Indian Institute of Space Science and Technology

Thiruvananthapuram

B Tech Internship Project

A

Report on

Pose Estimation
for
Autonomous Robotic Grasping

Submitted by

Sri Aditya Deevi

SC18B080

12 November, 2021

Department of Avionics

Certificate

This is to certify that the internship report titled ***Pose Estimation for 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 **Electronics and Communication Engineering (Avionics)** is a bona fide record of the original work carried out by him under my supervision. The contents of this internship 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

Dr. Deepak Mishra

Head of Department

ECE (Avionics)

Place: Thiruvananthapuram

Date: 12 November, 2021

Declaration

I declare that this project report titled ***Pose Estimation for Autonomous Robotic Grasping*** submitted in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Electronics and Communication Engineering (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: 12 November, 2021

Sri Aditya Deevi
SC18B080
ECE (Avionics)

Acknowledgements

.

I would like to thank my adviser, guide and mentor, **Prof. Deepak Mishra**, for his constant support and encouragement throughout the internship. 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.

.

Abstract

The ability of a robot to sense and "perceive" its surroundings to interact and influence various objects of interest by grasping them, using vision-based sensors, is the main principle behind vision based *Autonomous Robotic Grasping*. Incorporating such a crucial ability to grasp an object in a robotic arm, to perform certain activities, can be highly beneficial across a wide variety of domains. For example, industrial robots can be used for assisting human professionals in performing versatile and repetitive processing tasks such as pick-and-place, assembly, glue dispensing, material finishing, packaging, material removal, and quality inspection; whereas domestic robots can provide support to elderly or disabled people for their day to day grasping tasks.

In order to realize this task of autonomous object grasping, we can partition the entire task into a set of key sub-tasks. Execution of these sub-tasks properly leads to the final desired outcome. One of these critical sub-tasks is *object pose estimation*, which forms the central focus of this report. It involves estimating the pose of known objects in a given environment from sensory data. The sensory data can include RGB images and data from depth sensors, but being able to determine the pose of the object by using only a single RGB image is cost effective and highly desirable in many applications.

Object Pose Estimation mainly involves determining the 6D pose, which includes a translational component and a rotational orientation component, of an object of interest. Deep Learning techniques have revolutionized the field of Computer Vision, by outperforming conventional algorithms in terms of performance. The high representational power of Convolutional Neural Networks (CNNs), to express complex mathematical relations, can be exploited to devise an effective and efficient solution for the estimation of object pose.

Contents

Acknowledgements **II**

Abstract **III**

1 Introduction **1**
1.1 Problem Statement . . . 1
1.2 Basic Description of the Simulation Setup . . . 1
1.2.1 UR3 Robotic Arm . . . 1
1.2.2 Simulation Scenarios . . . 2
1.3 Report Organization . . . 4
1.4 Key Contributions . . . 4

2 A Brief Review of Literature **6**
2.1 Typical Robotic Grasping System . . . 6
2.2 6D Pose of an Object . . . 7
2.3 Overview of Object Pose Estimation Approaches . . . 8
2.4 Relevance of Pose Estimation . . . 10
2.5 Challenges of Pose Estimation . . . 11

3 A Basic Blueprint **12**
3.1 Main Phases of the Approach . . . 12
3.2 Setting up the Robotic Arm . . . 13
3.3 Synthetic Data Collection . . . 14
3.3.1 3D BBox Labelling . . . 15
3.3.2 Camera Intrinsic Matrix . . . 16
3.3.3 Domain Randomization . . . 17
3.4 Trajectory Planning and *MoveIt* . . . 18
3.4.1 What is a Trajectory? . . . 18
3.4.2 Stages of a Pick and Place Task . . . 18
3.4.3 Generation of Motion Plan . . . 18
3.5 Pick and Place . . . 20
3.5.1 High Level Flow Diagram . . . 20
3.5.2 Elements of the User Interface . . . 21
3.5.3 A Pictorial Demonstration . . . 21

4 Pose Estimation Models **23**
4.1 Model-1 : UnityVGG16 . . . 23
4.2 Transformations between Image and Real World . . . 24
4.2.1 Perspective Projection (Real World ⟶ Image) . . . 24
4.2.2 Perspective-n-Point (PnP) Algorithm . . . 26
4.3 Model-2 : Pose6DSSD . . . 27
4.3.1 Interpreting the Feature Extraction Output . . . 28
4.3.2 Modelling the Ground Truth Confidence . . . 28
4.4 Model-3 : DOSSE-6D . . . 29
4.4.1 Backpropagatable PnP (BPnP) Module . . . 31
4.4.2 Differences among Model Versions (*v_1,v_2* and *v_3)* . . . 32
4.4.3 Attention Module . . . 33
4.5 Model-4 : AHR-DOSSE-6D . . . 36
4.5.1 Intuition behind Model Design . . . 36
4.5.2 Model Architecture Details : A High-Level Overview . . . 36
4.5.3 AHRNet Feature Extraction Backbone . . . 37
4.5.4 Differential Spatial to Numerical Transform (DSNT) Block . . . 40
4.6 Comparison of Pose Estimation Model Architectures . . . 42

5 Training Configuration and Model Evaluation Setup **43**
5.1 Experimental Configurations . . . 43
5.2 Loss Functions . . . 44
5.3 Optimizer Details . . . 46
5.4 Training-Validation Curves . . . 47
5.5 Evaluation Metrics . . . 49
5.5.1 Average Distance (ADD) Metric . . . 49
5.5.2 Reprojection Error Metric . . . 49
5.5.3 Translational MSE Metric . . . 50
5.5.4 Quaternion Error Metric . . . 50

6 Results and Inferences : Unity Simulation Scenarios **51**
6.1 Average Distance (ADD) Metric : Results . . . 51
6.1.1 Quantitative Results . . . 51
6.1.2 Graphical Results . . . 53
6.2 Translational RMSE Metric : Results . . . 55
6.2.1 Quantitative Results . . . 55
6.2.2 Graphical Results . . . 56
6.3 Reprojection Error Metric : Results . . . 58
6.3.1 Quantitative Results . . . 58
6.3.2 Graphical Results . . . 59
6.4 Quaternion Error Metric : Results . . . 61
6.4.1 Quantitative Results . . . 61
6.4.2 Graphical Results . . . 62
6.5 Inferences . . . 64

7 Benchmarking Model Performance : LINEMOD Dataset **66**
7.1 Basic Description of the Dataset . . . 66
7.2 Training Configuration Details . . . 67
7.2.1 Dataset Augmentation . . . 67
7.2.2 Experimental Setup . . . 68
7.3 Average Distance (ADD) Metric : Results . . . 69
7.3.1 Quantitative Results . . . 69
7.3.2 Graphical Results . . . 70
7.4 Translational RMSE Metric : Results . . . 72
7.4.1 Quantitative Results . . . 72
7.4.2 Graphical Results . . . 73
7.5 Reprojection Error Metric : Results . . . 76
7.5.1 Quantitative Results . . . 76
7.5.2 Graphical Results . . . 77
7.6 Quaternion Error Metric : Results . . . 79
7.6.1 Quantitative Results . . . 79
7.6.2 Graphical Results . . . 80
7.7 Performance Comparison . . . 82
7.8 Inferences . . . 83

8 Conclusion & Future Work **85**
8.1 Conclusion . . . 85
8.2 Future Work Ideas . . . 86

Bibliography **87**

Chapter 1

Introduction

1.1 Problem Statement

Autonomous Robotic Grasping is the ability of an "intelligent" robot to perceive its immediate environment and *grasp* the objects under consideration. This fundamental ability to grasp object can prove to be invaluable in various applications across a variety of domains. The aim of this project is to create a *pose estimation pipeline* where a robotic arm needs to pick and place the object under consideration, after estimating its 6D pose, in a robotic simulation environment. Deep Neural Networks such as Convolutional Neural Networks (CNNs) can be explored to perform efficient and effective object pose estimation for this *Autonomous Robotic Grasping* task in simulated cluttered scenes.

1.2 Basic Description of the Simulation Setup

In order to create an end-to-end pose estimation pipeline for performing a pick and place task *Unity Editor* for *Robotic Simulation* is utilized. In this section, some of the different aspects of the simulation setup are described briefly.

1.2.1 UR3 Robotic Arm

The UR3 is a lightweight, adaptable, collaborative industrial robotic arm manufactured by Universal Robots. In this project, the UR3 robotic arm with the *Robotiq* end effector is utilized. UR3, with a 360° rotation range on all its joints, is designed to simulate repetitive manual tasks. Some more features of UR3 are mentioned in Table 1.1.

Fig. 1.1: UR3 Robotic Arm

Feature	Value
Max. Payload	3 Kg
Weight	11 Kg
Range	500 mm
No. of Joints	6

Table 1.1: Some features of UR3 Cobot

Fig. 1.2: *Robotiq* Gripper

Robotiq gripper is a 2-finger adaptive end effector with a maximum stroke of 140 mm. For the purpose of simulation, the definition present in the URDF (Universal Robot Description Format) file is used. Unity's URDF Importer package parses the URDF file and imports it into Unity scene using PhyX 4.0 articulation bodies. Building physics articulations such as robotic arms or kinematic chains, with hierarchically organized game objects, is facilitated through Unity's *Articulation Body* package.

1.2.2 Simulation Scenarios

Unity Editor is used to create the simulation scenes, which contains the environment in which the entire simulation takes place. The scene can also be designed to generate synthetic data (see Chapter 3, Section 3.3). To analyze the robustness of the pose estimation models discussed in Chapter 4, two different simulation scenarios are considered, namely *simple scene* and *cluttered scene*, which are described below.

(i) *Simple Scene :*

The various objects present are as follows :

- Directional Light
- Virtual Camera
- Table
- Floor
- UR3 Robotic Arm with *Robotiq* gripper
- A coloured face cube – Object of Interest (no axis of symmetry)
- Goal Mat – Indicates the final place location

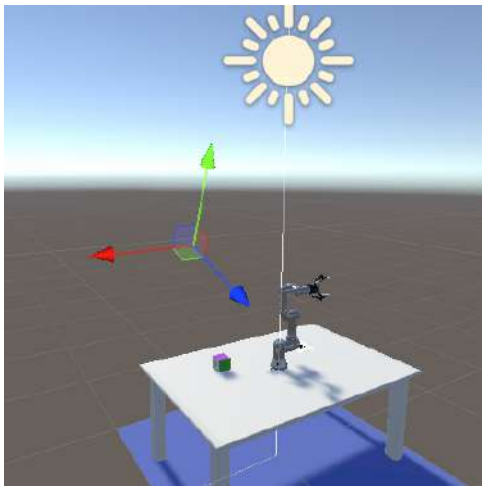


Fig. 1.3: A still from the Simple Scene – Unity Editor. The coordinate frame attached to the virtual camera is displayed. {Green Axis ↦ Y-Axis; Red Axis ↦ X-Axis; Blue Axis ↦ Z-Axis }

(ii) *Cluttered Scene :*

The Cluttered Scene is a more *challenging* version of the simple scene. All the objects present in the simple scene are a part of the cluttered scene as well. It is basically a general room environment with objects such as workbenches, paint buckets etc. Furthermore, on the table there are various real-life distractor objects along with the object of interest (see Fig. 1.5).

Fig. 1.4: A still from the Cluttered Scene – Unity Editor. This simulation scene is a representative of typical room environments with various objects placed at random positions. {Green Axis ↦ Y-Axis; Red Axis ↦ X-Axis; Blue Axis ↦ Z-Axis }

The 3D models of the distractor objects shown in Fig. 1.5 are taken from the YCB model set, provided as a part of the YCB benchmark dataset [2][3]. Also, since this work is mainly focussed on single-object & single-instance pose estimation, only one instance of the object of interest (*cube*) is present in the scene. Note that, other distractor objects can be present in multiple instances.

(a) Wooden Block (b) Cracker Box (c) Master Chef Can (d) Sugar Box

Fig. 1.5: Distractor Objects present in Cluttered Scene. These objects are typically found in real-life scenes.

Note

The 3D models are imported into the Unity scene by creating as `GameObjects`, using the Unity's *Prefab system*. This system allows the users to create, configure, and store a `GameObject` complete with all its components, property values, and child `GameObjects` as a reusable Unity Asset from 3D model files.

1.3 Report Organization

This report on "*Pose Estimation for Autonomous Robotic Grasping*" is organized as follows :

- ⋆ Chapter 2 provides a brief review of literature to setup the overall context for pose estimation approaches used in this work, which explaining relevance and typical challenges.
- ⋆ Chapter 3 serves as a simple blueprint by providing a roadmap of different aspects of this work. It helps in establishing a high-level understanding of the entire pipeline.
- ⋆ Chapter 4 provides detailed descriptions of various pose estimation models used in the pipeline, which is the primary focus of this work.
- ⋆ Chapter 5 specifies the training and testing configuration utilized for evaluating these pose estimation models based on various metrics of interest.
- ⋆ Chapter 6 includes various quantitative and graphical results along with inferences for the collected Unity Synthetic dataset.
- ⋆ Chapter 7 mainly focusses on benchmarking some of the best performing pose estimation models on a popular object pose estimation dataset LINEMOD.
- ⋆ Chapter 8 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 pose estimation pipeline using Unity and ROS Noetic, where a UR3 Robotic Arm was deployed in a simulated pick-and-place task.
- ✓ Creation of cluttered and simple simulation scenes in Unity Environment, equipped with synthetic data collection, to analyze the same-environment and cross-environment performance of the developed pose estimation models.
- ✓ Design and development of a series of convolutional neural network-based pose estimation models based on different methodologies. The model development was focussed to iteratively improve the efficiency (use of only RGB image and no depth information), accuracy (performance on relevant metrics) and speed (use of no post hoc refinement stages) of the approach.
- ✓ Incorporation of developed models in the designed pipeline demonstrating improved performance on the robotic pick-and-place task.
- ✓ Comprehensive Comparsion of the developed 6D object pose estimation models based on various design aspects.
- ✓ Extensive experimentation, analysis and inference based on obtained results, for all the developed models.

✓ Benchmarking of the developed pose estimation models on various real-life objects present in the LINEMOD dataset.

Note

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

https://drive.google.com/drive/folders/12IsTnNRi_nTKyrJbbwUtWqgDrjalL8Oe?usp=sharing

Chapter 2

A Brief Review of Literature

Autonomous Robotic Grasping aims at endowing robots with the ability to perceive and interact with the environment by performing robotic tasks such as pick and place effectively. It is a long-standing research topic that has been extensively researched over the years. It finds applications in a variety of environments such as industries, homes, laboratories, and spacecraft.

This chapter aims to introduce the main concepts of Robotic Grasping, particularly *Pose Estimation* which is an integral component of it. Relevant theory about various modules used in this work is described while reviewing a few existing methods.

2.1 Typical Robotic Grasping System

A robotic grasping system [7], similar to the system described in *Simple* and *Cluttered* simulation scenarios (see Chapter 1, Section 1.2), is illustrated in Fig. 2.1.

The major tasks involved in such a system are :

1. Object Localization
2. **Object Pose Estimation**
3. Grasp Estimation
4. Trajectory Planning and Motion Execution

The *Object Pose Estimation* task forms the central aspect of this work.

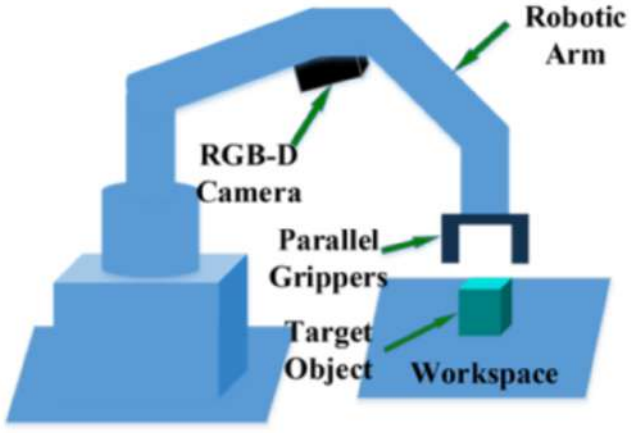


Fig. 2.1: An illustration of a robotic grasping system. Here, the robotic arm is equipped with an RGB-D camera and a gripper to pick the target object kept on the planar workspace.

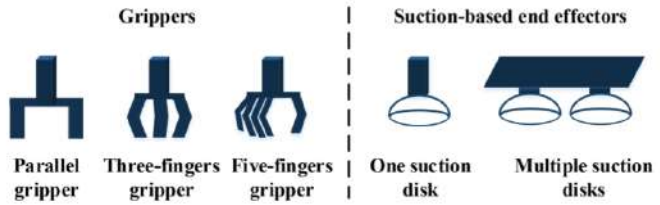


Fig. 2.2: Types of grippers for a robotic arm. In this work, parallel gripper with two fingers is used.

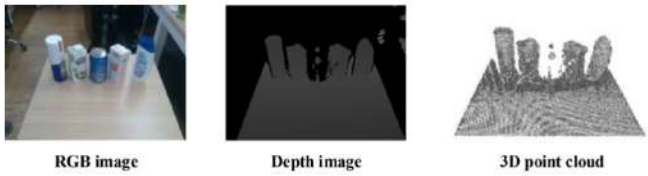
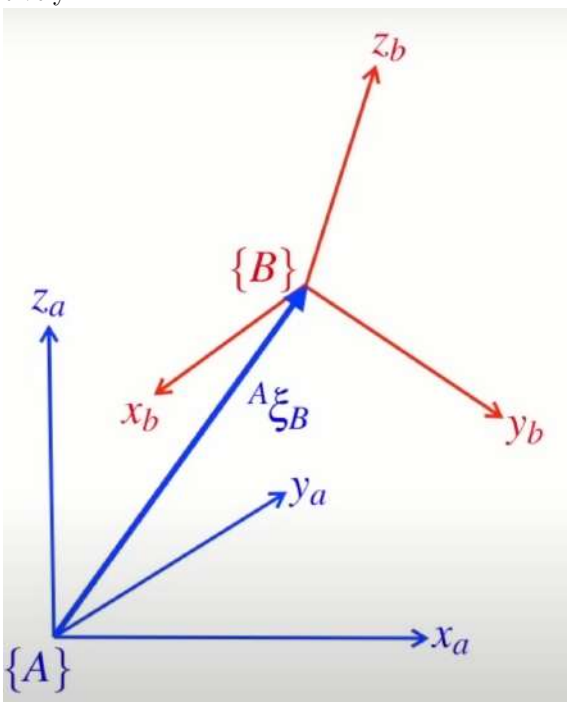


Fig. 2.3: Typical inputs available to a grasping system from a variety of visual sensors. In this work, we utilize only the RGB image assuming non-availability of any depth information.

Different systems can employ different types of grippers and different types of input data [7], as described in Fig. 2.2 and Fig. 2.3 repectively, depending upon the application and availability. In this work, the focus is on RGB image based Pose Estimation.

2.2 6D Pose of an Object

Fig. 2.4: An illustrative representation of 6D Pose of an Object [5]. The two coordinate frames associated with the definition of pose are shown as $\{A\}$ and $\{B\}$ respectively.



6D Pose of a 3D object consists of its 3D location and 3D orientation wrt a reference frame. Estimation of this quantity is crucial for robotic manipulation. Pose can be thought of consisting of a translational Component, $\mathbf{t}$ and a rotational component $\mathbf{R}$. In Fig. 2.4, $\{A\}$ denotes the reference world co-ordinate frame and $\{B\}$ denotes the co-ordinate frame attached to an object under consideration, whereas, $^A\xi_B$ represents the 6D object relative pose. The popular formats for representing $^A\xi_B$ are shown in Table 2.1.

A Note on Unit Quaternion

In general, the most convenient and efficient representation [5] to encode a 3D Rotation is that of a *Unit Quaternion* (See Table 2.1). It is represented as :

$$\mathring{q} = s < v_1, v_2, v_3 >$$
$$(or)$$
$$\mathring{q} = [x, y, z, w]$$

The axis–angle representation of a rotation parameterizes a rotation in a three-dimensional Euclidean space by two quantities: a unit vector $\hat{\boldsymbol{n}}$ indicating the direction of an axis of rotation, and an angle θ describing the magnitude of the rotation about the axis

Quaternion is a hyper-complex number used for representing relative object orientation. It has the following relation with the angle-axis representation of rotation*:

$$s = \cos\frac{\theta}{2} \quad , \quad \mathbf{v} = \hat{\boldsymbol{n}} \sin\frac{\theta}{2}$$

Format of Representation	# Parameters	Compounding Multiple Rotations
Translational Vector + Set of Euler Angles	$3+3$	Non-Trivial
Translational Vector + Roll-Pitch-Yaw Angles	$3+3$	Non-Trivial
Translational Vector + Unit Quaternion	$3+4$	Quaternion Multiplication
Homogenous Transformation (4x4 Matrix) $[R_{3\times3}, t_{3\times1}; 0, 0, 0, 1]$	$16-4=12$	Rotation Matrices can be multiplied

Table 2.1: Popular Formats for represnting relative 6D pose of a 3D object and their features. The effiency of the quaternion in terms of representing the rotation and ease of use for compounding is evident, when compared to other formats.

2.3 Overview of Object Pose Estimation Approaches

The 6D object pose transforms the object from its local co-ordinates into the camera co-ordinates, which can then be transformed into the world co-ordinates for robotic manipulation.

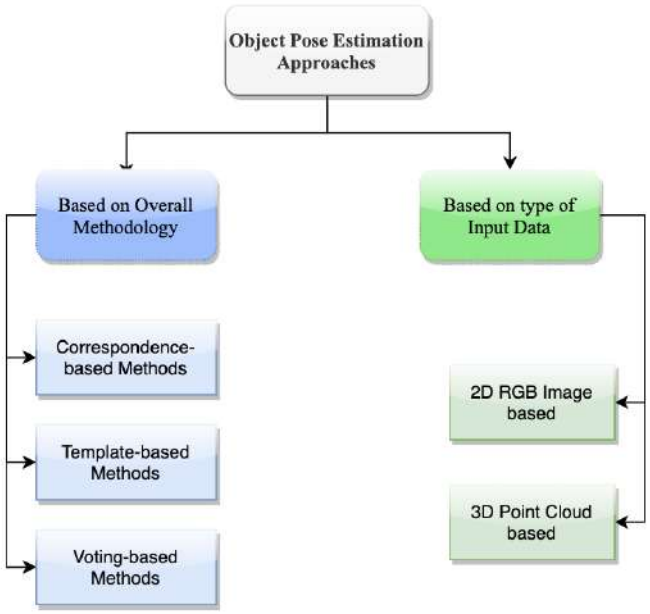


Fig. 2.5: Classification of typical Object Pose Estimation Approaches. Here, the dimensions explored for classification are the overall methodology and type of input data.

Estimating pose helps the robot to get aware of the target object. Some of the popular classes of methods typically used for this task are shown in Fig. 2.5.

Based on the overall methodology, the object pose estimation approaches can be classified into various types. Note that, each of these methods can be classified further depending upon whether depth information along with RGB data (RGB-D) is used or not. In this work, depth information is not utilized and the primary goal is RGB image based pose estimation.

Correspondence-based Methods

For 2D image based methods, this class of techniques [4][21] involves finding correspondences betweeen 2D image pixels and 3D model points using the Perspective-n-Point (PnP) algorithms (See Fig. 2.6).

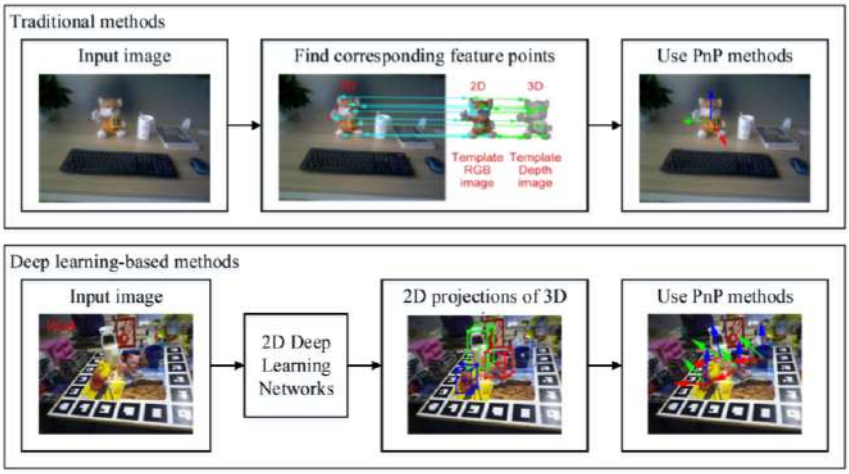


Fig. 2.6: An illustrative representation of Typical RGB based correspondence methods [7]. The objects shown correspond to the LINEMOD dataset [10]. 2D feature points of the target object are matched first, followed by establishment of 2D-3D correspondences.

This type of method can fail if the objects of interest do not have rich texture. A similar version utilizing the depth information, involves finding correspondences of 3D points between the observed partial-view point cloud and the complete 3D model.

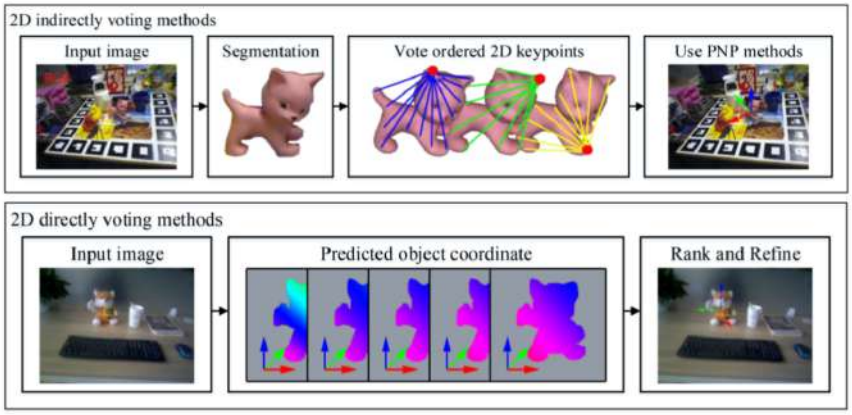


Fig. 2.7: An illustrative representation of Typical RGB based Voting methods [7]. The objects shown correspond to the LINEMOD dataset [10].

Voting-based Methods

For 2D image based methods, this class of techniques is again subclassified into indirect and direct voting methods. Indirect voting methods [14][25] (for the 2D case), can be regarded as *voting for correspondence*-based approaches, where 2D-3D correspondences are achieved by voting for 2D features or keypoints. Direct voting methods [1][18] can be similarly regarded as *voting for template*-based approaches.

Template-based Methods

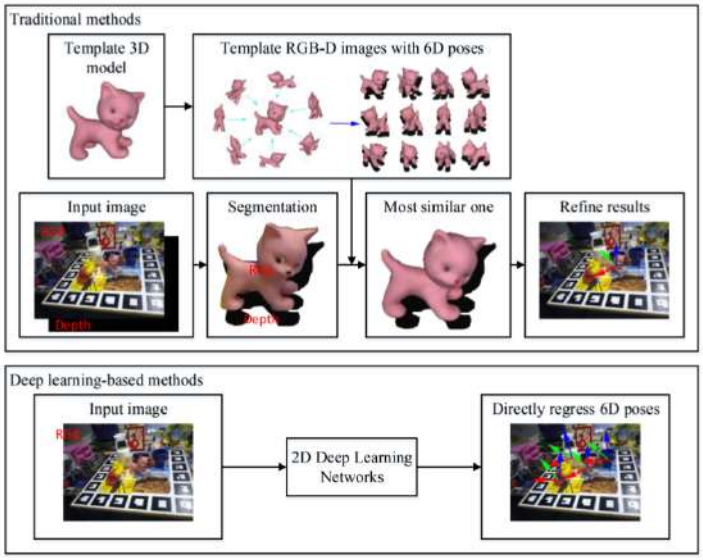


Fig. 2.8: An illustrative representation of Typical RGB based Template methods [7]. The objects shown correspond to the LINEMOD dataset [10]. Most similar template image is found either implicitly or explicitly.

For 2D image based methods, this class of techniques [20][24] is concerned with retrieving the template image, which is basically projected 2D images from known models, that is most similar with the observed input image (See Fig. 2.8). When Deep Learning is used for this task, it is reduced to directly regressing the 6D pose parameters from the RGB image. In 3D input case, the template is replaced by the full 3D point cloud of the target object.

2.4 Relevance of Pose Estimation

Apart from estimating the pose of objects for enabling autonomous robotic grasping, which is the main focus of this work, there exist many use cases, where deciphering the pose of objects can be critical. This makes the problem of pose estimation extremely relevant. Some of the typical applications requiring pose estimation as an essential component, other than autonomous robotic grasping are as follows [13] :

(i) *Human Activity Estimation* for fall detection in elderly people, body language analysis, and surveillance enhancement.

(ii) *Augmented Reality and Virtual Reality* applications such as design of realistic computer games (motion tracking consoles), graphics design for movies, and defence applications.

(iii) *Space Applications* such as satellite pose estimation and autonomous spacecraft docking maneuvering.

2.5 Challenges of Pose Estimation

Following are some of the typical challenges [13] that faced in estimating pose of an object :

1. Identifying the pose of objects that are intrinsically small in size is difficult. High resolution input images are required to be able to decipher the pose.
2. Real world settings may include cluttered environments with many background distractors objects and improper lighting conditions, which make it difficult to estimate the pose of the object of interest.
3. Occlusion of objects by other objects is also major concern. In such cases, the algorithms developed should be able to predict the pose of the object by using the information from just the partially visible keypoints.
4. Objects which contain a plane of symmetry require additional considerations while designing the pose estimation approach, as two or more object poses may have the same external visual appearance. In case of DL models, training conventionally may lead to reduction of performance due to improper feedback.
5. For data driven approaches including deep learning based algorithms, availability of data for estimating the pose might not be cheap or readily available. One of the ways overcoming this difficulty is to use synthetic data for training. But proper measures need to taken to bridge the simulation-reality gap, for good performance in real-life scenarios.
6. In many applications, there is a time constraint in estimating the pose of an object. Approaches that involve pose-prediction refinement stages tend to be slower and unsuitable for such applications.
7. Estimating the pose of objects using RGB images accurately is also a challenge. Depth information might be useful but may not be available always.

Chapter 3

A Basic Blueprint

In this chapter, a basic roadmap of the approach used for creating a *Object Pose Estimation* pipeline for performing a *Pick & Place* task will be provided. The intricacies of various components involved are revealed, analysed and explained over the course of remaining chapters.

3.1 Main Phases of the Approach

In Chapter-1, a basic description of the simulation setup was given, explaining details about the simulation scenarios and robotic arm setup. Now, building on these details, a bird's eye view of the entire approach is given in this section. The whole approach can be broadly divided into two main phases, namely : *Training Phase* and *Test Phase*. The flow diagrams for each of these phases are shown here.

Training Phase

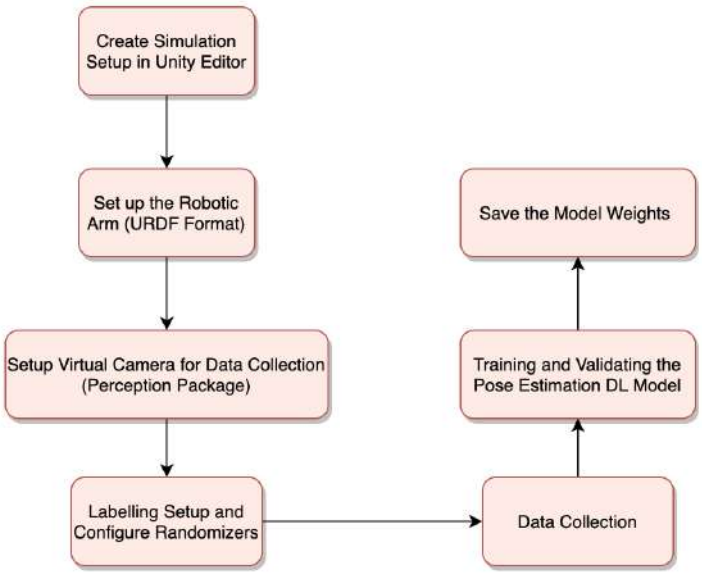


Fig. 3.1: The flowchart displaying various subtasks that are performed as a part of the training phase, in a chronological manner. In this phase, the pose estimation model is being *trained* to predict the pose of the object of interest.

Test Phase

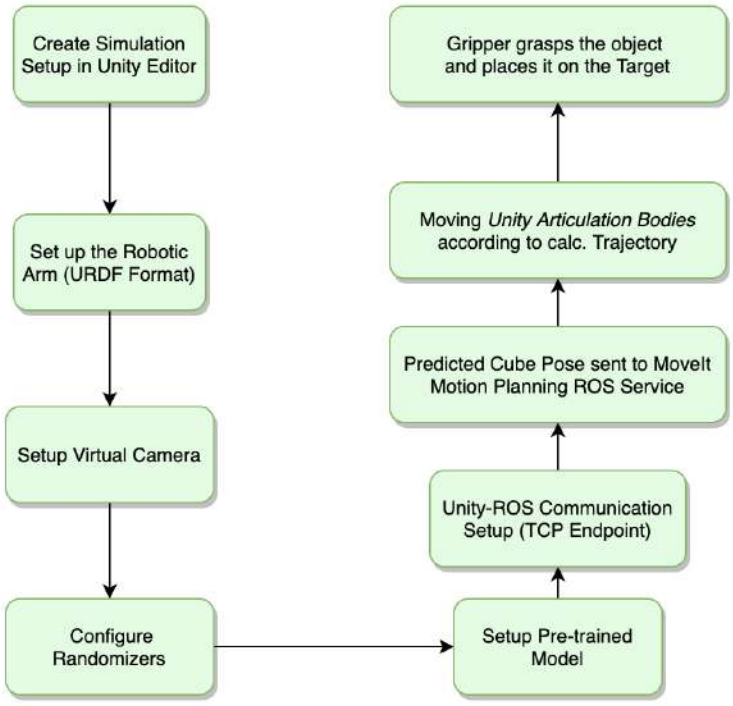
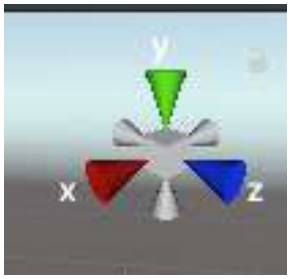


Fig. 3.2: The flowchart displaying various subtasks that are performed as a part of the test phase, in a chronological manner. In this phase, the pose estimation model is deployed and integrated into the simulated pick and place task.

These flowcharts depict the major tasks that need to be performed during each phase. More details on some of the tasks involved are provided* in the subsequent sections of this chapter.

3.2 Setting up the Robotic Arm

UR3 robotic arm with gripper, defined in URDF format, is imported into the simulation scene as a `GameObject` by using Unity's *URDF Importer* package. Some details regarding the `Controller` script of the arm, whose properties describe the physics of how robot moves, are specified in Fig. 3.4. Note that, Fig. 3.5 mentions the immovable base link parameters of the robotic arm.



Unity uses a left-handed coordinate system in which the y-axis points up (See Fig. 3.3). Therefore, the default axis needs to set as **Y**-Axis while importing the URDF files to avoid any discrepancies.

Fig. 3.3: Unity's Co-ordinate Axes

*Note that, the details about the Pose Estimation Models, which is the primary focus of this work, will be given in the forthcoming chapters, along with results and inferences.

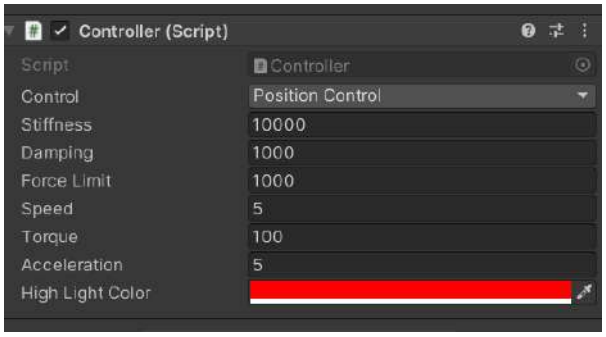


Fig. 3.4: Controller script configuration consisting of values set for various physical parameters such as stiffness, damping etc.

