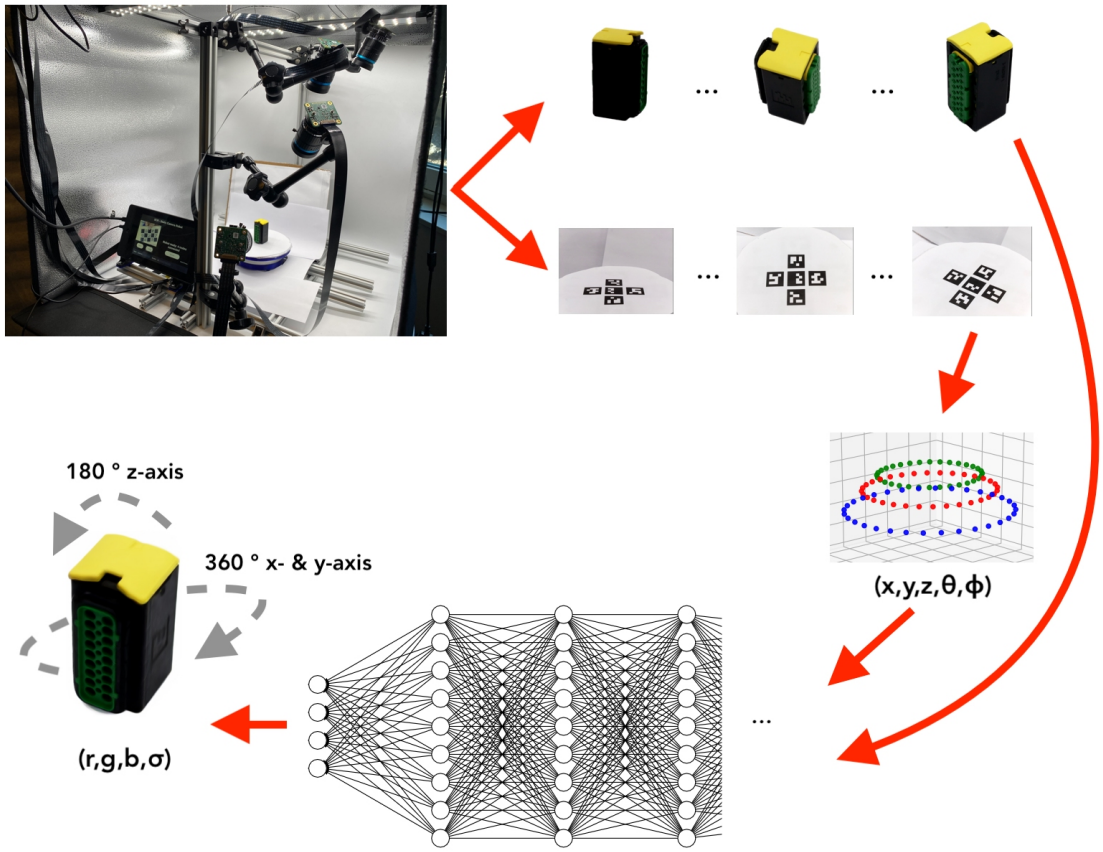


Camera Pose Estimation and Multiview 2D to 3D Reconstruction

Robust camera pose estimation utilizing ArUco markers for novel view synthesis using Neural Radiance Fields

Master's thesis in Systems, Control and Mechatronics

Filip Persson
Alfred Hazard

June 3, 2022
DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Camera Pose Estimation and Multiview 2D to 3D Reconstruction

Robust camera pose estimation utilizing ArUco markers for novel view synthesis using Neural Radiance Fields

Alfred Hazard
Filip Persson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2022

Camera Pose Estimation and Multiview 2D to 3D Reconstruction
Robust camera pose estimation utilizing ArUco markers for novel view synthesis using Neural Radiance Fields
Filip Persson and Alfred Hazard

Supervisor: Erik Sintorn, Department of Computer Science and Engineering (CSE)
Supervisor: Isak Ernstig, Wiretronic AB
Supervisor: Ludwig Friborg, Wiretronic AB
Examiner: Mats Rudemo, Department of Mathematical Sciences

Master's Thesis 2022
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg

Cover: A simplified overview of the pipeline presented in the project.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Camera Pose Estimation and Multiview 2D to 3D Reconstruction
Robust camera pose estimation using ArUco markers for novel view synthesis using Neural Radiance Fields
Alfred Hazard, Filip Persson
Department of Mathematical Sciences Chalmers University of Technology

Abstract

Generating 3D representations of objects is of interest in a wide breadth of industries. These representations are often created by hand through 3D modelling softwares such as CAD - derivatives. This in itself can be a complex process in order to capture desired detail and view dependent highlights. In this project, we investigate how Neural Radiance Fields (NeRF) can be used to extract the structure of an object within a bounded scene. NeRF encodes the structure of a scene in a Multi Layer Perceptron using a postional encoding heuristic. The training input is a 5-tuple consisting of a set of images and corresponding viewing directions of the cameras, and the output is the expected volume density and RGB color.

This project is a continuation of a Masters thesis by Isak Ernstig and David Olofsson [1], which has shown that accurate camera pose estimations are crucial to allow NeRF to render high fidelity views of a scene. In a feature poor environment traditional pose estimation pipelines using feature detection algorithms, such as the commonly used COLMAP estimator, have been shown to yield inadequate estimates. In this project, fiducial markers known as ArUco markers have been used to deduce accurate 2D-3D correspondences through detection and error correction. The detected markers allow for accurate usage of typical Computer Vision methodologies, such as Perspective-N-Point and Bundle Adjustment, allowing qualitative camera poses to be estimated. We conclude that our approach enables training a NeRF model, for 360 degree view datasets with symmetrical and uniform objects, which may then subsequently render high fidelity novel views of real life objects. We also show the importance of accurate camera calibration and correct sampling intervals for rays when querying a trained NeRF model from a given viewpoint.

Keywords: CAD, Neural Radiance Field, Multi Layer Perceptron, positional encoding, camera pose, COLMAP, ArUco, Perspective-N-Point, Bundle Adjustment

Acknowledgements

We would like to thank our supervisors at Wiretronic, Isak Ernstig and Ludwig Friborg, as well as members of their team, Patrik Andersson and Oscar Andersson, for all the help and enthusiasm regarding our work. They were all instrumental to the success of the project through their supply of necessary hardware, aid with practicalities and experience with certain pieces of software.

We would also like to give a big thank you to our supervisor Erik Sintorn, Associate Professor at Chalmers, for his expertise and attention to detail in issues where a certain bit of confusion could easily occur. Erik was of great importance to every part of the project, especially with regard to the workings of Neural Radiance Fields and how to approach areas in Computer Vision that pertained to camera pose estimation. Furthermore, we wish to extend our appreciation to our examiner Mats Rudemo, Professor Emeritus at Chalmers. Mats gave us constructive feedback and was always keen to help out with logistical issues and the presentation of our work.

Finally, we wish to thank our families and close friends for their support and encouragement.

Alfred Hazard, Filip Persson, Gothenburg, May 2022

List of Acronyms and Abbreviations

Below is the list of acronyms and abbreviations that have been used throughout this thesis listed in alphabetical order:

Adam	Adaptive Moment Estimation
ANN	Artifical Neural Network
ArUco	Markers used for point detection in images
Blender	Software that renders 2D images of 3D models
COLMAP	Computer vision software able of deducing camera poses and 3D point clouds from images
CPD	Coherent Point Drift
DLT	Direct Linear Transform
DNN	Deep Neural Network
GPU	Graphics Processing Unit
GUI	Graphical User Interface
ICP	Iterative Closeset Point
MVS	Multi View Stereo
NeRF	Neural Radience Fields
PSNR	Peak Signal-to-Noise Ratio
RANSAC	Random Sampling Consensus
ReLU	Rectified Linear Unit
RGB	Red Green Blue, a reference used for 2D images with the three color channels
SfM	Structure from Motion
SIFT	Scale Invariant Feature Transform
SVD	Singular Value Decomposition

Contents

List of Figures

List of Tables

1
Introduction

In order to create a virtual 3D environment, a natural necessity is the ability to render digital 3D objects. These could for example be a game map with trees and mountains or a film set of a city skyline. Traditionally assets of this nature are created using mesh based surfaces such as polygons and/or quads. However, the main downsides to these methods are the complexity in creating them with accurate specular highlights and material specific attributes since these features have to be skillfully modeled by hand.

Given that the implementation of Deep learning models is becoming more prevalent within an ever increasing breadth of industries the world of computer graphics has not been an exception, and recently there has been a massive surge in research regarding alternative ways to represent 3D surfaces. Notably, by using neural rendering techniques, wherein Artifical Neural Networks (ANN) are trained using a sparse set of images to generate *novel views*, which are views never previously seen by the network during training. One of the most promising and well researched approaches uses what is known as *Neural Radiance Fields* (NeRF). NeRFs have been shown to be able to render novel views with high fidelity while also not compromising specular features and view dependent highlights. [2].

A company located in central Gothenburg, named Wiretronic, aims to develop a robust pipeline that generates high fidelity 3D models of objects used within the company using images of those objects. Previous work on this pipeline has been done through another thesis project at Chalmers, where the pipeline consists of a turntable with a statically mounted camera that captures images by letting the object rotate with the turntable to get multiple viewpoints of a chosen object. The project uses the software COLMAP to estimate camera poses using the Scale-Invariant Feature Transform (SIFT) algorithm. The information from the pipeline is used to train Neural Radiance Fields (NeRF), which in turn may generate novel views of a scene which in conjunction with volumetric rendering techniques can generate a 3D model. Currently implementations of NeRFs are often trained using synthetic data with access to perfect camera poses, or the scenes that are to be rendered are by nature rich in features that jointly allow COLMAP to accurately estimate the camera poses. However, a problem may arise when the environment is lacking in features or an object is symmetrical. This can lead feature based algorithms, such as COLMAP, to generate inadequate camera pose estimations. Consequently, the input data to a NeRF is flawed which leads to poor results [1].

In this project we investigate two methods to accurately estimate camera poses of

the images taken by using ArUco fiducial markers as opposed to the very commonly used COLMAP estimator. Both methods use the open-source library OpenCV [3], where the first one uses a single ArUco marker and the second one uses five ArUco markers. The methods are similar, where the largest difference is simply the amount of points from detected markers. With the methods, our own Structure from Motion (SfM) pipeline is implemented in which the typical feature matching step is skipped by simply using the detected corners of the ArUco markers between each of the images instead. The estimated poses from each of the two methods are then used to train a NeRF model, where we then investigate whether these may enable NeRF to generate high fidelity novel views.

1.1 Related work

The idea of using certain sets of points in images and matching these points against images with similar points has been around since 1981 [4]. The early ideas consisted of finding corners and edges in images, which in short meant looking at areas with large gradients. The complexity of these ideas where later increased by introducing a correlation area at found corners, such that images could be matched that had close resemblance in correlation areas. Additions to these methods came in the form of matching using a descriptor of local image areas that was invariant to rotation, hence matching images with different orientations was possible. These earlier methods were improved by making the matching invariant to scale differences in images, which is a method called Scale-Invariant Feature Transform (SIFT) created by David G. Lowe [4]. The SIFT method is capable of discerning distinctive features in images and matching them against images that are different in scale, rotation, clutter and affine transformation.

A number of general purpose Structure-from-motion algorithms have been developed recently, where one of the newer and most prevalent is the COLMAP software by Johannes L. Schönberger et. al [5] [6]. From images of the same scene, this software is able to deduce 3D point clouds and camera poses for the images. The method relies on features found in the images, specifically using the SIFT method. Because this software is meant to be used for general purpose SfM, the disadvantages of the COLMAP software stand out with datasets consisting of images of uniformly coloured and textured objects.

Another method of estimating camera poses, valuable for fields as robotics and augmented reality, is using recognizable markers in the applicable environment. The usage of markers comes from the fact that they can be simple to localize, yield good precision and are fast to detect. There are a number of different markers and coherent detection algorithms that have been developed. Some of these methods use markers that are circular, where markers distinguish from each other with circular patterns in the markers. These markers come with a flaw, namely that a marker only yield one central point of detection, therefore creating the need of having several circular markers in order to estimate a coherent camera pose [7]. Other methods use markers that are outlined as a square, using the inner part of the marker for

identification and error correction. This sort makes it possible to estimate a camera pose from a detection of one marker, as the marker provides four points as the corners of the outer square. Some of these markers suffer from problems as being sensitive to different lightning in a scene, not being able to use the inner identification part of the marker for error correction and having a predetermined set of markers. The lastly mentioned issue gives that the method can not be generalized to applications needing many different sets of identification. The mentioned ArUco markers are one of the recent methods of dealing with marker detection in images. This method lets the user define their own configuration of marker sets, in terms of amounts of bits that make up the inner identification matrix and size of the whole marker. The marker and detection algorithm also deals with error correction, allowing alteration of found points to be as accurate as possible. The ArUco marker method also deals with varying lighting conditions by firstly segmenting images through finding the contours of markers, using a local adaptive threshold [7].

Several of the newly developed NeRF implementations are used with synthetic and real life datasets paired with camera poses deduced using COLMAP. In the work done by Ben Mildenhall et al. [2] and Alex Yu et al. [8] the implementations are tested with images from real life scenes, coupled with camera poses from COLMAP. There are implementations done with NeRF without having the camera poses pre-calculated using either synthetic data or COLMAP, which is done in the work by Yoonwoo Jeong et al. [9]. The optimization in their work consists of both learning Neural Radiance Fields and camera models jointly. Experimentation shows that the implementation can produce results that are similar to the original NeRF [2] without poses from COLMAP and results that are slightly better with camera poses initialized and optimized with COLMAP results [9]. The used datasets are the same ones used as in the work by Ben Mildenhall et al.

1.2 Objective

This project aims to further develop the mentioned pipeline and the novel view generation using synthetic images and images from real life scenes. As such this project aims to achieve the following goals:

1. Accurately estimate the pose of cameras from 2D images of an object, even in cases where the SIFT algorithm would otherwise struggle to detect keypoints consequently leading COLMAP to generate inadequate camera poses.
2. Using the camera pose estimates, create a structure of a specified object using a NeRF implementation.
3. Combine 1 and 2 into a clear and usable pipeline to generate visualizations of objects in the real world.

In order to evaluate the chosen methods and results during the project, it is relevant to propose research questions that can highlight how certain aspects of the project are developing. Therefore, a number of research questions are proposed:

- What are qualitative alternatives as compared to methods based on keypoint detection using the SIFT algorithm for pose estimation? How does using ArUco markers as a method of estimating poses compare to using the traditional feature-based methods?
- How close can a rendered novel 2D scene using real data and NeRF resemble a result with synthetic data and NeRF, where the comparison is done using PSNR [10] and other qualitative measurements?
- What are possible and meaningful ways of visualizing the captured scene by a NeRF and how can the quality of these representations be measured? How can these visualizations be generalized and are they accurate enough such that measurements from a scene can be used?

1.3 Scope

It is reasonable to narrow down the scope of this work because the project is time constrained and the scientific areas that are studied are broad. This also serves as a means to clarify the project, such as explaining what sort of data will not be used in the project. Hence, a few limitations are introduced:

- The datasets used in this project will be bounded, i.e the scenes that are used will contain a single object where novel views of the object is essentially what the NeRF algorithm should generate.
- The proposed pipeline from this project will not be scaleable in terms of introducing much smaller or larger objects than the ones used in the project. The objects will be of roughly the same size.
- Part of the project will contain gathering real-life data of objects where a environment will be chosen that fits with the setup of the pipeline. This means that specific environments, such as a black or white background, might be needed when collecting the data using the turntable.

2
Theory

This chapter aims to give the reader further insight regarding the areas involved in this project. The two main areas covered are Computer Vision and Deep Machine Learning, which are divided into separate important subsections.

2.1 Computer Vision

The human eye is an extremely comprehensive tool that is able to extract visual information from surroundings, while the human brain at the same time can decompose and perceive details at an impressive rate. The translated information could, as an example, be how many persons are visible in the surroundings, depth perception to certain objects and geometric understanding based on shadows cast on the ground. Computer Vision is an area that tries to encapsulate the way humans perceive scenes by researching mathematical techniques that allow a computer to understand and extract information from the scenes. The field has grown fast and modern algorithms in the area can produce information that are useful in several ways [11]. Cameras have a natural role in the field of Computer Vision and their properties related to mathematics are important to understand in order to correctly interpret information from relevant surroundings.

