Enhancing 6D Object Pose Estimation

Elias Noorzad
Student ID: s314515

Ayda Ghasemazar
Student ID: s338736

Filip Nykvist
Student ID: s347839

Hasti Azadnia
StudentID: s338984

***Abstract*—This project details the development of an end-to-end pipeline for 6D object pose estimation from RGB-D images. Addressing this critical task is vital for applications in robotics and augmented reality. Our pipeline uses distinct modules for 2D object detection and 6D pose prediction. For object detection, we used a pre-trained YOLOv8 model, fine-tuning it on a custom-formatted LineMOD dataset to accurately identify and localize objects, evaluating its performance using mean Average Precision (mAP).**

Furthermore, the detected object crops for the Yolo-model serve as input for a downstream simplified PoseNet, PoseCNN and DenseFusion-inspired pose estimator. We first constructed a baseline using an RGB-only model and then extended it to incorporate depth information (RGB-D. This updated model, while simplified compared to the full DenseFusion pipeline, proved to perform slightly better.

The models were evaluated using the Average Distance of Model Points (ADD) metric. Our results demonstrates the capabilities of YOLOv8 for object detection together with the advantages of incorporating depth data for enhanced 6D pose predictions.

***Index Terms*—6D Pose Estimation, Object Detection, Deep Learning, RGB-D, PoseNet, DenseFusion, YOLO, LineMOD.**

I. Introduction

Six-Degrees-of-Freedom (6D) pose estimation refers to computing an objects 3D position and orientation, commonly represented by a translation vector $t = [x, y, z]$ and a rotation matrix R or quaternion q. This task is fundamental in computer vision and robotics, supporting applications such as manipulation, augmented reality, and autonomous navigation.

Recent advances in deep learning have significantly improved 6D pose estimation, especially when leveraging RGB-D data. While RGB-only models extract appearance features, depth data contributes geometric structure, helping disambiguate object position and orientation in cases of occlusion or weak texture [1].

Central to pose estimation pipelines is the preceding task of object detection. Object detection locates and classifies objects in images, generating bounding boxes and class labels crucial for cropping object regions for the downstream estimation of the 6D pose [2].

This project introduces a pipeline for 6D object pose estimation from RGB-D images, combining a YOLOv8-based object detector with a pose regression network. Inspired by PoseNet [3] and PoseCNN [4], the baseline is extended using depth features via a simplified DenseFusion approach [1]. Evaluation is based on the ADD metric and the LineMOD dataset.

The report covers related work (Section II), methodology and data (Section III), experimental results (Section IV), and conclusions with future directions (Section V).

Code and resources are available at:
https://github.com/EliasNoorzad/Pose6d_project.

II. Related Work

Early 6D pose estimation relied on matching hand-crafted features (ex. SIFT) between 2D/depth data and 3D models. While effective in controlled settings, these methods faltered with occlusion, clutter, and texture-poor objects.

The advent of deep learning marked a major shift in the field. For example, PoseNet [3], originally proposed for camera relocalization, demonstrated that convolutional networks could directly regress 6-DOF poses from RGB images. Although not designed for object-level estimation, its architecture inspired other models to go further in that direction.

PoseCNN [4] introduced a two-stage RGB-D approach, integrating semantic segmentation with ICP-enhanced 6D pose regression. While accurate, its computational intensity stood in the way for real-time performance.

DenseFusion [1] countered these limitations with an efficient, modular framework. By independently extracting and pixel-wise fusing RGB and depth features, it enabled end-to-end 6D pose regression with optional refinement, demonstrating robust performance in cluttered or occluded scenes.

Alongside pose estimation, accurate object detection remains a key prerequisite. To address the critical need for efficient and accurate object detection, our project employs YOLOv8, the latest model in the widely used YOLO (You Only Look Once) family. YOLOv8 follows a single-stage detection strategy, dividing an input image into grids and directly predicting bounding boxes and associated class probabilities within a single forward pass. This streamlined design allows YOLOv8 to achieve real-time detection performance, significantly outperforming traditional region-based methods in terms of speed and computational efficiency [5].

In particular, YOLOv8 offers a notable advantage over R-CNN-based methods, which rely on multi-stage detection processes involving separate region proposal and classification steps. The inherent simplicity and speed of

YOLOv8s unified detection approach make it ideally suited for the real-time and dynamic environments required by our 6D pose estimation pipeline [6].