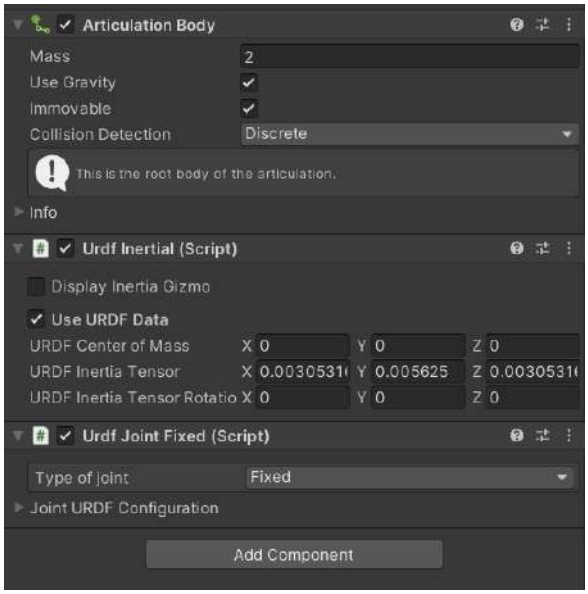


Fig. 3.5: Base Link Parameters. Note that, the base link is configured to be an immovable fixed joint.

3.3 Synthetic Data Collection

Once the simulation scenario has been created and robotic arm has configured, the virtual camera needs to be setup for Data collection. Virtual Camera is equipped with Unity's Perception Computer Vision Package, which provides support for this task. The collected data is utilized to train the pose estimation (DL) model.

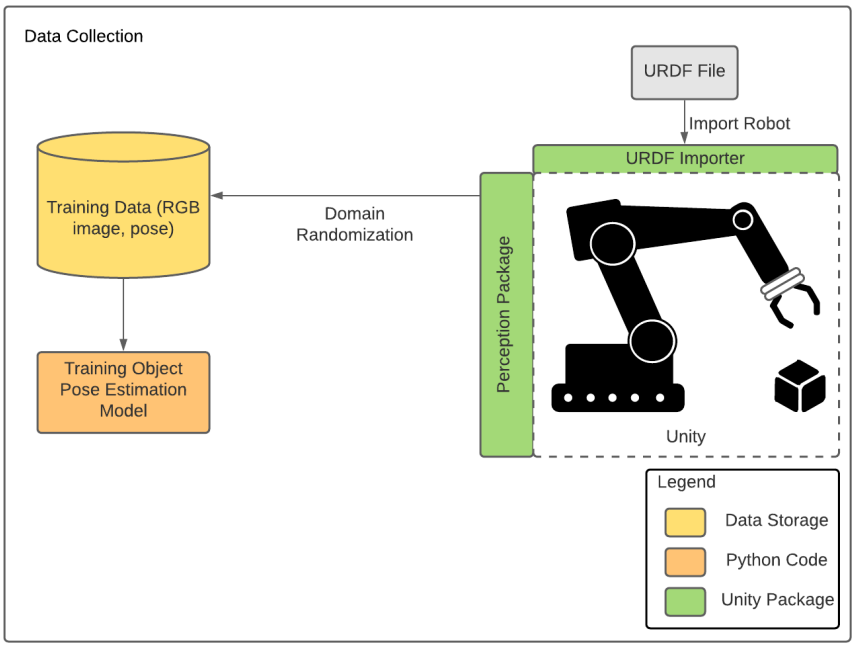


Fig. 3.6: An illustrative representation of Data Collection procedure. The collected synthetic data is utilized for training the pose estimation model to predict the object pose.

3.3.1 3D BBox Labelling

The collected data includes :

- RGB images (Here, resolution = 650 × 400 px)
- Capture Files containing the ground truth annotation information (in `.json` format)

The capture files are generated by the Unity Perception package's `3D BBox Labeller`. Each capture file consists of ground truth annotations corresponding to an instance of capture, stored according to a schema. This schema provides a generic structure for simulation output which can be easily consumed to show statistics or train machine learning models (See Fig. 3.8).

Fig. 3.7: An example of a captured RGB image by the virtual camera, in Cluttered Scene. The object under consideration (*cube*) is highlighted (green 3D BBox).


```
    },
    "filename": "RGB6318fadf-e77e-4ada-9da7-3ea85a797097/rgb_1502.png",
    "format": "PNG",
    "annotations": [
      {
        "id": "df68f078-3eec-4d83-9a1f-3deb4ffc1163",
        "annotation_definition": "0bfbe00d-00fa-4555-88d1-471b58449f5c",
        "values": [
          {
"label_id": 0,
"label_name": "cube_position",
"instance_id": 1,
"translation": {
  "x": 0.13716761767864227,
  "y": -0.23996292054653168,
  "z": 1.3521386384963989
},
"size": {
  "x": 0.10000002384185791,
  "y": 0.099999994039535522,
  "z": 0.10000002384185791
},
"rotation": {
  "x": 0.022925479337573051,
  "y": 0.976187527179718,
  "z": -0.17212818562984467,
  "w": -0.13001687824726105
},
"velocity": {
  "x": 0.0,
  "y": 0.0,
  "z": 0.0
```

```
"timestamp": 0.0,
"sensor": {
  "sensor_id": "33599f37-1c4a-4569-8b43-904491844dbb",
  "ego_id": "5bc12eee-2559-4066-a75f-ce1e22cb8865",
  "modality": "camera",
  "translation": [
    0.0,
    0.0,
    0.0
  ],
  "rotation": [
    0.0,
    0.0,
    0.0,
    1.00000012
  ],
  "camera_intrinsic": [
    [
      1.06587744,
      0.0,
      0.0
    ],
    [
      0.0,
      1.73205078,
      0.0
    ],
    [
      0.0,
      0.0,
      -1.0006001
    ]
  ]
```


Fig. 3.8: An example of a captured annotation record in `.json` format.

The interpretation of some of the annotation parameters in the file are described as follows :

```
bounding_box_3d {
  label_id:      <int>   -- Integer identifier of the label
  label_name:    <str>   -- String identifier of the label
  instance_id:   <str>   -- UUID of the instance.
  translation {          -- 3d bounding box's center location in meters with respect to the sensor's coordinate system
    x:           <float> -- The x coordinate
    y:           <float> -- The y coordinate
    z:           <float> -- The z coordinate
  }
  size {                 -- 3d bounding box size in meters
    x:           <float> -- The x coordinate
    y:           <float> -- The y coordinate
    z:           <float> -- The z coordinate
  }
  rotation {             -- 3d bounding box orientation as quaternion: w, x, y, z.
    x:           <float> -- The x coordinate
    y:           <float> -- The y coordinate
    z:           <float> -- The z coordinate
    w:           <float> -- The w coordinate
  }
  velocity {             -- [Optional] 3d bounding box velocity in meters per second.
    x:           <float> -- The x coordinate
    y:           <float> -- The y coordinate
    z:           <float> -- The z coordinate
  }
  acceleration {         -- [Optional] 3d bounding box acceleration in meters per second^2.
    x:           <float> -- The x coordinate
    y:           <float> -- The y coordinate
    z:           <float> -- The z coordinate
  }
}
```

3.3.2 Camera Intrinsic Matrix

Along with ground truth annotation pose labels, data regarding the sensor (in this case, a virtual camera) such as camera intrinsic matrix, the pose shift of the sensor w.r.t World frame, and sensor ID are also recorded (See Fig. 3.8). Unity uses a different representation for intrinsic matrix and a slightly different procedure for converting into 2D pixel co-ordinates. For convenience we can convert the camera intrinsic matrix from the unity format to the commonly used OpenCV format by denormalizing the matrix elements (w.r.t image dimensions), add the details of the optical centre and converting from Left-Handed to Right-Handed system. This can be represented by the following transformation (shown for a generic case).

$$C_{unity} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \longrightarrow C_{OpenCV} = \begin{bmatrix} \frac{-Wa}{2c} & 0 & \frac{W}{2} \\ 0 & \frac{Hb}{2c} & \frac{H}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

where $W \times H$ are the image dimensions.

In this report, the transformation done is shown below, for converting the intrinsic matrix of the virtual camera sensor :

$$\begin{bmatrix} 1.0659 & 0 & 0 \\ 0 & 1.7321 & 0 \\ 0 & 0 & -1.0006 \end{bmatrix} \longrightarrow \begin{bmatrix} 346.2024 & 0 & 325 \\ 0 & -346.2024 & 200 \\ 0 & 0 & 1 \end{bmatrix}$$

3.3.3 Domain Randomization

It is a simple technique for bridging the simulation-reality gap for synthetic data, where instead of collecting data and training a model on a single simulated environment, we randomize the simulator to expose the model to a wide range of environments at training time. The main idea involved is that if the variability in simulation is significant enough, models trained in simulation will generalize to the real world with no additional training.

Domain randomization provides an added benefit of data augmentation without redundancy. By bringing in sufficient variation into the generated data, the pose estimation model is forced to handle many small visual variations, making it more robust. In this work, this idea is used extensively, and the following custom types of randomizers[†] are configured :

- Rotation and Position Randomizer for object of interest (Here, *cube*)[‡]
- Rotation and Position Randomizer for objects present in the scene (on the table)
- Vitual Light Randomizer (Colour + Intensity + Direction)
- Camera Pose (w.r.t World Frame) Randomizer

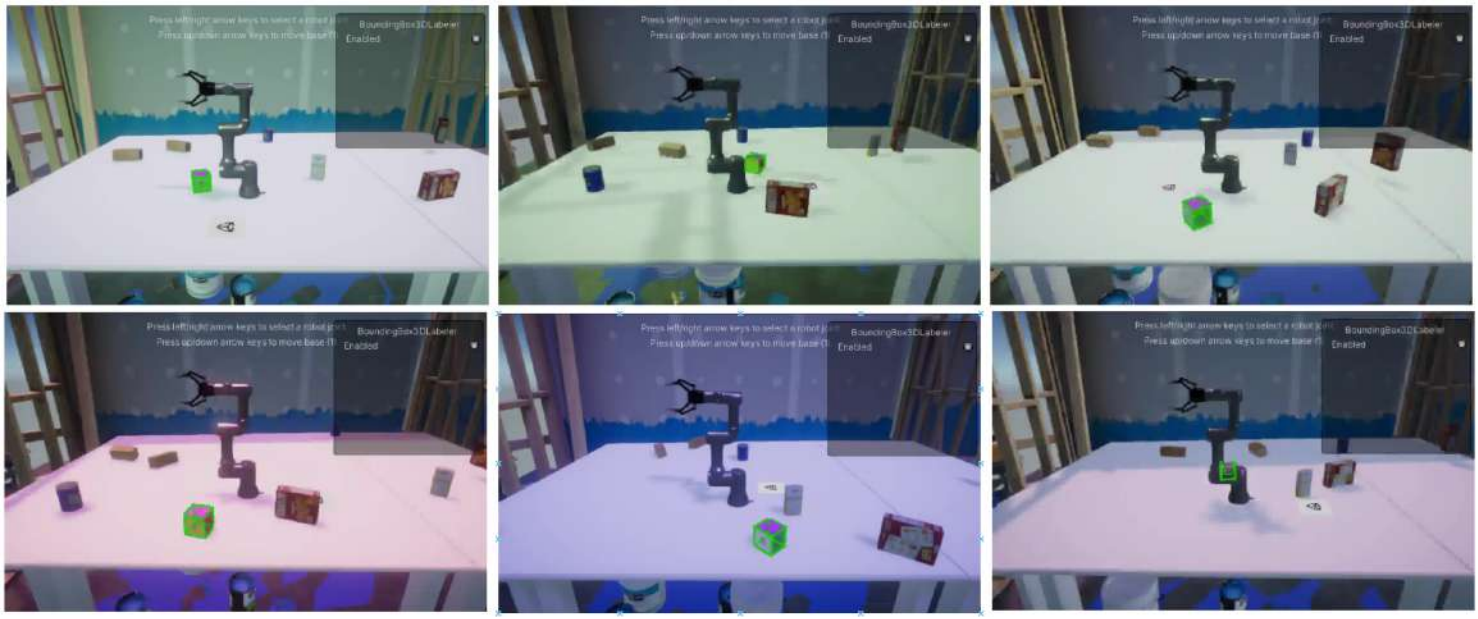


Fig. 3.9: A collection of images depicting the superimposed effect of all domain randomizers applied to the simulation scenario (specifically, here Cluttered Scene is considered)

[†]scripts are written in C#

[‡]One of the considerations here is that the object should always be within the "reach" of the robotic arm

3.4 Trajectory Planning and *MoveIt*

3.4.1 What is a Trajectory?

When the robotic arm in motion to execute a task such as pick and place, its relative pose w.r.t the world frame is a function of time. Ideally, we want the pose of the robot to vary smoothly with time with efficient use of motors, which can be made possible by trajectory planning. A trajectory consists of (See Fig. 3.10):

- a spatial construct for moving, which describes the physical path taken by the robot's end effector, and
- a schedule of motion, which is basically the time parametrization of the path.

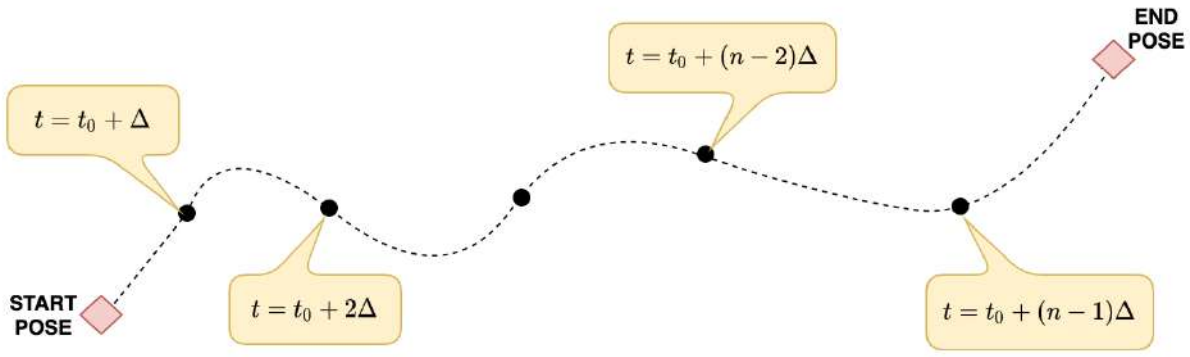


Fig. 3.10: An illustration of a trajectory. Note that it consists of a path with waypoints and time parametrization.

3.4.2 Stages of a Pick and Place Task

Specifically, in this case, the pick and place task can be divided into the following important stages[§] :

(i) *Pre-Grasp* – Involves motion plan for the robotic arm to move from initial pose to a pose convenient for grasping the object of interest (Here, a *cube*).

(ii) *Grasp* – Open the gripper (end-effector) once the robotic arm is in position, and depending on a preset gripping angle, close the gripper fingers grabbing the object securely.

(iii) *Pickup* – Keeping the gripper closed (holding the object) move the arm to a pose convenient for executing the place operation.

(iv) *Place* – Execute the motion plan for moving the robotic arm to a specified destination pose (Here, a target mat is the final pose for placing the object) and open the gripper once destination pose is achieved by the end-effector.

3.4.3 Generation of Motion Plan

There exist two popular methods for generating the motion plan for the robot by knowing the start and end poses of the end effector in each of these stages :

[§]Generally, separate trajectory plans are generated for each stage of the task

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.

MoveIt Package for Motion Planning

MoveIt 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.

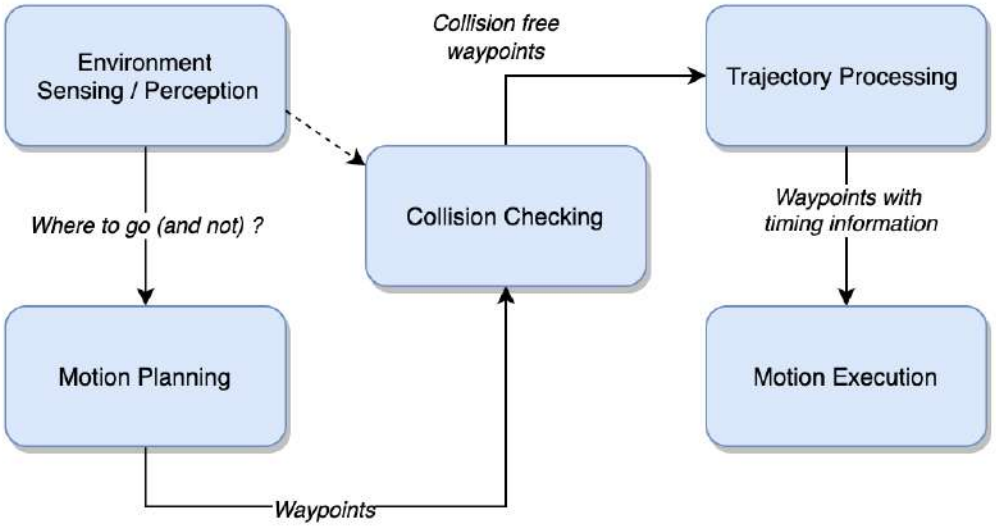


Fig. 3.11: Different Function Modules and their typical order of usage. This represents the typical flow used by MoveIt package to perform motion planning and execution.

Different packages and algorithms for each functional modules in MoveIt motion planner (See Fig. 3.12). In this pipeline, 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, MoveIt chooses between joint space and cartesian space interpolation.

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

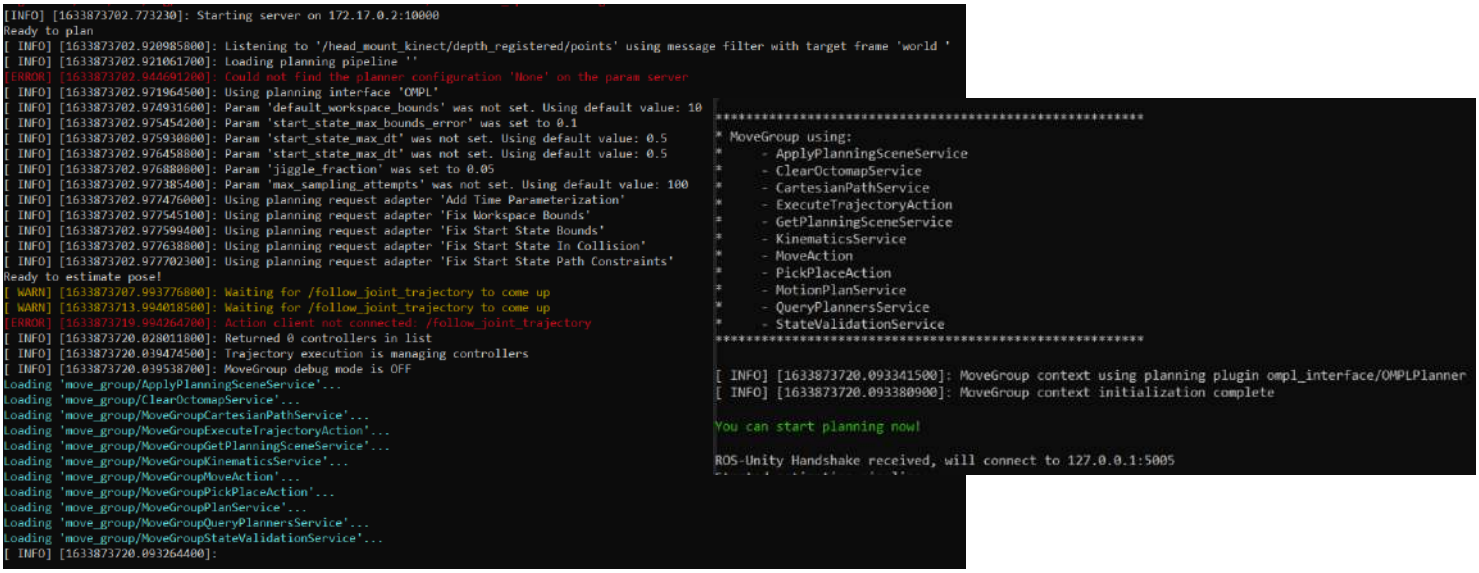


Fig. 3.12: MoveIt Motion Planner in action : Different ROS services, actions and parameters being loaded is shown. Also, note the ROS-Unity Communication through the TCP Endpoint.

3.5 Pick and Place

3.5.1 High Level Flow Diagram

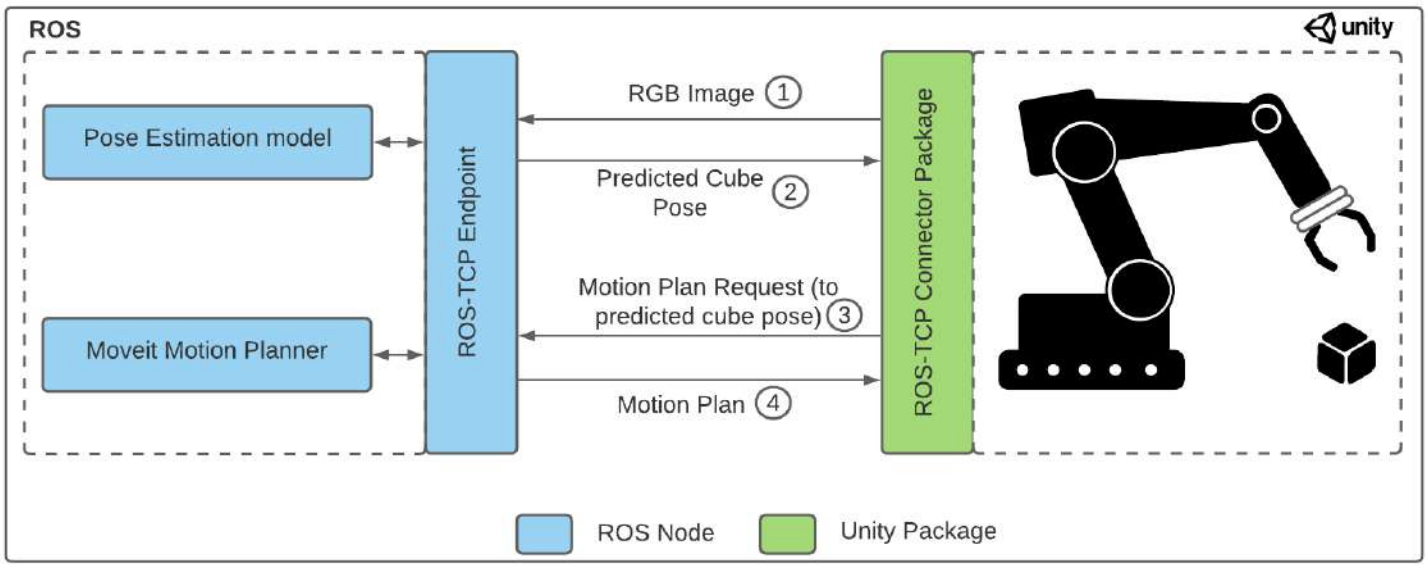


Fig. 3.13: Bird's Eye View of Pose Estimation Pipeline for object Pick and Place. All these subtasks are done in a chronological manner to realize the desired pick and place action.

A high level flow diagram for performing the complete pick and place task, using the pose estimation pipeline is shown in Fig. 3.13. The numbers ①, ②, ③ and ④ indicate the chronological sequence of events that take place during test phase.

3.5.2 Elements of the User Interface

In order to demonstrate the pick and place task using the pose estimation pipeline, a simple UI (User Interface) was designed (see Fig. 3.14).

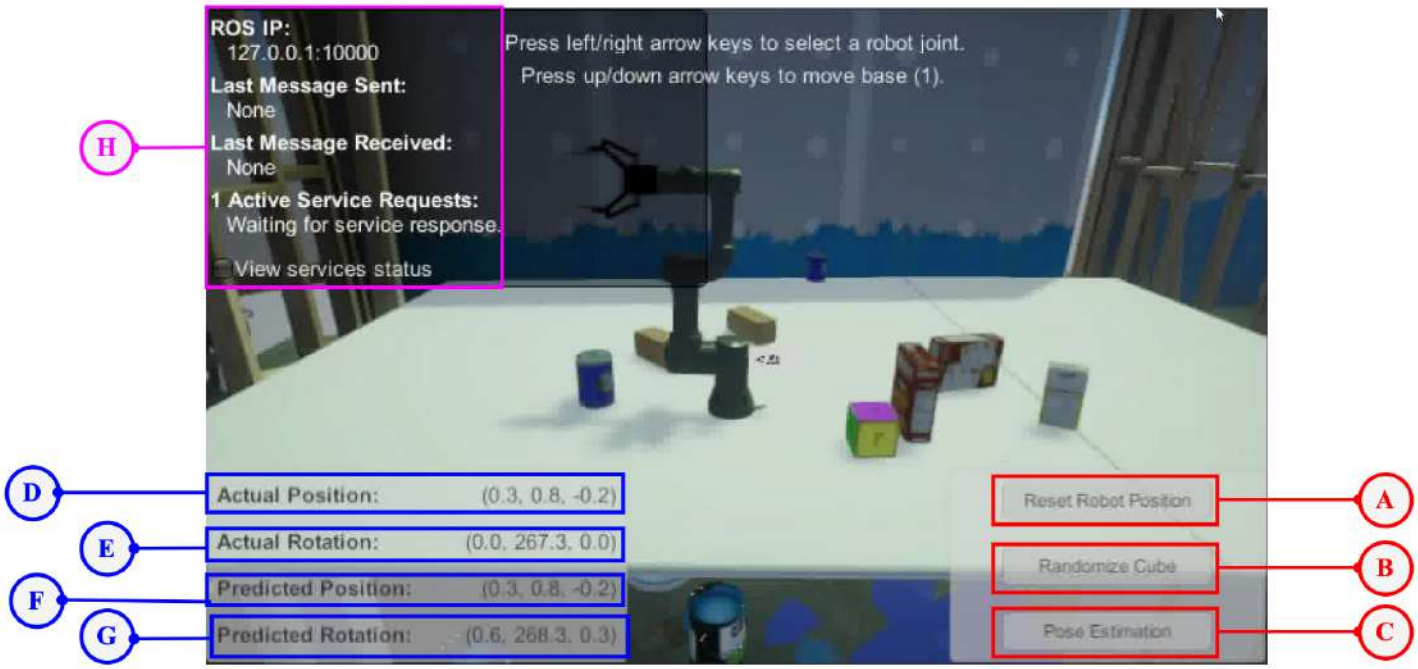


Fig. 3.14: Different Elements of Pick and Place UI. Red colour indicates control options. Blue colour indicates the display options. Pink colour indicates the status options.

(A) ⟼ Control Button for changing the robotic arm configuration to initial pose

(B) ⟼ Control Button for randomizing the pose of objects in the scene

(C) ⟼ Control Button for triggering Pose Estimation and activating Pick and Place action

(D) ⟼ Display Field for showing the ground truth position vector (in m) of the object of interest (*cube*) in the scene

(E) ⟼ Display Field for showing the ground truth rotation (euler) angles (in *deg*) of the object of interest (*cube*) in the scene

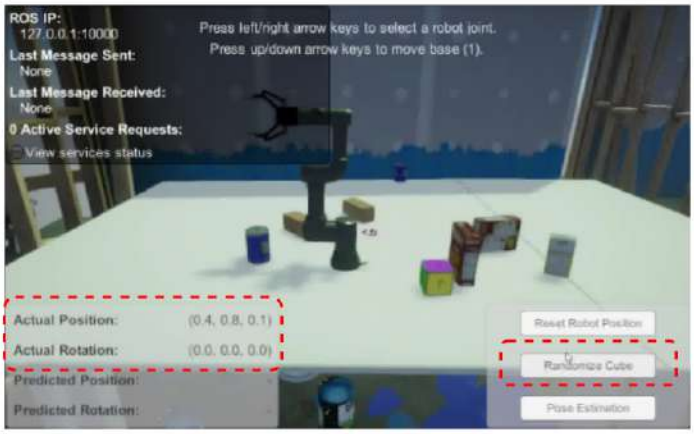
(F) ⟼ Display Field for showing the predicted position vector (in m) of the object of interest (*cube*) in the scene

(G) ⟼ Display Field for showing the predicted rotation (euler) angles (in *deg*) of the object of interest (*cube*) in the scene

(H) ⟼ Status Fields for indicating the progress of various events in the pose estimation pipeline

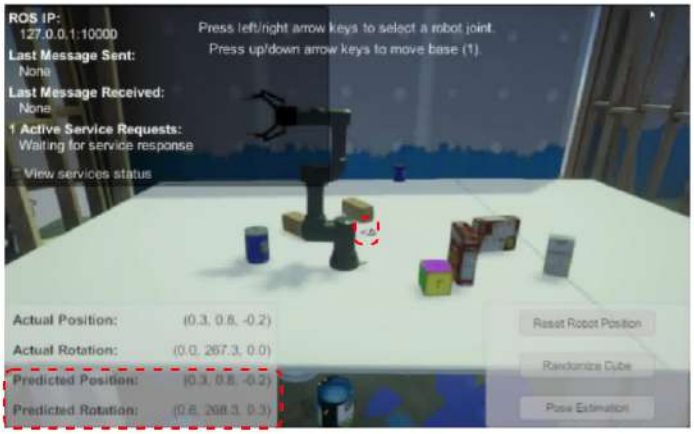
3.5.3 A Pictorial Demonstration

The following sequence of images is provided to give a demonstration of the pick and place task, by the UR3 Robotic Arm. Note that, the final desired outcome is for the cobot to *pick* the *cube* object and *place* it at the *Target* mat location.



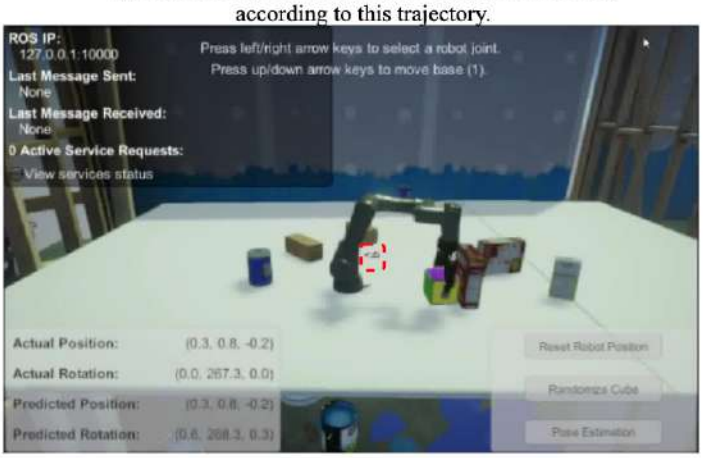
(a) Robotic Arm pose reset and *cube* is randomized

(b) Pose Estimation service requested. RGB image captured and sent for pose inference



(c) Predicted *cube* pose sent to MoveIt Motion Planner

(d) Motion Plan returned. Robotic Arm starts moving according to this trajectory.



(e) Robotic Arm *picks* up the *cube*

(f) Robotic Arm follows the rest of the trajectory while holding the *cube*. Finally, *cube* is placed at the *Target* Mat

Fig. 3.15: Pictorial Demonstration of *Pick and Place* task. The keyframes corresponding to various steps of the task are shown in a chronological manner. Notice that, the object of interest is picked up from the initial location after estimating its pose and placed in its target position, after planning the trajectory for the motion.

Chapter 4

Pose Estimation Models

In this chapter, a detailed description of the different object pose estimation models used in this project, for determining the position and orientation (6D Pose) of an object of interest (In this case, *cube* object) to enable tasks such as pick and place is given. Deep Learning based models are mainly utilized for this purpose in this work, for single-object, single-instance pose estimation. The overall architecture of each approach used and related concepts are described in this chapter. More details about the training configuration, performance evaluation and benchmarking of the various models will be presented in the forthcoming chapters.

4.1 Model-1 : UnityVGG16

The overall architecture of Model-1 *UnityVGG16* is shown in Fig. 4.1. It is basically a template based approach (see Chapter 2, Section 2.3) utilizing DL.

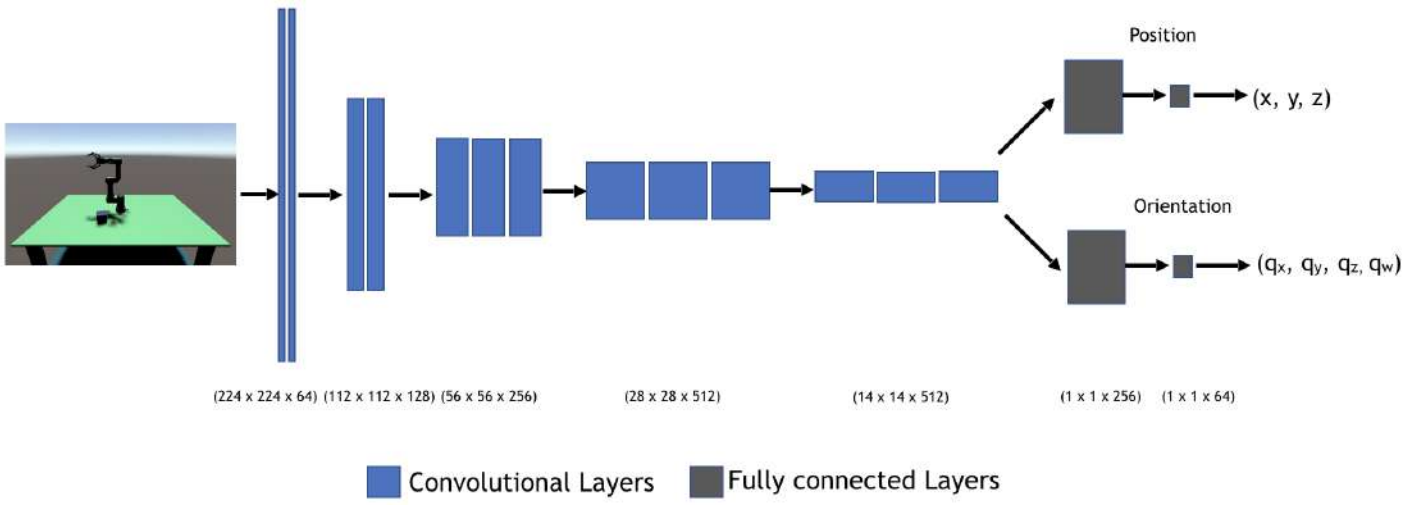


Fig. 4.1: Model Architecture of *UnityVGG16*. It is a template-based (implicit) approach that directly regresses the pose information from the RGB image.

In this work, this model was used a "place-holder" network for testing the pose estimation pipeline (for pick and place in the simulated environment) and was also utilized later as a baseline comparing performance of other models developed.

The following are some of the features of this model :

- Input to the network is an RGB image of size 224×224 ($= H \times W$).
- Image features are extracted by the VGG16 [16] backbone.
- The network consists two heads made up of fully-connected layers, for directly regressing the 3D position (x, y, z) and orientation quaternion (q_x, q_y, q_z, q_w) of the object of interest, by the utilizing the extracted features.
- Please note that, Transfer Learning was utilized here, where the weights of the feature extraction backbone were initialized from the pretrained model on ImageNet dataset [6].

4.2 Transformations between Image and Real World

For understanding some of the aspects of the subsequent models, it is required to understand how to convert real points (in the 3D space) into the image 2D pixel coordinates. Also, for correspondence based approaches, it is a common requirement to be able to get the transformation matrix (rotation matrix augmented with translational vector) from the 2D-3D correspondences. In this section, a brief overview about such concepts is provided.

4.2.1 Perspective Projection (Real World $\longrightarrow$ Image)

For converting 3D points in the real world to the 2D pixel coordinates, we can follow the steps given below :

(i) Consider a 3D point (U, V, W) in the real world. Note that these points can be one of the keypoints of the object of interest, expressed in the local coordinate frame with its origin at centroid.

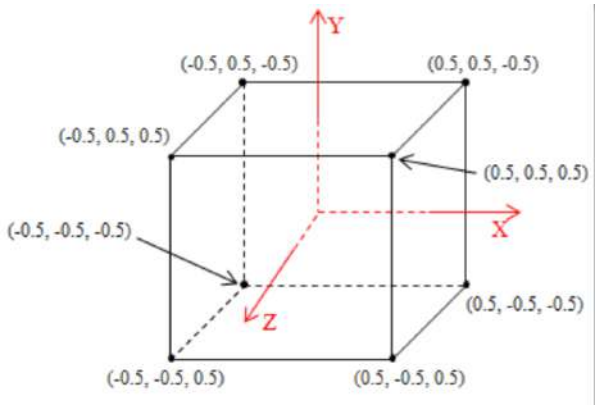


Fig. 4.2: Bounding box coordinates of *cube* object (Edge length = 10 cm) expressed in local coordinate frame

(ii) Now, we can express this point in the camera coordinate frame as (X, Y, Z) by multiplying with the following transformation matrix (rotation matrix augmented with translational vector) :

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \mathbf{R} & | & \mathbf{t} \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

(iii) Now, multiply the vector (X, Y, Z) with the camera intrinsic matrix (see Chapter 3, Section 3.3.2). The intrinsic parameters represent the optical center and focal length of the camera assuming a pin-hole camera model. It is one of the necessary steps for mapping the camera coordinates into the image plane[§] :

$$\begin{bmatrix} P'.x \\ P'.y \\ P'.z \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 346.2024 & 0 & 325 \\ 0 & -346.2024 & 200 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

(iv) Now, in order to project the points onto the image plane, we use the principle of foreshortening. Foreshortening describes the optical illusion, that an object or a distance is smaller than it really is, due to being angled towards the viewer. Another rule related to foreshortening states that vertical lines are parallel, while nonvertical lines converge to a perspective point, thereby appearing shorter than they really are. These effects give a sense of depth, which is useful in evaluating the distance of objects from the viewer.

