16811: Mathematical Fundamentals for Robotics

# Neural Radiance Fields for 6D Pose Estimation

Team members: Abhishek Tandon (atandon2), Poorvi Hebbar (phebbar), Siva Ramakrishnan (sivarams)

## Introduction

This project looks into Neural Radiance Fields (NeRF) for 6D pose estimation. Since its introduction in 2020, Neural Radiance Fields (NeRF)[9] learned image priors from large-scale image sets. NeRFs have found uses in robotics applications for visuomotor control and vision-only robot navigation (with a pre-mapped NERF environment). The end application analyzed here concerns how NeRFs can be employed for 6D pose estimation. 6D pose estimation is the task of detecting the location and orientation of an object given a 2D image of the object.

This project first looks into 6D Pose Estimation and its standard methods such as the Perspective-n-Point (PnP) method and its limitations. It then discusses how deep learning models are used in conjunction with PnP.

The project then explores NeRFs to form a thorough understanding of the topic and then evaluates the method Inverted Neural Radiance Fields (iNeRF) which is used for pose estimation.

# 6D Pose Estimation

6D Pose Estimation is the task of predicting the orientation and location of an object given an image taken from a camera. The knowledge about the object pose then allows the robot to reason about the object in the 3D world as the camera location is known with respect to the world center. Using the predicted pose, a robot arm can place its gripper accordingly and successfully manipulate the object. Pose Estimation also finds use in planning tasks such as planning robot motion around objects and to power augmented applications.

## Formal Problem Setup

Formally the problem setup consists of:

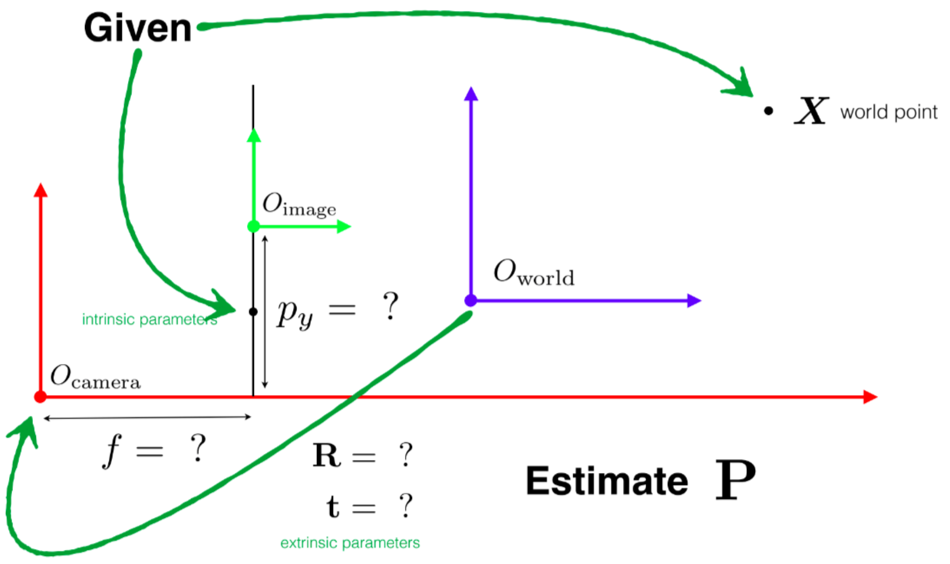
1. 3D Object
2. 2D image of the object

This is given through 2D points on the image matched to 3D points on object. These are known as 2D-3D correspondences.

The task is to estimate the camera matrix P for which we have these 2D-3D correspondences.

$$\boldsymbol{x} = \underbrace{\boldsymbol{f}}_{\text{camera model}}(\mathbf{X}; \underbrace{\boldsymbol{p}}_{\text{parameters}}) = \underbrace{\mathbf{P}}_{\text{Camera matrix}}\mathbf{X}$$

Figure 1: Pose Estimation equation [9]



Firgure 2: Pose Estimation problem [9]

## Method: Perspective-n-Point

Given the correspondences we can write the equation in matrix form.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Figure 3: Matrix form of 2D-3D correspondence [9]

