Technische Universität München

Department of Mathematics

Master Thesis

Satellite Pose Estimation

Salem Ben el cadi

Assessor: Nina Gantert

Contributor: Rudolph Triebel

Submission Date: 31 December

I assure the single handed composition of this master thesis only supported by declared resources.

Munich, 01/11/2021

signature :

Abstract

6D Satellite pose estimation is a crucial step of many on orbit servicing missions. Either for docking or debris removal, one need to know the pose of the target satellite. Throughout the time many methods were proposed to solve this problem, till the recent surge of monochrome based deep learning methods. We particularly get interested in data demanding direct regression methods using neural networks. We generate photorealistic data using Blenderproc and we divide many experiments to probe some architectural designs improving the networks performance, such as the choice of representations, hyperparameters and loss functions. Using all the feedback from previous experiments we go on optimizing the results.

Acknowledgement

First of all, I would like to Thank Prof. Dr. Rudolph Triebel for giving me the opportunity to conduct this work in collaboration with the German aerospace center (DLR), at the Robotics and Mechatronics institut during the year 2021. I would like to thank Prof. Dr. Rudolph Triebel additionally for providing me with a research direction and for guiding me throughout the thesis. I am very grateful to Martin Sundermeyer, with whom I have actively exchanged throughout these months, who guided my efforts and who introduced me to the software frameworks and methodologies that gave me a good start and a solid work foundation. I also want to thank Nassir Oumer who was willing to help with administrative issues as well as providing me with useful data. Further, I want to thank Nina Gantert my supervising professor at TUM for serving on my committee and providing crucial help and guidance. I would also like to express gratitude for much of the patience and diligence demonstrated by to the administration at DLR, and particularly in the RM institute : Ariane Schlinzig, Sylvia Scholz, Lilo Stieglmeier and Monika Schmidt. I'm very much appreciative as well for the active help and quick responses the administration in TUM and particularly in the Maths departement: Julia Cyllok, Anja Hoffman and Dörte Ptassek. Moreover, a big thanks goes to the administration in my home university and my professors in Toulouse that allowed the completion of this thesis in the context of an Erasmus exchange year : Anastasia Da Costa Velazquez, Cécile chouquet, Sophie Raynaud, Fabrice Gamboa, Zineb Zeinabou Ali and Paulo Carrillo Rouse.
Last but not least, I want thank my caring family for their support and my friends for making my time enjoyable in Munich and especially for my sister Hajar Ben el cadi for her unconditional care and most needed advice.
Salem,

Contents

1 Introduction

Estimating the position and orientation of a spacecraft is an essential part of many on-orbit missions e.g. debris removal, servicing or docking. Each of these has it's own scenario but all rely on a precise estimate of the relative pose between target object and the reference as shown in Figure 3. Pose determination may entail a wide range of engineering challenges, including the development of algorithms and sensor architectures suitable for the mission. As detailed in a review [21], pose estimation techniques may widely vary depending on the target object being cooperative or uncooperative. To help pose estimation the target object can make use of a radio communication, or is designed with referential passive/active markers (e.g. on it's corners). In this case, many features are acquired from sensor data and the object is so called cooperative. In the other case of uncooperative objects, very little features are known a priori, hence the need of more sophisticated methods to extract significant features for a precise pose estimate. Also these techniques are often required to be reliable and robust against harsh lighting conditions e.g. from very intense sun light to none, or against highly cluttered background, i.e earth. In order to obtain the required pose precision and robustness, the use of some sensors is preferred to others depending on mission scenario. For instance LiDAR technology provides an advantage in long range operations over RGB cameras, and is already in use in hybrid solutions like in TriDar [13]. While in uncooperative proximity operations, RGB or RGB-D cameras remain the sensors of choice because of their relative simplicity, low energy consumption and their low space and low mass compatibility with smaller platforms e.g. nano-satellites.

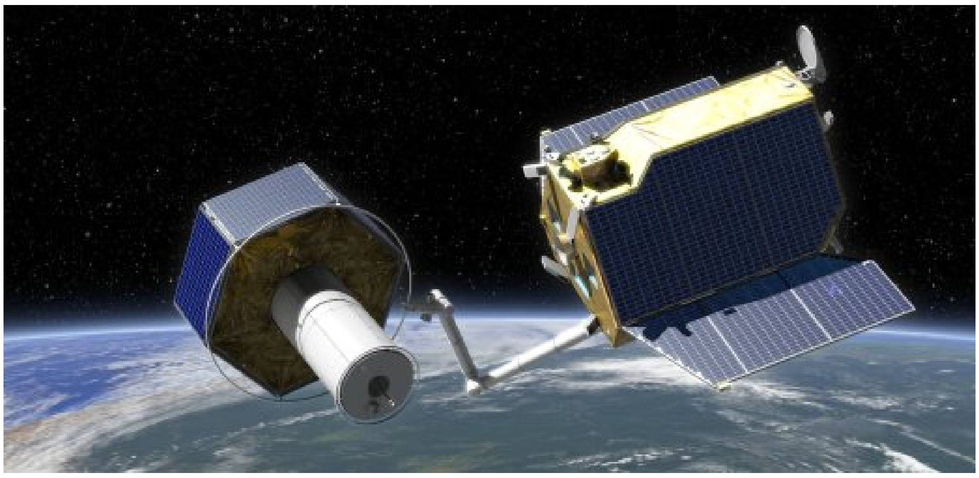


Figure 1: Deutsche Orbitale Servicing Mission

Among the vision based methods using these cameras, monocular pose estimation techniques [9, 8, 7] are drawing a special attention for their wide range of applications particularly in on orbiting servicing mission i.e OOS, shown in Figure 1. Section 2 includes a more detailed discussion on this type of application. Then Section 3 will briefly present the main types of monocular 6DoF pose estimation methods. Most notably among these a considerable interest is attributed to the subclass of supervised learning methods, which have grown in popularity hand in hand with the advancement of deep learning in the recent years.

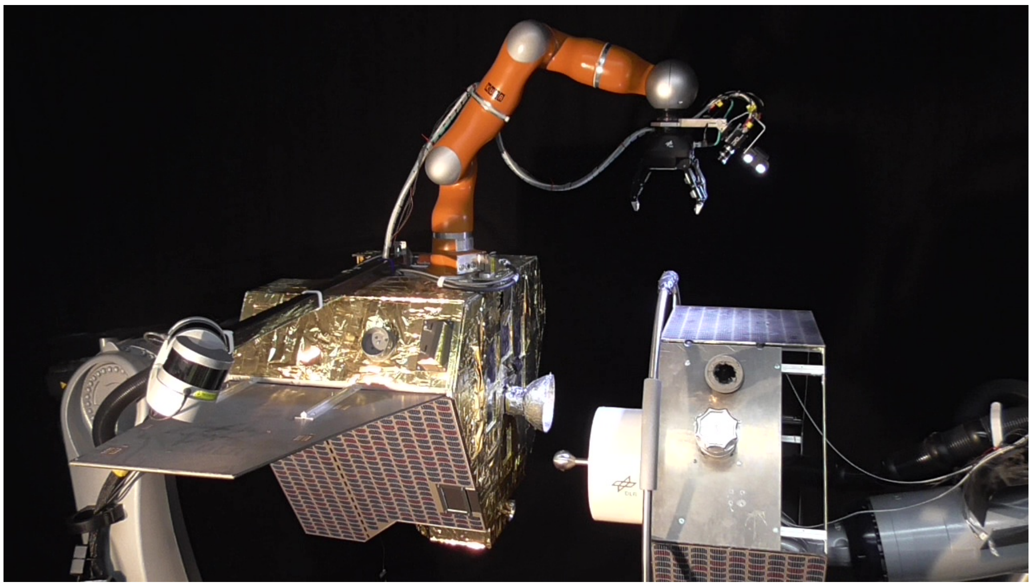


Figure 2: Simulating the On Orbit Servicing mission in a black room at DLR

Most of the state of the art monocular methods rely on deep learning [26] as their back bone or a part of a hybrid solution. However Deep learning is limited by the amount of data needed for training and validation, whereas real images of target space objects are not only sometimes not possible to obtain before mission, they are scarce and costly. Hence the need for simulated images, either in the lab e.g. at DLR as shown in Figure 2 or rendered from an available 3D model. My first contribution would be to propose a simulator built on Blenderproc that renders synthetic satellite images. The aim is to obtain a large data-set of photo-realistic RGB and depth images labeled by their relative pose, based a 3D model of an OOS satellite. My second contribution, is to experiment with training a Deep neural network for pose regression. I discuss the results of these experiments in terms of the impact of hyper-parameter choice and some aspects of the architecture that enhances the performance of the network. A special care is given to the choice of the rotation representation to be regressed.

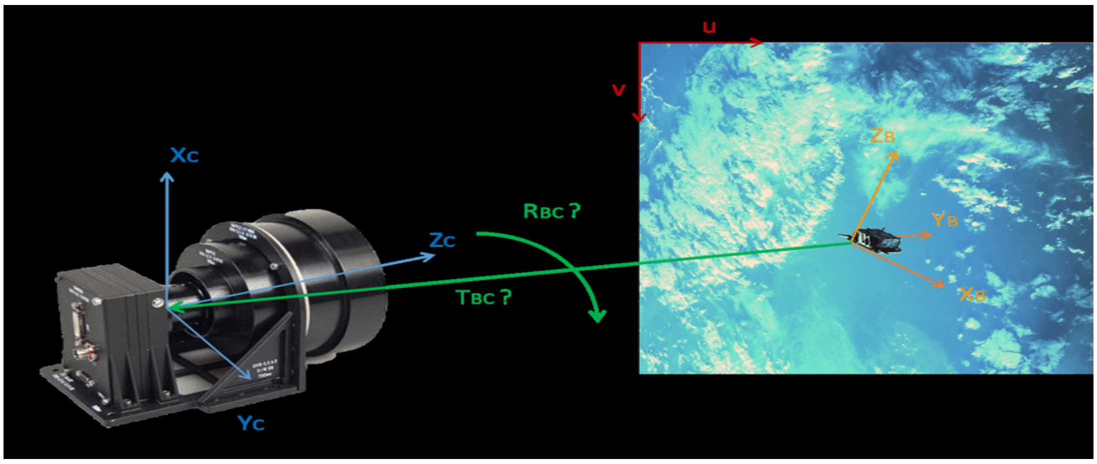


Figure 3: Illustration of the relative pose determination between camera and target object

2 Application

Monocular pose estimation techniques prove to be particularly efficient as well as being cost effective for proximity missions such as on orbit servicing. This type of mission might include :

1. Repair : Fixing simple damage of a component, e.g. a failure to move a sensor or extend an antenna
2. Upgrade : Replacement of outdated system components with components of the latest technology.
3. Refueling : Resupplying consumable e.g. fuel, coolant at the target spacecraft
4. Orbit maneuver : Orbit correction of the target space borne object. During the transfer stage of a satellite, it might get incorrectly positioned into another orbit rendering it non-functional or putting it at risk of collision with other space objects. Hence the need for a operation to put it back to the right intended orbit.
5. Space debris removal: Which is a serious issue since space debris continues to increase, lower orbits are more and more congested and the risk of collision is getting increasingly relevant. One collision generates thousands of damaging high energy particles and a few collisions could cause a chain reaction. As a result space exploration may no longer be possible if this issue remains ignored for long. Therefore there is a dire need for disposing of all kind of space debris e.g. defective or obsolete satellites, rocket parts and detached components.
6. Space construction : Less commonly, some proximity operations are aimed at building and assembling large structures in orbit to form an new ”space born” object.

