Analysis and Investigations of iNeRF and Directly Learning Pose Initializations

Levi Cai
cail@mit.edu

Daniel Yang
dxyang@mit.edu

Abstract

Recent work in neural radiance fields have allowed for the generation of novel viewpoints of objects using machine learning methods. These work by learning a fully connected network where inputs are light rays, originating from a given viewpoint, propagated through the volume of the scene. This has allowed for new methods of pose estimation in scenes, using analysis-by-synthesis approaches which leverage neural radiance fields to synthesize novel viewpoints in order to then estimate pose through optimization. However, some of these approaches are unable to be used in real-time, often requiring at least hundreds of iterations of stochastic gradient descent, and without an initial estimate of pose, can fail in many circumstances. In this work, we analyze details of one algorithm in particular, iNeRF, and how they contribute to performance, and investigate methods to improve runtime and error, specifically by providing initial pose estimates using ideas from more traditional approaches to pose estimation using neural networks. Our proposed architecture and results show promise in producing state of the art results in both error and runtime.

1. Introduction

Neural radiance fields (NeRFs) were developed in 2020 and led to impressive results on novel view synthesis of complex scenes [10]. NeRF works by learning a fully-connected network that maps a spatial location (x, y, z) and viewing direction (θ, ϕ) to a color, (r, g, b) and density α. Thus, by integrating over the network output from rays emitted from a specific camera pose, one can synthesize a novel view of a scene. The original NeRF paper has led to an explosion of works improving upon NeRF's shortfalls — its slow training and rendering speed and ability to represent only a single static (e.g., constant lighting and shape) scene — as summarized in a review [7].

Drawing heavy inspiration from one work, iNeRF [18], we focus on the inverse problem of pose estimation — given a rendered image and a pretrained NeRF for that scene, determining the pose of the object. Pose detection of objects has wide-ranging applications in mixed and augmented reality as well as robotic manipulation [8]. In this work, our contributions are as follows:

1. A custom implementation of iNeRF [18] in PyTorch
2. An analysis of the role of components in iNeRF — batch size of rays to render, ray sampling strategy — on the synthetic dataset used in the original NeRF paper [10]
3. Experimentation with more recent works that *learn* a pose estimation network that predicts pose from an image instead of a random initialization. We take inspiration from Direct-PoseNet [5] which learns a regressor neural network that can immediately generate an initial pose estimate from an image. We show that initializing iNeRF using the output of a pose regressor leads to high performance.

For the rest of the paper, we discuss related work in Section 2, provide our technical approach and implementation details in Section 3, demonstrate results in Section 4, and finally provide discussion and concluding remarks in Section 6.

Our code can be found at https://github.com/dxyang/6869_nerf.

2. Related Work

This work lies at the intersection of novel view synthesis and visual 6-DoF pose estimation.

2.1. Novel view synthesis

In novel view synthesis, the goal is typically to generate a never before-seen view from a small set of known views. This problem has been extensively studied in the graphics community or in situations where sophisticated sensors or light ray modelling is available. In these approaches, mesh-based representations of scenes are generally required [6, 2]. Alternatively, from a computer vision-based perspective, most traditional approaches have still required strong structural priors about the scene or object of interest, such as having full models of human faces [15, 1]. This can be extremely limiting.

More recently, neural network-based approaches have shown some promise in creating more generialized models of novel view synthesis. In particular generative adversarial networks (or GANs) and variational auto-encoders (VAEs), and typically a combination of the two approaches, have allowed for numerous examples of novel view synthesis [16, 11]. These systems allow users to train generic latent space representations of objects or scenes, and by then adjusting latent parameters, allows for the generation of novel views. However, these approaches can be difficult to control due to the inherent latent embeddings.

These works in graphics and machine learning have helped lead to the breakthrough development of NeRFs [10], which parameterize scenes based on light rays and volume densities. This representation is differentiable, and hence learnable using fully connected neural networks to learn mappings from this parameterization to fully rendered, novel RGB images.

2.2. Pose Estimation

Absolute pose estimation is a well-studied problem with broad coverage. In this work, we focus on the problem of 6-DoF pose estimation, or position and orientation, using visual information from RGB images. Key previous work may range from perspective-n-point (PnP) problems to visual simultaneous localization and mapping (SLAM) and visual odometry.

In traditional visual PnP problems, given a limited set of three dimensional points and image pixels, one must estimate the relative position of the camera. Iterative closest point (ICP) is one of the traditional algorithms used to solve this problem, and is closely related to our setting. However, ICP and similar algorithms, typically require an estimate of points, such as from point clouds or traditional 3D image features, and can be quite computationally expensive to solve [cite].

Visual SLAM and visual odometry algorithms, which often use ICP-style algorithms in conjunction, attempt to localize sequences of camera poses relative to one another according to some fixed global reference frame. Potentially of particular interest are active-SLAM approaches, in which the next camera pose can be selected in order to improve previous pose estimates. This literature is incredibly rich and well-studied, so we provide a survey reference [3].

Only recently however have neural network style approaches been utilized to estimate cameras poses relative to larger scenes [9] or smaller objects [14]. In these cases, neural networks are used to directly learn mappings from images to pose estimates. However, one issue with these approaches has been uncertain behavior when fairly novel view points are provided.

This is where pose estimation methods that utilize NeRF-style approaches can provide some additional benefit, in an *analysis-by-synthesis* approach [19]. With a differentiable renderer, we can generate novel viewpoints in an optimization pipeline and perform gradient descent on the renderer inputs. iNeRF [18] and Direct-PoseNet [5] are two such approaches, both utilize a pre-trained NeRF to render novel viewpoints. iNeRF performs a direct stochastic gradient optimization on the pose, while Direct-PoseNet attempts to learn a neural network, as in [9] to estimate poses.

Our work thus investigates the performance of iNeRF in this problem, and how we might be able to merge aspects of Direct-PoseNet for improvement.

3. Approach

3.1. iNeRF

The core of our work includes reimplementing iNeRF as described by [18]. Let us define our pretrained NeRF as a neural network parameterized by Θ and image I which was rendered from camera pose in world frame as ${}^{W}T_{C}$. Note that in the context of pose detection, we assume that the object or scene represented by the NeRF is centered in world frame such that estimating the camera pose is equivalent to estimating the object pose. Thus, our optimization problem is to find the camera pose, ${}^{W}\hat{T}_{C}$, that best explains the image as measured by a photometric loss function, $\mathcal{L}$, that is mean squared error in pixel space of the rendered pixels:

$$ {}^{W}\hat{T}_{C} = \underset{{}^{W}T_{C} \in \mathrm{SE(3)}}{\operatorname{argmin}} \mathcal{L}({}^{W}T_{C} | I, \Theta) \tag{1} $$

Whereas training NeRF only requires backpropagation through the weights of the neural network used for volume rendering, solving the inverse problem would require us to backpropagate further in the pipeline to the camera pose from which rays are sampled, inputted into the neural network, and then used to render an image from that camera pose. Following [18], we represent ${}^{W}\hat{T}_{C_i}$ at each optimization iteration, i, as a rigid body transform, parameterized by exponential coordinates, applied to an initial guess of the camera pose, ${}^{W}T_{C_0}$. [18] notes that they pre-multiply the optimized transform instead of post-multiplying it, so we can think of this as learning an offset to the world frame, ${}^{W_{\text{new}}}T_{W_{\text{old}}}$. With $\mathcal{S}$ being our screw axis with magnitude θ, we have:

$$ {}^{W}\hat{T}_{C_i} = {}^{W_{\text{new}}}\hat{T}_{W_{\text{old}}} {}^{W}T_{C_0} = e^{[\hat{\mathcal{S}}]\hat{\theta}}\, {}^{W}T_{C_0} \tag{2} $$

Thus, our optimization goal is to find the screw axis with magnitude that yields the offset that when applied to our initial guess gives the camera pose that most accurately recreates our given image with the given NeRF based on MSE.

$$ \hat{\mathcal{S}\theta} = \underset{\mathcal{S}\theta}{\operatorname{argmin}} \mathcal{L}(e^{[\mathcal{S}]\theta}\, {}^{W}T_{C_0} | I, \Theta) \tag{3} $$

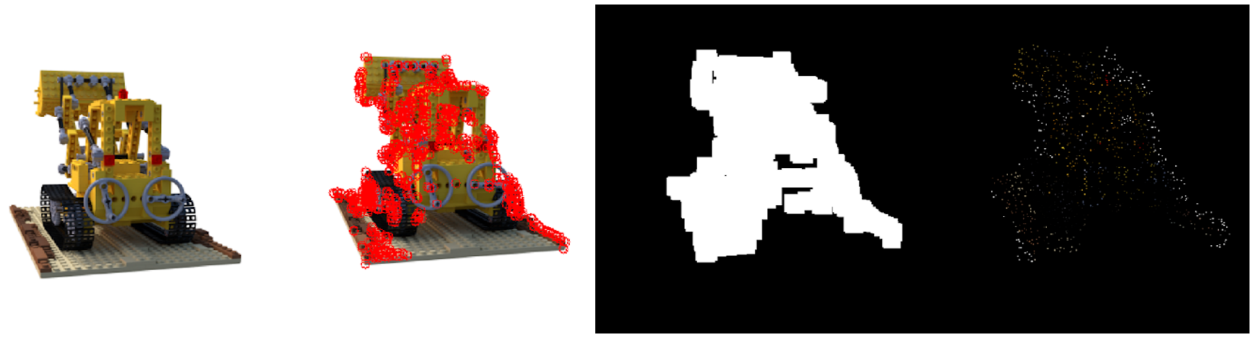


Figure 1: Ray sampling approach. From left to right: the image whose pose is to be estimated, ORB features as interest points, region sampling via a 3x dilation using 5x5 kernels, and finally the corresponding RGB values of the sampled 1024 rays, randomly selected and projected from the dilated mask. The last image is best viewed zoomed in on a computer due to the sparsity of 1024 out of 400×400 possible rays.