Using linear algebra, we can manipulate the above matrix to give the below equations:

$$\boldsymbol{p}_2^\top \boldsymbol{X} - \boldsymbol{p}_3^\top \boldsymbol{X} y' = 0$$

$$\boldsymbol{p}_1^\top \boldsymbol{X} - \boldsymbol{p}_3^\top \boldsymbol{X} x' = 0$$

Figure 4: Constraint Equations (p1.T, p2.T, p3.T are the first, second and third row of the P matrix) [9]

Since the camera matrix P is orthogonal, we can apply those constraints to the unknowns. Since each pont gives 2 equations, we now need only 3 correspondences to determine all the unknowns. This method is known as the Perspective - 3 - Point. (P3P) [8]

Using the n correspondences in the general case of Perspective-n-Point method, we can form the below matrix and then apply SVD to find the unknowns. The solutions is the eigenvector corresponding to the smallest eigenvector.

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\mathbf{A}\boldsymbol{x}\|^2 \text{ subject to } \|\boldsymbol{x}\|^2 = 1$$

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{X}_1^\top & \mathbf{0} & -x'\boldsymbol{X}_1^\top \\ \mathbf{0} & \boldsymbol{X}_1^\top & -y'\boldsymbol{X}_1^\top \\ \vdots & \vdots & \vdots \\ \boldsymbol{X}_N^\top & \mathbf{0} & -x'\boldsymbol{X}_N^\top \\ \mathbf{0} & \boldsymbol{X}_N^\top & -y'\boldsymbol{X}_N^\top \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{bmatrix}$$

Figure 5: Matrix format of system of equations [9]

The PnP method works well and is able to find good estimate the object pose but is limited on its dependency for perfect 2D-3D correspondences. RANSAC is used in conjunction with PnP to be robust to the noise in the correspondences. The standard implementations of PnP also use Levenberg-Marquadt (LM) optimization to further tune the P matrix found from PnP.

## Finding correspondeces

Perspective-n-Point method uses 2D-3D correspondences to estimate the object pose, but these correspondences also need to be determined. This is the blind PnP problem where the correspondences are unknown. Below are some methods to find the correspondences:

1. 2D-2D correspondences

   In this method, first the 3D model is projected to an image and then correspondences are found between the two images. These 2D-2D correspondences are then mapped back to the 3D model.

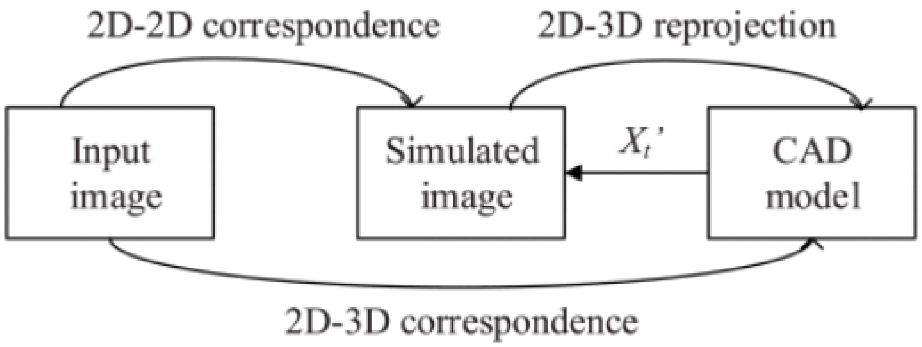
   

   Figure 6: 2D-2D-projection-3D [5]

2. Using Deep Learning Model

   In this case the strategy is to train a model to output the correspondences. These models take into account the 3D model of the object and the 2D image and directly regress for the correspondences.

After finding the correspondences, these methods apply PnP with RANSAC.

## Results

The PnP method was applied usign OpenCV implementation on the LineMOD[7] synthetic dataset. The correspondences were found using PVNet[6] deep neural network.