In order to apply, these principles we calculate the Perspective divide as follows :

$$P.x = \frac{P'.x}{P'.z}$$
$$P.y = \frac{P'.y}{P'.z}$$

This relation can be obtained easily using similarity of triangles.

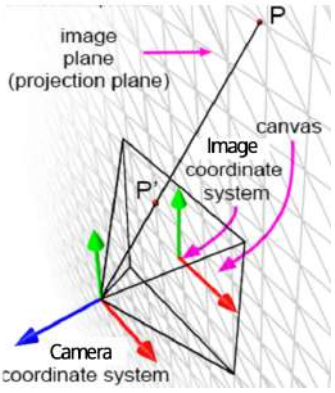


Fig. 4.3: Illustration representing projection of 3D points onto the image plane. P is the original point and P' is the projected point on the image plane.

(v) Finally, we can get the pixel locations by simply shifting the origin to the top left corner, as pixels are generally expressed in raster coordinate system (shown in Fig. 4.4)

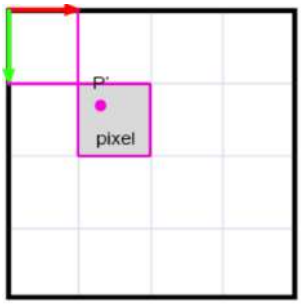


Fig. 4.4: Raster Coordinate System. This is the coordinate system generally used for representing the pixels of an image. The origin is at the top left corner.

[§]Here, we have considered the intrinsic matrix for the unity virtual camera sensor

4.2.2 Perspective-n-Point (PnP) Algorithm

The problem statement that PnP algorithm basically addresses is to find the 6D pose of the object w.r.t camera (or equivalently rotation + translation), given 3D points (keypoints in this case), corresponding 2D image coordinates and camera intrinsic matrix. Now, in order to give an overview of how the PnP algorithm basically finds the solution, a mathematical formulation of the problem is imperative.

Consider a perspective function $\mathbf{\Pi} : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ (computes the perspective divide) such that :

$$\mathbf{\Pi}\left(\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}\right) = \begin{bmatrix} \frac{a_x}{a_z} \\ \frac{a_y}{a_z} \end{bmatrix}$$

In a single equation the 2D image coordinates can be obtained as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{\Pi}\left(\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}\right)$$

$$s_{unknown} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where $s_{unknown}$ is an unknown scale factor, which arises because the perspective divide is not independent of the point which needs to be projected. The aim here is to the find the augmented transformation matrix, $\mathbf{T_y} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$, which cannot be found by simple matrix manipulation because of the unknown scale factor. Let $\mathbf{y} = g(\mathbf{x}, \mathbf{z}, \mathbf{K})$ denote the PnP solver. Then, this problem of finding the pose is formulated as an optimization problem.as follows :

$$\mathbf{y} = \underset{\mathbf{y} \in SE(3)}{\arg\min} \quad \|\mathbf{x} - \boldsymbol{\pi}\|_2^2 \quad ; \quad \boldsymbol{\pi}_i = \mathbf{\Pi}\left(\mathbf{K}\mathbf{T}_\mathbf{y}\mathbf{z}_i\right)$$

where

$\mathbf{K} \longrightarrow$ Camera Intrinsic Matrix

$\mathbf{y} \longrightarrow$ Predicted Object Pose

$\mathbf{x}_i \in \mathbb{R}^2 \longrightarrow$ i^{th} 2D image coordinate

$\mathbf{z}_i \in \mathbb{R}^3 \longrightarrow$ Corresponding i^{th} 3D keypoint coordinate

$\boldsymbol{\pi}_i \in \mathbb{R}^2 \longrightarrow$ Corresponding i^{th} 2D projected image coordinate using the predicted pose

$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1^T & \mathbf{x}_2^T & \dots & \mathbf{x}_n^T \end{bmatrix}^T \in \mathbb{R}^{2n \times 1} \longrightarrow$ Flattened array consisting of corresponding 2D image coordinates of 'n' keypoints

$\mathbf{z} = \begin{bmatrix} \mathbf{z}_1^T & \mathbf{z}_2^T & \dots & \mathbf{z}_n^T \end{bmatrix}^T \in \mathbb{R}^{3n \times 1} \longrightarrow$ Flattened array consisting of corresponding 3D coordinates of 'n' keypoints

This problem by using an iterative optimization procedure called "Levenberg-Marquad Algorithm". In this work, OpenCV's `SOLVE_PNP_ITERATIVE` is used to the same.

4.3 Model-2 : Pose6DSSD

In order to improve the performance of pose estimation, the second model designed and explored is *Pose6DSSD*[‡]. It is a correspondence based approach (see Chapter 2, Section 2.3). The complete architecture of the model is illustratrated in Fig. 4.5. Some of the ideas for designing the model were taken from [19].

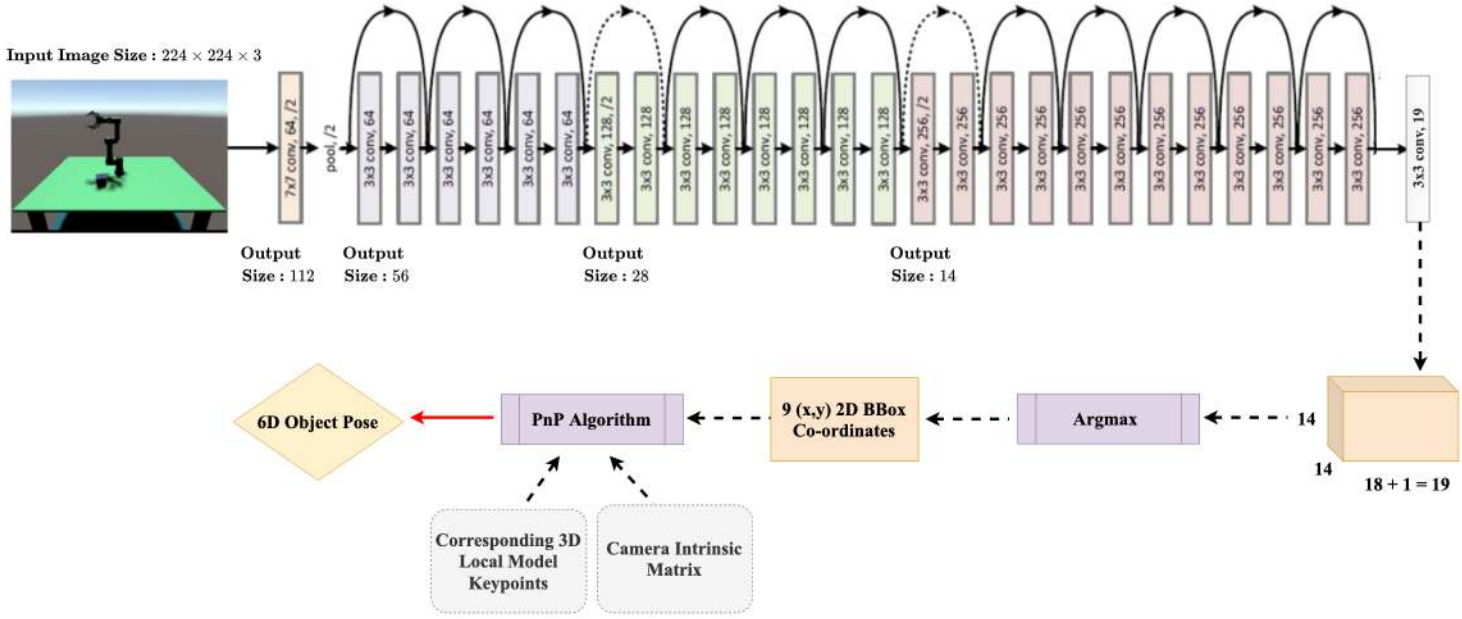


Fig. 4.5: Model Architecture of *Pose6DSSD*. It is a correspondence-based approach, which involves estimation of the 2D keypoints followed by extraction of pose information using the PnP algorithm.

The following are some of the main features of this approach[†] :

- Input to the network is an RGB image of size 224×224 ($= H \times W$).
- As it is a correspondence based approach, we first the regress the 2D image coordinates of certain kepoints, which in this approach, are the 8 corners and the centroid of the 3D bounding box around the object of interest.
- The main feature extraction backbone consists of 27 convolutional layers with residual skip links, and has been adapted from ResNet34 architecture[9].
- There are no fully connected layers used as opposed to Model-1 *UnityVGG16*, to limit the number of parameters and gain other typical advantages.
- Except the output layer, in all other blocks BatchNorm Layer is used followed by ReLU non-linear activation.
- Transfer Learning was utilized here, where the weights of the feature extraction backbone were initialized from the pretrained model on ImageNet dataset [6].

[‡]Stands for 6D Pose **S**ingle **S**tage **D**etector
[†]More details will be given shortly

4.3.1 Interpreting the Feature Extraction Output

In this subsection, details regarding the interpretation of the output tensor from the feature extraction backbone are given. For a single image input, the output of the main feature extraction backbone is 3D tensor of dimensions $S \times S \times (2K+1)$. The following information is helpful to understand how this output can be interpreted :

→ The input image is partitioned into a 2D regular grid (see Fig. 4.6) with $S \times S$ cells. In this work, $S = 14$ is considered.

→ For each grid, $2K+1$ values are predicted where K is the number of keypoints being considered. Note that, since we are considering a 3D bounding box based approach, $K = \underbrace{8}_{\text{8 corners}} + \underbrace{1}_{\text{the centroid}}$.

→ The remaining one value predicts the confidence value of the grid cell, i.e. how confident the model is that in a given grid cell the object of interest is present.

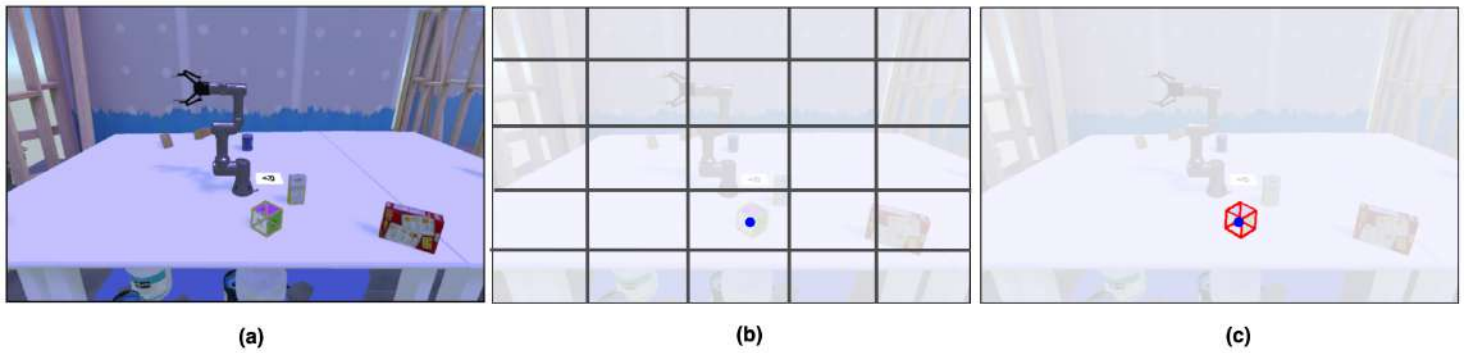


Fig. 4.6: (a) An example input RGB image. (b) The image is divided into $S \times S$ regions denoted by the square grid. (c) Each cell predicts 2D locations of the corners of the projected 3D bounding box in the image.

→ The model is designed to predict normalized offset values from the bottom-left grid point of each grid cell, (c_x, c_y). Then the effectively predicted normalized control point (g_x, g_y) is :

$$g_x = f(x) + c_x$$
$$g_y = f(y) + c_y$$

Note that, $f(z)^{\dagger}$ is the sigmoid function for the centroid coordinates' prediction where it is the identity function for other coordinates' prediction. The intuition behind this is to make sure that the model first learns to find in which grid cell, the object of interest is present. The coordinates of the centroid should fall inside the cell, which is predicting it but there are no constraints for other keypoints.

4.3.2 Modelling the Ground Truth Confidence

For the 2D object detection case, the YOLO single shot detector basically uses an IoU (Intersection over Union) score for between the true and predicted 2D bounding box. To do the same thing analogously in this case, would require the computation of the 3D convex hull of intersection between the predicted and ground truth 3D bounding boxes. This would computationally expensive and might increase the overall training time. So the ground truth confidence values for training

[†] z can be either x or y

the model is modelled in the following manner :

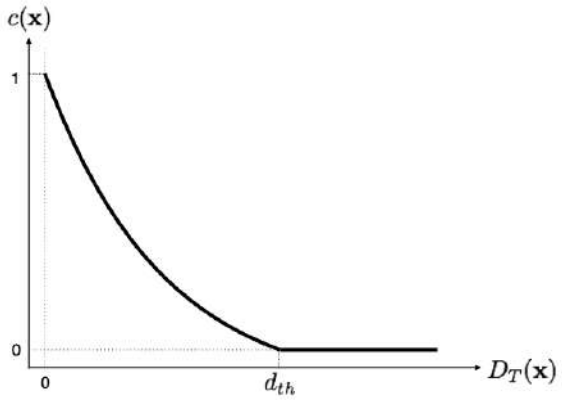


Fig. 4.7: Confidence $c(x)$ measured as a function of distance $D_T(x)$ between a predicted and ground truth point.

$$c(\mathbf{x}) = \begin{cases} e^{\alpha\left(1-\frac{D_T(\mathbf{x})}{d_{th}}\right)}, & \text{if } D_T(\mathbf{x}) < \mathbf{d}_{th} \\ 0 & \text{otherwise} \end{cases}$$

The intuition behind this is the idea that the confidence value is low when there is no object in grid cell, it is high when the object is present in the grid cell. The exponential function is used to model this. Here, the values chosen for some paramters are $\alpha = 2$ and $d_{th} = 80$ pixels. In practice, the confidence value is computed for all control points in all grid cells and the mean is assigned as ground truth confidence during training*.

After the feature extraction output tensor is obtained (Refer 4.3.1 for interpretation) the grid cell output with the maximum confidence value is selected as the final candidate and the $2K$(corresponding to 2D image coordinate predictions) values are converted into the 9 (x, y) coordinates (unnormalized). Please consider the Fig. 4.5 for details of the steps involvec. These along with the cooresponding 3D model points expressed in the local model frame and camera intrinsic matrix, form the input to the PnP algorithm (see Section 4.2.2) block. The output of the this block is the final predicted 6D pose for the object of interest present in the input image.

4.4 Model-3 : DOSSE-6D

We can observe that the previous model used a traditional correspondence based (using DL), so it is not an end-to-end approach, as we cannot directly utilize the output 6D pose information for training the model. We rely on indirect supervision (see Chapter 5) for training the model. The next model which is an improvement of the previous model (Model-2 *Pose6DSSD*) in this aspect as well as some other aspects which are discussed later, and is called *DOSSE-6D*§. Three versions of this model have been developed, each of them having small differences (improvements), compared to the other. More details about each one of them will be provided subsequently in this section. The complete architecture, represented in a high level block diagram fashion, of the first version of Model-3 *DOSSE-6D_v1* is shown in Fig. 4.8. The third version can also be represented by the same block diagram in a high level sense. The architecture for the second version of Model-3 *DOSSE-6D_v2* is shown in Fig. 4.9.

Some of the common features of this approach among all the versions are as follows :

- It is a correspondence based approach, we first the regress the 2D image coordinates of certain kepoints, which in this approach, are the 8 corners and the centroid of the 3D bounding box around the object of interest.
- The main feature extraction backbone consists of convolutional layers with residual skip links, and has been adapted from ResNet34 architecture[9].

*More details about the training configuration will be provided in the next chapters.
§Stands for **D**eep **O**bject **S**ingle **S**hot **E**stimator of **6D** object pose

- There are no fully connected layers used as opposed to Model-1 *UnityVGG16*, to limit the number of parameters and gain other typical advantages.
- The PnP block in Model-2 *Pose6DSSD* has been replaced by the BPnP module (see Section 4.4.1), to make the model end-to-end trainable.
- The interpretation of the output tensor and the approach for modelling the ground truth confidences remain the same, as discussed in Section 4.3.1 and Section 4.3.2 respectively.

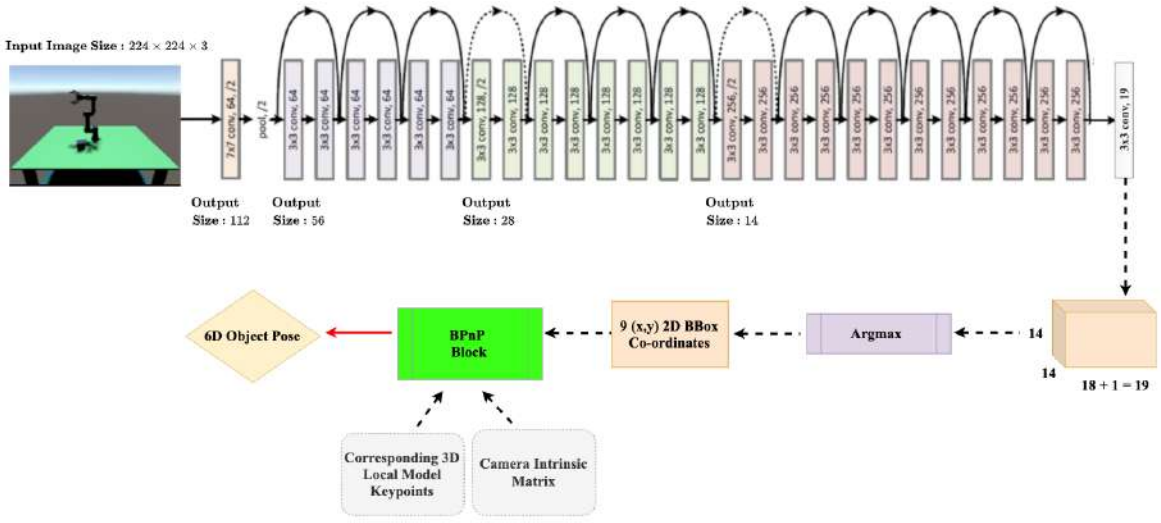


Fig. 4.8: High Level Model Architecture of *DOSSE-6D_v1* & *DOSSE-6D_v3*. It is a correspondence-based approach, which involves estimation of the 2D keypoints followed by extraction of pose information using the BPnP module (see Section 4.4.1). Input is an RGB image of resolution 224 × 224 px.

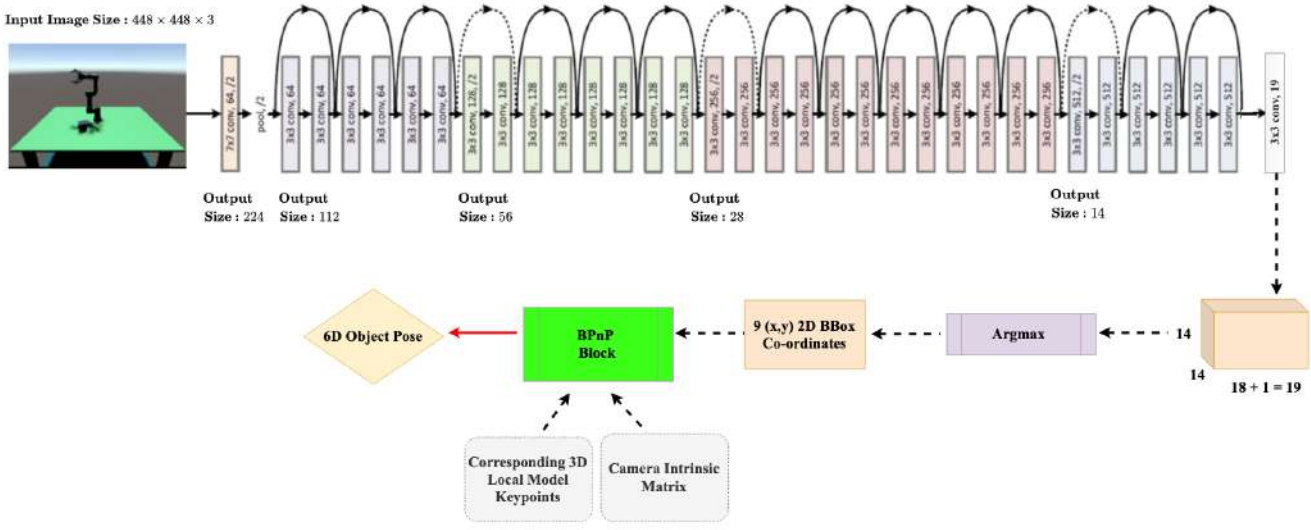


Fig. 4.9: High Level Model Architecture of *DOSSE-6D_v2*. It is a correspondence-based approach, which involves estimation of the 2D keypoints followed by extraction of pose information using the BPnP module (see Section 4.4.1). Input is an RGB image of resolution 448 × 448 px.

4.4.1 Backpropagatable PnP (BPnP) Module

The BPnP block [4] is a module Backpropagates gradients through a Perspective-n- Points (PnP) solver (see Section 4.2.2) "layer" to guide parameter updates of a neural network. Based on the concept of implicit differentiation, it helps to combine DL network and geometric vision to form an end-to-end trainable pipeline.

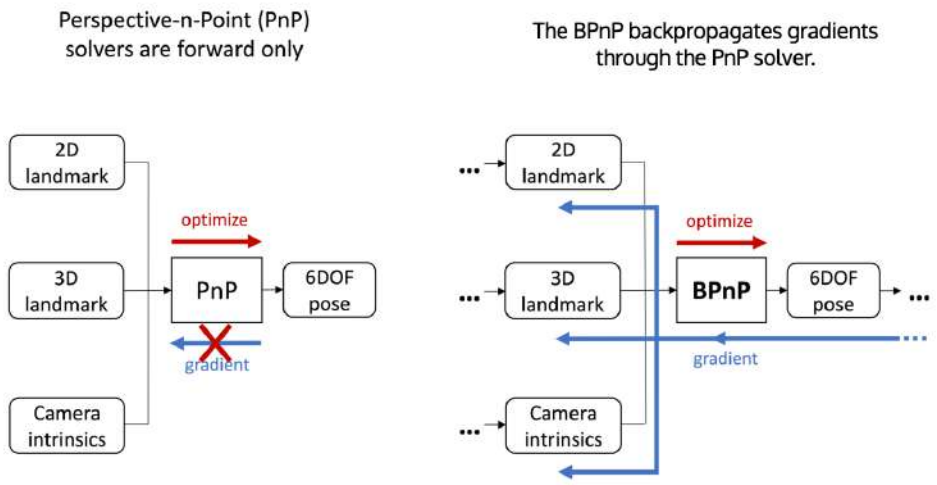


Fig. 4.10: Basic motivation of BPnP module. This module provides an elegant way to backpropagate gradients through the PnP Solver.

Implicit Function Theorem (IFT)

Let $f : \mathbb{R}^{n+m} \to \mathbb{R}^m$ be a continuously differentiable function with input $(\boldsymbol{a}, \boldsymbol{b}) \in \mathbb{R}^n \times \mathbb{R}^m$. If a point $(\boldsymbol{a}^*, \boldsymbol{b}^*)$ satisfies :

$$\underbrace{f\left(\boldsymbol{a}^*, \boldsymbol{b}^*\right) = \mathbf{0}}_{\text{(Stationary constraint)}} \quad \& \quad \frac{\partial f}{\partial \boldsymbol{b}}\left(\boldsymbol{a}^*, \boldsymbol{b}^*\right) \text{is invertible}$$

then there exists an open set $U \subset \mathbb{R}^n$ such that $\boldsymbol{a}^* \in U$ and a unique continuously differentiable function $g(\boldsymbol{a}) : \mathbb{R}^n \to \mathbb{R}^m$ such that :

$$\mathbf{b}^* = g\left(\boldsymbol{a}^*\right) \quad \& \quad f\left(\boldsymbol{a}', g\left(\boldsymbol{a}'\right)\right) = \mathbf{0}, \quad \forall \boldsymbol{a}' \in U$$

Using this, moreover, we can compute the derivatives of a function g with respect to its input a without an explicit form of the function as follows :

$$\begin{aligned} &\left[\frac{\partial f(\boldsymbol{a}, g(\boldsymbol{a}))}{\partial \boldsymbol{b}}\right] \frac{\partial g(\boldsymbol{a})}{\partial \boldsymbol{a}} + \frac{\partial f(\boldsymbol{a}, g(\boldsymbol{a}))}{\partial \boldsymbol{a}} = 0 \\ \implies &\frac{\partial g(\boldsymbol{a})}{\partial \boldsymbol{a}} = \left[\frac{\partial f(\boldsymbol{a}, g(\boldsymbol{a}))}{\partial \boldsymbol{b}}\right]^{-1} \left[\frac{\partial f(\boldsymbol{a}, g(\boldsymbol{a}))}{\partial \boldsymbol{a}}\right] \end{aligned} \tag{1}$$

Computing the Gradients

To use IFT to compute the req. gradients we define the constraint function $f(\boldsymbol{a}, \boldsymbol{b})$ as :

$$f(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{K}) = \frac{\partial o(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{K})}{\partial \mathbf{y}} = [f_1, \ldots, f_m]^T$$

where

$$o(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{K}) = \sum_{i=1}^{n} \|\mathbf{r}_i\|_2^2 = \sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\pi}_i\|_2^2 \quad ; \quad \boldsymbol{y} \in \mathbb{R}^m$$

and

$$f_j = \frac{\partial o(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{K})}{\partial y_j} = 2 \sum_{i=1}^{n} \left\langle \mathbf{r}_i, \frac{\partial \mathbf{r}_i}{\partial y_j} \right\rangle = -2 \sum_{i=1}^{n} \left\langle \mathbf{r}_i, \frac{\partial \boldsymbol{\pi}_i}{\partial y_j} \right\rangle$$

Note that, the terminology for all the other terms remains the same as that in Section 4.2.2. In this work, m=6 (angle-axis representation of pose) is considered, as suggested in [4].

Observe that, f satisfies stationary constraint as the output pose is local minimum for PnP solver. For the purpose of object pose estimation, we can consider $\mathbf{a} = \mathbf{x}$, $\mathbf{b} = \mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{z}, \mathbf{K})$. Then, finally, we can obtain the Jacobians of g w.r.t to its inputs using IFT as (From Equation (1)):

$$\frac{\partial g}{\partial \mathbf{x}} = -\left[\frac{\partial f}{\partial \mathbf{y}}\right]^{-1} \left[\frac{\partial f}{\partial \mathbf{x}}\right] \tag{2}$$

During backward pass, using we can get the input gradient (through the PnP solver) using chain rule as (From Equation (2)) :

$$\frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{x}} = \left[\frac{\partial g}{\partial \mathbf{x}}\right]^{T} \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{y}}$$

Note that these gradients can be computed with the help of PyTorch's autograd package. Using this (instead of Fully connected layers as in Model-1) we can optimise feature based loss and learn geometric constraints in an end-to-end manner.

4.4.2 Differences among Model Versions (*v_1*,*v_2* and *v_3*)

The common features listed in the beginning of Section 4.4 remain same for all the versions, only the constrast in other aspects is discussed here in Table 4.1. The attention mechanism employed in *v2* and *v3* will be discussed in the next section.

S.No.	Feature \ Model Version	*DOSSE-6D_v1*	*DOSSE-6D_v2*	*DOSSE-6D_v3*
1.	**Input Image Size (H × W)**	224 × 224	448 × 448	224 × 224
2.	**No. of Conv2D layers (in Feature Extraction Backbone)**	27	34	27
3.	**Attention Mechanism Used**	None	Channel + Spatial	Channel + Spatial
4.	**Transfer Learning Utilized**	Yes	No	No

Table 4.1: Table displaying the details of features in which the three *DOSSE-6D* versions differ.

4.4.3 Attention Module

The versions 2 and 3 of the *DOSSE-6D* utilize attention modules in their architectures. 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 maps, by telling the model "where" to focus and improve its hidden representations. Main Idea is to force the model to focus on important features and suppress unnecessary ones. Incorporation of Attention module was done by the considering the best empirical practices in [23] and [22], found by extensive experimentation. Each attention module basically consists of two sub-modules namely : *Channel Attention* and *Spatial Attention* as shown in Fig. 4.11.

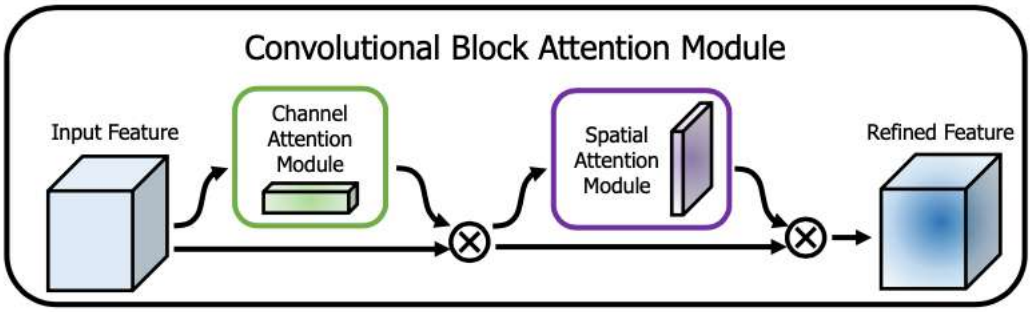


Fig. 4.11: Illustrative Representation of the Attention Module. Note that, the attention module is made up of submodules : *Channel Attention* (Section 4.4.3) and *Spatial Attention* (Section 4.4.3).

Channel Attention Sub-Module

The intuition behind channel attention sub-module is to improve the feature maps 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.

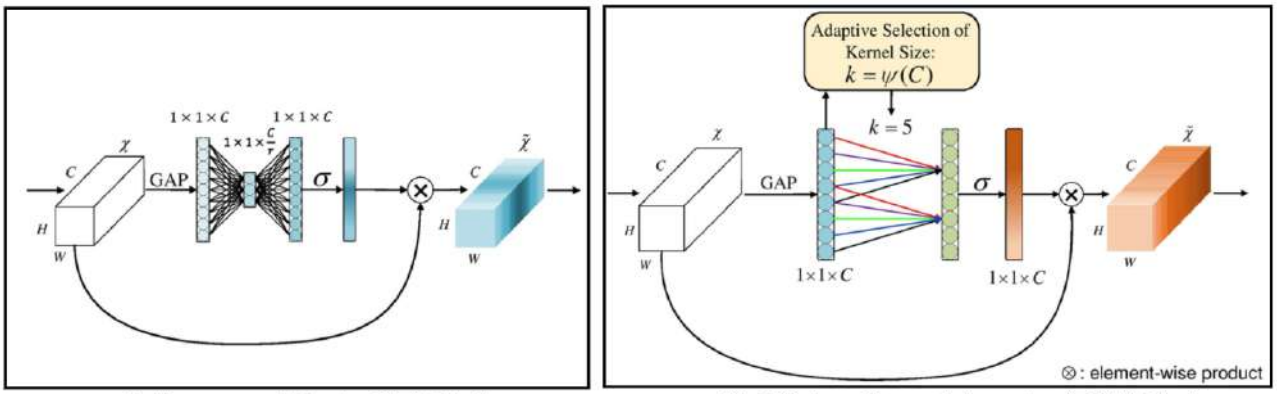


Fig. 4.12: Different types of Channel Attention mechanisms. These are popularly used in methods, based on different principles. GAP stands for **G**lobal **A**verage **P**ooling.

In general, there are two popular types of channel attention mechanisms used as shown in Fig. 4.12 : *SE block* and *ECA block*. In [22], it was shown empirically that for an 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. 4.13, in this work, both the Maxpool and Average pool 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

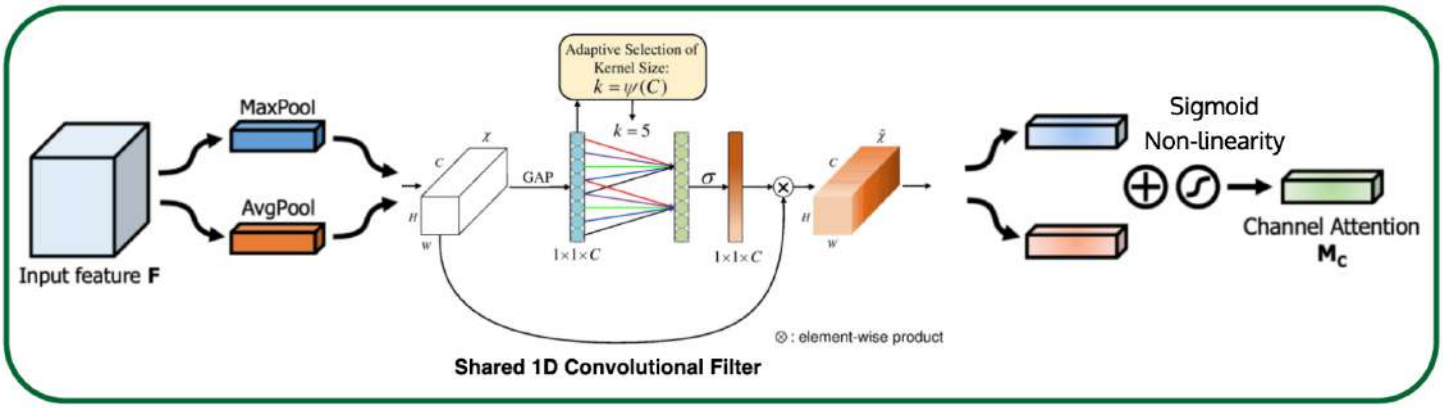


Fig. 4.13: Channe Attention Sub-Module Architecture. Notice the use of Effective Channel Attention (ECA) block due to its advantages.

The intuition behind using both the type of features is that (In [23], it is shown through some experiments that that both are complementary) :

→ Max-pooled features encode the degree of the most salient part in the feature map.

→ Average-pooled features encode global statistics softly.

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 :

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

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. 4.14. In this module, spatial attention map is obaitned which can be used to improve features utilizing the inter-spatial relationship of features. In other words, it is helps the model to basically decide "where" to focus in a feature map. As shown in Fig. 4.14, we are using both the Max-pool and Average pool features (performed along the channel dimension, to get 2D feature map) are used for the same reasons as discussed in Section 4.4.3. Both these feature maps are stacked together and 2D convolution is performed followed by

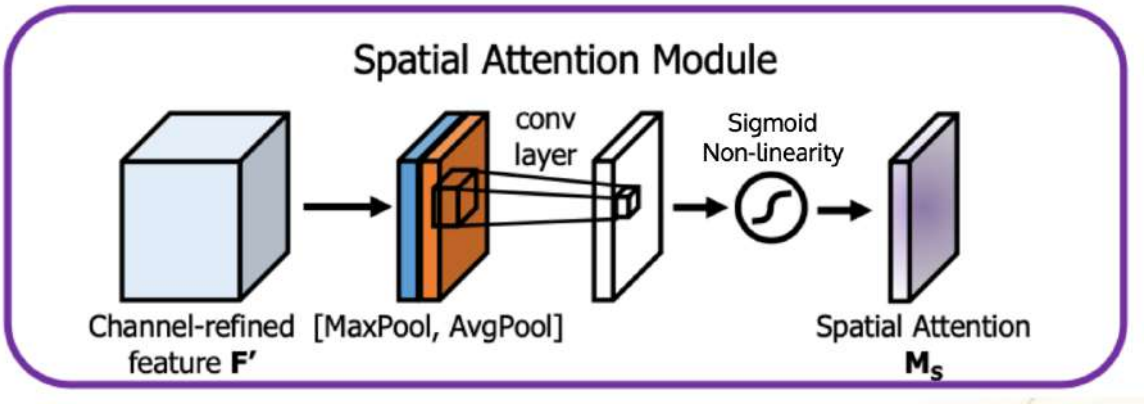


Fig. 4.14: Spatial Attention Sub-Module Architecture

passing the output through a sigmoid non-linearity to restrict the range of values to $[0, 1]$.