We can implement the operations within this function in PyTorch such that a loss can be backpropagated from a rendered image, to the SE(3) pose used to generate the rays rendering that image, to the underlying screw axis parameterization.

The authors in [18] note that merely using a random sampling of rays to render for Equation (3) is ineffective because many rays hit background pixels that do not provide any useful signal, especially for smaller batch sizes of rays. The authors instead suggest using interest point sampling, which we replicate with ORB feature detection [12], as well as interest region sampling which dilates the interest points and samples from there. Interest region sampling is motivated by how the interest points are detected on the given image, I, and not the rendered image for the current camera pose estimate, $\hat{I}$ which may differ in ORB features non trivially (e.g., an edge in I may not be at same pixel location in $\hat{I}$ which may lead to rendering a ray into background pixels). Figure 1 shows our interest region ray sampling and rendering approach. An image of the `lego` truck scene (left-most), from which we are detecting ORB features (as shown in the red circles) and dilating to create an interest region. We then render rays from our current estimate of the camera pose, as shown on the speckled image (right-most).

For our implementation, we draw attention to a few key differences between our implementation and the original work. These differences were gleamed from the paper as well as ad-hoc conversations with an author of [18] during office hours. We use pre-trained NeRF models from the `nerf-pytorch` repo which were trained at a lower resolution (400 x 400 on the synthetic dataset) and not for the same number of epochs as the models used in the original work. We also are never able to fit more than 1024 rays within GPU VRAM whereas the original work used much higher capacity GPUs that could. These are choices of convenience and scope as we wanted to focus on the algorithmic components of the work with relatively limited computation, but we do note the lower resolution and pre-trained models may explain performance differences as discussed later in Section 4.1.

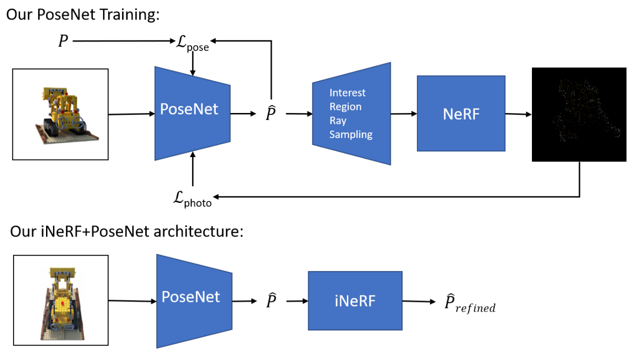


Figure 2: iNeRF+Direct-PoseNet architecture. On top, we show how we separately train PoseNet using the photometric and pose losses. At inference time and for evaluation, as shown on bottom, we show our combined process, which only needs an input image such that PoseNet outputs an estimated pose and then the optimization process of iNeRF further refines it.

3.2. Merging iNeRF with Direct-PoseNet

At the core of iNeRF is a stochastic gradient descent optimization problem that attempts to minimize a loss parameterized by the pose of an image in a scene. Stochastic gradient descent approaches, due to the non-convex solution space, are especially sensitive to poor initializations. The tests conducted in [18], as well as in our own vanilla iNeRF results, rely on initial conditions that are bounded relative to the ground truth estimate as specified by the user. In real conditions, this ground truth pose is not known, and thus the initial error in both translation and rotation can be extremely large, well outside the bounds provided in the test cases. Even in the best-case bounded scenarios, such as the synthetic datasets with a ray batchsize of 1024 and interest region sampling, iNeRF can fail to achieve decent predictions in 20% or more of cases [18].

Instead, we utilize a recent work, Direct-PoseNet [5], which attempts to learn a pose regression network that when given an input image, outputs the camera pose that was used to generate that image. We hypothesize that Direct-PoseNet could provide a better initialization that does not require any knowledge of the relative ballpark of the ground truth camera pose. Direct-PoseNet uses a MobileNet V2 backbone [13], pre-trained on ImageNet, with frozen batch norm layers, and replaces the last layer with a fully connected layer to regress the pose. It is then subsequently trained with two loss terms $\mathcal{L}_{\text{photo}}$ and $\mathcal{L}_{\text{pose}}$, a photometric loss similar to what iNeRF uses as well as a ground truth pose loss, a supervision signal available from the datasets, respectively. These losses are mixed with a hyperparameter λ for the overall loss, and the authors report success with $\lambda = 0.3$ as defined below:

$$\mathcal{L}_{\text{pose}} = ||\mathbf{P}_{gt} - \hat{\mathbf{P}}||_2 \tag{4}$$

$$\mathcal{L}_{\text{photo}} = ||\mathbf{I}_{gt} - \hat{\mathbf{I}}||_2 \tag{5}$$

$$\mathcal{L}_{\text{total}} = \lambda\mathcal{L}_{\text{photo}} + (1-\lambda)\mathcal{L}_{\text{pose}} \tag{6}$$