3 Related work

3.1 Direct regression

The recent advance in deep learning methods, particularly in image classification and object detection have driven many pose estimation research projects in the direction of direct regression or end-to-learning [27, 5, 17, 15, 18]. This methods use deep neural networks to extract complex non-linear features and outputs the pose given the input image. Although some works have achieved state of the art performance using a direct regression paradigm, they still remain behind Geometry based solutions in accuracy.

3.2 Keypoints techniques

This the type of cooperative problems that ave been around for a while. Traditionally these pose estimation methods usually use manually crafted descriptors or key points, for example SIFT [20], SURF [2], and BRIEF [6]. The key idea is to create a set of 2D-3D and 2D-2D keypoint correspondences, then using non-linear optimisation from the correspondence set a pose estimation step is carried out. Keypoint methods can fail when there is a big variation in the pose, background textures, lightning condition. Most famous of these are PnP solvers that are able to estimate the pose accurately and robustly, given well crafted correspondence sets.

3.3 Feature extraction methods

Similar to keypoints methods, feature extraction method don't rely on handcrafted descriptors, which can not always be chosen optimally against a high variety of poses, lighting condition and noise. They use rather machine learning methods to best identify descriptors from different perspectives. BRIEF [6], and [22] uses naive Bayes classifier to determine keypoints. Deep learning methods have also made a contribution to this research direction as deep CNN network can extract powerful and complex features to fix the 2D-3D keypoint correspondance, e.g. [24, 23, 28]. The complexity of a deep learning netwoek leave door to many innovations, and a number of other papers share the same idea although they differ in model architechture. Namely for example [23] defines and uses semantic keypoints, and [28] is limited to the corners of the object's bounding box as for keypoints.

4 Synthetic Data Generation

The recent popularity of deep learning methods is partly due to the increasing availability of data. However this data is not always easy to harness and particularly not at all to label. The cost to label data is usually high in terms of the necessary time and effort which makes real annotated data scarce or sometimes unavailable. This is case in all domains where neural networks could be used to solve problems, like 6d pose estimation in robotics for instance. While the performance and robustness of neural networks relies heavily on the quantity of the annotated data, a new approach have to be considered. One way is simulation. That is creating synthetic data out of simulated environments that matches the real domain as much as possible. The advantage being one can configure the simulated but realistic environment as convenient as one can wish. Then one could automate the extraction of the features needed from the simulated environment to build the best labeled dataset to implement the deep learning task in question. More specifically in the 6D pose estimation problem, given a 3D model of the satellite we can build a simulation of the serviced satellite, the camera sensor from the servicing satellite and the environment around including light conditions. Then we render the images of the satellite from the camera view, as realistically as possible, at so many different positions, keeping the relative camera-object 6D poses as the corresponding labels. As we can automate the configuration of many features including satellite geometry, texture, background, camera positions and lighting conditions we obtains a big and rich annotated dataset that would maximize the networks performance on a variety of real scenarios.

4.1 Blenderproc : Data generation framework

BlenderProc [3] offers exactly this possibility of generating synthetic data most needed in deep learning applications. According to [11], Blenderproc is ”a fully configurable pipeline for procedurally generating scenes and rendering photo-realistic training images. The pipeline is built on top of Blender, an open-source project which offers a variety of relevant features through a stable API with years of optimization and an active community”. Blenderproc contribution complements Blender and goes even beyond fully automating feature set. In a typical pipleline one could load many objects, sample their position, define their geometry and texture proprieties, sample cameras at different positions, randomize background texture, then finally it once could choose to output render color or depth images, semantic segmentation or optical flows. As discussed in [10] and [11] Blenderproc offers highly realistic renderings. The presented objective evaluation of how much realistic the renderings is the comparison results on the performance of deep learning tasks, e.g. segmentation on Blenderproc rendered data on the one hand, and on real data, on the other. As well one can notice how the rendered images are qualitatively realistic as shown in Figure 4.

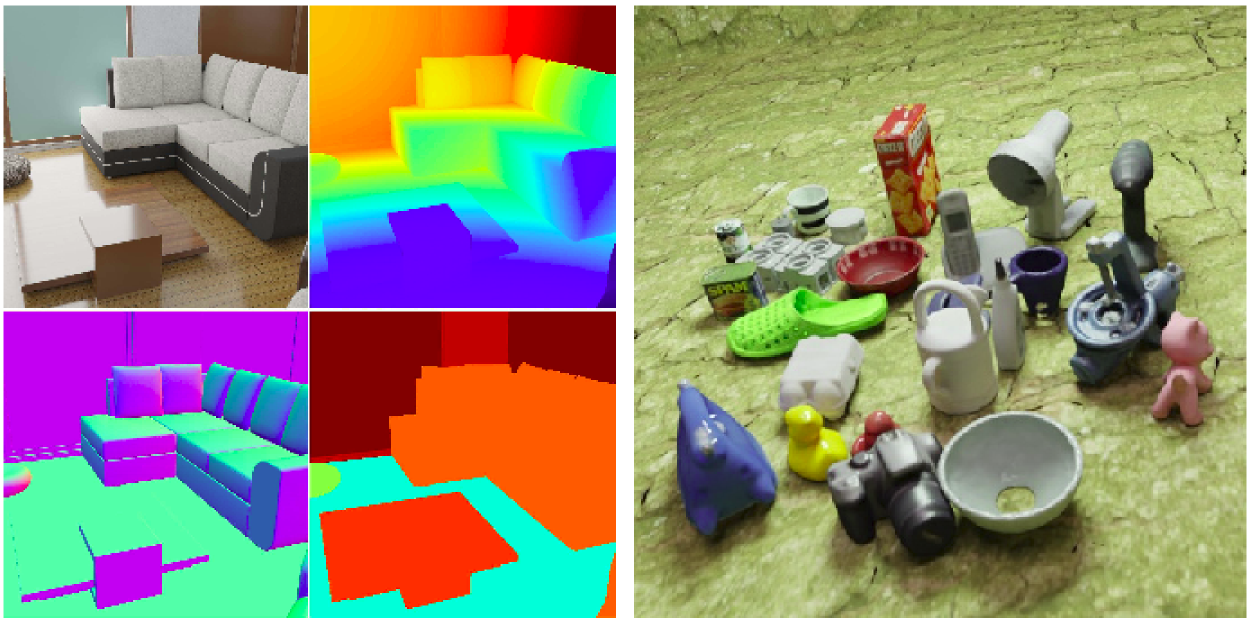


Figure 4: Blenderproc reducing the gap between reality and simulation

I've been introduced to BlenderProc as an internal project in DLR during development. Now BlenderProc is an open source project growing in popularity. Many features in the second version have been added and many are to come. I construct a synthetic annotated satellite 6D pose estimation dataset based on Blenderproc2 in it's current updated state.

4.2 Rendered images data

My first contribution in this thesis is to build a synthetic annotated dataset of images that would help me train a deep learning model to estimate the 6D pose of a target satellite in the context of a typical On Orbit Servicing mission. I build my rendering pipeline using Blenderproc. Then I generate 20000 photo-realistic images with corresponding pose as a vectorised SO3 rotation and location vector. I used 12000 images to for train and validation and 8000 images for the test dataset. I'll present here how I rendered these images and how I organised them into datasets.

4.2.1 Rendering pipeline

To obtain the photo-realistic rendered images a whole pipeline has to be set in place using the functionalities offered by BlenderProc. I have in my disposition a 3D model of an OOS satellite cf. the serviced Satellite in Figure 1 and 2. Using the python interface of Blenderproc I first load the 3D model, then I fix a position and orientation in the world's reference. I set a strong punctual light to simulate the sun light's intensity and direction. I load aluminium texture to simulate the Multi-layer insulation sheet on the front of the satellite. I set texture parameters as defined in Blender, such as base color, roughness, displacement, etc that would make more or less creases on the aluminium foil for example.

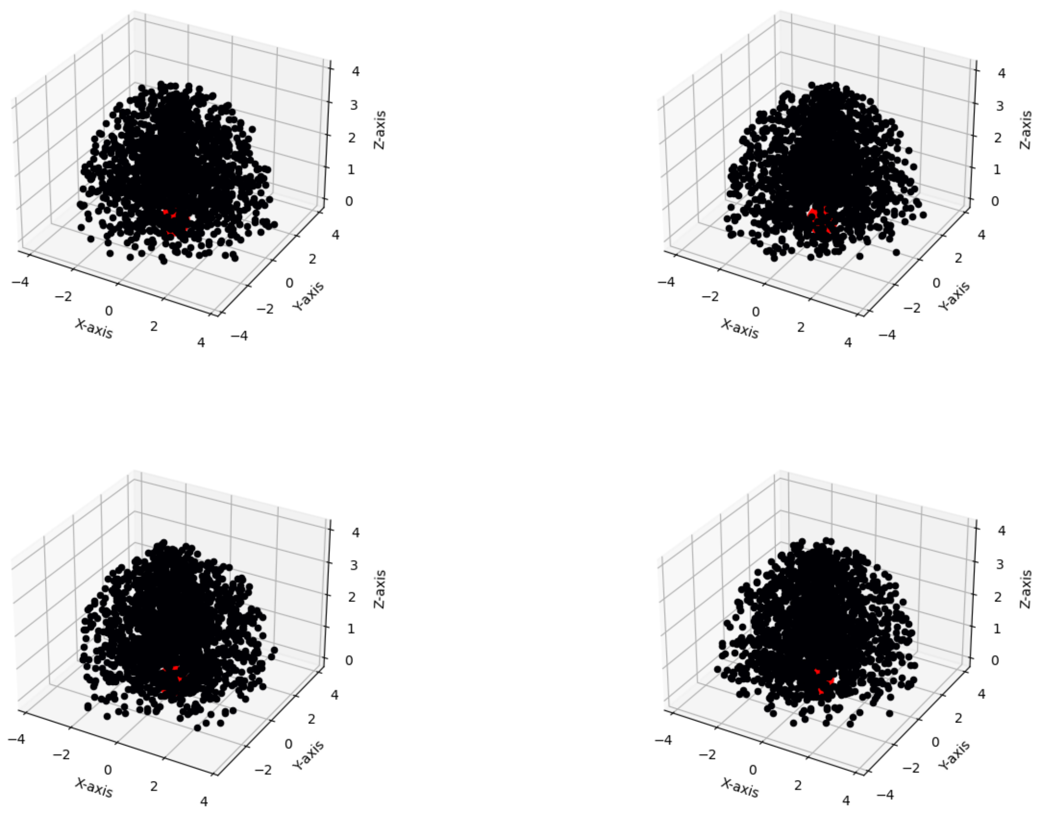


Figure 5: Sampled camera positions (black dots) on a half sphere around the object (red sphere)

Then I sample positions of the camera uniformly in the half shell around the fixed position of the satellite as the Figure 5 shows. The shell is contained between two sphere of radius 1m and 4m respectively from the center of satellite. The camera is pointing towards the center of satellite as well. Finally I render RGB and depth images and store them on the form of a Bop dataset cf.[4] where I get RGB and depth images and corresponding relative poses stored in a json file as a SO3 rotation 9D vector and a 3D location vector.