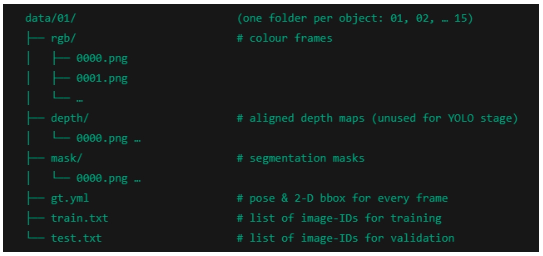
III. Methodology

The 6D object pose estimation pipeline is structured into two main sequential modules: a 2D object detection module, followed by a 6D pose estimation module. The 2D object detection module processes the RGB image to identify and localize objects, providing tight bounding boxes and class labels. These detection outputs, specifically the cropped RGB and (optionally) depth images, along with the object identifiers, are then fed into the 6D pose estimation module. This second module predicts the 3D translation and rotation for each detected object. The predicted 6D pose is subsequently used for evaluation against ground truth.

A. Object Detection Module

For the 2D object detection task, we selected **YOLOv8** due to its renowned balance of high accuracy and real-time inference capabilities, making it suitable for subsequent pose estimation.

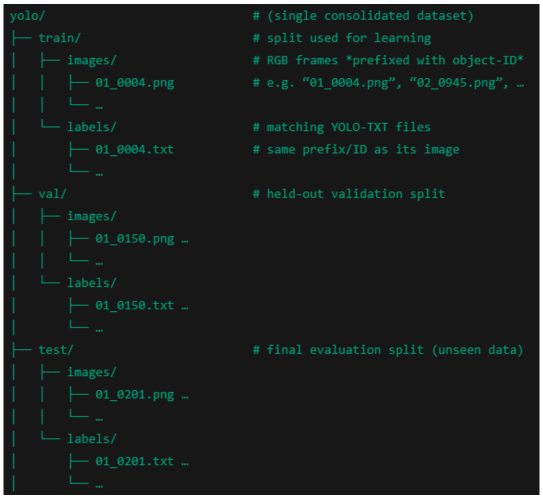
1) Dataset Preparation for YOLO: The original dataset was structured per object, containing individual folders named numerically from 01 to 15 (except for 03 and 07). Within each object-specific folder, three modalities were provided: RGB color images (rgb/), depth maps (depth/), and segmentation masks (mask/). Each image file was sequentially numbered, such as *0000.png*, *0001.png*, etc. Additionally, a file named *gt.yml* provided pose information and 2D bounding boxes for each frame. The dataset also included two separate text files, *train.txt* and *test.txt*, listing the specific image IDs designated for training and validation respectively.



```
data/01/                    (one folder per object: 01, 02, … 15)
├── rgb/                    # colour frames
│   ├── 0000.png
│   ├── 0001.png
│   └── …
├── depth/                  # aligned depth maps (unused for YOLO stage)
│   └── 0000.png …
├── mask/                   # segmentation masks
│   └── 0000.png …
├── gt.yml                  # pose & 2-D bbox for every frame
├── train.txt               # list of image-IDs for training
└── test.txt                # list of image-IDs for validation
```

Fig. 1: Original LineMOD dataset structure.

To convert the bounding boxes into YOLO compatible format, the original top-left coordinates (x, y) were first transformed into center coordinates (x_c, y_c) by adding half the bounding box width and height $(w/2, h/2)$, respectively. These center coordinates and dimensions were then normalized by dividing by the width and height of the image, resulting in values ranging from 0 to 1 as required by YOLO. Additionally, a class label was added at the beginning of each annotation entry to indicate the object type, resulting in the final format: `<class_id> <x_center> <y_center> <width> <height>`. After conversion, the dataset is organized specifically for YOLO training into three distinct subsets: training, validation, and testing, each represented by the top-level folders `train/`, `val/`, and `test/`. Each subset contains an `images/` directory holding RGB frames, and a parallel `labels/` directory with corresponding `.txt` files specifying bounding boxes in YOLO-compatible format. The file naming convention includes an object-ID prefix (ex. `01_0004.png` and `01_0004.txt`), ensuring that images of multiple objects coexist without naming conflicts. The `train/` split is used directly for model training, the `val/` split serves to tune hyperparameters and prevent overfitting, while the `test/` split is reserved strictly for evaluating the final model performance on unseen data. Figure 2 illustrates this conversion.



```
yolo/                        # (single consolidated dataset)
├── train/                   # split used for learning
│   ├── images/              # RGB frames *prefixed with object-ID*
│   │   ├── 01_0004.png      # e.g. “01_0004.png”, “02_0945.png”, …
│   │   └── …
│   └── labels/              # matching YOLO-TXT files
│       ├── 01_0004.txt      # same prefix/ID as its image
│       └── …
├── val/                     # held-out validation split
│   ├── images/
│   │   ├── 01_0150.png …
│   │   └── …
│   └── labels/
│       ├── 01_0150.txt …
│       └── …
├── test/                    # final evaluation split (unseen data)
│   ├── images/
│   │   ├── 01_0201.png …
│   │   └── …
│   └── labels/
│       ├── 01_0201.txt …
│       └── …
```

Fig. 2: YOLO-ready dataset layout.