2.1.1 Camera models and projective geometry

2.1.1.1 The Pinhole Camera Model and Homogenous Coordinates

The most widely used representation of a camera is the pinhole camera model, which is visualized in Figure 2.1.

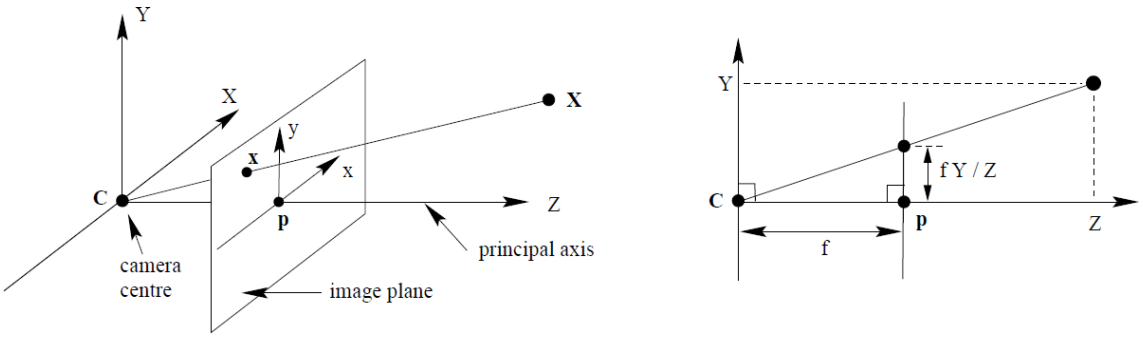


Figure 2.1: The pinhole camera model

This is an ideal model and as such an approximation of cameras in the real world. The different parts of the model can be explained as follows:

- **principal axis**: Viewing direction of the camera.
- **C**: Camera center, convergence point of rays going from points in 3D space through the image plane.
- **x**: An arbitrary point on the 2D image plane.
- **X**: An arbitrary point in the 3D world
- **p**: Principal point, the point which marks the intersection of the principal axis with the 2D image plane
- **f**: Focal length, distance between the camera center and the 2D image plane

2.1.1.2 Camera calibration (intrinsics)

Cameras have different properties compared to each other, hence these differences have to be taken into account when using standardized Computer Vision algorithms. A common way of referring to a certain cameras calibration settings is writing it as a matrix **K**, just like in equation (2.1). These settings are commonly known as camera intrinsics [11].

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

The variables f_x and f_y represent focal length with respect to the sensor of the camera for x and y dimensions. Variable s denotes skew, i.e accounting for the possibility that the camera sensor is not set perpendicular to the principal point. Lastly, the variables c_x and c_y represent the image center in pixel coordinates, which are often deduced as half of the height and width of the image. This calibration matrix makes it possible to transform 3D points that are camera-centered, to 2D points in correct pixel coordinates for a specific camera [11].
The intrinsics for a certain camera need to be estimated as these values are not often provided by manufacturers and might differ between camera units of the same model. There are a number of possible ways to find the intrinsics of a camera, where a common approach involves using objects of known measurements and mapping 3D scene points to corresponding 2D image points of said object [11].

Another camera property that is important in Computer Vision is the fact that a lot of camera lenses impose radial distortion in images. This means that straight lines in a scene tend to become curved in the final image taken with the camera. These distortions can be compensated for using estimated radial distortion parameters and low-order polynomials such that pixel values are altered [11].

2.1.1.3 Camera pose (extrinsics) and 3D transformations

There are a several possible ways to perform 3D transformations. These transformations maintain certain properties for points and objects, for example, an important property is preserving angles between lines and planes. A useful 3D transformation is the Euclidean transformation, which preserves a lot of important properties when

performing transformation. The Euclidean transformation only alters the rotation and translation of points. It can be written as a 3×4 matrix, where a part $\mathbf{R}$ (3×3) describes rotation and the other part $\mathbf{t}$ (3×1) describes the translation [11]. Such a 3D transformation can be seen in equation (2.2).

$$\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}} \tag{2.2}$$

The Euclidean transformation can be used to describe the extrinsics of a camera, in other words how a certain camera is placed and rotated in a coordinate system. So for instance, in equation (2.2), the variables $\bar{\mathbf{x}}$ and $\hat{\mathbf{x}}$ could respectively be the 3D point in a world coordinate frame and the transformed 3D point in the cameras coordinate system.

2.1.1.4 Camera matrix

In order to map 3D points to correct 2D points in accordance with a specific camera, it is needed to make use of a so called camera matrix (or projection matrix). This is a 3×4 matrix, and if the camera is said to be *calibrated* ($\mathbf{K}$ is known), the extrinsics are combined with the intrinsics to create the full camera matrix depicted in equation (2.3).

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tag{2.3}$$

Multiplying a 3D point with a camera matrix therefore transforms and projects a 3D point to a camera correct 2D pixel representation of the same point [11]. An important property, however, is *depth*. The pinhole camera model describes the relationship between a point in 3D along a ray originating from the camera center. This ray then intersects the corresponding 2D point on the image plane, and therefore the depth variable λ is introduced. From this, a fundamental Computer Vision relationship of projecting 3D points to 2D points can be formed:

$$\lambda \mathbf{x} = \mathbf{P}\mathbf{X} \tag{2.4}$$

2.1.1.5 Mapping points in 3D to image points in 2D

In projective geometry, points are often used in their so called *homogenous* represenation. A 2D point in the traditional setting described by two numbers $\in R^2$

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

can be extended to a homogeneous representation by introducing a third coordinate $\bar{w}$

$$\bar{\mathbf{x}} = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{w} \end{bmatrix} \in P^2$$

Here P^2 signifies a homogenous 2D point. The most commonly used form of projection is *perspective projection*. With this approach, a 2D coordinate is deduced by

dividing through the coordinates by the last entry. As such, a coordinate given in a homogeneous representation can be projected to 2D space by de-homogenizing it by dividing all the coordinate values by the last entry. Hence a de-homogenized point gets the coordinate values in (2.5).

$$\mathbf{x} = \frac{1}{\bar{w}} \begin{bmatrix} \bar{x} \\ \bar{y} \\ 1 \end{bmatrix} \tag{2.5}$$

All the same principles apply to 3D points. Homogeneous coordinates are used heavily since they allow representing rotations and translations as matrix operations. These two operations in particular are very important in Computer Vision as they describe the relation between a point in the 3D world and the 2D point of an image. With this in mind, transforming a point in the 3D world to a pixel on an image involves 3 main steps where we make use of the camera matrix in (2.3):

1. Transforming from world coordinates to camera coordinates

$$\bar{\mathbf{X}} = \mathbf{R}\mathbf{X} + \mathbf{t}, \qquad \mathbf{X} \in P^3, \bar{\mathbf{X}} \in R^3 \tag{2.6}$$

2. Transforming from camera coordinates to image coordinates using perspective projection

$$\bar{\mathbf{x}} = \frac{1}{\bar{X}_3} \begin{bmatrix} \bar{X}_1 \\ \bar{X}_2 \\ \bar{X}_3 \end{bmatrix} = \begin{bmatrix} \bar{X}_1/\bar{X}_3 \\ \bar{X}_2/\bar{X}_3 \\ 1 \end{bmatrix}, \qquad \bar{\mathbf{x}} \in P^2 \tag{2.7}$$

3. If the camera is calibrated ($\mathbf{K}$ is known), image coordinates can now be transformed to sensor coordinates (pixel values)

$$\mathbf{x} = \mathbf{K}\bar{\mathbf{x}}, \qquad \mathbf{x} \in R^2 \tag{2.8}$$

2.1.2 Structure From Motion

Structure from Motion (SfM) is the concept of using 2D point correspondences between several images to simultaneously estimate both the location of the 3D world points and the location of the cameras in relation to those 3D points [11]. SfM is a powerful tool to gain knowledge of given scene and has for example been used to construct sparse 3D pointclouds of entire cities. In this chapter, the main steps used in SfM will be explained.

2.1.2.1 Key Point Detection and Descriptors

In order to establish connections between images, points of interest most often referred to as keypoints must be found in the images. These are generally speaking points that are easily discernable in the scene due to for example the high contrast of a black spot on a white background. After the keypoints have been found, descriptors which describe the points themselves are created such that 2D-2D point correspondences between images can be established. However the task of extracting keypoints is somewhat non-trivial, since information in an image changes if it undergoes a change in scale or rotation. This could lead to different keypoints being

detected for the same object, with different images of the scene. One of the most well known and used algorithms to counteract the influence of scale and rotation in both the keypoint detection step and the descriptor generation step is the Scale-Invariant Feature Transform [4].

2.1.2.2 Scale-Invariant Feature Transform (SIFT)

SIFT is a method that can deduce distinctive features from camera images, which can be further used to match different views of a scene to each other. The feature detection is done in a number of stages. Firstly, a computation is done for the whole image with a "difference-of-Gaussian" function to find interest points invariant to scale and rotation. For these possible interest points a model is fit and keypoints are chosen based on stability computations of keypoints. Then orientations are paired with the keypoints deduced from local gradient directions. Lastly these local gradient directions for the keypoints are transformed to a new representation, invariant to shape distortion and changes in illumination [4].

A ”normal” image of, say, a landscape, a brick wall or something else, of resolution 500×500 pixel sized image yields about 2000 features. These features can be used to match against images from the same scene, with the same features. The matching is done by comparing features individually and a match of features is deduced based on the Euclidean distance for the features vectors [4]. SIFT finds features that are distinct, meaning that they somehow stand out and are relatively easy to discern between images. In most scenarios, SIFT is able to deduce distinct features between images - since the subject is most often not simply a blank wall and is not completely symmetrical. Images where there are a lot of the same features, for example images of an all white coffee mug, yields false matches that need to be filtered out.

2.1.2.3 ArUco markers

A way to bypass the issue of having to find keypoints between images is by introducing fiducial markers to the scene. There are several types of fiducial markers, one of them is the ArUco code that consists of a black border and an inner binary matrix that allows specific identification of ArUco code instances [12]. ArUco markers have different bitsizes ranging from 4×4 to 7×7 bits to differentiate the binary matrix patterns. The border of the marker yields fast detection possibilities while the inner binary matrix makes it possible to utilize correction methods and error detection. With calibrated cameras, it is possible to deduce the camera pose by going from the marker coordinate system to the camera coordinate system by using a 3D transformation. An example of this is shown in Figure 2.2.

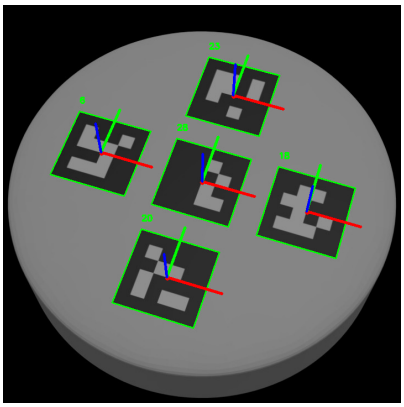


Figure 2.2: Detection of ArUco markers and their coherent poses visualized as green, red and blue vectors from which the camera pose can be deduced.

An important property with the usage of ArUco markers is the fact that the detection of the markers yield a known amount of points, which is the four corners of each marker. This means that no outlier methodology needs to be paired with the detection of ArUco markers, as the points are identified with the markers' different binary matrices and the amount of markers are known. With this in mind, it is also possible that the markers are not detected by the mentioned algorithms. Therefore it is important that images of ArUco markers are taken with a clear view of the marker.

2.1.2.4 Epipolar Geometry and Pose Estimation

Once 2D-2D point correspondences have been established, the next step is to generate a link between the two images. If the cameras in question are calibrated we may multiply their respective camera matrices with the inverse of their respective instrinsics matrices, $\mathbf{K_i}^{-1}$. Placing one of the cameras in the origin we therefore get the two matrices $[I\ \ 0]$ and $[R\ \ t]$. The link that ties these two cameras together is represented by the *Essential matrix* E. 2-view epipolar geometry is based on the principles in Figure 2.3.

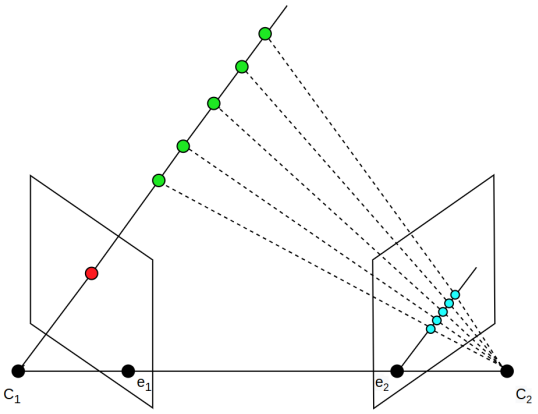


Figure 2.3: 2-view epipolar geometry

In this configuration, several different scene points (in green) all project to the same 2D point (in red) in the first camera C_1. However, the same scene points project onto the *epipolar line*, e_2 in the second image taken by camera C_2. This information is captured by E and satisfies the *epipolar constraint* in (2.9) [13].

Let x_1 be an image point in camera 1 and x_2 an image point in camera 2, then the following relation is to be satisfied.

$$x_2^T E x_1 = 0 \tag{2.9}$$

$E^T x_2$ is the epipolar line in camera 1 and Ex_1 is the epipolar line in camera 2.

If $\mathbf{x_i} = [x_i, y_i, z_i]$ and $\bar{\mathbf{x}}_\mathbf{i} = [\bar{x}_i, \bar{y}_i, \bar{z}_i]$ then we can expand (2.9)

$$\begin{aligned} \bar{\mathbf{x}}_\mathbf{i}^T E \mathbf{x_i} = {} & E_{11}\bar{x}_i x_i + E_{12}\bar{x}_i y_i + E_{13}\bar{x}_i z_i \\ & + E_{11}\bar{y}_i x_i + E_{12}\bar{y}_i y_i + E_{13}\bar{y}_i z_i \\ & + E_{11}\bar{z}_i x_i + E_{12}\bar{z}_i y_i + E_{13}\bar{z}_i z_i \end{aligned} \tag{2.10}$$

Doing this for all matching image points between two images then yields us the matrix representation for (2.10) in (2.11).

$$\begin{bmatrix} \bar{x_1}x_1 & \bar{x_1}y_1 & \bar{x_1}z_1 & \dots & \bar{z_1}z_1 \\ \bar{x_2}x_2 & \bar{x_2}y_2 & \bar{x_2}z_2 & \dots & \bar{z_2}z_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \bar{x_n}x_n & \bar{x_n}y_n & \bar{x_n}z_n & \dots & \bar{z_n}z_n \end{bmatrix} \begin{bmatrix} E_{11} \\ E_{12} \\ E_{13} \\ \vdots \\ E_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.11}$$

Given that this may not have an exact solution, we solve it as a homogenous least squares problem using *Singular Value Decomposition* (SVD).

$$\bar{E} = USV^T \tag{2.12}$$

From this we are given an approximate essential matrix $\bar{E}$. However, an important property for the true E is rank-deficiency. Additionally it must have two non-zero and equal singular values, such that a rotation and translation decomposition is possible. The latter condition in conjuction with the fact that scale is arbitrary allows us to enforce E to have two singular values equal to 1. As such we finally get the true E in (2.13) [13].

$$E = U\text{diag}\left(\begin{bmatrix} 1 & 1 & 0 \end{bmatrix}\right) V^T \tag{2.13}$$

From this matrix the camera extrinsics can be deduced by deconstructing E. Given it's Singular Value Decompostion, we can decompose part of using a skew symmetric

matrix and an orthogonal matrix.

$$Z = \underbrace{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{skew sym.}} \qquad W = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{orthogonal.}} \tag{2.14}$$

$$E = U\Sigma V^T = ZW = Z^T W^T \tag{2.15}$$

From this we are given two solutions

$$E = U\Sigma\, V^T = UZWV^T = UZU^T UWV^T \tag{2.16}$$

$$E = U\Sigma\, V^T = UZ^T W^T V^T = UZ^T U^T UW^T V^T \tag{2.17}$$

These two solutions are jointly often referred to as a "twisted pair". The decomposition of E makes up the rotational part of the second camera, which leaves a translational part. The "twisted pair" gives two options, t or $-t$, which in total leaves 4 possible combinations. With $P_1 = [I \mid 0]$ we get:

$$P_2 = [UWV^T \mid t], \quad P_2 = [UWV^T \mid -t]$$
$$P_2 = [UW^T V^T \mid t], \quad P_2 = [UW^T V^T \mid -t]$$

Selecting any combination of P_1 and P_2 allows scene points to be *triangulated*, by minimizing the algebraic expression in (2.18).

$$\begin{bmatrix} & \vdots & & \\ P_{11} - x_i P_{31} & P_{12} - x_i P_{32} & P_{13} - x_i P_{33} & P_{14} - x_i P34 \\ P_{21} - y_i P_{31} & P_{22} - y_i P_{32} & P_{23} - y_i P_{33} & P_{24} - x_i P34 \\ & \vdots & & \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \mathbf{0} \tag{2.18}$$

This gives the four possible configurations in Figure 2.4 [13].