4.2.2 The dataset

To build the dataset I need for training my neural network, I iterate the pipeline above 2000 using a GPU machine on a cluster in DLR and iterate this task 10 times. I obtain 10 folders on the Bop format then I aggregate all the data in one folder including the json files in one json file including all poses. Checking for errors during this process is necessary, re-indexing and copying many files can be subtle.So I make sure to check whether each original rendered image corresponds to it's original pose in the final aggregated folder. Once I have 20000 data sample and their pose labels, I split into 12000 images train-validation dataset and 8000 images dataset. Figure 6 shows some training samples and Figure 7 shows a sample from test dataset.

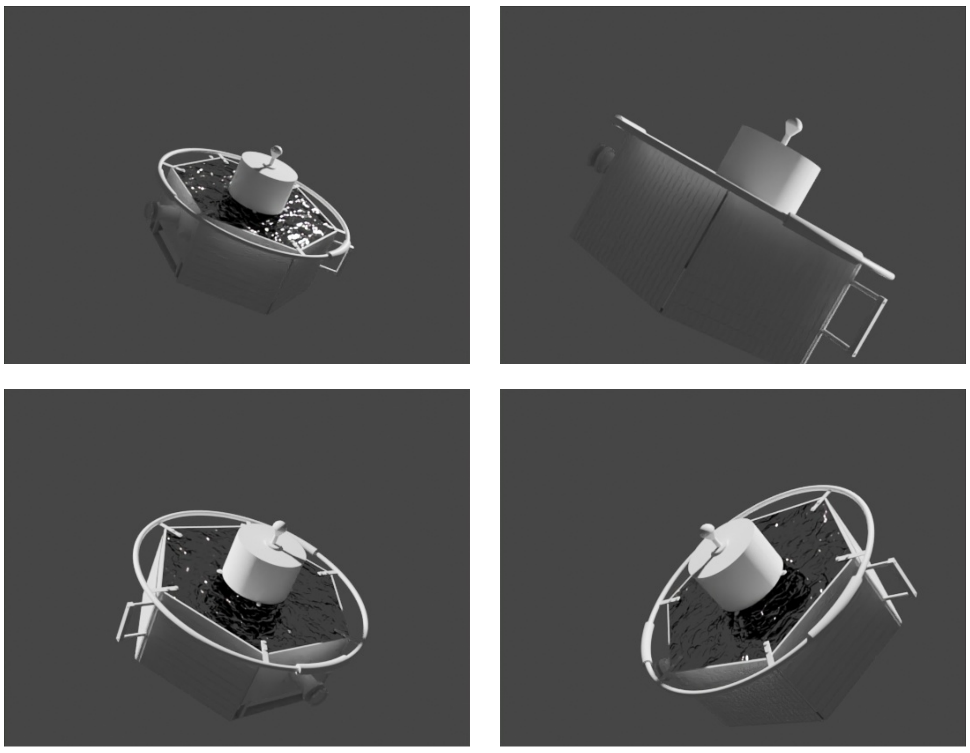


Figure 6: Train dataset images samples

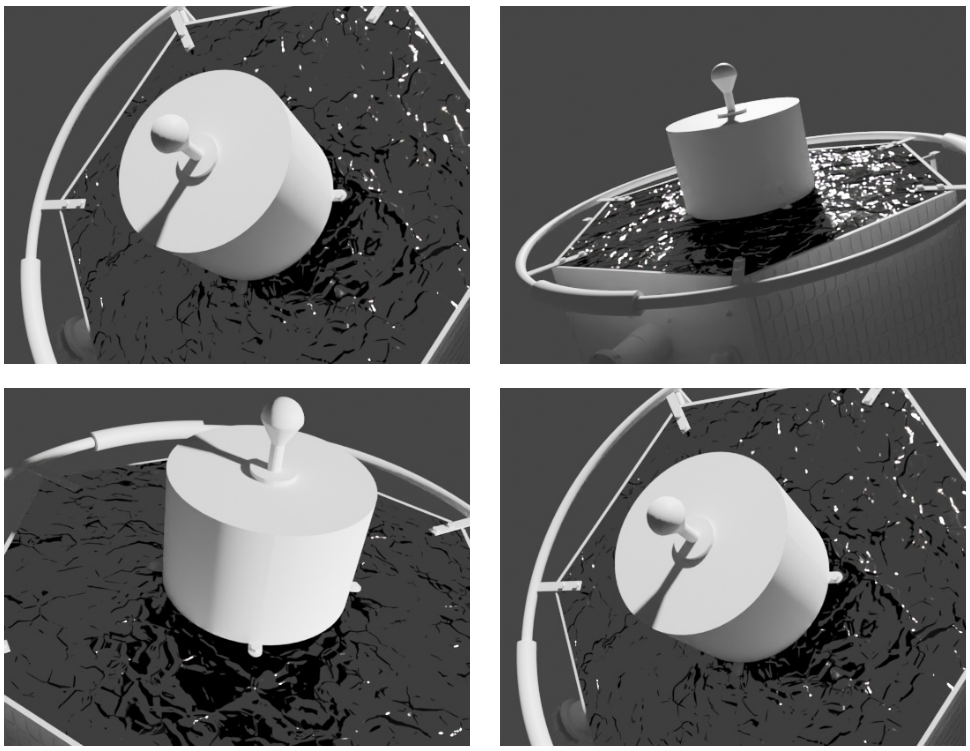


Figure 7: Test dataset images samples

4.3 Validating Data

There is a restriction to qualitative judgement in the scope of this thesis when it's about validating the generated data. There is a small sample of real data and I use it to compare it qualitatively to rendered synthetic data. Although there is some imperfections with the 3D model, as shown in Figure 9 the aluminum texture has similar light reflection as in the real images and is almost enough to substitute for Multi-layer insulation texture. More improvement can be made to make the renderings even more realistic but this level is enough for training a neural network.

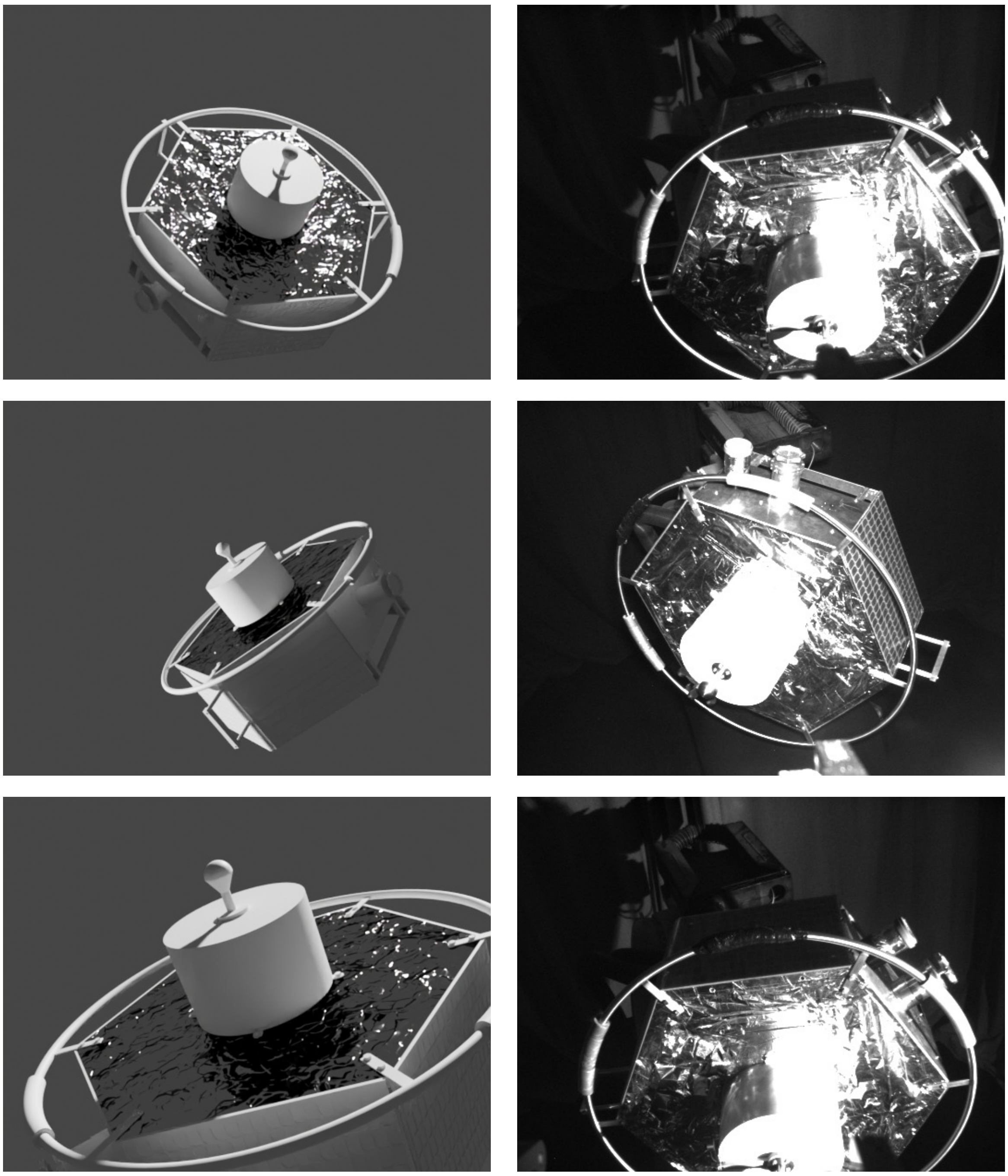


Figure 8: (a) Rendered images (b) Real images

5 Satellite pose estimation

In this section we'll discuss briefly about different concepts important for the presentation of the results and their evaluation. Mainly we'll define a continuous representation and discuss it's importance in network's performance. We'll see some common representations and briefly point to their advantages and weaknesses. Then follows a small presentation of presumably known concepts about neural networks. Finally a first description of the training pipeline is described.

5.1 Pose representation and SVD

Directly regressing rotation is a hard problem, much harder that simply regressing a 3D location vector. Depending on the choice of the regressed output the results could vary greatly. That's called the representation of rotation and it's of crucial importance to the results when following a direct regression approach of pose estimation. There is one propriety of the representation that changes drastically the performance of a regressor as discussed in [30]. That is the continuity of a representation. [30] rigorously defines continuity as follows : "Let R be a subset of a real vector space equipped with the Euclidean topology. We call R the *representation space*: in our context, a neural network produces an intermediate representation in R. Let X be a compact topological space (e.g. space of 3d rotations $SO3$) . We call X the *original space.* In our context, any intermediate representation in R produced by the network can be *mapped* into the original space X. Define the *mapping to the original space* $f : R \rightarrow X$, and the *mapping to the representation space* $g : X \rightarrow R$. We say (f, g) is a *representation* if for every $x \in X, f(g(x)) = x$, that is, f is a left inverse of g. We say the *representation is continuous* if g is continuous."

5.1.1 $SO(3)$ Rotation and SVD

More concretely on this definition, suppose the network regresses a 9D vector that's supposed to represent a rotation. Here the space of representation is R=$\mathbb{R}^9$ and X= $SO3$. From the space of 3D rotation $SO3$ we can simply obtain a 9D vector (in R^9) writing the elements of the matrix as a vector. However the inverse operation of writing a 9D vector as 3x3 matrix doesn't necessarily give a 3D rotation matrix, because one need to satisfy orthogonality. Hence this operation as a mapping from $\mathbb{R}^9$ to $SO3$, is not always invertible, i.e. if composed with it's the inverse operation as mapping from $SO3$ to $\mathbb{R}^9$doesn't always give the identity, and there is not continuous according to the definition above.

