Object Pose Estimation with Neural Radiance Fields

Internship report

Jakub Zadrożny

MPRI M1 2020-2021

September 7, 2021

Host institution

Czech Institute of Informatics, Robotics and Cybernetics

Supervisor

Dr. Torsten Sattler

Dates

June – September 2021

Abstract

This work introduces a method, inspired by CosyPose [1], for recovering the 6D pose and 3D shape of multiple objects in a scene captured by a set of input images with unknown camera viewpoints. Crucially, we relax the limiting assumption of CosyPose, namely that precise 3D models of all objects of interest are known beforehand. Instead it is sufficient to provide a few examples of 3D models for each category of objects that the user is interested in. We conduct initial experiments aimed at veryfying the practical applicability of our method. We describe the issues we've encountered throughout our experiments, we also provide solutions to some of the problems and analyze and narrow-down the cause of problems we were not able to fix along with some potential remedies. A partial implementation of our method is available online [1].

[1] `https://github.com/jakubzadrozny/nerfpose`

1 Introduction

The goal of object pose estimation is to detect, identify and accurately estimate the 6D pose (i.e. the 3D position and 3D rotation) of certain objects of interests in a scene. It is a central, long-standing and challenging problem of computer vision. Solving it would have important applications in robotics by allowing robots to plan, navigate and interact with the environment accurately and efficiently. It is therefore a very active area of research, where multiple approaches have been proposed and evaluated throughout the years.

Early works [2] usually relied on gradient-based image features, but they were superseded by convolutional neural networks [3] as deep learning methods were gaining momentum. Most of these learning-based methods focused on estimating the pose of each object independently, given only a single input view. This however is a significant limitation as depth estimation from a single image has inherent ambiguities that may not always be accurately resolved by prior knowledge.

Recently a novel approach dubbed CoysPose was proposed by Y. Labbe et al. [1]. It is designed to work with multiple views and multiple objects from the ground up and it outputs a single, multi-view consistent scene interpretation. A crucial requirement of CosyPose is that it needs access to the 3D models of the exact objects that it will estimate the pose of. This assumption can often be violated in real-world scenarios due to the large intra-class variability of objects, e.g. different types of chairs, which limits the practical applicability of CosyPose.

This work introduces a method, inspired by CosyPose, that relaxes the limiting assumption of knowing the exact 3D models of all objects of interest. Instead we only assume that we have access to 3D models of a few examples for each class of objects. Given several such exemplary models, our method would later predict poses of a large variety of objects.

This can be achieved through the use of Neural Radiance Fields (NeRFs) [4] that represent the scene as an implicit function, implemented via a multi-layer perceptron. Since the relative camera poses are assumed to be unknown, a variant of NeRF is required that is able to render novel views of a scene given only a single input view. This can be achieved by equipping the model with a set of prior experience and conditioning it on an intermediate latent code that describes the shape and general appearance of the rendered object. The NeRF model can later be *inverted* [5] to recover the pose of a novel view relative to some input frame.

Our approach first inverts a pixelNeRF [6] model based on individual input frames to establish relative camera poses and correspondences between object detections. Second, exactly as in CosyPose, errors in object detection are addressed using an object-level matching procedure based on RANSAC. Third, we place all cameras in some world coordinate frame and optimize their position to minimize NeRF render errors. Finally we leverage all available views to extract high-quality 3D models from NeRF and we use point-set registration techniques to align these models with with known similar objects.

In summary the contribution of this work is as follows:

- We describe an approach, based on CosyPose, that does not require the exact 3D models of all the objects of interest. Instead it is sufficient to provide a few examples of 3D models for each category of objects that the user is interested in.
- We conduct initial experiments aimed at veryfying the practical applicability of our method. We describe the issues we've encountered throughout our experiments, we also provide solutions to some of the problems and analyze and narrow-down the cause of problems we were not able to fix along with some potential remedies.
- We provide an open-source, partial implementation of our method (available here [2]).

2 Background

Our method directly builds on recent advancements in two loosely coupled research areas: object pose estimation and implicit scene representations.

2.1 Object pose estimation

Single-view single-object pose estimation

Most of the recent approaches for this scenario fall under one of the two categories: *template matching* or *regression techniques*. Methods based on template matching align known 3D

[2] `https://github.com/jakubzadrozny/nerfpose`

models of objects to observed 3D point clouds, images or learned keypoints. Regression techniques tackle object pose estimation as a classification or regression problem. However, to achieve high accuracy, these methods typically require a refinement step that is based on rendering the known 3D models. The multitude of methods falling under both of the categories above, e.g. [7], requires access to the exact 3D models of test objects and is hence fundamentally different from our approach.

