UCLouvain

École polytechnique de Louvain

Parallel Inversion of Neural Radiance Fields for Spacecraft Trajectory Estimation

Author: **Xavier Moonens**
Supervisor: **Christophe De Vleeschouwer**
Readers: **Antoine Legrand, Benoît Macq**
Academic year 2023–2024
Master [120] in Computer Science

Acknowledgments

I would like to express my gratitude to my supervisor, Professor Christophe De Vleeschouwer, for his guidance throughout the duration of my master's thesis. His expertise and insightful feedback have been instrumental in shaping this work.

I am especially grateful to Antoine Legrand, whose mentorship and dedication have greatly contributed to my learning and growth over the past year. His patience, constructive criticism, and constant availability made this challenging journey both manageable and enriching.

I would also like to extend my thanks to Benoît Macq for graciously agreeing to be a part of my jury. Your time and effort in reviewing my work are greatly appreciated.

Finally, I want to thank all those who supported me during this thesis—friends and family—your encouragement and belief in me have been a constant source of strength.

Contents

Abstract

This thesis focuses on spacecraft trajectory estimation. Knowing the 6D pose (position and orientation) of an unknown uncooperative space object relative to an on-board camera facilitates space missions such as debris removal, re-supplying and servicing. Making it possible for some of these missions to be executed without human intervention. Our method leverages recent researches around neural radiance fields[31] and their inversion [60], [22] to perform 6D pose estimation of an uncooperative space object based on sequences of RGB images.

The state of the art method for NeRF inversion is the parallel inversion technique[22]. It focuses on estimating the pose of an object by minimizing the error between pixels rendered from a NeRF, for different starting poses, and the corresponding pixels of an observed RGB image iteratively. The main issue when considering this approach for space missions is the computational power required. This kind of computational power is not available in a spacecraft. In this work, we show that we can improve the efficiency of these kind of methods when dealing with moving objects. As most space missions imply a spacecraft navigating on orbit around a target object we can leverage this observation and adapt the parallel inversion of NeRFs[22] to modify the target image during execution.

We demonstrate on synthetic terrestrial images that the method converges fast to a subsequent image of the sequence when it is able to predict a pose relatively close to the target for the previous image of the trajectory.

The particular conditions characterizing the space environment such as occlusion, symmetrical and almost texture less objects make it hard for most pose estimation techniques to achieve accurate results on spacecraft images. In this work we show how our method performs in these particular conditions both on synthetic and real images. We show that the method performance drops significantly when dealing with synthetic spacecraft images compared to the ideal case and even more when dealing with real images.

Finally, we study the impact of the starting pose candidates on the performance of the method.

Chapter 1

Introduction

In computer vision, pose (position and orientation) estimation is becoming increasingly important with various applications such as robotics [1] and autonomous driving. With the recent development of AI techniques such as neural networks, progress in this domain has significantly accelerated, allowing new approaches to be explored as mentioned in the Related works chapter (Chapter 2).

Our focus in this work is on uncooperative spacecraft pose estimation tasks such as servicing, debris removal missions and resupplying of on orbit satellites. By 'uncooperative' we mean that the target object is not communicating with the spacecraft trying to estimate its pose. The goal of estimating the pose of uncooperative objects is to enable these tasks to be performed autonomously, without human intervention. There are different techniques that enable to estimate the pose of an object in space but the first neural network-based approach was described in a publication [49] by S. Sharma and S. D'Amico. The main goal of the authors was to realise on-board pose estimation of a known uncooperative spacecraft using monocular cameras.

Knowing the target object's shape in advance facilitates the process of estimating its pose as a virtual representation of the object can be generated on Earth *e.g.* via a NeRF[31]. For debris removal missions though, such a model is not always available. This problem is not directly addressed in this thesis but we assume that images of the target are collected during the spacecraft's journey toward the target. These images are then sent to Earth where a 3D representation is trained based on them and subsequently sent back to the satellite.

For now, we define the NeRF[31] tool mentioned earlier in this chapter as a black box. It is used to represent scenes. It takes a camera pose as input and returns the images corresponding to the object's view from that pose as output. Providing it with a sufficient number of images to train on, results in a good 3D representation of the object. A framework called nerfstudio[53] exists for train-

ing and developing new NeRFs[31]. To ensure our method remains resilient to future improvements in NeRFs, we made it compatible with nerfstudio models.

In this thesis we study how we can invert this process to estimate the pose of objects for which we trained a NeRF[31] representation. This can be achieved by generating the image associated with a given pose and comparing it to the image observed by our on-board camera. By iteratively updating the pose based on this difference, referred to as the loss, we can generate an image similar to the one observed by the camera. This process is known as INeRF[60]. Since the generated image is produced based on a specific pose, we know the pose of the object relative to the camera.

While INeRF[60] is used to find the pose of an object it requires the initial pose to be already close to the target. This presents a challenge when we want to estimate the pose of an object but cannot easily obtain an initial pose close to the target without prior knowledge of its location. To address this issue we implement a version of parallel INeRF[22]. Parallel INeRF involves randomly sampling multiple starting poses in the 3D space around the target object and initiating an INeRF[60] process from each of them simultaneously. We then keep the poses whose associated images are the closest to the observed image. For each of these poses we generate new ones in their neighbourhood. Finally we repeat the entire process again with starting poses corresponding to both the best ones found in the previous step and the ones derived from them. This is called Monte Carlo resampling. By doing this we augment the likelihood that at least some process will converge to a pose for which the associated image is close to the target.

Training multiple INeRFs[60] in parallel requires significant computational power. This is an issue in a space environment for several reasons:

- Power constraints: spacecraft have limited power. This power often comes from batteries or solar panels. Thus, processor that consume less power are preferred even if that means lower computational power.
- Heat management: High-performance processor typically generate a lot of heat. Thus, as the cooling options are limited, processors that generate less heat are preferred.

Due to the researches on NeRFs[31] and NeRF inversion[60], [22] being recent (the paper on parallel inversion of NeRF[22] was published in march 2023), the literature on the subject is not well developed. To our knowledge two papers treats the subject of using NeRFs for 6D spacecraft pose estimation from RGB images in a space environment. The first one (June 2024) by A. Legrand et *al*[19] addresses the issue of not having a representation of the target object in advance and develops a spacecraft pose estimation network. The second (August 2023) by A. M. Heintz and M. Peck develops a method

for estimating the state of a spacecraft relative to an asteroid[10]. This lack of researches on the topic motivates the development of our method.

Our contribution is threefold: first, the development of a method for pose estimation over a sequence of images following a trajectory; second, the integration of nerfstudio models into our method, ensuring compatibility with future improved versions of NeRF due to the modular nature of the NeRF-Studio framework; and third, a study of the performance of this method, as well as Inerf-type methods more generally, on spacecraft images. Our approach leverages the fact that our camera is on-orbit around the target object. We do not compare the image(s) generated by our NeRF[31] to the same observed one each time we perform a Monte Carlo resampling. Instead, we change the target image to the one observed after the time needed to perform the computation of the first step. This process is detailed in the Method chapter (See chapter 3). The implementations details can be found in the Software implementation chapter (Chapter 4)

In the Results chapter (Chapter 5) we explore what this new framework is capable of on multiple datasets. Particularly on spacecraft synthetic and real images. We also study the influence of several model parameters, including: the number of pixels we want to generate through the NeRF[31] for each image and the influence of the starting pose candidates.

Some possible future improvements of our work are discussed in the Future works chapter (Chapter 6) and includes the development of new NeRFs[31] capable of dealing with the particular conditions posed by the space environment [19], the development of new sampling strategies better suited for space imagery and more.

Chapter 2

Related works

In this chapter, we explore three fundamental topics that serves as pivotal points in our exploration of the subject. They provide a comprehensive foundation for understanding the context of this thesis. These topics are Pose estimation, Neural Radiance Fields and Inversion of NeRFs for pose estimation.

The Pose Estimation section (See Section 2.1) outlines the fundamental approaches to pose estimation and their application to spacecraft pose estimation. The Neural Radiance Fields section (See Section 2.2) details how neural radiance fields function as representations of 3D scenes. Finally, the Inversion of NeRFs for Pose Estimation section (See Section 2.3) connects the previous sections by explaining how neural radiance fields can be leveraged for pose estimation.

2.1 Pose estimation

Pose estimation and more specifically 6D pose estimation is a prominent task in computer vision that involves determining the position and orientation of an "object" relative to a camera.

This thesis focuses on methods taking a single RGB image as input. Camera systems that incorporate depth perception or sensors like LIDAR are disregarded due to their weight, computational demands, and power consumption, all of which pose significant challenges in the space environment (See chapter 1). This section briefly describes all the families of methods used for pose estimation based on a single RGB image.

In the past, most methods were based on geometrical approaches [56] (sometimes relying on manual annotations [47]) or 2D representations of the object from different viewpoints[30]. Those 2 approaches are respectively called Feature-based and Template-based. Recently, with the advancements of AI

and neural networks most researches were directed toward Learning-based methods or leveraging existing approaches through neural networks. [30]. Those Learning-based methods can themselves be divided into 3 sub-categories: Bounding box prediction and Perspective-n-Point (PnP) algorithm-based methods, Classification-based methods, and Regression-based methods.

In the following sections we go into more details about the different methods we mentioned here. Then, we discuss the main learning-based method for spacecraft pose estimation.

2.1.1 Feature-based methods

Feature-based methods rely on two stages.[30] Firstly, local features such as keypoints or edges are extracted from an image. Those 2D features are then matched to 3D features from the corresponding 3D model, *e.g.* the CAD model[3]. CAD (Computer-Aided Design) models are digital representations of objects generated by computer software. They are used as a base for many pose estimation algorithms. Finally, the 6D pose is recovered by solving the underlying geometric problem, *e.g.* through a PnP[45] solver algorithm. The PnP (Perspective-n-Point) algorithm is a computer vision algorithm used to estimate the pose (position and orientation) of a calibrated camera relative to a known 3D scene, given a set of 3D points and their corresponding 2D image projections. This optional step is called the pose refinement step. Though optional, it is often necessary as the coarse 6D position resulting from the 2D-3D match is generally not precise enough[8]. One or multiple CNNs can be added at different stages to increase the performance. Though mostly used for feature extraction [43], [44], they were also used to predict 2D-3D correspondence in [44], object detection and keypoints estimation[64], or even the pose refinement step in [17].

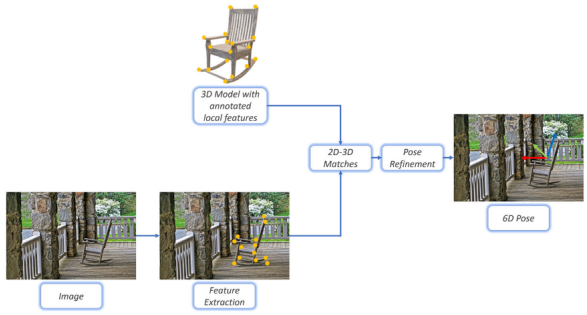


Figure 2.1: Pipeline overview of the features-based methods. Features are extracted from the input image. Then there is a matching between them and the ones annotated in the 3D model (could be a CAD model). Finally matches are given to a pose refinement step responsible of estimating the 6D pose.[30]

The main advantage of these methods is their speed[54] and robustness[23]. Though objects should have a very recognizable non-symmetrical shape[23] with well chosen keypoints[61] for the method to be efficient (which is an issue with space objects). Moreover the additional refinement step can be time consuming and is necessary in most cases to obtain precise pose estimation[8].

2.1.2 Template-based methods

In the template-based methods, a template database consisting in a set of synthetic renderings representing the object under different positions and orientations, is built offline from a 3D model. Then, a second stage is executed online and compares the input image with the set of renderings using a sliding window algorithm to find the best match and the corresponding 6D pose[30].