Empirical results in [23] show that larger kernel size generates better accuracy. The intuition is that larger kernel sizes are necessary for deciding spatially important regions. So, in this work, we use consider 7×7 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 [23] 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. 4.15.

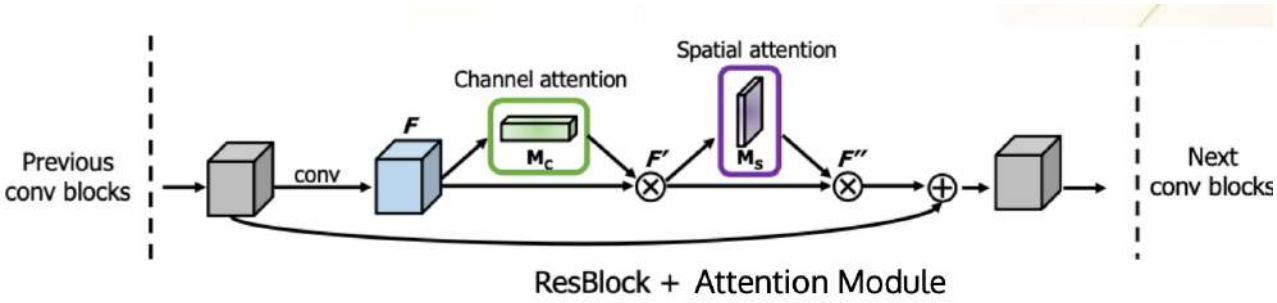


Fig. 4.15: Configuration of attention sub-modules within the main attention module

Since, we are using a ResNet adapted backbone in all the versions of *DOSSE-6D* model, the completer attention module is placed at the end of each ResBlock[§].

[§]Here, the terminlogy considered is each ResBlock consists of two convolutional layers before the skip connnections. So the arrangement of different layers is as follows : $Conv1 \rightarrow BatchNorm1 \rightarrow ReLU \rightarrow Conv2 \rightarrow BatchNorm2 \rightarrow Attention\ Block \rightarrow Skip\ Connection\ (H(x) + x) \rightarrow ReLU$

4.5 Model-4 : AHR-DOSSE-6D

In this section, a description of Model-4 *AHR-DOSSE-6D*[†] is given. The motivation for the design[§] of this model is provided first, followed by architectural details.

4.5.1 Intuition behind Model Design

The following are the some of the desirable attributes present in Model-1, Model-2 and Model-3, that we wish to preserve in the new model :

- Single Stage Correspondence approach without post-refinement stages
- End-to-end trainablity
- Use of Attention Module (Spatial + Channel)
- Use of only RGB image, without depth information

The new features that would (possibly) improve the performance of pose estimation (based on general observations as well as Results and Inferences in Chapter 6 and Chapter 7) are :

- Maintain High-Resolution representations throughout the backbone, which can extremely useful for estimating the pose of small objects.
- Use of more geometrical details of the object under consideration.
- Replacing the modelled confidence approach with HeatMap Estimation. Note that, this has been developed keeping in mind that it can be extended further easily to multi-object and multi-instance object pose estimation using Part Affinity Fields (PAFs).
- Increased input resolution image (leading to increased perfomance as seen in results of Chapter 6 and Chapter 7) without increase in parameters.

4.5.2 Model Architecture Details : A High-Level Overview

The basic high-level block diagram of various elements present in the model architecture of *AHR-DOSSE-6D* is shown in Fig. 4.16.

Some of the main features of this approach are :

- The input to the network is an RGB image of generic dimensions $H \times W$. Both the cases of $H = W = 448$ and $H = W = 224$ are considered (Results are presented in subsequent chapters).
- The feature extraction backbone in *AHR-DOSSE-6D* is explained in Section 4.5.3. It makes use of the attention module described in Section 4.4.3.

[†]Stands for **A**ttention **H**igh **R**esolution **D**eep **O**bject **S**ingle **S**ingle **E**stimator
[§]Please note that the motivation will be further reinforced after results and inferences present in Chapter 6 and Chapter 7

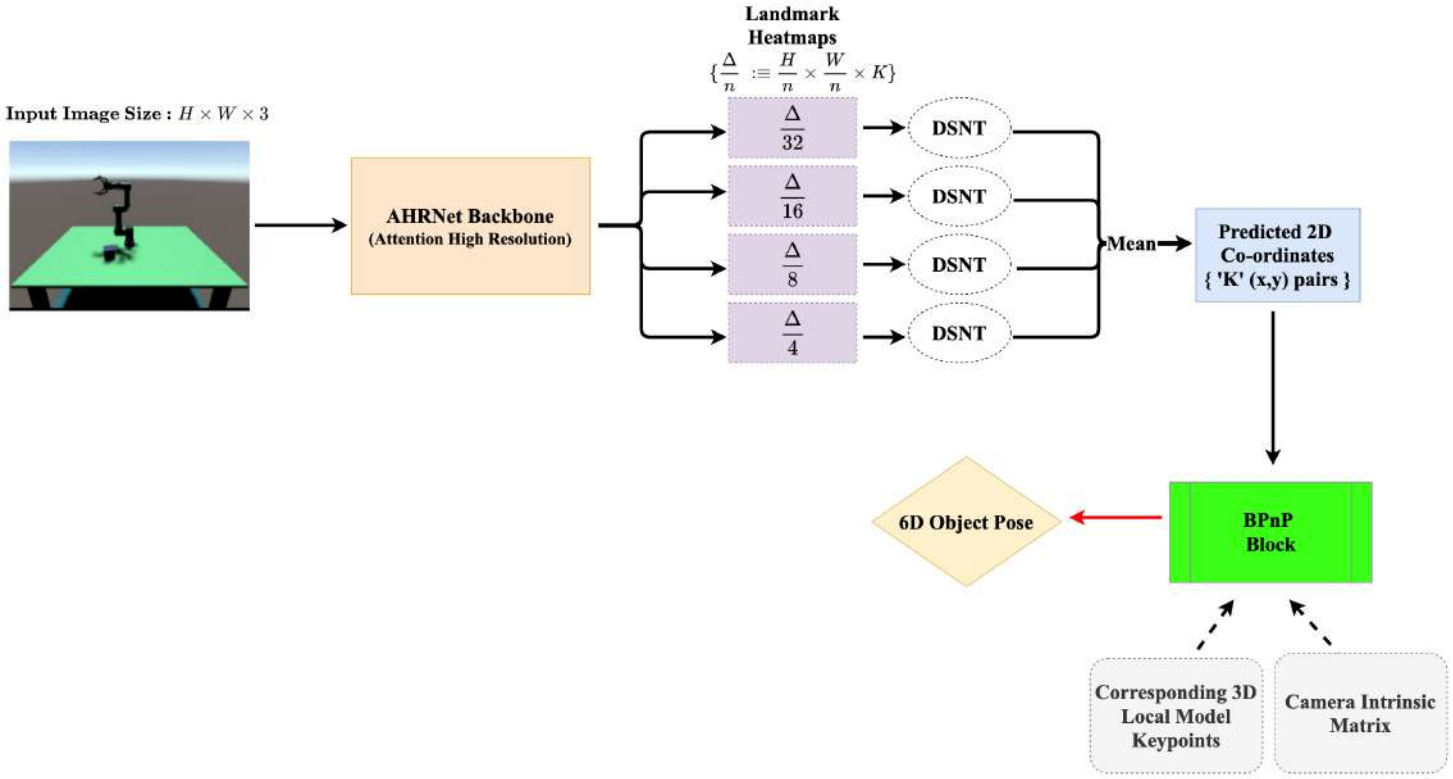
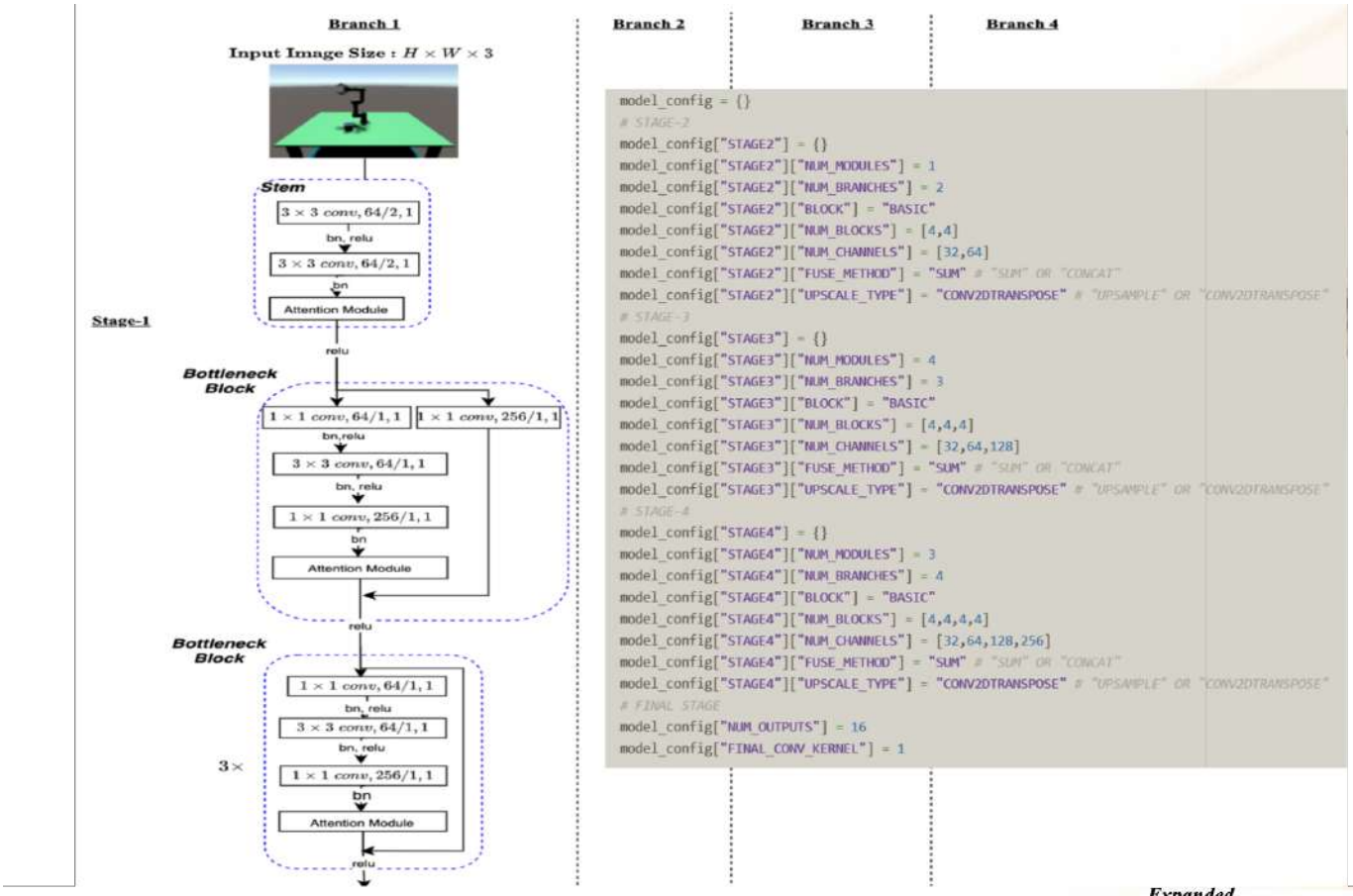


Fig. 4.16: High-Level Block Diagram of *AHR-DOSSE-6D*. It is a correspondence-based approach, which involves estimation of the 2D landmark heatmaps with keypoints, followed by extraction of pose information using the BPnP module (see Section 4.4.1).

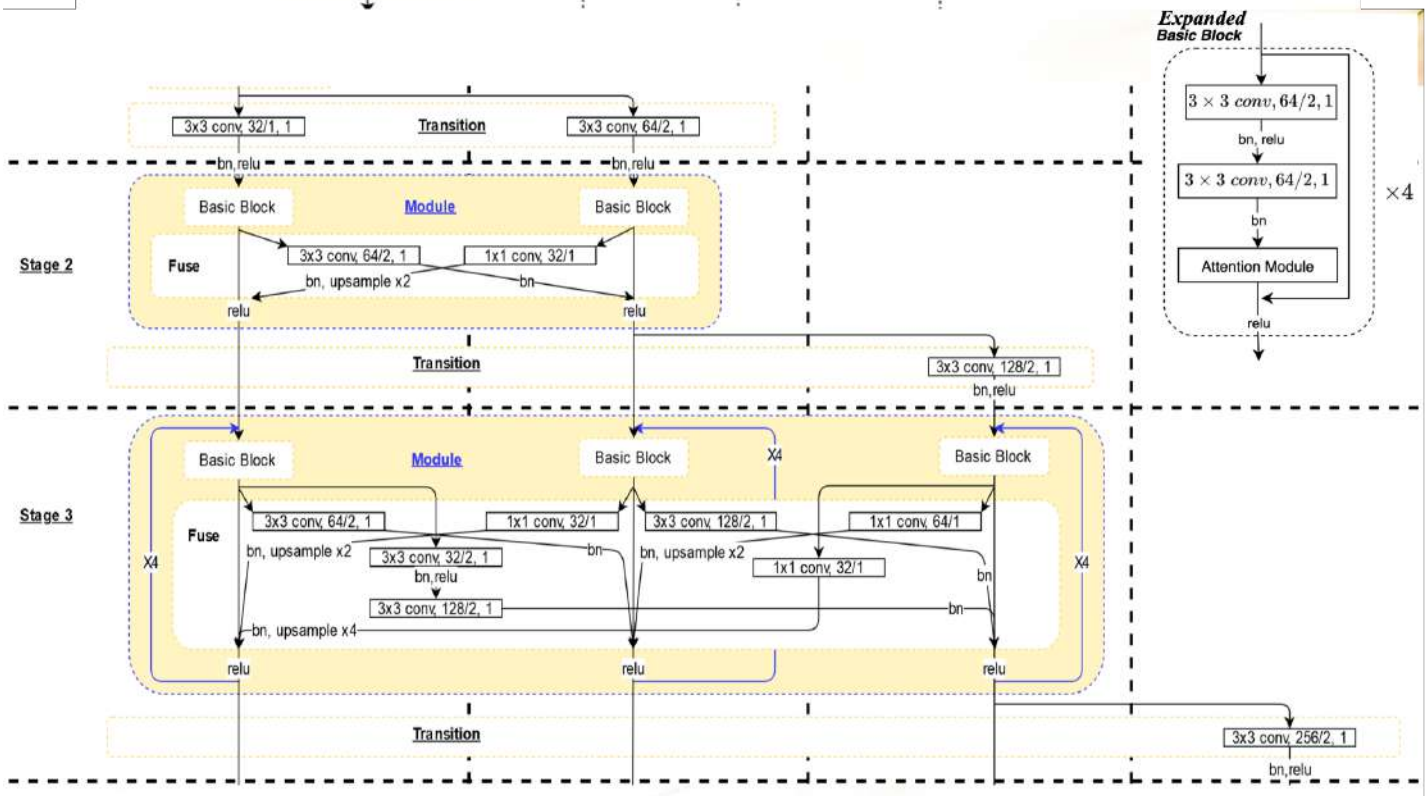
- The output of the *AHRNet Backbone* is a collection of landmark heatmaps $\{\frac{\Delta}{n} := \frac{H}{n} \times \frac{W}{n} \times K\}$, where $n = 4, 8, 16, 32$. K represents the number of keypoints under consideration. Here, $K = \underbrace{8}_{\text{8 corners}} + \underbrace{6}_{\text{6 face centers}} + \underbrace{1}_{\text{the centroid}} = 15$. Note that, for the *cube* object (the object of interest in this work), these are the farthest 15 points in the corresponding 3D model. In this way, addition of more geometrical details is ensured.
- The landmark heatmaps are used to obtain the 2D co-ordinates in normalized form, using the Differential Spatial to Numerical Transform (DSNT) block which is explained in Section 4.5.4. As it is a correspondence based approach, we first the regress the 2D image coordinates of certain keypoints.
- The BPnP block, as described in Section 4.4.1, is to obtain the final predicted pose to make the model end-to-end trainable.
- There are no fully connected layers used as opposed to Model-1 *UnityVGG16*, to limit the number of parameters and gain other typical advantages.

4.5.3 AHRNet Feature Extraction Backbone

The complete architecture of AHRNet backbone for feature extraction is illustrated in Fig. 4.17. Please note that the architecture has been divided into 4 branches and 5 stages for the ease of understanding (more details will be given soon). The convention used for representing *Conv2D* operations in the blocks of Fig. 4.17 is $\{K \times K$ conv, $O/S, P\}$; where K is the kernel size of the filter, O is the number of output channels, S is the stride used, and P represents the padding.




```
model_config = {}
# STAGE-2
model_config["STAGE2"] = {}
model_config["STAGE2"]["NUM_MODULES"] = 1
model_config["STAGE2"]["NUM_BRANCHES"] = 2
model_config["STAGE2"]["BLOCK"] = "BASIC"
model_config["STAGE2"]["NUM_BLOCKS"] = [4,4]
model_config["STAGE2"]["NUM_CHANNELS"] = [32,64]
model_config["STAGE2"]["FUSE_METHOD"] = "SUM" # "SUM" OR "CONCAT"
model_config["STAGE2"]["UPSCALE_TYPE"] = "CONV2DTRANSPOSE" # "UPSAMPLE" OR "CONV2DTRANSPOSE"
# STAGE-3
model_config["STAGE3"] = {}
model_config["STAGE3"]["NUM_MODULES"] = 4
model_config["STAGE3"]["NUM_BRANCHES"] = 3
model_config["STAGE3"]["BLOCK"] = "BASIC"
model_config["STAGE3"]["NUM_BLOCKS"] = [4,4,4]
model_config["STAGE3"]["NUM_CHANNELS"] = [32,64,128]
model_config["STAGE3"]["FUSE_METHOD"] = "SUM" # "SUM" OR "CONCAT"
model_config["STAGE3"]["UPSCALE_TYPE"] = "CONV2DTRANSPOSE" # "UPSAMPLE" OR "CONV2DTRANSPOSE"
# STAGE-4
model_config["STAGE4"] = {}
model_config["STAGE4"]["NUM_MODULES"] = 3
model_config["STAGE4"]["NUM_BRANCHES"] = 4
model_config["STAGE4"]["BLOCK"] = "BASIC"
model_config["STAGE4"]["NUM_BLOCKS"] = [4,4,4,4]
model_config["STAGE4"]["NUM_CHANNELS"] = [32,64,128,256]
model_config["STAGE4"]["FUSE_METHOD"] = "SUM" # "SUM" OR "CONCAT"
model_config["STAGE4"]["UPSCALE_TYPE"] = "CONV2DTRANSPOSE" # "UPSAMPLE" OR "CONV2DTRANSPOSE"
# FINAL STAGE
model_config["NUM_OUTPUTS"] = 16
model_config["FINAL_CONV_KERNEL"] = 1
```



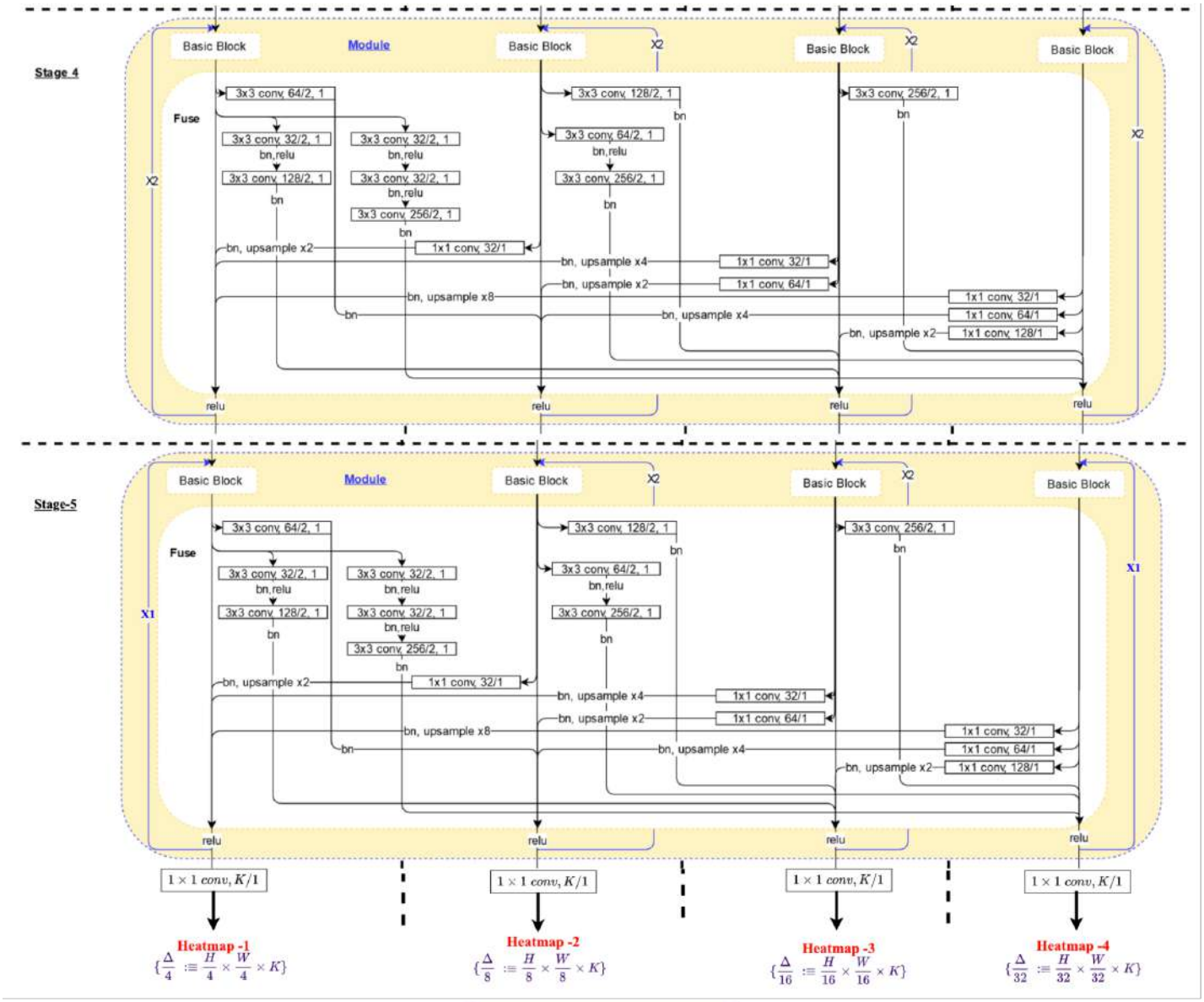


Fig. 4.17: Detailed architecture of the AHRNet Backbone. In the previous page, the moel configuration (different elements of the *AHRNet* backbone) is listed on the right. The legend that is useful for reading the figure and understanding the operations involved is provided in this page at the end of the figure.

Some of the ideas for designing AHRNet backbone have been taken from [17]. The main ideas and intuitions behind this type of architecture are as follows :

→ Repeated Multi-scale 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. 4.18.

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

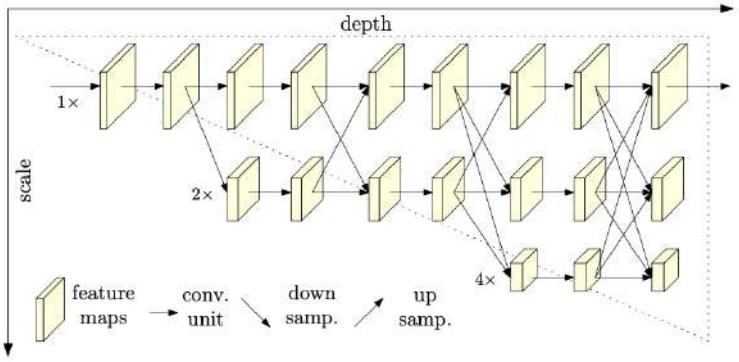


Fig. 4.18: Intuitive visualization of Multi-scale Fusions [17]

4.5.4 Differential Spatial to Numerical Transform (DSNT) Block

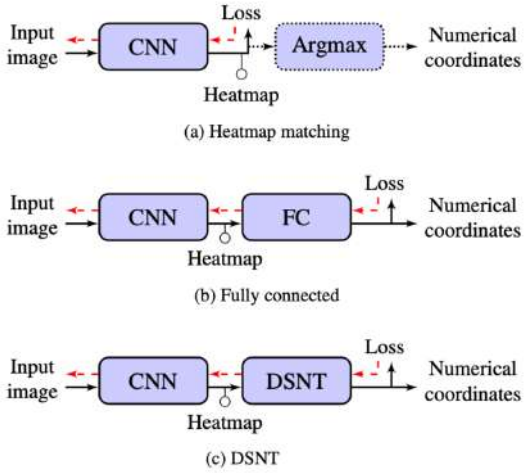


Fig. 4.19: Comparison of coordinate regression appraches[12]. The red dashed arrows indicate the flow of gradients during Backpropagation.

The DSNT [12] block, in simple words, is used for converting the landmark heatmaps produced by the AHRNet Backbone (see Section 4.5.3) to normalized 2D coordinates of the corresponding keypoints. It is basicaly a spatial "soft"-argmax over each feature channel. Commonly used approaches for performing this task of coordinate regression include *Heatmap Matching* and use of fully connected layers. But these approaches have flaws such as the former is not differentiable, and the latter lacks inherent spatial generalization while adding lot of trainable parameters.

Some of the desirable properties of the DSNT block are as follows :

→ It adds no additional trainable parameters.

→ It is fully differentiable and thereby, helps in designing end-to-end models.

→ It exhibits good spatial generalization, as it based on computing the spatial 2D expectation.

→ It performs well even with low heatmap resolutions.

.

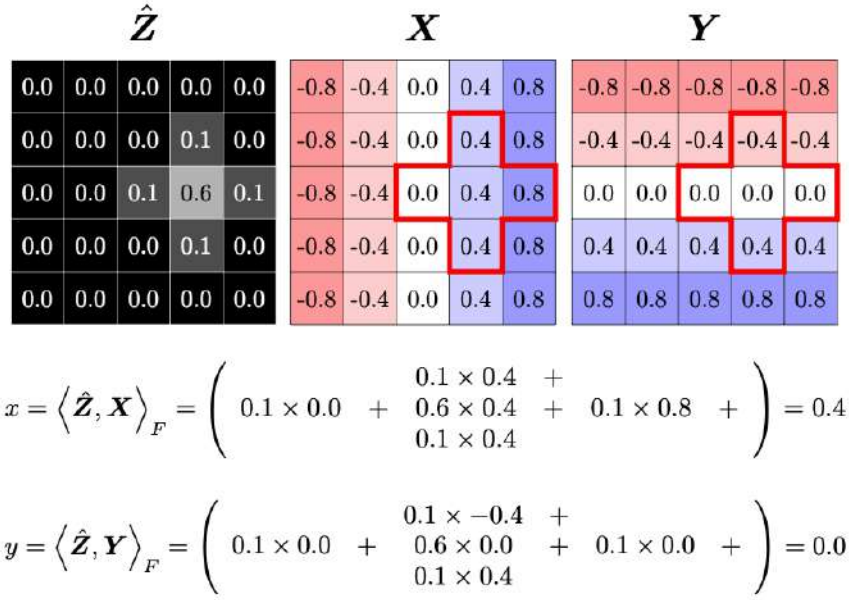


Fig. 4.20: An example illustrating how coordinates can be computed from heatmaps, in a normalized system [12]. Here, the pixel with weight of 0.6 in the heatmap $\hat{\mathbf{Z}}$ is the location of the predicted keypoint. The coordinate system has its origin $(0, 0)$ as the center pixel and the top left pixel has the coordinate $(-1, -1)$. The spatial 2D expectation provides the correct location as $(0.4, 0)$.

Note

Multiple landmark heatmaps (Here, 4) are generated by the AHRNet backbone. DSNT block is used to extract the 2D coordinates of the landmarks (which are bascially the keypoints). Mean of the predicted 2D co-ordinates (normalized) across multiple scales is considered. The intuition is that of "Multi-scale Supervision" while training, might help in flow proper flow of gradients during backpropagation for updating the parameters of all the branches of the network.

4.6 Comparison of Pose Estimation Model Architectures

The comparison of various pose estimation models used in this work are compared based on various criteria related to the architectural aspects is given in Table 4.2. Note that, in the subsequent chapters, more training configuration details and test results on various metrics will be discussed.

S.No.	Approach / Criterion	UnityVGG16	Pose6DSSD	DOSSE-6D_v1	DOSSE-6D_v2	DOSSE-6D_v3	AHR-DOSSE-6D
1.	**Type of Approach**	Template based	Correspondence based	Correspondence based	Correspondence based	Correspondence based	Correspondence based
2.	**Input Image Size (H × W)**	224 × 224	224 × 224	224 × 224	448 × 448	224 × 224	448 × 448
3.	**Feature Backbone**	VGG16 based	ResNet34 based	ResNet34 based	ResNet34 based	ResNet34 based	AHRNet
4.	**Fully Connected Layers Used**	Yes	No	No	No	No	No
5.	**End to End Trainable**	Yes	No	Yes	Yes	Yes	Yes
6.	**No. of Trainable Parameters**	27.594M	8.214M	8.214M	21.374M	8.215M	30.178M
7.	**Attention Mechanism Used**	None	None	None	Channel + Spatial	Channel + Spatial	Channel + Spatial
8.	**Transfer Learning Utilized**	Yes	Yes	Yes	No	No	No
9.	**Post-Refinement Stages Used**	No	No	No	No	No	No

Table 4.2: Table displaying the comparison of the Pose Estimation Models used in this work based on various architectural parameters.

Chapter 5

Training Configuration and Model Evaluation Setup

In the previous chapter, details regarding the architectures of various pose estimation models used in this approach were discussed. In this chapter, different attributes of the training configuration utilized such as experimental setup, loss function and optimizer details will be described. In addition, different aspects of the evaluation setup for testing the pose estimation models on *Unity Synthetic Data* on various evaluation metrics will be specified. In the next chapter, the actual quantitative results and inferences of object pose estimation based on these evaluation metrics will be provided along with graphical results. In the subsequent chapters, the benchmarking results of these models on a popular object pose estimation dataset will also be provided.

5.1 Experimental Configurations

The technique described in Chapter 3, Section 3.3 is used for collecting *domain randomized*, labelled data from both the simulation scenarios : *Simple Scene* and *Cluttered Scene* (see Chapter 1, Section 1.2). For both the scenarios, the following data split is used for all the models :

$$\{Training \mid Validation \mid Test\} \quad : \quad \{30000 \mid 3000 \mid 3000\} \quad \text{domain randomized RGB images}$$

For testing the performance of the model for estimating object pose, we consider the following cases :

(i) Training in Cluttered Scene + Testing in Cluttered Scene
(ii) Training in Simple Scene + Testing in Simple Scene
} Same Environment Cases

(iii) Training in Cluttered Scene + Testing in Simple Scene
(iv) Training in Simple Scene + Testing in Cluttered Scene
} Cross Environment Cases

The "Same Environment Cases" test the model performance with respect to the generalizability of its 6D pose predictions in a *known* environment, whereas, the "Cross" Environment Cases" test the model's robustness and its ability to generalize over both the environment and the 6D pose predictions. The corresponding quantitative and graphical results for these cases, for the pose estimation models described in Chapter 4, will be provided in Chapter 6.

5.2 Loss Functions

In this section, more details about the loss functions utilized for training the corresponding pose estimation models discussed in Chapter 4, will be given. Some of the common notations used for defining some terms in the loss functions are as follows :

$$
\begin{aligned}
\mathbf{x}_i \in \mathbb{R}^2 &\mapsto \text{Ground truth 2D image coordinates} \\
\mathbf{z}_i \in \mathbb{R}^3 &\mapsto \text{3D keypoint coordinates} \\
\boldsymbol{\pi}_i = \boldsymbol{\pi}\left(\mathbf{z}_i \mid \mathbf{y}, \mathbf{C}\right) \in \mathbb{R}^2 &\mapsto \text{Projected 2D image coordinates (using Pose Prediction)} \\
\mathbf{y} &\mapsto \text{Predicted 6D Pose of the object} \\
\mathbf{C} &\mapsto \text{Camera Intrinsic Matrix} \\
K &\mapsto \text{Number of Keypoints} \\
\mathbf{R}, \tilde{\mathbf{R}} \in \mathbb{R}^{3\times3} &\mapsto \text{Ground Truth and Predicted Rotation Matrices respectively} \\
\mathbf{T}, \tilde{\mathbf{T}} \in \mathbb{R}^{3\times1} &\mapsto \text{Ground Truth and Predicted Translational Vectors respectively} \\
\mathbf{v} &\mapsto \text{A point present in 3D object model (expressed in local coordinate frame)} \\
m &\mapsto \text{Number of Model points considered in the 3D model}
\end{aligned}
$$

- **Model-1 : UnityVGG16** (Chapter 4, Section 4.1) –

  A mixture loss function is used for training given by :

  $$\mathcal{L} = \lambda_{trans}\mathcal{L}_{trans} + \lambda_{orient}\mathcal{L}_{orient}$$

  → Here, $\mathcal{L}_{trans}$ and $\mathcal{L}_{orient}$ are the Mean Squared Errors (or equivalently, the squared L2 Norm) between the predicted translational vectors & ground truth translational vectors and the vectors representing the predicted & ground truth quaternions.

  → Also, equal weightage is given to both the terms $\implies$ $\lambda_{orient} = \lambda_{trans} = 1$.

- **Model-2 : Pose6DSSD** (Chapter 4, Section 4.3) –

  A mixture loss function is used for training given by :

  $$\mathcal{L} = \lambda_{reproj}\mathcal{L}_{reproj} + \lambda_{conf}\mathcal{L}_{conf}$$

  → $\mathcal{L}_{reproj}$ is the Mean Squared Error (MSE) between the predicted and true projected 2D image point (normalized) coordinates (Here, $K = 9$) :

  $$\mathcal{L}_{reproj} = \frac{1}{K}\sum_{i=1}^{K} \|\mathbf{x_i} - \boldsymbol{\pi_i}\|^2$$

  → $\mathcal{L}_{conf}$ is the Mean Squared Error (MSE) between the predicted and ground truth confidence values.

→ λ_{conf} is region-selective :

$$\lambda_{conf} = \begin{cases} 0.1 & \text{regions (grid cells) with no object,} \\ 5 & \text{regions (grid cells) with object} \end{cases}$$

More importance is given to grid cells with object of interest, whereas $\lambda_{reproj} = 1$ everywhere.

→ Note that for the first 15 epochs, confidence loss ($\mathcal{L}_{conf}$) is not considered (equivalently $\lambda_{conf} = 0$ for the first 15 epochs) to avoid unreliable confidence values in the initial stages.

- **Model-3 : DOSSE-6D** (Chapter 4, Section 4.4) –

A mixture loss function is used for training given by :

$$\mathcal{L} = \overbrace{\lambda_{reproj}\mathcal{L}_{reproj} + \lambda_{conf}\mathcal{L}_{conf}}^{\text{Indirect Supervision}} + \overbrace{\lambda_{add}\mathcal{L}_{add}}^{\text{Direct Supervision}}$$

→ $\mathcal{L}_{reproj}$ is the Mean Squared Error (MSE) between the predicted and true projected 2D image point (normalized) coordinates :

$$\mathcal{L}_{reproj} = \frac{1}{K}\sum_{i=1}^{K} \|\mathbf{x_i} - \boldsymbol{\pi_i}\|^2$$

→ $\mathcal{L}_{conf}$ is the Mean Squared Error (MSE) between the predicted and ground truth confidence values.

→ $\mathcal{L}_{add}$ is the average squared distance of 3D model points between predicted and ground truth configurations of the object.

$$\mathcal{L}_{\text{add}} = \frac{1}{m}\sum_{\mathbf{x}\in\mathcal{M}} \|(\mathbf{R}\mathbf{x} + \mathbf{T}) - (\tilde{\mathbf{R}}\mathbf{x} + \tilde{\mathbf{T}})\|^2$$

→ λ_{conf} is region-selective :

$$\lambda_{conf} = \begin{cases} 0.1 & \text{regions (grid cells) with no object,} \\ 5 & \text{regions (grid cells) with object} \end{cases}$$

More importance is given to grid cells with object of interest, whereas $\lambda_{reproj} = \lambda_{add} = 1$ everywhere.

→ Note that for the first 15 epochs, confidence loss ($\mathcal{L}_{conf}$) and add loss ($\mathcal{L}_{add}$) is not considered (equivalently $\lambda_{conf} = 0 \quad ; \quad \lambda_{add} = 0$ for the first 15 epochs) to avoid unreliable confidence values in the initial stages.