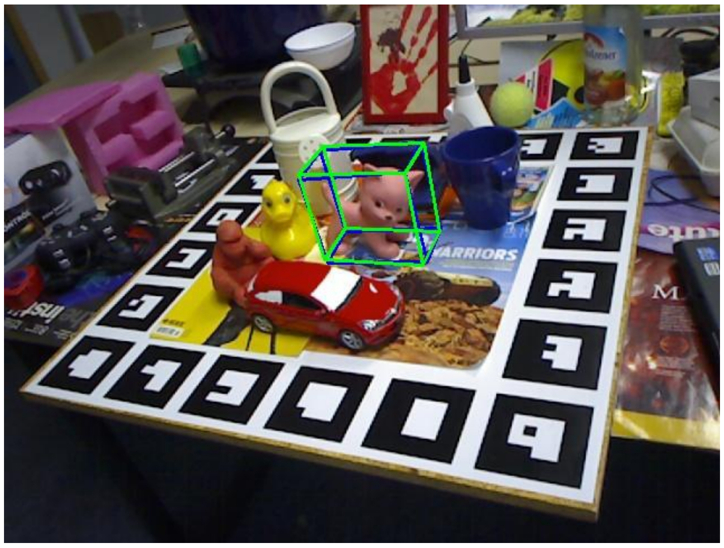


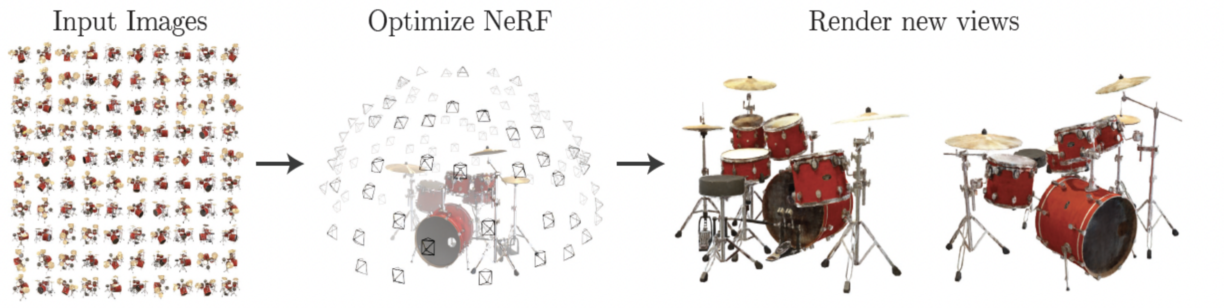
Figure 7: result figure (green is the ground truth box, blue is the predicted box)

As seen in the above figure the method works well even in cases of heavy occlusion and when there are multiple objects in the scene.

# Neural Radiance Fields (NeRF)

**What is NeRF?**

NeRF stands for Neural Radiance Fields. Let's understand a few concepts before understanding how exactly a NeRF works [8]:



**Rendering**: process of creating an image from a 3D model containing features such as textures, shading, shadows, lighting, and viewpoint. The role of the rendering engine is to process these features to create a realistic image. There are main methods being used for rendering:

1) Rasterization: projects objects geometrically based on information in the model, without optical effects
2) Ray casting: calculates an image from a specific point of view using basic optical laws of reflection
3) Ray tracing: uses Monte Carlo techniques to achieve a realistic image in a far shorter time

**Volume rendering**: creates a 2D projection of a 3D discretely sampled dataset. Obtains RGBa for every voxel in space in a particular ray direction and gets a RGB color for the corresponding pixel in 2D image.

**View synthesis**: creates a 3D view from a  series of 2D images by predicting the depth given different perspectives of an object

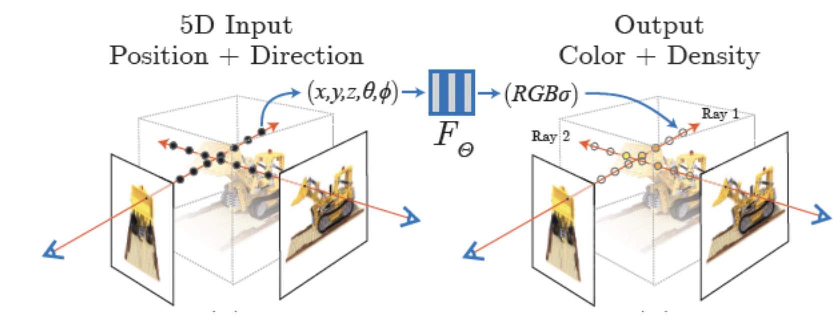
**NeRF**: A neural radiance field (NeRF) is a fully-connected neural network that can generate novel views of complex 3D scenes, based on a partial set of 2D images. It takes a set of input images of a scene and renders the complete scene by interpolating between the scenes. Interpolation between 2D scenes is done using a Continuous Volumetric Scene function F: (x,y,z,θ,Φ) -> (r,g,b,a). We try to optimize this *continuous volumetric scene function* which can be further used to produce novel views.