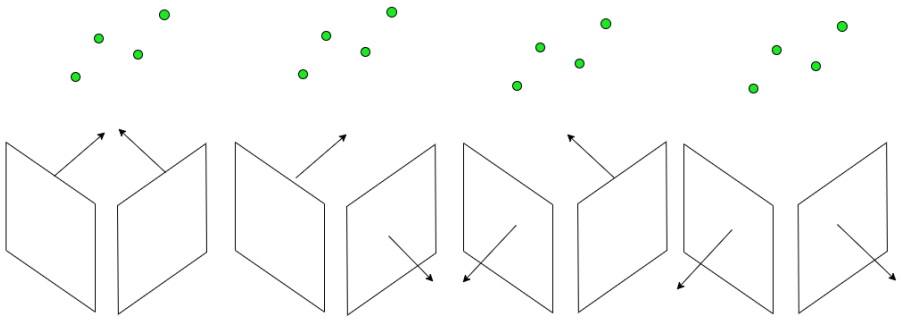


Figure 2.4: 3D reconstruction using triangulation, green dots are scene points, arrows are the principal axis of the cameras

Mathematically all these configurations project the scene points to the same 2D points. This is why during 3D reconstruction, the *Cheirality constraint* is established. This essentially means picking the two cameras where all scene points appear

in front of both of them, which in Figure 2.4 would be the leftmost configuration. After this, using the two correct cameras, the scene points can be reprojected to estimate the correctness of the cameras by comparing the reprojected 3D points to their true pixel equivalents. In order to construct an entire scene captured by multiple cameras, a new camera is added by establishing what 3D points that were triangulated can be seen in that camera and matching those to 2D image points captured by that camera. Then an essential matrix is created connecting the new camera with one of the previous cameras and new 3D points are thereafter triangulated. This procedure is then repeated until the entire scene has been constructed.

Once the entire scene has been created, or iteratively every time a new camera is introduced, an optimization step known as *Bundle adjustment* is performed. Bundle adjustment is a non-linear least squares problem that aims to optimize both the 3D scene points and the relative motion of the cameras by minimizing the error based on the criteria of minimal sum of reprojection errors [14]. The error in the most common setting is the Euclidean distance between a predicted reprojected 3D point and to where it was actually projected determined by the camera estimation. Given an example scene such as in Figure 2.5

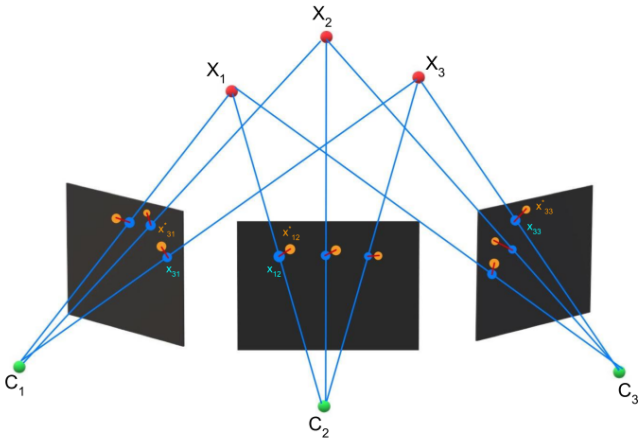


Figure 2.5: Scene showing cameras, 3D points, predicted reprojected 3D points and actual reprojected 3D points

wherein cameras C_i, 3D points X_i, intended reprojected 3D points x_{ij} (in blue) and actual reprojected 3D points x^*_{ij} (in orange) are available, the goal then becomes to minimize the sum in equation (2.19).

$$\min \sum_{i}^{n} \sum_{j}^{m} (x_{ij} - \pi(C_i, X_i))^2 \tag{2.19}$$

Here, $\pi(C_i, X_i)$ is a non-linear operation.

2.1.2.5 Camera Resectioning

In chapter 2.1.2.4, the term *triangulation* was mentioned. Briefly speaking, we project 3D world coordinates into a scene, giving us a sense of depth, by leveraging the essential matrix E and 2D-2D point correspondances - similar to how our eyes are able to perceive depth by having two reference angles. However, sometimes, 3D world coordinates and corresponding 2D image coordinates may already be known but the task of determining the camera extrinsics (pose) remains. This is instead known as *Camera Resectioning*. A very commonly used resectioning algorithm is the Direct Linear Transform (DLT). By cross-multiplying (2.4), we get (2.20)[15].

$$\mathbf{x}_i \times \mathbf{P}\mathbf{X}_i = 0 \tag{2.20}$$

This yields two equations shown in (2.21)

$$\begin{aligned} P_{11}X_i + P_{12}Y_i + P_{13}Z_i + P_{14} - x_i(P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34}) = 0 \\ P_{21}X_i + P_{22}Y_i + P_{23}Z_i + P_{24} - y_i(P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34}) = 0 \end{aligned} \tag{2.21}$$

Doing this for all scene points and corresponding image points then yields the matrix representation in (2.22)

$$\underbrace{\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ & & & & & & & \vdots & & & & \\ X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -x_iX_i & -x_iY_i & -x_iZ_i & -x_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -y_iX_i & -y_iY_i & -y_iZ_i & -y_i \end{bmatrix}}_{M} \begin{bmatrix} \mathbf{p_1} \\ \mathbf{p_2} \\ \mathbf{p_3} \end{bmatrix} = 0 \tag{2.22}$$

This problem is handled as a homogenous least squares such as in (2.11), therefore we can solve it by using eigenvalue computations. This is efficiently done by using Singular Value Decomposition.

$$M = USV^T \tag{2.23}$$

The most optimal solution is then found by reshaping the vector in V^T that corresponds to the smallest eigenvalue, which is the last vector. Therefore we finally get $\mathbf{P}$ in (2.24) [15].

$$\mathbf{P} = \mathbf{v}_* = V^T[:, -1] \in R^{12\times 1} \rightarrow R^{3\times 4} \tag{2.24}$$

Due to the linear nature of the relations between the camera matrix P, the scene points and the 2D image points, improvements may be attained by performing non-linear optimization. Given that the optimization is tied to the camera pose, this step is often referred to as *Pose Refinement*. An algorithm that is very often used for Pose Refinement is the *Levenberg-Marquardt* algorithm. This is a way of solving the non-linear least squares problem in (2.5), but in this case for one camera and it's 3D-2D point correspondances, by introducing the usage of a combination of an adaptive damping factor and an approximation of the 1st order Taylor series [11]. DLT and Pose Refinement are both used in algorithms known as *Perspective-n-Point* algorithms[3], which have been used in this project.

2.2 Deep Machine Learning

2.2.1 Training and network architecture

To differentiate between machine learning algorithms, the two typical training regimes for a network are separated as *supervised* or *unsupervised*. The first method consists of giving the learning network both the input and the wanted target output from the network. This means that the network parameters are changed and optimized such that the given input yields a result closely resembling the wanted target output. *Unsupervised learning* only uses sets of non-labeled input [11].

With machine learning, an important step is to categorize the available data for training and testing. The goal with a trained deep neural network (DNN) is that it should produce good results with input similar to the training data. After training, a validation subset of all of the available data (different from the training data) is used to evaluate if the network can produce good results with input it has not been exposed to before. If the output, using the validation data as input, is not closely resembling the target validation data, there is a risk that the network needs to be retrained with other parameters or other training data to generalize better to the given task [11].

In order to understand how a DNN is trained to compute a wanted output, some of the key components of a network need to be explained. A basic DNN architecture can be seen in Figure 2.6.

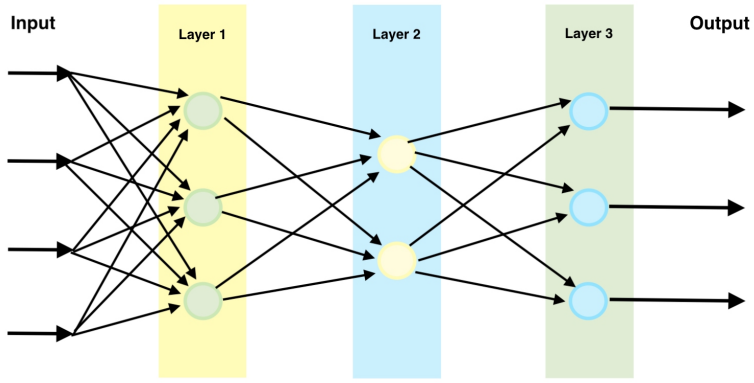


Figure 2.6: A DNN with three fully connected layers.

As seen in the figure, a network can be summed up into having an input, several consecutive layers and an output. The input into a network, x_i, is often a vector of some data. This data is fed into the first layer, which yields a new output that is fed into the following layer and so on. These different connections between input, layers and output are called nodes and are represented as circles in Figure 2.6. The layers consist partly of a weighted sum of the input, which is calculated by multiplying the input with a weight, w_i, and by adding a bias, b_i, shown in equation (2.25) [11].

$$s_i = w_i^T x_i + b_i \tag{2.25}$$

The other part of a layer consists of an activation function, used to remap the newly calculated weighted sum. There are a few different activation functions used with DNNs, but a commonly used one is the ReLU function. This function remaps all negative valued data to zero and keeps positive valued data as is. ReLU is often used for *hidden layers*, i.e layers before the last layer that produces the final output. The activation function is often refered to as $h(s_i)$, seen in equation (2.26) taking the weighted sum as an input and remapping the data to become the new input to the next layer $y_i = x_{i+1}$. As an example for classification tasks, a commonly used activation for the final layer is the softmax function, which remaps the data to probabilities for classes defined within the network, meaning to output the likelihood for a certain class based on the input data. A network could for example be trained to classify cats or dogs from images, where the network classes would be cats and dogs [11].

$$y_i = x_{i+1} = h(s_i) \tag{2.26}$$

The network represented in Figure 2.6 uses *fully connected layers* which means that the nodes in a layer is connected to all the inputs before that layer and that all the outputs from the layer is influenced by all of the earlier layers nodes. This is not the case with convolutional neural networks (CNNs), where just a portion of the input is weighted and summed together to form feature maps that describes the input. These sort of networks are useful to be able to describe images and their properties, as portions of the images are propagated through the network to deduce feature maps portraying information from the image [11].

2.2.2 Loss function and backpropagation

Another essential component of a neural network is the loss function, L, which enables the network to evaluate and enhance the performance of its output. The function compares the N number of outputs from the network, $\mathbf{y_i}$, against the N number of target outputs, $\hat{\mathbf{y}}_\mathbf{i}$. There are several different loss functions useful for different implementations, where an example of a simple loss function is the mean-squared error depicted in equation (2.27). As data might be noisy and the network might do faulty predictions there is a need to describe the optimization of the network as minimizing an expected loss. In equation (2.27) the expected loss is described as $E(\mathbf{w})$, which is the term we want to minimize by altering weights and biases in the network such that outputs of the network closer resemble the wanted target [11].

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{y_i}, \mathbf{f}(\mathbf{x_i}; \mathbf{w})) = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{y_i}, \hat{\mathbf{y}}_\mathbf{i}), \;\; L(\mathbf{y_i}, \hat{\mathbf{y}}_\mathbf{i}) = (\mathbf{y_i} - \hat{\mathbf{y}}_\mathbf{i})^2 \tag{2.27}$$

The alteration of weights and biases is closely related to the loss function and involves computing derivatives of the loss function with regards to the weights. Gradient descent is used to repeatedly alter the weights during training, where the derivatives throughout the network is calculated with the chain rule such that specific layer derivatives starting from the output all the way to the input of the network is derived. The method of calculating these derivatives is called backpropagation. Deriv-

ing how the weigths should be updated based on the backpropagation is done using a so called optimizer. There are several different variants of optimizers but nowadays the most popular one is called Adam. This optimizer is a framework consisting of several optimizers (AdaGrad, RMSProp, etc.) that uses newer techniques of updating weigths in the direction of weight derivatives. This involves having a decreasing learning rate, the influence from backpropagation used to update the weights, and using momentum, introducing an average of the gradients that is decaying over time to the update direction for the weights [11].

2.2.3 Neural Radiance Fields (NeRF)

In recent years, a method that uses machine learning with camera poses and 2D images as input to optimize a scene representation, has gained traction. With this method a non-dynamic scene can be represented as a continuous 5D function, where the input is the viewing directions (θ, ϕ) and spatial location (x,y,z) [2]. These parameters are fed to a deep fully-connected neural network which outputs a volume density and a view-dependent RGB color. This network closely resembles the DeepSDF [16] architecture which represents an objects surface with a continous volumetric field and is able to represent a whole class of shapes.

A sampling strategy is used, discussed later in this section, which is referenced as a vector $\mathbf{t}$, consisting of distances $t_k \in \mathbf{t}$ between the chosen scene bounds t_n and t_f. The distances t_k is used to calculate the 3D points along ray $\mathbf{x} = \mathbf{r}(t_k)$, which is further processed by transforming each position with a positional encoding. A visualization of the sampling along rays can be seen in Figure 2.7.

Figure 2.7: Visual representation of rays being emitted and sampled

This is depicted in equation (2.28), where the variable L is a hyperparameter. A positional encoding means that a heuristic sinusoidal mapping has been performed

for the positions [17] [18].

$$\gamma(\mathbf{x}) = \Big[\sin(\mathbf{x}), \cos(\mathbf{x}), ..., \sin(2^{L-1}\mathbf{x}), \cos(2^{L-1}\mathbf{x})\Big]^T \tag{2.28}$$

The positional encoding, $\gamma(\mathbf{r}(t_k))$, together with the viewing direction serves as input to the network. Transforming the input via positional encoding has been found to better approximate high frequency functions [18], which for the NeRF implementation would equate to being able to represent geometries and texture clearer, seen in the work by Ben Mildenhall et. al [2].

In order to optimize the representation, the output of the network is evaluated by rendering new 2D views of the scene. To render a scene from the Neural Radiance Field (NeRF) it is required to estimate the integral shown in equation (2.29), which depicts the expected color of camera ray $\mathbf{r}(t)$.

$$C(\mathbf{r}) = \int_{t_f}^{t_n} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt, \;\; T(t) = \exp\Big(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\Big) \tag{2.29}$$

The bounds t_n and t_f relate to the scene bounds and are decided for a wanted scale. As mentioned in the paper by Ben Mildenhall et. al [2], the volume density σ can be explained as the probability of a ray terminating at a particle in position x. The function $T(t)$ depicts the probability that the ray travels from t_n to t without colliding with another particle. To render a scene, a camera ray has to be traced for every pixel in the new 2D image, by approximating equation (2.29). This rendered scene is referenced as the output from a "coarse" network, and to achieve a better representation a weighted sampling along rays is done. With this data a probability density function is created that can be sampled again, which is referenced as sampling the "finer" network.

The propagation of the input through the network yields the color and volume densities which can now be represented as a 2D scene which is used in the optimization. In this step the coarse and fine generated scene are both compared to the ground truth scene, where the loss is calculated as the sum of the total squared error between each pixel in the corresponding RGB images from a given view and the pixel values generated by the coarse and fine networks respectively [2].

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

$\mathcal{R}$ are the rays in each batch, $C(\mathbf{r})$, $\hat{C}_c(\mathbf{r})$, and $\hat{C}_f(\mathbf{r})$ represent true colors, and predicted colors from the coarse and fine networks respectively for ray $\mathbf{r}$. A common way to quantitatively assess the novel views from NeRF is by calculating the Peak Signal-to-Noise Ratio (PSNR) for the images. The calculation is performed as depicted in equation (2.30), where the variables f and g describe the real image and the image produced by the NeRF model. The parameters M and N describe the size of image f and g in terms of resolution.

$$\text{PSNR}(f, g) = 10\log_{10}\frac{255^2}{\text{MSE}(f, g)}, \;\; \text{MSE}(f, g) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(f_{ij} - g_{ij})^2 \tag{2.30}$$

From this equation it can be seen that a lesser MSE error between two images yields a larger PSNR value. This means that comparing two images that are pixelwise similar, would give a relatively large PSNR value.

