Acta Astronautica 232 (2025) 231–243

Contents lists available at ScienceDirect

Acta Astronautica

journal homepage: www.elsevier.com/locate/actaastro

Research paper

Deep learning on 3D point clouds for fast pose estimation during satellite rendezvous

Léo Renaut [a,b,*], Heike Frei [a], Andreas Nüchter [b]

[a] German Aerospace Center (DLR), Spaceflight technology, Münchener Str. 20, Weßling, 82234, Germany
[b] Julius-Maximilians-Universität Würzburg, Informatics XVII - Robotics, Am Hubland, Würzburg, 97074, Germany

ARTICLE INFO

Keywords:
Lidar
Rendezvous
Non-cooperative
Pose estimation

ABSTRACT

Light detection and ranging (lidar) is valuable during non-cooperative space rendezvous scenarios. By processing the 3D point clouds, it is possible to provide a navigation solution, consisting of an estimate of the relative pose of the approached spacecraft. To enable a safe rendezvous, the pose estimation has to be precise, but also robust if the output is used as a primary navigation solution. Navigation has to be performed in real-time, and onboard computing hardware has a reduced processing capability. Therefore, the real-time requirement is a main driver of the design. Additionally, a spacecraft often has a symmetrical shape. In this case, the pose estimation method has to account for the fact that multiple attitudes represent the same configuration. This work investigates the use of a point-based neural network, or 3D neural network, for the pose estimation task. This network is integrated in a full pose estimation pipeline, where every component is optimized to achieve real-time requirements on a representative onboard computing hardware. After pre-processing, the neural network produces a relative position and attitude estimation in a single-stage, where the attitude estimation considers the symmetries of the spacecraft. Furthermore, a high-fidelity lidar simulator is used, which enables to generate an extensive synthetic dataset. The method is trained and optimized solely on synthetic data. After training, the pose estimation is evaluated on real lidar data acquired at a hardware-in-the-loop rendezvous facility. Results highlight that the method is accurate and robust, without a loss in performance when evaluated on real data. Finally, the flight-readiness is demonstrated by runtime evaluations on an onboard computer candidate, showing that the method is suited for real-time processing.

1. Introduction

Many modern satellite mission scenarios such as on-orbit servicing, active debris removal or inspection rely on the capability of a spacecraft to autonomously rendezvous with another satellite or object. Recent examples include the servicing of Intelsat telecommunication satellites by the two Mission Extension Vehicles (MEVs) [1], or the inspection flight of a rocket upper stage debris by Astroscale's ADRAS-J spacecraft [2].

A rendezvous involves two satellites, a "chaser" spacecraft which performs the approach, and a passive satellite or object which is referred to as "target". Targets include satellites to repair, refuel, or even a piece of space debris to be removed. It is assumed in the most general case that the target is non-cooperative, *i.e.*, that it does not communicate any information about its relative position and attitude to the chaser, and is not necessarily attitude controlled. In some cases, if the target satellite is inactive, it might be freely tumbling.

In order to rendezvous and perform proximity operations with the target, the chaser needs to estimate the relative position and attitude (pose) of the target spacecraft. Due to real-time constraints, the pose estimation is required to be performed onboard and autonomously. Full pose estimation, *i.e.*, including the relative attitude estimation, is typically only needed in the final phase of the rendezvous, when the inter-satellite distance is small. This distance is dependent on the target size, *e.g.*, it could be below 30 m for a small target, while full pose estimation is already started at larger distances for big targets.

Several electro-optical sensors are used to perform relative pose estimation, including optical cameras, infrared cameras or lidars [3]. Passive sensors such as optical or infrared cameras are sensitive to the illumination conditions: They are blended by direct or reflected sunlight, or not sensitive enough whenever the satellites are in eclipse phase [4,5]. On the contrary, lidars or time-of-flight cameras are active sensors which provide their own illumination source. Hence they are less sensitive to external light sources. A lidar also provides a direct 3D information of the target in form of a point cloud which enables simple and robust position estimation even at large distances [1]. In

* Corresponding author at: German Aerospace Center (DLR), Spaceflight technology, Münchener Str. 20, Weßling, 82234, Germany.
E-mail address: leo.renaut@dlr.de (L. Renaut).

https://doi.org/10.1016/j.actaastro.2025.03.009
Received 20 November 2024; Received in revised form 21 February 2025; Accepted 10 March 2025
Available online 22 March 2025

addition, the point clouds are free of any background, while optical camera images contain stars or portions of the Earth which need to be interpreted as background by the pose estimation method.

To perform pose estimation from lidar point clouds, it is assumed in this work that a 3D model of the target object is known. It is either available prior to the flight, or it is the result of an inspection of the target object in a first phase of the mission. The task of pose estimation resides in matching the "source" point cloud, collected by the lidar sensor in real-time, with the "target" point cloud, which is known and sampled from the 3D model. To estimate the 6 Degrees Of Freedom (DOF) of the relative pose, hand-crafted point cloud association methods are applied [6–12], but usually come with a high computing time which is not adapted for real-time processing.

The emergence of deep learning methods has led to significant improvement in terms of precision and efficiency for point cloud matching [13–16]. Yet, some neural network architectures rely on high performance computing hardware such as Graphics Processing Units (GPUs) which might not be available for such missions. In addition, real point cloud data contains important disturbances such as motion blur, reflections or outliers [17,18]. Hence deep learning methods trained on synthetic data simulated under ideal conditions might encounter difficulties when evaluated on real data, a problem known as "domain gap". Moreover, spacecrafts often have a symmetrical shape. In this case, the attitude of the target cannot be estimated unambiguously, and neural networks have difficulties estimating the relative pose [19].

This work introduces a deep learning based pose estimation architecture, tailored specifically for efficiency and real-time capability on computing hardware with limited processing power. The 3D point clouds are processed by a point based neural network model which is optimized to keep a similar precision while achieving better computational efficiency than the original model for this application. Using the attitude classification strategy developed in our previous work [16], the method accounts for potential symmetries of the target spacecraft. The network is trained on a fully synthetic dataset, using a lidar simulator which models some common disturbances and reflection effects that are observed on real data [16]. After training, evaluation is performed on a real lidar dataset generated at a hardware-in-the-loop rendezvous facility, the EPOS [20]. The runtime is assessed on flight representative computing hardware. The results highlight that the method is real-time capable, robust and able to generalize to real data without a performance loss.

The paper is structured as follows: Section 2 reviews the state-of-the art, and the proposed pose estimation method is detailed in Section 3. In particular, the neural network model optimization and the pose estimation logic involving the handling of symmetries are introduced. Also, the training method relying on a lidar simulator to generate synthetic data is described, as well as the hardware-in-the-loop test dataset collected at the EPOS test facility. The results of the different runtime evaluations on flight representative hardware and on the real lidar datasets are presented in Section 4, and discussed in Section 5. Finally, conclusions are drawn in Section 6.

2. Related work

The first lidar sensors for space rendezvous were developed in the context of the Space Shuttle program for rendezvous and docking with the International Space Station (ISS). Using the TriDAR sensor, Ruel et al. [6] designed a polygonal aspect hashing algorithm. It implies to extract sets of points from the source point cloud forming polygons, and to match them with a database of polygons of the target point cloud, which is searched through using a hashing method. The optimal pose is found after applying a Random Sample Consensus (RANSAC) voting scheme on the different matches. A similar algorithm for matching tetrahedrons was developed by Yin et al. [7]. Instead of RANSAC, a pose refinement is performed with Iterative Closest Point (ICP) [21] for every match, and only pose candidates with a sufficient score are retained. Likewise, Klionovska and Frei [8] apply a method to match pairs of oriented points when using a time-of-flight sensor for pose estimation.

A possibility is to try to find the global ICP optimum by the application of a branch-and-bound ICP [22]. This approach is applied to spaceborne pose estimation by Liu et al. [23], but is computationally expensive. Alternatively, Woods and Christian [9] suggest characterizing a point cloud using a single point cloud descriptor belonging to the family of point feature histograms [24]. The descriptor is then matched with an offline built database to retrieve the relative pose. Opromolla et al. introduce a model-based template matching method [10]. It consists in computing the ICP distance metric between the source point cloud and a database of point cloud views from the target seen from different orientations. The candidate with the minimum score is selected, before ICP refinement is performed. In follow-up work [11], the authors accelerate the method using Principal Component Analysis (PCA) to select the principal axis of the spacecraft. A similar approach is applied by Jasiobedzki et al. [25] to match a 3D point cloud constructed from stereo images with a template point cloud. Guo et al. [12] suggest accelerating the template matching by projecting the 3D point clouds to 2D binary silhouette images. The matching score between the silhouette image of the source point cloud and the images of the database is then a binary distance metric which is computed efficiently.

The initial pose estimate might be coarse, and is usually refined with ICP or one of its variants [3]. An alternative to ICP for pose refinement is the NDT [26], consisting in a probabilistic matching of the source and target point clouds. In previous work [18], we introduced a smoothed version of the NDT algorithm which shows better efficiency and robustness compared to ICP for satellite pose tracking.

With the popularization and success of neural networks, hand-crafted features and databases are being replaced by learnable features or matching logics. In this sense, Schmitt et al. [13] introduce a neural network aided polygon matching algorithm. The idea is to extract a polygon from the source scan, and to train a neural network to retrieve the corresponding polygon in the target coordinate frame. For efficiency, the neural network is implemented on a Field Programmable Gate Array (FPGA). Despite this acceleration, several polygons must be evaluated until a satisfactory match is found. This involves several ICP iterations each time, which is time-consuming. To directly estimate the parameters of the relative pose without additional matching steps, Pensado et al. [14] suggest using a fully connected neural network, also called Multilayer Perceptron (MLP). To do so, the source point cloud is first projected to a 2D depth image, before this image is flattened and processed by the MLP.

Yet, the usual way to process 2D images with neural networks is to use Convolutional Neural Networks (CNNs). Very efficient architectures have been developed, and are being extensively studied for camera based pose estimation [5]. Feature-based approaches rely on identifying features on the 2D image before applying a Perspective-n-Point (PnP) algorithm to retrieve the relative pose [27–30]. On the other hand, in the direct approach, the CNN is trained to directly predict the relative pose parameters [31–33]. Because a 3D point cloud projects to a 2D depth image, we have investigated the use of CNNs for lidar based pose estimation in previous work [16]. The scaled 2D depth image is processed by a CNN using MobileNet [34] as a backbone, to directly estimate the relative position and perform an attitude classification. Instead of processing a single frame at a time, some authors make use of recurrent CNNs to process a sequence of lidar depth images in the context of rendezvous [35] or lunar landing [36]. However, the computing requirements are such that a GPU is required for real-time processing.