The Continuous volumetric scene function can be viewed as 5D vector valued function which takes input as a 3D location x = (x; y; z) and 2D viewing direction (θ; Φ) and outputs an emitted color c = (r; g; b) and volume density (α).

Briefly, to understand how a NeRF works: We first generate a sampled set of 3D points—by marching camera rays through the scene. We then produce an output set of densities and colors—by inputting your sampled points with their corresponding 2D viewing directions into the neural network. Finally, we accumulate our densities and colors into a 2D image—by using classical volume rendering techniques. This overview of NerF can be better understood in the following detailed steps:

1) **Nerf Scene Representation:**
   Say d is a 3D cartesian unit vector representing the viewing direction. We approximate a continuous 5D scene representation with an MLP network F: (x; d) -> (c; a) and optimize its weights to map from each input 5D coordinate to its corresponding volume density and directional emitted color. We observe that the volume density, a, is a function of only the location x, while the RGB color c is predicted as a function of both location and viewing direction.

   MLP F first processes the input 3D coordinate x with 8 fully-connected layers (using ReLU activations and 256 channels per layer), and outputs and a 256-dimensional feature vector. This feature vector is then concatenated with the camera ray's viewing direction and passed to one additional fully-connected layer (using a ReLU activation and 128 channels) that outputs the view-dependent RGB color. Interpolation between the given set of images is done using this continuous volumetric scene function. If we consider a voxel at (x,y,z) and the viewing ray direction as (theta,phi) then the F function here basically maps (x,y,z,theta,phi) to (RGB sigma) of the voxel where sigma is volume density.



## 2) Volume Rendering with radiance fields

Once we have this RGB sigma for every voxel in a ray direction, we can get RGB of the 2D corresponding pixel. This color rendering basically creates a 2D projection of a 3D model.

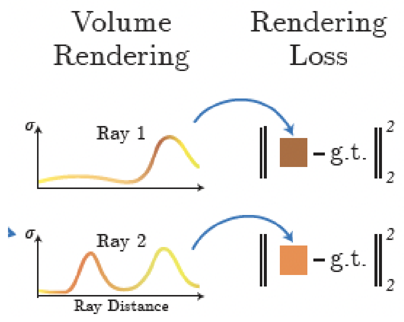
Estimated color C(r) of camera ray r(t) = o + t**d** with near and far bounds tn and tf is:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t),\mathbf{d})dt\,,\ \text{where } T(t) = \exp\Bigg(-\int_{t_n}^{t}\sigma(\mathbf{r}(s))ds$$

We use a stratified sampling approach to estimate this integral where we partition [tn; tf] into N evenly-spaced bins and then draw one sample uniformly at random from within each bin:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1-\exp(-\sigma_i\delta_i))\mathbf{c}_i\,,\ \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right),$$

Where delta_i=ti+1 - ti, distance between adjacent samples


Volume Rendering
Rendering Loss
$\sigma$
Ray 1
$\left\| \blacksquare - \text{g.t.} \right\|_2^2$
$\sigma$
Ray 2
$\left\| \blacksquare - \text{g.t.} \right\|_2^2$
Ray Distance

So overall, we synthesize images by sampling 5D coordinates (location and viewing direction) along camera rays, feeding those locations into an MLP to produce a color and volume density, and using volume rendering techniques to composite these values into an image. This rendering function is differentiable, so we can optimize our scene representation by minimizing the residual between synthesized and ground truth observed images.