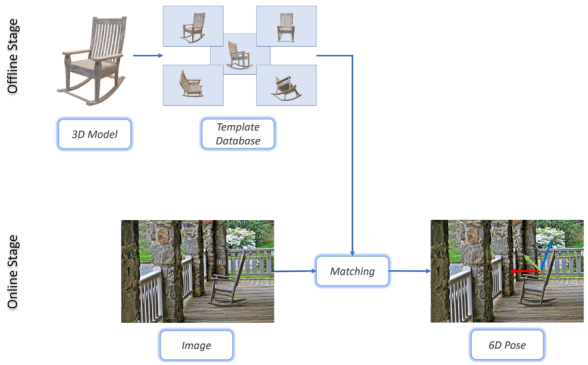


Figure 2.2: Pipeline overview of the template-based methods. First a template database is built from a 3D model. Then a matching step is performed between this database and the input image to find the 6D pose.[30]

These family of methods work even in case of object without distinguishable textures[23] and can achieve high accuracy given enough renderings creation in the offline stage. Though these methods are sensitive to lighting conditions and occlusion[64]. Moreover the execution time is proportional to the number of renderings[25] making it hard to find a good tradeoff between the accuracy and the computing costs[34]. Neural networks can be used to accelerate some parts of the pipeline, notably to predict the 6D position[5] or to learn representations from a CAD[52].

2.1.3 Learning-based methods

Learning-based methods rely on a training step which consist in training a model by giving training data to a/some CNN(s)[64],[8]. In this section, we firstly discuss the two categories of learning based methods; Single-stage meth-

ods (section 2.1.3.1) and Two-stages methods (section 2.1.3.2).

Then, we provide an overview of the main families of learning-based methods, namely: Bounding box prediction and Perspective-n-Point (PnP) algorithm-based (section 2.1.3.3), Classification-based (section 2.1.3.4) and Regression-based (section 2.1.3.5).

2.1.3.1 Single-stage methods

Single-stage methods consist of CNN(s) directly predicting the 6D pose.

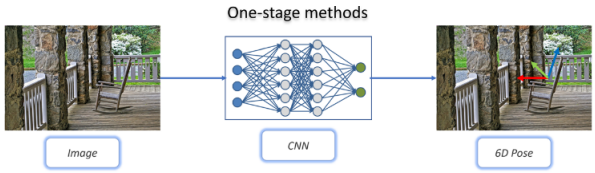


Figure 2.3: Pipeline overview of the single-stage methods. CNN(s) are charged of directly predicting the 6D pose based on an input image.

Those kind of methods are generally less accurate than two-stages ones [30]. Furthermore, their precision is affected by occlusion in the presence of multiple objects in the image[23].

2.1.3.2 Two-stages methods

Two-stages methods rely on CNN(s) to predict a bounding box around the object. Then, a pose refinement step is fed the output of the CNN to predict the 6D pose through PnP algorithm[30].

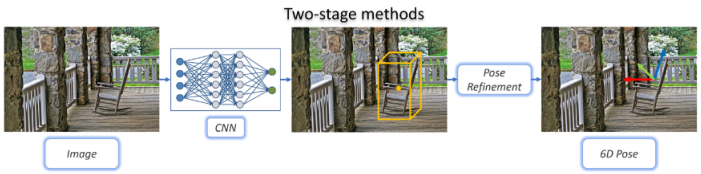


Figure 2.4: Pipeline overview of the two-stages methods. First CNN(s) is/are used to generate an output (bounding box) based on the input image. Then, a pose refinement step is fed the CNN output to refine the pose parameters and generate a 6D pose through PnP.[65]

Two-stages methods are generally more accurate than single-stage ones. This is particularly true for images with small or multiple objects[30].

2.1.3.3 Bounding box prediction and Perspective-n-Point (PnP) algorithm-based

The methods of this category are two-stages. Firstly, they estimate the 2D projections of the target object's corresponding 3D keypoints in the image. Then, they compute the actual 6D pose using the PnP algorithm[65].

These methods require a lot of manual annotations[64] which is be a problem when dealing with space images (See section 2.1.4). However some of these methods are able to work in real time [63],[24],[14].

2.1.3.4 Classification-based

Methods in this category are single-stage, performing 6D pose estimation by discretizing the pose space[30] and using CNNs to solve a classification problem. This is accomplished by computing a probability distribution that is then associated with a 3D model[64].

2.1.3.5 Regression-based

The methods of this category are single-stage. They train a neural network with images and their corresponding 6D poses, allowing the model to directly predict the 6D pose for a new image once trained[65]. Often, a preprocessing step detects objects in the images to simplify the pose estimation process.[41].

2.1.3.6 Pros and cons of learning based methods

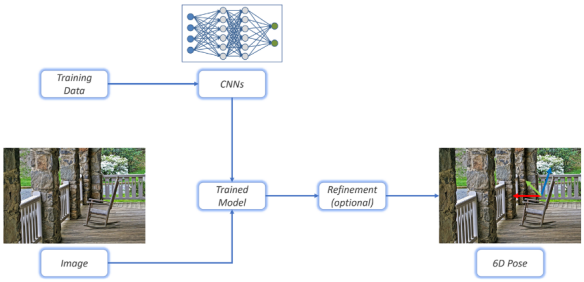


Figure 2.5: Pipeline overview of the learning-based methods. First a CNN or multiple CNNs are trained based on training data forming a trained model. Then the input image is fed to the CNN(s) to either directly predict the 6D pose, or predict a value which is then used in an optional refinement step to predict the 6D pose. [30]

These methods offer the advantage of being able to achieve highly accurate results ([46],[62]). However, they have significant drawbacks, including a time-consuming training process[13]. Some approaches address the lack of training

data by generating synthetic images or using data augmentation[21], but this can often result in overfitting and less generalizable models.

2.1.4 Spacecraft pose Estimation

After reviewing the fundamental approaches to pose estimation, our next step is to analyze the learning-based methods used for non-cooperative spacecraft pose estimation and how these methods respond to the specific challenges posed by the space environment. Indeed feature-based approaches suffer from harsh lighting conditions, illumination and high contrast present in space imagery and are not able to predict accurate 6D poses for space images on their own[49]. S. Sharma and S. D'Amico implemented a neural network based approach for pose estimation combining the strength of feature-based and learning-based approaches [49]. However, deep learning (DL) methods heavily rely on annotated images, which are difficult to obtain in space due to the high cost of missions and the limited access to space. The performance of deep learning methods also drops significantly when the training and testing image domains differ (*e.g.* training on synthetic and testing on real images). This is called "domain gap". [42]

The following sections describe DL-based spacecraft pose estimation algorithms. Recent challenges in the domain [16] have demonstrated that this is the preferred approach for addressing the problem of uncooperative spacecraft pose estimation. Most learning-based approaches in this field fall into one of two categories: hybrid modular algorithms ([11],[39],[4]) or direct end-to-end algorithms([50],[51],[38]).

2.1.4.1 Hybrid modular algorithms

These approaches have two steps in common: keypoint prediction and pose computation. Sometimes they also implement a preliminary step called spacecraft localisation.

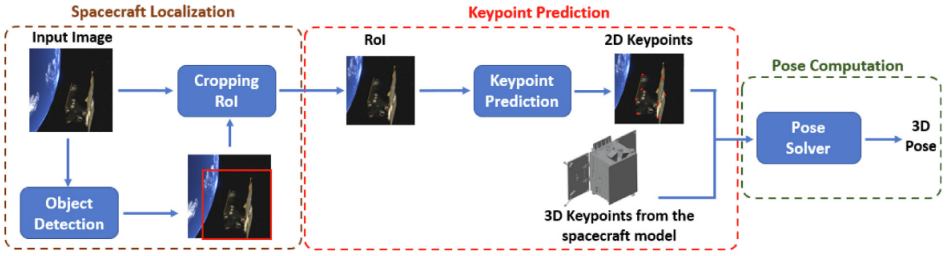


Figure 2.6: Pipeline overview of the hybrid modular approaches[42].

Spacecraft localisation consist in finding the spacecraft in the input image and cropping the region it is located in. This can be done either with one[39]

or two stage object detectors[11],[4]. Two stage detectors can achieve more accurate detection but are more computationally expensive. One stage detectors on the other hand, have less parameters and are thus more suitable for real-time detection.

The keypoint detection step uses a DL model to predict the 2D projections of a set of 3D keypoints defined by a CAD model. If no CAD model is available, multiview triangulation can be used instead[4],[12].

Finally, the pose computation step determines the pose using the 2D keypoints from the previous step *e.g.* through an IterativePnP solver[45], an EPnP solver[20] or a MLP[18].

2.1.4.2 Direct end-to-end algorithms

These approaches rely on a single neural network to predict the spacecraft pose directly from the input image. One of these approaches was described in a work by Sharma and D'Amico [49]. This SPN (spacecraft pose network) did not require the formulation of hand-engineered features and was able to find the spacecraft's pose only using a single greyscale image fed to a CNN.

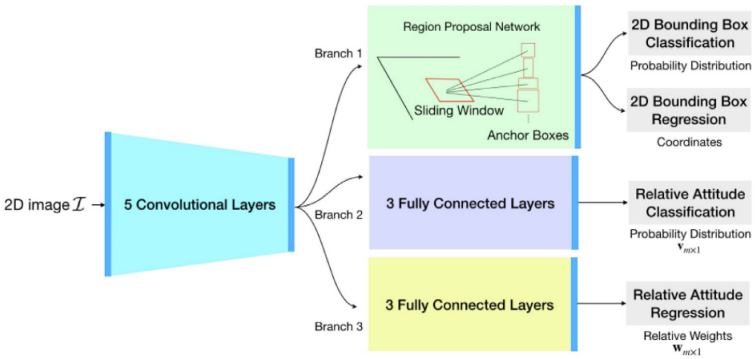


Figure 2.7: Pipeline overview of the SPN method[42] by S. Sharma and S. D'Amico[49]

The model represented in figure 2.7 uses three separate branches. The first branch detects the spacecraft in the input image and generates a bounding box around it. The output of the convolutional layers corresponding to the detected 2D bounding box is fed into Branches 2 and 3 to estimate the relative attitude $\hat{q}_{BC}$. Finally, the 2-D bounding box and $\hat{q}_{BC}$ are used as input to the Gauss–Newton algorithm[32] to estimate the relative position $\hat{t}_{BC}$.[49]

Following the development of SPN, T. H. Park and S. D'Amico introduced SPNv2[38] to address the domain gap problem (See section 2.1.4).

2.1.4.3 Limitations of the current algorithms

Despite the progress in the field of uncooperative spacecraft pose estimation several limitations persist.

- Deployability: The recent models are computationally expensive and are designed to work with advanced hardware, which is not available in space missions. This aspect is discussed in section 3.1. We show in Chapter 5 that our method partially solves this issue.
- Explainability: The black box nature of DL model is a problem when critical decisions are to be taken, as humans have to be able to understand why some choices are made.
- Robustness to illumination conditions: Lighting conditions in space are really harsh with high contrasts, shadows and reflections.
- Dataset availability: Real images of space are scarce, and it has been shown that synthetic, lab, and real images belong to different domains, leading to significant performance drops when a model is trained on one type of data and tested on another [9], [59].

2.2 Neural Radiance Fields

In parallel to the development of pose estimation techniques a new tool has emerged in the field of scene representation and novel view synthesis. This tool is called Neural radiance fields (NERFs[31]) and is capable of synthesizing novel views of complex scenes by using a sparse set of images and their corresponding camera coordinates [31]. More concretely the algorithm of the base version of Nerf uses a multi-layer-perceptron (MLP) whose input is a single continuous 5D coordinate (x, y, z,θ, Φ) which can be divided in a spatial location (x,y,z) and a viewing direction (θ, Φ) and whose output is an emitted radiance (amount of light that is emitted from a point) at that specific location depending on the view and a volume density.