A downside of NeRFs are their complexity in rendering making them impractical for real time rendering at useable framerates. This is due to the fact that to render a novel view with a NeRF a large amount of samples is required, each of which then requires a network inference. In the paper by Alex Yu et. al [8] the solution proposed is to first model the RGB values given from a NeRF as spherical harmonics - basis functions defined on the surface of a sphere, and then densely sample the NeRF after training and tabulate the resulting densities and harmonic coefficients in what the authors call a *PlenOctree* - a sparse voxel based tree structure. As such the model becomes aware of the geometry of the scene, and can therefore ignore parts of the scene that are of no significance and densely pack voxels where the object of interest is located. Also, given that the values for the voxels were tabulated during training of the NeRF model, network inference is not required to get densities as in Vanilla NeRF, and RGB colors are acquired simply by adding the weighted spherical harmonics from a given viewpoint. Furthermore since the rendering procedure remains differentiable, the Octree itself can be optimized by comparing to the ground images. All of this entails that the PlenOctree pipeline is not only much faster than Vanilla NeRF in both rendering and training, it also often leads to more high quality renders because of the extra optimization step.

3
Methods

In order to put together a pipeline that can produce novel views of a chosen object, methods of deducing camera poses and NeRF representation need to be created and implemented. Figure 3.1 depicts a flowchart over the pipeline, where there are some differences between working with synthetic data as compared to real data. The hardware used throughout the project is a computer equipped with 2 NVIDIA® GeForce® RTX 2080 Ti GPUs, supplied by Wiretronic. This chapter will go through the methods used in the project.

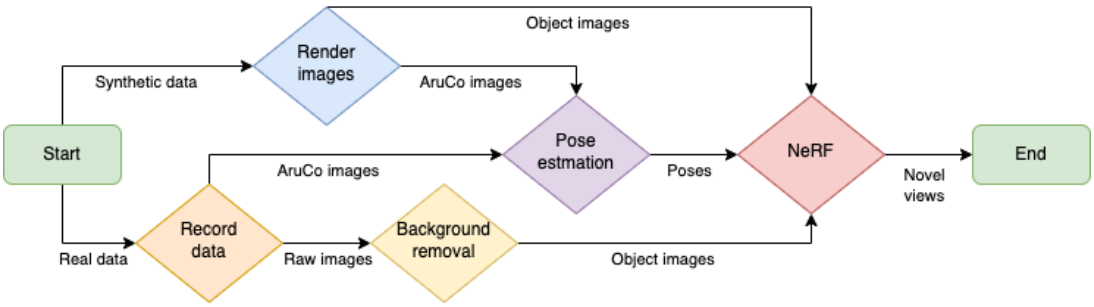


Figure 3.1: Flowchart for the pipeline.

3.1 Data generation

3.1.1 Synthetic data

The need for synthetic data in this project is essential in order to test out various methods and their results. With synthetic data it can be assured that the used input data is not flawed and the risk of misjudging a methods capabilities is minimized. Therefore a dataset from the creators of the first NeRF paper is used [2], specifically a file depicting a yellow lego truck seen in Figure 3.2. This file is useful because of the possibilities to create 2D images from a certain camera pose with the rendering program Blender.

Figure 3.2: A rendered image of the Blender file containing a yellow lego truck.

In other words, with this file it is possible to generate 2D images of the lego truck with associated camera poses. An example of this is shown in Figure 3.3.

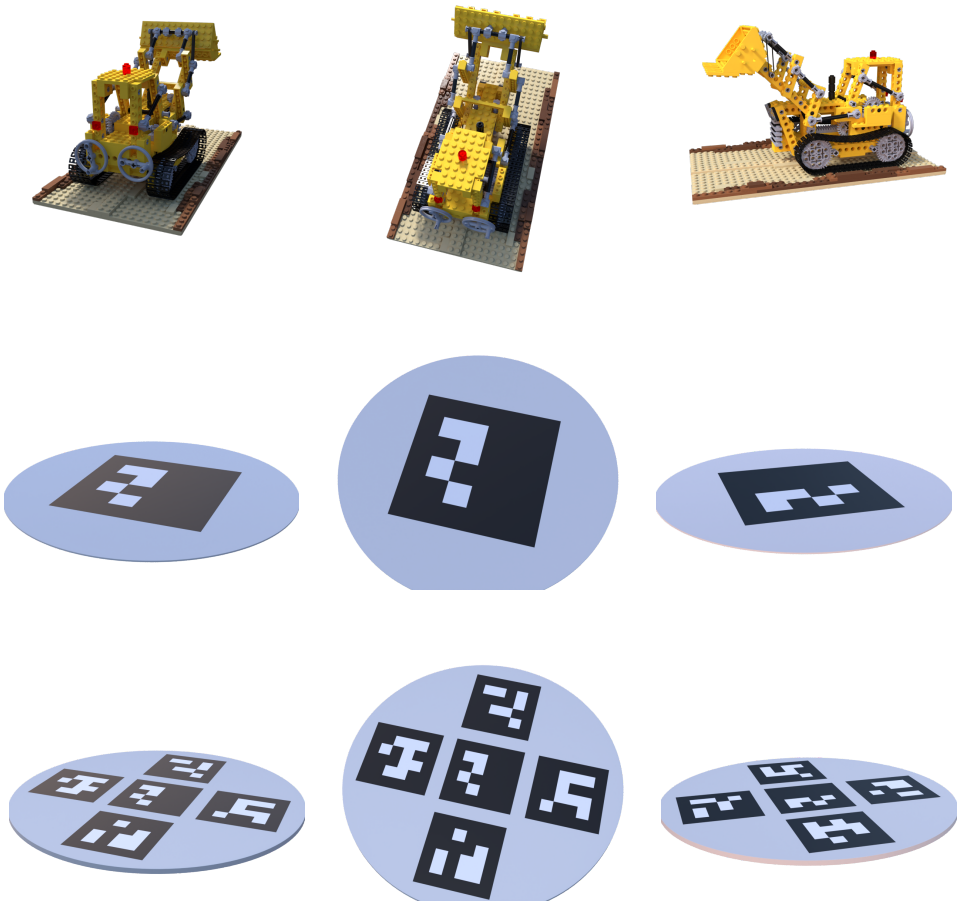


Figure 3.3: 2D views rendered with the same camera poses.

As this is the needed input to train a NeRF model, it is possible to train NeRF models with exact poses and corresponding 2D images. This gives a solid starting ground to qualitatively compare other possible pose estimation methods with the exact poses from Blender. For this project, one hundred random views of the truck have been rendered with associated poses for training the NeRF models. As some of the pose estimation methods depend on visible ArUco markers in the scene, 2D views of a turntable with ArUco markers with the same poses as the truck have been generated.

The idea is that the pose estimation methods will use the images of the ArUco turntable in order to estimate the camera poses, which with a perfect estimation would be exactly the same as the poses for the views of the lego truck. The input to train a NeRF model would then be the estimated poses from the ArUco marker images together with the corresponding images of the lego truck.

3.1.2 Real data

Recording a dataset was done in two separate ways. The first one is done with a setup of three cameras, with different viewing angles. The lighting in the scene is uniform and the background in the setup is white, this is in order to make it easier for the background removal algorithm to deduce the contrast between the wanted object and the background. The cameras are of the same model, namely a Raspberry Pi HQ module. The lenses used for these cameras differ a bit so calibration needs to be performed separately for each camera. With this setup, the used turntable rotates with the help of a stepper motor.

The other setup uses only one camera, the Intel Realsense L515. The cameras in both of these setups are especially useful because they have the property to set a fixed focal length whereas for example with a common mobile phone it is not as simple to set up. Autofocus and other commonly used features that generally speaking improve the quality of the image would change the intrinsic parameters of the camera, and thus invalidate any former calibration. Another difference between the two setups is the darker lighting in the single camera setup. The turntable with a stepper motor is also used within this setup, which yields the possibility to record footage with better precision than with a DC-motor turntable, which was the sort of turntable that was used at the start of the project. Better precision in this case means that the rotational increments of the turntable stay the same during the collection of footage. This is important, as it is needed that the object footage is associated with the same poses as estimated from the ArUco footage. In Figure 3.4 the two setups are visualized. The setups were used to capture 360 degree views of the intended object, a green and yellow Connector housing provided by Wiretronic. To be able to estimate camera poses of the object, the same capture procedure was used with two circular paper cut-outs of one and five ArUco markers placed on the turntable.

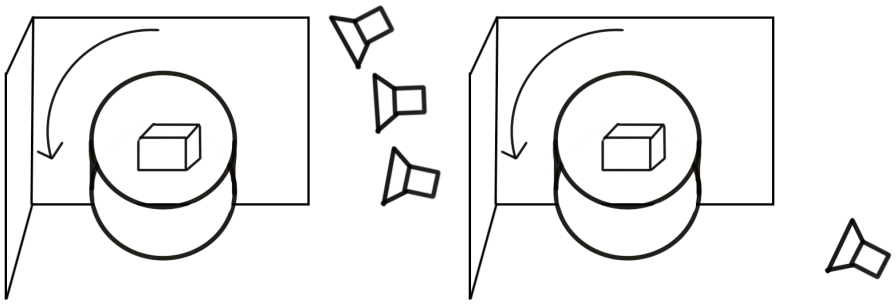


Figure 3.4: A overview of the two used camera setups in the project.

As the NeRF algorithms are intended to deduce the structure of the specific object on the turntable, the background in the images needs to be removed such that only the object itself is visible. As in the latest master thesis at Wiretronic [1], the background removal is done using a deep neural network trained for object detection [19].

3.2 Camera Calibration

There are a few ways in which one can calibrate a camera. For example, if 3D-2D point correspondences are available, one alternative is so set up a DLT problem (similar to (2.22)) and minimize a criteria of minimal reprojection errors. However, as is often the case, such 3D-2D correspondences are not readily known. A method proposed by Z. Zhang [20] is another way to calibrate cameras, and since it was published it has been one of the most used methods. By using a checkerboard (such as a chessboard or similar) printed onto a flat planar surface and taking multiple images of it from different angles and different distances, the intrinsics of a camera can be deduced. However, an issue with using a normal checkerboard is the problem of occlusion, i.e. not all parts of the checkerboard are visible in all images. Given that the space in which images of the real objects is quite small and confined, there was a risk that occlusion could very well occur. Therefore in this project, a Charucoboard was used instead. A Charucoboard is a combination of a checkerboard and ArUco markers, where the white squares of a checkerboard are instead ArUco markers - the differences are shown in Figure 3.5.

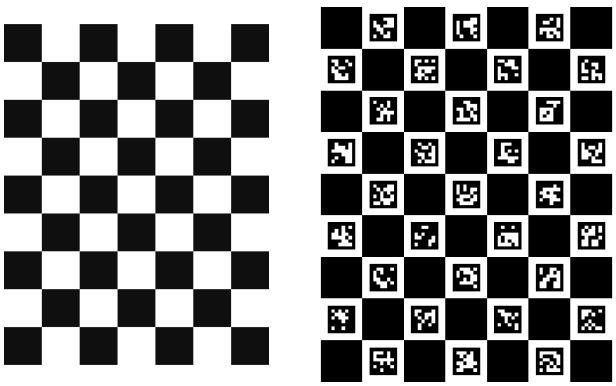


Figure 3.5: A Checkerboard and a Charucoboard of size 7x9 respectively

A Charucoboard is flexible with regards to occlusion due to the presence of ArUco markers. This is because each marker has a unique ID. Therefore if a marker is detected in an image, 2D-2D point correspondences can immediately be established between another image in which that same marker has been detected. Thus not all points need to be seen in all images, since points are connected through the markers unique ID:s. One difference though between ArUco markers versus a Charucoboard is that the Charucoboard does not use the corners of the ArUco markers, instead points where the black squares meet the white border of the ArUco squares are instead interpolated through the detection of the markers. This is because the interpolation of points by utilizing the clear contrast of black and white yields very high subpixel accuracy - leading to better calibration estimation.

When working with the synthetic data, the calibration matrix for the camera was simply taken from the camera parameters in Blender. However, with the real data, a Charucoboard was printed onto a piece of paper and attached to a flat piece of cardboard. Several images of it were then taken by each of the cameras used to capture footage of the objects with which to train NeRF. As explained previously, by using these images of the same Charucoboard, 2D-2D point correspondences were established as shown in Figure 3.6. The images used for calibration are of native resolution from the used cameras in this project.

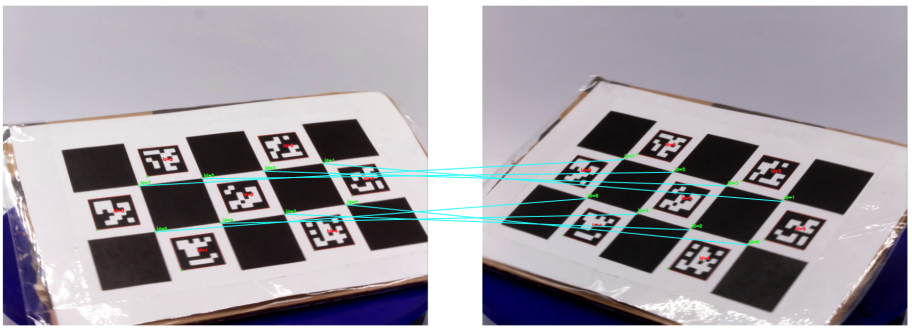


Figure 3.6: 2D-2D matches between two views of the same Charucoboard

The following equations and principles are taken from Z.Zhang [20], and are displayed to give the reader further insight into how cameras were calibrated in this project.

In each image of the board, the board is placed in the origin. Thus z-coordinates of all interpolated coordinates equal 0. Using homogenous coordinates, eq. (2.4) may be utilized, which is expanded in (3.1).

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{r_1} & \mathbf{r_2} & \mathbf{r_3} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{r_1} & \mathbf{r_2} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.1}$$

Here $\mathbf{r_i}$ signifies a column in the 3×3 rotation submatrix of the camera extrinsics, and t is the translation part. Let a point on the board be known as $\mathbf{M}$ and an image

point $\mathbf{m}$, then eq. (3.1) can be condensed

$$\lambda \mathbf{m} = \mathbf{HB} \tag{3.2}$$

From this, a point on the board and an image point are connected through $\mathbf{H}$, known as an *homography*. Equation (3.2) yields

$$\begin{bmatrix} \mathbf{h_1} & \mathbf{h_2} & \mathbf{h_3} \end{bmatrix} = \lambda \mathbf{K} \begin{bmatrix} \mathbf{r_1} & \mathbf{r_2} & \mathbf{t} \end{bmatrix}$$

Leveraging that $\mathbf{r_1}$ and $\mathbf{r_1}$ are orthonormal the following constraints can be established.

$$\mathbf{h_1}^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h_2} = 0 \tag{3.3}$$

$$\mathbf{h_1}^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h_1} = \mathbf{h_2}^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h_2} \tag{3.4}$$

The closed-form solution can now be formed

$$\mathbf{B} = \mathbf{K}^{-T} \mathbf{K}^{-1} \equiv \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} \tag{3.5}$$

$\mathbf{B}$ is symmetric, thus the vector (3.6) may be acquired.

$$\mathbf{b} = \begin{bmatrix} B_{11} & B_{12} & B_{22} & B_{13} & B_{23} & B_{33} \end{bmatrix} \tag{3.6}$$

Now, using the homography constraints in (3.3),(3.4) and letting the ith column vector of $\mathbf{H}$ be $\mathbf{h_i} = [h_{i1}, h_{i2}, h_{i3}]^T$, (3.7) is formed.

$$\mathbf{h_i}^T \mathbf{B} \mathbf{h_j} = \mathbf{v_{ij}}^T \mathbf{b} \tag{3.7}$$

This can now be rewritten as two homogenous equations as shown in (3.8).

$$\begin{bmatrix} \mathbf{v_{12}}^T \\ (\mathbf{v_{11}} - \mathbf{v_{22}})^T \end{bmatrix} \mathbf{b} = \mathbf{0} \tag{3.8}$$

By using n number of images of the board, n such equations are concatenated to get(3.9)

$$\mathbf{Vb} = \mathbf{0} \tag{3.9}$$

Similar to before, Singular Value Decomposition can now be performed and the solution is extracted from the vector that corresponds to the smallest eigenvalue. Finally, the result can be refined using non-linear optimization as explained in (2.5), with the difference being that the instrinsics $\mathbf{K}$ is now a also a parameter to be optimized. For each of the cameras, around 20 images of the board were taken.