A notable and important work in the single-view single-object case is [8]. They utilize a CNN-based neural renderer module that is conditioned on a shape/appearance latent code and also explicitly pose-dependant. This rendering process is fully differentiable and hence at test time they optimize a pose and a latent code so that the rendered image matches a given input view. Our work is based around the same core ideas, however it has several key differences: (1) [8] works only in the single-view case whereas our method is geared towards multiple input views (as is CosyPose); (2) [8] uses a custom neural rendering process that generates 2D images without understanding the underlying 3D structure of the object, hence the shape of the object with a given latent code cannot be easily extracted and only the pose can be returned; we instead use a radiance-field based approach that assigns density values to each continuous 3D location, which then enables the extraction of shapes in arbitrary resolution.

Multi-view multi-object pose estimation

This is the exact scenario that this work focuses on. In this case all objects in a scene are considered together in order to jointly estimate the state of the scene in the form of a compact representation of the object and camera poses in a common coordinate system. This problem is known as object-level SLAM and has been approached by the robotics community, but usually via heavy dependance on depth sensors and an assumption that the camera poses come from a continuous sensor motion [9].

Our work depends heavily on a recently proposed multi-view multi-object pose estimation method CosyPose [1]. It is designed to work with multiple RGB-only input images where the camera poses are unknown. One of its main limitations is the usual assumption of having access to all the 3D models of test objects, which is hard to meet in practice due to extreme intra-class variability of real-life objects. In this work we propose modifications to the CosyPose framework to eliminate this limiting assumption.

2.2 Implicit scene representations

Several traditional 3D representations are mostly used throughout the literature: (1) voxel-based representations, (2) point clouds and (3) meshes. All of these however exhibit severe limitations, e.g. large memory footprint and low resolution for voxels (1), no connectivity structure in case of point clouds (2) and not being able to represent arbitrary topologies with meshes (3). To address these problems occupancy networks [10] were introduced. Instead of predicting a voxelized representation at a fixed resolution, they predict the complete occupancy function with a neural network, which can be evaluated at arbitrary resolution.

This idea is further extended by Neural Radiance Fields (NeRFs) [4] that predict not only the density at each continuous location, but also an RGB color of that point if observed from a particular viewing direction. This field can be later queried along rays shot from a given camera location and by alpha-compositing points on those rays a view of the scene can be rendered. A traditional NeRF is trained for a single scene using multiple input views of it with known camera poses by minimizing the L2 loss between rendered and given images.

To avoid training occupancy networks and NeRFs per-scene, they can be conditioned on a latent code that usually describes the content and/or appearance of the scene. Multiple extensions of NeRF have been proposed [11, 12, 13] where the conditioning is handled in different ways, but in this work we focus on pixelNeRF [6] mainly because it does not require test-time optimization of latent codes, hence there is no need to enforce latent-space structure and training is easier.

Most of the NeRF-based methods (including pixelNeRF) are fully differentiable and hence the camera pose can be optimized with gradient-based algorithms to reproduce a given image best. This technique has been described in [5] as *iNeRF* for *inverting* NeRFs. We exploit this method heavily across our pipeline to establish camera poses relative to each other and also the pose estimates of individual objects.

3 Preliminaries

3.1 CosyPose

Since our work builds on CosyPose, it shares the general notation and certain elements of the pipeline with it. To make this report as self-contained as possible, we outline the basic steps of the CosyPose method.

CosyPose takes as input multiple photographs of a scene $\{I_a\}$ and a set of 3D models, each associated to an object label l. The intrinsic parameters of camera C_a corresponding to image I_a are assumed to be known. The method works in 3 different stages: (1) object candidate generation, (2) object candidate matching and (3) global scene refinement.

In stage (1) a detection model is used on each view I_a to find bounding boxes of objects of interest. Each 2D candidate detection in view I_a is identified by an index α and corresponds to an object candidate $O_{a,\alpha}$ associated with a predicted object label $l_{a,\alpha}$. For each object candidate $O_{a,\alpha}$ a single-view single-object pose estimation method is used to generate an initial 6D pose estimate $T_{C_a,O_{a,\alpha}}$ with respect to camera C_a. The specifics of the pose estimation method used are important for CosyPose, but irrelevant for our work, hence ommitted here. In both CosyPose and this work a 6D pose $T \in \mathrm{SE}(3)$ is modelled as a 4×4 homogeneous matrix composed of a 3D rotation matrix and a 3D translation vector.

An important issue in object pose estimation is the handling of object symmetries. Following [1] we define a symmetry of an object as a rigid transformation that leaves its appearance unchanged. CosyPose assumes that the set of symmetries $S(l)$ for a test object with label l is explicitly known for all test objects. This assumption obviously needs to be lifted in our scenario as we don't even assume the knowledge of 3D models of test objects.

In stage (2) object candidates from different views that correspond to the same physical object are matched and a single, consistent scene representation is obtained. At first, CosyPose focuses on a single pair of views (I_a, I_b) and looks for all pairs of object candidates $(O_{a,\alpha}, O_{b,\beta})$, one in each view, which correspond to the same physical object in these two views. It first finds a set of candidate relative poses $T_{C_bC_a}$ between cameras C_a and C_b.

Hypotheses are generated by assuming a particular pair of object candidates $(O_{a,\alpha}, O_{b,\beta})$ corresponds to the same physical object with label $l = l_{a,\alpha} = l_{b,\beta}$. Then

$$T^S_{C_bC_a} = T_{C_aO_{a,\alpha}} S T^{-1}_{C_bO_{b,\beta}}$$

for any symmetry $S \in S(l)$ is a hypothesis for the relative transformation between cameras C_a and C_b. To reduce the number of hypotheses a second pair of objects $(O_{a,\gamma}, O_{b,\delta})$ with equal labels $l' = l_{a,\gamma} = l_{b,\delta}$ is also assumed to correspond to the same physical object. Then only $T_{C_bC_a} = T^{S^\star}_{C_bC_a}$ is considered as a hypothesis where $S^\star \in S(l)$ is the symmetry of object l that minimizes the mis-alignment of object l' in poses $T_{C_aO_{a,\gamma}}$ and $T^S_{C_bC_a} T_{C_bO_{b,\delta}}$. Considering two objects is usually enough to disambiguate symmetries [1].

For each relative camera pose hypothesis $T_{C_aC_b}$ the number of objects complying with it (inliers) is calculated. Each object candidate $O_{a,\alpha}$ from view I_a is matched with the object candidate $O_{b,\beta}$ from view I_b that has the same label and minimizes the mis-alignment of the corresponding 3D model in poses $T_{C_aO_{a,\alpha}}$ and $T_{C_aC_b} T_{C_bO_{b,\beta}}$. Such a pair is considered an inlier if the mis-alignment is lower than some threshold A. The hypothesis with the most inliers is finally selected.

Finally all the initial object candidates are merged using the 2-view correspondences described above to find the set of distinct physical objects. Consider a graph where vertices correspond to object candidates in single views and edges correspond to pairs that were consistent (considered inliers) with the final relative pose hypothesis. Isolated vertices are first removed as they correspond to object candidates that have not been validated by other views. Then each connected component in the graph defines a unique physical object. There are N physical objects $P_1, \ldots, P_N$, where N is the number of connected components in the graph.

The goal of the last stage (3) is to estimate estimate poses of physical objects $P_1, \ldots, P_N$ represented by transformations $T_{P_1}, \ldots, T_{P_N}$ and cameras C_a represented by transformations T_{C_a} in a common world coordinate frame. At first a random camera's coordinate frame is initialized as the world coordinate frame and later transformations are initialized using the – already estimated – relative camera poses and initial object candidate poses. Later a bundle-adjustment step is performed where all the object and camera poses are refined together to minimize a global reprojection error.

3.2 pixelNeRF

We do not provide details on the inner-workings and the training process of pixelNeRF, however it is necessary to state what an already trained model is capable of doing. Given an input image I captured with a camera C with known intrinsic parameters, a 3D point x and a view direction d, both expressed in camera C's coordinate frame, pixelNeRF produces with a single forward pass the density of point x called $\sigma(x)$ and RGB colors $\boldsymbol{c}(x, d)$ of x as observed from direction d. It can be represented as a following transformation

$$f(x, d, I) = (\sigma(x), \boldsymbol{c}(x, d)). \tag{1}$$

Here, crucially, f is a fully differentiable transformation realized by a neural network. Moreover, pixelNeRF can also be conditioned on multiple input views to boost its representation quality, which can be leveraged later down the pipeline. Conditioned on a set of N images I_n with known poses T_{C_n} expressed in some common world coordinate frame

$$\boldsymbol{I} = \{(I_1, T_{C_1}), \ldots, (I_N, T_{C_N})\}$$

for a point x and view direction d (both expressed in the common world coordinate frame), pixelNeRF can be written as

$$f(x, d, \boldsymbol{I}) = (\sigma(x), \boldsymbol{c}(x, d)). \tag{2}$$

An important charachteristic of the pixelNeRF method is that it's invariant to the world coordinate frame chosen, i.e. the output of f is unchanged if the world is transformed with any rigid transformation $T \in \mathrm{SE}(3)$. For example, any of the input cameras' coordinate frame may be chosen as the world coordinate frame.

The transformation f described above can be used to render a view of the scene registered by some camera C in pose T_C with known intrinsic parameters. This is achieved through a process similar to the original NeRF [4], but we outline it briefly here. A ray is cast from the camera's origin through each pixel on the image plane and a color of each of these rays is determined. Every ray $r(t)$ is determined by its origin o and (unit-length) direction d

$$r(t) = o + td.$$

The color $\boldsymbol{C}(r)$ of ray $r(t)$ is determined by alpha-compositing densities and colors along it

$$\boldsymbol{C}(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))\boldsymbol{c}(r(t), d)dt,$$

where t_n, t_f are the near and far bounds of rendering (determined by the dataset, they correspond to the volume bounds) respectively and

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(r(s))\right) ds.$$

In practice, this integral is estimated by a quadrature based on N randomly sampled t_i's between the near and far bounds. For a particular t_i the density $\sigma(r(t_i))$ and color $\boldsymbol{c}(r(t_i), d)$ are obtained from (1) (if given a single view as input) or (2) (if given multiple views) evaluated at $r(t_i)$. Let us denote the process of rendering a complete image, i.e. generating rays and alpha-compositing along them, from pose T_C given posed input images $\boldsymbol{I} = \{(I_1, T_{C_1}), \ldots, (I_N, T_{C_N})\}$ as

$$\mathcal{R}(T_C, \boldsymbol{I}).$$

Moreover when referring to the rendering process based on a single input view I and when the transformation T_C is expressed in the input camera's coordinates, we'll simply write

$$\mathcal{R}(T_C, I).$$

4 Method

The key difference from CosyPose introduced in this work is the lifting of the assumption that the 3D models of all test objects are explicitly known along with all their symmetries. A preliminary attempt at adapting CosyPose to this new setup would be to first perform some image-to-3D processing, e.g. occupancy networks [10] and then apply CosyPose on them as-is.

Before CosyPose is ran however, the method does not have access to object correspondences in multiple views and the relative camera poses, so the only straightforward approach is to use a single-view-to-3D method, which inevitably has lower quality than a multi-view method as it's largely based on prior experience. Hence the rest of this section focuses on a comprehensive modification of the CosyPose method in every place that it requires the 3D models and their symmetries. Throughout the rest of this section we use notation as in the description of CosyPose.

Our method works in 4 stages. First, in stage (1) each view I_a is analyzed independently and a segmentation model is used to identify and precisely locate all object candidates $O_{a,\alpha}$. The detector in the original method is only supposed to identify the exact test objects (and not similar ones), but in this scenario the exact objects are not known beforehand, hence the segmentation model should locate all objects belonging to the expected classes (provided as input). In contrast to CosyPose we do not estimate poses $T_{C_a O_{a,\alpha}}$ for object candidates at this stage.

In stage (2) relative camera poses are established, just as in CosyPose. We consider a pair of views (I_a, I_b) and we generate a set of candidate relative poses $T_{C_a C_b}$ between cameras C_a and C_b from which we'll later select the one with the best inlier support.

The most straightforward way to estimate $T_{C_a C_b}$ would be to initialize it as identity and optimize it to minimize

$$\|\mathcal{R}(T_{C_a C_b}, I_a) - I_b\|_2^2.$$

For this approach to work however, we'd require a pixelNeRF model able to generate novel views of entire scenes based on single-view input. Although the authors of pixelNeRF claim it is able to work with full scenes, they always use multi-view input in such scenario. At this point in our method we only have access to a single view, hence we do not consider the approach above promising.

pixelNeRF is however supposed to generate novel views of single objects based on a single-view input. We therefore proceed similarly to CosyPose, i.e. take two particular pairs of object candidates $(O_{a,\alpha}, O_{b,\beta})$ and $(O_{a,\gamma}, O_{b,\delta})$, with one candidate from each view in both pairs, where both candidates in a pair have equal labels, and assume that each pair corresponds to the same physical object. Let us denote by $I_{a,\alpha}$ image I_a, but with all pixels not belonging to the object candidate $O_{a,\alpha}$ masked as background. We proceed by minimizing

$$\|\mathcal{R}(T_{C_a C_b}, I_{a,\alpha}) - I_{b,\beta}\|_2^2 + \|\mathcal{R}(T_{C_a C_b}, I_{a,\gamma}) - I_{b,\delta}\|_2^2 \tag{3}$$

over $T_{C_a C_b} \in \mathrm{SE}(3)$. In other words, we find a relative transformation between cameras C_a and C_b s.t. the renders of both objects are consistent with captured images.

There are two issues to consider with the solution presented above. First, the union of the two objects may be symmetric, in which case there may be multiple local minima in (3), all of which should be considered as hypotheseses. Second, at any position symmetric to the ground truth for at least one object, at least one of the two summands of (3) is likely to have reached a minimum and the whole sum as well. These minima should not be considered as hypotheses (although they can be, they will not have sufficient inlier support to be accepted), but the simple optimization methods used (based on SGD) may be unable to escape such a trap. To overcome both of these issues (3) has to be minimzed starting from multiple different initial positions. Morevoer, the minimum reached may be accepted as a hypothesis if and only if (3) falls below a given threshold A. This will capture both the ground truth solution and all solutions in case the union of objects is symmetric (because both objects will be well reconstrcuted), but it will reject the local minima caused by a single symmetric object as then one of the objects will have high error.

For each relative camera pose hypothesis $T_{C_a C_b}$ the number of inliers (as in CosyPose) is later calculated. For every object candidate $O_{a,\alpha}$ in view I_a an object candidate $O_{b,\beta}$ is selected in view I_b s.t. it has the same label as $O_{a,\alpha}$ and it minimizes

$$\|\mathcal{R}(T_{C_a C_b}, I_{a,\alpha}) - I_{b,\beta}\|_2^2.$$

The pair $(O_{a,\alpha}, O_{b,\beta})$ is considered an inlier if and only if

$$\|\mathcal{R}(T_{C_a C_b}, I_{a,\alpha}) - I_{b,\beta}\|_2^2 < C$$

for some threshold A. Just as in CosyPose, the hypothesis with the most inliers is finally selected.

We later also proceed as in CosyPose by merging single-view object candidates into distinct physical objects using the graph analysis approach described in section 3.1. For the sake of

further analysis let us assume the graph analysis found N physical objects $P_1, \ldots, P_N$. A physical object P_i is in fact a connected component containing some vertices that are object candidates $O_{a,\alpha}$. We'll write $(a, \alpha) \in P_i$ to say that object candidate $O_{a,\alpha}$ corresponds to the physical object P_i.

Stage (3) is an adaptation of the bundle adjustment step of CosyPose. At this point in our method we have a world coordinate frame (chosen as one of the cameras' coordinate frame), initial estimates for camera positions T_{C_a} in these world coordinates and coorespondences between object candidates and physical objects. We do not however have any estimates of the position of physical objects in our world coordinate frame (or any other coordinate frame actually). Our bundle adjustment step is therefore aimed to correct the poses of cameras only. Let

$$\boldsymbol{I}_n = \{(I_{a,\alpha}, T_{C_a}) \mid (a, \alpha) \in P_n\}$$

be the set of all (masked and posed) views of the physical object P_n. For any $(a, \alpha) \in P_n$ let also

$$\boldsymbol{I}_n^{a,\alpha} = \boldsymbol{I}_n \setminus \{(I_{a,\alpha}, T_{C_a})\}$$

be the set $\boldsymbol{I}_n$ without a single view. The bundle adjustment objective is

$$\min_{T_{C_1}, \ldots, T_{C_V}} \sum_{n=1}^{N} \sum_{(a,\alpha) \in P_n} \|\mathcal{R}(T_{C_a}, \boldsymbol{I}_n^{a,\alpha}) - I_{a,\alpha}\|_2^2. \tag{4}$$

Each term of the minimized loss is the L2 loss between a registered view of a single object and its render based on all other available views. The minimization objective in (4) is fully differentiable with respect to T_{C_n}'s, so it can be optimized using standard gradient-based tools.

In the last, fourth stage (4) the poses of physical objects P_i are established in world coordinates. Consider a single object P_n. At this point we have access to a multi-view based volumetric model of P_n as

$$\hat{f}(x, \boldsymbol{I}_n) = \sigma(x).$$

Here the dependance on view direction d was ommitted, because the density of a point does not depend on it. From this model we can extract a point cloud representing P_n in the following way. Let V be a volume fully containing P_n. We first divide V into voxels with some resolution R, let S denote the set of all resulting voxels. We then repeat the following process until we collect enough points. Sample a voxel from S with weighted sampling, where the weight of each voxel is the volumetric density σ of its center times its volume. Add the center of the sampled voxel to the point cloud, remove the voxel from S, divide it intro four subvoxels and add them to S.

Once the point cloud of P_n is sampled, we estimate the transformation between it and its canonical pose based on 3D models of similar objects. Let M_n be the sampled point cloud (i.e. a set of 3D points) of object P_n and assume we have access to m_n point clouds $S_{n,1}, \ldots, S_{n,m_n}$ of objects similar to object P_n (as input, they may be given as meshes and later converted to point clouds). We set

$$T_{P_n} = \operatorname*{arg\,min}_{T \in \mathrm{SE}(3)} \sum_{k=1}^{m_n} \mathrm{dist}(T * M_n, S_{n,k}),$$

where $T * M$ transforms a point cloud M by the rigid transformation T and

$$\mathrm{dist}(M, S) = \sum_{p \in M} \|p - c(p, S)\|_2^2,$$

$$c(p, S) = \operatorname*{arg\,min}_{q \in S} \|p - q\|_2^2.$$

Transformations T_{P_n} can be determined by point set registration techniques, e.g. variants of the iterative closest point (ICP) algorithm.

5 Practical aspects and experimental validation

In this section we look at the pipeline described in sec. 4 from a practical point of view. We decompose it into smaller, relatively independent pieces and describe requirements that each of the pieces should satisfy in order for the entire pipeline to work well.

