Autonomous Robotic Grasping

A project report submitted

in partial fulfillment for the award of the degree of

Bachelor of Technology

in

ECE (Avionics)

by

Sri Aditya Deevi

Department of Avionics

Indian Institute of Space Science and Technology

Thiruvananthapuram, India

May 2022

Certificate

This is to certify that the project report titled ***Autonomous Robotic Grasping*** submitted by **Sri Aditya Deevi**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, in partial fulfillment for the award of the degree of **Bachelor of Technology** in **ECE (Avionics)** is a bona fide record of the original work carried out by him under my supervision. The contents of this project report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Deepak Mishra
Professor and Head of Department

Dr. Deepak Mishra
Professor and Head of Department

Place: Thiruvananthapuram
Date: May 2022

Declaration

I declare that this project report titled ***Autonomous Robotic Grasping*** submitted in partial fulfillment for the award of the degree of **Bachelor of Technology** in **ECE (Avionics)** is a record of the original work carried out by me under the supervision of **Dr. Deepak Mishra**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgments have been made wherever the findings of others have been cited.

Place: Thiruvananthapuram
Date: May 2022

Sri Aditya Deevi
(SC18B080)

This project report is dedicated to my Mom and Dad.

Acknowledgements

I would like to thank my adviser, guide and mentor, **Prof. Deepak Mishra**, for his constant support and encouragement throughout the project. He supported me and he was always more than willing to help me in clearing my doubts and participate in discussions. These insightful discussions and ideas provided by him were invaluable for successfully realizing various aspects of this project.

It has been a great learning experience and opportunity to grow personally and academically.

Furthermore, I would like to express my deep and sincere gratitude to the Department of Avionics and the Indian Institute of Space Science and Technology, Thiruvananthapuram for granting me this opportunity as well as the resources to carry out this project.

I also sincerely thank everyone else who helped me in realizing this work successfully.

Sri Aditya Deevi

Abstract

The primary premise behind vision-based Autonomous Robotic Grasping is the ability of a robot to sense and "perceive" its environment to interact with and impact diverse objects of interest by grasping them using vision-based sensors. Incorporating such a critical ability to grab an object in a robotic arm to do certain tasks might be advantageous in many disciplines. Industrial robots, for example, may aid human experts in completing diverse and repetitive processing activities such as pick-and-place, assembly, glue dispensing, material finishing, packaging, material removal, and quality inspection, while domestic robots can assist elderly or disabled individuals with their day-to-day grasping chores. This work is mainly focused on developing effective approaches for two challenging Autonomous Robotic Grasping tasks encountered universally, namely: *Task I : Grasping Various Object in Diverse Environments* and *Task II : Dynamic Grasping of Moving Objects*. In Task I, potent Deep Reinforcement Learning algorithms are developed to train a robotic arm to grasp novel objects in random unforeseen scenes, whereas as a part of Task II, Deep Learning and Inverse Kinematics methods are used in tandem, as a part of a dynamic grasping pipeline, to make the robotic arm grasp known objects, whose 3D model is available apriori, moving in an unspecified trajectory. Extensive experimentation and analysis is performed to evaluate the performance of the presented approaches. Furthermore, basic tasks for laying the foundation of developing a perception based intelligent “real” robotic grasping system are explored, using a Kinova robotic arm.

Contents

Abbreviations

ARG	Autonomous Robotic Grasping
DoF	Degrees of Freedom
RL	Reinforcement Learning
NN	Neural Network
CNN	Convolutional Neural Network
FCN	Fully Convolutional Neural Network
MDP	Markov Decision Process
SGD	Stochastic Gradient Descent
DL	Deep Learning
DRL	Deep Reinforcement Learning
FK	Forward Kinematics
IK	Inverse Kinematics
TD	Temporal Difference
DDPG	Deep Deterministic Policy Gradient
TD3	Twin Delayed Deep Deterministic Policy Gradient
SAC	Soft Actor Critic
TQC	Truncated Quantile Critics
ROS	Robot Operating System
KDL	Kinematics and Dynamics Library

OMPL	Open Motion Planning Library
FE	Feature Extractor
O-Conv	Octree based Convolution Operation
O-Maxpool	Octree based Maxpool Operation
O-CNN	Octree Based Convolutional Neural Network
SE	Squeeze and Excite
ECA	Efficient Channel Attention
GAP	Global Average Pooling
GAN	Generative Adversarial Networks
RRT	Rapidly-exploring Random Trees
PRM	Probabilistic RoadMaps
CHOMP	Covariant Hamiltonian Optimization for Motion Planning
STOMP	Stochastic Trajectory Optimization for Motion Planning
SDF	Signed Distance Field
MLP	Multi Layer Perceptron
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory network
HR	High Resolution
AC	Actor Critic
SN	Spectral Normalization

Chapter 1

Introduction

1.1 Problem Statement and Objectives

Autonomous Robotic Grasping (ARG) is key to attaining the promise of intelligent robotics. The primary premise underlying vision-based ARG is the capacity of a robot to "perceive" its surroundings using vision sensors to constructively interact with various objects to accomplish a given task of interest. However, the real world consists of many highly variable aspects. It is neither tractable nor feasible for a robot to accurately represent its surroundings, the things in it, and the complex interactions among them. Therefore, learning is crucial in such intelligent autonomous systems to acquire the ability to perform skilled manipulation tasks. Effective ARG systems have many applications in various domains. They can be deployed in industries, spacecraft, restaurants, and homes to perform or assist human experts in performing versatile and repetitive manipulation tasks. In this project, the following two challenging ARG tasks (Objectives) are considered which are almost ubiquitously found problems that an intelligent robotic arm can automate:

- *Grasping Various Objects in Diverse Environments* – This task aims to use Deep Reinforcement Learning techniques to make a robotic arm intelligently grasp novel objects, i.e. objects whose 3D model is not known apriori, in novel random scenes.
- *Dynamic Grasping of Moving Objects* – This task aims to use Deep Learning and Inverse Kinematics based Motion Planning techniques to help a robotic arm in *learning* how to grasp dynamic items of interest, whose 3D model is known apriori.

In addition, the basic steps and tasks necessary for performing complex ARG tasks in a "real" robotic setup are taken into account as a part of the problem statement of this work.

1.2 Basic Description of Robotic Arms Utilized

One of the most important components of any ARG setup is the robotic arm. In this section, a brief description of the robotic arms that were utilized either in the real setup or the simulated setup is provided.

1.2.1 Kinova Jaco2 Arm

`j2s7s300` arm is a 7-DoF flexible service robotic arm manufactured by Kinova. It has extreme portalbility, ultra low-weight and excellent payload-to-weight ratio. The gripper considered is a three fingered `KG-3` gripper.

Feature	Value
Max. Payload	2.1 Kg
Weight	5.5 Kg
Range	985 mm
No. of Joints	7

Table 1.1: Some features of Kinova `j2s7s300` arm

Fig. 1.1: Kinova Jaco2 Arm
(Used in Real Setup)

1.2.2 UR5 Arm

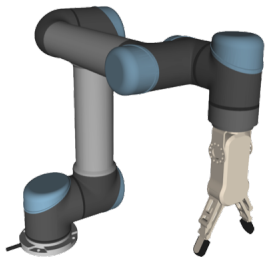


Fig. 1.2: UR5 Arm
(Used in Simulated Setup)

Feature	Value
Max. Payload	5 Kg
Weight	20.6 Kg
Range	850 mm
No. of Joints	6

Table 1.2: Some features of UR5 arm

The UR5 is a lightweight, adaptable, collaborative industrial robotic arm manufactured by Universal Robots.

It is designed to perform versatile and repetitive manual tasks consisting of payloads upto 5 Kg efficiently. The end effector considered is a *Robotiq* two-fingered adaptive gripper.

1.2.3 Franka Emika Panda Arm

Panda arm is a 7-DoF collaborative robotic arm developed by Franka Emika. It has high sensitivity, agility and ease of programming. The gripper attached to the arm is a two-fingered parallel FE gripper.

Feature	Value
Max. Payload	3 Kg
Weight	18 Kg
Range	855 mm
No. of Joints	7

Table 1.3: Some features of Panda arm

Fig. 1.3: Panda Arm
(Used in Simulated Setup)

1.3 Report Organization

This project report on "*Autonomous Robotic Grasping*" is organized as follows :

⋆ Chapter 2 provides a brief review of literature to setup the overall context for ARG approaches studied in this work, while explaining relevance and typical challenges associated with ARG.

⋆ Chapter 3 deals with developing effective DRL algorithms for *Task I - Grasping Various Objects in Diverse Objects* which involved training a robotic arm agent to grasp unseen objects in novel environments.

⋆ Chapter 4 provides detailed descriptions for the approach used for *Task II - Dynamic Grasping of Moving Objects*, where the aim is to make the robotic arm grasp known objects moving in an unknown trajectory.

⋆ Chapter 5 includes the results obtained by deploying the approaches used for both the tasks in various test scenarios.

⋆ Chapter 6 specifies the various tasks performed for setting up a "real" robotic grasping setup, while discussing the challenges associated.

⋆ Chapter 7 concludes the report, while providing ideas for future work to stimulate further research.

1.4 Key Contributions

Some of the major contributions made as part of this work and presented in the report are listed as follows :

✓ Demonstration of a complete end-to-end DRL pipeline, where a robotic arm was deployed in various random simulation scenes to learn a robust policy for grasping novel objects.

✓ Design and development of a series of octree-based convolutional neural network models for effective feature extraction in an Actor-Critic setting. The model development was focused to demonstrate the need for developing and incorporating complex DL models in DRL algorithms.

✓ Incorporation of developed models in the DRL pipeline demonstrating improved performance on the first task iteratively.

- ✓ Extensive comparison of various experiments conducted in the DRL setting on the first task. Some experiments involving the incorporation of novel DL ideas, such as advanced normalization techniques and improved architecture of Actor and Critic Networks, motivated the need to research DRL analogues for them.

- ✓ Demonstration of a Dynamic Grasping pipeline, where a robotic arm was deployed in a simulation scene to grasp dynamic objects whose 3D model is known apriori but the motion trajectory being followed by them is unknown.

- ✓ Incorporation of various object pose prediction algorithms into the dynamic grasping system and demonstrating improved performance on the second task.

- ✓ Extensive experimentation, comparison and inference based on obtained results, for all the developed methods as a part of the second task.

- ✓ Setting up a complete "real" robotic grasping setup mainly consisting of a Kinova `j2s7s300` arm and Intel Realsense `D415` camera.

- ✓ Successful demonstration of various tasks such as Hand-Eye Calibration & Blind Pick and Place, in the constrained robotic workspace of IIST's CVVR Laboratory.

Note

The implementation code and other files, used for realizing this work, can be found at :

https://drive.google.com/drive/folders/1nzImtpRu6TpJ83co94xY-qV8qdUJw-u2?usp=sharing

Chapter 2

A Brief Review of Literature

Autonomous Robotic Grasping aspires to give robots the capacity to "see" and interact with their surroundings via the efficient execution of skilled manipulation tasks. It is a long-standing scientific subject that has received significant attention throughout the years. It is used in a wide range of locations, including industries, residences, labs, and spaceships.

This chapter will discuss some fundamental ideas of ARG while illustrating a few of the works related to the problem statement at hand.

2.1 Typical Robotic Grasping System

A robotic grasping system [1] is shown in Fig. 2.1 where a robotic arm is fitted with an RGB-D camera and an end-effector to pick the object of interest kept on the flat workspace.
Different systems can employ different types of grippers and different types of input data, as described in Fig. 2.2 and Fig. 2.3 repectively, depending upon the application and availability.

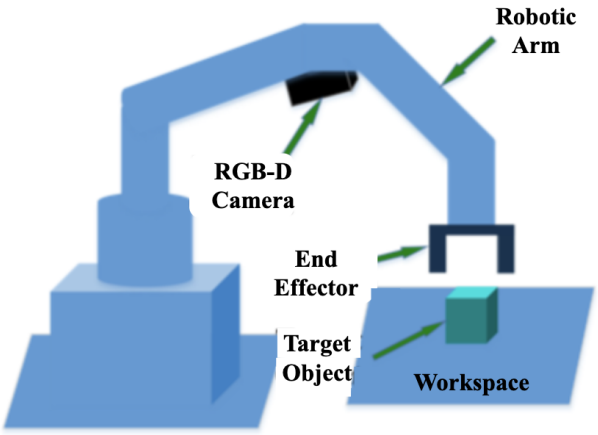


Fig. 2.1: An illustration of a robotic grasping system.

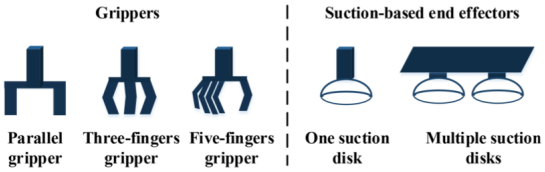


Fig. 2.2: Types of grippers for a robotic arm.

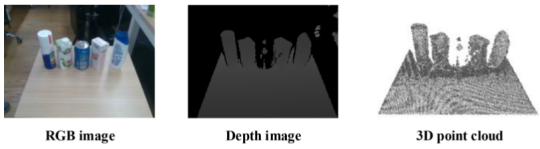


Fig. 2.3: Typical inputs available to a grasping system from a variety of visual sensors.

2.2 Categorizing Grasping Approaches

Grasping is a fundamental skill critical to perform “skilled" manipulation tasks such as :

- Pick & Place
- Assembly
- Manipulating deformable objects
- Handling common hand tools
- Dynamic Grasping of moving objects

Examining previous works in literature, two broad classes of grasping approaches can be identified as follows:

(1) *Grasping Known Objects* [2][3]

- 3D Model of Object of interest required
- Pose Estimation + Grasp (Trajectory) planning
- Generally, one model trained per object
- Objects can be dynamic

(2) *Grasping "Unknown" Objects* [4][5]

- 3D Model of Objects not required
- Deep RL based approaches can be used
- One agent trained can have a policy that can grasp diverse objects

2.3 Overview of Related ARG Approaches

In this section, reviews of some of the related research works from the ARG literature are specified.

2.3.1 Review of Paper-[6]

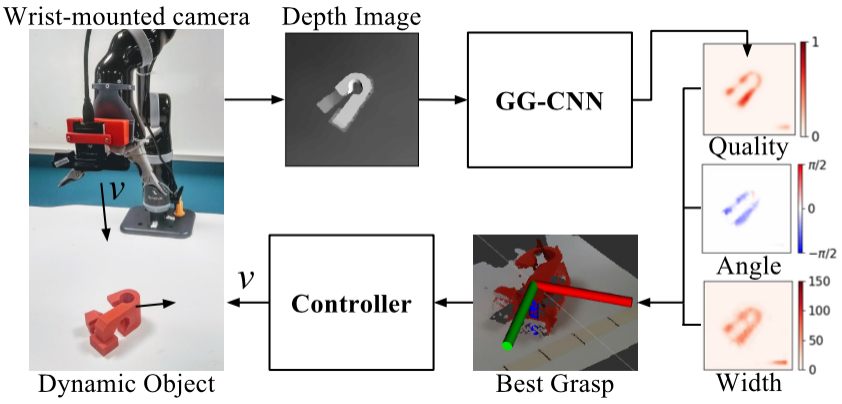


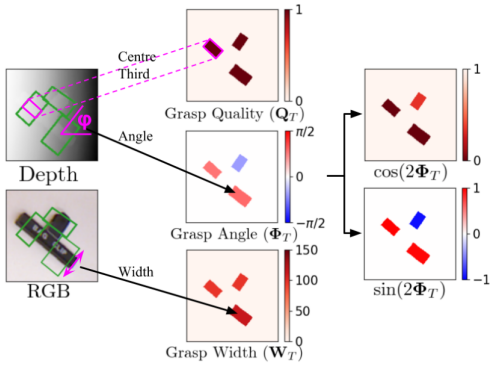
Fig. 2.4: Complete Pipeline of the GG-CNN approach. The CNN generates pixel-wise maps describing the quality, angle and width attributes of the grasp. Based on these best grasp is selected and given to the Controller to grasp the object.

An object-agnostic, real-time "generative" grasp construction technique is proposed by [6] that can prove to be helpful for closed-loop grasping. Some of the key contributions of the paper are :

(i) The method does not depend on sampling grasp candidates, but instead generates grasp poses pixel-by-pixel.

(ii) Light-weight CNN with fewer parameters to minimise latency

Fig. 2.5: Ground Truth Grasp Maps for Grasp Quality ($\mathbf{Q}_T$), Grasp Angle ($\mathbf{\Phi}_T$) and Grasp Width ($\mathbf{W}_T$).



Basically, a grasp is represented by $\mathbf{g} = (\mathbf{p}, \phi, w, q)$:

$\rightarrow$ Pose of the Gripper $(\mathbf{p}, \phi)$

$\rightarrow$ Width of the Gripper (w)

$\rightarrow$ Quality (chances of success) of grasp (q)

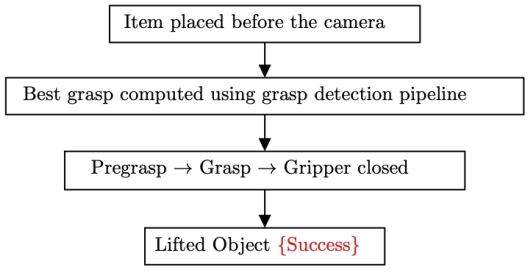
From the grasp maps $M(\mathbf{I}) = \mathbf{G} = (\Phi, W, Q) \in \mathbb{R}^{3\times H\times W}$, the best grasp in image space is obtained by :

$$\tilde{\mathbf{g}}^* = \max_{\mathbf{Q}} \mathbf{G}$$

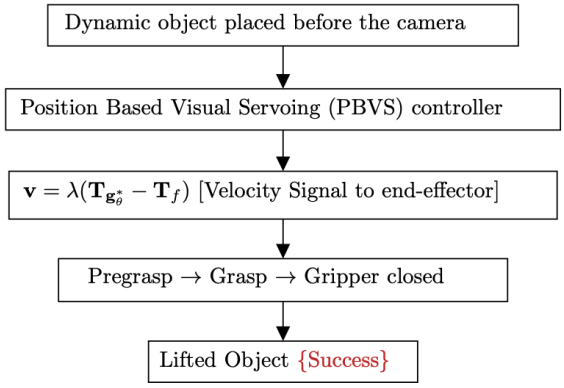
Fig. 2.5 describes the procedure for generating the ground truth maps (labels) for training the network.

Two types of grasp execution approaches were considered :

1. *Open Loop Grasping*



2. *Closed Loop Grasping*



2.3.2 Review of Paper-[5]

This work basically, considers the task of Robotic Kit Assembly, to learn to generalize to new unseen kits/objects. Some of the important aspects of this paper are :

(i) Establish the kit assembly job as a problem of matching shapes, with the objective of learning a shape discriminator that sets up geometric correspondences between item surfaces and the locations of their placement targets.

(ii) Data collection system that collects ground truth object-to-location correspondences without the need for supervision.

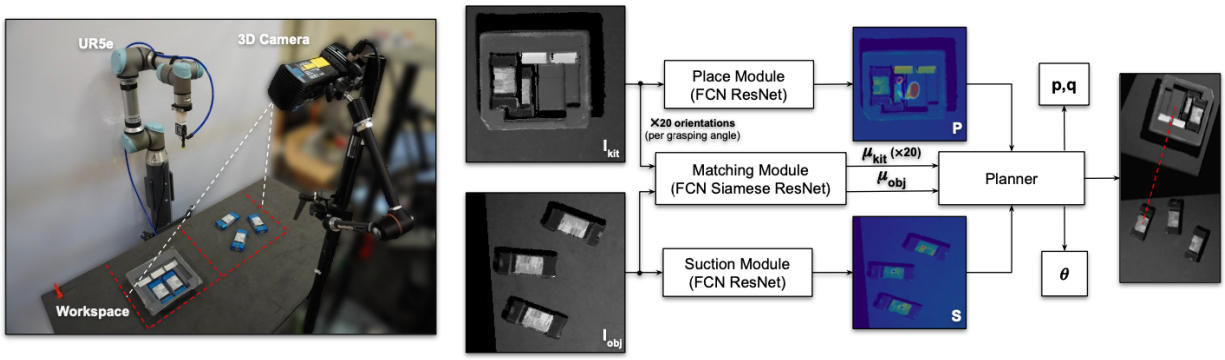


Fig. 2.6: Bird's eye view of different modules present in the Form2Fit pipeline.

Inputs of the approach are object heightmap I_{obj} and kit heightmap I_{kit}. Outputs are predictions of :

- Location of the pick, p
- Location of placement, q, and
- an angle $\theta \rightarrow$ measures difference in orientation between the placement and pick locations.

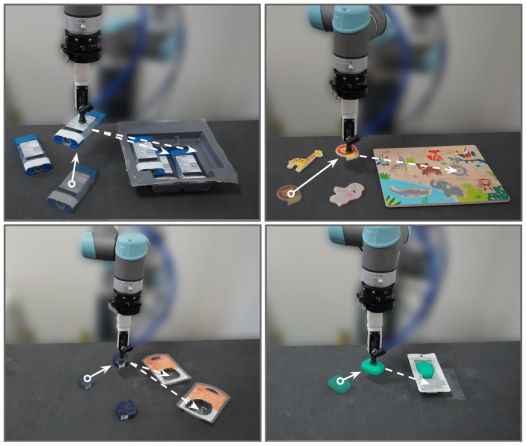


Fig. 2.7: Form2Fit in action. The system with suction effector generalises to new items and kits by leveraging data-driven priors of shapes learnt during training.

The system has three modules :

1. *Suction network* → Pixel-by-pixel forecasts of success probabilities (also known as affordances) of picking using a suction cup.
2. *Placing network* → Placement success percentages on the kit are predicted pixel by pixel.
3. *Matching network* → pixel-wise, rotation-sensitive feature descriptors to match between items and their corresponding fitting locations in the kit.

Self-monitoring from disassembly in reverse is used to teach the system :

- Utilizing random selection and trial & error to deconstruct a completely constructed kit.
- To get supervision labels for the suction, placement, and matching networks, reverse the dismantling procedure.

2.3.3 Review of Paper-[4]

[4] demonstrates that model-free deep Reinforcement Learning can be used to gain an understanding of the synergistic interactions between grasping and pushing Some of the important contributions are :

(i) The process of self-supervised trial and error is used for development of joint pushing and grasping policies.

(ii) Policies trained end2end whose input is visually perceivable observations and output is Expected Return (in Q-values).

The problem is treated as an MDP where the aim of the agent is to follow a strategy that will improve the discounted sum of future rewards. The state is characterized by RGB-D

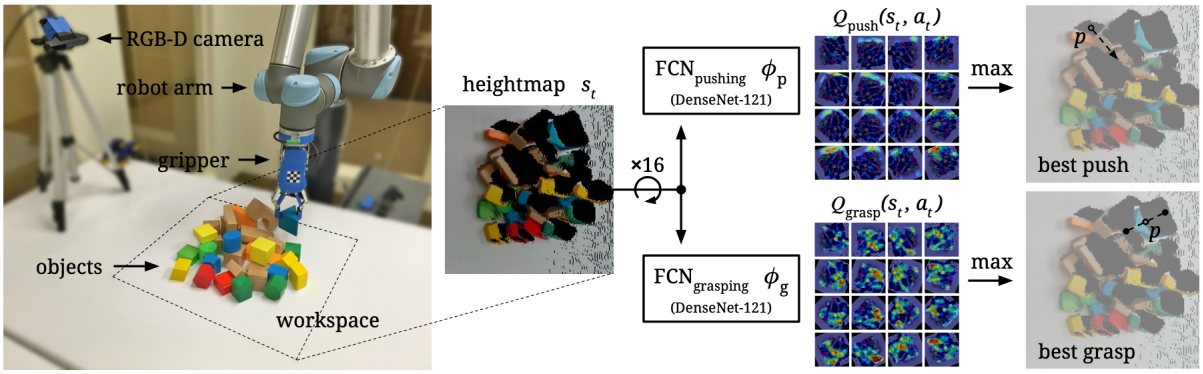


Fig. 2.8: Overview of the complete system and the different modules present.

heightmap picture of the scene. Off-policy Q-learning is utilized to guide the policy, where a deterministic but greedy policy picks actions by maximizing the $Q_\pi\left(s_t, a_t\right)$.

The definitions of two motion primitive activities are:

1. Pushing $\rightarrow$ $10\,\mathrm{cm}$ push in 16 different directions.
2. Grasping $\rightarrow$ move $3\,\mathrm{cm}$ below and grasp in 16 different orientations.

$$a = (\psi, q) \mid \psi \in \{grasp, push\},$$
$$q \rightarrow p \in s_t$$

Rewards are :

1. 1 if a grasp is successful
2. 0.5 for efforts that produce observable changes in the environment

Two Fully Convolutional Neural Networks (FCNs) are considered, one for each motion primitive :

- Two parallel 121-layer DenseNet towers $\rightarrow$ Channel Wise Concat. + 2 additional 1 x 1 conv. layers $\rightarrow$ Bilinear Upsampling.
- One tower takes input as RGB and other takes DDD map.

Q-Learning was implemented through the Huber Loss function, as it is less sensitive to outliers as compared to Mean Squared Loss :

$$\mathcal{L}_i = \begin{cases} \frac{1}{2}\left(Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-}\right)^2, \text{ for } \left|Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-}\right| < 1 \\ \left|Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-}\right| - \frac{1}{2}, \text{ otherwise} \end{cases}$$

The optimizer used is the SGD with momentum along with Weight Decay. In addition to these, prioritized Experience Replay for training was used and ε-greedy exploration strategy was considered.

All the models were trained by self-supervision :

→ *n* toy blocks are dropped randomly in the workspace

→ Automatic data gathering via trial and error until the workspace is completely devoid of items

→ Afterwards, a total of *n* items are thrown onto the workspace at random locations again.

Both real and simulated experiments were conducted and many ablation studies were done.

2.4 Relevance of ARG Systems

ARG systems have an immense capability to add value by enhancing the operations of traditional approaches. They have advanced to the point that they can operate around the clock with a higher degree of consistency in terms of quality and productivity, completing activities that people are unable or unwilling to undertake. Their relevance is captured by understanding the humongous benefits that they offer:

(i) Enhance efficiency and productivity by lowering the rate of mistake, re-work, and risk.

(ii) Increase the safety of workers who operate in high-risk areas.

(iii) Optimize the speed and precision of mundane processes, notably in the warehouse and manufacturing industries.

(iv) Work with humans in a safe and collaborative manner so that people can give more attention to strategic tasks that are difficult to automate.

(v) Increase income through increasing the rate of flawless order fulfilment, the speed of delivery, and ultimately, the level of customer happiness.

2.5 Challenges of ARG

Despite the tremendous advances made in the fields of AI and intelligent robotics, there are still some crucial challenges [7] that need to be addressed by ARG researchers:

1. Radically enhancing the ability of the robot to learn from less number of samples or demonstrations, so that the learning algorithms become more resource efficient.

2. Imbibing the concept of safe robotic manipulation workspaces into the grasping algorithms so that damage avoidance can be guaranteed at different stages of trajectory execution.

3. Development of task agnostic algorithms that may have the capacity to transfer well among drastically different grasp task families.

4. Forming a proper framework of continual learning where learned components from regular interactions of the robotic arm with the external environment can be used seamlessly for the upcoming interactions.

5. Coming up with robust algorithms that can exploit "common-sense" knowledge and explore using strategies with proper reasoning.

6. Providing the ARG system with a framework to infuse useful data from multiple sensor modalities to accomplish the task at hand more effectively.

2.6 Chapter Summary

This chapter briefly reviews the literature to set up the overall context for ARG approaches studied in this work. In the later sections, various details regarding the relevance and typical challenges associated with ARG are discussed. The next chapter provides a detailed description of Task I : Grasping Various Objects in Diverse Environments.

Chapter 3

Task I - Grasping Various Objects in Diverse Environments

This chapter describes various elements involved in realizing the first task involving picking up novel objects across varied unforeseen environments. Deep Reinforcement Learning (DRL) techniques are mainly utilized in this work. Details regarding the formulation of the task, design of the entire approach, implementation details and architectures of the various DL models utilized are illustrated in this chapter. Some of the ideas for the task design were taken from [8]. The results of various experiments conducted as a part of this task will be presented in a forthcoming chapter.

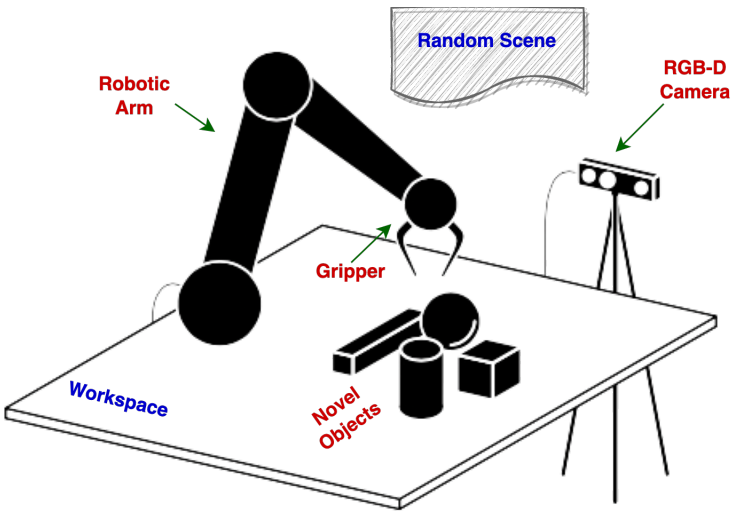


Fig. 3.1: Illustration of basic problem setup of Task-1. Note the use of an RGB-D camera as a perception module.

3.1 Task Formulation

The first task taken up, *Grasping Various Objects in Diverse Environments*, is conceived as a Markov Decision Process (MDP), where the agent is a high-level controller for sequential decision making in the forms of actions in cartesian space. The following episodic formulation is considered :

- ⋆ *Episode Start* → New set of objects (3D Model is not known) is given.
- ⋆ *During Episode* → Aim is to grasp an object and lift in 12.5 cm above the ground.
- ⋆ *Episode End* → Success (or) 100 time-steps* {Failure}

3.1.1 Observation Space

The following observations were considered that would be further processed:

- *Visual Octree Observations* – The following data is organized in the finest leaf octants (primitive features) to form the input signal :
  - average unit normal vector, $\bar{n}$
  - average distance from centre, $\bar{d}$
  - average colour, $\overline{rgb}$
- *Proprioceptive Observations* – State of the gripper (open/closed) and gripper pose (position + orientation) contribute to this.

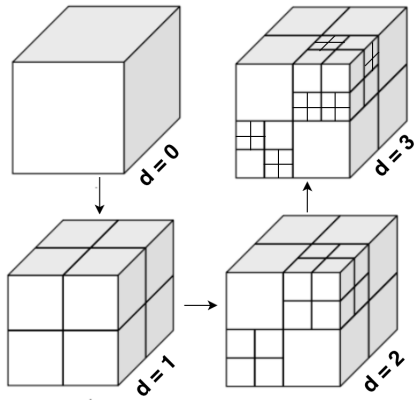


Fig. 3.2: Visual illustration of hierarchical octree data structure at different depths, *d*. Note that, RGB image and depth maps are used to make coloured point clouds, which are further processed to form octrees.

A Note on Octree Observations

Why not RGB or RGB-D observations ?

2D RGB or 2.5 RGB-D image observations can be processed by 2D CNNs but they do not provide required level of generalisation over the depth and spatial orientation [8]. Hence,

*Note that here time-step indicates the minimum interval where physics parameters can be changed in the simulation. Ignition Gazebo allows tunablity in time-step size so that the simulation can run faster than real-time, real-time or even slower than real-time.

there is a need for a compact 3D data structure that provides both tractability and detail.

Background on Octrees

Some other possible 3D representations of data other than octrees are :

- 3D Mesh → Irregular 3D shape → Typical CNNs cannot be used.
- Full Voxels → Regular 3D shape → CNNs can be used but not efficiently.

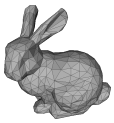


Fig. 3.3: Example of Mesh representation

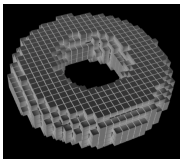


Fig. 3.4: Example of Full Voxel representation

Octree is a 3D representation as a hierarchical tree (see Fig. 3.2) where each cell can be recursively decomposed into eight octants. Note that, size of voxels can be varied according to object occupancy. Octree based observations provide more spatial generalization than 2D RGB (or) 2.5 RGB-D images while being more tractable and compact than full voxel representation.

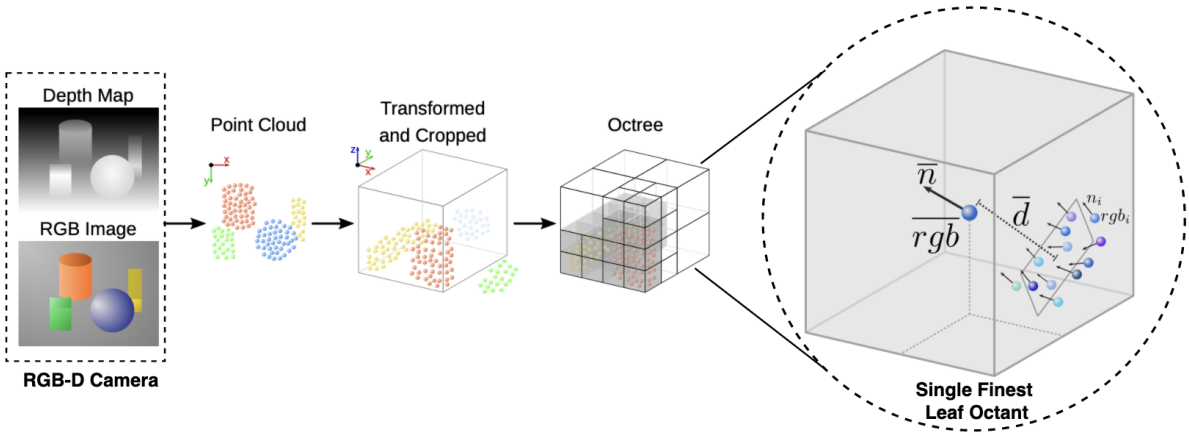


Fig. 3.5: Chronological Order of various pre-processing steps that are involved in the formation of a Octree 3D representation. On the extreme right, the data stored in a finest leaf octant (See Section 3.1.1) is pictorially illustrated.