3.3 Camera Pose Estimation

3.3.1 Single ArUco marker

Some of the subfigures shown in Figure 3.3 depict the rendered 2D images of a turntable with a single ArUco marker on top. With only having one ArUco marker

present in the view, it is possible to deduce the camera pose for the synthetic data. With the ArUco marker footage from the two camera setups, *Single* and *Triple*, camera poses could also be estimated for the real data. This method is possible due to the python library openCV [3] which contains multiple implementations of important Computer Vision methodologies.

Firstly the marker in an image has to be detected, where the sort of ArUco marker that is present needs to be specified. Indeed there are multiple different appearances of ArUco markers. Therefore whichever type is to be detected needs to be explicitly specified. Given that the markers are put on a white background, a contour step is introduced where the clear contrast of black and white yields the borders of the markers. Because of this, false positive detections of markers can be ruled out. Then the four corners of the marker are detected and refined using subpixel interpolation. Finally, by specifying the true length of the marker sides, 3D points coordinates with z = 0 are created which are connected to the corresponding pixel values of the corners. From these connections the pose of the marker relative to the camera is solved using a Perspective-n-Point algorithm. This algorithm tries to minimizes the reprojection error for the 2D-3D correspondences by optimizing the rotation and translation of the camera poses. The specific Perspective-n-Point algorithm used in this project is based on Marquardt-Levenberg optimization.

After the poses have been deduced with the PnP algorithm, a joint pose refinement is performed by using bundle adjustment. A minimization of the reprojection error is done iteratively and the used solver is called the trust region reflective algorithm. The method computes different regions of the solution and bounds the final result until a convergence tolerance is met [21].

3.3.2 Multiple ArUco markers

Another method of estimating camera poses in this project uses synthetic and real images of the turntable with multiple ArUco markers placed on it. This method is similar to the single marker method, with some exceptions. A clear distinction with this method is that it deals with more markers, giving more information to the algorithms in the form of more detected corners. The refined corner positions are coupled with a set of 3D points which are hardcoded with this method, such that each found corner represents a 3D point. The 3D points being hardcoded just means that the coordinate system for the points are set at $z = 0$ and centered around the origin. The 2D-3D correlation is fed to a PnP algorithm implemented in openCV. This method itself uses Marquardt-Levenberg refinement, yielding camera poses that are more precise. The poses from the PnP method is also refined using bundle adjustment. An important note is that the Perspective-n-Point algorithm yields the pose of the marker/markers with respect to the camera. However, to train a NeRF, camera poses with respect to the marker/markers are required. Therefore, for both the single marker method and the multiple marker method, each pose from PnP *after* Bundle Adjustment needs to be inverted. An arbitrary camera pose of dimension 3×4 is inverted as shown in (3.10).

$$\begin{bmatrix} R & t \end{bmatrix}^{-1} = \begin{bmatrix} R^T & -R^T t \end{bmatrix} \tag{3.10}$$

For further insight in the ArUco marker methods, pseudo code for the two algorithms are presented in the Appendix section.

3.3.3 COLMAP

The COLMAP software was used to estimate the poses for the synthetic and real datasets composed of 2D images. This software is capable of creating a 3D reconstruction of a scene comprised of 2D images, using Computer Vision methods such as SfM, RANSAC, Bundle Adjustment and MVS [5] [6]. The methodology used in this project with COLMAP can be summarized into a number of stages:

1. Use SIFT with the dataset to retrieve image features.
2. Match features against each other and perform geometric verification.
3. Use SfM to reconstruct the motion of dataset, i.e the camera poses for the dataset.

An important difference between feature based methods, such as COLMAP, and marker based methods with known true measurements, such as ArUco, is scale. Indeed scale is not a deterministic parameter when trying to estimate the pose of camera from images - therefore a COLMAP estimate is often very close to the true pose *up to scale* within a given reference frame, provided that enough features can be extracted between images. Thus training a NeRF by treating raw COLMAP poses the same way as with Blender poses would not represent a fair comparison, since their relative positions with regard to the object (in the synthetic case the yellow Lego truck) are completely different. Therefore before any NeRF training is performed with COLMAP poses, a transformation is performed to adhere to the same reference frame as Blender. By treating the positions of the cameras as a point cloud, a point cloud registration algorithm can be used that finds the rigid transformation that transforms the COLMAP estimates to appear as similar to the Blender poses as possible. The algorithms used in this case are *Coherent Point Drift* (CPD), which predicts a preliminary transformation, which is then refined using the *Iterative Closest Point* (ICP) algorithm.

3.4 NeRF implementation

A Tensorflow implementation of NeRF made by Ben Mildenhall et al. [2] was used in this project. Naturally, the architecture of the deep learning network used in this implementation also comes from the work done by Mildenhall et al. [2]. An overview of the architecture can be seen in Figure 3.7.

The network is comprised of 8 fully connected layers with 256 channels using the ReLU activation function, depicted by black arrows. The green and red arrows respectively depict no activation and sigmoid activation. The 9th layer is concatenated with the input viewing direction, as seen by the blue box $\gamma(d)$. The input to the 1st layer is the positional encoding $\gamma(x)$ which is also concatenated with the 5th layer. The output from the network is a volume density, σ, and the released RGB radiance at position x.

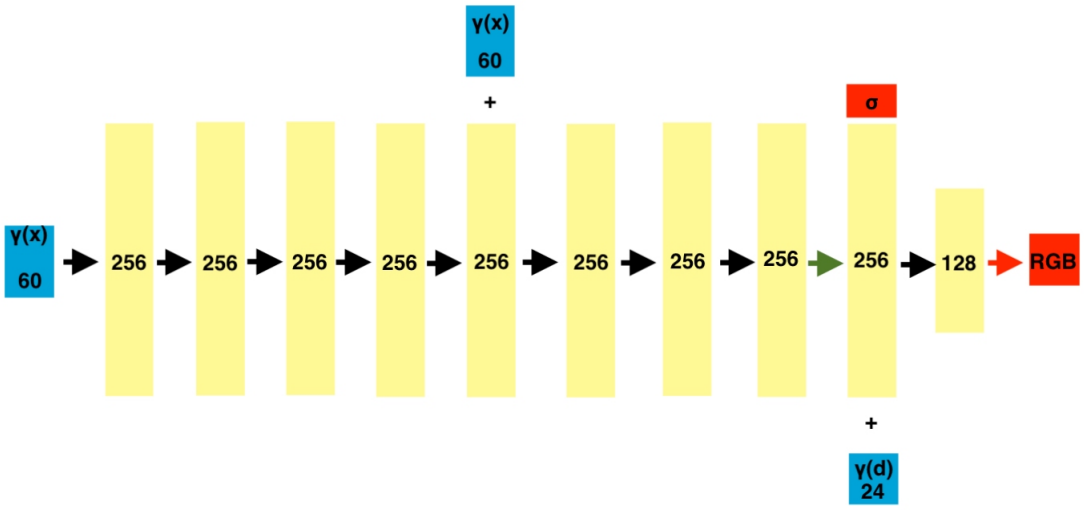


Figure 3.7: The architecture used for NeRF models from the work of Mildenhall et al. [2]

Some alteration of the NeRF implementation was needed for the datasets recorded with the *Triple* camera setup. This is because the three cameras needed individual calibrations, where the found focal lengths for the cameras are different. This is not the case with the original implementation, as it uses the same focal length because the method uses images from Blender (where one camera is used). Therefore a alteration was made such that depending on the image used for training, validation and testing, the correct focal length was used with the image. Using the *Single* camera setup needed no alteration as it uses one camera, i.e the focal length is fixed while training the NeRF model.

Training a NeRF model with the implementation consists of:

1. Choosing a dataset type (Synthetic data or real data), such that the implementation that takes different cameras in account is used.
2. Select appropriate near and far bounds for the ray sampling with NeRF. This is essential for training with real life data, as the scale of the poses differs a lot from the used Blender file. Values of near and far bounds need to be adapted to the diameter of the sphere that makes up the camera poses. An approximation is done by investigating estimated camera poses for the used dataset.
3. Training and sampling is done for intervals of 500 iterations, then a validation of the network is performed by propagating a validation camera pose through the network. The output of the network is used to produce an RGB image by approximating the expected color, seen in equation (2.29), for the corresponding camera rays. This is done for 250 000 iterations for the synthetic and real data. Learning rate is decreased during training.
4. After every 50 000 iteration, a set of test camera poses are fed to the trained network to evaluate how well the model can produce novel views of the scene/object. These are camera poses coupled to a RGB image never seen before by the trained model.

Methods, parameters and important metrics used for the synthetic and real datasets used in this project are listed in Table 3.1. The two camera setups are mentioned as *Single* and *Triple*. An important note is that the datasets use different image resolutions and different amount of images for training. This is simply due to the fact the the cameras used in the two setups are different from one another, and thus yield different image resolutions.

Table 3.1: Different datasets used throughout the project.

Synthetic object	**Camera setup**	**All images**	**Train images**	**Image resolution**	**Pose method**
Lego	-	400	100	800 x 800	Blender
Lego	-	400	100	800 x 800	Sin. ArUco
Lego	-	400	100	800 x 800	Mul. ArUco
Lego	-	400	100	800 x 800	COLMAP
Real object	**Camera setup**	**All images**	**Train images**	**Image resolution**	**Pose method**
Connector housing	*Single*	32	24	720 x 720	Sin. ArUco
Connector housing	*Single*	32	24	720 x 720	Mul. ArUco
Real object	**Camera setup**	**All images**	**Train images**	**Image resolution**	**Pose method**
Connector housing	*Triple*	96	72	896 x 896	Mul. ArUco

4
Results

In this chapter, the evaluation of the camera pose estimations and the novel view generation with the chosen NeRF implementations are shown.

4.1 Camera pose estimation

4.1.1 Synthetic data

4.1.1.1 ArUco marker methods

The input to the camera pose estimation methods are the datasets of synthetic ArUco images rendered with Blender, which are visualized in Figure 3.3. The resulting camera poses of the methods can be assessed visually in Figure 4.2, where true camera poses from Blender are also present. A numeric assessment of the poses is done by reprojecting the 3D marker points to 2D with the calculated poses. The distance between the projected and the initially detected 2D points can be seen in Figure 4.1. The visualizations in this figure might be misleading, as the left image depicting the single marker results seem noisier than the multiple marker result. In fact, the scale of the y-axis is 10^{-7} for the single marker result compared to the scale 10^{-6} for the multiple marker result.

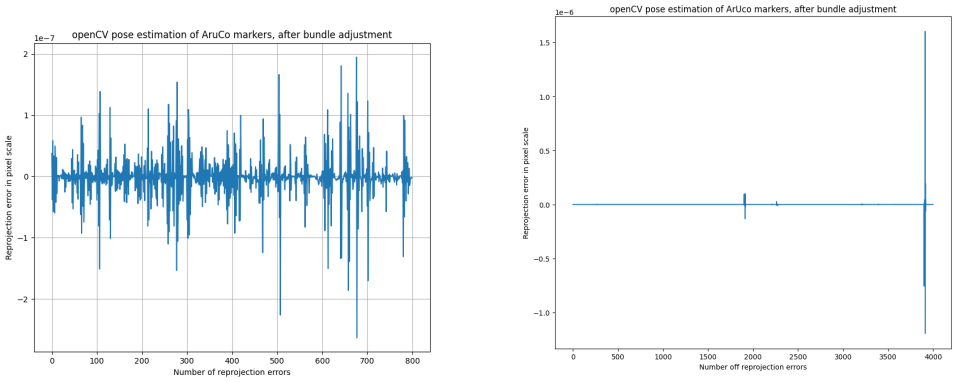


Figure 4.1: Reprojection errors of 3D points after bundle adjustment, for single and multiple marker methods respectively.

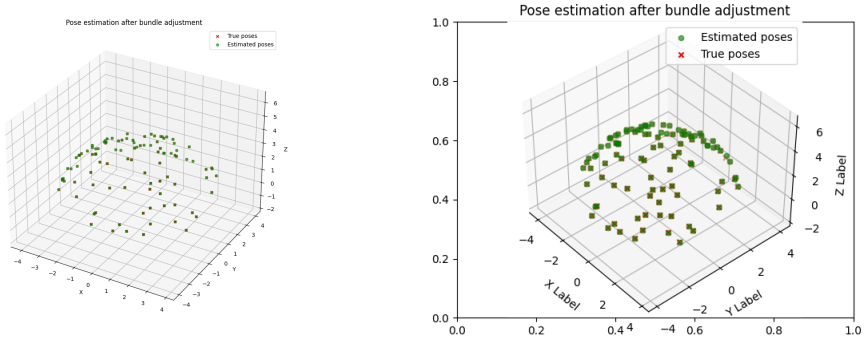


Figure 4.2: 3D plots of estimated camera poses obtained after performing bundle adjustment, for single and multiple marker methods respectively.

In Table 4.1, for both the single marker and multiple markers, the maximum and the mean error of the euclidean distance between the estimated - and true poses as well as the difference in Euler angles w.r.t to the x, y and z - axis are shown. True poses in this case is a reference to the poses from Blender.

Table 4.1: Pose estimation error for synthetic data using ArUco marker methods.

Method	**Max dist.** [m]	**Mean dist.** [m]	$\theta_{\mathbf{x}}[^\circ]$	$\theta_{\mathbf{y}}[^\circ]$	$\theta_{\mathbf{z}}[^\circ]$
Single	0.0167	0.006	-0.0235	-0.0091	-0.0135
Multiple	0.1490	0.012	-0.0335	-0.0106	-0.0156

4.1.1.2 COLMAP

Estimation of camera poses using COLMAP was done using the images of the lego truck as opposed to the ArUco marker methods, which uses images with markers. This is because the COLMAP method relies on SIFT and the possibility to deduce features in images, where using images of ArUco markers likely would yield fewer usable features. The resulting poses are visualized in Figure 4.3, taken from COLMAPs own GUI.

Figure 4.3: Camera poses visualized in the COLMAP application, where the left image shows the most common result and the right image shows the most accurate result.

A common occurrence with the COLMAP software was that some camera poses could be deduced as clearly wrong, just by visual inspection. This result can be

seen in the left image of Figure 4.3, where a "hole" in the sphere of camera poses can be seen. The better result was achieved by altering parameters in COLMAP, which for example was parameters related to minimum amount of inliers for matching and allowing two-view feature tracks in triangulation. The lastly mentioned change can, according to the COLMAP authors, yield more 3D points in triangulation and therefore refine the stability of the model [22]. The better result was hard to replicate in different sessions of camera pose estimation, where the common mentioned result often was the result from the software.

In Figure 4.4, the appearances of the COLMAP estimates compared to the Blender poses before and after CPD+ICP are shown. Table 4.2 shows the pose estimation error, where the results come from a comparison between COLMAP estimated poses and ground truth Blender poses.

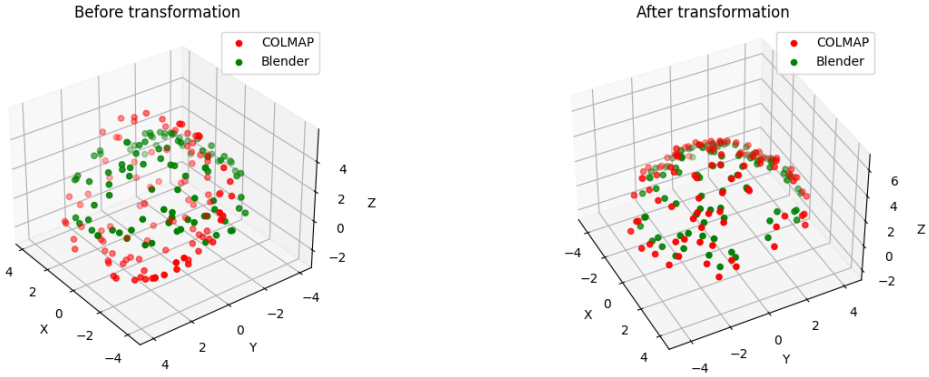


Figure 4.4: Transformation of COLMAP pose estimates to Blender reference frame

Table 4.2: Pose estimation error for synthetic data using COLMAP.

Method	**Max dist.** [m]	**Mean dist.** [m]	$\theta_{\mathbf{x}}[^\circ]$	$\theta_{\mathbf{y}}[^\circ]$	$\theta_{\mathbf{z}}[^\circ]$
COLMAP	0.622	0.334	0.119	0.004	-0.128

What may immediately be apparent is the large discrepancy in the translational position of the COLMAP estimates, but the rotational differences are on par with the ArUco methods. This means that COLMAP has been able to accurately estimate the viewing directions of the cameras, but the ambiguity in scale still remains. In the real world, backing up and simultaneously zooming in with a camera can make the subject appear virtually the same, but the background would change. However, in these images, there is no background. On top of this, COLMAP is unaware of the true calibration of the cameras, so it tries to estimate it online. Therefore COLMAP becomes restricted with regard to how accurately it can estimate how close the object is to the camera in each image since it does not know the true focal length of the camera. All of this entails that the COLMAP estimates may appear worse than they actually are in a sense, given how it can still produce poses that are somewhat correct, up to scale, with the little amount of information it has at its disposal.

