NTNU
Norwegian University of
Science and Technology

# Deep Reinforcement Learning for Gripper Vector Estimation

Thomas Olsen
Birk Midtbø Ottesen

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Keith Downing, IDI
Co-supervisor: Ekrem Misimi, SINTEF Ocean AS

Norwegian University of Science and Technology
Department of Computer Science

# Abstract

The problem of gripper vector estimation, also referred to as gripper pose estimation, is the problem of constructing a vector describing the pose of the end-effector of a robotic gripper, which enables it to grasp an object. Recent work have investigated the use of Artificial Intelligence (AI) for constructing the gripper logic, where among others the use of Deep Learning (DL) and supervised learning has been applied. However, as supervised algorithms require large labeled datasets to be constructed, we propose the use of Deep Reinforcement Learning (DRL). The utilization of DRL mitigates the concerns regarding constructions of labeled datasets, but often requires more training and data than supervised algorithms. To ease these challenges, we propose the use of a simulated environment, where training of DRL algorithms can be conducted significantly faster than in the real world. It can be expected that a simulated environment will differ from the real world environment in many aspects, and thus, problems regarding transfer learning will arise. To increase the prospects of transfer learning, we propose the use of Domain Randomization (DR) in the simulated environment. To ensure good and descriptive information is available for the DRL agent, we propose the use of a state space consisting of color images combined with their respective depth images (RGB-D).

To investigate the prospects of DRL in the context of gripper pose estimation and dexterous robotic manipulation, a state-of-the-art literature review was conducted. The DRL algorithms Deep Deterministic Policy Gradients (DDPG) and Proximal Policy Optimization (PPO) were deemed promising and therefore investigated. The simulation environment constructed was realized using the Unity Game Engine, exploiting its newly released Machine Learning (ML) library which enabled communication with the TensorFlow library. To enable evaluations of the agents trained in simulation, a real world setup for the gripper pose estimation task was constructed. This setup was realized with the Panda robot and a two finger gripper, both made by Franka Emika, and the color and depth sensing camera, Intel Realsense SR300.

The main question this thesis addresses is whether a Deep Reinforcement Learning (DRL) agent, solely trained in simulation, can achieve satisfactory results for the problem of gripper pose estimation in a real world environment without any domain adaption or fine tuning. Our best DRL agent achieved a successful grasp prediction rate of 60% when evaluated for 60 gripper pose estimation attempts, and 88.3% of the grasp attempts were either successful or within one centimeter and five degrees from a valid grasp. Additionally, the mean positional and rotational offsets from a gripper pose that would have resulted in a valid grasp were respectively; 0.47 centimeters with a standard deviation of 0.75 centimeters and 0.6 degrees with a standard deviation of 2.1 degrees. Our main contributions to the field are; evaluations for specific application domains of the PPO DRL algorithm, additional evidence that DR positively impacts transfer learning when transferring from simulation to real world environments. Last but foremost, we have contributed to our field by demonstrating that an agent trained completely and solely in a simulation environment is able to perform successful grasping predictions for semi-compliant objects in the real world after transfer learning, without any domain adaptation.

Despite aspects of Domain Randomization (DR) being incorporated in our simulation environment, we observed that the Deep Reinforcement Learning (DRL) agents were sensitive to lighting conditions in our real world setup. In light of this, we suggest the inclusion of more expressive DR aspects regarding lighting conditions, in the simulation environment. The results obtained with a generic Franka gripper, not customized for semi-compliant objects, along with our general observations, lead us to strongly suspect that DRL agents only trained in simulation can produce satisfactory results in a real world environment for the problem of gripper pose estimation for semi-compliant objects.

# Sammendrag

Griper vektor estimerings problemet, også kalt griper poserings problemet, er problemet som omhandler konstruksjonen av en vektor som beskriver posisjonen og rotasjonen til ende-effektoren til en robot griper, der denne vektoren gjør roboten i stand til å gripe et objekt. Nyere forskning har undersøkt bruken av kunstig intelligens for å konstruere griper logikk, hvor blant annet bruken av dyp læring og ”supervised” læring har blitt testet. Ettersom ”supervised” lærings algoritmer trenger store mengder data med fasit inkludert, foreslår vi å bruke dyp forsterkende læring (DRL). Bruken av DRL eliminerer bekymringene rundt å konstruere store datasett med fasit, men trenger til gjengjeld mer trening enn ”supervised” algoritmer. For å takle denne utfordringen foreslår vi bruken av et simuleringsmiljø hvor trening av DRL algoritmer kan bli utført betydelig raskere enn i den virkelige verden. Man kan forvente at et simuleringsmiljø vil være ulikt det virkelige miljøet på flere områder, der dette medfører problemer når man skal overføre den lærte kunnskapen fra det ene miljøet til det andre. For å øke muligheten for en suksessfull overføring av kunnskap, foreslår vi å bruke domene randomisering (DR) i simuleringsmiljøet. For å sikre at god og informativ data er tilgjengelig for DRL agenten, foreslår vi bruken av et ”state space” bestående av farge bilder kombinert med deres respektive dybde bilder (RGB-D).

For å kunne utforske mulighetene for bruken av dyp forsterkende læring (DRL) i sammenheng med griper posering estimering og fingerferdig robot manipulering, ble gjennomlesning av ”state-of-the-art” litteratur utført. De to DRL algoritmene ”Deep Deterministic Policy Gradients (DDPG)” og ”Proximal Policy Optimization (PPO)” ble sett på som lovende og ble derfor utforsket. Det konstruerte simuleringsmiljøet ble utviklet med bruk av spillmotoren Unity, hvor det nylig publiserte maskin lærings biblioteket som muliggjør kommunisering med ”TensorFlow” ble brukt. For å muliggjøre evaluering av agenter som er trent i simulering, ble et oppsett for griper posering estimerings oppgaven satt opp i den virkelige verden. Dette oppsettet besto av Panda robot griperen produsert av Franka Emika og farge og dybde kameraet Intel Realsense SR300.

Hovedspørsmålet som vår oppgave adresserer er hvorvidt en DRL agent som bare er trent i simulering kan produsere tilstrekkelige resultater for griper posering estimering problemet i den virkelige verden uten å benytte seg av videre læring etter overføring. Vår beste DRL agent oppnådde 60% godkjente griper prediksjoner når den ble evaluert på 60 griper posering estimerings forsøk. I tillegg estimerte denne agenten griper poseringer der hvor 88.3% av forsøkene var enten vellykkede eller under 1 centimeter og 5 grader fra en vellykket estimering. I tillegg var gjennomsnittsavvikene i forhold til posisjon og rotasjon fra en vellykket griper posering estimering henholdsvis; 0.47 centimeter med et standard avvik på 0.75 centimeter og 0.6 grader med et standard avvik på 2.1 grader. Våre hovedbidrag til fagfeltet er følgende; evalueringer av spesifikke applikasjonsdomener for DRL algoritmen PPO, videre bevis på at DR har en positiv innvirkning på kunnskaps overføring når man overfører fra en simulering til den virkelige verden. Sist og viktigs, har vi bidratt til vårt fagfelt med å demonstrere at en agent som bare er trent i et simuleringsmiljø er i stand til å utføre vellykkede griper estimeringer for halv-myke objekter i den virkelige verden uten noe videre læring etter kunnskapsoverføring.

Til tross for at aspekter fra DR har blitt brukt i vårt simuleringsmiljø, observerte vi at DRL agentene var sensitive til forskjeller i lysforhold i vårt oppsett i den virkelige verden. Som en konsekvens av dette foreslår vi at det inkluderes sterkere DR aspekter når det kommer til lysforhold i simuleringsmiljøet. Resultatene vi oppnådde med en generisk Franka griper som ikke er spesialisert for griping av halv-myke objekter sammen med våre generelle observasjoner, gir oss sterk tro på at DRL agenter som bare er trent i simulering kan tilegne seg tilstrekkelige resultater i den virkelige verden for problemet som omhandler griper posering estimering for halv-myke objekter.

# Preface

This report was written by Thomas Olsen and Birk Midtbø Ottesen as a part of TDT4900 Computer Science, Master's Thesis at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU). We would like to thank Keith Downing for valuable guidance and the facilitation of rewarding gatherings with peers investigating similar areas. In addition, we would like to thank Ekrem Misimi at SINTEF Ocean for providing valuable insights, guidance and meticulous and invaluable feedback. Furthermore, we would like to thank Elling Ruud Øye at SINTEF Ocean for helping us create the castings of the silicone based objects and SINTEF Ocean for providing the equipment that made this work possible.

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

**A2C** Advantage Actor Critic. 19

**AC** Actor-Critic. 15, 19

**ACER** Actor Critic with Experience Replay. 19

**Adam** Adaptive Moment Estimation. 6, 15, 50

**AI** Artificial Intelligence. i, 1, 16

**ANN** Artificial Neural Network. 5, 6, 8, 11, 21, 22, 29

**ANNs** Artificial Neural Networks. xv, 1, 5, 6, 7, 8, 9, 17, 30

**CAS** Continuous Action Space. xv, 1, 2, 11, 12, 13, 17, 19, 39, 49, 75

**CEM** Cross Entropy Method. 22

**CL** Curriculum Learning. xi, xii, xix, xx, 17, 27, 28, 46, 55, 60, 61, 62, 63, 64, 67, 71, 72, 75, 78

**CNN** Convolutional Neural Network. xv, xvii, xviii, xix, 6, 8, 39, 40, 42, 44, 55, 56, 58, 59, 60, 61, 62, 63, 64, 67, 71, 73

**CNNs** Convolutional Neural Networks. xv, 8, 17, 44

**DAS** Discrete Action Space. 11, 12, 13

**DCNN** Deep Convolutional Neural Network. 21

**DCNNs** Deep Convolutional Neural Networks. 21

**DDPG** Deep Deterministic Policy Gradients. i, xi, xv, xvii, xviii, 13, 14, 19, 22, 28, 39, 49, 50, 52, 53, 54, 75

**DL** Deep Learning. i, 1, 2, 7, 9, 12, 16, 19, 21, 22, 28, 75

**DPG** Deterministic Policy Gradient. 19

**DPPT** Deep Predictive Policy Training. 22

**DQN** Deep Q Network. 19

**DR** Domain Randomization. i, xi, xii, xvi, xix, xx, 2, 3, 4, 16, 21, 22, 25, 27, 28, 34, 35, 46, 52, 53, 55, 58, 59, 60, 61, 62, 63, 64, 67, 71, 72, 73, 75, 78

**DRL** Deep Reinforcement Learning. i, xvi, xxi, 1, 2, 3, 4, 11, 12, 13, 15, 17, 19, 21, 22, 25, 26, 28, 29, 30, 31, 33, 37, 38, 39, 67, 75

**LRN** Local Response Normalization. xviii, 44

**LSTM** Long Short-Term Memory. 22

**ML** Machine Learning. i, 1, 6, 9, 27, 29, 30, 31, 39

**NAF** Normalized Advantage Functions. 22

**PPO** Proximal Policy Optimization. i, xi, xvii, xviii, xix, xx, 15, 19, 28, 39, 40, 42, 44, 49, 50, 52, 53, 54, 55, 56, 58, 59, 60, 61, 62, 63, 75

**ReLU** Rectified Linear Unit. 7

**REPS** Relative Entropy Policy Search. 22

**RL** Reinforcement Learning. xv, 1, 2, 9, 10, 11, 12, 17, 19, 22, 26

**RNN** Recurrent Neural Network. 6

**RNNs** Recurrent Neural Networks. 22, 78

**SGD** Stochastic Gradient Descent. 15, 19

**TRPO** Trust Region Policy Optimization. 15, 19, 22

**VPG** Vanilla Policy Gradient. 22

Chapter 1

# Introduction

## 1.1 Background

SINTEF Ocean is leading a research project called iProcess, which aims to develop novel concepts for flexible robot based automation in the food processing industry. One of their research areas focuses on vision-guided Machine Learning (ML) robots that can grasp and manipulate compliant food objects based on 3D images, where the use of Deep Reinforcement Learning (DRL) is of current interest.

The grasping and manipulation of objects in robotic applications is a complex task, where early systems required specialized algorithms and human expertise for the particular grasping problems. This in turn often lead to either highly specialized systems with low generality, or with systems of poor performance due to errors in sensors or insufficient modeling of the problem. However, recent developments in Artificial Intelligence (AI) using Deep Learning (DL) and supervised learning have accomplished good results on tasks like robotic manipulation (Kumra and Kanan, 2016; Levine et al., 2016; Johns et al., 2016) and image recognition (He et al., 2015; Krizhevsky et al., 2012). Supervised learning algorithms learn by being fed the correct output, often called a label, given an input. This imposes the challenge of collecting a labeled dataset that is sufficiently large and representative for the problem to be learned. The collecting of a labeled dataset can often be time consuming if the nature of the problem dictates the need of a human expert to label the data. The challenge of dataset generation is especially valid in the context of robotic learning and robotic gripper pose estimation. Consequently, learning methods which do not require labeled datasets are of particular interest.

Reinforcement Learning (RL) as opposed to supervised learning, does not require a labeled dataset. Instead these algorithms learn by exploring the problem space and observing the rewards received for their actions. RL thus requires rewards instead of labeled data to train, where the rewards are constructed based on a heuristic function relevant for the task. Historically, RL has not been well suited for real world problems, as early RL algorithms did not adapt well to problems with a Continuous Action Space (CAS) or with high dimensional state spaces. However, with the rise of DL, new RL methods using Artificial Neural Networks (ANNs) have been devised, creating the field of Deep Reinforcement Learning (DRL). DRL adapts more effectively to the problems present in the real world, where the methods can handle both CAS, and inputs of high dimensions. Despite DRL being a rather novel field, promising results have been achieved on challenging and complex problems including games (Mnih et al., 2013; Silver et al., 2017) and robotic control (Gu et al., 2016; Popov et al., 2017; Inoue et al., 2017; Ghadirzadeh et al., 2017; Koch et al., 2018).

In light of recent developments in the field of DRL, the investigation of such methods applied to the challenge of estimating gripper poses, for grasping of compliant objects, is of great interest for the iProcess project and for robotic handling of compliant objects in general.

## 1.2 Motivation

The current Reinforcement Learning (RL) algorithms face primarily two challenges. Firstly, RL performs poorly in domains with large or continuous action spaces, which makes it unfit for most real world problems. However, recent studies (Popov et al., 2017; Gu et al., 2016; Lillicrap et al., 2015; Mnih et al., 2013; Schulman et al., 2017; Ghadirzadeh et al., 2017; Inoue et al., 2017) have achieved promising results in such domains by combining RL and Deep Learning (DL), giving rise to Deep Reinforcement Learning (DRL). The emergence of DRL has led to renewed research in the training of such algorithms on real world challenges, including robotic appliances. From the utilization of DRL, good policies can be learned in problem domains with large state spaces, such as raw images and with Continuous Action Space (CAS), as demonstrated in the work of Ghadirzadeh et al. (2017), Schulman et al. (2017) and Pinto et al. (2017). In order to achieve good results on the challenge of gripper pose estimation for compliant objects, investigating state-of-the-art DRL methods in general and applied to similar challenges is of the essence.

Secondly, both RL and especially DRL require an extensive amount of training in order to learn viable policies and thus be able to perform well for the given problems. Both methods learn by exploring combinations of states and actions available, and inspecting the rewards achieved for these combinations. The number of possible combinations grows rapidly along with the growth of state and action spaces, and in problems with CAS the number of combinations are practically infinite. In real world applications such as robotic control where the execution of each combination is expensive in terms of runtime, the exploration rapidly becomes constrained.

Recent studies (Gu et al., 2016; Popov et al., 2017; Ghadirzadeh et al., 2017; Pinto et al., 2017) show that being able to simulate the environment and having an agent primarily to learn in the simulated environment reduces this drawback. The hyperparameter tuning can be performed more efficiently in a simulator and an agent can be trained in a simulation, providing a starting point for the real world challenge as a form of transfer learning.

The simulation of the environment also enables the agent to explore in a less constrained manner during the first stages of training, as no physical equipment is at risk. Simulations provide the possibility of high scalability and parallel exploration given sufficient computing power, and can thus greatly reduce the time needed to train a good policy. The investigation of simulation possibilities for the gripper pose estimation task is thus of great interest.

A common challenge in both DL and DRL methods is the generalization capabilities of the learner, where poor generalization often leads to poor performance in the case of error prone sensors or unseen variations of the environment. When using simulations, the simulated environment will always be an approximation of the real world environment. Specializing too much on this approximation may reduce the quality of the learned agent in respect to transfer learning, and methods mitigating this effect are consequently of interest. A common method for increasing the generalization of learning algorithms is to use Domain Randomization (DR), where alterations that should not change the output of the learner are applied to the environment. Some of the benefits of using DR on different, but related problems, can be seen in the work of Tobin et al. (2017a,b), Pinto et al. (2017) and Peng et al. (2017). In light of this, an interesting aspect to investigate is the effect of Domain Randomization (DR) in regard to transfer learning of the gripper pose estimation agent.

## 1.3 Problem Formulation

This work will focus on the investigation of viable Deep Reinforcement Learning (DRL) solutions for solving the problem of gripper pose estimation with regards to grasping semi-compliant objects based on visual input. The gripper pose estimation problem concerns estimations of feasible sets of positions and rotations for the end-effector of a robotic gripper, which enables it to successfully grasp an object. A good corpus of research regarding gripper pose estimation for rigid objects through the means of supervised learning exists, and promising results have been achieved (Kumra and Kanan, 2016; Levine et al., 2016). Training DRL agents to perform robotic manipulations based on high dimensional inputs such as images, is more novel, but promising results have been achieved as shown in the work of Ghadirzadeh et al. (2017) and Pinto et al. (2017). The estimation of gripper poses in appliances with compliant or soft objects is rather novel and introduces more levels of complexity, where successful gripper poses for rigid objects may no longer be sufficient due to deformations of the objects during grasping. The state space of the DRL agents will consist of visual inputs in the form of color and depth images, commonly referred to as RGB-D images, which will be captured by a camera mounted on the end-effector of a robotic gripper. The use of RGB-D images as a state representation also seems rather novel, where related work using only RGB or only depth images seem more common.

The methodology and the approach for this work is to perform a state-of-the-art literature review, select and implement two promising DRL algorithms, train them in a simulation environment and evaluate different variations of the most promising algorithm in a real world environment without any domain adaption. In order to increase the generalization capabilities of the algorithms, Domain Randomization (DR) is to be incorporated in the simulated environment and a frame work for facilitating transfer learning to the real world setup is to be constructed. The real world setup will contain molded silicone copies of the same objects that are to be used in simulation, as well as a robotic gripper with a wrist mounted depth sensing camera.

## 1.4 Goals and Research Questions

This thesis will have one primary goal:

*Investigate the feasibility of using Deep Reinforcement Learning (DRL) to solve the problem of gripper pose estimation for grasping of semi-compliant objects based on visual inputs in the form of RGB-D images, after only training in simulation.*

To achieve this goal, this work will also focus on three respective sub goals:

*Conducting a state-of-the-art literature review and selecting two promising state-of-the-art DRL algorithms for evaluation.*

*Developing a base simulation for the gripper environment which incorporates Domain Randomization (DR) aspects.*

*Constructing a framework which enables a DRL agent to obtain RGB-D images from a depth sensing camera and to perform robotic actions in a real world setup.*

Along with the goals of this thesis, three research questions regarding the gripper pose estimation problem for semi-compliant objects will be investigated:

**RQ1:**
*Can the benefits of Domain Randomization enable an agent solely trained in simulation to produce satisfactory results in a real world environment?*

**RQ2:**
*Is training in a simulation environment with rigid objects sufficient for developing policies for grasping semi-compliant objects in the real world?*

**RQ3:**
*Can the use of Curriculum Learning decrease the amount of training needed for developing good policies in a randomized simulation environment?*

Chapter 2

# Theory

This chapter will present and briefly cover relevant theory behind the work conducted.

## 2.1 RGB-D Images

An RGB-D image is the combination of a normal color image showing the different color intensities captured by the camera and the corresponding depth image. Depth images can vary in encoding, but similar for all types are that instead of the colors in the image reflecting the colors captured by the camera, the different colors encode the spatial distance from the given pixel to the surface covered by this pixel. A common encoding for these kinds of images are to use gray-scale values for the pixels, ranging either from black to white or white to black. An illustration of a depth image can be seen in Figure 2.1, where gray-scale and white to black encoding is used. From this illustration one can see that the cube is closer to the camera than the sphere as the pixels in the area covered by the cube are whiter.

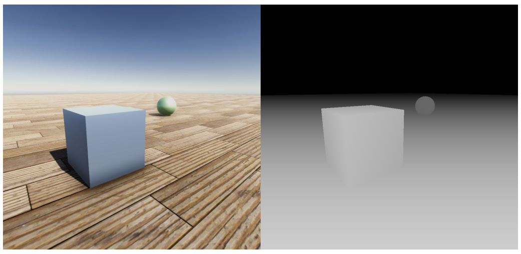


**Figure 2.1:** RGB (left) and Depth (right) images showing a blue cube and a green sphere resting on a wooden floor. White to black encoding is used on the depth image, resulting in close pixels being whiter and pixels further away being darker. In the depth image one can see that the cube is closer to the camera than the sphere, as the pixels covered by the cube are whiter than the pixels covered by the sphere. In addition, the depth values for the floor ranges from white to black as the surface of the floor gets further away.

## 2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by biological counterparts in humans. An ANN typically consists of five main components: nodes, weights, input, output and activation functions. The nodes in a neural network can be thought of as neurons in a brain, where information is altered and transmitted based on certain conditions. In ANNs these conditions are referred to as activation functions and the information relayed is encoded as numerical values. Activation functions can either transmit the input or a zero value based on a threshold, or perform some other linear or non-linear transformation on the input before transmitting. The weights in a neural network are the connections binding the network together, where each weight encodes a connection from one node to another. In addition to encoding connections, the weights contain a numerical scalar which is tuned during training and used to alter the information as it passes from one node to another. Input is the data sent in to the network and can be everything from pixels in an image to the sensor readings of a robot. The output of a neural network is what the network returns after running the input through its nodes, which are typically structured in layers. The output can, among other things, be encoded to represent a classification of an image or how much a robotic arm should rotate each joint. All these components together build an Artificial Neural Network (ANN), and ANNs have proven themselves to be powerful learners.

### 2.2.1 Network Structures

A common type of ANN is the feedforward fully connected neural network. In these networks the nodes and the weights form an acyclic directed graph, where each node in one layer is connected to all the nodes in the next layer and information moves steadily from the input to the output layer. In recent years several other versions of ANNs have been devised. Among these are the Recurrent Neural Network (RNN), which contain weights pointing to previous layers in the network, thus creating cyclic graphs enabling the network to learn features across multiple input sets and retain internal states. Another version of the ANN devised in recent years is the Convolutional Neural Network (CNN) which have shown promising results in the field of image recognition and pushed the state-of-the-art substantially. A CNN, further introduced in section 2.3, uses sparse connections, exploits spatial closeness and sharing of weights. This reduce the number of parameters to be tuned and consequently increases training speed.

### 2.2.2 Gradient Based Learners

Most Machine Learning (ML) algorithms use an optimization algorithm to minimize a given loss function. A subset of these optimization algorithms are gradient based, and work by calculating gradients relative to the loss function and nudge the trainable weights in the opposite direction of the gradient. These algorithms often use backpropagation to calculate the gradients of the loss function. Backpropagation is an algorithm for calculating partial derivatives, and in an ANN, it derives the gradients of the error with respect to each weight, $\frac{\partial E}{\partial w_{ij}}$, and feeds these to the optimization algorithm. The optimization algorithm then uses these gradients to change the respective weights in the network in order to minimize the loss function. A variety of different gradient based optimization algorithms have been proposed through the years. Some of the most popular optimization algorithms are AdaGrad, Adaptive Moment Estimation (Adam) and RMSProp (Géron, 2017).

### 2.2.3 Activation Functions

Artificial Neural Networks (ANNs) have evolved through the years, and with it, the choice of preferred activation functions. Activation functions are primarily used in hidden layers and on the output layer. The classical and most used activation functions early on were the sigmoid activation function (Equation 2.1), hyperbolic tangent activation function (Equation 2.2) and the identity activation function (Equation 2.3). Common for all activation functions are that they take a numerical value $x$ as input and produce a numerical value. The sigmoid activation function was very popular in the early research of neural networks, as it closely resembles how neurons in the brain activate. Additionally, the output of the sigmoid activation function is scaled between zero and one, making it ideal to model probabilities.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Another commonly used activation function in the early work was the hyperbolic tangent function, which outputs are scaled between minus one and one, and which holds the nice property of tanh of zero being zero. The hyperbolic tangent function also closely resembles the identity function for inputs near zero, which produces good gradients for such values.

$$Tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.2}$$

However, both the sigmoid and the hyperbolic tangent functions saturate at large positive and negative values. This becomes a problem when gradient based optimizers [2.2.2] are applied for tuning the weights in the network, as each layer's gradients are multiplied by its succeeding layer's gradients during training. This makes the gradients in the early layers very small, and consequently, the updates to the weights of these layers are minimal. This phenomenon is commonly known as the vanishing gradients problem. The identity activation function forwards the input $x$ to the output applying no changes and cannot be used to solve non-linear problems without the use of additional non-linear activation functions. However, the derivatives of the identity function are constant for all values, which result in good weight updates for all inputs.

$$Identity(x) = x \tag{2.3}$$

One of the factors contributing heavily to recent years development in Deep Learning (DL) is the use of the Rectified Linear Unit (ReLU) activation function. ReLU outputs the input $x$ when $x$ has a value greater than zero and outputs zero when $x$ is below or equal to zero, as shown in Equation 2.4. The ReLU activation function benefits from the derivative properties of the identity function, while also introducing non-linearity and sparsifying the activations. The use of ReLU as an activation function has enabled training of networks substantially deeper than could be achieved with other previous activation functions. However, activation functions like sigmoid and tanh are still used in the output and later hidden layers of networks, but ReLU and variations of ReLU are dominantly used in the hidden layers of deep networks. A plot of the values produced by the activation functions mentioned, can be seen in Figure 2.2.

$$ReLU(x) = max(0, x) \tag{2.4}$$

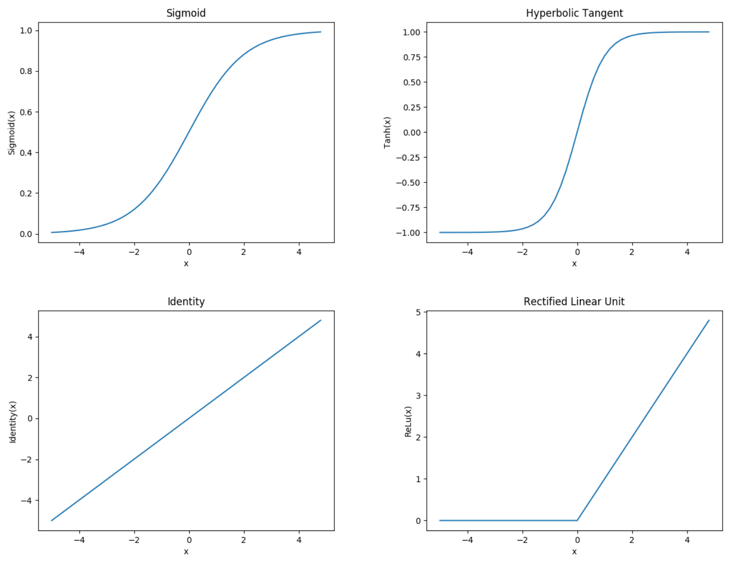


**Figure 2.2:** Four common activation functions used in Artificial Neural Networks (ANNs), respectively: Sigmoid, Hyperbolic Tangent, Identity and Rectified Linear Unit

## 2.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of an ANN which has been especially successful when applied to problems with raw images as input and have shown promising results in problem domains with inputs that embed spatial relationships, such as sound recordings and natural languages. CNNs utilize sparse connections, with weights connecting spatially close inputs to an output. These weights, referred to as a kernel, are shared across the processing of different inputs, rendering the CNN capable of learning features for multiple input locations without having to learn the feature in each location. The strides in a CNN are commonly represented as vectors, and describes the number of inputs to skip in each respective dimension of the input before applying another round of convolution with the kernels, producing an output value for the next layer. Another common component of a CNN is a pooling layer which, like the convolutional layer, also works by convolving over a sparse and spatially close set of the input with the stride parameter dictating how many inputs to skip in each dimension. Popular versions of pooling layers are max pooling and average pooling. Max pooling outputs the highest of the inputs and average pooling outputs the average of the inputs. Pooling layers are often combined with strides higher than one to decrease the dimensions of the next layer. A diagram showing an example of a CNN structure can be seen in Figure 2.3.

