

ABSTRACT

Title of Thesis: UNBLOCK: INTERACTIVE
PERCEPTION FOR DECLUTTERING

Thesis Presented by: Krithika Govindaraj
Master of Science, 2021

Thesis Directed by: Dr. Yiannis Aloimonos
Department of Computer Science

Novel segmentation algorithms can easily identify objects that are occluded or partially occluded, however in highly cluttered scenes the degree of occlusion is so high that some objects may not be visible to a static camera. In these scenarios, humans use action to change the configuration of the environment, elicit more information through perception, process the information before taking the next action. Reinforcement learning models this behaviour, however unlike humans, the phase where perception data is understood is not included, as images are directly used as observations. The aim of this thesis is to establish a novel method that indirectly uses perception data for reinforcement learning to address the task of decluttering a scene using a static camera.

UNBLOCK: INTERACTIVE PERCEPTION FOR
DECLUTTERING

by

Krithika Govindaraj

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2021

Advisory Committee:

Dr. Yiannis Aloimonos, Chair/Advisor

Dr. Cornelia Fermuller

Dr. Jeffrey Herrmann

© Copyright by
Krithika Govindaraj
2021

Dedication

I would like to dedicate this work to my parents Govindaraj Kashappa and Manjula Chinnappa, and my sister, Deepika Govindaraj, who continue to shower their love and blessings, and to whom I owe everything in my life.

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever. First and foremost I'd like to thank my advisor, Dr Yiannis Aloimonos for providing me with this extremely interesting research topic and supporting me throughout the duration of my work. He has been extremely helpful and has continually provided me with useful suggestions, without which this thesis would not have been possible. It has been a genuine pleasure to work as a part of the research group advised by Dr. Yiannis Aloimonos and Dr. Cornelia Fermuller. I would also like to thank Dr. Jeffrey W. Herrmann for agreeing to serve as a member of my thesis committee. I would also like to thank my research group colleague Xiaomin Lin for his inputs from his previous work on developing the simulation framework and models for counting experiments on static scenes which provided me with a strong foundation for conducting this research. My family — my father, mother and sister — deserve special acknowledgement for their constant support and faith in me, and for making me the person I am today. I would also like to acknowledge the huge role that all my friends played during my work. It is due to their friendship and support that I have been able to maintain balance in my life.

Table of Contents

Dedication	ii
Acknowledgements	iii
Table of Contents	iv
List of Tables	vi
List of Figures	vii
Chapter 1: Introduction	1
1.1 Problem Statement:	1
1.2 Problem Setup:	2
1.3 Proposed Solution:	3
Chapter 2: Chapter Roadmap	5
Chapter 3: Related Work	7
3.1 Reinforcement Learning:	7
3.2 Perception:	8
Chapter 4: Preliminaries	9
4.1 Problem Statement:	9
4.2 Assumptions:	9
Chapter 5: Methodology	11
5.1 Environment Setup	11
5.2 Actions and Observations:	14
5.3 Reward:	14
Chapter 6: Algorithm Design	15
6.1 General Design	15
6.2 Neural Network Design	16
6.2.1 Actor-Critic as the main Network for Learning	16
6.2.2 Holistically-Nested Edge Detection(HED)	18
6.2.3 Noisy Net	21
6.2.4 Hindsight Experience Replay	23

6.3	Integrated Design	25
Chapter 7: Design of Experiments		29
7.1	General Structure	29
7.2	Single Object	30
7.2.1	Simulation Framework	30
7.2.2	Reward Function	30
7.2.3	Observation and Goal	31
7.2.4	Hindsight Experience Replay	32
7.2.5	Noisy Net	32
7.3	Multiple Objects	32
7.3.1	Simulation Framework	32
7.3.2	Holistically Nested Edge Detection	33
7.4	Realistic Scene	33
Chapter 8: Experiments		34
8.1	Single Objects	34
8.1.1	Reward Function Design	34
8.1.2	Observation and Goal Setup	43
8.1.3	Hindsight Experience Replay	45
8.1.4	Noisy Net	47
8.2	Multiple Objects	48
8.2.1	Holistically Nested Edge Detection	48
8.3	Realistic Scene	50
Chapter 9: Results		52
Chapter 10: Future Work		53
Bibliography		54
Bibliography		54

List of Tables

7.1	Overview of Reward	31
7.2	Overview of Observations and Goals	31
7.3	Overview of HED	33
8.1	Overview of HED	48

List of Figures

5.1	FetchReach Arm	12
5.2	FetchReach Arm Custom	13
6.1	Reinforcement learning methods	16
6.2	Actor Critic Network with HER	17
6.3	Holistically Nested Edge Detection	19
6.4	Hindsight Experience Replay	20
6.5	Noisy Net	24
6.6	Hindsight Experience Replay Buffer Size	26
8.1	Experiment1 for Reward Function Design	35
8.2	Experiment2 for Reward Function Design	36
8.3	Experiment3 for Reward Function Design	38
8.4	Experiment4 for Reward Function Design	39
8.5	Experiment5 for Reward Function Design	41
8.6	Experiment6 for Reward Function Design	42
8.7	HER comparison of modes future and episodic	46
8.8	HER modes	46
8.9	Experiment for Noisy Net without Noise	47
8.10	Experiment for Noisy Net with Noise	48
8.11	HED modes	49
8.12	YOLO Realistic Image	50
8.13	HED Realistic Image	51

Chapter 1: Introduction

1.1 Problem Statement:

Deriving information from a scene that is dynamic ensures that objects at some point will be in the view of a static camera and hence, get detected. In static scenes however, we cannot guarantee that occluded objects are in clear view until a change is induced in the environment.

Occlusion can be partial or complete and more often the amount of occlusion can be controlled based on camera height and angle. However, in extremely cluttered scenes positioning the camera cannot guarantee that all objects can be properly detected, counted or segmented. A moving camera can also be a possible alternative, however, if an object is completely occluded underneath another object, a moving camera can still fail to see the object. Humans change the scene to extract more information and improve the action that needs to be taken to accomplish a task. We derive information from what we see and we may pay attention to the characteristics of an object like the color, size, shape, texture, and number of objects in our environment based on the task to guide our actions. We emphasize the need for combining perception and learning to model human-like behaviour and address

the need for extracting appropriate information from perception based modules for learning actions that can optimally declutter a scene.

1.2 Problem Setup:

We model a static cluttered scene as an environment with a table and many objects of different sizes, shapes, colors and textures lying randomly in the scene. The scene is mainly composed of cluttered objects and a few decluttered objects produced on randomizing the scene. Imagine a human standing at the table and to model a static camera, we assume that the human is not allowed to move. The table is placed such that the arm of the human can reach its full extent by width and height. Now, if the human is tasked with decluttering the objects on a table, naturally the human will move towards the area with the most number of objects and pick that area apart to find objects hidden away due to occlusion. Now, imagine a robot having visual capabilities looking at a table top that contains many objects, on top of each other, in a cluttered scene. The robot's objective is to find as many objects by taking appropriate actions. Given one or several images, any segmentation algorithm will not produce the correct answer, simply because most objects are visible enough. The way to solve the problem is similar to the case discussed earlier for humans, where the objects that are stacked or very close together would have to be picked apart. The robot arm is placed overlooking the table to simulate a human who is not allowed to change position. It uses a single overhead static camera with a field of view covering the perimeter of the table as its perception module to model

the gaze of the human. The entire robot arm can be considered as the arm of the human, and the goal is to sway, push, and reach areas on the table knocking down objects that are stacked and separate them to clearly distinguish and find as many objects as possible.

1.3 Proposed Solution:

Reinforcement learning (RL) combined with neural networks has recently led to a wide range of successes in learning policies for sequential decision-making problems. This includes simulated environments, such as playing Atari games, and defeating the best human player at the game of Go, as well as robotic tasks such as helicopter control, hitting a baseball, screwing a cap onto a bottle, or door opening. The main issue with reinforcement learning however, has been reward engineering, but current algorithms use entire images with a convolutional neural network to convert states into actions. We pick a middle ground for reward engineering, by using information from images, and not the images themselves as a part of the reward function. In using this method, we reduce the computational complexity of using entire images but at the same time introduce perception based understanding into the system. In summary, there are two problems here, the first is the vision based which emphasizes on providing the learning module an appropriate representation of the environment. The other is a reinforcement learning problem and amounts to learning where to place the arm and in which direction to move to place objects in clear view of the camera. Hence, this work uses a combination of computer vision

and reinforcement learning to derive understanding of its environment and perform the task of decluttering.

Chapter 2: Chapter Roadmap

In this chapter, we provide a brief description of all the concepts that are discussed in this thesis.

In chapter 3, we provide a brief description of all the research in the area of reinforcement learning and computer vision, and compare our work to the previous research in these areas.

In chapter 4, we provide a description of the problem as well as the assumptions that are made to appropriately set up the environment.

In chapter 5, we provide an overview on setting up the simulation of the environment. The simulation consists of various parameters that need to be customized to make the simulation behave as close to the real-time scenario. The process of arriving at the final result is discussed in this chapter.

In chapter 6, we provide an overview on the design of the algorithm, which consists of a discussion on arriving at the final neural network design for the system.

In chapter 7, we plan out the set of experiments to determine the effect of different parameters on the system.

In chapter 8, we provide an analysis on the different parameters as set up in the previous chapter and conclude on the final parameters chosen of the system.

In chapter 9, we provide a concise overview on the conclusion of the system design for the problem.