Another pose estimation with COLMAP, using the synthetic dataset, was done using the calibrated camera directly from Blender as input to COLMAP. Unfortunately, no result could be achieved where camera poses are deduced from all lego images, which yields a similar result to the left image in Figure 4.3. Inspection of feature matches in these images show that similar features on both side of the truck yield faulty matches, which produces contradicting scene information to COLMAP. Using a dataset consisting of an object that is more asymmetric would probably achieve better results.

4.1.2 Real data

The input to the camera pose estimation methods are the datasets of images collected with the setups mentioned in chapter 3.1.2, where the datasets are shown in Table 3.1. For the setup with one camera, the resulting camera poses are visualized in Figure 4.6 and the reprojection errors using these poses can be seen in Figure 4.5. This dataset consists of 32 images of multiple ArUco markers and 32 images of a connector housing, recorded with the same alteration of rotation for each image. A similar result is achieved with this camera setup, using images of a single ArUco marker and coherent pose estimation method.

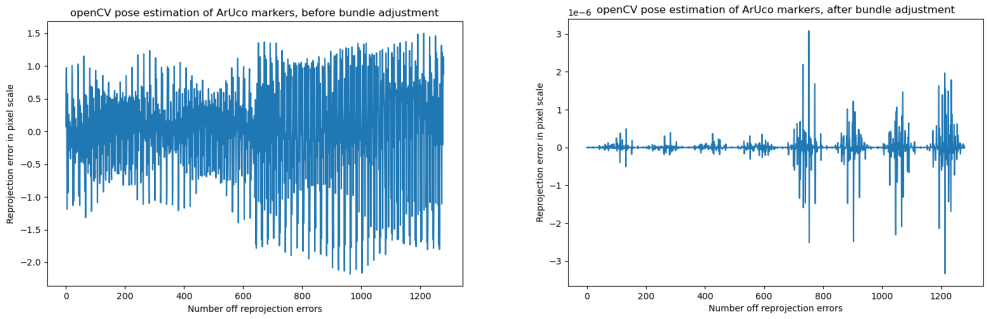


Figure 4.5: 2D to 3D reprojection error before and after Bundle Adjustment.

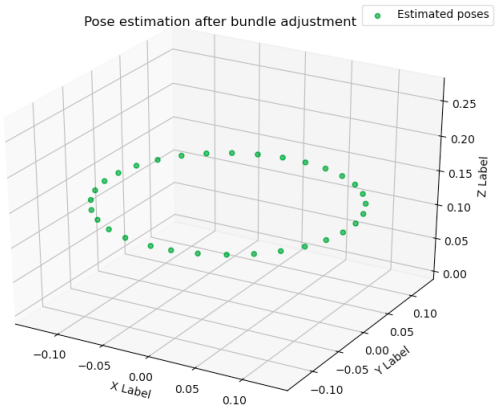


Figure 4.6: Pose estimation of 32 images taken in a 360 degree view with a static camera mount.

The reprojection error before and after performing Bundle Adjustment for the camera poses, are shown in Figure 4.5. The reprojection error with the altered camera poses is decreased by a magnitude of 10^{-5} with the bundle adjustment.

A dataset was recorded with the setup consisting of three cameras. The footage only contains images of multiple ArUco markers, as the previous result of the two methods were similar. This data consists of 96 images, 32 images per camera. These cameras are denoted as *VB1, VB3* and *VB4*. 192 images were taken in total, where the first half make up a scene with multiple ArUco markers and the other half of the same scene with a green and yellow connector housing. The estimated camera poses of the 96 ArUco marker images are visualized in Figure 4.7.

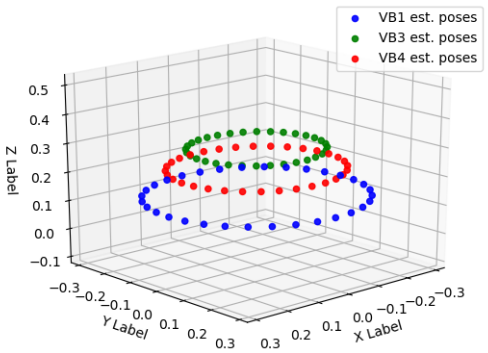


Figure 4.7: Pose estimation of 96 images taken in a 360 degree view with three static camera mounts.

The best camera pose estimation that was achieved, using COLMAP, is visualized in Figure 4.8. Obejct images from camera *VB1* in the *Multiple* camera setup was used. The poses that are estimated form a half circle and there are only 12 poses estimated, despite that there are 32 images given to COLMAP that form a 360 degree view of the object. Because of this, COLMAP poses were not used to train a NeRF with real data, given that the result would inevitably be very poor.

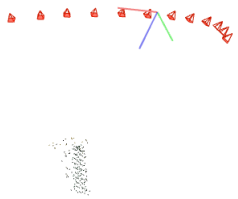


Figure 4.8: Camera pose estimation of dataset with camera setup *Multiple*, only using images from one of the three cameras, using the COLMAP software.

The found SIFT features in these object images are scarce, seen in Figure 4.9. These are images taken from COLMAP's GUI, that shows the found features in the images (red dots) and their coherent feature-matches (the green lines).

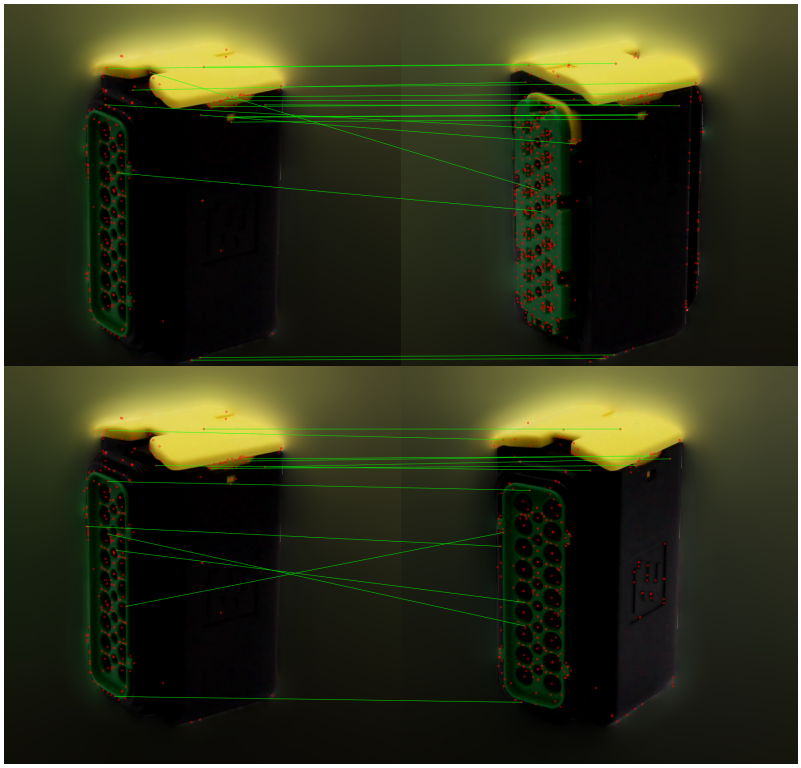


Figure 4.9: Feature matches for different views on a connector housing.

Few features are found in the black part of the connector housing and the ones found are hard to discern and properly match against each other. The top image in Figure 4.9 displays two images of the connector that are rotated approximately 180° compared to each other. These images yield 18 feature matches as compared to 13 matches of connectors with similar view, seen in the bottom image of Figure 4.9. This behaviour is not favorable as it is wanted to find more matches in images with similar viewpoint. This dataset is hard to use with SIFT matching as several features are very alike and uniform, and concurrently wrongly matched against each other. The top view in Figure 4.9 show several matches that are wrong, most probably matched because the orientations of the connector are similar and as such features are found that have similar lighting conditions and colour. The bottom view shows the importance of the lighting in the scene, as the left connector does not find any features in the black area whereas the right one finds a few features which is visible due to the lighting.

4.2 NeRF Renders

4.2.1 Synthetic data

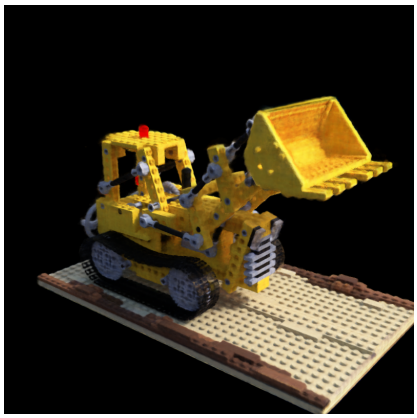
The NeRF implementation by Mildenhall et. al [2] was used to train a network with the dataset consisting of lego truck images together with the included poses taken from the camera parameters in Blender. The network was also trained with the same images again but using camera poses estimated using the ArUco marker methods and the COLMAP software. A visual comparison of novel views can be seen in Figure 4.10.

(a) Blender render

(b) NeRF render using exact poses from Blender

(c) NeRF render using poses estimated with single ArUco marker

(d) NeRF render using poses estimated with multiple ArUco markers

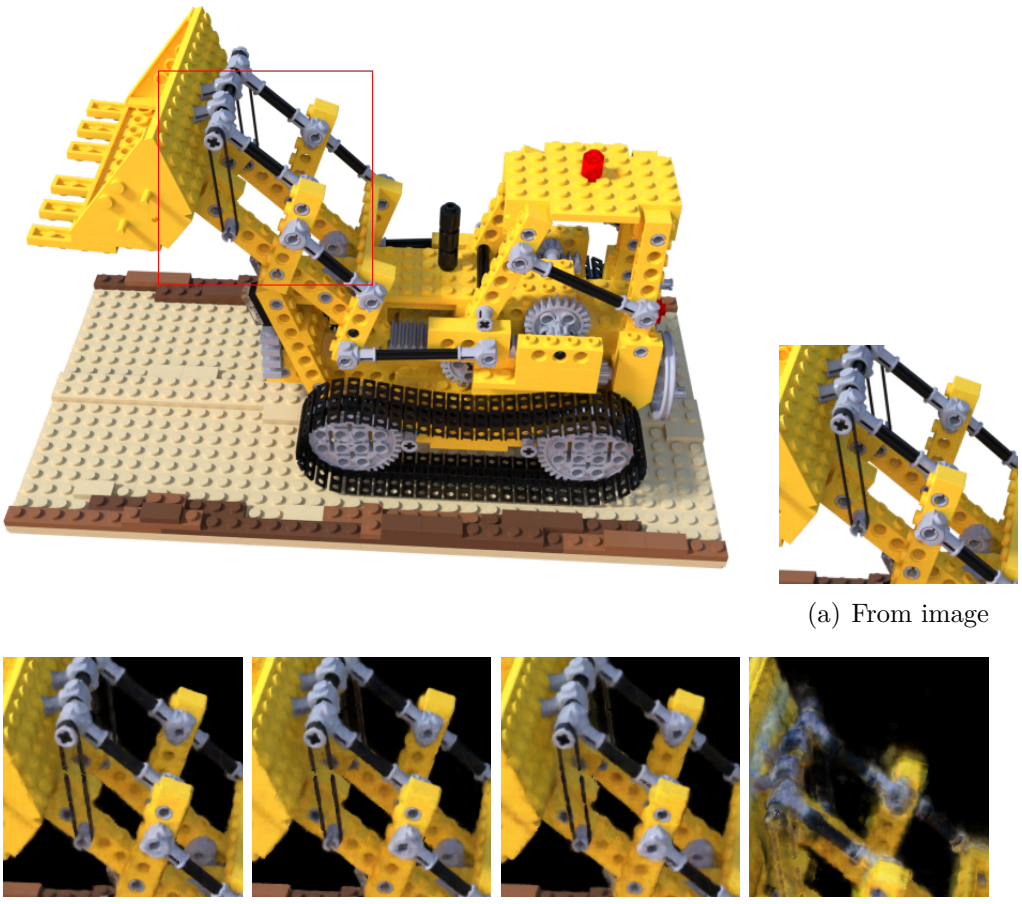


(e) NeRF render using poses estimated with COLMAP

Figure 4.10: Comparison between original render from Blender, NeRF render using exact poses from Blender as well as NeRF renders using the different methods of pose estimation.

The novel views are each computed using a network that has been trained for 250

000 iterations, which took 26-27 hours, to complete using one of the provided GPUs. From these images, it is clear that using either a single marker or multiple markers yields poses that allow NeRF to generate views of nearly identical fidelity as compared to the true poses. Using COLMAP poses, a somewhat satisfactory result can be seen as well with regard to picture quality. However, there is a distinctive shift in the translational position of the truck in COLMAPs result. This is because the network has encoded positional information of the object with respect to the cameras using poses that are too far away from the true poses. Subsequently, when asked to generate a novel view of the truck, the truck will appear to have moved. Given that NeRF is supposed to be able to generate high fidelity novel views of a scene, how well the estimated poses work with NeRF can be gauged by comparing finer detail of the truck in another view as rendered using the different pose estimation methods. This is shown in Figure 4.11.



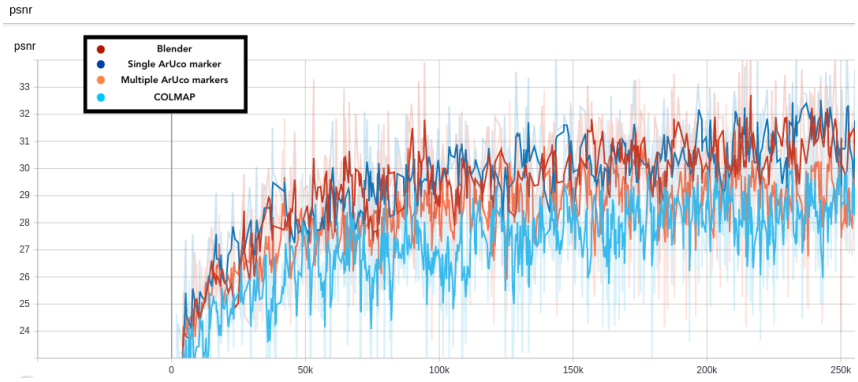
(a) From image

(b) Blender poses (c) One marker (d) Multiple markers (e) COLMAP

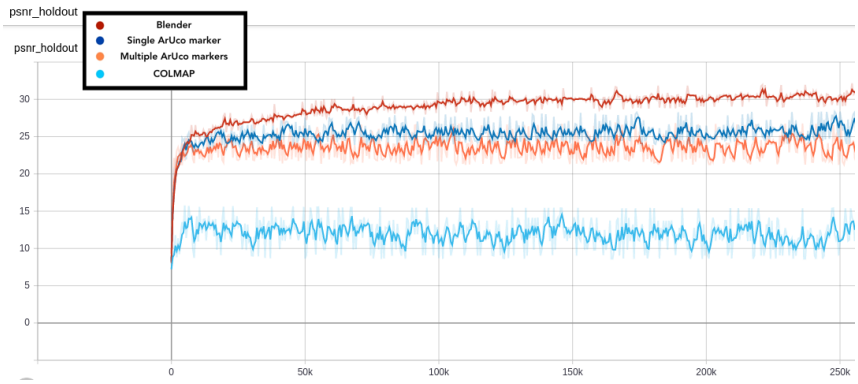
Figure 4.11: Comparison of detail in cropped region of the image

Here we see a much larger difference between the different renders. Both marker methods yield detailed views that closely resemble views with the true Blender poses. The novel views with COLMAP are not as detailed, where artifacts and visual blemishes are apparent. Also, just as in Figure 4.10, there is a shift in the

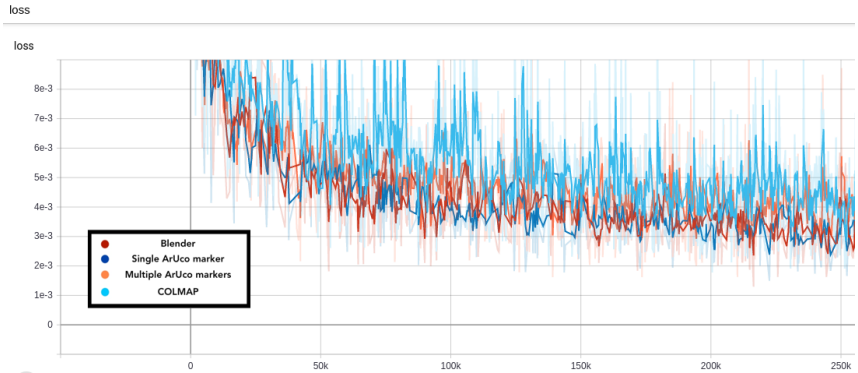
translational position of the frame. Another way of measuring the resulting views is by numeric evaluation, where metrics for training sessions with NeRF models and usage of corresponding synthetic datasets can be seen in Figure 4.12. The figure depicts graphs that correspond to results with usage of different camera poses.