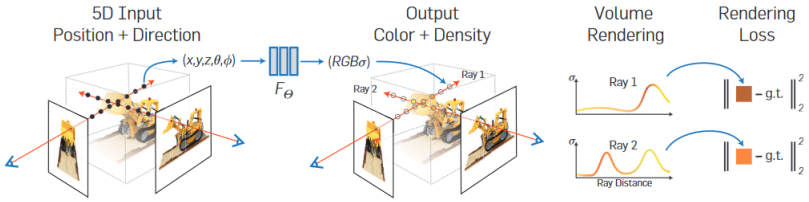


Figure 2.8: Overview of nerf rendering procedure by [31]. This figure illustrates the different steps of generating a Nerf model. Namely generating rays, feeding them to the MLP to get RGB colors and density σ corresponding to the rays and finaly using volume rendering to create an image based on these values

Figure 2.8 shows how this process is done. To predict the RGB value of a particular pixel, Nerf generates a ray r originating from the center point of the camera and passing through a pixel in the image plane. After that two types of samples are generated along the ray. We then simultaneously optimise 2 networks. A "coarse" one is fed N_c locations using stratified sampling. The output resulting from equations (1) and (2)[31] is then used to produced a more informed sampling biased toward relevant parts of the volume.

$$t_i \sim U\left[t_n + \frac{i-1}{N}, (t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)\right] \tag{1}$$

$$C(r) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))c_i, \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right) \tag{2}$$

To perform this we rewrite formula (2) as a weighted sum of all sampled colors c_i along the ray[31]:

$$C_c(r) = \sum_{i=1}^{N_c} w_i c_i, \quad \text{where } w_i = T_i(1 - \exp(-\sigma_i\delta_i)) \tag{3}$$

We then normalize the weights: $\hat{w}_i = \frac{w_i}{\sum_{j=1}^{N_c} w_j}$ and sample N_f locations from this distribution[31]. Finally we evaluate our "fine" network on the union of N_c and N_f to get the final output.

The rendering function being differentiable we are able to optimise the scene representation by minimizing the residual between the generated and the observed image.

Following this work several papers discussing improvements over the Nerf technique or use cases emerged. The following sections described the most interesting ones.

2.2.1 Instant NGP

One of the works following the NeRF[31] paper is "Instant neural graphics primitives" (Instant NGP) [33]. This implementation of NeRF[31]decreases the training time by several orders of magnitude. This is achieved by utilizing a new, more versatile input encoding that allows the MLP network to be smaller without compromising reconstruction quality, thereby reducing the number of memory access operations.

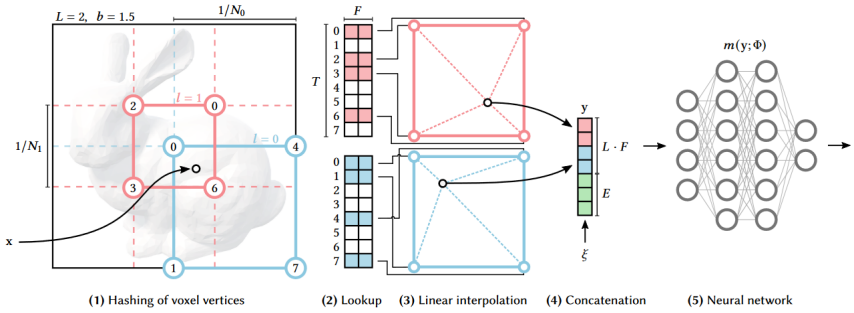


Figure 2.9: Instant NGP input encoding [33]. This figure illustrates the transformation of a point into a NN input. This is done by hashing voxel vertices, looking up the feature vectors in the hashtables and linearly interpolate them. Finally we concatenate the interpolated values and this forms the NN input.

More precisely their neural network has trainable encoding parameters θ which are arranged into L levels of T feature vectors of dimension F. The precise encoding is illustrated in figure 2.9. For a given point x NGP finds surrounding voxels (In 3D computer graphics, a voxel represents a value on a regular grid in three-dimensional space. [58]) at L resolution levels, assigning an index to each of them corresponding to their hashed integer coordinates. It then looks up the feature vectors present in the hash tables and linearly interpolate them with respect to the position of x in each particular voxel. The

result of the different levels are then concatenated to form the input of the neural network. Lastly the encoding is trained by backpropagating the loss gradients through each step and accumulating them in the looked-up feature vector.

2.2.2 NeRF in the wild

Another interesting work that followed NeRF[31] is NeRF in the wild (NeRF-W) [29]. Classical NeRF models have proven effective in static environments, but real-world scenes often exhibit varying parameters over time, such as changing lighting conditions or the presence of occluders. This is illustrated in Figure 2.10.

Figure 2.10: This figure illustrates the same image with different photometric variations (right) and occluders (left) [29]

When observing internet photos of notable places, two distinct phenomena can be observed:

- Photometric variation: Factors such as the time of day or atmospheric conditions can alter both our perception and a camera's perception of a scene. In space environments, light reflection directly affects illumination and, consequently, the emitted radiance of objects within the scene.
- Transient objects: Real-world images often contain moving objects or occluders. This is demonstrated in the Nerf-w model, which uses tourist photos where elements like people or moving vehicles are commonly present in the images.

NeRF-W aims at addressing those two challenges. To mitigate the impact of photometric variation on the model, NeRF-W incorporates appearance em-

beddings, making the radiance image dependant rather than independent of it. This approach allows the model to adjust the emitted radiance for each image while ensuring that the 3D geometry predicted by the NeRF's MLP remains static and consistent across all images.[29]

As illustrated in figure 2.11, NeRF-W also separates the scene in a static (a) and a transient (b) part. Those two parts are then composite together (c). To mitigate the impact of occluders the training minimizes the difference between the composite (c) and the real image (d) weighted by uncertainty (e) [29].

Figure 2.11: Overview of NeRF-W decomposition of a scene[29]

2.2.3 Nerfstudio

Nerfstudio [53] is a Python framework designed for NeRF [31] development. Despite the growing popularity of NeRFs, a lack of standardized development tools exist. Nerfstudio addresses this challenge by offering a modular framework that supports various input data pipelines, divide NeRFs into modular core components, integrates with a real-time web viewer (See annex: A.1), and provides diverse export options. The primary aims of this framework are to simplify the creation of custom NeRF methods, to enable easy processing of real-world data, and to improve user interaction with 3D reconstructions. Figure 2.12 illustrates nerfstudio's pipeline.

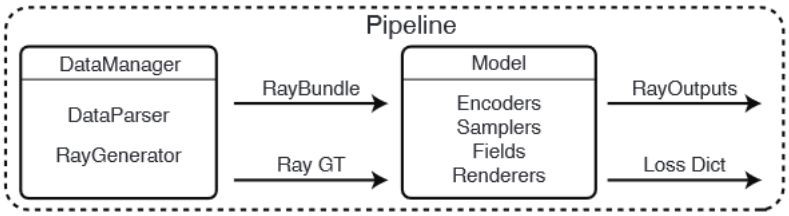


Figure 2.12: Pipeline overview of nerfstudio by [53]. A DataManager is responsible for processing input images into bundles of rays. These Raybundle objects get passed to the model network and produce the corresponding ray outputs (RGB color, depth, normals,...).

In addition to the framework itself Nerfstudio was released with a new Nerf model implementation named Nerfacto which is the NeRF[31] model we

chose for our method. This model integrates ideas from multiple research papers. Primarily inspired by Mip-NeRF 360 [2] (proposal network sampler, scene contraction), Nerfacto also incorporates performance optimizations from other models, such as hash encoding and fused MLP from Instant-NGP[33], normals from Ref-NeRF[57], and appearance embedding from NeRF-W[29]. You can see an illustration of the nerfacto model pipeline in this annex: A.2.

2.3 Nerf Inversion for pose estimation

The following sections describe the preliminary works that aimed at performing 6D pose estimation through Neural Radiance Fields.

2.3.1 Inerf

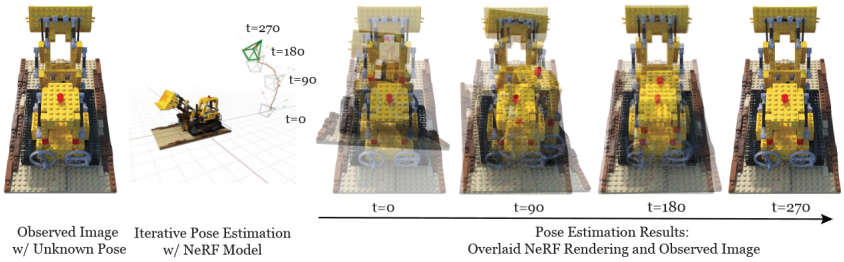


Figure 2.13: This figure illustrates the Inerf process. We start from a pose relatively close to the target and we can see that the model converges to the real image by iteratively updating the pose. [60]

Figure 2.13 illustrates Inerf[60]. The way Inerf works is by "Inverting" a Nerf to find the pose of a camera based on a single observed RGB image. Firstly, a NeRF[31] model is trained on a target object using multiple training images. The Trained Nerf is then taken as fixed and we feed it an initial pose candidate.The main issue with Inerf is that we need this initial pose candidate not to be too far from the target one as otherwise it might not converge or converge to a local minimum. By giving our Nerf an input pose and sampling pixels of the image in one of the way presented in the Inerf paper; namely random, interest point, interest region. (See section 3.2) we get their corresponding RGB color and we can compute a loss between these generated pixels and the corresponding pixels in the observed image. By backpropagating this loss over our pipeline we can iteratively modify the input pose until convergence to obtain the target camera pose. This is illustrated in the following figure:

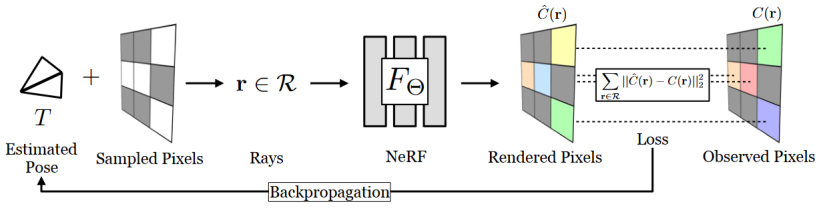


Figure 2.14: Pipeline overview of an INeRF[60] iteration. We start by sampling pixels of the image and generating the corresponding rays. After that, those rays are passed as input to the network that outputs an RGB color for each of them. Finally, those colors are compared to the color of the pixels in the target image and a loss is computed. This loss is backpropagated through the network and the pose is updated. [60]

2.3.2 Loc-Nerf

Loc-Nerf[28] is a real-time, vision-based approach for robot localization that leverages NeRF[31] and Monte Carlo localization[7] to perform 6D pose estimation. Unlike INeRF[60], it does not require a precise initial pose estimate. Firstly, a NeRF map model M of the scene is pre-trained. Then, at each time step t, the model is given an observed RGBD image I_t and a motion estimate O_t from time $t-1$ to t. Using this information, the model predicts the 6D pose of the robot at time t. Monte Carlo localization models the posterior probability $P(X_t|M, I_{1:t}, O_{1:t})$ as a set of weighted particles, each representing a possible pose with an associated weight. In this context, $I_{1:t}$ and $O_{1:t}$ refer to the sets of images and motion measurements from time 1 to t, respectively. At each step, n particles are resampled based on a likelihood determined by their weights.

There are two key differences between our approach and the Loc-NeRF method. Firstly, Loc-NeRF is scene-centric, meaning it focuses on the entire scene rather than a specific target object. Secondly, Loc-NeRF relies on a depth camera, which simplifies pose estimation. However, this type of camera is not commonly available in space missions, as discussed in the pose estimation section (See section 2.1). Additionally, their method runs on an RTX 5000 GPU, which, as discussed in Chapter 1, demands too much power for space missions.

2.3.3 Parallel Inversion of Nerfs

The goal of the Parallel INeRF[22] method is to estimate the 6D pose of a known object from a single RGB image. In their paper, Lin, Y. et *al*, address the INeRF[60] issue of requiring an initial pose candidate close to the actual camera pose. By sampling initial pose candidates in the 3D space surrounding

the target object they assume at least some will converge to a good estimate after the optimization process. The Results chapter (See Chapter 5) demonstrates that this assumption is not always valid.

The optimization process of parallel inversion of NeRF[22] iteratively updates a set of pose candidates in parallel. For each of these pose candidates, the model generates the corresponding image by passing the pose as input to a trained NeRF model. The residuals between these generated images and the observed image are then backpropagated through the fixed network to update the poses. This is repeated for a certain number of iteration after which we keep candidates with the lowest residual (lowest MSE loss) and resample new poses around them. This whole process is repeated 5 times in total. Figure 2.15 illustrates this process at different stages of the execution.