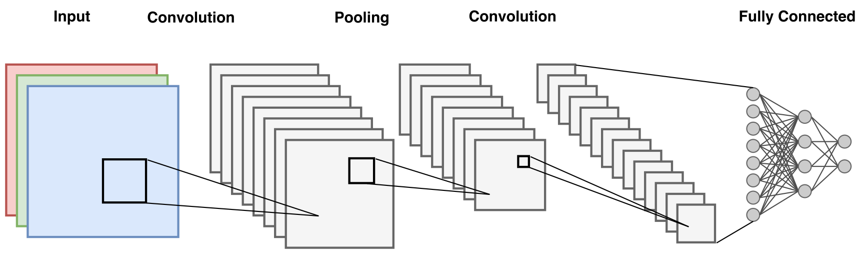


**Figure 2.3:** Visualization of a Convolutional Neural Network (CNN) architecture. As can be seen, convolutional layers often produce deeper feature spaces, while the pooling layers do not alter the depth of the features. Both convolution and pooling can reduce the dimension of the feature space depending on the strides used, however, dimensionality reduction is more common in pooling layers. As illustrated, CNNs are often combined with fully connected Artificial Neural Networks (ANNs).

## 2.4 Deep Learning

Conventionally the training of deep Artificial Neural Networks (ANNs) were problematic due to activation functions such as sigmoid and tanh being used in the hidden layers, causing the vanishing gradients problem [2.2.3]. However, with the introduction of the ReLU activation function, deeper networks could efficiently be trained with gradient descent methods, and impressive results on previous challenging tasks such as image classification were obtained. This introduced the concept of Deep Learning (DL), where the networks used were much deeper than previously. Recent studies (He et al., 2015) have also introduced a concept where shortcut connections between layers are used, called Residual layers or Residual networks, which in turn lead to successful training of even deeper networks, further increasing the potential of deep ANNs.

## 2.5 Reinforcement Learning

Reinforcement Learning (RL) is an area of Machine Learning (ML), where an agent strives to maximize its expected long-term reward. Available for such an agent is a set of observations and a set of possible interactions, often referred to as state space and action space (Géron, 2017). The concept of RL stems from behaviour psychology observed in intelligent beings, where they tend to behave in respect to what they have experienced as good or bad. RL tries to mimic this learning strategy by giving the agent feedback or a reward [2.5.2], based on its action in a given state. A simple diagram of a typical RL cycle can be seen in Figure 2.4. In the early training of an agent it is important to explore the given environment, typically by doing random actions. Over time the agent starts to utilize what it has learned by doing random actions, often referred to as exploitation. By exploring the environment and over time reducing exploration while increasing exploitation, the agent discovers the consequences of different state action combinations and learns how to maximize its long-term reward. As it is unknown when an agent has learned enough to predict the optimal action given the state observed, exploration versus exploitation is often a challenging aspect of RL. RL algorithms are commonly divided into two categories, respectively policy and value based algorithms. Policy based algorithms iteratively improves a policy based on a previously calculated value function and value based algorithms improve the value function directly for a given policy.

A common challenge regarding rewards in RL is the credit assignment problem. This is a challenge when rewards are sparse, as it is often intractable to uncover which actions that directly contributed to the received reward. In other words, it is unknown whether it was the most recent action or a number of preceding actions that led to a given reward. Due to this challenge, an RL agent often needs to heavily explore the environment and discover which action sequences, given a state sequence, that most effectively contribute towards a good expected long-term reward.

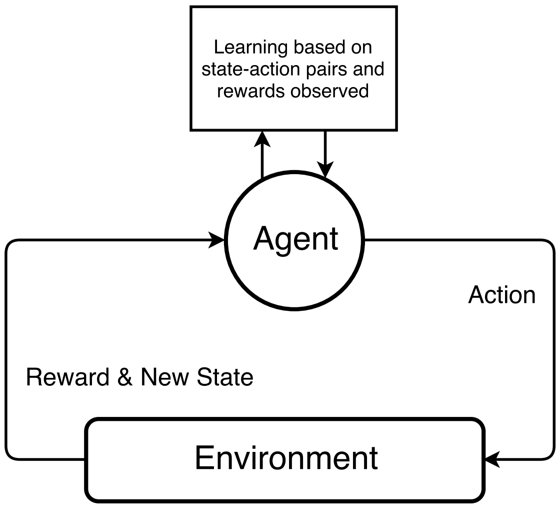


**Figure 2.4:** Illustration of a typical Reinforcement Learning (RL) cycle, where the agent observes the state, performs an action and observes a reward and a new state based on the action.

### 2.5.1 Environments

The environment includes all the aspects of the world the agent is to interact with. The environment can be either fully observable, where the agent has access to the full state of the environment, or partially observable, where only certain aspects of the environment can be perceived by the agent. Most real world environments are partially observable, where the agent perceives the world through sensors like cameras, GPS or motor readings, which are often inaccurate and prone to fault.

### 2.5.2 Rewards

The rewards given to an agent, based on its action in a given state, are often encoded as numerical values, where high positive values enforce good actions, and less positive or negative values punish inferior or poor actions. The design of the reward function or a critic is an important aspect of the design of an RL agent, and poor reward functions might lead to poor performance for the trained agent.

### 2.5.3 Exploration Noise

The choice of the exploration noise impacts both the training times and the results of the trained agent, but plays a bigger role in Continuous Action Space (CAS) than in Discrete Action Space (DAS) [2.6.1]. In DAS, the noise is typically used to choose an action from a defined set, whereas in CAS the noise is often added to the predicted action of the agent. In environments incorporating CAS, the use of a temporally correlated noise functions is often preferred over uniform noise functions. Nevertheless, performance of different noise functions may vary greatly depending on the nature of the problem.

An example of a temporally correlated noise function used in CAS, is the Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930), which will also be used in our work. A plot of the different values produced by this process can be seen in Figure 2.5.

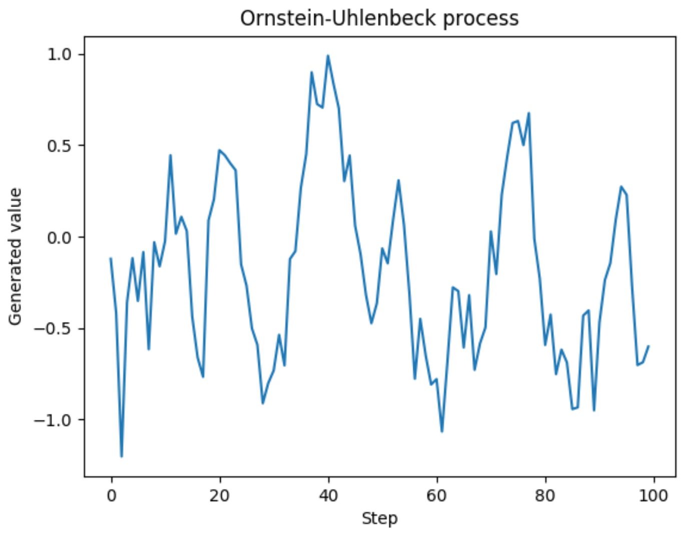


**Figure 2.5:** Temporally correlated noise values produced by the Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930) over 100 steps. Noise functions similar to this process are often preferred over uniform functions in Continuous Action Space (CAS) environments, where a common strategy is to add the noise to the actions, enabling more systematic exploration of the environment.

## 2.6 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) shares many of the same concepts behind RL, but uses an Artificial Neural Network (ANN) for predictions instead of methods like look-up tables.

### 2.6.1 Action Spaces

Action spaces in RL and DRL problems are commonly divided into two categories, respectively, Discrete Action Space (DAS) and Continuous Action Space (CAS). An example of a problem containing a DAS is the game of Pong, where given each input state the agent has to choose between three actions, either move up, down or stand still. An example of a problem containing a CAS is the problem of controlling the driving direction of a car, where given the state input, the agent is to predict continuous steering angles.

The algorithms concerning DAS work with a finite set of distinct actions and often converge and learn relatively fast as the number of possible actions are limited. The limited set of possible actions in DAS algorithms enables the exploration of the problem domain to be faster and more exhaustive. Nonetheless, algorithms in DAS often do not perform too well in problems with a large set of possible actions. The algorithms concerning DAS can also be used for problems with a CAS, in the example of the car control, each steering angle can be discretized into a set of actions with a set angle difference between them. However, this yields a challenge with discretization of the action space, and often leads to either low coverage of the action space or rapidly growing discrete action sets in relation to number of action parameters.

The algorithms concerning CAS often learn slower than the algorithms concerning DAS. These algorithms also rely more heavily on a good action noise [2.5.3] in order to properly explore the action space. However, many problem domains require algorithms which support continuous actions, as discretization of the action spaces is not feasible. The stability and ability to converge has long been a challenge in the development of CAS algorithms, but recent systems utilizing target networks [2.6.3] and a replay memory [2.6.2] have shown to help with these aspects in both CAS and DAS algorithms.

### 2.6.2 Replay Memory

A replay memory is often applied to DRL algorithms and increases the learning stability in both DAS and CAS algorithms. In Deep Learning (DL), most algorithms assume that the samples to learn from are independent and have a nearly uniform distribution. This is not the case in DRL, as samples are generated in a exploration sequence and thus training on each sample as they are gathered can lead to instability in the learning algorithms and possibly unwanted feedback loops. In order to mitigate this problem a replay memory or sometimes called an experience buffer is used, where the samples gathered through exploration are stored in a buffer and sampled during training. An example of how a replay memory can be used is to have a buffer of fixed size and have new experiences replace the oldest ones, and to sample a subset of these experiences during training. The utilization of a replay memory also enables more efficient use of hardware during training, as the agent can train on multiple samples, commonly referred to as a mini-batch, instead of a single sample at each training step.

### 2.6.3 Target Networks

Target networks can be applied to both DAS and CAS algorithms and have shown to reduce instability in the training of these. A target network is implemented by having a second network with the same architecture as the main network and making the target network either slowly update its weights towards the weights of the main network, or having it fixed and periodically setting its weights equal to the weights of the main network. During exploration the main network is used to decide actions, and in the learning phase the target network is used to evaluate losses.

### 2.6.4 Deep Deterministic Policy Gradients

Lillicrap et al. (2015) proposed a DRL algorithm, named Deep Deterministic Policy Gradients (DDPG), for use on continuous and high dimensional problems. Pseudocode for the DDPG algorithm can be seen in algorithm 1. In total of four networks are created; critic $Q(s, a|\theta^{Q})$, actor $\mu(s|\theta^{\mu})$ and their respective target networks $Q'$ and $\mu'$. The DDPG algorithm implements both a replay buffer and target networks in order to mitigate the problems of data correlation and non-stationary distributions. An important aspect of CAS algorithms is the action exploration noise [2.5.3]. In algorithm 1 the exploration noise is denoted by the random process $\mathcal{N}$, which was a Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930) in the original work of Lillicrap et al. (2015). For each step, exploration noise is added to the action from the actor network before being executed in the environment. After executing action $a_t$, the reward $r_t$ and new state $s_{t+1}$ is observed, and previous state, action, reward and new state is added to the replay buffer as an experience. Thereafter, a mini-batch, which is multiple experiences previously gathered by the agent, is randomly selected from the replay buffer. This mini-batch is used to update the critic network by minimizing the loss function seen in Equation 2.6. This loss function is the average squared difference between $y_i$ and the critic's estimated reward value for state $s_i$ and action $a_i$ for each sample in the mini-batch. The $y_i$ component, as seen in Equation 2.5, is an estimate of the long-term reward given state $s_i$ and action $a_i$, where the observed reward $r_i$ is added to the $\gamma$ discounted estimate produced by the target critic network applied to the observed next state $s_{i+1}$ and the action proposed by the target actor network given $s_{i+1}$. The actor network is updated by applying the sampled policy gradients from the aforementioned loss with respect to the action. As can be seen in Equation 2.7, the gradients for updating the actor network is computed by taking the average of the product of the gradients of the produced action $a$ with respect to the critic's value estimate and the actor's weight gradients produced in state $s_i$. At the end of each step the weights of the target networks are slowly updated towards the weights of the actor and critic networks based on a set value $\tau$. This is done for a set amount of steps $T$ for a set amount of episodes

$M$, unless an episode is terminated by, for example, reaching a goal state.

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{2.5}$$

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{2.6}$$

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} \tag{2.7}$$

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$
  and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1,T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and
          exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random mini-batch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:

            $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$

          Update the target networks:

            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$

            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$

    end
end
```

**Algorithm 1:** The Deep Deterministic Policy Gradients (DDPG) algorithm proposed by Lillicrap et al. (2015).

### 2.6.5 Proximal Policy Optimization Algorithms

In the work of Schulman et al. (2017), a new family of promising policy gradient based Deep Reinforcement Learning (DRL) algorithms were proposed, namely Proximal Policy Optimization (PPO). The motivation behind creating a new policy based DRL algorithm was to retain the data efficiency and stability of Trust Region Policy Optimization (TRPO) methods while keeping the algorithm simple using only first-order optimization. PPO alternates between sampling data from the current policy and optimizing this policy based on the data sampled through previous interactions with the environment. Pseudocode of an Actor-Critic (AC) version of the PPO algorithm is shown in algorithm 2.