(a) Training PSNR



(b) Validation PSNR



(c) MSE loss

Figure 4.12: Numeric evaluation of NeRF training, where training PSNR, validation PSNR and MSE loss is shown rowwise.

As seen in Figure 4.12 (b), the validation PSNR with COLMAP poses compared to Blender poses, indicate that there is a large difference between novel views. The same figure also shows that there are minor differences between novel views for

NeRF models trained with ArUco marker methods.

4.2.2 Real data

Using the dataset recorded with the single camera setup and poses estimated with the multiple marker method, a NeRF model was trained for 250000 iterations. 75 percent of the images were used for training, 12.5 percent for validation and 12.5 percent for testing. Cropped novel views of the green and yellow connector housing can be seen in Figure 4.13. These were created by propagating camera poses through the trained network, which are novel views as the poses coheres with the test images never seen by the network.

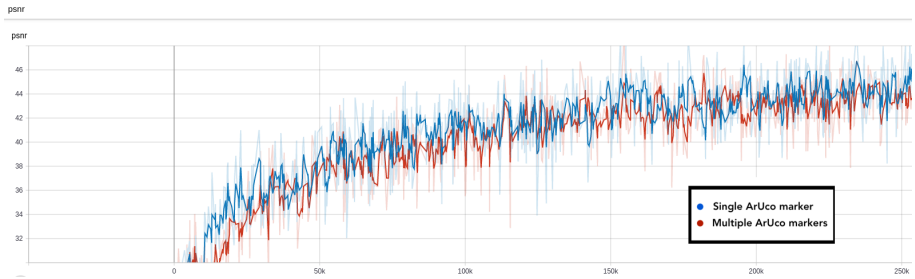
(a) Blurry edge (b) Detailed holes (c) Cut-out (d) Blurred edge

Figure 4.13: Image from test set and Figures (a) - (d) showing detailed areas of novel views.

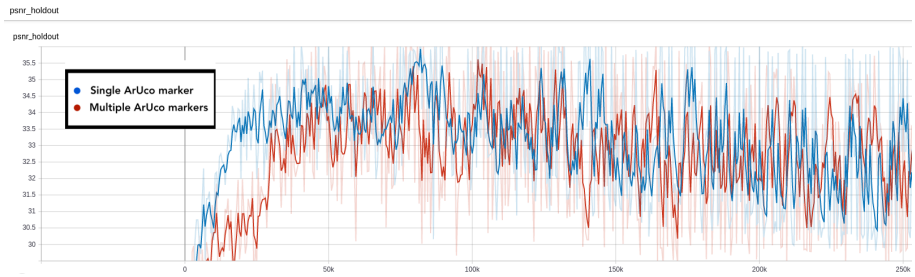
The results seen in Figures 4.13 (a) and (d) show that there are a few edges on the object that are blurry. It can also be seen that details of the object have been captured with high fidelity, such as detailed rectangular and circular holes in Figure 4.13 (b) and a cut-out in the plastic hull in Figure 4.13 (c).

A numeric evaluation of training sessions for NeRF models and usage of corre-

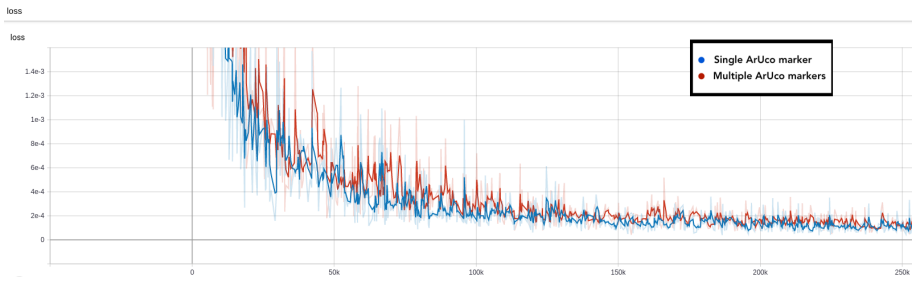
sponding real datasets can be seen in Figure 4.14. The figure depicts graphs that correspond to a result with usage of the *Single* camera setup, where poses are derived with the multiple and single ArUco marker methods.



(a) Training PSNR



(b) Validation PSNR



(c) MSE loss

Figure 4.14: Numeric evaluation of NeRF training, where training PSNR, validation PSNR and MSE loss is shown rowwise. The blue and red graph corresponds to novel view results coherent with usage of the multiple marker poses and the single marker poses.

With the dataset recorded with the *Triple camera setup* and poses estimated with the multiple marker method, a NeRF model was trained for 250000 iterations. 75 percent of the images were used for training, 12.5 percent for validation and 12.5 percent for testing. Novel views of the green and yellow cabelhouse can be seen in Figure 4.15. As the novel view result is similar using camera poses from the multiple marker and single marker methods, it was decided that this dataset was trained only using poses deduced with the multiple marker method.

(a) Camera VB1 (b) Camera VB4 (c) Camera VB3

Figure 4.15: Figures (a) - (c) showing novel views from the three different cameras in the setup.

A big difference between the novel views from the two camera setups is the illumination of the object. In Figure 4.13 there is a darker tint to the colours compared to the more vibrant colours seen in Figure 4.15, depending on the lighting that was used with the two setups. The novel views from this setup also capture details of the object with high fidelity. More novel view results with this dataset can be found in the Appendix section of this report, in Figure A.1.

Another numeric evaluation was done by training NeRF models with the real data, using the triple camera setup. The poses used with this round of training was introduced to Gaussian noise (random numbers sampled from a normal distribution) to evaluate how the NeRF model handles deviations in both translation and rotation for input camera poses. Three different noise cases were used for the camera poses, resulting in PSNR-values for the novel views of the trained models, which can be seen in Table 4.3. The average direction deviation is computed by measuring a mean of angle errors between the camera rays from the non noisy poses versus the poses with added rotational noise. The average distance deviation means the added noise to the translational part of the camera poses.

Table 4.3: Numeric NeRF results with variations of Gaussian noise added to poses.

Case	**Avg. direction deviation** [°]	**Avg. distance deviation** [mm]	**Avg. train PSNR**	**Avg. val. PSNR**
1	0	2	29.16	11.39
2	0.980	0	29.17	8.67
3	1.133	2	29.47	8.50

A final evaluation of the NeRF model trained with real data was done, by having random poses on a sphere as input to the trained NeRF model. The result gives that the NeRF model quite well handles camera poses that are different from the ones used for training, both in terms of translational and angular difference in the x-, y- and z-axis. The sphere coheres with the camera poses, deduced with the ArUco marker methods, used for the training, validation and testing sessions for the NeRF model. The four random poses that were fed to the NeRF network can be seen in

Figure 4.16 and also the corresponding novel views. These new views show sharp details, albeit the view corresponding to the camera pose close to the top of the sphere has blemishes apparent in the view. In terms of rendering speed, a camera pose is propagated through the model trained with real life images of resolution 896×896 pixels, and a single novel view is then rendered after about 40 seconds.

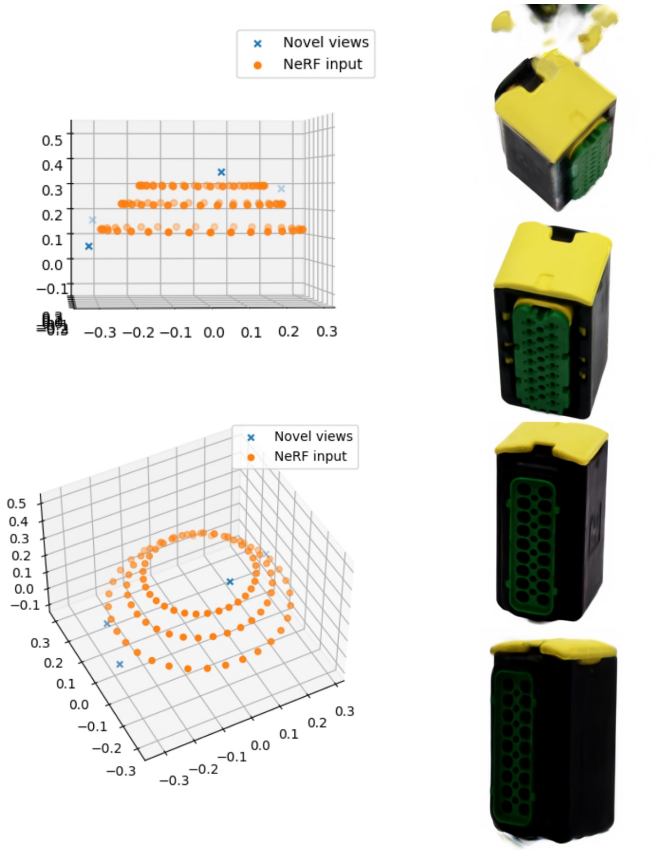


Figure 4.16: The leftmost images depict the training, validation and test camera poses, as orange dots, used for the NeRF model. These plots also show the random poses, as blue crosses, that lie on a sphere having a radius of 0.34 meters, which approximately match the sphere making up the training poses. The novel views resulting from the usage of random poses with the NeRF model are seen in the rightmost images.

5
Discussion

The following section aims to discuss the results of the project and evaluate the used methodologies.

5.1 Pose estimation methods

The camera pose estimation methods using ArUco markers, gave results that showed reprojection errors of magnitude around 10^{-6}-10^{-7} in pixel coordinates. This means that estimated 2D points were matched almost perfectly against ground truth 2D points. As seen in Figure 4.5, the poses firstly deduced by the algorithm has a reprojection error of 1-2 pixel coordinates. Only after the bundle adjustment has been performed, a pixel to pixel accuracy is achieved. Still, it is needed to keep in mind that the ground truth points are found by the detection algorithms of ArUco, which internally relies on the accuracy of the camera calibration, so technically there are no ground truth points. During evaluation of the methods, visual inspection was done by plotting the points in the images of ArUco markers, where points seemed to match corners in the markers. As visual inspection is not a rigorous way of evaluating results, the usage of synthetic data proved to be a really great tool to ensure that results were qualitative. In Table 4.1 the comparison between ground truth poses from Blender and estimated ArUco marker methods can be seen, where the average errors in terms of translation circulates around millimeters of error. Comparing this error to the scale of the lego truck in Blender shows that this is a small error, since the poses of the truck span a sphere with a radius of approximately 4 meters. In terms of rotational errors, a factor of at most $10^{-2}[^\circ]$ could be seen.

However, it must be noted that there are some obvious downsides to the usage of markers. As mentioned earlier, the accuracy of the markers heavily depend on the camera calibration. This means that large errors could fairly easily start to mount right at the beginning if the calibration parameters are not estimated well enough. Moreover, there is the added issue of having to place the markers *in the same place* as the object, or vice versa, since the poses will be with respect to a given origin determined by the markers. Thus unsatisfactory NeRF renders can be achieved if the object is placed far enough off center compared to the markers. This problem is compounded by the fact that our methods rely on having to place the markers, gather footage, remove the markers and then place the object and gather footage again. Therefore, it is difficult to guarantee that the object and the markers were placed in the exact same spot. Another issue with using markers is the critical need for having a clear view of the markers. This means that images that are somewhat

blurry may be inadequate, and images that observe the turntable completely level to it would not see the marker at all. The former would likely yield a wrongly estimated pose, whereas the latter would not yield a pose at all, given that the marker simply is not visible. Despite this, we have shown that a trained NeRF network is generalized enough to be able to render novel views that observe the object completely from the front, provided that poses that are fairly close to the same level are used for training. This can be seen in Figure 4.16.

A similar comparison of the camera poses for synthetic data, is shown in Table 4.2. The table shows the average difference in translation and Euler angles for the camera poses deduced with COLMAP as compared to ground truth Blender camera poses. This result shows a larger difference to ground truth poses compared with ArUco marker results. The transformed COLMAP poses are seen visually in Figure 4.4, where the translational difference to Blender poses is apparent. If the estimation using COLMAP would have resulted in poses closely resembling the Blender poses, a transformation would have been found through CPD+ICP that yields a result closer to the ones of the ArUco marker methods.

As mentioned earlier in the report, the usage of COLMAP with our synthetic data and *Connector housing* dataset to estimate poses had a stochastic characteristic to it. The camera poses estimated with COLMAP, having the lego truck images as input, rarely gave results that accurately could represent the spherical nature of the camera poses. Looking at Figure 4.3 depicting camera poses for the lego truck, there are results where a larger part of the sphere does not contain poses. Further investigation of this figure entails that a number of points, depicting the bucket of the truck, is located on the other end of the truck. Features are matched against each other pair-wise for images, so there is a risk that some features are wrongly matched because there are similar features on both sides of the truck. Example of wrong matches can be seen in Figure 4.9, for the *Connector housing* dataset. This behaviour was also seen in the synthetic lego data, where similar features are found on both of the symmetrical sides of the truck. Therefore using triangulation to find 3D points and deduction of camera poses with these matches might yield results like the ones in Figure 4.3.

Using COLMAP to estimate camera poses for the connector housing dataset did not produce poses that correspond to the 360 degree view of input images. This highlights the most apparent issue of using feature based methods in some feature poor environments, just like our dataset. SIFT simply can not establish enough correct connections between images. Consequently, the estimated poses are inadequate or are downright discarded entirely, which is why only 12 cameras are shown instead of 32 in Figure 4.8. Therefore these poses were never used in conjunction with the object images to train a NeRF model. Another likely reason as to why COLMAP had a hard time estimating the poses of the real dataset, could be that the input images to the method only consists of the object itself. This input is the result of the background removal software, meaning that COLMAP only has the object at hand to deduce features for matching points. The traditional input to the method would be images where all the parts of the image can be used to find features,

giving the method more features which in extension yields more reliable 3D points and camera poses. A viable option could have been to introduce more structure on the turntable in the recording of the footage, such that a possible COLMAP estimation would have more features to yield poses from. Then the poses would be estimated from one recording, with a 360 degree rotated feature heavy scene, and the object footage would be recorded with the white background. This approach was considered but as the methodology would be dealing with the issue that parts of the background would be static, the usage of markers was pursued instead. This input would most likely confuse COLMAP as large parts of the images would be rotated between pairs of images, while at the same time there would be some static features that are placed at the same coordinates in all images.

The results of this project show that a viable method for scenes with poor SIFT conditions, could use ArUco markers as a mean of performing camera pose estimation. Bear in mind, the datasets that we have used might be the worst case scenario of data for general SfM algorithms, such as COLMAP. Using ArUco markers is probably not the most viable option in general settings of camera pose estimation, as compared to the many scene setups the COLMAP software can handle. For the camera setups used in this project the ArUco markers have proven to be a great option for estimating poses.

5.2 Camera pose influence on novel view fidelity

The estimated camera poses for synthetic data, using ArUco marker methods, gave results that closely resemble the poses from Blender. This meant that the marker methods using real data had a good chance of estimating qualitative poses for the connector housing datasets. As there are no ground-truth poses available for the real data, visualizations of the estimated poses and the resulting fidelity of the novel NeRF views was the only way of measuring the quality of the camera poses. The resulting novel views show very qualitative images, seen in Figure 4.13 and 4.15, where several key details of the object have been captured. This result in conjunction with visual inspection of the poses proves that the ArUco marker pose estimation works well in both simulated and real life environments. Earlier results from work done by Wiretronic [1], shows that to deduce novel views of high fidelity with NeRF, highly accurate camera poses are crucial. This is also seen in the results for training NeRF models of real data with noisy poses, seen in Table 4.3. The resulting validation PSNR-values are low, and visual inspection of novel views also prove that the trained NeRF models, queried with validation camera poses, yield novel views that do not represent the object well. The noise added to the poses is relatively small, since the poses of the connector housing span a sphere with a radius of approximately 0.34 meters and the added translational noise on average was 0.002 meters. The rotational noise was on average 1° for rays. This shows that small additions of noise on camera poses is enough to greatly impair results of NeRF models.