Chapter 3: Related Work

3.1 Reinforcement Learning:

Coupling the perception module and the robot arm to develop hand eye coordination seems to be the most intuitive solution to the problem. Cheng *et al.* (2018) [1] suggest a method which consists of having a single object of interest, with objects that occlude the scene called distractors, while the arm pushes the object towards the target. In their paper, it is suggested that an active camera provides a better view of the scene through which the arm can make accurate decisions to push the object. However, the concept of target and the object of interest were initially known and the situation of multiple objects of interest was not handled. Johns *et al.* (2016) [2] use adapted versions of Convolutional Neural Network (CNN) architectures that are designed for object recognition for reinforcement learning based object manipulation tasks. Morrison *et al.* (2018) [3] suggest a novel CNN that uses the depth map of an object to produce an antipodal grasp pose and quality measure for every pixel in an image, from a camera mounted to the wrist of the robot. We determine the location of the objects and determine the objects to push based on contour data and derive estimated location of objects in the scene through our simulation. Zeng *et al.* (2018) [4] propose a method that could learn these syn-

ergies from scratch through model-free deep reinforcement learning. Their method involves training two fully convolutional networks that map from visual observations to actions: one infers the utility of pushes for a dense pixel-wise sampling of end effector orientations and locations, while the other does the same for grasping. They also propose the use of a heightmap i.e the depth map generated from an overhead RGB-D camera whose difference is used to provide a reward of 0.5 if it increases over a threshold.

3.2 Perception:

Attention based perception plays a crucial role in determining the focus of the perception module. Nishigaki *et al.* (2016) [5] proposed a mid-level operator to determine the closed contour, using a that torque operator takes as input the raw image and creates an image map by computing from the image gradients within regions of multiple sizes a measure of how well the edges are aligned to form closed, convex contours hence segmenting objects in the scene. Xie *et al.* (2015) [6] propose an end-to-end edge detection system, holistically-nested edge detection (HED), that automatically learns the type of rich hierarchical features that are crucial, to mimicking human ability to resolve ambiguity in natural image edge and object boundary detection. Joshi *et al.* (2020) [7] propose a network that merges information from an overhead RGB-D depth sensor and an RGB camera mounted on the wrist of the robot into a CNN for modelling vision into their network.

Chapter 4: Preliminaries

4.1 Problem Statement:

Reinforcement learning problems are designed around agents that perform actions in the environment with the intention of reaching a goal. The environment consists of a cluttered scene with rigid, non-hollow and non-deformable objects $O = o_1, o_2, \dots, o_N$ that may or may not be occluded. We define occlusion as a state where an object or a group of objects remain out of view of the overhead camera due to the height of the object in front of it. The goal of the arm is to move towards objects that are stacked to potentially uncover hidden objects behind them or separate objects that are stacked so that they can be detected as two or more separate objects. In this thesis, we aim to prove the following

- Need for action in static environments with occlusion.
- Emphasis on mid-level vision as a reward signal.

4.2 Assumptions:

We assume that the only equipment we have is a camera and distance sensor. In order to model realistic scenarios, we maintain that through the distance sen-

sensor, the only information we have is the position of the gripper and the location of 5% of objects within the scene. The rest of the objects whether actually occluded or unoccluded are considered to be undetected by arm. Hence, we know the relative position between the gripper and 5% of randomly placed objects through the distance sensor. The overhead camera is angled at 45 degrees so that even partially hidden objects are in view of the camera. This is to prove that novel segmentation methods fail and action in realistic scenarios and action is required for object-oriented tasks.

Chapter 5: Methodology

5.1 Environment Setup

With the introduction of open-source OpenAI’s Fetch manipulator environment, which offers a simulation of the 7-DoF Fetch robotics arm, has a two-fingered parallel gripper there has been extensive reinforcement learning research for manipulators. The Fetch environment can be used to perform four different tasks, sliding, reaching, pick and place and pushing. In our application, since the primary goal is to declutter the scene, we are using the Fetch-Push environment, which includes the same structure of observation and goals as mentioned above, but the primary goal being pushing the objects with its gripper to a target goal.

We customized our environment to house multiple objects without target locations as given in the default MuJoCo environment. We introduce objects that are occluded and unoccluded, in the form of objects that are stacked and unstacked. The camera is placed externally to the robot arm, and on the opposite end of the table to the arm, in order to provide a clear view with an angle of 45 degrees. We also categorize objects as being registered and unregistered, with the assumption that by calibrating a depth sensor mounted on the gripper, we can estimate the relative location from the gripper to a few clusters in the scene. Registered objects

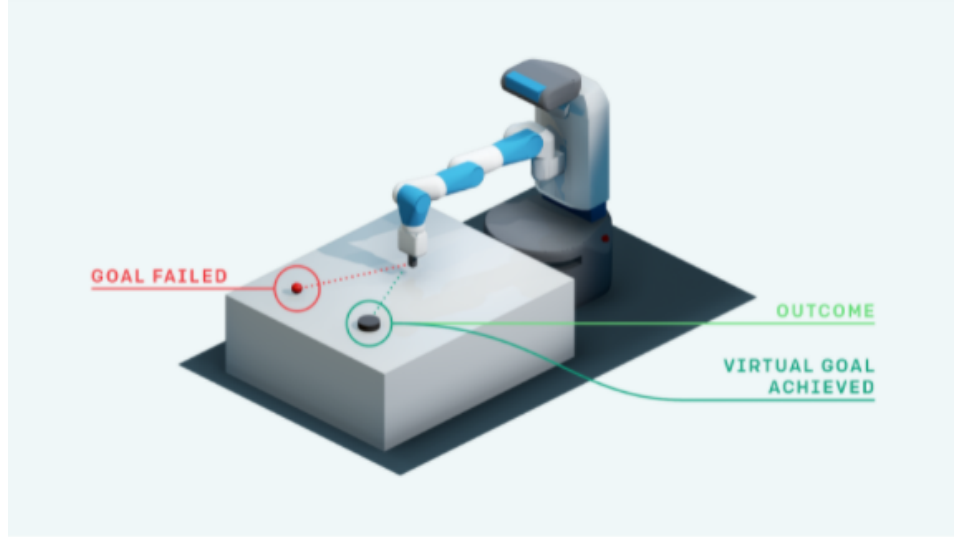


Figure 5.1: FetchReach Arm in original configuration

are those objects whose distance from the gripper can be estimated. Unregistered objects are those objects whose distances are unknown. These objects are chosen randomly and placement of these objects varies throughout the training. There is also no constraint on the number of unregistered objects, however, we have limited the number of registered objects to be two occluded objects and a single unoccluded object. Our training mainly depends on the number of contours as our reward and since this is very sparse i.e obtained after a few iterations of exploration in the scene, we need a dense reward to ensure proper training and eventually convergence. Hence, we introduce the concept of registered objects. The arm is also not bound to the constraints of the table, as it has to move away after each rollout to obtain a view of the table and the objects only, for contour detection.

In order to develop custom environments we need to include the files in the following structure:

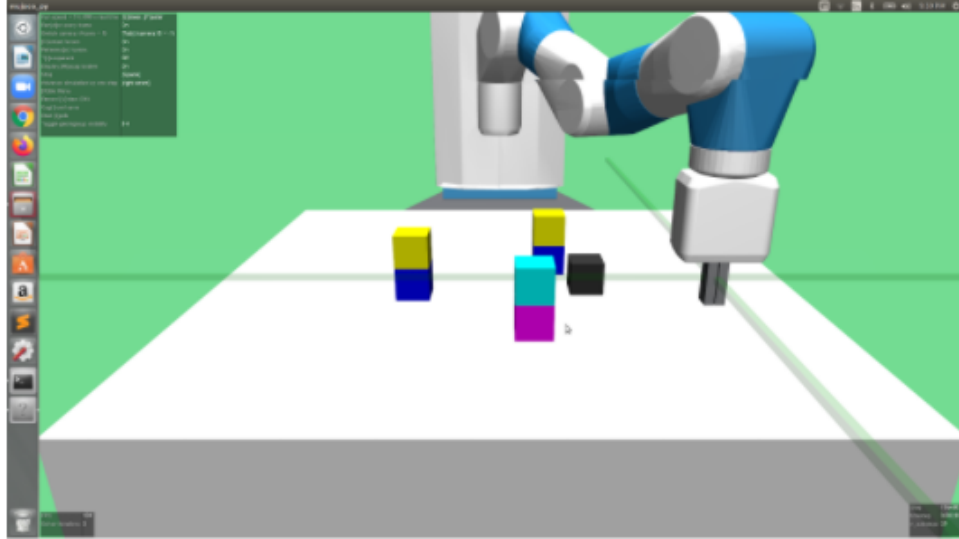


Figure 5.2: Customized FetchReach Arm

- robot.xml : Holds the model of the robot arm only along with the gripper and the textures and stls associated with it.
- shared.xml : Holds the textures and stls associated with common objects in the environment.
- push.xml : Holds the information for spawning models of the robot arm only along with the objects in the environment.
- fetch-env.py: Contains all the functions for reward design, viewer setup, goal design, observation design and environment reset.
- robot-env.py : Contains the functions to load the model of the robot into the environment and functions for OpenAI to interact with MuJoCo to enable the robot model to operate as intended.

- Setup.py: This file includes all the dependencies to set up the environment for launching the robot.

5.2 Actions and Observations:

Actions are 4-dimensional: 3 dimensions specify the desired gripper movement in Cartesian coordinates and the last dimension controls opening and closing of the gripper. Observations include the Cartesian position of the gripper, its linear velocity as well as the position and linear velocity of the robot's gripper. If an object is present, we also include the object's Cartesian position and rotation using Euler angles, its linear and angular velocities, as well as its position and linear velocities relative to gripper [8].

5.3 Reward:

In all Fetch tasks, the goal is 3-dimensional and describes the desired position of the object. Rewards are sparse and binary: The agent obtains a reward of 0 if the object is at the target location and -1 otherwise. The reward can either be set to sparse or binary. In the sparse reward mode, we obtain 1 if the block moves within 0.05m to the target, and 0 otherwise. The distance is calculated using the norm from the target to the object. In the binary reward mode, the distance itself is used as the reward. In our application we use the binary reward mode, with additional reward signals derived from edge detected images.

Chapter 6: Algorithm Design

6.1 General Design

The main goal is to use contours in order to determine the area of maximum occlusion. The reward function is designed to mimic human understanding of occlusion, as we consider the different manipulations to the number of contours as the signal as a measure for the degree of occlusion. Considering a scene where there are items placed randomly on the table, the most occluded part is the area on the table that has numerous objects lying in such a way that another object or object cannot be seen. As the robot moves around the environment, its goal is to remove stacked objects in the scene, so that any hidden object or any stacked object that may be perceived as a single object can be seen in clear view without changing the direction or orientation of the camera. This can be considered a scenario similar to when a human stands in front of the table, trying to figure out the number of objects. The human doesn't necessarily make major movements with their head or eyes, the displacement in the scene is produced by moving the arms around to take apart objects. In this work we use neural networks pertaining to noise based exploration, reinforcement learning based robot decision making and contour detection using image torque to design dynamic scene understanding.

6.2 Neural Network Design

6.2.1 Actor-Critic as the main Network for Learning

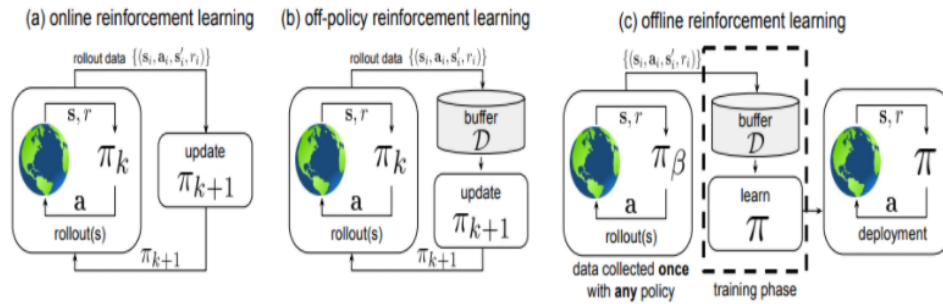


Figure 6.1: Reinforcement learning methods a) on policy b) off policy c) offline

In evaluating the types of networks to be used for this task, we could classify them into three basic categories which are on-policy, off-policy and offline reinforcement learning. We know that an agent within a network uses a policy to determine the next set of actions it needs to take within an environment.

The on-policy does not take into account previous experiences and simply uses the current policy to determine the next action. The off policy methods take into consideration the previous experiences to determine the next set of actions. Offline reinforcement learning essentially uses an agent that no more has access to the environment to derive experiences from to determine the optimal policy.

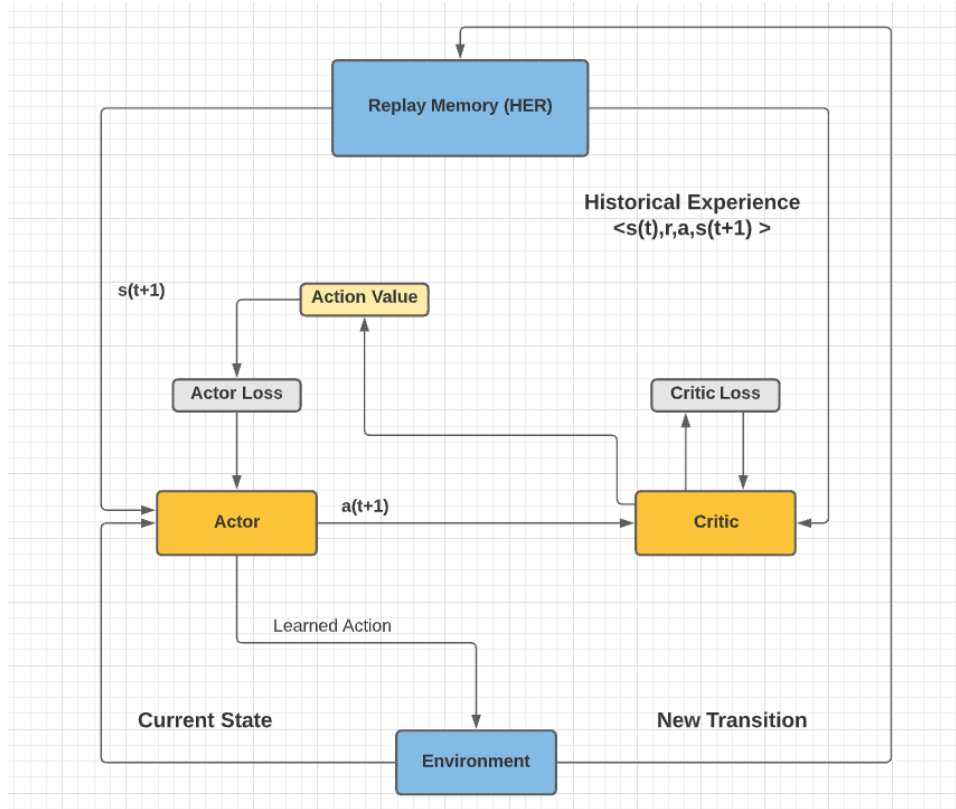


Figure 6.2: Actor Critic Network with buffer

Algorithm 1: DDPG Algorithm

Randomly Initialize critic network $Q(s, a | \theta_c)$ and actor $A(s|\theta_a)$ with weights θ_c and θ_a
Initialize target network Q' and A' with weights $\theta_c' \leftarrow \theta_c, \theta_a' \leftarrow \theta_a$
Initialize Replay Buffer R
for $episode \leftarrow 0$ **to** M **do**
 Initialize a random process N for action exploration
 Receive initial observation state s_1
 for $t \leftarrow 0$ **to** T **do**
 1. Select action $a(t)$ according to the current policy and exploration based on NoisyNet
 2. Execute action $a(t)$ and observe reward $r(t)$ and observe new state $s(t+1)$
 3. Store transition $(s(t), a(t), r(t), s(t+1))$ in R
 4. Sample a random minibatch of N transitions $(s(t), a(t), r(t), s(t+1))$ in R
 5. Set $y(i) = r(i) + Q'(s(i+1), A(s | \theta_a)|\theta_c')$
 6. Update critic by minimizing the loss $L = 1/N (y(i) - Q(s(i), a(i) | \theta_c))^2$
 7. Update the actor policy using the sampled policy gradient:
 $\nabla J \approx (1/N) \sum \nabla Q(s, a|c)|_{s=s(i), a=c(s(i))} \nabla A(s|\theta_a)|_{s(i)}$
 8. Update the target networks:
 $\theta_c' \leftarrow \tau\theta_c + (1 - \tau)\theta_c'$
 $\theta_a' \leftarrow \tau\theta_a + (1 - \tau)\theta_a'$
 end
end

6.2.2 Holistically-Nested Edge Detection(HED)

Initially we use the concept of image torque to determine the number of contours in the view of the camera. However, the algorithm was computationally intensive and with the increase in machine learning applications for detecting contours and object segmentation, we transitioned to the use of holistically nested edge detection. Every object can be considered as a collection of edges. If a scene has occluded objects then the amount of edges increases as we progress through the training process. Using HED the number of contours detected was accurate. We also found that image torque was highly susceptible to changes in scaling in the image of the scene. After increasing the scaling of the image the number of contours changed drastically. HED internally evaluates scaling, to determine the number of edges in the objects of the scene, so this method provides more consistent edge information

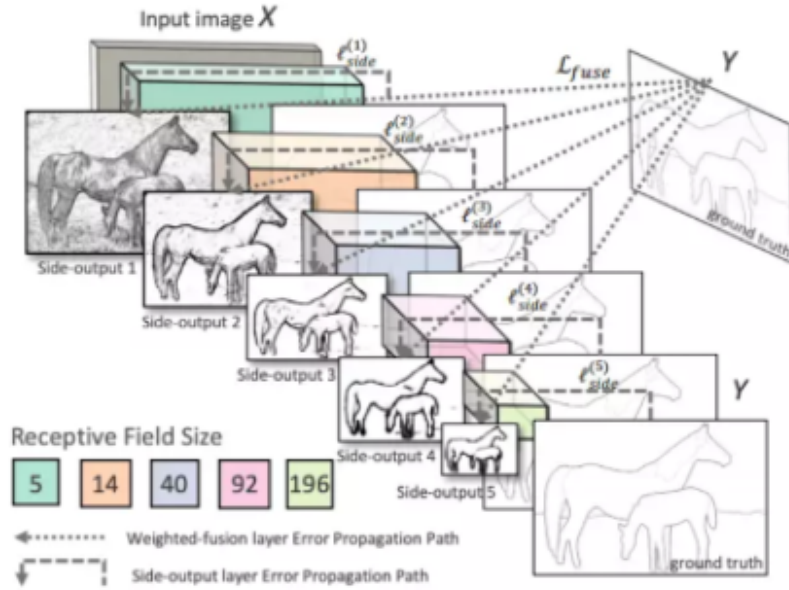


Figure 6.3: Holistically Nested Edge Detection

in the scene. We also do not require edge detection to an extent that it finds the details on the objects themselves.