To get a continuous representation an orthogonalization step is necessary after regression. Gramm-schmit procedure could be used but a discussion in [19] disfavor it over an SVD orthogonalization that yields better results on a regression task. Directly regressing a 9D for rotation is not a good idea.

5.1.2 Euler angles

3D euler angles vector is neither a good idea for a representation to regress rotation. Although it's a straightforward way to understand rotation, it's easy to observe the discontinuity of this representation. An Euler angle of θ=2π is the same as the angle θ=0 but if the network out put 0.1 and the ground truth would be 2π, then the loss would be quiet big, penalising the whole network when it shouldn't change that much since we are regressing rotations and the predicted and ground truth are almost the same ($0.1 \approx 0 = 2\pi$).

5.1.3 The Quaternion

A unit quaternion q represents rotation as :

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} = w + \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \cos(\alpha/2) + \vec{u}\sin(\alpha/2) \tag{1}$$

Where $\vec{u}$ is unit vector. This is a compact representation of rotation, not requiring orthogonalisation as a normalization of the regressed 4D suffices. However if one rotates around $\vec{u}$ with an angle α, it's the same as rotation as when one rotates around -$\vec{u}$ with an angle 2π-α , yet it could be concluded from the right hand side of formula 1 that it give a different unit quaternion -q. The quaternion representation is therefore discontinuous however more convenient.

5.1.4 6d representation

In the work of [30] a 6D rotation representation is defined and proved to be continuous. From an $SO3$ matrix you get this 6D representation from simply removing one column or row vector. The inverse of this map is not unique as we retrieve back an $SO3$ matrix by Gramm-schmit orgonalization procedure.

5.2 Modeling and training pose regression network

I present here the common model and training pipeline to all of the experiments that were carried out and discussed in details in the next section. Each will slightly modify this model out of a curiosity to know how the networks performance reacts to the changes.

5.2.1 Model

The model is an untrained VGG16 network with a modified encoder. the new "classifier" branches out from a Linear(512 * 7 * 7, 1024) layer, with an added Relu unit, to two separate ”classifiers”. One to predict rotation and the other one to regress location. The rotation ”classifier” starts with a Linear(1024, 512) layers and chains to another Linear(512, 256) layer and then to a Linear(256, 128) layer until finally regressing a 9D vector with Linear(128, 9) layer. Relu unit are kept in between to add more non-linearity while also adding a dropout of 0.5 each time to avoid over-fitting. Simultaneously the location encoder also starts with a Linear(1024, 512) layer and progressed to another Linear(512, 256) layer to finally regress the 3D vector with a linear Linear(256, 3) layer. Rectified linear units and a 0.5 dropout are also added between layers.

5.2.2 Training Overview

Before feeding data to the network a pre-processing phase has to make sure data is well conditioned. First of all the initial train dataset is split into a validation and training dataset with 20% and 80% ratios of the initial dataset respectively. A training dataloader then decomposes the data into batches of equal size and sampled randomly and feed it to the network. During validation the random batch sampling is deactivated, as well as the dropout phase. many metrics are computed then out of the prediction and ground truth values and logged for network performance monitoring. The loss and the gradients, and the updated weights are computed as well one batch a time and through the whole epochs and the loop repeated again a number of epochs as the losses decrease and the network learns.

5.2.3 Testing

In the test phase we go through the same pipeline as the validation with the exception that we evaluate the model one time on all individual 8000 validation images. We store these results in a json file for later use to validate the results qualitatively.

6 Experiments and evaluation

Deep learning methods are limited by the data needed to train and validate them. In the previous section I describe the generation process of pose annotated images using BlenderProc. I use 12000 training images to train and validate a deep neural network directly regressing the pose in different configurations. A brief description of the experimental settings follows.

6.1 Settings and training configurations

It's important to present the experimental setting before an evaluation of the results to be able to interpret them in the context of how they were produced. I used Pytorch Lightning as the deep learning framework that follows a clear stylistic guideline for the whole pipeline decomposing it in simple blocks. This simplicity doesn't come at the cost of generality as it offers customisation handles till the level of the GPU. Another advantage offered by Lightning is the acceleration capabilities for Multi-GPU training which I needed to run my code on a SLURM GPU cluster in the RM institute inside DLR. In a first experiment I run the training for 4h20m on parallel on two different machines one using 2xGeForce RTX 2080 Ti GPU and the other a 2xTITAN RTX GPU, in a second experiment meant to improve results, I run the training on a 2xTITAN RTX GPU for 24 hours. For that I opted for the Distributed Data Parallel Multi-GPU training acceleration strategy. In all experiments I used 4 CPUs for each machine and a batch size of 16 images. This choice was mainly made because this configuration offered the best time performance. With a learning rate of 0.001 I have obtained poor results so I fixed it to 0.0001 which already yielded a way better training and validation performance. Further-more, to optimize the performance, I inspected for a better learning rate using Lightning *auto_lr_find* option that finds the best learning rate based on [25]. The optimizer I used is Adam, which I preferred to an SGD because of the small batch size. To avoid the gradients exploding or vanishing at some point after a day long training, I used the scheduler, *ReduceLROnPlateau*. The inputs I use are RGB images of size 224x224x3 which have enough resolution to capture all the details including the small variations of the texture. Using a higher resolution didn't improve the results by much. I normalize the inputs before feeding them to the network. The advantageous effect of this was lower loss values at the start of training and validation, compare to no normalize step. To monitor the performance of the network I used logged different relevant metrics and visualized them with the python framework "Weights and Biases" which offered a real time view on the progression of the training, validation and testing.

6.2 Hyperparameter search

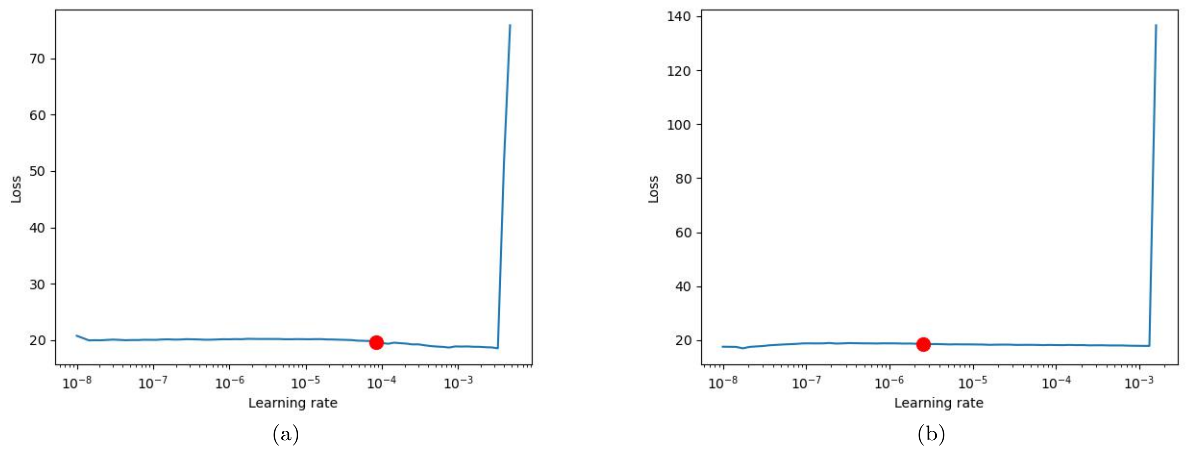


Figure 9: learning rate graphs : with SVD (a) no SVD (b)

To increase to performance of training a learning rate search was done as shown in the figures. Depending on whether or not we include an SVD transformation to the regressed 9D rotation vector, the optimal learning rate varies in the range $[10^{-8}, 10^{-4}]$. It could be clearly observed from the graphs that 10^{-3} is bad choice of the learning results, which matches the intuition on the results before the learning rate search. However beyond 10^{-3} the search of the best learning rate gets tricky because the graphs almost flattens for both case leaving a margin of doubts on the best learning rate. After a few runs, the optimal choice was for learning rates in the smaller range $[10^{-5}, 10^{-4}]$ as lower learning values would yield worse results. The other hyperparameters where kept as they didn't present any enhanced performances, namely the batch size and the dropout rate.

6.3 Evaluation metrics

The choice of evaluation metrics is most crucial to make sens of the results and interpret the networks performance, in terms of time, precision and generality. Most important of all these metrics is the chosen loss that sets the direction of training as it's gradient governs all the weights updates during back-propagation.

6.3.1 Axis-angle rotation loss

Also refereed to as the geodesic loss. This is the most important metric to compute when estimating a rotation. it's computes the shortest distance between two rotations in $SO3$ as explained briefly [12] or [14] in more details. More concretely we compute the axis angle of the product quaternion of the first rotation quaternion and the conjugate of the second rotation quaternion. This angle appears in the graphs in degrees.

6.3.2 MSE losses

This section deals with the number of losses computed as a mean square error of two vectors. In a first experiment the chosen loss for training the network is an MSE loss of the 12 dimensional vector, 9 components for rotation and 3 components for location. Additionally other MSE losses where chosen as a measure for network training performance and include : the Quaternion vector loss, the 6d-representation vector loss, the 9D rotation loss, the 3D location vector loss. In a second experiment, the chosen loss was a hybrid loss of rotation and location inspired from [29] and [1]. Since we have the 3D model of the satellite, we take a uniform sample of all the vertices (10 percent of the number of vertices) and compute their images with both the estimated rotation and the ground truth rotations. An MSE loss is then used to compute the error between the estimated output and the ground truth. Then an MSE loss component of the estimated and ground truth location vectors is then added to made the hybrid loss.

$$\mathcal{L}_{hybrid}(R_i, t_i, \hat{R}_i, \hat{t}_i) = \mathcal{L}_R(R_i, \hat{R}_i) + ||t_i - \hat{t}_i||, \tag{2}$$

$$\mathcal{L}_R = \frac{1}{|\mathcal{M}|} \sum_{\mathrm{x} \in \mathcal{M}} ||(R_i \mathrm{x} - \hat{R}_i \mathrm{x})|| \tag{3}$$

where $\mathcal{M}$ indicates the set of 3D model points. Here, we subsample 2172 points from provided 3d point cloud. R_i is the ground truth rotation and t_i is the ground truth translation. $\hat{R}_{\sigma(i)}$ and $\hat{t}_{\sigma(i)}$ are the predicted rotation and translation, respectively.

6.3.3 Euler angles losses

Euler angles were introduced as an additional metric. However as expected they are not the the best target for a rotation regressor because of the inherent discontinuity of this representation, cf. Section 5.1.1.

6.4 Evaluation of training on Synthetic Data

Initially the intention behind experimenting with a network following a direct 6d pose regression paradigm was to compare the choice of different representations of the regressed output. However along the trials, many modifications to network architecture, while preserving the simple direct regression paradigm, have proved to be crucial for network performance. Moreover these made the difference between the effects of representation choice even more evident. One particular choice that made a huge difference is adding an SVD decomposition before regressing the final output, of namely the intermediate 9D vector that is supposed to regress a rotation matrix. Once this choice was adopted and the comparison between different representation was established, the intention for the subsequent experiments shifted toward making the results even more accurate. Although no further experiments were carried out toward comparing the finals results to state-of-the-art methods on benchmark data, the observed accuracy on the test dataset was distinctively low on both rotation and location metrics.

