Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Aleksi Hämäläinen

Learning Affordance Representations:

An Efficient Learning Approach for End-to-End Visuomotor Control

Master's Thesis
Espoo, July 2, 2019

Supervisors: Professor Ville Kyrki, Aalto University
Advisor: Karol Arndt M.Eng.

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Aleksi Hämäläinen		
Title: Learning Affordance Representations: An Efficient Learning Approach for End-to-End Visuomotor Control			
Date:	July 2, 2019	**Pages:**	60
Major:	Machine Learning and Data Mining	**Code:**	SCI3044
Supervisors:	Professor Ville Kyrki		
Advisor:	Karol Arndt M.Eng.		

The development of data-driven approaches, such as deep learning, has led to the emergence of systems that have achieved human-like performance in wide variety of tasks. For robotic tasks, deep data-driven models are introduced to create adaptive systems without the need of explicitly programming them. These adaptive systems are needed in situations, where task and environment changes remain unforeseen.

Convolutional neural networks (CNNs) have become the standard way to process visual data in robotics. End-to-end neural network models that operate the entire control task can perform various complex tasks with little feature engineering. However, the adaptivity of these systems goes hand in hand with the level of variation in the training data. Training end-to-end deep robotic systems requires a lot of domain-, task-, and hardware-specific data, which is often costly to provide.

In this work, we propose to tackle this issue by employing a deep neural network with a modular architecture, consisting of separate perception, policy, and trajectory parts. Each part of the system is trained fully on synthetic data or in simulation. The data is exchanged between parts of the system as low-dimensional representations of affordances and trajectories. The performance is then evaluated in a zero-shot transfer scenario using the Franka Panda robotic arm. Results demonstrate that a low-dimensional representation of scene affordances extracted from an RGB image is sufficient to successfully train manipulator policies.

Keywords: robotics, representation learning, end-to-end visuomotor control, variational autoencoder, zero-shot transfer, deep learning, affordance

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Aleksi Hämäläinen		
Työn nimi: Matalaulotteisen affordanssiesityksen oppiminen ja tämän hyödyntäminen robottijärjestelmän koulutuksessa			
Päiväys:	2. heinäkuuta 2019	**Sivumäärä:**	60
Pääaine:	Koneoppiminen ja tiedonlouhinta	**Koodi:**	SCI3044
Valvojat:	Professori Ville Kyrki		
Ohjaaja:	Diplomi-insinööri Karol Arndt		

Tietopohjaisten oppimismenetelmien etenkin syväoppimisen viimeaikainen kehitys on synnyttänyt järjestelmiä, jotka ovat saavuttaneet ihmistasoisen suorituskyvyn ihmisälyä vaativissa tehtävissä. Syväoppimiseen pohjautuvia robottijärjestelmiä ollaan kehitetty, jotta ympäristön ja tehtävän muutoksiin mukautuvaisempia robotteja voitaisiin ottaa käyttöön.

Konvoluutioneuroverkkojen käyttö kuvatiedon käsittelyssä robotiikassa on yleistä. Neuroverkkomallit, jotka käsittelevät anturitietoa ja suorittavat päätöksenteon ja säädön, voivat oppia monimutkaisia tehtäviä ilman käsin tehtyä kehitystyötä. Näiden järjestelmien kyky mukautua ympäristön muutoksiin on kuitenkin suoraan verrannollinen koulutustiedon monimuotoisuuteen. Syväoppimiseen pohjautuva robottijärjestelmä vaatii oppiakseen suuren määrän ympäristö-, tehtävä-, ja laitteisto-ominaista koulutustietoa, mikä joudutaan yleensä kerätä tehottomasti käsin.

Tämän työn tarkoitus on esittää ratkaisu yllämainittuun tehottomuuteen. Esittelemme neuroverkkoarkkitehtuurin, joka koostuu kolmesta erillisestä komponentista. Nämä komponentit koulutetaan erikseen ja koulutus ollaan ainoastaan toteutettu simulaatiossa tai synteettisellä tiedolla ilman fyysisen maailman lisäkouluttautumista Ensimmäinen komponentti tuottaa RGB-kuvasta matalaulotteisen affordanssiesityksen. Tämän esityksen pohjalta toinen komponentti tuottaa matalaulotteisten liikeratаesityksen. Kolmas komponentti luo tämän esityksen pohjalta täysimittaisen liikeradan teollisuusrobotille. Järjestelmän suorituskykyä arvioidaan fyysisessä ympäristössä ilman lisäkoulutusta Franka Panda -teollisuusrobotilla. Tulokset osoittavat, että kuvatieto voidaan esittää matalaulotteisena affordanssiesityksenä ja tätä esitystä voidaan käyttää säätötehtävän oppimiseen.

Asiasanat:	robotiikka, esitysoppiminen, päästä päähän visuomotorinen säätö, variaationaalinen autoenkooderi, nollaotossiirtymäoppiminen, syväoppiminen, affordanssi
Kieli:	Englanti

Acknowledgements

Thank you Karol for all the help and unforgettable moments while working on the thesis. You are the best thesis advisor one could have! Thank you Ville for the strong leadership and trust in achieving the final outcome.

Finally, thank you my beloved one for giving me so much support and time in this journey. I would not be here without you Stella.

Helsinki, July 2, 2019

Aleksi Hämäläinen

Abbreviations and Acronyms

CNN	Convolutional neural network
DAE	Denoising autoencoder
IROS 2019	2019 IEEE/RSJ International Conference on Intelligent Robots and Systems
KL-divergence	Kullback-Leibler divergence
MDP	Markov decision process
MSE	Mean squared error
ROS	Robot Operating System
RRT	Rapidly exploring random tree
RRT*	Rapidly exploring random tree star
SRL	State representation learning
VAE	Variational autoencoder
VAED	Variational affordance encoder-decoder
YCB	Yale-CMU-Berkeley

Symbols

a_t	action at time t
$\hat{a}_t$	predicted action
D_{KL}	KL-divergence
$\mathcal{N}$	normal distribution
o_t	observation at time t
$\hat{o}_t$	predicted observation
s_t	environment state at time t
$\hat{s}_t$	predicted state
$\mathcal{L}$	loss function
$\mathcal{L}_{recon}$	reconstruction loss function
β	coefficient hyperparameter for KL-divergence
θ	neural network parameters
τ	length of an episodic task
ϕ_θ	decoder network
ψ_θ	encoder network

Contents

Chapter 1

Introduction

In robotics research, the objective is to move us towards a world that we have only witnessed in science fiction so far — a world where not only all the repetitive and tedious tasks are automated by robotic systems, but also those that require human-like intelligence. We humans could then focus on tasks that motivate us and require skills that we are the most capable of, such as understanding highly complex conceptual problems, and being creative and emotionally intelligent — creating and enjoying art, and solving the most notorious problems.

We have witnessed the introduction of systems that have achieved adequate or superhuman performance in complex tasks, such as image classification, playing Atari games, and Go [Krizhevsky et al., 2012, Mnih et al., 2013, Silver et al., 2017]. These systems have been built to understand and process complex semantics of the problem domains. The critical factors of these recent successes are improved utilization of computational resources and the recent development of data-driven approaches, especially deep learning methods.

Deep learning is a data-driven approach, where the model learns to perform its objective by identifying meaningful features from its training data. Deep learning models include multiple non-linear function layers. In other words, they are neural networks with multiple hidden layers. Each of the layers include a large number of non-linear functions that pass their outputs to the following layer. Here, the layers in the model process increasingly abstract features from its input. This deep structure enables the model to learn semantic features that cannot be feature engineered. Different deep neural network structures have performed successfully in many complex tasks where sufficient data has been available. Especially, convolutional neural networks (CNNs), a class of deep neural networks, have become the *de facto* standard way of processing visual input [Krizhevsky et al., 2012].

1.1 Data-Driven Robot Control

In robot control, the system observes its surrounding environment, makes a decision, and then interacts with its environment. The decision is made based on the state of the environment and the system's internal status. The environment state is estimated from observed data. In visuomotor control, the observed data is high-dimensional visual data.

Currently in industrial settings, robotic arms are mainly utilized in tasks where repetition is high and expected changes to the task are small. Common tasks are, for example, welding and painting. In such applications, robots' behavior can be programmed by experts, as these working environments are predictive, and no major changes occur in the production line. However, for many tasks, foreseeing all the possible scenarios and programming desired behaviors for all of these scenarios is difficult, or even impossible, and expensive. Due to the time and expenses needed for the development, many repetitive physical tasks are still more efficiently performed by humans.

To introduce robots to more complex tasks, recent years have seen widespread research and adoption of deep learning in robotics. With convolutional neural networks, the environment state is estimated from observed visual data. Visual data can enable a robot system to be utilized in a more complex environment, as it can generally capture a wide variety of information about the working environment. Additionally, recent works have introduced end-to-end approaches, where a deep neural network model directly computes the entire mapping from observations to actions. As in other task domains, such as playing Atari games, end-to-end approaches have the potential to produce systems that can perform their tasks robustly and more efficiently than systems programmed by human experts — sometimes performing better than originally expected by their creators [Ecarlat et al., 2015]. This can be essential for tasks where human-like intelligence is required.

Despite the potential of introducing these deep data-driven models for robotic tasks, a few issues limit their further adoption. Specifically, (i) learning an end-to-end neural network system requires vast amounts of task-specific training data and months of real world experience [Levine et al., 2018], which are simply unfeasible to provide; (ii) using task- and domain-specific training data means that changing the task, such as transferring it to another robot or changing the task objective, requires often entirely new training data, which is costly to obtain [Levine et al., 2016]. In other words, the adaptivity of a learned system goes hand in hand with the level of variation in training data; (iii) the behavior of deep models can be difficult to predict and interpret, as the intermediate representations learned by the neural network rarely provide meaningful information to humans — despite recent successes in interpreting neural networks, it is still difficult to gain understanding of why the model behaves the way it does.

1.2 Objectives and Contributions

The goal of this thesis is to build an end-to-end robotic system that minimizes the need for task- and domain-specific training data and enables transferring its learned behavior to other tasks. Additionally, one of the aims is to make the learned behavior of the system interpretable.

We introduce a modular neural network structure, which consists of three parts: perception, policy, and trajectory generation. Each part of the system is a separately trained neural network. For the perception part, we introduce an approach to learn low-dimensional affordance representations of visual observations. Affordances — an idea originated from perceptual psychology — express semantic information of what can be done with each part of the object in a perceived environment [Gibson, 2014]. Representing visual observations as affordances is a common approach to model a surrounding environment for a robot to interact with [Do et al., 2018, Hassanin et al., 2018]. Representing affordances in low-dimension enables a policy to be more efficiently learned. Affordances are environment invariant, and therefore, the learned system can perform its task with the same low-dimensional affordance representation in different environments. The research questions for this thesis are:

- How can low-dimensional affordance representations be learned?
- How can the learned representations be used in robot control?
- Can the learned system be transferred to another environment?

For learning low-dimensional affordance representations, we use a variational encoder-decoder neural network structure [Kingma and Welling, 2013]. Using variational encoder-decoders ensures neighborhood preservation and disentanglement of the low-dimensional representation space [Higgins et al., 2016].

The trajectory generation part learns a low-dimensional trajectory representation. A similar variational encoder-decoder structure as for the perception part is used to learn low-dimensional trajectory representations. The policy part learns an intermediate relationship between the low-dimensional affordance and trajectory representations, which maximizes the system's performance for its task. Low-dimensionality of both trajectory and perception representations allows the policy layer to be retrained with relatively little training data.

Our system has been entirely trained on synthetic data or in simulation, with no real-world adaptation. Training images for the perception part are generated using randomized textures, object shapes, distractor objects and camera viewpoints to increase the model's capability in adapting to new environments [James et al., 2017, Tobin et al., 2017]. For the trajectory generation part, the training dataset includes task-suitable trajectories that are generated with a motion planning algorithm. The policy part, trained in a physics simulator, maps low-dimensional affordance representations to representations of trajectories, which are then passed to the trajectory decoder.

We demonstrate that the system performs well in the task of inserting a ball into a container both in a simulated and real environment. The real environment experiments are conducted with a Franka Panda robotic arm, without the need for any real world adaptation of our neural network parts. This is referred to as a zero-shot sim-to-real transfer scenario. We also evaluate the system's susceptibility to clutter in the vicinity of the cup. The results demonstrate that the robotic arm can complete the task even in heavily cluttered environments.

Based on the research conducted for this thesis, a research paper has been submitted to 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019) [Hämäläinen et al., 2019]. The source code of the work is publicly available [1].

1.3 Outline

The structure of the thesis is the following. First, Chapter 2 studies previous data-driven approaches for physical robot control tasks. Chapter 3 introduces our approach for learning low-dimensional affordance representation and our modular neural network system for the end-to-end visuomotor task. Chapter 4 describes the overall visuomotor control setup. Chapter 5 empirically evaluates the introduced system. Finally, the overall conclusions of the topic are provided in Chapter 6.

[1] https://github.com/gamleksi/affordancegym

Chapter 2

Learning Robotic Tasks

In this work, we are interested in studying data-driven approaches that are suitable for robot control tasks — the systems learn the desired behavior, called a policy, with a sufficient of amount training data.

In Section 2.1, the policy learning problem is described. For robot control tasks, information about the working environment is obtained from the sensors. However, raw sensor data is often high-dimensional, making it difficult to directly extract task-relevant information. In Section 2.2, we introduce state representation learning (SRL) for identifying the necessary information from the high-dimensional data. By introducing an auxiliary learning task, a state representation space can be learned in an unsupervised manner. This can improve the policy learning process, as acquiring state values for training can be inefficient or even impossible. In Section 2.3, we study the required characteristics for learning state representation in an unsupervised manner. In Section 2.4, we study approaches to learn state representation in an unsupervised manner for robotic control tasks.

In Section 2.5, we introduce a domain randomization and trajectory optimization method that can decrease the amount of task- or domain-specific training data required in the policy learning process.

2.1 Policy Learning Problem

In policy learning for a robot control task, the robot, called an agent, finds a policy that performs the task. The policy selects the motor commands, called actions, based on received observations. The policy can be described as a function $a = \pi_\theta(o)$ that maps observations o to actions a, where θ expresses the learned parameters of the policy. These observations include data from the environment captured by sensors.

For learning the policy, we consider that the policy learning problem is a Markov Decision Process (MDP) [Sutton and Barto, 2018]. The task is episodic, and thus consists of a finite number of time steps, τ. At each time step t, the policy receives a state s_t and a reward r_t from the environment, and then selects an action a_t based on the state s_t. Figure 2.1 visualizes the process.

The received rewards of the entire episode guide the robot to find the policy that performs the task. The objective of the robot is to find a policy π_θ that maps a state s_t to an action a_t, which produces the maximum expected cumulative reward of the task $\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\tau} r_t\right]$.

In an MDP, the state s_t maintains the Markovian property — the state s_t includes information about all the aspects of previous actions and states that have an effect on the future. Based only on the received state s_t, the policy can select the action a_t that maximizes the future expected reward.

A state space expresses the set of possible values of state s_t at any t. The definition of a state space is not unique — the necessary information about the environment can be expressed in multiple ways. The policy can be learned, for example, with reinforcement learning or Value Iteration [Sutton and Barto, 2018].

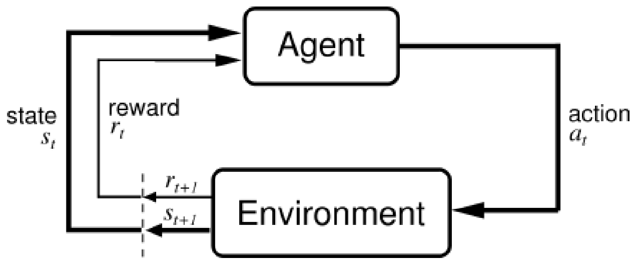

Figure 2.1: The agent-environment interaction process for policy learning (Source: [Sutton and Barto, 2018])

2.2 State Representation Learning

For robotic control tasks, the robot does not usually receive the state s_t, but instead, the observation o_t from the sensors. The observation is often high-dimensional which includes information about the state s_t. It can be considered that the observation o_t is an outcome of the state s_t — the observation o_t is a high-dimensional description of the state s_t [Ghadirzadeh, 2018]. For robotic control, we are interested in finding the inverse of that description: the objective is to find a representation mapping from the observation o_t to the state s_t.

This representation mapping should represent the observation space in a state space, called the representation space. The problem of finding this representation mapping is referred to as the state representation learning (SRL) problem.

Many feature engineering approaches exist for extracting the state from the observations, but they fail in terms of flexibility — it is difficult to foresee everything that can occur in the environment. The state representation can also be learned in a supervised manner. However, obtaining the labeled training data is inefficient.

For policy learning, it is not necessary that a learned state representation space explicitly expresses the physical, human interpretable, state values of the environment. A mapping from an abstract state representation to the desired action can be learned, if the representation maintains the Markovian property [Böhmer et al., 2015]. The abstract state representation can decrease the complexity of the policy learning problem, if the representation is low-dimensional.

The unsupervised approaches for SRL can be divided into end-to-end policy learning and auxiliary task learning approaches. The abstract state representation can be learned in an unsupervised manner — a neural network indirectly learns the representation while learning some other task.

Learning the state representation in an unsupervised manner can be more efficient and feasible than learning the physical state values of the environment. The training data for an auxiliary task can be obtained more easily than the physical state values of the environment.

In end-to-end policy learning approaches, the agent learns a policy that directly maps the observations to the actions. Here, the entire system is defined as a single neural network that, when trained with reinforcement learning, learns the mapping by interacting with the environment. The agent receives rewards from the environment based on its interactions. The reward directs the agent to find a policy that maximizes the reward. The end-to-end policy model does not learn directly the state representation but the intermediate value vector of the neural network model can be considered as an implicit state representation of the observation [Ghadirzadeh et al., 2017].

Based on previous examples in other domains, such as in computer vision, introducing a single neural network achieves more robust performance than a multi-component system [Krizhevsky et al., 2012, Szegedy et al., 2015]. End-to-end learning approaches have been successfully introduced for various complex policy learning problems, for example, arcade games and humanoid movements in simulation [Mnih et al., 2013, Schulman et al., 2017].

Similarly, in policy learning in robotics, one can presume to improve performance by introducing a single deep neural network structure. However, the end-to-end learning process requires vast amounts of training data, which is not applicable for robotic tasks [Levine et al., 2018].

The initial idea of learning the state representation with auxiliary tasks is that the same state representation, can be utilized for multiple tasks. For example, the state presentation of cup locations on a table can be used to pour tea into the cups, and to generate an image of the cups. By introducing an auxiliary task, the objective is to improve the learning process of the state representation. The auxiliary task can be learned either separately or simultaneously with the policy model.

When separately learning the state representation with the auxiliary task, the learned state representation is transferred to the target task. This can be beneficial when the training data for the auxiliary task can be efficiently obtained without performing the control task [Ghadirzadeh et al., 2017].

By simultaneously learning the auxiliary task and the policy model, the objective is to strengthen reward signals that direct the policy learning process. The end-to-end learning process requires vast amounts of training data, because the received reward signal from the environment is usually weak in proportion to the complexity of the problem [Ghadirzadeh, 2018]. The agent is required to learn both identifying state information, and choosing the action from one-dimensional reward signal. The auxiliary task can be utilized to direct the learning of the state representation [Finn et al., 2016, van Hoof et al., 2016].

2.3 Abstract Representation Characteristics

In this thesis, we study SRL in an unsupervised manner, specifically, SRL approaches that introduce auxiliary tasks. By introducing an auxiliary task for the policy learning process, the objective is to reduce the required amount of training data and speed up the training. The state representation is learned in an unsupervised manner. A neural network for the auxiliary task learns indirectly the state representation of the observations. This state representation is an abstract representation of the environment's state — it cannot be directly interpreted by humans.

However, the abstract representation has to maintain some properties for the policy model to learn to interpret the representation, and map the abstract representation to the desired actions. In this section, we study these properties. From now on, the state representation expresses the abstract representation that is learned in an unsupervised manner, and s_t expresses the abstract state representation of the observation o_t.

In policy learning, the objective is to learn to interpret the relation of the state representations and actions. In practice, different state-action combinations are explored. The complexity of the policy search can be decreased by introducing simpler state representations — a lower dimensional state representation space [Böhmer et al., 2015].

A state representation space should preserve predictable structure for a policy to learn to interpolate unseen states that have similar features as previously seen states [Böhmer et al., 2015]. This is achieved if the representation space is disentangled and exhibits the neighborhood preservation property.

In a disentangled space, each state feature of a system, such as position, orientation and size, is separately represented in the state representation space [Burgess et al., 2018]. Ideally, each dimension represents a separate feature. In a neighborhood preserved state representation space, observations that have similar features in the observation space, are located closely [Böhmer et al., 2015]. These properties promote interpolation, which decreases the complexity of policy learning. A policy can predict the correct action for a given state s_t based on its previous experiences with similar states [Böhmer et al., 2015].

The described state representation properties are hard to verify directly when trained in an unsupervised manner, as the state representation values are abstract. The properties are usually indirectly reviewed with a control task — how efficiently a policy for the task can be learned — or by studying the intermediate relationship between the observation and abstract spaces [Lesort et al., 2018]. To encourage to learn the representation with the properties, previous works have included learning constraints [Finn et al., 2016, Higgins et al., 2016].

2.4 Auxiliary Task Approaches

This section introduces SRL approaches that learn the representation in an unsupervised manner with auxiliary tasks. These approaches are reconstruction, forward, inverse, and end-to-end approaches.

In the reconstruction approach, the state representation is learned with an encoder-decoder neural network structure, that first encodes an observation o_t to a low-dimensional representation [Finn et al., 2016, Mattner et al., 2012]. From this low-dimensional representation, a decoder generates the same observation o_t or another high-dimensional target value. The state representation is the encoder. In the forward learning approach, state representation of an observation o_t is simultaneously learned with a transition function that predicts a future state s_{t+1} from an encoded s_t and an action a_t [Goroshin et al., 2015, Watter et al., 2015]. In the inverse model approach,

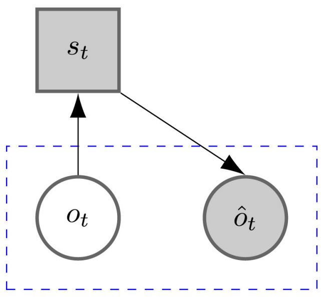

Figure 2.2: The reconstruction approach: the error is between o_t and $\hat{o}_t$. The white components are input data, and the gray components are generated by the model. (Source: [Lesort et al., 2017])

states s_t and s_{t+1} are used to predict an action a_t [Agrawal et al., 2016]. Here, a state representation network outputs both states $s_{t,t+1}$.

In the end-to-end approach, a mapping from an observation o_t to a task related target is learned. An intermediate value vector of the learned neural network model is an implicit state representation of the observation. This implicit state representation is transferred to the target policy.

Learning the state representation and policy for a physical task is in principle a slow process. It requires task- and environment-specific data which is obtained by, for example, changing objects in the environment, or letting a robot interact with the environment. For each of the described approaches, we describe previously proposed methods with a real physical robot. These works have shown how prior knowledge can be included to a learning process to improve the learning processes [Agrawal et al., 2016, Devin et al., 2018, Finn et al., 2016, Ghadirzadeh, 2018, Watter et al., 2015]. Here, priors cover preliminary knowledge that can be incorporated to a system to improve its performance in a control task [Ghadirzadeh, 2018].

2.4.1 Reconstruction Approach

The reconstruction approach utilizes an encoder-decoder structure. Both encoder ψ_θ and decoder ϕ_θ are neural networks that are parametrized with θ. An autoencoder is a special case of this structure, where the reconstruction target is the same as the input. It learns to encode an observation o_t to a low-dimensional state representation $s_t = \psi_\theta(o_t)$, which is then used to generate $\hat{o}_t = \phi_\theta(s_t)$. An error signal is the reconstruction loss between the original o_t and generated $\hat{o}_t$ [Bourlard and Kamp, 1988], which is usually the mean squared error (MSE) expressed as:

$$\mathcal{L}_{recon}(o_t, \hat{o}_t) = \|o_t - \hat{o}_t\|_2^2. \tag{2.1}$$

Figure 2.2 shows the semantic structure of this approach.

To learn a state representation space, no other information, than a low-dimensional representation s_t, is shared between encoder and decoder layers. For state representation learning, the dimension of the encoded space is lower than the dimension of the observation space. Hence, the low-dimensional representation can be considered as an information bottleneck of the autoencoder model. Not all necessary information can be included to s_t to obtain an accurate reconstruction of o_t. To obtain a minimal error between o_t and $\hat{o}_t$, s_t includes information of a given o_t that maximizes its difference from the other observations — s_t ignores information about features that has the smallest variance in the observation domain.

For the purpose of policy learning, an autoencoder model can learn to encode necessary state information about a system for a policy learning problem. However, this depends on the observation domain. An encoder-decoder model may fails to learn state representation for a policy task, when, meaningless features, such as the texture and color of the manipulated object, for the policy task have higher variance in the observation domain than the necessary state features. Here, the encoder learns to ignore necessary information to the low-dimensional representation.

Prior knowledge of the task can be included to an encoder-decoder learning process to ignore unnecessary information in the low-dimensional space: the decoding target can differ from the original observation o_t. Here, the model is an encoder-decoder model.

For instance, the target can be gray-scaled for tasks where color and texture information of objects are unnecessary [Finn et al., 2016]. Ghadirzadeh et al. [2017] introduced a reconstruction prior for the task of throwing a ball to hit a specific object on the table. Multiple clutter objects varied in each observed sample. For the ball throwing policy, only information about the target object was required. Ghadirzadeh et al. [2017] removed all clutter objects from the target image. Here, their encoder-decoder model needed to encode information about the target object to a low-dimensional representation space. Decreasing complexity of SRL can reduce the amount of training data needed, but modifying target samples as in [Ghadirzadeh et al., 2017] is not a scalable approach.

Increasing the complexity of an encoder-decoder task can act as a learning regularizer. Vincent et al. [2008] introduced a denoising autoencoder (DAE) approach, where, during the learning phase, a DAE model reconstructs an observation o_t from corrupted version of the observation. By increasing the noise complexity, the DAE model learns to operate more robustly, and focuses on critical features in the observations [Vincent et al., 2008].