2) YOLOv8 Training Setup: Our YOLO v8 model was trained for 30 epochs using images resized to 640 × 640 pixels, a resolution chosen to optimize detection accuracy while maintaining computational efficiency. We initialized the training from the lightweight YOLOv8-nano variant (*yolov8n.pt*), which provided a balanced trade-off between model complexity and performance. To leverage accelerated computing, training was conducted on a GPU (CUDA-enabled). Additionally, we employed a structured and clearly defined configuration file (*dataset.yaml*) to specify the paths to our datasets, detailing explicit splits for training, validation, and testing sets, along with the corresponding class labels, ensuring a robust and organized training pipeline.

3) LineMOD-to-YOLO Mapping: To align YOLO detections with LineMOD object IDs, a mapping dictionary is defined that converts YOLO class indices to their corresponding LineMOD object identifiers. This is necessary because YOLO assigns class indices based on training order, while LineMOD uses fixed IDs for each object (for example, YOLO class 0 might correspond to LineMOD object ID 1). The mapping take on the following form:

```
YOLO  LineMOD: {"0":1, ..., "13":15}.
```

This mapping is implemented within the dataset class that will be discussed later in the report (see section III-F), ensuring consistent use of object embeddings and 3D models during training and evaluation. After creating an instance of `dataset`, the mapping can be saved to a local file using the call `dataset.save_mapping()` for reproducibility across training and evaluation stages.

B. 6D Pose Estimation Module

For the 6D object pose estimation task, we designed lightweight neural network architectures that directly regress both the 3D translation vector $\mathbf{t}$ and the 3D orientation, represented as a unit quaternion $\mathbf{q}$. These models operate on cropped RGB or RGB-D image regions based on bounding boxes predicted by the detection stage. Inspired by prior work on 6D pose estimation, our implementations draw from PoseNet [3], PoseCNN [4], and DenseFusion [1], while simplifying the architectures to maintain clarity and educational feasibility.
We implemented two main variants:

- **PoseNet6D (RGB-only):** This baseline model uses only the cropped RGB image as input. The image is resized to 224×224 to ensure consistent dimensions and compatibility with pre-trained backbones.
  - *Backbone Network:* A ResNet-18 pretrained on ImageNet is used to extract visual features. The final classification layers are removed, and a global average pooling reduces the feature map to a 512-dimensional vector. We chose ResNet-18 for its balance between accuracy and efficiency, offering fast training and inference while remaining lightweight.
  - *Object Embedding:* Each object is associated with a class ID (mapped from LineMOD to YOLO format), which is embedded into a learnable vector space.
  - *Pose Regression Head:* The RGB features, object embedding, and normalized bounding box coordinates are concatenated and passed through fully connected layers. The network outputs:
    - ∗ A 4D rotation quaternion $\hat{\mathbf{q}}$, normalized via ℓ_2 norm.
    - ∗ A scalar depth value $\hat{Z}$, clamped to the range $[0.1, 1.5]$ meters for stability.
  - *Back-Projection to 3D:* The 3D translation vector is reconstructed by projecting the center of the bounding box into 3D space using the predicted depth and the cropped intrinsics matrix K_{crop}:

    $$\mathbf{t} = \hat{Z} \cdot \begin{bmatrix} (u - c_x)/f_x \\ (v - c_y)/f_y \\ 1 \end{bmatrix} \tag{1}$$

    where (u, v) is the bounding box center in pixels, and $(f_x, f_y), (c_x, c_y)$ are from K_{crop}.

TABLE I: PoseNet (RGB-only) Configuration Summary.

Component	**Details**
Input Modality	RGB
Input Image Size	224×224
Backbone	ResNet-18 (pretrained)
Object ID Encoding	Learnable embedding
Extra Inputs	Normalized bounding box coordinates
Rotation Output	ℓ_2-normalized quaternion $\hat{\mathbf{q}}$
Translation Output	Predicted depth $\hat{Z}$
Back-projection	Using K_{crop}
Mask Usage	None

- **PoseNet6D_RGBD (RGB + Depth):** This extended model incorporates both RGB and depth modalities and builds on the PoseNet architecture with influence from DenseFusion [1] and PoseCNN [4] papers.
  - *Dual Backbones:* An RGB ResNet-18 processes the RGB crop, while a lightweight CNN processes the depth map. Both output spatial feature maps of compatible size (e.g., 7×7).
  - *Global Feature Fusion:* The RGB and depth feature maps are concatenated along the channel dimension and passed through a 1×1 convolution to produce a fused representation. This is followed by global average pooling to obtain a single vector per sample.
  - *Pose Regression Head:* The fused feature vector is concatenated with the object embedding and normalized bounding box. Fully connected layers output a 4D quaternion and scalar depth value.
  - *3D Translation Computation:* As in the RGB-only model, the predicted depth and bounding box center are back-projected into 3D coordinates using the cropped intrinsic matrix.