- **Model-4 : AHR-DOSSE-6D** (Chapter 4, Section 4.5) –

A mixture loss function is used for training given by :

$$\mathcal{L} = \overbrace{\lambda_{reproj}\mathcal{L}_{reproj} + \lambda_{heat}\mathcal{L}_{heat}}^{\text{Indirect Supervision}} + \overbrace{\lambda_{add}\mathcal{L}_{add}}^{\text{Direct Supervision}}$$

→ $\mathcal{L}_{heat}$ represents the multi-scale supervision. It is basically the Mean Squared Error (MSE) loss between predicted and ground truth heat maps :

$$\mathcal{L}_{\text{heat}} = \frac{1}{S}\sum_{s=1}^{S}\frac{1}{K}\sum_{k=1}^{K}\left\|\mathbf{H}_k^{s,pred} - \mathbf{H}_k^{s,\text{true}}\right\|_F^2$$

where

- The heatmap corresponding to the i^{th} keypoint and of scale (dimension) $\frac{\delta}{z}$ is :

$$\{\mathbf{H}_i^{j,pred}, \mathbf{H}_i^{j,true} \in \mathbb{R}^{\frac{\delta}{z}} \mid \frac{\delta}{z} :\equiv \left(\frac{H}{z} \times \frac{W}{z}\right), z = 4 \cdot 2^{j-1}\} \quad ; \quad i = 1, 2, \ldots, K \text{ and } j = 1, 2, \ldots, S$$

  Here, $K = 15$ and $S = 4$ is considered (see Chapter 4 for details.)
- $\|\cdot\|_F$ represents the Frobenius Norm of matrices.
- $(H \times W)$ is the input image dimension. Here $H = W = 448$ is considered.
- S represents the number of "scales" of heatmaps generated. Here, $S = 4$ is considered.

→ For the ground truth heatmap generation, we render a heatmap (of required size based on the scale value) of a gaussian blob with mean positioned at true 2D image keypoint and standard deviation of σ_{true}. Here, $\sigma_{true} = 1$ px is considered.

→ $\mathcal{L}_{reproj}$ is the Mean Squared Error (MSE) between the predicted and true projected 2D image point (normalized) coordinates :

$$\mathcal{L}_{reproj} = \frac{1}{K}\sum_{i=1}^{K}\|\mathbf{x_i} - \boldsymbol{\pi_i}\|^2$$

→ $\mathcal{L}_{add}$ is the average squared distance of 3D model points between predicted and ground truth configurations of the object.

$$\mathcal{L}_{\text{add}} = \frac{1}{m}\sum_{\mathbf{x}\in\mathcal{M}}\|(\mathbf{R}\mathbf{x} + \mathbf{T}) - (\tilde{\mathbf{R}}\mathbf{x} + \tilde{\mathbf{T}})\|^2$$

→ In this work, $\lambda_{add} = \lambda_{reproj} = \lambda_{heat} = 1$ is considered (equal weightage).

5.3 Optimizer Details

For training all the pose estimation models described in Chapter 4, the *ADAM* optimizer is considered with the following parameters :

$$\beta_1 = 0.9 \quad , \quad \beta_2 = 0.999$$

Note that, in this work checkpointing of models is considered, which is basically saving model files with weights. The checkpointing of the models is done at the epoch at which the model achieves best validation performance (upto that point).

This is done for training with both *Simple Scene* and *Cluttered Scene* synthetic data. The evaluation results presented in Chapter 6 and Chapter 7, based on various evaluation metrics (will be discussed next), are also based on these checkpointed models.

5.4 Training-Validation Curves

In this section, some of the curves corresponding to the training and validation of the pose estimation models are presented. Training curves related to the *Cluttered Scene* synthetic data and *Cluttered Scene* synthetic data are considered. The metrics plotted are average *quaternion error* (angle between predicted and ground truth quaternions) and average *translational error* (average mean squared error between predicted and ground truth translational vectors)[‡]

Model-1 : UnityVGG16

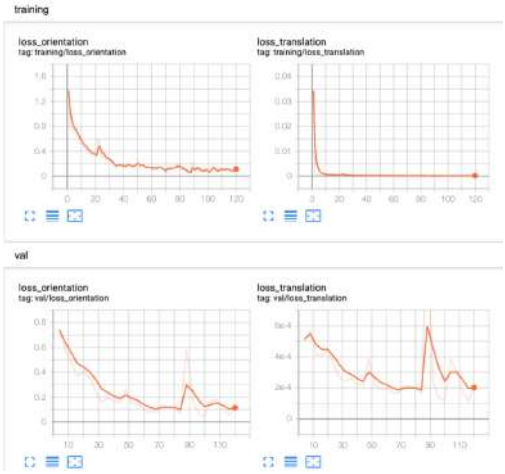


Fig. 5.1: Trained in *Simple Scene*

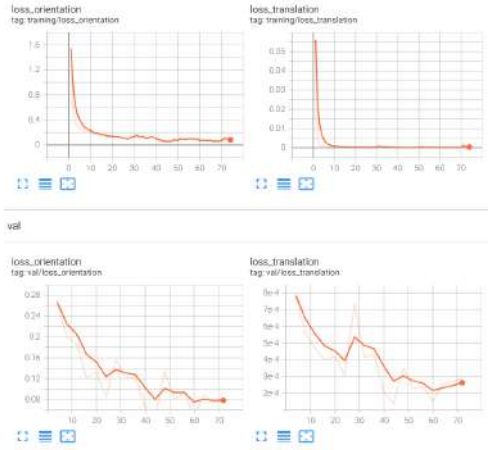


Fig. 5.2: Trained in *Cluttered Scene*

Model-2 : Pose6DSSD

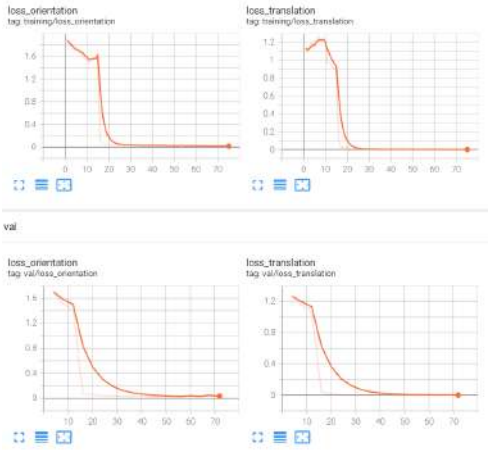


Fig. 5.3: Trained in *Simple Scene*

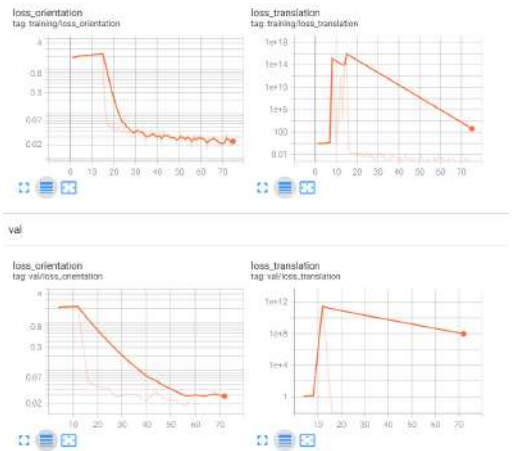


Fig. 5.4: Trained in *Cluttered Scene*

[‡]Note that, in some cases, the total loss (see Section 5.2) is also plotted.

Model-3 : DOSSE-6D (v2)[†]

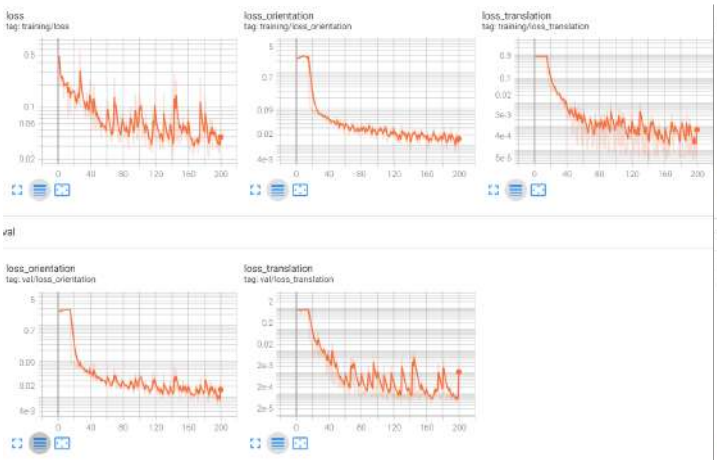


Fig. 5.5: Trained in *Simple Scene*

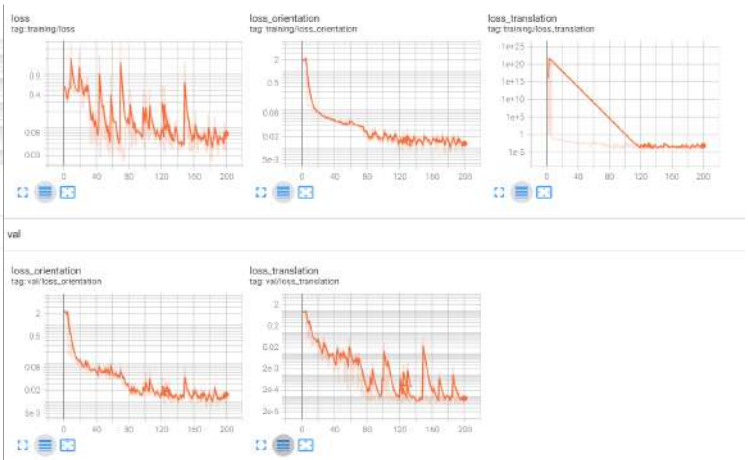


Fig. 5.6: Trained in *Cluttered Scene*

Model-4 : AHR-DOSSE-6D

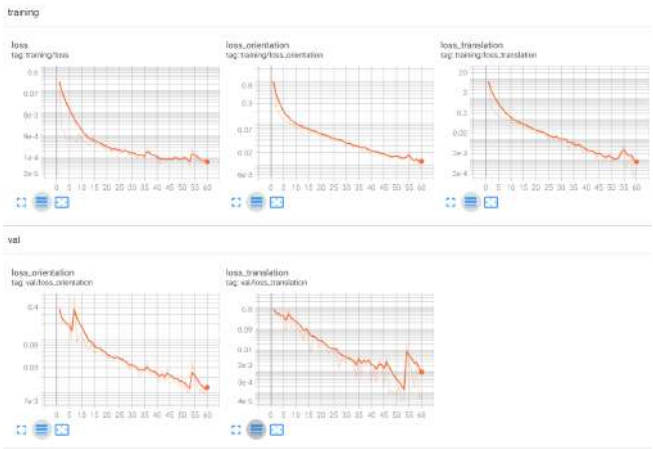


Fig. 5.7: Trained in *Simple Scene*

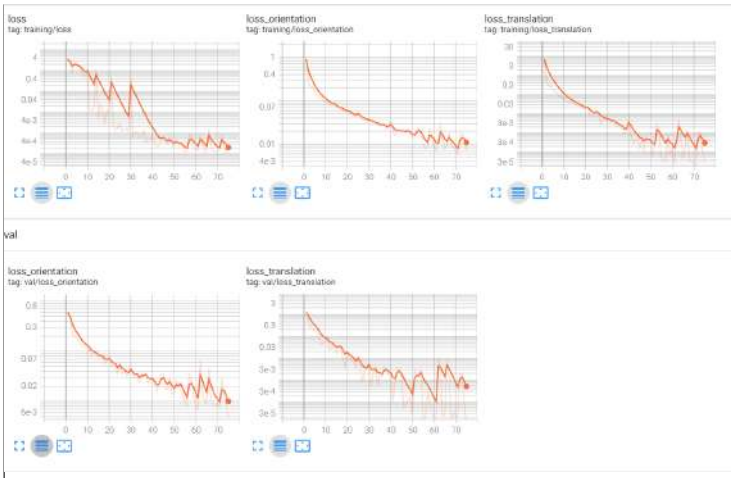


Fig. 5.8: Trained in *Cluttered Scene*

[†]v1, v3 are not represented here, so as to not increase the length of the report

5.5 Evaluation Metrics

In this section, the different evaluation metrics used for testing the object pose estimation performance of the different models presented in Chapter 4 are discussed. Note that, both quantitative and graphical results based on these metrics are presented in the next chapter along with some inferences.

We can quantify the model perfomance in absolute terms, on an average sense (over the test dataset). Some of the metrics have some popular thresholds for measuring performance in terms of pass rate. Metric Pass rate of a model corresponding to a predefined threshold, is the fraction (or equivalently percentage) of examples in the test dataset that have a metric score less than threshold. The threshold can be varied over a range, to get Pass Rate Curves, corresponding to the metric under consideration.

5.5.1 Average Distance (ADD) Metric

This metric measures the average distance of 3D model points between predicted and ground truth configurations of the object. It is measured in metres (m) or centimeters (cm).

$$\text{ADD} = \frac{1}{m} \sum_{\mathbf{x} \in \mathcal{M}} \|(\mathbf{R}\mathbf{x} + \mathbf{T}) - (\tilde{\mathbf{R}}\mathbf{x} + \tilde{\mathbf{T}})\|$$

where

$\mathbf{R}, \tilde{\mathbf{R}} \in \mathbb{R}^{3\times3} \mapsto$ Ground Truth and Predicted Rotation Matrices respectively

$\mathbf{T}, \tilde{\mathbf{T}} \in \mathbb{R}^{3\times1} \mapsto$ Ground Truth and Predicted Translational Vectors respectively

$\mathbf{v} \mapsto$ A point present in 3D object model (expressed in local coordinate frame)

$m \mapsto$ Number of Model points considered in the 3D model

The commonly used thresholds for measuring pass rate performance are 10%, 30% and 50% of the diameter (or in this case side length) of the object of interest (Here, *cube* object).

5.5.2 Reprojection Error Metric

This metric measures the average distance between projected ground truth and predicted 2D image (normalized) coordinates. It is measured in pixels (px).

$$\text{REPROJ} = \frac{1}{K} \sum_{i=1}^{K} \|\mathbf{x_i} - \boldsymbol{\pi_i}\|$$

where

$\mathbf{x}_i \in \mathbb{R}^2 \mapsto$ Ground truth 2D image coordinates

$\mathbf{z}_i \in \mathbb{R}^3 \mapsto$ 3D keypoint coordinates

$\boldsymbol{\pi}_i = \boldsymbol{\pi}(\mathbf{z}_i \mid \mathbf{y}, \mathbf{C}) \in \mathbb{R}^2 \mapsto$ Projected 2D image coordinates (using Pose Prediction)

$\mathbf{y} \mapsto$ Predicted 6D Pose of the object

$\mathbf{C} \mapsto$ Camera Intrinsic Matrix

$K \mapsto$ Number of Keypoints

The commonly used threshold for measuring pass rate performance is 5 px.

5.5.3 Translational MSE Metric

This metric measures the average distance mean squared error between the predicted and ground truth translational vectors. We can also it in terms of RMSE (root mean squared error)

$$\text{TMSE} = \frac{1}{3}\|\mathbf{T} - \tilde{\mathbf{T}}\|^2$$

where $\mathbf{T}, \tilde{\mathbf{T}} \in \mathbb{R}^{3\times 1} \mapsto$ Ground Truth and Predicted Translational Vectors respectively.

The thresholds for measuring pass rate performance are 10%, 30% and 50% of the diameter (or in this case side length) of the object of interest (Here, *cube* object).

5.5.4 Quaternion Error Metric

This metric measures the angular distance between ground truth and predicted quaternions. It is measured in radians (rad).

$$\text{QE} = \cos^{-1}\left(2\,|\langle \mathbf{q}, \tilde{\mathbf{q}}\rangle|^2 - 1\right)$$

where $\mathbf{q}, \tilde{\mathbf{q}} \in \mathbb{R}^{4\times 1} \mapsto$ Ground Truth and Predicted Quaternions respectively[§].

[§] $\langle . \rangle$ represents the inner product operator between vectors.

Chapter 6

Results and Inferences : Unity Simulation Scenarios

In this chapter, the results are provided for the test performance of the pose estimation models described in Chapter 4, on the Unity Synthetic data (see Chapter 3.3). The experimental configurations and evaluation metrics defined in Chapter 5, will be used to compare and analyze various aspects of the model performance.

Some more details, common for all the results in this chapter, are :

- **Object of Interest :** A coloured face *cube* object
- **Side Length (a) :** 10 cm

6.1 Average Distance (ADD) Metric : Results

6.1.1 Quantitative Results

The following table provides the results for the average (over the test set) ADD metric (in cm) for various models and in different experimental configurations :

S.No.	Approach \ Expt. Config.	Train-Clutter + Test-Clutter	Train-Clutter + Test-Simple	Train-Simple + Test-Simple	Train-Simple + Test-Clutter
1.	**UnityVGG16**	1.6801	16.5287	2.0248	53.7345
2.	**Pose6DSSD**	1.3976	9.0066	1.0054	39.0549
3.	**DOSSE-6D_v1**	1.2150	3.9213	0.9789	58.1505
4.	**DOSSE-6D_v2**	0.8836	10.5477	0.7604	41.8551
5.	**DOSSE-6D_v3**	0.9540	30.3129	1.0083	48.6070
6.	**AHR-DOSSE-6D**	0.4192	22.6130	0.4685	92.2395

Table 6.1: Table displaying the average ADD metric values (in cm). The model architectures and other details are described in Chapter 4 and Chapter 5.

The following table provides the results for the ADD metric, in terms of pass rates (in %), for various models and in different experimental configurations. For each Model :

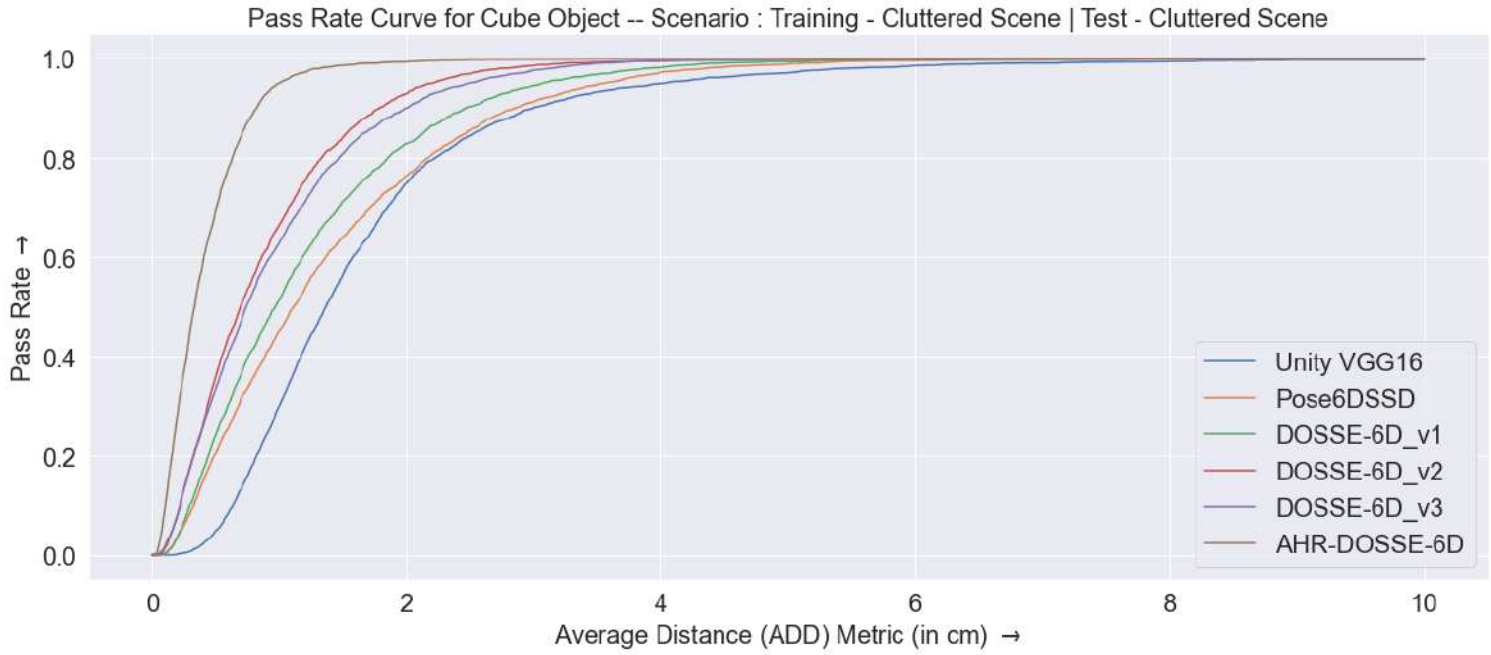
- 1^{st} sub-row corresponds to the 10% of the diameter (Here, side length) of object threshold for pass rate. It is represented in red colour.
- 2^{nd} sub-row corresponds to the 30% of the diameter (Here, side length) of object threshold for pass rate. It is represented in green colour.
- 3^{rd} sub-row corresponds to the 50% of the diameter (Here, side length) of object threshold for pass rate. It is represented in blue colour.

S.No.	Approach \ Expt. Config.	Train-Clutter + Test-Clutter	Train-Clutter + Test-Simple	Train-Simple + Test-Simple	Train-Simple + Test-Clutter
1.	UnityVGG16	30.10	0.00	33.33	0.00
		90.22	0.00	86.87	0.00
		97.13	0.00	92.73	0.00
2.	Pose6DSSD	45.15	12.10	62.77	3.90
		91.42	34.80	96.77	14.78
		98.97	51.60	99.50	21.92
3.	DOSSE-6D_v1	51.38	17.20	63.60	3.94
		94.59	60.43	96.97	15.98
		99.43	81.37	99.27	26.83
4.	DOSSE-6D_v2	66.53	5.30	74.63	0.43
		98.67	20.10	99.00	6.64
		99.83	39.33	99.90	10.34
5.	DOSSE-6D_v3	62.86	5.57	61.50	3.87
		97.63	18.93	96.53	16.95
		99.93	25.93	99.80	26.06
6.	AHR-DOSSE-6D	95.06	2.33	93.07	3.44
		99.87	10.13	98.87	12.11
		100.00	16.90	99.37	18.65

Table 6.2: Table displaying the ADD metric pass rates (in %). The model architectures and other details are described in Chapter 4 and Chapter 5.

6.1.2 Graphical Results

Train-Clutter + Test-Clutter



Train-Clutter + Test-Simple

Train-Simple+ Test-Simple

Train-Simple + Test-Clutter

6.2 Translational RMSE Metric : Results

6.2.1 Quantitative Results

The following table provides the results for the average (over the test set) translational RMSE metric (in cm) for various models and in different experimental configurations :

S.No.	Expt. Config. / Approach	Train-Clutter + Test-Clutter	Train-Clutter + Test-Simple	Train-Simple + Test-Simple	Train-Simple + Test-Clutter
1.	**UnityVGG16**	1.1845	9.8310	1.3502	32.2180
2.	**Pose6DSSD**	1.0119	16.4012	0.7571	46.6069
3.	**DOSSE-6D_v1**	0.8790	8.5814	0.7445	73.1984
4.	**DOSSE-6D_v2**	0.6463	16.9794	0.6665	0.4346
5.	**DOSSE-6D_v3**	0.6996	29.4211	0.7475	55.5968
6.	**AHR-DOSSE-6D**	0.3054	19.570	0.6678	82.1462

Table 6.3: Table displaying the average Translational RMSE metric values (in cm). The model architectures and other details are described in Chapter 4 and Chapter 5.

The following table provides the results for the translational RMSE metric, in terms of pass rates (in %), for various models and in different experimental configurations. For each Model :

- 1^{st} sub-row corresponds to the 10% of the diameter (Here, side length) of object threshold for pass rate. It is represented in red colour.
- 2^{nd} sub-row corresponds to the 30% of the diameter (Here, side length) of object threshold for pass rate. It is represented in green colour.
- 3^{rd} sub-row corresponds to the 50% of the diameter (Here, side length) of object threshold for pass rate. It is represented in blue colour.

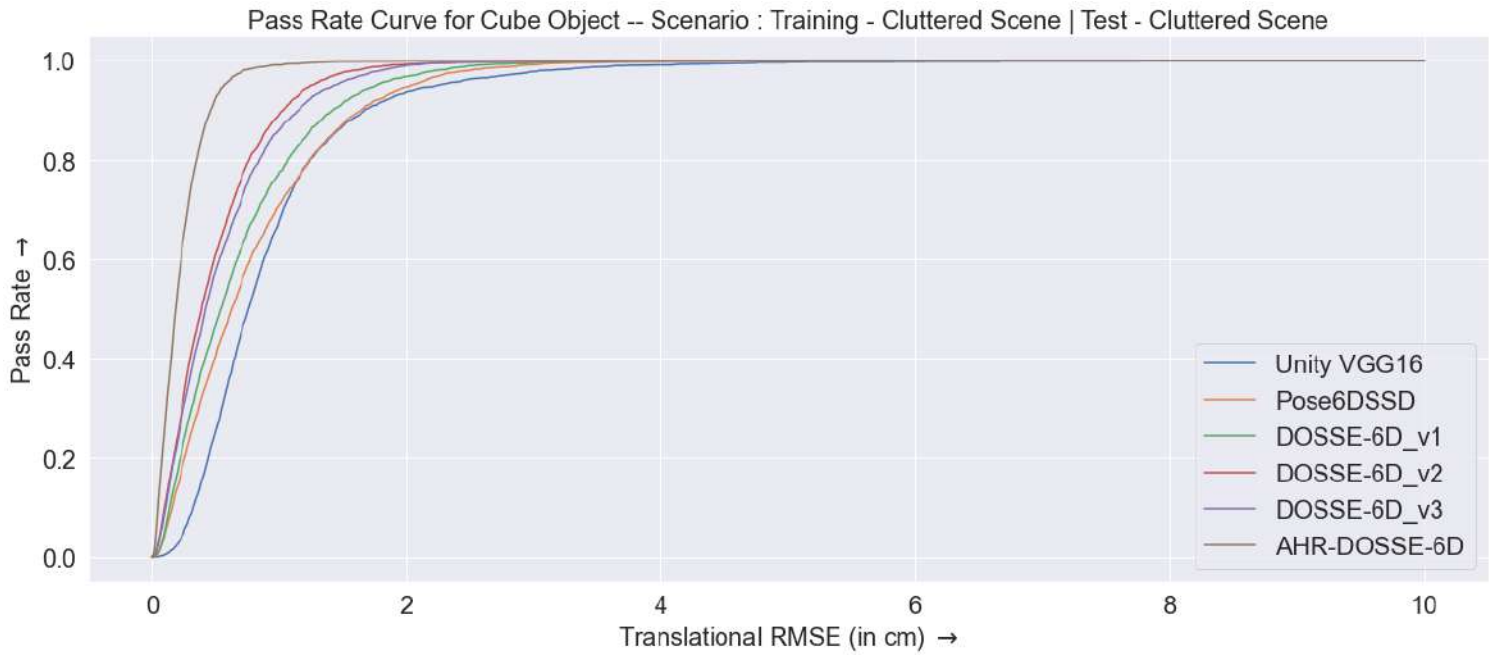
S.No.	Expt. Config. / Approach	Train-Clutter + Test-Clutter	Train-Clutter + Test-Simple	Train-Simple + Test-Simple	Train-Simple + Test-Clutter
1.	**UnityVGG16**	67.93	0.00	70.23	0.00
		97.80	0.00	96.03	0.00
		99.70	0.07	98.93	0.00
2.	**Pose6DSSD**	70.94	22.23	86.10	8.78
		99.20	53.10	99.53	23.12
		99.93	63.47	99.93	37.30
3.	**DOSSE-6D_v1**	77.54	35.93	87.60	8.74
		99.57	84.87	99.43	27.79

		100.00	96.97	99.97	45.65
4.	**DOSSE-6D_v2**	89.26	10.90	94.10	3.87
		99.87	41.97	99.93	10.98
		100.00	63.57	99.97	19.19
5.	**DOSSE-6D_v3**	86.02	11.60	84.60	9.84
		99.93	26.47	99.87	27.33
		100.00	28.73	100.00	38.84
6.	**AHR-DOSSE-6D**	99.13	5.00	97.33	7.11
		100.00	17.53	99.40	19.39
		100.00	29.00	99.80	28.10

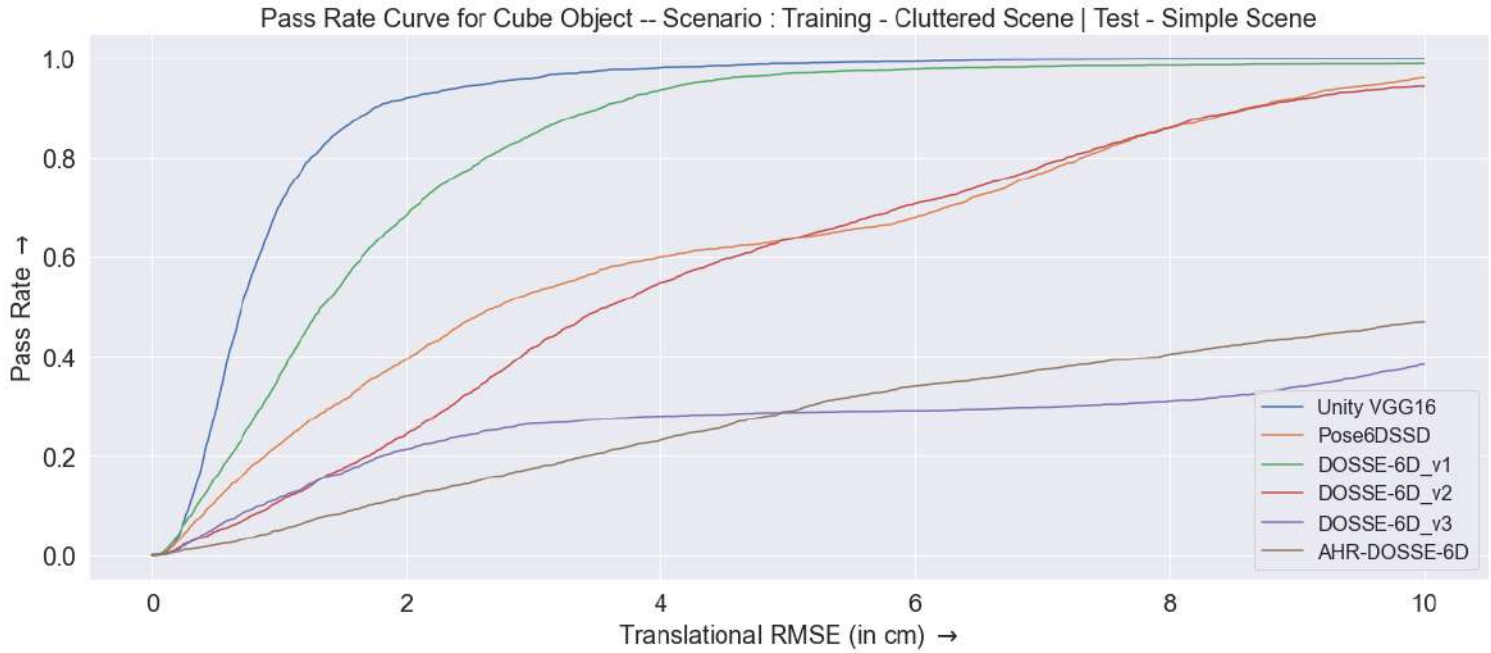
Table 6.4: Table displaying the Translational RMSE metric pass rates (in %). The model architectures and other details are described in Chapter 4 and Chapter 5.

6.2.2 Graphical Results

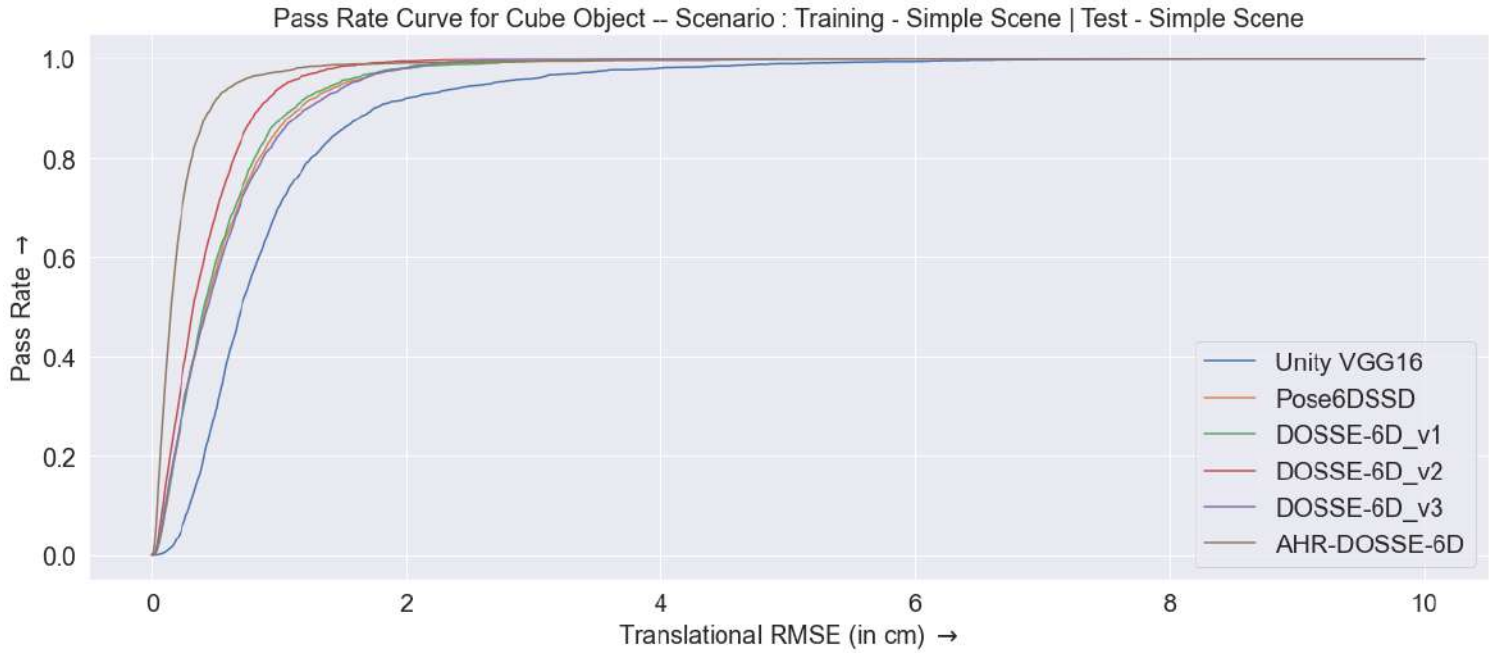
Train-Clutter + Test-Clutter



Train-Clutter + Test-Simple



Train-Simple+ Test-Simple



Train-Simple + Test-Clutter

6.3 Reprojection Error Metric : Results

6.3.1 Quantitative Results

The following table provides the results for the average (over the test set) reprojection error metric (in px) for various models and in different experimental configurations :

S.No.	Approach \ Expt. Config.	Train-Clutter + Test-Clutter	Train-Clutter + Test-Simple	Train-Simple + Test-Simple	Train-Simple + Test-Clutter
1.	**UnityVGG16**	2.2855	6.5294	2.7313	30.5473
2.	**Pose6DSSD**	0.5149	1.9097	0.4256	51.9298
3.	**DOSSE-6D_v1**	0.5258	1.9551	0.4251	52.8834
4.	**DOSSE-6D_v2**	0.3660	5.2495	0.3101	21.2240
5.	**DOSSE-6D_v3**	0.3360	14.7534	0.3759	60.2965
6.	**AHR-DOSSE-6D**	0.1972	3.6707	0.2395	112.58

Table 6.5: Table displaying the average Reprojection Error metric values (in px). The model architectures and other details are described in Chapter 4 and Chapter 5.

S.No.	Expt. Config. / Approach	Train-Clutter + Test-Clutter	Train-Clutter + Test-Simple	Train-Simple + Test-Simple	Train-Simple + Test-Clutter
1.	**UnityVGG16**	93.69	46.90	89.90	0.27
2.	**Pose6DSSD**	100.00	93.73	100.00	72.54
3.	**DOSSE-6D_v1**	100.00	94.37	100.00	71.20
4.	**DOSSE-6D_v2**	100.00	95.27	100.00	74.11
5.	**DOSSE-6D_v3**	100.00	71.10	100.00	73.04
6.	**AHR-DOSSE-6D**	100.00	93.90	99.97	61.16

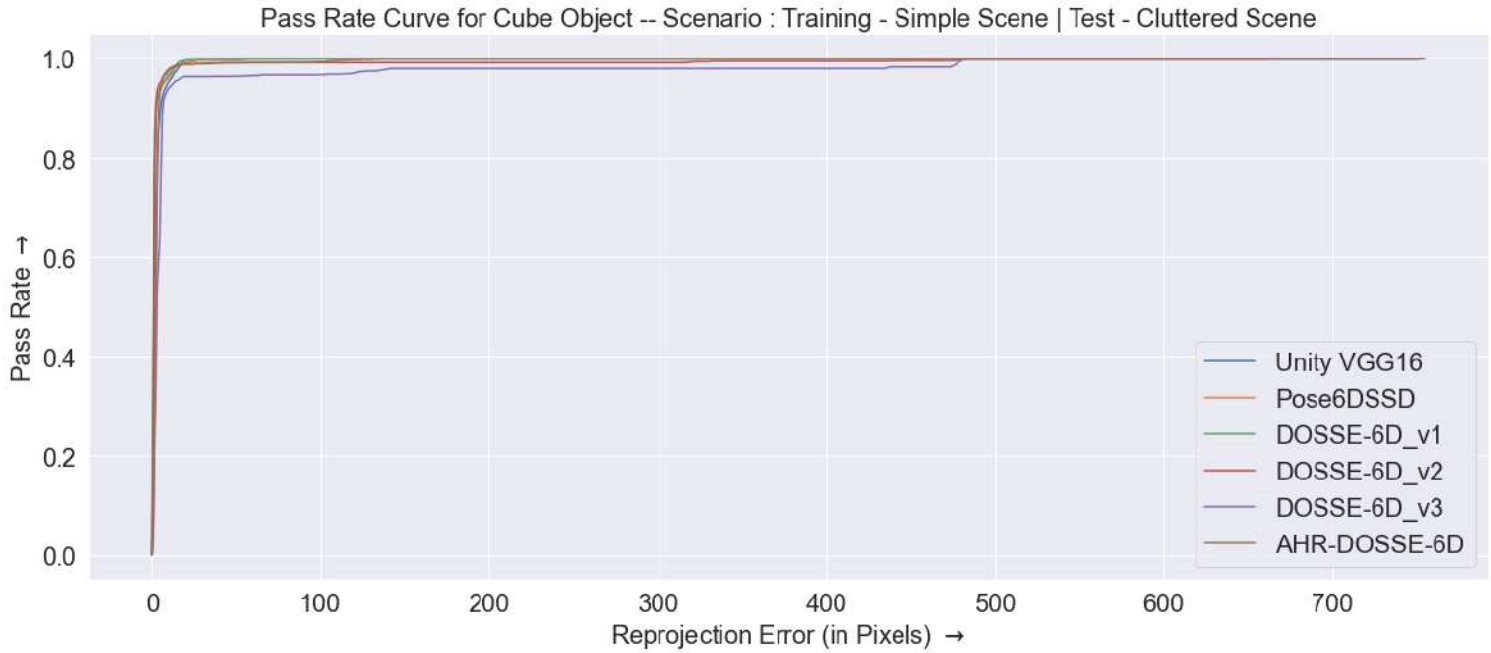
Table 6.6: Table displaying the Reprojection Error metric pass rates (in %) – 5 px threshold. The model architectures and other details are described in Chapter 4 and Chapter 5.

The following table provides the results for the reprojection error metric, in terms of pass rates (in %), for various models and in different experimental configurations. The threshold considered is 5 px.

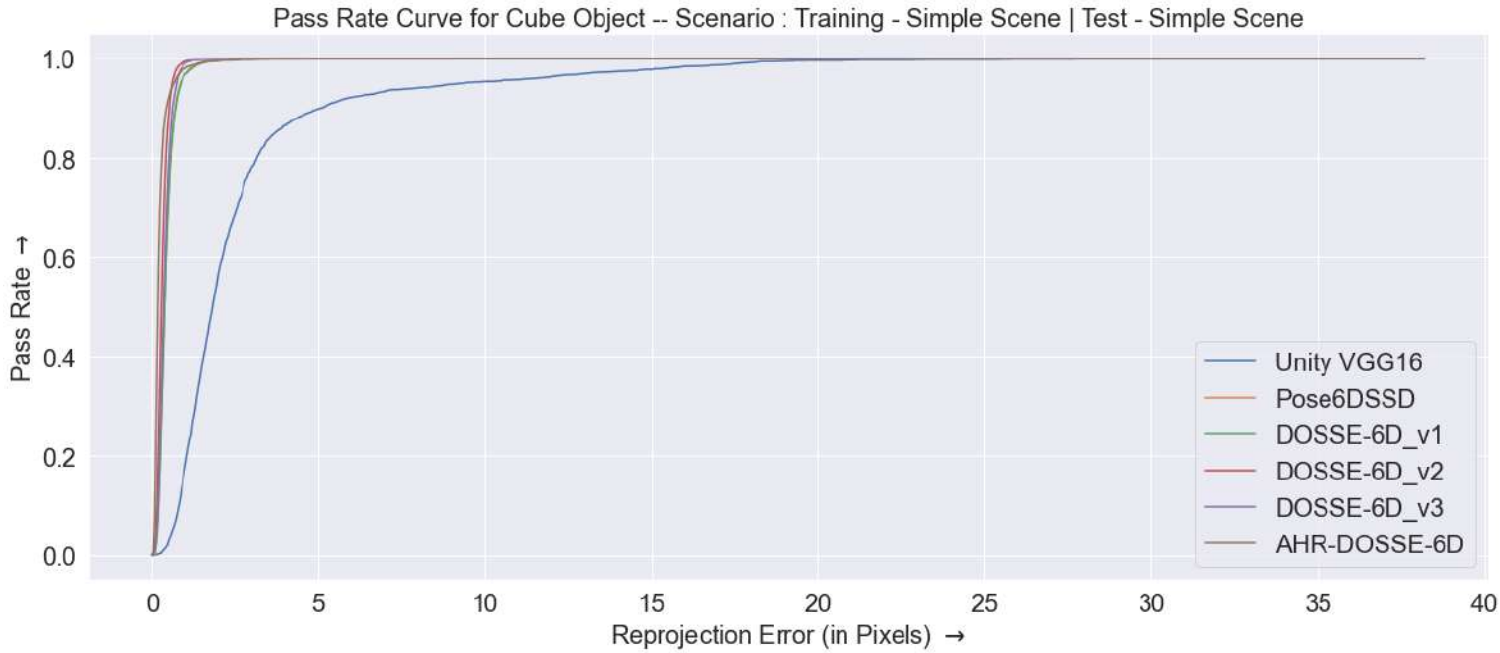
6.3.2 Graphical Results

Train-Clutter + Test-Clutter

Train-Clutter + Test-Simple



Train-Simple+ Test-Simple



Train-Simple + Test-Clutter

6.4 Quaternion Error Metric : Results

6.4.1 Quantitative Results

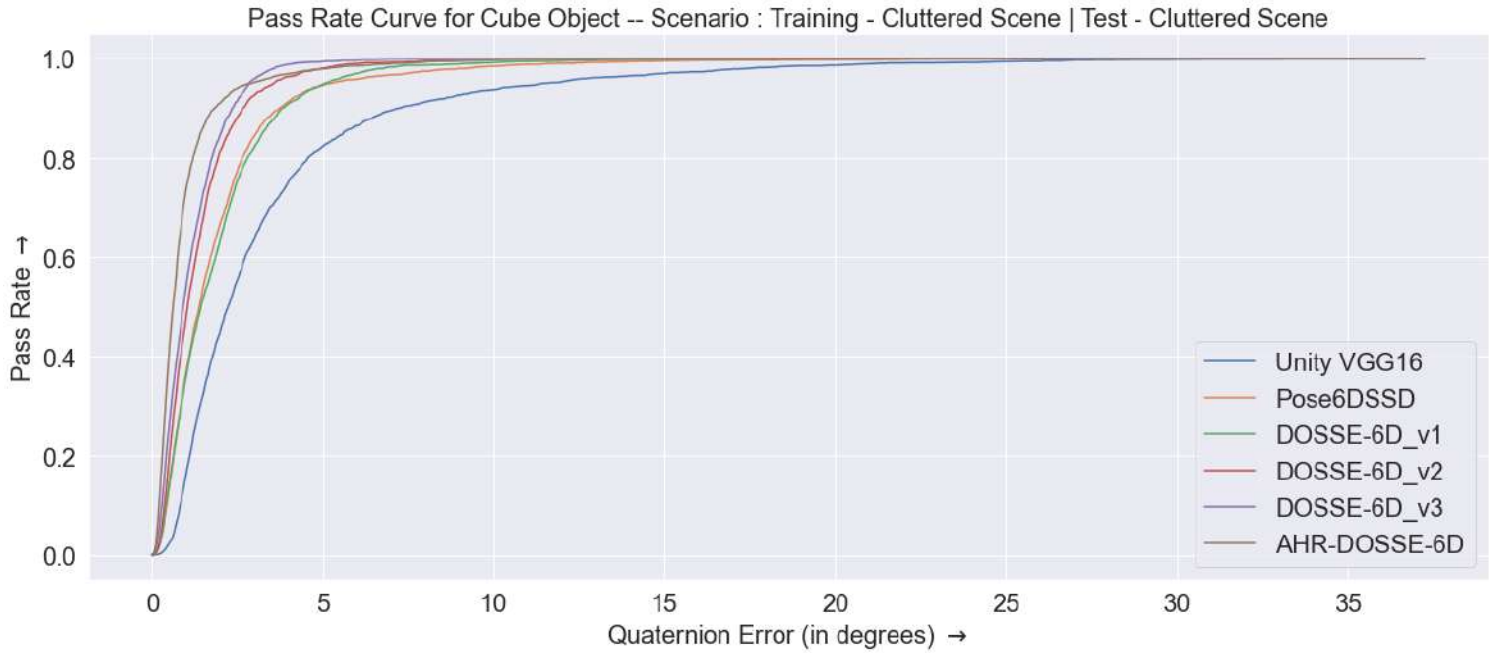
The following table provides the results for the average (over the test set) quaternion error (in rad) for various models and in different experimental configurations :

S.No.	Approach \ Expt. Config.	Train-Clutter + Test-Clutter	Train-Clutter + Test-Simple	Train-Simple + Test-Simple	Train-Simple + Test-Clutter
1.	**UnityVGG16**	0.0468	0.1930	0.1381	0.2524
2.	**Pose6DSSD**	0.0140	0.1441	0.0101	0.4927
3.	**DOSSE-6D_v1**	0.0175	0.1298	0.0133	0.5962
4.	**DOSSE-6D_v2**	0.0076	0.1365	0.0059	0.4120
5.	**DOSSE-6D_v3**	0.0045	0.26	0.0102	0.5091
6.	**AHR-DOSSE-6D**	0.0049	0.1429	0.0104	0.9139

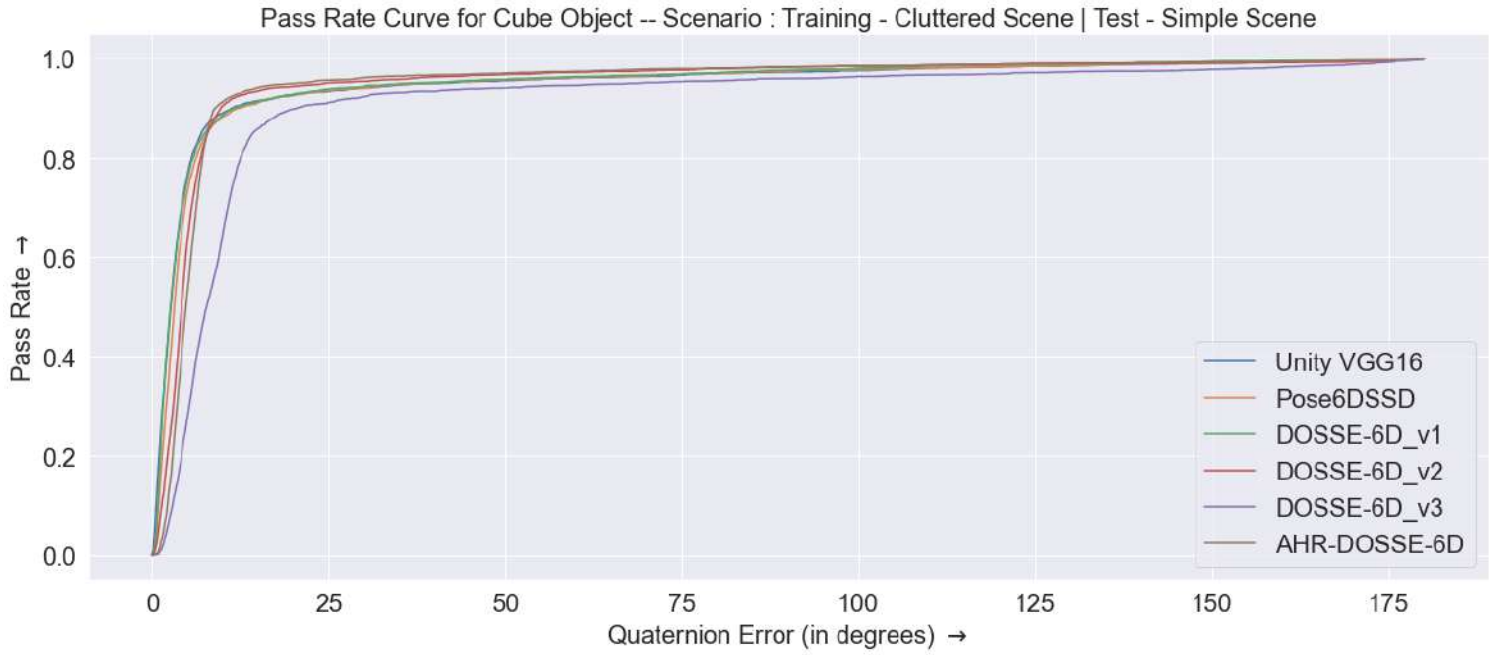
Table 6.7: Table displaying the average Quaternion Error metric values (in rad). The model architectures and other details are described in Chapter 4 and Chapter 5.

6.4.2 Graphical Results

Train-Clutter + Test-Clutter

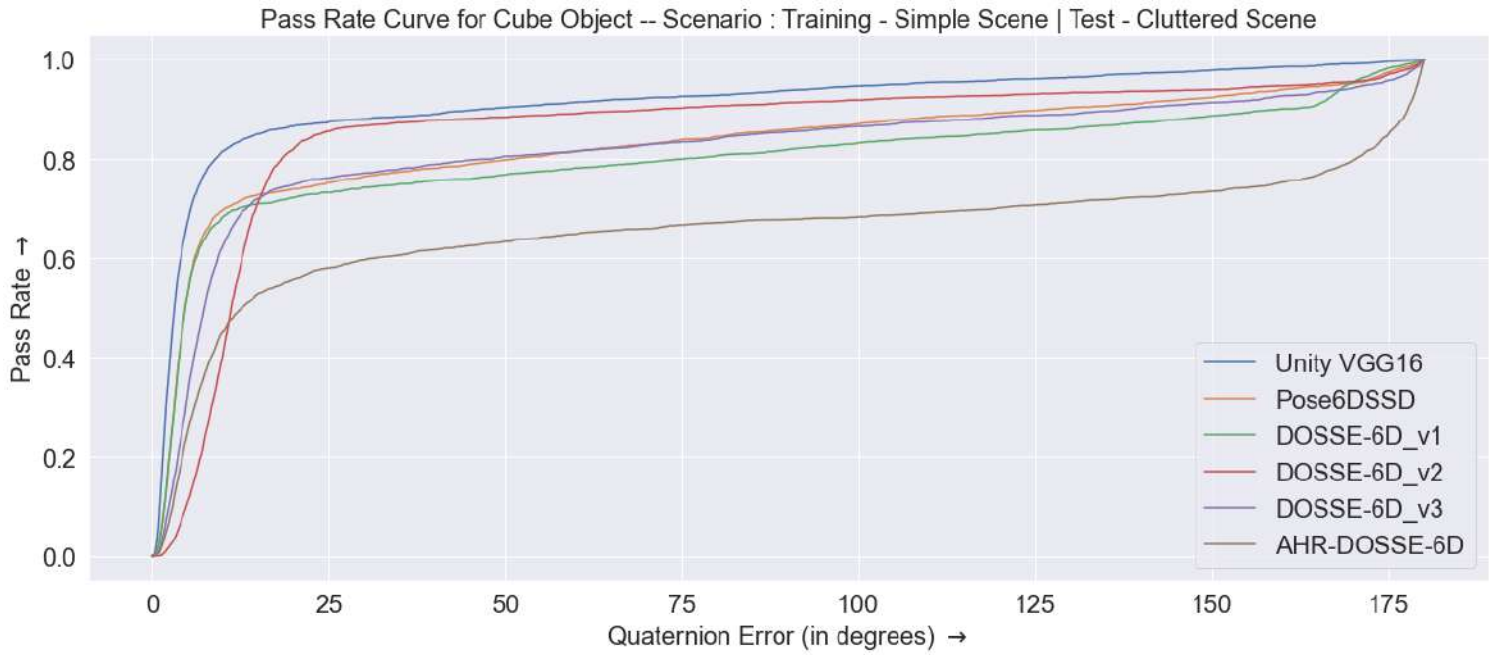


Train-Clutter + Test-Simple



Train-Simple+ Test-Simple

Train-Simple + Test-Clutter



6.5 Inferences

Now, that the quantitative results and graphical results of different metrics are given, in this section, some of the inferences[‡] extracted from these test experiments will be discussed. Note that more results and inferences based on testing the some of the models on a benchmark dataset will be discussed in the next chapter.

1. **Model Performance in *Same Environment Cases* :**

   → We can observe that from results for the *Same Environment Cases*, the *AHR-DOSSE-6D* model performs the *best* as compared to the other models. This can be observed by examining Table 6.1, Table 6.2 and also the relevant pass rate curves shown in Section 6.1.2. The improvement in performance can be attributed to the following factors (see Chapter 4, Section 4.5) :

   - Maintenance of high resolution hidden representations in the AHRNet backbone
   - Incorporation of attention module
   - Increased input image resolution
   - Use of more geometrical details by considering keypoints that lie on the object's surface instead of the 3D bounding box points, which is the case in *Pose6DSSD* and *DOSSE-6D*.

   → In general, the performance of *DOSSE-6D* (all versions) models is higher than that of *Pose6DSSD* and *UnityVGG16* model. This can attributed to the use of Direct Supervision (see Chapter 5, Section 5.2), which was made possible due to the incorporation of BPnP module (see Chapter 4,Section 4.4.1).

   → Also, the performance of *UnityVGG16* model is not as high as compared to the remaining models. This shows the merit in using *correspondence-based* approaches rather than using template-based approaches.

   → Note that, good performance in the *Same Environment Cases*, indicates that the model is model is able to generalize well with respect to object pose prediction in a "known" environment.

2. **Model Performance in *Cross Environment Cases* :**

   → Firstly, it can be seen that the all the models perform poorly as compared to the *Same Environment Cases* as expected, due to the fact that it is a much difficult problem. Here, essentially the "transfer" of pose prediction performance across environments is being tested.

   → As expected the generalization capability of the models trained in *Cluttered Scene* is higher, i.e. these models are more robust to changes in environment, due to presence of background distractor objects.

   → Transfer Learning is utilized in models *UnityVGG16*, *Pose6DSSD*, *DOSSE-6D_v1* as discussed in Chapter 4. Specifically, all these models consisted of a ResNet34 based feature extraction backbone, whose weights were initialized by pretraining on the ImageNet dataset for classification (a different task). Interestingly, it can be observed such pretraining is highly beneficial for improving generalization ability of the model across environments and leads to improvement in the *Cross Environment* performance of the models. Therefore, the declined performance of *DOSSE-6D_v2*, *DOSSE-6D_v3* and *AHR-DOSSE-6D* can be attributed to the fact that transfer learning is not utilized in these models.

[‡]Note that, generally in object pose estimation, the Average Distance Metric (see Chapter 5, Section 5.5.1) is considered to be the most important and relevant metric for quantifying the model performance.

→ Note that, good performance in the *Cross Environment Cases*, indicates that the model is model is able to generalize well with respect to the environment of the object as well as the object pose prediction.

3. **Input Image Resolution :**

   From the results, it can be inferred that the use of higher resolution input images tends to improve the prediction performance of the models. This can be seen by observing the enhanced performance of *DOSSE-6D_v2* as compared to *DOSSE-6D_v1* and *DOSSE-6D_v3* (see Table 6.1 and Table 6.2), which use an input image resolution of 224×224. The same idea is considered in *AHR-DOSSE-6D* model as well. Also, use of higher input image resolution RGB tends to an improvement in reprojection error metric performance for both *Same Environment* and *Cross Environment* cases.

4. **Incorporation of Attention :**

   The incorporation of attention module consisting of both spatial and channel attention sub-modules also proves to be useful, as can be seen by comparing the increased performance of *DOSSE-6D_v3* to *DOSSE-6D_v1* model. The same idea again used in *DOSSE-6D_v2* and *AHR-DOSSE-6D* model along with increased input resolution leads to pose prediction enhancememt.

5. **Parameter Efficiency :**

   → It can be observed from Table 4.2 given in Chapter 4, that the number of trainable parameters in *Pose6DSSD* (8.214M) has $3.36X$ less parameters than *UnityVGG16* model, even though performing much better than it. This reinforces the fact that correspondence-based methods are much better suited for object Pose Estimation.

   → For increasing the model's input image resolution, for *DOSSE-6D* (from $v1 \rightarrow v2$), more convolutional layers were added making the model *deeper* (i.e increasing total number of parameters). *AHR-DOSSE-6D*, on the other hand, is parameter-efficient in the sense that we can increase the input image resolution (because it is shown to improve performance) without an increase in the trainable weights[†]. The only difference arises in the output heatmap (set) dimensions, which is not an issue, as DSNT block (see Chapter 4, Section 4.5.4) takes in these heatmaps and regresses "normalizes" 2D image coordinates.

[†]One of the experimental configurations in Chapter 7 clearly demonstrates this idea.

Chapter 7

Benchmarking Model Performance : LINEMOD Dataset

In the previous chapter, the results of pose estimation models considered in this work (see Chapter 4) evaluated on the synthetic datasets collected from *Unity Simulation scenarios* (see Chapter 3) were presented along with inferences. In this chapter, the results and corresponding analyses of some of the models (that displayed better pose estimation performance than the others on the Unity synthetic data) tested on a popular benchmark dataset, LINEMOD [10], for object pose estimation, using RGB images only.

7.1 Basic Description of the Dataset

The LINEMOD [10] dataset is a standard benchmark dataset for 6D object pose estimation of objects in cluttered scenes. The dataset contains poorly textured objects in a cluttered scene. The dataset contains many object sequences (see Table 7.1). The RGB images (640×480) in the dataset contain multiple objects, but every image of a particular object sequence, has only one object of interest. In Table 7.1, the rows highlighted in yellow, represent the objects which are considered for testing the performance in this work.

Along with each RGB, correspoding ground truth pose information (Rotation + Translation) is provided for use in supervised learning setups. For each object, an ID.No. is allocated and a 3D model saved as a point cloud is given.

Object	**Diameter (in m)**
Iron	0.3032
Benchvise	0.2869
Lamp	0.2852
Can	0.202
Cat	0.155
Driller	0.262
Duck	0.109
Egg	0.1764
Glue	0.176
Holepuncher	0.162
Ape	0.103
Cam	0.173
Phone	0.213

Table 7.1: Table displaying the commonly used object sequences for 6D pose estimation in LINEMOD dataset.

Also, the internal camera matrix parameters (camera intrinsic matrix) are also made available.

$$C_{intrinsic} = \begin{bmatrix} 572.4114 & 0 & 325.2611 \\ 0 & 573.5704 & 242.0489 \\ 0 & 0 & 1 \end{bmatrix}$$

There are around 15783 images in total and nearly 1200 images per object.

7.2 Training Configuration Details

In this section, a brief description of the training configuration, used for benchmarking, is given.

7.2.1 Dataset Augmentation

Since the amount of data available (in terms of number of images per object), it becomes necessary to perform data augmentation to effectively increase the amount of training data available.

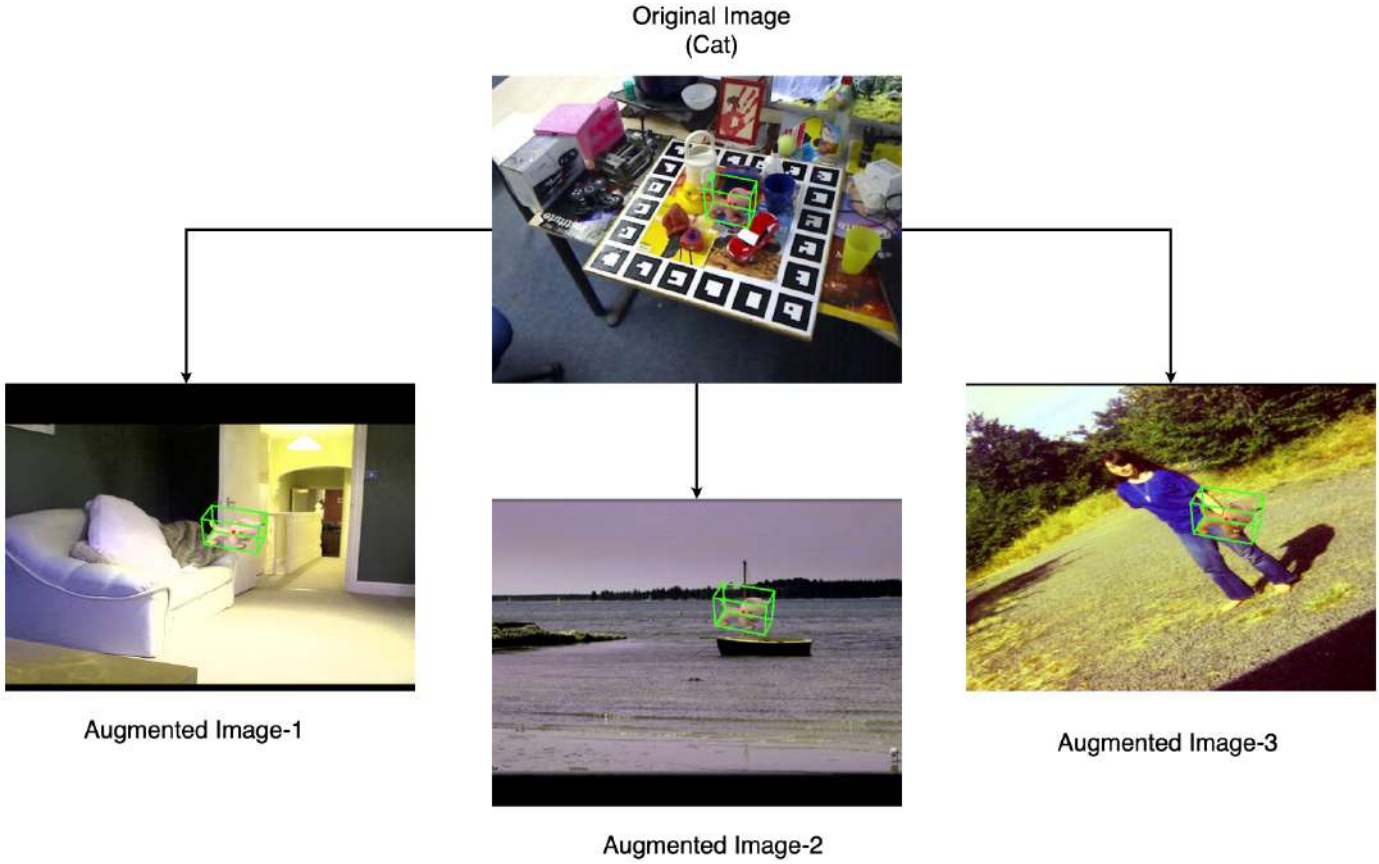


Fig. 7.1: An example of Data Augmentation used in this work. The bounding box around the object of interest (Here, cat) is shown, in green. The centroid of the object is indicated using a red point.

[19] provide segmentation maps (for the object of interest) for each RGB image in every object sequence. These maps are utilized and the following tasks are perfomed , during training the model, for augmenting the dataset :

- Randomly change the object's background by considering images from the PASCAL VOC dataset [8].

- Change the hue, exposure and saturation parameters of the image randomly
- Scale, translate and distort the image slightly.

The use of such data augmentation, also provides an additional benefit to improve the generalization capability of the models, by avoiding background overfitting.

7.2.2 Experimental Setup

For all the experiments, the following data split (popularly used in many research papers) is considered, for each object sequence[†] :

$$\{\textit{Training} \mid \textit{Validation} \mid \textit{Test}\} \quad : \quad \{15\ \% \mid \sim 12.75\ \% \mid \sim 72.25\ \%\} \quad \text{RGB images}$$

The following approaches, based on some pose estimation models described in Chapter 4, are considered for training and evaluation on the benchmark dataset :

(i) *DOSSE-6D_v1*
(ii) *DOSSE-6D_v2*
} (Chapter 4, Section 4.4)

(iii) *AHR-DOSSE-6D_LR*
(iv) *AHR-DOSSE-6D_HR*
} (Chapter 4, Section 4.5)

Note :

→ *AHR-DOSSE-6D_LR* and *AHR-DOSSE-6D_HR* utilize the same pose estimation model *AHR-DOSSE-6D*, but the difference lies in the resolution of the image which is given as an input for object pose estimation to the network. In the case of *AHR-DOSSE-6D_LR*, the input RGB image size is 224×224 px, whereas, for *AHR-DOSSE-6D_HR*, the input image size is 448×448 px.

→ The number of parameters in the pose estimation model for *AHR-DOSSE-6D_HR* and *AHR-DOSSE-6D_LR* remains the same (unlike in *DOSSE-6D* model). The difference arises in the output heatmap (set) dimensions, which is not an issue, as DSNT block (see Chapter 4, Section 4.5.4) takes in these heatmaps and regresses "normalizes" 2D image coordinates.

→ It should be noted that the keypoints considered for *DOSSE-6D_v1* and *DOSSE-6D_v2* models are $K = 9$ in number, corresponding to the 8 corners of the 3D bouunding box and the centroid of the object. [19] provide a processed version of the dataset with normalized 2D co-ordinates (vector of length $2 \times 9 = 18$) for each image, which form the ground teuth labels.

→ As mentioned in Chapter 4, Section 4.5, more geometrical details are considered while training *AHR-DOSSE-6D* models. So, in this case, we choose the keypoints (K_{LM}) as follows [Here, $K_{LM} = 16$] :

- $(K_{LM} - 1)$ points are selected based on the Farthest Point Sampling (FPS) algorithm [14]. All of these points lie on the object's surface, thus providing more information about the geometrical shape of the object (which might be useful).

[†]Note that, in this work, the main focus is *single object, single instance* pose estimation

- The remaining point chosen is the Object Centroid.

It should be kept in mind that the details, other than the above-mentioned aspects, such as the *Loss Functions*, *Optimizer setup* and the *Model Evaluation metrics* remain the same as discussed in Chapter 5.

7.3 Average Distance (ADD) Metric : Results

7.3.1 Quantitative Results

The following table provides the results for the average (over the test set) ADD metric (in cm) for various models and objects :

S.No.	Object / Approach	Cat *(d = 15.50 cm)*	Benchvise *(d = 28.69 cm)*	Lamp *(d = 28.52 cm)*	Can *(d = 20.20 cm)*	Iron *(d = 30.32 cm)*
1.	**DOSSE-6D_v1**	3.1831	1.7091	2.1349	2.1471	3.2184
2.	**DOSSE-6D_v2**	2.0117	1.1971	1.7420	1.6979	2.0958
3.	**AHR-DOSSE-6D_LR**	2.5921	1.2866	1.2467	1.2626	1.9153
4.	**AHR-DOSSE-6D_HR**	1.9293	3.1612	1.2967	0.8502	2.5020

Table 7.2: Table displaying the average ADD metric values (in cm). The model architectures and other details are described in Chapter 4 and Chapter 5.

The following table provides the results for the ADD metric, in terms of pass rates (in %), for various models and objects. For each Model :

- 1^{st} sub-row corresponds to the 10% of the diameter of object threshold for pass rate. It is represented in red colour.

S.No.	Object / Approach	Cat *(d = 15.50 cm)*	Benchvise *(d = 28.69 cm)*	Lamp *(d = 28.52 cm)*	Can *(d = 20.20 cm)*	Iron *(d = 30.32 cm)*
1.	**DOSSE-6D_v1**	33.45	86.77	74.94	60.19	60.22
		80.52	99.20	99.10	96.07	97.72
		94.37	99.54	100.00	99.42	99.40
2.	**DOSSE-6D_v2**	50.23	94.53	85.55	78.01	82.45
		92.96	99.32	99.89	98.96	99.28
		99.18	100.00	100.00	99.77	99.64
3.	**AHR-DOSSE-6D_LR**	45.8920	94.30	94.36	84.14	88.10
		88.97	99.77	100.00	98.96	99.16
		97.89	99.89	100.00	99.77	99.52
4.	**AHR-DOSSE-6D_HR**	68.31	96.69	97.86	95.02	93.63
		98.47	98.63	99.89	99.42	98.92
		99.53	98.75	99.89	99.77	99.16

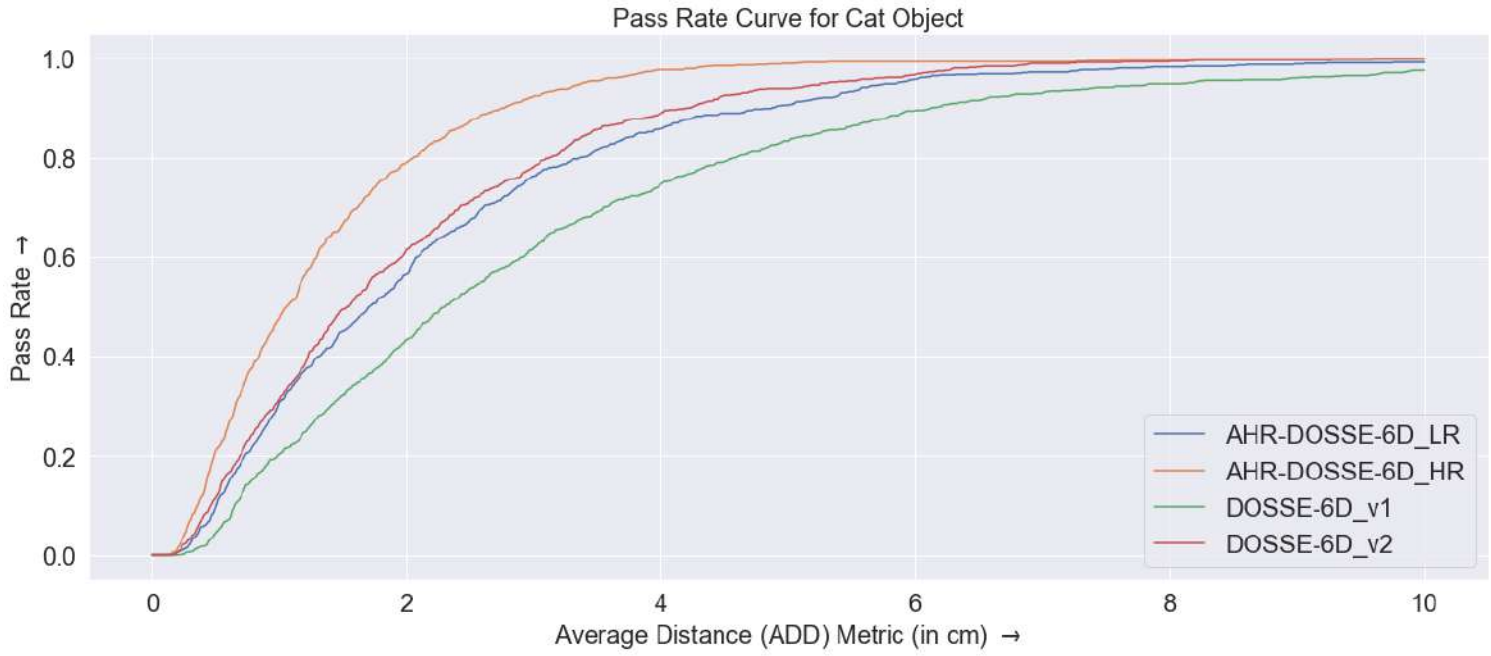
Table 7.3: Table displaying the ADD metric pass rates (in %). The model architectures and other details are described in Chapter 4 and Chapter 5.