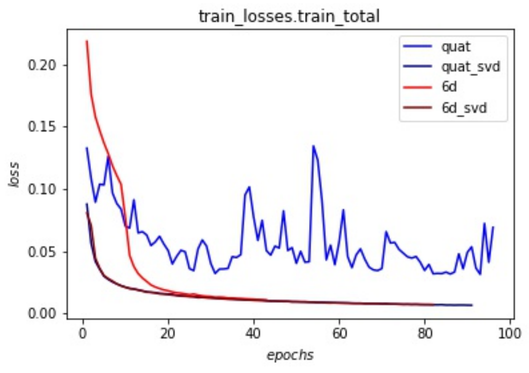
6.4.1 The effects of rotation representations

To probe the importance of different representations in terms of regression convergence speed and accuracy, I divided four experiments of neural network training running in parallel. The same settings where preserved for these to account only for the representation changes among the reasons that would explain the apparent differences in the results. Each have in the common the pipeline described in 5.2.2. Depending on representation in question different additional layers have been added to the last 9D vector output meant previously to directly regress the rotation matrix of the relative pose. In one experiment, I add a linear layer of 4D vector output with an Rectified linear unit, to introduced more non-linearity and 50% dropout rate to avoid over-fitting. In a second experiment, I map the 9D output to another 9D vector representing an $SO3$ using an SVD based orthogonalization method as described in [19]. Then I add a Quaternion representation as a 9x4 linear layer with Relu and 0.5 dropout. For the third and fourth experiments I repeat these last two but I switch to a 6d rotation representation instead a Quaternion, by adding rather a (9,6) linear layer with Relu and 0.5 dropout, once with an SVD orthogonalization and once without. The loss used for training in these experiments is the MSE loss joined rotation and location vector. It's the quadratic error between predicted and ground truth 7D vector in the case of 4D quaternion vector and 3D location vector or the 9D in the case of 6d rotation representation. The choice of the loss function will prove later to be of crutial importance to the network's performance.

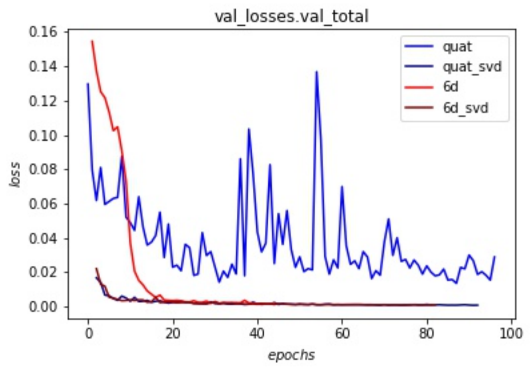
As shown in the figure 10 the 6d representation (red line) provides a clear advantage over the Quaternion representation (blue line) on all metrics. from one hand all the losses of the 6d representation converge, and progress faster toward a limit value. From the other hand, the loss in the 6d representation case decreases smoothly as the curve does not present any irregularities. The opposite could be said about the loss curves in the case of the Quaternion representation. The loss curves in this case seem to converge however slower and with much irregularities. For simplicity many other graphs of the tracked metrics where omitted and only four metrics where kept here to show the progression of the network's performance during training and validation. other figures could be found annexed.

Figure 10 also shows obvious performance margin between the use of the SVD orthogonalization method and without, especially in the case of Quaterninon. However in the case of 6d representation the margin is a little bit less clear in terms of accuracy and after many enough epochs as all the curves converge toward the same value soon or later, using SVD or not. This convergence is slower nevertheless when not using the SVD as the curve decreases from much higher initial values after more than 40 epochs before reaching a comparable accuracy to when using SVD. It follows then the importance of using SVD orthogonalization preferably coupled with a continuous representation starting 5D or higher to avoid irregularities.

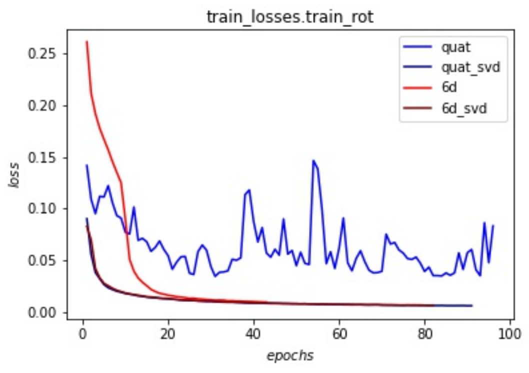
If these results underline the performance difference between representations and the advantage of using an SVD orthogonalization, they do not compare to state-of-the-art methods in solving the main problem, that is of 6D satellite pose estimation. Further investigation for ways to optimize performance had to be carried out.



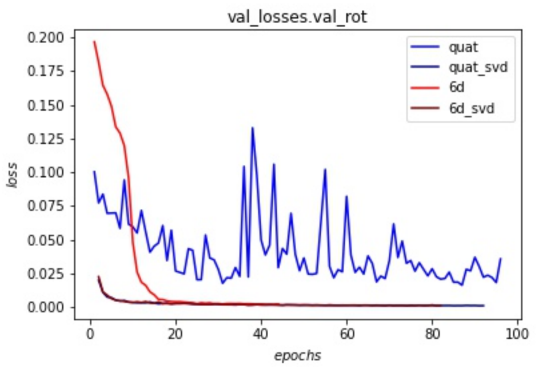
(a)



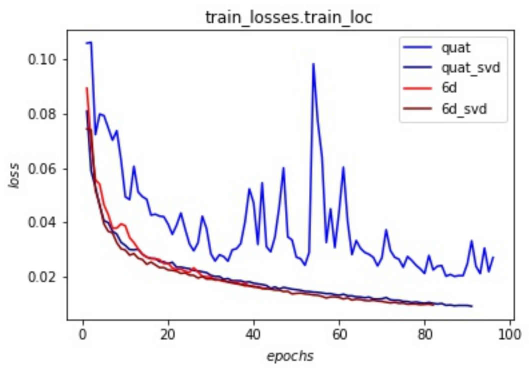
(b)



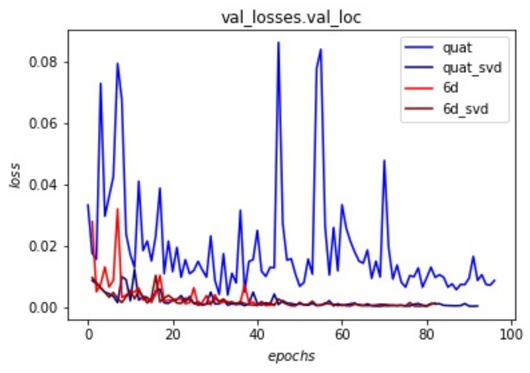
(c)



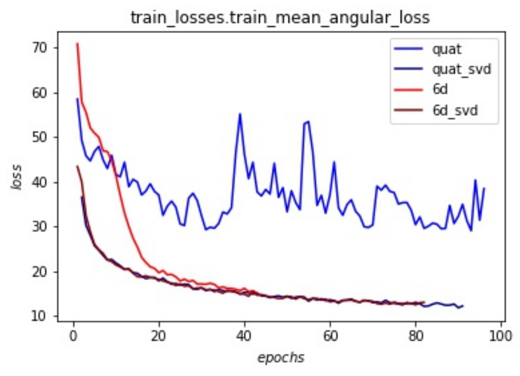
(d)



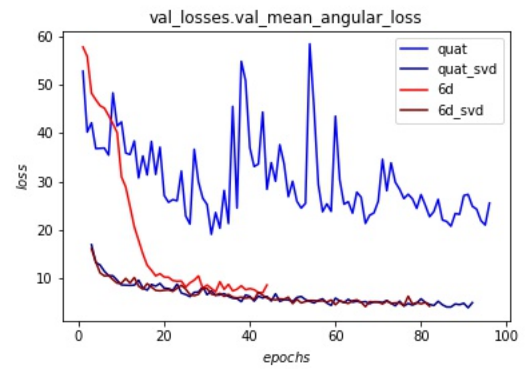
(e)



(f)



(g)



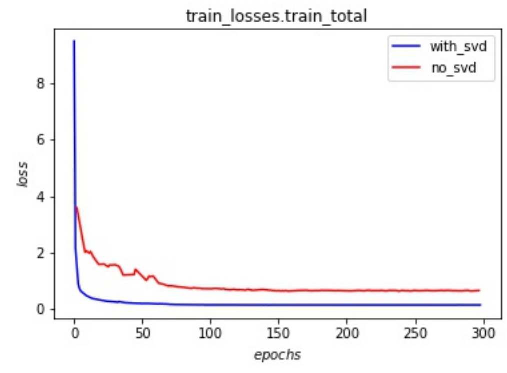
(h

Figure 10: The evolution of different metrics during training and validation including (in meters) the MSE loss of the joined rotation and location vector (a) & (b), MSE loss of the rotation vector (c) & (d), the MSE loss of the location vector (e) & (f) and the mean angular loss of the Euler angles (g) & (h) in degrees.

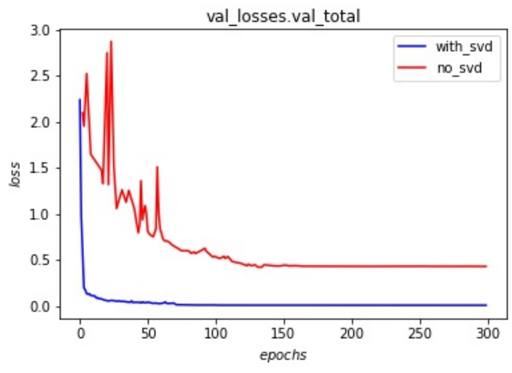
6.4.2 Further optimization

In this section I present a trial to get the best result out of a direct pose regression having learnt the benefits of a good representation, an SVD orthogonalization and having discovered new advantages about the loss function choice inspired by [1]. First of all to be able understand the level of performance of network two metrics are essential. First the geodesic loss which is the best measure of how far two rotation are and is expressed in degrees, and second the quadratic error between predicted and ground truth location vectors.

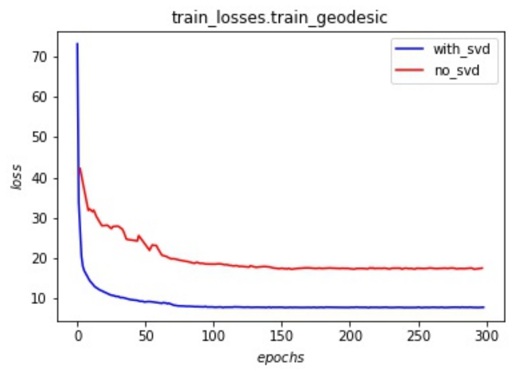
Looking to optimize the networks performance, the first thing was to do a hyperparameter search. Mainly a learning rate search yielded already better results as described in 6.2. Although modifying the learning rate had a positive impact on performance, it wasn't as obvious as when I tried a different loss function. So I defined a new loss function that takes into account both location loss and rotation loss, while in the same time being a meaningful measure, expressed in meters, and a big boost to network performance. The naive joint loss is relatively inefficient when trying to simultaneously learn position and orientation, as well as it doesn't make much sens as physical measure as it adds up two quantities of different nature. The quest for a better loss was inspired by this paper [16] defining a number of loss functions and exploring their efficacy for camera pose regression, especially against the naive joint loss. I implemented my loss after further readings and was inspired by [29] and [1] who also tried to implement a direct regression approach for 6D estimation given a 3D model. According to 6.3.2 defining the Hybrid loss and shown in equation 2 and 3 one gets over the ambiguity of the joint loss by computing the quadratic sum of the image vectors of the 3D models vertices by the computed and label pose. Since the 3d model contains over than 21000 vertices, only a much smaller uniform sample (10% of total vertices) was kept and was already sufficient in getting way better results.



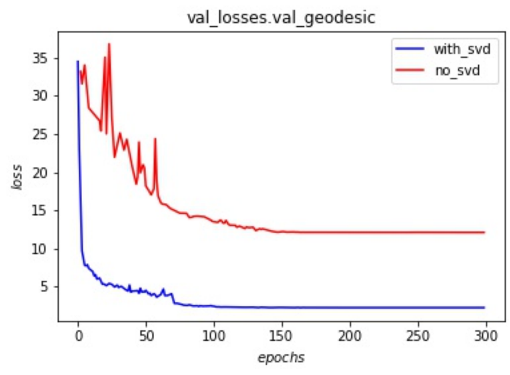
(a)



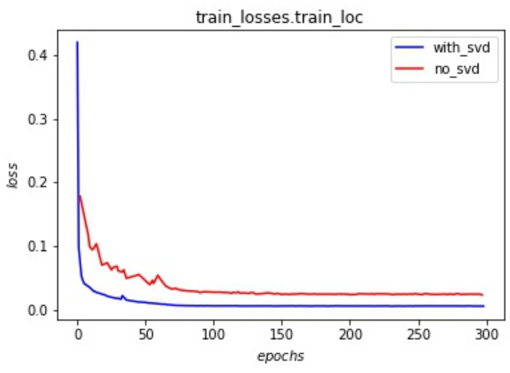
(b)



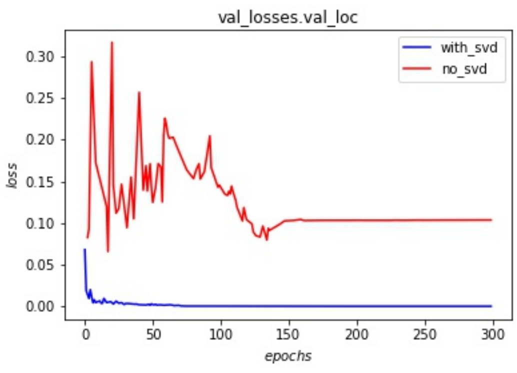
(c)



(d)



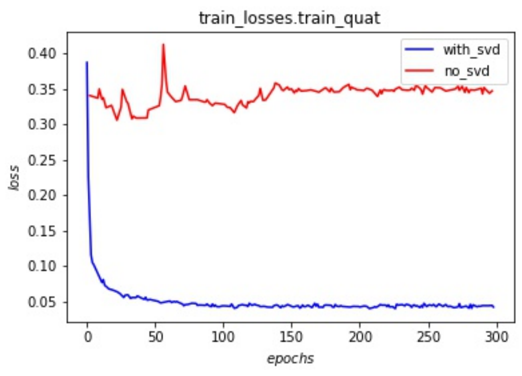
(e)



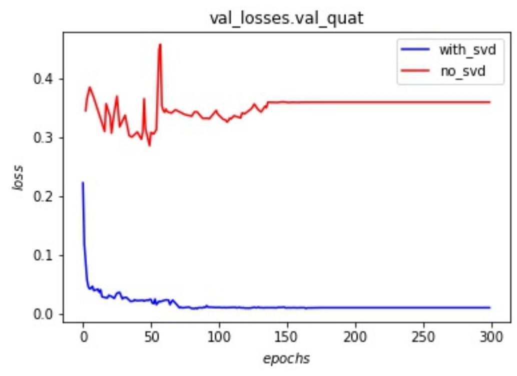
(f)

Figure 11: The SVD model outperforming the no SVD model and scoring the lowest accuracy score during both training and validation on the hybrid loss in meters (a) & (b), the geodesic loss in degrees (c) & (d), and the location loss is meters (e) & (f).

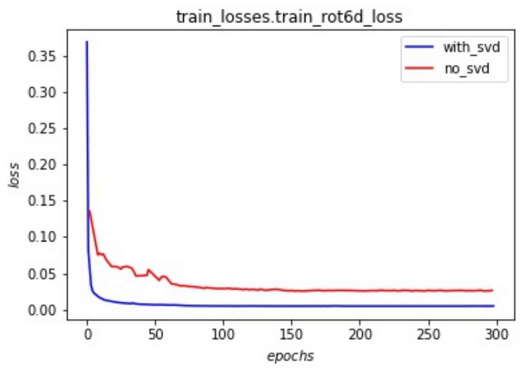
To see the consequent difference in performance, I kept the Quaternion representation and I implemented another version with the SVD orthogonalization block before finale regression. As shown in figure 11 the network implementation without SVD, as the blue line, performs really well and on all measures, compared to the one without the SVD, in red line. The location loss of the SVD model even goes relatively down with a margin during the whole training while it was observed that's it easier to regress location. Even if the losses of both models converge, the one in SVD consistently do so at much lower level and much more smoothly as shown by lack of irregularities on the loss curves contrarily to the one without SVD. The performance of the SVD model in it's own is quiet impressive given the used simple direct regression scheme as it's score pretty low on accuracy. Always on the same figure, it could be seen that the geodesic loss goes below 5 degrees on the validation and training set, while the location loss goes under 6mm on the training and validation set. More specific losses are presented in figure 12, to highlight even more the importance of the SVD orthogonalization. This figures are not meant nevertheless to judge upon the importance of representation, as the the 6d loss presented is just an MSE loss of the 6d representation vector obtained only after directly regressing the quaternions.



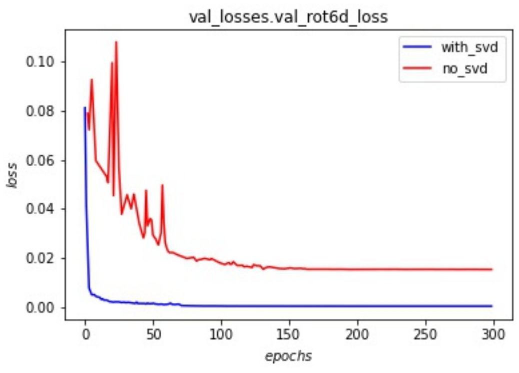
(a)



(b)



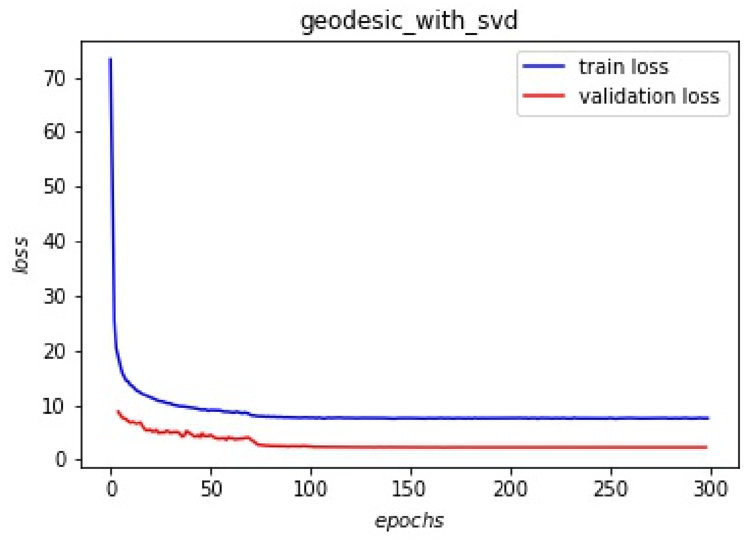
(c)



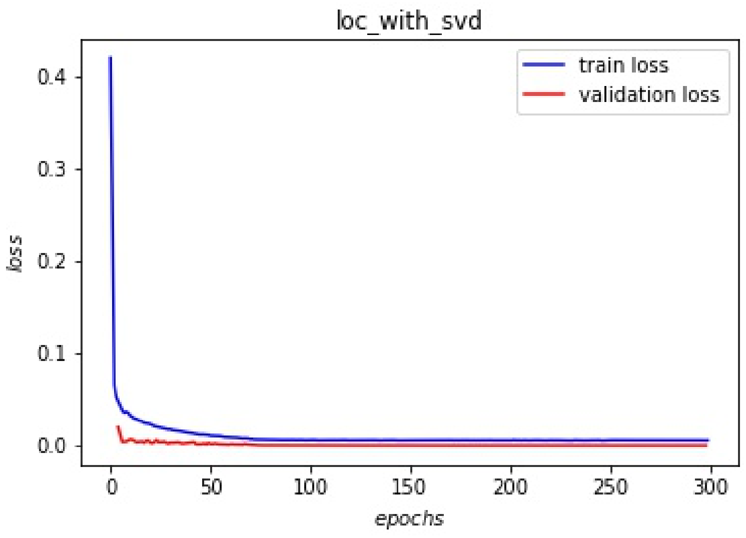
(d)

Figure 12: Comparison of both models during training and validation :
(a)&(b) Quaternion vector MSE loss.
(c)&(d) Quadratic error of the 6d representation vector.

Among all experiments the one using the last described SVD model provides so far the best results. These results also seem to generalize pretty good looking at the graphs in figure 13 when comparing the same loss, location and geodesic, between the validation phase and training phase. There is an apparent gap between the curves, although less in location loss graph as models capacity is big enough to predict the 3d vector of location than the any representation of rotation. In seems therefore that the performance goals are reached both on the level of accuracy or generalization.