In recent years, neural network architectures have been developed for direct 3D point cloud understanding. A first possibility is to use 3D convolutions on a voxelized representation of the point cloud [37]. Because the complexity explodes when the resolution of the voxel grid increases, concepts of efficient and learnable point-wise convolutions

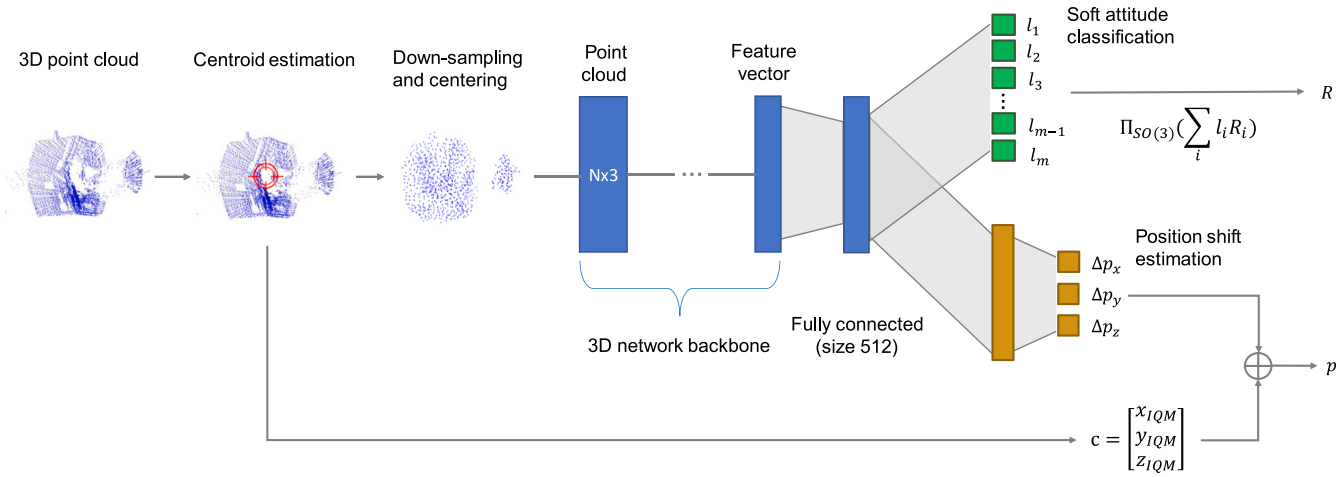


Fig. 1. Overview of the point cloud processing pipeline for pose initialization.

have been developed [38,39]. A major evolution in the field of 3D point cloud understanding was the introduction of PointNet [40]. This network processes each point of a set individually using a shared MLP, before a global max pooling operation on all features enables to interpret the relations between points. To be able to capture finer details, the follow-up architecture, PointNet++ [41], is a hierarchical stacking of multiple PointNets. To further exploit the relationships between neighboring points, Dynamic Graph CNN (DGCNN) [42] introduces a graph architecture. The graph connects each point with its k-Nearest Neighbors (k-NN), and the network applies a shared MLP to process each point with its associated edges, forming an edge feature. A recent interesting approach is PointTransformer [43], which successfully applies the self-attention layer of Large Language Models (LLMs) to exploit and learn relationships between neighboring points.

For lidar based pose estimation during rendezvous, Zhang et al. [44] introduce a hybrid algorithm mixing a classical ICP matching with a deep learning method. The point-to-point association step of ICP is performed by a PointTransformer network. Likewise, in their next work [15], the authors implement a mixture between a deep learning model, and another classical local matching method, Coherent Point Drift (CPD) [45]. This time, the network features layers from both DGCNN and PointTransformer. It is used at each iteration to assign the points of both the source and target point clouds to their associated component in a Gaussian Mixture Model (GMM) representation, before finding the transformation which maximizes the likelihood for this model. The two methods are evaluated on simulated datasets of several spacecrafts. However, they involve to run a neural network inference at each iteration of the algorithm, which is computationally expensive.

Given the rapid development of neural networks for 3D point cloud understanding, we contribute in this work to apply a point-based neural network to the problem of lidar pose estimation during space rendezvous. Unlike other approaches, we propose to train such a neural network to directly estimate the relative pose of the target spacecraft, in the form of a position estimation and an attitude classification. Furthermore, we focus on optimizing the runtime of the pose estimation method in order to achieve real-time performance on space hardware.

3. Methods

3.1. Pose initialization pipeline

The pipeline for pose initialization is illustrated on Fig. 1. The overall logic is inherited from [16], with the difference that a 3D neural network is used instead of a 2D CNN. The pre-processing is also different, as the network directly processes 3D point clouds. The different steps will be detailed in the following.

3.1.1. Point cloud pre-processing

The first step of the pre-processing is to estimate the centroid of the point cloud, which will give a first coarse position estimate of the target. To have an estimation resilient to potential outliers in the point cloud, the centroid is computed as a trimmed mean with a trimming factor of 25%, as in [16]. The trimmed mean with a factor of 25% is also known as interquartile mean. Given n sorted values $a_1 \leq \cdots \leq a_n$, the interquartile mean a_{IQM} is defined as the mean of all values remaining after having discarded the first and last quartile of the data:

$$a_{IQM} = \frac{2}{n} \sum_{i=n/4}^{3n/4} a_i \,. \tag{1}$$

For a point cloud with n points with coordinates $(x_i, y_i, z_i)^T$ for $i = 1, \ldots, n$, the first step is to compute the interquartile mean along each coordinate x_{IQM}, y_{IQM} and z_{IQM}. The trimmed centroid is then defined as

$$\boldsymbol{c} = \begin{pmatrix} x_{IQM} \\ y_{IQM} \\ z_{IQM} \end{pmatrix} . \tag{2}$$

The trimmed mean is only used to get a coarse estimate of the satellite's position, which will be corrected in the next steps. While the estimate is different from the one of a standard centroid, the rejection of outliers enables to have a more robust and consistent estimate of the main location of all points, independently of outliers such as double reflections.

Having determined the centroid, the point cloud is shifted to be centered around zero, and only the points within a so-called Region Of Interest (ROI) are kept. The target considered in this work, which is the spacecraft illustrated on Fig. 2, has a hexagonal body with 1.6 m diameter, and a length of approximately 1.8 m. It would fit in a bounding box of 2 m side, but to ensure that no points are cropped, the ROI is defined as a cubic bounding box of 4 m side. Through the centering and ROI selection, the point cloud undergoes a normalization which is helpful in neural network training [46]. In addition, discarding points outside the ROI enables to filter out obvious outliers or ghost reflections, effects which are observed with lidar sensors [17,18].

While point based neural networks are theoretically able to handle varying input sizes, a standard practice for computationally efficient training through batching is to define a fixed point cloud size [40,41]. However, the data registered by the lidar might have a variable number of points. A typical input size for point-based neural networks is 1024, which is used as a baseline for many models when evaluated on benchmark datasets [40–43]. This size will be used as point cloud size for the neural network in this work. If the source point cloud contains more than 1024 points, which is expected to be the most common

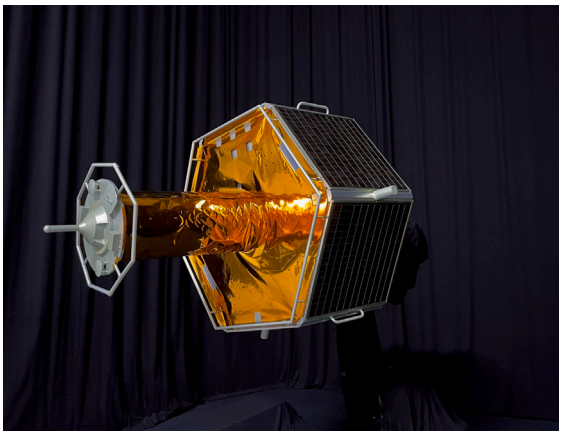


Fig. 2. View of the full-size satellite mockup mounted at the EPOS facility.

case, the point cloud needs to be down-sampled before being passed to the neural network. In the opposite case, we randomly duplicate points to ensure that the input size is 1024. The lidar used in this work provides a dense point cloud, similarly to current space lidars [47,48], so that downsampling is more frequent than having to duplicate points. However, if the sensor is expected to provide a very sparse point cloud, the model input size should be reduced to avoid unnecessary computations.

A standard method for point cloud down-sampling in deep learning is Farthest Point Sampling (FPS). It is used in most neural network architectures [40–43]. Compared to other methods such as random sampling, FPS produces better results when training and testing the models [49]. However, it is computationally intensive, especially if the input point cloud is large. Even when using a GPU, the sampling step with FPS can take up a large part of the total processing time [49]. Therefore, we suggest using an accelerated version of the algorithm introduced by Han et al. QuickFPS [49]. It is a k-d tree based version of the FPS algorithm, and is particularly adapted for implementation on a CPU.

3.1.2. 3D neural network backbone optimization

The input point cloud is processed by a neural network backbone in order to estimate the relative pose. The considered neural networks produce a feature vector as output, which is then typically used for a classification task. In our case, this feature vector will be further processed to estimate the relative position and attitude of the target. The feature vector, which is of different sizes depending on the backbone model, is further processed by one Fully Connected (FC) layer of size 512, as illustrated on Fig. 1. Finally, the network is split in two output heads to estimate both the position and the attitude. These output heads will be described in detail in the following Sections (3.1.3 and 3.1.4).

We evaluate different point-based neural network backbones for this task. All networks were re-implemented in Python using the TensorFlow library. For the model comparison, PointNet [40], PointNet++ [41], DGCNN [42] and PointTransformer [43] are compared. The results of the evaluations (Section 4.3) highlight that PointNet++ is the most promising network in terms of accuracy for this application. Therefore, we will look into this model more deeply in order to see how the model runtime can be optimized.

PointNet++ is a hierarchical stacking of mini-PointNet models. PointNet processes input points of size $N \times c$, where c is the number of channels, which is typically 3 in the beginning for the 3 spatial coordinates. Then, each point is processed by the same shared MLP with multiple layers to obtain a point cloud of size $N \times f$, where f is the number of output features. Finally, a max pooling operation selects, for every feature coordinate, the maximum value of all points [40]. After this pooling, the output is a flattened feature vector of size f.

Table 1
Layers of the PointNet++ classification model.

Layer name	Output size
input scan	$N \times 3$
clustering	$N_1 \times k_1 \times (3+3)$
MLP {64, 64, 128}	$N_1 \times k_1 \times 128$
max pool	$N_1 \times 128$
clustering	$N_2 \times k_2 \times (3+128)$
MLP {128, 128, 256}	$N_2 \times k_2 \times 256$
max pool	$N_2 \times 256$
MLP {256, 512, 1024}	$N_2 \times 1024$
max pool	1024

These operations are also at the basis of the PointNet++ model [41]. Its layers are listed in Table 1. The different operations are:

- Clustering: The input is a point cloud of size $N_i \times f$, where f is the feature size of each point, which additionally has 3 position coordinates (in the beginning, the features equal the coordinates). Given a fixed number of clusters $N_e < N_i$ and a value k for the k-NN algorithm, the clustering applies FPS to select N_e sample points. For each sample point, the k-NN are selected to form a cluster. Finally, the 3D position of each point of a cluster relatively to the center of the cluster is concatenated with the point feature. Thus, the output consists of N_e clusters of k points with each $(3+f)$ features.
- MLP{...} + max pool: This sequence is a mini-PointNet model (with a smaller MLP) applied separately to each cluster of points. The sizes between the brackets indicate the sizes of the hidden and output layers. Given an input of size $N \times k \times c$, each cluster of size $k \times c$ is seen as a small point cloud processed by a mini-PointNet. The output, after max-pooling, of processing a cluster with a mini-PointNet is a feature vector of size f. By aggregating the feature vectors for all clusters, the overall output of this sequence has dimensions $N \times f$.

In the original PointNet++ classification model [41], the input has $N = 1024$ points. Additionally, the number of clusters at each clustering step given in Table 1 are $N_1 = 512$ and $N_2 = 128$. The clusters have sizes $k_1 = 32$ and $k_2 = 64$. To optimize the model for runtime, an approach resides in optimizing these cluster sizes N_1, N_2, k_1, k_2, in order to reduce the number of operations performed by the network. However, reducing the number and size of the clusters translates to the neural network using less and smaller subregions of the point cloud to detect patterns. With less information a decrease in performance occurs.

Additional directions are explored to further simplify the network, such as reducing the size of the three MLP layers of PointNet++, or removing some layers. The result of this optimization is presented in Section 4.4. It seeks to achieve real-time efficiency while maintaining a good performance in terms of position and attitude error for the pose estimation task.

3.1.3. Position estimation

The first prediction head of the network is charged with predicting the position of the target's center of mass relative to the chaser, which is denoted by $\boldsymbol{p}$. In the pre-processing step, the centroid $\boldsymbol{c}$ of the point cloud was already computed (2), and the point cloud was centered around $\boldsymbol{c}$. Hence the network only needs to predict the remaining position shift

$$\Delta \boldsymbol{p} = \boldsymbol{p} - \boldsymbol{c}\,. \tag{3}$$

The first layer of the top network (rightmost blue layer in Fig. 1) is of size 512. The position shift is obtained by processing this layer by two fully connected layers of size 128 and 3, illustrated in orange. The 3 coordinates are then passed through a tanh activation

function [50], followed by a scaling by $l/2$, which is half the length of the bounding box. This activation allows for negative values in the position prediction, but ensures that the shifted position estimate $\Delta \boldsymbol{p}$ will be within the ROI, as each of its coordinates is within $[-l/2, l/2]$.

3.1.4. Attitude classification for a symmetrical target

The second prediction head of the network is tasked with estimating the attitude of the target satellite relatively to the chaser. The rotation matrix R will represent this attitude. A first idea would be to directly estimate the parameters of the attitude matrix, or the parameters of a lower dimensional representation of the attitude. Such a representation should have at least 5 parameters [51], as lower dimensional representations are discontinuous, and discontinuous outputs are difficult to learn for neural networks.

However, in the case of a symmetrical target object, one configuration of the target might be equivalently represented by several attitudes. Therefore, only one attitude should be considered valid for each configuration. This introduces an additional discontinuity between the inputs and outputs, as nearby input point clouds are not represented by nearby output rotation matrices anymore [19]. To ensure that the neural network learns a continuous function, we suggest using a classifier to estimate the attitude of a symmetrical target. In this way, the discontinuity is isolated in the post-processing step of the classifier scores. The classifier is used for a target with or without symmetries, and is identical to [16]. To make the paper self-contained, we recall overall logic here.

The first step is to divide the attitude space into a discrete number of bins M. The sampling is performed with an angular resolution $\Delta\alpha$. Using a spiral sampling strategy [52], care is taken to have bins which sample the attitude space uniformly. Given $\Delta\alpha$, the sampling produces discrete attitudes $R_1, \dots, R_M$, obtained by following the logic detailed in [16]. Similarly to [32], the attitude R is then encoded by the labels vector

$$L(R) = \begin{pmatrix} l_1 \\ \dots \\ l_M \end{pmatrix}, \quad \text{where } l_i = K \exp\left(-\frac{d(R, R_i)^2}{2\sigma^2}\right). \tag{4}$$

Here, K is a scaling constant such that $\sum_i l_i = 1$. The standard deviation σ is dependent on the angular resolution of the attitude sampling, $\Delta\alpha$, and chosen such that $\sigma = \frac{2}{3}\Delta\alpha$. We justified this value empirically in [16], finding that it leads to a low conversion error when encoding an attitude into a weights vector, before decoding it. Finally, d is the angular distance function between two attitudes. Given two attitude matrices R, Q, it is defined as

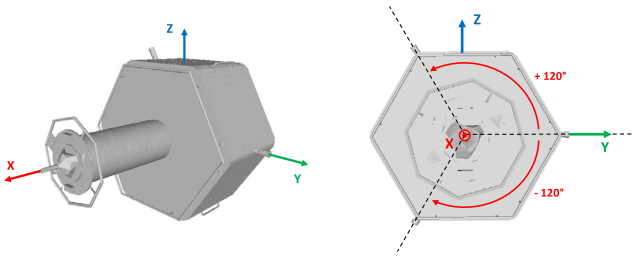
$$d(R, Q) = \arccos\left(\frac{\mathrm{tr}\,(R^T Q) - 1}{2}\right). \tag{5}$$

Eq. (4) shows how at training time, the labels that the network learns are obtained from the ground truth R. However, at test time, the inverse transformation is required. The network produces a labels vector $(l_1, \dots, l_M)^T$, which needs to be transformed into an attitude matrix to estimate the relative attitude. This is achieved by computing the orthogonal projection of the weighted sum of the attitude samples on the 3D rotation group $SO(3)$, as in [31]:

$$R = \Pi_{SO(3)}\left(\sum_{i=1}^{M} l_i R_i\right), \tag{6}$$

where $\Pi_{SO(3)}$ is the projection function.

For now, the symmetries of the target satellite have not been considered. In case the target presents no symmetries, no further modifications are needed. However, in this work, we consider as a use case the symmetrical spacecraft illustrated on Fig. 2. The coordinate system of the target is defined on Fig. 3(a). As illustrated on Fig. 3, when viewed from the front, the main part of the spacecraft body has a hexagonal shape with antennas on three of the corners. Therefore,



(a) (b)

Fig. 3. **(a)** Definition of the target coordinate frame. **(b)** Front view of the target, showing a symmetry around the roll axis X.

a rotation around the roll axis X by ± 120 deg leaves the spacecraft unchanged from an observer point of view.

Given one input attitude of the target R, from an observer point of view, three attitudes $R^{(1)}, R^{(2)}, R^{(3)}$ equivalently represent this configuration. Only one of them corresponds to the actual attitude R. We say that these attitudes are the symmetrical equivalents of R and write $R^{(1)} \sim R^{(2)} \sim R^{(3)}$.

The distance function (5) is modified to define the symmetrical distance function

$$d_{sym}(R, Q) = \min\left(d(R^{(1)}, Q), d(R^{(2)}, Q), d(R^{(3)}, Q)\right). \tag{7}$$

This distance function is continuous, and it holds $d_{sym}(R, Q) = 0$ if and only if $R \sim Q$.

Due to the symmetries of the target, we remove all symmetric equivalents from the M attitude classes initially defined. These are the attitudes with a roll angle outside the range [0 deg ,120 deg]. The new number of attitude classes is $m = M/3$. With the reduced number of attitude classes, the expression of the labeling function (4) is modified to replace the classical distance function by the angular distance function:

$$L_{sym}(R) = \begin{pmatrix} l_1 \\ \dots \\ l_m \end{pmatrix}, \quad \text{where } l_i = K \exp\left(-\frac{d_{sym}(R, R_i)^2}{2\sigma^2}\right). \tag{8}$$

Given an attitude labels vector, the attitude estimation function (6) is also modified. The first step consists in finding the maximum index of the vector

$$i_{max} = \underset{i=1,\dots,m}{\mathrm{argmax}}\, l_i. \tag{9}$$

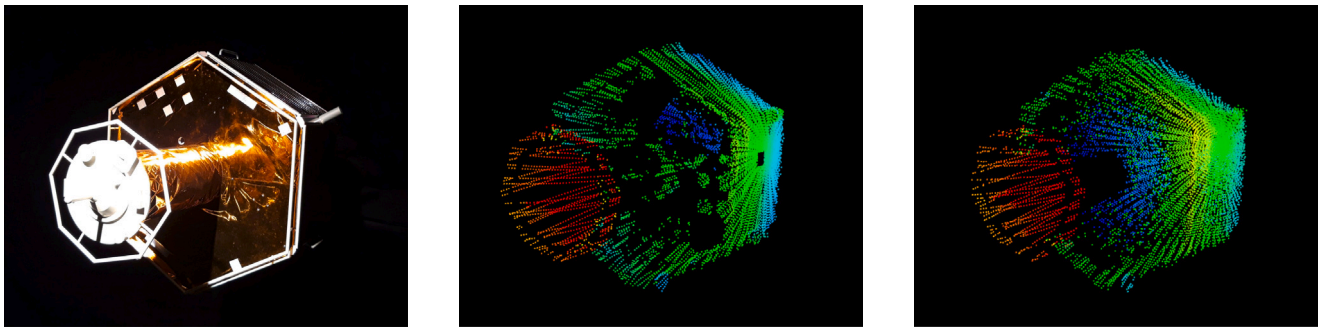
The corresponding attitude $R_{i_{max}}$ will serve as reference. For each other attitude sample R_i, the symmetric equivalent $R_i^{(j_i)}$ closest to $R_{i_{max}}$ is chosen, *i.e.*, the attitude $R_i^{(j_i)} \sim R_i$ such that $d(R_i^{(j_i)}, R_{i_{max}}) = d_{sym}(R_i, R_{i_{max}})$.

Finally, given an angular threshold β, the attitude estimated from a labels vector is

$$R = \Pi_{SO(3)}\left(\sum_{\substack{i=1,\dots,m \\ d_{sym}(R_i, R_{i_{max}}) < \beta}} l_i R_i^{(j_i)}\right). \tag{10}$$

The threshold β enables to discard attitude samples located too far away from the reference attitude, and to avoid ambiguous cases where the network would be undecided between two possible attitudes. In the experiments, it is set to $\beta = 50$ deg.

In summary, the attitude prediction head is tasked with predicting a "soft labels" vector L_{sym}. For the experiments of this work, a sampling step $\Delta\alpha = 20$ deg is chosen, which corresponds to a number of attitude classes $m = 618$. The attitude prediction layer is thus a fully connected layer between the last shared layer of dimension 512 and this layer. Finally, "softmax" activation [50] is applied to ensure that the labels



(a) (b) (c)

Fig. 4. Comparison of real and synthetic point clouds: **(a)** Image of the satellite mockup at the rendezvous facility. **(b)** Real point cloud registered by the Livox® lidar at the facility. Warm colors represent points closer to the sensor. **(c)** Synthetic point cloud simulated by the lidar simulator for the same pose and integration time.

are normalized. The output L_{sym} predicted by the neural network is a continuous function of the possible input target poses. This facilitates the training of the network. However, the attitude predicted by the network after applying the projection (10) is a discontinuous function. Given the target attitude R, the network prediction Q is only ensured to be symmetric equivalent of R, such that $Q \sim R$.

The proposed pose estimation method for a symmetrical object does not intend to deliver an unambiguous estimation of the relative attitude, but only to find one possible symmetrical equivalent of the attitude. In case the object symmetries are perfect, this is the only alternative: it is not possible to find the attitude unambiguously only from 3D data. On the contrary, if the satellite is not perfectly symmetrical, further disambiguation strategies might be explored, but are not within the scope of this work.

3.2. Datasets for domain gap bridging

Deep learning methods need a substantial amount of data to be trained properly without overfitting, *i.e.*, without loosing their ability to generalize to unseen data. Moreover, in the scenario of a rendezvous mission, real lidar data of the target might not be available prior to the flight. Therefore, the approach developed in this work consists in training the method only on simulated data. Because synthetic data is generated on demand, this ensures that the method is easily deployable, and that as much data as required is used for training.

To demonstrate Simulation To Reality (Sim2Real) transfer, *i.e.*, to show that the method is able to generalize to real lidar data, the test dataset is a real dataset gathered at the EPOS rendezvous facility with a scanning lidar. The datasets used in this work are the same datasets that were used to train and test the CNN-based pose estimation method developed in [16]. This enables to compare both methods under the same conditions. The main characteristics of both the synthetic and the real lidar datasets are summarized in the following.

3.2.1. Synthetic training dataset

If synthetic data is simulated under too ideal conditions, then the "domain gap" with the real data is important. In this case, a learning based method might be unable to produce good results on a real dataset, even though it performs well on synthetic datasets. For Sim2Real, an advanced lidar simulator was developed in previous work [16]. It is designed specifically to model the effects that arise when scanning a target spacecraft with a lidar in close range. Its main characteristics are listed in the following.

The lidar simulator is implemented as a ray tracer. Given a 3D model of the target, rays emerging from the sensor are propagated to see if they cross the 3D mesh. At first, the sensor properties need to be modeled. Because the lidar used to collect the test datasets at the EPOS facility is a Livox® Mid-40, this sensor's scan pattern needs to be replicated. The characteristics of the non-repetitive rosette scan pattern are reproduced and implemented following the modeling suggested by Brazeal et al. [53]. This lidar and scan pattern are taken as a use case in this work, but the simulator is implemented in a generic way. If needed, it is possible to simulate the scan pattern corresponding to a different sensor.

In addition to random noise and measurement errors, the effects of laser beam divergence is observed on point clouds. The laser being not perfectly directive, it emits not a perfect ray but rather a light cone with a very small aperture. Because this light cone hits surfaces located at different distances, the point estimated by the sensor is in between those distances. To model beam divergence, similarly to [54], we simulate that several "perfect" rays are emitted within the laser light cone. The measured distance is then defined as the intensity weighted average of the different distances measured by the rays.

Compared to a visual camera, a lidar sensor has a high exposure time, the scan time, which is in the order of a second. Therefore, especially for a fast tumbling target, motion blur is an important effect [18]. Motion blur describes the distortion of the point cloud due to the relative motion between both satellites during the scan. This effect is modeled relatively simply in the simulator, by assuming that every ray emitted by the sensor is emitted from a different location, dependent on the relative velocity and angular rate.

Satellites are often coated with very reflective material for heat shielding, such as golden Multi-layer Insulation (MLI) sheets. Other components of the satellite, such as solar panels, have a high reflectance. When the laser is reflected and hits other surfaces, the lidar registers erroneous measurements and ghost reflections [17,18]. Therefore, the reflectivity of the surfaces is taken into account using a simplified Phong model [55,56]. Instead of propagating a ray in all possible directions when a surface is hit, the simulator only propagates the refracted ray. It also assumes that a ray is reflected at most once. While this is a strong simplification, the lidar simulator is experimentally found to generate synthetic point clouds which are similar to the point clouds gathered at the hardware-in-the-loop facility, as illustrated on Fig. 4.

The real point cloud shown in Fig. 4(b) contains 9679 points, while the synthetic point cloud simulated for the same pose and scan time (0.9 s) contains 11 362 points. This is mostly explained by the difference between the properties of the real and simulated MLI: on the real point cloud, more points which would be located on the MLI become invisible to the sensor compared to the simulated point cloud. In addition, the MLI is folded, while it is assumed to be flat in the simulator. To quantify the similarity between both point clouds, we use the Chamfer distance [57], which is the average of the distance of each point of one cloud with its closest point in the second cloud. The Chamfer distance between the real and simulated point clouds in Fig. 4 is of 4.6 cm. As a comparison, the distance between two real point clouds rotated with respect to each other by 1 deg is of only 1.4 cm. The higher distance between the real and simulated point cloud is explained by the difficulty to model reflections accurately.

The reflectivity properties of the materials on the target cannot be exactly known, and the modeling of the reflection effects is imperfect. Therefore, following a "domain randomization" [58] approach, the reflectivity properties of each material on the target are randomized within a certain range. The baseline for these values is derived from the satellite materials reflectivity analysis of Nakajima et al. [59], and we provide further detail of the range in [16]. For each new simulated point cloud, the material properties are different. At test time, the real properties of the materials will only be one of the multiple variations of these properties learned by the model during training.

Using the lidar simulator, an extensive synthetic dataset of 500 000 point clouds is generated. This dataset is used for training pose estimation in close range. The target is positioned at a random range to the chaser, selected uniformly between 2.5 m and 20 m, and the attitude is randomized ensuring a uniform distribution in the attitude space. The position within the field-of-view of the sensor is also selected at random, following a normal distribution centered around the sensor's principal direction. The standard deviation of the angle between the target's position and the lidar's principal direction is a third of the sensor's field-of-view. Finally, for simulating motion blur, the relative velocities during a scan are bound to maximally 3 cm/s and 5 deg/s. The dataset is split into 400 000 point clouds used for training, and 100 000 for validation.

3.2.2. Test dataset from the EPOS facility

The test data is collected at the EPOS rendezvous facility [20]. This facility is located at the German Aerospace Center (DLR) in Oberpfaffenhofen, Germany. It consists of two robotic arms which move with 6 degrees of freedom and each carry a charge. For this test, the first arm holds a mockup of the target satellite, as illustrated on Fig. 2. The second robotic arm holds a sensor plate on which different sensors, amongst which the Livox® lidar, are attached. Additionally, this arm is mounted on a linear rail, so that a translational movement up to a relative distance of 25 m is simulated.

On the raw point clouds captured by the lidar, all the surroundings of the facility such as the robotic arm, the floor, ceiling and curtains, are visible. These elements are cropped out in a pre-processing step in order to replicate space conditions. Also, the frame rate of the lidar is dynamically adjusted so that after this pre-processing step, all point clouds of the target contain approximately 10 000 points. Existing lidar sensors for space usually have a dynamically adaptable field-of-view [47,48], so that it is expected that all point clouds have approximately the same density. The resolution varies depending on the type of sensor considered: it is adaptable for Jena Optronik's scanning RVS® 3000 lidar [47], and it is 128 × 128 pixels for ASC's GSFL-16KS Flash lidar® [48], so that 10 000 points is an achievable number of points.

Four datasets were collected at the EPOS facility at different distances between the target and sensor: 5 m, 10 m, 15 m and 20 m. For each of these distances, the target was oriented so that it takes as many attitudes as possible, and multiple point clouds were recorded. Yet, because of the limitations of the robotic facility, not all attitudes are simulated. In particular, the test dataset contains no views of the target satellite "from behind", *i.e.*, where the sensor would be placed on the $-X$ direction in Fig. 3(a). When dividing the attitude space into bins of 20 deg resolution, the EPOS datasets contain poses covering 18% of all attitude bins, while the training data covers 100% of the bins. In total, aggregating the four EPOS datasets, the test dataset contains 8648 lidar scans, with ground truth poses. The facility is calibrated with an accuracy of 1.56 mm in position and 0.2 deg in orientation [20]. The ground truth pose associated with a point cloud is considered to be the pose at the end of the scan time. In case of fast tumbling target leading to strong motion blur, strategies to correct the distorted scan should be investigated [18], but are not in the scope of this study.

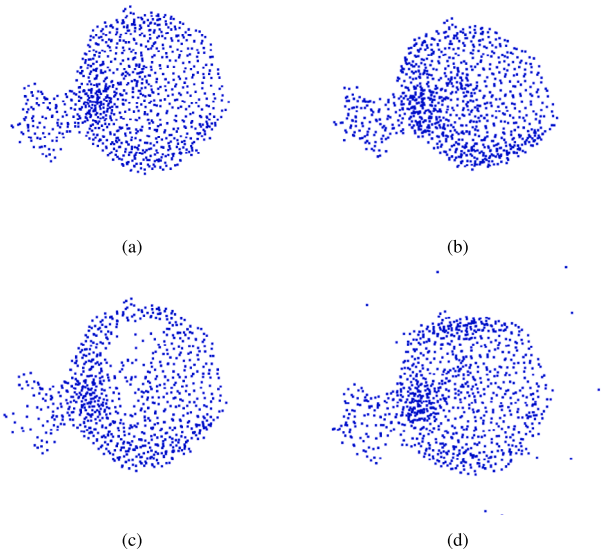


Fig. 5. Different types of data augmentation: **(a)** Reference point cloud; **(b)** Jitter and deformation; **(c)** Region dropout; **(b)** Outliers.

3.3. Training

As mentioned in the previous sections, the neural network is trained only with the synthetic point clouds of the training dataset. While several parameters of the simulator are randomized, adding further randomization and variation to the input point clouds during training adds robustness to the network: this is the process of data augmentation. In this work, we use different types of data augmentation layers, shown on Fig. 5. Data augmentation is applied to all point clouds during training.

The first type of modification of the point cloud performed during training is similar to the training procedure of [42]. It consists in the addition of jitter and deformation, as illustrated on Fig. 5(b). Jitter is the addition of noise on the position of the individual points, following a normal distribution with a standard deviation of 1 cm. Deformation involves expanding or shrinking the point cloud with a different dilatation factor along each axis, set here to ±15%. This deformation enables to account for the potential inaccuracies of the target's 3D model, by learning to recognize variable geometries.

To account for additional reflection effects and avoid that the neural network relies only on specific regions of the point cloud, region dropout is also performed, as shown on Fig. 5(c): a region around a point is selected and all points from this region are removed. This region dropout is only performed with a 50% probability, and applied with a variable radius between 20 cm and 80 cm. To ensure that the point cloud still contains 1024 points after this dropout, other points are randomly duplicated. Finally, some points are randomly removed while outliers are added within the bounding box of the target, as illustrated on Fig. 5(d). This is also only performed with a 50% probability, and at most 50 outliers are added.

In addition to data augmentation, a dropout layer is added after the fully connected layer of size 512, before the two output heads. This layer has a dropout ratio of 0.3, which is applied only during training. Using dropout layers in the last FC layers before classification is common to regularize the weights of the network and avoid overfitting [40,42].

As in [16], the network learns to minimize the sum of two losses: for the attitude classification, the cross-entropy loss between the real labels $(\overline{l_1}, \dots, \overline{l_m})^T$ and the predicted labels $(l_1, \dots, l_m)^T$ is computed. For the position, the mean squared error between the real position shift $\overline{\Delta \boldsymbol{p}}$ and the prediction $\Delta \boldsymbol{p}$ is used. In total, the loss is

$$\text{loss} = -\sum_{i=1}^{m} \left(\overline{l_i} \log l_i \right) + \lambda \frac{\|\overline{\Delta \boldsymbol{p}} - \Delta \boldsymbol{p}\|^2}{3}, \tag{11}$$

where λ is a weighting coefficient between both losses which was tuned to $\lambda = 10$ in our experiments. This weight also has a physical interpretation: with $\lambda = 10$, and for the attitude sampling resolution $\Delta\alpha = 20$ deg, numerical evaluations show that an error of 1 deg between the true and the estimated attitude leads approximately to the same increase in the cost function (11) as a 3 cm error on the position estimation.

The network is trained on 400 000 synthetic point clouds with a batch size of 32, using the Adam optimizer [60] with cosine decay and an initial learning rate of 10^{-3}. The number of epochs is set to 35. No additional termination criteria is used, and the training stops after all epochs are complete.

3.4. Pose refinement

The neural network does not require an a priori estimate of the relative pose to provide a pose estimation. Therefore, the proposed method is used for pose initialization. Once the initial pose is found, it is possible to refine it with a local registration method such as ICP or NDT. In [18], we introduced a smoothed version of the NDT algorithm tailored specifically for efficiency in the prospect of onboard implementation. We showed that this method is precise and fast when compared to ICP. This method will be used to refine the initial pose estimate delivered by the neural network in a post-processing step.

4. Results

4.1. Computing hardware for runtime evaluation

The objective is to select a neural network model and optimize it for runtime, in order to have a real-time capable pose estimation method. The processing time of an algorithm strongly depends on the type of hardware it is evaluated on. If a GPU is used, runtime is usually not an issue. On a CPU however, the processing time strongly varies from one platform to another.

Onboard computers developed for space have to be resilient to difficult thermal, radiation and power conditions. This resiliency usually comes at the cost of processing power, so that space onboard computers typically have much lower processing power than processors used for ground applications. For complex processing or AI applications in space, Commercial Off-The-Shelf (COTS) System-on-Chip (SoC) are becoming increasingly popular [61,62]. In particular, DLR is developing an onboard computer based on an AMD Zynq® 7000 SoC [62]. Therefore, the Zynq 7000 is used as a reference platform for runtime evaluation after training. It consists of a CPU unit paired with an FPGA. However, for practical reasons (ease of implementation, faster development and flexibility), only the CPU is used for evaluating the pose estimation methods. The neural networks are written in Python, and trained on a GPU. Only after training, they are evaluated in C++ using the TensorFlow Lite library. The pre- and post-processing of the network's results is also implemented in C++.

To enable computationally intensive AI applications, the field of onboard computing is also going in the direction of testing and validating GPUs for space applications [63,64]. In this work, we assume that such a platform is not available, and take the Zynq 7000 CPU as a reference onboard computer for our evaluations.

For all evaluations, the ground truth relative attitude and position $(\overline{R}, \overline{p})$ are compared to the values (R, p) estimated by the pose initialization. Since the method aims at finding one of the symmetric equivalents of the real attitude, the angular and position error are defined as follows:

- angular error: $d_{sym}(\overline{R}, R)$;
- position error: $\|\overline{p} - p\|$.

Table 2
Evaluation of different pre-processing methods when paired with PointNet. The runtime, computed on the Zynq 7000 CPU, is shown as mean ± standard deviation. The success rate (error below 5 deg and 15 cm) is evaluated on the synthetic validation dataset.

Sampling method	Validation success [%]	Pre-proc. time [ms]
random	89.07	**15 ± 1**
octree + random	90.86	35 ± 1
FPS	**93.04**	1785 ± 111
QuickFPS [49]	**93.04**	118 ± 6

4.2. Pre-processing

The first evaluation concerns the sampling method used in the pre-processing step. The synthetic and real point clouds contain in average about 10 000 points each. However, the neural network expects a fixed number of points as input, which is set to 1024 in this work. Therefore, several strategies are used to downsample the point clouds:

- Random sampling: Points are randomly removed or duplicated to obtain 1024 points;
- Octree downsampling + random sampling: The point cloud is first downsampled using an octree with a certain voxel size (here 4 cm). Because the resulting point cloud does not have a deterministic number of points, random sampling is additionally applied afterwards;
- FPS: A standard implementation of FPS;
- QuickFPS: A k-d tree version of FPS as in [49]. The maximum number of points per cell for the k-d tree subdivision of QuickFPS is set to 200.

Table 2 presents a comparison of these different methods, when used for pre-processing the point clouds at training and testing time. For this evaluation, PointNet is used as a backbone model. The trained model with the pre-processing method is evaluated on the synthetic validation dataset. To quantify the performance of a model, a success criteria is defined, which corresponds to a position and attitude error of the estimation below 15 cm and 5 deg. When the initial error is below this threshold, it is expected that the estimate will converge towards a precise pose estimate after the refinement step.

While random sampling is the fastest pre-processing method, FPS leads to a better model accuracy. The octree downsampling paired with random downsampling also leads to a significantly lower accuracy compared to FPS. Therefore, the efficient, k-d tree based implementation of the FPS algorithm is chosen as a pre-processing method for all subsequent evaluations.

4.3. Base model selection

To select the neural network backbone, several state-of-the art models, presented in Section 2, are compared: PointNet, PointNet++, DGCNN and Point Transformer. These networks are re-implemented in Python using the TensorFlow library. The different networks are indifferently inserted in the pose estimation pipeline, with the only difference that Point Transformer has an output feature layer of size 512, while the other networks have a feature layer of size 1024. The results when training and testing these networks on the synthetic datasets are presented in Table 3.

From all evaluated models, PointNet++ presents the best accuracy, with a success rate on the validation dataset of 99.23%. The evolution of the training and validation losses for this model during the training process are presented in Fig. 6. Due to the use of data augmentation and dropout regularization, the validation loss is lower than the training loss.

However, the precision of PointNet++ also comes at the cost of an important inference runtime when evaluated on the onboard-representative hardware. Selecting this model as a baseline, we will analyze in the following Section 4.4 how the model's parameters are adapted to reduce the runtime.

Table 3
Success rate (error below 5 deg and 15 cm) and runtime (on the Zynq 7000 CPU) of different 3D network models when evaluated on the synthetic validation dataset. The runtime is shown as mean ± standard deviation, and does not include pre-processing.

Model name	Validation success [%]	Runtime [ms]
PointNet [40]	93.04	**264 ± 0**
PointNet++ [41]	**99.23**	1719 ± 1
DGCNN [42]	98.82	2455 ± 0
Point Transformer [43]	98.31	676 ± 0

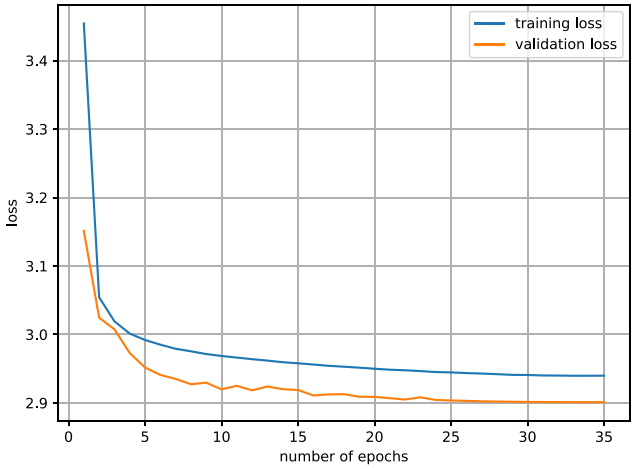


Fig. 6. Training and validation loss when training PointNet++ for 35 epochs with data augmentation.

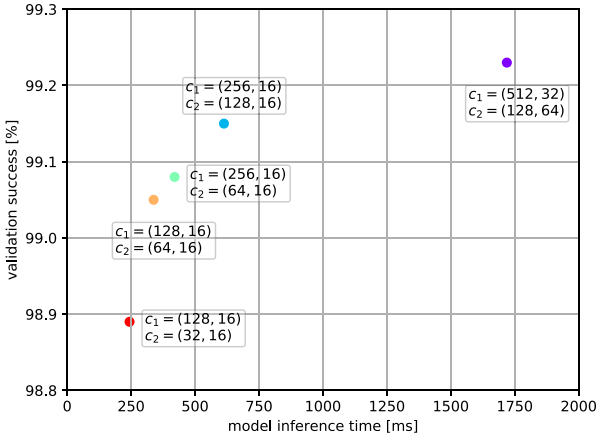


Fig. 7. Average runtime on the Zynq 7000 CPU vs accuracy on the synthetic validation dataset of different PointNet++ models. The labels $c_1 = (N_1, k_1)$ and $c_2 = (N_2, k_2)$ indicate the number of clusters and the cluster size of the first and second clustering layers, respectively.

Table 4
Ablation study of the optimized network: comparison of the optimized PointNet++ model with the same model where one component has been modified. The success rate (error below 5 deg and 15 cm) is evaluated on the synthetic validation dataset, and the runtime shown as mean ± standard deviation on the Zynq 7000 CPU.

Model name	Validation success [%]	Runtime [ms]
reference (optimized)	99.08	420 ± 0
– shrinkage $\alpha_{MLP} = 7/8$	99.01	392 ± 0
– one FC layer less	98.30	420 ± 0
– input points $N = 768$	98.91	387 ± 0

4.4. Optimized PointNet++ model

The original PointNet++ when evaluated on the Zynq CPU, takes about 1.7 s to run an inference, which is considered too long for real-time pose estimation in this case. While the exact timing requirements vary depending on the mission, it is typically desired to provide a pose estimation solution with at least 1 Hz. This is also in line with the expected frame rate of a space qualified scanning lidar. Accounting for pre-processing time and margin, we seek a runtime of the model below 0.5 s.

Given a fixed number of input points $N = 1024$, optimizing the model for runtime involves reducing the number of operations performed in the model. In Table 1, we provided an overview of the layers of the PointNet++ model. A first possibility to reduce the model size is to tune the parameters associated with the two clustering operations: the first clustering produces N_1 clusters of k_1 points, and the second one N_2 clusters of k_2 points.

The original PointNet++ model has sizes $N_1 = 512$, $k_1 = 32$ and $N_2 = 128$, $k_2 = 64$. We find out that reducing these parameters leads to an important reduction of the runtime, while the performance loss is less drastic. We explore several variations of these parameters. Fig. 7 presents the results of this parameter variation, where only the best models, which had a high success rate compared to their runtime, are presented. For each model identified by its two clusters $c_1 = (N_1, k_1)$ and $c_2 = (N_2, k_2)$, the model is positioned on the graph depending on its success rate on the validation dataset, as a function of its runtime on the Zynq CPU.

With a decreasing number of clusters and cluster size also comes a decrease in performance. Therefore, the choice of the suited model on Fig. 7 strongly depends on the mission requirements. For the case of the study, the selected model has a number of first clustering layer with dimensions $(N_1, k_1) = (256, 16)$, and the second one with size $(N_2, k_2) = (64, 16)$. This model (in green on Fig. 7) shows to be a suited compromise between runtime and accuracy. The average runtime of this model on the Zynq CPU is of 420 ms, so less than 0.5 s, with a success rate (initial estimate below 5 deg and 15 cm) on the synthetic validation dataset of 99.08%. In the following, this model will be referred to as the optimized reference model.

A further direction for optimization is to modify the width of the MLP layers. In the reference configuration of Table 1, PointNet++ contains three shared MLP layers of dimensions $\{64, 64, 128\}$, $\{128, 128, 256\}$, and $\{256, 512, 1024\}$. Reducing the sizes of these MLP layers is also considered. To do this, a shrinking factor $\alpha_{MLP} \leq 1$ is introduced, by which all the MLP dimensions are multiplied. When $\alpha_{MLP} = 1$, the configuration corresponds to the default configuration. In Table 4, we present the success rate on the synthetic validation dataset, and runtime, of the optimized model with reduced cluster sizes, compared with the same model where a shrinkage factor $\alpha_{MLP} = 7/8$ has been applied.

While reducing the width of the network enables to reduce the runtime accordingly, the performance also decreases by 0.07% when using a shrinking factor $\alpha_{MLP} = 7/8$ compared to the reference model. Reducing the width of the MLP layers also reduces the number of parameters of the network. It is a modification that is to be considered if the runtime is to be further optimized, but is not retained here as the performance also decreases. Another possibility to strongly reduce the number of parameters of the network, is to remove one of the top layers. In the third line of the ablation study presented in Table 4, the Fully Connected (FC) layer of size 512, which is between the feature vector and the two classification heads, has been removed. However, the performance drop compared to the reference model is significant, by around 0.8%, highlighting the importance of this intermediate layer between the feature vector and the output heads. In addition, removing the fully connected layer does not visibly reduce the runtime, which is dominated by the evaluations in the previous layers of the network.

Finally, it is also possible to reduce the number of input points to the model, N. This could also be a mission requirement in case the point clouds collected by the sensor are very sparse. On the last line

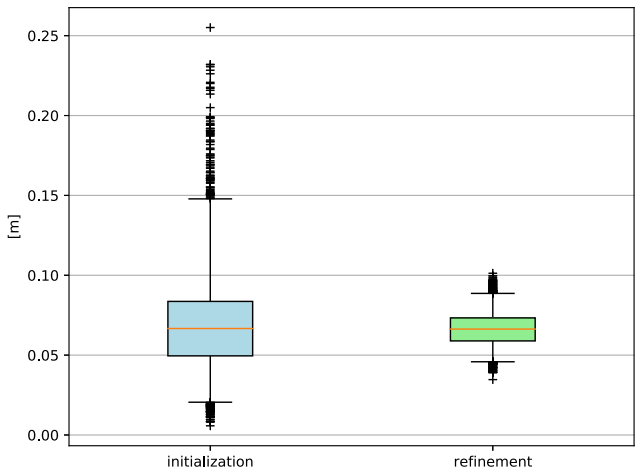


Fig. 8. Distribution of position errors before and after NDT refinement, when evaluating the method on the EPOS dataset. Whiskers extend to the 1st and 99th percentiles.

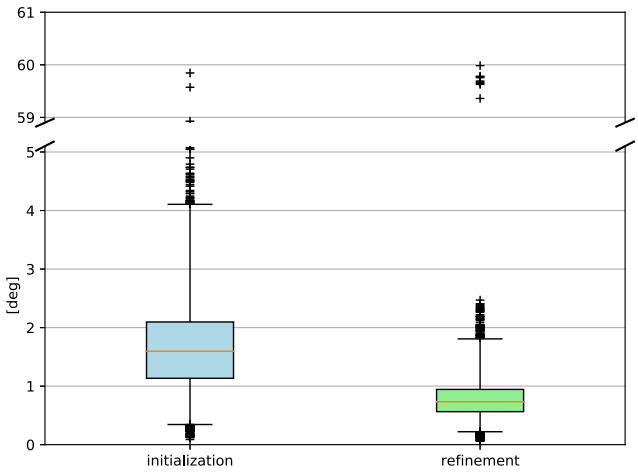


Fig. 9. Distribution of angular errors before and after NDT refinement, when evaluating the method on the EPOS dataset. Whiskers extend to the 1st and 99th percentiles.

of Table 4, the reference model has been modified to take as input a point cloud of size $N = 768$ (25% less points than the reference model, where $N = 1024$). As it is observed, the performance drops to 98.91%, mainly because the network has less information as input. The decrease in runtime is mainly due to the fact that the first clustering step is performed with less points, and is less computationally expensive. The following layers have the same size as the model with 1024 points.

4.5. Results on real dataset

After tuning and training the model on synthetic data, evaluation is performed on the EPOS test dataset containing real lidar scans. For this evaluation, the NDT pose refinement presented in Section 3.4 is applied after an initial estimate has been provided by the neural network. The NDT algorithm is set with following parameters, which are further detailed in [18]: the voxel grid size for computing the distributions, as well as the maximum point-to-cell distance, are set to 7.5 cm. The iterative algorithm stops either once 30 iterations have been performed, or if the increment to perform is below 0.05 deg and 1 mm. Finally, before being registered, the source point clouds are downsampled with a voxel filter of 2 cm.

We evaluate the pose estimation method, using the optimized PointNet++ model presented in the previous Section 4.4 as a backbone. This model has dimensions $(N_1, k_1) = (256, 16)$ and $(N_2, k_2) = (64, 16)$. The distribution of errors of the method on the EPOS dataset before and after the NDT refinement step is presented on Figs. 8 and 9. The attitude error (Fig. 9) is computed using the symmetrical distance function (7). The boxes extend to the first and third quartile, and the orange line represents the median. The whiskers extend to the first and 99th percentiles, while outliers are marked with crosses.

From Figs. 8 and 9, it is seen that the refinement step helps to correct initial estimates, and to reduce the error spread. After the refinement step, 99% of the estimate have a position error below 9 cm, and an attitude error below 2 deg. Interestingly, in most cases where the attitude prediction is wrong, it is wrong by an error of 60 deg. This indicates that due to the hexagonal shape of the target, the neural network might confuse two possible configurations of the hexagon if the three antennas could not be clearly identified in the point cloud, see Fig. 3.

We compare our method with the CNN based pose estimation that we introduced in previous work [16]. This second method is trained and tested on the same datasets, and also combined with NDT for pose refinement. Additionally, we evaluate the results of our method when using the original (baseline) PointNet++ model as a backbone. The result of this comparison is presented in Table 5. It includes the results of the runtime evaluation of the three methods on the Zynq 7000 CPU.

From Table 5, it is seen that the position errors of the three methods are very similar. However, the two PointNet++ based models outperform the CNN based method when it comes to robustness of the attitude estimation. Overall, the method presented in this work, using the optimized network combined with NDT refinement, predicts a pose with an error below 15 cm and 5 deg for 99.79% of the point clouds of the EPOS dataset. This number even increases to 99.83% when using the original PointNet++ model, not optimized for runtime. Yet, the baseline model is more than four times slower than the optimized model.

5. Discussion

5.1. Relevance of the different processing steps

The proposed pose estimation pipeline consists of several steps, all of which have their importance. In particular, the importance of the FPS as a sampling method during pre-processing was presented in Section 4.2. The neural network was also optimized to remove parameters and dimensions which do not significantly affect the performance, with results presented in Section 4.4. The relevance of data augmentation during training is evaluated on the real dataset. In the first two lines of Table 6 we compare the success rate of the reference model optimized for runtime, with the same model trained without data augmentation.

Without data augmentation, the performance decrease is of around 0.6%, demonstrating that the proposed data augmentation method enables to increase the robustness of the network when evaluated on real lidar data. While this shows the relevance of using data augmentation, the gap between the model trained with and without data augmentation is much smaller than what is observed when using a CNN instead of a 3D network for this task [16].

In addition, we evaluate the importance of using a trimmed centroid (2) instead of a standard centroid (*i.e.*, mean) during pre-processing. The result of the model where a standard centroid is used both during training and evaluation is presented on the third line of Table 6. Using a standard centroid leads to a strong performance loss, as the overall accuracy drops by over 16% compared to the reference model. This is explained by the fact that the real point clouds contain a high proportion of outliers and ghost reflections, which are located *e.g.*, at twice the distance of the sensor to the satellite. Compared to the trimmed centroid, the regular centroid is less robust, and affected by these outliers. By not discarding them, the selected ROI is less guaranteed to contain all relevant points.

Table 5
Precision and runtime comparison of three methods evaluated on the EPOS lidar dataset, after NDT refinement. The baseline corresponds to the original PointNet++ model, while the optimized version uses the model optimized for runtime. The runtime is evaluated on the Zynq 7000 CPU.

Method	Position error [cm]		Attitude error [deg]		Success rate [%]	Runtime [ms] (avg. ± stddev.)			
	Median	99th %ile	Median	99th %ile	(<5 deg and 15 cm)	Pre-proc.	Initialization	Refinement	Total
CNN based [16]	**6.63**	8.87	0.74	59.7	97.94	24 ± 1	737 ± 0	347 ± 101	1108 ± 101
ours (baseline)	6.64	8.87	0.74	1.85	**99.83**	118 ± 6	1719 ± 1	345 ± 114	2182 ± 114
ours (optimized)	**6.63**	8.87	**0.73**	**1.82**	99.79	118 ± 6	420 ± 0	341 ± 109	**879 ± 109**

Table 6
Ablation study: success rate (error below 5 deg and 15 cm) when evaluated on the EPOS test datasets of the reference model compared to the same model where only the listed component was removed.