The importance of accurate camera poses is also seen in the novel views from NeRF

models trained with poses from COLMAP. The new views of the lego truck, seen in Figure 4.11, and their coherent validation PSNR values, Figure 4.12, compared to the other methods are relatively low. For this case the PSNR value in itself is misleading, as PSNR is a pixelwise comparison of images, and novel views with COLMAP is slightly shifted in a translational manner, compared to ground truth Blender images. By visual inspection of the whole rendered image, these novel views show quality resembling the result with the other pose estimation methods. However, by zooming in on a given area, such as in Figure 4.11, it can be seen that sharpness in details is lost with the usage of COLMAP camera poses. The fidelity of details is kept using data from the ArUco marker methods, which is both seen visually in Figure 4.11 and numerically in Figure 4.12, as validation PSNR-values show a relatively small error compared with values that correspond to usage of Blender poses.

5.3 Single marker compared to Multiple markers

When using the synthetic dataset, the single marker method managed to estimate poses that were closer to the Blender poses, both in terms of rotational and translational error. This in itself made NeRF render novel views with larger PSNR-values for the single marker method, compared to the method using multiple markers. There are few visual differences between novel views using poses from single or multiple marker methods, but as the PSNR value is highly sensitive pixelwise differences might be clearer with PSNR. However, the opposite behavior, at least in terms of render quality, was seen with the real data. The reason for this is not entirely clear. The most probable reason has to do with the placing of the object on the turntable as compared to the placing of the ArUco markers. Currently, there is no obvious point marking the middle of the turntable, which means that the placing of an object is done by eye. Given that NeRF is sensitive to shifts in both translation and rotation, the single marker sheet may have been placed slightly more off center as compared to the multiple marker sheet, yielding somewhat worse renders due to the relative position of the poses compared to the object. Another thing may simply be the stochastic nature of the NeRF network itself, which means that getting a better result with a single marker during another run of training can not be ruled out.

5.4 Novel NeRF views with random poses

The usage of synthetic data to train a NeRF model yielded a network than can produce qualitative images with random poses on a sphere coherent with the training data. The synthetic novel views show high fidelity independent of which camera pose on the sphere is used (not accounting for poses from the bottom half of the sphere). The reason for this is that the NeRF model trained on synthetic data, uses random views from a half sphere with coherent camera poses for training. The NeRF model therefore is generalized to handle random camera views from the same radial distance. This is hard to replicate using real life data, as it would mean that footage of the object is taken with a camera moved around perfectly as a

half-sphere, and subsequently reproducing this exact movement for ArUco marker footage. The dataset that was recorded that closest represented the Blender data, used three cameras with different viewing angels and translations to the turntable. As seen in Figure 4.16, the NeRF model handles poses different from the training poses relatively well. Camera poses propagated through the model, that lie between the three camera angles, produced novel views that are of high fidelity. This is interesting, as it shows that the model is able to generalize well to a smaller amount of angular differences for training images. The biggest issue for this trained network is apparent when using poses that lie close to the top of the sphere, i.e camera poses where the object is seen from above. This is seen in Figure 4.16 where several visual abnormalities can be seen with a top view on the object. The result is reasonable, as the network has not been trained with images and poses that closely resemble the top view of the object. A better result could reasonably have been achieved by training NeRF models using images and camera poses with more angular differences in the z-axis. This would mean that recording footage should be done using more cameras at different viewing angles. The question at hand would then be, how many cameras and views are needed in order to optimize the output of a NeRF model? Likely, there is a threshold were more cameras are redundant as results with less cameras would be similar. For the pipeline used in this project, introducing a fourth camera with a top view of the object could reasonably have yielded a better result. Another option could have been to mount cameras such that the overlapping areas would be smaller, so cameras would cover larger angular differences of the object.

5.5 Important factors for novel view quality

The usage of different cameras in the *Triple* camera setup was at first generalized to using a single approximation of a focal length, used for all of the images with NeRF. We saw that resulting novel views were not as qualitative compared with data from the *Single* camera setup, therefore leading us to experiment with usage of all three different focal lengths with the NeRF implementation. This gave results with similar quality to the usage of the *Single* camera setup. Using different cameras in the context of rendering a novel view, is hard to interpret as new poses to the network does not correspond to a "real" camera with a certain focal length. Therefore, the usage of the earlier approximated focal length was used with the network to produce novel views, seen in Figure 4.16 for random camera poses. These results are of good quality, albeit leaving us with the question how much the used focal length affects the result and whether another methodology for focal lengths could be used when several cameras are present in the pipeline. The resulting novel views can be interpreted as not depending that much on the approximation of focal length, as the novel views are of similar quality compared to results with poses from the test set.

Additional parameters that were important for the NeRF model were the near and far bounds for ray sampling, specifying the interval on the ray where the sampling should be performed. For the synthetic data, this near and far bound were already deduced from the Blender file, corresponding to the scale of the lego truck and coherent camera poses. For our own recorded footage, it was needed to find reasonable

near and far bounds such that sampling is performed in accordance to the scale of the connector housing object and the deduced camera poses from the ArUco methods. Several combinations of near and far bounds were used with NeRF training, where results show the importance of having accurate near and far values. This was especially seen when we went from training NeRF models with data from the *Single* setup to the *Multiple* setup. We trained a NeRF model using the *Triple* setup data, using the same near and far bounds as for the *Single* setup, resulting in poor novel views from the NeRF model. In hindsight, this is reasonable as camera poses from the *Triple* setup spans a larger sphere compared to the *Single* setup data, meaning that rays were sampled at intervals not centered around the object. Trying different combinations of bounds therefore consisted of finding reasonable values such that the object is centered in the sample interval. Interesting ground to cover regarding near and far bounds would be to evaluate if smaller intervals of sampling could be used to yield more precise results, as it would result in finer sampled rays.

6
Conclusion

Using synthetic datasets, the quality of the proposed ArUco marker pose estimation methods could be deduced as satisfactory. The results using real data show that training a NeRF model using poses deduced from ArUco marker images enable the network to render qualitative novel views. The model can produce images coherent with camera poses forming a sphere around the object at hand, albeit novel top views of the object are of lesser quality. We conclude that important aspects with our camera setups, for achieving NeRF models that produce views of high fidelity, are ray sampling intervals, accurate camera poses and well calibrated cameras. Furthermore, we have shown that using ArUco markers is a viable option with datasets that contains few distinct features as an alternative to SIFT-based SfM pipelines. This project has resulted in enhancements of a pipeline meant to produce high fidelity novel views of objects. The result can be used to capture several objects appearances, where usage could be to simply show products from a wanted view, use the trained models to extract 3D models, generate datasets for objects in conjunction with usage of other machine learning applications and possibly AR-related applications.

6.1 Future work

The models trained with footage from three cameras have a hard time producing novel top views of the object. Therefore it would be of interest to experiment with using more cameras or altering the mounting of existing cameras, such that trained models can produce high fidelity novel views independent of chosen camera pose (coherent with half-sphere of training poses). Another topic of interest is the sampling interval of the rays. Currently as recorded objects might be of different sizes, the interval is deduced empirically. Therefore another method that can more accurately bound the sampling interval of the rays to the object itself, based on its dimensions, could perhaps help generalize the pipeline.

Producing a novel view from a camera pose takes about 40 seconds using the models trained in this project. Other implementations of NeRF would be of interest to speed up this process. The Plenoctree implementation could in future work be of interest [8], given the possibility for faster rendering. For example, this could be of use in AR-applications where generating novel views could cohere with the camera pose of a person holding a mobile phone, putting the novel view in context to rotations and translations in the real world. Lastly, an interesting direction of this project would be to deduce 3D models using the trained networks and evaluate whether they provide accurate measurements of the real world object.

References

[1] D. O. Isak Ernstig, "3d reconstruction using deep learning view synthesis", M.S. thesis, Chalmers University of Technology, 2021.

[2] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, *Nerf: Representing scenes as neural radiance fields for view synthesis*, 2020. arXiv: 2003.08934 [cs.CV].

[3] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[4] D. G. Lowe, *Distinctive image features from scale-invariant keypoints*, 2004. DOI: 10.1023/B:VISI.0000029664.99615.94. [Online]. Available: https://link.springer.com/article/10.1023/B:VISI.0000029664.99615.94.

[5] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo", in *European Conference on Computer Vision (ECCV)*, 2016.

[7] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion", *Pattern Recognition*, vol. 47, pp. 2280–2292, Jun. 2014. DOI: 10.1016/j.patcog.2014.01.005.

[8] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, *Plenoctrees for real-time rendering of neural radiance fields*, 2021. arXiv: 2103.14024 [cs.CV].

[9] Y. Jeong, S. Ahn, C. Choy, A. Anandkumar, M. Cho, and J. Park, *Self-calibrating neural radiance fields*, 2021. DOI: 10.48550/ARXIV.2108.13826. [Online]. Available: https://arxiv.org/abs/2108.13826.

[10] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, *The unreasonable effectiveness of deep features as a perceptual metric*, 2018. arXiv: 1801.03924 [cs.CV].

[11] R. Szeliski, *Computer Vision: Algorithms and Applications, 2nd edition*. University of Washington: Springer, 2021.

[12] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers", *Image and Vision Computing*, vol. 76, pp. 38–47, 2018, ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2018.05.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885618300799.

[13] C. Zach, *Computer vision lecture 6 : Supplementary notes*, Dec. 2021.

[14] Y. Chen, Y. Chen, and G. Wang, *Bundle adjustment revisited*, 2019. DOI: 10.48550/ARXIV.1912.03858. [Online]. Available: https://arxiv.org/abs/1912.03858.

[15] C. Zach, *Computer vision lecture 4*, Dec. 2021.

[16] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, *Deepsdf: Learning continuous signed distance functions for shape representation*, 2019. arXiv: 1901.05103 [cs.CV].

[17] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, *Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields*, 2021. DOI: 10.48550/ARXIV.2103.13415. [Online]. Available: https://arxiv.org/abs/2103.13415.

[18] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng, *Fourier features let networks learn high frequency functions in low dimensional domains*, 2020. DOI: 10.48550/ARXIV.2006.10739. [Online]. Available: https://arxiv.org/abs/2006.10739.

[19] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. R. Zaiane, and M. Jagersand, "U2-net: Going deeper with nested u-structure for salient object detection", *Pattern Recognition*, vol. 106, p. 107 404, Oct. 2020, ISSN: 0031-3203. DOI: 10.1016/j.patcog.2020.107404. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2020.107404.

[20] Z. Zhang, "A flexible new technique for camera calibration", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000. DOI: 10.1109/34.888718.

[21] N. Mayorov, *Trust region reflective algorithm*, https://nmayorov.wordpress.com/2015/06/19/trust-region-reflective-algorithm/, 2015.

[22] J. S. et. al, *Frequently asked questions*, https://colmap.github.io/faq.html, 2022.

A

Appendix

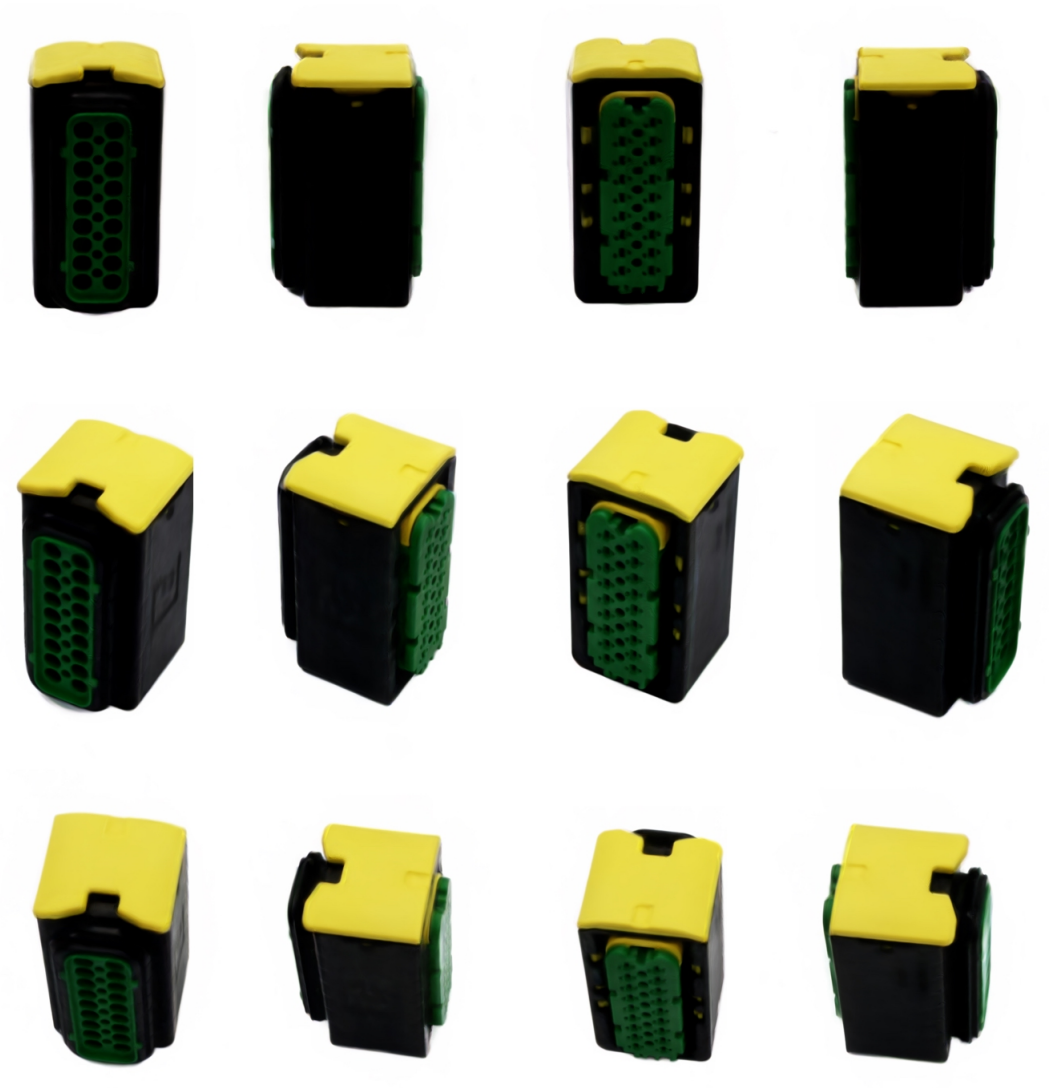


Figure A.1: Novel views of a green and yellow cabelhouse created with a trained NeRF model, trained with input using footage from the *Triple* camera setup and poses estimated using images of multiple ArUco markers.

Algorithm 1 Single Marker Pose Estimation

```
det_params = detection parameters for marker
all_poses = empty array
all_corners = empty array
all_scene_points = empty array
for i, img in images do
    corners, scene_points = detect_marker(det_params)
    R, t = Perspective_N_Point(corners, scene_points)
    pose = concatenate(R, t, axis = column)
    refined_pose = Levenberg_Marquardt(pose, corners, scene_points)
    all_poses[i] = refined_pose
    all_corners[i] = corners
    all_scene_points[i] = scene_points
end for
bundle_adjustment_poses = least_squares(all_poses, all_corners, all_scene_points)

camera_to_marker_poses = empty array
for marker_to_camera_pose in bundle_adjustment_poses do
    R = marker_to_camera_pose.R
    t = marker_to_camera_pose.t
    camera_to_marker_poses[i] = concatenate(R.T, −R.T ∗ t, axis = column)
end for
camera_to_marker_poses → NeRF training
```

Algorithm 2 Multiple Markers Pose Estimation

```
det_params = detection parameters for marker
all_poses = empty array
all_corners = empty array
scene_points = hardcoded array from measurements
for i, img in images do
    corners = detect_marker(det_params)
    R, t = Perspective_N_Point(corners, scene_points)
    pose = concatenate(R, t, axis = column)
    refined_pose = Levenberg_Marquardt(pose, corners, scene_points)
    all_poses[i] = refined_pose
    all_corners[i] = corners
end for
bundle_adjustment_poses = least_squares(all_poses, all_corners, scene_points)

camera_to_marker_poses = empty array
for marker_to_camera_pose in bundle_adjustment_poses do
    R = marker_to_camera_pose.R
    t = marker_to_camera_pose.t
    camera_to_marker_poses[i] = concatenate(R.T, −R.T ∗ t, axis = column)
end for
camera_to_marker_poses → NeRF training
```

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY