Neural Radiance Fields of Robots for Planning with Contact

Gadiel Sznaier Camps Keiko Nagami JunEn Low

Stanford University
Stanford, CA

Abstract

We propose a method of applying image segmentation and converting inertial pose information to relative pose in order to generate a Neural Radiance Field (NeRF) model of a target system using real world images. Our approach is motivated by planning problems that involve making contact with a target system, that could leverage the volumetric information of an object for onboard planning. Our codebase for this project can be found at `https://github.com/kayn329/drone_nerf`.

1. Introduction

For this project we will consider a problem in which a quadrotor with a forward facing monocular camera (ego drone) must identify and plan a path to make contact with a dynamic target. This problem presents challenges in both perception and planning but for the purpose of this course we will focus on the perception part of this problem. In this pipeline it will be important to develop a perception method that effectively informs our planning method.

To achieve this, we will use Neural Radiance Fields (NeRFs) [10] that are trained using real data obtained via a drone to gain an understanding of the density map of the target during flight time. A benefit of NeRFs is that for a given spatial location (x, y, and z) and viewing direction (ϕ, θ), the corresponding output is a view dependent radiance and density for that location. Additionally, rays can be generated based on the viewing direction, and then integrated with respect to the density to obtain a corresponding depth map of the environment from that point of view. If the planner receives this information rich output, it can then determine which parts of the target would result in contact that is most likely to lead to successful contact.

Our approach will be to collect image data using the camera mounted on the ego drone as shown in Figure 2, as well as pose information of the camera and the target robot obtained via an OptiTrack motion capture system shown in Figure 1. After this data collection stage, the data will then be loaded onto a separate GPU capable computer to train a NeRF that can output the density map of the target based on a given relative pose. One difficulty in using NeRFs as they are presented in prior works is that NeRFs will learn not only the density field of a single object in the field of view, but also the environment. Therefore, NeRFs are typically trained on static environments, where a global pose of the camera is used to obtain viewing positions and angles. In our problem, we are only interested in the relative pose between the camera and the target drone, and not of the environment that this target drone may be found in. To combat this difference, we propose to use a segmentation method to first segment the images from our dataset such that only the pixels in the images corresponding to the target robot are preserved. From here, we will train a NeRF using these segmented images along with *relative* pose information of the camera with respect to the target. This way, even if the target were to move with respect to the global frame, we would still be able to recover a density map of the target system in any environment.

Thus, our expected results will be to have a high quality NeRF that describes our target robot with a relative frame of reference, which can then be used to plan a capture trajectory. To evaluate our approach qualitatively, we observe the predicted RGB images at novel view directions. To evaluate our approach quantitatively, we will look at the PSNR and loss metrics during training. To tie our approach back to our initial goal of using the nerf for planning, we will then query the NeRF for dense regions of the target object, and produce a trajectory that passes through this high density region.

2. Related Work

NeRFs, first proposed by Mildenhall et. Al. [9], have only very recently appeared in the literature. In their method, they utilize a fully connected deep neural network to learn an implicit continuous volumetric scene function which takes as input a 5D spatial and view dependent coordinate obtained from an image with a known camera pose. The network then returns a corresponding volume density and radiance value for the given spatial coordinate. This approach is fully differentiable and can be used to learn a high quality geometric and visual representation of an environment at a cost of a slow training speed. Their ability to learn complex scenes and render new images from novel views has made them extremely attractive for vision based tasks and has garnered a wide range of interest in various fields of robotics, such as manipulation [14, 4], pose estimation [15], multi-object dynamics [2, 5, 18], and trajectory planning [1].

An important aspect of trajectory planning is knowledge of one's own pose. While it is possible to obtain pose information through GPS or from tracking software, often in real world scenarios this information is not accurate or available. However, cameras are cheap sensors that are common in most robot platforms, and therefore extremely useful for pose estimation. Common methods utilize CNNs or stereo camera setups to predict pose based on identified features [8, 3]. Yen-Chen et. al. [15] instead propose to use a trained NeRF and a given image to estimate the pose of the camera that took the image relative to the object in the scene. This allows them to avoid using expensive sensors or complicated setups while also opening the possibility of a fully differentiable unified trajectory planner based solely on NeRFs. However a limitation of this work is that its performance can be severely effected by occlusions and lighting effects. Moreover, it relies on the NeRF only containing a single object that remains static in the environment. In our problem, we instead assume that we have knowledge of our pose, and treat the problems of estimating poses and generating a trajectory as out of scope for this project.