```
for iteration=1,2,... do
    for actor=1,2,...,$N$ do
        Run policy $\pi_{\theta_{old}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
    end
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and mini-batch size $M \leq NT$
    $\theta_{old} \leftarrow \theta$
end
```

**Algorithm 2:** The Proximal Policy Optimization (PPO) algorithm Actor-Critic (AC) Style proposed by Schulman et al. (2017). Where the advantage estimate $\hat{A}_t$ is produced by the function seen in Equation 2.8 and the surrogate loss $L$ is the function seen in Equation 2.12.

For each iteration in algorithm 2, $N$ actors are collecting experiences in parallel for $T$ time steps by doing actions dictated by the policy $\pi_{\theta_{old}}$ and observing the resulting rewards and next states. After time step $T$, advantage estimates are calculated and used in combination with the data collected by the $N$ agents to construct the surrogate loss $L$. The advantage estimate $\hat{A}_t$ is calculated as can be seen in Equation 2.8, where the difference between the value estimated by the network for state $s_t$ and the sum of; the reward $r_t$, the $\gamma$ discounted rewards for step $t+1$ to step $t+T-1$ and the discounted value estimate produced by the network for step $t+T$ is calculated. The surrogate loss $L$ is optimized based on mini-batches with a gradient based learner [2.2.2], like Stochastic Gradient Descent (SGD) or Adaptive Moment Estimation (Adam), for $K$ epochs. The surrogate loss function found to produce the best results in the work of Schulman et al. (2017) was the clipped surrogate objective which can be seen in Equation 2.10, where $r_t(\theta)$ denotes the probability distribution seen in Equation 2.9, $\hat{A}_t$ is the advantage estimate computed for step $t$ seen in Equation 2.8 and $\epsilon$ is a hyper parameter dictating the clipping bounds.

$$\hat{A}_t = -V(s_t) + \gamma^T V(s_{t+T}) + \sum_{i=0}^{T-1} \gamma^i r_{t+i} \tag{2.8}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{2.9}$$

$$L_t^{CLIP}(\theta) = min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{2.10}$$

The clipped surrogate loss function depicts how much the new policy obtained can differ from the old policy, limiting the variation of the algorithm. Additionally, as the neural network architectures used with PPO shares parameters between the policy and the value function, a loss function combining the surrogate loss and the value function error is used. The value function error term $L_t^{VF}(\theta)$ is the squared error loss seen in Equation 2.11 with respect to the weights, where $V_\theta(s_t)$ is the current value estimate and $V_t^{targ}$ is the discounted rewards. A third term $S[\pi_\theta](s_t)$ was added to the surrogate loss, which depicts the changes to be applied to the weights of the random normal distributed exploration noise, based on the state $s_t$. This produced a combined surrogate loss $L$ which can be seen in Equation 2.12, where $\hat{\mathbb{E}}_t$ denotes the empirical average over a fixed batch, and this loss is used to train the network. The variables $c_1$ and $c_2$ of this equation are parameters, where $c_2$ can also be referred to as $\beta$. Where the $\beta$ parameter dictates the strength of the entropy regularization, in other words the balance between exploration and exploitation. In our implementation of the PPO algorithm, the entropy regularization loss will be the equation seen in Equation 2.13, which is used to calculate the gradients for the weights ($\sigma$) associated with the normal distribution exploration noise.

$$L_t^{VF}(\theta) = (V_\theta(s_t) - V_t^{targ})^2 \tag{2.11}$$

$$L_t(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{2.12}$$

$$S[\pi_\theta](s_t) = \frac{1}{2}\log(2\pi e^{\sigma+1}) \tag{2.13}$$

## 2.7 Domain Randomization

Domain Randomization (DR), sometimes referred to as perturbation, is a method commonly used in order to cope with the generalization problem of Deep Learning (DL) algorithms (Tobin et al., 2017a,b; Pinto et al., 2017; Peng et al., 2017). If a DL algorithm is to learn and generalize, it is important to have enough data to train on, and this data should cover enough of the state and action space. DR techniques in relation to the problem of gripper pose estimation can be to change textures, sizes, rotations or colors of objects to be grasped. Additionally, aspects relevant to the environment itself can be randomized, including, camera field of view, table texture and color, lighting, shadows and reflections. The general assumptions behind DR in regard to learning, is that by exposing the learning algorithm to enough variations of the task given, it will learn which aspects of the input that are most relevant to the solutions and enable it to be more invariant to changes that do not affect the problem. As an analogy, a person knows how to pick up a pink ball even though it has only seen balls of other colors.

## 2.8 Transfer Learning

Transfer learning is a challenge in Artificial Intelligence (AI) regarding the utilization of knowledge gained from solving previous problems on a new but often related challenge (Silver et al., 2012). The base assumption is that solutions to similar problems relies on common or similar features. The use of transfer learning seems to be a natural aspect of learning in biological systems. For example, it can be assumed that knowledge gained from learning to recognize cats can be applicable when learning to recognize other similar animals like dogs. In our work, transfer learning will pose a challenge when introducing an agent trained in a simulated environment to the real world version of the challenge.

## 2.9 Curriculum Learning

Curriculum Learning (CL) is a learning regime inspired by how humans and animals learn better when challenges are presented in an ordered manner, introducing more complex concepts gradually, instead of in a random order (Bengio et al., 2009). Typically, the difficulty of the challenges increase with each challenge as in the curriculum structure students face in school, enabling the learner to gradually extend its generality and comprehension of the challenge. In our work, curriculum learning will be investigated as a tool for decreasing the amount of training needed for learning policies in a randomized environment [2.7].

## 2.10 Robotic Gripper Pose Estimation

As mentioned in the introduction of this thesis, robotic gripper pose estimation concerns the estimation of both the positioning and the orientation of the end-effector of a robotic gripper. This estimate should ideally allow the gripper to perform some task in the operative environment. For our work, this task is to grasp a semi-compliant object that is within the reach of the robotic gripper. Conventionally robotic grippers are controlled by instructions dictating the different rotations for each joint in the robot, which can range from one to several joints. However, satisfactory algorithms exist for extracting these rotations from the pose of the end-effector.

## 2.11 Summary

This section has briefly covered several theoretical concepts relevant for our thesis, introducing the fundamentals along with more advanced concepts. As our work will focus on training an agent to estimate gripper poses based on RGB-D images, such images have been introduced and explained both textually and visually. The concepts of Deep Reinforcement Learning (DRL) along with conventional Reinforcement Learning (RL) environments have been described, introducing action spaces and observation spaces, where this work will concern Continuous Action Space (CAS). Different Artificial Neural Networks (ANNs) architectures have been introduced, covering both feedforward fully connected networks and Convolutional Neural Networks (CNNs) which will be used in our work. Two state-of-the-art DRL algorithms, which will be evaluated through experiments, have been presented, and a textual explanation of their process has been given along with pseudocode and corresponding equations. Finally, a brief description of the robotic gripper pose estimation challenge has been presented.

Chapter 3

# Related work

Several studies have been conducted regarding the estimation of gripper poses in specific, and robotic manipulation in general, through the means of Deep Learning (DL) and Deep Reinforcement Learning (DRL). The act of estimating gripper poses for semi-compliant objects using DRL and RGB-D visual input however, is novel. Most of the systems proposed in the studies found either; concern rigid-objects, use low dimensional state spaces or only use depth images as input. Additionally, related work using DRL seem more sparse compared to supervised approaches. Nevertheless, gripper estimation for rigid objects is considered highly relevant, and the different solutions applied to handle gripper pose estimations will thus be an important aspect of this section. A common challenge among these studies and along with our work, is the challenge of data collection in real world environments. Consequently, this section will also focus on the different methodologies applied to tackle the challenge of data collection. Specifically, different approaches for pre-training or solely training in a simulated environment before transfer to the real world will be of particular interest.

## 3.1 Deep Reinforcement Learning

In recent years the combination of Deep Learning (DL) and Reinforcement Learning (RL), namely Deep Reinforcement Learning (DRL), has increased the prospects of using RL on more complex problems. In the work of Mnih et al. (2013) a DRL method which learns control policies directly from raw pixels is presented. They propose a new DRL method based on the popular Q-learning algorithm (Watkins and Dayan, 1992), with Stochastic Gradient Descent (SGD) to update the weights. In order to mitigate the problems of data correlation and non-stationary distributions, the training distribution is smoothed through the means of an experience replay buffer [2.6.2], proposed by Lin, Long-Ji (1993). The method of Mnih et al. (2013) was named Deep Q Network (DQN) and the algorithm learned to play seven different Atari 2600 games implemented in The Arcade Learning Environment by Bellemare et al. (2012). The DQN algorithm performed better on the seven Atari 2600 games than any other previous methods and outperformed a human expert in three of them.

Lillicrap et al. (2015) researched the possibility of transferring the success of Mnih et al.'s DQN to the Continuous Action Space (CAS) [2.6.1]. They proposed an Actor-Critic (AC), model-free method, based on the Deterministic Policy Gradient (DPG) algorithm (Silver et al., 2014), which handles problems with CAS. Lillicrap et al. (2015) incorporated the ideas from DQN with training the agent off-policy with an experience replay buffer [2.6.2] and two target networks [2.6.3] along with batch normalization (Ioffe and Szegedy, 2015). The original DPG algorithm was more data efficient than stochastic AC methods and solved a challenging task which involved a multi-joint arm. However, the algorithm was used on toy-problems and did not take into account large high-dimensional observation spaces. Making the DPG algorithm applicable to these kinds of problems led to the development of the Deep Deterministic Policy Gradients (DDPG) algorithm. The DDPG algorithm by Lillicrap et al. (2015) solves more than 20 simulated physics tasks in the MuJoCo simulator (Todorov et al., 2012). Their method is able to find policies that are competitive with those found by planning algorithms with full access to the dynamics of the environment. Some of the tasks where the DDPG algorithm was able to learn policies from raw pixels are shown in Figure 3.1.

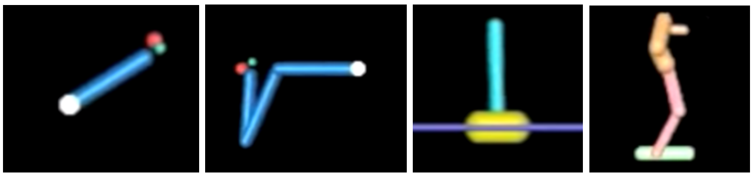


**Figure 3.1:** Four simulated physics tasks where the Deep Deterministic Policy Gradients (DDPG) algorithm of Lillicrap et al. (2015) is able to learn policies end-to-end from raw pixels. *Left to right: Singe joint reaching, Reaching fixed target, Cartpole swingup, Monoped fall and balance* (Lillicrap et al., 2015).

In the more recent work of Schulman et al. (2017) a new family of policy gradient methods for DRL is proposed. This new method alternates between executing environment interactions and optimizing a new objective function that enables multiple mini-batch updates, where standard policy gradient methods only allows one gradient update per experience. The proposed method of Schulman et al. (2017) is called Proximal Policy Optimization (PPO) and the results obtained indicate that it is easier to implement, has better sample complexity and is more general than other policy gradient methods. Results obtained by testing on several benchmarks, including Atari games and simulated physics tasks, show that the PPO algorithm for the most part outperforms other policy gradient methods in the CAS, such as Trust Region Policy Optimization (TRPO) and different versions of Advantage Actor Critic (A2C). Schulman et al. (2017) tested their algorithm on several tasks in the MuJoCo simulator environment and achieved at least as good and often better results than the other algorithms in the CAS. In the Atari games the PPO algorithm was evaluated against the A2C algorithm (Mnih et al., 2016) and the Actor Critic with Experience Replay (ACER) algorithm (Wang et al., 2016) by running all three algorithms on 49 games.

Their results showed that the PPO method learned games the fastest on most occasions, the ACER algorithm overall got the best final scores and the A2C algorithm was on one occasion fastest and the best.

In the work of Koch et al. (2018) different state-of-the-art RL algorithms were investigated, including DDPG, PPO and TRPO, on the problem of stability and control for an autopilot system. In their experiments the investigated algorithms were compared using various metrics, including; sample efficiency, training time, parameter tuning and final policy accuracy. In order to investigate these aspects, Koch et al. (2018) developed a high-fidelity simulator, incorporating CAS, where they trained and tested quadcopter attitude control agents using the different RL methods. The results of Koch et al. (2018) show that for the problem of quadcopter attitude control, PPO performed better than the other RL algorithms investigated on all occasions and even better than a fully trained Proportional-Integral-Derivative (PID) controller on nearly all metrics. These results further motivate us to evaluate the PPO algorithm on the problem of gripper pose estimation.

## 3.2 Dexterous Robotic Manipulation

In the early history of robot control and dexterous robotic manipulation, the systems applied tended to use manually crafted features extracted by specialized algorithms, and manually crafted algorithms for inferring actions based on these features. However, with the increasingly successful utilization of Deep Learning (DL) in the fields of image recognition, object detection and image segmentation (Krizhevsky et al., 2012; He et al., 2015), attempts have been made to apply DL techniques in robotics as well. This has yielded some success, as can be seen in the results of Kumra and Kanan (2016) and Ghadirzadeh et al. (2017). According to Levine et al. (2016), most successful applications of either DL or DRL have focused on producing relatively simple behaviour in relatively constrained environments. Many successful implementations of DRL in respect to robot manipulation, as in the work of (Gu et al., 2016; Popov et al., 2017; Inoue et al., 2017), also tend to restrain the state space, using low dimensional state spaces such as relative positions, joint rotations etc., as opposed to using raw camera inputs. Several of the implementations of DL algorithms using visual inputs for robotic problems also tend to use pre-trained feature extractors for processing of the visual inputs, such as the work of Ghadirzadeh et al. (2017) and Johns et al. (2016). In contrast to these, our work will focus on the use of raw image data as the input for the learning agent, which results in much larger state spaces but has the potential to be more general.

## 3.3 Deep Learning for Grasp Pose Estimation

The last couple of years many have applied DL to grasp pose estimation in various environments. Kumra and Kanan (2016) redefined the state-of-the-art in 2016 for robotic grasp detection with their multi-modal model which achieved 89.21% accuracy on the standard Cornel Grasp Dataset (CU, 2009). Their model consists of two Deep Convolutional Neural Networks (DCNNs), one for RGB image input and one for depth image input, which is then combined in a shallow fully connected network to predict the final grasp pose. Kumra and Kanan (2016) used a pre-trained ResNet, as proposed by He et al. (2015), for both their RGB DCNN and depth DCNN. During training the fully connected end part of the network was trained first, and then the whole network was trained.

Levine et al. (2016) describe a self learning based approach to gripper pose estimation with DL and large scale data collection. They had 6-14 real world robots collecting data with wrist mounted cameras throughout their experiment, learning hand-eye coordination for grasping. Levine et al. (2016) made a DCNN with many layers of convolution and pooling. In contrast to our work, this network was trained in a supervised manner, where large data collection phases were initiated in the real world and labeled based on the success of each grasp attempt. Their network could then be trained on the labeled dataset, enabling it to predict probabilities of a grasp being successful.

A challenge when making DL-based approaches to robotic grasping is generalization. Most state-of-the-art approaches in this domain are often trained on a low number of objects, which in turn makes generalization challenging. In the work of Tobin et al. (2017b) a method for data generation with the idea of applying Domain Randomization (DR) to the creation of objects is proposed. The proposed method procedurally generates millions of unique objects and trains a deep Artificial Neural Network (ANN) on estimating grasp poses for these objects. This resulted in a learner that generalized well and achieved 92% accuracy on grasp pose estimations for the objects present in the YCB dataset (Çalli et al., 2015), despite only training on randomly generated objects. Tobin et al. (2017b) identified three categories where the method had problems making a successful grasp: objects that were close to the max size of the gripper hand, curved objects that might cause a collision with the gripper hand on the widest area and highly irregular objects like a chain. For the method to be able to grasp such objects, the authors proposed to add more examples of the edge cases to the training set and aid the agent with visual servoing. The core idea behind the work of Tobin et al. (2017b) is to train a DL agent on different randomizations of the simulation environment to make it generalize better. In our work, similar techniques will be investigated, where randomizations of the scenery will be applied to change several aspects of the simulation environment.

## 3.4 Deep Reinforcement Learning in Robotic Manipulation and Grasping

In addition to DL, Deep Reinforcement Learning (DRL) has been applied to the problem of robotic grasping and robotic object manipulation. Gu et al. (2016) implemented the DDPG algorithm, the Normalized Advantage Functions (NAF) algorithm and a variant of the NAF algorithm called Linear NAF. They trained a robotic agent with the different algorithms in a simulated environment to achieve the task of reaching and opening a door and compared the success of each algorithm with regards to how fast they learned. The results of Gu et al. (2016) showed that for the task of reaching and opening a door, the NAF algorithm converged to a good solution faster than the DDPG algorithm, while Linear

NAF did not reach a good solution with the time frame given. In the simulation experiments both DDPG and NAF reached comparably good solutions. However, as the focus of the project was to find an algorithm well suited for learning from scratch in real world robotic appliances with respect to training times, the authors selected NAF to conduct further experiments with. Gu et al. (2016) evaluated and tuned the hyperparameters for their algorithms in a simulated environment in the simulation tool MuJoCo, made by Todorov et al. (2012), and then used the best hyperparameters found in the simulation phase to train the agent in the real world setup from scratch. The learning agents used relatively simple state spaces, consisting of joint angles and door handle positions, and their respective time derivatives. The ANN used in the agents was rather shallow consisting of two hidden layers with 100 nodes each, and thus training of the network weights could be conducted relatively fast. To increase the learning speed of the agent, Gu et al. (2016) expanded the NAF algorithm to handle asynchronous off-policy updates, where multiple robots gathered experiences in parallel. Their results showed that learning with two robots greatly outperformed learning with one robot for their problem, where the agent both learned faster and achieved better final solutions in the time frame given. Adding another robot, increasing the number of parallel workers from two to three, further increased this benefit, but the differences in final solutions and the speed of learning was smaller when going from two to three, than when going from one to two.

Popov et al. (2017) investigated the use of DRL on the problem of picking up and stacking bricks, with focus on data efficiency in the learning phase. They implemented the DDPG algorithm and extended it to handle multiple mini-batch updates per step, where in the original algorithm only one update is conducted per step. Additionally, they extended the DDPG algorithm to handle asynchronous updates to the agent. The experiments were conducted in a simulated environment using MuJoCo (Todorov et al., 2012). The state space in the experiments consisted of the angles and angular velocities of the joints and fingers, the position and orientation of the two bricks and the relative distance between the bricks and the pinch position of the gripper. To reduce training time and increase the probability of the agent finding a good solution, a form of apprentice learning was implemented, where the agent was trained on different initial positions sampled from a solution trajectory of the problem. Two different methods for initializing the starting states were used. The first method was initializing the gripper with the brick already picked up, as if it had previously conducted a successful grasp. The second method was initializing the gripper with different starting distances from the brick it should pick up, where the agent initially started close to the brick, and was then initialized iteratively further away from the goal position.

Ghadirzadeh et al. (2017) proposed a data-efficient Deep Predictive Policy Training (DPPT) framework with a deep ANN policy architecture, to solve the problem of skilled object grasping and ball throwing. The network responsible for converting image input to motor commands on the robot is conceptually divided into three sub-networks by the authors, respectively, the perception, policy and behaviour super layers. The training of the different super layers is conducted separately, where both the perception super layer and the behaviour super layer is trained prior to training the policy super layer. The perception super layer consists of an auto-encoder which maps an input image into a set of spatial feature points containing points of interest in image space. The behaviour super layer takes an action as input for the robot and uses an auto-encoder to map this action into a motor trajectory for the seven joints of the robotic arm. The policy super layer is more shallow than the other super layers, and has thus fewer parameters to be tuned by the learning algorithm. The policy layer was trained in a RL manner and different DRL algorithms were tested, including Vanilla Policy Gradient (VPG), Relative Entropy Policy Search (REPS), Cross Entropy Method (CEM) and Trust Region Policy Optimization (TRPO). However, training the policy layer with TRPO in simulations yielded the best results regarding convergence rates.