In Octree Data Structure, each octant has [9] :

- Input Signal
- Shuffle Keys
- Labels

Input Signal, which can be regarded as the primitive features extracted, is the actual "data" that will be processed to form high-level features. Shuffle Keys and Labels, collectively known as property vectors, are additional auxiliary data vectors extracted to facilitate efficient computation. More details regarding efficient implementation of octree processing will be discussed in Section 3.5.1.

In this work, the *maximum octree depth* is 4, the *observable workspace* is a cube of volume $24 \times 24 \times 24\ cm^3$ and *metric resolution* of leaf octant is $\{1.5 \times 1.5 \times 1.5\}\ cm^3$.

3.1.2 Action Space

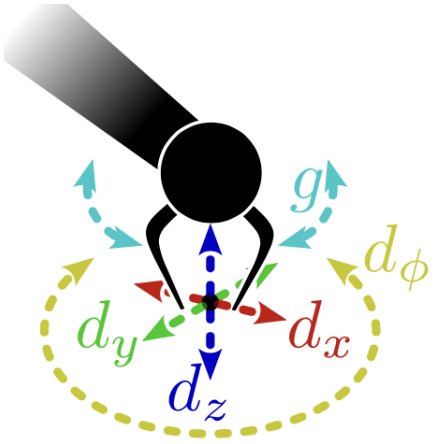


Fig. 3.6: Illustrative representations of different possible actions that can be predicted by the agent. The idea is to use optimized traditional motion planning approaches to execute these actions without many collisions.

The action space consists of continuous actions in cartesian space rather than low level joint space observations. Different possible actions are :

(i) translational displacement, (d_x, d_y, d_z)

(ii) rotation around z-axis, d_ϕ

(iii) gripper closing and opening, g

Traditional IK and motion planning approaches (inbuilt in *MoveIt2*) used to provide commands for low-level joint controllers and avoid self-collisions.

3.1.3 Reward Function

It may be generally desirable to give reward only on success of the episode, but training might get prolonged due to sparsity of success through random exploration. To avoid such problems, the reward structure shown in Fig. 3.7 is designed to guide training.

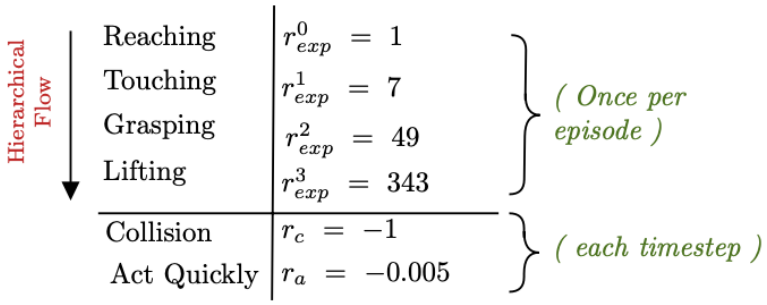


Fig. 3.7: Structure of the Reward Function

The intuition is that the robotic arm should first *reach*, then *touch*, then *grasp* and finally *lift* an object without *colliding*, while acting as *quickly* as possible. Note that, the theoretical maximum achievable reward is $r_{\max} = 400$.

3.2 Approach Design

The task formulation suggests the use of Deep Reinforcement Learning (DRL) algorithms to learn robust and effective policies to enable this specific grasping task. In this section, an overview of the RL setup is provided followed by an outline of relevant DRL algorithms to this work.

3.2.1 A Brief Overview of RL

When an RL agent interacts with the environment sequentially, the aim is to maximise the total expected return as much as possible for the agent. This problem can be expressed using MDPs (Markov Decision Processes), which are typically described as a tuple $(\mathcal{S}, p, \mathcal{A}, r)$. Here, $\mathcal{A}$ and $\mathcal{S}$ are the action and state spaces respectively (here, continuous) . The state transition probability $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ denotes the probability density of the succeeding state $s^+ \in \mathcal{S}$, given current state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ as :

$$p\left(s^+ \mid s, a\right) = \Pr\left\{S_{t+1} = s^+ \mid A_t = a, S_t = s\right\} \tag{3.1}$$

The sum of discounted rewards, $r(s_i, a_i)$ (with a discount factor) that an agent seeks to maximize is given by :

$$G_t = \sum_{i=t}^{T} \gamma^{i-t} r\left(s_i, a_i\right) \tag{3.2}$$

The behaviour of an agent is defined by a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The action-value function for taking an action a in state s and then following policy π is : $Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t \mid A_t = a, S_t = s\right]$. For the continuous control problem, the agent's aim is to discover the optimal policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the action-value function :

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{3.3}$$

To make RL scalable to real life problems, the following tricks are commonly utilized:

- Estimating Expectations using Sampling:

$$\mathrm{E}[f(\boldsymbol{x})] \approx \frac{1}{N}\sum_{i} f\left(\boldsymbol{x}_i\right) \tag{3.4}$$

- Approximating Functions by Deep Neural Networks (DNNs):

$$f(\boldsymbol{x}) \approx f_{\theta}(\boldsymbol{x}) \quad ; \quad \theta \rightarrow \text{ DNN parameters} \tag{3.5}$$

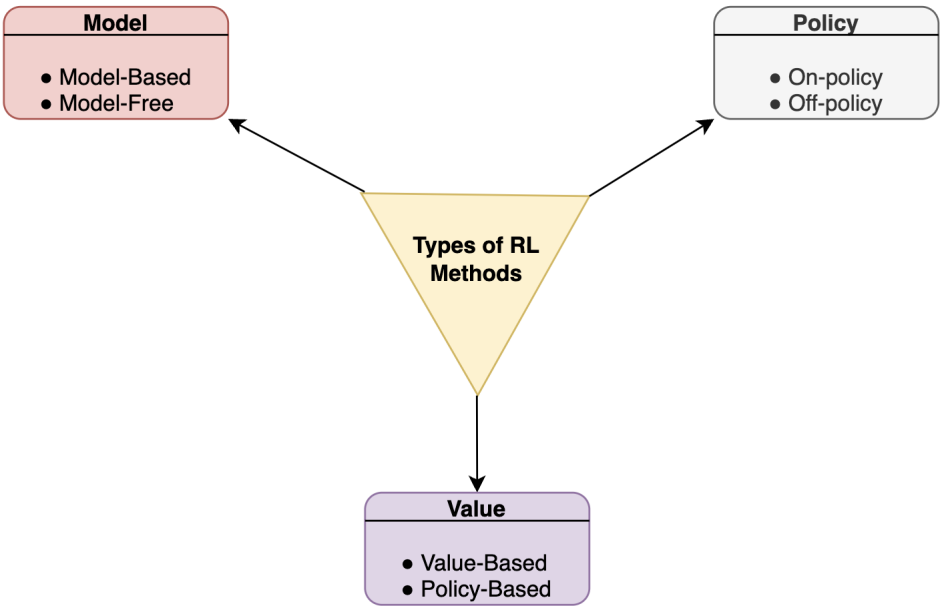


Fig. 3.8: An illustration of various classes of RL algorithms available in Literature

3.2.2 Relevant DRL Algorithms

In this work, the main focus is on the usage of Actor-Critic algorithms. These algorithms can be viewed as a combination of value-based and policy-based methods. Actor network learns the parametrized policy $\pi(a \mid s, \theta)$ whereas Critic network *critiques* the actor based on how good the selected action is by estimating the value function $Q(s, a)$ by minimising the Temporal Difference (TD) error δ_t :

$$\delta_t = r_t + \gamma \max_{a'} Q^{\pi}\left(s_{t+1}, a'\right) - Q^{\pi}\left(s_t, a_t\right) \tag{3.6}$$

Some popularly used *Model-free* , Off-policy Actor-Critic algorithms relevant to this work are briefly described as follows :

1. **Deep Deterministic Policy Gradient (DDPG) : [10]**

   - Off-policy algorithm and makes use of Experience Replay buffer to be sample-efficient.
   - Deals with continuous action spaces and trains a deterministic policy.
   - Target networks are utilized to ensure training stability, whose weights are exponentially averaged versions of the original network.
   - To accommodate exploration, noise is added to actions during training.

2. **Twin Delayed Deep Deterministic Policy Gradient (TD3) : [11]**

   - It is an extension to DDPG, designed to alleviate the typical Q-value overestimation bias.
   - The following three tricks (contributions) are incorporated into the framework to performance imporvement :
     (i) Use of two individual critics where the smaller of the two is used to compute the Temporal Difference (TD) error.
     (ii) Updating actor network less often than critic networks.
     (iii) Addition of smoothing noise for the actor network, which makes it difficult for the policy to capitalize on the errors of the critic.

3. **Soft Actor Critic (SAC) :** [12]

   - It is one of the state-of-the-art algorithms for continuous control, which inherits the use of two critics and actor network smoothing noise from the previous methods.
   - It optimizes a stochastic policy and the main addition is the use of entropy-regularized reinforcement learning framework, where the agent's aim to learn an effective policy by optimizing a trade-off between expected return and entropy (analogy to trading off exploration and exploitation) :

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T} \gamma^t \left(r\left(s_t, a_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid s_t\right)\right)\right)\right] \tag{3.7}$$

     where entropy is defined as:

$$\mathcal{H}(P) = \underset{x \sim P}{\mathrm{E}}[-\log P(x)] \tag{3.8}$$

   - The intuition is that larger entropy leads to more exploration.

4. **Truncated Quantile Critics (TQC) :** †[13]

   - It is a recently added extension to the SAC algorithm, which addresses the following shortcomings of the previous approaches :
     (i) Overestimation control is coarse (two Q-networks and min. is chosen)
     (ii) Min. of estimators diminishes the power of the ensemble of approximators.
   - The main ideas used in this algorithm are :
     (i) It models the distribution of the random return rather than expectation of the return. The action-value function $Q(s, a)$ of critics is modelled as a distribution with n-number of atoms.
     (ii) To keep the overestimation under control, truncate the right tail of the return distribution approximation by removing the atoms at the very top of the distribution.
     (iii) Mixture of multiple critics is considered (Ensemble advantage).

†Mainly used in this work

3.2.3 High Level Block Diagram

The environment under consideration can provide information of the current state (S_t) and reward (r_t). The *Feature Extractor Backbone* processes the state information to form high level features. Actor network learns the odds of selecting a specific action a in a given state s as $\pi(a \mid s, \theta)$. The critic network estimates action-value function $Q(s, a)$ by minimising TD error δ_t, which is used to critique the actor pased on how good the predicted action is. This process has been illustrated in Fig. 3.9.

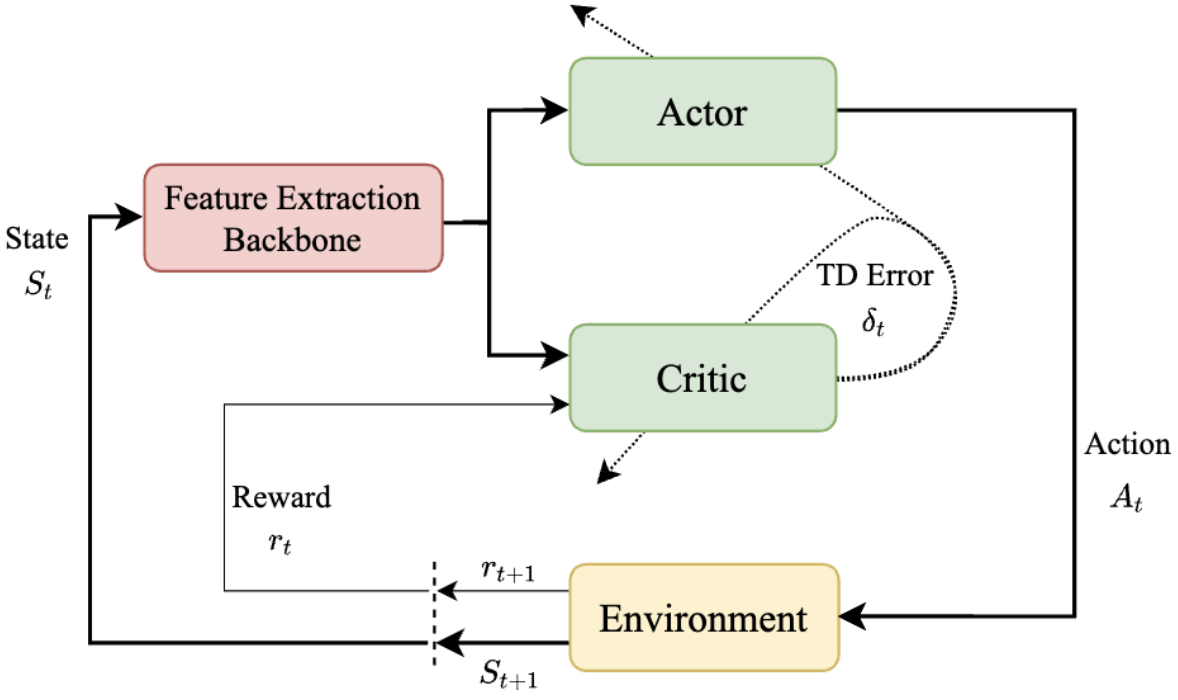
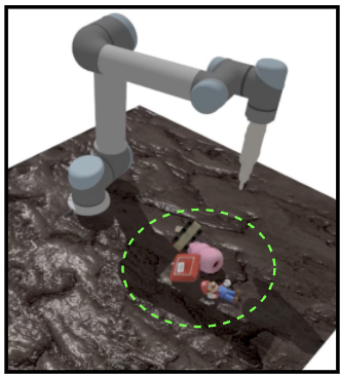
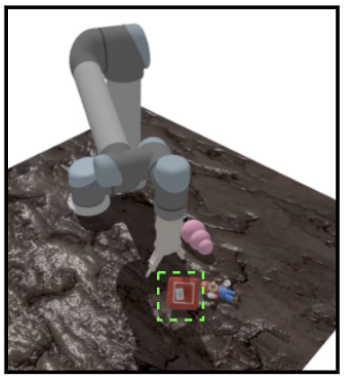


Fig. 3.9: Illustrative high level block diagram of the approach used in this work. Note that here, feature extraction backbone, actor and critic blocks are DL based neural network function approximators.

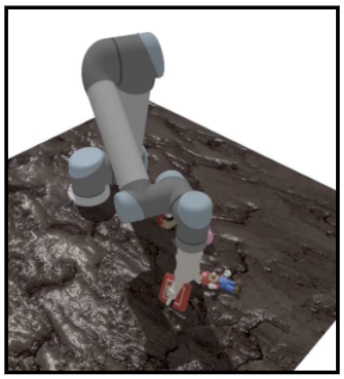
3.3 Pictorial Demonstration



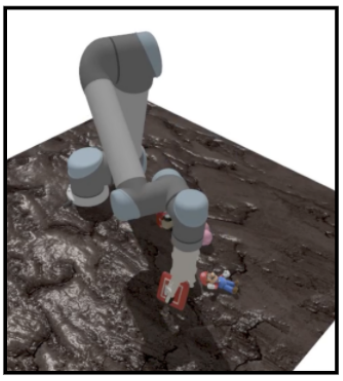
(a) Start of a new episode. A group of novel objects spawned



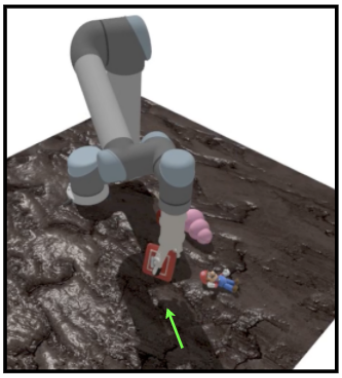
(b) Robotic arm clears up the object clutter. "Intention" seems to be to pick "red bowl"



(c) Robotic Arm "reaches" the object and enters into a pregrasp pose



(d) "red bowl" object is grasped successfully



(e) Robotic Arm "lifts" the grasped object 12.5 cm above ground

(f) Previous episode ended successfully. New episode evaluation begins

Fig. 3.10: Pictorial Demonstration of Task-1. The keyframes corresponding to various steps of the task are shown in a chronological manner, in a novel simulation environment with a UR5 arm. Notice that the robotic arm first *reaches* a novel object, makes contact with it in a pregrasp, then *grasps* the object before lifting it into the air. This marks the end of a successful episode.

3.4 Implementation Details

In this section, some aspects related to the implementation of the virtual setup for training and testing DRL algorithms is discussed. Note that more details regarding the Configuration and hyperparameters will be provided in Chapter 5, Section 5.1.1.

Simulation Environment

Ignition Gazebo[‡] is used as the simulation environment to facilitate realistic rendering. It is an open-source robotics simulator derived from gazebo. Some of its prominent features can be listed as follows:

- It allows the use of rendering engines such as OGRE v2 for realistic rendering of environments with superior quality lighting, textures and shadows.
- It provides access to high performance physics engines through *Ignition Physics*.
- It is possible to use a variety of virtual sensors, such as 2D/3D cameras, IMU, GPS, etc., with an additional option to add noise for more realistic data.
- *Ignition GUI* is a plugin-based graphical interface that allows users to create, inspect, and interact with your simulations using graphical interfaces.

Middleware Software

ROS2 is used as middleware for this work which handles communication between primary nodes such as RGBD data stream and processes requests from motion planner and the simulation environment itself.

Motion Planning

For trajectory generation, solving kinematics and motion planning MoveIt2[§] framework is utilized. There exist two popular methods for generating the motion plan for the robot by knowing the start and end poses of the end effector:

[‡] https://ignitionrobotics.org
[§] https://moveit.ros.org

1. **Joint Interpolated Motion –**

   → From known start and end poses of the end effector, inverse kinematics is used to get the intial and final joint angle configuration vectors.

   → Linear (vector) interpolation is then done with time parametrization to get motion plan.

   → In this case, the end effector may not follow a straight line path but computational burden is less.

2. **Cartesian Interpolated Motion –**

   → From known start and end poses of the end effector, directly interpolation of poses is done (interpolation of translational vector and rotational quaternion)

   → Then, we get joint configuration vectors using inverse kinematics at every time sample for the motion plan.

   → In this case, the end effector follows a straight line path in 3D but computational burden is high.

MoveIt2 is an open-source software library used to plan and execute motion for serial link manipulators. In this work, the functional modules (See Fig. 3.11) of this package are extensively used for the purpose of trajectory planning for the pick and place task.

Different packages and algorithms for each functional modules in MoveIt2 motion planner. The default configuration is utilized, where for Inverse Kinematics, `KDL Kinematics Plugin` is used and for Motion planning `OMPL Library` is used. In OMPL planner, the `RRTConnect` algorithm is used to generate the path[¶].

Note that, taking the tradeoff between speed and optimality for a given motion planning problem, MoveIt2 chooses between joint space and cartesian space interpolation.

Perception

During simulation virtual camera is used to provide RGBD data of the scene, which is further processed into octree observations. Note that, noise is added to this data to account for real world sensor noise.

[¶]Samples random valid joint configs. between start and target states and generates path

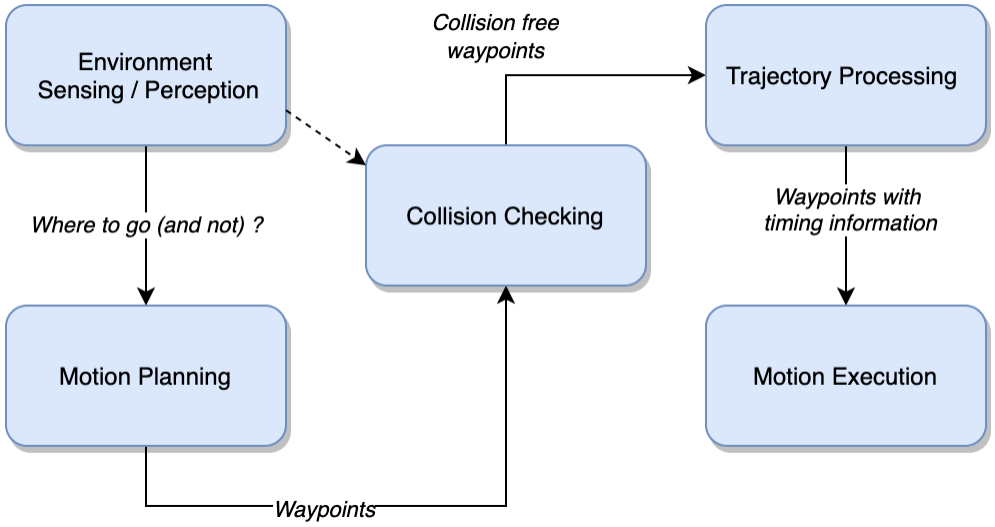


Fig. 3.11: Different Function Modules and their typical order of usage. This represents the typical flow used by MoveIt2 package to perform motion planning and execution.

Robotic Arms

In this work, mostly UR5 arm with RG2 gripper and Franka Emika Panda robotic arm are used for various experiments.

Objects

For training and testing the robotic arm's performance inside the simulation environment[8], dataset of scanned objects provided by Google Research is being utilized

Fig. 3.12: Pictorial Representation of various objects being utilized in different stages is shown. Notice how the test objects are not present in the train subset, to test the performance of the robotic arm on novel objects.

Domain Randomization

A basic strategy for bridging the simulation-reality gap for synthetic data is to randomise the simulator's surroundings throughout the training process, rather than training an agent on a single simulated environment. This exposes the model to a broad variety of environments during training time. The core concept is that, if the variability in simulation is substantial enough, models trained in a simulated world will generalise to reality without the need for extra training or experimentation.

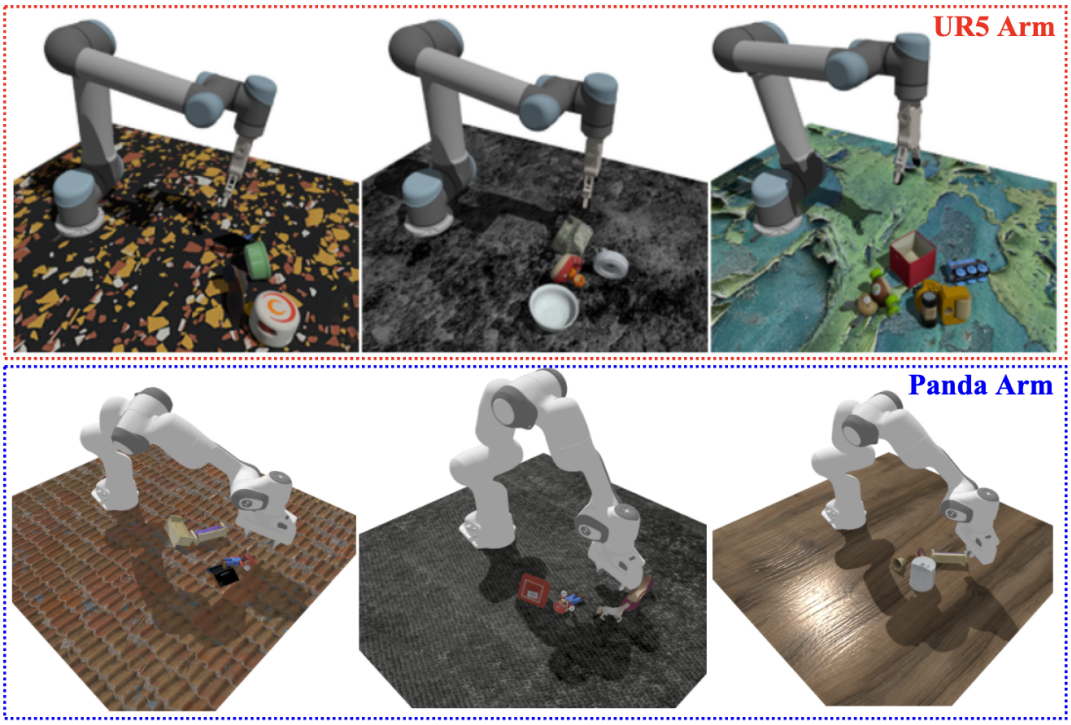


Fig. 3.13: A collection of images depicting the superimposed effect of all domain randomizers applied to the simulation scenario for both the robotic arms under consideration.

In this work, the following types of domain randomization aspects are considered [8] :

- *Object Property Randomizer* – A few objects from the dataset under consideration (See Fig. 3.12) are spawned by varying different inertial and mechanical properties.
- *Object Pose Randomizer* – The position and orientation of the objects chosen are also randomized in each episode.
- *Environment Randomizer* – The ground plane textures are also chosen randomly at every reset to randomize the visual environment in each episode. Note that, it is made

sure that the test time material textures are unseen during training time to evaluate the robot's performance in novel environments.

- *Camera Pose Randomizer* – In order to make the training of the agent more robust, small random disturbances are added to the pose of camera.

- *Initial Joint Configuration Randomizer* – To make the setting more realistic, the initial joint angles of the robotic arm are set randomly at the beginning of each episode.

Framework

Implementation of DRL algorithms from scratch is error prone and time consuming, so for the implementation of RL algorithms the open source and reliable Stable Baselines3 (based on PyTorch) framework is used as a baseline for further developement.

3.5 Deep Learning (DL) Architectures

In this section, a detailed description of the different DL architectures designed, for different parts of the approach as shown in Fig. 3.9, is given. The overall architecture of each model and related concepts are described in this chapter. More details about the training configuration and performance evaluation of the various models will be presented in a forthcoming chapter.

3.5.1 Octree Processing

As mentioned in Section 3.1.1, in octree data structure, each octant consists of input signal and property vectors [9]. Input signal is the actual "data" that will be processed to form high-level features, whereas property vectors are additional auxiliary data vectors extracted to facilitate efficient computation. The property vectors include [9] :

- *Shuffle Key* –

  → $3l$ bit key for each octant encoding the position in 3D space at depth l.

  → In the form of $(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_l, y_l, z_l)$ where $x_i, y_i, z_i \in \{0, 1\}$ denotes the relative position in parent octant.

→ Octants are sorted according to shuffle keys.

- *Label* –

→ To rapidly determine the parent-child link of the octants at adjoining depths.

→ Label p specifies that it is the p^{th} non-vacant octant at at depth l.

→ Empty octants are labelled 0.

Now, more details about how some of the typical operations are implemented more efficiently is discussed [9].

Operations : 3D Convolution

Convolution operator can be written in unrolled form as :

$$\Phi_c(O) = \sum_n \sum_i \sum_j \sum_k W_{ijk}^{(n)} \cdot T^{(n)}\left(O_{ijk}\right) \tag{3.9}$$

where:

$O_{ijk} \rightarrow$ represents an adjacent octant of O

$T^{(n)}(\cdot) \rightarrow$ represents the n^{th} channel of the feature vector associated with associated with O_{ijk}

$W_{ijk}^{(n)} \rightarrow$ represents the weights of convolution

For efficient implementation of Octree 3D convolutions[9], the following tricks were used :

(i) For performing the convolution operation, we need quick access to the neighbours of the octant, so hash table $\mathcal{H} : \text{key}(O) \mapsto \text{index}(O)$ is implemented to search for the neighbours in constant time.

(ii) Parallelly calculate the 3D convolutions of eight sibling octants by creating a shared neighbourhood.

(iii) For batch processing, the individual octrees are merged into one super-octree.

Operations : Pooling

In terms of functionality, pooling is primarily concerned with gradually shrinking the spatial extent of the representation:

- Each of the eight related octants of the same parent are kept in sequence.
- Pick out the largest from every 8 continuously stored elements.
- Depth reduces by one level when a pool layer is used.

Similarly, *Transposed Convolution* operations are also implemented efficiently.

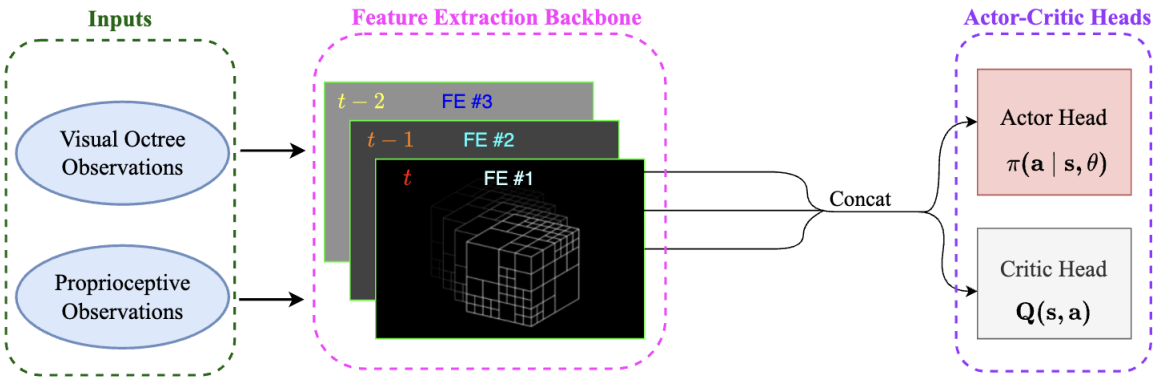


Fig. 3.14: An illustration of various DL models utilized in the Actor-Critic Algorithm Setting. Note the stacking of historical observations in the Feature Extraction Backbone.

3.5.2 Feature Extraction Backbone

Now, the various DL models that were designed to improve the Feature Extraction from Octrees are described. The intuition is that the architectures of the feature extractor, actor and critic network play a pivotal role in the learning process and hence, designing effective and efficient architectures can lead to better performance.

Note

At each timestep t observations at $t-1$ and $t-2$ are also considered and are processed in separate *copies* of the backbone before concatenating the final high-level feature vectors (See Fig. 3.14), to provide temporal context to the agent. Common backbone for processing observations at all timesteps $\{t, t-1, t-2\}$ was not considered because different type of

features maybe required to be extracted from different timesteps. This gave better results than its counterpart experimentally as per [8].

3.5.2.1 Model-1 : Vanilla O-CNN

The baseline architecture [8] of the Octree processing Feature Extraction backbone *Vanilla O-CNN*. The complete architecture of Model-1 *Vanilla O-CNN* is shown in Fig. 3.15.

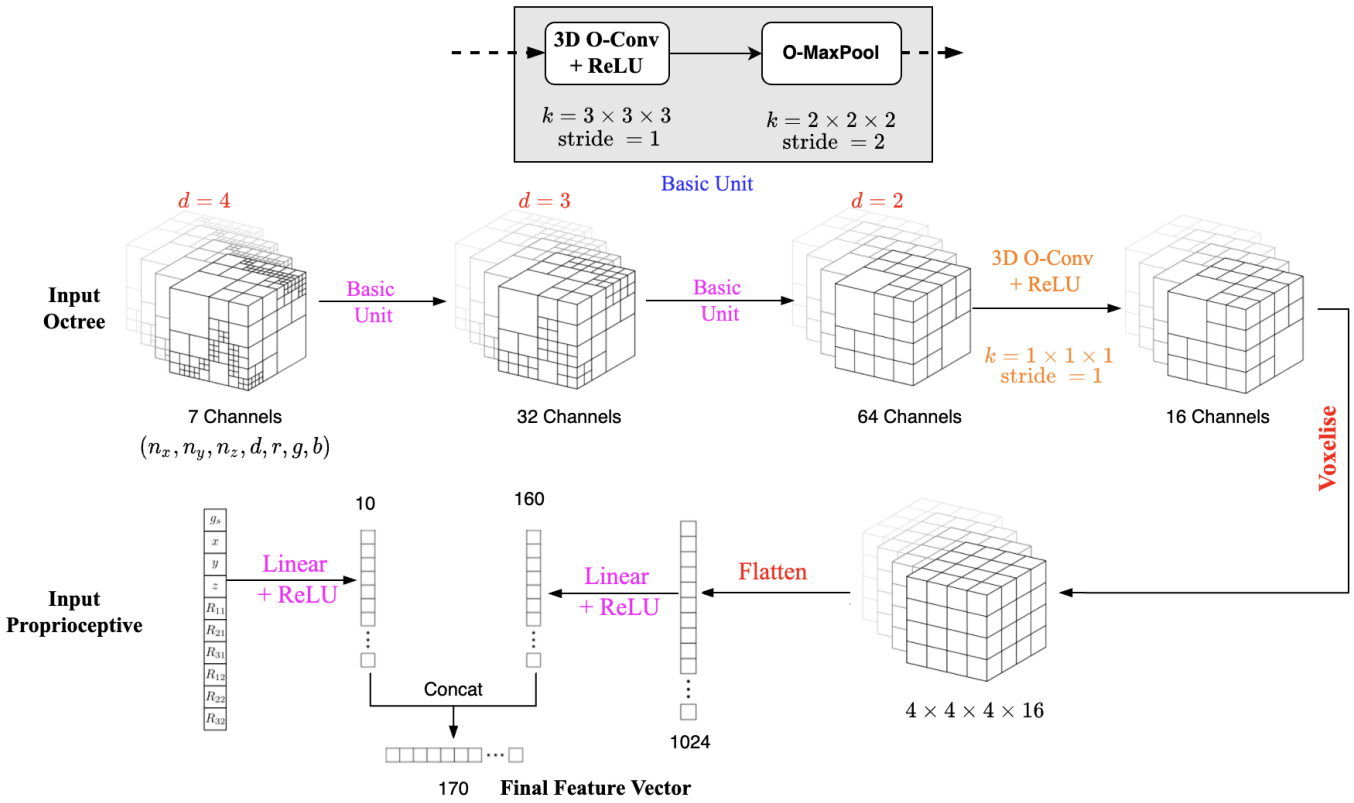


Fig. 3.15: Architecture of Vanilla O-CNN feature extraction backbone. d in the figure indicates the octree depth. Note that, this is the architecture for the observation stack at one timestep. The feature extractor is duplicated for each stack in order to process two historical observations in addition to the current one.

The following are some of the features of this model:

- The input octree to the model has 7 channels $(n_x, n_y, n_z, d, r, g, b)$ as described in Section 3.1.1.
- The further processing of the octree can be explained by using a *Basic Unit*, consisting of a 3D *O-Conv*, followed by a ReLU non-linearity and then a *O-Maxpool* operation. The *Basic Unit* reduces octree depth by single unit.

- Before the processed octree is voxelised, 3D *O-Conv* is performed with a $1 \times 1 \times 1$ kernel. Note that, since all spatial elements of the octree are stored in an array, 3D O-CNN for kernel size of $1 \times 1 \times 1$ is implemented using 1D Convolution of kernel size 1.

- Then the full-voxel feature units are flattened before being processed by a fully connected layer.

- Finally, this feature vector is concatenated with the processed input proprioceptive observations to form the *Final Feature Vector*.

Here in the *Vanilla O-CNN* model, increasing the depth of the network for obtaining a better hierarchy of feature representations may lead to *Vanishing Gradient* problem. So we need to look for more advanced feature extraction architectures that allow this increase in network depth while maintaining parameter efficiency and not encountering vanishing gradients.

3.5.2.2 Model-2 : Residual O-CNN

Model-2 *Residual O-CNN* architecture was designed by incorporating the idea of residual skip connections from [14]. It provides the typical advantages such as:

- Better gradient flow during backpropagation

- Identity mapping to avoid excessive morphing of hidden representations.

The detailed architecture of *Residual O-CNN* is illustrated in Fig. 3.16. It should noted that n here is a design hyperparameter[†]. n can be chosen by performing an experimental tradeoff study between network depth (more layers may lead to better feature representations) and parameter efficiency (more parameters may cause overfitting).

The step-through of different stages of the model remains the same as Model-1 *Vanilla O-CNN* as described in Section 3.5.2.1. The main modification is made in the *Basic Unit* of the model.

[†]Experiments with different n values and corresponding results are discussed in Chapter 5, Section 5.2

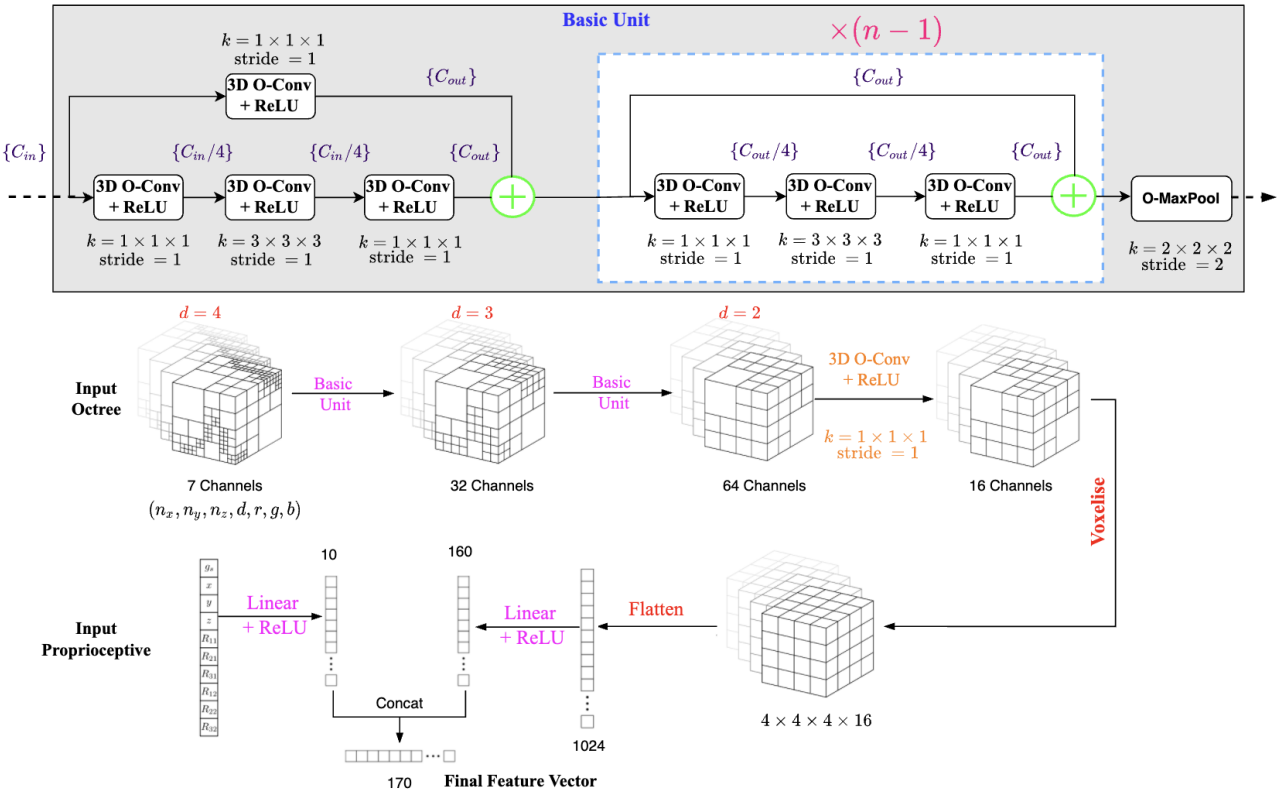


Fig. 3.16: Architecture of *Residual O-CNN* feature extraction backbone. Note that, here n is a design parameter. Similar to *Vanilla O-CNN*, this is the architecture for the observation stack at one timestep.

3.5.2.3 Model-3 : O-AHRNet

The next model designed for extraction octree features is *O-AHRNet*[§]. The high level block diagram is shown in Fig. 3.17 and the detailed architecture of this model is pictorially illustrated in Fig. 3.23.

The main components of Model-3 *O-AHRNet* are:

- Attention Module → Channel + Spatial Attention
- High Resolution → Repeated Multi-depth Octree Fusion Features

Now, each of these components are explained below.

Attention Module

In this section, we will focus on the various components that make up the attention module. The attention module was introduced for adaptive feature refinement of intermediate feature

[§]Stands for **O**ctree based **A**ttention **H**igh **R**esolution **Net**work

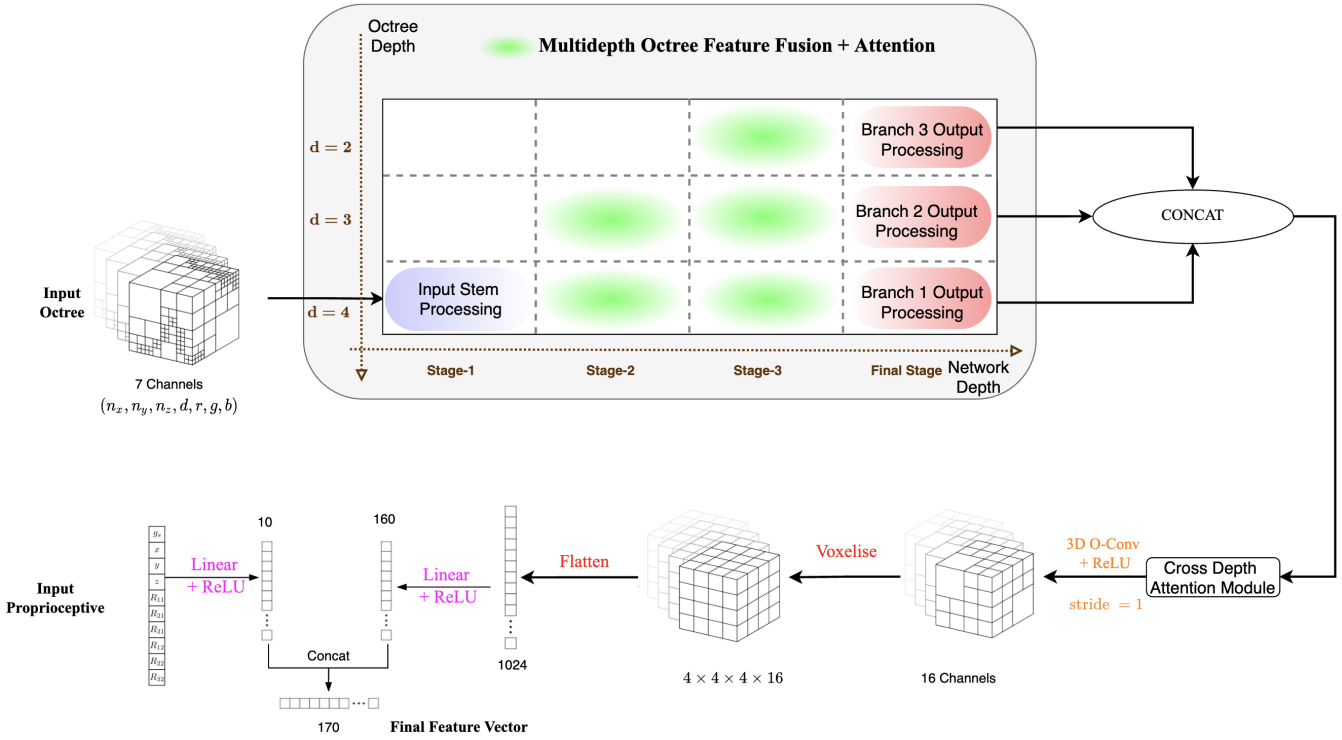


Fig. 3.17: High Level Block Diagram of *O-AHRNet* architecture. The main components of the approach such as High Resolution and use of attention are illustrated. Note that, details about the input stem processing and the output branch processing is described in the Detailed Architecture.

maps, by telling the model "where" to focus and improve its hidden representations. Main idea is to compel the model to concentrate on essential traits and ignore unimportant ones. Incorporation of Attention module was done by the considering the best empirical practices in [15] and [16], found by extensive experimentation. Note that, this attention module was originally designed for typical 2D Convolution operations but later it was adapted for 3D *O-Conv* operations. Each attention module basically contains two sub-modules namely : *Channel Attention* and *Spatial Attention* as shown in Fig. 3.18.

■ *Channel Attention Sub-Module* –

The intuition behind channel attention sub-module is to improve the feature blocks by cross-channel interaction. One way that can be done is by selectively weighting each feature channel adaptively. In other words, we are refining features in a channel by using information from all channels.

In general, there are two popular types of channel attention mechanisms used as shown in Fig. 3.19 : *SE block* and *ECA block*. In [16], it was shown empirically that for an

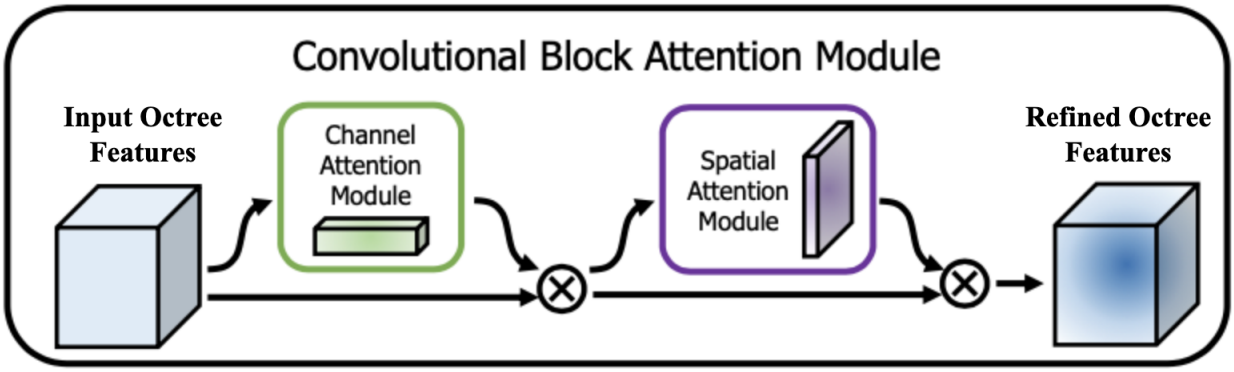


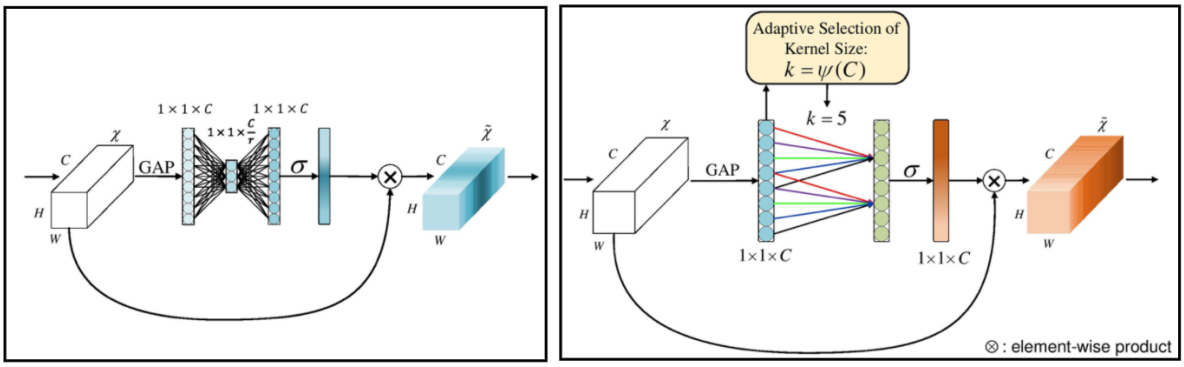
Fig. 3.18: Illustrative Representation of the Attention Module. Note that, the attention module is made up of submodules : *Channel Attention* and *Spatial Attention*.

improvement in performance avoiding dimensionality reduction is important. The empirical reasons why the ECA block was chosen for channel attention are :

→ SE block destroys direct correspondence (due to dimensionality reduction) between channel and weight, which might be useful for deciding the importance of a particular channel.

→ ECA block uses a 1D convolution, hence limiting the number of parameters as compared to SE block, which uses fully connected layers.

Now, more details regarding the implementation of the channel attention sub-module are specified. As shown in Fig. 3.20, in this work, both the O-MaxPool and O-AvgPool features (pooling performed along the spatial dimensions, to get a 1D vector of length equal to number of channels) are passed through a shared 1D convolutional layer.



(a) Squeeze and Excite (SE) Block (b) Efficient Channel Attention (ECA) Block

Fig. 3.19: Different types of Channel Attention mechanisms. These are popularly used in methods, based on different principles. GAP stands for **G**lobal **A**verage **P**ooling.

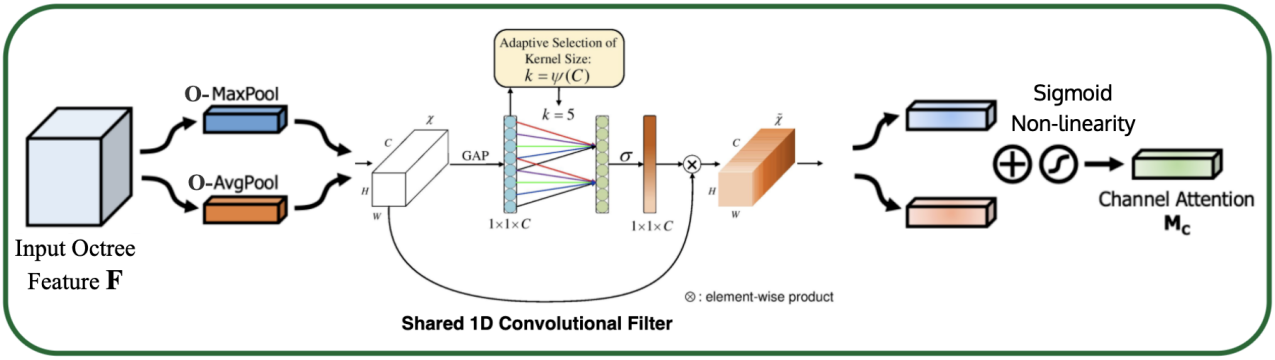


Fig. 3.20: Channel Attention Sub-Module Architecture. Notice the use of Effective Channel Attention (ECA) block due to its advantages.

The intuition behind using both the type of features is that (In [15], it is shown through some experiments that that both are complementary) :

→ Max-pooled features convey the degree of the octree feature block's most prominent element.

→ Average-pooled features capture global statistics in a soft manner.

Now, the kernel size (k) for the shared 1D convolutional layer is selected adaptively (to avoid extensive hyperparameter tuning), based on the number channels (C) involved, given by the following expression :

$$k = \psi(C) = \left| \frac{\log_2(C)}{\gamma} + \frac{b}{\gamma} \right|_{\text{odd}} \tag{3.10}$$

Here, $b = 1, \gamma = 2$ is considered and C is the number of channels which have to be selectivey weighted. Observing this, we can see that the intuition is higher number of channels should undergo longer range of interaction, hence larger kernel size.

■ *Spatial Attention Sub-Module* –

The basic architecture for spatial attention sub-module is shown in Fig. 3.21. In this module, spatial attention feature block is obtained which can be used to improve features utilizing the spatial link between features. In other words, it is helps the model to basically decide "where" to focus in a feature octree. As shown in Fig. 3.21, we are using both the O-MaxPool and O-AvgPool features (performed along the channel dimension, to get 3D feature octree) are used for the same reasons as discussed in the previous section on

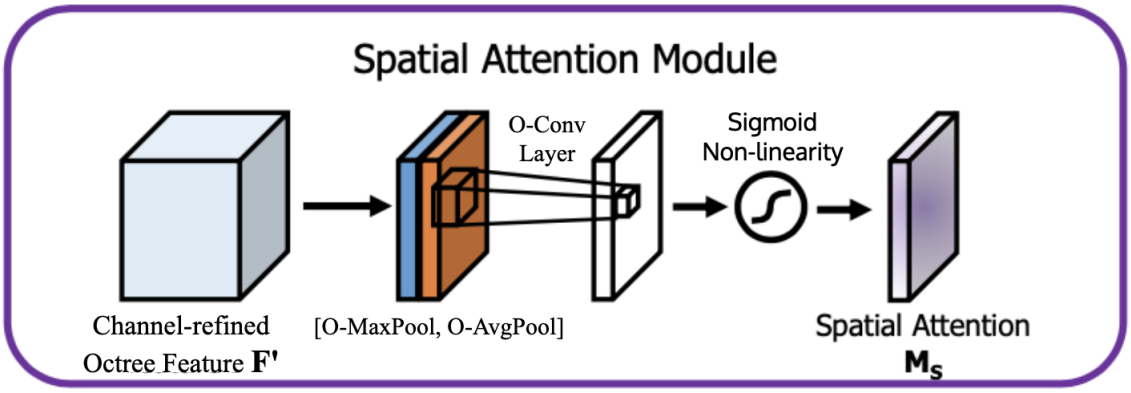


Fig. 3.21: Spatial Attention Sub-Module Architecture

Channel Attention Submodule. Both these feature octrees are stacked together and 3D octree based convolution is performed followed by passing the output through a sigmoid non-linearity to restrict the range of values to $[0, 1]$. In this work, we use consider $3 \times 3 \times 3$ kernels.

■ *Relative Placement of Sub-Modules* –

There exist many configurations for the placement of each attention sub-module to form the complete module. Based on experiments, it was shown in [15] that a *series configuration with channel attention sub-module preceeding the spatial attention sub-module* gives the best results. So, in this work, this configuration is considered as shown in Fig. 3.18.

High Resolution

Some of the ideas for designing *0-AHRNet* have been taken from [17]. The main ideas and intuitions behind this type of architecture are as follows (See Fig. 3.17):

→ Repeated Multi-depth Fusions are performed to improve quality of hidden representations.

→ Maintaining high-resolution representations is important.

→ Along depth axis, feature map size remains same as shown in Fig. 3.22.

→ Along scale axis, typical feature map size reduction happens as in any typical CNN.

The main features of Model-3 *O-AHRNet* are as follows:

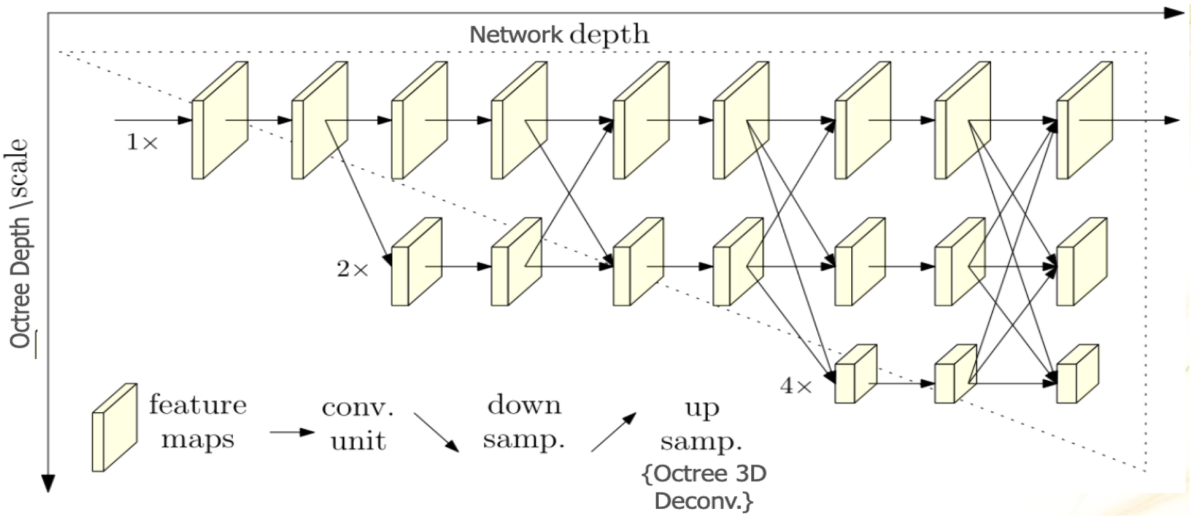


Fig. 3.22: Intuitive visualization of Multi-scale Fusions. Adapted from [17].

- The input octree to the model has 7 channels $(n_x, n_y, n_z, d, r, g, b)$ as described in Section 3.1.1.

- As shown in Fig. 3.23, the whole architecture can be divided into multiple branches and stages for the ease of understanding.

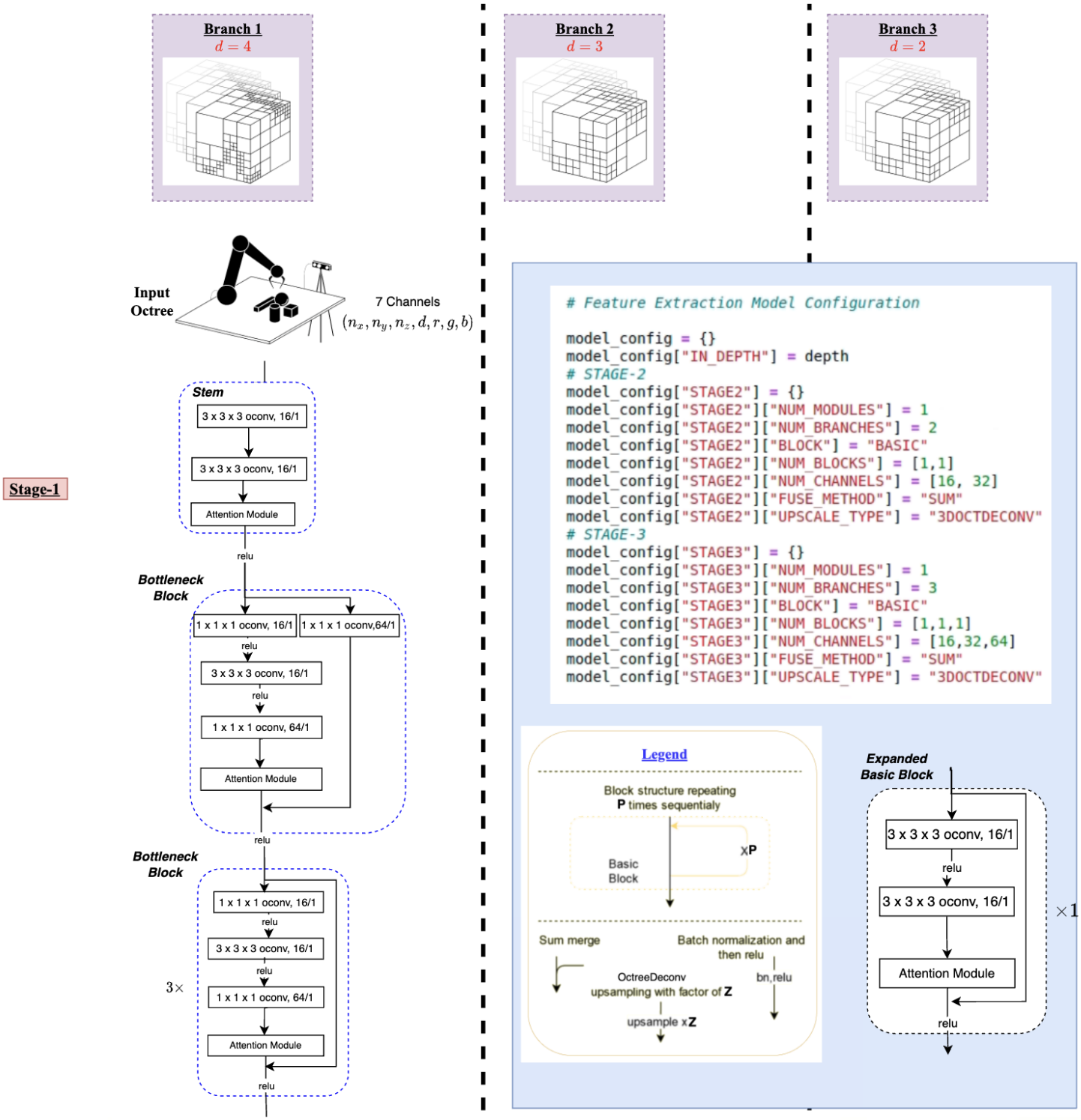
- Each branch processes octree features of a particular depth. For example, *Branch-1* processes octree features of $d = 4$ (See Fig. 3.23).

- In this model, for 3D *O-Conv* a kernel of size $3 \times 3 \times 3$ is utilized.

- To enable multi-depth fusion of features, *upsampling* and *downsampling* (in terms of octree depth) is necessary. For the purpose of downsampling Strided 3D *O-Conv* operations are utilized, whereas for *upsampling* Transposed 3D *O-Conv* operations (3D Octree Deconvolutions) are used.

After voxelisation, the remaining step-through of different stages of the model remains

the same as Model-1 *Vanilla O-CNN* as described in Section 3.5.2.1.



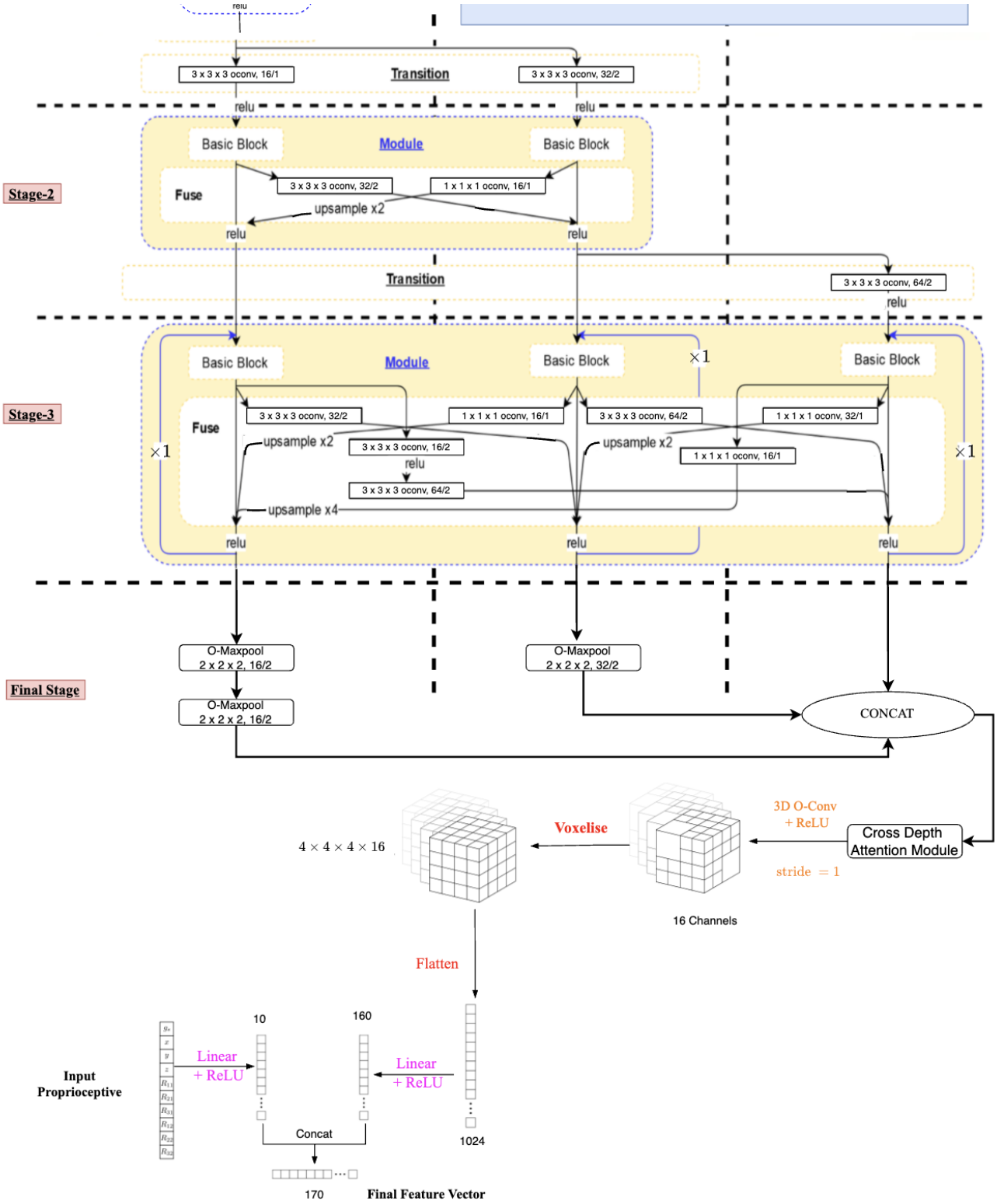


Fig. 3.23: Detailed architecture of the *O-AHRNet*. In the previous page, the model configuration (different elements of the *O-AHRNet*) is listed on the right. The legend that is useful for reading the figure and understanding the operations involved is provided in this page at the end of the figure. Note that, this is the architecture for the observation stack at one timestep. The feature extractor is duplicated for each stack in order to process two historical observations in addition to the current one.

3.5.3 Actor-Critic Heads

In this section, more details regarding the DL models of Critic and Actor Heads are specified.

3.5.3.1 Vanilla Architectures

The vanilla baseline architectures for the actor and critic networks are shown in Fig. 3.24. An identical fully connected neural network architecture with different set of parameters is used for the actor and critic.

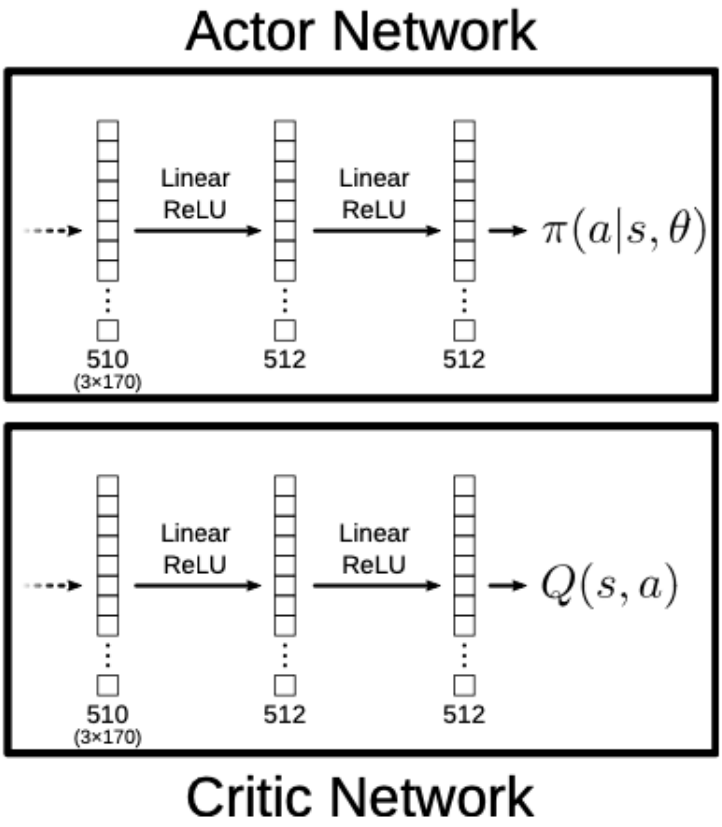


Fig. 3.24: Architectures of Vanilla Actor and Critic Networks. Note that, for output of the actor network for TQC algorithm is interpreted as mean and std. of the action distribution. The output of the critic network is interpreted as the distributional representation of the action-values (with finite support) predicted by multiple critics.

It should be noted that the input to both actor and crtic networks is the resultant feature vector obtained after concatenating the *Final Feature Vector*s from the three copies of Feature Extractors {FE#1, FE#2, FE#3} processing observations from timesteps respectively $\{t, t-1, t-2\}$ (Refer Fig. 3.14).

3.5.3.2 Innovations in Architectures of Heads (Digression)

Most of the DRL research has been focussed into algorithmic innovations, where DL is generally treated as a blackbox function approximator. Some ideas that were introduced to break this trend are *Spectral Normalization* (inspired from GANs) and *Duelling Networks*. In this section, the focus is on innovating the architectures of actor and critic networks (also called heads).

■ *Spectral Normalization* –

In an entropy regularized setting (such as the one used in TQC), critic is trained by minimizing the soft bellman residual :

$$\min_{\psi} \mathbb{E}\left(Q_{\psi}\left(\mathbf{s}_t, \mathbf{a}\right) - \left[r_t + \gamma \mathbb{E}\left[\hat{Q}\left(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}\right) + \alpha \mathcal{H}(\pi)\right]\right]\right)^2 \tag{3.11}$$

For getting a gradient step for actor, derivatives are passed through the critic $\nabla_\theta = \frac{\partial Q_\psi(\mathbf{a}_\theta, \mathbf{s})}{\partial \theta}$. Some experiments conducted in [18] revealed that on use of modern head architectures, the suspected problem of overfitting was not the cause of poor performance but exploding gradients can be a possible issue. It was hypothesized that taking gradients through the critic can cause unstable behaviour, similar taking the gradient through the discriminator network in GANs.