Another related work that uses NeRFs in planning is shown in [1]. In their approach, they use a pretrained NeRF to represent the environment and then perform trajectory planning using only an on board RGB camera. The approach takes advantage of the differentiable nature of NeRFs and differential flatness which constrains the full pose of the robot to perform trajectory optimization to avoid high-density regions in the NeRF. While very impressive, this approach assumes that the start and end points necessary to calculate the trajectory are already provided. In contrast our approach focuses on providing a goal point based on the provided NeRF for the target and could potentially be integrated into this pipeline.

However, most works using NeRFs use pose information from a global reference frame, and do not account for dynamic objects in the scene. One approach that considers object dynamics is [2], which utilizes a combination of encoders, NeRFs, and graph neural networks to create a compositional object centric framework that can generalize across different scenes. One issue with this approach is that the examined scenes and objects are very basic and so it is unclear how well it would work for a more realistic scene. In addition, their focus was on multiple objects moving at once whereas in our problem we wish to focus on a single object. In comparison, the authors in [18] address this issue by training multiple networks for the elements in the scene that are static and dynamic. In our problem, our planner is concerned predominantly with the dynamic object, and less about the environment. For this reason, we plan to include a segmentation step in our pipeline to train the NeRF on relative pose information.

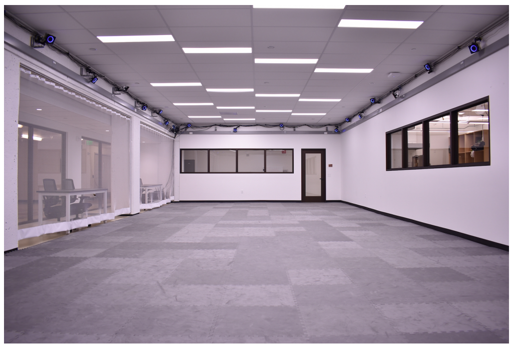
Figure 1. OptiTrack System in Boeing Flight and Autonomy Lab

3. Approach

There are three main components to our project, which involve data collection with robot hardware, image segmentation and computing the relative pose given global pose labels, and lastly training a NeRF on the segmented images and relative poses. Each of these components are detailed below.

3.1. Data Collection

Data was collected in the Boeing Flight and Autonomy Lab on Stanford campus. This flightroom is equipped with an OptiTrack motion capture system, with infrared cameras lining the room. These cameras enable us to obtain sub millimeter pose estimation on objects in the room. We leverage this setup to be able to obtain the ground truth pose of the target drone and camera during data collection. An image of the flightroom is shown in Figure 1.

To perform the data collection we then use a modified Carl drone, which we denote as the ego drone, which is shown in Figure 2. Mounted on the drone is an Intel RealSense Depth camera to the bottom of the drone which we use to obtain RGB images. While the RealSense is capable of extracting depth images as well, we only use monocular images for our dataset. To obtain accurate camera pose information, spherical motion capture markers are mounted onto the camera. These allow us to determine the camera's global pose using the OptiTrack system. To define

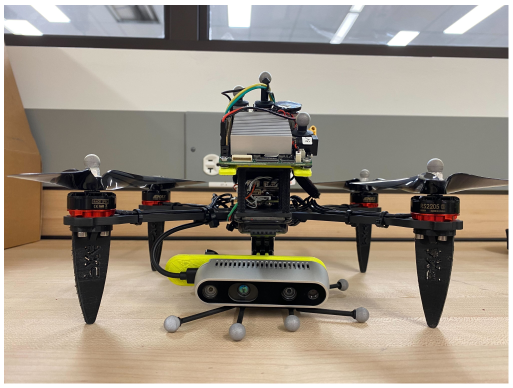
Figure 2. Front view of ego drone with RealSense camera

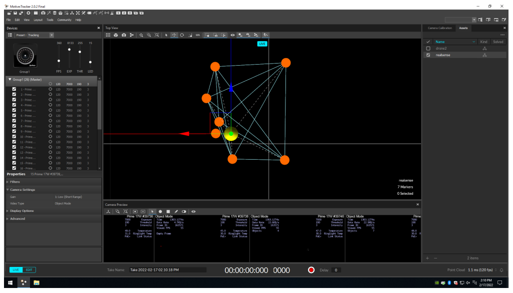
Figure 3. RealSense Camera definition in Motive OptiTrack Software