We need to simply highlight the region that encapsulates an entire object. This is important in obtaining an accurate reading for the increase in contours. We need an algorithm that ensures that an object that is occluded is detected as a single object, so that the robot arm declutters the scene and these occluded objects are split up into two or more objects, giving an indication to that agent that its primary goal is to find and remove occlusion. If these occluded objects are already detected as individual objects, this hinders the learning. In order to accomplish the task of edge detection using machine learning many methods have been proposed, of which the notable ones that apply to this task are RefineContourNet, and Robust

Algorithm 2: Hindsight Experience Replay Algorithm

Given:

- An off-policy RL algorithm A e.g DQN, DDPG, NAF
- A strategy S for sampling goals for replay
- a reward function $r : S \times A \times G \rightarrow R$

Initialize A

Initialize replay buffer R

for $episode \leftarrow 0$ to M do

 Sample a goal g and initialize the state s_0 .

 for $episode \leftarrow 0$ to $T-1$ do

 Sample an action $a(t)$ using the behavioural policy from A: $a(t) \leftarrow \pi(s(t)||g)$

 Execute action $a(t)$ and observe reward $r(t)$ and observe new state $s(t+1)$

 end

 for $episode \leftarrow 0$ to $T-1$ do

$r(t) = (s(i), a(i), g)$ in R

 Store the transition $(s(t)||g, a(t), r(t), s(t+1)||g)$ in R

 Sample a set of additional goals for replay $G = S$ (current episode)

 for $g' \in G$ do

$r' = r(s(t), a(t), g')$

 Store the transition $(s(t)||g', a(t), r', s(t+1)||g')$ in R

 end

 end

 for $episode \leftarrow 0$ to $T-1$ do

 Sample a minibatch B from the replay buffer R

 Perform one step of optimization using A and minibatch B

 end

end

Figure 6.4: Hindsight Experience Replay buffer size comparison

CNN for edge detection. These methods directly derive from HED and are able to find edges more robustly, however using these methods could lead to detection that may hinder our learning. Hence, we use an algorithm that finds the overall segmentation of objects within a scene. In order to address the issue of multiple scales, traditional methods train using multiple independent networks with different depths and different output loss layers. However, implementing such an architecture would increase the amount of computational resources required for training. HED achieves scale-invariance using a single stream deep network with multiple side-outputs. The multiple side outputs also provide flexibility in adding additional fusion layers for unified output. The architecture of HED is based on VGGNet that has been seen to achieve state-of-art performance in the ImageNet challenge. Essentially this network has depth (16 convolutional layers), great density (stride-1 convolutional kernels) and multiple stages (five 2-stride down-sampling layers). Additional modifications such as connecting the side output to the last convolutional layer in each stage, respectively and cutting the last stage of the VGGNet were introduced. We are using the HED that is trained on the Berkeley Segmentation Dataset and Benchmark (BSDS500) images.

6.2.3 Noisy Net

Exploration allows an agent to improve its knowledge about which action to pick to lead to a long-term benefit. Improving the accuracy of the estimated action-values, enables the agent to make more informed decisions in the future. Ex-

exploitation, however, chooses the greedy action to get the most reward by exploiting the agent’s current action-value estimates. However, if we increase the exploitation without properly exploring the environment it could lead to a situation where the system has not properly understood its environment and takes actions that could lead to sub-optimal behaviour. There needs to be a balance between the amount that the arm explores and exploits. Our network should be able to understand when to utilize exploitation and when to switch to exploration of the reward signal is not improving. Epsilon-greedy action selection is a simple method that is usually used to balance exploration and exploitation by choosing between the two randomly. The epsilon refers to the probability of choosing to explore. If the epsilon is high then it is more likely for the system to explore than to exploit, and vice versa. Initially, the epsilon is maintained at a low value and then increased as the number of epochs increases to induce the switching from exploration to exploitation. Meire *et. al* (2019) [9] introduce a new method to replace conventional exploration heuristics, where they train a deep reinforcement learning agent with parametric noise added to its weights. This induced stochasticity aided efficient exploration and ensured that the agent learned the appropriate policy. Considering the epsilon-greedy method, we manually tune the hyperparameter and determine the amount that epsilon should increase in through the epochs. However, in using the NoisyNets we are automatically tuning the level of noise added to an agent for exploration. This network determines the amount of noise that is injected into the parameters of the network as needed. The NoisyNet samples a new set of parameters after every step of optimization, and between optimization steps the agent acts according to a fixed

set of parameters(weights and biases). This ensures that the agent always acts according to the parameters drawn from the current noise distribution. In this paper, two options are explored: Independent Gaussian noise, which uses an independent Gaussian noise entry per weight and Factorised Gaussian noise, which uses an independent noise per each output and another independent noise per each input. The main reason to use factorised Gaussian noise is to reduce the compute time of random number generation in our algorithms. In the paper factorised Gaussian Noise is used on RL agents such as DQN and Duelling networks, and independent Gaussian Noise was applied on A3C. We are using a DDPG based A2C network and A2C is a variant of the A3C, so we use independent Gaussian noise as described in this paper. The following is the representation of the noisy net layer. The parameters μ^w, μ^b, σ^w and σ^b are the learnables of the network whereas w and b are noise variables which can be chosen in factorised or non-factorised fashion. The noisy layer functions similarly to the standard fully connected linear layer. The main difference is that in the noisy layer both the weights vector and the bias is perturbed by some parametric zero-mean noise, that is, the noisy weights and the noisy bias can be expressed as $w = \mu^w + \sigma^w \cdot \epsilon^w$ and $b = \mu^b + \sigma^b \cdot \epsilon^b$, respectively. The output of the noisy layer is then simply obtained as $y = wx + b$.

6.2.4 Hindsight Experience Replay

Off policy networks rely on a replay buffer in order to maintain the record of all the experiences of the agent. These experiences are then recalled and used for

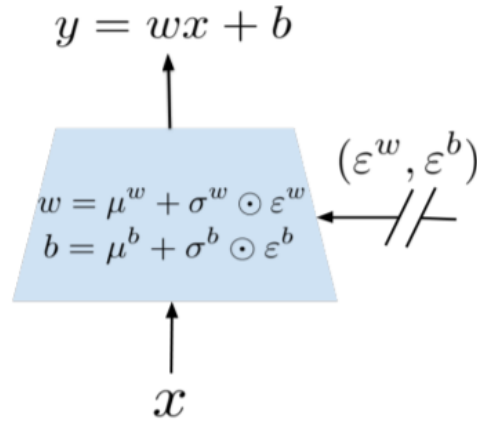


Figure 6.5: Noisy Net Architecture

determining the next action that the agent needs to take. Essentially the internal belief of an agent is stored in the form of this replay buffer, just like humans store memories. Replay buffers are used to break temporal correlations in the data by removing older experiences and holding onto a set of the most recent experiences. The replay buffers are flushed after it is completely full, and these experiences can be tuned to be obtained at certain intervals or throughout the training process. It was found that on training an RL agent to play the MountainCar game under the standard deep Q-Network and changing only the size of the memory buffer, the agent's performance depended non-monotonically on the memory size. Hence, both too little and too much memory substantially degraded the agent's learning. There are various types of buffers that we could use to record the information collected over various epochs. As mentioned, we are dealing with sparse rewards, due to which failures are more likely to occur than successes. We need a buffer that can help us learn from failures rather than from successful scenarios. Marcin Andrychowicz *et*

al. [10] suggest hindsight experience replay as the best alternative as it is sample-efficient in learning from rewards which are sparse and binary and therefore avoids the need for complicated reward engineering. In this paper, they demonstrate the same robotic arm used in our simulation and using a DDPG based actor-critic network. It was found that the results improved drastically, and the agent was able to learn at a much faster pace as it used its failures during training.

We use the same concept in this thesis, where we use a DDPG based actor-critic network, with a hindsight experience replay buffer to deal with sparse rewards even in this environment. There are three different modes that can be used to extract information for training:

- future — replay with k random states which come from the same episode as the transition being replayed and were observed after it
- episode — replay with k random states coming from the same episode as the transition being replayed
- random — replay with k random states encountered so far in the whole training procedure

6.3 Integrated Design

The network consists of three parts:

- Holistically Nested Edge Detection
- NoisyNet

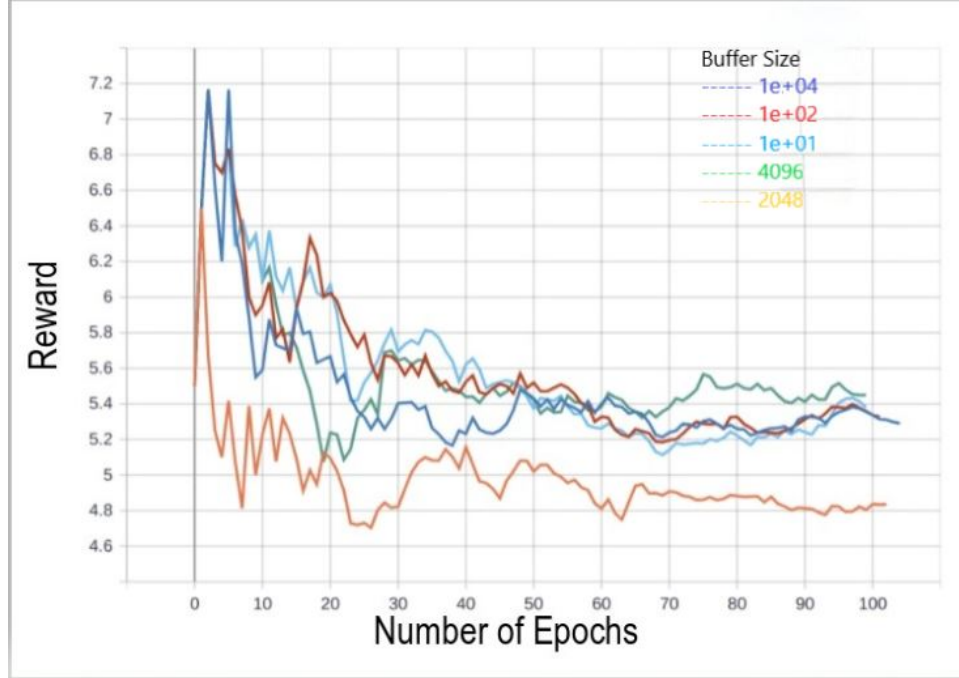


Figure 6.6: Hindsight Experience Replay buffer size comparison

- DDPG based Actor-Critic Network

Holistically Nested Edge Detection is used to obtain the number of edges within the network. This is the network responsible for deriving the input for the reinforcement learning. The image is placed into the HED network to obtain an edge detected image. The number of edges detected are counted and this is used as an input to the reward signal. It should however, be noted that this is a sparse component. We only collect this information once every epoch as this provides enough time for the arm to explore the environment and take movements that are significant enough to cause a distortion in the scene. We are using policy gradient methods, and in particular the actor-critic based network as it works best for this environment.

Algorithm 3: DDPG + HER using NoisyNet and HED for Decluttering

```
Randomly Initialize critic network  $Q(s, a | \theta_c)$  and actor  $A(s|\theta_a)$  with weights  $\theta_c$  and  $\theta_a$ 
Initialize target network  $Q'$  and  $A'$  with weights  $\theta_c' \leftarrow \theta_c, \theta_a' \leftarrow \theta_a$ 
Initialize Replay Buffer  $R$ 
for  $episode \leftarrow 0$  to  $M$  do
    Initialize a random process  $N$  for action exploration
    Receive initial observation state  $s_1$ 
    for  $t \leftarrow 0$  to  $T$  do
        Select action  $a(t)$  according to the current policy and exploration based on
        NoisyNet
        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 set of additional goals for replay  $G$ 
        for  $g' \in G$  do
             $\bar{r} = r(s(t), a(t), g')$ 
            Store the transition  $(s(t)||g', a(t), r', s(t+1)||g')$  in  $R$ 
        end
    end
    for  $episode \leftarrow 0$  to  $M$  do
        Sample a random minibatch of  $N$  transitions  $(s(t), a(t), r(t), s(t+1))$  in  $R$ 
        Set  $y(i) = r(i) + Q'(s(i+1), A(s|\theta_a)|\theta_c')$ 
        Update critic by minimizing the loss  $L = 1/N (y(i) - Q(s(i), a(i) | \theta_a))^2$ 
        Update the actor policy using the sampled policy gradient:
         $\nabla J \approx (1/N) \sum \nabla Q(s, a|c)|_{s=s(i), a=c(s(i))} \nabla A(s|\theta_a)|_{s(i)}$ 
        Update the target networks:
         $\theta_c' \leftarrow \tau \theta_c + (1 - \tau) \theta_c'$ 
         $\theta_a' \leftarrow \tau \theta_a + (1 - \tau) \theta_a'$ 
    end
end
```

The actor-critic in general has:

- The critic estimates the value function. This could be the action-value or state-value.
- The actor updates the policy distribution in the direction suggested by the critic.

Essentially the actor is responsible for taking steps to explore or exploit the environment based on the policy that is learned, and the critic essentially guides the actor to make better decisions in order to generate the optimal policy.

The overall action essentially is determined from the reward function which now takes into consideration the number of edges that are found in the environment.

However, we need to tune the amount of exploration so that the actor does not settle at one particular action. We use Noisy Net to determine the amount of exploration based on the variance that is predetermined based on our experiments. When the actor has to pick between exploration or exploitation it still uses a threshold determined by a random number, however due to the noise that is directly input into the network, the arm tends to explore its environment better, without saturating to a particular set of actions.

Chapter 7: Design of Experiments

7.1 General Structure

Each experiment is designed to depict the changes in reward with the aim of maximizing it over a period of training. The arm is trained for until the reward begins decreasing over ten iterations, on an average it is hundred epochs.

In order to define an epoch, we need to define an episode. An episode consists of moving the arm in the environment over fifty iterations, the movements are defined by the exploration algorithm and the policy that is learned by the system over time.

An epoch consists of a combination episodic tasks as well as recall from the replay buffer. In every epoch, for every two episodes we have one active recall that is used during the training.

After the policy is learned the system is tested for twenty epochs and the training is validated by the increase in the number of contours in every epoch.

In the experiments below, our aim is to gauge the best methods for training the system, so we have the system run for hundred epochs and find the best combination of parameters or tune the parameters to increase the cumulative reward over time.

7.2 Single Object

7.2.1 Simulation Framework

The experiment is carried out using MuJoCo and OpenAI in a linux environment. The framework consists of occluded objects, unoccluded objects and a robot arm. In this experiment, we determine the performance of the algorithm using a single occluded object in the scene. In total, we house three objects, one which is unoccluded(unstacked) and another that is occluded(stacked). All the objects within the scene are registered with the assumption that we have a depth sensor that can gauge the distance from itself to these objects.

7.2.2 Reward Function

In this section we outline the different reward functions that were chosen and evaluated to enable the system to learn appropriate actions.

Experiments	Reward Function
Experiment 1	(current contours - previous contours)
Experiment 2	(current contours - previous contours) + sum(distances to all registered objects <0.1)
Experiment 3	(current contours - starting contour) + 5 x sum (distances to all registered objects <0.1)
Experiment 4	(current contours - previous contours) + 10 x sum (distances to all registered objects <0.1)
Experiment 5	0.5 x (current contours - starting contour) + 0.2 x sum (distances to all registered objects <0.1) + 0.3 x (increase of contour from previous iteration)
Experiment 6	5 x (current contours - starting contour) + 10 x sum (distances to all registered objects <0.1) + 15 x (increase of contour from previous iteration)

Table 7.1: List of Reward Functions

7.2.3 Observation and Goal

In this section we outline the different observations and goal functions that were chosen and evaluated to enable the system to learn appropriate actions.

Sl no.	Observation	Desired Goal	Achieved Goal
1	gripper state and the objects detected by the depth sensor	gripper position	object position
2	gripper state, objects detected by the depth sensor, number of contours detected	gripper position, number of contours in previous iteration	object position, number of contours in current iteration
3	gripper state, objects detected by the depth sensor, number of contours detected	gripper position, maximum number of contours detected over all iterations	object position, number of contours in current iteration

Table 7.2: List of Observations and Goals

7.2.4 Hindsight Experience Replay

We use different modes of hindsight experience replay to determine the correct operation for our application. We compare the total reward obtained over number of epochs, each evaluating future mode to episodic and random.

7.2.5 Noisy Net

We evaluate the network with noise and without noise, to determine the amount of stability on the network. We compare the total reward obtained over a number of epochs and determine whether or not to use noise.

7.3 Multiple Objects

7.3.1 Simulation Framework

The experiment is carried out using MuJoCo and OpenAI in a linux environment. The framework consists of occluded objects, unoccluded objects and a robot arm. In this experiment, we determine the performance of the algorithm using multiple occluded objects in the scene. In total, we house multiple objects, one which is unoccluded(unstacked) and another that is occluded(stacked). All the objects within the scene are registered with the assumption that we have a depth sensor that can gauge the distance from itself to these objects.

7.3.2 Holistically Nested Edge Detection

This experiment consists of varying scales of HED, where we gauge the effects of changing the size of the images to find the number of edges detected, and in turn, gauge the effects produced on the reward signal for its training.

Experiments	HED Scaling
Experiment 1	360 x 420
Experiment 2	420 x 560
Experiment 3	560 x 720
Experiment 4	720 x 1080
Experiment 5	1080 x 2060

Table 7.3: HED based scaling for multiple objects

7.4 Realistic Scene

In this section, we compare the results obtained from realistic scenes using segmentation methods such as YOLO and show that objects that are occluded are not detected correctly. We then present our approach for reward signal design using mid-level vision. We also compare the quality of images based on the number of edges seen.

Chapter 8: Experiments

We visualize the experiment results using plots of the rewards for every change in the parameters. First, we consider single objects, with different noise parameters and HED scales. Then, we consider the multiple objects scenario, with different noise parameters and HED scales.