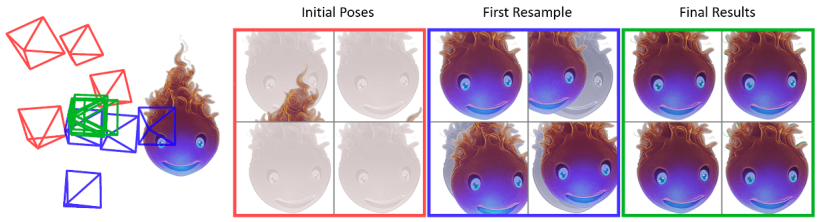


Figure 2.15: Illustration of the parallel NeRF inversion process[22]. The figure is decomposed to represent three stages of the execution. The red rectangle corresponds to four images generated based on initial pose candidates. The blue rectangle corresponds to four images generated after a first resample. Finally, the green rectangle corresponds to four images generated after the final resample.

The parallel inversion of NeRF[22] approach leverages instant-NGP[33] to speed up the INeRF[60] process. The paper also explores the impact of different loss functions on the performance of the model. It shows that occlusion and lighting conditions can be partially addressed by modifying the loss function to "Mean absolute percentage error" (MAPE[6]) instead of the common practice to employ L2 loss like it was done in Inerf.

2.4 Our contribution

As mentioned in the introduction (See chapter 1) our contribution is threefold.

Firstly, we develop a method for pose estimation over a sequence of RGB images. Our approach is based on neural radiance fields[31] for two key reasons: first, S. Sharma and S. D'Amico [49] demonstrated that classical feature-based methods underperform compared to learning-based approaches in spacecraft pose estimation; second, NeRF representations have shown outstanding results

in novel view synthesis [31], [29]. Section 2.3.2 details how loc-nerf[28] employs a similar approach to ours for pose estimation. However, Loc-NeRF relies on an RGBD camera, which is not available for space missions, and operates in a terrestrial, texture-rich environment. Instead of that, our method is based on INeRF [60] (See section 2.3.1) and addresses the challenge of requiring a good initial pose candidate by integrating a version of the Monte Carlo sampling discussed in the parallel inversion section (See section 3.4).

Secondly, we integrate nerfstudio [53] models into our framework. As discussed in Section 2.2.3, the nerfstudio framework is modular by design. By incorporating nerfstudio models, our framework becomes resilient to the development of new and improved versions of NeRF [31]. To use our framework with a different NeRF model, we only need to generate the corresponding configuration file via nerfstudio and use it as the model in our framework.

Finally, we evaluate the performance of our method on spacecraft images. As discussed in Section 2.1.4, the unique conditions in space environments can impact the accuracy of pose estimation models. Therefore, we investigate how INeRF[60] based techniques perform under these challenging conditions.

Chapter 3

Method

This chapter describes our method for estimating the 6D pose of a known spacecraft relative to a camera using a flow of RGB images following a trajectory. The actual implementation is discussed in the next chapter (See chapter 4).

Firstly, section 3.1 gives an overview of the method. Then, we provide a more in-depth explanation of several key components of the method. Section 3.2 discusses the different ray sampling strategies. Section 3.3 covers the gradient optimization process. Finally, section 3.4 discusses the Monte Carlo strategy used in our method and how we can use it to predict the 6D pose of a sequence of RGB images following a trajectory.

3.1 Method overview

Our method estimates the 6D pose (position and orientation) of a known spacecraft relative to a camera. It uses a sequence of RGB images captured by the camera that is orbiting around the spacecraft to predict its pose at different time steps.

To predict the pose of the spacecraft our method renders an image from a starting pose through a model and compares it to the image observed by the camera at that particular time. The residual is then backpropagated through the fixed model to iteratively update the starting pose, via gradient optimization (See section 3.3), for a variable number of iterations (See chapter 5).

The model used to generate the image corresponding to the pose is a pretrained (See section 4.2) NeRF[31]. Before rendering the image, we select pixels in the observed image following a sampling strategy (See section 3.2) and only generate the RGB colors corresponding to the selected pixels, via the NeRF. As this reduces the execution time we are able to launch this process from multiple starting poses in parallel, reducing the chances of falling in a local minimum. After all the initial pose candidates have undergone a specified number of iter-

ations through the NeRF model, the pose of the first frame is predicted to be the updated pose with the lowest loss at the end of the last iteration.

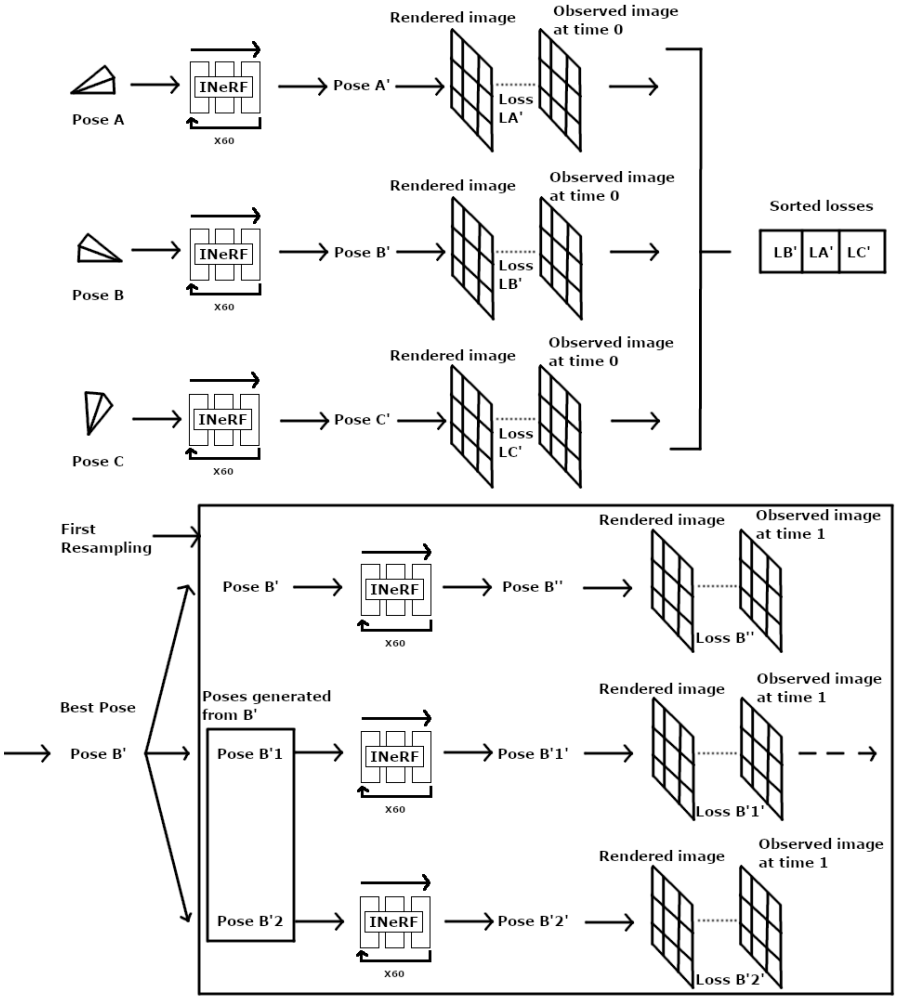


Figure 3.1: Pipeline overview of our method. We lauch the INeRF process from multiple starting positions for 60 iterations. The images rendered from the resulting poses are then compared to the first image of the target trajectory sequence of images. The losses are sorted and we keep the best pose candidates of the first step as input to the next one in addition to the poses generated from them. The process is then repeated for each image of the trajectory sequence. (Annex A.3 shows an uncut sideways view of this figure)

We then perform Monte Carlo sampling (See section 3.4) by selecting the

"best" poses of the previous step and resample a certain number of poses around these ones. The selected poses in addition to the ones derived from them serve as starting pose candidates for the prediction of the next image of the sequence. This process is repeated until reaching the last image of the target sequence. Figure 3.1 represent an overview of the method.

The Monte Carlo sampling strategy discussed above is inspired by the parallel inversion of NeRF paper[22]. Though, we leverage the fact that we have a sequence of images to accelerated the pose estimation process. Instead of comparing our produced pixels to the corresponding ones in the same target image at each Monte Carlo resampling, we change the target image to be the next one in the sequence. It is shown in the results chapter (See chapter 5) that doing so causes the number of starting poses needed at each resampling to decrease extremely fast once the method starts converging.

3.2 Ray sampling strategies

Sampling pixels of the target image to generate rays for, was a key step in the development of the method. As the images we used were smaller than the ones used in the Inerf[60] framework, we decided to sample less points but the precise number vary depending on the case studied (See chapter 5). The base version of the Inerf code[35] defined "interest point" as a patch of pixels in the center of the image. It also defined "interest region" as the same patch that can be shifted in the x and y axis. Instead, we decided to follow the guidelines of the Inerf[60] paper itself and defined "interest point" and "interest region" as follows.

3.2.1 Random Sampling

Pixels are selected randomly in the target image.

3.2.2 Interest Point Sampling

Keypoints are detected in the target image (We used the Sift[26] keypoint detection algorithm for this). We then randomly select points from these keypoints. If the number of points to be selected is larger than the actual number of keypoints detected in the target image, we fall back to the random strategy for the remaining number of points.

3.2.3 Interest Region Sampling

Keypoints are detected in the target image. Then, we form a dilated mask by creating a field of 5x5 pixels around those keypoints. Finally we select pixels in this mask. If the number of pixels to be sampled is greater than the

total number of pixels in the mask we fall back to the random strategy for the remaining ones.

3.3 Gradient-based optimization

This section explains the gradient-based optimization process used in our method, the formulas shown in this section come from the INeRF[60] paper and are modified to correspond to our method.

We define as Θ the parameters of the NeRF of a scene. The pose T of a target image I is what is to be determined.

$$\hat{T} = \arg\min_{T \in \mathrm{SE}(3)} L(T \mid I, \Theta) \tag{4}$$

We thus want to solve equation (4) for all the starting poses (T_0 to T_n) in parallel. The equation becomes:

$$\hat{X} = \arg\min_{T_{0\ldots n} \in \mathrm{SE}(3)} \sum_{i=0}^{n} L(T_i \mid I, \Theta) \tag{5}$$

To solve this optimization problem, we leverage NeRF's ability to generate an image from a given pose. We apply the same photometric loss function L that was originally used to update the MLP weights through backpropagation, but this time, we update the pose while keeping the weights fixed.

Given $\hat{T}_{i0} \in \mathrm{SE}(3)$ an initial pose estimate for pose i. We can represent $\hat{T_{ij}}$, the estimated camera pose for the current optimization step j of pose i as:

$$\hat{T}_{ij} = e^{[S_i]\theta_i} \hat{T}_{i0} \tag{6}$$

$$\text{where } e^{[S]\theta} = \begin{bmatrix} e^{[\omega]\theta} & K(S,\theta) \\ 0 & 1 \end{bmatrix}$$

where $S = [w, v]^T$ represents the screw axis, θ denotes the magnitude, [w] refers to the skew-symmetric 3x3 matrix associated with w and

$$K(S,\theta) = \left(I\theta + (1 - \cos\theta)[\omega] + (\theta - \sin\theta)[\omega]^2\right)\nu[27]$$

Finally, the loss function is differentiated iteratively through the MLP and we obtain the following gradient: $\nabla_{S\theta} L\left(e^{[S]\theta} T_{i0} \mid I, \Theta\right)$ for each of the starting poses T_i. Similar to INeRF[60], we use an Adam optimizer with exponentially decaying learning rate [15].

3.4 Parallelized Monte Carlo sampling and trajectory integration

As briefly explained in section 3.1, our method performs Monte Carlo sampling to select the pose candidates for the current image of the trajectory sequence based on the best computed poses from the previous step. For clarity reasons we define the pose estimating process for a particular image of the target sequence as a **step** for the rest of this section.

At first, our method selects a certain number of 6D poses sampled in the 3D space around the target. We also select pixels in the first image of the trajectory sequence to generate rays for.