Our implementation and experiments are constructed towards running the pipeline on the Linemod (LM) [14] / Linemod-occluded (LM-O) [15] dataset. This dataset was chosen, because it is probably the simplest among all the datasets that CosyPose has been evaluated on, so that a comparison would be possible, but at the same time there would be as little data-related problems as possible.

5.1 The pixelNeRF model

The authors of pixelNeRF [6] provide an open-source working implementation along with their pretrained models. One of their training scenarios was to build a category-agnostic model for the 13 largest object categories in the ShapeNet [16] dataset. Because there is some (although small) overlap between those ShapeNet categories and LM test objects, we decided to test the pretrained model as a baseline.

Unfortunately the results, both qualitative displayed in fig. 2 and quantitative displayed in fig. 1, indicate the pretrained model is clearly insufficient in this case. To remedy this problem we fine-tune the pretrained model using synthetic renders of the LM test objects. We closely follow the training procedure and parameters described by the authors of pixelNeRF. After a relatively short training time (12 hours) the model reaches good performance, demonstraiting pixelNeRF's flexibility. Here, we've used renders of the exact test objects from the LM dataset, while ideally we'd like to rely only on renders of *similar* objects, however this is mostly due to time constraints and a lack (to our best knowledge) of a sutiable set of similar objects. In [6] authors experimentally demonstrated that pixelNeRF generalizes well to unseen examples, so significant obstacles are not expected here (other than assembling a good dataset).

Because pixelNeRF operates in view-space coordinates it is supposed to be invariant to changes in world coordinates, as long as all the relative poses remain unchanged. Specifically for a set of N posed images $\boldsymbol{I} = \{(I_1, T_{C_1}), \ldots, (I_N, T_{C_N})\}$, a camera pose T_C and any transformation $T \in \mathrm{SE}(3)$ we expect

$$\mathcal{R}(T_C * T, \{(I_1, T_{C_1} * T), \ldots, (I_N, T_{C_N} * T)\}) = \mathcal{R}(T_C, \boldsymbol{I}). \tag{5}$$

While running experiments concering the later stages of the pipeline, we've notived that the (5) does not hold for the fine-tuned model. After closer inspection we've discovered that the reason why our model was not invariant to world coordinate changes was a particular flag in pixelNeRF, called `normalize_z`, which essentially assumes that all cameras are looking directly towards the origin of the world coordinate space and that the object is located precisely at the origin. This assumption usually holds in most datasets focusing on scenes with a single object (like ShapeNet) and it resolves scale ambiguities that would otherwise have to be figured out by the model. When a large object is seen by a camera, it is not clear if it's a small object placed close to the camera or a large one placed far away. This can be disambiguated by the model when information from multiple views is used, but it's easier to simply place the object in the origin by assumption. Enabling this flag accelerates the training process, but after applying arbitrary transformations to the world coordinate system the assumption no longer holds (even if it originally did) and hence we don't have (5). In our use case it is crucial not to depend on this assumption, because in case of scenes with multiple objects the camera rarely (if ever) points directly at the center of an object. As the `normalize_z` flag is enabled by default, we had to manually turn it off and retrain the model to handle the ambiguities itself. Fortunately after another 12 hours of training we've obtained a model with good performance that also satisfies (5).

While inspecting the ShapeNet render dataset used by [6] we noticed that the variety of camera poses in it is heavily limited, specifically the camera moves over a sphere at a fixed elevation, always points directly towards the object, which is placed in the center of the sphere. In the Linemod dataset on the other hand the variety of poses is much wider. One particularly differentiating factor between ShapeNet and Linemod is the presence of in-plane rotations of the input image. We tested how adding in-plane rotations to input images affects the performance of the pretrained model and also how well can we fine-tune it to LM objects with in-plane rotation augmentation. It turns out that, even though pixelNeRF has no troubles fitting to LM objects themselves, it can no longer easily do so if we add in-plane rotation augmentation. The training was conducted similarly to [6] and it lasted for ca. 1 day with no visible improvements. Our hypothesis on why in-plane rotations pose a difficulty for pixelNeRF is that they directly increase the variety of images seen by the CNN encoder. Ideally, we'd like the encoder to be invariant to arbitrary rotations, but this imposes a major limitation on the CNN used, which may in turn be lacking capacity. Another issue to consider

is that we haven't been able to fit the model to a small set of test objects (only from LM), let alone achieve good generalization with a diverse dataset. If the pretrained model is already lacking capacity with the limited test set, it is likely to fall into further troubles when trying to achieve good generalization. Due to time constraints, we did not investigate how using larger (and perhaps different) encoders affects this problem.

model	no aug.	pos. aug.	rot. aug.
pretrained (ShapeNet)	28.3	21.2	19.0
pretrained (LM)	12.1	12.7	12.3
fine-tuned 1	26.0	12.1	15.9
fine-tuned 2	26.2	25.8	15.9
fine-tuned 3	15.5	16.3	16.1

Figure 1: PSNR scores of renders based on single views (averaged over 100 renders). The first column is evaluated without any augmentations, second is with world coordinate transformation and third is with rotation augmentation. The first two rows are the same pretrained model supplied by the authors of pixelNeRF [6], but evaluated on different datasets. Fine-tuned 1 is the model after fine-tuning, but with `normalize_z` turned on. Fine-tuned 2 is with `normalize_z` turned off and fine-tuned 3 is with `normalize_z` turned off, but also trained with in-plane rotation augmentation.

5.2 Rest of the pipeline

The issues with pixelNeRF turned out to be a major blocker for the rest of the pipeline. Without first fixing them, we would not be able to test the entire approach, so we focused on better understanding and trying to find a solution there. Certain experiments could be conducted however, even without a fully-capable pixelNeRF model, to emprically demonstrate that the rest of the pipeline is practical.

Regarding iNeRF and relative camera pose estimation, we'd like to answer at least these basic questions:

- if both pairs of object candidates do corrsepond to the same physical objects, can we recover the correct relative pose between cameras with low final loss?
- if at least one pair of object candidates does not correspond to the same physical object does the loss stay above the threshold C?
- does the optimization get stuck in local minima when at least one object has symmetries?
- how many initial candidates should be used to avoid the local minima and find all possible solutions?

Unfortunately there is no official implementation of iNeRF available yet and we did not find one that would work well in our setup, hence these questions are yet to be answered.

Other minor steps of the method could also be tested, i.e. point cloud extraction from pixelNeRF and the applicability of aligning a point cloud of an object to point clouds of similar objects. Due to time constraints we did not implement these experiments.

6 Conclusion

This report can be treated as preliminary work on an approach based on CosyPose [1] that recovers the 6D pose and 3D shape of multiple objects viewed from several cameras and that does so without the explicit knowledge of the exact 3D models of test objects. Through experimental work we've identified the main practical blocker for our approach, i.e. the pixelNeRF model, and we've narrowed down the source to the capacity / architecture of the encoder part. A number of tasks are yet to be completed before our approach can be considered complete, they include (1) either fixing or circumventing the problem with pixelNeRF encoder,