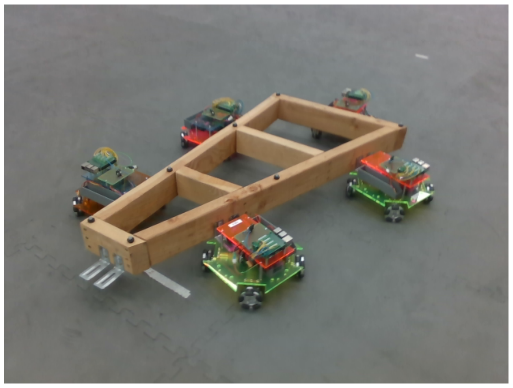
Figure 4. Target robot system

the camera's global pose, we must map the IR markers to a rigid body in the Motive program associated with OptiTrack. This process is shown in Figure 3. We then perform a similar process to obtain the pose information for the target robot. Lastly, we designate a target robot system shown in Figure 4. Our goal is to take images of this target system with the ego drone to obtain images and pose information of both robot systems.

Using this setup, we then collect two datasets. In the first, we capture the images with the ego drone by walking the ego drone around the target system. This allows us to obtain high quality images where the target system is fully in frame, and the camera is held steady. In a second dataset, we fly the ego drone manually using an RC to collect images and pose information. In this second dataset, the images contain higher levels of motion blur from the vibrations of the quadrotor, and the target system sometimes falls out of frame. Our goal is to see if our method would work with an ideal dataset with high quality data, and how our pipeline's results degrade as the data quality degrades, but becomes closer to what would be observed in a real-time application. All data was collected using the Robot Operating System (ROS) [12] to synchronize the data collected to the same clock.

3.2. Post Processing Data

All of our data is initially collected in a world or inertial frame, where the target system's pose and the camera pose are defined in the inertial frame, and the images of the target system contain information about the background. In order to remove the dependence of the final NeRF model on the background environment, we first post process our data by segmenting out the background of our images and converting our camera poses to poses relative to the target system frame.

3.2.1 Pose Transforms

In converting our dataset to be applicable to local reference frames, we must convert our pose labels from global poses to relative pose. Using the global pose labels of the robot system and the ego drone, we can compute the relative pose of the camera to the target as follows:

$$M_{c\to t} = M_{t\to w}^{-1} M_{c\to w}$$

where M is a transformation matrix composed of the rotation matrix R and translation vector T as follows:

$$M = \begin{bmatrix} R & T \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix},$$

and the subscripts denote an active transformation between frames. c denotes the camera frame, w denotes the world frame, and t denotes the target robot system's frame. Figure 8 shows the different reference frames and the transformations between each frame.

Additionally, we found that the NeRF codebase required that our input poses for training used the convention that the camera frame used an x downward, y right, and z backward convention. For consistency we also transformed our world frame to match that of the example dataset in the NeRF codebase. These transforms are shown in Figure 6.

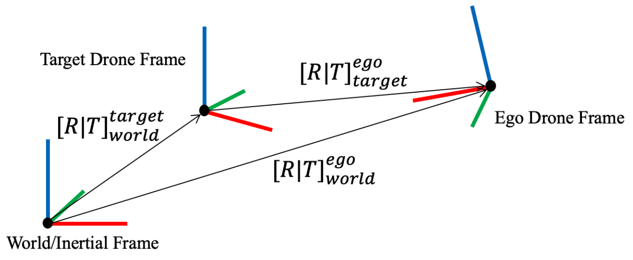

Figure 5. Reference frames for our collected data

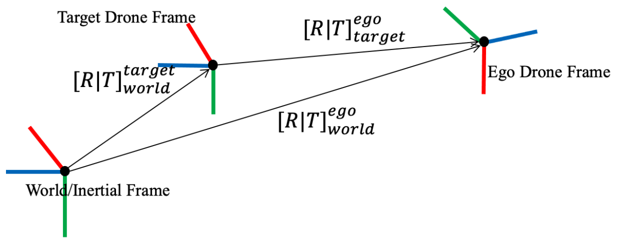

Figure 6. Reference frames for NeRF codebase

To convert our poses to use these conventions, we used the following transformations:

$$M_{nc\to nt} = M_{t\to nt} M_{c\to t} M_{nc\to c},$$

where the subscripts nc and nt represent the NeRF codebase camera frame and the NeRF codebase target frame respectively. From these relations we can obtain the relative position and orientation of the ego drone with respect to the target robot coordinate frame. These relative poses will then be used in conjunction with the segmented images to train the NeRF.