We note that $\mathbf{P}$ is parameterized as an $[\mathbf{R}|\mathbf{t}] \in \mathbb{R}^{3\times4}$ matrix. As in the original work, we also post-process the rotation $\mathbf{R}$ of the matrix by taking its SVD decomposition and using those components to create a well-conditioned rotation matrix. We also note that $\mathbf{I}$ in the original work is the full rendered image for a given pose. This is quite computationally expensive in terms of GPU VRAM to keep track of the gradients for backpropagation for a batch of $H \times W \times C$ rays. We instead utilize a ray sampling strategy similar to what is done in iNeRF. Our coupled iNeRF+Direct-PoseNet architecture, including our photometric loss modifications, are shown in Figure 2. Also of note is that the GPU VRAM usage of this photometric loss prevents utilizing batch sizes larger than 1 for large number (e.g. 1024) of rays. We also investigate the use of the exponential screw representation discussed in Section 3.1 which is $\mathbb{R}^6$ instead of the $\mathbb{R}^{3\times4}$ matrix representation for the output $\hat{\mathbf{P}}$.

4. Experimental Results

4.1. Vanilla iNeRF

Here we evaluate our iNeRF implementation according to similar metrics as present by [18] on the 8 synthetic objects test datasets used by the original NeRF work. For each object, we randomly select 5 test images and for each of the 5 test images we apply a random rigid body transformation

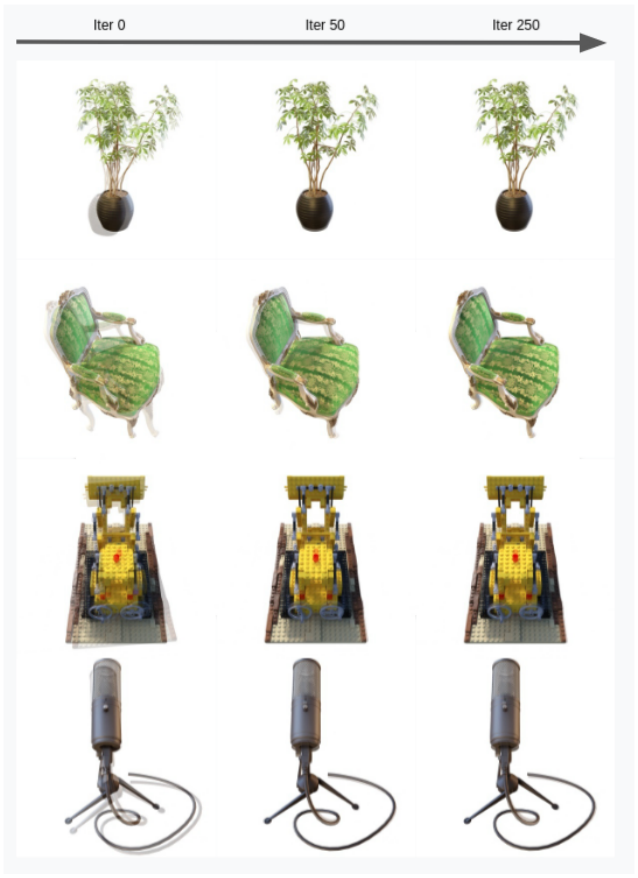
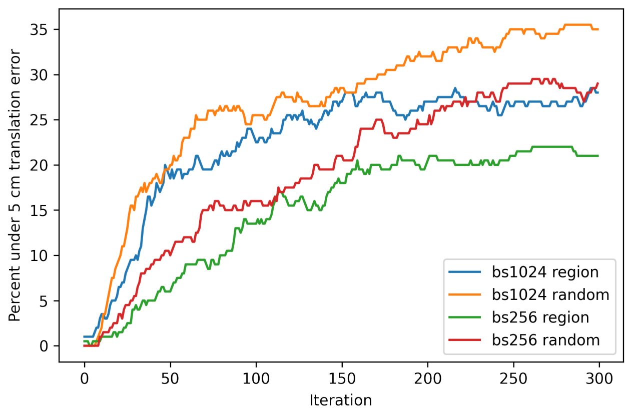
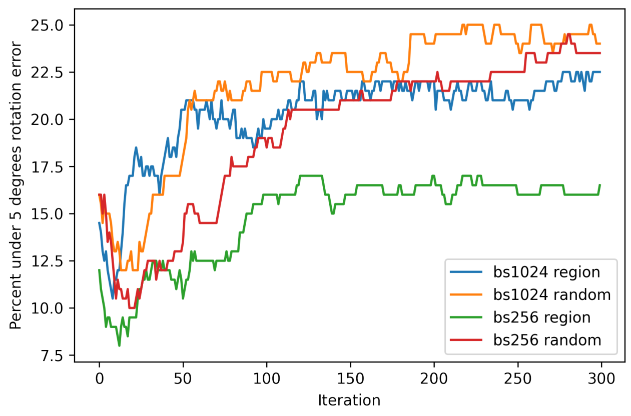


Figure 3: Here we show some qualitative successful results of our vanilla iNeRF implementation on the synthetic dataset (ficus, chair, `lego`, and mic, respectively). Initial estimates were sampled between 40 degree rotation errors and 0.2 m translation errors. We note that successful results tend to have smaller initial rotational and translational error offsets from the ground-truth, and thus also converge quite quickly, when compared to [18], which we describe earlier.

to the ground truth camera pose to generate our initialization. These random rigid body transformations consist of a rotation, defined by sampling a random axis by sampling a point from the unit sphere and then choosing a random amount of rotation between $[-40, 40]$ degrees around that axis, as well as a translation, consisting of a random amount of movement between $[-0.2, 0.2]$ meters along each axis. We use pretrained models provided by the original repo that are trained at half-resolution, 400×400, instead of full resolution, 800×800. We train models using the provided configs for synthetic objects where the pretrained model is not available. We initialize the screw parameters by sampling each parameter from a normal distribution centered at 0 with a standard deviation of 10^{-6}. Over 300 iterations we use an Adam optimizer with an initial learning rate of

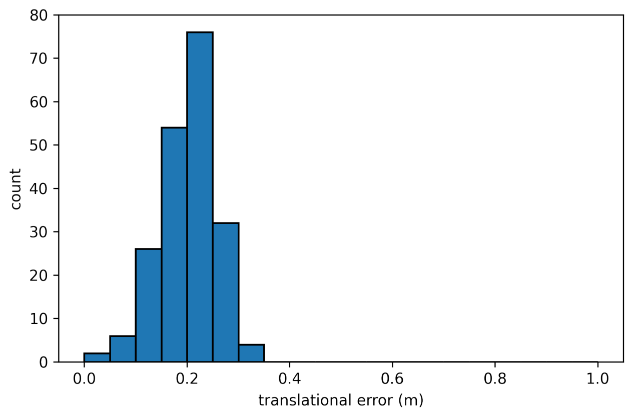


(a) % of poses at each iteration under 5 cm of translational error

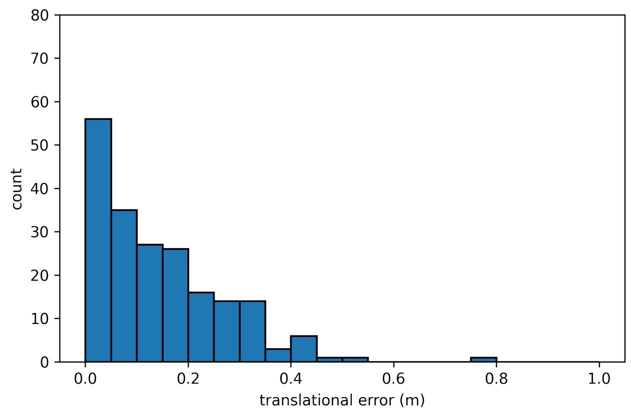


(b) % of poses at each iteration under 5 degrees of rotational error

Figure 4: Quantitative analysis over the 8 synthetic benchmark objects using only iNeRF. Our best configuration achieves under 5 cm of translational error 35% of the time and under 5 degrees of rotational error 23.5% of the time at the end of optimization. Percent of accurate poses increases over iteration, though these plot are notably monotonically increasing.



(a) Before optimization (iteration = 0)



(b) After optimization (iteration = 299)

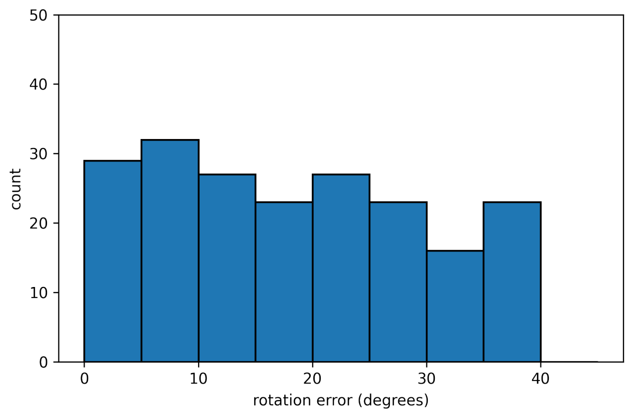
Figure 5: Distribution of initial and final translation error using only iNeRF for rendering 1024 rays and using region sampling (the blue line, `bs1024 region`, in Figure 4. Note the overall shift left from plot (a) to plot (b). Also note that the spread of errors increases, highlighting that optimization may bring some poses further away from the target pose.

0.01, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and a learning rate schedule that decays by 0.6 every 100 steps. We experiment with sampling 256 and 1024 rays as well as using random pixel sampling and region sampling where we dilation the region of orb keypoints.

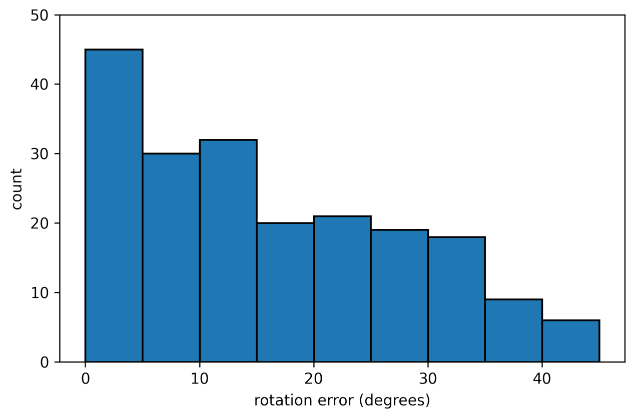
In Figure 4, we report the percentage of predicted poses with translational error less than 5 cm and with rotational error less than 5 degrees over each iteration of optimization. To evaluate its effectiveness relative to initial conditions, we report histograms of errors before and after the optimization process in terms of rotations in Figure 6 and translations in Figure 5 for rendering a batch of 1024 rays using region sampling.

4.2. iNeRF + Direct-PoseNet

Due to time constraints, we only evaluate our iNeRF + Direct-PoseNet architecture on the synthetic `lego` dataset. We first evaluate different methods of training Direct-PoseNet as presented in Section 3.2. We utilize the same standard training and testing splits for the synthetic dataset as used in NeRF and iNeRF to ensure Direct-PoseNet does not see an iNeRF evaluation image at train time. We compare between which losses are used, specifically, (1) only using $\mathcal{L}_{pose}$ as in Equation (4), and (2) with the mixed loss using our modified ray sampling procedure for the photometric loss as in Equation (6) and Section 3.2. We also

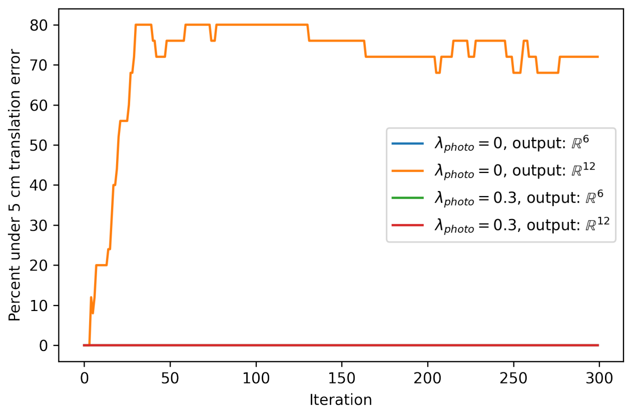


(a) Before optimization (iteration = 0)

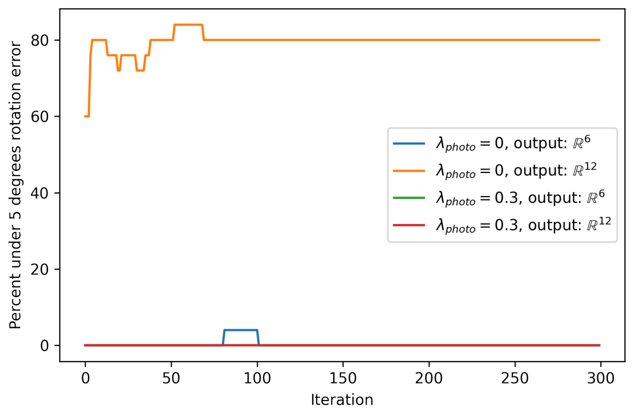


(b) After optimization (iteration = 299)

Figure 6: Distribution of initial and final rotation error using only iNeRF for rendering 1024 rays and using region sampling (the blue line, `bs1024 region`, in Figure 4. Note the overall shift left from plot (a) to plot (b) though less markedly than in Figure 5. This suggests that outputting optimizing for rotation may be a harder task than optimizing for translation which intuitively makes sense. Also note that the spread of errors increases, highlighting that optimization may bring some poses further away from the target pose.



(a) % of poses at each iteration under 5 cm of translational error



(b) % of poses at each iteration under 5 degrees of rotational error

Figure 7: Quantitative analysis of iNeRF + Direct-PoseNet on just the `lego` synthetic object. We initialize iNeRF with the output of the given image on our pretrained PoseNet, thus removing the requirement to initialize iNeRF nearby the target pose. We trained four versions of PoseNet, varying either the output dimensionality between $\mathbb{R}^6$ and $\mathbb{R}^{12}$ as well as changing the λ value mixing the photometric and pose losses as shown in Table 1. The version of PoseNet trained to predict the pose $\mathbb{R}^{12}$ without the photometric loss (e.g., just loss from the pose error) performs well. The rest fail to achieve the desired error rates. We can think of the iNeRF framework as refining the PoseNet output successfully near 80% of the time. We also note that performance peaks at roughly 50 iterations (30 for rotation and 52 for translation in this specific example and experiment).

compare between two different representations of SE(3), the first using the Direct-PoseNet method of using $\mathbb{R}^{3\times4}$ or flattened to $\mathbb{R}^{12}$, and the second using the screw exponential representation of the original iNeRF [18] in $\mathbb{R}^6$. The results of only the Direct-PoseNet training for the four combinations are shown in Table 1.

Finally, we couple each of the trained PoseNet architectures with our iNeRF implementation, and evaluate them on the synthetic dataset in the same quantitative manner as in Section 4.1. Instead of initializing the pose estimate of iNeRF with a randomly sampled offset, we use the output of each PoseNet method. The results are shown in Figure 7.

Dim	λ	t_{train}	r_{train}	t_{test}	r_{test}
12	0.3	3.94	76.24	3.93	89.86
6	0.3	3.94	98.29	3.93	103.09
12	0.0	2.58	42.49	0.20	3.32
6	0.0	63.97	998.37	2.54	41.76

Table 1: Training a PoseNet [5], according to Section 3.2 on just the `lego` synthetic dataset varying the loss mixing parameter λ as well as the output dimensionality. When λ is 0.3, the photo loss is weighted by 0.3 and the pose loss is weighted by 0.7. When λ is 0, we only use the pose loss. t is the translation error in meters while r is the rotation error in degrees. Using the photometric loss, we are unable to obtain low prediction error on both translation and rotation for either $\mathbb{R}^6$ or $\mathbb{R}^{12}$. However, for using just the pose loss we are able to obtain low error when predicting $\mathbb{R}^{12}$ but not $\mathbb{R}^6$, which has quite high error.

5. Discussion

5.1. Vanilla iNeRF compared with original iNeRF

Given the qualitative results in Figure 3 and the quantitative results of Figure 6 and Figure 5, it seems clear that our iNeRF implementation is generally performing correctly, as it is able to effectively reduce pose error estimates given the initial offsets, and qualitatively is able to estimate poses that align the corresponding images with the ground truth. However, when comparing with the performance of the original iNeRF, specifically our Figure 4 with Fig. 6 in [18], ours generally performs nearly 30% to 50% worse on the synthetic benchmarks. We also notice that region sampling is not as helpful as anticipated. We note some primary differences in implementation that could result in these issues. First, the feature sampling approach is not clearly outlined in the original iNeRF paper. Specifically, we chose to use ORB features and a dilation size using a 5x5 kernel that is then run 3 times, whereas it is not clear the exact procedure used in the original paper. More likely of importance however, is that we use an output image resolution of 400×400 pixels, whereas [18] uses a higher resolution of output images of 800×800, which we understood from discussions directly with Yen-Chen. Intuitively, this makes sense in that a higher resolution preserves more unique high dimensional features that would provide better signal in matching images purely based off pixel error. In addition, [18] is able to use larger batch sizes by running on hardware with more VRAM, whereas we were limited by access to only a Nvidia RTX 2080 Max-Q and Nvidia RTX 2080 Ti GPUs with up to 11GB VRAM. We also note that we use the pre-trained `lego` NeRF model and our own custom trained synthetic models, using [17]. Our trained models are available in the Github repository linked in the introduction.

5.2. Direct-PoseNet

In the training of Direct-PoseNet, as shown in Table 1, it is interesting to note that using the $\mathbb{R}^{12}$ pose representation along with using *only* the pose loss $\mathcal{L}_{pose}$ not only achieves the best results, but is also the only one to converge to reasonable error rates. The in-effectiveness of the photometric loss is contradictory to the results discussed in [5]. It is very important to note, however, that we do not render the entire image as in [5] which is likely the primary contributing factor because we only had access to limited hardware and the NeRF implementation of [17]. The authors also use a much larger and more complex dataset that could be better suited as well. Like before, our trained models are available in the Github repository.

Also of interest is that the $\mathbb{R}^{12}$ representation is significantly more effective for this style of network training than the $\mathbb{R}^6$ representation used for iNeRF. It is possible that using $\mathbb{R}^{12}$ better decouples the problem for usage as output of the fully connected layer compared with the $\mathbb{R}^6$ approach when used in direct optimization. More investigation should be done in order to better understand this relationship. That being said, the results using only $\mathcal{L}_{pose}$ are by themselves promising for their use as initialization values for iNeRF, as they bring the translation into a 0.2 meter range of error and rotation error into single-digit degrees, which are precisely within the bounds of the tests given by the original iNeRF paper.

5.3. iNeRF + Direct-PoseNet

Indeed, when coupling our best learned PoseNet network to provide an initialization for our own iNeRF implementation, we achieve state-of-the-art results in terms of both accuracy and time on the synthetic `lego` dataset as shown in Figure 7. This can be thought of as the PoseNet network first providing a fast initial pose estimate, and iNeRF then iteratively refining that estimate. The other variants of PoseNet, as expected from their training performance, cause initializations that are too far away from the true pose for iNeRF to be able to correct them reasonably. The latter is unsurprising given the average translation and rotation errors shown in Table 1.