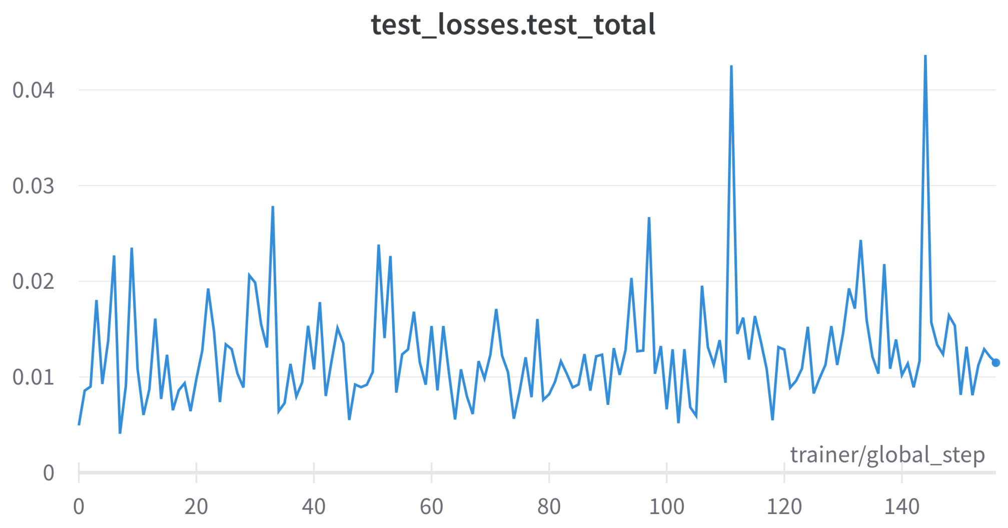


(a) geodesic loss training vs validation



(b) geodesic loss training vs validation

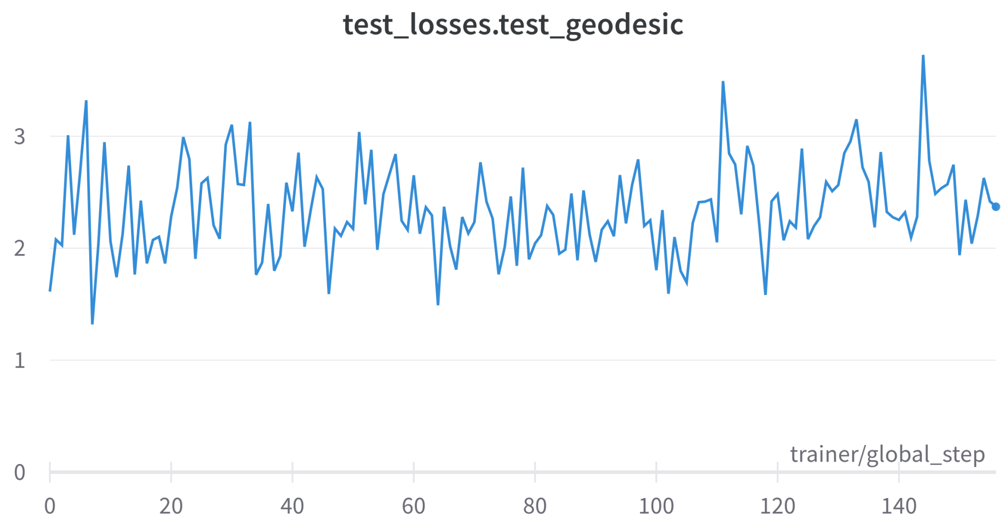
Figure 13: Generalisation of the best results



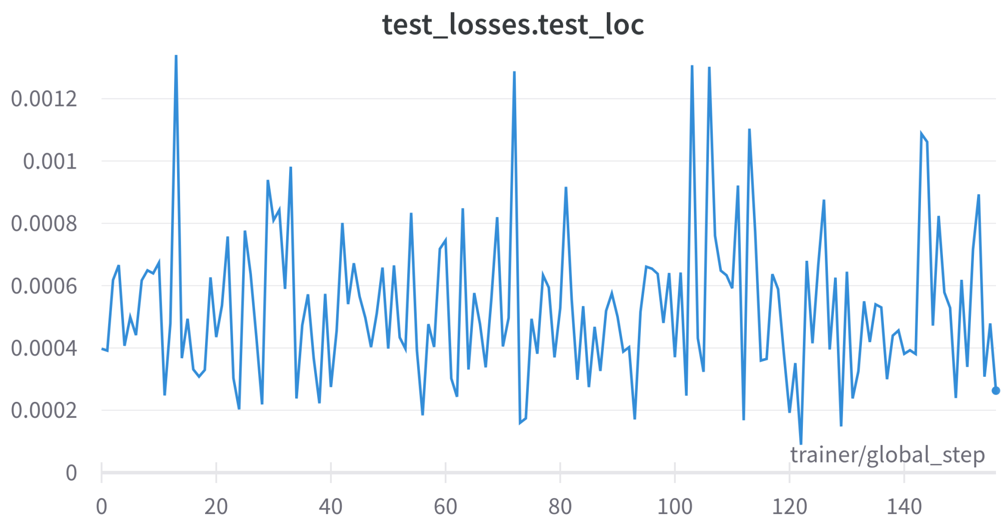
(a) Hybrid loss values on the test dataset subset

Figure 14: of the best model (SVD)

Although it might seem that one can conclude from the graphs that the results are satisfactory for most known applications, it remains still to hard test them further. With this intention the following graphs in figure 15 show the performance of the network on individual images from the test dataset containing in total 8000 items. The values are all contained in a small band between sufficiently small values, similar or lower to results on the validation dataset. The mean test geodesic loss is around 2,5 degrees, when location loss oscillates around 5mm and the total hybrid loss around 1cm.

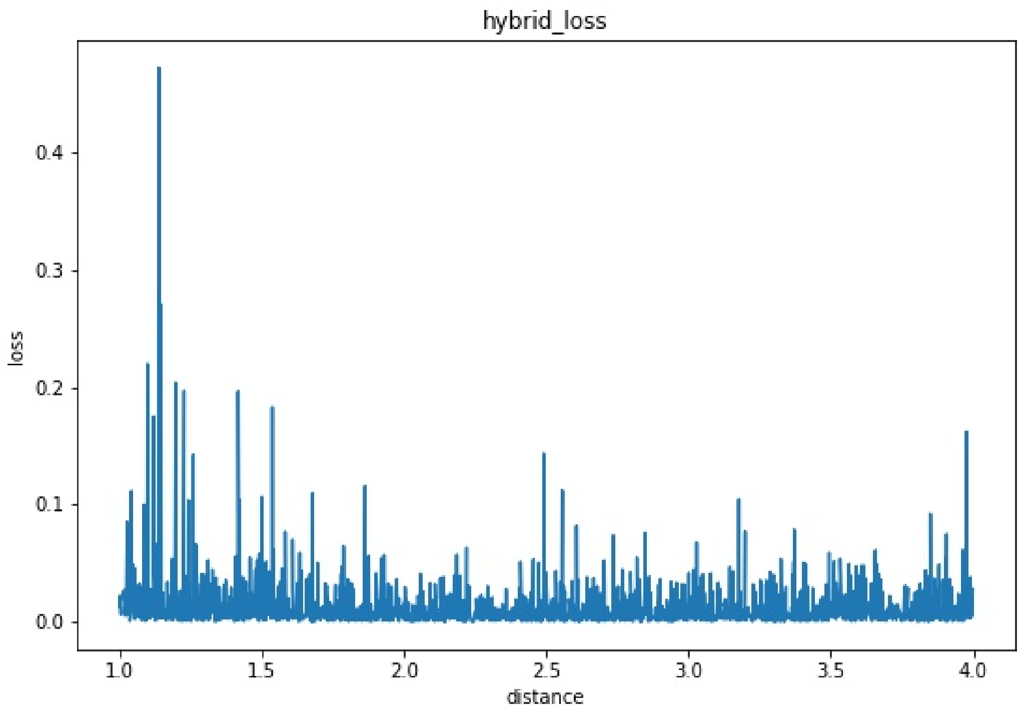


(a) geodesic loss on a test dataset subset



(b) location loss on a test dataset subset

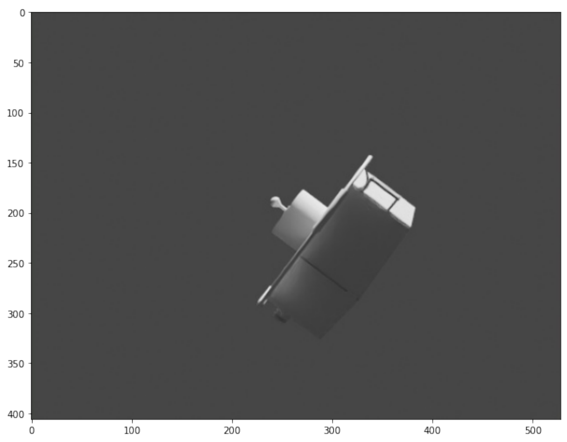
Figure 15: test results of the best model (SVD)



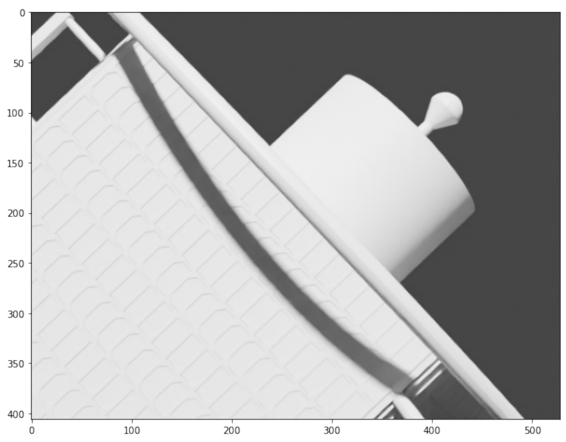
(a) location loss on the test dataset

Figure 16: Test results on the entire test dataset, sorted by distance

After sorting the values of the losses in an ascendant way according to the distance of the camera from the object, we observe the results in figures 18 and 16. As the graphs show the network have a harder time predicting the pose of either close up images or the those where the satellite is furthest. I give in figure 17 examples of these extreme cases.