3.2.2 Image Segmentation

To train a NeRF using relative poses only, we need to first segment the training images so that only the target robot is shown in the image frame. The purpose of this step is to prevent the NeRF from learning background environmental features such as windows, chairs and other such objects which will keep the same position as the target robot moves in the scene. Furthermore, this also prevents the NeRF from assigning non-zero density to those features, and thus the only regions with non-zero density will be our desired target robot. The target robot is segmented out of the image by first converting the image from an RGB representation to HSV. Next, a gray color threshold is used to generate an image mask that covers the gray background. This mask is then inverted to obtain one that only covers the target robot. To ensure that the mask is mostly continuous and covers

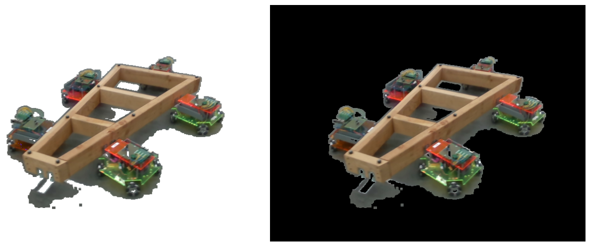

Figure 7. Segmented image of the Target drone system using either a white or black background

the features of interest, we perform an image dilation step which involves convolving the created mask with a identity kernel and applying the following operation:

$$d(x, y) = \max_{(x',y'):\ elem(x',y')\neq 0} src(x + x', y + y')$$

which replaces each pixel with the maximal value of the convolution.

Once the image mask has been created, we then have to select a background that would ensure that the empty regions in the image would be associated with the far camera plane and thus given an infinite depth. The purpose of this step is to prevent the NeRF from assigning arbitrary density in the empty regions. To achieve this, two approaches were tried. The first approach was to assign empty pixels a value of 255 to generate white background, which was more aligned with what the original NeRF pipeline expected. The second approach was to assign empty pixels with the value of zero across the color channels as it was reasoned that zero is easier for a network to learn.

3.3. Training and Evaluating a NeRF

The NeRF was trained using the original NeRF codebase written using TensorFlow. The input data was modified so that training would use the relative poses described in the previous section. The NeRF architecture used eight layers of channel width 256 for its fine and coarse networks. It was then trained using an Adam optimizer with a learning rate of 5×10^{-4} and a learning rate decay of 250. In addition, the NeRF architecture was set to use random ray batching and sampled each image using between 500 and 1000 rays that shot through the image plane at random uniformly. For the inertial and relative experiments using high quality images, the coarse network was given 32 samples along each ray and while 32 different positions were sampled along the ray for the fine network. However, to get the network to converge for the inertial experiment using the low quality images taken from the flying drone, 120 different positions were sampled for the coarse and fine networks instead. For all experiments, the network was trained under the assumption that the camera model was a pinhole model and the

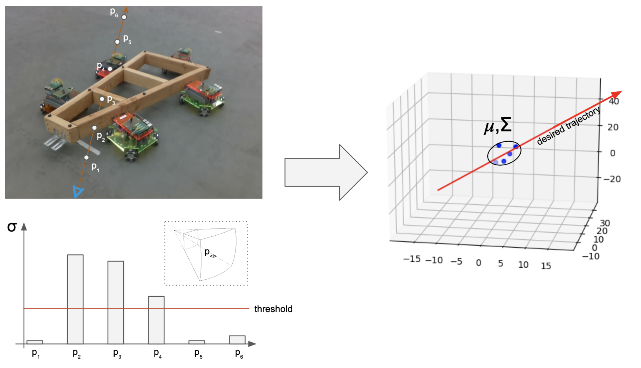

Figure 8. Diagram describing method of obtaining a potential capture trajectory from a NeRF based on the distribution of density.

near and far clipping planes were set according to outputs from the LLFF codebase. In all experiments, except for inertial low quality image experiment which was trained for 10 thousand iterations, the networks were then trained for 50 thousand iterations.

3.4. Capture Trajectory Planning

Lastly for capture trajectory planning, where we attempt to attach a drone mounted grasp mechanism onto the object, we generate a pair of weight points using the density information provided by the NeRF. The process is as follows: 1) We discretize the 3D space and populate it with points where the density is above a certain threshold. 2) We then take the variance of these points to assess the quality of our NeRF. A high variance implies a large amount of NeRF 'dust'; either the NeRF needs to be regenerated or the scene is not optimal for capture trajectory planning. 3) We generate a line of best fit in the 3D space using principal component analysis. 4) We generate a starting waypoint and ending waypoint using the line by stepping a user defined distance in both directions along the vector, starting from the centroid of the points. By travelling in a straight line between these two waypoints, we give the drone the best chance for its grasp mechanism to attach to the object.

4. Experiment

Using our method, there are three main variables that we look to observe the impact of. These are the use of relative pose instead of poses from an inertial frame, using segmented images instead of original images, and the effect of using a dataset with lower quality images that are more realistic to what a flying quadrotor with a camera would see onboard. To address these effects, we show the results of four different tests shown in Table 1.

Test Case	Relative vs. Inertial Pose	Segmented vs. Unsegmented	Image Quality
1	Inertial	Unsegmented	High
2	Inertial	Segmented	High
3	Relative	Segmented	High
4	Inertial	Unsegmented	Low

Table 1. Description of test cases for each NeRF that was trained.

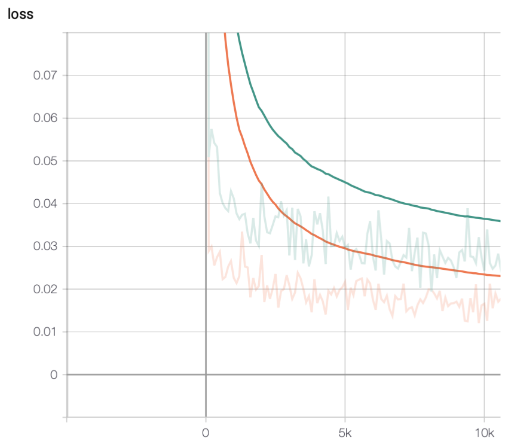

Figure 9. Loss comparison plot for segmented images with black (orange) and white (green) backgrounds respectively after 10K iterations.

4.1. Segmented vs. Unsegmented

Here, we will discuss the differences found between Tests 1 and 2, where the only difference is that Test 1 does not use image segmentation while Test 2 does. From Figures 11 and 12 we can see that the NeRF is able to produce reasonable images for novel views not seen during training for both cases. However, we can see in Figures 14 and 15 that the segmentation has a reduced PSNR value of 23.56 compared to the PSNR of Test 1 which is 29.16 and correspondingly has a higher loss value. This result mostly likely stems from the fact that the NeRF no longer has the background to provide additional guidance during training

We additionally observed that the effect of the values on the background of the segmented images also made an impact on the PSNR and loss metrics during training. Figure 10 shows the different PSNR progressions through training iterations for using a white background vs a black background in the segmentation process. We can see that using a black background for segmentation resulted in a higher PSNR value. We therefore use a black background for the tests in which segmentation is performed.

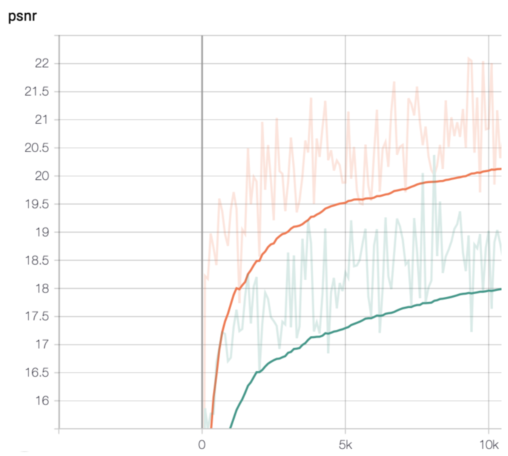

Figure 10. PSNR comparison plot for segmented images with black (orange) and white (green) backgrounds respectively after 10K iterations.

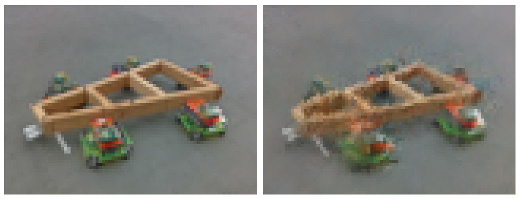

Figure 11. NeRF test output for Test 1. The left image is from the validation test set, unseen during training. The right image is the output from the nerf after training for the camera view of the image on the left.

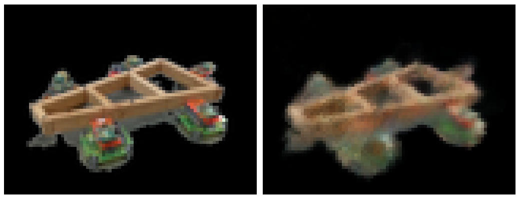

Figure 12. NeRF test output for Test 2. The left image is from the validation test set, unseen during training. The right image is the output from the nerf after training for the camera view of the image on the left.