Inoue et al. (2017) investigated the use of DRL on the high precision task of assembling with robotic grippers. More precisely the problem investigated was that of guiding a peg through its designated hole. The learning agent contains two Recurrent Neural Networks (RNNs) with two Long Short-Term Memory (LSTM) layers each which outputs a preference regarding which action to choose. The two different networks are used in two different phases of the task, where the first phase is to search for the hole, and the second phase is inserting the peg into the hole. The two different phases have different state spaces and action spaces. Where the states in both tasks encode continuous valued readings of the forces and the moments on the peg, and in the phase of searching, the state also contains the position of the peg relative to the hole. The action space for both phases are heavily discretized, containing five different actions for the search phase and four different actions for the insertion phase. The learning of the agent for the two tasks is facilitated by the Q-learning algorithm, where the RNNs replace the q-value tables. Three different reward functions were used, depending on whether an episode was completed within the maximum steps allowed or not and which phase the agent was to complete. The results of Inoue et al. (2017) show that DRL can be used to successfully teach an agent to perform a tight clearance peg-in-hole task and that the proposed solution shows robustness against real world sensor errors.

The work of Pinto et al. (2017) investigated the use of a simulated environment for multiple robotic manipulation tasks, including picking and pushing of a cube. The simulated environment was realized in MuJoCo (Todorov et al., 2012) and enabled the training of DRL algorithms before transferring the obtained knowledge to a real world setup. They used the DDPG algorithm as the base of their DRL agent and proposed several extensions, including the use of asymmetric actor critic state spaces and hindsight experience replay (Andrychowicz et al., 2017). These extensions yielded better results in respect to the stability of the learning and the number of environment interactions needed. To improve the expected generalization capabilities of the trained algorithm regarding transfer learning [2.8], they extended the simulation environment to incorporate Domain Randomization (DR) aspects. Experiments investigating both the performance of an agent trained in the simulation incorporating DR and without were conducted, where the agent trained without DR did not produce satisfactory policies in the real world setup, even though obtaining near perfect performance in the simulated environment. The agent trained in the simulation containing aspects of DR however, produced good policies in the real world environment without training on any real world interactions.

## 3.5 Simulation of Environment

In order to mitigate the challenge of time constraints in experience gathering for DRL and to ease the safety concerns regarding exploration in the context of robot manipulation, robotic manipulation projects often turn to the utilization of simulators. Execution of actions on a real world robot can often take time in the order of seconds or even minutes, but in simulations the execution of these actions can often be carried out multiple times in a second depending on the complexity of the simulation. By conducting a preliminary training phase in the simulated environment of the problem, it is expected that the trained agent will have a good starting point with good prior assumptions when it is introduced to the real world environment. This effect is often prominent even when the simulated environment does not fully model the true environment or when the tasks trained on in the simulator differs on some aspects from the final tasks.

The method of first training an agent on a related problem before introducing it to another is often referred to as transfer learning [2.8]. Transfer learning in combination with Domain Randomization (DR) can be a powerful tool, where the work of Pinto et al. (2017) and Peng et al. (2017) show DRL algorithms that are able to perform complex tasks in real world environments after only training in simulators. Among the mentioned implementations for solving problems concerning robotic manipulations, quite a few have turned to simulators and achieved promising results with relatively low training times (Ghadirzadeh et al., 2017; Gu et al., 2016; Popov et al., 2017; Pinto et al., 2017). However, only a few of these incorporated transfer learning, while most of them used the simulations exclusively to tune their hyperparameters before training their agents in real world from scratch.

In simulations of robotic interactions, especially gripper related simulations, the physics capabilities of the simulation tools are of high importance. Hence, most of the simulation tools investigated had high quality physics simulations but poor and seemingly lowly prioritized rendering capabilities, as can be seen in Figure 3.2. Due to the importance of high quality physic simulations for robotic grasps of compliant objects, the quality of the physics libraries included in the different simulators is important. A common physics library for simulating complex interactions is the Bullet (Bullet, 2018) physics library, which is incorporated in the Gazebo (Gazebo, 2018) simulator.

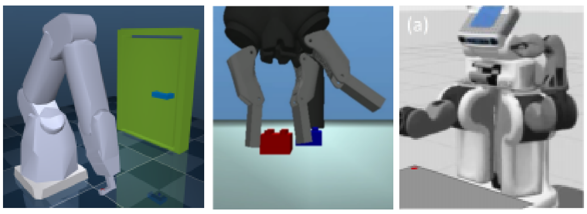


**Figure 3.2:** Three simulated environments respectively from left to right; the door opening environment in MuJoCo from Gu et al. (2016), the brick stacking environment in MuJoCo from Popov et al. (2017) and the ball throwing environment in Gazebo from Ghadirzadeh et al. (2017).

## 3.6 Parallel Experience Gathering

Another method for reducing the time spent on experience gathering, either in a simulator or in a real world environment, is the use of parallel experience gathering. In parallel experience gathering multiple instances of the agent interact with the environment in parallel, and thus multiple combinations of states and actions can be explored in one step. An implementation of this approach was used in the work of Levine et al. (2016), where multiple physical robots interact with the real world environment in parallel and uses these experiences to train the shared agent which governs their predictions. Using multiple instances of physical robots rapidly becomes expensive, and the number of robots available is thus limited by financial aspects. In simulations however, the expenses are often relatively low for computer hardware as opposed to robotic hardware, and scaling is thus easier, where parallel experience gathering can either be conducted on the same machine or on multiple machines, or a combination of the two. The method of conducting experience gathering in parallel can also be thought to ease the training sample set distribution problem, where most gradient based optimizers assume uniform distributions of the samples. In Reinforcement Learning (RL) this is not true, as the new predictions or experiences depends on the previous predictions trained on. Parallel experience gathering explores more combinations of state and action pairs for each instance of the trained agent, and thus reduces the influence of each prediction on the predictions in the next iteration of the agent. Training the agent with parallel experience gathering can thus be a possible method for increasing stability in the learner, as predictions are less heavily influenced by single previous predictions and instead influenced by a multiple of different previous predictions. The benefits of this method can be seen in the work of Gu et al. (2016) and Popov et al. (2017), where the agents both converge towards good solutions faster than their single experience gathering counterparts, and converge to better solutions during the training iterations performed.

## 3.7 Pre-trained Feature Extractors

As mentioned in section 3.2, some work investigating the applicability of DRL methods to robotic manipulation incorporates pre-trained networks when visual inputs are used. These networks function as feature extractors and might enable an agent to more rapidly tune the parts of the network responsible for high level reasoning, assuming good pre-trained networks which extract discriminating and informative features. Two common approaches for learning with pre-trained feature extractors are a), to only train the reasoning layers, and b), to first train the reasoning layers while keeping the feature extractor part of the network constant and then fine tune the whole network. The first approach can be seen in the work of Ghadirzadeh et al. (2017) which used the ResNet (He et al., 2015), and the latter approach can be seen in the work of Johns et al. (2016) which pre-trained their own feature extractor. Both of these achieved promising results, and similar techniques will be investigated in our work.

## 3.8 Domain Randomization for Transfer Learning to the Real World

In order to cope with the challenge of generalization when transferring an agent trained in a simulation to the real world, many have opted to use Domain Randomization (DR). As can be seen in the work of (Tobin et al., 2017a,b; Pinto et al., 2017; Peng et al., 2017), DR has been of great value for real world results after training in simulated environments. The work of Tobin et al. (2017a) shows that they are able to bridge the gap between simulation and real world with training an agent only on RGB images in a heavily randomized environment. Their results show that their system is able to train a real world object detector which is able to grasp objects in noisy environments and has a precision of detecting objects with an accuracy of 1.5 cm. The focus of Tobin et al. (2017a) was to train an agent in a simulation with low-quality rendered RGB images and transferring it directly, without any additional training, to the real world. Low-quality RGB images were used in order to achieve high speed when training the agent and making the input to the agent invariant to real world components such as light, camera field of view, textures and scene configurations.

The work of Zhang et al. (2017) investigate the use of DR when training in a simulation before transferring to the real world in the context of visual motor control for reaching objects with a 7-DoF robotic arm, concerning 3D positions and ignoring the orientation of the end-effector. In contrast to previously mentioned work, Zhang et al. (2017) focused mainly on different domain adaption and fine-tuning methods for further improvements to the performance of the algorithm after transfer. The results produced in their work showed that the algorithm evaluated did not produce satisfactory results after only training in simulation, where the end-positions had a large mean error of 47.1 centimeters in their real world experiments. This error was suspected by the authors to stem from too few and little expressive DR methods being incorporated. However, the benefits of training in a simulation with DR were strongly indicated where training on relatively few real world examples enabled the algorithm to produce impressive results with a final success rate of 97.8% and with a median control error of 1.8 centimeters.

## 3.9 Curriculum Learning

Many have incorporated variants of Curriculum Learning (CL) in their trials and achieved good results on complex problems both in relation to robotic control and otherwise (Popov et al., 2017; Wu and Tian, 2017; Florensa et al., 2017; Bengio et al., 2009). The work of Bengio et al. (2009) compare the results of training ML algorithms with and without CL and conclude that the inclusion of this method can positively and severally impact convergence rates, final accuracies and generalization. However, Bengio et al. (2009) also theorized that CL might not be applicable to all problems and that it might be difficult to develop suitable curriculum strategies. In our work, CL is to be incorporated in a somewhat novel extent. Instead of gradually altering the scope of the problem, the extent of randomness applied to the environment and thus the observations of the agent, is to be gradually increased, investigating if Curriculum Learning (CL) is applicable in this manner.

## 3.10 Summary

In this chapter a selection of relevant work applying Deep Learning (DL) to complex problems, like robotic manipulation and gripper pose estimation, have been presented. For these problems, both supervised and Deep Reinforcement Learning (DRL) methods have yielded promising results, and key ideas and results from both approaches are deemed relevant. Among the methods using DRL, the algorithms Deep Deterministic Policy Gradients (DDPG) and Proximal Policy Optimization (PPO) seem to produce overall satisfactory results on several complex and related problems. Both these algorithms are thus of interest in our work. Several of the studies presented opted to use simulators to mitigate time constraints in experience gathering and hyperparameter tuning, motivating us to investigate the use of a simulator. Some of the work, which successfully transferred agents trained in simulations to real world setups, used aspects of Domain Randomization (DR) and showed its impact on the quality after transfer. Consequently, DR will be of great interest in our thesis. Finally, studies investigating Curriculum Learning (CL) and its benefits have been presented, motivating the investigation of its potential impact on training times in domain randomized environments.

Chapter 4

# Methodology

In this chapter we describe the methodology and approaches used in this thesis. This includes tools, such as Unity, Blender and TensorFlow, two Deep Reinforcement Learning (DRL) methods, some Artificial Neural Network (ANN) architectures, as well as methods for increasing the learning prospects in regard to generalization and transfer learning.

## 4.1 Tools

This section briefly describes the tools used in this project accompanied by their relevance and utilization.

### 4.1.1 Unity

As mentioned in the relevant work section [3.5] most simulators used in previous work regarding gripper pose estimations, incorporate low rendering capabilities and focus primarily on realistic physics simulations. This poses no challenge for DRL methods that do not include RGB-D visual inputs as their state representation and works well for DRL methods that uses depth images or non-visual inputs. Rendering of high quality depth images can often be achieved in these kinds of simulators (Todorov et al., 2012; Gazebo, 2018). However, in our project, we aim to use high quality RGB and depth renderings as input states for the DRL algorithms. Simulators that meet these prerequisites are thus favorable. The Unity Game Engine (Unity, 2018) offers high quality rendering, as Unity is a high-end game engine created for modern game development. The physics simulation capabilities of Unity, however, are optimized and simplified to be used in real-time applications, such as games, and do not meet the accurate simulation requirements of compliant object grasping. Nonetheless, popular physics simulation libraries such as Bullet (Bullet, 2018), also incorporated in Gazebo (Gazebo, 2018), have recently been made available for use in the Unity Game Engine. Recently the developers of Unity released a Machine Learning (ML) API (Unity-ML, 2018) enabling it to communicate with common ML libraries, such as TensorFlow, making it easier to incorporate DRL methods.

We suspect that training a DRL method on low quality RGB-D renderings that poorly matches the visual inputs produced by a real world RGB-D camera, such as Intel RealSense [4.3], can hamper the results after transfer learning [2.8]. Unity shows promising prospects and fulfills the main expectations both regarding rendering quality and physics simulation capabilities, and was therefore chosen as a basis for the simulation environment in our research trials.

### 4.1.2 Blender

Simulation of physics and rendering of 3D environments requires 3D models to be developed and incorporated in the simulation tool. Available for this work were several 3D CAD models of some relevant objects, which can be seen in Figure 4.1. The 3D CAD objects were intended to be used in the simulator and to be produced for experiments in our real world setup. Along with the CAD models of the objects, a model of the Panda gripper [4.3] was made available. However, as Unity does not support CAD models, we used the modelling tool Blender (Blender, 2018) to convert these models to a format supported by Unity. Blender is a free open source 3D modelling software which supports importing and exporting of several common 3D formats, as well as other standard modelling tools. Additionally, simulations often require models of relatively less resolution than of those used in printing and visualization. In order to make the simulations tractable, down sampling of the models was conducted in Blender.

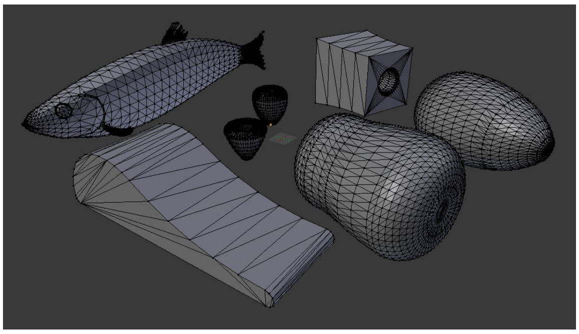


**Figure 4.1:** Wire-frame render in Blender (Blender, 2018) of the 3D CAD objects made available for this thesis by PhD student Aleksander Eilertsen at SINTEF Ocean.

### 4.1.3 TensorFlow

TensorFlow (TensorFlow, 2018) is an open-source graph computation framework developed by the Google Brain Team and has grown considerably in popularity among the researchers of Machine Learning (ML). It was originally developed to be used in ML and deep neural network research, but is now used in a variety of areas. TensorFlow includes functionality for handling the computation of derivatives and structuring computational graphs, which can then be effectively exploited on several CPUs and GPUs. We chose to use this framework as it enables easy construction and high speed training of ANNs, because of its good community, online support and the fact that it is well documented. In our work, TensorFlow was used to implement the networks of the DRL agents with the Python API. The GPU version of TensorFlow was chosen as it enables a substantial speed-up to the training of large networks compared to the CPU version, given adequate GPUs.

## 4.2 Simulation

In order to be able to run experiments on simulations of the gripper pose estimation problem, a simulation environment was created. As mentioned in section 4.1.1, we chose to use Unity as the basis for the simulation as it incorporates good rendering capabilities and prospects of extensions with the Bullet Physics Library (Bullet, 2018). Initially, a baseline for the simulator was developed, containing the necessary functionality for interactions with the TensorFlow API through the use of Unity's new ML library (Unity-ML, 2018). This baseline enables a DRL agent to observe and interact with the environment and facilitates easy replacement and testing of different algorithms. The simulation environment is realized in a way that enables extensions and changes to be made with relatively low effort, depending on the complexity of the change itself, not the integration of the change. Furthermore, an environment incorporating the Panda gripper along with the provided 3D models was developed, where the task of reaching a close and properly orientated gripper pose could be trained on. As the problem investigated only concerns the position and orientation of the end-effector of the gripper, the simulation environment contains only the head of the gripper. Additionally, as one of our research questions concerns the potential of using an agent trained in simulation with rigid objects, compliant physic simulations are not incorporated in this version of the simulator. These design choices make the simulator more lightweight and enable faster environment interactions, resulting in faster training times. Renderings from the simulation environment can be seen in Figure 4.2, where a three step episode with a slightly trained DRL agent is shown.

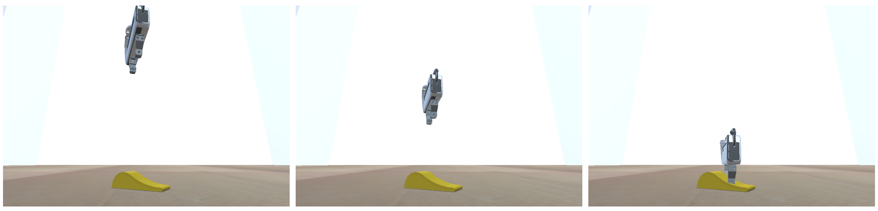


**Figure 4.2:** Renderings of the simulation environment created in our work, where a slightly trained Deep Reinforcement Learning agent is performing a three step episode, estimating a gripper pose for the Fish Fillet object.

### 4.2.1 Environment

In the simulation environment developed, the actions provided are interpreted as movement and rotations relative to the current pose of the gripper, enabling an agent to predict actions relative to its current observation. In other words, the simulation receives a 6-DoF transformation, where the first three parameters represent the offsets of the x, y, and z positions in local coordinate space, and the last three represent the additional Rx, Ry and Rz intrinsic rotations. A visualization of intrinsic rotation and its effect on the local coordinate system can be seen in Figure 4.3.

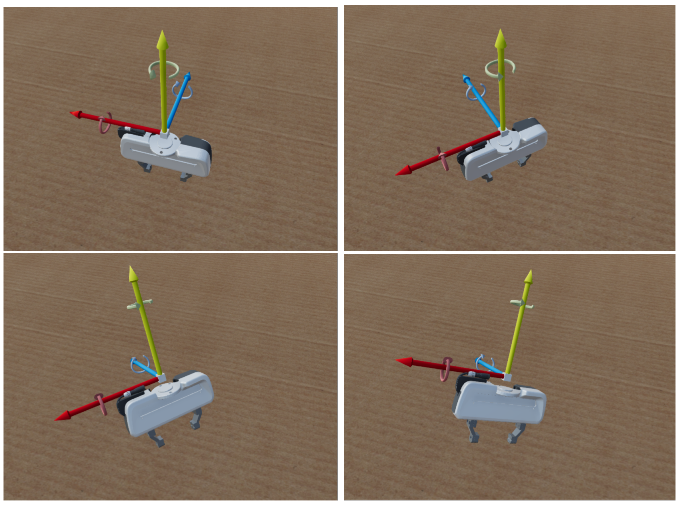


**Figure 4.3:** Four renderings of the gripper's end-effector along with a visualization of its local coordinate system. The straight arrows represent translational axes and the curved arrows represent rotations around these axes. The axes are color coded, where the x axis is red, the z axis is blue and the y axis is yellow. In the top left render a base rotation of the gripper is shown, where the local coordinate system of the end-effector and the global coordinate system are identical. In the top right render the end-effector has rotated around the y axis. In the bottom left render the end-effector has rotated around the x axis after rotating around the y axis. Finally, in the bottom right render the end-effector has rotated around the z axis after the previously mentioned rotations. As can be seen in the renderings, rotations impact how new rotations and translations are interpreted, keeping all axes relative to the end-effector and not a global coordinate system.

The simulation developed is configured to handle an arbitrary amount of steps, produced by an agent, before an episode is completed. However, for the experiments conducted in this work, three environment steps per episode was chosen. Furthermore, the observations gathered from the environment are represented by two images, one for the RGB render and one for the depth render. Both renderings are produced by simulated cameras positioned relative to the gripper's end-effector, approximating the real world setup. For the depth renderings a thresholded noise texture, generated by the Perlin Noise (Perlin, 1985) function, is applied. This produces black artifacts in the image, loosely mimicking the depth shadows and other artifacts produced by a real world depth sensing camera. Renderings showing examples of color and depth visual observations from the simulation can be seen in Figure 4.4.

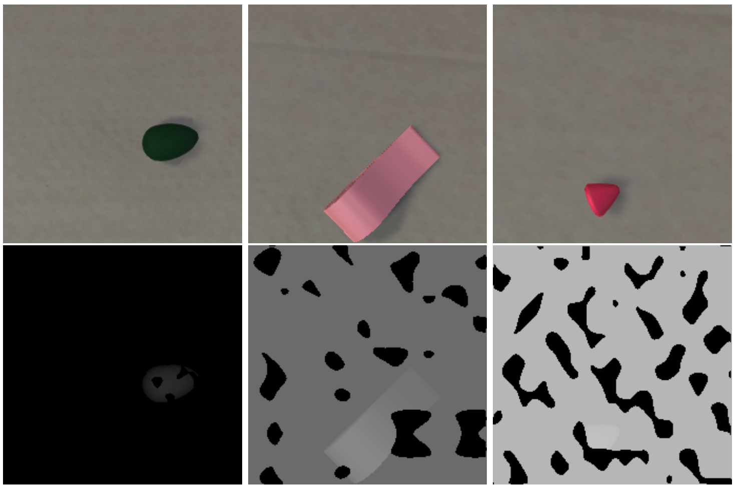


**Figure 4.4:** Examples of visual observations gathered from the simulation environment for three different steps from different episodes. Both the RGB (top) and the depth (bottom) renderings are shown for the objects; Avocado, Fish Fillet and Raspberry. As can be seen in the gray-scale intensity of the depth renderings, both the object and the background gets closer for each step (left-to-right). Additionally, the added thresholded Perlin Noise (Perlin, 1985), producing black artifacts, can be seen in the depth renderings.

The different graspable objects have been integrated in the simulation along with two additional objects. For each episode of an environment instance, one of these objects are selected randomly and placed at a random location with a random orientation. All objects are placed within a bounding volume of 40 by 40 centimeters below the initial pose of the gripper, ensuring that the current object is observable in the first observation of each episode. A render of the objects included in the simulation environment can be seen in Figure 4.5, where some of the objects have two material definitions.

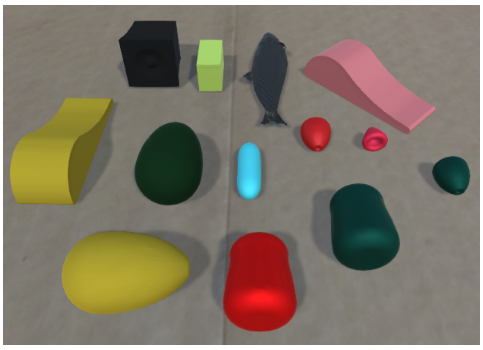


**Figure 4.5:** A render of the 3D models incorporated in our simulation. The conversion of the objects from 3D CAD, seen in Figure 4.1, to Unity (Unity, 2018) compatible formats was done with relatively low effort in Blender [4.1.2]. After conversion, appropriate textures and materials were added to each object.

### 4.2.2 Reward Function

To enable DRL agents to learn gripper pose estimation in our simulation, a three component reward function incorporating relevant aspects for the task was developed. During training, the agent perceives the environment as partially observable as it can only sense the environment through its visual observations, as described in section 4.2.1. However, the reward function exploits the entire state of the environment in order to calculate the reward.

The first component of the reward function consists of a distance measure, calculating the Euclidean distance between the midpoint of the gripper and the closest of a set of good gripper points on the object. Additionally, to increase the difference in rewards positively as the positioning gets better, the distance measure is evaluated on a non-linear function. The formula for the distance component can be seen in Equation 4.1, and the non-linear function used to evaluate the distance score can be seen in Equation 4.2.

$$r_1 = Evaluate(1 - \min_{point}(DistanceBetween(gripperMidpoint, point))) \tag{4.1}$$

$$Evaluate(x) = x^2 \tag{4.2}$$

The second component of the reward function, seen in Equation 4.3, measures the difference in orientation between the gripper and a set of good gripper orientations for the object to be grasped, where the smallest difference is used.

$$r_2 = 1 - \min_{orientation}(\frac{RotationBetween(gripperAngle, orientation)}{90}) \tag{4.3}$$

The third and final component, seen in Equation 4.4, checks whether one or more points on the gripper are below the ground, indicating that the gripper has moved below the ground, and either returns zero or a static penalty of -0.05 based on the results.

$$r_3 = \begin{cases} -0.05, & \text{if one or more points on the gripper are below the ground} \\ 0.0, & \text{otherwise} \end{cases} \tag{4.4}$$

The three components are combined as seen in Equation 4.5, where DistanceScale and AngularScale are set to 0.5 for the experiments in our work. This function then produces a reward function that scales between -1.1 and 1.0 for the possible combinations of gripper poses and object instantiations in the environment and indicates the quality of the current gripper pose given the current object to be grasped.

$$Reward = \frac{(r_1 * DistanceScale + r_2 * AngularScale + r_3) - 0.5}{0.5} \tag{4.5}$$

### 4.2.3 Domain Randomization

Related work investigating transfer learning [2.8] from a simulated environment to a real world setup (Pinto et al., 2017; Peng et al., 2017; Tobin et al., 2017a) showed consistently better results when training in a simulation incorporating Domain Randomization (DR). These findings indicate that the agents learned more general features relevant for their problems, making them more robust to differences between the simulation and the real world setup. Motivated by this, aspects of DR were incorporated in our simulation environment giving variations to the visual observations received in each instance. These aspects were implemented to support an arbitrary amount of randomization based on a scalar provided by the agent, enabling the amount of randomization applied to each episode to vary throughout the course of training. This implementation facilitates data gathering for our first research question: *Can the benefits of Domain Randomization enable an agent solely trained in simulation to produce satisfactory results in a real world environment?* and our third research question: *Can the use of Curriculum Learning decrease the amount of training needed for developing good policies in a randomized simulation environment?*

Several aspects of the environment were implemented with support for randomization. A list of these can be seen in Table 4.1. Some examples of visual observations from the environment with different scales of DR applied can be seen in Figure 4.6. The effect of DR in our simulation environment scales linearly for all objects in the environment except for the ground, where random textures are first applied when the scale of randomization exceeds a threshold of 0.5.

| **Domain Randomization Aspects** | |
|---|---|
| Object | Affected Aspects |
| Graspable Object | Color, Size |
| Ground | Color, Texture |
| Light | Color, Intensity, Orientation |
| Camera | Position, Field of View, Depth Range |
| Gripper | Initial Position |

**Table 4.1:** Table showing the different aspects of our simulation environment affected by Domain Randomization (DR).

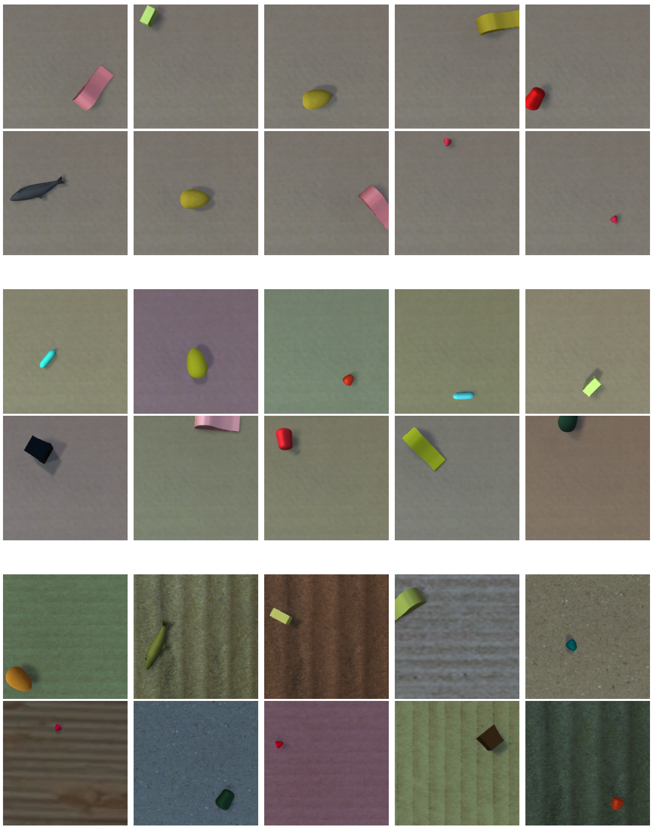


**Figure 4.6:** Examples of RGB visual observations from our simulation environment, where the depth images are omitted as they are mainly black in the first step of an episode. The first two rows show visual observations without DR applied, the two middle rows show visual observations with a Domain Randomization (DR) factor of 0.5 and the last two rows show visual observations with a factor of 1.0. As can be seen in these images, many aspects of the observations are randomized as the power of randomization increases.

## 4.3 Real World Setup

In order to evaluate the policies learned in the simulation environment, a real world setup with a robotic gripper and a RGB-D camera was configured. The gripper used in our work was the Panda robot by Franka Emika, shown untethered in Figure 4.7. Panda has seven degrees of freedom, a maximum payload of three kilograms and a maximum reach of 855 millimeters. The Panda robot is inspired by the human arm, striving to mimic its aspects regarding agility, dexterity and sensitivity (Franka Emika, 2018a). Along with the Panda robot, the depth sensing camera Intel RealSense SR300 (Intel, 2018) was deployed. The RealSense camera renders high quality 1080p resolution images in 30 frames per second and has an optimal depth sensing range between 0.2 and 1.5 meters. The RealSense camera, seen in Figure 4.7, is light-weight and is therefore suitable for robotic applications as it can easily be mounted on a robot, like Panda.

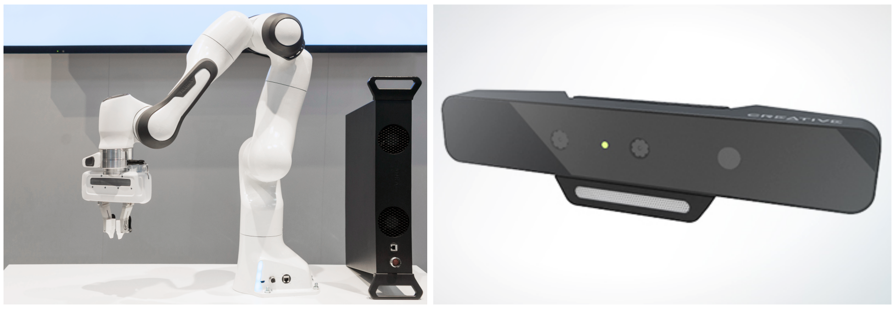


**Figure 4.7:** The Panda robot by Franka Emika in a real world setup (Franka Emika, 2018a) and the Intel RealSense SR300 depth sensing camera (Intel, 2018), both shown untethered.

In our setup the Intel RealSense depth sensing camera has been mounted to the wrist of the Panda robot, enabling the visual inputs to be influenced by the movements and the current pose of the gripper. Photographs from our real world setup with the camera mounted on the end-effector of the Panda gripper can be seen in Figure 4.8, where the setup is fully tethered and operational.

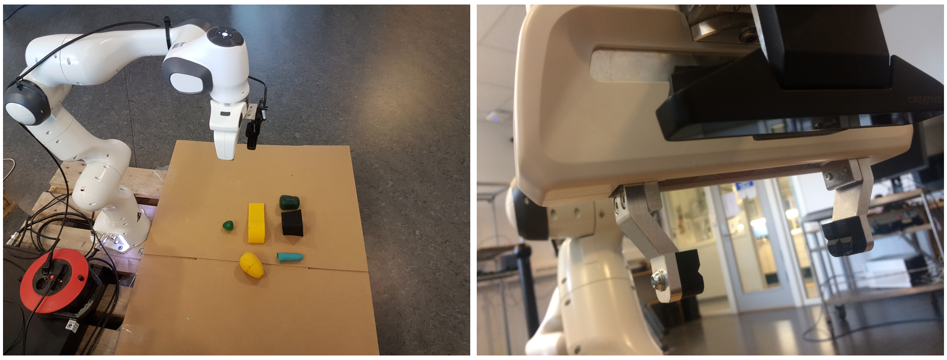


**Figure 4.8:** Photographs showing our real world setup for the gripping pose estimation problem, with the Panda robot by Franka Emika (Franka Emika, 2018a) and a RealSense SR300 camera (Intel, 2018) mounted on its end-effector.

As mentioned in subsection 4.1.2, replicas of the objects trained on in the simulation environment were used in the real world setup to evaluate the quality of the trained DRL agents. In order to replicate these 3D objects, inverted models of the objects were created using Blender. The inverted models were then printed in plastic using a 3D printer, producing moulds for the objects, where some of these can be seen in Figure 4.9. Following, the objects were produced by pouring a silicone compound into the plastic moulds, resulting in semi-compliant replicas. Additionally, a solid plastic version was constructed of the Speaker object. A photograph showing the manufactured objects in our real world setup can be seen in Figure 4.10.

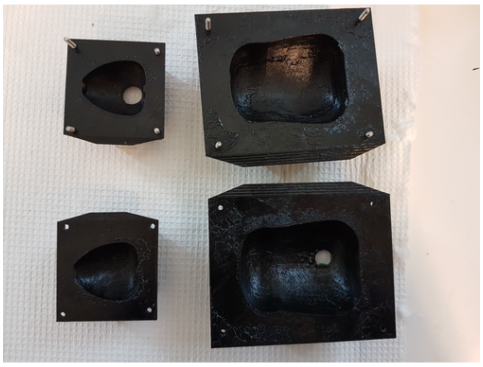


**Figure 4.9:** Image showing two of the 3D printed moulds used for producing the silicone based objects for our real world experiments. The moulds shown were used for producing the Strawberry object (left) and the Paprika object (right). Additionally, the moulds were greased to ensure slip when extracting the solidified objects.

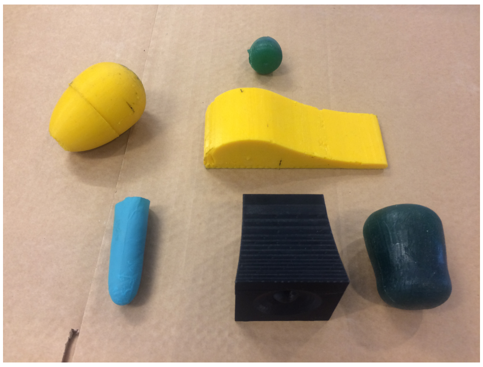


**Figure 4.10:** Photograph of the 3D models used in simulation, moulded in silicone for our real world setup. The semi-compliant objects shown are respectively top-to-bottom and left-to-right; Avocado, Pickle, Strawberry, Fish Fillet, Speaker and Paprika.

A framework building on Franka's existing controller interface (Franka Emika, 2018b) was developed, enabling movement and rotations of the end-effector of the gripper in local coordinate space. This framework enables a Deep Reinforcement Learning (DRL) agent, given equal actions, to produce the same behaviour in the real world setup as in the simulation environment. Furthermore, a framework enabling a DRL agent to sample RGB-D images from the wrist mounted camera and execute actions through the aforementioned framework was constructed. This resulted in a complete framework for interacting and sensing in the real world. The range of the depth images produced were modified in both our real world and simulation setup in order to produce similar observations given similar environment states. A collection of visual observations sampled from different states in our real world setup can be seen in Figure 4.11.

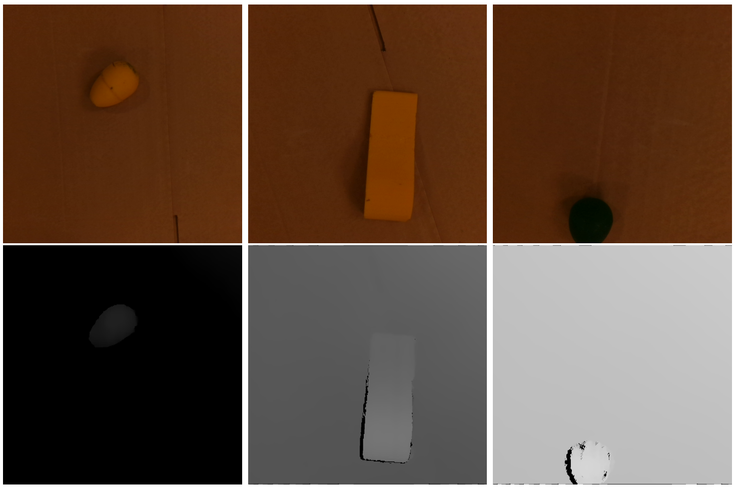


**Figure 4.11:** Examples of visual observations gathered from our real world setup for three different steps from different episodes. Both the RGB (top) and the depth (bottom) renderings are shown for the objects; Avocado, Fish Fillet and Strawberry.

## 4.4 Deep Reinforcement Learning Algorithms

Investigation of related work in regards to gripper pose estimation and robotic control, revealed the Deep Deterministic Policy Gradients (DDPG) algorithm to be a suitable basis for DRL agents working in large state spaces and in a Continuous Action Space (Lillicrap et al., 2015; Gu et al., 2016; Popov et al., 2017; Pinto et al., 2017). Additionally, reviews of related work applying DRL methods to different, but seemingly related and complex problems, revealed the Proximal Policy Optimization (PPO) algorithm to be a recent and promising method for effectively learning in environments with high dimensional state spaces and CAS (Schulman et al., 2017; Koch et al., 2018). Motivated by this, both the DDPG and the PPO algorithms were implemented and evaluated in our preliminary experiments. The DDPG algorithm was developed using the pseudo code shown in subsection 2.6.4, and like the work of Lillicrap et al. (2015) we used the Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930) as our exploration noise. For the PPO algorithm a basis included in the Unity ML library (Unity-ML, 2018) was modified, where the exploration noise incorporated is an action noise sampled from a random normal distribution which decays throughout training.

The preliminary experiments were designed to enable the selection of one algorithm for use in further experiments, where sample efficiency, training times and performance were of the essence. To ensure gathering of data from multiple training runs, creating a basis for better comparison, relatively few training iterations and relatively shallow networks were used. The network architectures chosen for our preliminary experiments built on an instance of the pre-trained AlexNet (Krizhevsky et al., 2012) classifier, where only the weights of the added fully connected end part of the networks were tuned. From AlexNet only the five first CNN layers were used, and a diagram of the full AlexNet structure can be seen in Appendix A. The DDPG and PPO network architectures used for these experiments can be seen in Figure 4.12 and Figure 4.13 respectively. The results of the preliminary experiments are presented in section 5.1, and PPO was chosen for further experiments.

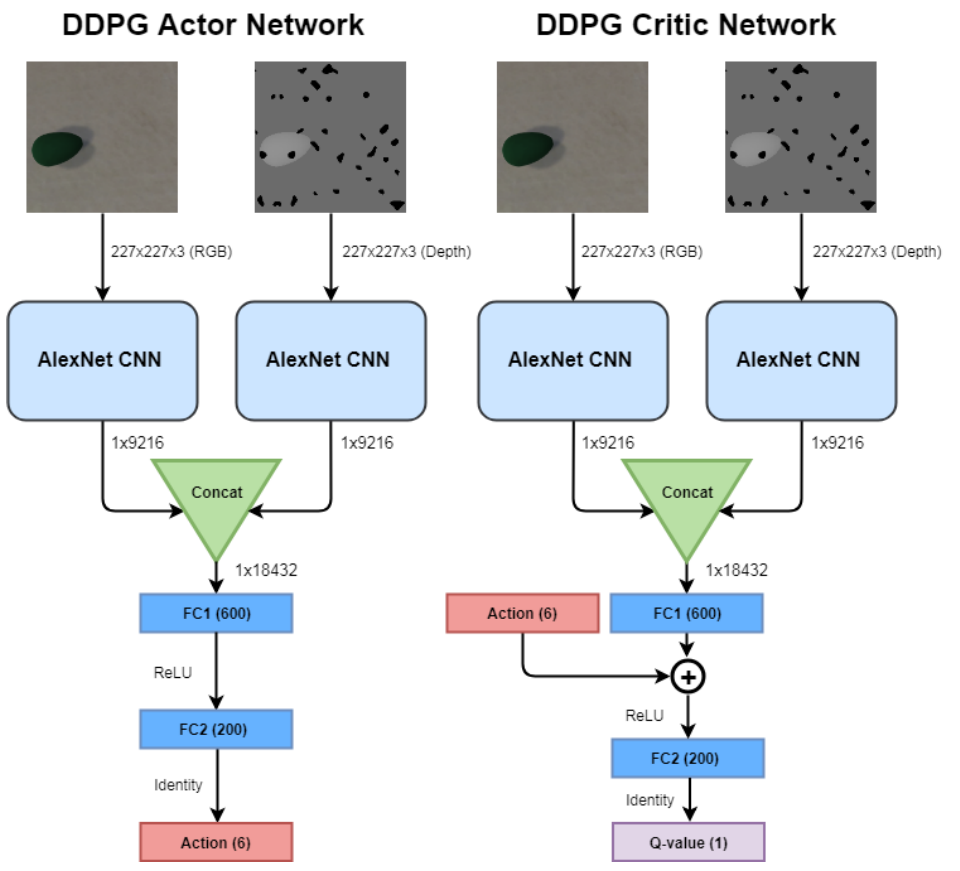


**Figure 4.12:** Deep Deterministic Policy Gradients (DDPG) actor and critic networks learning the reasoning on features extracted by a pre-trained feature extractor. The light blue squares represent the CNN component from AlexNet (Krizhevsky et al., 2012), which can be seen more detailed in Figure 4.15. The AlexNet CNN component receives a 227x227 dimension image with three channels and outputs 9216 features. The green triangle represents a layer where the inputs are concatenated. The dark blue squares represent fully connected networks, where the number of nodes in each layer is presented in parentheses. Both networks receive the RGB and the depth images which are fed separately through their AlexNet CNN components resulting in 9216 features each. These features are then concatenated and fed to the first fully connected layer. The actor network outputs an action based on a state, while the critic network outputs an estimated long-term reward (Q-value) based on a state-action pair.

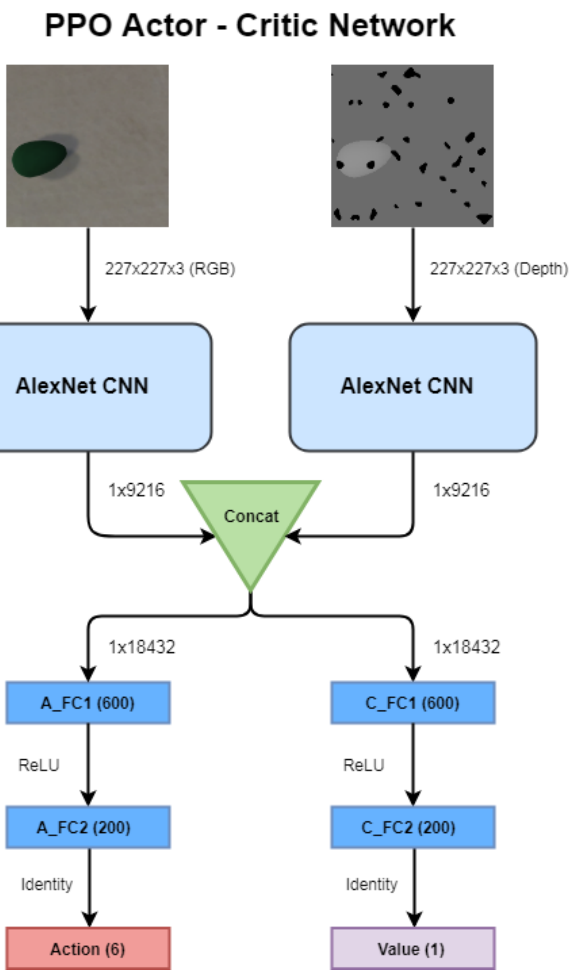


**Figure 4.13:** Proximal Policy Optimization (PPO) actor and critic network learning the reasoning on features extracted by a pre-trained feature extractor. The light blue squares represent the CNN component from AlexNet (Krizhevsky et al., 2012), which can be seen more detailed in Figure 4.15. The AlexNet CNN component receives a 227x227 dimension image with three channels and outputs 9216 features. The green triangle represents a layer where the inputs are concatenated. The dark blue squares represent fully connected networks, where the actor sub-network is prefixed with the letter ”A” and the critic sub-network is prefixed with the letter ”C”. The number of nodes in each of the fully connected layers is presented in parentheses and the used activation function is shown along the output. The RGB and the depth images are both fed to their dedicated instance of the AlexNet CNN component, producing features for each input. These features are then concatenated and fed to the first fully connected layer of both the actor and the critic sub-networks. The actor sub-network outputs a predicted action and the critic sub-network predicts an estimated long-term reward (Value).

In the main part of our experiments two different approaches were investigated regarding network architectures and training regimes for PPO. The two different architectures both consist of a deep CNN followed by a shallow fully connected network, however, the CNN components differ. In the first version of the network, a custom deep CNN architecture is constructed. This network takes the whole RGB-D image as input and is initialized with random weights, enabling the agent to train fully end-to-end, specializing for the problem given and jointly reason on both color and depth data. A diagram of the first network can be seen in Figure 4.14.

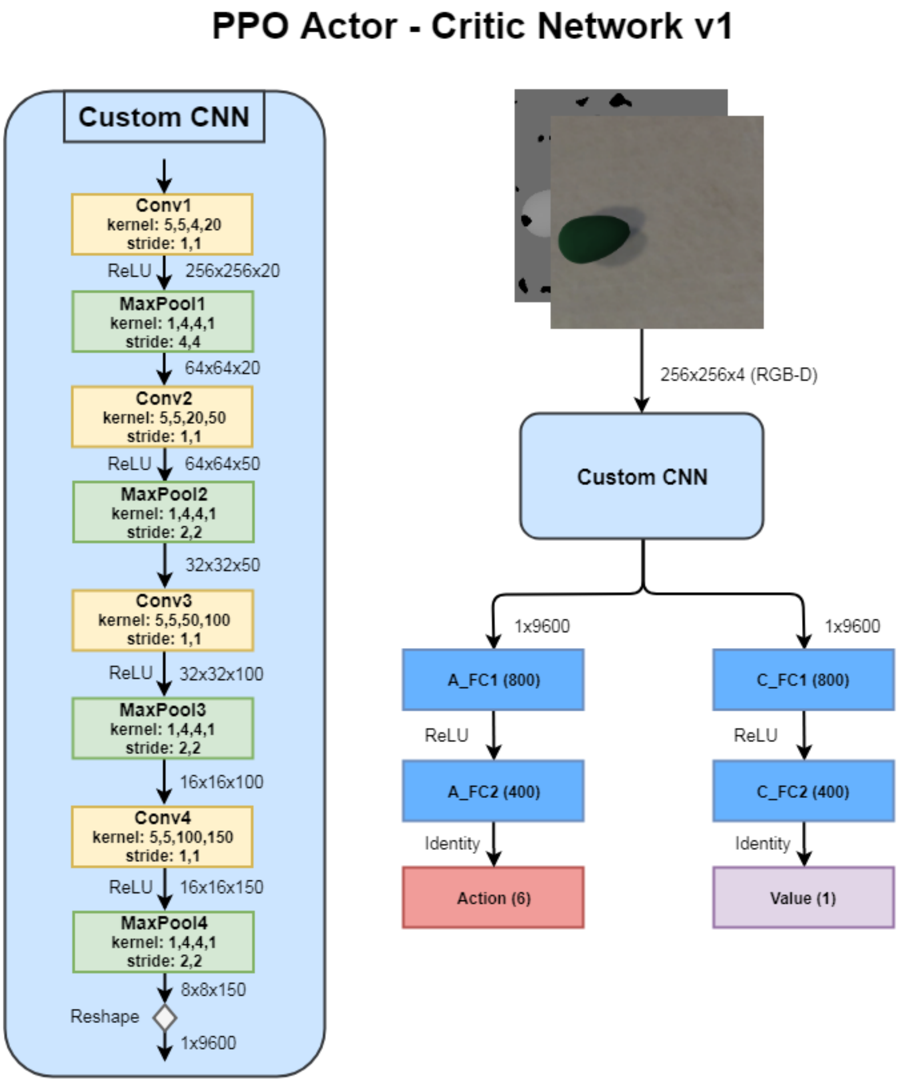


**Figure 4.14:** Proximal Policy Optimization (PPO) network architecture with a custom CNN component which operates on RGB-D images jointly. The custom CNN component (left) is shown with corresponding kernels, strides and activation functions for each layer [2.3]. Additionally, resulting dimension of the data is shown between the layers. Before being fed to the fully connected sub-networks, the output of the CNN component is reshaped to a one-dimensional tensor. The fully connected sub-networks are prefixed with the letter ”A” for the actor sub-network the letter ”C” for the critic sub-network. These sub-networks produce an action and an estimated long-term reward (Value) respectively.

The second network architecture, like in the work of Ghadirzadeh et al. (2017) and Johns et al. (2016), incorporates pre-trained feature extractors. We chose to use AlexNet with weights trained on the ImageNet dataset (ImageNet, 2018) as our feature extractor. Since AlexNet uses three channel images (RGB) as input, two instances taking the RGB and the depth images separately are incorporated. It can be expected that CNNs trained on large image classification datasets extracts general features, for instance edges, in the early layers of the network. Motivated by this, we chose not to tune the weights of the first and the second CNN layers of the AlexNet instances. From the pre-trained AlexNet the first five layers are used, enveloping the entirety of the CNN layers. The second network architecture used in our experiments can be seen in Figure 4.15.

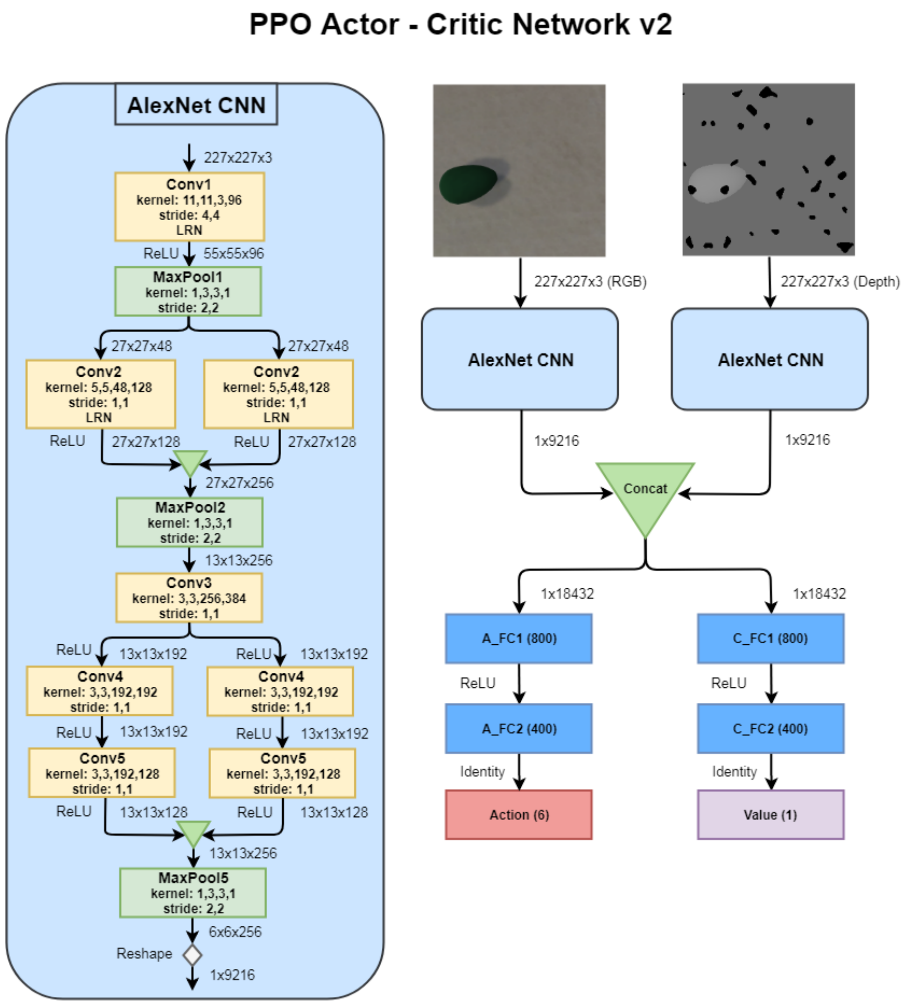


**Figure 4.15:** Proximal Policy Optimization (PPO) network architecture building on a pre-trained sub-network of AlexNet (Krizhevsky et al., 2012). The AlexNet CNN component (left) is shown with corresponding kernels, strides, activation functions and output dimensions [2.3]. Additionally, the first two convolutional layers use a Local Response Normalization (LRN) function, which normalizes the output of the layer based on its activations (described detailed in the work of Krizhevsky et al. (2012)). The output of each AlexNet CNN component is reshaped to a single dimension tensor before being concatenated and fed to the actor and critic sub-networks. The fully connected sub-networks are prefixed with the letter "A" for the actor sub-network the letter "C" for the critic sub-network. These sub-networks produce an action and an estimated long-term reward (Value) respectively.

By conducting experiments with the aforementioned network architectures, data giving indications on the difference between training on the color and depth jointly and separately can be acquired. Additionally, some indications to the prospects of tuning a network specialized for color images and location and rotation invariant reasoning on RGB-D images for a problem that require spacial and rotational features, can be obtained.

## 4.5 Experiments

As sample efficiency and training times were of importance in our work, we decided to impose some minor constraints on the environment interactions for the agents during training and evaluation. The first constraint discards the rotations around the x and z axis for the first two steps of each episode. This results in the agent only rotating around the y axis for the first two steps producing rotations around an axis that is perpendicular to the ground. Our second constraint limits the total transformation of an episode to 75 centimeters translation in each axis and 180 degrees rotation around the y axis and 60 degrees around the x and z axes. This is enforced by dividing the output of the agents, which are scaled between -1 and 1, by four for the translations and three for the rotations. The motivation behind enforcing the transformation limitations with a division, instead of clipping the actions estimated by the network, is to retain an expressive range for the actions. Additionally, a third constraint is enforced as the RealSense camera starts to produce more artifacts when the distance between the camera and the object in focus is outside its optimal depth sensing range [4.3]. This last constraint limits the gripper movement to 15 centimeters along the y axis towards the ground for each of the two first steps of an episode, resulting in the camera mounted on the gripper's end-effector being roughly 25 centimeters above the ground before the last step is executed.

Pre-processing methods were applied to the RGB images before feeding them to the networks as related work showed noteworthy results obtained with such methods (Ghadirzadeh et al., 2017). This pre-processing was done by computing and subtracting the mean of each channel in the image, resulting in pixel values that range from -1 to 1. The motivation behind this was to lower the differences between images produced with different lighting and contrast. For the depth images, however, no pre-processing was applied as we suspect that valuable depth information could be lost.

### 4.5.1 Simulation Experiments

In order to gather data for the research question regarding the benefits of Domain Randomization (DR) two different experiments were conducted in the simulation environment. The first experiment consisted of training the different agents on an instance of the environment where no aspects were randomized, producing a baseline to measure against. The second experiment concerned training the different agents on an instance where many aspects of the environment were randomized, as described in subsection 4.2.3, producing trained policies to evaluate against the aforementioned. Additionally, to investigate if Curriculum Learning (CL) can positively impact the amount of training necessary for developing good policies in a randomized environment, a third experiment was conducted. In this experiment the different agents were trained in an environment where the extent of DR increased iteratively during the training phase.

### 4.5.2 Real World Experiments

To evaluate the policies produced by training in the simulated environment, experiments in the real world setup were conducted. These experiments consisted of several grasping attempts for each object in the setup, producing the data necessary for estimating overall grasp success and grasp success per object for each of the different agents. However, as a binary classification of whether a grasp was successful or not does not provide information regarding the proximity of the failed attempts, additional measurements were performed. These measurements evaluated if the pose estimated was within a certain threshold regarding errors in position and angle, thus giving a more accurate representation of the quality of the policy applied.

As a final experiment, the different agents were evaluated on a set of novel objects which were not included in the simulation. This produced indicative data as to the generalization capabilities of the policy beyond the examples included in the training set.

Chapter 5

# Results and Discussion

This chapter presents the results, discussions and analysis of our work, where the problem of gripper pose estimation for semi-compliant objects was investigated. As described in section 4.4, both the Deep Deterministic Policy Gradients (DDPG) algorithm and the Proximal Policy Optimization (PPO) algorithm were deemed promising based on related work. Both algorithms produced satisfactory policies on various complex problems incorporating high dimensional state spaces and a Continuous Action Space (CAS). To select the most promising of these, preliminary experiments were conducted, comparing the two algorithms based on multiple criteria. The most promising algorithm was then implemented with two different network architectures and trained in three different versions of the simulation environment. Following, the trained agents were evaluated in our real world setup. The simulation experiments presented in this chapter all conduct 500 episodes to estimate the current quality of the agent, ensuring good estimations. Additionally, for both our simulation and our real world environments, all episodes are divided into three environment interactions, often referred to as steps, constituting a complete grasp pose estimation sequence.

## 5.1 Preliminary Experiments

Early experiments were conducted, applying the DDPG and the PPO algorithms to the problem of gripper pose estimation in our simulation environment. Each algorithm was trained for 50000 episodes, equivalent to 150000 environment interactions, over four runs with random simulation seeds. Multiple and relatively short runs were conducted to generate data for comparing the two algorithms with regards to; training times, final policies and sample efficiency. This section presents the setup used for these experiments, along with the results obtained and a discussion of these.

### 5.1.1 Network Architecture

As shown in section 4.4, the networks used in our preliminary experiments used a pre-trained instance of AlexNet (Krizhevsky et al., 2012) as a feature extractor paired with custom relatively shallow fully connected networks. In order to constrain training times and total execution times for these experiments, only the fully connected end part of the networks was tuned during training, leaving the weights of AlexNet constant. This enabled several runs to be executed, creating a better basis for selecting the most promising algorithm.

### 5.1.2 Hyperparameters

The hyperparameters used for the DDPG algorithm can be seen in Table 5.1, and the hyperparameters used for the PPO algorithm can be seen in Table 5.2. For both algorithms the hyperparameters were found by empirical testing. The DDPG algorithm seemed to be sensitive to changes in hyperparameters, where small changes could severally decrease learning stability. The PPO algorithm, however, responded better to changes in hyperparameters, making it easier to tune.

| **Hyperparameters for the Deep Deterministic Policy Gradients (DDPG) algorithm** | | |
|---|---|---|
| Parameter | Value | Description |
| Actor Learning Rate | $1^{-5} \to 1^{-6}$ | The decaying learning rate for the actor's Adam optimizer. |
| Actor Epsilon | 0.1 | The Epsilon for the actor's Adam optimizer. |
| Critic Learning Rate | $1^{-4} \to 1^{-5}$ | The decaying learning rate for the critic's Adam optimizer. |
| Critic Epsilon | $1^{-8}$ | The Epsilon for the critic's Adam optimizer. |
| Mini Batch Size | 64 | The number of sampled interactions from the Replay Memory the algorithm is trained on for each step. |
| Tau ($\tau$) | $1^{-4}$ | Value dictating how fast the target networks should approximate the base networks each update. |
| Gamma ($\gamma$) | 0.99 | Value dictating how important later rewards are compared to the current reward. |
| Replay Memory Size | 3500 | The maximum number of experiences the replay memory holds. |
| Pre-training Environment Episodes | 100 | The number of interactions that are performed prior to initiating learning. |
| Number of Episodes | 50000 | The total number of episodes the algorithm is trained on. |
| Number of Steps | 3 | The number of steps in each episode. |
| Positional Noise Amplitude | $2 \to 0.1$ | The decaying amplitude multiplied by the Ornstein-Uhlenbeck noise (Uhlenbeck and Ornstein, 1930). |
| Rotational Noise Amplitude | $3 \to 0.1$ | The decaying amplitude multiplied by the Ornstein-Uhlenbeck noise (Uhlenbeck and Ornstein, 1930). |

**Table 5.1:** Hyperparameters used for the Deep Deterministic Policy Gradients algorithm [2.6.4] in our preliminary experiments. The hyperparameters were found through empirical testing. The positional noise amplitude, the rotational noise amplitude and the learning rates are all decayed linearly throughout the course of training.

| **Hyperparameters for the Proximal Policy Optimization (PPO) algorithm** | | |
|---|---|---|
| Parameter | Value | Description |
| Learning Rate | $1^{-5} \rightarrow 1^{-6}$ | The decaying learning rate for the Adam optimizer. |
| Epsilon $\epsilon$ | $0.2 \rightarrow 0.1$ | The decaying epsilon dictating the bounds for the clipped surrogate objective. |
| Beta $\beta$ | $1^{-2} \rightarrow 1^{-5}$ | The decaying beta dictating the strength of the entropy regularization. |
| Number of Epochs | 21 | The number of epochs that the algorithm is trained for each training interval. |
| Mini Batch Size | 64 | The number of sampled interactions from the buffer the algorithm is trained on for each epoch. |
| Buffer Size | 512 | The number of experiences that are stored in the buffer. |
| Lambda $\lambda$ | 0.95 | The lambda value used to calculate the Generalized Advantage Estimate where this value dictates how much the agent relies on its current value estimate when calculating a new one. |
| Gamma ($\gamma$) | 0.99 | Value dictating how important later rewards are compared to the current reward. |
| Number of Episodes | 50000 | The total number of episodes the algorithm is trained on. |
| Number of Steps | 3 | The number of steps in each episode. |
| Noise Amplitude | $1 \rightarrow 0$ | The decaying amplitude multiplied with a random normal distribution. |

**Table 5.2:** Hyperparameters used for the Proximal Policy Optimization algorithm [2.6.5] in our preliminary experiments. The hyperparameters were found through empirical testing. The learning rate, the epsilon and the beta parameters are decayed linearly throughout training. The noise amplitude is decayed towards zero throughout training, based on the calculated loss and the beta parameter, where high losses give more exploration and lower losses give more exploitation.

### 5.1.3 Results and Discussion

The results produced by training the DDPG and the PPO algorithms in our simulation environment without Domain Randomization (DR) for the gripper pose estimation problem can be seen in Figure 5.1. Both algorithms seemed to learn relatively fast and achieved adequate results given the fact that shallow networks and few episodes were used. However, as can be seen in Figure 5.1, the PPO algorithm both learned faster and ended at a significantly better policy than the DDPG algorithm in our experiments. Additionally, the results show that PPO has slightly less variance in performance between different runs.

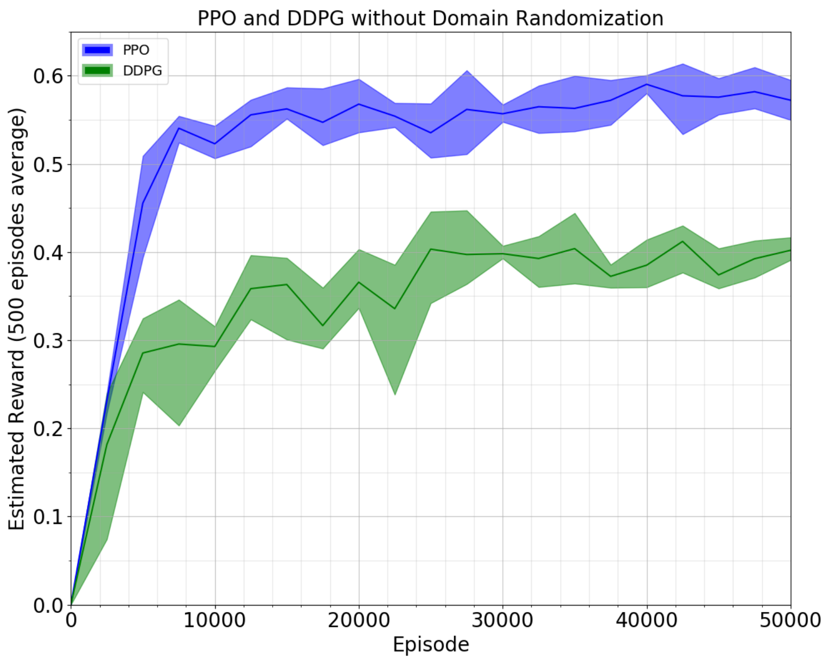


**Figure 5.1:** Graph showing the results produced by training the Proximal Policy Optimization (PPO) algorithm (blue) and the Deep Deterministic Policy Gradients (DDPG) algorithm (green) for 50000 episodes (150000 environment interactions) in a **non-randomized** instance of our simulation. For each algorithm the mean rewards and upper and lower bounds obtained from four individual runs are shown.

The results produced by training the algorithms our simulation environment incorporating Domain Randomization (DR) can be seen in Figure 5.2, where the power of randomness was set to 1.0. Both algorithms learned relatively fast for these experiments as well, however, to our expectations, the final results were not as good as in the non-randomized instance. Both algorithms seemed to vary more across different runs, with DDPG varying slightly more. Interestingly, the DDPG algorithm seemed to be only marginally affected by the introduction of DR, whereas the PPO algorithm was notably affected. We suspect that this is caused by the difference in training strategies, where DDPG used a larger replay memory and only tuned the weights in the network slightly for each training update. The PPO algorithm, however, incorporated a smaller replay memory in our experiments, and used a more aggressive tuning strategy, where the advantage estimates, along with the clipped surrogate loss, were used to tune the weights for each update. Despite being less impacted by DR, DDPG did not produce quite as good final policies as PPO, however, the difference in results were less prominent for these experiments.

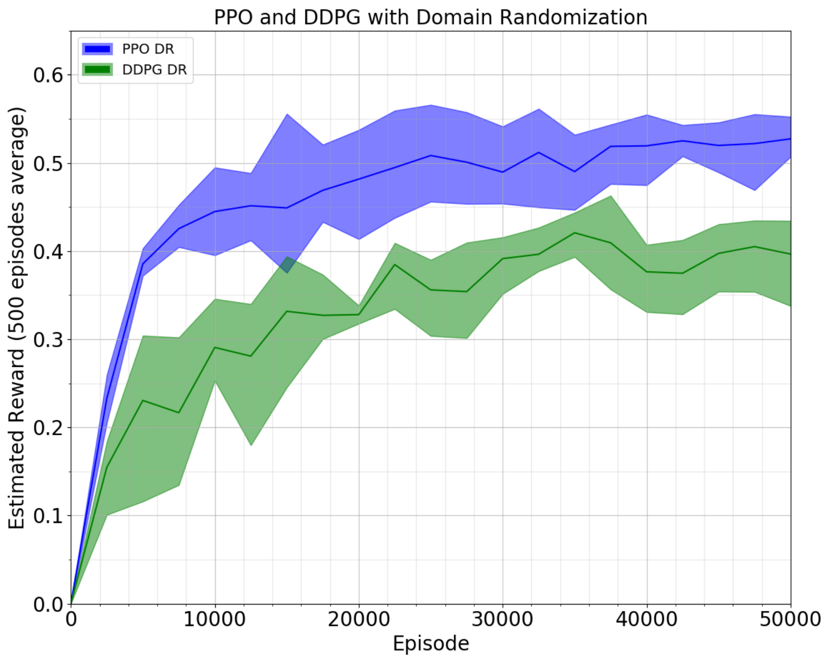


**Figure 5.2:** Graph showing the results produced by training the Proximal Policy Optimization (PPO) algorithm (blue) and the Deep Deterministic Policy Gradients (DDPG) algorithm (green) for 50000 episodes (150000 environment interactions) in a **domain randomized** instance of our simulation. For each algorithm the mean rewards and upper and lower bounds obtained from four individual runs are shown.

We suspect that better hyperparameters can be obtained for DDPG, rendering it capable of competing with the resulting policies of PPO. However, the difference in training times and sample efficiency are significant. For these experiments, the DDPG algorithm trained for one epoch each step, while the PPO algorithm only trained 24 epochs at intervals of 512 steps, resulting in DDPG training roughly 21 times more than PPO. This impacts the difference in training times as the complexity of the networks applied increases. Additionally, DDPG seemed less stable than PPO, where the difference in fluctuations and convergence rates seemed to increase with the increase of network complexity. Based on these results, along with the observations regarding training times and hyperparameter tuning, the Proximal Policy Optimization (PPO) algorithm was chosen for further experiments.

## 5.2 Simulation Experiments

This section presents the setup for the experiments conducted in the different instances of our simulation environment along with the results obtained by training the PPO algorithm with two different network architectures. Each architecture was evaluated over two runs and trained for 600 thousand episodes in the base simulation, the Domain Randomization (DR) version and the Curriculum Learning (CL) version of our simulation.

### 5.2.1 Network Architectures

As described in section 4.4, the different architectures used in our simulation experiments differed in the Convolutional Neural Network (CNN) part of the networks. The first architecture, seen in Figure 4.14, used a custom CNN component. The second architecture, seen in Figure 4.15, used the CNN component of a pre-trained feature extractor, namely AlexNet (Krizhevsky et al., 2012). In contrast to our preliminary experiments, most of the CNN layers in the second architecture were tuned during training, where only the first two CNN layers were kept constant as it can be expected that these early layers extract highly general features. In the first architecture, the whole network was tuned as it was initialized with random weights and no prior assumptions.

### 5.2.2 Hyperparameters

The hyperparameters used for these experiments were similar to the parameters used in our preliminary experiments for the PPO algorithm, which can be seen in Table 5.2. As we suspected that the network architecture building on a pre-trained instance of AlexNet contained a good starting point for the weights, a smaller learning rate was chosen for this configuration. This was done in order to not tune the pre-trained weights of the AlexNet CNN layers too heavily with the early and rough weight updates produced by randomly initialized fully connected networks. As the whole network of the first architecture was initialized with random weights, a larger learning rate was used for these runs. The learning rate for the two architectures both decayed throughout learning and were $2^{-4} \rightarrow 1^{-5}$ for the first architecture and $2^{-5} \rightarrow 1^{-6}$ for the second architecture.

### 5.2.3 Results and Discussion

The agents trained in the base simulation without Domain Randomization (DR) obtained the highest estimated final rewards for these experiments. The results after training the PPO algorithm with our custom CNN component in a non-randomized version of our simulation environment can be seen in Figure 5.3. The rewards produced throughout these runs show that the agent quickly learns decent policies, requiring only 100000 episodes for obtaining an estimated reward of roughly 0.8. As can be seen in the plot, this configuration ended with an estimated reward of roughly 0.94. Additionally, as can be seen in Table 5.3, this indicates that the average gripper poses estimated by this agent lies within three centimeters and six degrees of a good grasping pose for all the objects in the simulation environment.

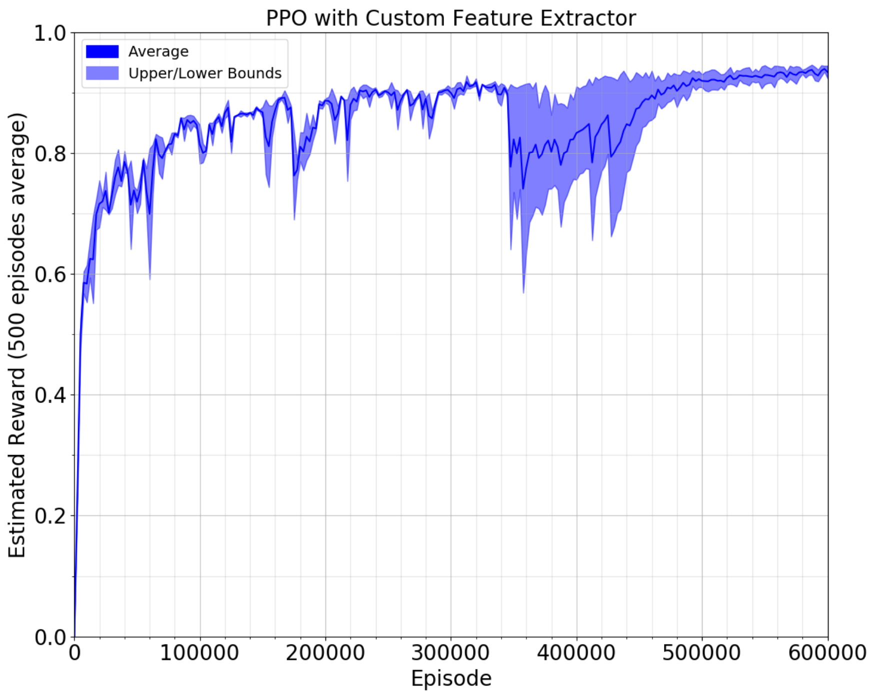


**Figure 5.3:** Graph showing the results after running the Proximal Policy Optimization (PPO) algorithm with our **custom Convolutional Neural Network (CNN)** component in a **non-randomized** instance of our simulation environment for 600000 episodes (1.8 million environment interactions). The average, the upper bound and the lower bound are generated by two separate runs of this configuration. The estimated reward for this agent is measured every 2500 episode, where each estimate is an average calculated over 500 episodes.

**Rewards Produced by Proximate Gripper Poses**

| | 0cm | 1cm | 2cm | 3cm | 4cm | 5cm | 6cm |
|---|---|---|---|---|---|---|---|
| 0° | 1.00 | 0.98 | 0.96 | 0.94 | 0.92 | 0.90 | 0.88 |
| 2° | 0.98 | 0.96 | 0.94 | 0.92 | 0.90 | 0.88 | 0.86 |
| 4° | 0.96 | 0.94 | 0.92 | 0.90 | 0.88 | 0.86 | 0.84 |
| 6° | 0.93 | 0.91 | 0.89 | 0.87 | 0.85 | 0.84 | 0.82 |
| 8° | 0.91 | 0.89 | 0.87 | 0.85 | 0.83 | 0.81 | 0.79 |
| 10° | 0.89 | 0.87 | 0.85 | 0.83 | 0.81 | 0.79 | 0.77 |
| 12° | 0.87 | 0.85 | 0.83 | 0.81 | 0.79 | 0.77 | 0.75 |

**Table 5.3:** Table showing the values produced by the reward function in our simulation environment, given small positional and rotational errors.

The results produced by training the PPO algorithm with parts of AlexNet as a starting point for a feature extractor can be seen in Figure 5.4. Similar to the agents with the custom CNN component, the results for these runs show that the agents quickly develop decent policies. Additionally, the fluctuations in these runs seem to be less prominent than those produced by the agents using the custom CNN. We suspect that this difference might stems from the fact that the first two layers of the pre-trained CNN component contain good weights and are kept constant, reducing the cascading effect of heavy weight updates in these early layers. Despite less fluctuations and a steep learning curve, this configuration ended with a slightly lower estimated reward of roughly 0.92, which indicates that the average gripper poses estimated lies within four centimeters and eight degrees from a good gripper pose.

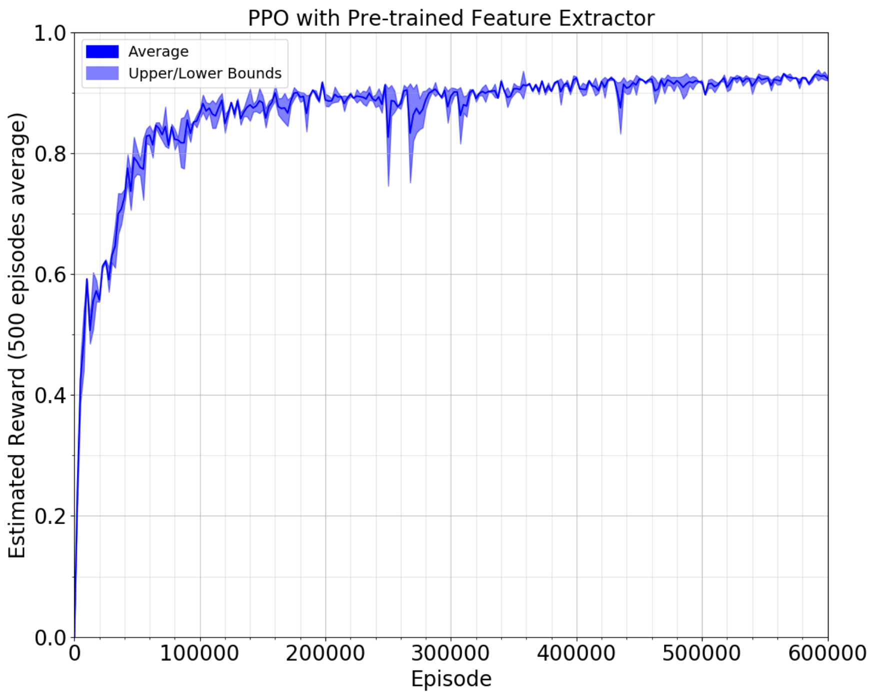


**Figure 5.4:** Graph showing the results after training the Proximal Policy Optimization (PPO) algorithm with a **pre-trained feature extractor** in a **non-randomized** instance of our simulation environment for 600000 episodes (1.8 million environment interactions). The average, the upper bound and the lower bound are generated by two separate runs of this configuration. The estimated reward for the agent is measured every 2500 episode, where each estimate is an average calculated over 500 episodes.

The results produced by training the PPO algorithm with a custom feature extractor in an environment incorporating Domain Randomization (DR) are presented in Figure 5.5. Despite being exposed to a heavy randomized environment, with great differences between different visual observations, the results of these runs show that the agents quickly learns decent policies. These runs ended with a policy that produced an estimated reward of roughly 0.92, which, as can be seen in Table 5.3, signifies that the average estimated gripper poses are within four centimeters and eight degrees of a good gripper pose.

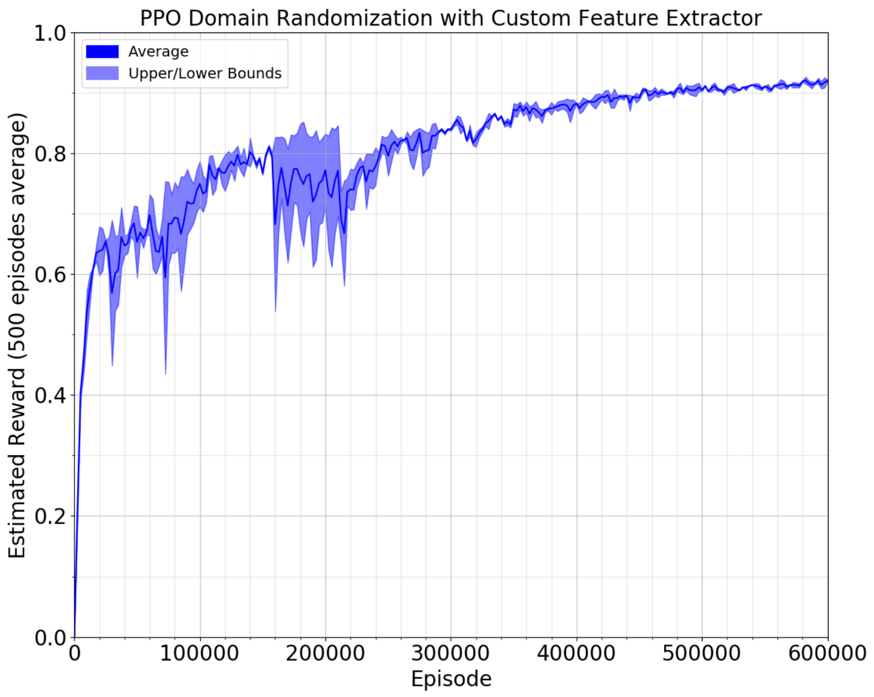


**Figure 5.5:** Graph showing the results after running the Proximal Policy Optimization (PPO) algorithm with our **custom feature extractor** in a **Domain Randomized (DR)** instance of our simulation environment for 600000 episodes (1.8 million environment interactions). The average, the upper bound and the lower bound are generated by two separate runs of this configuration. The estimated reward for the agent is measured every 2500 episode, where each estimate is an average calculated over 500 episodes.

Running the PPO algorithm with the second network architecture, which builds on a pre-trained CNN component, in our DR simulation environment, produced the results shown in Figure 5.6. Similar to the configuration using a custom CNN component, the agents learned rapidly despite DR being applied. The resulting policies obtained a final estimated reward of roughly 0.9, which like in the experiments for the non-randomized environment, were slightly inferior to the results obtained by the configuration using our custom CNN.

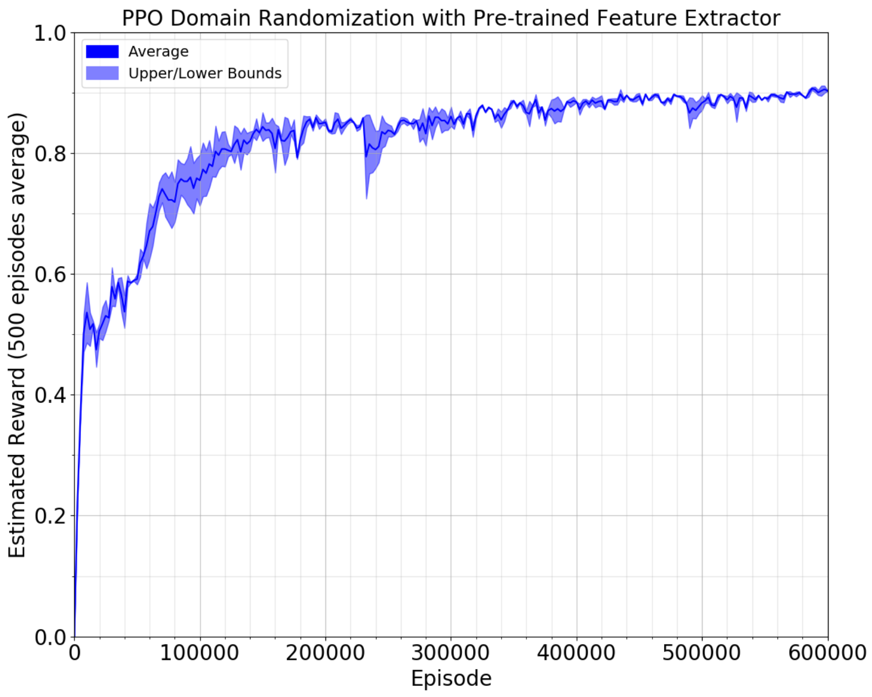


**Figure 5.6:** Graph showing the results after training the Proximal Policy Optimization (PPO) algorithm with our **pre-trained feature extractor** in a **Domain Randomized (DR)** instance of our simulation environment for 600000 episodes (1.8 million environment interactions). The average, the upper bound and the lower bound are generated by two separate runs of this configuration. The estimated reward for the agent is measured every 2500 episode, where each estimate is an average calculated over 500 episodes.

A plot showing the results from both the configuration using our custom feature extractor and the configuration using a pre-trained feature extractor trained in our DR simulation can be seen in Figure 5.7. As can be seen in this plot, the configuration using our custom CNN component produces more fluctuations than the configuration using the pre-trained CNN. Despite of this, the first architecture, using our custom CNN component, yields better final policies. We suspect that this difference in final policies stems from the fact that our custom CNN extracts features from the RGB and the depth images jointly, whereas the second network architecture extracts features from the RGB and the depth images separately. Additionally, the second network architecture utilizes a feature extractor trained only on RGB images. Consequently, it is not necessarily guaranteed that this CNN component extracts descriptive and discriminating features from depth images in the early layers.

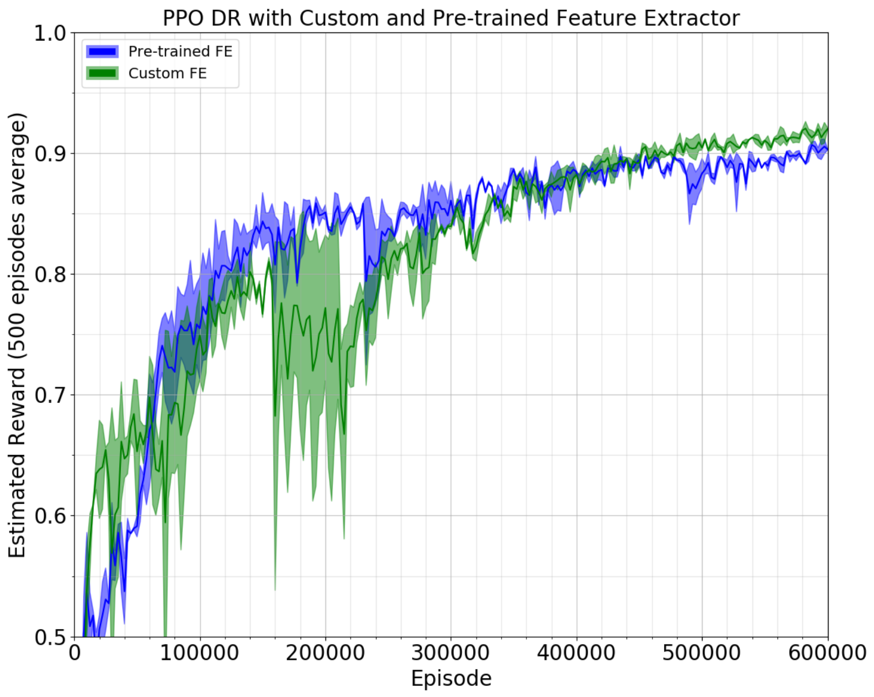


**Figure 5.7:** Graph showing a comparison between the Proximal Policy Optimization (PPO) algorithm with **pre-trained** and **custom feature extractors** in the **Domain Randomized (DR)** version of our simulation environment. The average, the upper bound and the lower bound are generated by two separate runs of both configurations. The blue graph shows the results produced with a pre-trained feature extractor, and the green graph shows the results produced with a custom feature extractor.

The results obtained by running the PPO algorithm with the network architecture building on our custom CNN component on an instance of the simulation environment incorporating CL, can be seen in Figure 5.8. The Curriculum Learning strategy implemented, linearly increased the amount of randomization from 0.0 to 1.0 throughout training. The amount of randomization was increased by 0.1 every 15000 episode starting at episode 32500 and reaching a DR factor of 1.0 at episode 167500. The points of randomization increase are represented in the plot as dotted black lines along the axis of the corresponding episode. For most of the randomization increases the agent seemed to adapt quickly to the changes, even to the introduction of texture changes which occurs when the DR factor exceeds 0.5 at episode 107500. The final estimated reward obtained by this agent was roughly 0.92, rendering it comparable with the agent trained in the environment with full DR from start.

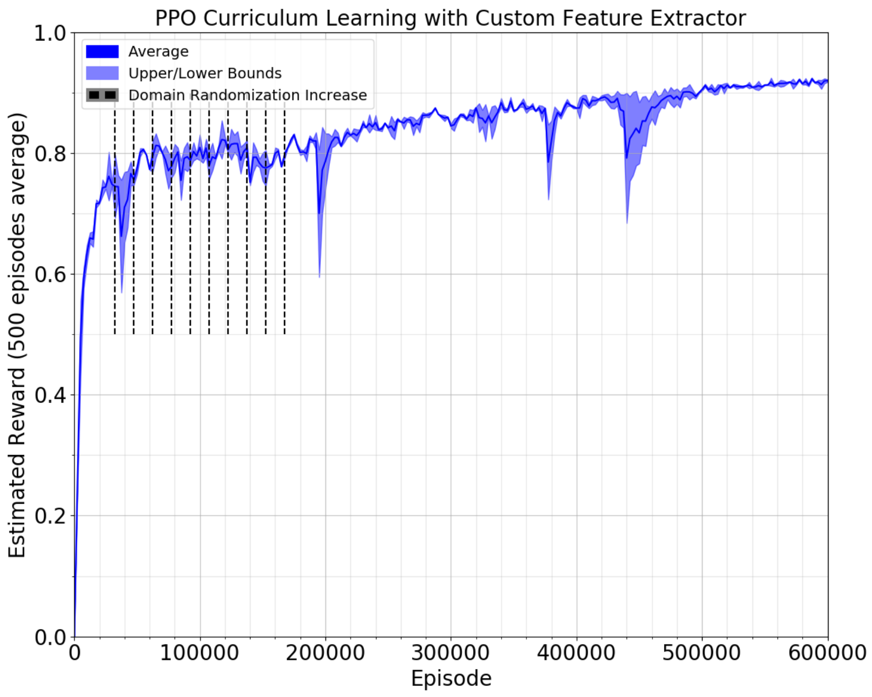


**Figure 5.8:** Graph showing the results after running the Proximal Policy Optimization (PPO) algorithm with our **custom feature extractor** with **Curriculum Learning (CL)** in a Domain Randomized (DR) instance of our simulation environment for 600000 episodes (1.8 million environment interactions). The average, the upper bound and the lower bound are generated by two separate runs of this configuration. The estimated reward for the agent is measured every 2500 episode, where each estimate is an average calculated over 500 episodes. The dotted black vertical lines show where the amount of Domain Randomization (DR) is increased. Our Curriculum Learning (CL) strategy increases the amount of DR linearly from 0.0 to 1.0, where 0.1 is added every 15000 episode starting at 32500 and ending at 167500.

A plot showing both the results of training the PPO algorithm with a custom CNN in the DR and the CL instances of the simulation environment, can be seen in Figure 5.9. As can be seen in the plot, the agent trained with CL learns faster than the agent trained with full DR from start. However, after the first 100000 episodes, the difference between the agents decreases. Both graphs in the plot inhabits large fluctuations, but increase somewhat proportional to each other after the first 150000 episodes. The final results of each agent were nearly identical with little variance around an estimated reward of 0.92.

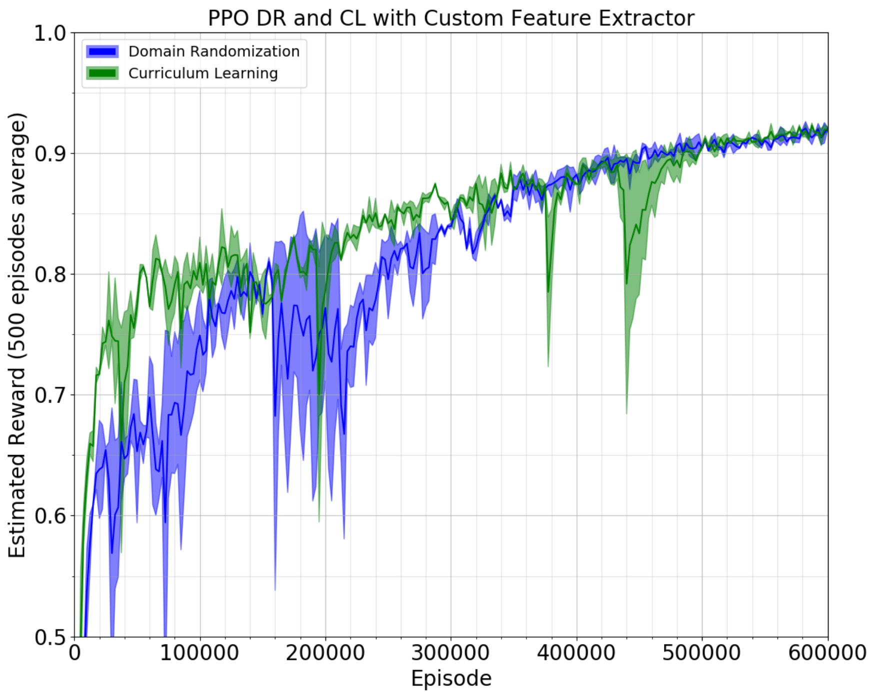


**Figure 5.9:** Graph showing a comparison between the Proximal Policy Optimization (PPO) algorithm with **custom feature extractor** in the **Domain Randomized (DR)** and the **Curriculum Learning (CL)** version of our simulation environment. The average, the upper bound and the lower bound are generated by two separate runs of both configurations. The blue graph shows the results produced in the DR environment, and the green graph shows the results produced in the CL environment.

Training the PPO algorithm with a pre-trained CNN component on an instance of the simulation environment incorporating Curriculum Learning (CL) produced the results seen in Figure 5.10. The same CL strategy was used for this configuration as for the configuration using a custom CNN component. Likewise, the points of randomization increase are shown in the plot as dotted black lines along the axis of the corresponding episode. The agent seemed to learn quickly and adapted well to increases in DR strength, however, some medium fluctuations can be seen in the plot. The final estimated reward obtained by this configuration was roughly 0.89, resulting in slightly poorer results than those obtained by training the same network in an environment with full DR from start.

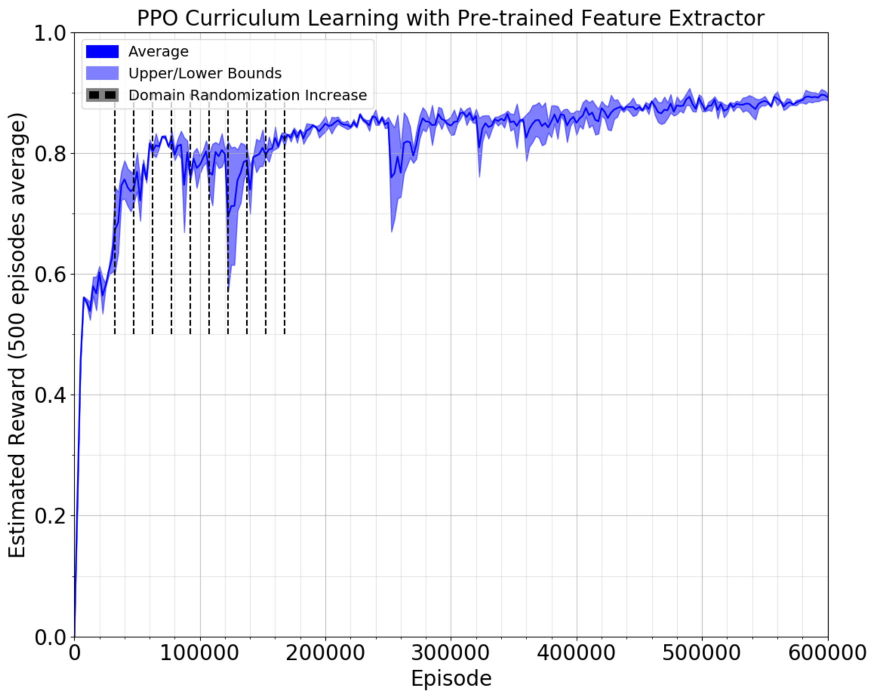


**Figure 5.10:** Graph showing the results after training the Proximal Policy Optimization (PPO) algorithm with a **pre-trained feature extractor** and **Curriculum Learning (CL)** in a Domain Randomized (DR) instance of our simulation environment for 600000 episodes (1.8 million environment interactions). The average, the upper bound and the lower bound are generated by two separate runs of this configuration. The estimated reward for the agent is measured every 2500 episode, where each estimate is an average calculated over 500 episodes. The dotted black vertical lines show where the amount of DR is increased. Our CL strategy increases the amount of DR linearly from 0.0 to 1.0, where 0.1 is added every 15000 episode starting at 32500 and ending at 167500.

A plot showing both the results of training the PPO algorithm with a pre-trained CNN component in the DR and the CL instances of the simulation environment can be seen in Figure 5.11. As can be seen in the plot, the CL configuration seemed to learn more rapid for the first 80000 episodes, but then approached the results obtained by the DR agent and they increased proportionally throughout the rest of the training. The rewards produced by the CL configuration seemed to be more prone to fluctuations. The final results of each learning configuration were comparable, with the DR configuration achieving slightly better results.

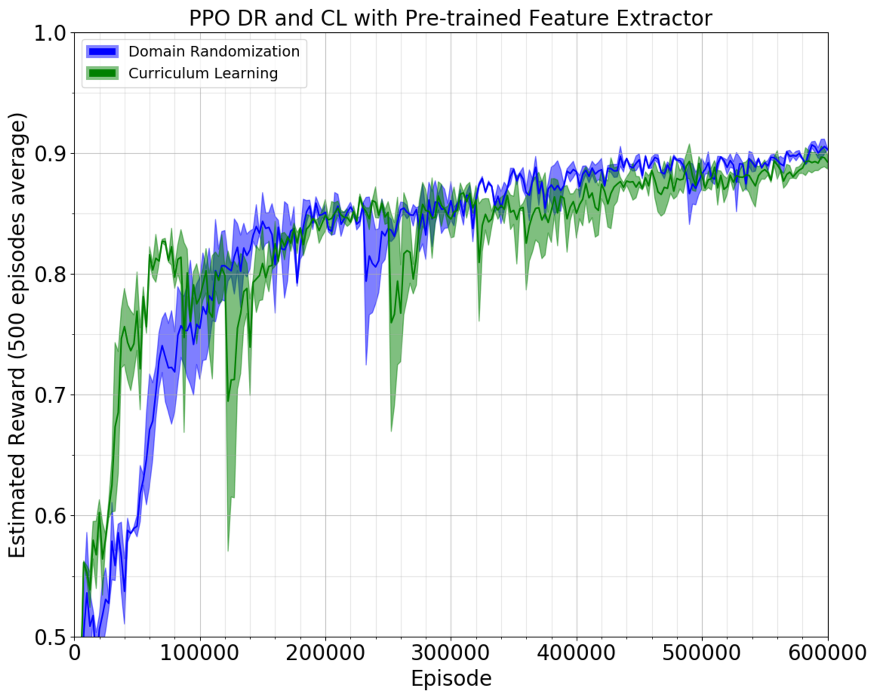


**Figure 5.11:** Graph showing a comparison between the Proximal Policy Optimization (PPO) algorithm with **pre-trained feature extractor** in the **Domain Randomized (DR)** and the **Curriculum Learning (CL)** versions of our simulation environment. The average, the upper bound and the lower bound are generated by two separate runs of both configurations. The blue graph shows the results produced in the DR environment, and the green graph shows the results produced in the CL environment.

### 5.2.4 General Observations

Overall, the different approaches regarding network architectures and training regimes all yielded satisfactory results in our simulation experiments with final estimated rewards ranging from roughly 0.89 to 0.94. As can be seen in Table 5.3, these results indicate that all agents estimates gripper poses that on average are six centimeters and twelve degrees or closer to a good gripper pose. The agents concerned with the base environments produced higher final estimated rewards than the agents coping with Domain Randomization (DR). This coincide with the assumption that learning more general concepts over diverse data is a more complex task. Across all simulation experiments, minor and a few major fluctuations occurred in the quality of the estimated gripper poses, with the agents using a network architecture building on the AlexNet CNN producing fewer and smaller fluctuations. We suspect that these fluctuations occur due to large weight updates in the network based on poorly distributed experiences, where a majority of these experiences require similar changes to be made, resulting in exaggerated updates. Given sufficient hardware capabilities, we suspect that increasing the buffer and mini-batch sizes along with decreasing the epsilon ($\epsilon$) hyperparameter can ease this effect. Despite some fluctuations in the quality of the estimated grasp poses, the agents quickly stabilize and rebounds. Comparisons between configurations using DR and CL show indications of faster learning during the first few stages of randomization increase, but this head start declines before reaching full DR in both experiments. Additionally, the configurations using CL do not seem to produce better final policies in our experiments. We believe that the reason why our CL agents did not end up learning faster or producing better policies than the agents trained with full DR from start, is that the CL strategy chosen did not benefit the agents sufficiently during training.