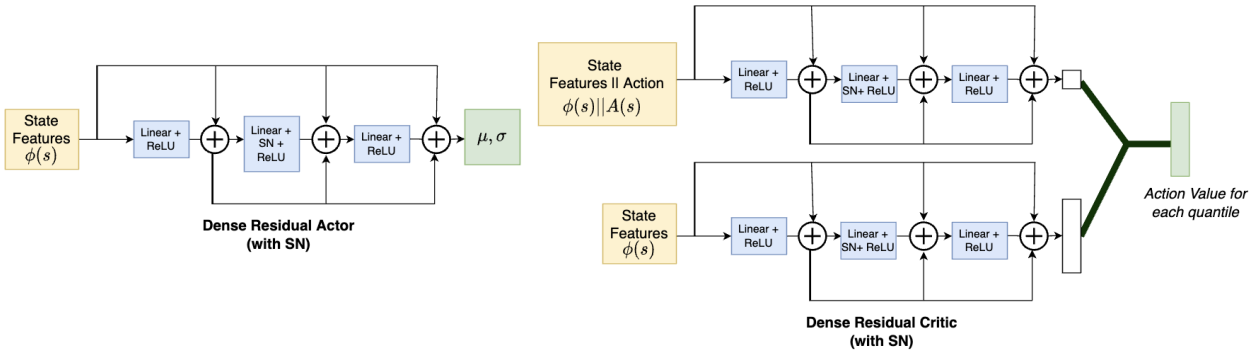


Fig. 3.25: Architectures of Actor and Critic Networks after incorporating some ideas such as dense connections [19], spectral norm. [18] and duelling critic [20]. It has been used for some initial experiments that will be shown in the Chapter 5.

We know that a function is called L-Lipschitz smooth when its gradient is limited by L. So, the idea of spectral normalization [18] is that if we can make sure that critic function is

L-Lipschitz smooth, then gradients $\frac{\partial Q_\psi}{\partial a_q}$ being propagated into the actor are bounded.

In GAN literature, spectral norm applied in the following way, ensures that all layers have operator norm of 1 :

$$y = \frac{Wx}{\sigma_{\max}(W)} \approx \frac{Wx}{p^T W q} \tag{3.12}$$

For this to work, p and q need to be the Eigen vectors that correspond to the biggest eigen value (since we are trying to estimate $\sigma_{\max}$), which is made sure by using Power Iteration method.

■ *Duelling Networks* –

Dueling network (originally introduced for discrete action spaces) represents two separate estimators for the critic: a state-dependent action advantage function estimator and a state-independent value function estimator[21]. Intuition behind Duelling Networks is that may learn which states are useful without having to understand the effects of each action on each state.

Module that combines two streams of duelling network is given by (carefully designed) :

$$Q(a, s; \alpha, \theta, \beta) = V(s; \theta, \beta) + \left(A(a, s; \alpha, \theta) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(a', s; \alpha, \theta) \right) \tag{3.13}$$

This idea has been adapted to continuous action spaces in [20] by discretizing the action space can be discretised into intervals and decoupling Q value as action interval advantages and action independent state values.

3.6 Chapter Summary

In this chapter, details about developing effective DRL algorithms for *Task I - Grasping Various Objects in Diverse Objects* which involved training a robotic arm agent to grasp unseen objects in novel environments, were provided. A complete end-to-end DRL pipeline was demonstrated, where a robotic arm was deployed in various random simulation scenes to learn a robust policy for grasping novel objects. The design and development of a series of octree-based convolutional neural network models for effective feature extraction in an Actor-Critic setting were discussed. The model development was focused on demonstrating

the need for developing and incorporating complex DL models in DRL algorithms. The developed models were incorporated into the DRL pipeline. Results for various experiments conducted as a part of this first task will be analyzed in Chapter 5. In this task, only static but novel objects of interest were considered. Chapter 4 explores the other side of the coin, where dynamic but known objects of interest are considered whose motion trajectories are unknown apriori.

Chapter 4

Task II - Dynamic Grasping of Moving Objects

This chapter explains various components involved in realizing the second task dealing with grasping and picking up known but dynamic (moving) objects whose motion path is not known apriori. Deep Learning (DL) and Inverse Kinematics (IK) techniques are mainly used in this work. Details regarding the task description, design of the entire approach, implementation details and architectures of the various DL models utilized are illustrated in this chapter. Ideas and setup for this task design were taken from [3]. The results of various experiments conducted as a part of this task will be presented in a forthcoming chapter.

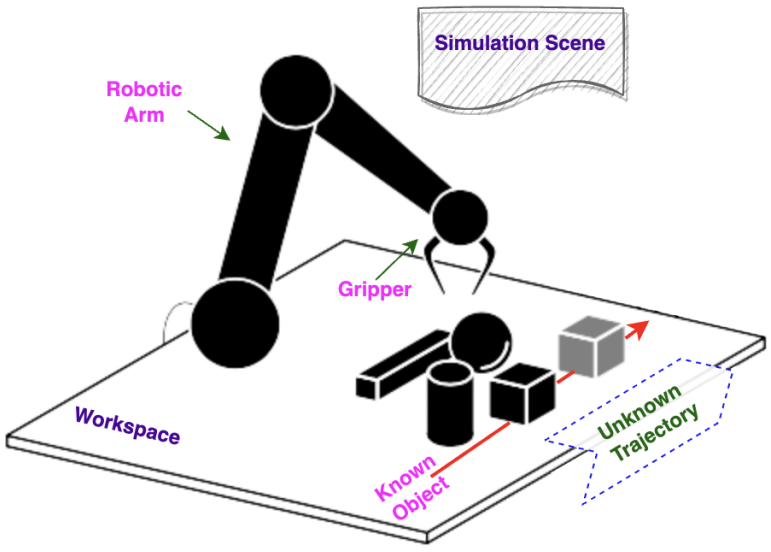


Fig. 4.1: Illustration of basic problem setup of Task-2. Note the 3D model of the target object is known apriori but not its motion trajectory. Here, the moving target object is the "cube".

4.1 Task Description

The second task considered is *Dynamic Grasping of Moving Objects*. The definition of the problem can be characterized using the following pointers:

- ⋆ The target object which is moving w.r.t robot's world frame is *known*, i.e. existence and apriori access to its 3D model is assumed.
- ⋆ The motion trajectory which is being followed by the target object is unknown.
- ⋆ The aim of the robotic arm is to first *reach*, then *grasp* and finally *lift* the dynamic target in a stable manner.
- ⋆ Grasping the dynamic target is considered to be a success if the robotic arm picks up the object of interest *quickly* but does not overturn any obstacles present as a result of a collision in the process.

4.2 A Note on Inverse Kinematics (IK)

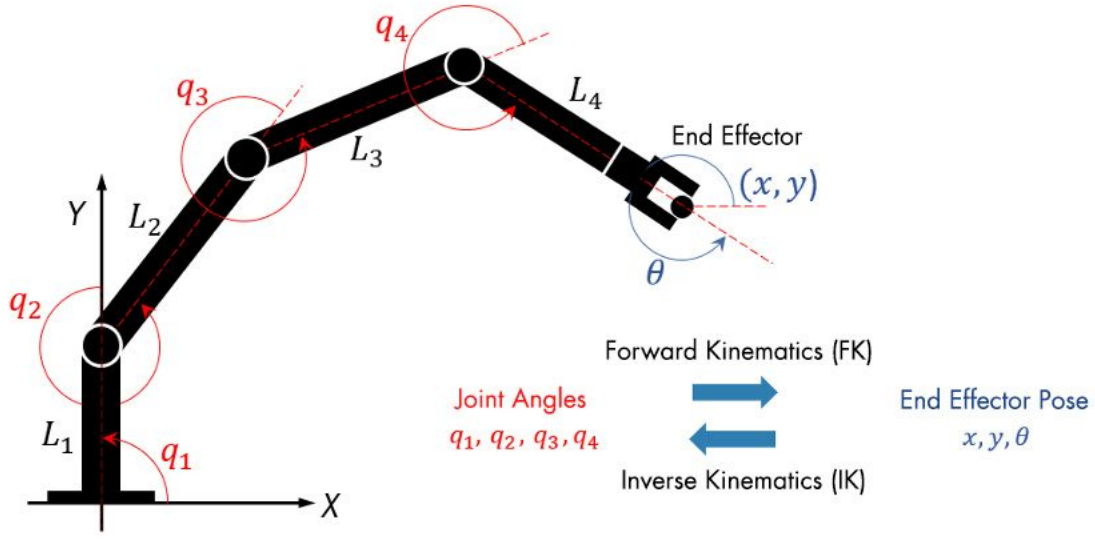


Fig. 4.2: Illustration of Forward and Inverse Kinematics in a articulated robotic arm with multiple joint links. Note that, here L_i's denote the robot's links and q_j's denote the joint angles.

We know that Forward Kinematics (FK) deals with determining the pose of the end effector, given the joint angles of different joints present in the robotic arm. Inverse

Kinematics (IK) tells us the joint angle configuration required so that the end effector can reach a particular pose of interest to us for manipulation (See Fig. 4.2).

There are two broad classes of IK methods of relevance, used popularly for motion planning :

1. *Sampling based Methods* –

    - This class of methods make use of random sampling in a high dimensional joint configuration space and constructing tree structures, to plan trajectories.
    - Examples are Rapidly-exploring Random Trees (RRT) [22] and Probabilistic RoadMaps (PRM) [23].

2. *Optimization based Methods* –

    - This type of algorithms start with an initial trajectory and then perform mathematical optimization in trajectory space.
    - Examples are Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [24] and Stochastic Trajectory Optimization for Motion Planning (STOMP) [25].

4.3 Integral Components of the Approach : An Overview

In this section, a brief overview of various components [3] that are involved in the design of the approach is provided.

4.3.1 Object Pose Retrieval

Since we are considering a simulated setup and the main focus of the problem is not estimation of object pose, we retrieve or get the object pose using the simulation software w.r.t. world coordinate system. Note that, in a real-world setup, this pose retrieval part can be replaced with DL-based pose estimation models that RGB or RGB-D data of the scene's state as input and estimate the 6D pose consisting of the position and orientation of the target object. In that case, a perception sensor such as an RGB-D camera can be a part of the robotic setup. One popular method that can be used for estimating pose is DOPE [2].

4.3.2 Object Pose Prediction

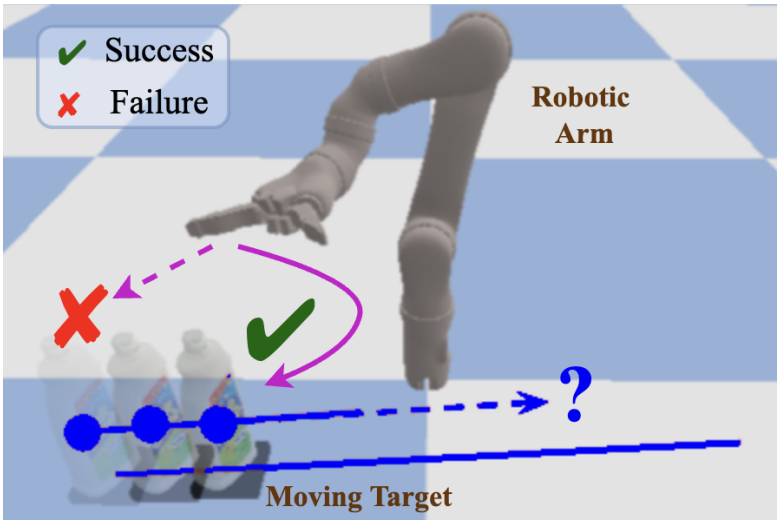


Fig. 4.3: Illustration demonstrating the need for pose prediction. A *Kinova Mico* robotic arm and a known moving target (*Bleach Cleanser*) are shown in Pybullet Simulation environment.

The problem statement suggests that target object of interest would probably be dynamic and therefore planning motion trajectories for the end effector at the retrieved pose of the target would be wasteful. This is because when the grasp would be executed the object would not be in the same pose due to its motion. This ultimately leads to failure of the grasp. Hence, object pose prediction becomes necessary in such environments (See Fig. 4.3). Different techniques explored to ensure effective estimation of target object's future pose are described in Section 4.7.

4.3.3 Grasp Database

Since the 3D model of the target object is known apriori, a grasp database can be created by pre-computing the various stable grasps in simulation. The grasps are basically obtained by putting the target at different random poses in the workspace. The procedure employed [3] for getting final condensed grasp database for each object is as follows :

- First, without considering the stability of grasp, 5000 candidates of grasp are considered.
- Then, a small random noise is added to pose of the object at which the grasp happened.
- In the new displaced pose, the same grasp is carried out. The outcome of this grasp is stored as either a success or failure.

- This experiment is repeated multiple times for each grasp. The rate of success of this experimentation is an indicator of the reliability of the grasp.

- From the 5000 candidates, the top 100 grasps (for each object) are selected and stored as the database, G_{DB}, to be used in the dynamic grasping pipeline.

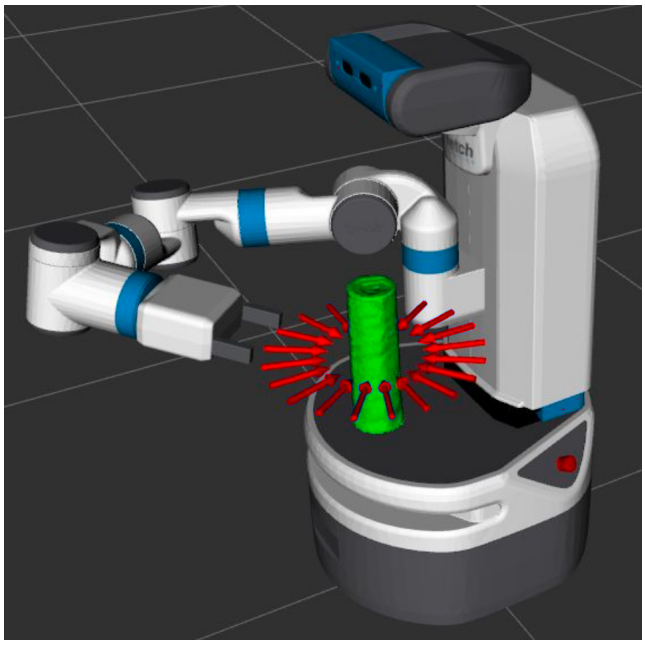


Fig. 4.4: Pictorial Illustration of robust grasps from Grasp Database (G_{DB}). The simulation scene consists of a *Fetch* robotic arm and a cylinder object. The red arrows indicate some of the stable grasps directions for picking up the object. Adapted from [26].

Note that, here a trade off exists between the number of grasps in the database (accounting for diversity of object approach) and the time to iterate through all grasps.

4.3.4 Grasp Ranking Functions

In this subsection, details about various ranking functions, utilized to filter the grasps online from the offline computed grasp database (See Section 4.3.3) and get a final grasp trajectory to be executed *live*[*] in the dynamic grasping pipeline, are discussed.

[*]When a test trial is running

4.3.4.1 Reachability Awareness

Intuition

One of the most important indicators of a correct and feasible grasp is *reachability*. It is important for the robotic arm to be reachability aware because:

- A reachability unaware motion planner will return some unrealistic and impossible grasp candidates. This will lead to waste of computation because of motion trajectory replanning.
- For dynamic objects, reachability can serve as an indicator of manipulability i.e., it is highly likely that most reachable grasp will remain attainable in the future as well.

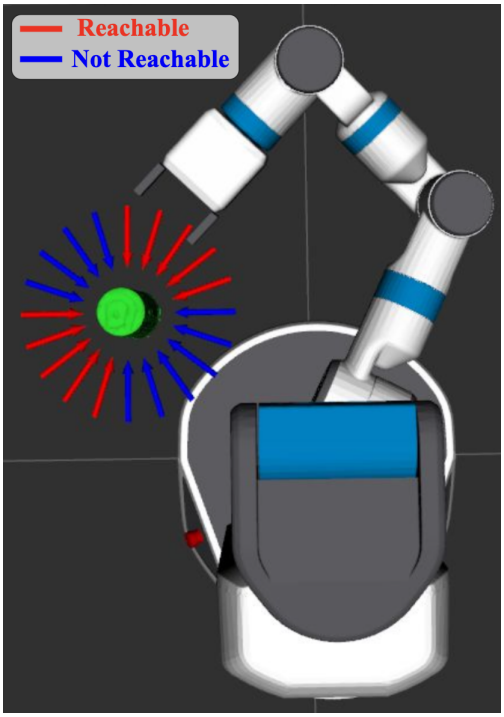


Fig. 4.5: Image depicting the need for a notion of reachability. It should be noted that, all robust grasps need not reachable as shown.

One naive solution for incorporating reachability awareness is that if we have a grasp database (such as G_{DB} in Section 4.3.3) for a known target, then we can compute IK for all of them and check if they are reachable. But this process is time-consuming and compute-intensive; hence, not suitable for dynamic settings.

The following are some of the favourable traits of a reachability awareness system:

→ Predetermined (offline) space of reachability, unique for every robotic arm.

→ This space can be used to rate the grasp without computing IK.

→ Specifically, for a dynamic setting, it is beneficial to have a "smooth" Signed Distance Field (SDF) than binary reachability values in the field. This is because the value should indicate not only reachability now but after some time steps when the object has moved around.

→ A gradient to indicate the direction from inaccessible areas to accessible areas of the workplace, can be informative.

Reachability Space Representation

A grasp is reachable when a motion path can be devised to take the arm from its present configuration to a target configuration that puts the arm at the intended grasp location. Therefore, by this original definition, reachability space is binary in nature (For example, { Reachable (1), Not Reachable (0) }).

In order to obtain the features of an ideal system described earlier, a Signed Distance Field (SDF) $d_{sdf} = SDF(\text{ pose })$ can be defined [26]. By taking into account the grid resolution and distance from interface:

$$d_{sdf} = \pm\sqrt{\left\|\frac{\Delta xyz}{res_{lin}}\right\|^2 + r\left\|\frac{\Delta rpy}{res_{rot}}\right\|^2} \tag{4.1}$$

where:

$$\begin{aligned}
\Delta xyz(\text{ in cm}) &\rightarrow \text{translational distance} \\
res_{lin}(\text{ in cm}) &\rightarrow \text{translational resolution} \\
\Delta rpy(\text{ in rad}) &\rightarrow \text{rotational distance} \\
res_{rot}(\text{ in rad}) &\rightarrow \text{rotational resolution} \\
r &\rightarrow \text{relative metric ratio} \implies res_{lin}\ \text{cm} \equiv r * res_{rot}\text{rad}
\end{aligned}$$

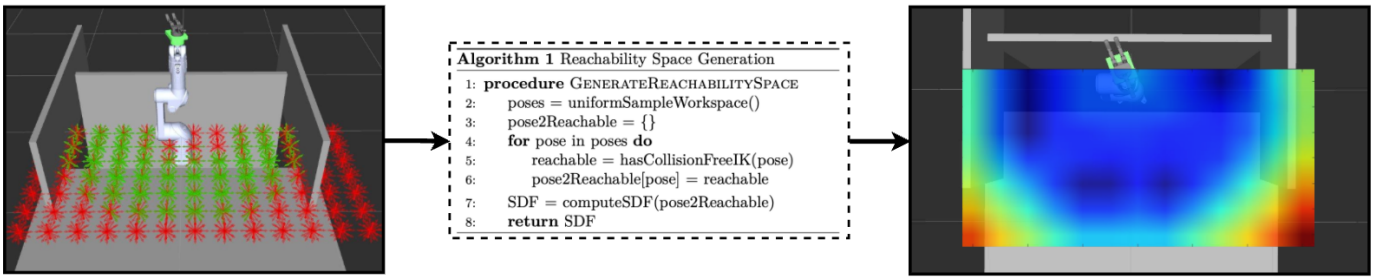


Fig. 4.6: Conversion of Binary Reachability Notion to smooth SDF representation. Note that, in the left figure green arrows indicate reachable and red arrows indicate unreachable grasps. Adapted from [26].

This SDF can be interpreted as reachable grasps lead to positive d_{sdf} value and unreachable grasps will have a negative d_{sdf} value.

4.3.4.2 Motion Awareness

Another important ranking function is that of *Motion Awareness*. This function basically maps each grasp to a value denoting the feasibility of the grasp based on the way in which the object of interest is moving.

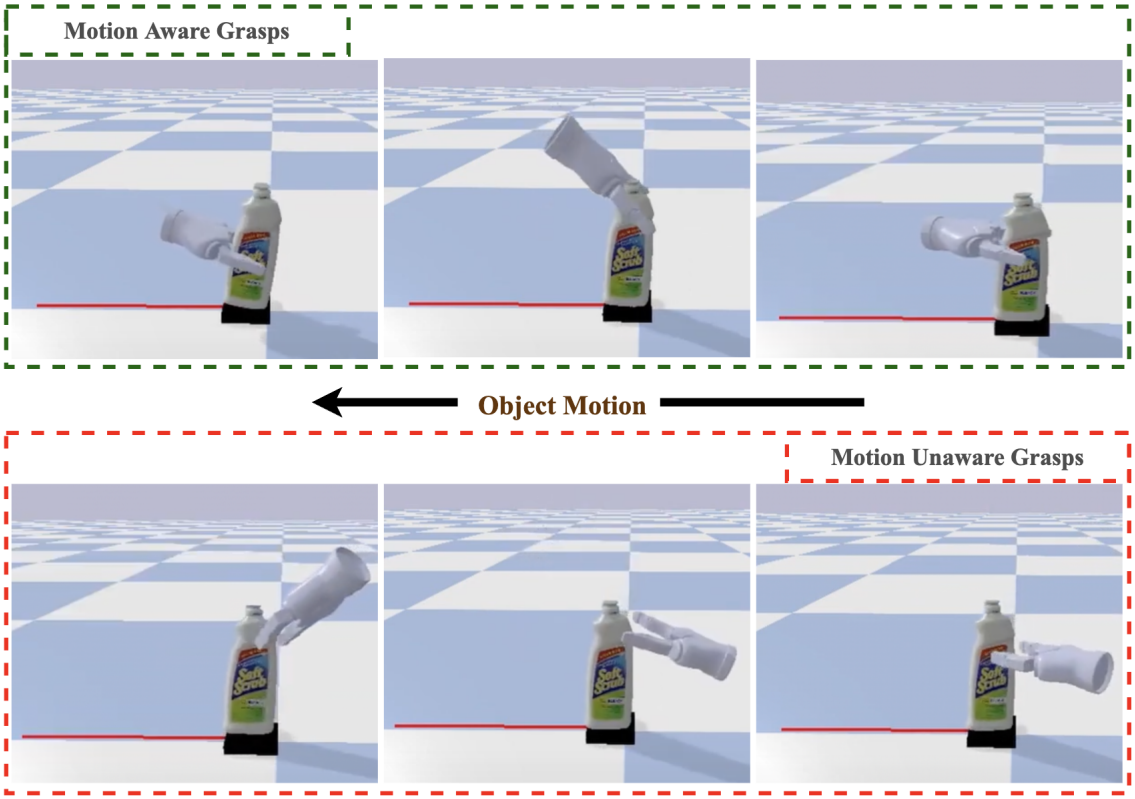


Fig. 4.7: Motivation for considering type of motion as a discriminating function for planning grasps. From the figure, it is clear that motion aware grasps have a higher probability of succeeding compared to its counterpart, even though both of them maybe stable.

A DL based Neural Network model was used for getting motion quality score of grasps. The complete architecture of the model is provided in Fig. 4.8. A dataset for supervised training of this network was generated [3] and then the training was carried out. Note that, this NN function is used to filter grasps based on the attribute of motion. More details about this would be given in Section 4.4.

4.3.4.3 Integration of Ranking Functions

Now, note that both the ranking functions are used for the filter of grasps in the Grasp Database. There are many ways to combine both of them for use. After experimentation,

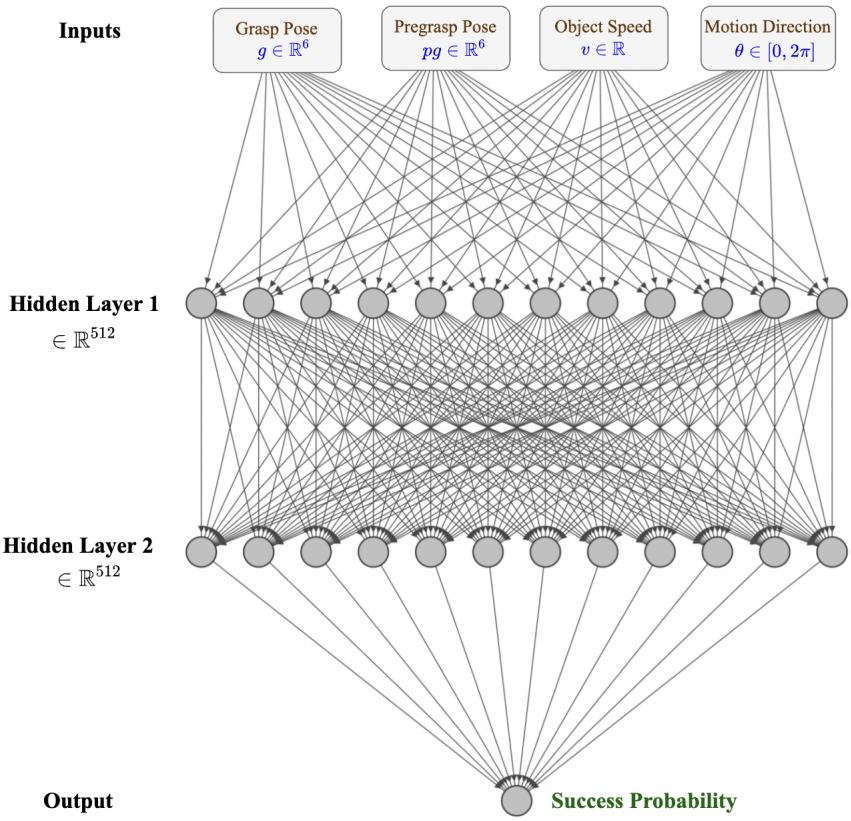


Fig. 4.8: Architecture of the Motion Aware Network used [3]. Note that, the output is a probability indicating how successful a given grasp can be depending on the object motion aware quality.

[3] found that including five of the top grasps based on both qualities works the best, grasping success result-wise.

4.3.5 Adaptive Trajectory Synthesis

In Section 4.2, different class of IK methods were discussed. It should be noted both these methods are not directly suitable for handling dynamic environments, where the target object is moving in an unknown trajectory. This is because of the following reasons:

(i) Optimization based algorithms such as Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [24] and Stochastic Trajectory Optimization for Motion Planning (STOMP) [25] can be expensive in time and since dynamic settings require fast motion replanning, these methods do not really scale here.

(ii) Sampling based methods such as Rapidly-exploring Random Trees (RRT) [22] and Probabilistic RoadMaps (PRM) [23] in the vanilla form are faster but they can generate

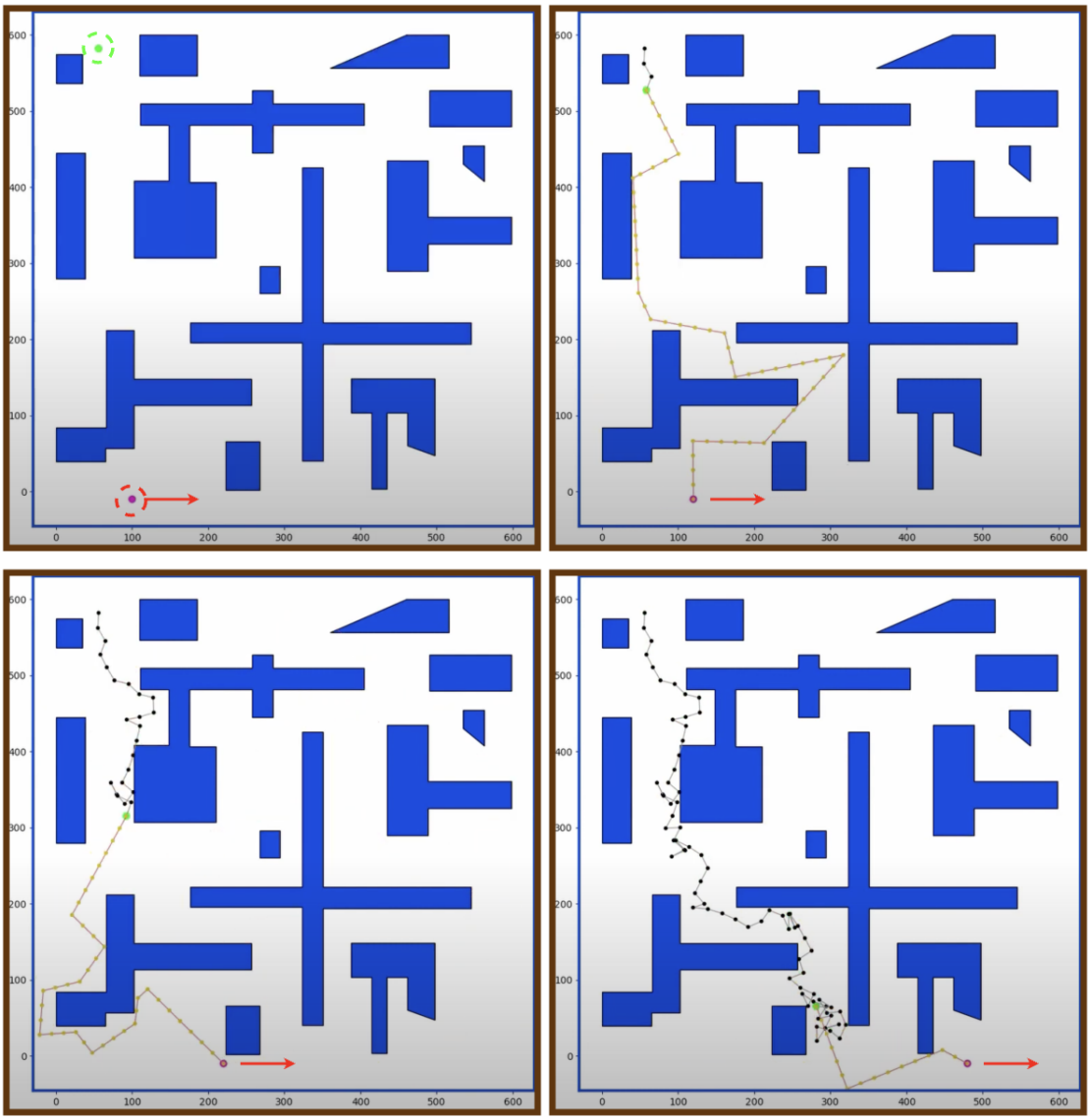


Fig. 4.9: A toy example for demonstrating the problem with vanilla RRT and dynamic targets. Here, the goal of the green dot is to catch the moving red one (both circled in the top left sub-image). RRT is used as the planning algorithm to generate motion plans. We can see that the motion trajectory being followed by the green dot is not at all smooth and very random.

random trajectories that may be significantly different from the previous solution, owing to the motion of the target. Such random motion plans at every time instant, ultimately leads to the robotic arm having wavy motion, which can be dangerous and prone to collisions. This is demonstrated using a toy example in Fig. 4.9.

In order to enable fast motion replanning and avoid jerky motions of the robotic arm, we can use a trajectory seeding approach [3], where the motion planning algorithm (sampling

based method such as RRT) incorporates the solution from the previous timestep as a initial point for generating current trajectory. Using this technique has the following advantages:

- Smoother transition between plans.
- Temporal consistency in generated trajectories at different timesteps.
- Speed up in computation.

4.4 Algorithmic View of the Approach

Some details about a few functions involved in the algorithm [3] are provided:

- *RetrieveGraspDatabase(.)* loads the object specific database of robust grasps precomputed (Refer to Section 4.3.3).
- *RetrieveObjectPose(.)* is used to get the pose of the dynamic target at the current instant (Refer to Section 4.3.1).
- *PredictObjectPose(.)* is used to forecast the pose of the dynamic target at a future instant (Refer to Section 4.3.2).
- *ConvertGrasps(.)* basically does a co-ordinate frame transformation from the frame of target to the robotic arm's frame.
- *FilterGrasps(.)* reduces the size of G_{DB} by performing the filtering operation and selecting grasps according to the motion aware and reachability aware ranking functions (Refer to Section 4.3.4).
- *ChooseGrasp(.)* finds the grasp that is nearest to the present configuration of the arm from G_F.
- *GetEuclideanDistance(.)* calculates the differential euclidean distance between the present gripper location and the chosen grasp pose.
- *GetQuaternionDistance(.)* calculates the differential quaternion distance between the present gripper location and the chosen grasp pose.
- *CheckGraspSuccess(.)* confirms whether the executed *dynamic grasp* was a success or failure.

Algorithm 1: Dynamic Grasping Algorithm

Function *DynamicGrasp(O)*
- $G_{DB} \leftarrow RetrieveGraspDatabase(O)$;
- **while** *True* **do**
  - $p_c \leftarrow RetrieveObjectPose(O)$;
  - $t_{pred} \leftarrow CalcPredTime(d) p_f \leftarrow PredictObjectPose(O, t_{pred})$;
  - $G_W \leftarrow ConvertGrasps(G_{DB}, p_f)$;
  - $G_F \leftarrow FilterGrasps(G_W, p_f)$; `/* Using Ranking Functions */`
  - $g_c \leftarrow ChooseGrasp(G_F)$
  - If this chosen grasp is unreachable, continue to next iteration;
  - Move arm to g_c;
  - $d_p \leftarrow GetEuclideanDistance()$;
  - $d_q \leftarrow GetQuaternionDistance()$;
  - **if** *ObjectGrasp*(d_p,d_q,b) **then**
    - $p'_c \leftarrow RetrieveObjectPose(O)$;
    - $t_pred' \leftarrow 1s$
    - $p'_f \leftarrow PredictObjectPose(O, t'_{pred})$;
    - $g_c \leftarrow ConvertGrasps(G_{DB}, p'_f)$;
    - Move arm to g_c;
    - Close hand while moving with the target for t'';
    - Break Loop;
  - **end**
- **end**
- **return** CheckGraspSuccess();

end

Function *ObjectGrasp*(d_p,d_q,b)
- `/* b → backoff distance */`
- **if** $d_p \leq 1.1b$ ***and*** $d_q \leq 20°$ **then**
  - **return** True;
- **end**

end

Function *CalcPredTime(d)*
- **if** $0.1\ m < d \leq 0.3\ m$ **then**
  - $t_{pred} \leftarrow 1s$;
- **else if** $d \leq 0.1\ m$ **then**
  - $t_{pred} \leftarrow 0s$;
- **else**
  - $t_{pred} \leftarrow 2s$;
- **end**
- **return** t_{pred};

end

4.5 Pictorial Demonstration

Fig. 4.10: Pictorial Demonstration of Task-2. The keyframes corresponding to (A), (B) and (C) in each set of images are various steps of the task are shown in a chronological manner, in Pybullet simulation environment with a UR5 arm. Notice that the robotic arm first *reaches* a dynamic object, makes contact with it in a pregrasp, then *grasps* the object before lifting it into the air. This marks the end of a successful trial. Note that the object of interest is circled in the first frame of every motion sequence.

4.6 Implementation Details

In this section, some aspects related to the implementation of the *dynamic grasping* setup is discussed. Note that more details regarding the Configuration will be provided in Chapter 5, Section 5.2.1.

Simulation Environment

For all experiments and trials, the *Pybullet*[†] simulation environment was utilized. It focuses on sim-to-real transfer and is a simple and straightforward Python tool for deep learning and simulating robots. Some of its prominent features are:

- Allows loading from URDF, SDF, MJCF, and more file formats.
- It offers simulations of forward dynamics, computation of inverse dynamics, forward and inverse kinematics, collision detection, and ray intersection queries.
- In addition to physics simulation, bindings exist for rendering, including a CPU renderer (TinyRenderer) and OpenGL 3.x rendering.
- PyBullet is effortlessly compatible with TensorFlow and OpenAI Gym.

Middleware Software

ROS Melodic is used as middleware for this work which handles communication between primary nodes and processes requests from motion planner and the simulation environment itself.

Robotic Arm

In this work, UR5 arm with *Robotique* gripper is utilized.

[†] https://pybullet.org

Objects

For training and testing the robotic arm's performance inside the simulation environment, the following objects are taken from the YCB model set, provided as a part of the YCB benchmark dataset [27][28]. Note that, the 3D models of these objects are available apriori.

Fig. 4.11: Stills of various YCB objects used in Task-2 on dynamic grasping.

Grasp Collection

For collecting various robust grasps to store in the Grasp Database, the *GraspIt!*[‡] software is utilized. It is a simulator for grasping research that may accept various hand and robot designs. Some of its main features are as follows:

- 3D user interface that allows the user to observe and interact with a virtual environment filled with robots, objects, and obstacles.
- Provides support for grasp planning, numerical grasp quality metrics calculation and visualisation approaches for the Grasp Wrench Space.
- It consists of a dynamics engine and allows interaction with hardware and sensors.

Motion Planning

For trajectory generation, solving kinematics and motion planning MoveIt[§] framework is utilized. Details about this were provided in Chapter 3 Section 3.4.

[‡] https://graspit-simulator.github.io
[§] https://moveit.ros.org

Types of Object Motion

Here, we discuss various types of object motion that are used various experiments of this task. More details about the performance will be provided in Chapter 5 Section 5.2. The following types of motion were considered in simulation environment [3] using conveyor belts:

1. *Linear Motion* – The dynamic target advances at the same velocity along a straight line.
2. *Linear Motion with Obstacles* – Similar to linear motion, but with added complexity due to the presence of background clutter objects.
3. *Linear Motion with Top Slab* – Here, the level of challenge is increased in linear motion by keeping a flat top slab directly above the conveyor belt.
4. *Circular Motion* – The dynamic target executed uniform circular motion along a non-linear circularly shaped conveyor belt.
5. *Sinusoidal Motion* – This is a more difficult version of nonlinear motion in which the target goes along a sinusoidal path (superimposed on linear motion).

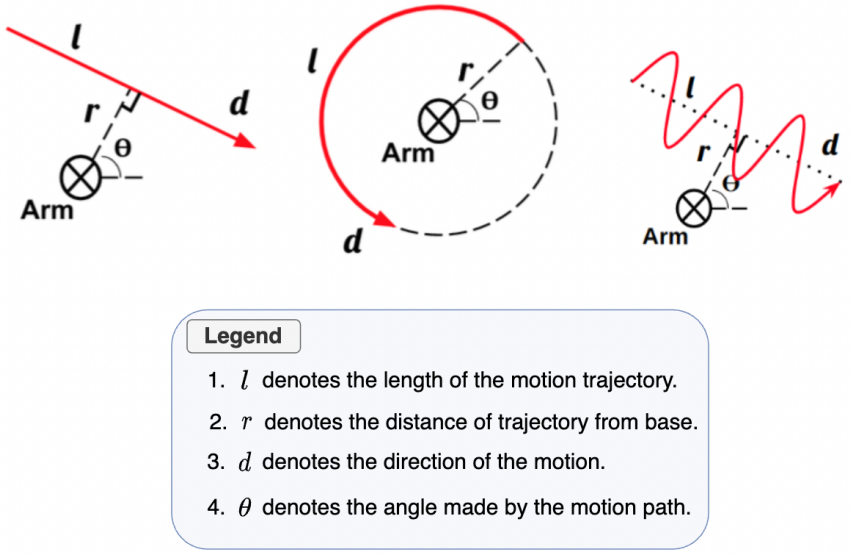


Fig. 4.12: Illustration of different types of motion types. Note that, parameters that can be randomized are displayed in the legend.

It should be noted that since we are considering a setup where the motion of the object is not known apriori, there are a necessity to randomize various parameters of the motion

trajectory. This is because exposing the model to different kinds of trajectories during training and help in its generalization aspect. The various aspects that are considered for randomization are shown in Fig. 4.12.

4.7 Object Pose Prediction : Approach Design

In this section, we discuss about various techniques utilized for forecasting the future pose of the object of interest.

4.7.1 Need for Pose Prediction

Consider the following arguments:

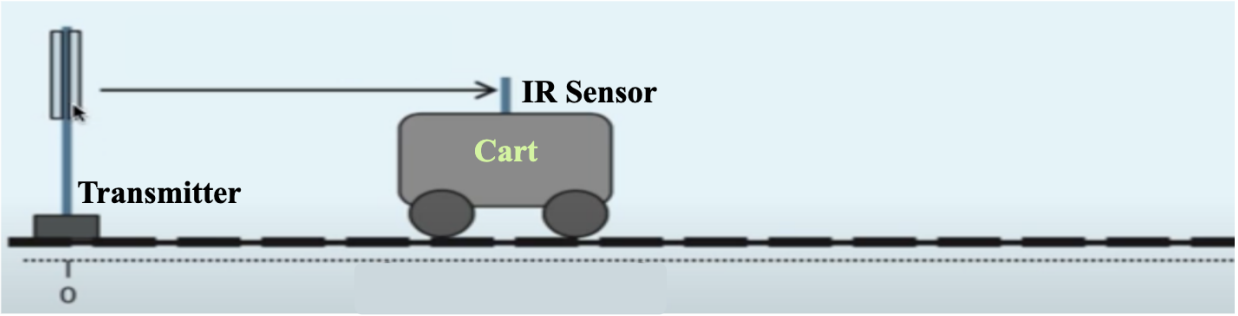
- Motion planning consumes time and computational resources.
- As an item moves in a dynamic grasping situation, the calculated motion plan might soon become outdated.

Hence, since the type of motion of the target is unknown in the task that is being considered, it is needed to be able to forecast the location of the target in future.

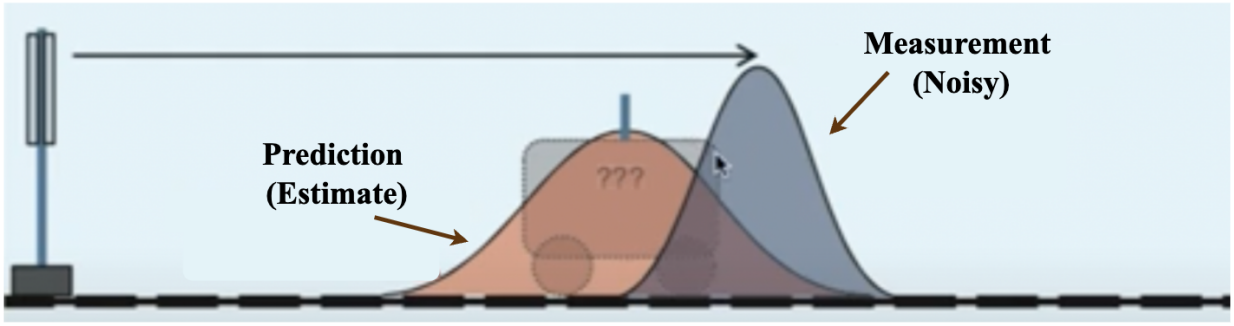
4.7.2 Method-1 : Kalman Filter

4.7.2.1 Main Idea

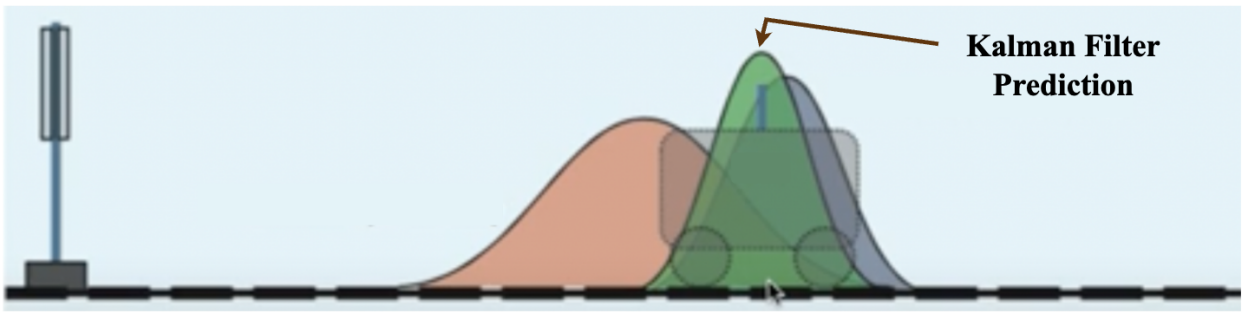
The usage of Kalman Filters is a common and conventional technique. The Kalman Filter forecasts the upcoming state of the system based on prior assessments. It is often used in target tracking (radar), location and navigation systems, control systems etc.



(a) Cart with an IR sensor. Distance from transmitter needs to estimated correctly.



(b) Prediction is made using equations of motion and measurement is done by using IR sensing arrangement.



(c) Kalman Filter makes use of the two available sources of information and makes the forecast of the final position.

Fig. 4.13: Toy Example describing how Kalman Filter can be useful for prediction of motion.

4.7.2.2 Operation

The main steps of involved in the main operation of a Kalman Filter are as follows [29] :

1. *Time Update (Predict) Step* : Dynamical Model and equations of motion from physics are used to make the forecast. Note that, the uncertainities are taken into account as Process Noise.

2. *Measurement (Update) Step* : This step basically indicates the updation in the prediction made after taking into account the sensor measurements. Note that, here Sensor Noise is considered.

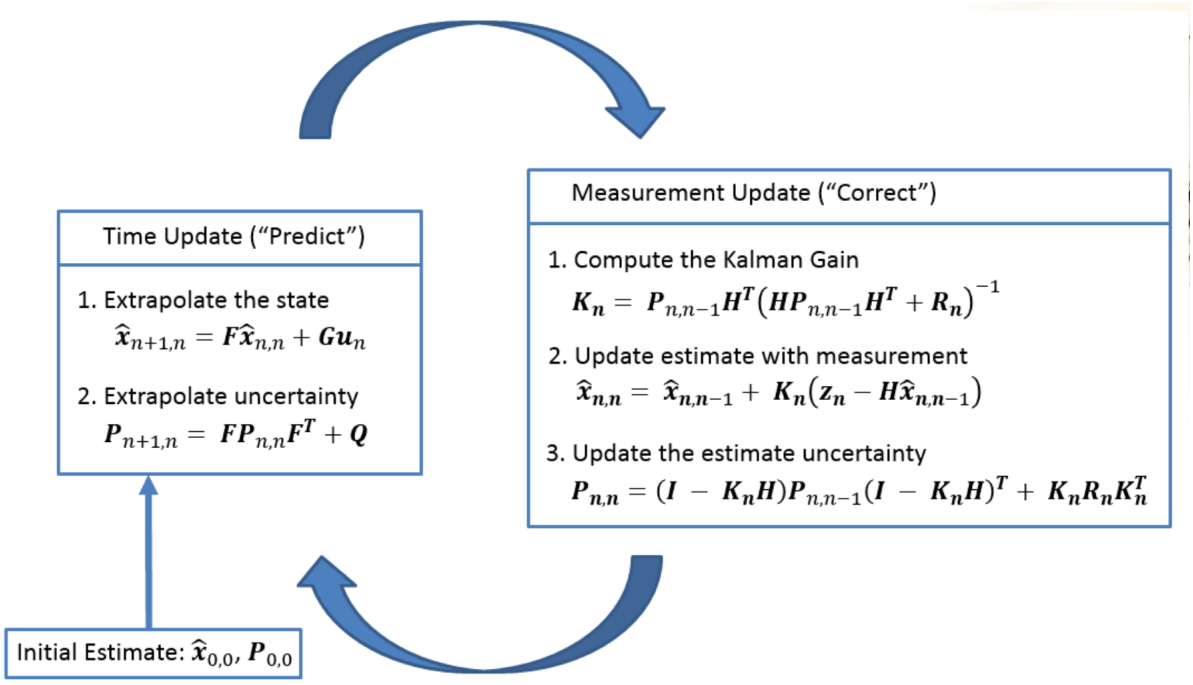


Fig. 4.14: An illustration of whole filter operation with important equations under each of the two steps [29].

4.7.2.3 State Extrapolation

The general form of the state extrapolation equation is :

$$\hat{\boldsymbol{x}}_{n+1,n} = F\hat{\boldsymbol{x}}_{n,n} + Gu_n + w_n \tag{4.2}$$

In this current case of object pose prediction, we consider:

- No control inputs $u_n = 0$
- Constant Acceleration Dynamical Model

Therefore, the equation becomes:

$$\hat{\boldsymbol{x}}_{n+1,n} = F\hat{\boldsymbol{x}}_{n,n} + w_n \tag{4.3}$$

The constant acceleration dynamical model can be represented as (9D State vectors) :

$$\hat{\boldsymbol{x}}_{n+1,n} = F\hat{\boldsymbol{x}}_{n,n} + w_n$$

$$\begin{bmatrix} \hat{x}_{n+1,n} \\ \hat{y}_{n+1,n} \\ \hat{z}_{n+1,n} \\ \hat{\dot{x}}_{n+1,n} \\ \hat{\dot{y}}_{n+1,n} \\ \hat{\dot{z}}_{n+1,n} \\ \hat{\ddot{x}}_{n+1,n} \\ \hat{\ddot{y}}_{n+1,n} \\ \hat{\ddot{z}}_{n+1,n} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0.5\Delta t^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0.5\Delta t^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0.5\Delta t^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_{n,n} \\ \hat{y}_{n,n} \\ \hat{z}_{n,n} \\ \hat{\dot{x}}_{n,n} \\ \hat{\dot{y}}_{n,n} \\ \hat{\dot{z}}_{n,n} \\ \hat{\ddot{x}}_{n,n} \\ \hat{\ddot{y}}_{n,n} \\ \hat{\ddot{z}}_{n,n} \end{bmatrix}$$

After matrix multiplication, we get the familiar equations of motion as:

$$\begin{aligned}
\hat{x}_{n+1,n} &= \hat{x}_{n,n} + \hat{\dot{x}}_{n,n}\Delta t + \frac{1}{2}\hat{\ddot{x}}_{n,n}\Delta t^2 \\
\hat{y}_{n+1,n} &= \hat{y}_{n,n} + \hat{\dot{y}}_{n,n}\Delta t + \frac{1}{2}\hat{\ddot{y}}_{n,n}\Delta t^2 \\
\hat{z}_{n+1,n} &= \hat{z}_{n,n} + \hat{\dot{z}}_{n,n}\Delta t + \frac{1}{2}\hat{\ddot{z}}_{n,n}\Delta t^2 \\
\hat{\dot{x}}_{n+1,n} &= \hat{\dot{x}}_{n,n} + \hat{\ddot{x}}_{n,n}\Delta t \\
\hat{\dot{y}}_{n+1,n} &= \hat{\dot{y}}_{n,n} + \hat{\ddot{y}}_{n,n}\Delta t \\
\hat{\dot{z}}_{n+1,n} &= \hat{\dot{z}}_{n,n} + \hat{\ddot{z}}_{n,n}\Delta t \\
\hat{\ddot{x}}_{n+1,n} &= \hat{\ddot{x}}_{n,n} \\
\hat{\ddot{y}}_{n+1,n} &= \hat{\ddot{y}}_{n,n} \\
\hat{\ddot{z}}_{n+1,n} &= \hat{\ddot{z}}_{n,n}
\end{aligned}$$

4.7.2.4 Measurement Model

The measurement model can be represented as follows:

$$z_n = Hx_n + v_n \tag{4.4}$$

where

$$z_n \in \mathbb{R}^{3\times 1} \quad x_n \in \mathbb{R}^{9\times 1} \quad H \in \mathbb{R}^{3\times 9}$$

Note that, H is the observation matrix, which is typically used for useful state selection.

In this work, the following H matrix is considered (for position selection) :

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.5}$$

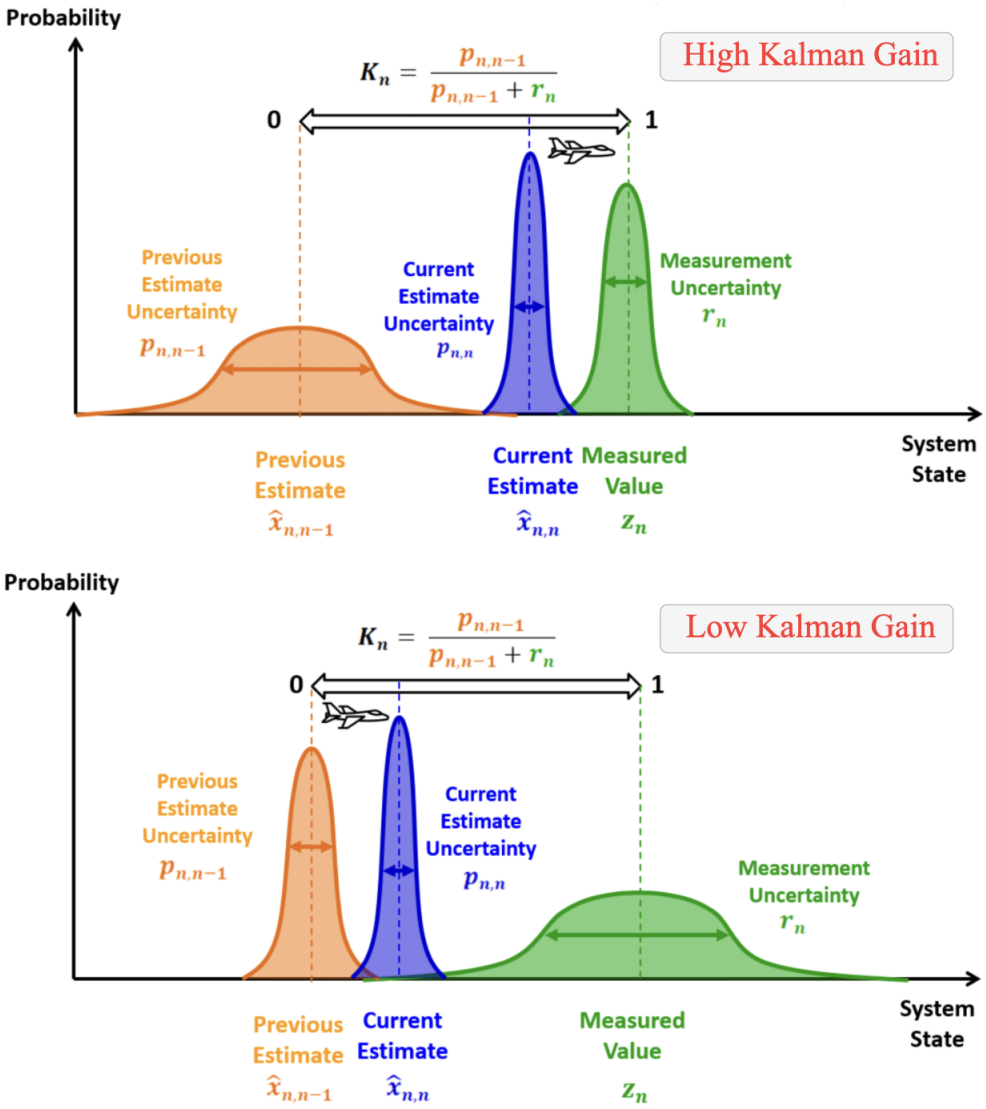


Fig. 4.15: An illustration of a simple 1D example describing the intuition of kalman gain in the state update equation. Adapted from [29].

4.7.2.5 Kalman Gain

The expression for Kalman Gain can be given by:

$$K_n = P_{n,n-1}H^T \left(HP_{n,n-1}H^T + R_n\right)^{-1} \tag{4.6}$$

Intuition is that when creating a new estimate, Kalman Gain considers both the measurement and the prior estimate. For intuition, it can be thought of as follows:

$$K_n = \frac{P_{n,n-1}H^T}{(HP_{n,n-1}H^T + R_n)} \tag{4.7}$$

Note that, for good sensors, R_n is small and therefore Kalman Gain is high. The intuition behind Kalman Gain is pictorially illustrated in Fig. 4.15.

4.7.2.6 State Update Equation

The equation is given as :

$$\hat{\boldsymbol{x}}_{n,n} = \hat{\boldsymbol{x}}_{n,n-1} + K_n \left(z_n - H\hat{\boldsymbol{x}}_{n,n-1}\right) \tag{4.8}$$

From this equation, we can see that if Kalman Gain is high, we are giving more importance to measurements. This can also be understood by observing Fig. 4.15.

4.7.3 Method-2 : Multi Layer Perceptron (MLP)

Following recent advances in DL due to greater computational capabilities and democratisation, the creation of efficient neural network topologies has resulted in performance breakthroughs for a range of issues, such as sequence-to-sequence cognitive tasks. In Methods-2 based on *Multi Layer Perceptron (MLP)*, the Kalman Filter (See Section 4.7.2) for object pose prediction is replaced with a simple Fully Connected Neural Network with non-linear ReLU activations. The complete architecture along with sequence based input and outputs is illustrated in Fig. 4.16.

Note

It should be noted in this work, one of the assumptions made is that the target object orientation remains the *same* as the previous time step (i.e. the timestep when we are predicting the future pose). Therefore in all object pose prediction methods of this section, we only estimate the future pose, whereas, orientation is duplicated from the previous timestep.

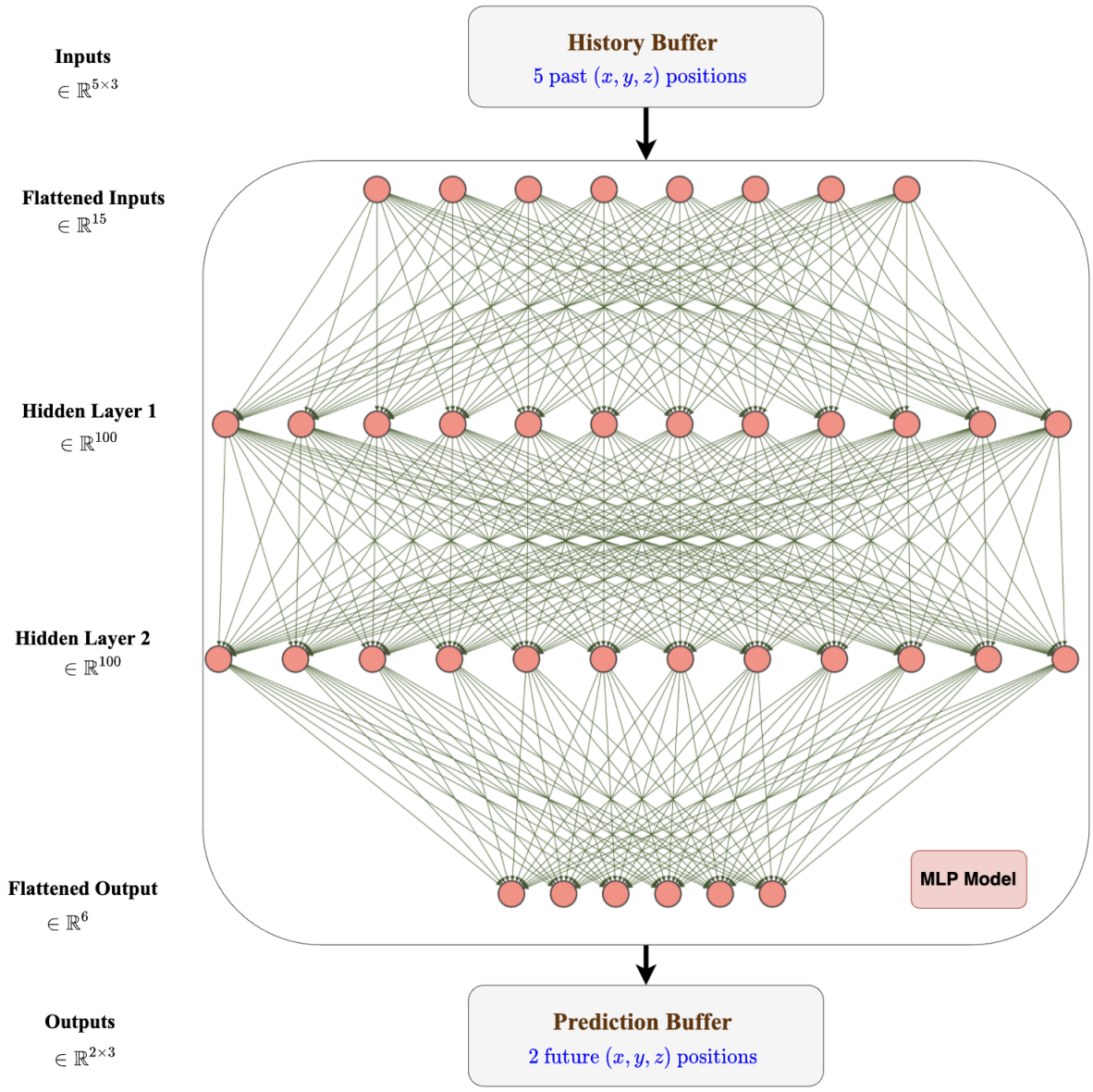


Fig. 4.16: Architecuture of MLP Model for Object Pose Prediction. Note that, here position predictions are made at two future timesteps. Also, dropout is used as a regularization mechanism here.

4.7.4 Method-3 : Long Short Term Memory Network (LSTM)

The next method that is considered an improvement to the previous method is the utilization of *Long Short Term Memory* (LSTM) Networks. The following characteristics of the LSTM networks have motivated its usage for this specific subtask of object pose prediction:

(i) Efficiently process and prioritize historical information valuable for future prediction and handle highly complex temporal data with grace.

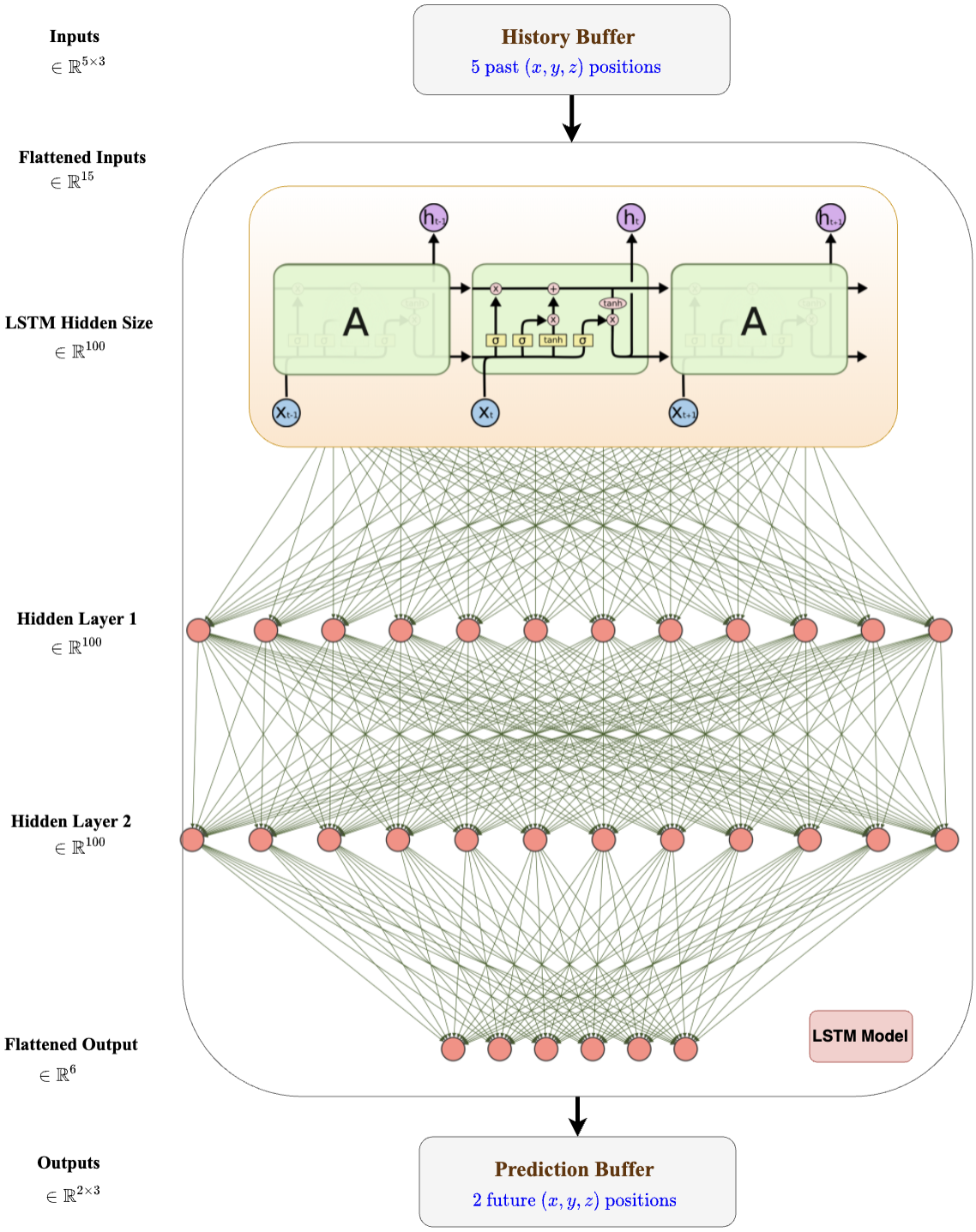


Fig. 4.17: Architecuture of LSTM Model for Object Pose Prediction. Note that, here position predictions are made at two future timesteps. Also, dropout is used as a regularization mechanism here.

(ii) LSTMs have been demonstrated to be superior than dense DNNs and early Recurrent Neural Networks (RNNs) in retaining long-term dependencies due to the addition of a

context-dependent weighted self-loop that enables them to forget previous knowledge in addition to acquiring it.

(iii) To express the connection between past and present data values, LSTMs use learned weights to reflect that link.

(iv) They can also manage multivariate time data without the requirement for dimensionality reduction or application-specific domain expertise, allowing for generalizability across a variety of domains.

(v) Furthermore, LSTM techniques have been proven to represent complicated nonlinear feature interactions and eliminate the requirement to define a time-window for considering data values owing to the usage of shared parameters over time.

Please note that in this work, predictions of the pose of the dynamic target are made at two future timesteps $t = t_0+1$ and $t = t_0+2$, respectively. The prediction to be used further for motion planning is chosen based on the conditions mentioned in the *CalcPredTime(.)* function of the dynamic grasping algorithm (Refer Section 4.4).

4.8 Chapter Summary

In this chapter, a detailed description of the approach used for *Task II - Dynamic Grasping of Moving Objects*, where the aim is to make the robotic arm grasp known objects moving in an unknown trajectory, is provided. A demonstration of a Dynamic Grasping pipeline is given, where a robotic arm was deployed in a simulation scene to grasp dynamic objects whose 3D model is known apriori, but the motion trajectory is unknown. Various object pose prediction algorithms were incorporated into the dynamic grasping system. The next chapter provides results for various experiments conducted as a part of both the tasks.

Chapter 5

Results and Inferences

In this chapter, the results are provided for the test performance of the various models and methods described in Chapter 3 on Task-1 and Chapter 4 on Task-2. The experimental configurations used and any other details will be provided wherever necessary. The results from various trials and experiments will be used to compare and analyze various aspects of the model performance.

5.1 Task-1 : Grasping Various Objects in Diverse Environments

This section provides details about the test setup, configuration, results, and analysis of different experiments conducted as part of Task-1.

5.1.1 General Configuration and Hyperparameter Details

As mentioned in Chapter 3, DRL algorithm used in various experiments is the TQC (Truncated Quantile Critics) [13] algorithm. The following table provides information regarding the hyperparameters used and configuration, specific to this DRL algorithm.

Configuration Parameter	Value
# Critics	2
Experience Replay Buffer Size	40000
Mini-batch Size	32
Activation Function	ReLU
# Training Iterations	500 K
Discount Factor γ	0.999
Learning Rate Scheduler	Linear $(1.5 \times 10^{-4} \rightarrow 0)$
Optimizer	Adam
Target Update Rate τ	5×10^{-5}
Exploratory Action Noise	$\mathcal{N}(0, 0.025)$
Initial Entropy Coefficient	0.1
# Atoms	25
# Truncated Atoms	3

Table 5.1: List of various general configuration parameters that are used in the experiments of Task-1.

Note that, more configuration details specific to an experiments or group of experiments will be provided with the results.

5.1.2 Test Setup and Evaluation Metrics

In Task-1, we are considering the task of making the robotic arm learn how to grasp novel objects in novel random scenes. Therefore, for testing we use objects and scenes (floor textures etc.) that the model has not been exposed to, during training. The test setup consists of evaluating the robot's performance and the learnt policy in 200 episodes with novel scenes and novel objects.

The different evaluation metrics that are considered for understanding the performance of different models is as follows:

1. *Success Rate* – The percentage of episodes where the robotic arm was able to successfully *reach*, *touch*, *grasp* and *lift* any object 12.5 cm above the ground.
2. *Mean Reward* – The average reward obtained by the DRL agent calculated on a total of 200 test episodes.
3. *Mean Episode Length* – The average length of episode, calculated on a total of 200 test episodes.

4. *Mean Successful Episode Length* – The average length of episode, calculated over only the successful episodes.

5.1.3 Results and Analysis

The results of various experiments that are conducted as a part of this task are shown in a comparative fashion. This for the ease of understanding and making inferences.