4.2. World Frame vs. Relative Pose

Here, we will discuss the differences found between Tests 2 and 3, in which Test 3 uses relative poses whereas Test 2 uses inertial poses. We can see in Figure 13 that the NeRF using relative poses is able to generate reasonable images from novel view points. Additionally, we can see that using the relative pose seems to have little effect on the PSNR and Loss metrics as shown in Figures 14 and 15 as it has a final PSNR of 24.56 and loss of 6.929e-3 respectively in comparison to the PSNR 23.56 and loss 9.566 e-3 for Test 2, which are quite close to one another.

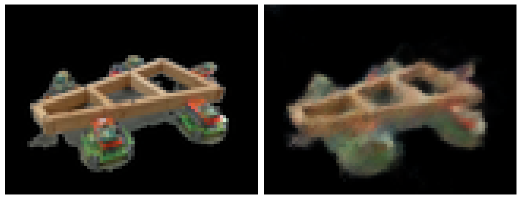

Figure 13. NeRF test output for Test 3. The left image is from the validation test set, unseen during training. The right image is the output from the nerf after training for the camera view of the image on the left.

4.3. Dataset Image Quality

As detailed in our approach, we collected two datasets; one in which the camera system was held to take higher quality images, and another in which the drone with a mounted camera was flown to capture the image data. In the latter test, there were more frames that contained motion blur, and frames in which the target system was not fully in frame. In Test case 4, we use the flown dataset to see how well a NeRF could train on a more realistic dataset like this. We found that there is a large performance degradation between training on high quality images versus low quality images. This makes sense as the NeRF tries to match the input images and as such cannot be better than them. Furthermore, the low quality images make it harder for the NeRF to identify the important fine details necessary for training a high quality NeRF.

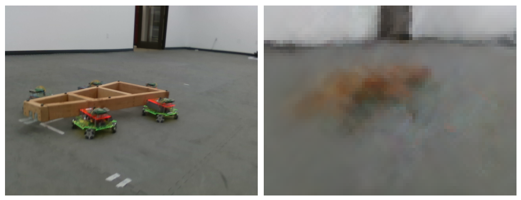

Figure 16. NeRF test output for Test 4. The left image is from the validation test set, unseen during training. The right image is the output from the nerf after training for the camera view of the image on the left.

4.4. Capture Trajectory Planning using Simulated Target

After training NeRFs for the various tests, we found that while the images and PSNRs were reasonable, the network did not learn well the underlying geometry of the problem. Even for our best NeRF that resulted from Test 1, the NeRF only learned a rough and indistinct representation that only vaguely matched the target robot shape as illustrated in Figure 17.

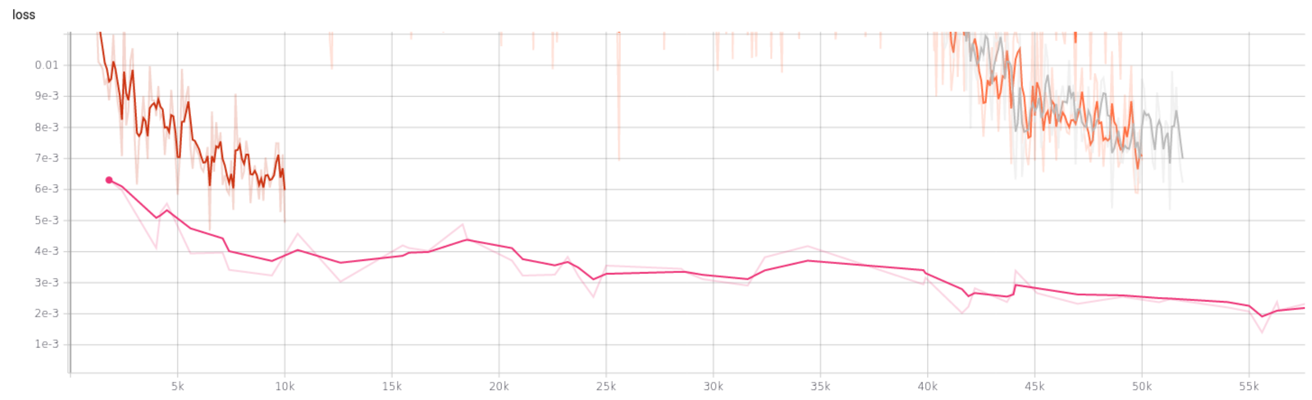

Figure 14. Comparison of Loss after 50k iterations for Tests 1 (pink), 2 (gray), and 3 (orange), and after 10k iterations for Test 4 (dark red)