8.1 Single Objects

8.1.1 Reward Function Design

In this section, we provide an overview of all the reward functions designed and discussions for arriving at the final reward function. The following are the reward functions that were designed and modified to arrive at the final reward function:

We started with designing the simple reward function as a measure of increasing contours. If the number of contours increases with respect to the previous iteration, we set a reward of 1 else we set a reward of -1.

$$\mathbf{Reward = (current\ contours - previous\ contours)}$$

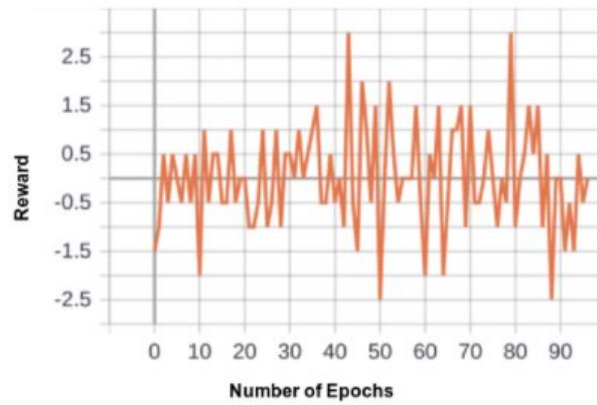
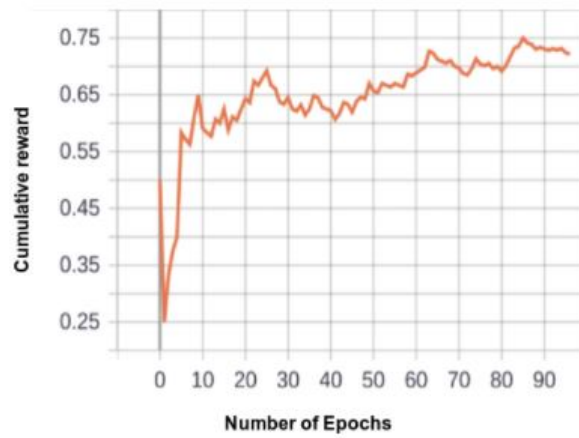


Figure 8.1: a) Cumulative Reward over epochs b) Reward per epoch

However this reward did not optimize the movements of the arm. In our algorithm we move the arm away from the table to take in the scene and determine the number of contours. There was no criteria to enforce the arm to stay near the table. There is also no incentive for staying within the bounds of the table.

We then introduced the concept of registered objects with the assumption that

using a depth sensor we can obtain the distance from the arm to the object. We cannot obtain the distance of all the objects within the scene, so we classified objects into two parts, registered and unregistered objects. Registered objects only form a very small portion of the scene and the majority of the objects are unregistered. With this in mind the function is now:

$$\text{Reward} = (\text{current contours} - \text{previous contours}) + \text{sum}(\text{distances to all registered objects} < 0.1)$$

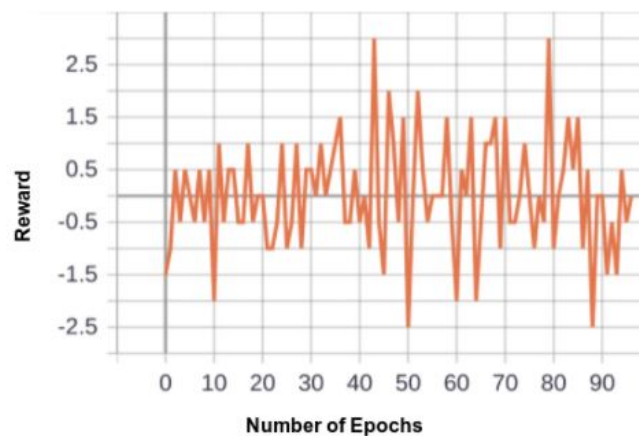
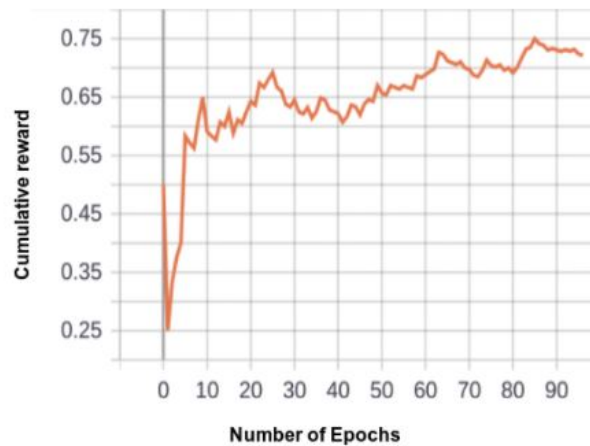


Figure 8.2: a) Cumulative Reward over epochs b) Reward per epoch

The movement of the arm was now close to the objects on the table but it was not learning to hit the stack objects or occluded objects in the scene. It moved in a predetermined trajectory and was not hitting objects that can be randomly spread out.

Reward scaling was used to prioritize increasing the number of current contours over the previous iteration instead of moving towards a registered object. The reward was scaled as follows:

$$\begin{aligned} \text{Reward} = & (\text{current contours} - \text{previous contours}) \\ & + 10 \times \text{sum} (\text{distances to all registered objects} < 0.1) \end{aligned}$$

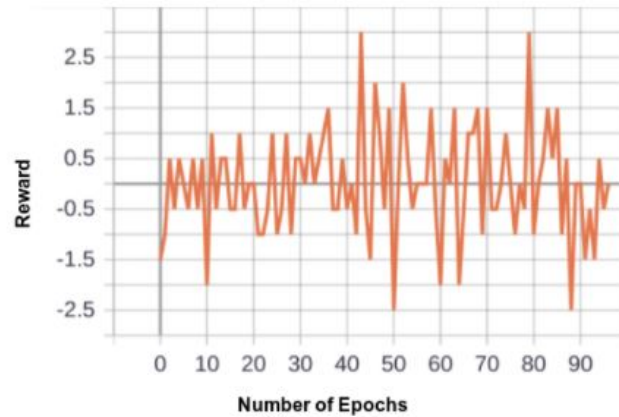
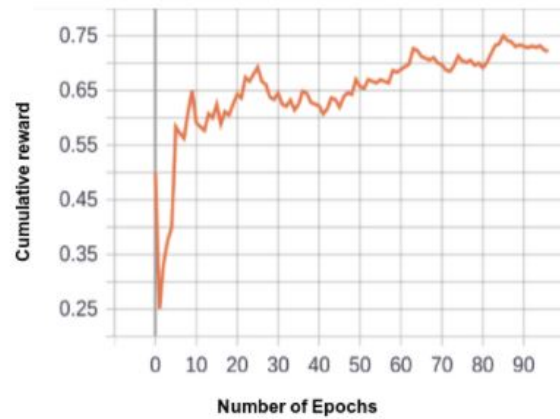


Figure 8.3: a) Cumulative Reward over epochs b) Reward per epoch

However, the arm still moved randomly and this can be attributed to the concept of fixing the goal. The previous contours and the current contours are constantly changing, due to which the arm learns and unlearns an existing policy over various epochs, very quickly. There is no stability in the reward function, and the only stable reward obtained is from the distances to registered objects, due to which the arm has an inclination to move towards the registered objects without

realizing that the actual goal is to remove occlusion.

In most reinforcement learning problems, the goal is to scale the amount of reward obtained. Extending the same concept, we start with the initial number of contours in the environment. The initial contours is set as the starting point and the agent is said to be learning if the total amount of rewards improves over time.

$$\text{Reward} = (\text{current contours} - \text{starting contour}) \\ + 5 \times \text{sum} (\text{distances to all registered objects} < 0.1)$$

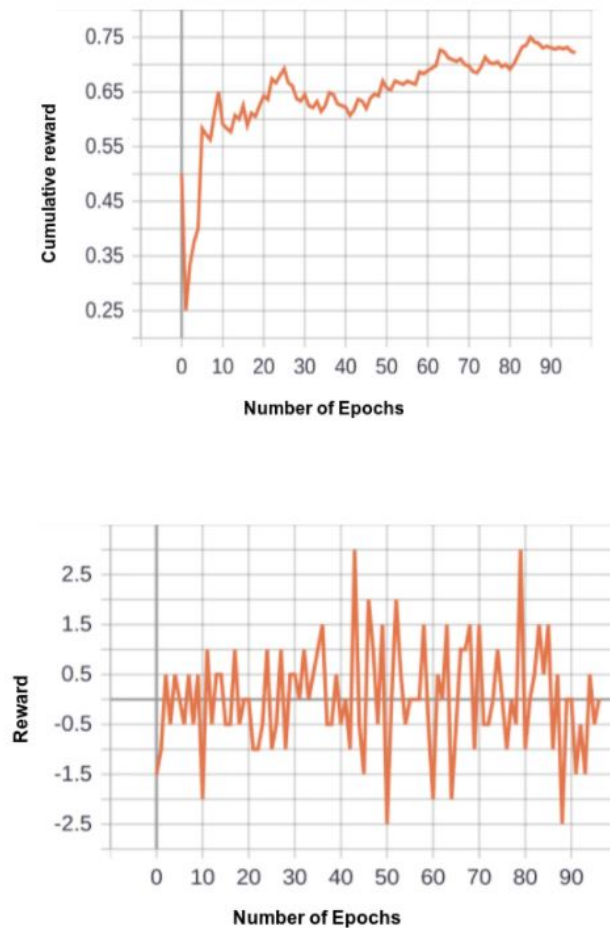


Figure 8.4: a) Cumulative Reward over epochs b) Reward per epoch

The issue with this function however, was that the increase from the starting point was accounted for, we are not accounting for the overall increase. At every epoch there are new contours that are explored, and we want to ensure that at every iteration there are new contours that are found. In order to account for this, we use the previous contours as we did at the very beginning, but assign a lower scaling to it.