Then, we perform a **step** and all the resulting poses are associated with a loss. This loss is computed using Mean squared error (MSE). It corresponds to the difference between the RGB color of the pixels sampled in the first image of the target sequence and the corresponding RGB color of the pixels generated by the NeRF[31] model for the candidate pose. We then perform Monte Carlo sampling by taking a certain percentage of the poses for which the loss is the smallest. For each of these poses, x new poses are created by randomly sampling in their neighbourhood. This is achieved by adding small random perturbations to the 4x4 matrix representing the pose. To apply these perturbations we define a rotational range r and translational range tr that are fixed throughout the whole execution process. A ϕ, θ and ψ rotations, each randomly sampled in [-r,r], are then applied to the pose. The pose is also translated along each axis by a value randomly sampled within the range [-tr,tr].

Next, we use these generated poses, along with the best ones from the previous step, as input for the subsequent step and continue the process. This time sampling the pixels to generate rays for in the second image of the trajectory sequence.

This process is repeated for all the images in the sequence. It is shown in the results chapter (See chapter 5) that once the method find a pose relatively close to the target it only requires a small number of starting poses to achieve good accuracy on the next image of the sequence.

Chapter 4

Software Implementation

This chapter details the implementation of our method, focusing on the core functions and hardware constraints. The chapter is organized as follows:

Firstly, section 4.1 provides an overview of the method and describes how the various functions interact. Then, section 4.2 discusses the pre-training of the NeRF model and sections 4.3 and 4.4 explore the main functions used in the method in greater detail. Finally, section 4.5 discusses the hardware limitations encountered during the implementation.

Our framework is based on the official INeRF[60] implementation[35]. However, as this initial code did not include all the features described in their paper[60], we adopted an updated version that aligns more closely with their guidelines[55] as our baseline. We then adapted all the functions related to the NeRF[31] implementation, allowing a model generated by nerfstudio to be seamlessly integrated into the INeRF[60] framework. The rest of our method was built upon this foundation, forming our final pipeline.

4.1 Implementation overview

Figure 4.1 provides an overview of the code pipeline.

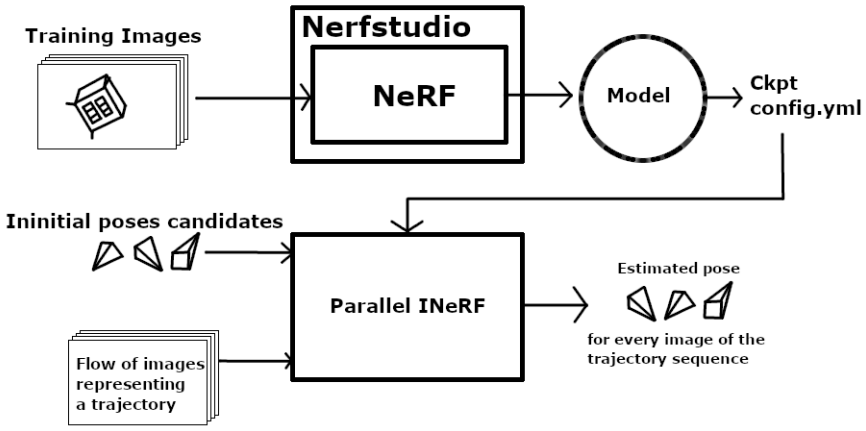


Figure 4.1: Pipeline overview of our code.

When launching the script of our framework the *main()* function is called. This function is responsible for generating starting poses and formatting them. The starting poses are typically sampled in a ".json" file containing all the images used to train the NeRF[31] model. Once they are all sampled, these poses are converted to a tensor object for which the device is set to GPU.

After that, we launch the *run()* function with the following inputs:

- The tensor object with all the starting poses
- The initial learning rate (usually set to 0.01)
- A number n associated with one of the target images in the trajectory sequence
- A number i of iterations

This *run()* function represents the INeRF[60] process executing for i iterations, with a target image corresponding to the nth image of the trajectory sequence. The function returns the updated poses, each one associated with a loss corresponding to the difference between the image generated by the NeRF[31] from this particular pose and the target image.

We then rank the poses generated by the *run()* function in ascending order based on their loss values. The pose associated with the lowest loss is the estimated pose for the current image. We then select the first x poses and generate y poses candidates for each of them with the *generate_candidate_from_pose()* function. This generated poses are created by adding small random rotation

and translation to the initial pose (See section 3.4).

The *run()* function is then called again with the same learning rate and number of iterations parameters. We replace the starting poses from the previous step by a tensor object composed of the x best poses selected on the previous step and their associated newly generated poses. The target image number n is also changed to become n+1 (the next image in the trajectory sequence).

The whole process is then repeated for each image in the trajectory sequence.

4.2 Pre-training of the NeRF model

We chose for our method to be built around a NeRF[31] model pre-trained on the nerfstudio[53] framework. The resulting format is a "config.yml" file which contains a nerfstudio pipeline object. This object includes all the information about our trained NeRF[31] model. Notably the weights associated with the MLP. More specifically we chose their nerfacto implementation 2.2.3. Integrating our method directly in the nerfstudio framework would have required too much time for it to be considered.

The use of Instant-NGP[33] was considered as it was the implementation used in the parallel inversion of neural radiance fields paper[22]. Despite its promising results this implementation was not compatible with our framework version. Though its use would most certainly speed up the method, we decided not to explore it further for several reasons.

Firstly, the parallel inversion paper is missing some key elements for replicability: number of points sampled from the target image(See section 3.2), the size of the target image and the exact Monte Carlo sampling strategy. Secondly, some parts of the parallel inversion of neural radiance fields[22] implementation were encrypted[36]. Finally, the focus of this thesis is on the use of NeRF inversion for spacecraft pose estimation and the addition of a flow of target images following a trajectory, as opposed to relying on a single fixed image. The emphasis is less on the actual execution time of the method itself.

Considering these facts it would have been hard for us to obtain a baseline to compare our method to. However, using Instant-NGP[33] to pre-train the NeRF[31] model is still something to consider for future works.

4.3 The iterating function

The function that performs the iterations is called *run()*. The input and output of that function are discussed in section 4.1. An iteration correspond to the

following steps:

Firstly, we sample pixels to generate rays for in the target image following the chosen sampling strategy (See section 3.2). Then, we create a nerfstudio camera object with all the input starting poses. For each of these poses, we generate the rays corresponding to the sampled pixels of the target image. This is done by calling the *generate_rays()* function that returns a raybundle object for each of the strating poses. Subsequently, these raybundles are used as input to the *get_outputs_for_camera_ray_bundle()* function that computes the rgb color associated with each ray of each starting pose. A loss is then calculated for each initial pose by computing the mean squared error (MSE) between the generated RGB output and the actual RGB color of the target image. Finally, we stack the losses and backpropagate them through the NeRF[31] to update the initial poses using the gradients accumulated by the tensors (*loss.backward*).

After updating the initial poses in the first iteration, we proceed to the next, repeating the process until reaching the total number of iterations specified in the input of the *run()* function. Finally, we return all the final poses along with their associated losses.

4.4 The rays and output generating functions

Both functions originate from the nerfstudio[53] framework. The *generate_rays* function, which generates all the rays associated with the selected pixels of the target image, was kept unchanged. However, the *get_outputs_for_camera_ray_bundle()* function required modifications to work in our framework, mainly because the nerfstudio version did not retain gradients during computation. While this function is quite complex and involves two subfunctions, its core task is to perform a forward pass through the MLP of the NeRF[31] model.

4.5 Hardware limitations

All the experiments conducted in this thesis were done on a laptop with an NVIDIA RTX 3060 GPU, an intel core i9 processor and 16GB of RAM (which might not be in a perfect state).

The maximum number of poses for which we can run our method in parallel is limited to 10-15 with this setup. To counter that effect we implemented a version of the method capable of functioning per batches instead of true parallelization. This version is slower but enables us to increase the maximum number of starting poses usable as input to 80. This number is the upper limit throughout this thesis.

Chapter 5

Results

This chapter presents the results obtained over the different experiments conducted throughout this thesis. First, section 5.1 presents the different datasets used to conduct our experiments. Then, section 5.2 demonstrates how our framework can accelerate the pose estimation of an object using synthetic terrestrial images captured along a trajectory. Section 5.3 evaluates the method's performance with synthetic spacecraft images compared to terrestrial ones. Section 5.4 then presents the method's performance on real spacecraft images. Finally, section 5.5 summarizes the main results and discusses the limiting factors of our approach.

5.1 Validation datasets

5.1.1 Lego Dataset

The synthetic LEGO dataset is a collection of rendered images used for evaluating NeRF[31] models. This dataset was generated as part of NeRF's evaluation to assess model performance in synthetic, controlled settings. All images present in this dataset are of size 800x800 pixels. It is divided in a training set of 100 images, a validation set of 100 images and a testing set with 200 images following a trajectory. We use this dataset as a texture-rich, terrestrial synthetic baseline to evaluate the performance of our method under ideal conditions. The trajectory sequences of images on which the model is tested are extracted from the testing part of the dataset.

5.1.2 Speed+ Dataset

The Speed+[40] dataset is the improved version of the SPEED[48] dataset. It is specifically designed to address the domain gap problem (See section 2.1.4) of its predecessor. This dataset is composed of 60,000 synthetic images for training and 9,531 hardware-in-the-loop images of a spacecraft mockup model. These hardware-in-the-loop images were captured from the TRON (Testbed for Rendezvous and Optical Navigation) facility. A robotic "arm" took pictures

of a mockup model of the studied spacecraft in an environment reproducing spaceborne illumination conditions with high-fidelity. These particular images will be referred to as real. This dataset was used to train both our synthetic and real NeRF models respectivelly on the synthetic and real part of the images.

5.1.3 Shirt Dataset

Shirt[37] dataset represents the same spacecraft as the speed[48] and speed+[40] datasets. It is composed of two trajectory scenarios (ROE1 and ROE2). The synthetic part of the dataset was created using the OpenGL-based computer graphics renderer and the real part was created in the TRON facility. The trajectory sequences of images used to test our method are extracted from this dataset.

5.2 Experiments on Synthetic images

This section covers the results of our method on the synthetic terrestrial texture-rich images. Section 5.2.1 shows the overall performance of our method on these types of images. Then, section 5.2.2 explores the influence of the sampling strategy chosen on the angular error. Finally, section 5.2.3 shows the influence of the number of points sampled both on the execution time and angular error. Throughout this section the images are resized to be 100x100 pixels in size.

5.2.1 Results

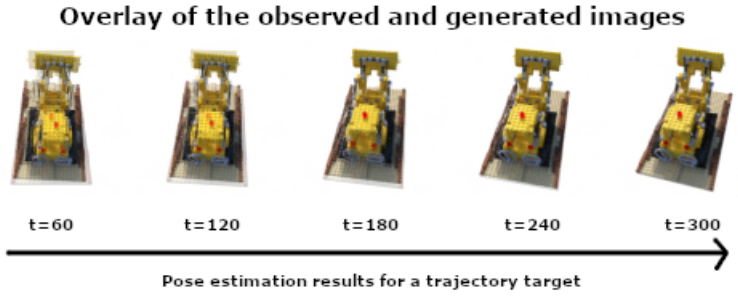


Figure 5.1: Overview of the pose estimation results for a flow of target images following a trajectory. t represent the current iteration. The images represent the overlay of the image generated at iteration t by the NeRF[31] and the observed image at time t. The observed image correspond to the low opacity part of the image and the estimated pose correspond to the high opacity part.

Figure 5.1 illustrates the output of our method when using synthetic images from Lego. This result was obtained for 15 starting poses and it can be ob-

served that the estimated pose is already close to the target after 60 iteration (14° angular error). Once the estimation is close to the observed image the approximation error drops fast (2° error at t=180, 0.2° at t=300). The method is consistent with 10-15 starting poses sampled randomly in the training images of our NeRF[31] model. It always estimates the pose with an angular error of less than 3° over three different sequences tested 10 times each.