- 2^{nd} sub-row corresponds to the 30% of the diameter of object threshold for pass rate. It is represented in green colour.

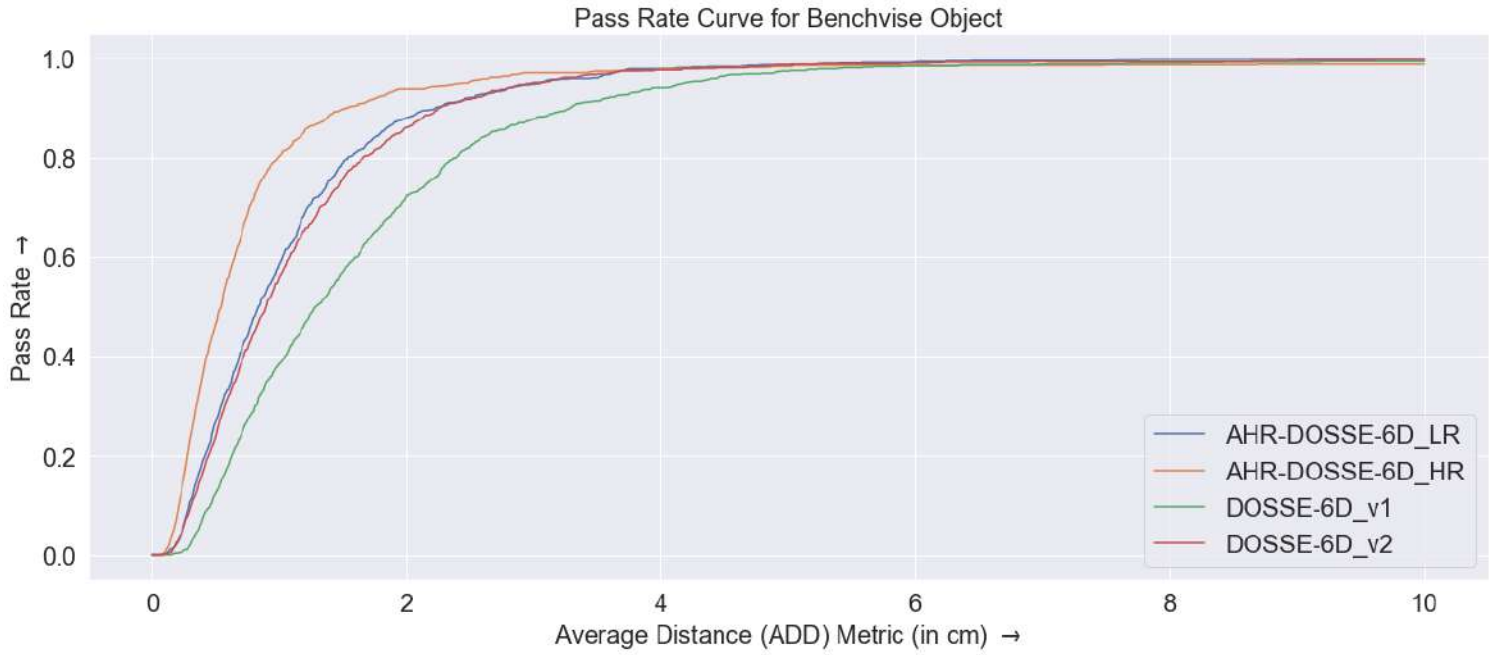
- 3^{rd} sub-row corresponds to the 50% of the diameter of object threshold for pass rate. It is represented in blue colour.

7.3.2 Graphical Results

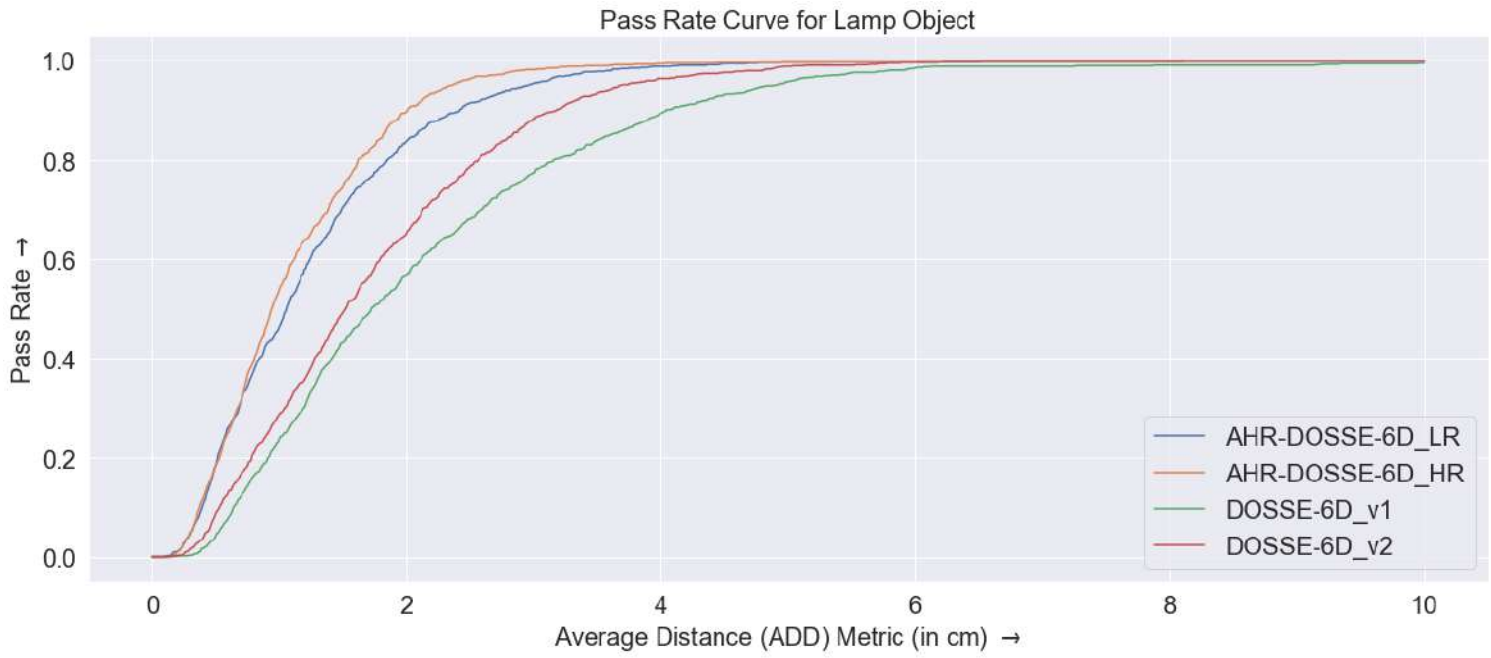
Cat



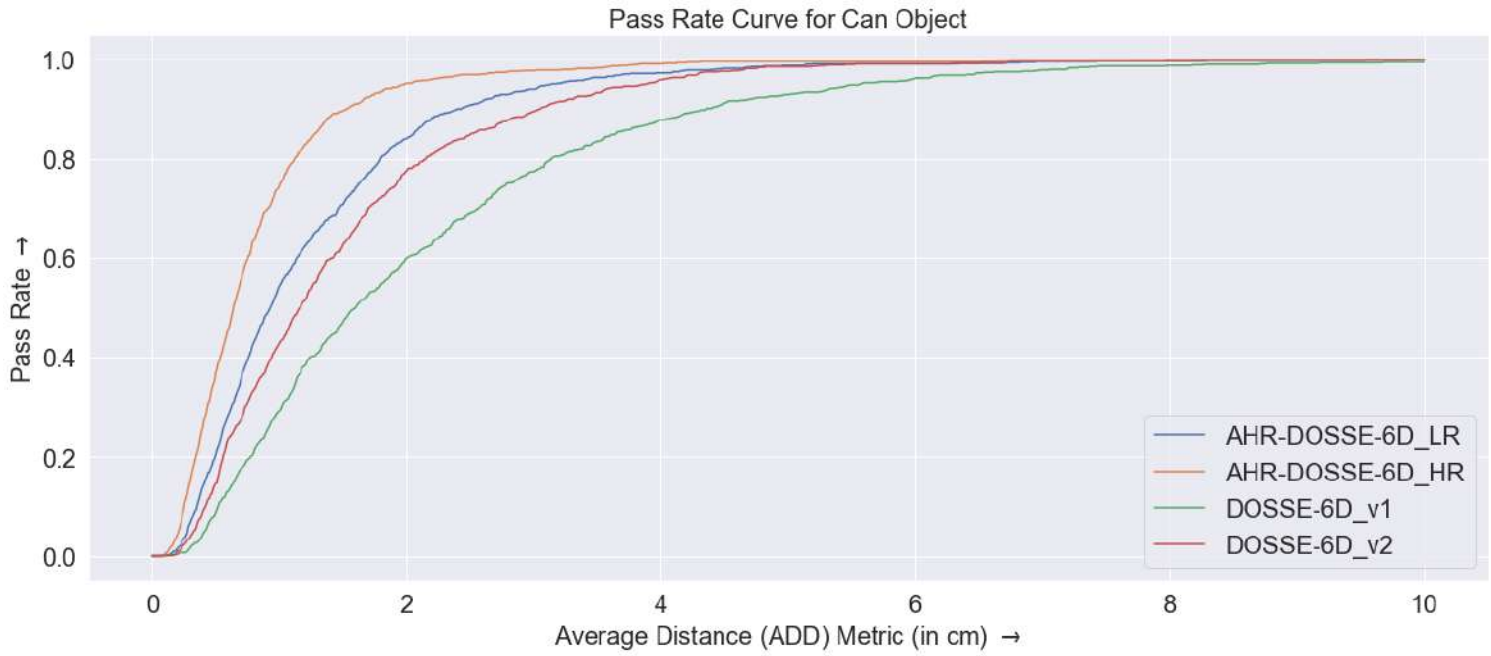
Benchvise



Lamp


Pass Rate Curve for Lamp Object
Pass Rate →
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
Average Distance (ADD) Metric (in cm) →
AHR-DOSSE-6D_LR
AHR-DOSSE-6D_HR
DOSSE-6D_v1
DOSSE-6D_v2

Can


Pass Rate Curve for Can Object
Pass Rate →
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
Average Distance (ADD) Metric (in cm) →
AHR-DOSSE-6D_LR
AHR-DOSSE-6D_HR
DOSSE-6D_v1
DOSSE-6D_v2

Iron

7.4 Translational RMSE Metric : Results

7.4.1 Quantitative Results

The following table provides the results for the average (over the test set) translational RMSE metric (in cm) for various models and objects :

S.No.	Object / Approach	Cat *(d = 15.50 cm)*	Benchvise *(d = 28.69 cm)*	Lamp *(d = 28.52 cm)*	Can *(d = 20.20 cm)*	Iron *(d = 30.32 cm)*
1.	**DOSSE-6D_v1**	3.0916	1.3903	1.4937	1.6155	2.5879
2.	**DOSSE-6D_v2**	1.4772	0.9548	1.1841	3.7014	1.7146
3.	**AHR-DOSSE-6D_LR**	4.5607	3.0240	0.8560	0.9647	2.3141
4.	**AHR-DOSSE-6D_HR**	5.8455	12.5857	3.2542	0.7023	7.6040

Table 7.4: Table displaying the average translational RMSE metric values (in cm). The model architectures and other details are described in Chapter 4 and Chapter 5.

The following table provides the results for the translational RMSE metric, in terms of pass rates (in %), for various models and objects. For each Model :

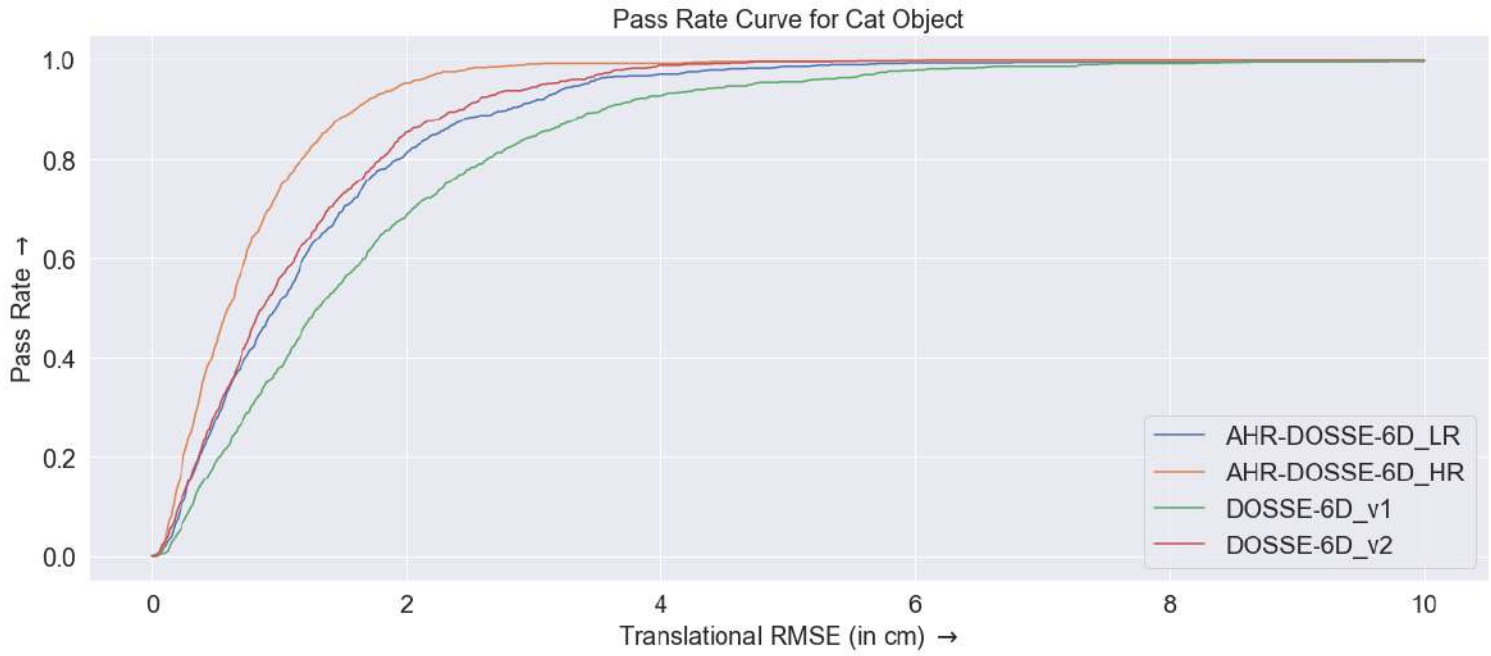
S.No.	Approach \ Object	Cat (d = 15.50 cm)	Benchvise (d = 28.69 cm)	Lamp (d = 28.52 cm)	Can (d = 20.20 cm)	Iron (d = 30.32 cm)
1.	**DOSSE-6D_v1**	56.92	97.38	95.37	83.33	87.02
		94.72	99.54	100.00	99.54	99.40
		99.18	100.00	100.00	99.88	99.64
2.	**DOSSE-6D_v2**	73.71	98.63	98.76	93.17	96.03
		99.30	100.00	100.00	99.77	99.64
		100.00	100.00	100.00	99.88	99.88
3.	**AHR-DOSSE-6D_LR**	70.89	98.63	99.66	96.18	97.24
		98.12	99.89	100.00	99.77	99.52
		99.41	99.89	100.00	100.00	99.76
4.	**AHR-DOSSE-6D_HR**	89.32	98.40	99.66	98.26	98.44
		99.53	98.75	99.89	99.77	99.16
		99.65	98.75	99.89	100.00	99.16

Table 7.5: Table displaying the translational RMSE metric pass rates (in %). The model architectures and other details are described in Chapter 4 and Chapter 5.

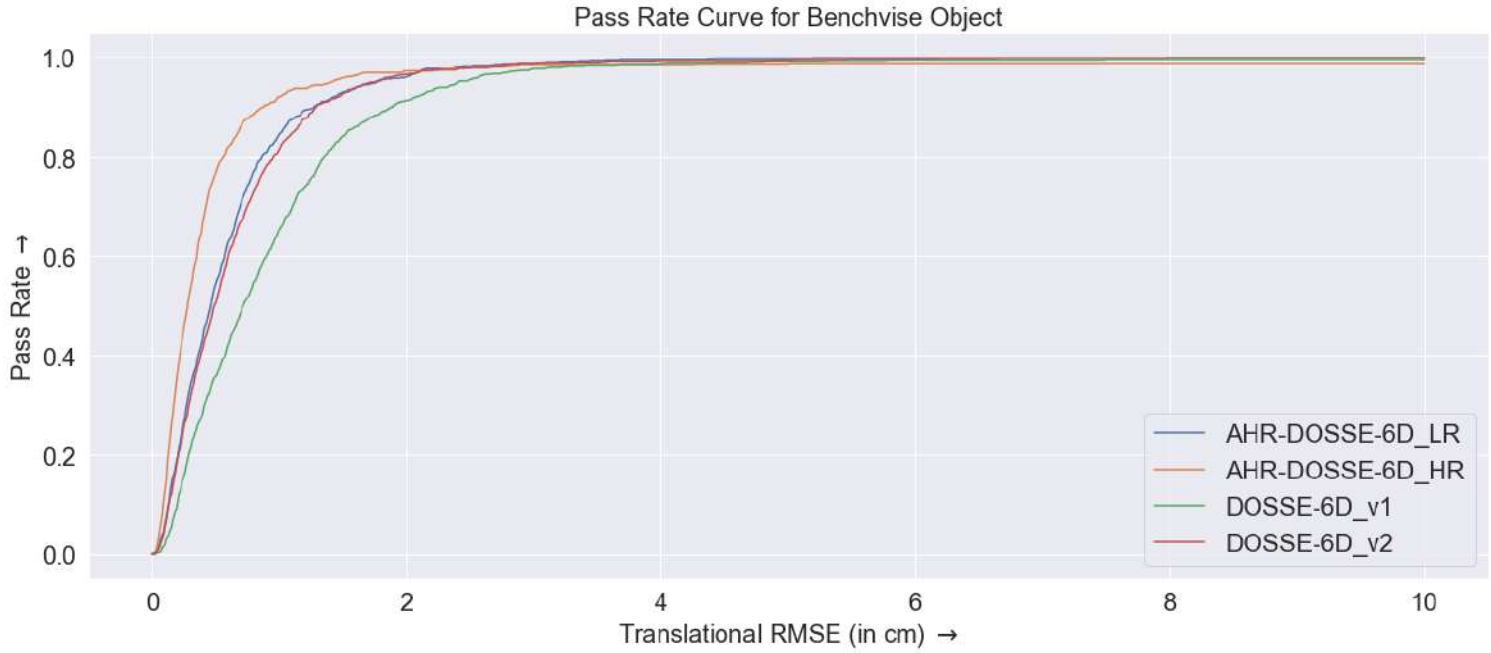
- 1^{st} sub-row corresponds to the 10% of the diameter of object threshold for pass rate. It is represented in red colour.
- 2^{nd} sub-row corresponds to the 30% of the diameter of object threshold for pass rate. It is represented in green colour.
- 3^{rd} sub-row corresponds to the 50% of the diameter of object threshold for pass rate. It is represented in blue colour.

7.4.2 Graphical Results

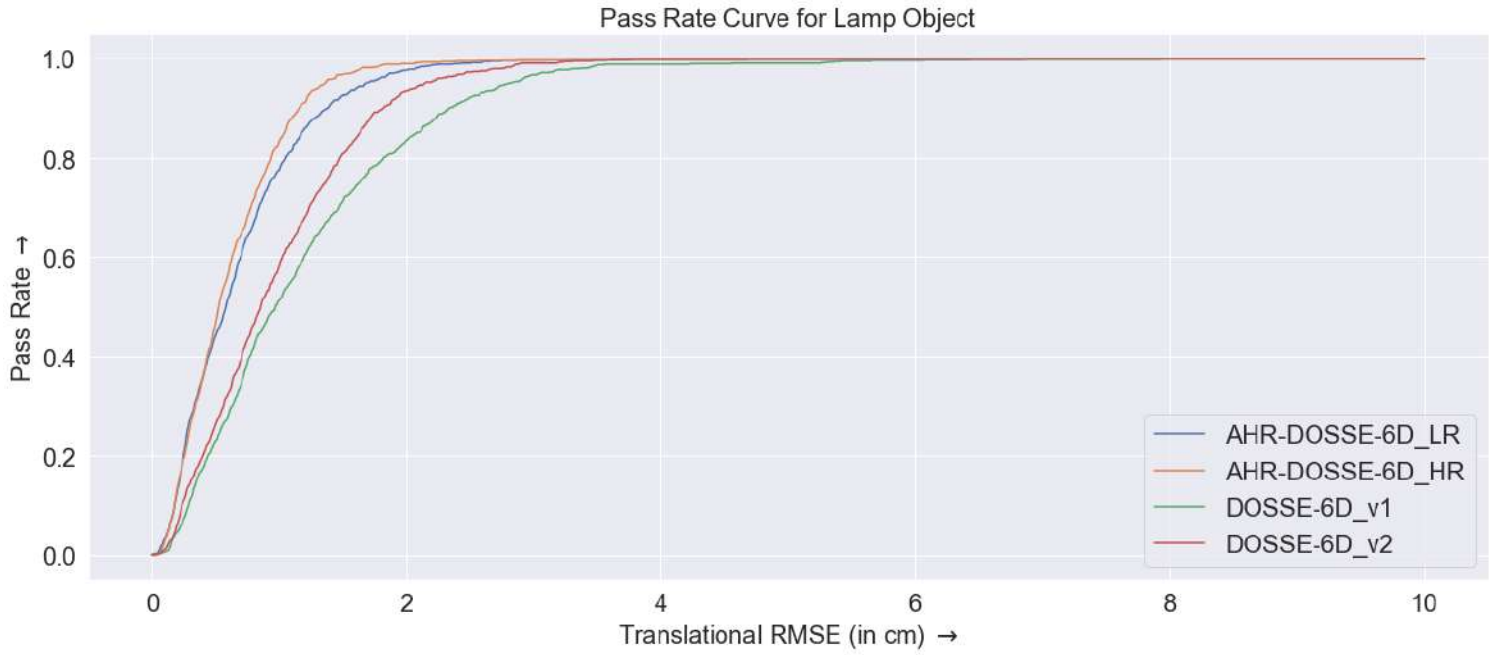
Cat



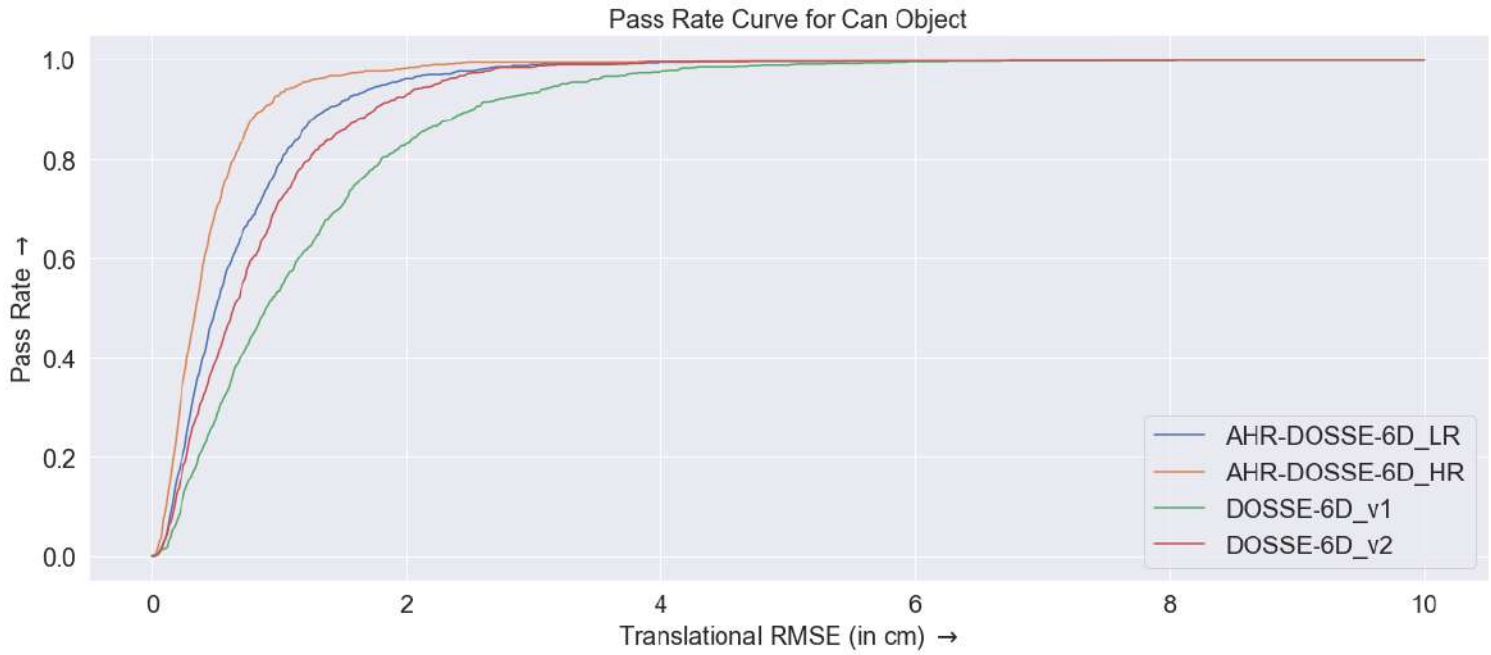
Benchvise



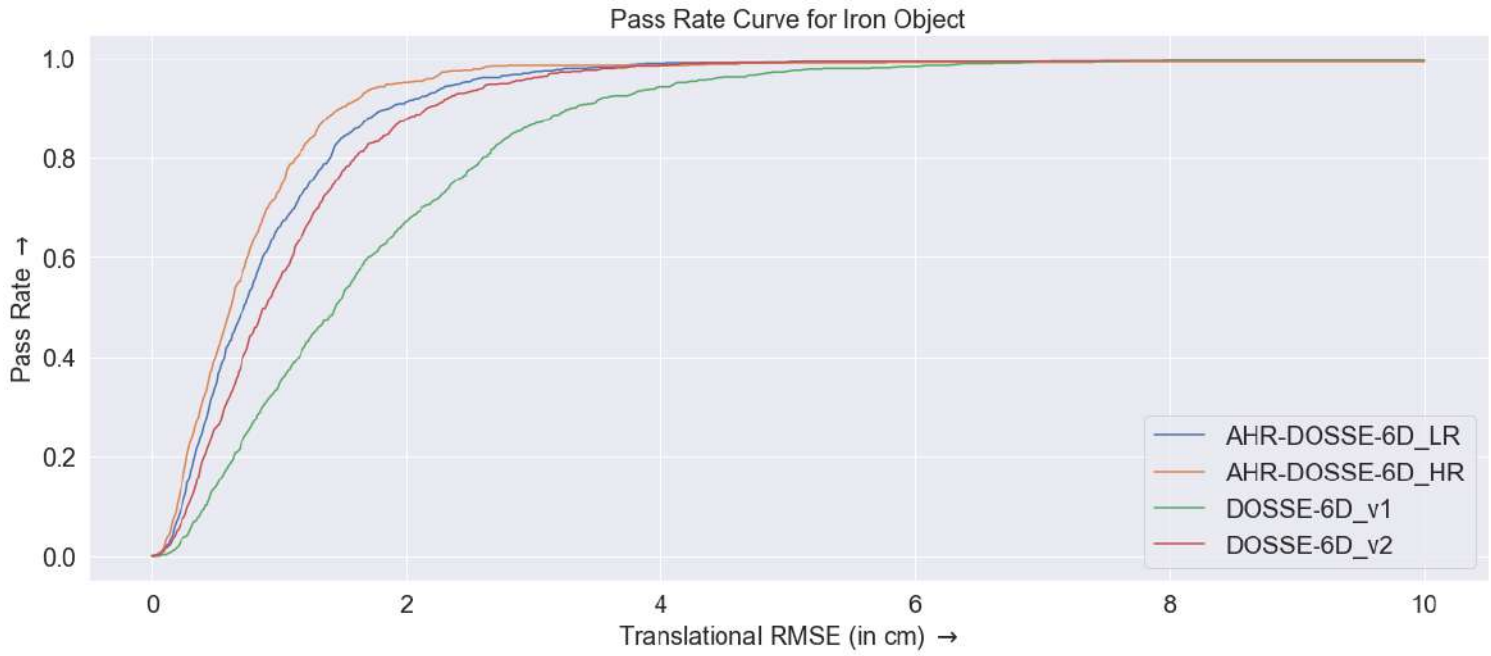
Lamp



Can


Pass Rate Curve for Can Object
Pass Rate →
Translational RMSE (in cm) →
AHR-DOSSE-6D_LR
AHR-DOSSE-6D_HR
DOSSE-6D_v1
DOSSE-6D_v2
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10

Iron


Pass Rate Curve for Iron Object
Pass Rate →
Translational RMSE (in cm) →
AHR-DOSSE-6D_LR
AHR-DOSSE-6D_HR
DOSSE-6D_v1
DOSSE-6D_v2
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10

7.5 Reprojection Error Metric : Results

7.5.1 Quantitative Results

The following table provides the results for the average (over the test set) reprojection error metric (in px) for various models and objects :

S.No.	Object / Approach	Cat *(d = 15.50 cm)*	Benchvise *(d = 28.69 cm)*	Lamp *(d = 28.52 cm)*	Can *(d = 20.20 cm)*	Iron *(d = 30.32 cm)*
1.	**DOSSE-6D_v1**	3.9603	2.7420	3.6497	3.0645	4.6495
2.	**DOSSE-6D_v2**	2.0605	2.1459	2.7505	2.4283	3.5350
3.	**AHR-DOSSE-6D_LR**	12.9549	2.2051	2.5063	2.3291	12.0925
4.	**AHR-DOSSE-6D_HR**	1.7143	1.6847	2.3473	1.6618	27.7547

Table 7.6: Table displaying the average Reprojection Error metric values (in px). The model architectures and other details are described in Chapter 4 and Chapter 5.

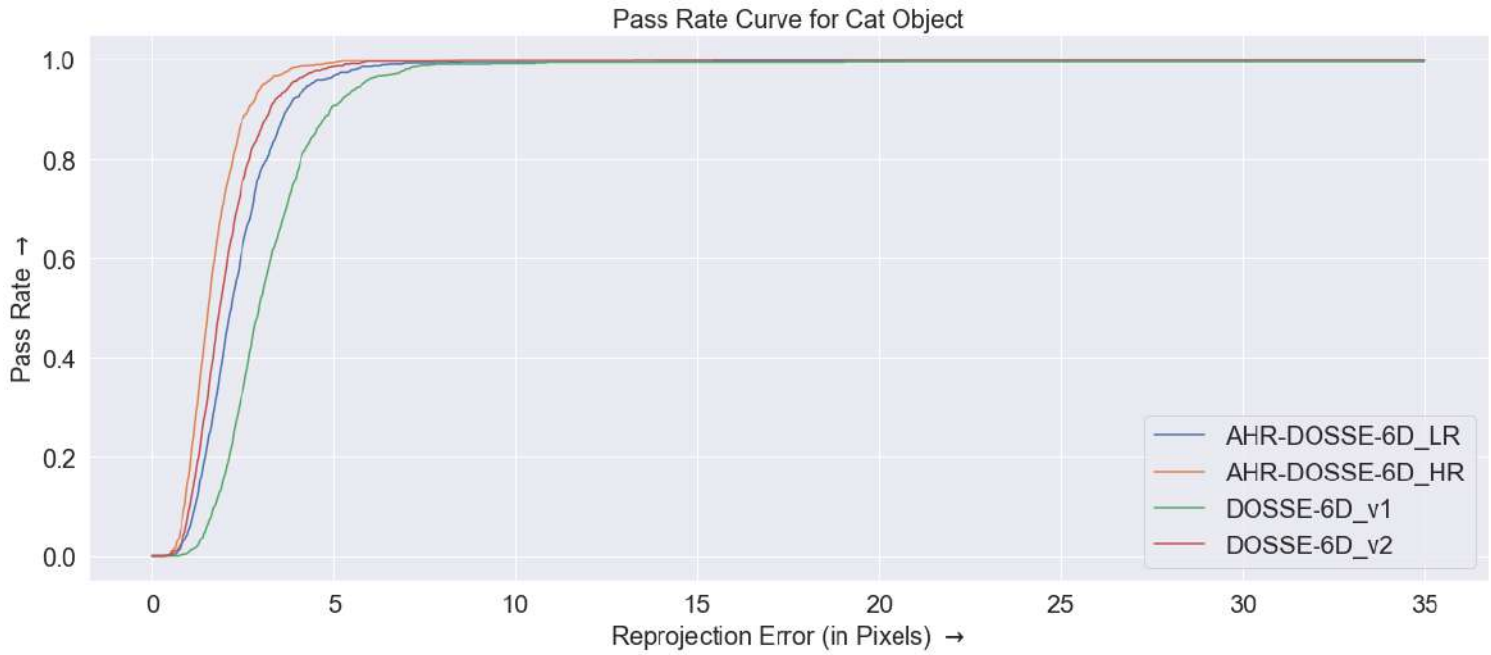
The following table provides the results for the reprojection error metric, in terms of pass rates (in %), for various models and objects. The threshold considered is 5 px.

S.No.	Object / Approach	Cat *(d = 15.50 cm)*	Benchvise *(d = 28.69 cm)*	Lamp *(d = 28.52 cm)*	Can *(d = 20.20 cm)*	Iron *(d = 30.32 cm)*
1.	**DOSSE-6D_v1**	90.85	93.16	84.65	91.90	66.47
2.	**DOSSE-6D_v2**	98.59	95.21	91.31	95.37	83.65
3.	**AHR-DOSSE-6D_LR**	96.60	95.32	94.02	96.06	87.86
4.	**AHR-DOSSE-6D_HR**	99.30	97.04	94.58	98.84	91.71

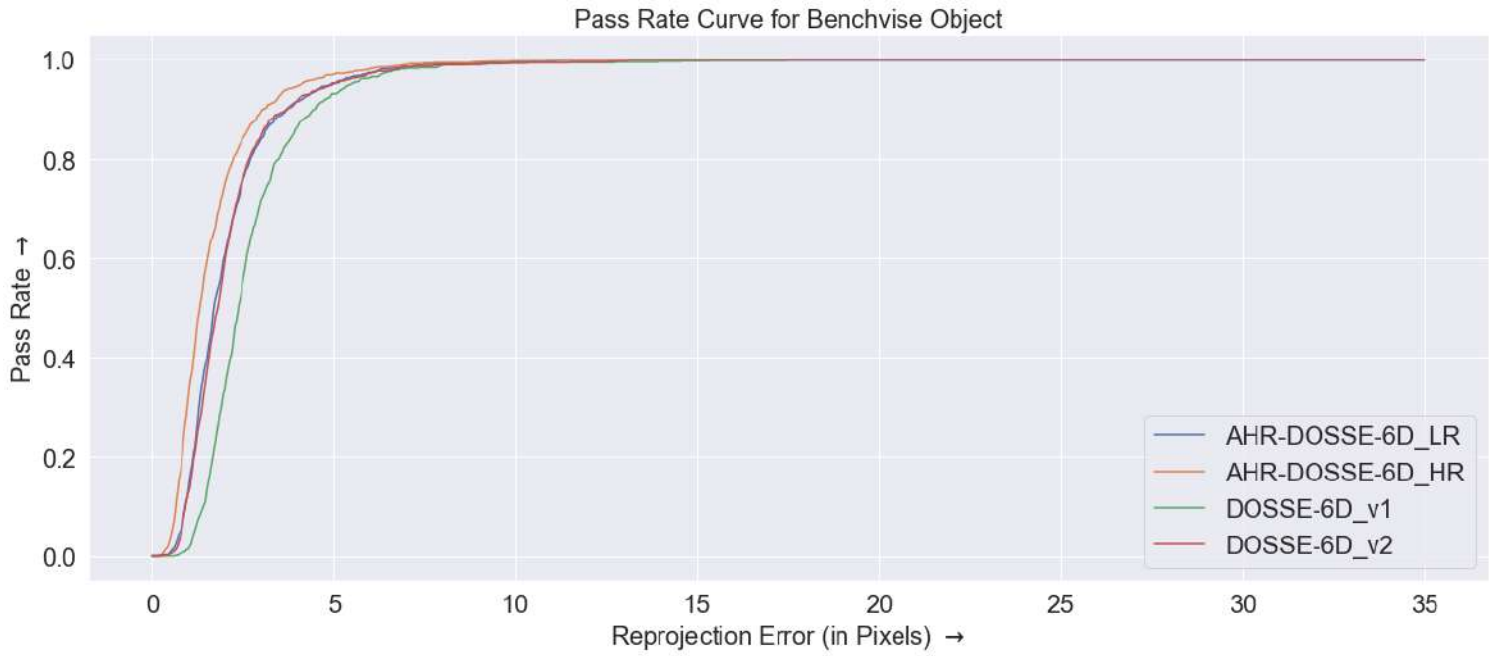
Table 7.7: Table displaying the Reprojection Error metric pass rates (in %) - 5 px threshold. The model architectures and other details are described in Chapter 4 and Chapter 5.