The reward function is now updated to increase the number of contours for every iteration, but also increase the contours for its starting point. We introduce a new variable called increase, which acts as an indicator for increase. If the number of contours is more than the previous, it produces a value of 1, else it generates a value of -1. We also changed the scaling to ensure the priority of the task is maintained, such that the first priority is to increasing the number of contours from the initial scene, after which we need to keep increasing the number of contours after every iteration and finally enforce the arm to stay close to the table. We also scale down to control the arm's swing. We distribute the values in such a way that it sums up to 1. Now, the reward function obtained is as follows:

$$\begin{aligned} \text{Reward} &= 0.5 \times (\text{current contours} - \text{starting contour}) \\ &+ 0.2 \times \text{sum} (\text{distances to all registered objects} < 0.1) \\ &+ 0.3 \times (\text{increase of contour from previous iteration}) \end{aligned}$$

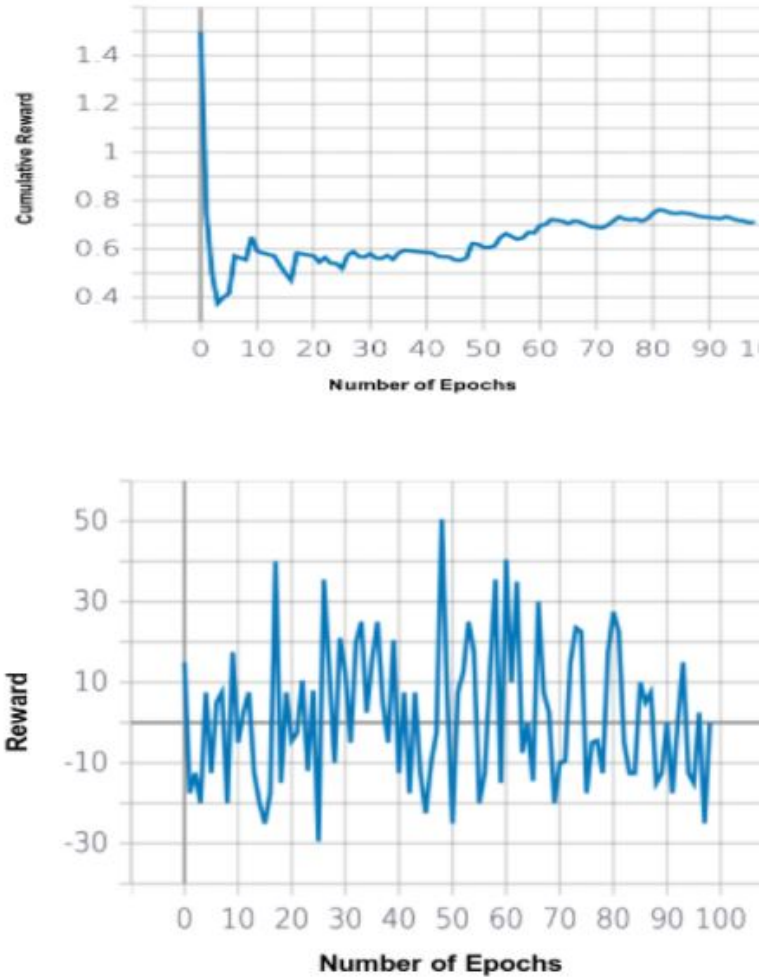


Figure 8.5: a) Cumulative Reward over epochs b) Reward per epoch

However, we found that the swing of the arm now is very less. The arm also tends to move slower and it can be attributed to the fact that the rewards obtained for every action are not scaled properly.

We used integers as rewards to amplify the movements of the arm, and assigned higher points for increasing the number of contours, next highest for increasing the contours with respect to the starting contour, and finally maintaining the distance

from the registered objects. The reward function now becomes:

$$\begin{aligned} \text{Reward} = & 5 \times (\text{current contours} - \text{starting contour}) \\ & + 10 \times \text{sum} (\text{distances to all registered objects} < 0.1) \\ & + 15 \times (\text{increase of contour from previous iteration}) \end{aligned}$$

Using this reward scheme allowed in maintaining the swing of the arm and moving the arm through larger movements while training it on the find more objects in the scene.

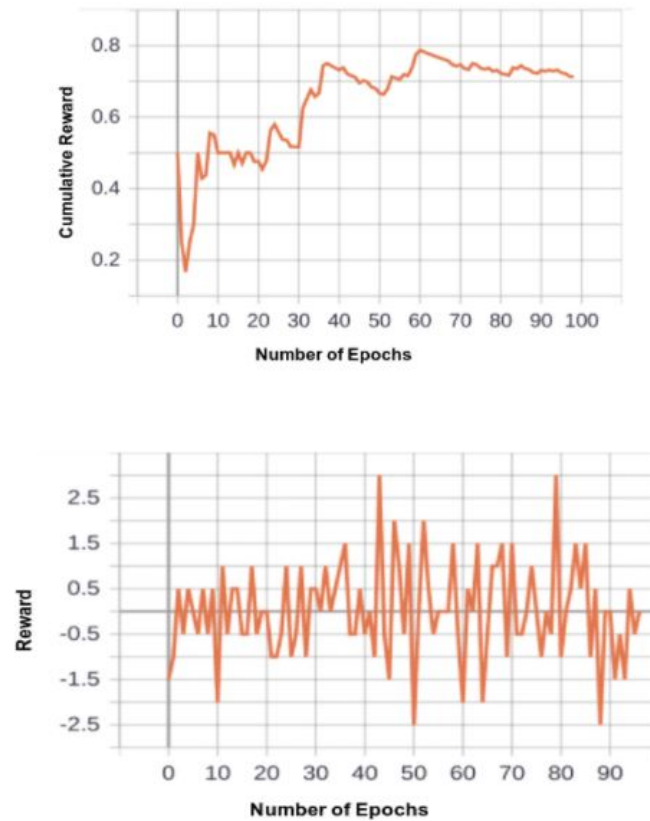


Figure 8.6: a) Cumulative Reward over epochs b) Reward per epoch

8.1.2 Observation and Goal Setup

For our application we wanted to modify the environment as much as possible to replicate real-time scenarios. The following changes were introduced to the observation and goals: The observation was initially provided only with the relative object positions of registered objects, and the gripper position and the gripper states. The observation or goals did not have any information about the number of contours.

observation = gripper state and the objects detected by the depth sensor

The desired goal used information of gripper positions only.

desired goal = gripper position

The achieved goal used the registered object positions only.

achieved goal = object position

Since the number of contours is not considered within the observation and goal states the robot arm focused on moving toward the registered object over the unregistered objects, with the goal of reducing the distance between the arm and the objects.

The observation contains the relative position from registered objects to the gripper and the gripper position itself. This information is obtained with the as-

sumption that we can find locations of registered objects using a depth sensor mounted on the gripper.

**observation = gripper state, objects detected by the depth sensor,
number of contours detected**

The desired goal contains the location of the gripper and the prev number of contours detected so far.

**desired goal = gripper position, number of contours in previous
iteration**

The achieved goal contains the location of the registered objects and the number of contours detected in that particular iteration.

achieved goal = object position, number of contours in current iteration

This structure was unstable as the target goal contours constantly changed, due to which the arm moved more randomly within the environment.

The observation contains the relative position from registered objects to the gripper and the gripper position itself. This information is obtained with the assumption that we can find locations of registered objects using a depth sensor mounted on the gripper.

**observation = gripper state, objects detected by the depth sensor,
number of contours detected**

The desired goal contains the location of the gripper and the max number of contours detected so far.

**desired goal = gripper position, maximum number of contours detected
over all iterations**

The achieved goal contains the location of the registered objects and the number of contours detected in that particular iteration.

achieved goal = object position, number of contours in current iteration

This was the final combination of observation, desired goal and achieved goal structure chosen. Using the maximum number of contours found over all epochs produced a stable target to reach. The current number of contours was also retained as a parameter in the observation.

8.1.3 Hindsight Experience Replay

All of these strategies have a hyper parameter k which controls the ratio of HER data to data coming from normal experience replay in the replay buffer. The following graph shows the comparison between using future vs episodic replay in HER. We find that HER with future has a better performance than HER with episodic.

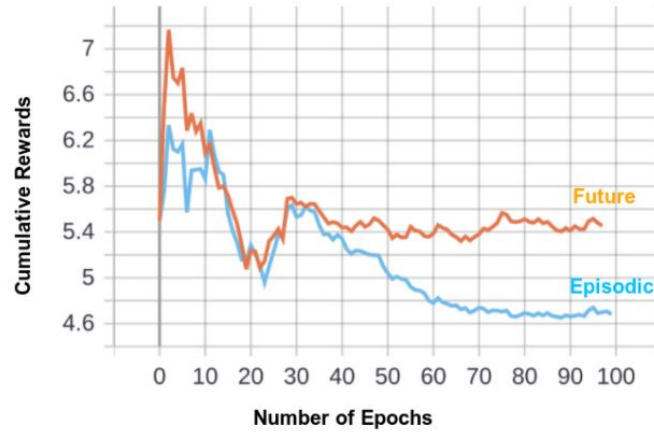


Figure 8.7: HER comparison of modes future and episodic

The following graph shows the comparison between using future vs random replay in HER. We find that HER with future has a better performance than HER with random.

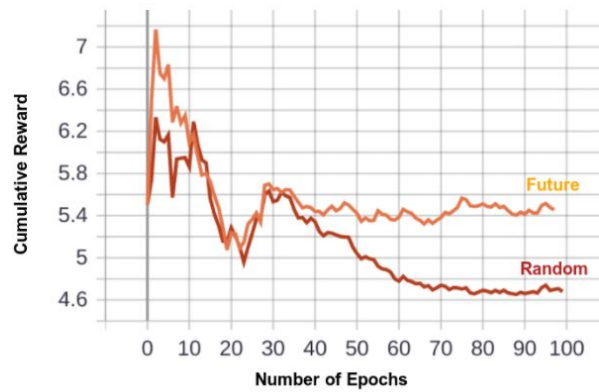


Figure 8.8: HER comparison of modes future and random

8.1.4 Noisy Net

Considering the network without noise, we can find that the running mean does not stabilize. This can be attributed to the fact that not many actions are explored and since HER converges faster as it learns from failure, without enough exploration it learns the wrong policy.

