Point Cloud Simulator for Space in-Orbit Close Range Autonomous Operations.

L.M. González-deSantos [1*], H. González-Jorge [1], M. Sanjurjo-Rivo [2], H. Michinel[1].

[1] Engineering Physics Group. School of Aerospace Engineering, University of Vigo, Campus Ourense, 32004 Ourense, Spain.
[2] Department of Bioengineering and Aerospace Engineering, Universidad Carlos III de Madrid, Leganés, Spain.

Commission I, WG I/2

KEY WORDS: LIDAR, Point Cloud, Aerospace, Rendezvous operations, on-orbit services, Pose tracking.

ABSTRACT:

In recent years, many different in-orbit close-range autonomous operations have been developed for multiple purposes, such as rendezvous and docking operations or ADR operations. In both cases, the systems have to calculate the relative position between the spacecraft and the target in order to control the orbital manoeuvres and the physic interaction between both systems. One of the sensors used for the pose calculation for these operations are LiDAR sensors, developing pose calculation algorithms that process the point cloud acquired by these sensors. One of the main problems for the development and testing of these algorithms is the lack of real data acquired in orbit and the difficulty of acquiring this data. This makes it fundamental to develop a simulator to generate realistic point clouds that can be used to develop and test pose calculation algorithms. This work presents a simulator developed for this purpose, that is the generation of realistic point clouds for algorithm development for pose calculation using LiDAR sensors for space in-orbit close range autonomous operations. The simulator uses the LiDAR sensor specifications, in order to introduces measurement errors and the scanning pattern, and 3D model of the satellite or object that is scanned.

1. INTRODUCTION

Nowadays, new systems for close-range autonomous operations have been developed to perform many different operations where the spacecraft have to interact with different targets. One example of these operations are the rendezvous and docking (R&D) operations, that consists of a series of orbital manoeuvres for approaching to the target ending with the mating process. The R&D operations are used for many different applications, such as on-orbit servicing (OOS) and active debris removal (ADR). OOS (Kaiser et al., 2008) basically consists of a R&D operation for different service propose, such as refuelling or repairing tasks. The objective of these operations is to enlarge operational lifetime of satellites, which is especially interesting for geostationary satellites. ADR operations (Liou, 2011) basically consist of the removal of obsolete satellites in orbit, that suppose a collision risks for other satellite and spacecrafts.

Two different targets can be considered in these R&D operations, cooperative and uncooperative targets, depending on if they are equipped or not with a dedicated communication link for make its pose know by other satellites. The R&D operations with uncooperative targets are especially challenging, since it is necessary first to calculate its pose. Many different pose tracking systems have been developed, using different kind of sensors, some passive sensors, such as monocular cameras or stereo vision systems, or active sensors, such as LiDAR (Light Detection and Ranging) sensors. Different algorithms have been developed for pose tracking operations. Most of them are model-based algorithms (Opromolla et al., 2017), which uses the geometry of the target, that is previously known, to calculate its pose. In recent years, new deep learning and machine learning algorithms have been developed for these pose tracking operations.

One of the main problems in the development of these algorithms is the access to data (images or point clouds) from real cases of study. The aim of this work is to create a simulator able to generate point clouds from satellite models using the specifications of different LiDAR sensors and the relative pose between the sensor and the target. In this way, the system will be able to generate the point cloud that a LiDAR sensor should acquire for different trajectories. This simulator will take into account the errors of the selected LiDAR sensor to generate the most realistic point cloud possible.

There are two major reasons to create this point cloud generator:

• In the aerospace field, it is necessary a TRL (Technology Readiness Levels) 8 at least for hardware and software developments to be considered a “flight qualified” system. To reach this level for algorithms, it is necessary an exhaustive testing, which carries the use of a great amount of data, or in this case, point clouds.
• For deep learning algorithms training, it is necessary a great amount of classified data, which can be generated with the simulator proposed.

Other authors (He et al., 2017) have already used simulated point clouds to test pose tracking algorithms, but in this case the point cloud is generated without considering different important parameters, such as occlusions, acquisition time, LIDAR parameter (errors, point rate, range…), so the point cloud generated is not as realistic as it should be. In recent years, new deep learning algorithms (Sharma and D’Amico, 2020) have been developed for pose tracking of satellites using monocular camera systems, where synthetic images generated from the satellite model are used, so something similar can be done but using synthetic point clouds instead of images.

There are currently point cloud simulators, such as the Helios++ simulator (Winiwarter et al., 2022), from the University of Heidelberg, but they are orientated to be used in other fields, such

* Corresponding author (luismgonzalez@uvigo.es)

as civil engineering, agroforestry applications, among others. The presented point cloud simulator has been specifically designed for space operations.

2. METHODOLOGY

The developed point cloud simulator has been tested with the Envisat satellite (Figure 1) (ESA, n.d.), which is a satellite developed by ESA (European Space Agency) in 2002. It was developed for Earth Observations tasks and was equipped with different sensors, such as radiometers, radar and multispectral cameras.

Figure 1. Envisat satellite.

Figure 2 shows the workflow of the point cloud simulator developed. It is composed of three main parts: the satellite modelling, the LiDAR modelling and a raytracing algorithm. The first part, that is the satellite modelling, consist of generate a 3D model to be used by the raytracing algorithm. Then, the LiDAR model is defined, using the datasheet of a commercial LiDAR to extract its specifications. Both the LiDAR simulated and the satellite model are positioned and orientated, and after that a ray tracing algorithm is executed.

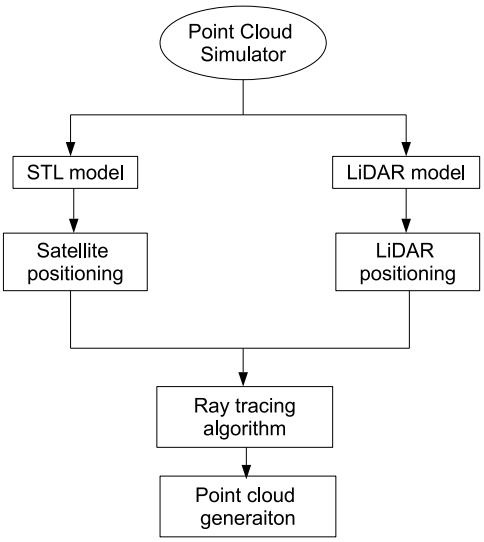


Figure 2. Workflow of the point cloud simulator

2.1 Satellite modelling and positioning.

The developed simulator uses a mesh of the satellite to generate the point cloud, tracing multiple rays and calculating the intersection point between the ray and the 3D model. In this case, the STL format is used to capture the geometry of the satellite. STL (Standard Triangle Language) is a format used by many different CAD (Computer Aided Design) programs to describe the surface geometry of an object or design using an unstructured triangulated discretization, where the corners of each triangle are points in the surface of the object modelled. Depending on the accuracy of the surface described by the triangulated surface, the number of triangles needed increase of decrease.

As was mentioned previously, for this work the satellite selected was the Envisat, so a STL model of the satellite was used (Figure 3). This STL archive is basically composed of two matrices. The first one contains the cartesian coordinates of the corners of each triangle with a similar format of a point cloud, while the second one contains the reference of the points that forms each triangle.

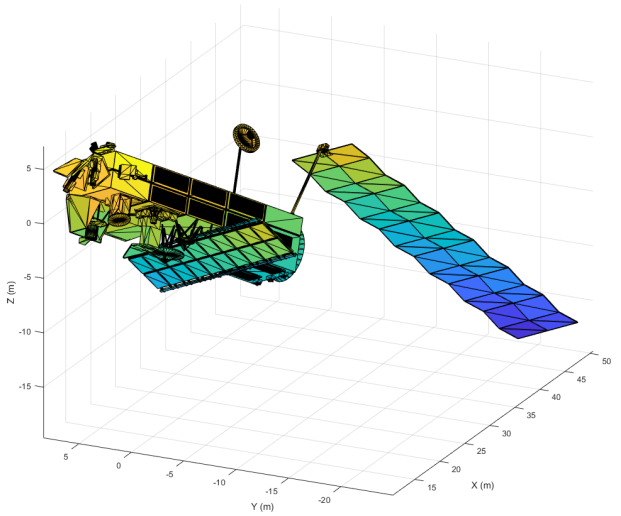


Figure 3. STL model of the Envisat satellite

For the satellite positioning, the position and orientation of the satellite are defined, applying them using rigid body transformations, that basically are three rotations (around X, Y and Z axis) and one translation. As was mentioned before, the first matrix of the STL model contains the cartesian coordinates of the corners of each triangle, so these transformations are applied to this matrix, without any modification over the second matrix, that contains the reference of the points from the first matrix that define each triangle.

2.2 LiDAR modelling and positioning.

In order to simulate a realistic point cloud, the specifications of the LiDAR have to be taken into account, not only introducing the errors of the specific sensor, but also the scanning pattern and acquisition speed. For the LiDAR modelling, the specifications of a commercial LiDAR are needed, introducing these data in the point cloud simulator. The specifications used by the simulator are:

• FoV (Field of View) type: this variable defines the type of LiDAR used (mechanical or solid-state LiDAR), and also defines the type of coverage of the LiDAR (usually 360º horizontal FOV in the case of the mechanical LiDAR sensors and a smaller one

for the solid-state LiDAR sensors). This variable is used for the scanning pattern definition.