7.5.2 Graphical Results

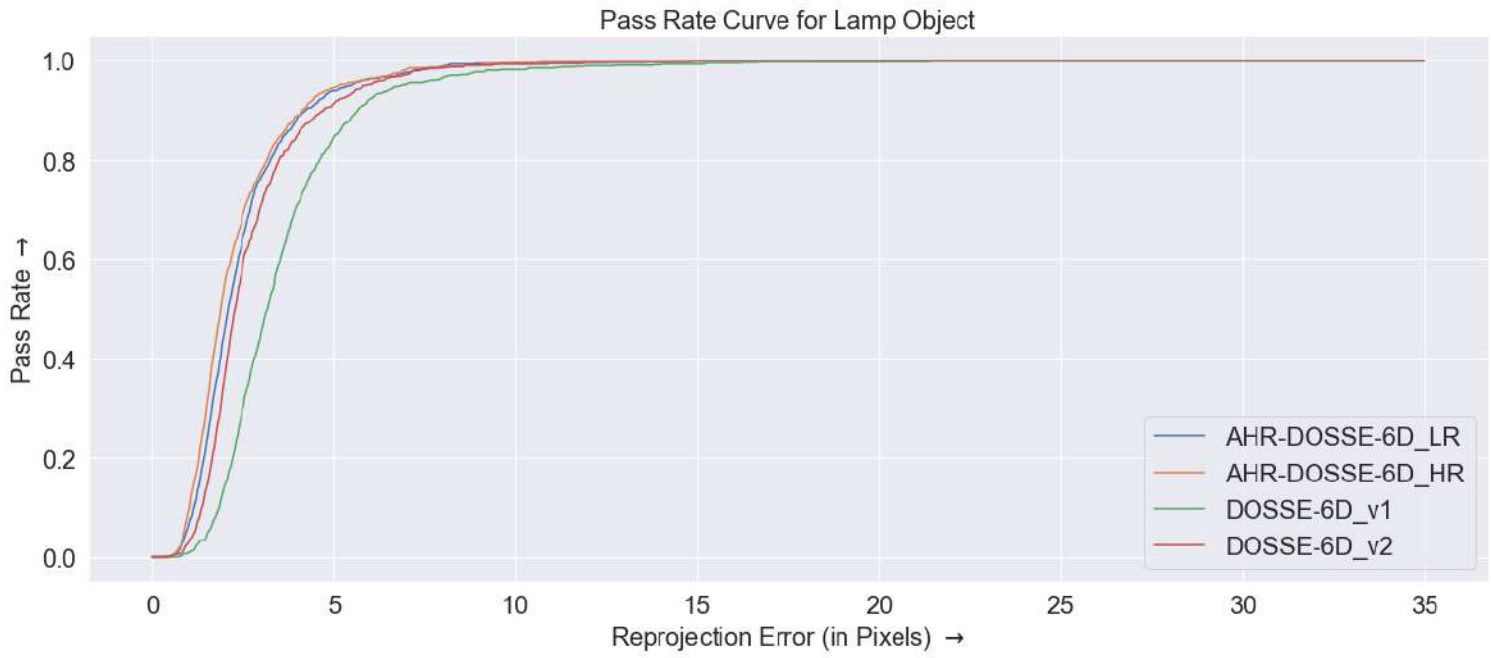
Cat



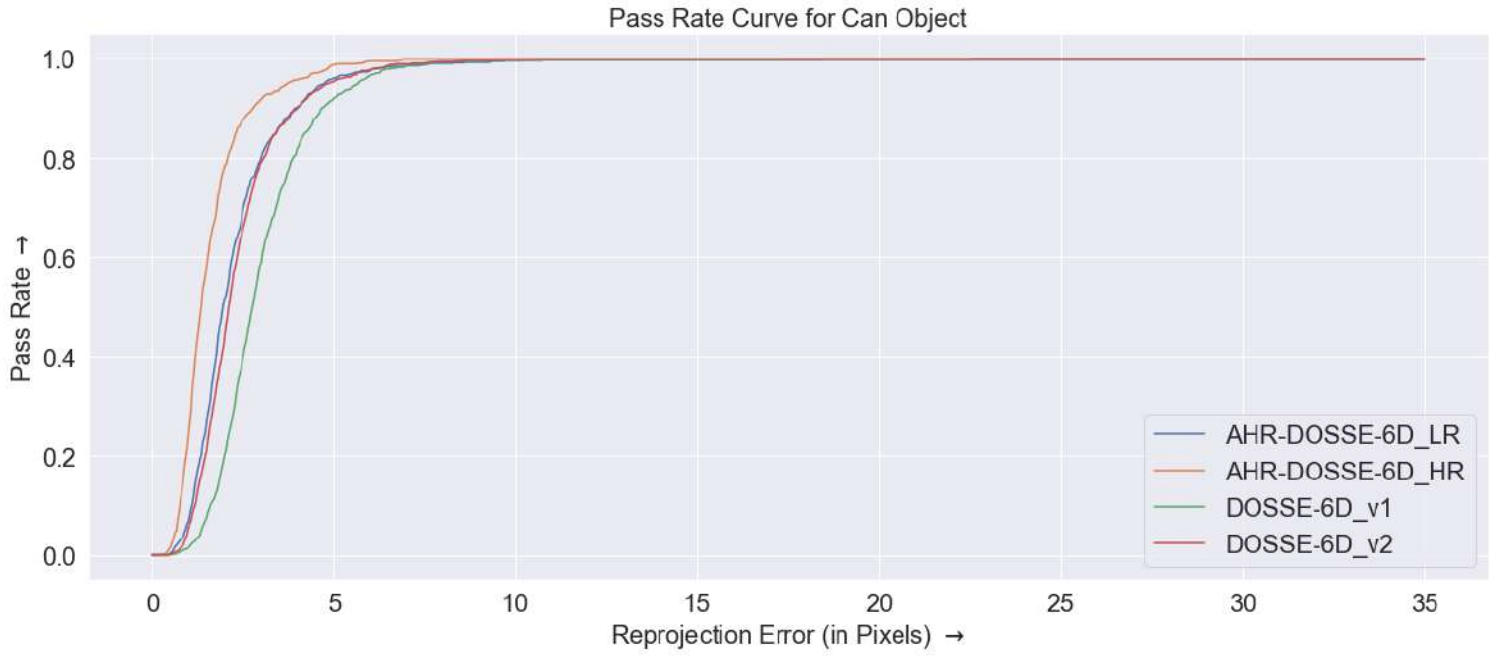
Benchvise



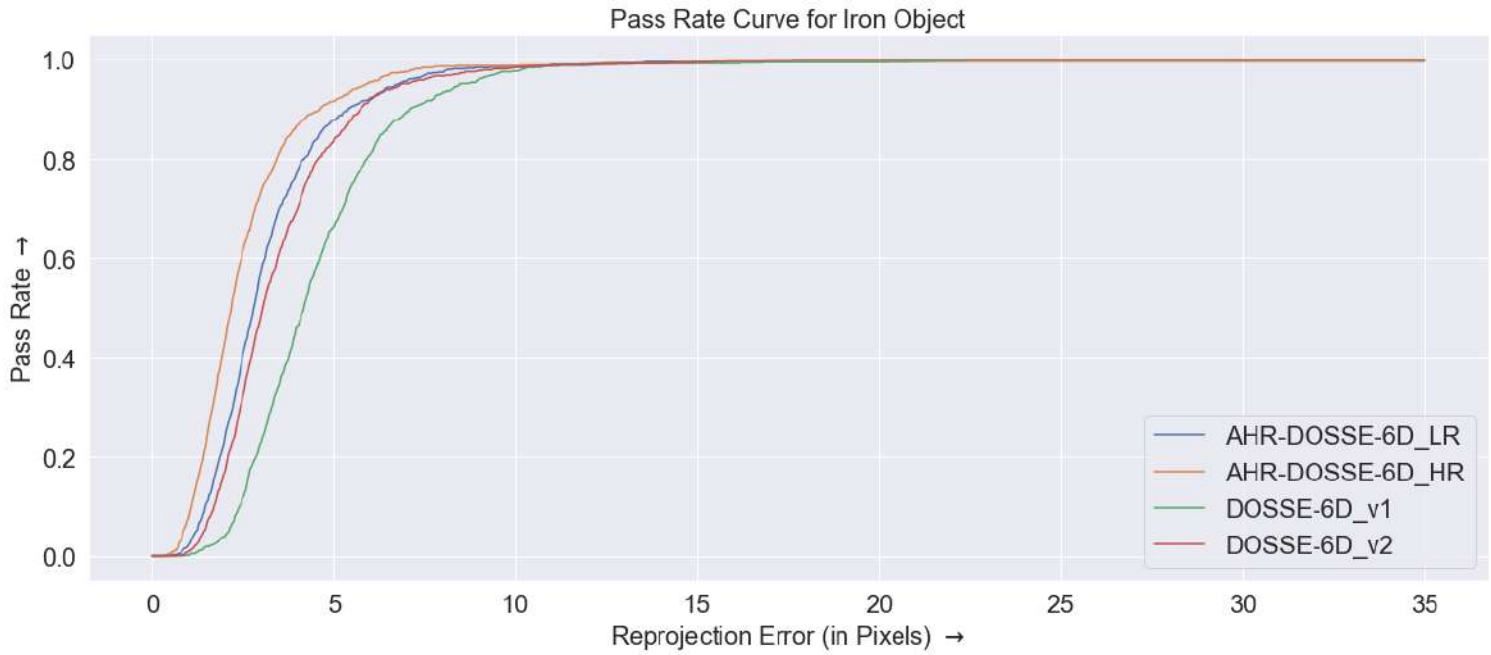
Lamp


Pass Rate Curve for Lamp Object
Pass Rate →
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
35
Reprojection Error (in Pixels) →
AHR-DOSSE-6D_LR
AHR-DOSSE-6D_HR
DOSSE-6D_v1
DOSSE-6D_v2

Can


Pass Rate Curve for Can Object
Pass Rate →
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
35
Reprojection Error (in Pixels) →
AHR-DOSSE-6D_LR
AHR-DOSSE-6D_HR
DOSSE-6D_v1
DOSSE-6D_v2

Iron



7.6 Quaternion Error Metric : Results

7.6.1 Quantitative Results

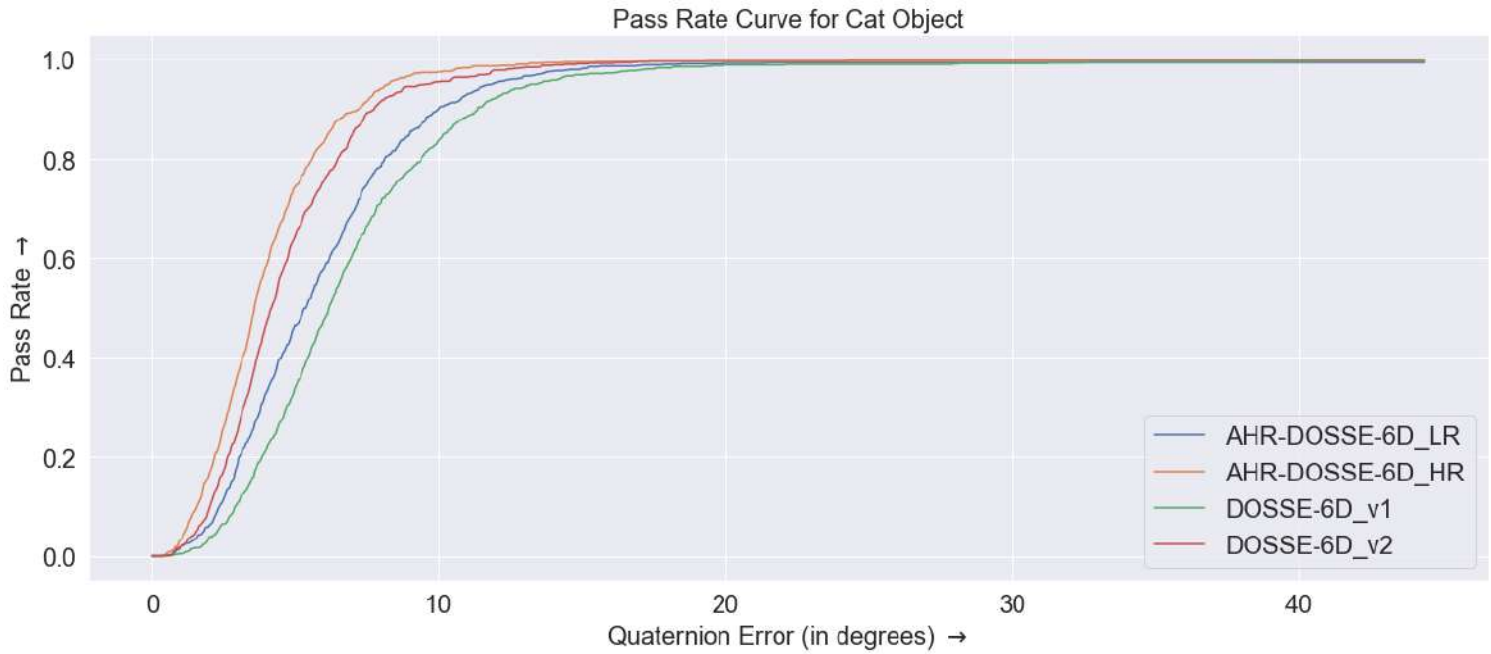
The following table provides the results for the average (over the test set) quaternion error (in rad) for various models and objects :

S.No.	Object / Approach	Cat *(d = 15.50 cm)*	Benchvise *(d = 28.69 cm)*	Lamp *(d = 28.52 cm)*	Can *(d = 20.20 cm)*	Iron *(d = 30.32 cm)*
1.	**DOSSE-6D_v1**	0.1217	0.0565	0.0641	0.0714	0.1161
2.	**DOSSE-6D_v2**	0.0762	0.0363	0.0462	0.0574	0.0884
3.	**AHR-DOSSE-6D_LR**	0.1077	0.0405	0.0416	0.0512	0.0877
4.	**AHR-DOSSE-6D_HR**	0.0681	0.0453	0.0383	0.0273	0.0751

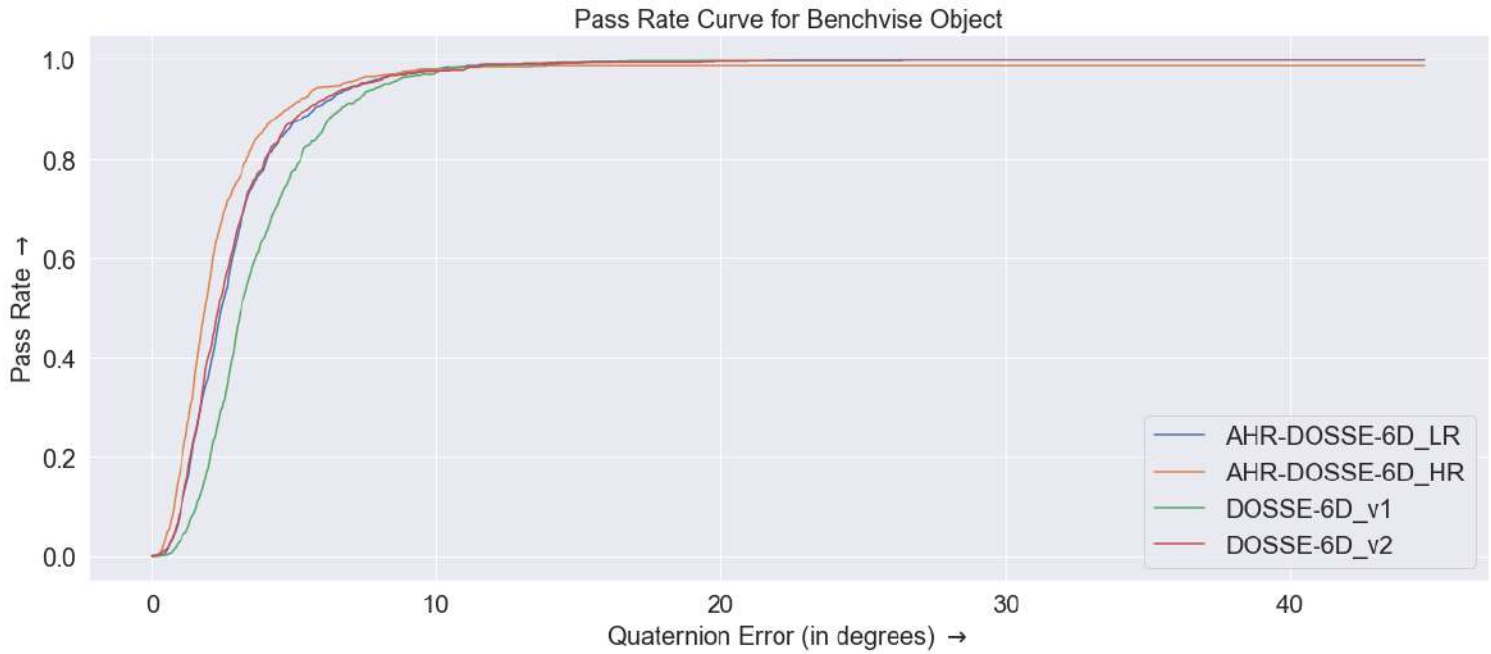
Table 7.8: Table displaying the average Quaternion Error metric values (in rad). The model architectures and other details are described in Chapter 4 and Chapter 5.

7.6.2 Graphical Results

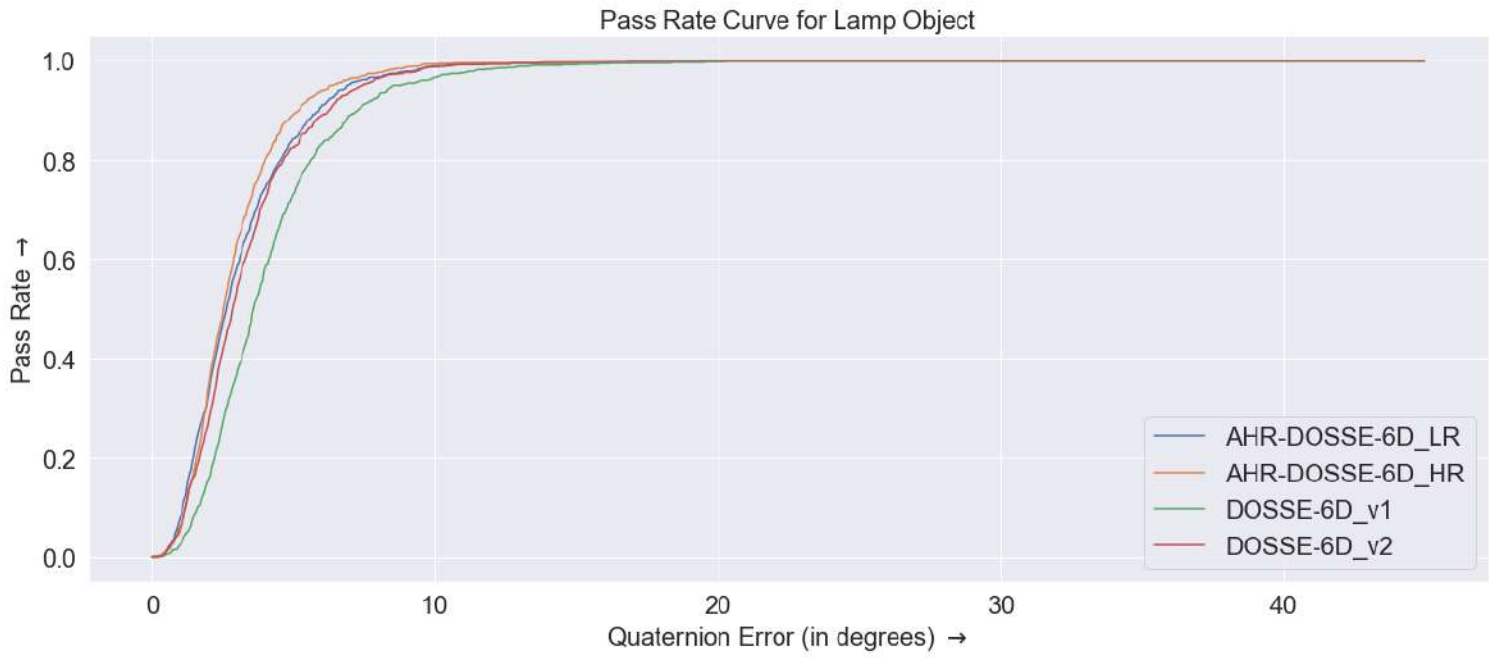
Cat



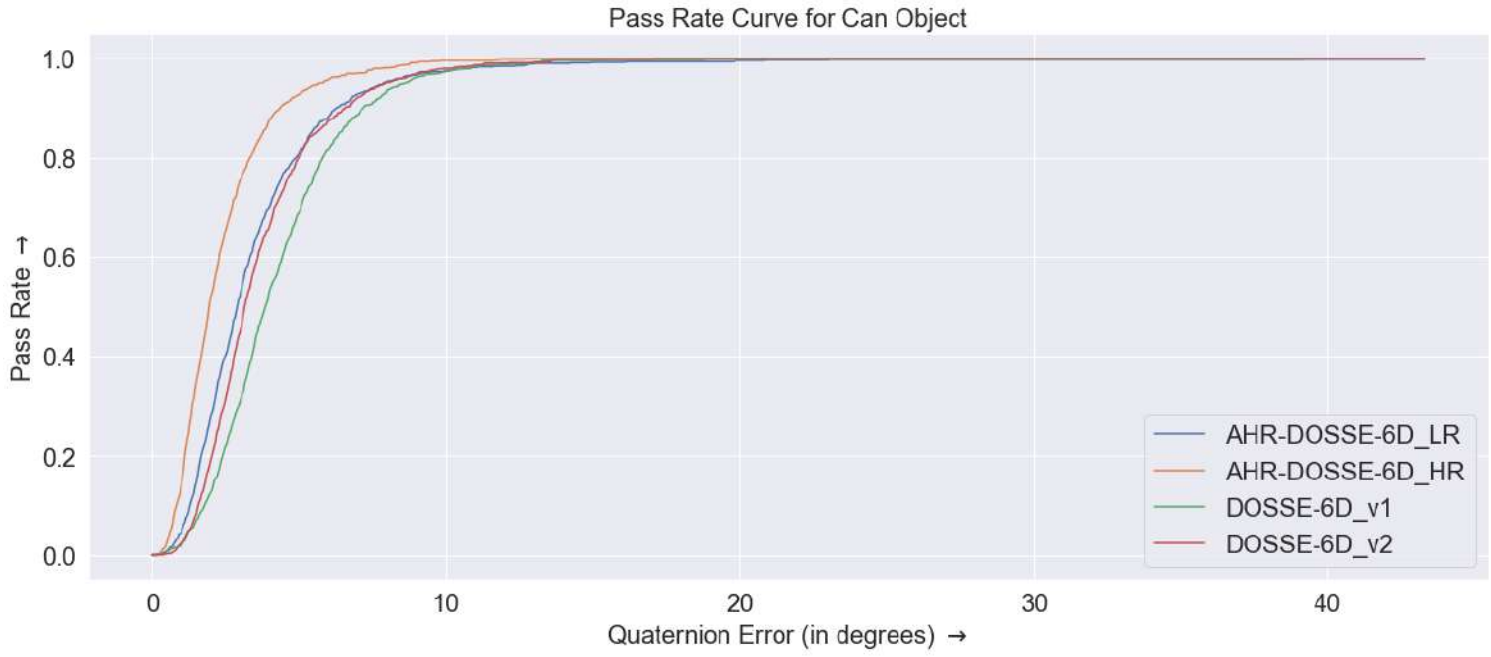
Benchvise



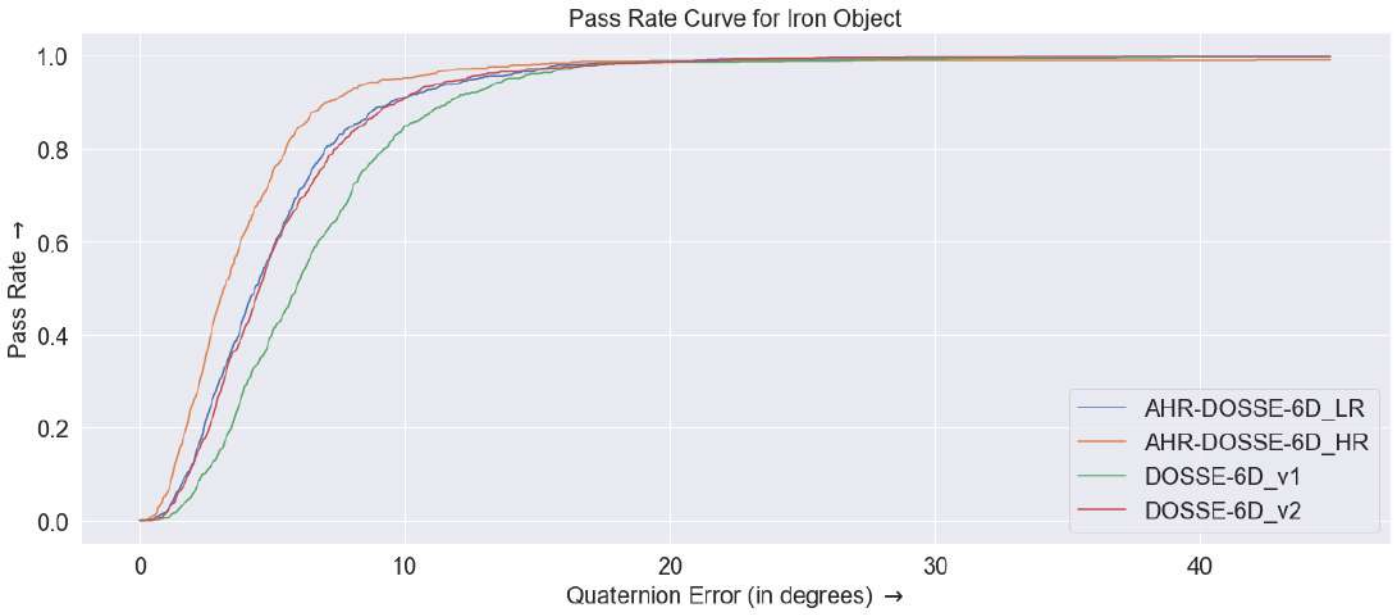
Lamp


Pass Rate Curve for Lamp Object
Pass Rate →
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
Quaternion Error (in degrees) →
AHR-DOSSE-6D_LR
AHR-DOSSE-6D_HR
DOSSE-6D_v1
DOSSE-6D_v2

Can


Pass Rate Curve for Can Object
Pass Rate →
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
Quaternion Error (in degrees) →
AHR-DOSSE-6D_LR
AHR-DOSSE-6D_HR
DOSSE-6D_v1
DOSSE-6D_v2

Iron



7.7 Performance Comparison

The following table provides the results for the ADD metric, in terms of pass rates (in %), for various models and objects. For each Model, 10% of the diameter of object threshold for pass rate.

S.No.	Object / Approach	Cat *(d = 15.50 cm)*	Benchvise *(d = 28.69 cm)*	Lamp *(d = 28.52 cm)*	Can *(d = 20.20 cm)*	Iron *(d = 30.32 cm)*
1.	**SSD-6D** [11]	0.51	0.18	8.20	1.35	8.86
2.	**Tekin et al.** [19]	41.82	81.80	71.11	68.80	74.97
3.	**DOSSE-6D_v1**	33.45	86.77	74.94	60.19	60.22
4.	**DOSSE-6D_v2**	50.23	94.53	85.55	78.01	82.45
5.	**AHR-DOSSE-6D_LR**	45.89	94.30	94.36	84.14	88.10
6.	**AHR-DOSSE-6D_HR**	68.31	96.69	97.86	95.02	93.63

Table 7.10: Table displaying the ADD metric pass rates (in %).

The following table provides the results for the reprojection error metric, in terms of pass rates (in %), for various

models and objects. The threshold considered is 5 px.

S.No.	Object / Approach	Cat *(d = 15.50 cm)*	Benchvise *(d = 28.69 cm)*	Lamp *(d = 28.52 cm)*	Can *(d = 20.20 cm)*	Iron *(d = 30.32 cm)*
1.	**BB8** [15]	97.00	80.00	74.40	84.10	78.90
2.	**Tekin et al.** [19]	97.41	95.06	76.87	97.44	82.94
3.	**DOSSE-6D_v1**	90.85	93.16	84.65	91.90	66.47
4.	**DOSSE-6D_v2**	98.59	95.21	91.31	95.37	83.65
5.	**AHR-DOSSE-6D_LR**	96.60	95.32	94.02	96.06	87.86
6.	**AHR-DOSSE-6D_HR**	99.30	97.04	94.58	98.84	91.71

Table 7.12: Table displaying the Reprojection Error metric pass rates (in %) - 5 px threshold.

The rows highlighted in grey colour are some of the popular methods that provide results for similar *single stage, single object, single instance*[‡] object pose estimation using RGB images only. These are presented here for comparison of the performance of the models proposed in this report to some of the existing methods.

7.8 Inferences

In this section, the performance of different models based on the quantitative and graphical results presented in this Chapter is analyzed.

1. **Comparison to Existing Methods :**

   As we can from Table 7.10 and Table 7.6, the model *AHR-DOSSE-6D_HR* seems to give a relatively superior performance on all objects. In fact, for most of the objects tested, the model achieves ADD metric pass rate and Reprojection error pass rate greater than 90% for commonly used thresholds of $< 10\%$ object's diameter and 5 px respectively, using only RGB images. The improvement in performance can be attributed to the following factors (see Chapter 4, Section 4.5) :

   - Maintenance of high resolution hidden representations in the AHRNet backbone

[‡]Note that, for some of the methods only pre-refinement stage results are provided for fair comparison.

- Incorporation of attention module
- Increased input image resolution
- Model's ability to generate spatially precise heatmaps as opposed to modelled confidence approaches as in *Pose6DSSD* and *DOSSE-6D*.
- Use of more geometrical details by considering keypoints that lie on the object's surface instead of the 3D bounding box points, which is the case in *DOSSE-6D*.

In general, the performance of the model is better than most of the other existing object pose estimation methods and is comparable to the state-of-the-art techniques' performance.

2. **Model Performance based on Object's Size (Diameter) :**

   → It is observed that predicted pose of objects having smaller diameter such as *Cat* is more difficult for the models.

   → The improved performance of *AHR-DOSSE-6D_HR* is due to the usage of AHRNet feature extraction backbone which is capable of maintaining high resolution, spatial information-rich representations, due to the use of multipe multi-scale fusions. In other words, the model is able to utilize whatever image is given as input, in an effective manner.

3. **Input Image Resolution :**

   → It is clear by observing the performance of the model pairs *DOSSE-6D_v1* & *DOSSE-6D_v2* and *AHR-DOSSE-6D_LR* & *AHR-DOSSE-6D_HR*, the models which take high input resolution image as input generally tend to perform much better than their counterparts.

   → The Reprojection Error performance also improves a lot when high resolution input images are used for estimating object pose.

4. **Parameter Efficiency :**

   For increasing the model's input image resolution, for *DOSSE-6D* (from $v1 \rightarrow v2$), more convolutional layers were added making the model *deeper* (i.e increasing total number of parameters). *AHR-DOSSE-6D*, on the other hand, is parameter-efficient in the sense that we can increase the input image resolution (because it is shown to improve performance) without an increase in the trainable weights[†]. The only difference arises in the output heatmap (set) dimensions, which is not an issue, as DSNT block (see Chapter 4, Section 4.5.4) takes in these heatmaps and regresses "normalizes" 2D image coordinates.

[†]One of the experimental configurations in Chapter 7 clearly demonstrates this idea.

Chapter 8

Conclusion & Future Work

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

8.1 Conclusion

This report on *Pose Estimation for Autonomous Robotic Grasping* is aimed at developing robust, efficient and effective object 6D pose estimation techniques. A complete end to end pose estimation pipeline where a UR3 robotic arm is deployed in a simulated pick and place task is demonstrated in this work. The basic outline of this pipeline is discussed in Chapter 3. A brief literature review concerning object pose estimation techniques is provided in Chapter 2. The majority of this work has been focussed at developing improved CNN based models for estimating pose from a single RGB image, utilizing neither depth information nor post hoc refinement stages. A series of such pose estimation models are designed, compared and analyzed in Chapter 4. In order to test the efficacy of the approaches,they are trained and tested on synthetic data from simple and cluttered scenes, in a same-environment and cross-environment setting (see Chapter 5 & Chapter 6). The developed models were also tested on various objects from the LINEMOD benchmark dataset and the results, exhibiting the high performance of the models, were illustrated and analyzed in Chapter 7.

8.2 Future Work Ideas

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

- Creation of more simulation scenarios in Unity Editor and studying the *Same Environment* and *Cross Environment* performance more extensively.
- Collection of data for other real life objects and testing the model's pose estimation performance.
- Developing an improved pose estimation approach for *Multi-object, Multi-instance* prediction. One way of doing this is the inclusion of Part Affinity Fields (PAFs) prediction component into the *AHR-DOSSE-6D* model (see Chapter 4, Section 4.5).
- Working on improving the next step of the pose-estimation in the *Pick-and-Place* pipeline, called Grasp Estimation, for understanding the optimal way of grabbing an object to minimize grasp failures.
- Testing the performance of the pose estimation models on real data collected from a digital camera. One can look into Domain adaption techniques to bridge the gap between simulation and reality.
- Incorporation of additional collision avoidance modules to develop a more robust "safety-critical" approach to minimize the collisions, while executing the planned robotic arm trajectory.

Bibliography

[1] Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., and Rother, C. (2014). Learning 6d object pose estimation using 3d object coordinates. In *European conference on computer vision*, pages 536–551. Springer.

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

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

[4] Chen, B., Parra, A., Cao, J., Li, N., and Chin, T.-J. (2020). End-to-end learnable geometric vision by backpropagating pnp optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8100–8109.

[5] Corke, P. P. (2019). Qut robot academy. https://robotacademy.net.au/.

[6] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[7] Du, G., Wang, K., Lian, S., and Zhao, K. (2021). Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review. *Artificial Intelligence Review*, 54(3):1677–1734.

[8] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[9] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[10] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. (2013). Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In Lee, K. M., Matsushita,

Y., Rehg, J. M., and Hu, Z., editors, *Computer Vision – ACCV 2012*, pages 548–562, Berlin, Heidelberg. Springer Berlin Heidelberg.

[11] Kehl, W., Manhardt, F., Tombari, F., Ilic, S., and Navab, N. (2017). Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *Proceedings of the IEEE international conference on computer vision*, pages 1521–1529.

[12] Nibali, A., He, Z., Morgan, S., and Prendergast, L. (2018). Numerical coordinate regression with convolutional neural networks. *arXiv preprint arXiv:1801.07372*.

[13] Odemakinde, E. (2021). Human pose estimation with deep learning - ultimate overview in 2021. https://viso.ai/deep-learning/pose-estimation-ultimate-overview/.

[14] Peng, S., Zhou, X., Liu, Y., Lin, H., Huang, Q., and Bao, H. (2020). Pvnet: pixel-wise voting network for 6dof object pose estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[15] Rad, M. and Lepetit, V. (2017). Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3828–3836.

[16] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

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

[18] Tejani, A., Tang, D., Kouskouridas, R., and Kim, T.-K. (2014). Latent-class hough forests for 3d object detection and pose estimation. In *European Conference on Computer Vision*, pages 462–477. Springer.

[19] Tekin, B., Sinha, S. N., and Fua, P. (2018). Real-time seamless single shot 6d object pose prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 292–301.

[20] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE.

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

[22] Wang, Q., Wu, B., Zhu, P., Li, P., Zuo, W., and Hu, Q. (2020). Eca-net: Efficient channel attention for deep convolutional neural networks. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11531–11539.

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

[24] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2017). Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*.

[25] Yu, X., Zhuang, Z., Koniusz, P., and Li, H. (2020). 6dof object pose estimation via differentiable proxy voting loss. *arXiv preprint arXiv:2002.03923*.