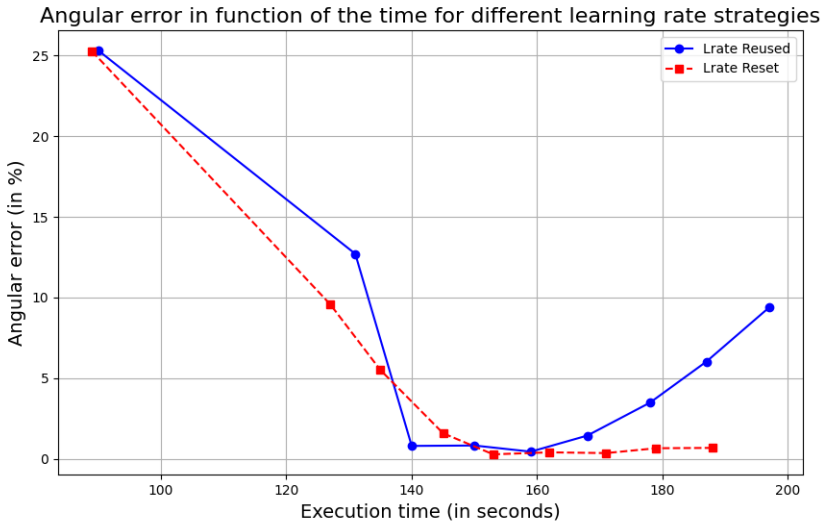


Figure 5.2: This graph represents the angular error in function of the time. Every node is a step of 60 iterations and correspond to each resampling. It can be observed that the graph starts at 89 seconds when the learning rate is reset and at 90 seconds when it is not. This corresponds to the execution time required for the first 60 iterations over 15 initial poses. After this first step 2 pose candidates are generated for the 3 best poses (6 poses in total). For all the remaining steps of 60 iterations only the best prediction is kept, making each step way faster.

The graph represented in figure 5.2 illustrates two key observations. The first one is the importance of resetting the learning rate at each resampling. The Blue line (no reset) shows an increase in the angular error after a certain number of steps while the red line (reset) does not. The second one is the considerable time gain obtained by starting from a close pose. Once an error of less than 5° is obtained, subsequent predictions remain accurate while less starting poses are needed, decreasing the execution needed to perform a step.

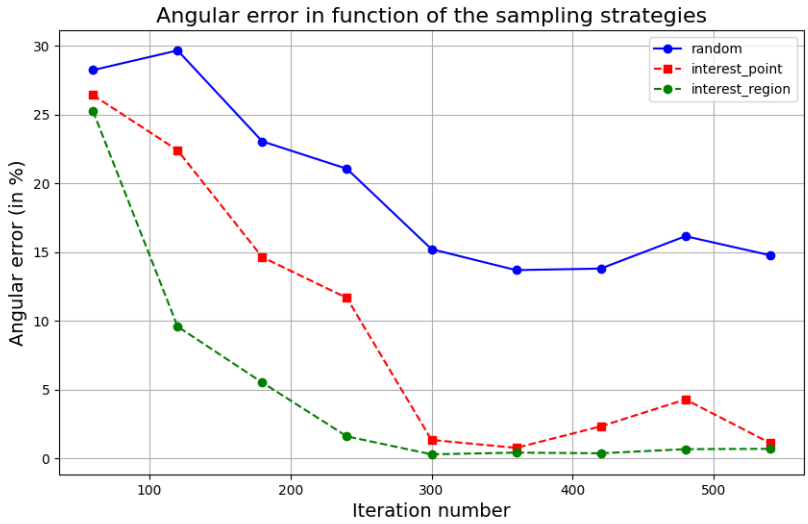


Figure 5.3: This graph represents the angular error in function of the iteration number for our three sampling strategies. It can be noted that the interest point strategy worked decently well in this case. Though it still converges slower than the interest region strategy. It is also more prone to falling into local minima.

5.2.2 Influence of the sampling strategy

Figure 5.3 shows the importance of a good sampling strategy in our framework. The random sampling strategy is not effective. Although figure 5.3 depicts a single execution of the method, a similar trend has been consistently observed across 10 executions on 3 different trajectories. The interest point sampling strategy performs reasonably well but fell into a local minimum 6 times out of 30 executions with 15 starting poses. In contrast, the interest region strategy not only converges faster than the interest point strategy but also consistently found a pose with an angular error below 5° in all 30 executions. Those results align with the ones obtained in the Inerf[60] paper. Further improving those strategies could thus be a good avenue for future research.

5.2.3 Influence of the number of points sampled

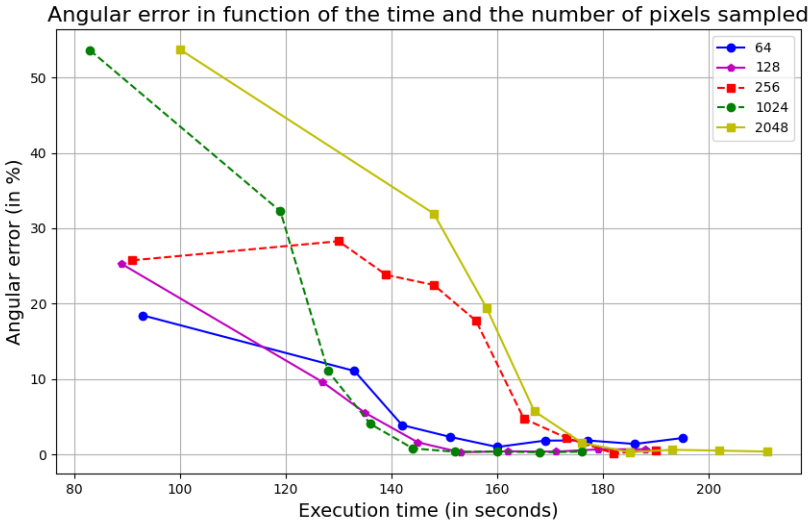


Figure 5.4: This graph illustrates the time taken to perform a total of 540 iterations with different quantities of sampled points. All other parameters are kept the same. The angular error also appears as an indication of the quality of the model in function of the number of points sampled.

From figure 5.4 we can conclude that the number of points sampled does not play a big role in the quality of the model. The influence on the execution time is also unclear. As pixels are treated in batches this can be explained by the fact that the memory is not full so the execution time is not affected that much.

5.3 Experiments on synthetic spacecraft images

This section covers the results of our method on the synthetic spacecraft images. More specifically, the NeRF[31] model is trained on 500 synthetic images from speed+[40]. Those images are selected with no earth in the background. The flow of target images following a trajectory are synthetic images from Shirt.

Section 5.3.1 discusses the performance of our method on synthetic spacecraft images. Then, section 5.3.2 explores the influence of the translation on the accuracy of the method. In this section, images are resized to be 240x150 pixels in size. Due to the nature of the spacecraft images, our experiments with lower resolution images did not yield any results.

5.3.1 Results

For the first experiments we sampled the starting poses in Shirt. This means all the images have a similar distance separating them from the target. This observation is key to our analysis. Previous works [60], [22] never addressed the impact of adding perturbation in the translation of the starting poses over the performance of their framework. Nor did they cover the impact of off-centered objects in the starting poses. We show in section 5.3.2 that these kinds of perturbations prevents the framework to achieve good results even with a large number of starting poses and iterations.

When testing our framework with the same parameters as in the Lego experiments (See section 5.2) the model converges most of the time to an angular error of less than 10°. Over 30 experiments on 3 trajectories (10 times each) the method converges to an angular error of less than 10° 22 times. In 9 out of those 22 the method converges to an angular error of less than 5°. These results were obtained by adjusting the number of pose candidates retained in the Monte Carlo sampling process compared to the Lego experiments: keeping 5 instead of 3 for the second set of 60 iterations, and 5 instead of 1 for the remaining sets. Additionally, the number of starting poses was increased from 15 to 20. This result is illustrated in figure 5.5

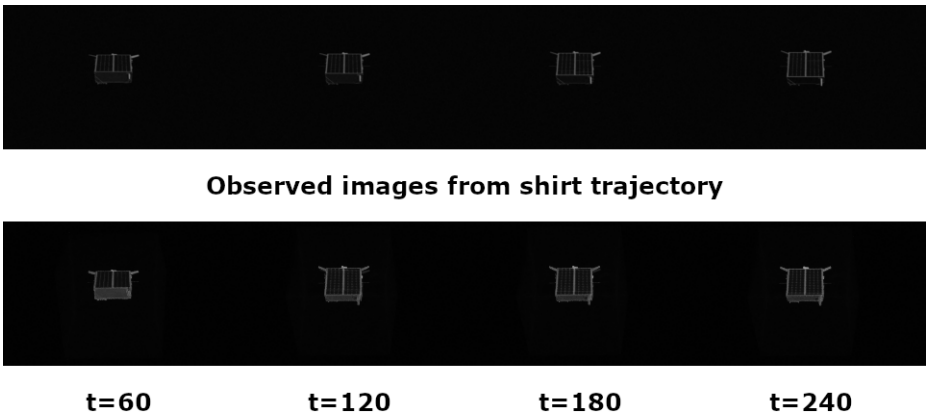


Figure 5.5: Overview of the pose prediction of our framework. The upper line represent the observed image at a certain point t in time. The bottom line correspond to the predicted images at certain iterations.

From this first experiments we can already conclude that the nature of the space images are detrimental to the performance of our framework. This can be explained by several factors.

Firstly, the poor lighting conditions in space hinder SIFT's ability to detect keypoints in the target images, even in synthetic images. Secondly, the

monochromatic nature of the images makes it difficult for the model to distinguish between black pixels of the satellite and those of the background. Finally, the textureless and symmetrical nature of the satellite reduces the number of keypoints that can be extracted from Shirt images compared to the Lego images, even though the Shirt images are 3.6 times larger.

5.3.2 Influence of the translation

After these experiments we tested the method with starting poses sampled in speed+[40]. These poses are characterized by a big difference in their distance relative to the target. Thus, sampling starting poses in this dataset correspond to random sampling in the 3D space surrounding the target object up to 9m distance relative to the satellite.

The result of our experiments when sampling starting poses in speed+ showed that the method cannot converge with the parameters used in the Lego experiments.

Following this observation we tried setting the number of iterations to a larger number (512 per resampling) and followed the Monte Carlo sampling proposed by [22]. Despite the paper lacking critical details about their exact setup, we replicated their method and arrived at the following configuration:

We performed 512 iterations per pose per resampling, utilizing Monte Carlo sampling to retain the top 25% of candidate poses at each resampling stage, with this percentage halved in subsequent resampling steps. Additionally, we generated 4 extra pose candidates for each retained pose by introducing slight random perturbations to the 4x4 pose matrices. Specifically, perturbations were applied by randomly selecting values between -3 and +3 for each axis, resulting in the pose being rotated accordingly. The translation was adjusted by adding or subtracting a value sampled between -0.5 and +0.5 meters along each axis. Furthermore, we increased the number of starting poses to 80.

The results of this experiment were not better than the ones obtained in the previous setup.

To confirm our work hypothesis that this absence of convergence was caused by the translation part of the starting poses we tried to add random translational perturbation to our starting poses while testing our method on the lego images. This added translational perturbation simulates a similar spread of poses than the speed+ training images. This caused the model to no longer converge. This was expected and can be explained by the following reasons:

When sampling poses at the same distance respective to the target object we limit the search space to the border of a sphere (Approximately 10 meters of diameter sphere in the case of the Shirt dataset). When sampling our starting poses in speed+, we change the search space to become the border of a 18m

diameter sphere and everything inside. Making it very unlikely to have a starting pose candidate close to the observed image even with 80 starting poses. Having a starting pose close to the observed image is a mandatory requirement for Inerf[60] to converge. In the parallel inversion paper[22] they assume that at least some of the starting poses will converge in the right direction. This can no longer be assumed in this case while sampling poses in speed+.

5.4 Experiment on real Spacecraft images

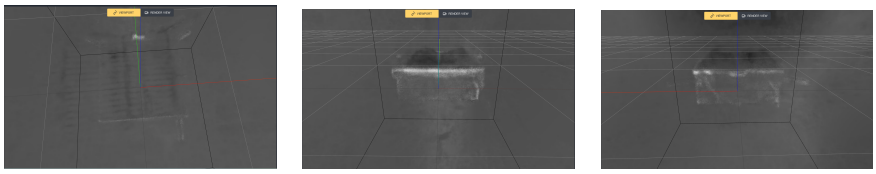
This section covers the results of our method on "real" images of the spacecraft. More specifically, the NeRF[31] model is trained on 500 real images of the speed+ dataset. Those images are selected with no earth in the background. The flow of target images following a trajectory are real images from Shirt.

It was shown in the previous section that the model cannot converge when sampling starting poses in speed+ even on synthetic images (with the upper limit of 80 starting poses imposed by the memory of the computer used to conduct these experiments). Thus, we focus solely on sampling poses in Shirt for the experiments on real spacecraft images.

The images shown in this section are resized to be of 240x150 pixels in size.

5.4.1 NeRF[31] representation from real speed+ images

As shown in figure 5.6 the quality of the NeRF[31] scene representation generated using real images from speed+ is of low quality, making it hard to distinguish the front view 5.6(b) from the back view 5.6(c).



(a) Top view (b) Front view (c) Back view

Figure 5.6: This figure represents different views of the NeRF[31] model generated using 500 real images from the speed+ dataset

5.4.2 Results

Following the observations made in section 5.4.1 we didn't expect the method to be able to predict an accurate pose with real images of the space dataset. Though it can be observed in figure 5.7 that the method converges to a plau-

sible pose.

The low quality of the NeRF representation of the scene makes it hard to distinguish certain symmetric views. This sometimes causes the method to converge to a local minima of the inverted satellite (+-180° angular error).

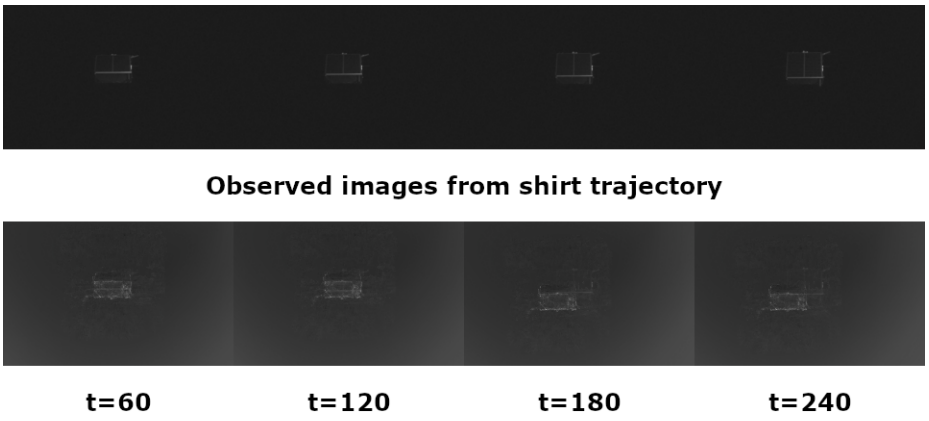


Figure 5.7: Overview of the pose prediction of our framework over time. The upper line represents the observed images and the bottom line corresponds to the corresponding predicted images.

The method either converges to the right pose or a plausible local minima approximately 50% of the time. Over 30 experiments on 3 trajectories (10 times each) the method either converges to an error of less than 10° or 190° on a plausible local minimum 14 times.

5.5 Discussions

From all our experiments, we can see that three key elements affect the accuracy of our method.

Firstly, the nature of the images influences the accuracy of our method. Section 5.2 shows that our method performs well on texture-rich terrestrial synthetic images while section 5.3 shows a decrease in performance on synthetic space images. This is because terrestrial synthetic images generally offer higher contrast, making keypoints more detectable. Specifically, the Lego object has numerous edges and distinct features, which makes keypoint detection easier compared to the symmetrical and texturless spacecraft.

Secondly, the initial pose candidates are critical to the performance of the model. Sections 5.2 and 5.3 show that the method predicts accurate poses when sampling initial pose candidates at the same distance to the object than the

target pose. This considerably reduces the search space compared to random sampling in the 3D space around the target (See section 5.3.2).

Finally, section 5.4 shows the importance of a good representation of the target (a good NeRF[31] model). When the model's quality is poor, our method is more susceptible to falling into local minima (See section 5.4.2).

Chapter 6

Future works

Section 5.5 discusses the main factors affecting the accuracy of our method, namely: the nature of the target images, the initial pose candidates and the quality of the NeRF model. Throughout this thesis, it was also shown that the main limiting factors for our method were its speed and robustness to space environments. In this chapter we will show how these observations can be leveraged in future works.

Firstly, we discuss the development of better versions of NeRF (See section 6.1) both in term of speed and robustness when handling space imagery. By improving the quality of the NeRF models we could solve one of the limiting factor that was observed in section 5.4 and increase the robustness of our method.

Then, section 6.2 discusses potential improvements to sampling strategies. Section 5.2.2 shows the influence of a good sampling strategy improving them could decrease the execution time needed for the method to find an accurate estimation of the target pose. Furthermore improving the keypoint detection algorithm could partially solve the issues associated with the nature of space images.

6.1 Development of improved versions of NeRF

The Results chapter (See Chapter 5) discussed the quality of the NeRF[31] representations of scenes when dealing with real spacecraft datasets. A very recent paper (June 2024) by A. Legrand et *al*[19] addresses this issue by developing a NeRF[31] which employs learnable appearance embeddings to handle the diverse appearance conditions present in real images. Integrating this new NeRF[31] in our framework could lead to improved performances when dealing with spacecraft datasets. Furthermore, we could consider comparing multiple renderings of the initial pose to the observed image in our INeRF[60] framework. Though implementing this would further increase the computational

complexity of our methods. Tradeoffs would thus need to be explored between execution time and faster convergence.

For our method to automate space missions, it needs to be able to perform trajectory estimation in real time. To achieve this, its speed needs to be considerably improved. Developing faster NeRFs[31] is another avenue for future researches.

6.2 Exploration of better sampling strategies

As mentioned in the Results chapter (See section 5.3.1) less keypoints are detected by the Sift[26] algorithm when dealing with spacecraft datasets. Exploring new sampling strategies and keypoint detecting techniques specifically designed for space environment could prove interesting as we showed that the sampling strategy had a big impact on the performance of the method (See section 5.2.2).

Chapter 7

Conclusion

In this work, we present a method to estimate the 6D pose of an uncooperative spacecraft relative to a camera by leveraging neural radiance fields [31]. The spacecraft's pose is derived from a flow of RGB images captured along a trajectory.

We show throughout our experiments that our methods is effective at finding the 6D pose of a known object using synthetic terrestrial images. However, it is limited by several factors including the nature of the images, the quality of the NeRF[31] model and the initial pose candidates.

Performance begins to decline when using synthetic spacecraft images, as the characteristics of these images make keypoints detection more challenging. This emphasizes the importance of an effective sampling strategy. We also demonstrate that the search space around a target object is too extensive for our method to effectively explore on personal laptop setups without an initial coarse estimate of the object's pose. The impact of translation on the accuracy of our model is specifically analyzed by applying random translational perturbations to the starting poses, which results in less accurate predictions.

Experiments on real images highlight the need for a high-quality NeRF[31] model in INeRF[60] approaches. This encourages the development of improved versions of NeRF tailored to the specific conditions of space environments.

These observations show that we are still far from real time 6D pose estimation of an uncooperative spacecraft using a single RGB camera. Significant advancements are still needed to achieve fully autonomous rendezvous operations or debris removal missions. However this work is a first step which serves as entry point in a complex field and there is still room for future improvements
.

Bibliography

[1] Michal Adamkiewicz et al. "Vision-Only Robot Navigation in a Neural Radiance World". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4606–4613. DOI: 10.1109/LRA.2022.3150497.

[2] Jonathan T. Barron et al. "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 5470–5479.

[3] *CAD model definition*. URL: https://en.wikipedia.org/wiki/Computer-aided_design.

[4] Bo Chen et al. "Satellite pose estimation with deep landmark regression and nonlinear pose refinement". In: *Proceedings of the IEEE/CVF international conference on computer vision workshops*. 2019, pp. 0–0.

[5] Enric Corona, Kaustav Kundu, and Sanja Fidler. "Pose Estimation for Objects with Rotational Symmetry". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 7215–7222. DOI: 10.1109/IROS.2018.8594282.

[6] Arnaud de Myttenaere et al. "Mean Absolute Percentage Error for regression models". In: *Neurocomputing* 192 (2016). Advances in artificial neural networks, machine learning and computational intelligence, pp. 38–48. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.12.114. URL: https://www.sciencedirect.com/science/article/pii/S0925231216003325.

[7] Frank Dellaert et al. "Monte carlo localization for mobile robots". In: *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*. Vol. 2. IEEE. 1999, pp. 1322–1328.

[8] Thanh-Toan Do et al. *Deep-6DPose: Recovering 6D Object Pose from a Single RGB Image*. 2018. arXiv: 1802.10367 [cs.CV]. URL: https://arxiv.org/abs/1802.10367.

[9] Yuqi Fang et al. "Source-free unsupervised domain adaptation: A survey". In: *Neural Networks* (2024), p. 106230.

[10] Aneesh M Heintz and Mason Peck. "Spacecraft State Estimation Using Neural Radiance Fields". In: *Journal of Guidance, Control, and Dynamics* 46.8 (2023), pp. 1596–1609.

[11] Wenxiu Huan, Mingmin Liu, and Qinglei Hu. "Pose estimation for non-cooperative spacecraft based on deep learning". In: *2020 39th Chinese Control Conference (CCC)*. IEEE. 2020, pp. 3339–3343.

[12] Yurong Huo, Zhi Li, and Feng Zhang. "Fast and accurate spacecraft pose estimation from single shot space imagery using box reliability and keypoints existence judgments". In: *IEEE Access* 8 (2020), pp. 216283–216297.

[13] Josip Josifovski et al. "Object Detection and Pose Estimation Based on Convolutional Neural Networks Trained with Synthetic Data". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 6269–6276. DOI: 10.1109/IROS.2018.8594379.

[14] Linh Kästner, Daniel Dimitrov, and Jens Lambrecht. "A markerless deep learning-based 6 degrees of freedom pose estimation for mobile robots using rgb data". In: *2020 17th International Conference on Ubiquitous Robots (UR)*. IEEE. 2020, pp. 391–396.

[15] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[16] M. Kisantal et al. "Satellite pose estimation challenge: Dataset, competition design, and results". In: *IEEE Transactions on Aerospace and Electronic Systems* (2020).

[17] Jogendra Nath Kundu et al. "Object Pose Estimation from Monocular Image Using Multi-view Keypoint Correspondence". In: *Computer Vision – ECCV 2018 Workshops*. Ed. by Laura Leal-Taixé and Stefan Roth. Cham: Springer International Publishing, 2019, pp. 298–313. ISBN: 978-3-030-11015-4.

[18] Antoine Legrand, Renaud Detry, and Christophe De Vleeschouwer. "End-to-end neural estimation of spacecraft pose with intermediate detection of keypoints". In: *European Conference on Computer Vision*. Springer. 2022, pp. 154–169.

[19] Antoine Legrand, Renaud Detry, and Christophe De Vleeschouwer. *Leveraging Neural Radiance Fields for Pose Estimation of an Unknown Space Object during Proximity Operations*. 2024. arXiv: 2405.12728 [cs.CV]. URL: https://arxiv.org/abs/2405.12728.

[20] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "EP n P: An accurate O (n) solution to the P n P problem". In: *International journal of computer vision* 81 (2009), pp. 155–166.

[21] Zhigang Li et al. "Robust rgb-based 6-dof pose estimation without real pose annotations". In: *arXiv preprint arXiv:2008.08391* (2020).

[22] Yunzhi Lin et al. "Parallel Inversion of Neural Radiance Fields for Robust Pose Estimation". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 9377–9384. DOI: 10.1109/ICRA48891.2023.10161117.

[23] Fuchang Liu et al. "Recovering 6D object pose from RGB indoor image based on two-stage detection network with multi-task loss". In: *Neurocomputing* 337 (2019), pp. 15–23. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.12.061. URL: https://www.sciencedirect.com/science/article/pii/S0925231218315236.