The learning process of an encoder-decoder model does not encourage to learn a state representation that preserves a neighborhood and disentangled structure [Higgins et al., 2016]. To solve this challenge, previous works have included prior constraints or auxiliary tasks. Finn et al. [2016] included a prior constraint that promotes an encoder to represent only spatial features of the observed objects. They used the spatial soft-max function as the activation function of the encoder's last layer. The spatial features describe where to move the end-effector. The spatial representation space is continuous and predictable, which satisfies the desired properties described in Section 2.3. However, the spatial representation can solely handle situations where each learned visual feature occurs exactly once in the image [Finn et al., 2016].

Variational autoencoders (VAEs) include a Kullback-Leibler divergence (KL-divergence) term to the loss function [Higgins et al., 2016]. The KL-divergence encourages neighborhood preservation and disentanglement in state representation space [Burgess et al., 2018]. Instead of producing a low-dimensional vector, the encoder of VAE produces a low-dimensional Gaussian distribution $\psi_\theta(o_t) = \mathcal{N}(\boldsymbol{\mu}_\theta(o_t), \boldsymbol{\Sigma}_\theta(o_t))$, where μ_θ and Σ_θ are neural networks, $\boldsymbol{\mu}_\theta(o_t)$ is a mean vector, and $\boldsymbol{\Sigma}_\theta(o_t)$ is a diagonal covariance matrix. The KL-divergence term is calculated between the encoded distribution and a normal distribution with zero mean and unit variance:

$$D_{KL}(\mathcal{N}(\psi_\theta(o_t) \| \mathcal{N}(\mathbf{0}, \mathbf{1})) = \frac{1}{2} \sum_{i=1}^{k} (\sigma_i^2 + \mu_i^2 - \ln(\sigma_i^2) - 1), \tag{2.2}$$

where k is the number of dimensions in the representation space, μ_i and σ_i^2 are elements from $\boldsymbol{\mu}_\theta(o_t)$ and $\operatorname{diag}\big(\boldsymbol{\Sigma}_\theta(o_t)\big)$ respectively. The minimum of the KL-divergence term is obtained when the representation distribution of an observation o_t is $\psi_\theta(o_t) = \mathcal{N}(\mathbf{0}, \mathbf{1})$.

The KL-divergence term can be considered as a regularizer for learning a state representation space in an unsupervised manner. In the learning phase, a VAE model makes a trade-off between the KL-divergence and the reconstruction loss. The KL-divergence encourages information to be packed closer to the origin in the low-dimensional space, which decreases the model's capacity to represent information. When less capacity is available, the encoder learns to represent features in a more general and robust manner, with the cost of increasing the reconstruction loss [Burgess et al., 2018, Higgins et al., 2016].

The trade-off between the densely packed low-dimensional space and the reconstruction quality can be balanced by the β hyperparameter introduced by Higgins et al. [2016]. The loss function of VAE can then be expressed as

$$\mathcal{L}_{vae} \equiv \mathcal{L}_{recon}(o_t, \hat{o}_t) + \beta D_{KL}(\mathcal{N}(\psi_\theta(o_t) \| \mathcal{N}(\mathbf{0}, \mathbf{1})). \tag{2.3}$$

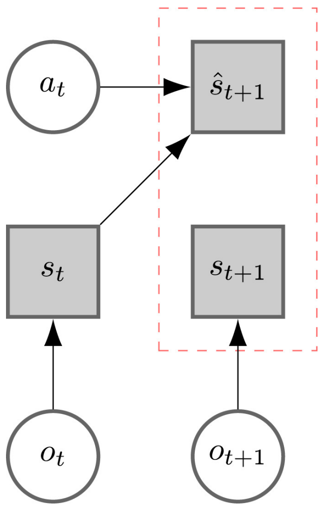

Figure 2.3: The forward approach: the error is calculated between the predicted $\hat{s}_{t+1}$ and the actual s_{t+1}. (Source: [Lesort et al., 2017])

By introducing a representation distribution $\psi_\theta(o_t)$, the neighborhood preservation can be assured [Ghadirzadeh, 2018]. During the learning process, a representation variable s_t is sampled from the encoded distribution $\psi_\theta(o_t)$. Then, the reconstruction loss is the error between the target value and the value generated by the sampled s_t. The error is a similarity metric which is chosen based on the data. To minimize the reconstruction loss, neighborhood values in the learned representation space generate similar features also in the target domain.

2.4.2 Forward Approach

A forward model consists of two parts: 1) a representation model encodes an observation o_t to a state s_t, and 2) a transition model predicts $\hat{s}_{t+1}$ from the state s_t and action a_t pair [Lesort et al., 2018]. The training loss is the error between the predicted $\hat{s}_{t+1}$ and the true s_{t+1} state from o_{t+1}. To fit the model, this error is then back-propagated through a transition function and a representation model. Figure 2.3 shows the semantic structure of this approach.

The learned state representation of the forward model maintains the Markovian property. Predicting s_{t+1} from s_t and a_t requires the transition model to learn the dynamics of the task environment. A state s_t should represent the system's state in order for the next state to be predicted based on an action a_t. Additionally, learning the dynamics of the system can be utilized to learn a policy [Watter et al., 2015].

A system prior can be included to a forward model by defining the structure of the transition model. For example, the transition model can linearly transform s_t and a_t to $\hat{s}_{t+1}$:

$$\hat{s}_{t+1} = f_\theta(s_t, a_t) = W_\theta s_t + U_\theta a_t + V_\theta$$

where θ represents either constant or learned parameters of W, U, and V [Goroshin et al., 2015, van Hoof et al., 2016, Watter et al., 2015]. Including the linear dynamic prior has shown to result in more stable policy learning results with a real robotic task [van Hoof et al., 2016].

van Hoof et al. [2016] included both the reconstruction and the forward approach to their state representation learning method. The reconstruction loss was obtained between $\hat{o}_{t+1}$ and o_{t+1} when o_t was given as an input to their forward model. An observation o_t was first encoded to s_t, from which the transition model produced a state prediction $\hat{s}_{t+1}$. The reconstructed $\hat{o}_{t+1}$ was computed from $\hat{s}_{t+1}$.

In their real robot experiment, the observed data was tactile, which has lower dimensionality than image observations [van Hoof et al., 2016]. Their results showed that the best performance was achieved with a VAE model that was retrained after each policy update. Learning simultaneously both the state representation and the policy for the task was achieved by including old and new policy data to state representation learning, and by introducing a prior to sample actions in policy learning.

2.4.3 Inverse Approach

An inverse model includes two different parts. Figure 2.4 shows the semantic structure of this approach. As in the forward model, a representation model encodes an observation o_t to a state s_t [Agrawal et al., 2016]. A prediction model receives s_t and s_{t+1} from the representation model, and predicts the action $\hat{a}_t$ that caused the transition from s_t to s_{t+1}. An error measure for learning is the difference between the correct a_t and the predicted $\hat{a}_t$. This value is then back-propagated through the prediction and representation parts. Here, the inverse model structure requires the representation model to encode sufficient information about o_t to find an action a_t that caused from s_t to s_{t+1}.

An inverse model for physical visual control was introduced to a task, where a robot pokes an object to another position on the table [Agrawal et al., 2016]. In this approach, the learned inverse model also acts as the policy. At each time step t, the current observation o_t and image of the final position of an object are given to the inverse model. The inverse model predicts the next

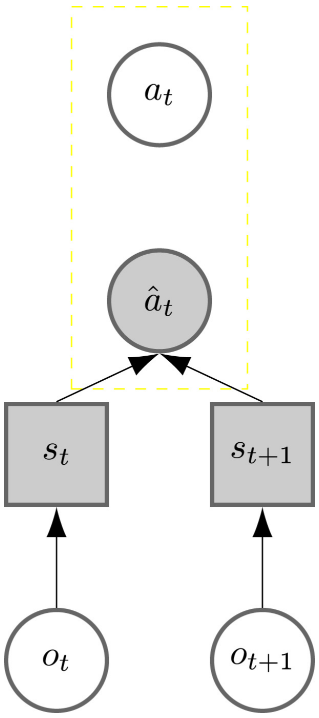

Figure 2.4: The inverse approach: the error is computed between the predicted action $\hat{a}_t$ and the actual a_t. (Source: [Lesort et al., 2017])

action a_t that then moves the object closer to its final position. The outcome of this action is observed at time $t+1$, which is then recursively used as the next input together with the image of the final position. Training data for learning the inverse model was generated by randomly poking objects on the table.

The forward model was included to the learning process to regularize inverse model learning. In the model, each training sample t includes observations o_t and o_{t+1}, and a corresponding action a_t. An action a_t includes the target poke position, and the length and angle of that poke. Observations o_t and o_{t+1} are passed to the inverse decoder that produces low-dimensional s_t and s_{t+1}. The prediction model results in an action prediction $\hat{a}_t$ with obtained states. Additionally, the forward model produces $\hat{s}_{t+1}$ from the decoded s_t and the action sample a_t. The inverse and forward losses were combined

$$\mathcal{L}_\theta(o_t, a_t, o_{t+1}, \theta) = \mathcal{L}_{inv}(a_t, \theta) + \lambda \mathcal{L}_{fwd}(s_{t+1}, \theta), \tag{2.4}$$

where θ is network parameters of the inverse and forward model, and λ is a regularizer hyperparameter. Agrawal et al. [2016] experimented with a simulation task, where less training data was required when the forward model was included. However, with enough training data, no performance improvement was achieved by including the forward model.

2.4.4 End-to-End Learning Approach

In the end-to-end approach, we learn a model that maps an observation o_t to a task related target. An intermediate value vector of the learned model can be considered as an implicit state representation of the observation [Ghadirzadeh et al., 2017].

The implicit state representation can be learned, for example, with another end-to-end policy task that can then consider an auxiliary task. This auxiliary task may require fewer interactions with the environment. The learned state representation can then be transferred to the target policy task.

The learned state representation is, however, difficult to transfer to other task settings. The end-to-end network is of a black-box nature, which makes it challenging to verify that the intermediate layer representations of the network preserve the properties introduced in Section 2.3 — the Markovianity, and the neighborhood preservation and disentanglement properties.

The intermediate representation can be learned by introducing a constrained neural network structure — the neural network structure enforces to learn a certain intermediate representation. Devin et al. [2018] introduced the a Deep Object-Centric Representation method that produced an intermediate priority map representation. This priority map was utilized for policy learning. Their neural network structure consists of meta-attention, the task-specific attention and the policy part.

The meta-attention part, that consists of the pre-trained layers, produces semantic and positional information about the recognized objects in an observed image. The task-specific attention part produces the priority-map of the semantic object information for the task. For example, for the task to move a mug to a desired position, the priority-map would pay attention to semantic information about the mug. The policy part receives a compound of the position information map from the meta-attention part and a priority-map from the task-specific attention part. The information combination expresses where important objects for the task are located in an image. Additionally, the current robot pose is given to the policy part.

For the meta-attention part, the first layers of AlexNet [Krizhevsky et al., 2012] were utilized to learn more general policies with fewer training samples. They argued that the number of pre-trained layers to be utilized is essentially a trade-off between flexibility and generalization of policy learning. The policy tends towards flexibility in identifying meaningful features, the less the pre-trained layers from the classifier network are used. In contrast, using too few of the pre-trained layers (or none at all), the policy focuses on overly general features, such as raw pixels or histogram of gradients, which restricts a policy to operate in a limited environment. This is essentially a challenge in policy learning in robotics when a limited amount of training data is available.

The task-specific attention part is trained in a supervised manner, where the task is to predict the next robot joint pose at time $t + 1$ from an image and robot state pair at time t. A policy network, separate from the one used for the final control task, is trained to map the output of the task-specific attention to the desired trajectory pose. Training data included demonstrations of a manually moved robot arm performing the same task. The priority-map representation of the task-specific attention part can be transferred to the final task.

2.5 Priors for Visuomotor Robot Control

The particular bottleneck in learning a robotic control task is the need for task- and domain-specific training data. In this section, we introduce domain randomization and trajectory optimization methods that have shown efficiency improvements in the learning process.

2.5.1 Domain Randomization

The idea of domain randomization is to synthetically produce artificial training data that enables a model to learn to behave in a desirable manner in a task domain [Tobin et al., 2017]. Figure 2.5 shows domain randomized training samples and the corresponding sample of the target task domain. The task domain covers all the features and values that can exist when performing the task. Here, the domain randomized dataset should cover the range of values of features that can exist in the task domain.

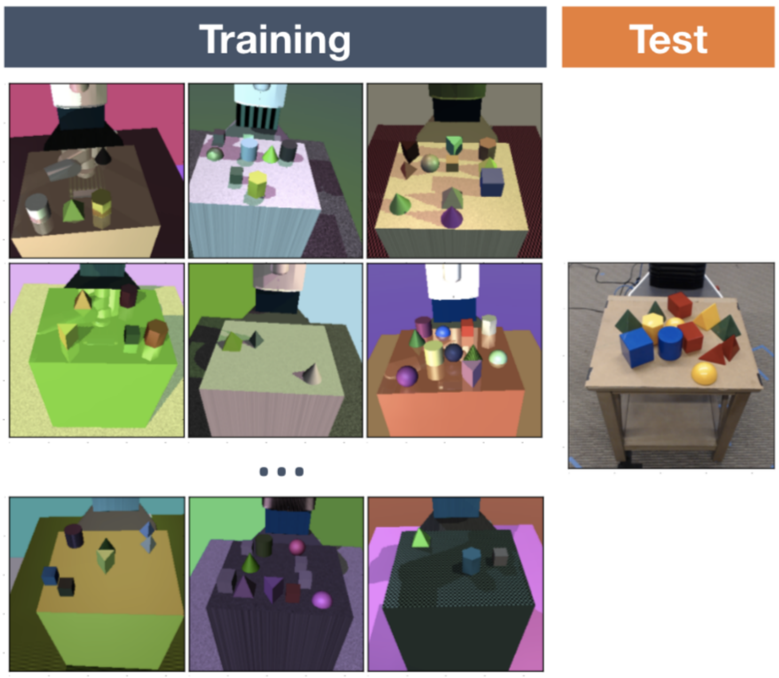

Figure 2.5: Domain randomized training data was generated to learn to detect object poses in the test environment [Tobin et al., 2017]. (Source: [Tobin et al., 2017])

Generating a large training dataset is faster than building an accurate computer model of the task domain and defining precise feature value regions to be learned. Especially, rendering high quality RGB images is inefficient. In domain randomization, the goal is to learn a model that can achieve its objective without disturbance from task-invariant changes in the environment [James et al., 2017]. Training samples do not need to correspond to those in the task domain when the training data includes high variation in the task-invariant features.

In Tobin et al. [2017], a neural network, trained with a domain randomized dataset, learned to find a desired object position from an RGB image. Features, such as textures, object materials, and lights, were randomly sampled for each training image. Samples of the domain randomization dataset are shown in Figure 2.5. James et al. [2017] introduced a domain randomization method for transferring an end-to-end learned policy from simulation to the real environment.

2.5.2 Trajectory Optimization

In a trajectory optimization problem, the policy learns to generate a sequence of actions, which successfully performs the task from a single observation [Ghadirzadeh, 2018]. This problem is considered a feedforward control problem. The length of an episode is one step, and after that, the policy receives a reward of the generated sequence of actions.

Previously described policy learning problems received an immediate observation feedback after each action, which is considered a feedback control problem. Learning the feedback control is suitable for more complex tasks than learning a feedforward control. Nevertheless, the feedforward control problem requires less data for learning than the feedback control task [Ghadirzadeh et al., 2017]. This can be beneficial for tasks, where the task environment is static while the trajectory is executed. In other words, an observation at the beginning of the episode contains enough information to perform the task.

Ghadirzadeh et al. [2017] introduced a VAE model that learned low-dimensional representations of task specific trajectories. Training data was generated with a planning algorithm, and the dataset included trajectories that produced desired outcomes for the task. Here, they were able to separately train the trajectory generation model and the feedforward policy. Their approach took advantage of their prior knowledge on successfully performed trajectories for the task.

2.6 Discussion

Section 2.4 introduced previous works on end-to-end visuomotor robotics control with different SRL approaches learned with auxiliary tasks. The works introduced methods to achieve more suitable learning processes for robotic control by decreasing the amount of training data required. For example, Ghadirzadeh et al. [2017] modified the reconstruction targets of their training data, Finn et al. [2016] constrained the state representation space, and Devin et al. [2018] utilized the pre-trained network layers and transferred the task-specific learnings from the demonstration task to the control task. However, these SRL methods all obtained task- and domain-specific training data manually.

Domain randomization methods enable transferring a simulation-learned model into the real environment without real-world adaptation. However, the previously introduced works have solely studied end-to-end learning approaches. These approaches require the system to be trained from the beginning, if the task is altered, for example, when a robot or an objective is changed.

A learned state representation model can be utilized for tasks related to the original task. For example, the state representation of tea pouring may capture necessary information for a cup grasping task. However, this can be verified only through the means of experimental evaluation [Lesort et al., 2018].

The representation learning approach can indicate which information the model represents [van Hoof et al., 2016]. Additionally, the model can learn the representation without interacting with a robot arm [Ghadirzadeh et al., 2017]. They modified the target domain of their encoder-decoded model, which enforced the model to learn the spatial representation of the target object. However, their approach required domain-specific data, as well as manually modifying the reconstruction targets.

In Chapter 3, we introduce an affordance prior for learning a state representation. The prior enforces the model to learn a low-dimensional state representation of the environment affordances. We argue that the prior enables the state representation to be used for new tasks. Additionally, the domain randomization method is used for training our SRL model, which enables the model to be utilized in a variety of domains. For efficiently reusing a learned state representation in other tasks, we propose a modular neural network structure that utilizes the trajectory optimization method.

Chapter 3

Affordance Learning for Robot Control

The affordances of an object express possible interactions with the object's parts [Gibson, 2014]. For example, the handle of a cup represents the *grasp* affordance, and the cup's interior part represents the *contain* affordance. Gibson argued that people perceive affordances in their surrounding environment. Affordances can be visually expressed by labelling affordance regions of received images. Figure 3.1 show example images of different tools and their corresponding affordances.

Learning an affordance state representation would enable utilizing the same representation for a variety of tasks. Each object type has its own general affordances. For example, cups in general have the *contain* and *grasp* affordances. The same affordances can be utilized in multiple tasks without retraining.

In Section 3.1, we study previous works in visual affordance learning. Section 3.2 proposes a modular neural network system that utilizes visual affordance prior.

3.1 Visual Affordance Learning

Visual affordance learning approaches extract affordances from visual observations. Several recent studies have introduced affordance learning as a segmentation problem [Chuang et al., 2018, Hassanin et al., 2018, Luddecke and Worgotter, 2017, Myers et al., 2014, 2015, Roy and Todorovic, 2016]. Here, a learning model is required to identify affordance features from an image. From these features, the model generates a segmented affordance image that expresses the corresponding pixel-level affordance labeling of the

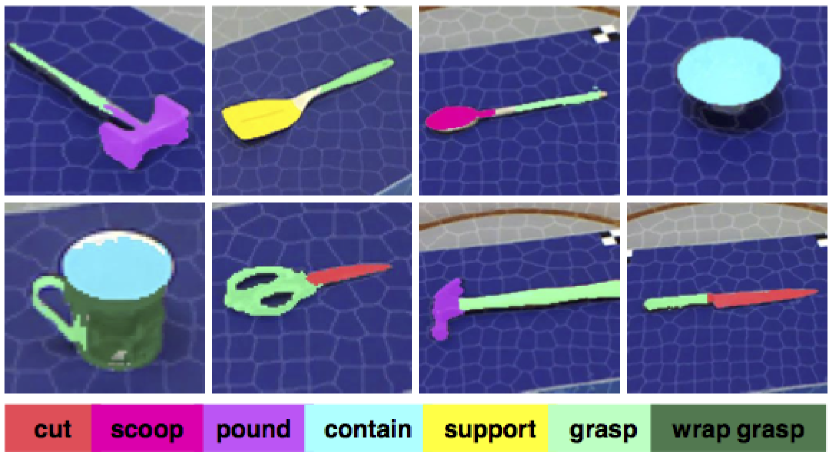

Figure 3.1: Samples from the UMD dataset are labeled by the affordance segmentation method introduced in [Myers et al., 2014]. The UMD dataset includes the shown affordance categories. (Source [Myers et al., 2014])

observed image.

Learning to generate affordance images requires a model to be trained in a supervised manner. The objective is to generate the target affordance image from the corresponding observation. A common dataset for this task is the UMD dataset that includes 30 0000 samples of 105 different kitchen, workshop, and garden tools [Myers et al., 2014]. These tools have seven different affordances in total. However, the dataset only includes samples from a single, simplistic environment with a single camera viewpoint and no clutter objects. Figure 3.1 shows samples from the UMD dataset.

Nguyen et al. [2017] introduced the IIT-AFF dataset, that includes labeled affordances of images of 10 object types. The dataset includes 8 835 samples. 60% of the images are from the ImageNet dataset, which enables utilizing layers from pre-trained deep network models. Figure 3.2 show samples from the IIT-AFF dataset, the corresponding affordances of the samples, and generated grasping frames. They extracted the grasping frames from the obtained affordance images, and used the frames to search trajectory.

A major bottleneck in the affordance segmentation learning is generating training data from the task environment, as affordance images usually need to be manually labeled. To mitigate this problem, Srikantha and Gall [2016] proposed an expectation-maximization method for learning pixel-level affordances trained with weakly labeled image data.

Nguyen et al. [2016] proposed an encoder-decoder architecture to learn different affordances from RGB-D images. For learning the model, they utilized the UMD dataset. In a more recent work, Do et al. [2018] introduced

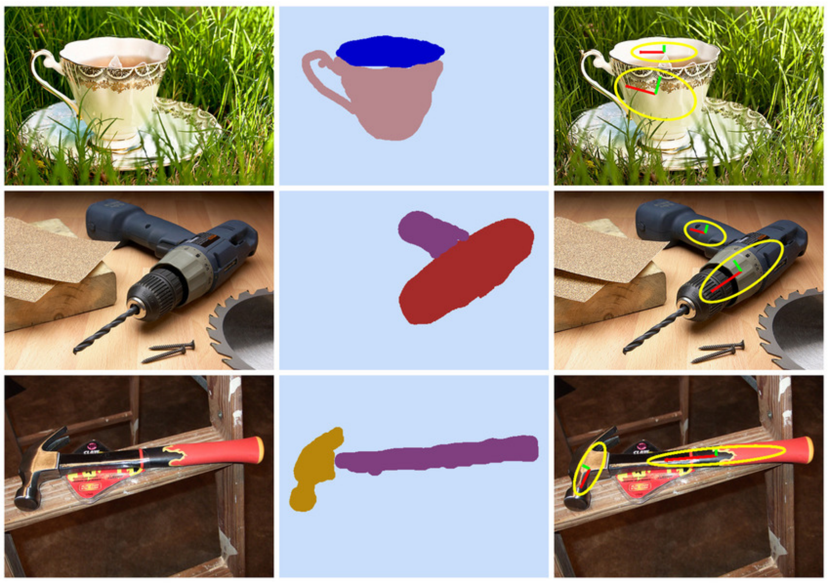

Figure 3.2: RGB images and their corresponding affordance images from the IIT-AFF dataset. Third column shows the grasp frames that were extracted from the affordance images [Nguyen et al., 2017]. (Source [Nguyen et al., 2017])

an object detector network to narrow down the image regions from which object affordances are extracted. They utilized the IIT-AFF dataset. They simultaneously trained the object detection and the affordance labeling tasks, which improved the learning process.

Previous works have extracted the desired positions of the robot end-effector from the affordance images [Do et al., 2018, Nguyen et al., 2016, 2017]. The trajectory to the desired position is planned with a path planner.

3.2 Modular Neural Network System for End-to-End Control

Our system consists of three separately trained parts—affordance perception, policy, and trajectory generation, similar to the approach in [Ghadirzadeh et al., 2017]. The policy problem is described as a feedforward control problem that was defined in Section 2.5.2 — the length of an episode is one. Here, we can express the process accordingly: the perception part encodes an RGB image o to a low-dimensional affordance representation s. The policy part maps the state s to a desired low-dimensional action vector a. Finally, based

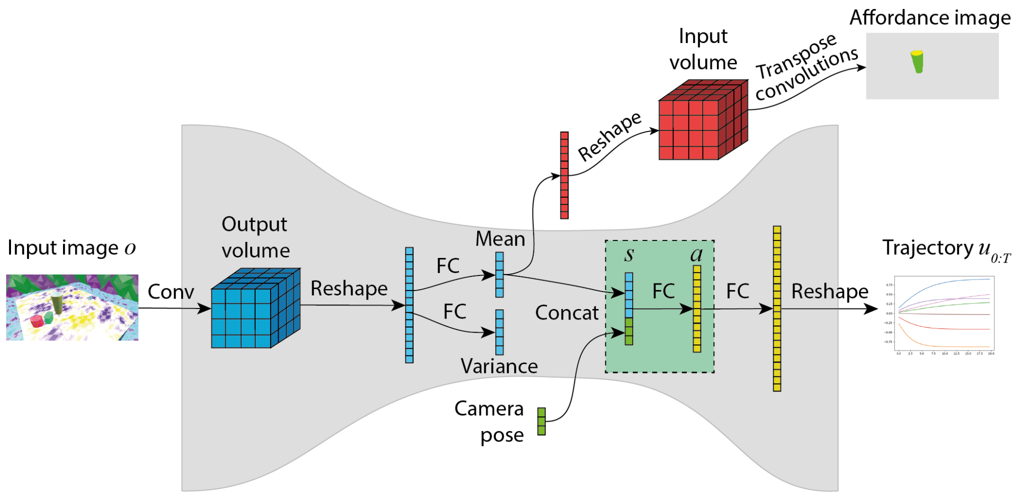

Figure 3.3: Structural overview of the system. An input image o is first processed by the affordance encoder (blue). The policy part (green rectangle) produces action a, which is then decoded to a robot trajectory $u_{0:T}$. The affordance image is also generated by the affordance decoder (red).

on action a, the trajectory part creates trajectory positions for each joint of the robot $u_{0:T}$.

This structure is shown in Figure 3.3. The affordance encoder consists of convolutional and fully connected layers (blue), while the decoder has the inverse structure — it consists of fully connected layers followed by transposed convolutions (red). The yellow blocks represent the policy and trajectory generation parts. In addition to a state s, information about camera pose is added to the input of the policy. This allows the policy to estimate the 3D pose of the target object with respect to the camera and removes the need for the simulation viewpoint to exactly match the real one.

The gray background emphasizes the hourglass shape of the network with a low-dimensional information bottleneck in the middle. Both the perception and the trajectory blocks are built using variational encoder-decoder structures that were introduced in Section 2.4.1.

The novelty of our method is representing visual affordances in a low-dimensional representation space — no previous work has trained an affordance representation that could directly be utilized in robotic control tasks. This low-dimensional, disentangled affordance representation enables efficient policy learning by making the representation space faster to explore.

3.2.1 Affordance Representation Learning

The perception part represents observations in a low-dimensional affordance representation space. The representation space follows the reconstruction learning approach introduced in Section 2.4.1.

We include the affordance prior to the learning process. Similarly as in [Ghadirzadeh et al., 2017], we modify the decoding target. In our method, the decoding target is the corresponding pixel-level affordance labeling of the observation o. The same affordance image target has been utilized in several visual affordance learning approaches that were introduced in Section 3.1. However, their works have focused on learning to output affordance images. To encourage the representation space to preserve neighborhood and disentangled structure, we included the KL-divergence term as in VAE [Higgins et al., 2016]. We refer to our structure as a variational affordance encoder-decoder (VAED) structure.

The encoder of VAED consists of a series of convolutional layers followed by one fully connected layer. The mean and covariance are produced by separate fully connected layers. The latent representation, drawn from the normal distribution with given μ and σ, is then passed as an input to the decoder. The decoder has a reversed structure—a fully connected layer followed by a series of transposed convolution layers. The decoder produces a multi-channel probability map $\hat{y}$ with each channel corresponding to a specific affordance. A pixel value in an affordance channel k describes how likely that affordance is to occur in each pixel. The probability output is achieved by using the sigmoid activation function. As the reconstruction loss, we use binary cross entropy [Brink and Pendock, 1996]:

$$\mathcal{L}_{recon}(y, \hat{y}) \equiv H_y(\hat{y}) := \sum_{k}^{M} \Bigg(- \sum_{i} \big(y_i^k \log(\hat{y}_i^k) + (1 - y_i^k) \log(1 - \hat{y}_i^k) \big) \Bigg),$$

where y is the target affordance image, M is the number of affordance channels and i expresses the ith pixel index of an affordance image.

Training the affordance detection network requires corresponding labeled affordance images. The target, together with the RGB inputs, are generated in Blender, an open source 3D modeling and animation software [1]. We use domain randomization of textures, clutter and lighting to increase the robustness of the model.

[1]https://www.blender.org/

3.2.2 Trajectory generation

To build a trajectory generation model, we first train a variational autoencoder on a set of task-specific trajectories, similar to the approach in [Ghadirzadeh et al., 2017]. Their method was presented in Section 2.5.2. The decoder block of this autoencoder converts low-dimensional latent trajectory representations of actions a to their corresponding trajectories in joint space $u_{0:T}$. This model can therefore be used as a task-specific trajectory generator — it converts each latent action a to a trajectory that is useful for accomplishing the given task. To achieve this, we rely on the observation that trajectories, useful for a specific task, often exhibit structural similarities. For example, trajectories useful for pouring tea into a cup consist of motions to a point above the target location followed by a wrist rotation. We generate such trajectories using MoveIt!, an open source motion planning software [Coleman et al., 2014]. The encoder and the decoder of the trajectory VAE consist of three fully connected layers each. We use mean squared error (MSE) as the reconstruction loss.

3.2.3 Policy layer

The policy part maps the affordance information s to an action vector a. The policy part includes three fully connected layers. In addition to the affordance information s, camera parameters are included as input to the policy network. This allows the policy to account for the relative position of the camera with respect to the identified affordances. The policy part can be trained either in a supervised manner or by using reinforcement learning.

We utilize supervised learning to train the policy. This is a more convenient learning approach for feedforward control problems than reinforcement learning. The feedforward control problems do not require sequential decision making. Additionally, the supervised learning approach is more sample efficient in learning the policy than by the trial and error method used in reinforcement learning.

Chapter 4

Visuomotor Control Application

The software architecture of our system can be divided into three processes: (i) the agent process, (ii) the control process, (iii) the backend process. Figure 4.1 shows the overview of the architecture. The agent process produces the trajectory commands based on received images, and it manages the simulation environment. Trajectories generated by the agent are then executed by the control process which uses MoveIt! Commander and ROS Control [Chitta et al., 2017, 2012]. MoveIt! Commander provides an interface for several state-of-the-art motion planning algorithms, and sets the trajectory target the ROS Control. ROS Control operates the low-level control task. The backend process describes the robot environment. The environment is either the MuJoCo simulation environment [Todorov et al., 2012], or the real environment. The environment provides the robot hardware interface for control, and the camera interface for the agent.

We used the open source Robot Operating System (ROS) framework to build the system [Quigley et al., 2009]. ROS offers a middleware for the processes to communicate, and to share global state and configuration information, such as the current state of the robot joints. Additionally, ROS provides a wide variety of libraries and tools that are commonly needed in robotic software development. These are, for example, the standard message definitions for geometric concepts and for common sensor information, and the methods for describing robot models, estimating poses, and visualizing processes. Additionally, ROS provides services for controlling the actuators of joints [Chitta et al., 2017].

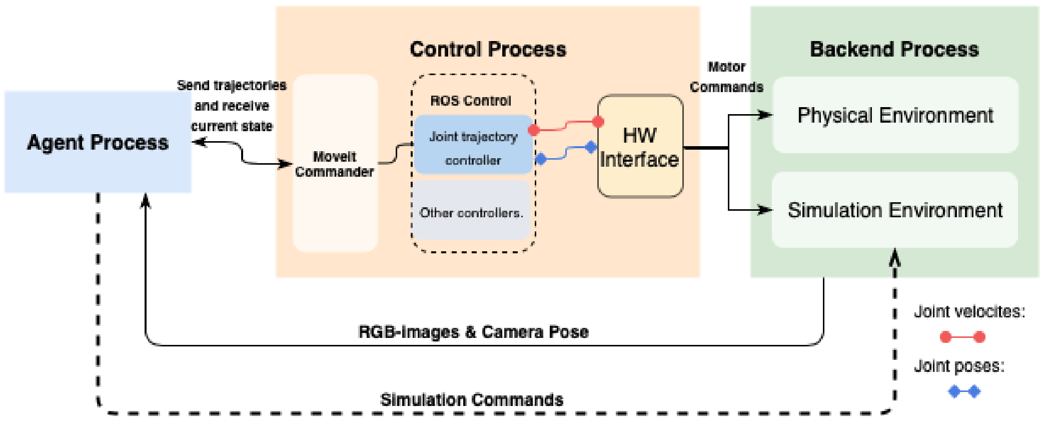

Figure 4.1: Our application architecture can be divided into three separate processes: the agent process, the control process, the backend process.

4.1 Hardware Setup

We conducted experiments on the Franka Panda robot shown in Figure 4.2. It is a 7 degrees of freedom industrial robot arm produced by Franka Emika. Franka Emika provides ROS configurations for the robot, which include, for example, the robot description, the model structure of the robot, and the hardware interface for the physical robot.

In the real environment, we used Kinect V1 for capturing RGB images. The resolution of the Kinect RGB camera is 640×480 pixels, and the field of view is 84.1×53.8 degrees [Wasenmüller and Stricker, 2016]. These same parameters were used in the simulation environment.

4.2 Agent Process

The agent process executes the modular neural network system part for end-to-end control shown in Figure 4.1. The neural network structure is built with PyTorch, an open source machine learning library [Paszke et al., 2017]. The agent process receives an RGB image of the environment from the backend process, and passes it to the perception part. The perception part produces a low-dimensional representation of the image. The camera pose, with respect to the robot, is received from the backend process and concatenated to the state representation. Based on the representation and the camera pose, the policy part produces the desired action, which is then used to generate the manipulator trajectory.

The trajectory is a $\tau \times n$ size vector, where τ is the number of steps in

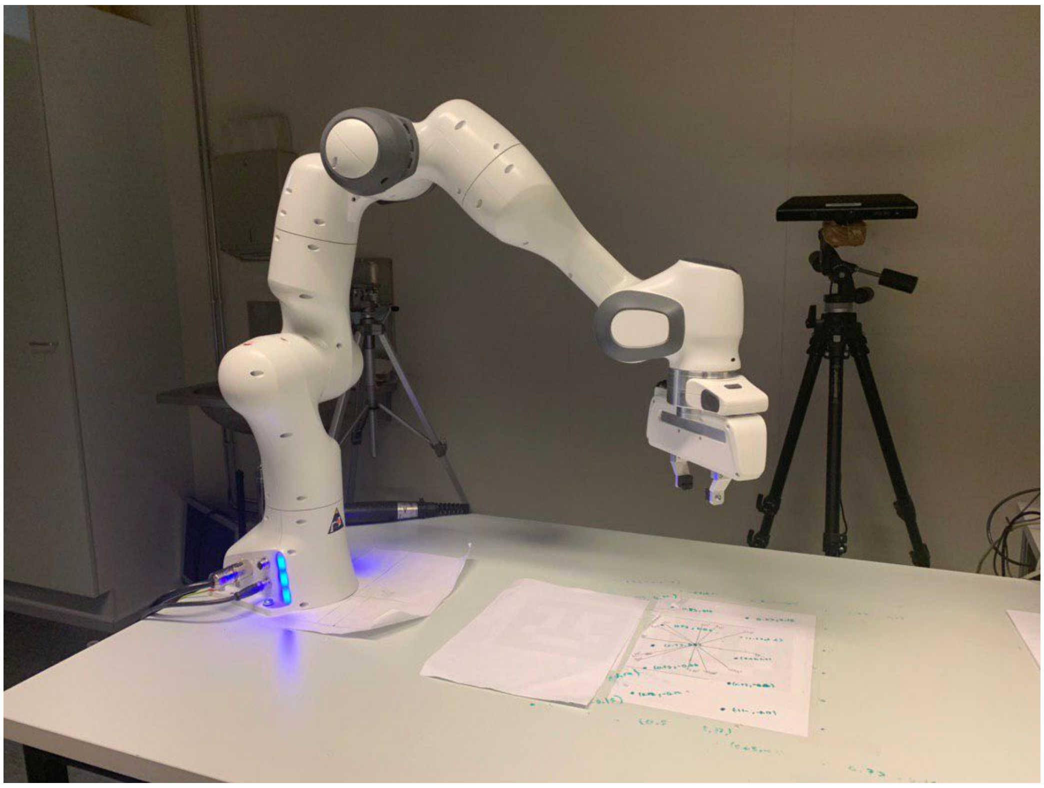

Figure 4.2: The Franka Panda robotic arm used for the experiment.

the trajectory, and n is the number of robot joints. ROS provides a standard message format for communicating the trajectories. The trajectory message of the description is created, and then sent to Moveit! Commander.

Additionally, when running in simulation, the agent process manages the simulation environment. The process changes the poses of the camera and the task objects, and resets the simulation to the initial setup after the episode.

For generating the training datasets for the trajectory and the policy parts, we use different agent processes than the one described above. For generating the trajectory dataset, the agent process requests the trajectory plans from Moveit! Commander. No ROS Control and simulation are needed for the planning task. For generating the policy dataset, the process determines the poses of the objects and camera. From the determined layout, the process captures the RGB image, and the corresponding trajectory target.

4.3 Control Process

In the robot task, Moveit! Commander receives the trajectory message from the agent process. Additionally, the agent process specifies the execution duration for the trajectory. The duration is evenly divided for the τ trajectory points. Moveit! Commander sends the trajectory and the trajectory duration to the joint trajectory controller that is operated by ROS Control [Chitta et al., 2017].

The controller executes the trajectory points in an iterative fashion — after the robot has reached the current target point, the joint trajectory controller transforms the next trajectory point to the motor commands. The communication between the controller and the robot is operated by the abstract hardware interface class provided by ROS Control [Chitta et al., 2017]. The abstract hardware interface receives the motor commands from the state trajectory controller, and sends them to the robot actuators. The interface also provides the updated information about the current robot state to the controller.

The physical robot has its own internal controller that transforms the received target joint poses to the torque commands. Here, the joint trajectory controller directly sends the joint poses and their duration to the robot via the abstract hardware interface [Chitta et al., 2017].

In the MuJoCo simulation, the robot actuators receive angular velocities which are then used to calculate the torque commands required for accurate trajectory following [Todorov et al., 2012]. Here, the joint trajectory controller sends the angular velocities to the abstract hardware interface, which further sends them to the simulated robot. The joint trajectory controller transforms the joint poses to the angular velocity commands with a PID controller [Chitta et al., 2017].

4.4 Backend Process

The backend process describes the working environment. The process provides the abstract hardware interface for the joint trajectory controller. This interface sends the motor commands to the robot and then updates the robot state values.

Additionally, the backend process sends the RGB images of the environment and the poses of the camera to the agent process. This communication is handled with a ROS topic [Quigley et al., 2009].

We utilize the OpenNI ROS package for capturing and broadcasting the images from the camera. To compute the camera pose, we place an Aruco marker [Garrido-Jurado et al., 2014] next to the robot on the table. The camera position, with respect to the marker, is computed by identifying the location and size of the marker in RGB images of the environment. The regular and distinctive patterns of the marker make it easy to identify and to estimate its pose. This computation is performed using the Aruco ROS package. Then, with the known relative position of the marker and the robot, we compute the camera position with respect to the robot.

In the MuJoCo environment, the images are rendered with Open Graphics Library. The backend process of the simulation environment changes the positions of the objects and camera based on the requests of the agent process. This communication is managed by a ROS Service [Quigley et al., 2009].

Chapter 5

Experiments and Results

The goal of the experimental evaluation was to study if the low-dimensional affordance representation space can be used to successfully train policies for visuomotor robot control. We first evaluated if affordances can be accurately represented in a low-dimensional space using the UMD dataset [Myers et al., 2015], a standard benchmark for the visual affordance learning task.

For holistically studying the approach, we developed an experimental setup presented in Figure 5.1. In this setup, the task of the Franka Panda robotic arm was to insert a ball into a container that is located on the table, with the scene observed by an RGB-camera. We used the setup to evaluate the performance quantified as position accuracy and task success both in simulation and on physical hardware.

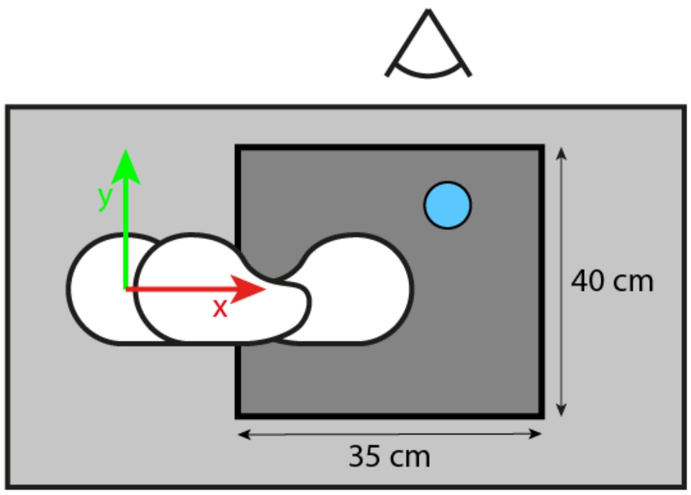

Figure 5.1: Experimental setup, as viewed from top. The robot is marked in white. The dark gray area shows the workspace and the blue circle an example location of a target container. The camera is located on the left side from the robot's perspective.

5.1 UMD Dataset Benchmark

We first evaluated the capabilities of VAED to detect object affordances on the UMD dataset [Myers et al., 2015]. The dataset was randomly divided into training (70%) and validation (30%) data. For this evaluation, we used a 20-dimensional representation space and KL-divergence penalty $\beta = 4$. We used F_β^w as the evaluation metric [Margolin et al., 2014]. The results are presented in Table 5.1 and are compared to the CNN-RGB and CNN-RGBD networks introduced in [Nguyen et al., 2016].

We can notice that for two out of seven affordances — *contain* and *wrap-grasp* — our model slightly outperformed the baseline, while for the other affordances, the results were the same or slightly worse. Comparing the RGB and RGB-D results, we can also see that discarding the depth information does not have much impact on the performance.

This evaluation shows that the variational encoder-decoder structure using RGB images can be used for the affordance segmentation problem.

Affordance	VAED		Nguyen et al.	
	RGB	RGB-D	RGB	RGB-D
grasp	0.667	0.643	**0.719**	0.714
cut	0.720	0.654	**0.737**	0.723
scoop	**0.760**	0.758	0.744	0.757
contain	**0.859**	0.858	0.817	0.819
pound	0.757	0.748	0.794	**0.806**
support	0.792	0.787	0.780	**0.803**
wrap-grasp	0.774	**0.777**	0.769	0.767
Average	0.761	0.747	0.766	**0.770**

Table 5.1: Results of the UMD dataset evaluation

5.2 Learning Affordance Representation

We used Blender's Python API to generate a domain randomized dataset of 1 million images. The development of this dataset generation program was a separate work from this thesis. Example samples from the generated dataset are shown in Figure 5.2. The Blender environment was constructed similarly to the experimental setup: it included a table with a cup and clutter objects, and the table was surrounded by walls. Each sample includes a rendered RGB image and the corresponding affordance image, which was used as the target output during training.

A simple cylinder model was built in Blender to represent the cup-like object. Its shape was randomly generated by smoothly changing its diameter along the height. Textures of the inner and outer parts of the object were separately randomized. In our setting, two different affordances occur: the outer part has the *wrap-grasp* affordance and the inner part has the *contain* affordance. The clutter objects were located on the table and their affordances were ignored. In total, 66 clutter objects from the Yale-CMU-Berkeley (YCB) Object and Model set were used [Calli et al., 2015].

The following features were randomized uniformly within reasonable limits: (i) positions, scales, and textures of the clutter objects and the cup, (ii) shape of the cup, (iii) texture and scale of the table, (iv) camera pose, (v) the number of clutter objects on the table, and (vi) the number and positions of lights in the scene. For the textures, we built a Blender Node structure that generated random textures based on randomly sampled parameters. The random textures were combination of two different textures. Based on the sampled parameters, the structure generated a new texture that was modification of the two textures in terms of scales, colors, rotations, and the level

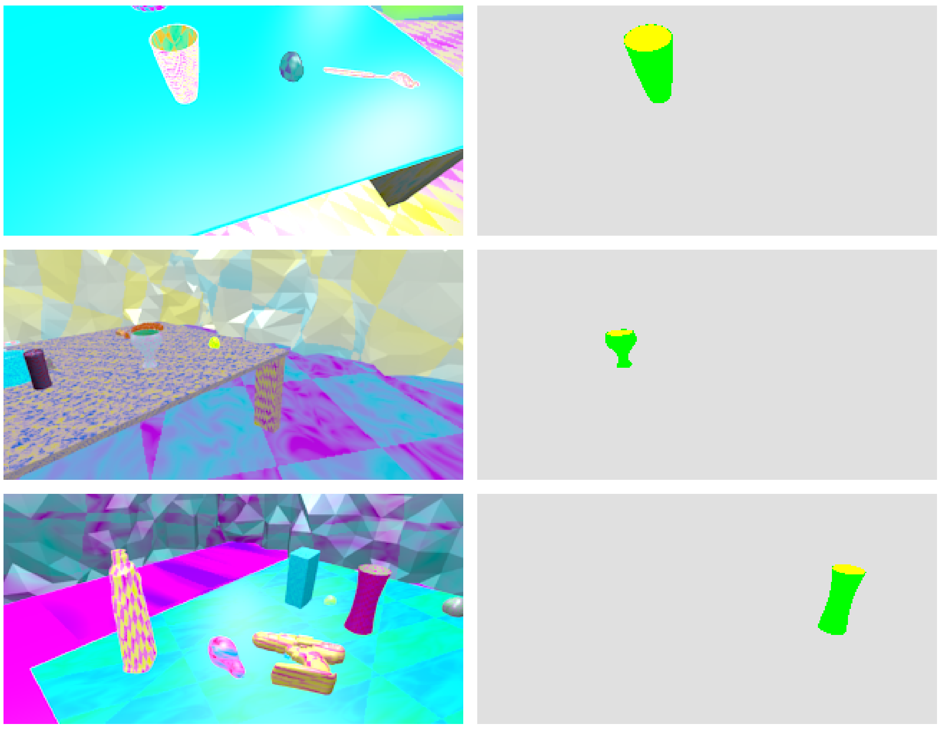

Figure 5.2: Samples from the affordance dataset. Green indicates the *wrap-grasp* affordance and yellow indicates the *contain* affordance. Gray pixels mean no affordance at all.

of noise.

The affordance representation network had four convolutional and corresponding transposed layers. We used KL-divergence penalty $\beta = 4$ and a 10-dimensional representation space throughout the experiments. The optimizer algorithm was the Adam optimizer [Kingma and Ba, 2014].

Only RGB data was used as the input of the model. We first included the depth information to the input of the model, which slightly improved the affordance detection in the training phase. However, the model performed poorly with RGB-D images captured by Kinect V1, as these depth images were of poor quality —the depth information was only partly captured, and it included a lot of noise. Figure 5.3 demonstrates the difference between these depth images. The model was not robust to these inaccuracies, as the training data only included precise depth images from Blender.

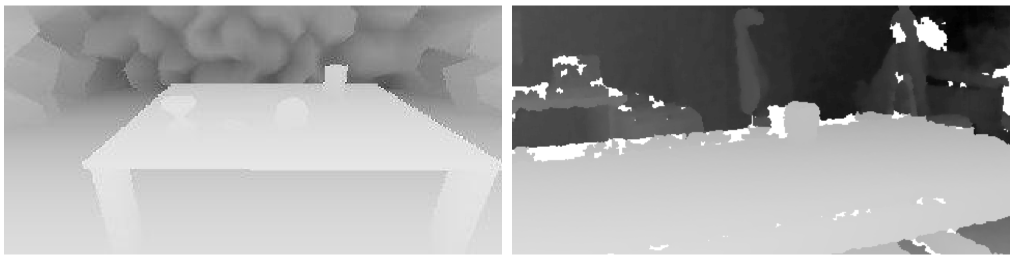
(a) (b)

Figure 5.3: Depth samples from Blender (a) and Kinect V1 (b).

5.3 Learning Trajectory Representation

Training data for the trajectory model included evenly distributed samples of the trajectories that moved the end-effector on top of the workspace region. Samples of the trajectory dataset were generated with the rapidly exploring random tree star (RRT*) planning algorithm using MoveIt! Commander. The initial pose of the robot in the training dataset was the same as in the real environment.

RRT* is a variant of a rapidly exploring random tree (RRT) algorithm [Karaman and Frazzoli, 2011], which searches for trajectories to target positions by randomly exploring the joint space of the robot arm [Lavalle, 1998]. The RRT* algorithm converges towards an optimal solution [Karaman and Frazzoli, 2011]. In other words, with sufficient time, it finds trajectories whose lengths are close to the shortest possible.

Multiple trajectories exist which move the end-effector to the same end position. Here, if trajectories are randomly generated, learning the low-level structure of the trajectories is difficult or impossible, because of the inconsistency and randomness of the trajectories. We found that the trajectory generation model learns a more accurate low-dimensional trajectory representation with training data, generated by RRT* instead of RRT. RRT* produces more consistent trajectories than RRT, as RRT* converges towards an optimal solution.

The encoder and the decoder of the trajectory VAE consist of three fully connected layers each. We used the Adam optimizer for updating the parameter values [Kingma and Ba, 2014]. The value of the β coefficient was increased while training such that it was initialized with $\beta = 10^{-8}$, and then β was iteratively increased in 400 epoch intervals towards $\beta = 10^{-5}$. By iteratively increasing β, the model learns first to reconstruct trajectories accurately without considering the smoothness and neighborhood preservation properties of the low-dimensional space. Without this, the model got stuck to local minima, where its trajectory reconstruction accuracy was poor.

We found that the KL-divergence penalty was essential in learning the trajectory representation. Without KL-divergence, the produced trajectories were not sufficiently smooth for the controller to execute them. We assume that the KL-divergence resulted in a more robust representation space of the model. Here, the generated trajectories were more consistent with the trajectories in the training dataset.

Trajectories were encoded to a 5-dimensional action space, and the number of steps in each trajectory was 24.

5.4 Policy Learning

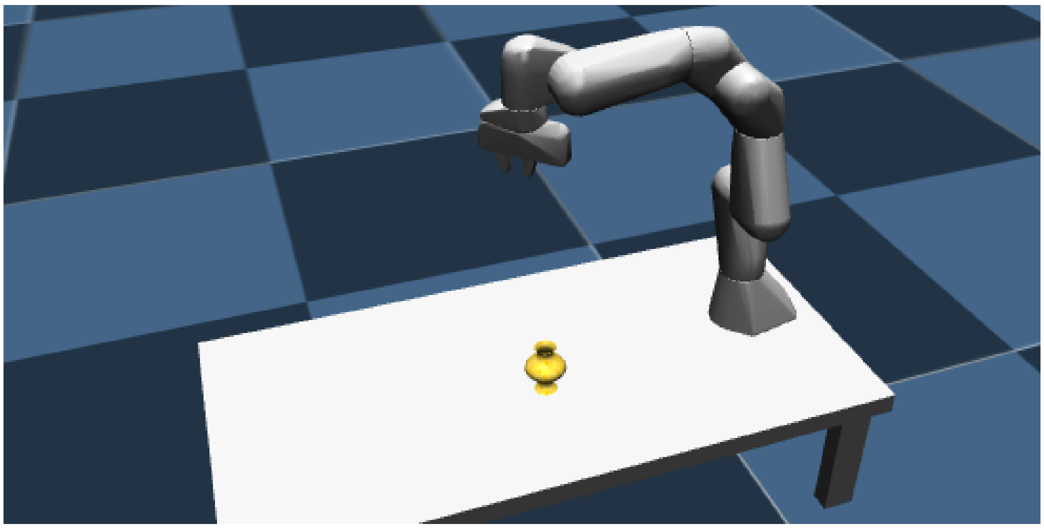
Figure 5.4: The policy training samples were generated in simulation. The relative position of the camera and the cup to the robot corresponded to the real environment.

The policy training setup was constructed in simulation mimicking the physical setup. Figure 5.4 shows the simulation environment.

In order to train the policy, the affordance perception, policy and trajectory generation parts were combined into a single network. The learned parameters of the affordance and trajectory models were not updated during the policy training phase. The policy network consists of three fully connected layers. Like previously, the weights were updated using the Adam optimizer.

The target of the policy was to determine the low-dimensional trajectory representation that generates the trajectory whose final point lies above the cup. The training loss was calculated as the mean squared error between the target end-effector position and the actual final position.

Since the trajectory is expressed in the joint space, the forward kinematics solution for the final point u_T was computed. In order to backpropagate the error through this computation, the base-to-hand transformation matrix — which depends on the joint configuration — was expressed as a PyTorch computation graph.

As the affordance representation is different for various cup shapes, positions, and camera angles, we used 15 different cup shapes and randomized the camera and cup positions. In total, our training set contained 3 million image and end-effector position target pairs. The average distance error was $0.9cm$ with validation data.

5.5 End-to-end experiments

We evaluated the system with and without clutter in terms of detection accuracy and task success rate. The detection accuracy expresses the position error of the end-effector againts manually labeled groundtruth, and the detection accuracy respectively describes the success rate of placing the ball inside the container.

5.5.1 No clutter

Without clutter, 13 different cup like objects shown in Figure 5.5 were used. Each of the objects was placed in 10 to 12 positions, with a total number of 143 trials. The diameter of the ball was 4 cm.

The position errors and success rates are reported in Table 5.2. The average error over all objects was 2.2 cm with three objects having over 3 cm average error. The success rate was 100% for 8/13 containers and 73–91% for the rest. The accuracy can be considered good, taking into account that the system has been trained to detect any general cylindrical containers.

The variation in performance between objects seems to be explained by the appearance of each object: the objects with the highest average distance error—yellow and jar—differ significantly from those in the training set. The yellow cup, for example, is smaller than the ones in training set, with a height of 5.7 cm. The system therefore expects the cup to be taller, and hence there is systematic error along the y axis. The object labeled as jar is a large aluminum container with a mirror texture on the outside. Even though the

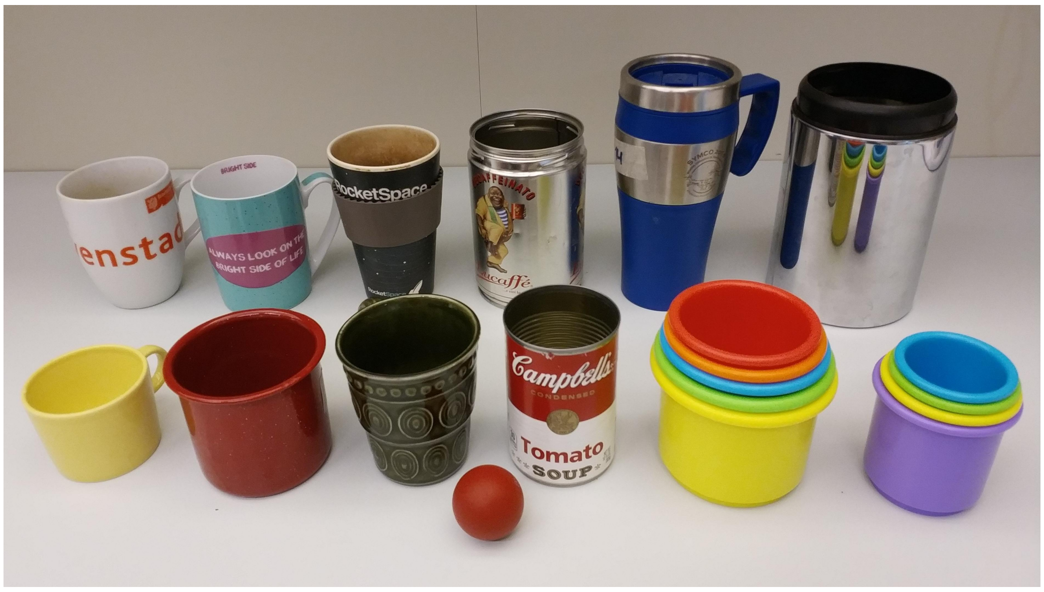

Figure 5.5: Various cups and the ball used throughout the experiments. Left to right, back row: white, pastel, rocket, can2, blue, jar. Front row: yellow, red, green, can, stack1, stack3. stack2 is composed from two bottom layers of stack 1, i.e. the large yellow and green parts.

training data did not contain any reflective objects, the system still managed to detect the opening and successfully place the ball inside of it 82% of the times.

To study the position dependency of error, all cup positions used for the experiments, together with their average error ellipses are shown in Figure 5.6. There are no consistent differences in accuracy across positions. However, we can see that the system performs more accurately along the x-axis than the y-axis. The higher error along y can be explained by the lack of depth information in the system and by the camera placement. While the position of the object along the x-axis can easily be deduced from the input image, calculating the y-position without knowledge of depth or object size is difficult.

Table 5.2: Experiment results for different positions in clutterless environment. All distances are given in centimeters.

Cup	Radius	Height	x error	y error	Error	Success
blue	3.45	16.60	0.73	2.48	2.68	82%
can	3.32	10.15	0.76	1.23	1.57	100%
can2	4.22	13.76	0.52	1.48	1.63	100%
green	3.92	9.33	0.74	1.00	1.35	100%
white	4.06	10.44	0.67	2.41	2.53	91%
jar	4.75	16.30	1.52	3.20	3.84	82%
red	4.30	8.08	0.59	1.60	1.76	100%
rocket	4.06	13.06	0.54	0.99	1.17	100%
stack1	3.70	10.52	0.52	1.99	2.07	100%
stack2	4.46	8.361	0.56	3.07	3.14	75%
stack3	2.72	8.76	0.51	1.15	1.31	100%
pastel	3.83	10.67	0.56	1.68	1.84	100%
yellow	4.06	5.69	0.95	3.45	3.69	73%
Average	3.91	10.90	0.71	1.97	2.19	92.5%

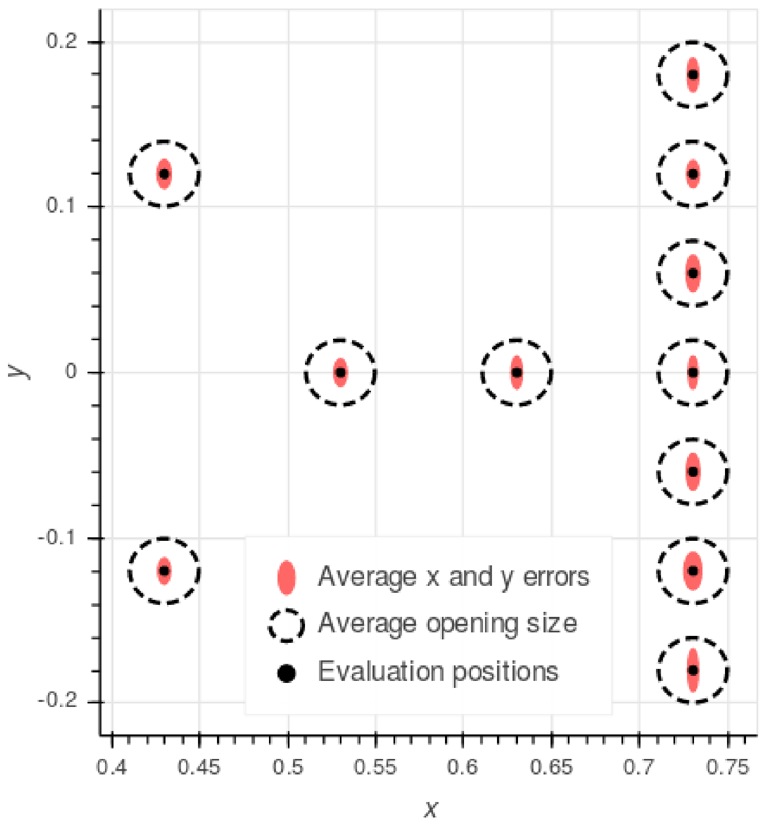

Figure 5.6: Evaluation positions with their respective average error ellipses marked.

5.5.2 Cluttered Scene

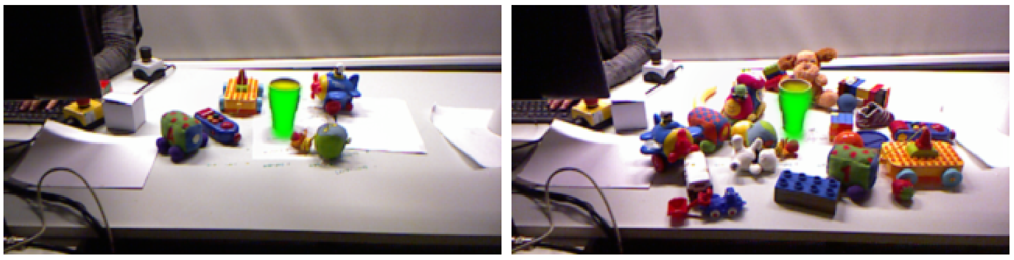
(a) (b)

Figure 5.7: Successful affordance detection with different amounts of clutter.

To evaluate the system's susceptibility to clutter, we chose three objects (rocket, red, can) from the ones used for previous experiments, providing a variety of different shapes, sizes and texture. The clutter objects were toys from a toy dataset[1], which were not seen during training. They were randomly shuffled on the table between each trial and their number was steadily increased from 0 to 25. The cup object position was fixed during these experiments.

Example scene images captured during these experiments are shown in Figure 5.7. The *wrap-grasp* affordance of the cup is visualized in green and the *contain* affordance in yellow. We can see that despite quite heavy clutter the cup was successfully identified in both situations.

Clutter objects	Rocket			Can			Red		
	d	%	n	d	%	n	d	%	n
0-6	1.25	97	45	1.64	92	28	2.02	94	69
7-12	1.86	96	30	2.93	76	26	2.4	82	64
13-18	2.84	100	6	4.47	73	30	4.76	63	36
19+	2.27	80	15	6.04	35	14	4.84	60	10

Table 5.3: Results of experiments in the cluttered environment. Columns d, % and n indicate, respectively, the average position error in cm, percentage of successful trials and the total number of trials.

The position errors and success rates are presented in Table 5.3. Not surprisingly, the errors increase and success rates decrease with increasing clutter, but the performance deterioration is gradual. Despite the training

[1]http://irobotics.aalto.fi/software-and-data/toy-dataset

data having only 0 to 10 clutter object on the table, our system proved to be resilient to a larger amount of clutter. This is most likely caused by applying heavy texture randomization when the training data was generated. The perception model thus learned to perceive clutter as just a different texture of the table - both of these look identical if there is no depth information available to the model.

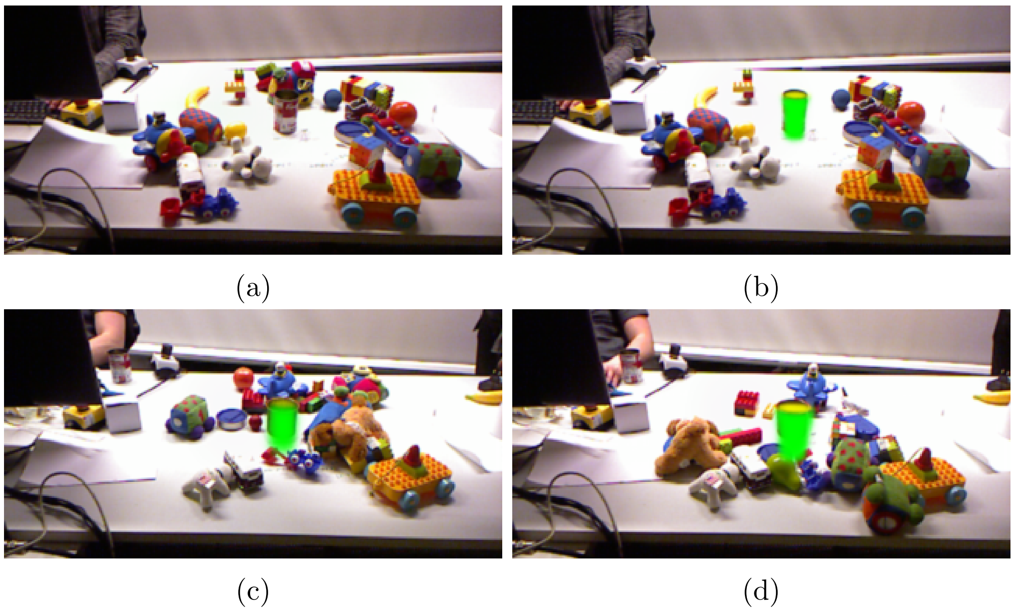
Figure 5.8: Examples of failure cases in different clutter configurations: the detection fails with clutter behind the object (a) and succeeds when the clutter is removed (b). With heavy clutter in front of the object its height is incorrectly estimated (c, d)

Noticing that the task can succeed even in heavily cluttered scenes, such as the one shown in Figure 5.7b, we further analyzed the data to determine failure cases. We found that the amount of clutter was not as important as its positioning—when clutter objects were located directly in front of the cup, covering its bottom part, the system quite often failed to correctly position the arm on the y axis. This is expected to happen—when the bottom part of the cup is not visible, it is impossible for the model to deduce its height, and thus estimate the y position. The situation is illustrated in Figures 5.8c and 5.8d.

