6D Pose Estimation via Keypoint Heatmap Regression with Depth-Driven Residual Neural Networks

Ismail Aljosevic
Politecnico di Torino
s337769

Amir Masoud Almasi
Politecnico di Torino
s337006

Ana Parovic
Politecnico di Torino
s344174

Ashkan Shafiei
Politecnico di Torino
s342583

Abstract

1. Introduction

6D Pose Estimation is the problem of figuring out where an object is in 3D space and how it is oriented, using just a single RGB image. It's a key task in many areas of computer vision, especially in robotics, augmented reality, and autonomous systems, where understanding how an object is positioned is crucial for interaction and navigation.

Earlier methods for pose estimation often relied on depth sensors or handcrafted geometric techniques. In recent years, deep learning has made it possible to estimate object pose directly from RGB images, which simplifies the pipeline and makes it more flexible. Still, the task remains difficult—especially when objects are partially hidden, appear in cluttered scenes, or have similar textures.

In this project, we design a complete deep learning pipeline that predicts the 6D pose of objects using only RGB data. Our method starts with object detection using YOLOv10m, then uses a ResNet18 model to predict 2D heatmaps for selected 3D keypoints. We compare two ways of choosing these 3D keypoints: Curvature Point Sampling (CPS), which focuses on detailed surface areas, and Farthest Point Sampling (FPS), which spreads keypoints evenly. After predicting the 2D keypoints, we use a Perspective-n-Point (PnP) algorithm to estimate the full 6D pose. Ground-truth labels are generated by projecting the 3D keypoints using the known camera intrinsics. All experiments are carried out on a subset of the LINEMOD dataset.

2. Related Work

This section discusses some of the main methods that have recently driven progress in 6D pose estimation, with a particular focus on approaches that use keypoint predictions.

The LINEMOD dataset [3], introduced in 2012, was one of the first standardized benchmarks for 6D object pose estimation. It includes several texture-less objects under different lighting and occlusion settings, and has been widely used to evaluate pose estimation methods. Its simplicity and controlled setup made it a preferred choice for initial experimentation and model validation.

Zhao et al. [8] proposed one of the early deep learning approaches that detects keypoints on the object's surface using heatmaps. The 2D keypoint locations are then used with a PnP solver to estimate the full 6D pose. Their method demonstrated that keypoint-based supervision could outperform direct pose regression under many conditions.

PVNet [4] improved robustness by introducing a pixel-wise voting strategy, where each pixel predicts the direction to each keypoint. They also used Farthest Point Sampling (FPS) to select keypoints that are spread across the object surface. This approach achieved high accuracy even in scenes with heavy occlusion and partial views.

Aing and Lie [2] proposed Curvature Point Sampling (CPS), which selects keypoints from high-curvature regions of the object. These points are usually easier to detect and less ambiguous. CPS is particularly useful for objects with smooth surfaces where traditional keypoints may be hard to localize.

Finally, Zappel et al. [7] extended the heatmap approach by adding part affinity fields to learn the spatial relationships between keypoints, which helps when some keypoints are occluded or close together.

3. Methodology

This section describes the pipeline we used to estimate the 6D pose of objects from RGB images. We explain the dataset setup, how we detect objects, how we choose 3D

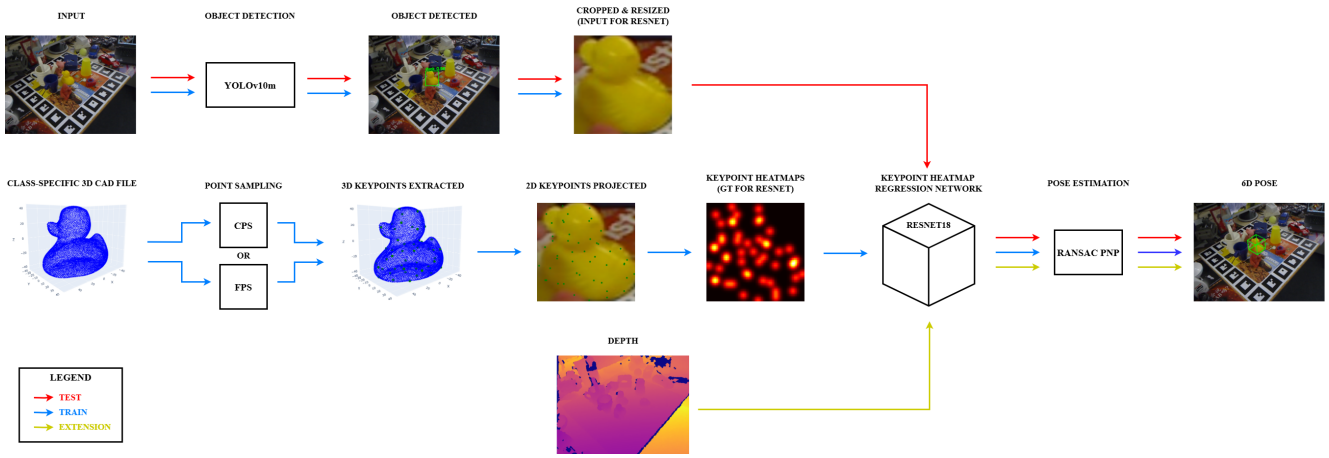

Figure 1. Pipeline overview with clear separation of training supervision (blue), test flow (red), and depth extension (yellow).

keypoints, and how we train a neural network to predict 2D heatmaps. Finally, we describe how the 6D pose is recovered using the predicted keypoints.

3.1. Hardware and Resources

All models in our entire pipeline were trained using Google Colab Pro with access to NVIDIA Tesla T4 GPUs (16 GB VRAM) and 32 GB RAM.

3.2. Dataset

The dataset used in our project was a subset of the LINEMOD dataset [3], a popular benchmark for 6D object pose estimation. LINEMOD includes 15 texture-less object classes captured in real-world environments under varying poses, occlusions, and lighting conditions. Each object instance is provided with a 3D mesh model, RGB images, depth maps, masks, and corresponding ground-truth poses.

In our setup, we excluded object classes ‘03‘ and ‘07‘ due to inconsistency in annotations, resulting in a total of 13 object categories: ‘01 (ape)‘, ‘02 (bench vise)‘, ‘04 (camera)‘, ‘05 (can)‘, ‘06 (cat)‘, ‘08 (driller)‘, ‘09 (duck)‘, ‘10 (eggbox)‘, ‘11 (glue)‘, ‘12 (hole puncher)‘, ‘13 (iron)‘, ‘14 (lamp)‘, and ‘15 (phone)‘.

We used a total of 15,800 RGB images from 13 object classes in the LINEMOD dataset. For each image, we extracted the camera intrinsic parameters and the ground-truth object pose. The dataset was split into 90% for training (14,220 images) and 10% for testing (1,580 images), with the test set reserved for final evaluation of the complete pipeline.

3.3. Object Detection

The first stage of our 6D pose estimation pipeline involved detecting the object of interest in each RGB image. We used the YOLOv10m [6] model to locate and classify the object present in each RGB image. YOLOv10m is a recent version of the YOLO (You Only Look Once) family of object detectors [5]. Its speed and low resource usage made it a good fit for our hardware setup without decreasing accuracy.

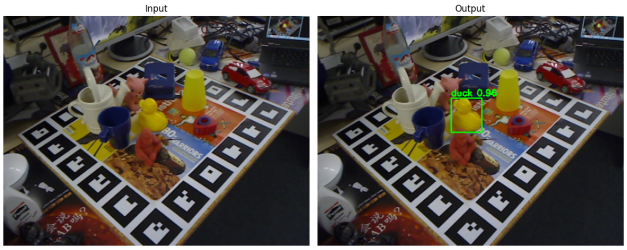

Figure 2. YOLOv10m inference results for a validation image of the “duck” class, including the model’s predicted bounding box, label, and confidence score.

YOLO is a one-stage object detector, meaning it predicts both bounding boxes and class labels in a single forward pass through the network. The input image is divided into a grid, and each cell is responsible for predicting:

- The bounding box coordinates (x, y, w, h), representing the normalized center location and size,
- An objectness score p_{obj}, indicating the likelihood that an object is present in the box,
- A set of class probabilities $p_{cls}^{(c)}$ for each class c.

The final confidence score for class c at a given location is computed as:

$$\text{Confidence}_{s,b}^{(c)} = p_{obj}^{(s,b)} \cdot p_{cls}^{(s,b,c)}$$

In our case, this score estimates how likely it is that the bounding box contains an object of the correct class.

We trained the YOLOv10m model on 13 object classes from the LINEMOD dataset, using 11,376 training images. The model is trained by minimizing a weighted sum of classification loss, objectness loss, and bounding box regression

loss, defined as:

$$\mathcal{L} = \lambda_{cls}\mathcal{L}_{cls} + \lambda_{obj}\mathcal{L}_{obj} + \lambda_{bbox}\mathcal{L}_{bbox}$$

where the λ terms control the contribution of each term to the total loss.

In object detection, mean Average Precision (mAP) is a standard way to measure performance. First, for each object class, the model's precision and recall are plotted at different confidence thresholds. The area under this curve gives the Average Precision (AP) for that class. Then, mAP is calculated by averaging all the AP values across the C classes:

$$\text{mAP} = \frac{1}{C}\sum_{c=1}^{C} \text{AP}_c$$

A high mAP means that the model is good at both finding objects and drawing accurate bounding boxes around them.

Once trained, we tested our YOLOv10m detector on a validation set of 2,844 images from 13 object classes. The model performed very well and gave strong results, as shown below:

- **mAP@50** (accuracy at a basic overlap level): **0.995**
- **mAP@50:95** (average accuracy over stricter overlap levels): **0.974**
- **Precision** (how many predicted boxes were correct): **0.997**
- **Recall** (how many actual objects were detected): **0.996**

Table 1. YOLOv10m performance per class on the validation set

Class	Instances	Precision	Recall	mAP@50:95
ape	253	1.000	0.999	0.961
benchvise	225	0.991	1.000	0.981
camera	203	0.998	1.000	0.976
can	204	0.994	1.000	0.989
cat	207	0.998	1.000	0.980
driller	210	1.000	0.995	0.980
duck	209	0.998	0.981	0.967
eggbox	210	0.995	0.987	0.977
glue	242	0.996	0.998	0.945
holepuncher	238	0.999	1.000	0.970
iron	180	0.994	0.999	0.984
lamp	245	1.000	0.995	0.978
phone	218	1.000	0.997	0.979
Mean	2844	**0.997**	**0.996**	**0.974**

As presented in Table 1, YOLOv10m detected objects very accurately in the 13 classes. Most classes had precision and recall above 0.99, meaning the model rarely missed any objects or made wrong predictions. Even the lowest score (0.945 for "glue") was still high. These results show that the detector worked well and gave reliable outputs for the next steps in our pose estimation pipeline.

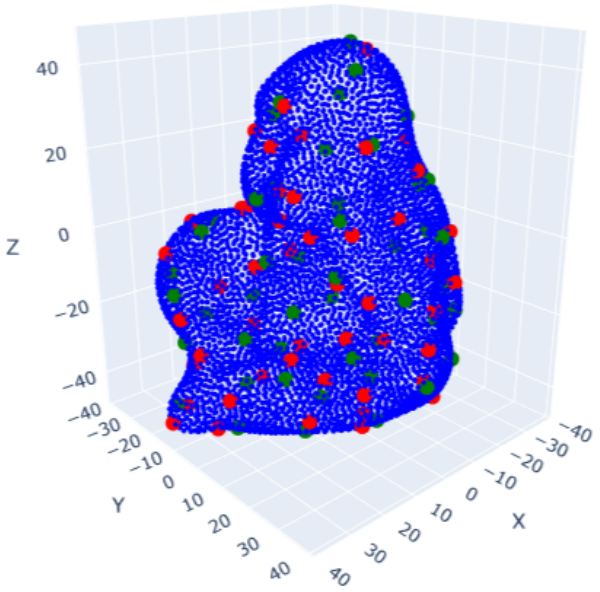

Figure 3. Overlay of CPS (red) and FPS (green) keypoints on the 3D CAD model of an object from the class 'ape'. CPS concentrate on high-curvature regions, while FPS uniformly distributes points.

After training the object detection model, we used the predicted bounding boxes from YOLO to crop each object from the RGB images. These cropped regions were then resized to 256×256 pixels to keep the input size consistent for the next stage. This size was chosen because it offers a good balance between speed and accuracy, and it easily scales down to the 64×64 resolution needed for the heatmap labels.

3.4. 3D Keypoint Sampling

To train the keypoint regression model, we first extracted a consistent set of 3D keypoints from each object's CAD model. These keypoints were later projected to 2D image space using the known object poses and camera intrinsics to generate heatmap labels. We selected **50 keypoints per object** to balance geometric richness with computational efficiency.

We implemented two different point sampling strategies: *Farthest Point Sampling (FPS)* and *Curvature Point Sampling (CPS)*, and later compare their impact on the final 6D pose estimation results.

Farthest Point Sampling (FPS). FPS selects points that are maximally spread out over the object surface. Starting from a randomly chosen seed point, the method iteratively selects the next point that maximizes the minimum distance to the set of already chosen keypoints. Formally, given a point cloud $P = \{p_1, p_2, \ldots, p_N\} \subset \mathbb{R}^3$, FPS constructs the keypoint set S as follows:

$$\text{Initialize: } S = \{p_{i_0}\} \text{ (random)}$$
$$\text{For } k = 1 \text{ to } K:$$
$$p_{i_k} = \arg \max_{p \in P \setminus S} \min_{q \in S} \|p - q\|_2$$
$$S \leftarrow S \cup \{p_{i_k}\}$$

This process continues until $K = 50$ keypoints are selected. FPS ensures even spatial coverage and is widely used in pose estimation pipelines such as PVNet [4].

Curvature Point Sampling (CPS). CPS selects keypoints from the most detailed parts of the object surface. We first estimate the curvature at each point on the 3D model by looking at how much the surface bends around it. This is done using the eigenvalues of the covariance matrix of its k-nearest neighbors. The curvature for point p_i is computed as:

$$\kappa(p_i) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2 + \varepsilon}$$

where $\lambda_0, \lambda_1, \lambda_2$ are the eigenvalues sorted in ascending order, and ε is a small constant to avoid division by zero.

Then, to encourage diversity and avoid selecting keypoints too close to the center, we assign a final score to each point based on both its curvature and its distance from the object center $\bar{p}$:

$$\text{score}(p_i) = \kappa(p_i) \cdot \|p_i - \bar{p}\|_2$$

We then pick the top 50 points with the highest scores. This ensures that the selected keypoints are both detailed and well spread across the object. This method is inspired by the approach used by Aing and Lie [1], who applied curvature sampling to select points on the surface.

3.5. Keypoint Heatmap Generation

Since our keypoint regression model takes cropped and resized RGB images as input and produces heatmaps as output, we first needed to generate ground truth heatmaps for each training image. These heatmaps serve as supervision signals that guide the model to learn the spatial distribution of keypoints.

To construct them, we began by projecting each object's 3D keypoints into the 2D image plane. This projection used the known camera intrinsics $K \in \mathbb{R}^{3\times3}$ and the object's rotation and translation matrices $R \in \mathbb{R}^{3\times3}$, $T \in \mathbb{R}^{3\times1}$, both provided in the dataset annotations. The 2D location (u, v) of each keypoint was obtained using the standard perspective projection:

$$[u, v, 1]^T \propto K \cdot (R \cdot X + T)$$

where $X \in \mathbb{R}^3$ is a 3D keypoint.

Because the model is trained on 256×256 image crops predicted by YOLOv10m, all projected 2D keypoints were normalized to match this cropped coordinate frame. This ensures that the supervision and input image are spatially aligned.

Each 2D keypoint was then converted into a heatmap by placing a 2D Gaussian function centered at its location. For a given keypoint located at (x, y), the corresponding heatmap $H(x', y')$ was defined on a 64×64 grid as:

$$H(x', y') = \exp\left(-\frac{(x' - x)^2 + (y' - y)^2}{2\sigma^2}\right)$$

We used $\sigma = 2.0$, which creates a sharper heatmap and helps the model learn more precise keypoint locations.

This process was repeated for all keypoints, and the resulting heatmaps were stacked to form a target tensor of shape $(N, 64, 64)$, where $N = 50$ is the number of keypoints per object. These heatmap tensors were used as ground truth outputs for training the regression model.

An example of this process is illustrated in Figure 4, where we show the original cropped image, the 2D keypoints sampled using FPS, and the combined heatmap used as supervision.

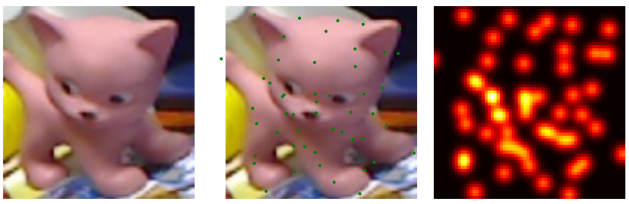

Figure 4. Visualization of the keypoint heatmap supervision process. Left: Cropped and resized image of an object from the class 'cat'. Center: 50 2D FPS keypoints overlaid on the image. Right: Combined heatmap formed by summing the 50 individual keypoint heatmaps.

References

[1] Lay Aing and Wernhuar Lie. Detecting object surface keypoints from a single rgb image via deep learning network for 6-dof pose estimation. *IEEE Access*, 9:77729–77741, 2021.

[2] Louis Aing and Weng Nam Lie. Detecting object surface keypoints from a single rgb image via deep learning network for 6-dof pose estimation. *IEEE Access*, 9:77729–77741, 2021.

[3] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision*, pages 548–562. Springer, 2012.

[4] Sida Peng, Yuan Liu, Qixing Huang, Hujun Bao, and Xiaowei Zhou. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *Proceedings of the IEEE/CVF Conference on*

Computer Vision and Pattern Recognition, pages 4561–4570, 2019.

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[6] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, and Youn-Long Lin. YOLOv10: Real-Time Object Detection and Recognition. `https://github.com/WongKinYiu/yolov10`, 2024. Accessed: 2025-06-05.

[7] Moritz Zappel, Simon Bultmann, and Sven Behnke. 6d object pose estimation using keypoints and part affinity fields. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2596–2603, 2022.

[8] Zelin Zhao, Gu Peng, Haoyu Wang, Hao-Shu Fang, Cewu Li, and Caiming Lu. Estimating 6d pose from localizing designated surface keypoints. *arXiv preprint arXiv:1809.08550*, 2018.