Model name	Success rate [%]
reference (optimized)	99.79
– no data augm.	99.20
– standard centroid	83.18
– no NDT refinement	98.50

The gain in precision added by the NDT refinement step was already presented in Section 4.5. The last line of Table 6 summarizes the importance of the NDT refinement. Without this step, the performance decreases by over 1%.

Point clouds are visualized either as a loose collection of 3D coordinates, or as a 2D depth image, especially when a matrix based sensor such as a time-of-flight lidar is used. In the same way, during processing, it is possible to either directly process the 3D point cloud, as in this work, or process the depth image using a CNN [14,16]. While both methods are effective, we have observed in this work the superiority of using point-based neural networks for this task, as highlighted by the comparison presented in Table 5.

5.2. Applicability to onboard implementation

Neural networks are inherently adapted for real-time requirements, as they perform a deterministic number of operations to evaluate an input. Therefore, the runtime of a neural network is nearly always the same. Yet depending on the selected model and the computing platform, the runtime is high. In this work, the runtime was optimized for a specific computing platform, the AMD Zynq® 7000, which is a potential candidate for an onboard computer. However, for other types of onboard computers, it might not be needed to optimize the model, or on the opposite to further optimize it.

In this work, we have proposed a way to significantly reduce the runtime of a standard architecture, PointNet++, without a consequential loss in performance. The inference time of the optimized model compared to the baseline is reduced by more than a factor 4, see column "initialization" of Table 5. For the PointNet++ architecture, reducing the number of clusters, and cluster sizes, is an efficient way to reduce the runtime while maintaining a good performance. Depending on the requirements on the model runtime, further optimization might be performed. The final result is a trade-off between the desired runtime, and the model's accuracy.

For a rendezvous mission, testing with real lidar data prior to the mission might not be possible. If data is collected in a hardware-in-the-loop facility, the data could also be included in the training process to form a hybrid training dataset. Still, a domain gap between the data collected in the facility and flight data might be observed. The approach presented in this work consists in using only synthetic data for training. This enables to quickly generate a training dataset as large as desired, and requires only to have a 3D model of the target. By training and optimizing the model solely on synthetic data, and testing it on real point clouds, we have shown that the pose estimation method is robust and able to bridge the "domain gap", *i.e.*, that no noticeable difference is observed between the results obtained on synthetic and real data.

With 99% of the pose estimates having a position error below 9 cm and an attitude error below 1.9 deg, the method was shown to be precise, reliable and fast enough on onboard hardware for real-time pose estimation. While the training is done with Python, the model is evaluated on the hardware in C++, using the TensorFlow Lite library. The optimized model contains around 1.7 million parameters, taking up 7.1 MB of space, which is usually not a problem for onboard storage.

The presented pose initialization method does not intend to replace pose tracking methods such as ICP or NDT, but is necessary as a complement to a pose tracking method, which requires an initial estimate to converge. In operational scenarios, pose initialization is run during the first target acquisition when entering close range. Afterwards, a tracking mode is entered, where the previous navigation estimate is used as an initial estimate for the current scan [3]. Given its precision, the method might also be used as a consistency check to be run periodically.

6. Conclusion

In this work, we introduced a method for lidar based pose estimation of a non-cooperative spacecraft during space rendezvous. In a single stage, a neural network computes an estimate of both the relative position and the attitude, given an input point cloud. The attitude classification logic is adapted to account for estimating the pose of a symmetrical spacecraft, where several attitudes might represent the same configuration. A refinement method is further applied to increase the precision of the initial estimate. The network is trained and optimized for runtime on synthetic data. Evaluation on onboard representative computing hardware and on a real lidar dataset shows that the method is flight-ready: it is precise, efficient and reliable, with over 99.7% of the pose estimates on real data having an error below 5 deg and 15 cm.

While being tested on a real dataset, the lidar sensor used in this work originates from the automotive domain: further work might include tests with a space-representative lidar sensor, and possibly other target satellites with a different shape. Finally, a strategy or measure to detect potential erroneous initial estimates could be investigated.

CRediT authorship contribution statement

Léo Renaut: Writing – original draft, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Heike Frei:** Writing – review & editing, Validation, Supervision, Methodology, Investigation. **Andreas Nüchter:** Writing – review & editing, Validation, Supervision, Methodology, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Pyrak, J. Anderson, Performance of Northrop Grumman's Mission extension vehicle (MEV) RPO imagers at GEO, in: Autonomous Systems: Sensors, Processing and Security for Ground, Air, Sea and Space Vehicles and Infrastructure 2022, vol. 12115, SPIE, 2022, pp. 64–82, http://dx.doi.org/10.1117/12.2631524.

[2] Astroscale, Astroscale's ADRAS-J continues to make history: Successfully demonstrates fly-around observations of space debris, 2024, [Press release]: https://astroscale.com/astroscales-adras-j-continues-to-make-history-successfully-demonstrates-fly-around-observations-of-space-debris/.

[3] R. Opromolla, G. Fasano, G. Rufino, M. Grassi, A review of cooperative and uncooperative spacecraft pose determination techniques for close-proximity operations, Prog. Aerosp. Sci. 93 (2017) 53–72, http://dx.doi.org/10.1016/j.paerosci.2017.07.001.

[4] L.P. Cassinis, R. Fonod, E. Gill, Review of the robustness and applicability of monocular pose estimation systems for relative navigation with an uncooperative spacecraft, Prog. Aerosp. Sci. 110 (2019) 100548, http://dx.doi.org/10.1016/j.paerosci.2019.05.008.

[5] L. Pauly, W. Rharbaoui, C. Shneider, A. Rathinam, V. Gaudillière, D. Aouada, A survey on deep learning-based monocular spacecraft pose estimation: Current state, limitations and prospects, Acta Astronaut. (2023) http://dx.doi.org/10.1016/j.actaastro.2023.08.001.

[6] S. Ruel, T. Luu, A. Berube, Space shuttle testing of the TriDAR 3D rendezvous and docking sensor, J. Field Robot. 29 (4) (2012) 535–553, http://dx.doi.org/10.1002/rob.20420.

[7] F. Yin, W. Chou, Y. Wu, G. Yang, S. Xu, Sparse unorganized point cloud based relative pose estimation for uncooperative space target, Sensors 18 (4) (2018) 1009, http://dx.doi.org/10.3390/s18041009.

[8] K. Klionovska, H. Benninghoff (now Frei), Initial pose estimation using PMD Sensor during the rendezvous phase in on-orbit servicing missions, in: 27th AIAA/AAS Space Flight Mechanics Meeting, 2017.

[9] J.O. Woods, J.A. Christian, LIDAR-based relative navigation with respect to non-cooperative objects, Acta Astronaut. 126 (2016) 298–311, http://dx.doi.org/10.1016/j.actaastro.2016.05.007.

[10] R. Opromolla, G. Fasano, G. Rufino, M. Grassi, A model-based 3D template matching technique for pose acquisition of an uncooperative space object, Sensors 15 (3) (2015) 6360–6382, http://dx.doi.org/10.3390/s150306360.

[11] R. Opromolla, G. Fasano, G. Rufino, M. Grassi, Pose estimation for spacecraft relative navigation using model-based algorithms, IEEE Trans. Aerosp. Electron. Syst. 53 (1) (2017) 431–447, http://dx.doi.org/10.1109/TAES.2017.2650785.

[12] W. Guo, W. Hu, C. Liu, T. Lu, Pose initialization of uncooperative spacecraft by template matching with sparse point cloud, J. Guid. Control Dyn. 44 (9) (2021) 1707–1720, http://dx.doi.org/10.2514/1.G005042.

[13] C. Schmitt, J. Both, F. Kolb, RVS3000-3D: LIDAR meets neural networks, in: International Symposium of Artificial Intelligence, Robotics and Automation in Space, (Jun. 4, 2018), 2018, pp. 1–7.

[14] E.A. Pensado, L.M.G. de Santos, H.G. Jorge, M. Sanjurjo-Rivo, Deep learning-based target pose estimation using LiDAR measurements in active debris removal operations, IEEE Trans. Aerosp. Electron. Syst. 59 (5) (2023) 5658–5670, http://dx.doi.org/10.1109/TAES.2023.3262505.

[15] S. Zhang, W. Hu, W. Guo, C. Liu, Neural-network-based pose estimation during noncooperative spacecraft rendezvous using point cloud, J. Aerosp. Inf. Syst. 20 (8) (2023) 462–472, http://dx.doi.org/10.2514/1.I011179.

[16] L. Renaut, H. Frei, A. Nüchter, CNN-based pose estimation of a non-cooperative spacecraft with symmetries from lidar point clouds, IEEE Trans. Aerosp. Electron. Syst. (2024) http://dx.doi.org/10.1109/TAES.2024.3517574.

[17] Q. Wang, T. Lei, X. Liu, G. Cai, Y. Yang, L. Jiang, Z. Yu, Pose estimation of non-cooperative target coated with MLI, IEEE Access 7 (2019) 153958–153968, http://dx.doi.org/10.1109/ACCESS.2019.2946346.

[18] L. Renaut, H. Frei, A. Nüchter, Lidar pose tracking of a tumbling spacecraft using the smoothed normal distribution transform, Remote. Sens. 15 (9) (2023) 2286, http://dx.doi.org/10.3390/rs15092286.

[19] G. Pitteri, M. Ramamonjisoa, S. Ilic, V. Lepetit, On object symmetries and 6D pose estimation from images, in: 2019 International Conference on 3D Vision, 3DV, IEEE, 2019, pp. 614–622, http://dx.doi.org/10.1109/3DV.2019.00073.

[20] H. Benninghoff (now Frei), F. Rems, E.-A. Risse, C. Mietner, European proximity operations simulator 2.0 (EPOS) - A robotic-based rendezvous and docking simulator, J. Large- Scale Res. Facil. JLSRF (2017) http://dx.doi.org/10.17815/jlsrf-3-155.

[21] P. Besl, N.D. McKay, A method for registration of 3-D shapes, IEEE Trans. Pattern Anal. Mach. Intell. 14 (02) (1992) 239–256, http://dx.doi.org/10.1109/34.121791.

[22] J. Yang, H. Li, D. Campbell, Y. Jia, Go-ICP: A globally optimal solution to 3D ICP point-set registration, IEEE Trans. Pattern Anal. Mach. Intell. 38 (11) (2015) 2241–2254, http://dx.doi.org/10.1109/TPAMI.2015.2513405.