Note

- In the comparison-specific configuration, the actor and critic network architectures used are Vanilla (See Chapter 3 Section 3.5.3.2), unless it is explicitly mentioned.
- The training plots described in all of the comparison have smoothed to eliminate noise (caused due to mini-batch nature of training) and interpret the underlying trend. Therefore, the smoothed out curve will appear opaque and the actual curve will appear slightly transparent.

5.1.3.1 Comparison-1

■ *Configuration* –

- Robotic Arm : Panda
- Feature Extractor : Vanilla O-CNN (Refer Chapter 3 Section 3.5.2.1)
- Aim : To analyze the change in performance when coloured octree features are used.

■ *Test Results* –

Type	With Colour Features	Without Colour Features
Success Rate	45.5%	40.5%
Mean Reward	188.42 ± 191.20	174.01 ± 186.73
Mean Episode Length	65.71 ± 39.38	74.59 ± 34.54
Mean Successful Episode Length	24.78 ± 19.725	37.26 ± 24.56

Table 5.2: Test results obtained on evaluating the learnt policy on novel objects and novel scenes over 200 episodes

■ *Training Plots* –

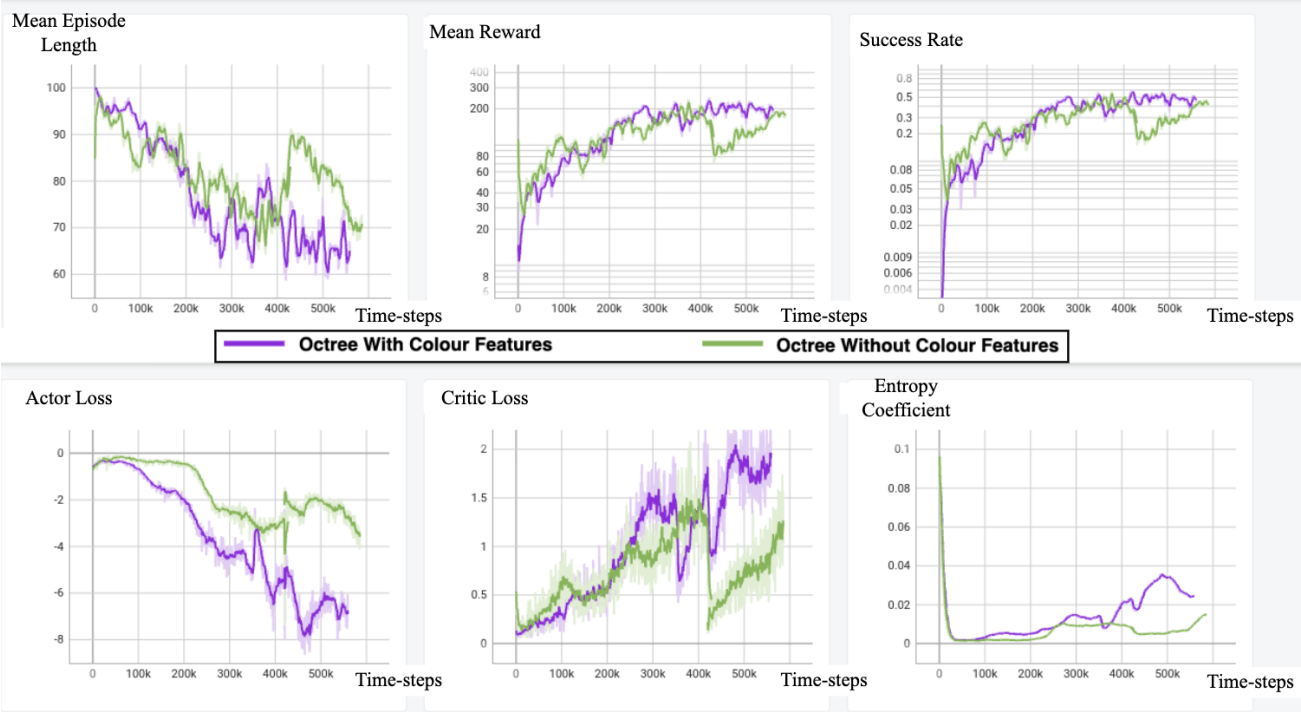


Fig. 5.1: Training Plots obtained after training the agent and the model with the specified configuration. Note that, initially the success rate of *Octree Without Colour Features* is better but later the performance of *Octree Without Colour Features* improves and is relatively more stable.

■ *Inferences* –

→ Inclusion of colored features[†] leads to the following:

[†]Octree With Colour Env consists of scene randomization and RGB features (input signal) where Octree With Colour Env does not contain them.

- Robust policy is learnt by the agent due to more domain randomization.
- Training success rate remains more stable as shown in the training plot.
- Overall performance is slightly better.

→ Initially, coloured features can be confusing and difficult to learn but leads to better policies being learnt by the agent.

5.1.3.2 Comparison-2

■ *Configuration* –

- Robotic Arm : Panda
- Aim : To analyze the change in performance when feature extractor is modified.

■ *Test Results* –

Arch.	Vanilla O-CNN	Residual O-CNN ($\boldsymbol{n} = 2$)
# Parameters	0.6795M	0.5675M
Success Rate	45.5%	45.5%
Mean Reward	188.42 ± 191.20	185.955 ± 194.76
Mean Episode Length	65.71 ± 39.38	63.965 ± 40.615
Mean Successful Episode Length	24.78 ± 19.725	20.835 ± 14.54

Table 5.3: Test results obtained on evaluating the learnt policy on novel objects and novel scenes over 200 episodes

■ *Inferences* –

→ Residual O-CNN as compared to Vanilla O-CNN :

- Learns faster due to better gradient flow.

- Finally obtains similar performance with 0.1 M parameters less.
- Better features are extracted, which is indicated by the Successful Episode length being much less.

→ Residual O-CNN performs more reliably with less variable success rates on different tests on novel scenes and new objects.

■ *Training Plots* –

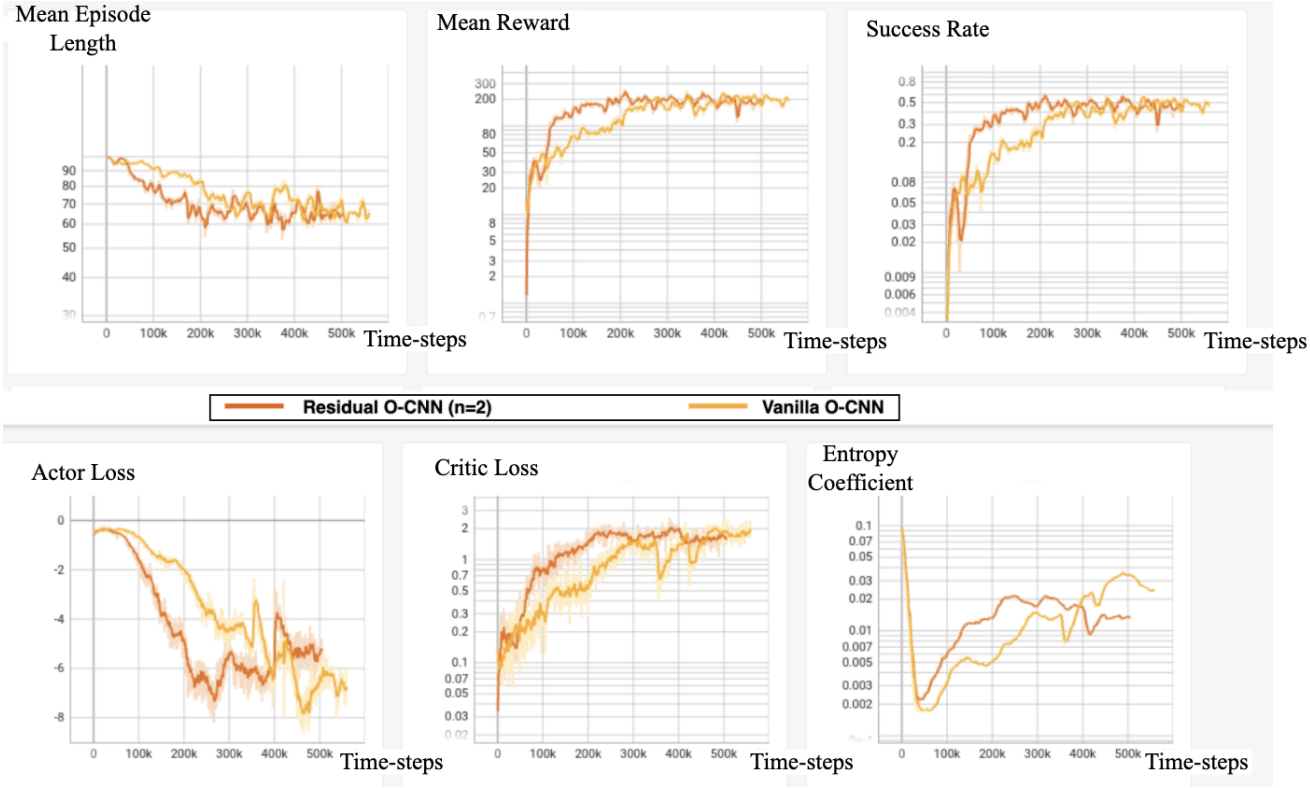


Fig. 5.2: Training Plots obtained after training the agent and the model with the specified configuration. We can see how the *Residual O-CNN* learns faster and extracts better features while having less parameters.

5.1.3.3 Comparison-3

■ *Configuration* –

- Robotic Arm : UR5 arm with RG2 gripper
- Feature Extractor : Residual O-CNN (n=4) (Refer Chapter 3 Section 3.5.2.2
- Aim : To analyze the change in performance when Batch Normalization is used in the Feature Extractor.

■ *Basic Unit Arch. (when BatchNorm is used)* –

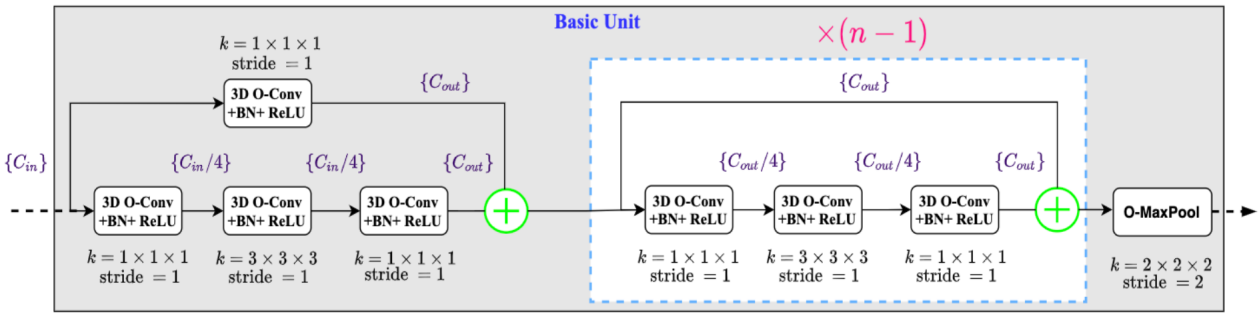


Fig. 5.3: Architecture of Basic Unit when BatchNorm is used

■ *Inferences* –

→ Use of BatchNorm in the Residual O-CNN severely reduces the performance :

- This is commonly encountered in Deep RL.
- One of the possible reasons can be that the value function is not stationary and is estimated by critic.
- There is a need to design an RL aware normalization method.

■ *Training Plots* –

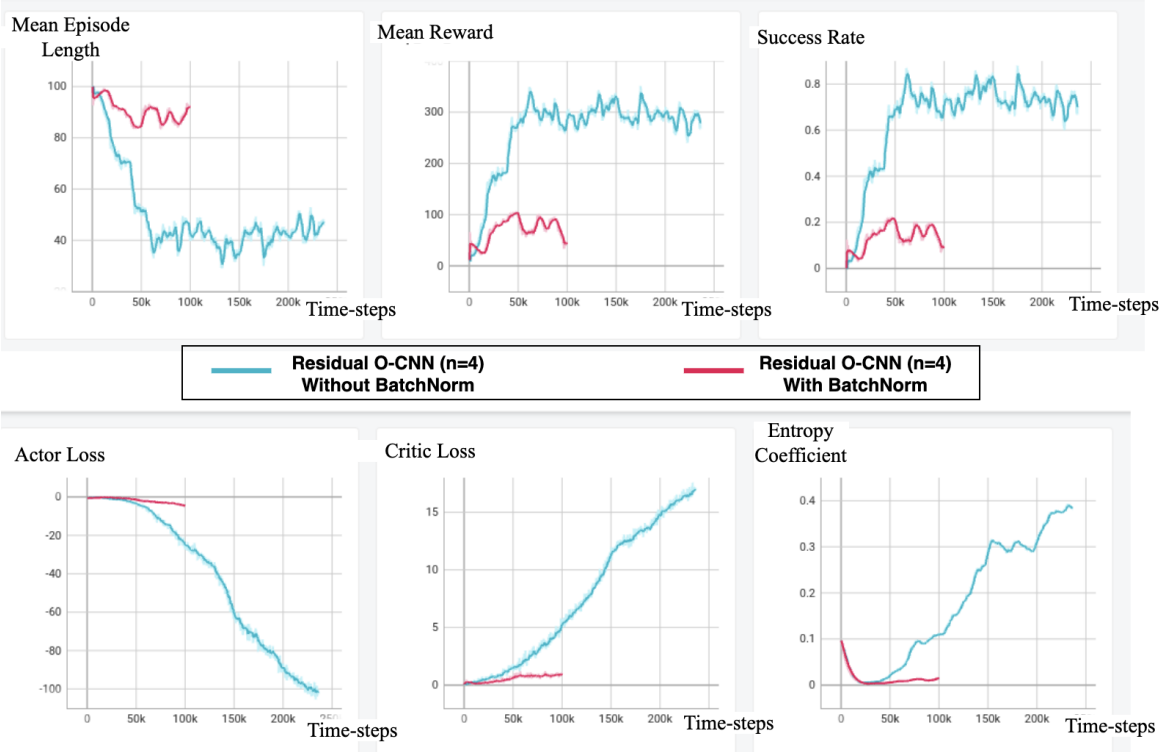


Fig. 5.4: Training Plots obtained after training the agent and the model with the specified configuration. The performance of *Residual O-CNN* is worse when BatchNorm is introduced into its architecture.

5.1.3.4 Comparison-4

■ *Configuration* –

- Robotic Arm : UR5 arm with RG2 gripper
- Feature Extractor : Residual O-CNN (n=4) (Refer Chapter 3 Section 3.5.2.2
- Aim : To analyze the change in performance when feature extractor and robotic arm is modified.

■ *Test Results* –

Arch.	Residual O-CNN $(\boldsymbol{n} = \mathbf{4})$
# Parameters	0.6348M
Success Rate	80.5%
Mean Reward	319.37 ± 152.32
Mean Episode Length	39.42 ± 34.70
Mean Successful Episode Length	25.71 ± 21.95

Table 5.4: Test results obtained on evaluating the learnt policy on novel objects and novel scenes over 200 episodes

■ *Inferences* –

→ The Residual O-CNN (n=4) + UR5 seems to perform better than others :

- Manipulation is easier to learn with UR5 arm :
  - Panda arm's gripper needs to be very precisely located for executing a successful grasp.
  - UR5 arm + RG2 gripper design allows it pick up objects in much simpler way.
- Residual O-CNN architecture has the following advantages :
  - Skip connections help in better flow of gradients during backpropagation (vanishing gradient problem solved upto some extent).
  - Identity mapping reduces excessive morphing of hidden representations.

■ *Training Plots* –

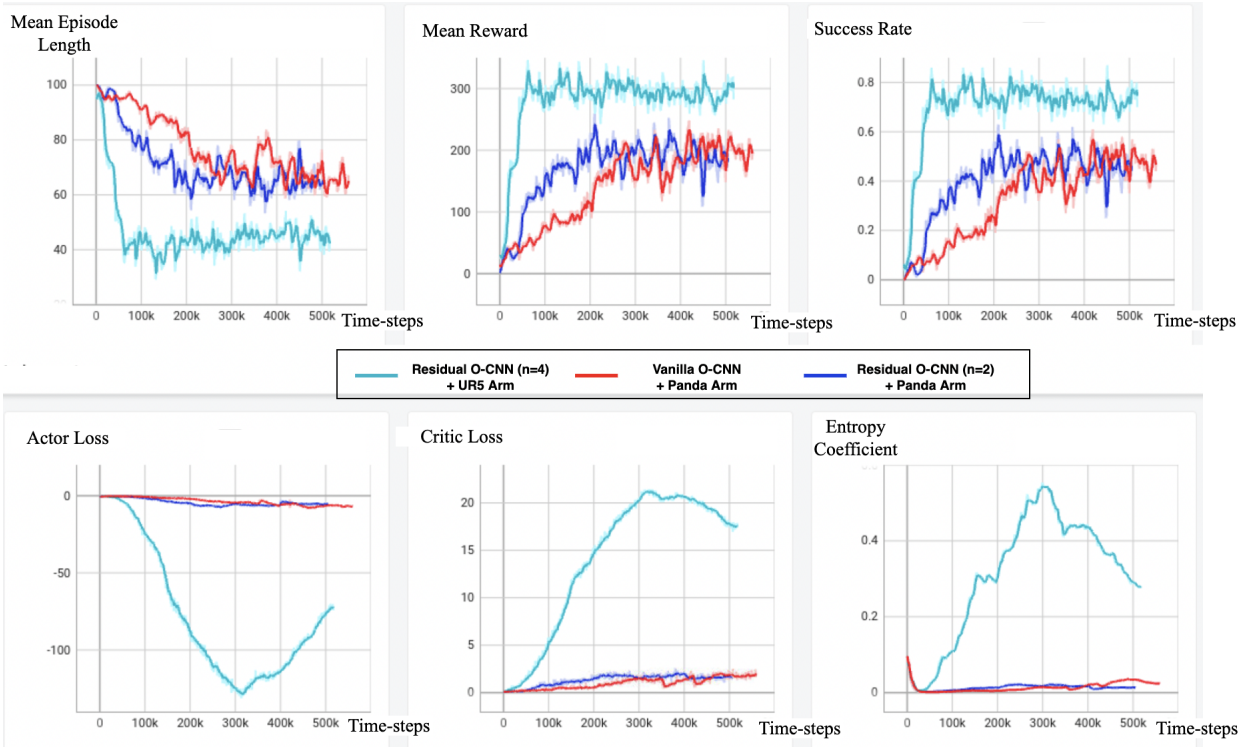


Fig. 5.5: Training Plots obtained after training the agent and the model with the specified configuration. We can clearly see how the *Residual O-CNN (n=4) + UR5 Arm* performance much better.

5.1.3.5 Comparison-5

■ *Configuration* –

- Robotic Arm : Panda Arm
- Feature Extractor : Residual O-CNN (n=4) (Refer Chapter 3 Section 3.5.2.2)
- Actor Arch. : Dense Residual (Refer Chapter 3 Section 3.5.3.2)
- Critic Arch. : Dense Residual + Duelling Arch. (Refer Chapter 3 Section 3.5.3.2)
- Description : Some initial experiments to understand the efficacy of Spectral Normalization (SN) and Duelling Networks.

■ *Training Plots* –

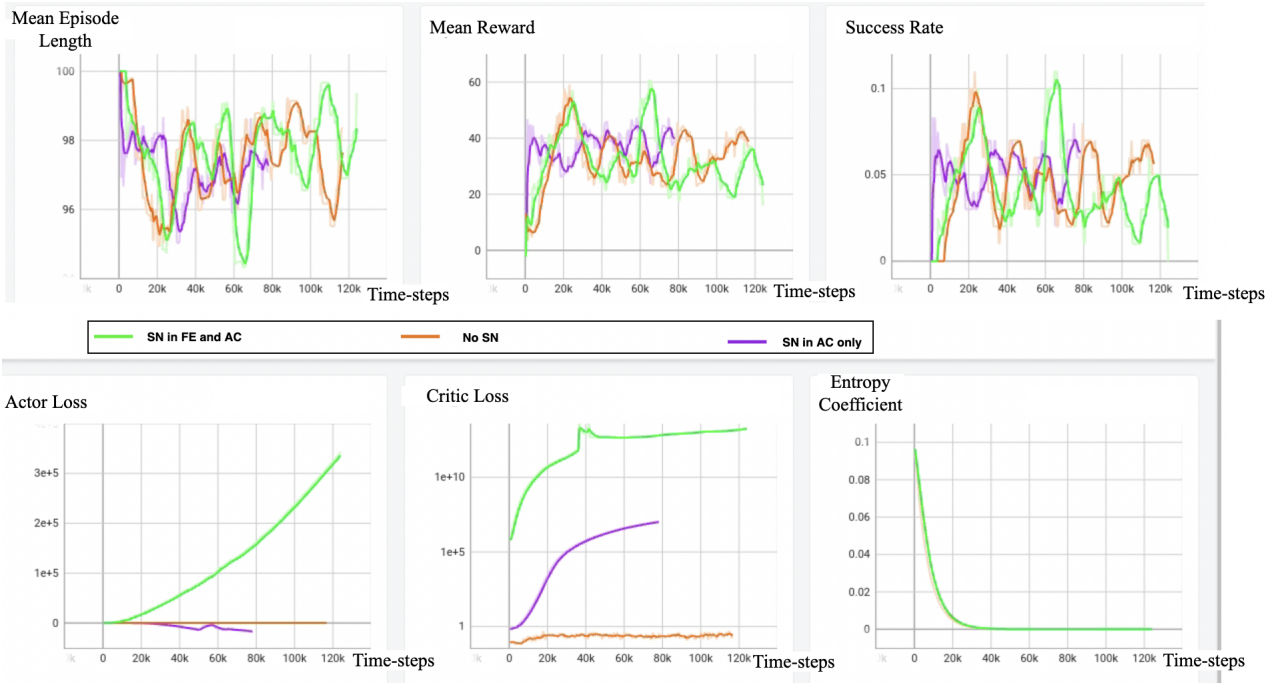


Fig. 5.6: Training curves describing the mean length of episode, mean reward of episode and success rate plotted against timesteps. Here, {FE = Feature Extractor, AC = Actor and Critic Networks}

■ *Inferences* –

→ The success rate (performance) of the experiments in the limited training time considered is not that high.

→ The critic loss seems to increase at much faster rates to exponentially higher values as the number of Spectral Norm (SN) layers increase in the whole architecture.

→ This needs to be investigated further by visualizing the gradients and digging deeper into the problem.

5.1.3.6 Comparison-6

■ *Configuration* –

- Feature Extractor : Residual O-CNN (n=4) (Refer Chapter 3 Section 3.5.2.2)
- Actor Arch. : Dense Residual (Refer Chapter 3 Section 3.5.3.2)
- Critic Arch. : Dense Residual + Duelling Arch. (Refer Chapter 3 Section 3.5.3.2)
- Description : Some initial experiments to understand the efficacy of Spectral Normalization (SN) and Duelling Networks.

■ *Inferences* –

→ The success rate (performance) of the experiments in the limited training time considered is not that high.

→ The critic loss seems to increase at much faster rates to exponentially higher values as the number of SN layers increase in the whole arch.

→ The success rate of the experiment with UR5 arm is higher because manipulation is easier in this case, but it is not very high.

→ This needs to be investigated further by visualizing the gradients and digging deeper into the problem.

■ *Training Plots* –

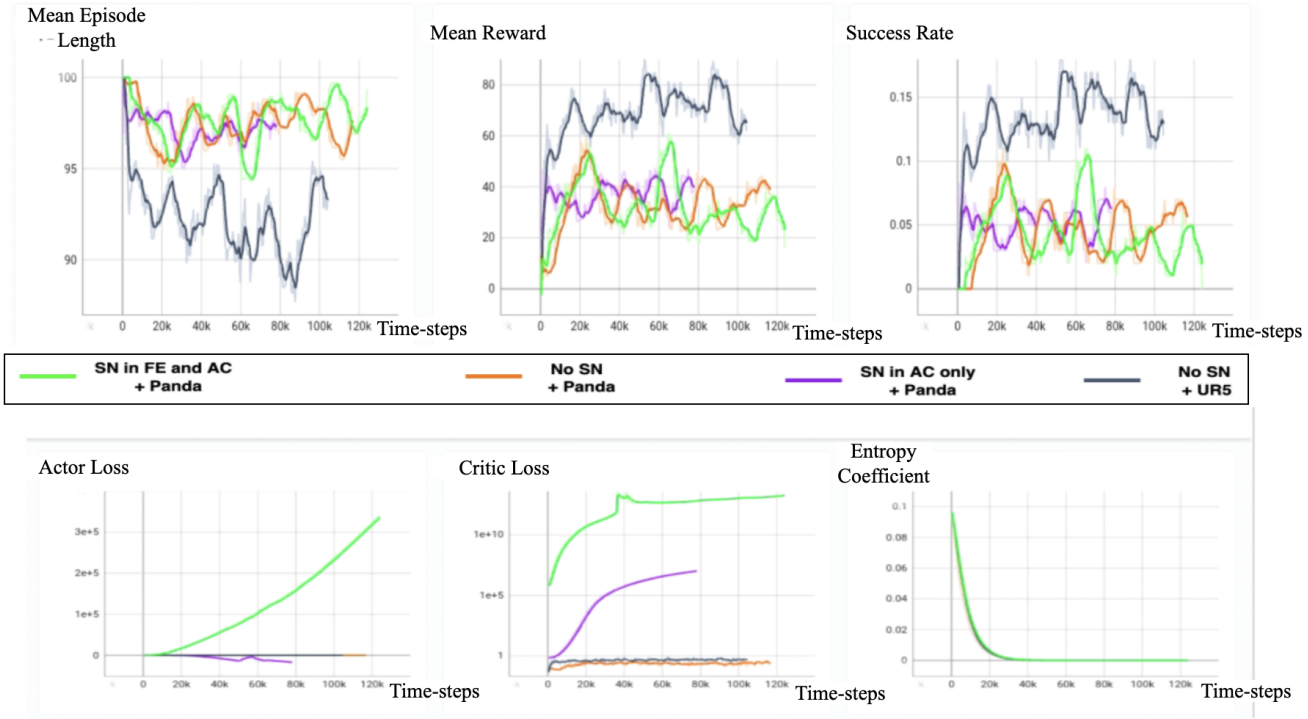


Fig. 5.7: Training curves describing the mean length of episode, mean reward of episode and success rate plotted against timesteps. Here, {FE = Feature Extractor, AC = Actor and Critic Networks, SN = Spectral Normalization}. Note that, the performance with UR5 arm is better in relative way but not that great when compared absolutely.

5.1.3.7 Comparison-7

■ *Configuration* –

- Feature Extractor : O-AHRNet (Refer Chapter 3 Section 3.5.2.3)
- Batch Size : 8
- Robotic Arm : UR5 arm with RG2 gripper
- Description : To understand the change in performance when batch size is modified.

■ *Inferences* –

When a batch size of 8 (i.e. lower batch size) is used :

→ Training becomes noisy with larger variations in success rate than usual.

→ Also performance seems to increase gradually but takes more time steps to reflect in the curves.

■ *Training Plots* –

Fig. 5.8: Training curves describing the mean length of episode, mean reward of episode and success rate plotted against timesteps. We can see that when a lower batch size is used, the success rate curve is very noisy but tends to improve gradually.

5.1.3.8 Comparison-8

■ *Configuration* –

- Robotic Arm : UR5 with RG2 Gripper
- Aim : To analyze the change in performance when feature extractor is modified.

■ *Test Results* –

Arch.	Residual O-CNN ($\boldsymbol{n} = 4$)	O-AHRNet
# Parameters	0.6348M	2.6614M
Success Rate	80.5%	87.5%
Mean Reward	319.37 ± 152.32	349.34 ± 123.41
Mean Episode Length	39.42 ± 34.70	29.57 ± 31.03
Mean Successful Episode Length	25.71 ± 21.95	19.50 ± 17.04

Table 5.5: Test results obtained on evaluating the learnt policy on novel objects and novel scenes over 200 episodes

■ *Inferences* –

→ O-AHRNet seems to perform better :

- It has a built-in capability to maintain good High Resolution feature representations due to repeated multi-depth octree fusion features.
- Attention module is helpful for assisting the model to understand where to focus.

→ Qualitatively, after observing the performance in simulation, the action of the object pickup was calculated and less wavering as compared to previous Feature Extraction Architectures.

■ *Training Plots* –

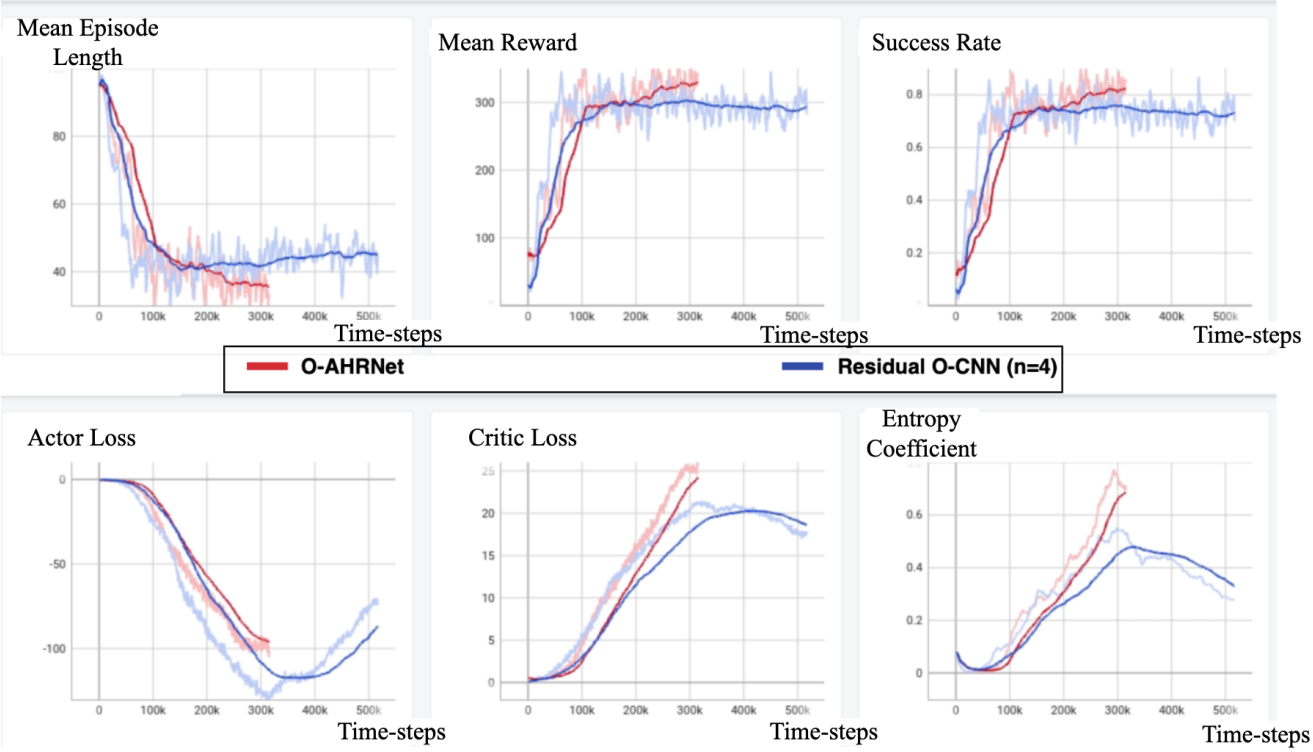


Fig. 5.9: Training Plots obtained after training the agent and the model with the specified configuration. We can see that the performance of the *O-AHRNet* model tends to be better than the *Residual O-CNN (n=4)* due to its advanced architecture design.

5.1.4 Summary of Notable Experiments

The following table describes a summary of notable experiments conducted as a part of the Task I : Grasping Various Objects in Diverse Environments. Please note that more details and metrics of each experiments can be found in the comparisons of Section 5.1.

Expt No.	Robotic Arm	Feature Extractor (FE)	# Parameters in FE	Actor-Critic Arch.	Other Configuration	Success Rate (in %) / Comments
1.	Panda	Vanilla O-CNN	0.6795 M	Vanilla	No Colour Features	40.5 %
2.	Panda	Vanilla O-CNN	0.6795 M	Vanilla	Colour Features	45.5 %
3.	Panda	Residual O-CNN ($n = 2$)	0.5675 M	Vanilla	-	45.5 %
4.	UR5	Residual O-CNN ($n = 4$)	0.6348 M	Vanilla	BatchNorm Present	Performance degraded during training
5.	UR5	Residual O-CNN ($n = 4$)	0.6348 M	Vanilla	-	80.5 %
6.	Panda	Residual O-CNN ($n = 4$)	0.6348 M	Advanced	-	Performance does not improve
7.	UR5	Residual O-CNN ($n = 4$)	0.6348 M	Advanced	-	Absolute performance is low but relatively better than Panda
8.	UR5	O-AHRNet	2.6614 M	Vanilla	Low Batch Size	Noisy but gradual performance increase
9.	UR5	O-AHRNet	2.6614 M	Vanilla	-	87.5 %

Table 5.6: List of Notable Experiments conducted as a part of Task I. Here, for Actor Critic architectures, "Advanced" means the Actor network has Dense Residual architecture and Critic network has Dense Residual + Dueling architecture (Refer Chapter 3 Section 3.5.3.2).

5.2 Task-2 : Dynamic Grasping of Moving Objects

This section provides details about the test setup, configuration, results, and analysis of different experiments conducted as part of Task-2.

5.2.1 Configuration and Hyperparameter Details

As mentioned in Chapter 4, Object Pose Prediction (See Chapter 4 Section 4.3.2) is an important part of the *Dynamic Grasping* setup. The following table provides information regarding the hyperparameters used and configuration, specific to this pose prediction subtask.

5.2.1.1 Dataset Used

For the DL models considered, a dataset needs to be created for providing supervision through training. The input to these models is a sequence of object poses $(o_{t-n}, \cdots o_{t-1}, o_t)$ and the output is the predictions of future poses at different horizons $\left(o_{t_{f_1}}, o_{t_{f_2}}, \cdots, o_{t_{f_m}}\right)$ {Here, $t_{f_m} = 2$}. The generated dataset has the following features:

Configuration Parameter	Value
Activation Function	ReLU
Regularisation	Dropout
Optimizer	Adam
Loss Function	Mean Squared Error
Fully Connected Hidden Layer Size	100
LSTM Hidden Units	100
Batch Size	1000
# Training Epochs	200
Input Sequence Length	5
Output Sequence Length	2

Table 5.7: List of various general configuration parameters that are used in the experiments of Task-2, for Object Pose Prediction component.