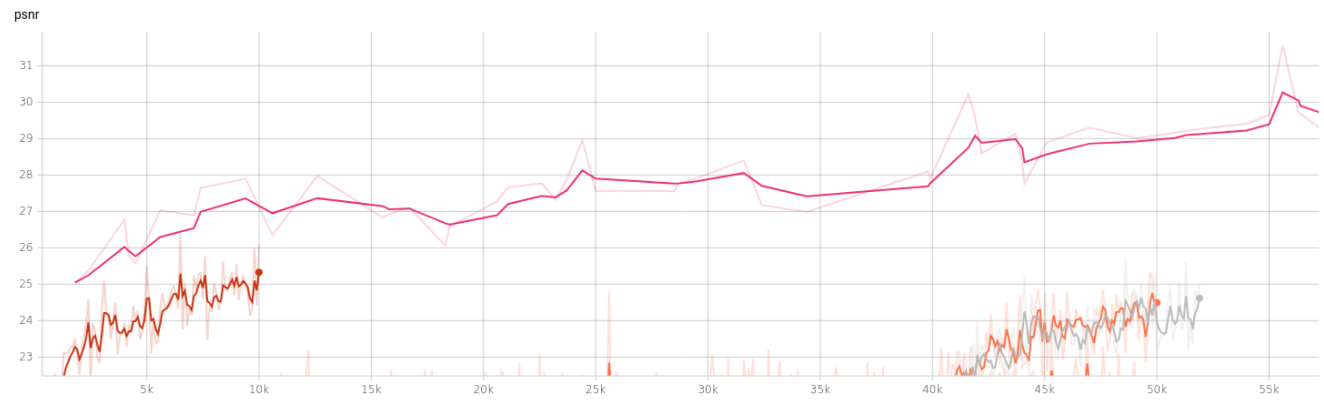

Figure 15. Comparison of PSNR after 50k iterations for Tests 1 (pink), 2 (gray), and 3 (orange), and after 10k iterations for Test 4 (dark red).

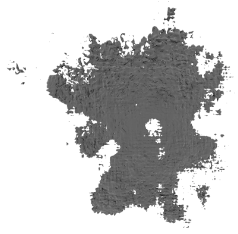

Figure 17. mesh generated from result of Test 1

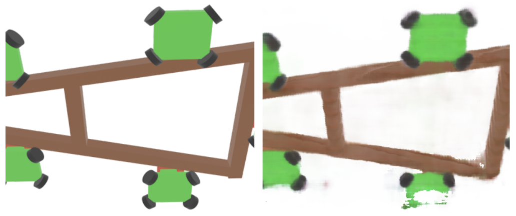

Figure 18. NeRF test output for Simulated target. The left image is from the test set. The right image is the output from the nerf after training for the camera view of the image on the left.

Thus, in order to properly test our capture trajectory algorithm, we trained a NeRF on a simulated target robot made using Blender modeling software, which resulted in a much crisper final NeRF as shown in Figure 18.

Using the NeRF trained on simulated data, we then applied our capture trajectory algorithm and generated the path shown in Figure 19.

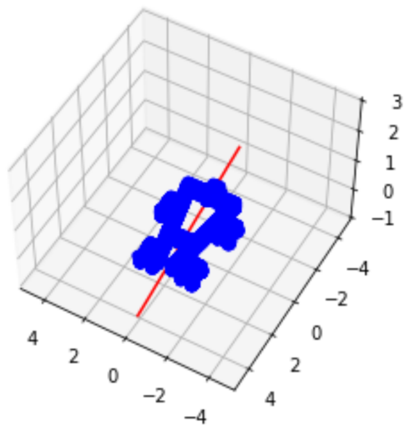

Figure 19. Generated capture trajectory (in red) using the simulated data. The 2.4m x 2.4m x 2.4m scene was discretized into a 256x256x256 grid and populated with a density threshold of 10. Given the size of the object, we chose a step size of 5m for the waypoints.

As seen in in Figure 19, given a high enough quality NeRF, we are able to obtain a reasonable trajectory that passes through the highest density region of the target, thus supporting our hypothesis that we can obtain a capture trajectory from a NeRF.