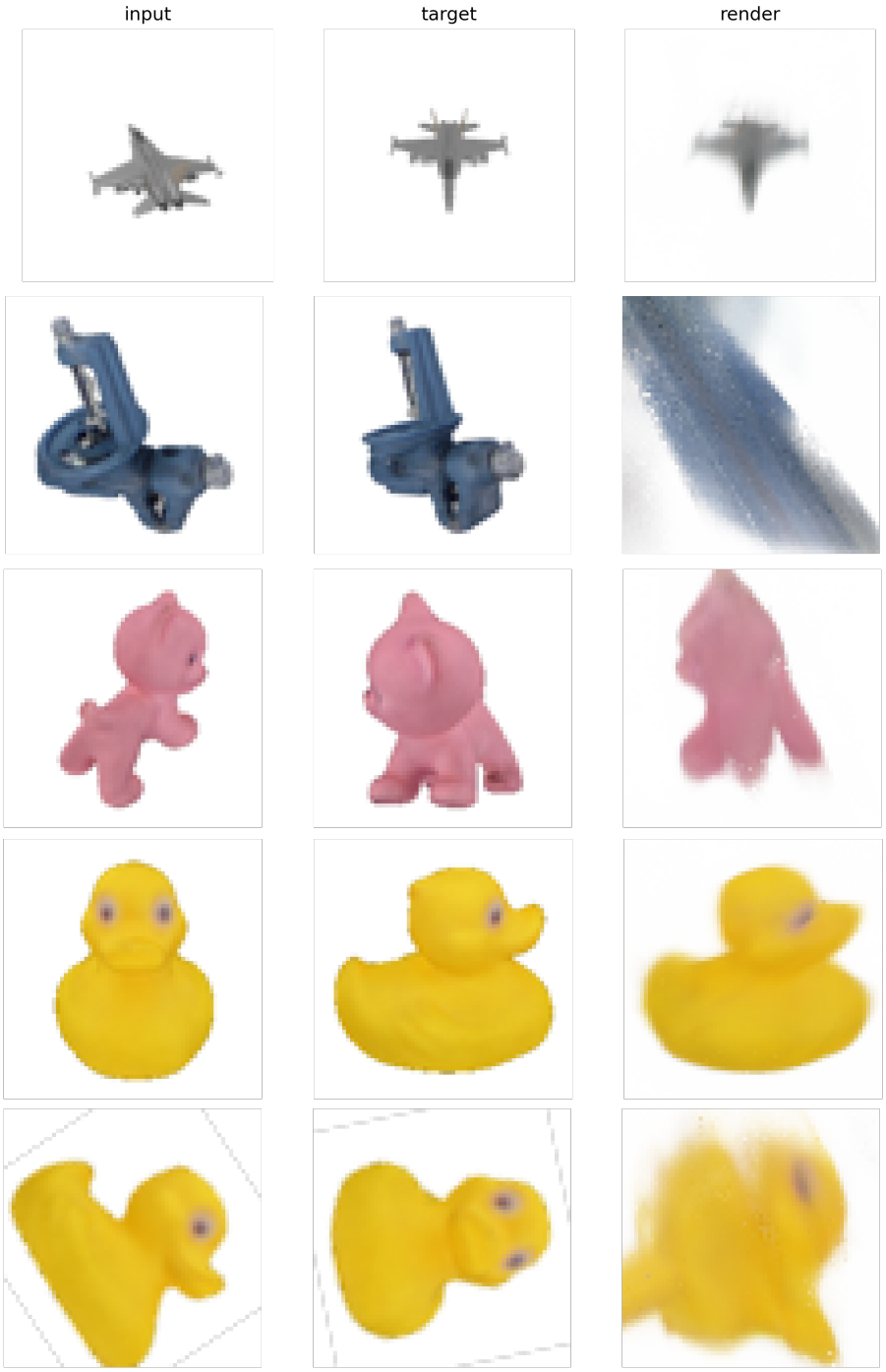

Figure 2: The first row is a render from the pretrained model on the original dataset. The second row is also the pretrained model, but applied on the LM dataset. The object color and placement is roughly preserved, but no detail are represented. The third row is a render from the fine-tuned model with `normalize_z` enabled with world coordinate transformation. The details are better preserved, but the render is shifted from its correct place. The fourth row is also a render with world coordinate transformation, but now with `normalize_z` disabled. The render is finally with details and in the correct place. Fifth row is a render from the same model as fourth row, but here an in-plane rotation was added, which had a significant negative impact on the result. The PSNR scores of these renders are: 27.3, 11.8, 16.8, 25.1, 13.8 respectively.

(2) validating the practicality of other steps of our pipeline and (3) assembling the pieces into a complete pipeline that can be evaluated and compared to CosyPose.

If (1) can not be fixed by using an encoder with higher capacity, the approach can be slightly modified so that pixelNeRF is no longer necessary. Our method can be adapted to use a method based on an explicit latent space, such as Sharf [11], where instead of using a CNN encoder (as in pixelNeRF) we optimize over an explicit latent space to find the code that produces renders most similar to our observations. This however also comes at a cost, because in this case we have to make sure that our latent space is properly regularized and highly expressive and that our optimization over latent codes does not get out of hand. Exploring the tradeoffs here is another interesting direction for future research.

References

[1] Y. Labbé, J. Carpentier, M. Aubry, and J. Sivic, "Cosypose: Consistent multi-view multi-object 6d pose estimation," in *European Conference on Computer Vision*, pp. 574–591, Springer, 2020.

[2] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, pp. 1150–1157, Ieee, 1999.

[3] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "Deepim: Deep iterative matching for 6d pose estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 683–698, 2018.

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

[5] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin, "inerf: Inverting neural radiance fields for pose estimation," *arXiv preprint arXiv:2012.05877*, 2020.

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

[7] M. Rad and V. Lepetit, "Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3828–3836, 2017.

[8] X. Chen, Z. Dong, J. Song, A. Geiger, and O. Hilliges, "Category level object pose estimation via neural analysis-by-synthesis," in *European Conference on Computer Vision*, pp. 139–156, Springer, 2020.

[9] B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model globally, match locally: Efficient and robust 3d object recognition," in *2010 IEEE computer society conference on computer vision and pattern recognition*, pp. 998–1005, Ieee, 2010.

[10] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

[11] K. Rematas, R. Martin-Brualla, and V. Ferrari, "Sharf: Shape-conditioned radiance fields from a single view," *arXiv preprint arXiv:2102.08860*, 2021.

[12] Q. Wang, Z. Wang, K. Genova, P. P. Srinivasan, H. Zhou, J. T. Barron, R. Martin-Brualla, N. Snavely, and T. Funkhouser, "Ibrnet: Learning multi-view image-based rendering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4690–4699, 2021.

[13] A. Trevithick and B. Yang, "Grf: Learning a general radiance field for 3d scene representation and rendering," *arXiv preprint arXiv:2010.04595*, 2020.

[14] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes," in *Asian conference on computer vision*, pp. 548–562, Springer, 2012.

[15] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6d object pose estimation using 3d object coordinates," in *European conference on computer vision*, pp. 536–551, Springer, 2014.

[16] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.