- Separately created for Linear, Circular and Sinusoidal Cases.
- Consists of a sequence of 2000 waypoints*.
- In order to ensure variety in training data, the paths generated have different start points and are in different directions.

Also, to make it easier for the model to learn, generalize well and be able to predict even on starting from a random point on the trajectory, each sequence created from the dataset is normalized to the start point i.e., normalized to the start of the sequence i.e.

$$\left\{ \underbrace{(0, \cdots, o_{t-1} - o_{t-n}, o_t - o_{t-n})}_{\text{Normalized Input Sequence}} \rightarrow \underbrace{\left(o_{t_{f_1}} - o_{t-n}, o_{t_{f_2}} - o_{t-n}\right)}_{\substack{\text{Normalized Output Sequence} \\ \text{[Ground Truth]}}} \right\}$$

*Different points sampled on the trajectory at regular intervals of the path

5.2.2 Test Setup and Evaluation Metrics

In Task-2, we are considering the task of making the robotic arm learn how to grasp dynamic known objects, whose motion is not known apriori. The test setup consists of evaluating the robot's performance and the dynamic grasping algorithm in 100 trials for each of the object and each type of motion mentioned in Chapter 4 Section 4.6.

The different evaluation metrics that are considered for understanding the performance of different methods is as follows:

1. *Success Rate* – The percentage of trials where the robotic arm was able to successfully *reach*, *grasp* and *lift* the moving target.

2. *Dynamic Grasping Time* – The average length of trial, calculated over all the 100 trials.

5.2.3 Results

The results of various experiments that are conducted as a part of this task are shown method by method.

5.2.3.1 Method-1 : Kalman Filter

Test Results –

The following table provides the results for Method-1, using Kalman Filter, for various objects and different types of motion. For each object :

- 1^{st} sub-row corresponds to the success rate (in %) out of the 100 test trials conducted.
- 2^{nd} sub-row corresponds to the average dynamic grasping time (in s) out of the 100 test trials conducted.

Object \ Type of Motion	Linear	Linear with Obstacles	Linear with Top Shelf	Circular	Sinusoidal
Bleach Cleanser	82	82	53	74	15
	17.1268	18.0885	25.4492	16.2566	26.8206
Potted Meat Can	86	82	55	85	8
	12.6949	13.5474	25.6200	13.2181	25.0831
Power Drill	88	84	31	81	16
	9.8143	9.1870	22.1733	9.7166	17.9101
Mustard Bottle	87	82	47	78	10
	15.5122	18.6734	26.3625	15.1800	27.2544
Sugar Box	86	84	47	88	5
	13.3597	13.8027	25.4042	12.4203	22.9465
All Objects	85.8	82.8	46.6	81.2	10.8
	13.7016	14.6598	25.0018	13.3583	24.0029

Table 5.8: Table displaying the test results for Method-1 used in the Object Pose Prediction subtask. Details of the method are described in Chapter 4.

5.2.3.2 Method-2 : Multi Layer Perceptron (MLP)

Training and Validation Loss –

Dataset Type \ Metric	Training Error	Validation Error
Linear Case	4.4124×10^{-7}	6.6252×10^{-5}
Sinusoidal Case	2.8202×10^{-4}	2.4454×10^{-4}

Test Results –

The following table provides the results for Method-2, using MLP model, for various objects and different types of motion. For each object :

- 1^{st} sub-row corresponds to the success rate (in %) out of the 100 test trials conducted.
- 2^{nd} sub-row corresponds to the average dynamic grasping time (in s) out of the 100 test trials conducted.

Object \ Type of Motion	Linear	Linear with Obstacles	Linear with Top Shelf	Sinusoidal
Bleach Cleanser	88	78	51	66
	17.3113	18.3565	24.9106	22.5683
Potted Meat Can	84	80	55	79
	14.3890	13.5307	25.8155	15.7835
Power Drill	91	91	30	88
	10.2867	8.7359	21.9116	11.6545
Mustard Bottle	88	78	49	63
	16.4776	19.2276	26.4544	22.8853
Sugar Box	74	79	46	78
	15.1663	15.5597	25.9263	17.1639
All Objects	85	81.2	46.2	74.8
	14.7262	15.0821	25.0037	18.0111

Table 5.9: Table displaying the test results for Method-2 used in the Object Pose Prediction subtask. Details of the method are described in Chapter 4.

5.2.3.3 Method-3 : Long Short Term Memory (LSTM)

Training and Validation Loss –

Dataset Type \ Metric	Training Error	Validation Error
Linear Case	6.7592×10^{-6}	1.8709×10^{-6}
Sinusoidal Case	2.0078×10^{-4}	7.8373×10^{-5}

Test Results –

The following table provides the results for Method-1, using LSTM model, for various objects and different types of motion. For each object :

- 1^{st} sub-row corresponds to the success rate (in %) out of the 100 test trials conducted.
- 2^{nd} sub-row corresponds to the average dynamic grasping time (in s) out of the 100 test trials conducted.

Object \ Type of Motion	Linear	Linear with Obstacles	Linear with Top Shelf	Sinusoidal
Bleach Cleanser	82	82	49	75
	16.5867	18.6834	25.2586	21.0621
Potted Meat Can	90	86	54	85
	12.2976	13.0665	25.0033	15.6227
Power Drill	85	89	30	82
	9.3358	8.7544	21.6791	11.6880
Mustard Bottle	87	77	49	63
	17.2409	18.9082	25.7427	23.4247
Sugar Box	83	88	49	74
	14.0838	14.5649	25.2388	17.6545
All Objects	85.4	84.4	46.2	75.8
	13.9090	14.7954	24.5845	17.8904

Table 5.10: Table displaying the test results for Method-3 used in the Object Pose Prediction subtask. Details of the method are described in Chapter 4.

5.2.4 Inferences

The following are some of the inferences that can be made by observing the results of various experiments mentioned in the previous section:

- Kalman Filter works fairly well for predicting the pose of objects moving in a linear path, for all the three linear cases namely: *Linear*, *Linear with Obstacles* and *Linear with Top Shelf* cases (Refer Table 5.8).
- Kalman Filter's performance is poor for the *Sinusoidal* motion case because constant acceleration model is considered. So success rate is low and grasping time is high for sinusoidal motion (Refer Table 5.8).

- For the sinusoidal case, the MLP model performs *fairly* well but the best performing model is the LSTM model which matches the common intuition. LSTM is able to capture the non-linear temporal relations with ease (Refer Table 5.9 and Table 5.10).
- For all the methods, the performance for the case of *Linear with Top Shelf* is not that great because of the top shelf obstacle right above the dynamic target. The grasping time is also higher since more time is required for planning due to restrictions on some trajectories.
- For all the methods, the performance for the case of *Linear with Obstacles* is success rate is slightly lower and grasping time is also slightly higher, as compared to the *Linear* case. This is expected due to presence of random obstacles. Note that, the performance gap between both these cases is the least for the LSTM model, showing its effectiveness.

Method \ Type of Motion	Linear	Linear with Obstacles	Linear with Top Shelf	Sinusoidal
Kalman Filter	85.8	82.8	46.6	10.8
	13.7016	14.6598	25.0018	24.0029
Multi Layer Perceptron (MLP)	85	81.2	46.2	74.8
	14.7262	15.0821	25.0037	18.0111
Long Short Term Memory Network (LSTM)	85.4	84.4	46.2	75.8
	13.9090	14.7954	24.5845	17.8904

Table 5.11: List of notable experiments performed as a part of Task II : Dynamic Grasping of Moving Objects.

5.2.5 Summary of Notable Experiments

Table 5.11 describes a summary of notable experiments conducted as a part of the Task II : Dynamic Grasping of Moving Objects. Please note that more details and metrics of each experiments can be found in the Section 5.2. In this table, the success rates and dynamic grasping time are average over all the five objects considered. For each method :

- 1^{st} sub-row corresponds to the success rate (in %) out of the 500 test trials conducted.

- 2^{nd} sub-row corresponds to the average dynamic grasping time (in s) out of the 500 test trials conducted.

5.3 Chapter Summary

This chapter included the results of deploying the approaches developed for both tasks in various test scenarios. For Task I, an extensive comparison of various experiments conducted in the DRL setting was performed. Some experiments involving the incorporation of novel DL ideas, such as advanced normalization techniques and improved architecture of Actor and Critic Networks, motivated the need to research DRL analogues for them. For the second task, extensive experimentation, comparison and inference based on obtained results were made for different methods. Both tasks discussed until now were performed and tested in simulation. Chapter 6 explores a more challenging aspect of setting up a "real" robotic grasping system using a Kinova arm.

Chapter 6

Setting up a "Real" Robotic Grasping System

In this chapter, details about some of the basic tasks necessary to setting up a real-life robotic system are discussed. Note that, it is possible to use perception to perform intelligent grasping activities as an extension to this work.

6.1 Kinova `j2s7s300` Robotic Arm

6.1.1 General Details

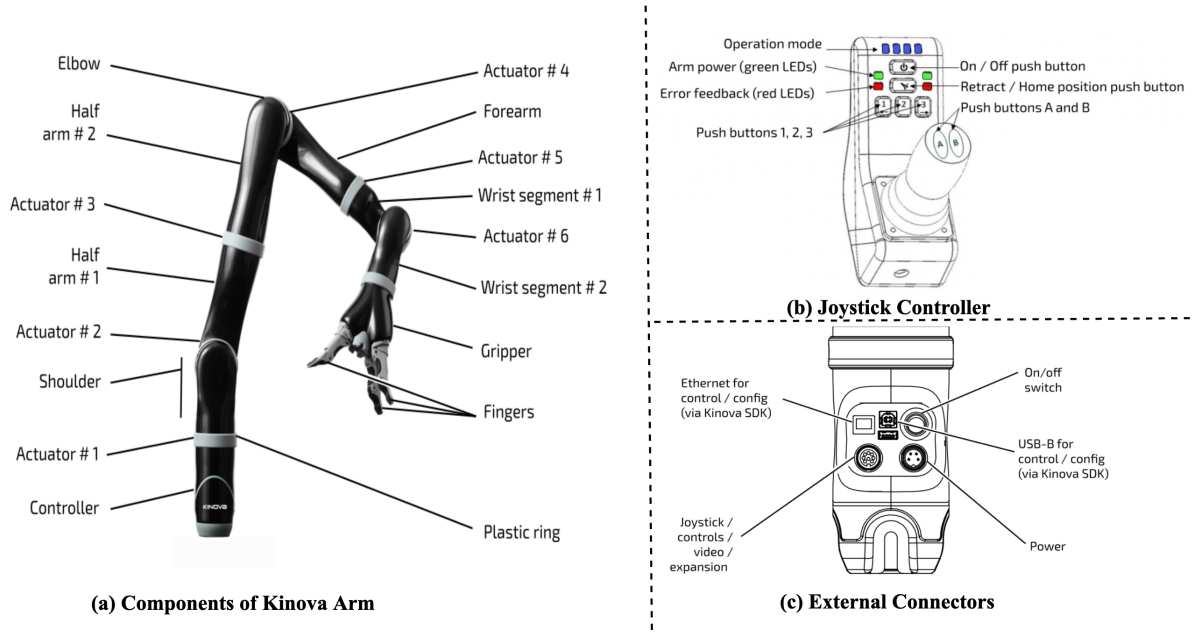


Fig. 6.1: Illustration of different components present in the Kinova `j2s7s300` Arm and the Joystick Controller. Note that, the joystick controller can be used as a safety stop to avoid damage in emergency situations.

`j2s7s300` arm is a 7-DoF flexible service robotic arm manufactured by Kinova (See Chapter 1 Section 1.2.1). It is designed to work with human beings in a collaborative and constructive manner, enhancing their abilities. mm. Striking a perfect balance between size and power, the `j2s7s300` arm is designed for seamless integration into a wide range of applications. More technical details about it can be found in the datasheet*.

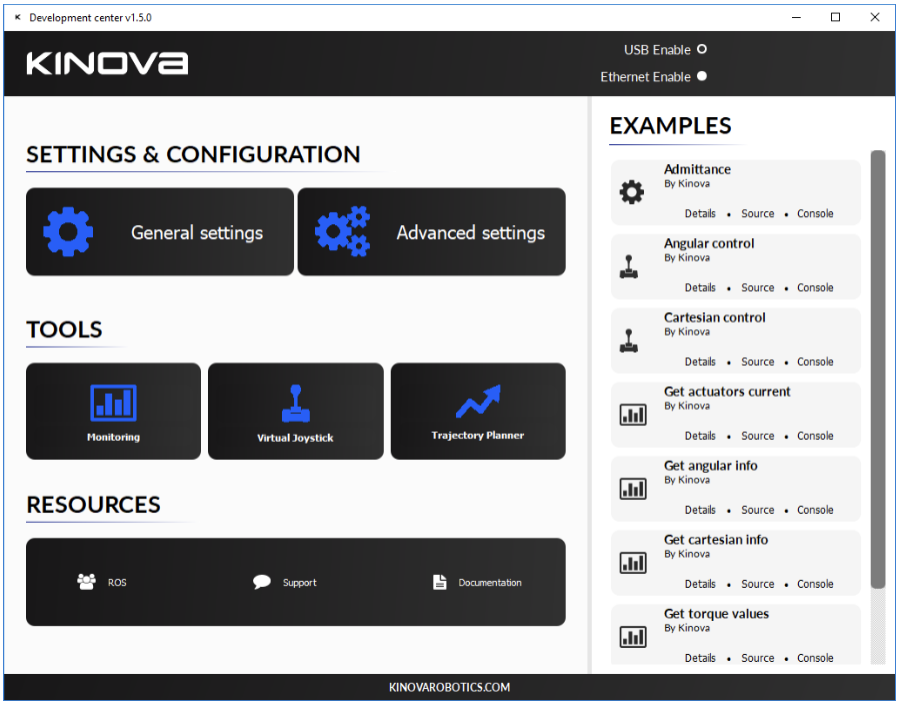


Fig. 6.2: A Glimpse of the Kinova Software Development Studio. It can be used for high-level control of the robotic arm.

The Kinova arm can be controlled from the *Kinova SDK* as shown in Fig. 6.2. It provides a wide range of features and capabilities. Users may transmit trajectories, check their robot's condition, enable admittance, and switch between Cartesian and angular control. It is also possible to utilise the Kinova software to upgrade the robot's firmware.

For getting more fine-grained control over the robotic arm and setting it up performing complex ARG tasks, the better alternative would be to use the ROS stack† for Kinova `j2s7s300` arm. With the drivers provided in the ROS stack, it is possible to perform the following, using the robotic arm:

*https://cutt.ly/lHwlZe0
†https://github.com/Kinovarobotics/kinova-ros

(i) *Joint position control* – It can be used to move the joint angles of various joints present in the arm in a relative or absolute manner. The following is an exemplary command that will move the 4th joint of `j2s7s300` robotic arm to rotate -20 degree relatively:

```
 $ rosrun kinova_demo joints_action_client.py -v \
-r j2s7s300 degree - 0 0 0 -20 0 0 0
```

(ii) *Cartesian position control* – It can be used to move the robotic arm's end effector in the cartesian space (w.r.t base frame of the robot) in a relative or absolute manner. The following is an exemplary command that will drive the `j2s7s300` robotic arm to traverse along -y axis for 3cm and rotate the gripper for +40 degree along arm axis, relatively :

```
 $ rosrun kinova_demo pose_action_client.py -v \
-r j2s7s300 mdeg - 0 -0.03 0 0 0 40
```

(iii) *Finger position control* – It can be used to move the fingers of the robotic arm's end effector. The following is an exemplary command that will close the fingers completely :

```
 $ rosrun kinova_demo fingers_action_client.py \
j2s7s300 percent - 100 100 100
```

6.1.2 Integration with MoveIt

MoveIt is a popularly used motion planning software compatible with ROS. Kinova ROS stack also provides support for MoveIt. The repository consists of default configuration files for `j2s7s300` robotic arm. Some details regarding the MoveIt setup assistant configuration is illustrated in Fig. 6.3.

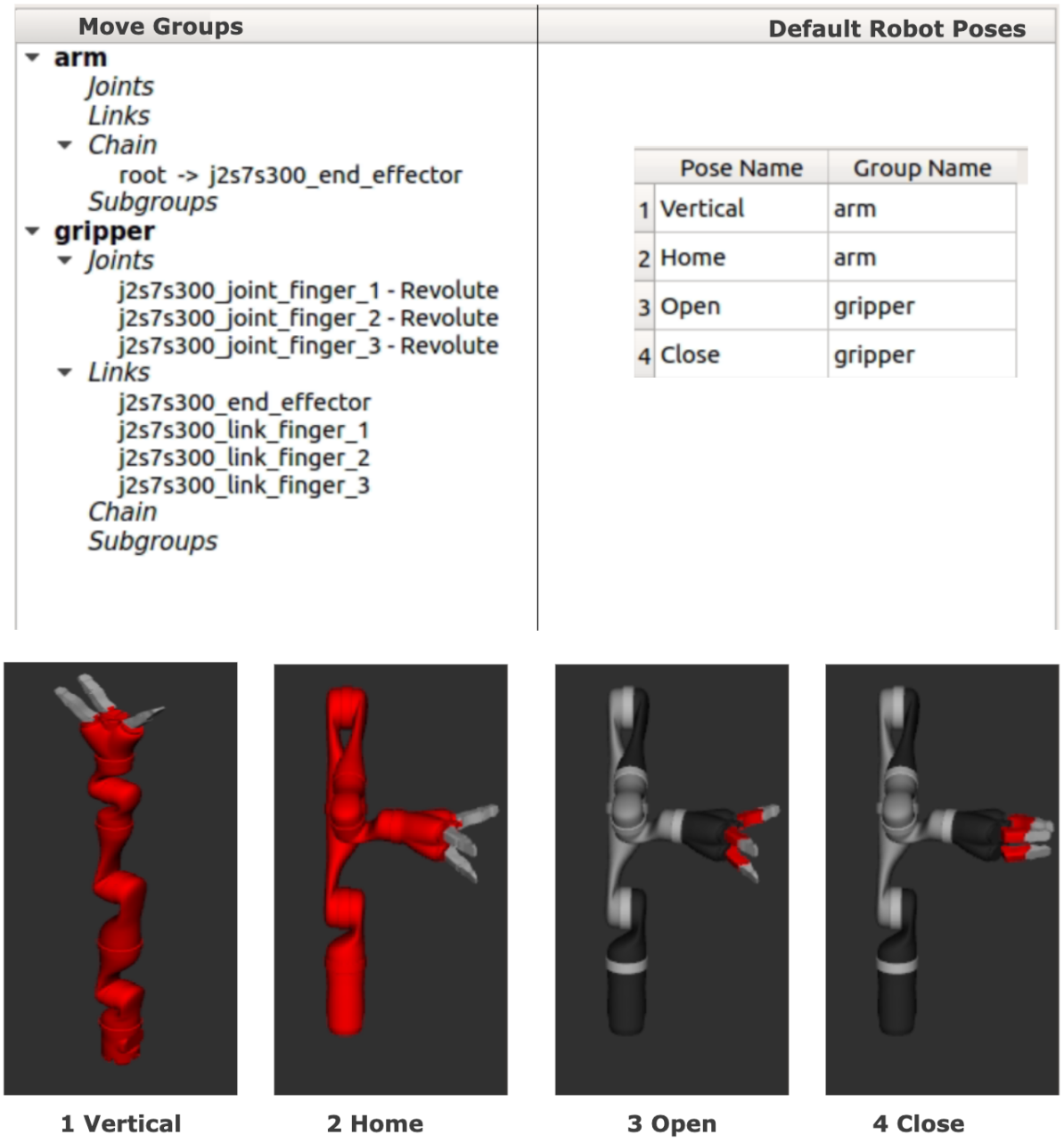


Fig. 6.3: Illustration of the various components present in the Kinova `j2s7s300` robot's MoveIt Setup Assistant. Note the demonstration default poses present in the Configuration.

The robotic arm has been setup to use two move groups *arm* and *gripper*, as shown in Fig. 6.3.

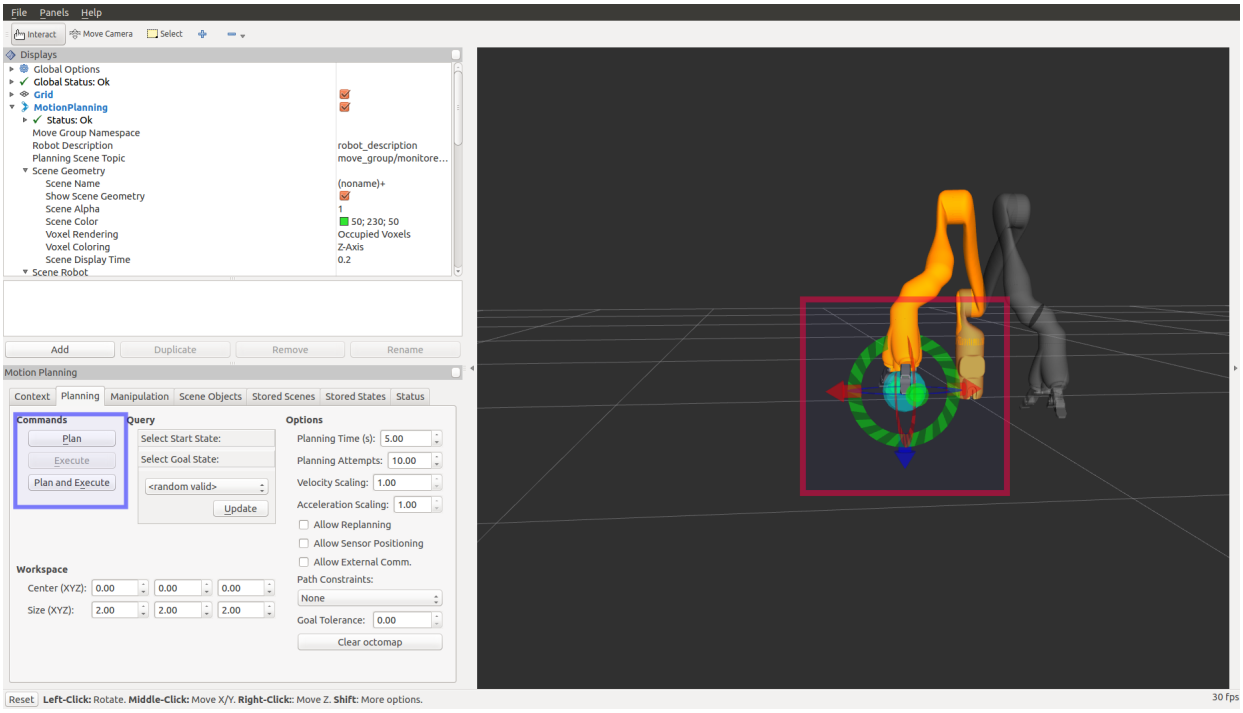


Fig. 6.4: A Glimpse of the RViz window of Kinova MoveIt setup. On the left, we can see different tabs for performing various activities. On the right, the robotic arm can be visualized and controlled with the interactive markers.

The following is a brief list of some of the different functionalities that can be used after integration of `j2s7s300` robotic arm with MoveIt is successful :

- *RViz MoveIt Plugin* –

  RViz can be used to visualize the robotic arm and other related aspects. When the real robot is connected, the movements of the virtual and real arms are synchronized. It also provides interactive markers (as highlighted in Fig. 6.4) so that the end effector can be moved in a graphical manner with ease.

- *Start and Goal Poses* –

  Under *Motion Planning*, the *Planning* tab provides the functionality to set a start and goal pose of the arm to be used later. We can define custom poses to execute complex trajectories.

- *Plan and Execute Trajectories* –

  As shown in the blue rectangle of Fig. 6.4, we can plan the motions of the arm before executing them. If there is any issue with the current trajectory, we can also replan the motion.

- *Adding Obstacles* –

  If the workspace is full of static obstacles, we can model the obstacles, so that the robotic arm can be aware of them and avoid collisions. More details about this will be given in Section 6.4.2.

6.2 Intel Realsense `D415` RGBD Camera

Intel Realsense `D415` RGBD Camera is a stereo vision depth camera system. The camera subsystem's compact size and simplicity of integration provide system integrators the ability to build into a broad variety of applications. More technical specifications can be found in the datasheet[‡].

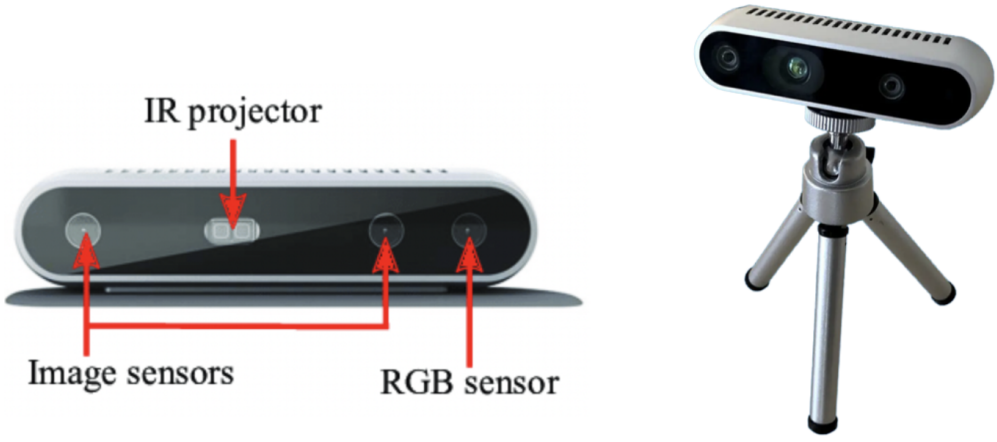


Fig. 6.5: Illustration of different components present in the Realsense `D415` camera. The image on the right shows the camera mounted on a tripod.

The ROS wrapper[§] for the camera is also provided so that it can be easily integrated into robotics applications.

[‡] https://cutt.ly/WHwSpwa
[§] https://github.com/IntelRealSense/realsense-ros

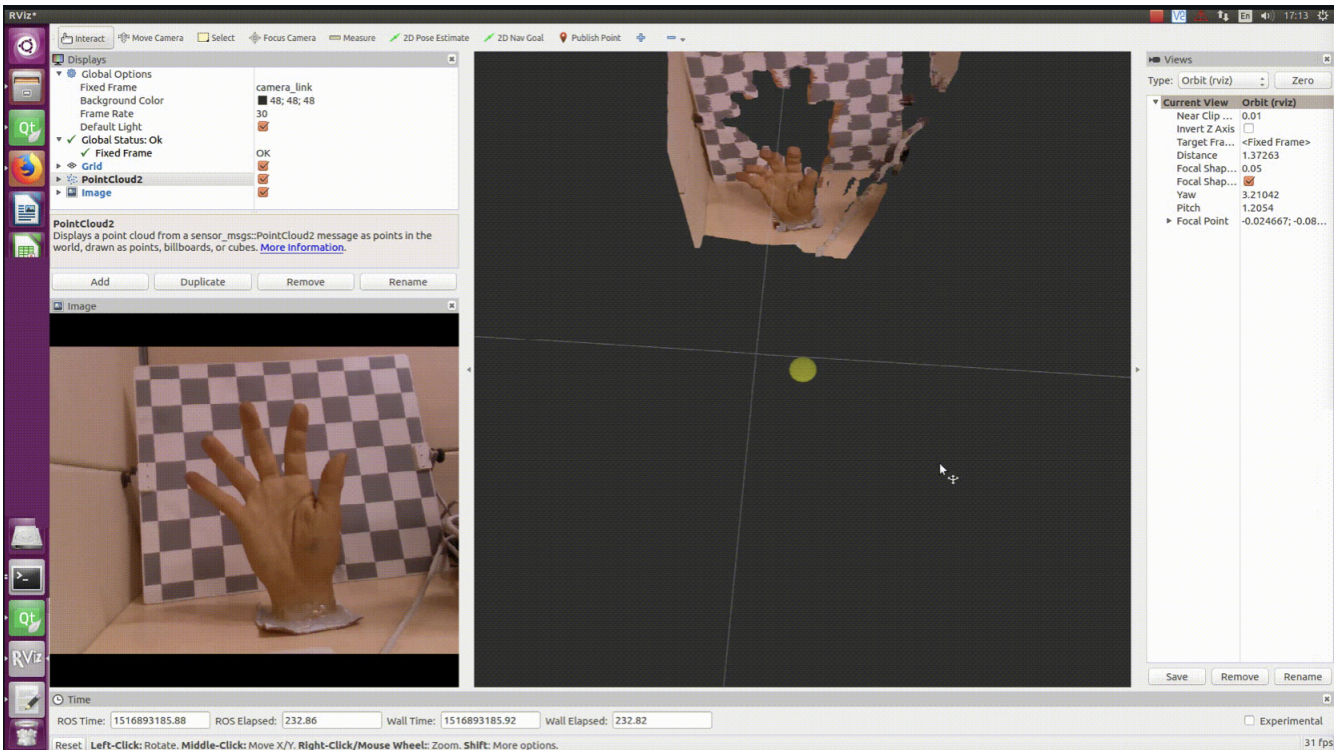


Fig. 6.6: A Glimpse of the RViz window, consisting of a camera connected and publishing point cloud data of the scene. On the bottom left, we can also see the normal RGB image captured by the camera.

6.3 Hand-Eye Calibration

One of the important tasks while creating a perception based "real" robotic grasping setup is *Hand-Eye* Calibration. It essentially binds the camera and robotic arm so that both of them can work together in synergy to pick the desired object.

6.3.1 Types of Setup

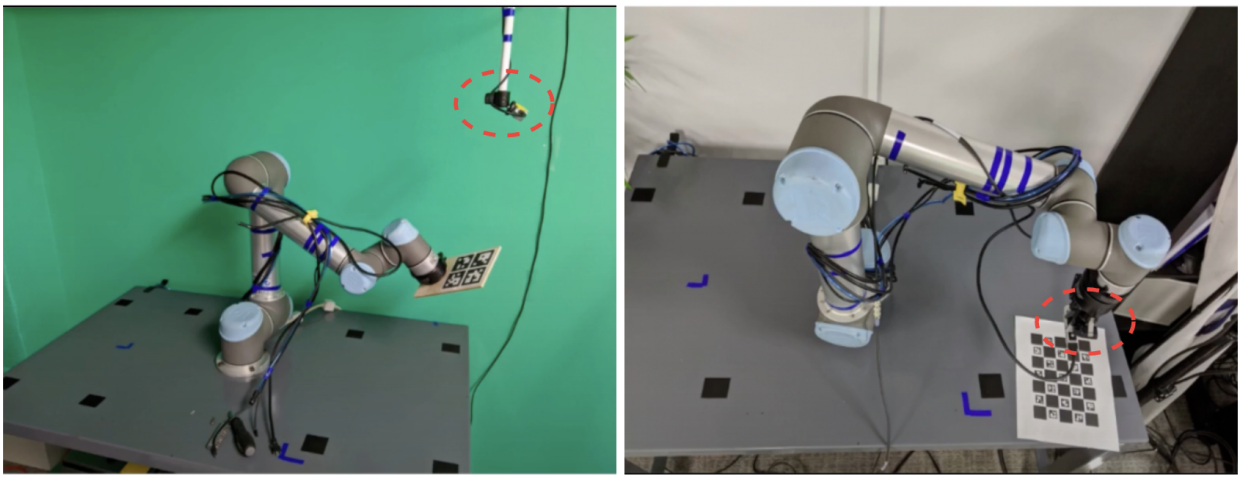
There are two common types of setup for Hand Eye Calibration (See Fig. 6.7):

1. *Eye-on-Base Setup* –

   The perception module is mounted in a fixed place, so that it remains stationary when the robotic arm moves.

2. *Eye-in-Hand Setup* –

   The perception sensor is mounted directly on the robotic arm. The camera also moves as the robotic arm moves.



(a) Eye-on-Base (b) Eye-in-Hand

Fig. 6.7: Different Kinds of Common Hand-Eye Calibration setups. Note that, the camera is highlighted in both of the images.

6.3.2 Problem Description and Solution Theory

The camera's coordinate system is used for all visual information it collects. In order for the arm to utilise visual input, we must determine the transformation between robotic arm's end effector and camera coordinate systems. This is the central problem solved by *Hand-Eye Calibration*.

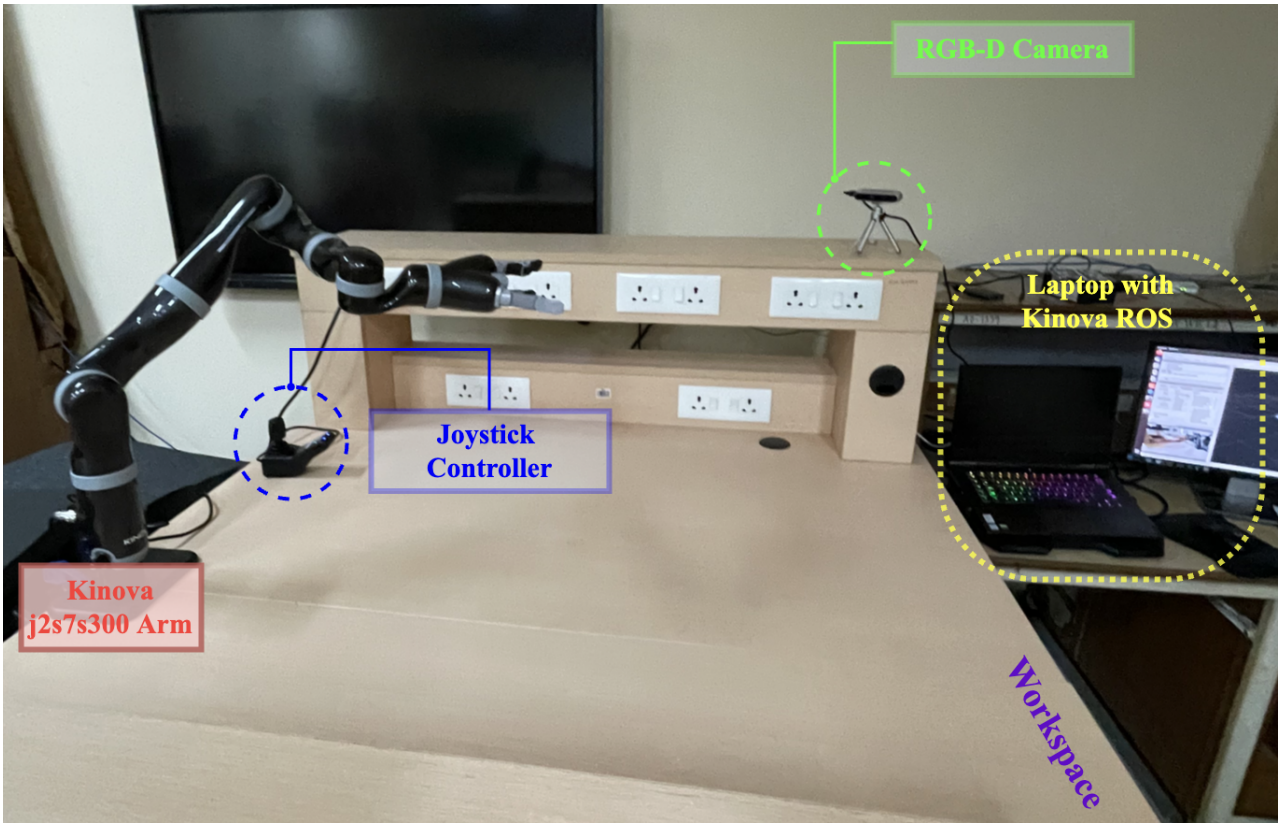


Fig. 6.8: "Real" robotic grasping setup made in IIST's CVVR Laboratory. Note that, the workspace is naturally constrained on both the sides by the wooden barriers of the table (one of them is visible in the image with the switch slots). This setup has been used for Hand-Eye Calibration.