[23] L. Liu, G. Zhao, Y. Bo, Point cloud based relative pose estimation of a satellite in close range, Sensors 16 (6) (2016) 824, http://dx.doi.org/10.3390/s16060824.

[24] R.B. Rusu, N. Blodow, Z.C. Marton, M. Beetz, Aligning point cloud views using persistent feature histograms, in: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2008, pp. 3384–3391, http://dx.doi.org/10.1109/IROS.2008.4650967.

[25] P. Jasiobedzki, M. Greenspan, G. Roth, H. Ng, N. Witcomb, Video-based system for satellite proximity operations, in: 7th ESA Workshop on Advanced Space Technologies for Robotics and Automation, ASTRA 2002, ESTEC, Noordwijk, the Netherlands, 2002.

[26] P. Biber, W. Straßer, The normal distributions transform: A new approach to laser scan matching, in: Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), vol. 3, IEEE, 2003, pp. 2743–2748, http://dx.doi.org/10.1109/IROS.2003.1249285.

[27] B. Chen, J. Cao, A. Parra, T.-J. Chin, Satellite pose estimation with deep landmark regression and nonlinear pose refinement, in: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, 2019, http://dx.doi.org/10.48550/arXiv.1908.11542.

[28] K. Cosmas, A. Kenichi, Utilization of FPGA for onboard inference of landmark localization in CNN-based spacecraft pose estimation, Aerospace 7 (11) (2020) 159, http://dx.doi.org/10.3390/aerospace7110159.

[29] Y. Hu, S. Speierer, W. Jakob, P. Fua, M. Salzmann, Wide-depth-range 6D object pose estimation in space, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 15870–15879, http://dx.doi.org/10.48550/arXiv.2104.00337.

[30] K. Black, S. Shankar, D. Fonseka, J. Deutsch, A. Dhir, M.R. Akella, Real-time, flight-ready, non-cooperative spacecraft pose estimation using monocular imagery, in: 31st AIAA/AAS Space Flight Mechanics Meeting, 2021, http://dx.doi.org/10.48550/arXiv.2101.09553.

[31] S. Sharma, S. D'Amico, Neural network-based pose estimation for noncooperative spacecraft rendezvous, IEEE Trans. Aerosp. Electron. Syst. 56 (6) (2020) 4638–4658, http://dx.doi.org/10.1109/TAES.2020.2999148.

[32] P.F. Proença, Y. Gao, Deep learning for spacecraft pose estimation from photorealistic rendering, in: 2020 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2020, pp. 6007–6013, http://dx.doi.org/10.1109/ICRA40945.2020.9197244.

[33] T.H. Park, S. D'Amico, Robust multi-task learning and online refinement for spacecraft pose estimation across domain gap, Adv. Space Res. (2023) http://dx.doi.org/10.48550/arXiv.2203.04275.

[34] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al., Searching for MobileNetV3, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1314–1324, http://dx.doi.org/10.1109/ICCV.2019.00140.

[35] O. Kechagias-Stamatis, N. Aouf, V. Dubanchet, M.A. Richardson, DeepLO: Multi-projection deep LIDAR odometry for space orbital robotics rendezvous relative navigation, Acta Astronaut. 177 (2020) 270–285, http://dx.doi.org/10.1016/j.actaastro.2020.07.034.

[36] Z. Chekakta, A. Zenati, N. Aouf, O. Dubois-Matra, Robust deep learning LiDAR-based pose estimation for autonomous space landers, Acta Astronaut. 201 (2022) 59–74, http://dx.doi.org/10.1016/j.actaastro.2022.08.049.

[37] D. Maturana, S. Scherer, VoxNet: A 3D convolutional neural network for real-time object recognition, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2015, pp. 922–928, http://dx.doi.org/10.1109/IROS.2015.7353481.

[38] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, B. Chen, PointCNN: Convolution on X-Transformed points, Adv. Neural Inf. Process. Syst. 31 (2018) http://dx.doi.org/10.48550/arXiv.1801.07791.

[39] H. Thomas, C.R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, L.J. Guibas, KPConv: Flexible and deformable convolution for point clouds, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 6411–6420, http://dx.doi.org/10.1109/ICCV.2019.00651.

[40] C.R. Qi, H. Su, K. Mo, L.J. Guibas, PointNet: Deep learning on point sets for 3D classification and segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 652–660, http://dx.doi.org/10.1109/CVPR.2017.16.

[41] C.R. Qi, L. Yi, H. Su, L.J. Guibas, PointNet++: Deep hierarchical feature learning on point sets in a metric space, Adv. Neural Inf. Process. Syst. 30 (2017) http://dx.doi.org/10.48550/arXiv.1706.02413.

[42] Y. Wang, Y. Sun, Z. Liu, S.E. Sarma, M.M. Bronstein, J.M. Solomon, Dynamic graph CNN for Learning on Point Clouds, ACM Trans. Graph. (Tog) 38 (5) (2019) 1–12, http://dx.doi.org/10.1145/3326362.

[43] H. Zhao, L. Jiang, J. Jia, P.H. Torr, V. Koltun, Point transformer, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 16259–16268, http://dx.doi.org/10.1109/ICCV48922.2021.01595.

[44] S. Zhang, W. Hu, W. Guo, 6-DoF pose estimation of uncooperative space object using deep learning with point cloud, in: 2022 IEEE Aerospace Conference, AERO, IEEE, 2022, pp. 1–7, http://dx.doi.org/10.1109/AERO53065.2022.9843444.

[45] A. Myronenko, X. Song, Point set registration: Coherent point drift, IEEE Trans. Pattern Anal. Mach. Intell. 32 (12) (2010) 2262–2275, http://dx.doi.org/10.1109/TPAMI.2010.46.

[46] J. Sola, J. Sevilla, Importance of input data normalization for the application of neural networks to complex industrial problems, IEEE Trans. Nucl. Sci. 44 (3) (1997) 1464–1468, http://dx.doi.org/10.1109/23.589532.

[47] C. Schmitt, M. Windmüller, M. Schwarz, M. Möller, RVS® 3000-3D LiDAR–next stop: Gateway, in: Proceedings of the 44th Annual American Astronautical Society Guidance, Navigation, and Control Conference, Springer, 2022, pp. 489–496, http://dx.doi.org/10.1007/978-3-031-51928-4_29.

[48] B.A. Sornsin, B.W. Short, T.N. Bourbeau, M.J. Dahlin, Global shutter solid state flash LIDAR for spacecraft navigation and docking applications, in: Laser Radar Technology and Applications XXIV, vol. 11005, SPIE, 2019, pp. 229–240, http://dx.doi.org/10.1117/12.2519178.

[49] M. Han, L. Wang, L. Xiao, H. Zhang, C. Zhang, X. Xu, J. Zhu, QuickFPS: Architecture and algorithm co-design for farthest point sampling in large-scale point clouds, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 42 (11) (2023) 4011–4024, http://dx.doi.org/10.1109/TCAD.2023.3274922.

[50] C.E. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall, Activation functions: comparison of trends in practice and research for deep learning, in: 2nd International Conference on Computational Sciences and Technology, 2021, http://dx.doi.org/10.48550/arXiv.1811.03378.

[51] Y. Zhou, C. Barnes, J. Lu, J. Yang, H. Li, On the continuity of rotation representations in neural networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 5745–5753, http://dx.doi.org/10.1109/CVPR.2019.00589.

[52] S.T. Wong, M.S. Roos, A strategy for sampling on a sphere applied to 3D selective RF pulse design, Magn. Reson. Med. 32 (6) (1994) 778–784, http://dx.doi.org/10.1002/mrm.1910320614.

[53] R.G. Brazeal, B.E. Wilkinson, H.H. Hochmair, A rigorous observation model for the Risley prism-based Livox Mid-40 lidar sensor, Sensors 21 (14) (2021) 4722, http://dx.doi.org/10.3390/s21144722.

[54] L. Winiwarter, A.M.E. Pena, H. Weiser, K. Anders, J.M. Sánchez, M. Searle, B. Höfle, Virtual laser scanning with HELIOS++: A novel take on ray tracing-based simulation of topographic 3D laser scanning, Remote Sens. Environ. 269 (2022) 112772, http://dx.doi.org/10.1016/j.rse.2021.112772.

[55] B.T. Phong, Illumination for computer generated pictures, in: Seminal Graphics: Pioneering Efforts that Shaped the Field, 1998, pp. 95–101, http://dx.doi.org/10.1145/360825.360839.

[56] B. Jutzi, H. Gross, Normalization of lidar intensity data based on range and surface incidence angle, Int. Arch. Photogramm. Remote. Sens. Spat. Inf. Sci. 38 (2009) 213–218, http://dx.doi.org/10.24406/publica-fhg-362664.

[57] H. Fan, H. Su, L.J. Guibas, A point set generation network for 3D object reconstruction from a single image, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 605–613, http://dx.doi.org/10.1109/CVPR.2017.264.

[58] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2017, pp. 23–30, http://dx.doi.org/10.1109/IROS.2017.8202133.

[59] Y. Nakajima, T. Sasaki, N. Okada, T. Yamamoto, Development of LiDAR measurement simulator considering target surface reflection, in: Proceedings of the 8th European Conference on Space Debris, Darmstadt, Germany, 2021, pp. 20–23.

[60] D. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations, ICLR, 2015, http://dx.doi.org/10.48550/arXiv.1412.6980.

[61] G. Furano, G. Meoni, A. Dunne, D. Moloney, V. Ferlet-Cavrois, A. Tavoularis, J. Byrne, L. Buckley, M. Psarakis, K.-O. Voss, et al., Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities, IEEE Aerosp. Electron. Syst. Mag. 35 (12) (2020) 44–56, http://dx.doi.org/10.1109/MAES.2020.3008468.

[62] D. Lüdtke, T. Firchau, C.G. Cortes, A. Lund, A.M. Nepal, M.M. Elbarrawy, Z.H. Hammadeh, J.-G. Meß, P. Kenny, F. Brömer, et al., ScOSA on the way to orbit: Reconfigurable high-performance computing for spacecraft, in: 2023 IEEE Space Computing Conference, SCC, IEEE, 2023, pp. 34–44, http://dx.doi.org/10.1109/SCC57168.2023.00015.

[63] B. Zhang, Y. Wu, B. Zhao, J. Chanussot, D. Hong, J. Yao, L. Gao, Progress and challenges in intelligent remote sensing satellite systems, IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens. 15 (2022) 1814–1822, http://dx.doi.org/10.1109/JSTARS.2022.3148139.

[64] T. Herbst, O. Balagurin, T. Greiner, T. Kaiser, H. Kayal, A. Maurer, T. Schwarz, 6U+ CubeSat SONATE2: Operation of an optical AI payload in low earth orbit, in: 75th International Astronautical Congress, IAC, 2024.