  Additionally, object masks from the LineMOD dataset are applied externally during training to suppress background pixels in both RGB and depth inputs. This is also done in the DenseFusion paper [1] and allows the model to focus on learning from the object region itself.

TABLE II: PoseNet6D_RGBD Configuration Summary.

Component	**Details**
Input Modality	RGB + Depth
Input Image Size	224×224
RGB Backbone	ResNet-18 (pretrained)
Depth Backbone	Custom CNN
Feature Fusion	Concatenation + 1×1 conv + pooling
Object ID Encoding	Learnable embedding
Extra Inputs	Normalized bounding box coordinates
Rotation Output	ℓ_2-normalized quaternion $\hat{\mathbf{q}}$
Translation Output	Predicted depth $\hat{Z}$
Back-projection	Using K_{crop}
Mask Usage	External (applied during training)

The quaternion output $\hat{\mathbf{q}}$ is later converted to a rotation matrix and evaluated using the ADD metric. This representation was selected due to its numerical stability.

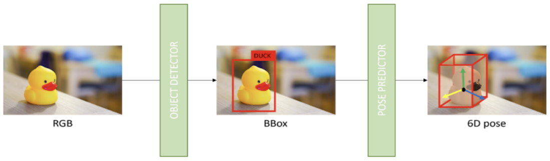


Fig. 3: Illustration of the 6D Pose Estimation Module Pipeline.

C. Model Simplifications and Design Choices

Our implementations draw inspiration from well-established 6D pose estimation frameworks, including PoseNet [3], PoseCNN [4], and DenseFusion [1]. While these models offer high accuracy and robust design, they often involve additional complexity that can be challenging to manage in limited academic settings such as courses. To align with the scope and constraints of this course, we intentionally adopted a simplified and modular design.

Specifically, our models does not do the following:

- **From DenseFusion:** We exclude per-pixel fusion and iterative refinement modules, opting instead for global feature fusion to reduce computational cost and training time.
- **From PoseCNN:** We incorporate object-centric cropping and direct 6D pose regression, but do not include segmentation masks as network outputs, nor do we use multi-stage pipelines.
- **From PoseNet:** We retain the CNN backbone and quaternion-based rotation output, but simplify the loss function by removing uncertainty modeling.

These design choices allow us to maintain the core ideas of typical 6D pose estimation, while creating models that are easier to train, debug, and interpret. The resulting architectures are thus lightweight but are well-suited for demonstrating the fundamental concepts of pose regression using RGB and RGB-D data.

D. Loss Function

The training objective combines Mean Squared Error (MSE) losses for translation and rotation. To improve stability and account for scale differences across axes, we use per-axis weights for translation and a scaling factor β to control the contribution of the rotation loss. The total loss is defined as:

$$\mathcal{L}_{\text{total}} = \frac{1}{N} \sum_{i=1}^{N} \left(\mathcal{L}_{\text{trans}}^{(i)} + \beta \cdot \mathcal{L}_{\text{rot}}^{(i)} \right), \tag{2}$$

where N is the batch size, and $\mathcal{L}_{\text{trans}}^{(i)}$ and $\mathcal{L}_{\text{rot}}^{(i)}$ denote the translation and rotation losses for the i-th sample.

The translation loss is computed as a weighted sum of axis-specific MSE terms:

$$\mathcal{L}_{\text{trans}} = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}^T \cdot \begin{bmatrix} \text{MSE}(\hat{t}_y, t_y) \\ \text{MSE}(\hat{t}_x, t_x) \\ \text{MSE}(\log \hat{t}_z, \log t_z) \end{bmatrix} \in \mathbb{R}, \tag{3}$$

where $(\hat{t}_x, \hat{t}_y, \hat{t}_z)$ and (t_x, t_y, t_z) are the predicted and ground truth translation components, and (w_x, w_y, w_z) are tunable weights. The z-component is log-transformed to reduce the impact of large depth values and improve numerical stability. For rotation, we use unit quaternions. The predicted quaternion $\hat{q}$ is normalized to ensure it represents a valid rotation. The loss is thus defined as:

$$\mathcal{L}_{\text{rot}} = 1 - \left(\langle \hat{q}, q \rangle^2 \right), \tag{4}$$

where $\langle \cdot, \cdot \rangle$ is the quaternion dot product. This formulation accounts for the fact that q and $-q$ represent the same rotation. Optionally, the weights β, w_x, w_y, and w_z can be adjusted dynamically during training to emphasize different components at different stages of training.