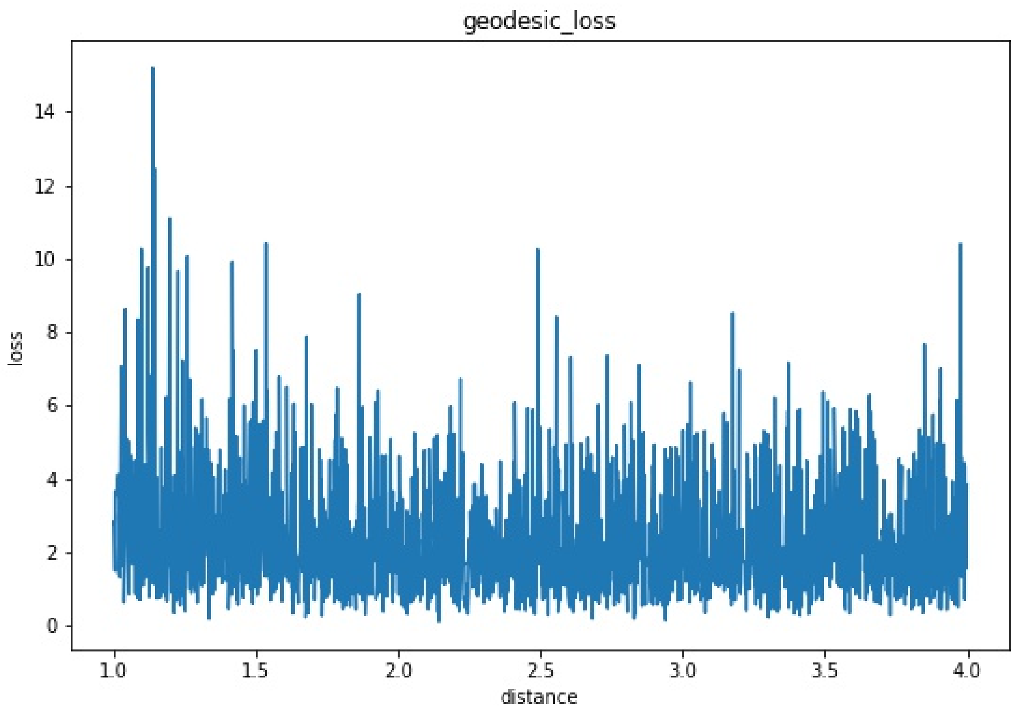


(a) Far away shot :10.4° geodesic loss

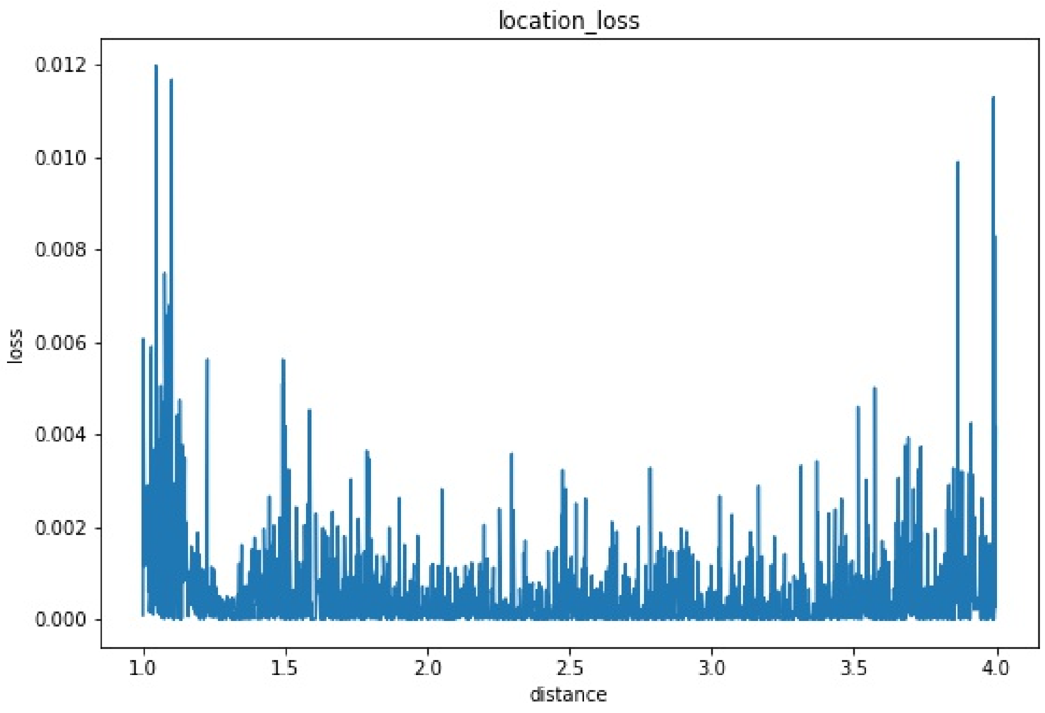


(b) Closeup shot : 15.2° geodesic loss

Figure 17: Hard cases



(a) geodesic loss on the test dataset



(b) location loss on the test dataset

Figure 18: Test results on the entire test dataset, sorted by distance

6.5 Visualisation : Axis projection

To highlight the pose we show the axes projected on the image given camera intrinsic parameters.

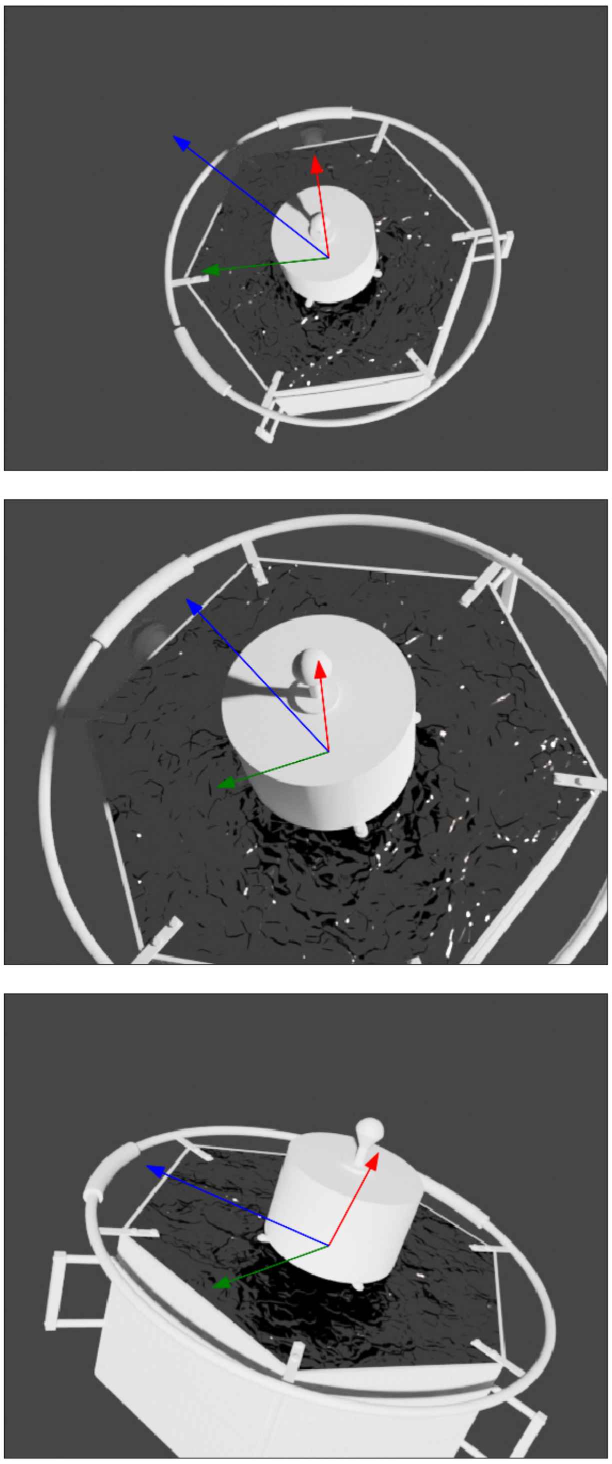


Figure 19: Pose axes on corresponding sample images

7 Conclusion

The goal of this thesis was to discover the Satellite 6D pose estimation problem under the light of recent advances in direct regression methods using deep learning. To compensate for the lack of data given the data hungry deep learning approach, I rendered annotated photorealistic data using an initially DLR internal framework. Then I implemented a direct pose regression method using deep neural networks. The initial goal was to understand the importance of different representations and the crutial role of an intermediate SVD orthogonalization in increasing the performance of the network. A further optimization of the results was carried out and only made the positive performance difference resulting from the latter choices even more relevant. Many trials lead to discovery of new ideas, the implementation of a new loss and a better search of relevant hyperparameters. Good results were obtained on both rotation and location regression metrics.

Bibliography

[1] Arash Amini, Arul Selvam Periyasamy, and Sven Behnke. “T6D-Direct: Transformers for Multi-Object 6D Pose Direct Regression”. In: *CoRR* abs/2109.10948 (2021). arXiv: 2109.10948. URL: https://arxiv.org/abs/2109.10948.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “Surf: Speeded up robust features”. In: *ECCV*. 2006.

[3] *BlenderProc2*. URL: https://dlr-rm.github.io/BlenderProc/index.html.

[4] *Bop Dataset format*. URL: https://github.com/thodan/bop_toolkit/blob/master/docs/bop_datasets_format.md.

[5] Samarth Brahmbhatt et al. “Geometry-aware learning of maps for camera localization”. In: *CVPR*. 2018.

[6] Michael Calonder et al. “Brief: Binary robust independent elementary features”. In: *ECCV*. 2010.

[7] Vincenzo Capuano et al. “Monocular-Based Pose Determination of Uncooperative Known and Unknown Space Objects”. In: Oct. 2018.

[8] Vincenzo Capuano et al. “Robust Features Extraction for On-board Monocular-based Spacecraft Pose Acquisition”. In: *AIAA Scitech 2019 Forum*. 2019, p. 2005.

[9] Lorenzo Pasqualetto Cassinis, Robert Fonod, and Eberhard Gill. “Review of the robustness and applicability of monocular pose estimation systems for relative navigation with an uncooperative spacecraft”. In: *Progress in Aerospace Sciences* (2019). ISSN: 0376-0421. DOI: https://doi.org/10.1016/j.paerosci.2019.05.008. URL: http://www.sciencedirect.com/science/article/pii/S0376042119300302.

[10] Maximilian Denninger et al. “BlenderProc”. In: *CoRR* abs/1911.01911 (2019). arXiv: 1911.01911. URL: http://arxiv.org/abs/1911.01911.

[11] Maximilian Denninger et al. “BlenderProc: Reducing the Reality Gap with Photorealistic Rendering”. In: 2020.

[12] *Distance between rotations.* URL: http://www.boris-belousov.net/2016/12/01/quat-dist/#using-rotation-matrices.

[13] Chad English et al. “Tridar: A hybrid sensor for exploiting the complimentary nature of triangulation and LIDAR technologies”. In: *Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*. 2005.

[14] Du Q. Huynh. “Metrics for 3D Rotations: Comparison and Analysis”. In: *Journal of Mathematical Imaging and Vision* 35.2 (Oct. 2009), pp. 155–164. ISSN: 1573-7683. DOI: 10.1007/s10851-009-0161-2. URL: https://doi.org/10.1007/s10851-009-0161-2.

[15] Alex Kendall and Roberto Cipolla. “Geometric loss functions for camera pose regression with deep learning”. In: *CVPR*. 2017.

[16] Alex Kendall and Roberto Cipolla. “Geometric loss functions for camera pose regression with deep learning”. In: *CoRR* abs/1704.00390 (2017). arXiv: 1704.00390. URL: http://arxiv.org/abs/1704.00390.

[17] Alex Kendall and Roberto Cipolla. “Modelling uncertainty in deep learning for camera relocalization”. In: *ICRA*. 2016.

[18] Alex Kendall, Matthew Grimes, and Roberto Cipolla. “Posenet: A convolutional network for real-time 6-dof camera relocalization”. In: *ICCV*. 2015.

[19] Jake Levinson et al. “An Analysis of SVD for Deep Rotation Estimation”. In: *CoRR* abs/2006.14616 (2020). arXiv: 2006.14616. URL: https://arxiv.org/abs/2006.14616.

[20] David G Lowe et al. "Object recognition from local scale-invariant features." In: *ICCV*. 1999.

[21] Roberto Opromolla et al. "A review of cooperative and uncooperative spacecraft pose determination techniques for close-proximity operations". In: *Progress in Aerospace Sciences* 93 (2017), pp. 53–72. ISSN: 0376-0421. DOI: `https://doi.org/10.1016/j.paerosci.2017.07.001`. URL: `https://www.sciencedirect.com/science/article/pii/S0376042117300428`.

[22] Mustafa Ozuysal et al. "Fast keypoint recognition using random ferns". In: *IEEE transactions on pattern analysis and machine intelligence* 32.3 (2009), pp. 448–461.

[23] Georgios Pavlakos et al. "6-dof object pose from semantic keypoints". In: *ICRA*. 2017.

[24] Sida Peng et al. "PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation". In: *CVPR*. 2019.

[25] Leslie N. Smith. "No More Pesky Learning Rate Guessing Games". In: *CoRR* abs/1506.01186 (2015). arXiv: `1506.01186`. URL: `http://arxiv.org/abs/1506.01186`.

[26] *State of the art : 6D Pose Estimation using RGB*. URL: `https://paperswithcode.com/task/6d-pose-estimation`.

[27] Martin Sundermeyer et al. "Implicit 3D Orientation Learning for 6D Object Detection from RGB Images". In: *ECCV*. 2018.

[28] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. "Real-time seamless single shot 6d object pose prediction". In: *CVPR*. 2018.

[29] Gu Wang et al. *GDR-Net: Geometry-Guided Direct Regression Network for Monocular 6D Object Pose Estimation*. 2021. arXiv: `2102.12145 [cs.CV]`.

[30] Yi Zhou et al. “On the Continuity of Rotation Representations in Neural Networks”. In: *CoRR* abs/1812.07035 (2018). arXiv: `1812.07035`. URL: `http://arxiv.org/abs/1812.07035`.