5. Conclusion

In our project, we proposed an approach to generate a NeRF using data collected with hardware to predict a capture trajectory for contact with the target. We trained four separate NeRFs in order to capture the effects of each stage in our pipeline; segmentation, use of relative pose information, and use of high quality data. From our experiments in comparing segmentation, we found that there was a quantitative difference between using white or black backgrounds for the segmented images when training a NeRF, with black backgrounds on average performing better than those with white backgrounds. We suspect that this was due to it being easier for networks to learn zero than it is to learn a large integer number. Additionally, we found that overall the PSNR and loss metrics performed better with the unsegmented images.

In evaluating the effect of using relative and inertial poses, we found that these tests performed very similarly. Since our experiments used data from a stationary target, we would expect to have similar performance between using relative and inertial poses. In future work it would be interesting to see how these differences would change when using a dynamic target. Lastly, we confirmed that using a dataset collected with the drone and camera system being flown produced images that the NeRF had more difficulty training on. Through this test, we learned how sensitive NeRFs are to the dataset that is being used to train them.

A major issue of using the original NeRF implementation is the slow training time. On average, it took more than a day to train a single NeRF. This made it difficult to prototype and debug our results. One possible solution would be to use more advanced, but more difficult to implement, NeRF solutions such as instant neural graphics primitives [11] or plenoctrees [16] which have been shown to have several orders faster training speed. This slow training speed was exacerbated by the fact that forward passes through a NeRF are extremely memory intensive. This had the added effect of forcing us to reduce the amount of rays and points we could sample per image, which in turn reduce the quality of the resulting NeRFs. Some possible ways to rectify it would be to use approaches such as PixelNeRF [17] which reduces the number of images necessary for training or using DIVeR which uses deterministic integration to improve the quality and speed for the NeRF pipeline [13]. A final issue with the original NeRF pipeline is that it is extremely sensitive to noise and pose misalignment. This resulted in a large amount of debugging being required to determine the correct pose transformations and network parameters needed for training the NeRFs. A possible approach would be to utilize the techniques proposed in either bundle adjustment NeRF (BARF) [6] or Deblur-NeRF [7] which focus on learning high quality NeRFs from misaligned poses and from blurry images respectively.

References

[1] M. Adamkiewicz, T. Chen, A. Caccavale, R. Gardner, P. Culbertson, J. Bohg, and M. Schwager. Vision-only robot navigation in a neural radiance world, 2022.

[2] D. Driess, Z. Huang, Y. Li, R. Tedrake, and M. Toussaint. Learning multi-object dynamics with compositional neural radiance fields. *arXiv preprint arXiv:2202.11855*, 2022.

[3] S. Hong and Y. Kim. Dynamic pose estimation using multiple rgb-d cameras. *Sensors*, 18(11):3865, 2018.

[4] J. Ichnowski, Y. Avigal, J. Kerr, and K. Goldberg. Dex-nerf: Using a neural radiance field to grasp transparent objects. *arXiv preprint arXiv:2110.14217*, 2021.

[5] Y. Li, S. Li, V. Sitzmann, P. Agrawal, and A. Torralba. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR, 2022.

[6] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey. Barf: Bundle-adjusting neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5741–5751, 2021.

[7] L. Ma, X. Li, J. Liao, Q. Zhang, X. Wang, J. Wang, and P. V. Sander. Deblur-nerf: Neural radiance fields from blurry images. *arXiv preprint arXiv:2111.14292*, 2021.

[8] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu. Relative camera pose estimation using convolutional neural networks.

In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 675–687. Springer, 2017.

[9] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[10] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[11] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022.

[12] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, may 2009.

[13] L. Wu, J. Y. Lee, A. Bhattad, Y. Wang, and D. Forsyth. Diver: Real-time and accurate neural radiance fields with deterministic integration for volume rendering. *arXiv preprint arXiv:2111.10427*, 2021.

[14] L. Yen-Chen, P. Florence, J. T. Barron, T.-Y. Lin, A. Rodriguez, and P. Isola. Nerf-supervision: Learning dense object descriptors from neural radiance fields. *arXiv preprint arXiv:2203.01913*, 2022.

[15] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin. inerf: Inverting neural radiance fields for pose estimation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1323–1330. IEEE, 2021.

[16] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021.

[17] A. Yu, V. Ye, M. Tancik, and A. Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021.

[18] W. Yuan, Z. Lv, T. Schmidt, and S. Lovegrove. Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13144–13152, 2021.