[24] Jin Liu et al. "Realtime RGB-based 3D object pose detection using convolutional neural networks". In: *IEEE Sensors Journal* 20.20 (2019), pp. 11812–11819.

[25] Yuanpeng Liu et al. "Regression-Based Three-Dimensional Pose Estimation for Texture-Less Objects". In: *IEEE Transactions on Multimedia* 21.11 (2019), pp. 2776–2789. DOI: 10.1109/TMM.2019.2913321.

[26] D.G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[27] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.

[28] Dominic Maggio et al. "Loc-NeRF: Monte Carlo Localization using Neural Radiance Fields". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 4018–4025. DOI: 10.1109/ICRA48891.2023.10160782.

[29] R. Martin-Brualla* et al. "NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021).

[30] G. Marullo et al. "6D object position estimation from 2D images: a literature review." In: *Multimedia Tools and Applications* (2022).

[31] B. Mildenhall et al. "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *Communications of the ACM* (2021).

[32] Ron Mittelhammer, George G Judge, and Douglas J Miller. *Econometric foundations pack with CD-ROM*. Cambridge University Press, 2000.

[33] T. Müller et al. "Instant neural graphics primitives with a multiresolution hash encoding". In: *ACM Transactions on Graphics* (2022).

[34] E. Muñoz et al. "Fast 6D pose from a single RGB image using Cascaded Forests Templates". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 4062–4069. DOI: 10.1109/IROS.2016.7759598.

[35] *Official repository for Inerf paper*. URL: https://github.com/yenchenlin/iNeRF-public/blob/master/pixel-nerf/pose_estimation.ipynb.

[36] *Official repository for the parallel inversion of neural radiance fields paper*. URL: https://github.com/NVlabs/ParallelInversion/tree/master/scripts.

[37] Tae Ha Park and Simone D'Amico. *SHIRT: Satellite Hardware-In-the-loop Rendezvous Trajectories Dataset*. Stanford Digital Repository. Available at https://purl.stanford.edu/zq716br5462. 2022. DOI: 10.25740/zq716br5462.

[38] Tae Ha Park and Simone D'Amico. "Robust multi-task learning and online refinement for spacecraft pose estimation across domain gap". In: *Advances in Space Research* 73.11 (2024), pp. 5726–5740.

[39] Tae Ha Park, Sumant Sharma, and Simone D'Amico. "Towards robust learning-based pose estimation of noncooperative spacecraft". In: *arXiv preprint arXiv:1909.00392* (2019).

[40] Tae Ha Park et al. "SPEED+: Next-Generation Dataset for Spacecraft Pose Estimation across Domain Gap". In: *2022 IEEE Aerospace Conference (AERO)*. 2022, pp. 1–15. DOI: 10.1109/AERO53065.2022.9843439.

[41] Aniruddha V Patil and Pankaj Rabha. "A survey on joint object detection and pose estimation using monocular vision". In: *arXiv preprint arXiv:1811.10216* (2018).

[42] Leo Pauly et al. "A survey on deep learning-based monocular spacecraft pose estimation: Current state, limitations and prospects". In: *Acta Astronautica* 212 (Nov. 2023), pp. 339–360. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2023.08.001. URL: http://dx.doi.org/10.1016/j.actaastro.2023.08.001.

[43] Georgios Pavlakos et al. "6-DoF object pose from semantic keypoints". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2011–2018. DOI: 10.1109/ICRA.2017.7989233.

[44] Sida Peng et al. "PVNet: Pixel-Wise Voting Network for 6DoF Pose Estimation". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4556–4565. DOI: 10.1109/CVPR.2019.00469.

[45] *Perspective-N-point (PNP) pose computation, OpenCV*. URL: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html.

[46] Jason Rambach et al. "Learning 6DoF Object Poses from Synthetic Single Channel Images". In: *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. 2018, pp. 164–169. DOI: 10.1109/ISMAR-Adjunct.2018.00058.

[47] Benjamin Sapp, David Weiss, and Ben Taskar. "Parsing human motion with stretchable models". In: *CVPR 2011*. 2011, pp. 1281–1288. DOI: 10.1109/CVPR.2011.5995607.

[48] S Sharma, TH Park, and S D'Amico. "Spacecraft pose estimation dataset (SPEED)". In: *Stanford Digit. Repository*. 2019.

[49] S. Sharma and S. D'Amico. "Neural network-based pose estimation for noncooperative spacecraft rendezvous". In: *IEEE Transactions on Aerospace and Electronic Systems* (2020).

[50] Sumant Sharma, Connor Beierle, and Simone D'Amico. "Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks". In: *2018 IEEE Aerospace Conference*. IEEE. 2018, pp. 1–12.

[51] Sumant Sharma and Simone D'Amico. "Pose estimation for non-cooperative rendezvous using neural networks". In: *arXiv preprint arXiv:1906.09868* (2019).

[52] M. Sundermeyer et al. "Augmented Autoencoders: Implicit 3D Orientation Learning for 6D Object Detection". In: *Int J Comput Vis 128*. 2020, pp. 714–729. DOI: https://doi.org/10.1007/s11263-019-01243-8.

[53] Matthew Tancik et al. "Nerfstudio: A Modular Framework for Neural Radiance Field Development". In: *ACM SIGGRAPH 2023 Conference Proceedings*. SIGGRAPH '23. , Los Angeles, CA, USA, Association for Computing Machinery, 2023. ISBN: 9798400701597. DOI: 10.1145/3588432.3591516. URL: https://doi.org/10.1145/3588432.3591516.

[54] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. "Real-Time Seamless Single Shot 6D Object Pose Prediction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[55] *Updated Inerf repository more in line with the paper itself in the sampling method*. URL: https://github.com/salykova/inerf/tree/main.

[56] Teodora Vatahska, Maren Bennewitz, and Sven Behnke. "Feature-based head pose estimation from images". In: *2007 7th IEEE-RAS International Conference on Humanoid Robots*. 2007, pp. 330–335. DOI: 10.1109/ICHR.2007.4813889.

[57] Dor Verbin et al. "Ref-nerf: Structured view-dependent appearance for neural radiance fields". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2022, pp. 5481–5490.

[58] *Voxel Definition by wikipedia*. URL: https://en.wikipedia.org/wiki/Voxel.

[59] Mei Wang and Weihong Deng. "Deep visual domain adaptation: A survey". In: *Neurocomputing* 312 (2018), pp. 135–153.

[60] Lin Yen-Chen et al. "iNeRF: Inverting Neural Radiance Fields for Pose Estimation". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 1323–1330. DOI: 10.1109/IROS51168.2021.9636708.

[61] Jiun-Kai You et al. "Object Pose Estimation Incorporating Projection Loss and Discriminative Refinement". In: *IEEE Access* 9 (2021), pp. 18597–18606. DOI: 10.1109/ACCESS.2021.3054493.

[62] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. "DPOD: 6D Pose Object Detector and Refiner". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 1941–1950. DOI: 10.1109/ICCV.2019.00203.

[63] Xin Zhang, Zhiguo Jiang, and Haopeng Zhang. "Real-time 6D pose estimation from a single RGB image". In: *Image and Vision Computing* 89 (2019), pp. 1–11.

[64] Wanqing Zhao et al. "6D object pose estimation via viewpoint relation reasoning". In: *Neurocomputing* 389 (2020), pp. 9–17. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.12.108. URL: https://www.sciencedirect.com/science/article/pii/S0925231220300333.

[65] Guoyu Zuo et al. "Low-Quality Rendering-Driven 6D Object Pose Estimation from Single RGB Image". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207286.

Appendix A

Appendix

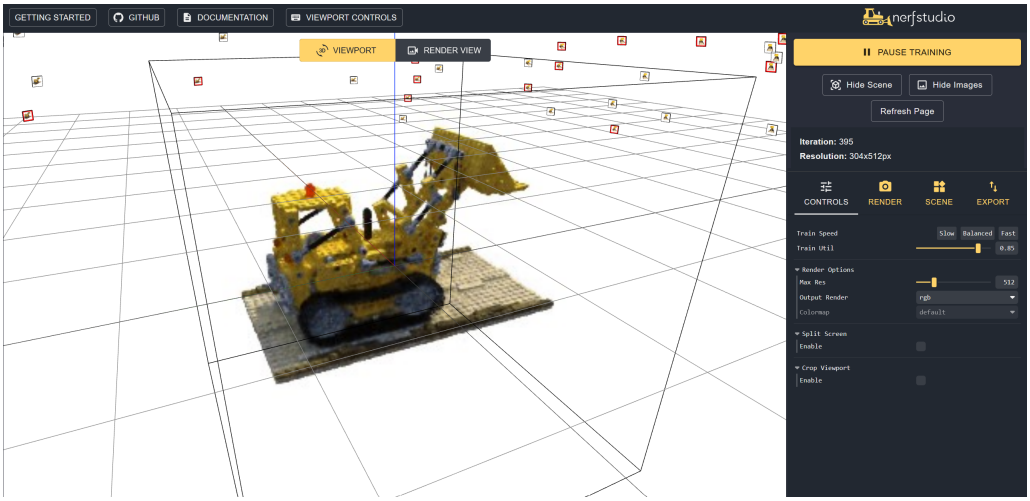


Figure A.1: Nerfstudio Viewer on Lego Dataset

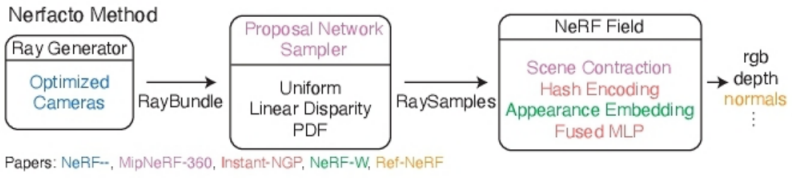


Figure A.2: Nerfstudio Pipeline [53]

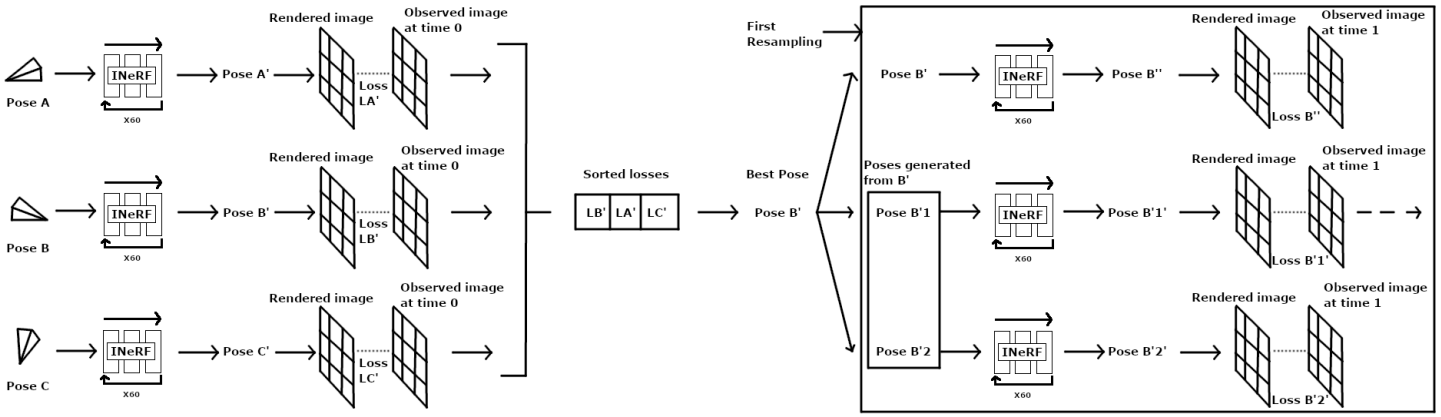


Figure A.3: Pipeline overview of our method. We lauch the INeRF process from multiple starting positions for 60 iterations. The images rendered from the resulting poses are then compared to the first image of the target trajectory sequence of images. The losses are sorted and we keep the best pose candidates of the first step as input to the next one in addition to the poses generated from them. The process is then repeated for each image of the trajectory sequence.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**