## 3) Optimizing a NERF

There are 2 main ways to optimize the NeRF that the original paper presents:

1. Positional encoding

Deep networks are biased towards learning low frequencies. Therefore the network F directly operating on (x,y,z,θ,Φ) input coordinates results in rendering that performs poorly at representing high-frequency variation in color and geometry. Also, mapping the inputs to a higher dimensional space using high frequency functions before passing them to the network enables better fitting of data that contains high frequency variation. Positional encoding basically maps the inputs to a higher dimensional space so that minute details with high frequency variation in color and geometry are captured. Therefore, we find improved performance by reformulating F_theta as (F_theta’,gamma) where gamma is the mapping from R to a higher dimensional space R^2L and F_theta’ is still an MLP.

2. Hierarchical volume sampling

The color rendering faces one major backfoot in color rendering, i.e Free space and occluded regions that do not contribute to the rendered image are still sampled repeatedly. We can use hierarchical representation that increases rendering efficiency by allocating samples proportionally to their expected effect on the final rendering. Instead of just using a single network to represent the scene, we simultaneously optimize two networks: one “coarse" and one “fine". We first sample a set of Nc locations using stratified sampling, and evaluate the “coarse" network at these locations. Given the output of this “coarse" network, we then produce a more informed sampling of points along each ray where samples are biased towards the relevant parts of the volume. We now sample a second set of Nf locations from this distribution using inverse transform sampling, evaluate our “fine" network at the union of the first and second set of samples, and compute the final rendered color of the ray ^ Cf ® using all Nc+Nf samples.

For volume sampling, let’s take an example. Suppose we have a laptop in front of us. We can say that the air between us and the laptop and the stuff behind the laptop screen are in no way contributing to the color of a 2D pixel in the laptop’s image. To optimize, we therefore sample course points and then some fine points in the regions that matter the most. This 2 step sampling to ignore free space and occluded regions is called Hierarchical Volume sampling.

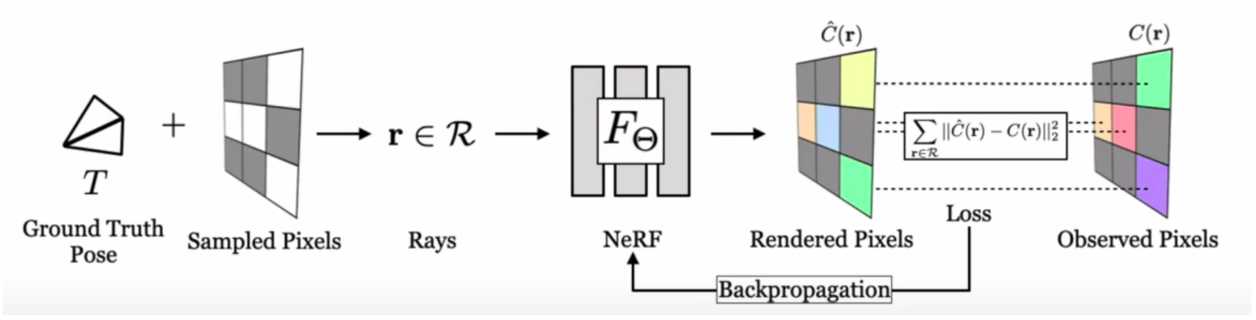
**4) Implementation details**

We optimize a separate neural continuous volume representation network for each scene. This requires only a dataset of captured RGB images of the scene, the corresponding camera poses and intrinsic parameters, and scene bounds (assumed to be given). At each optimization iteration, we randomly sample a batch of camera rays from the set of all pixels in the dataset, and then follow the hierarchical sampling to query Nc samples from the coarse network and Nc + Nf samples from the  ne network. We then use the volume rendering procedure to render the color of each ray from both sets of samples. Our loss is simply the total squared error between the rendered and true pixel colors for both the coarse and  ne renderings:

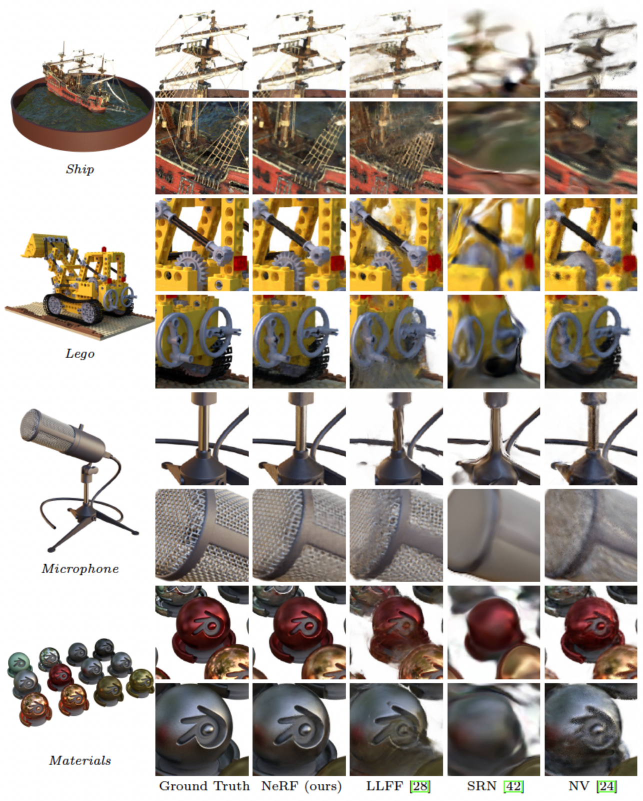
$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \left[ \left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right]$$

where R is the set of rays in each batch, and C(r), C^c(r), and C^f(r) are the ground truth, coarse volume predicted, and fine volume predicted RGB colors for ray r respectively. Note that even though the final rendering comes from C^f(r), we also minimize the loss of C^c(r) so that the weight distribution from the coarse network can be used to allocate samples in the fine network.

An overview of our neural radiance field scene representation and differentiable rendering procedure can be seen here:



We can observe from the following result on the LLFF dataset that the 3D model captures the intricate details very well, and can also render novel views very efficiently.



# iNeRF: Inverting Neural Radiance Fields for Pose Estimation

## Overview

Now that we understand how NeRF learns the 3D scene representation, we'll look at leveraging Inverse NeRFs for direct Pose Estimation instead of using 3d-2d correspondences. iNeRF takes 3 inputs: an observed image, an initial estimate of the pose (identity matrix), and a NeRF model representing a 3D scene or an object in the image. Different from NeRF, we start from an estimated initial camera pose and iteratively refine the pose (through gradient descent) following the similar rendering procedure as before to compute the loss with the observed image. iNeRF instead propogates the gradients to the estimated pose. After this is completed, iNeRF is able to recover the correct camera poses by aligning the rendered and observed images. In the following sub-sections, we'll cover the iNeRF formulation and how to leverage them for pose estimation followed by the current approach's limitations and scope.

## Leveraging Neural 3D representations

Neural 3D representations are essentially used to model the scene dynamics which can then later be leveraged for estimating the pose of the object. Niemeyer et al. [1] propose the approach of representing a surface as a neural 3D occupancy field and the texture as a neural 3D texture field in which the ray intersections are computed with numerical methods. NeRF in the Wild [2] extends NeRF to account for each image's individual appearance which in turn helps in high quality 3D reconstruction of landmarks using unconstrained photo collections.

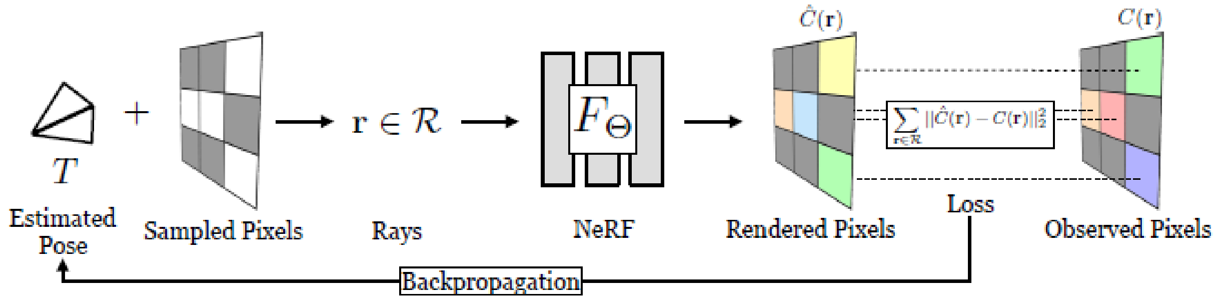
Besides the PnP based classical methods that were proposed in the earlier sections for pose estimation, NeRFs can also be leveraged for pose estimation of objects in context. On a tangential note, differentiable mesh renderers [3] have also been explored for pose estimation prior to the iNeRF approach. Prior work include Chen et al. [4] approached the category-level pose estimation problem using single-image reconstruction with a 3D voxel-based feature volume and then estimating pose iteratively by modelling it as an image alignment task. Since NeRF models scale reasonably well with dynamic and large scenes, the same formulation can be leveraged to perform localization or pose estimation. One potential limiting factor with this methodology of using NeRFs might be their memory run-time inference limitations.

## iNeRF Formulation

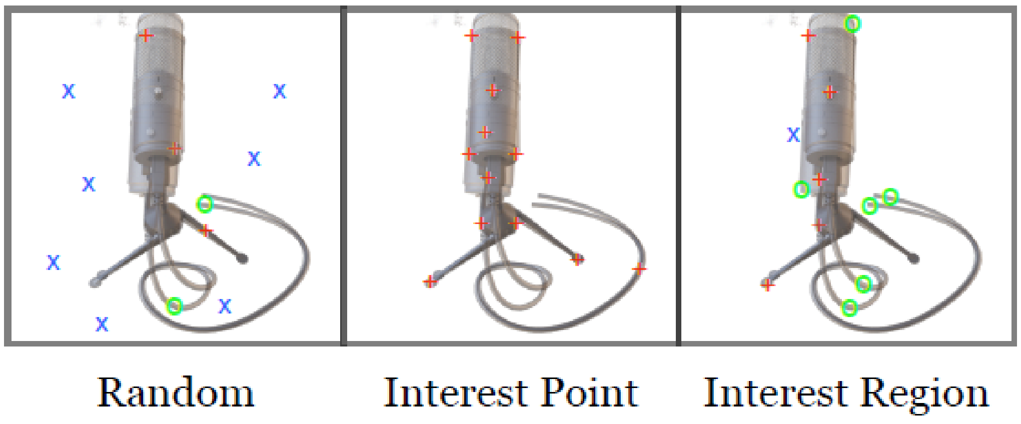
We begin by assuming that we have already modelled the NeRF representation of a scene and the camera intrinsics are given and we only need to predict/estimate the pose of the object. One another limiting assumption of this approach is assuming there is only one object in the scene for which we're estimating the object pose - this is due to the fact that NeRF based pose estimation approaches can't handle when multiple (possible overlapping occluding) objects are present in the same scene. Unlike NeRF, which optimizes using a set of given camera poses and image observations, we instead solve the inverse problem of recovering the camera pose T given the weights and the image I as input:

$$\hat{T} = \underset{T \in \mathrm{SE}(3)}{\operatorname{argmin}} \; \mathcal{L}(T \mid I, \Theta)$$

This equation can be solved by leveraging the NeRF's ability to estimate an arbitrary camera pose T and render the corresponding image observation. Once we have the reconstructed image for a corresponding pose, we can can then devise a photometric loss function L (as in NeRFs) and then backpropogate the updated weights ot pose T to minimize L (instead of to the NeRF model as in the typical NeRF implementation like in the previous section.