The initial PoseNet estimate can be performed in real-time, since it is just an inference pass of a MobileNet V2 network. Then, as we see in Figure 7, iNeRF only needs around 50 iterations to achieve less than 5 cm of translation error and less than 5 degrees of rotational error, with 80% effectiveness, even with our sub-optimal iNeRF implementation. Extrapolating, the final error rate is potentially on-par with the results from the original iNeRF implementation on the synthetic dataset using its best settings, but requires 200 less iterations. It is also clear that iNeRF's refinement is useful on top of just a PoseNet implementation, since the vanilla PoseNet performance is shown at iteration 0 in Figure 7. This means that the iNeRF+Direct-PoseNet architecture is likely capable of very accurate pose estimation in many more instances without relying on random initializations that could result in failures, and is significantly faster than vanilla iNeRF.

6. Conclusion

In summary, we have implemented a custom version of iNeRF in PyTorch, provided some analysis and investigation into different aspects that could be used for improvements, and developed an initial implementation of a novel architecture that combines iNeRF with the methodologies of Direct-PoseNet that achieves state-of-the-art speed performance on the synthetic `lego` dataset and comparable error performance compared to the best-performing vanilla iNeRF implementation. With our architecture that can provide a better initialization to iNeRF, iNeRF can be much more useful in real-world conditions. Though it is still not quite real-time, as 50 iterations still requires on the order of a minute (hardware dependent), it is much faster than when using a random initialization that requires some ground truth knowledge. More evaluation needs to be done on both the remaining synthetic dataset and the real datasets, such as LLFF, to conclusively say that this architecture is better — particularly with higher resolution NeRF models — but our results are very promising. Further work could investigate working with the YCB dataset [4] of household objects and integrating a NeRF-based pose estimation system onto a robotic manipulator with clear use cases in daily life.

7. Individual Contributions

Daniel Yang provided the initial iNeRF with random ray sampling and feature sampling implementations, ran all of the quantitative experiments, and combined the iNeRF and Direct-PoseNet implementations. Levi Cai implemented the region sampling method, developed the initial Direct-PoseNet implementation, ran the qualitative experiments, and explored and implemented incorporating disparity/depth into iNeRF (though these experiments were not included in the final paper since they were unsuccessful). Both authors provided debugging and general coding support for all aspects and implementations, and both authors contributed to the writing and presentation equally.

8. Acknowledgements

We would like to thank the entire 6.869 staff for a great course, even during a pandemic, and would especially like to thank Yen-Chen Lin for his useful insights, and his `nerf-pytorch` implementation and iNeRF paper that originally motivated this work.

References

[1] V. Blanz and T. Vetter. Face recognition based on fitting a 3D morphable model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1063–1074, Sept. 2003. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. 1

[2] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, pages 425–432, Not Known, 2001. ACM Press. 1

[3] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. 32(6):1309–1332. Conference Name: IEEE Transactions on Robotics. 2

[4] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, 2017. 8

[5] Shuai Chen, Zirui Wang, and Victor Prisacariu. Direct-posenet: Absolute pose regression with photometric consistency, 2021. 1, 2, 4, 7

[6] Wenzheng Chen, Jun Gao, Huan Ling, Edward J. Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer. *arXiv:1908.01210 [cs]*, Nov. 2019. arXiv: 1908.01210. 1

[7] Frank Dellaert and Lin Yen-Chen. Neural volume rendering: Nerf and beyond, 2021. 1

[8] Xinke Deng, Yu Xiang, Arsalan Mousavian, Clemens Eppner, Timothy Bretl, and Dieter Fox. Self-supervised 6d object pose estimation for robot manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3665–3671. IEEE, 2020. 1

[9] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: A convolutional network for real-time 6-DOF camera relocalization. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2938–2946. IEEE. 2

[10] 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. In *The European Conference on Computer Vision (ECCV)*, 2020. 1, 2

[11] Pierluigi Zama Ramirez, Alessio Tonioni, and Federico Tombari. Unsupervised Novel View Synthesis from a Single Image. *arXiv:2102.03285 [cs]*, Feb. 2021. arXiv: 2102.03285. 2

[12] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011. 3

[13] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the*

IEEE conference on computer vision and pattern recognition, pages 4510–4520, 2018. 4

[14] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. 2

[15] Thomas Vetter and Spemannstrae Tbingen#. #germany. *Synthesis of Novel Views From a Single Face Image*. 1996. 1

[16] Xiaogang Xu, Yingcong Chen, and Jiaya Jia. View Independent Generative Adversarial Network for Novel View Synthesis. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7790–7799, Seoul, Korea (South), Oct. 2019. IEEE. 2

[17] Lin Yen-Chen. Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/, 2020. 7

[18] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. iNeRF: Inverting neural radiance fields for pose estimation. *https://arxiv.org/abs/2012.05877*, 2020. 1, 2, 3, 4, 6, 7

[19] Ilker Yildirim and Winrich A Freiwald. Efficient analysis-by-synthesis in vision: A computational framework, behavioral tests, and comparison with neural representations. page 6. 2