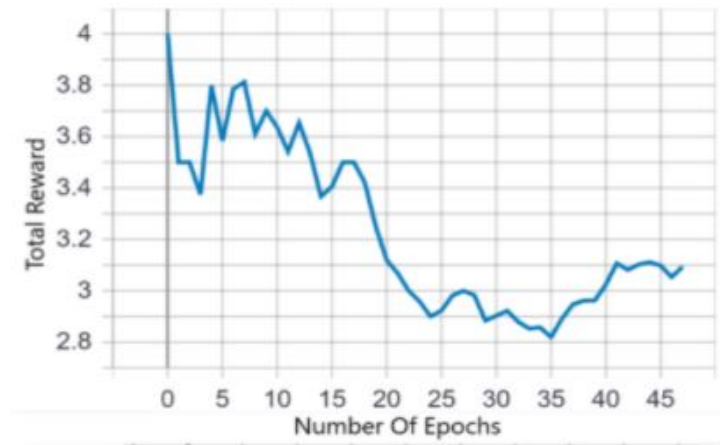


Figure 8.9: Experiment for Noisy Net without Noise

Considering the network with noise, we can find that the running mean stabilizes. This can be attributed to the fact that actions are explored and with enough exploration it learns the correct policy.

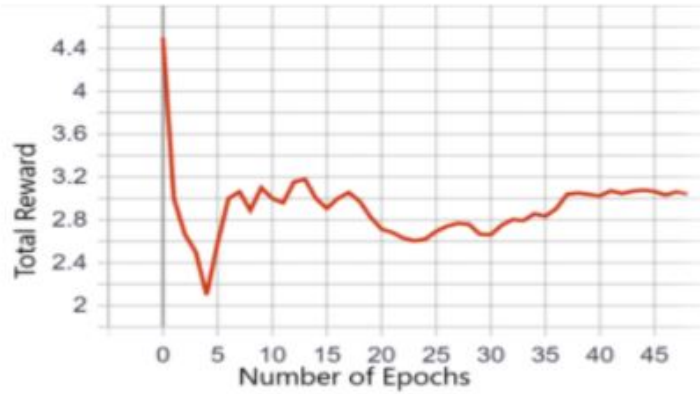


Figure 8.10: Experiment for Noisy Net with Noise

8.2 Multiple Objects

8.2.1 Holistically Nested Edge Detection

This experiment consists of varying scales of HED, where we gauge the effects of changing the size of the images to find the number of edges detected, and in turn, gauge the effects produced on the reward signal for its training.

Experiments	HED Scaling
Experiment 1	360 x 420
Experiment 2	420 x 560
Experiment 3	560 x 720
Experiment 4	720 x 1080
Experiment 5	1080 x 2060

Table 8.1: HED based scaling for multiple objects

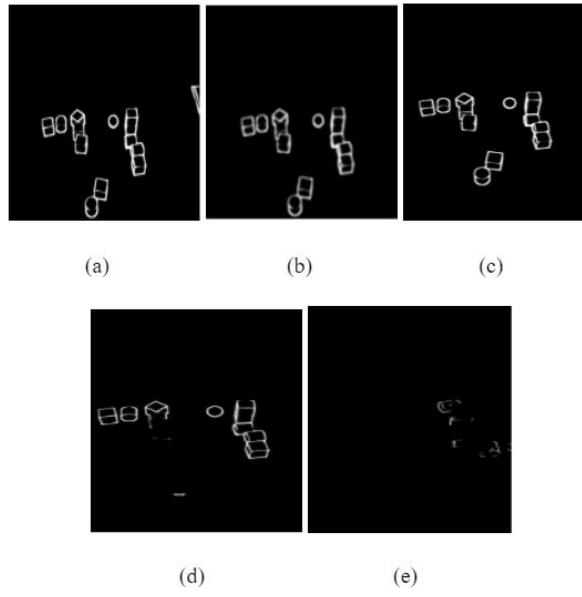


Figure 8.11: a) size 360 x 420 b) size 420 x 560 c) size 560 x 720 d) size 720 x 1080
e) size 1080 x 2060

8.3 Realistic Scene



Figure 8.12: Realistic Images using YOLO

Considering YOLO, we find that in highly cluttered scenes, the number of objects detected are very few. We also find that novel segmentation techniques cannot handle occlusion or partial occlusion.

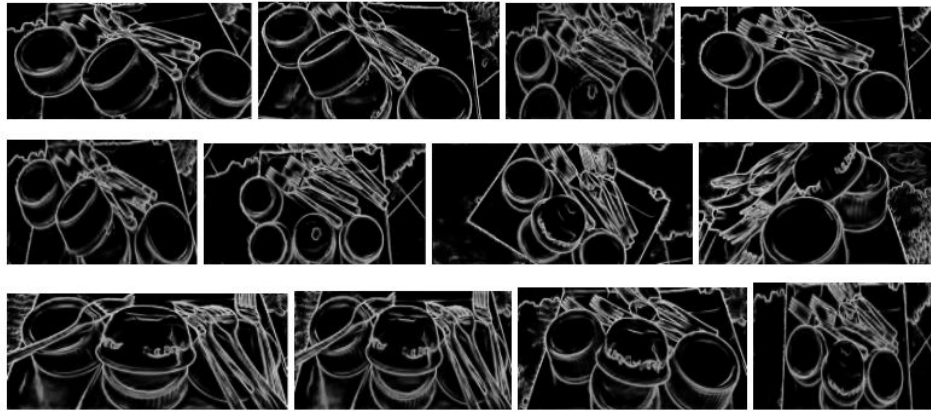


Figure 8.13: Realistic Images using HED

Considering an edge-detected image, such as holistically nested edge-detection, we find that edges of the occluded and unoccluded objects can be found. Hence, using this technique, we can obtain the count of the edges, and enable actions within the environment to place objects in clear-view.

Chapter 9: Results

Using real-time object detection on the scene, we have established that novel algorithms like YOLO cannot segment objects that are partially or fully occluded. We aim to declutter the scene before using segmentation for the same reasons, and the experiment is carried out in simulation. In environments with static objects in the scene, our aim was to establish the need for action in order to move the objects around. We found that instead of using images themselves, we obtained an optimal result from utilizing mid-level vision for reward signal design. We also found that changing the observation and goals was crucial for successful reinforcement learning. We shape the observation to include a number of edges along with information from the gripper, and with the assumption of using a depth sensor, we can include the information from the registered objects as well. The training involves scenes with single occlusion, and multiple occlusions to establish that algorithm works for all scenarios. In conclusion we find that the combination of Holistically Nested Edge Detection for mid-level vision, DDPG with hindsight as the main algorithm with NoisyNet to improve exploration for reinforcement learning produces best results.

Chapter 10: Future Work

Transfer learning is a research problem that aims to use knowledge gained in solving one problem to solve another problem. The hardest problem in robotics is to make a model transfer to the real world.

In our thesis, the main issue would be using holistically nested edge detection for real objects and we have shown that this method still works for real objects. The next steps would be including domain randomization into the simulation framework to mimic realistic scenarios such as changes in lighting, texture and color for hardware testing.

The Robot Operating System(ROS) comes with a combination of drivers which allows API's to connect with hardware. We can use the ROS software to model the same environment as that of MuJoCo and OpenAI. Using the knowledge of trained data from this simulation, we can transfer this to the hardware.

Bibliography

- [1] Ricson Cheng, Arpit Agarwal, Katerina Fragkiadaki, *Reinforcement Learning of Active Vision for Manipulating Objects under Occlusions* (2nd Conference on Robot Learning (CoRL 2018), Zurich, Switzerland).
- [2] Edward Johns, Stefan Leutenegger, and Andrew J. Davison *Deep Learning a Grasp Function for Grasping under Gripper Pose Uncertainty* (In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2016) pages 4461–4468.
- [3] Douglas Morrison, Peter Corke and Jurgen Leitner, *Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach*(arXiv:1804.05172v2 [cs.RO] 15 May 2018).
- [4] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, Thomas Funkhouser, *Learning Synergies between Pushing and Grasping with Self-supervised Deep Reinforcement Learning*,(arXiv:1803.09956v3 [cs.RO] 30 Sep 2018).
- [5] Morimichi Nishigaki, Cornelia Fermüller, *The Image Torque Operator for Contour Processing* (arXiv:1601.04669v1 [cs.CV] 18 Jan 2016).
- [6] Saining Xie, Zhuowen Tu, *The Holistically-Nested Edge Detection* (arXiv:1504.06375v2 [cs.CV] 4 Oct 2015).
- [7] Shirin Joshi, Sulabh Kumra, Ferat Sahin, *Robotic Grasping using Deep Reinforcement Learning*(arXiv:2007.04499v1 [cs.RO] 9 Jul 2020).
- [8] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba, *Multi-Goal Reinforce-*

ment Learning: Challenging Robotics Environments and Request for Research (arXiv:1802.09464v2 [cs.LG] 10 Mar 2018).

[9] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, Shane Legg, *Noisy Networks for Exploration* (arXiv:1706.10295v3 [cs.LG] 9 Jul 2019).

[10] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, Wojciech Zaremba, *Hindsight Experience Replay*(arXiv:1707.01495v1 [cs.LG] 5 Jul 2017).