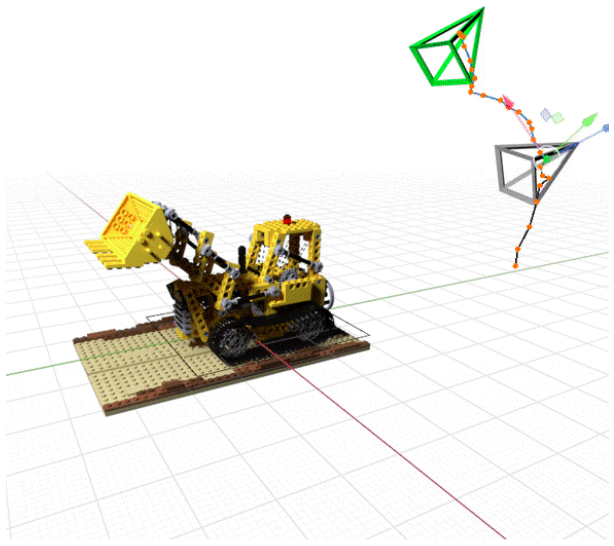


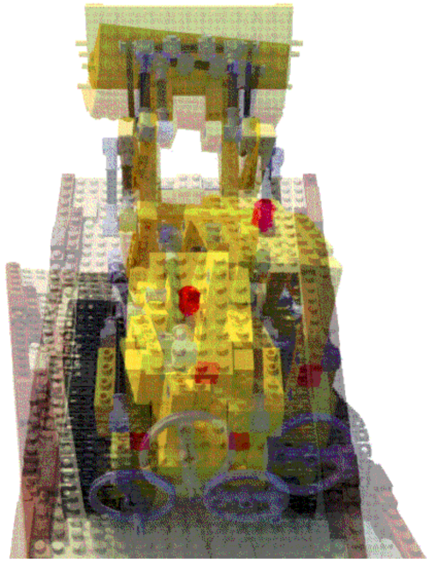
While sampling for rays during the aforementioned reconstruction approach, sampling of rays to reconstruct and backpropogate pose estimation values plays a critical role in the convergence of the algorithm. A naive approach of using random sampling, most of the sampled pixels provide no gradients for updating the pose of the object and hence do not contribute to the pose correction step. Instead, if we choose an interest region sampling as below, where pixels that fall on the object of interest are chosen, then we achieve faster convergence and higher accuracy.



## Results

The iNeRF model was tested on the LLFF dataset to evaluate the 6 DoF accuracy on pose estimation. It achieves competitive results when compared to feature-based methods without having the need for accessing CAD object mesh models or 2d-to-3D correspondences during training or testing time.





The above example shows using iNeRF in action, wherein we reduce the rotation error when starting from 45 degrees down to 0 degrees after running for 800 iterations.

## Limitations and future scope

Since, rendering the iNeRF (like NeRF) requires pixel-wise inference, multiple evaluations of the iNeRF model makes the runtime inference slow. Using guided sampling around object of interest is one way to improve on the baseline latency. Another drawback of this method is that it only supports pose estimation of object currently and cannot handle occluded objects in the scene, thus further limiting its usefulness and applicability. These are potential usecases to explore in the future for increasing the scope of this approach for Pose estimation.

## References

[1] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. CVPR, 2020.

[2] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. arXiv preprint arXiv:2008.02268, 2020.

[3] Andrea Palazzi, Luca Bergamini, Simone Calderara, and Rita Cucchiara. End-to-end 6-DOF object pose estimation through differentiable rasterization. ECCV, 2018.

[4] Xu Chen, Zijian Dong, Jie Song, Andreas Geiger, and Otmar Hilliges. Category level object pose estimation via neural analysis-by-synthesis. ECCV, 2020.Limitations and future scope

[5] Wang, Zi, Yang Shang, and Hongliang Zhang. "A Survey on Approaches of Monocular CAD Model-Based 3D Objects Pose Estimation and Tracking." 2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC). IEEE, 2018.

[6] Peng, Sida, et al. "Pvnet: Pixel-wise voting network for 6dof pose estimation." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

[7] Aubry, Mathieu, et al. "Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

[8] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. arXiv preprint arXiv:2003.08934, 2020

[9] "Perspective-n-Point." Wikipedia, Wikimedia Foundation, 19 Jul. 2022, en.wikipedia.org/wiki/Perspective-n-Point. Accessed 29 Oct. 2022.

[10] Kitani, Kris. Introduction to Computer Vision (CMU 16-720 B) . https://kriskitani.github.io/courses/16720B/index.html.