## 5.3 Real World Experiments

In our real world experiments, the different agents trained in the gripper pose estimation simulation environment were evaluated over a set of semi-compliant objects. The objects were molded in silicone as described in section 4.3 and can be seen in Figure 4.10. For all the experiments conducted in our real world setup, the agents evaluated were only trained in simulation and no fine tuning or domain adaption was performed. Early trials of the different agents revealed that the agents trained in non-randomized environments produced unsatisfactory gripper pose estimations which discriminated minimally between different visual observations, and no successful grasp poses were estimated by these agents. This observation led us to terminate the experiments regarding the agents trained in non-randomized environments. Additionally, we observed that all the agents benefited from the addition of thresholded Perlin Noise (Perlin, 1985) maps, like the ones used for the depth renderings in our simulations. We suspect this can be derived from the fact that the networks were trained on certain distributions of zero values in the depth images. To minimize the difference in estimated gripper poses for the same states in these experiments, the Perlin Noise added to the depth images was kept constant.

To ensure good indications for the quality of the different policies, ten grasp attempts were performed for each agent for each object, resulting in a total of 400 grasp attempts for the first part of the real world experiments. A visualization of the different random positions and rotations used during evaluation can be seen in Figure 5.12, where each unique object configuration is shown in one image per object. The second part of the experiments concerned five gripper pose estimations for four novel objects not trained on in simulation. The novel objects along with their random initial positions and rotations can be seen in Figure 5.13.

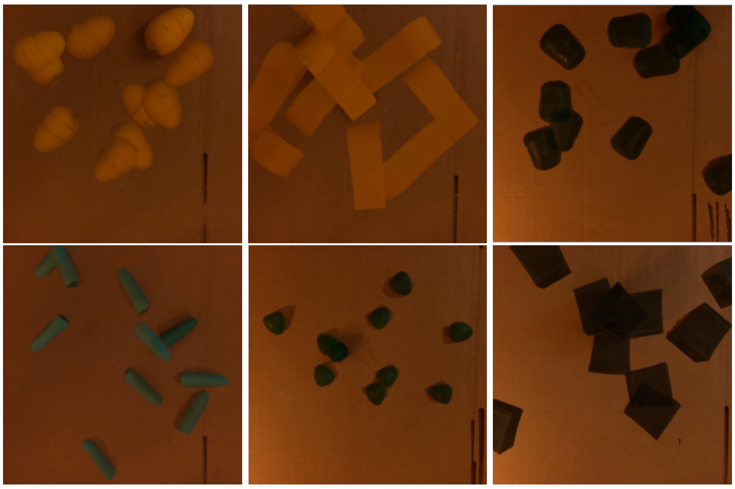


**Figure 5.12:** Visualization of the ten random initial configurations for each object used in our real world experiments. For each object, each unique position and rotation is shown per image, visualizing the distribution of our test set. The objects shown are respectively left to right, top row; Avocado, Fish Fillet and Paprika, bottom row; Pickle, Strawberry and Speaker.

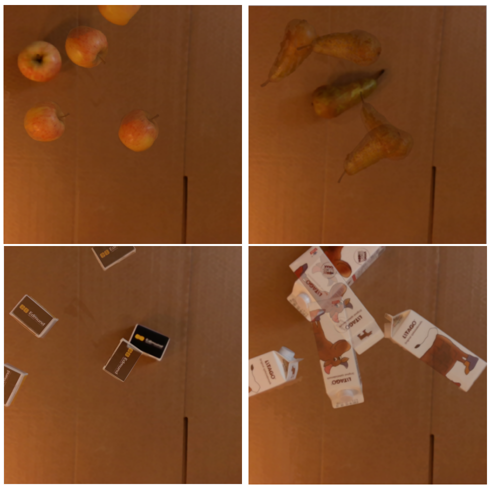


**Figure 5.13:** Visualization of the five random initial configurations for each object used in our real world experiments for novel objects not seen during training. For each object, each unique position and rotation is shown per image, visualizing the distribution of the test set. The objects shown are respectively left to right, top row; Apple and Pear, bottom row; Match Box and Milk Carton.

In contrast to our simulation, where the agents were rewarded based on the offset from good grasping poses with sufficient margins both regarding positional and rotational errors, the error measurements in our real world experiments were based on the minimal offsets required for obtaining successful grasps. For both the first and the second part of our real world experiments, the manually measured offsets from viable grasps were rounded to the nearest 0.5 centimeters and the nearest 5 degrees.

An example of a successful grasp pose estimation sequence for the Speaker object can be seen in Figure 5.14, where a viewer's perspective, the visual observations perceived by the agent and the resulting grasp pose can be seen. Common for most of the gripper pose estimation sequences and the sequence shown in this figure, is that the agents iteratively improves the end-effector's position and rotation, exploiting the several steps of an episode.

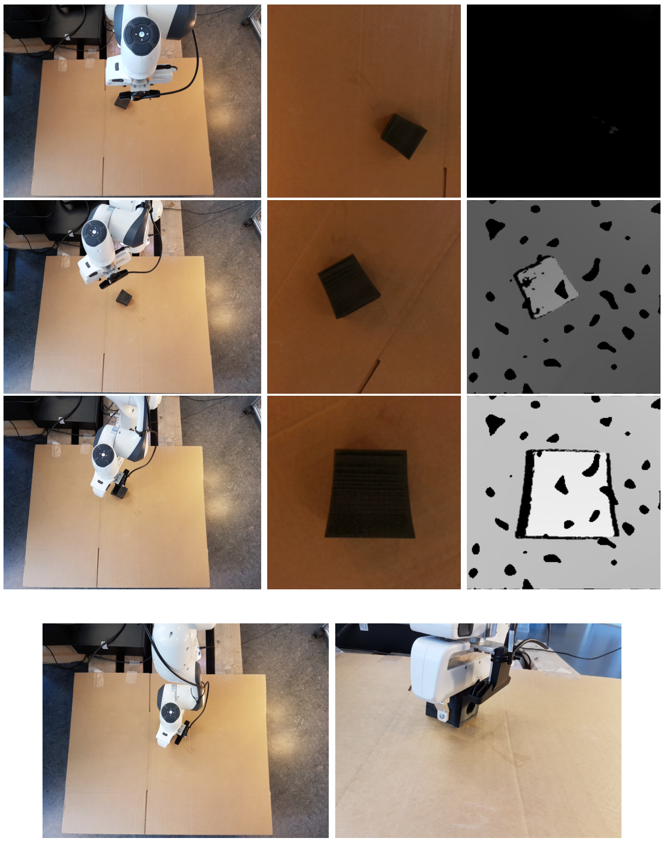


**Figure 5.14:** Photos and visual observations from an episode in our real world setup where the agent estimates a successful gripper pose for the Speaker object. The first column shows the episode seen from a viewer's perspective. The second and third columns show the RGB and the Depth visual observations perceived by the Deep Reinforcement Learning (DRL) agent. Finally, the last two images show the resulting grasping pose from top and side view.

### 5.3.1 Results and Discussion

The results from the first part of our experiments can be seen in Table 5.4 and Table 5.5, where the grasping success rates of each agent are presented in the first table, and the mean and standard deviation for each configuration is shown in the second table. As can be seen by these results, the agent which used our custom CNN component trained in simulation with full DR achieved the best overall results with a grasp success rate of 60% and 88.33% of the grasps being either successful or within one centimeter and five degrees of a viable grasp. This coincide with the results obtained in our simulation experiments, where the agents using our custom CNN component achieved some of the best policies. Interestingly, the agent using our custom CNN component along with the Curriculum Learning (CL) strategy, did not obtain equally good results in these experiments despite achieving equally good policies in our simulation experiments. However, as can be seen in Table 5.5, the custom CL agent estimated gripper poses with errors whose mean and standard deviation were comparable to the custom DR agent.

| **Real World Experiments - Grasp Success Rates** | | | | | | |
|---|---|---|---|---|---|---|
| | Pre-trained DR | | | Pre-trained CL | | |
| Object | M1 | M2 | M3 | M1 | M2 | M3 |
| Fish Fillet | 40% | 70% | 100% | 10% | 10% | 90% |
| Avocado | 40% | 50% | 100% | 20% | 30% | 60% |
| Pickle | 60% | 60% | 90% | 10% | 20% | 60% |
| Paprika | 40% | 50% | 70% | 30% | 40% | 60% |
| Strawberry | 70% | 90% | 90% | 30% | 90% | 90% |
| Speaker | 0% | 70% | 80% | 30% | 40% | 90% |
| **Total** | 41.66% | 65.00% | 88.33% | 21.66% | 38.33% | 75.00% |
| | Custom DR | | | Custom CL | | |
| Object | M1 | M2 | M3 | M1 | M2 | M3 |
| Fish Fillet | 40% | 70% | 90% | 10% | 10% | 90% |
| Avocado | 70% | 90% | 100% | 70% | 70% | 100% |
| Pickle | 100% | 100% | 100% | 40% | 90% | 100% |
| Paprika | 70% | 90% | 100% | 70% | 80% | 100% |
| Strawberry | 30% | 80% | 100% | 0% | 60% | 100% |
| Speaker | 20% | 90% | 100% | 0% | 60% | 90% |
| **Total** | **60.00%** | **88.33%** | **98.33%** | 36.66% | 73.33% | 96.66% |

**Table 5.4:** Table showing grasp success rates in our real world experiments for the four different training and network configurations after only training in simulation. Both Pre-trained DR (top-left) and Pre-trained CL (top-right) used the same network architecture, building on a pre-trained feature extractor, where Pre-trained DR was trained in the Domain Randomization (DR) version of our simulation and Pre-trained CL was trained in the Curriculum Learning (CL) version. Furthermore, Custom DR (bottom-left) and Custom DL (bottom-right) used the network architecture incorporating our custom feature extractor, where Custom DR was trained in a DR version of our simulation and Custom CL was trained in the CL version. Each configuration was evaluated on the six objects trained on in simulation with ten different initial positions and rotations. The first measurement **M1** is a binary measurement of how many grasps were successful, enabling the gripper to grasp and lift the object. The second measurement **M2** shows the percentage of attempts that were not classified as successful grasps, but were within 1 centimeter and 5 degrees of success. Finally, the measurement **M3** shows the percentage of grasps that were within 3 centimeters and 10 degrees of a successful grasp. The **Total** row shows the measurements for each configuration for the whole dataset.

| **Real World Experiments - Mean and Standard Deviation** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Pre-trained DR | | | | Pre-trained CL | | | |
| Object | P$\mu$ | P$\sigma$ | R$\mu$ | R$\sigma$ | P$\mu$ | P$\sigma$ | R$\mu$ | R$\sigma$ |
| Fish Fillet | 0.95cm | 1.08cm | 0.0° | 0.0° | 2.25cm | 1.45cm | 0.0° | 0.0° |
| Avocado | 1.05cm | 0.93cm | 0.5° | 1.5° | 2.25cm | 1.63cm | 0.0° | 0.0° |
| Pickle | 1.05cm | 1.37cm | 2.5° | 7.5° | 4.50cm | 5.82cm | 5.0° | 9.5° |
| Paprika | 2.70cm | 3.82cm | 2.5° | 4.0° | 3.50cm | 5.37cm | 6.0° | 8.0° |
| Strawberry | 0.85cm | 2.23cm | 0.5° | 1.5° | 2.75cm | 6.93cm | 0.5° | 1.5° |
| Speaker | 1.20cm | 0.98cm | 3.0° | 4.6° | 2.10cm | 3.40cm | 3.8° | 6.2° |
| **Total** | 1.30cm | 2.12cm | 1.5° | 4.2° | 2.89cm | 4.68cm | 2.6° | 6.2° |
| | Custom DR | | | | Custom CL | | | |
| Object | P$\mu$ | P$\sigma$ | R$\mu$ | R$\sigma$ | P$\mu$ | P$\sigma$ | R$\mu$ | R$\sigma$ |
| Fish Fillet | 0.7cm | 1.33cm | 0.0° | 0.0° | 0.95cm | 1.13cm | 0.5° | 1.5° |
| Avocado | 0.35cm | 0.63cm | 1.5° | 3.2° | 0.45cm | 0.69cm | 0.5° | 1.5° |
| Pickle | 0.00cm | 0.00cm | 0.0° | 0.0° | 0.55cm | 0.52cm | 0.0° | 0.0° |
| Paprika | 0.35cm | 0.55cm | 0.0° | 0.0° | 0.40cm | 0.70cm | 0.0° | 0.0° |
| Strawberry | 0.75cm | 0.51cm | 2.0° | 3.3° | 0.95cm | 0.47cm | 0.0° | 0.0° |
| Speaker | 0.65cm | 0.45cm | 0.0° | 0.0° | 0.95cm | 0.76cm | 3.0° | 9.0° |
| **Total** | **0.47cm** | **0.75cm** | **0.6°** | **2.1°** | 0.71cm | 0.78cm | 0.7° | 3.9° |

**Table 5.5:** Table showing mean and standard deviation for the error in our real world experiments for the four different training and network configurations after only training in simulation. Each configuration was evaluated on the six objects trained on in simulation with ten random initial positions and rotations. The first measurement **P**$\mu$ shows the mean positional offset from a successful grasp in centimeters. The second measurement **P**$\sigma$ shows the standard deviation of the positional offset. The third measurement **R**$\mu$ shows the mean rotational offset from a successful grasp in degrees, and the last measurement **R**$\sigma$ shows the rotational offset's standard deviation. The **Total** row shows the different measurements for the whole dataset.

Common for all the agents in these experiments were poor grasp success rates for the Speaker object, where we suspect that this is caused by the low clearance for this object. However, as can be seen in the results obtained with the M1 and the M2 measurements in Table 5.4, along with the mean positional offsets seen in Table 5.5, the gripper poses estimated were close to viable. The different clearance measurements for all the moulded objects can be seen in Table 5.6, where the clearances were measured based on a good rotation and some objects contains two measurements due to the objects being graspable from multiple orientations.

| **Real World Object Gripper Clearance** | |
|---|---|
| Object | Clearance |
| Fish Fillet | 2.0 cm |
| Avocado | 1.8 cm |
| Pickle | 5.5 cm |
| Paprika | 2.6 cm & 0.8 cm |
| Strawberry | 4.7 cm & 4.3 cm |
| Speaker | 2.0 cm & 0.7 cm |

**Table 5.6:** Table showing the clearance between the different objects used in our real world experiments and the gripper's actuators in centimeters. Two clearance measurements are included for the objects Paprika, Strawberry and Speaker, as they can be grasped from multiple orientations.

Overall, for the first part of our real world experiments, the agents using our custom CNN component as a feature extractor performed better than the agents using a pre-trained instance of AlexNet as a feature extractor. This also coincide with our simulation experiments where all the agents using the AlexNet CNN component achieved final policies that obtained lesser rewards. As mentioned in subsection 5.2.3, we suspect that this difference stems from the fact that AlexNet was trained purely on RGB images, and the fact that the first two layers were not tuned during training, potentially resulting in poor features being extracted from the depth images.

To give some indications to the similarities between the results obtained in our real world setup and the rewards obtained in simulation, some rough estimations were performed based on the gripper poses estimated. As stated, the error measurements in these experiments are based on the offsets from a viable grasp pose, as opposed to a good grasp pose in the simulation. However, the clearance measurements for the different objects seen in Table 5.6, can be used along with the mean offsets seen in Table 5.5 to roughly estimate the average distance to a centered pose within a centimeter for each object. Additionally, we observed that the viable grasps required an average of roughly 3-5 degrees of rotation for obtaining good rotational clearances. Based on these estimations an indication of the rewards can be produced, resulting in a value of 0.91 $\pm$ 0.02 for both the custom DR agent and the custom CL agent, which both achieved 0.92 in simulation. These estimations seem to correlate for the agents building on a pre-trained feature extractor as well, where the Pre-trained DR agent achieved 0.90 in simulation and 0.89 $\pm$ 0.02 in our real world experiments, and the pre-trained CL agent achieved 0.89 in simulation and 0.85 $\pm$ 0.02 in the real world. Although having notable differences in the amount of successful grasps, these estimations show that the gripper poses estimated by the custom DR and the custom CL agents produce grasp poses with similar average offsets, where this can also be derived from Table 5.5. This coincide with the results obtained in simulation where the agents achieved roughly the same estimated rewards. We suspect, that more accurate error measurements along with more gripper pose estimations per object would further the similarities between simulation and real world results.

The agents evaluated for all the aforementioned experiments were trained on a relative small set of different objects and evaluated on semi-compliant replicas of these. However, a final set of experiments were conducted to reveal some indications as to the capability of generalizing beyond these. A set of five different objects both semi-compliant and relatively rigid, which was not seen during training, was presented to the agents and evaluated over five different grasps. The different random initial positions for these novel objects can be seen in Figure 5.13, and the resulting grasping accuracy of the experiments can be seen in Table 5.7. Interestingly, the agent obtaining the highest accuracy for the seen objects did not obtain the highest accuracy for these unseen objects. Another interesting observation is that both the pre-trained CL and the custom CL agents obtain higher scores for the unseen objects. We observed that for the estimated gripper poses in the experiments regarding seen objects, the CL agents often estimated grasps that were above the small objects; Pickle and Strawberry. This did not occur in the experiments concerning the unseen objects, which we suspect comes from the objects being larger and thus covering more of the depth images while having larger ranges for valid estimations in the height dimension. We suspect that these observations, along with relatively few grasp attempts, can be the reason why the CL agents perform better on the novel objects. Considering the relatively small set of different objects presented to the agents during training, these results were noteworthy, with all the agents estimating gripper poses with at least 75% of the attempts being within three centimeters and ten degrees of a valid grasp.

| **Real World Experiments on Novel Objects** | | | | | | |
|---|---|---|---|---|---|---|
| | Pre-trained DR | | | Pre-trained CL | | |
| Object | M1 | M2 | M3 | M1 | M2 | M3 |
| Apple | 0% | 40% | 100% | 20% | 40% | 60% |
| Matchbox | 60% | 80% | 80% | 80% | 80% | 100% |
| Pear | 20% | 20% | 60% | 60% | 60% | 60% |
| Milk Carton | 0% | 100% | 100% | 20% | 80% | 100% |
| **Total** | 20% | 60% | **85%** | 45% | 65% | 80% |
| | Custom DR | | | Custom CL | | |
| Object | M1 | M2 | M3 | M1 | M2 | M3 |
| Apple | 40% | 60% | 80% | 60% | 60% | 80% |
| Matchbox | 40% | 80% | 100% | 80% | 80% | 80% |
| Pear | 20% | 20% | 40% | 60% | 80% | 80% |
| Milk Carton | 60% | 80% | 100% | 20% | 60% | 60% |
| **Total** | 40% | 55% | 75% | **55%** | **70%** | 75% |

**Table 5.7:** Table showing grasp success rates in our real world experiments for the **novel objects** not seen during training, for the four different training and network configurations after only training in simulation. Both Pre-trained DR (top-left) and Pre-trained CL (top-right) used the same network architecture, building on a pre-trained feature extractor, where Pre-trained DR was trained in the Domain Randomization (DR) version of our simulation and Pre-trained CL was trained in the Curriculum Learning (CL) version. Furthermore, Custom DR (bottom-left) and Custom DL (bottom-right) used the network architecture incorporating our custom feature extractor, where Custom DR was trained in a DR version of our simulation and Custom CL was trained in the CL version. Each agent configuration was evaluated on four different objects that were not trained on in simulation with five different initial positions and rotations. The first measurement **M1** is a binary measurement of how many grasps were successful, enabling the gripper to grasp and lift the object. The second measurement **M2** shows the percentage of attempts that were not classified as successful grasps, but were within 1 centimeter and 5 degrees of success. Finally, the measurement **M3** shows the percentage of grasps that were within 3 centimeters and 10 degrees of a successful grasp. The **Total** row shows the measurements for each configuration for the whole dataset.

### 5.3.2 General Observations

To our expectations, the agents trained in a non-randomized version of our simulation environment did not produce satisfactory results when transferred to our real world environment. Consequently, these agents were omitted from further experiments. The other agents evaluated seemed to have a consistent error of one centimeter along the negative y-axis and 0.75 centimeters along the negative z-axis, consequently, adjustments were made to the final actions to correct this. We assume that this effect stems from poor correlation between the camera position on the gripper's end-effector in the simulation and the camera's position in our real world setup. Another observation seen across all agents was that lighting conditions impacted the performance of the agents to a certain extent. Interestingly, different lighting conditions seemed to impact the agents that built on a pre-trained feature extractor more than the agents using our custom CNN architecture. We believe that this stems from our decision to not tune the first two CNN layers of the AlexNet component, and suspect that the lighting conditions in our real world setup differs from the lighting conditions present in the ImageNet (ImageNet, 2018) dataset. However, as both architectures were impacted by lighting conditions in these experiments, we suspect that incorporating more expressive Domain Randomization (DR) aspects regarding lighting, such as light intensity, shadows, number of light sources etc., will produce better results after transfer learning. Finally, we observed that the agents struggled more with the estimation of good gripper poses for objects that were almost as wide as the full extent of the gripper's actuators. This was not surprising as the error tolerance decreases severally for these objects where even slight displacements can result in gripper poses going from viable to failed grasps. We suspect that the utilization of robotic gripper actuators which are more customized for grasping of compliant objects, and that also adapts moderately to minor collisions instead of terminating, can increase the grasp success rates of the agents. Despite sensitivity to lighting conditions and the use of a gripper which is not optimized for grasping semi-compliant objects, many successful and nearly successful gripper pose estimations were produced by our agents. A collection of six grasp attempts can be seen in Figure 5.15, where the minor offsets required for moving from nearly successful to successful can be seen. Additionally, the complete list of the results obtained in our real world experiments can be seen in Appendix B.

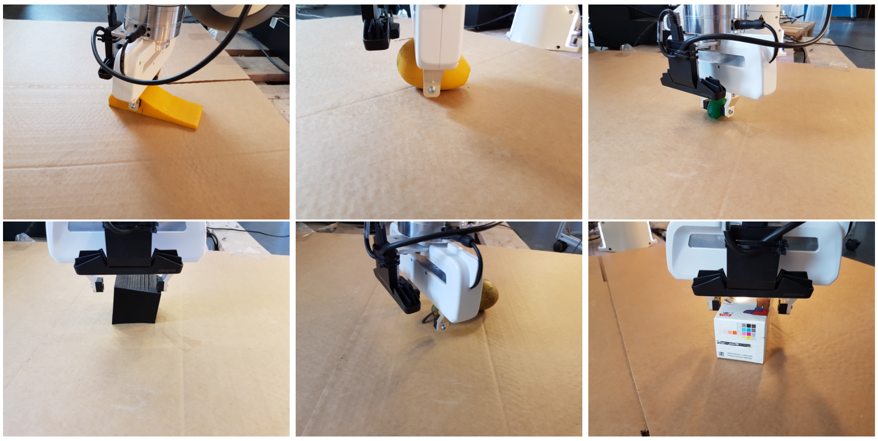


**Figure 5.15:** Photographs of different successful (top) and nearly successful (bottom) gripper pose estimations taken during our real world experiments. For the grasp attempts shown in these images, nearly perfect rotations were estimated. As can be seen in the bottom row, these attempts only miss a viable grasp pose by a few centimeters. Additionally, the gripper robot halts when the end-effector collides, resulting in the elevated end-positions in the bottom left and bottom right images.

Chapter 6

# Conclusion and Future Work

This chapter presents a brief summary of the problem formulation, motivation and the work of this thesis. Following, the key experimental results are discussed in relation to our research questions along with our contributions to the field of Deep Reinforcement Learning (DRL) and dexterous robotic manipulation. Finally, a brief section covering a collection of further interesting alterations and extensions to our work, is presented.

## 6.1 Conclusion

The task of using Deep Reinforcement Learning (DRL) in the context of gripper pose estimation for semi-compliant objects is rather novel. However, work investigating related tasks such as gripper pose estimation for rigid objects or vision guided robotic manipulation in general is of great interest. A structured state-of-the-art literature review, investigating work applying both Deep Learning (DL) and DRL to related tasks, was performed. The review revealed the DRL algorithms Deep Deterministic Policy Gradients (DDPG) and Proximal Policy Optimization (PPO) to be interesting candidates for experiments. A simulation environment for training and evaluating DRL agents on the problem of gripper pose estimation was developed along with support for Domain Randomization (DR) and Curriculum Learning (CL). The simulation environment was developed to loosely mimic the real world setup and incorporated rigid objects.

We performed a set of research experiments to reveal 1) which of the DRL agents that produce the most satisfactory results based on a set of criteria including, training times and final policies, and 2) to answer our research questions:

**RQ1:**
*Can the benefits of Domain Randomization enable an agent solely trained in simulation to produce satisfactory results in a real world environment?*

**RQ2:**
*Is training in a simulation environment with rigid objects sufficient for developing policies for grasping semi-compliant objects in the real world?*

**RQ3:**
*Can the use of Curriculum Learning decrease the amount of training needed for developing good policies in a randomized simulation environment?*

The research experiments regarding the selection of one of the two DRL agents (1), yielded results that strongly suggested PPO to be more suitable for the problem definition of grasp pose estimation. Consequently, PPO was chosen for our second set of research experiments.

To evaluate the implemented PPO agents in our real world setup, the agents were first trained in our simulation environment. Three different training regimes were chosen for these experiments. The first regime concerned training in a base instance without Domain Randomization (DR), the second regime concerned training in an instance with full Domain Randomization (DR), and the last regime concerned training with Curriculum Learning (CL). These training regimes produced the prerequisites necessary for gathering data for our first and second research questions, while the second and third training regimes produced data for our third research question.

Research experiments were conducted investigating the precision of different gripper pose estimations in our real world setup, produced by several variations of the PPO algorithm that were only trained in simulation. The trained agents produced by learning with the DR and CL regimes produced the data necessary for discussing our first and second research questions. The first training regime, concerning training an DRL agent in a base version of our simulation, was evaluated in our real world setup, creating a basis for observing the benefits of DR. Our experiments, in line with Pinto et al. (2017), revealed that the DRL agents trained without DR, despite obtaining near perfect results in the simulation environment, did not produce satisfactory results in the real world. The gripper poses estimated by these agents only showed marginal discrimination across different states and no successful grasps were executed. However, our experiments evaluating agents trained with DR showed promising results, where the grasp poses estimated by the best agent achieved a success rate of 60% on our test set, with 88.3% of the grasp attempts being either successful, or within one centimeter and five degrees of a successful grasp. Additionally, the mean positional and rotational offsets from a gripper pose that would have resulted in a valid grasp were respectively; 0.47 centimeters with a standard deviation of 0.75 centimeters and 0.6 degrees with a standard deviation of 2.1 degrees. The gripper used for these experiments was a crude two finger gripper, leading many of the estimated grasp poses to fail on account of minor offsets, where the griper aborts grasps with minor collisions instead of having fingers that adapts moderately to the objects during grasp initiation. Based on the results obtained, we suspect that the use of a gripper that is more customized for grasping of compliant and semi-compliant objects would lead to even higher grasping accuracies. In light of this, we deem the results obtained by our best agent to be promising.

Analysis of the results obtained in our real world experiments showed a notable correlation between the results obtained in simulation and the accuracy in our real world setup. This suggests that agents learning better policies in our simulation will produce better results in our real world setup. The best PPO agent using DR showed some weaknesses regarding the lighting conditions and positional reasoning. However, we suspect that these flaws can be mitigated by changes to the lighting DR aspects and changes to the camera position in both simulation and real world. Based on our results and these hypothesis, we strongly suspect our first research question **RQ1** to be true, stating that DRL agents trained solely in simulation can produce satisfactory results in a real world environment by exploiting Domain Randomization (DR). The same results also indicate our second research question **RQ2** to be true, stating that training in a simulation incorporating rigid objects is sufficient for developing policies for grasping semi-compliant objects in the real world. However, we observed some dubious grasps, which given more compliant objects, would most likely not be successful. These observations lead us to believe that training agents in a simulation that incorporates physics that enables the simulation of compliant objects and the grasping of these would be, if not crucial, at least beneficial for learning to grasp highly compliant objects. Based on the results obtained by our experiments concerning the training of agents in DR and CL simulation environments, we conclude that there are not sufficient indications in our results to claim that our third research question **RQ3** is true, stating that CL reduces the amount of training needed for learning good policies in a DR environment. We believe that an explanation for why training with CL did not outperform training in a DR environment from scratch, can be that the learning strategy chosen does not benefit the agent sufficiently during training.

To the best of our knowledge, a novel approach using Curriculum Learning (CL) for reducing the amount of training necessary for learning good policies in highly domain randomized environments was proposed. However, the CL strategy explored in this thesis failed to meet the expectations regarding the reduction of training times and the improved quality of final policies. Results supporting the hypothesis that DRL agents trained solely in simulations can produce satisfactory results in real world environments were obtained. Our work is in line with the results achieved in the work of Koch et al. (2018), where PPO was deemed more reliable and time-efficient than DDPG for learning a problem with a high dimensional state space and a Continuous Action Space (CAS). Furthermore, our work is in line with the results of Tobin et al. (2017a,b), Pinto et al. (2017) and Peng et al. (2017), where the use of Domain Randomization (DR) improved the results obtained in our real world environment after transfer from simulation. However, the results obtained in our experiments differed from the results of Zhang et al. (2017), where their algorithm produced poor estimations without the use of domain adaption. In contrast to the work of Tobin et al. (2017a,b), our work investigated the prospects of dividing the gripper pose estimation task into a sequence of three estimations. Our results indicate the benefits of this approach, where it is shown that instead of estimating the full pose from only one observation, the agents gradually fine-tune their estimates.

Our results along with the work of Pinto et al. (2017) and Inoue et al. (2017) support the claim that the use of DRL has great potential for the field of dexterous robotic manipulation. Ultimately, the results obtained in our thesis shows that the training of Deep Reinforcement Learning (DRL) agents in a simulator holds great potential for appliances in the real world, where many satisfactory and close 6-DoF gripper pose estimations were produced without any fine tuning or domain adaption after transfer.

## 6.2 Future Work

The simulation environment currently incorporates no physics, and outputs rewards as described in subsection 4.2.2. To increase the benefits of using a simulator in regards to transfer learning, we recommend some extensions to the simulation environment, which enables the simulation of non-rigid physics, as well as friction physics. These extensions will enable the environment to simulate grasping of semi-compliant and compliant objects, and thus the current rewards can be used for initial guidance combined with rewards for successful simulated grasps.

Compliant objects deform both during a grasp initiation and during the execution of a grasp. Consequently, it can be assumed that receiving additional inputs during the grasp execution, incorporating information about the deformation or other related behaviour of the object, can be valuable. Extensions enabling the inputs to the gripper agent to additionally incorporate tactile information in a visual form, providing the agent information about the touch points of the gripper fingers and the deformation of the object in the contact areas through the use of a gel based sensing technology, is therefore an intriguing aspect to consider for future work.

As the camera used for gathering the observations from our environment is mounted to the end-effector of the gripper, the placement of the camera relative to the gripper in simulation can be expected to be sensitive to errors. To reduce this sensitivity we believe that an interesting alteration to explore in further work, is the re-positioning of the mounted camera such that parts of the gripper, including the actuators, are shown in the visual observations of the agents. This was not an easy change to perform in our real world setup, but we suspect that this additional information might ease some of the errors regarding positional and rotational estimations.

Some of the the results covered in related work (Levine et al., 2016; Gu et al., 2016; Popov et al., 2017) show the benefit of having multiple agent instances explore the problem domain simultaneously, where good improvements are obtained from the inclusion of more agents. Consequently, we deem the investigation of using parallel experience gathering in simulation to be of interest in future work. Another interesting aspect to investigate regarding parallel experience gathering, is the use of a hybrid solution where both agents in simulation and in the real world setup explore the problem domain simultaneously.

As shown in our results and reiterated in our conclusion, the agents using our Curriculum Learning (CL) strategy did not end up learning faster or end up producing better policies than the agents learning with full Domain Randomization (DR) from start. We suggest that further research should be conducted investigating other strategies regarding the changes for each step of the CL sequence. For example, instead of increasing the extent of DR, one could increase the number of aspects being randomized for each CL step.

The benefits of dividing the gripper pose estimation task into a sequence has been shown in our results. However, the agent only reasons on one of the three observations included in the sequence for each step. We believe that an interesting area to investigate is the utilization of Recurrent Neural Networks (RNNs) for this task, enabling the agent to reason on multiple observations for each grasp attempt.

Finally, a last interesting concept to explore in future work is the use of procedurally generated objects in the simulation environment like the work of Tobin et al. (2017b). We suspect that the incorporation of such methods can both increase the grasp success rate for the pre-defined objects trained on, and for novel objects.

# Bibliography

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., Zaremba, W., 2017. Hindsight experience replay. CoRR abs/1707.01495.
URL `http://arxiv.org/abs/1707.01495`

Bellemare, M. G., Naddaf, Y., Veness, J., Bowling, M., 2012. The arcade learning environment: An evaluation platform for general agents. CoRR abs/1207.4708.
URL `http://arxiv.org/abs/1207.4708`

Bengio, Y., Louradour, J., Collobert, R., Weston, J., 2009. Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning. ICML ’09. ACM, New York, NY, USA, pp. 41–48.
URL `http://doi.acm.org/10.1145/1553374.1553380`

Blender, 2018.
URL `https://www.blender.org/`

Bullet, 2018.
URL `https://pybullet.org/wordpress/`

Çalli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., Dollar, A. M., 2015. Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols. CoRR abs/1502.03143.
URL `http://arxiv.org/abs/1502.03143`

CU, 2009. Robot learning lab.
URL `http://pr.cs.cornell.edu/grasping/rect_data/data.php`

Florensa, C., Held, D., Wulfmeier, M., Abbeel, P., 2017. Reverse curriculum generation for reinforcement learning. CoRR abs/1707.05300.
URL `http://arxiv.org/abs/1707.05300`

Franka Emika, 2018a.
URL `https://www.franka.de/`

Franka Emika, 2018b.
URL https://frankaemika.github.io/docs/

Gazebo, 2018. Robot simulation made easy.
URL http://gazebosim.org/

Géron, A., 2017. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media.

Ghadirzadeh, A., Maki, A., Kragic, D., Björkman, M., 2017. Deep predictive policy training using reinforcement learning. CoRR abs/1703.00727.
URL http://arxiv.org/abs/1703.00727

Gu, S., Holly, E., Lillicrap, T. P., Levine, S., 2016. Deep reinforcement learning for robotic manipulation. CoRR abs/1610.00633.
URL http://arxiv.org/abs/1610.00633

He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep residual learning for image recognition. CoRR abs/1512.03385.
URL http://arxiv.org/abs/1512.03385

ImageNet, 2018.
URL www.image-net.org/

Inoue, T., Magistris, G. D., Munawar, A., Yokoya, T., Tachibana, R., 2017. Deep reinforcement learning for high precision assembly tasks. CoRR abs/1708.04033.
URL http://arxiv.org/abs/1708.04033

Intel, 2018.
URL https://software.intel.com/en-us/realsense/sr300

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167.
URL http://arxiv.org/abs/1502.03167

Johns, E., Leutenegger, S., Davison, A. J., 2016. Deep learning a grasp function for grasping under gripper pose uncertainty. CoRR abs/1608.02239.
URL http://arxiv.org/abs/1608.02239

Koch, W., Mancuso, R., West, R., Bestavros, A., 2018. Reinforcement learning for UAV attitude control. CoRR abs/1804.04154.
URL http://arxiv.org/abs/1804.04154

Krizhevsky, A., Sutskever, I., Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C. J. C., Bottou, L., Weinberger, K. Q. (Eds.), Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp. 1097–1105.
URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep
pdf

Kumra, S., Kanan, C., 2016. Robotic grasp detection using deep convolutional neural networks. CoRR abs/1611.08036.
URL `http://arxiv.org/abs/1611.08036`

Levine, S., Pastor, P., Krizhevsky, A., Quillen, D., 2016. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. CoRR abs/1603.02199.
URL `http://arxiv.org/abs/1603.02199`

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. CoRR abs/1509.02971.
URL `http://arxiv.org/abs/1509.02971`

Lin, Long-Ji, 1993. Reinforcement learning for robots using neural networks. In: Technical report, DTIC Document.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. CoRR abs/1602.01783.
URL `http://arxiv.org/abs/1602.01783`

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. A., 2013. Playing atari with deep reinforcement learning. CoRR abs/1312.5602.
URL `http://arxiv.org/abs/1312.5602`

Peng, X. B., Andrychowicz, M., Zaremba, W., Abbeel, P., 2017. Sim-to-real transfer of robotic control with dynamics randomization. CoRR abs/1710.06537.
URL `http://arxiv.org/abs/1710.06537`

Perlin, K., Jul. 1985. An image synthesizer. SIGGRAPH Comput. Graph. 19 (3), 287–296.
URL `http://doi.acm.org/10.1145/325165.325247`

Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W., Abbeel, P., 2017. Asymmetric actor critic for image-based robot learning. CoRR abs/1710.06542.
URL `http://arxiv.org/abs/1710.06542`

Popov, I., Heess, N., Lillicrap, T. P., Hafner, R., Barth-Maron, G., Vecerik, M., Lampe, T., Tassa, Y., Erez, T., Riedmiller, M. A., 2017. Data-efficient deep reinforcement learning for dexterous manipulation. CoRR abs/1704.03073.
URL `http://arxiv.org/abs/1704.03073`

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. CoRR abs/1707.06347.
URL `http://arxiv.org/abs/1707.06347`

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., 2014. Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32. ICML'14. JMLR.org.
URL `http://dl.acm.org/citation.cfm?id=3044805.3044850`

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D., 10 2017. Mastering the game of go without human knowledge 550, 354–359.

Silver, D. L., Guyon, I., Taylor, G., Dror, G., Lemaire, V., 2012. Icml2011 unsupervised and transfer learning workshop. In: Proceedings of ICML Workshop on Unsupervised and Transfer Learning. Vol. 27 of Proceedings of Machine Learning Research. PMLR.
URL `http://proceedings.mlr.press/v27/silver12a/silver12a.pdf`

TensorFlow, 2018.
URL `https://www.tensorflow.org/`

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P., 2017a. Domain randomization for transferring deep neural networks from simulation to the real world. CoRR abs/1703.06907.
URL `http://arxiv.org/abs/1703.06907`

Tobin, J., Zaremba, W., Abbeel, P., 2017b. Domain randomization and generative models for robotic grasping. CoRR abs/1710.06425.
URL `http://arxiv.org/abs/1710.06425`

Todorov, E., Erez, T., Tassa, Y., 2012. Mujoco: A physics engine for model-based control. IEEE/RSJ International Conference on Intelligent Robots and Systems.
URL `https://homes.cs.washington.edu/~todorov/papers/TodorovIROS12.pdf`

Uhlenbeck, G. E., Ornstein, L. S., 1930. On the theory of the brownian motion. Physical Review 36 (5), 823841.

Unity, 2018.
URL `https://unity3d.com/`

Unity-ML, 2018.
URL `https://unity3d.com/machine-learning/`

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N., 2016. Sample efficient actor-critic with experience replay. CoRR abs/1611.01224.
URL `http://arxiv.org/abs/1611.01224`

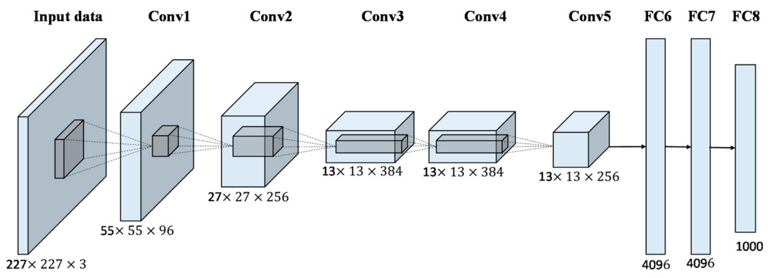
Watkins, C. J. C. H., Dayan, P., 1992. Q-learning. In: Machine Learning. pp. 279–292.

Wu, Y., Tian, Y., 2017. Training agent for first-person shooter game with actor-critic curriculum learning.

Zhang, F., Leitner, J., Milford, M., Corke, P., 2017. Sim-to-real transfer of visuo-motor policies for reaching in clutter: Domain randomization and adaptation with modular networks. CoRR abs/1709.05746.
URL `http://arxiv.org/abs/1709.05746`

# Appendix A

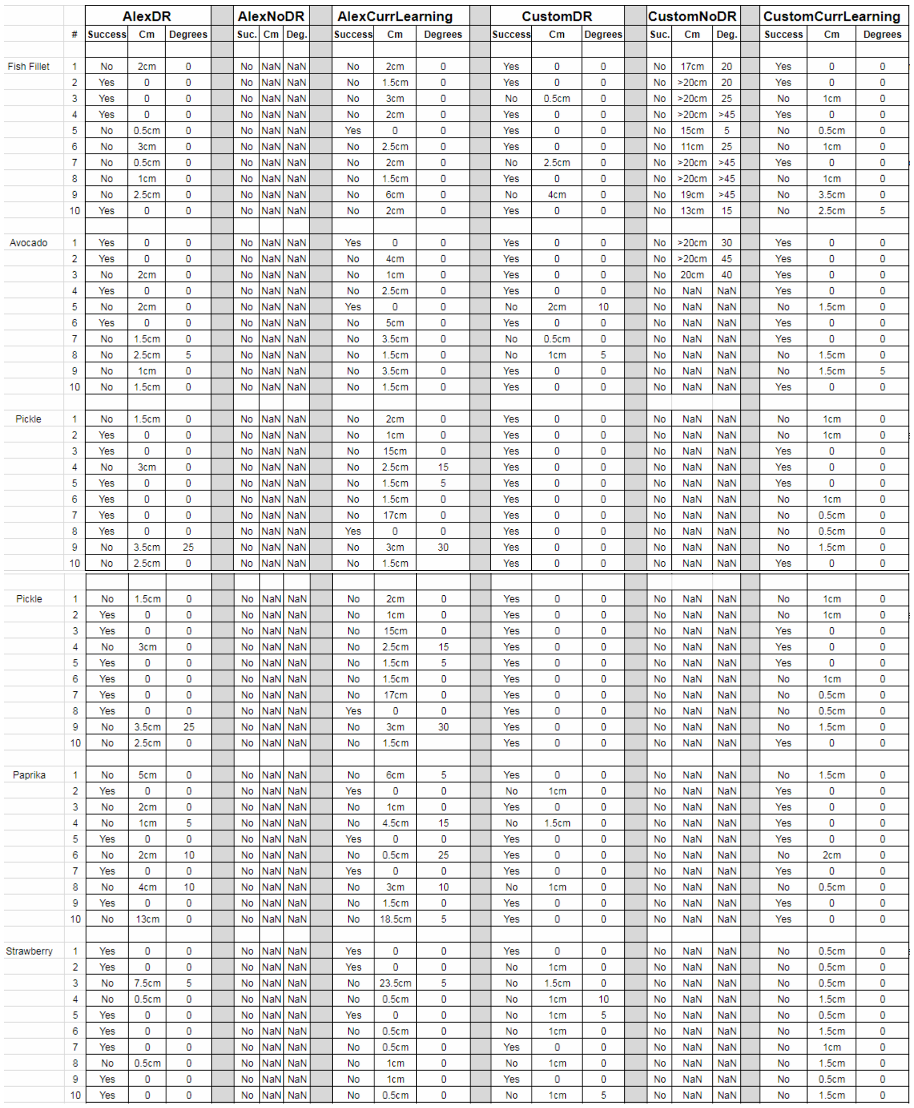
Below is the AlexNet (Krizhevsky et al., 2012) structure visualized with labeled layers and the pooling layers are omitted from the illustration.



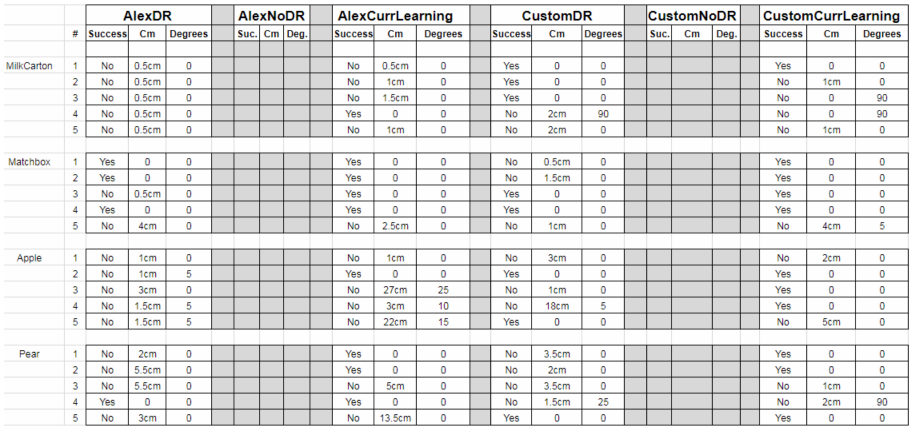
*Source: http://www.mdpi.com/remotesensing/remotesensing-09-00848/article_deploy/html/images/remotesensing-09-00848-g001.png*

# Appendix B

The complete list of the results obtained in our real world experiments.



| | | AlexDR | | | AlexNoDR | | | AlexCurrLearning | | | CustomDR | | | CustomNoDR | | | CustomCurrLearning | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | # | Success | Cm | Degrees | Suc. | Cm | Deg. | Success | Cm | Degrees | Success | Cm | Degrees | Suc. | Cm | Deg. | Success | Cm | Degrees |
| Fish Fillet | 1 | No | 2cm | 0 | No | NaN | NaN | No | 2cm | 0 | Yes | 0 | 0 | No | 17cm | 20 | Yes | 0 | 0 |
| | 2 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 0 | Yes | 0 | 0 | No | >20cm | 20 | Yes | 0 | 0 |
| | 3 | Yes | 0 | 0 | No | NaN | NaN | No | 3cm | 0 | No | 0.5cm | 0 | No | >20cm | 25 | No | 1cm | 0 |
| | 4 | Yes | 0 | 0 | No | NaN | NaN | No | 2cm | 0 | Yes | 0 | 0 | No | >20cm | >45 | Yes | 0 | 0 |
| | 5 | No | 0.5cm | 0 | No | NaN | NaN | Yes | 0 | 0 | Yes | 0 | 0 | No | 15cm | 5 | No | 0.5cm | 0 |
| | 6 | No | 3cm | 0 | No | NaN | NaN | No | 2.5cm | 0 | Yes | 0 | 0 | No | 11cm | 25 | No | 1cm | 0 |
| | 7 | No | 0.5cm | 0 | No | NaN | NaN | No | 2cm | 0 | No | 2.5cm | 0 | No | >20cm | >45 | Yes | 0 | 0 |
| | 8 | No | 1cm | 0 | No | NaN | NaN | No | 1.5cm | 0 | Yes | 0 | 0 | No | >20cm | >45 | No | 1cm | 0 |
| | 9 | No | 2.5cm | 0 | No | NaN | NaN | No | 6cm | 0 | No | 4cm | 0 | No | 19cm | >45 | No | 3.5cm | 0 |
| | 10 | Yes | 0 | 0 | No | NaN | NaN | No | 2cm | 0 | Yes | 0 | 0 | No | 13cm | 15 | No | 2.5cm | 5 |
| | | | | | | | | | | | | | | | | | | | |
| Avocado | 1 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 | Yes | 0 | 0 | No | >20cm | 30 | Yes | 0 | 0 |
| | 2 | Yes | 0 | 0 | No | NaN | NaN | No | 4cm | 0 | Yes | 0 | 0 | No | >20cm | 45 | Yes | 0 | 0 |
| | 3 | No | 2cm | 0 | No | NaN | NaN | No | 1cm | 0 | Yes | 0 | 0 | No | 20cm | 40 | Yes | 0 | 0 |
| | 4 | Yes | 0 | 0 | No | NaN | NaN | No | 2.5cm | 0 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 5 | No | 2cm | 0 | No | NaN | NaN | Yes | 0 | 0 | No | 2cm | 10 | No | NaN | NaN | No | 1.5cm | 0 |
| | 6 | Yes | 0 | 0 | No | NaN | NaN | No | 5cm | 0 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 7 | No | 1.5cm | 0 | No | NaN | NaN | No | 3.5cm | 0 | No | 0.5cm | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 8 | No | 2.5cm | 5 | No | NaN | NaN | No | 1.5cm | 0 | No | 1cm | 5 | No | NaN | NaN | No | 1.5cm | 0 |
| | 9 | No | 1cm | 0 | No | NaN | NaN | No | 3.5cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 5 |
| | 10 | No | 1.5cm | 0 | No | NaN | NaN | No | 1.5cm | 0 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | | | | | | | | | | | | | | | | | | | |
| Pickle | 1 | No | 1.5cm | 0 | No | NaN | NaN | No | 2cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 |
| | 2 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 |
| | 3 | Yes | 0 | 0 | No | NaN | NaN | No | 15cm | 0 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 4 | No | 3cm | 0 | No | NaN | NaN | No | 2.5cm | 15 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 5 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 5 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 6 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 |
| | 7 | Yes | 0 | 0 | No | NaN | NaN | No | 17cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 0.5cm | 0 |
| | 8 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 0.5cm | 0 |
| | 9 | No | 3.5cm | 25 | No | NaN | NaN | No | 3cm | 30 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 0 |
| | 10 | No | 2.5cm | 0 | No | NaN | NaN | No | 1.5cm | | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | | | | | | | | | | | | | | | | | | | |
| Pickle | 1 | No | 1.5cm | 0 | No | NaN | NaN | No | 2cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 |
| | 2 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 |
| | 3 | Yes | 0 | 0 | No | NaN | NaN | No | 15cm | 0 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 4 | No | 3cm | 0 | No | NaN | NaN | No | 2.5cm | 15 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 5 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 5 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 6 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 |
| | 7 | Yes | 0 | 0 | No | NaN | NaN | No | 17cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 0.5cm | 0 |
| | 8 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 0.5cm | 0 |
| | 9 | No | 3.5cm | 25 | No | NaN | NaN | No | 3cm | 30 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 0 |
| | 10 | No | 2.5cm | 0 | No | NaN | NaN | No | 1.5cm | | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | | | | | | | | | | | | | | | | | | | |
| Paprika | 1 | No | 5cm | 0 | No | NaN | NaN | No | 6cm | 5 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 0 |
| | 2 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 | No | 1cm | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 3 | No | 2cm | 0 | No | NaN | NaN | No | 1cm | 0 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 4 | No | 1cm | 5 | No | NaN | NaN | No | 4.5cm | 15 | No | 1.5cm | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 5 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 6 | No | 2cm | 10 | No | NaN | NaN | No | 0.5cm | 25 | Yes | 0 | 0 | No | NaN | NaN | No | 2cm | 0 |
| | 7 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 8 | No | 4cm | 10 | No | NaN | NaN | No | 3cm | 10 | No | 1cm | 0 | No | NaN | NaN | No | 0.5cm | 0 |
| | 9 | Yes | 0 | 0 | No | NaN | NaN | No | 1.5cm | 0 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | 10 | No | 13cm | 0 | No | NaN | NaN | No | 18.5cm | 5 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 |
| | | | | | | | | | | | | | | | | | | | |
| Strawberry | 1 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 0.5cm | 0 |
| | 2 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 | No | 1cm | 0 | No | NaN | NaN | No | 0.5cm | 0 |
| | 3 | No | 7.5cm | 5 | No | NaN | NaN | No | 23.5cm | 5 | No | 1.5cm | 0 | No | NaN | NaN | No | 0.5cm | 0 |
| | 4 | No | 0.5cm | 0 | No | NaN | NaN | No | 0.5cm | 0 | No | 1cm | 10 | No | NaN | NaN | No | 1.5cm | 0 |
| | 5 | Yes | 0 | 0 | No | NaN | NaN | Yes | 0 | 0 | No | 1cm | 5 | No | NaN | NaN | No | 0.5cm | 0 |
| | 6 | Yes | 0 | 0 | No | NaN | NaN | No | 0.5cm | 0 | No | 1cm | 0 | No | NaN | NaN | No | 1.5cm | 0 |
| | 7 | Yes | 0 | 0 | No | NaN | NaN | No | 0.5cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 |
| | 8 | No | 0.5cm | 0 | No | NaN | NaN | No | 1cm | 0 | No | 1cm | 0 | No | NaN | NaN | No | 1.5cm | 0 |
| | 9 | Yes | 0 | 0 | No | NaN | NaN | No | 1cm | 0 | Yes | 0 | 0 | No | NaN | NaN | No | 0.5cm | 0 |
| | 10 | Yes | 0 | 0 | No | NaN | NaN | No | 0.5cm | 0 | No | 1cm | 5 | No | NaN | NaN | No | 1.5cm | 0 |



| | | AlexDR | | | AlexNoDR | | | AlexCurrLearning | | | CustomDR | | | CustomNoDR | | | CustomCurrLearning | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | # | Success | Cm | Degrees | Suc. | Cm | Deg. | Success | Cm | Degrees | Success | Cm | Degrees | Suc. | Cm | Deg. | Success | Cm | Degrees |
| MilkCarton | 1 | No | 0.5cm | 0 | | | | No | 0.5cm | 0 | Yes | 0 | 0 | | | | Yes | 0 | 0 |
| | 2 | No | 0.5cm | 0 | | | | No | 1cm | 0 | Yes | 0 | 0 | | | | No | 1cm | 0 |
| | 3 | No | 0.5cm | 0 | | | | No | 1.5cm | 0 | Yes | 0 | 0 | | | | No | 0 | 90 |
| | 4 | No | 0.5cm | 0 | | | | Yes | 0 | 0 | No | 2cm | 90 | | | | No | 0 | 90 |
| | 5 | No | 0.5cm | 0 | | | | No | 1cm | 0 | No | 2cm | 0 | | | | No | 1cm | 0 |
| Matchbox | 1 | Yes | 0 | 0 | | | | Yes | 0 | 0 | No | 0.5cm | 0 | | | | Yes | 0 | 0 |
| | 2 | Yes | 0 | 0 | | | | Yes | 0 | 0 | No | 1.5cm | 0 | | | | Yes | 0 | 0 |
| | 3 | No | 0.5cm | 0 | | | | Yes | 0 | 0 | Yes | 0 | 0 | | | | Yes | 0 | 0 |
| | 4 | Yes | 0 | 0 | | | | Yes | 0 | 0 | Yes | 0 | 0 | | | | Yes | 0 | 0 |
| | 5 | No | 4cm | 0 | | | | No | 2.5cm | 0 | No | 1cm | 0 | | | | No | 4cm | 5 |
| Apple | 1 | No | 1cm | 0 | | | | No | 1cm | 0 | No | 3cm | 0 | | | | No | 2cm | 0 |
| | 2 | No | 1cm | 5 | | | | Yes | 0 | 0 | Yes | 0 | 0 | | | | Yes | 0 | 0 |
| | 3 | No | 3cm | 0 | | | | No | 27cm | 25 | No | 1cm | 0 | | | | Yes | 0 | 0 |
| | 4 | No | 1.5cm | 5 | | | | No | 3cm | 10 | No | 18cm | 5 | | | | Yes | 0 | 0 |
| | 5 | No | 1.5cm | 5 | | | | No | 22cm | 15 | Yes | 0 | 0 | | | | No | 5cm | 0 |
| Pear | 1 | No | 2cm | 0 | | | | Yes | 0 | 0 | No | 3.5cm | 0 | | | | Yes | 0 | 0 |
| | 2 | No | 5.5cm | 0 | | | | Yes | 0 | 0 | No | 2cm | 0 | | | | Yes | 0 | 0 |
| | 3 | No | 5.5cm | 0 | | | | No | 5cm | 0 | No | 3.5cm | 0 | | | | No | 1cm | 0 |
| | 4 | Yes | 0 | 0 | | | | Yes | 0 | 0 | No | 1.5cm | 25 | | | | No | 2cm | 90 |
| | 5 | No | 3cm | 0 | | | | No | 13.5cm | 0 | Yes | 0 | 0 | | | | Yes | 0 | 0 |