E. Evaluation Metric (ADD)

To assess the performance of our 6D pose estimation models, we utilized the Average Distance of Model Points (ADD) metric. The ADD metric calculates the average Euclidean distance between corresponding points on a 3D object model transformed by the predicted pose ($\hat{P} = [\hat{R}|\hat{t}]$) and the ground truth pose ($P = [R|t]$).

$$\text{ADD} = \frac{1}{M} \sum_{j=1}^{M} \|(Rx_j + t) - (\hat{R}x_j + \hat{t})\|$$

where x_j denotes the j-th point of M randomly selected 3D points from the object's 3D model. For symmetric objects, a variation known as ADD-S (Average Distance of Model Points for Symmetric Objects) is often used, which computes the average closest point distance, thereby accounting for rotational ambiguities. This metric is crucial for determining how accurately the predicted pose aligns with the actual object's spatial configuration. Only the regular ADD metric was used in this project.

F. Dataset

The LineMOD dataset served as the foundation for our experiments. It consists of RGB and depth images, ground truth 6D object poses (translation and rotation), object masks, and textured 3D CAD models for 13 distinct objects. All cropped RGB- and depth-images inputs were resized to 224×224 pixels before feeding into the pose estimation models. In this project we used the following components:

- **RGB and Depth Images:** These are the primary input modalities for our models. Cropped RGB and depth regions, based on 2D bounding box detections, were used to train the pose estimation networks. All crops were resized to 224×224 pixels.
- **Ground Truth 6D Poses (R, t):** Each object instance is annotated with a 3D rotation matrix $R \in \mathbb{R}^{3x3}$ and a translation vector $t \in \mathbb{R}^3$, used to assess the regression of pose parameters.
- **3D Object Models:** These CAD models are used during evaluation to compute the ADD metric, which compares the predicted pose to the ground truth pose by

measuring the average distance between corresponding model points as described in section III-E.

- **Camera Intrinsic Matrix K:** The original intrinsic matrix is used to project and deproject between image and 3D space. For pose estimation, a cropped and rescaled version of K, denoted K_{crop}, is computed per detection to account for the region of interest and image resizing. This enables correct mapping of 2D bounding box centers to 3D camera coordinates using the back-projection formula, see (1).

To maintain consistency between YOLO-based detections and LineMOD pose annotations, we used a mapping dictionary (see section III-A3) that aligns YOLO class indices with LineMOD object IDs.

IV. Experiments and Results

A. Experimental Setup

All experiments were conducted in a Kaggle environment equipped with dual-core Xeon CPUs, 32GB of RAM, and dual Nvidia Tesla T4 GPUs. All models were implemented using the PyTorch framework.

B. Object Detection Evaluation

The trained YOLO model demonstrated strong performance on unseen test data, achieving an overall precision (Box P) of 99.6%, recall of 99.1%, mAP50 of 99.2%, and mAP50-95 of 91.3%. These results closely align with training-set metrics (98.0% precision, 98.3% recall, mAP50 99.1%, and mAP50-95 91.5%), indicating robust generalization capability. Notably, performance across most object classes was consistently high, with the majority surpassing 90% in mAP50-95. However, the "squirrel" (Object 02) class exhibited slightly lower recall (89.7%) and mAP50-95 (69.4%), highlighting potential challenges arising from limited data quality or variability. Specifically, during training, several annotations for this object were found to be inaccurate or missing, which likely contributed to its comparatively lower detection performance. Overall, the model demonstrates excellent detection and localization performance on novel data, with minor opportunities for improvement in specific object classes.

1) Confusion Matrix Analysis: The confusion matrix can be seen in figure 4. It was obtained from the test set confirms strong classification performance for nearly all object classes, as indicated by the dominant diagonal entries representing correct detections. Most classes such as "ape," "camera," "pitcher," "cat," and others achieved perfect accuracy, with all 100 ground-truth instances correctly identified. However, the "squirrel" (Object 02) class showed some difficulty, with 89 out of 96 instances correctly detected, while 7 false positives and 8 missed detections appeared, contributing to its lower recall. The "glue", "phone" classes also experienced minor detection errors. Overall, the matrix indicates excellent generalization capability, highlighting only slight areas for further refinement, particularly for the "squirrel" class.

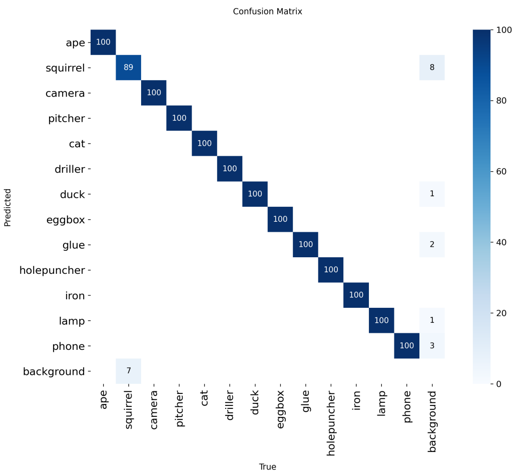


Fig. 4: Confusion Matrix for YOLOv8 Object Detection on the LineMOD dataset.

C. Pose Estimation Evaluation

For training both models we used the RGB images, depth images, segmentation masks, ground truth 6D poses, and 3D CAD models for all 13 object classes. The dataset was organized by object folder, and we adopted the same folder-based train/validation/test splits as used in the DenseFusion [1] and PoseCNN [4] papers. From the validation set, 20% of the data was held out as a final unseen test set to evaluate generalization.

1) Input Pre-processing: To improve model robustness and ensure compatibility with pretrained CNNs, we applied the following preprocessing steps to all cropped object images:

- **Resize and Normalize (RGB):** All RGB crops were resized to 224×224 pixels and normalized using mean $\mu = [0.485,\ 0.456,\ 0.406]$ and standard deviation $\sigma = [0.229,\ 0.224,\ 0.225]$). This normalization matched the expectations of the ResNet-18 backbone.
- **Segmentation Masking (RGB-D only):** Binary masks from the LineMOD dataset were applied to both RGB and depth crops during training. This suppresses background pixels, ensuring the model learns primarily from true object regions. Masking was performed externally in the dataset class, not within the model itself.
- **Depth Smoothing:** A Gaussian blur filter was applied to depth images during training to reduce noise and improve spatial coherence in the extracted depth features. This can be beneficial in early epochs since it may stabilize the learning.

2) Dynamic Loss Weighting (RGB-only model): For the RGB-only model, we used a dynamic loss weighting schedule to emphasize different pose components at different stages of training, see section III-D for details. Early epochs focus highly on depth (Z) prediction, while later stages shift attention to full 3D translation and rotation. The loss weights were adjusted as in table III.

TABLE III: Dynamic Loss Weighting Schedule (RGB-only Model).

Epochs	Translation Weights (w_x, w_y, w_z)	Rotation Weight β
0-4	(0.0, 0.0, 0.1)	1
5-9	(0.5, 0.5, 0.4)	5
10+	(0.1, 0.1, 1.0)	10

This approach allowed the model to first anchor itself in depth estimation, and then refine the translational understanding by sequentially increasing the weight of rotation and translation terms. For the RGB-D model we used $(w_x, w_y, w_z) = (1, 1, 3)$ and $\beta = 1$ for all epochs during training since the yield of employing scheduling was not significant.

TABLE IV: ADD Metric for Pose Estimation Models (lower is better).

Model	Input Modality	ADD (m)
PoseNet	RGB	0.095
PoseNet6D_RGBD	RGB + Depth	**0.0795**

As shown in Table IV, the RGB-only baseline (PoseNet6D) achieved an average ADD of **0.095 m**, while the enhanced PoseNet6D_RGBD model, which incorporates depth input, achieved a lower ADD of **0.0795 m** on the test set. This corresponds to an approximate 17% relative improvement in pose accuracy.

We also examined per-object ADD scores to assess model consistency across different classes apparent in the LineMOD dataset. Table V summarizes these results.

The RGB-D model performed best on Object 01 (0.0512 m) and worst on Object 13 (0.1031 m), likely due to variations in object size, geometry, or occlusion frequency. It is also notable that the total loss was dominated by rotation error. These insights suggest future gains could be achieved by enhancing rotational accuracypotentially via refinement modules or alternative loss functions.

TABLE V: Per-Object ADD Scores on the best performing model (RGB-D).

Object ID (LineMOD)	Name	Average ADD (m)
01	Ape	0.0512
02	Squirrel	0.0870
04	Camera	0.0844
05	Can	0.0761
06	Cat	0.0992
08	Driller	0.0967
09	Duck	0.0644
10	Eggbox	0.0916
11	Glue	0.0632
12	Hole Punch	0.0723
13	Iron	0.1031
14	Lamp	0.0906
15	Phone	0.0757

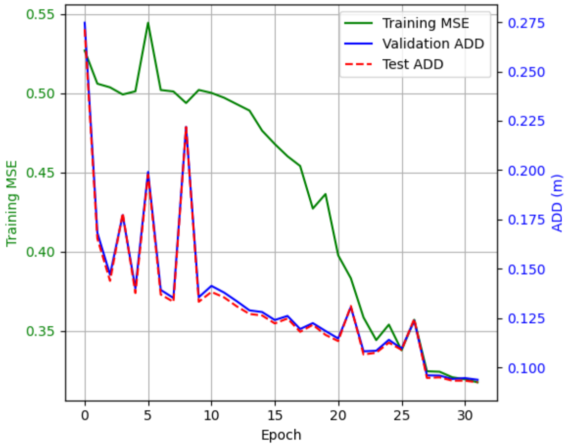


Fig. 5: Training and validation curves for the RGB-only PoseNet6D model.

D. Visualization of Results

Visualizations play a critical role in understanding and evaluating model behavior throughout the training and inference stages. We use both quantitative metrics together with qualitative assessment to interpret the overall performance and identify potential bottlenecks with our models.

1) Training Curves and Generalization Performance: Figure 5 shows the training loss (green) and the ADD metrics for the validation (blue) and test sets (red) for the RGB-only PoseNet model across 30 epochs. The consistent downward trends indicate that the model is learning effectively. The close alignment between validation and test ADD scores reflects generalization with low overfitting. Around epoch 30, all curves begin to plateau, signaling convergence. The relatively smooth evolution of losses across training and evaluation sets suggests stable learning behavior and confirms that the network seems to be capturing features from the RGB input.

Figure 6 displays the training loss for the RGB-D model. The training loss (blue) rapidly decreases in the first 10 epochs, reflecting relatively fast convergence. The validation loss (blue) also converges fast then for the RGB model, with small fluctuations occuring during later epochs. The closeness of the test and validation lines further confirms the model's ability to generalize.

2) Qualitative Visualization Results: To qualitatively assess 6D pose predictions, we visualized model outputs by overlaying 3D object meshes and projected coordinate axes into the original image frames using the predicted camera pose.

Figure 7 illustrates an overlay of the predicted and ground truth object poses using the RGB-D model. Visual alignment between the red (predicted) and green (ground truth) meshes is particularly tight for well-visible objects, while slight misalignments highlight challenges for predicting objects under occlusion.

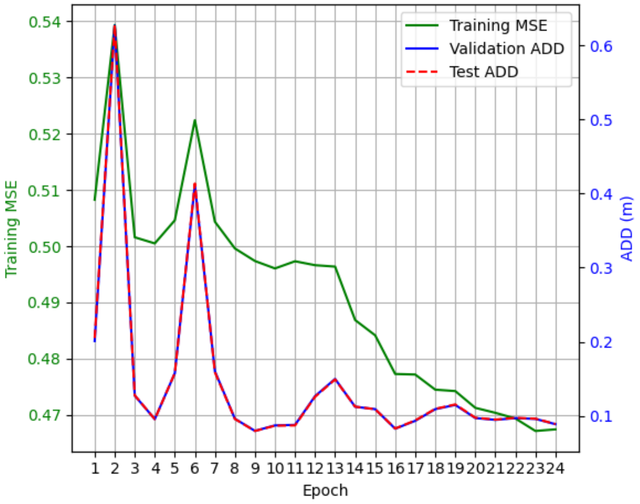


Fig. 6: Training and validation loss curves for the RGB-D model. The scale makes the blue and red dotted lines appear identical while in reality they are not.

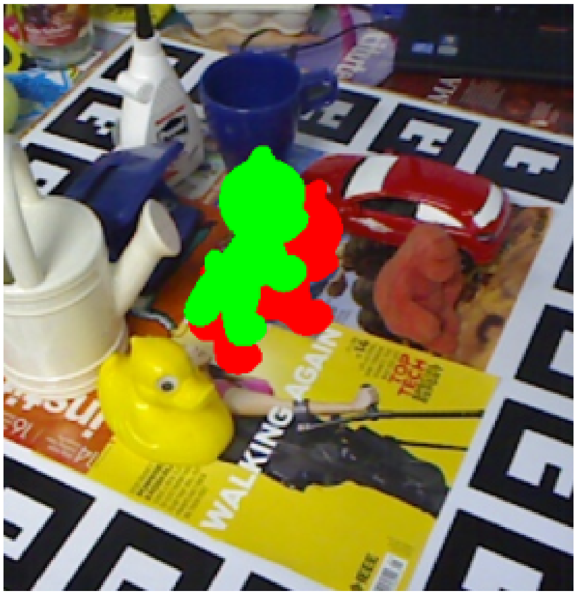


Fig. 7: Overlayed prediction for the RGB-D model. Ground truth pose (green mesh), predicted pose (red mesh). This particular prediction has $ADD \approx 0.045$ (m).

In Figure 8, we visualize the predicted rotation using 3D coordinate axes rendered at the object origin. This visual aid is convenient for inspecting rotational accuracy, especially for objects with rotational symmetries or odd shapes.

Across multiple images, we observed that the RGB-D model performs more reliably under occlusion, benefiting from depth cues where RGB features alone may fail. For example, objects like the glue stick and eggbox, both highly symmetric, were more accurately localized using depth-enhanced fusion.

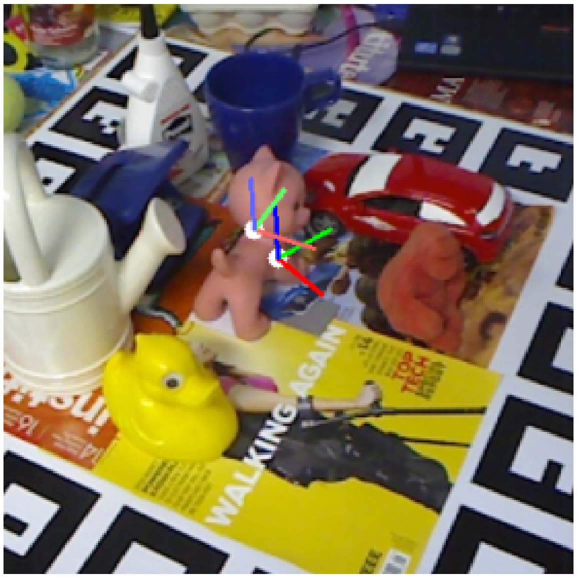


Fig. 8: Pose visualization with 3D coordinate axes (red: x, green: y, blue: z). This particular prediction has $ADD \approx 0.045$ (m).