Putting an object directly behind the cup opening also had a noticeable impact on performance, although the effect was less significant. In that case, the problems are likely to be caused by the perception model having problems estimating where the opening of the cup ends. It is, however, a difficult task to perform without depth data or stereovision. This is illustrated in Figure 5.8a, where a distractor object is placed directly behind the opening of a can. As the distractor and the inside of the can have a similar color, it becomes difficult to identify the opening in the image, and so the affordance part fails. When the distractors are removed from the back (Figure 5.8b), the object is successfully identified again.

5.6 Discussion

Section 5.1 showed that our variational affordance encoder-decoder (VAED) model can learn to identify different affordances from the RGB images, and represent them in a low-dimensional space. However, the model cannot work in a different domain, as the UMD dataset only includes samples from a single environment with a single viewpoint and no clutter objects. Additionally, the dataset was randomly divided into the training and the evaluation sets — the same objects appeared in both datasets, and only the orientations of the objects were different.

By introducing the affordance prior to the control task, and generating the domain randomized dataset, we produced the successful results presented in Section 5.5. The low-dimensional affordance representation space included no task-invariant information. The policy, trained in the simple simulation environment, learned to interpolate the affordance representations of the real observations.

The affordance images from VAED provide a solution to the need in transparency in robot control tasks, where the predictability and safety of the policy behavior are vital. In our experiments, the behavior of the policy was consistent with the affordance images from the VAED. When the affordance detection succeeded, we achieved desired performance of the policy. In turn, when the affordance detection failed, the policy failed. Even here, the failed trajectories moved the end-effector to the position that corresponded to the incorrectly detected affordance.

Learning the low-dimensional trajectory representation also improved the predictability of the system. By including KL-divergence to the learning phase, the trajectory part only generated trajectories similar to the ones appearing in the training data.

We observed that the VAED model failed when the cup was only partially shown due to the clutter objects — a scenario that was not included in the domain randomized affordance dataset used for training. In such situations, the detected affordances were inaccurately positioned with respect to the y-axis. Additionally, predicting the correct position and the shape is impossible when the relative depth distance between the clutter objects and the cup is unknown.

The results showed statistically significant success in the performance of the system in the non clutter experiment. The 95% confidence interval for successfully accomplishing the task was $93\% \pm 4\%$. The experiment included 13 different cup-like objects that covered a wide range of different textures and shapes, and were not seen in the training data.

For the cluttered experiment, the performance of the system showed promising results. With 7 to 12 cluttered objects on the table, the 95% confidence interval for successfully accomplishing the task was $96\% \pm 7\%$ for the rocket cup, $76\% \pm 16\%$ for the can, and $82\% \pm 9\%$ for the red cup. With a larger number of cluttered objects, the performance decreased. Nevertheless, with the rocket cup that more closely corresponded to the objects seen in training, the 95% confidence interval with over 19 clutter objects was $80\% \pm 20\%$. Despite large confidence intervals in the cluttered experiment, the results consistently showed over 50% success rate which can be considered favorable, given that the system was trained to detect any general cylindar containers.

The experimental task itself was simple which limits the generalization of the system performance to other control tasks. Nevertheless, we argue that the same affordance representation can be utilized in various robot control tasks, as the low-dimensional representation of the cup encodes its general affordance information. Additionally, our experiments showed that our VAED model ignored task-invariant changes in the real environment. We showed that learning the policy required only a simple replica of the real environment, and the dataset of the task-specific trajectories. However, the experimental task tolerated some level of inaccuracy, as a successful trajectory of the end-effector was only needed to end within, and not necessarily in the center of, the opening of a target in which the ball was placed. For many tasks, such as grasping, less inaccuracy is tolerated.

The performance of the system cannot be directly generalized for more complex affordance detection tasks where more target objects or affordances of multiple objects exist in the environment. Only one target object was placed on the table for each experiment, and we experimented the accuracy of the system solely with cup affordances. Nevertheless, we showed that low-dimensional representations of multiple object affordances can be learned with the UMD dataset. Additionally, previous works have shown that a single network can detect affordances of multiple objects in the same observation [Do et al., 2018].

Chapter 6

Conclusion and Future Work

The objective of this thesis was to study how to overcome the following limitations in applying deep data-driven models for robotics control: (i) the deep data-driven models, especially end-to-end trained models, require vast amounts of task-specific training data, and a lot of interactions with the physical robotic arm; (ii) if the task or the environment is changed, the model, trained with the domain- and task-specific data, needs to be trained from scratch with an entirely new dataset; (iii) the black-box nature of the deep models limits interpretability and predictability of the models.

We studied previous works that introduced methods to achieve more suitable learning processes for robotic control by decreasing the amount of training data required. However, these methods all obtained task- and domain-specific training data manually.

We demonstrated that a policy, trained in a simple simulation environment, can learn to generalize to the low-dimensional representations of the real observations. We achieved this by introducing an affordance prior to learn a low-dimensional affordance representation, and generating a domain randomized affordance dataset. The low-dimensional affordance representation space included no task-dependent information. With the domain randomized training data, the variational affordance encoder-decoder (VAED) model can be made invariant to various changes in the environment, such as adding distractor objects and changing textures. The entire system successfully performed the experiments in a zero-shot sim-to-real transfer manner.

We demonstrated that the behavior of the policy was consistent with the affordance images from the learned VAED model. This provides a solution for the transparency need of the deep data-driven approaches in robot control tasks.

Future work should be conducted on the generality of the low-dimensional affordance representation — whether a wider variety of affordances can be represented in an observation for robot control tasks. We showed that with the UMD dataset, our VAED model learned to represent multiple affordances in the low-dimensional space. However, the environment in the dataset is simple, and each of the samples includes only affordances of one object.

The variational encoder-decoder is a feedforward neural network that has its limitation to represent several affordance objects from an image in low-dimension. One approach to solve the described problem could be to structure the model as an object detector network. This architecture would produce a feature map that includes the separated semantic description of each of the detected objects, as in the approaches in [Devin et al., 2018, Do et al., 2018]. These semantic descriptions of the objects could be separately used as the input of a VAED model so that the affordance representation problem could be narrowed down. The affordances of only one object would be simultaneously represented in the low-dimensional space.

We argue that the performance of the system could be improved by introducing the depth information to be used by the VAED model. The challenge here is that the low-quality depth images captured by the real camera are far away from the rendered precise depth images. Figures 5.3a and 5.3b demonstrate the mismatch. This problem could be approached in both directions — either generating domain randomized depth images that include similar error structure as in the real depth images, or improving the quality of the real depth images to correspond to the ones generated by Blender. Generative adversarial networks (GANs) could help in transforming the precise depth images to match the real depth images [Goodfellow et al., 2014]. The quality of depth images for VAED could be improved with a depth in-painting method introduced in [Miao et al., 2012].

Finally, our system performs the task in a feedforward control manner. The system produces the entire trajectory based on a single observation, which limits the system's ability to adapt to changes in the environment while executing the trajectory. The low-dimensional affordance representation space could be utilized as the state space of feedback policies to learn, for example, closed-loop visual alignment tasks, or even active closed-loop search strategies for objects with particular affordances.

We believe that combining the generality of affordance information with modular architecture will bring us closer to creating autonomous systems with robust and reusable perception models. Such systems would be ready to operate in diverse and constantly changing environments that we humans are so used to in our every day lives.

Bibliography

Agrawal, P., Nair, A. V., Abbeel, P., Malik, J., and Levine, S. (2016). Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082.

Böhmer, W., Springenberg, J. T., Boedecker, J., Riedmiller, M., and Obermayer, K. (2015). Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations. *KI-Künstliche Intelligenz*, 29(4):353–362.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294.

Brink, A. and Pendock, N. (1996). Minimum cross-entropy threshold selection. *Pattern recognition*, 29(1):179–188.

Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., and Lerchner, A. (2018). Understanding disentangling in β-vae. *CoRR*, abs/1804.03599.

Calli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2015). Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics & Automation Magazine*, 22(3):36–52.

Chitta, S., Marder-Eppstein, E., Meeussen, W., Pradeep, V., Rodríguez Tsouroukdissian, A., Bohren, J., Coleman, D., Magyar, B., Raiola, G., Lüdtke, M., and Fernández Perdomo, E. (2017). ros_control: A generic and simple control framework for ros. *The Journal of Open Source Software*.

Chitta, S., Sucan, I., and Cousins, S. (2012). Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19.

Chuang, C.-Y., Li, J., Torralba, A., and Fidler, S. (2018). Learning to act properly: Predicting and explaining affordances from images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 975–983.

Coleman, D., Sucan, I., Chitta, S., and Correll, N. (2014). Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*.

Devin, C., Abbeel, P., Darrell, T., and Levine, S. (2018). Deep object-centric representations for generalizable robot learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7111–7118. IEEE.

Do, T.-T., Nguyen, A., and Reid, I. (2018). Affordancenet: An end-to-end deep learning approach for object affordance detection. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–5. IEEE.

Ecarlat, P., Cully, A., Maestre, C., and Doncieux, S. (2015). Learning a high diversity of object manipulations through an evolutionary-based babbling. In *Workshop on Learning Object Affordances, 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–2. IEEE.

Finn, C., Tan, X. Y., Duan, Y., Darrell, T., Levine, S., and Abbeel, P. (2016). Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., and Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292.

Ghadirzadeh, A. (2018). *Sensorimotor Robot Policy Training using Reinforcement Learning*. PhD thesis, KTH Royal Institute of Technology.

Ghadirzadeh, A., Maki, A., Kragic, D., and Björkman, M. (2017). Deep predictive policy training using reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2351–2358. IEEE.

Gibson, J. J. (2014). *The ecological approach to visual perception: classic edition*. Psychology Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Goroshin, R., Bruna, J., Tompson, J., Eigen, D., and LeCun, Y. (2015). Unsupervised learning of spatiotemporally coherent metrics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4086–4093. IEEE.

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

Hassanin, M., Khan, S., and Tahtali, M. (2018). Visual affordance and function understanding: A survey. *arXiv preprint arXiv:1807.06775*.

Higgins, I., Matthey, L., Glorot, X., Pal, A., Uria, B., Blundell, C., Mohamed, S., and Lerchner, A. (2016). Early visual concept learning with unsupervised deep learning. *CoRR*, abs/1606.05579.

James, S., Davison, A. J., and Johns, E. (2017). Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *1st Annual Conference on Robot Learning, CoRL 2017*, pages 334–343.

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

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

Lavalle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical report.

Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., and Filliat, D. (2018). State representation learning for control: An overview. *Neural Networks*, 108:379–0392.

Lesort, T., Seurin, M., Li, X., Rodríguez, N. D., and Filliat, D. (2017). Unsupervised state representation learning with robotic priors: a robustness benchmark. *arXiv preprint arXiv:1709.05185*.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436.

Luddecke, T. and Worgotter, F. (2017). Learning to segment affordances. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 769–776.

Margolin, R., Zelnik-Manor, L., and Tal, A. (2014). How to evaluate foreground maps? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE.

Mattner, J., Lange, S., and Riedmiller, M. (2012). Learn to swing up and balance a real pole based on raw visual input data. In *International Conference on Neural Information Processing*, pages 126–133. Springer.

Miao, D., Fu, J., Lu, Y., Li, S., and Chen, C. W. (2012). Texture-assisted kinect depth inpainting. In *2012 IEEE International Symposium on Circuits and Systems*, pages 604–607. IEEE.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Myers, A., Kanazawa, A., Fermuller, C., and Aloimonos, Y. (2014). Affordance of object parts from geometric features. In *Workshop on Vision Meets Cognition, 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 9. IEEE.

Myers, A., Teo, C. L., Fermüller, C., and Aloimonos, Y. (2015). Affordance detection of tool parts from geometric features. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1374–1381. IEEE.

Nguyen, A., Kanoulas, D., Caldwell, D. G., and Tsagarakis, N. G. (2016). Detecting object affordances with convolutional neural networks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2765–2770. IEEE.

Nguyen, A., Kanoulas, D., Caldwell, D. G., and Tsagarakis, N. G. (2017). Object-based affordances detection with convolutional neural networks and dense conditional random fields. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5908–5915. IEEE.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *Workshop on Open Source Software, 2009 IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, page 5. IEEE.

Roy, A. and Todorovic, S. (2016). A multi-scale cnn for affordance segmentation in rgb images. In *European conference on computer vision*, pages 186–201. Springer.

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

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

Srikantha, A. and Gall, J. (2016). Weakly supervised learning of affordances. *arXiv preprint arXiv:1605.02964*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE.

van Hoof, H., Chen, N., Karl, M., van der Smagt, P., and Peters, J. (2016). Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3928–3934. IEEE.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

Wasenmüller, O. and Stricker, D. (2016). Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In *Asian Conference on Computer Vision*, pages 34–45. Springer.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754.