In this work, we consider the Eye-on-Base setup as shown in Fig 6.8. As pictorially represented in Fig. 6.9, when we move the end effector to two different poses, such that the marker can be seen by the camera in both the cases. It can be observed that in both the cases, the transformation between the camera and the robotic arm's arm remains fixed.

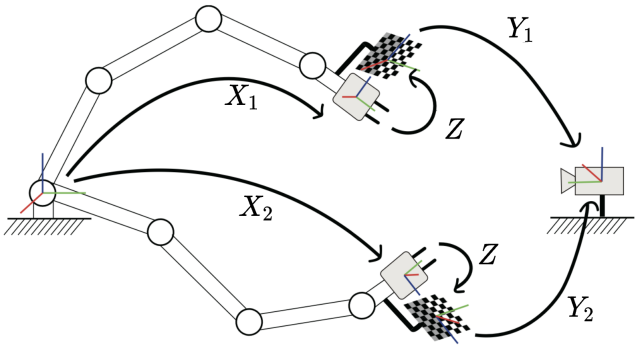


Fig. 6.9: Illustration for understanding the Hand-Eye Calibration using the Eye-on-Base setup. Adapted from [30].

Using this fact, we can write the following equations:

$$X_1 \cdot Z \cdot Y_1 = X_2 \cdot Z \cdot Y_2$$
$$\Rightarrow \quad \left(X_2^{-1} \cdot X_1\right) \cdot Z = Z \cdot \left(Y_2 \cdot Y_1^{-1}\right)$$

where

$X_p \longrightarrow$ Transformation from Robotic Arm base to the end effector for the p^{th} sample.

$Y_q \longrightarrow$ Pose of the camera in the Marker coordinate system for the q^{th} sample.

$Z \longrightarrow$ Transformation from the Robotic Arm's end effector to the marker.

Hence, if we solve for Z, we can equivalently the coordinate transformation from the robotic arm's base to the camera, which is of interest to us.

One way to solve this problem is to move the end effector of the robotic arm to various sufficiently distinct poses and then create a dataset with (X_i, Y_i) pairs. This can later be given to an $AX = XB$ solver such as [31] to get the desired transformation.

6.3.3 Solution Approach

MoveIt's HandEye Calibration package[¶] was used for performing the calibration using the Kinova `j2s7s300` arm (See Section 6.1) and Intel Realsense `D-415` RGB-D camera (See Section 6.2), in the setup shown in Fig. 6.8. Note that, the Aruco Board shown in Fig. 6.12 (a) was used as the marker.

The following is a summary of the procedure followed for performing the Hand-Eye Calibration:

1. Move the robotic arm to a unique pose.
2. Register the end-effector pose.
3. Image the calibration object (marker) and obtain its pose.
4. Repeat the above three steps 10-15 times. In this work, these 11 samples were collected.
5. Compute hand-eye transformation using the solver [31].

[¶] https://github.com/ros-planning/moveit_calibration

Fig. 6.10: A glimpse of the Setup during the Hand-Eye Calibration. Note that, the Aruco board is held in the Kinova arm's gripper. The robotic arm is in a such a pose that the marker is visible to the RGB-D camera.

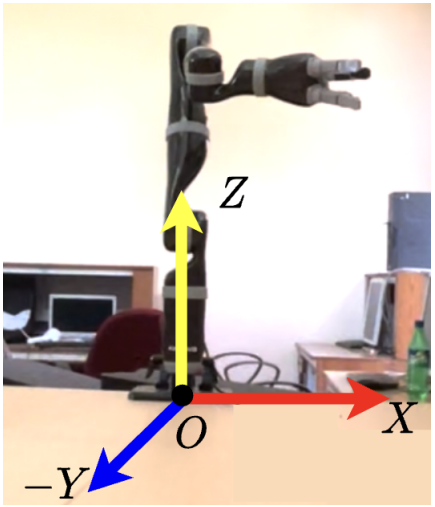
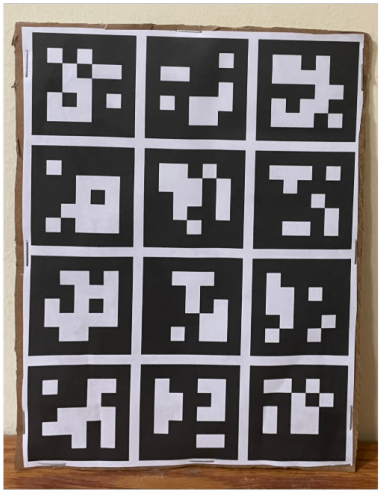
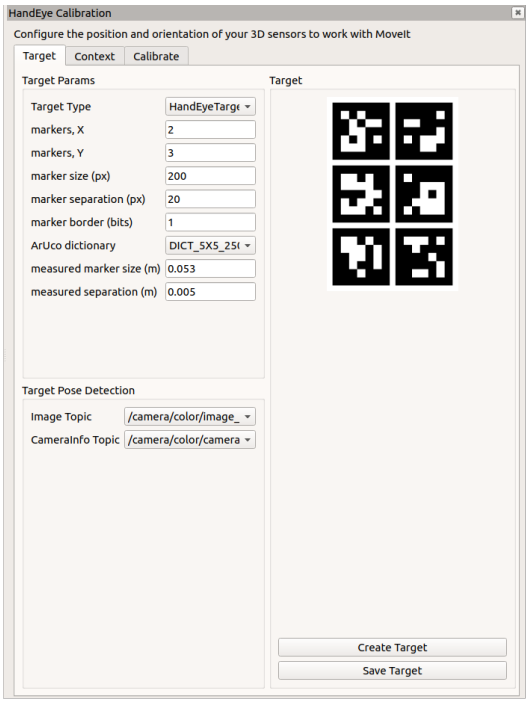


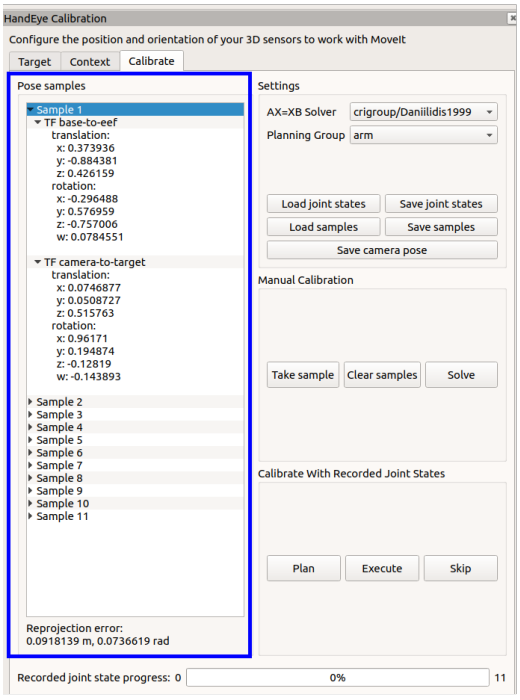
Fig. 6.11: Illustration of the Robot Base Coordinate Frame. Note that $-Y$ is the direction pointing out of the image.



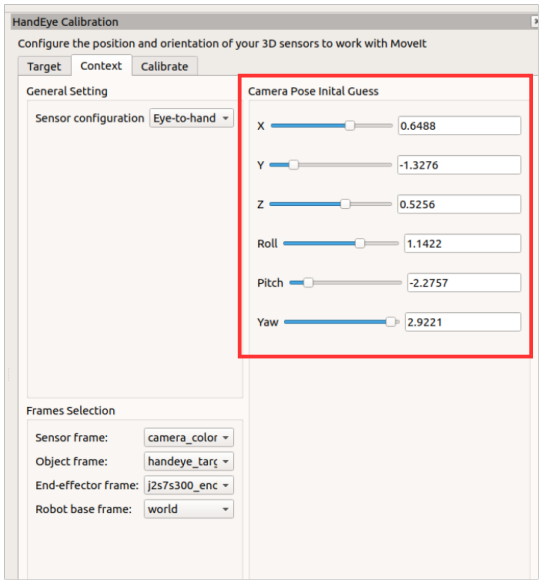
(a) Real Aruco Marker mounted on a cardboard



(b) Tab showing details of the Target



(c) Tab showing details of sample pairs collected (Here, 11)



(d) Tab showing details of the Final Calibration Result

Fig. 6.12: Various Stages of Hand-Eye Calibration experiment carried out in IIST's CVVR Laboratory. Note, the final robotic arm's base to camera transformation highlighted in red on the bottom right.

6.4 Blind Pick and Place

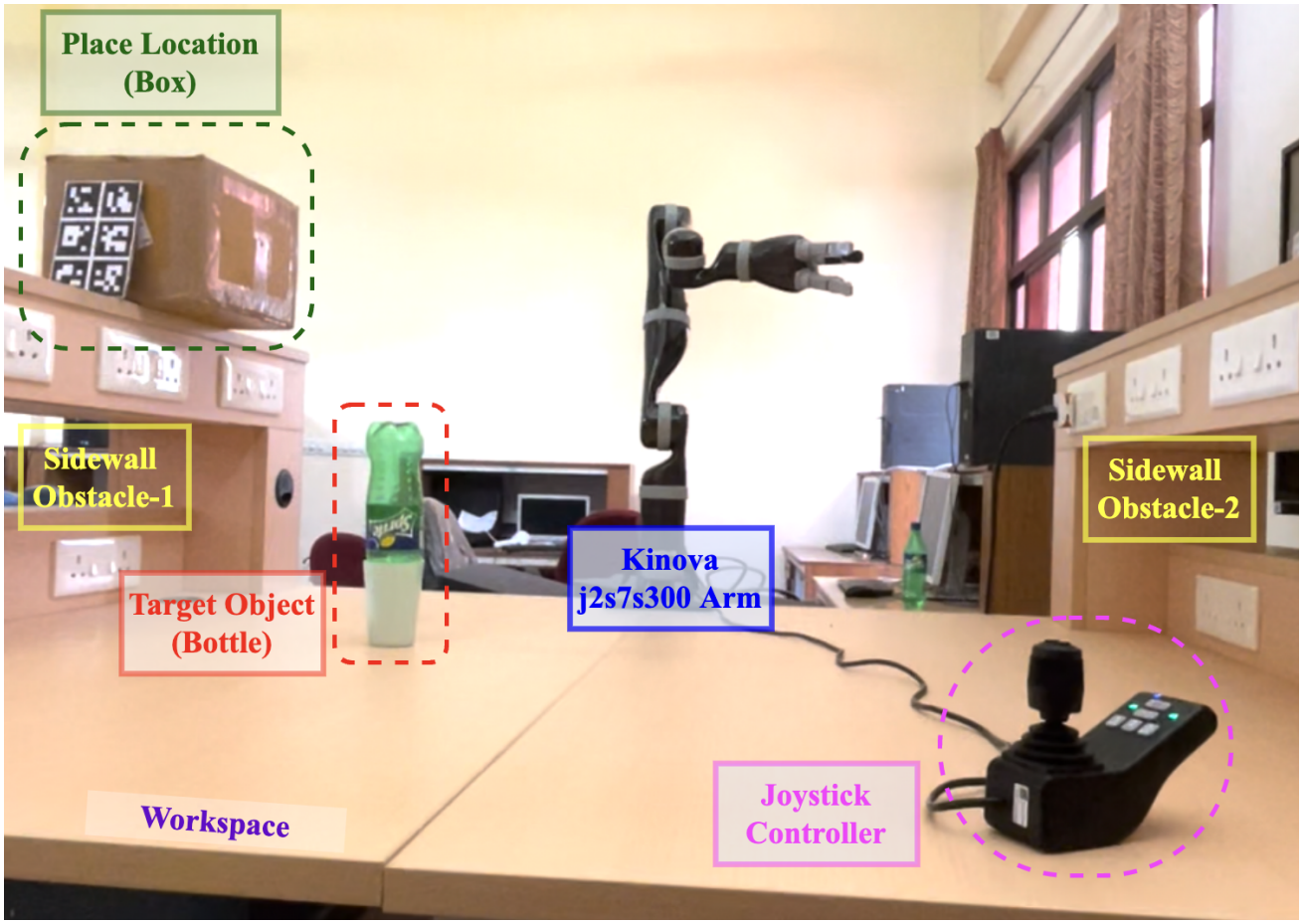


Fig. 6.13: "Real" robotic grasp setup in IIST's CVVR Laboratory, adapted for the Blind Pick and Place Task. Note that, the workspace is naturally constrained on both the sides by the wooden barriers of the table (Called Sidewall Obstacle). The Joystick Controller is kept in close proximity to execute manual stop in case of an emergency.

One of the most common and useful application that a robotic arm can be used for is to *Pick* an object of interest from a particular location and *Place* it at the desired location. In this work, we perform the Blind Pick and Place task using the Kinova `j2s7s300` arm, i.e. Pick and Place task is done without using any perception module. This is done by feeding the system with predetermined *Pick* and *Place* locations. Note that, an immediate extension to this work can be making the robotic arm *intelligent* by bringing a perception sensor such as the Intel Realsense `D415` camera.

6.4.1 Different Stages of the Task

The task of blind pick and place can be explained in the following chronological procedure:

1. Move to the *HOME* pose.
2. Move to the *Pre-Grasp* pose (backed off by some distance from the *Grasp* Pose).
3. Move to the *Grasp* Pose. Here the predetermined grasp pose is as follows (in Robotic Arm's Base Coordinate System):

Position : $(-0.2962, -0.5724, 0.1354)$; Orientation Quaternion : $(0.7158, -0.0893, 0.0541, 0.6904)$

4. Close the fingers of the gripper to grasp the Object of Interest. Here, the object of interest considered is a *Bottle*.
5. Move to the *Post-Grasp* pose. Here, considered same as the *Pre-Grasp* Pose.
6. Move to the *Place* pose. Here the predetermined place pose is as follows (in Robotic Arm's Base Coordinate System):

Position : $(-0.7770, -0.1960, 0.7189)$; Orientation Quaternion : $(0.5589, -0.3802, -0.2864, 0.6790)$

7. Open the fingers of the gripper to release the object of interest in the desired Place Location. Here, the Place Location considered is a cardboard *Box*.
8. Move to *Post-Place* pose.
9. Finally, move back to the *HOME* pose.

6.4.2 Modelling the Sidewall Obstacles

Note that, as shown in Fig. 6.13 there exist two sidewall obstacles on opposite sides of the robotic arm. We need to take into account and *inform* the robotic arm about their presence to avoid any collisions while executing motion trajectories. The MoveIt's Motion Planning Tab is used to do so as shown in Fig. 6.14.

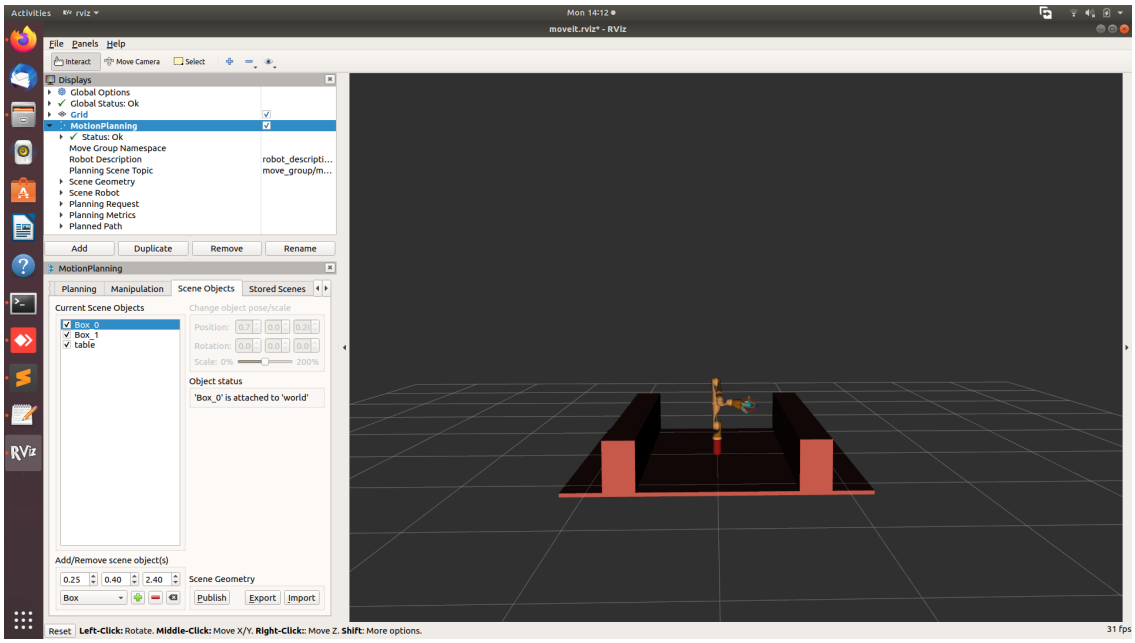


Fig. 6.14: Modelling the Sidewall Obstacles to avoid collisions. Note that the side wall obstacles are modelled as boxes as shown.

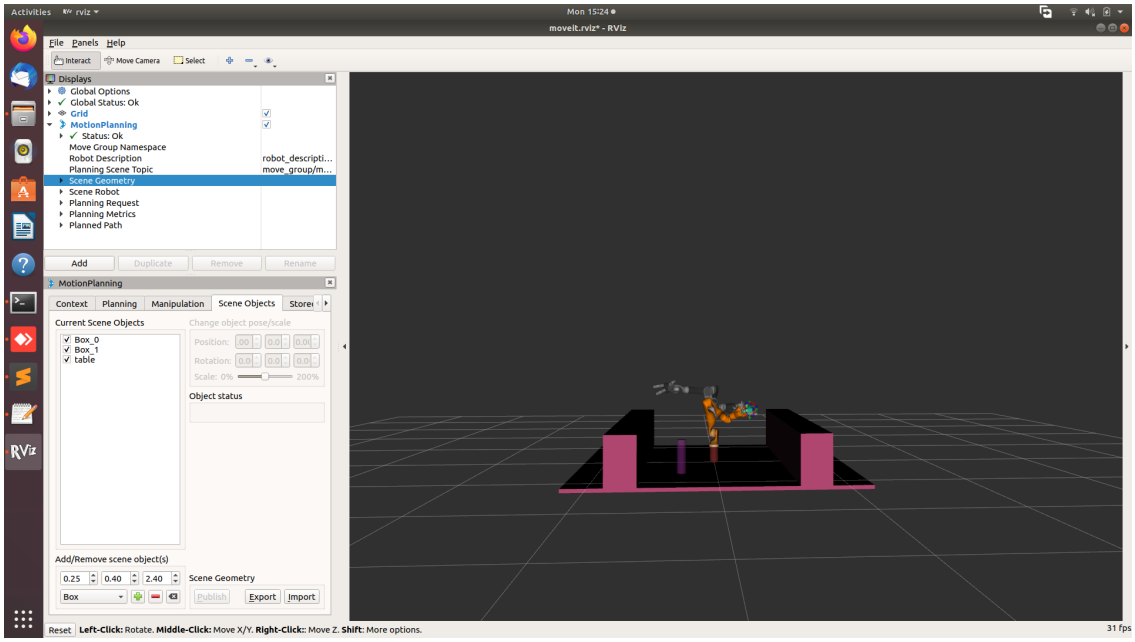
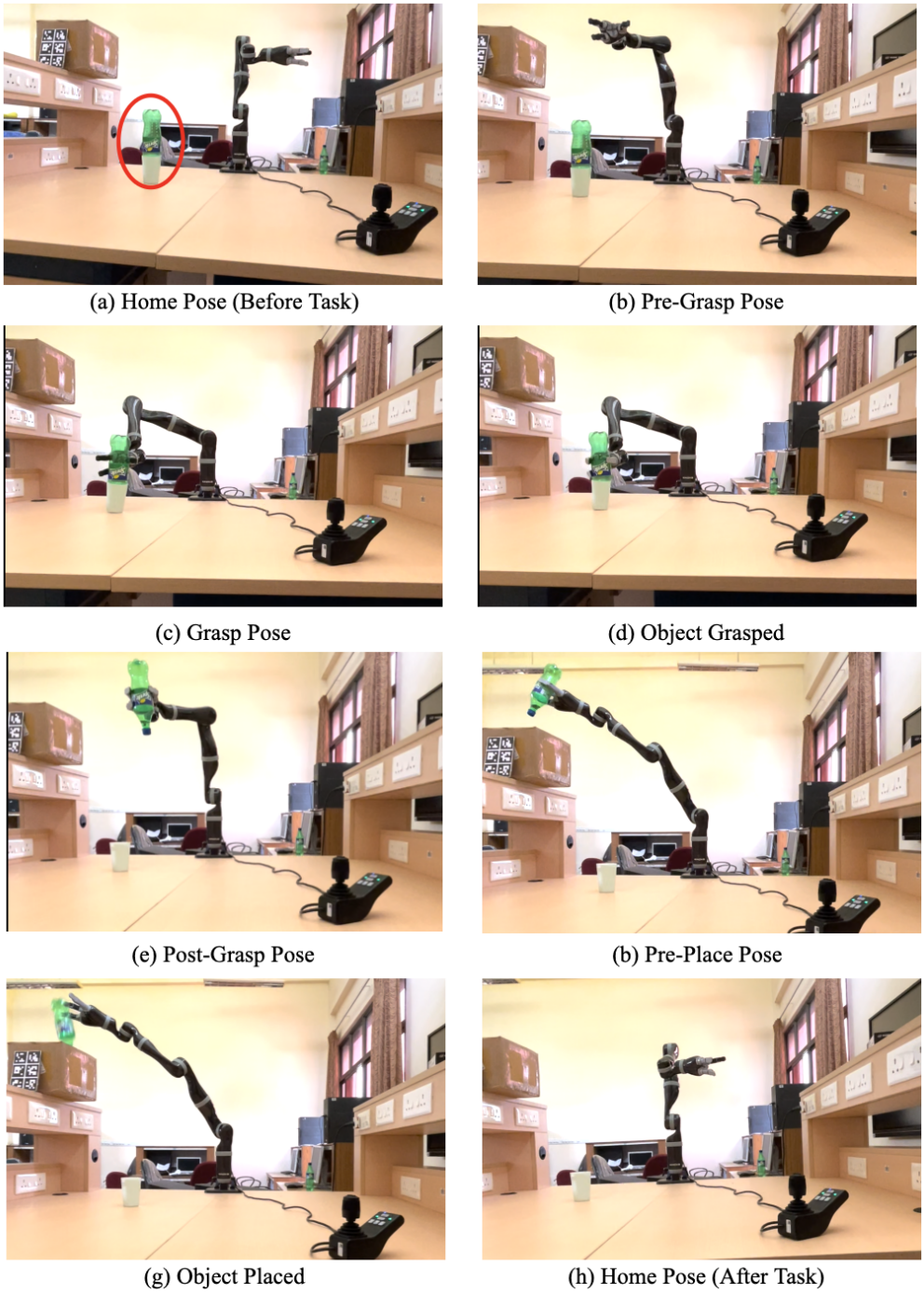


Fig. 6.15: A Glimpse of the RViz screen during the execution of the Blind Pick and Place Task. Note that, the object of interest *Bottle* is modelled as a cylinder as shown.

6.4.3 Pictorial Demonstration



(a) Home Pose (Before Task) (b) Pre-Grasp Pose

(c) Grasp Pose (d) Object Grasped

(e) Post-Grasp Pose (b) Pre-Place Pose

(g) Object Placed (h) Home Pose (After Task)

Fig. 6.16: Pictorial Demonstration of the Blind Pick and Place Task. The keyframes corresponding to various steps of the task are shown in a chronological manner, in real robotic setup of IIST's CVVR Laboratory with a Kinova `j2s7s300` arm. Note that the object of interest is circled in the first frame.

6.5 Chapter Summary

This chapter specifies the various tasks performed for setting up a "real" robotic grasping setup while discussing its challenges. The complete "real" robotic grasping setup mainly consisting of a Kinova `j2s7s300` arm and Intel Realsense `D415` camera. Later, details about the successful demonstration of various tasks such as Hand-Eye Calibration & Blind Pick and Place in the constrained robotic workspace of IIST's CVVR Laboratory were given.

Chapter 7

Conclusion and Future Work

This chapter includes a conclusion to this report on *Autonomous Robotic Grasping* and also provides interesting future work ideas for pursuing further research.

7.1 Conclusion

This report on "Autonomous Robotic Grasping" was aimed at developing effective techniques for skilled robotic manipulation tasks. A brief study of relevant ARG approaches was discussed in Chapter 2. In this work, specifically two intricate ARG tasks were studied extensively, namely :

- *Task I - Grasping Various Objects in Diverse Objects*, described in Chapter 3, involved development of effective DRL algorithms for training a robotic arm agent to learn how to grasp novel objects in random environments.

- *Task II - Dynamic Grasping of Moving Objects*, described in Chapter 4, required the design of a systematic approach for grasping "known" objects traversing a motion trajectory that is not known apriori.

As a part of both tasks, a series of effective Deep Learning algorithms were developed for accomplishing various subtasks embedded in the main task. The results obtained on testing the developed techniques are presented along with inferences in Chapter 5 for both tasks.

In Task I, it was shown through various experiments that in addition to improving the RL algorithm side of DRL techniques, it might be beneficial to do research for developing advanced DNN architectures. In this work, considering the improvements made in the architectures of Feature Extractor (See Chapter 3), the best performing model was the *O-AHRNet* model, which used repeated multi-depth octree fusion features (to maintain high-resolution representations) and an attention module consisting of the channel and spatial attention. Using *O-AHRNet* model as the Feature Extraction Backbone, the RL agent was able to achieve more than 87% success rate for grasping novel objects in random scenes, showing its efficacy. Other experiments have motivated the need for designing novel RL aware normalization techniques and effective architectures for Actor-Critic Heads.

As a part of Task II, it was discovered that the usage of DL models could prove useful for predicting the motion of dynamic known objects. In this work, considering the improvements made in the Object Pose Prediction component (See Chapter 4), the best performing model for objects moving in Sinusoidal non-linear trajectories was the *Long Short Term Memory (LSTM)* model. Incorporating this model into the dynamic grasping pipeline, the average success rate obtained for grasping dynamic objects in Sinusoidal, random motion was more than 75%, showing the power of LSTMs to model complex non-linear interactions in sequence to sequence problems. Some of the results have hinted at combining DL methods with non-DL methods, such as Kalman Filter, to get better future pose estimates, thereby improving dynamic grasping.

Furthermore, in this work, various challenges and tasks, such as Hand-Eye Calibration & Blind Pick and Place, that are involved in setting up a real-world robotic grasping system, were explored in Chapter 6 along with results and demonstrations conducted in IIST's CVVR Laboratory. These motivated the use case of having a perception-based intelligent grasping system for performing skilled manipulation tasks.

Moreover, such intelligent ARG systems with the ability to efficiently manipulate objects can prove to be invaluable in challenging space environments. They can be incorporated into space missions to perform various intravehicular and extravehicular activities efficiently. For example, a potential application can create a system to perform maintenance or repair tasks independently or cooperate in space with an astronaut. Integrating such technologies in future space missions can increase exploration productivity, maximize performance and improve scientific return.

7.2 Future Work

Some of the directions for future work, both immediate and long term, that can be pursued further as an extension of this project are as follows :

- *Task-1 : Grasping Various Objects in Diverse Environments* –

  → Currently most the DRL research is focused on improving the algorithmic part of the approach. But as shown in this work*, development of effective DL models can also lead to significant improvemnts in the agent's performance. Hence, improving the architectures of various DL models further can be a interesting idea for extension to this project.

  → As shown in Chapter 5 Section 5.1 (specifically Comparison-5 and Comparison-6), the performance of the agent is not that great when architectures of actor and critic network are modified. So, there is a need for performing more experiments in different combinations and for longer to isolate the problem (either in *Dense Residual*, *Duelling Critic*, *Spectral Normalisation* or any combination of them). This may lead to some exciting insights.

  → Also, as shown in Comparison-3 of Chapter 5 Section 5.1, the use of BatchNorm layer in the *Basic Unit* (Refer Chapter 3) reduced the performance of the agent drastically. More research and experimentation can done for finding what an effective normalization strategy can be for DRL algorithms.

  → Furthermore, incorporating the Adversarial framework inside the DRL setup can prove to be useful. More work needs to be done on how one can formulate this.

- *Task-2 : Dynamic Grasping of Moving Objects* –

  → In Chapter 4 Section 4.7, different methods to predict the future pose of a dynamic target were discussed. Here, either DL methods (MLP model and

*(Refer Approach from Chapter 3 and corresponding results from Chapter 5)

LSTM model) or other methods such as Kalman Filter were described. One possible extension to this work is to design a method that uses both of these approaches [32] [33] constructively to get better estimates.

→ In this work, the dynamic grasping algorithm was designed using various components such as *Object Pose Retrieval*, *Object Pose Prediction*, *Grasp Database*, *Grasp Ranking Functions* and *Adaptive Trajectory Synthesis* (See Chapter 4 Section 4.3). Most of these involved usage of DL and IK methods. One possible future work is to design an RL framework that takes RGB-D or RGB observations of the scene at regular intervals of time and then learns a policy to perform *Dynamic Grasping*.

- *"Real" Robotic Grasping Setup* –

→ One immediate extension of this work can be incorporating perception-based intelligence into the Blind Pick and Place task (See Chapter 6 (See Section 6.4). Specifically a perception sensor such as Intel Realsense `D415` RGB-D camera can be used with Object Pose Estimation algorithms such as DOPE [2] to automate the grasping workflow. An example of this type of work can be found in [34].

→ An interesting application that can be developed is to mount the robotic arm on a mobile platform and then perform *Mobile Manipulation*. Indoor Navigation techniques can be used to move the mobile base in an known environment. An example of this type of work can be found in [30].

Bibliography

[1] G. Du, K. Wang, S. Lian, and K. Zhao, "Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review," *Artificial Intelligence Review*, vol. 54, no. 3, pp. 1677–1734, 2021.

[2] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," *arXiv preprint arXiv:1809.10790*, 2018.

[3] I. Akinola, J. Xu, S. Song, and P. K. Allen, "Dynamic grasping with reachability and motion awareness," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 9422–9429.

[4] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[5] K. Zakka, A. Zeng, J. Lee, and S. Song, "Form2fit: Learning shape priors for generalizable assembly from disassembly," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9404–9410.

[6] D. Morrison, P. Corke, and J. Leitner, "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach," *arXiv preprint arXiv:1804.05172*, 2018.

[7] O. Kroemer, S. Niekum, and G. D. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *Journal of machine learning research*, vol. 22, no. 30, 2021.

[8] A. Orsula, "Deep Reinforcement Learning for Robotic Grasping from Octrees," Master's thesis, Aalborg University, 2021.

[9] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-cnn: Octree-based convolutional neural networks for 3d shape analysis," *ACM Transactions On Graphics (TOG)*, vol. 36, no. 4, pp. 1–11, 2017.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

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

[13] A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov, "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5556–5566.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[15] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[16] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "Eca-net: Efficient channel attention for deep convolutional neural networks," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11 531–11 539, 2020.

[17] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep high-resolution representation learning for human pose estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5693–5703.

[18] N. Bjorck, C. P. Gomes, and K. Q. Weinberger, "Towards deeper deep reinforcement learning with spectral normalization," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[19] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[20] M. Wu, Y. Gao, A. Jung, Q. Zhang, and S. Du, "The actor-dueling-critic method for reinforcement learning," *Sensors*, vol. 19, no. 7, p. 1547, 2019.

[21] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[22] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[23] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[24] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 489–494.

[25] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[26] I. Akinola, J. Varley, B. Chen, and P. K. Allen, "Workspace aware online grasp planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2917–2924.

[27] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in *2015 International Conference on Advanced Robotics (ICAR)*, 2015, pp. 510–517.

[28] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set," *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[29] A. B. (www.kalmanfilter.net), "Online Kalman Filter Tutorial."

[30] J. Hou, Y. Zhang, A. Rosendo, and S. Schwertfeger, "Mobile manipulation tutorial," 2020.

[31] K. Daniilidis, "Hand-eye calibration using dual quaternions," *The International Journal of Robotics Research*, vol. 18, no. 3, pp. 286–298, 1999.

[32] Z. Shi, "Incorporating transformer and lstm to kalman filter with em algorithm for state estimation," *arXiv preprint arXiv:2105.00250*, 2021.

[33] T. Bao, Y. Zhao, S. A. R. Zaidi, S. Xie, P. Yang, and Z. Zhang, "A deep kalman filter network for hand kinematics estimation using semg," *Pattern Recognition Letters*, vol. 143, pp. 88–94, 2021.

[34] S. Rakhimkul, A. Kim, A. Pazylbekov, and A. Shintemirov, "Autonomous object detection and grasping using deep learning for design of an intelligent assistive robot manipulation system," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 2019, pp. 3962–3968.