3) Interpretation: The combination of evaluation metrics and visualizations gives us a more complete picture of how well our models are working. From the training curves, we can see that both the RGB-only and RGB-D models improve steadily during training and generalize well to new data. The RGB-D model, in particular, benefits from having access to depth information.

The visual results support this, showing that in many examples, the predicted object poses line up fairly well with the actual objects in the scene. This is especially true when the objects are clearly visible and not too cluttered or occluded. However, we also see some failure cases, such as objects being slightly misaligned or rotated incorrectly. These cases can most of the times be detect just by looking at average error numbers.

Overall, the visualizations help us spot both strengths and weaknesses in the model. They show where it performs reliably and where it struggles, offering helpful clues for possible improvements in future versions, like dealing better with symmetry, occlusion, or adding a refinement step to improve pose estimates.

V. Conclusion

This project successfully created a modular pipeline for 6D object pose estimation. It began with effective 2D object detection using **YOLOv8** and then employed a *PoseNet*-inspired model for the final pose regression. The core achievement was the development of an **RGB-D** model that improved accuracy by approximately 17% over its RGB-only counterpart, proving particularly beneficial for challenging objects that are symmetric, have low-texture, or are partially occluded.

Despite these advancements, our models feature several simplifications, including the omission of dense per-pixel fusion and iterative refinement steps, reliance on external segmentation masks, and a manually defined loss weighting schedule. These limitations present clear directions for future enhancements, such as adopting more sophisticated architectures (ex. transformer-based backbones or diffusion-based regressors), utilizing advanced loss functions like the geodesic distance for rotation, and integrating learned segmentation masks.

This project highlights the benefits of integrating depth data for 6D pose estimation, showing that a streamlined and well-structured pipeline can balance computational efficiency, clarity of design, and strong performance.

Limitations and Future Work

Despite promising results, our models have clear limitations:

- **Simplified Architectures:** Our implementation does not include dense per-pixel feature fusion or iterative refinement, both of which are known to improve pose accuracy in complex scenes.
- **No End-to-End Segmentation:** Unlike PoseCNN, we do not predict segmentation masks; instead, we use ground-truth masks externally during training to suppress background noise.
- **Fixed Loss Scheduling:** We use a manually defined loss weighting schedule rather than learning the optimal balance between rotation and translation dynamically.

These limitations point to several natural directions for future work:

- **Stronger Architectures and Loss Functions:** Incorporate advanced architectures, such as transformer-based backbones or diffusion-based pose regressors, and explore more robust loss functions (for example the geodesic distance for rotation).
- **Learned Segmentation and Masking:** Add a segmentation head to jointly predict object masks, which could improve cropping accuracy and reduce reliance on external features.

This project has deepened our understanding of 6D pose estimation and shown how modular designs, when thoughtfully constructed, can balance clarity, computational feasibility, and performance. While there is still room for improvement, our results already demonstrate the value of combining RGB and depth for spatial understanding.

References

[1] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, “Densefusion: 6D object pose estimation by iterative dense fusion,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 3343–3352. [Online]. Available: https://arxiv.org/abs/1901.04780

[2] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, “SSD-6D: Making RGB-based 3d detection and 6d pose estimation great again,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1521–1529. [Online]. Available: https://arxiv.org/abs/1711.10006

[3] A. Kendall, M. Grimes, and R. Cipolla, “Posenet: A convolutional network for real-time 6-dof camera relocalization,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2938–2946. [Online]. Available: https://arxiv.org/abs/1505.07427

[4] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, “Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes,” in *Proceedings of Robotics: Science and Systems (RSS)*, 2018. [Online]. Available: https://arxiv.org/abs/1711.00199

[5] A. Vijayakumar and S. Vairavasundaram, “YOLO-based object detection models: A review and its applications,” *Multimedia Tools and Applications*, vol. 83, pp. 83 535–83 574, 2024. [Online]. Available: https://doi.org/10.1007/s11042-024-18872-y

[6] T. Diwan, G. Anirudh, and J. V. Tembhurne, “Object detection using YOLO: challenges, architectural successors, datasets and applications,” *Multimedia Tools and Applications*, vol. 82, pp. 9243–9275, 2023. [Online]. Available: https://doi.org/10.1007/s11042-022-13644-y