• Horizontal and vertical FoV (in degrees).

• Maximum detection range (in meters).

• Range precision (in meters)

• Angular precision (in degrees).

•Point rate: acquisition speed (in points/s).

• Scanning pattern.

Figure 4 shows the workflow implemented for the LiDAR modelling. Once the specifications of the sensor are defined, the scanning pattern is simulated. Depending on the LiDAR selected for the simulation, the scanning pattern can be different. This process basically consists of the determination of the scanning ray's angles and its order.

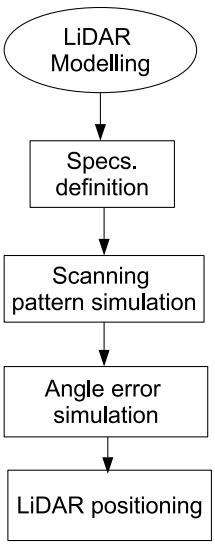


Figure 4. LiDAR modelling workflow

For this work, the LiDAR simulated is the Livox MID-70 (LIVOX, n.d.), which is a solid-state sensor designed for autonomous driving applications and mobile robots. This sensor uses a non-repetitive petal scanning pattern, which is defined using polar coordinates, using Equation 1:

$$r = A * \sin(B * \varphi), \qquad \textbf{(1)}$$

where r = radius of each point of the pattern
A = Variable defined by the FoV of the sensor
B = number of petals of the scanning pattern
φ = angle of each point of the pattern

Each time the petal pattern completes a turn ($\varphi = 360°$), a new pattern starts, but with a small rotation of 5° around its central axis, achieving in this way a complete coverage of the FoV. Figure 5 shows the intersection between the rays defined in three consecutive turns of the petal pattern and an orthogonal plane to the sensor placed one meter away from it.

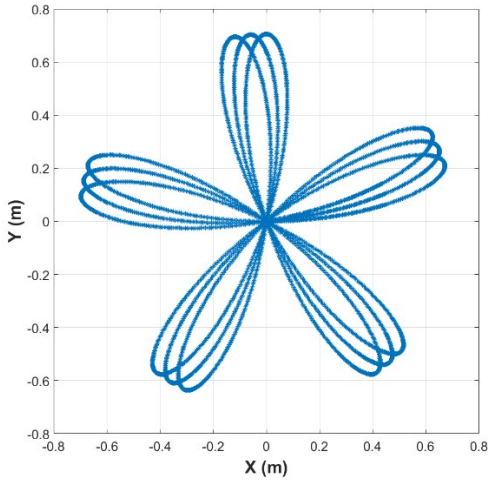


Figure 5. Three consecutive petal patterns.

The last step for the LiDAR modelling is the angular error introduction, which is the angular difference between the theorical direction of each scanning ray and the real direction of that ray during the scanning process. This error usually has a normal distribution and the value given in the datasheet of the sensor is its 1σ value. In order to introduce this error in the ray's direction generated by the scanning pattern, two rotations around the perpendicular axis of the ray are carried out, using a normal distributed random number with the same distribution of the angular error of the LiDAR. Figure 6 shows the histogram of the angular error introduced to the rays defined by the scanning pattern.

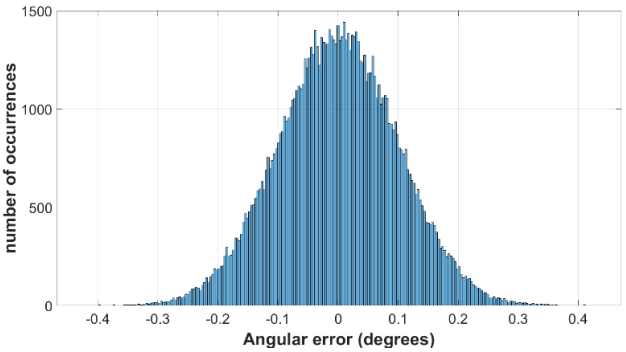


Figure 6. Normal distribution of the angular error introduced.

The last step of the LiDAR modelling is defining the pose (position and orientation) of the sensor for the simulation. Once the pose is defined, each ray of the scanning pattern is rotated and translated using rigid body transformations.

2.3 Ray tracing algorithm

Once the LiDAR and the satellite are defined and positioned (Figure 7), the next step is to carry out a ray tracing algorithm with the rays of the scanning pattern, defined in the previous section.

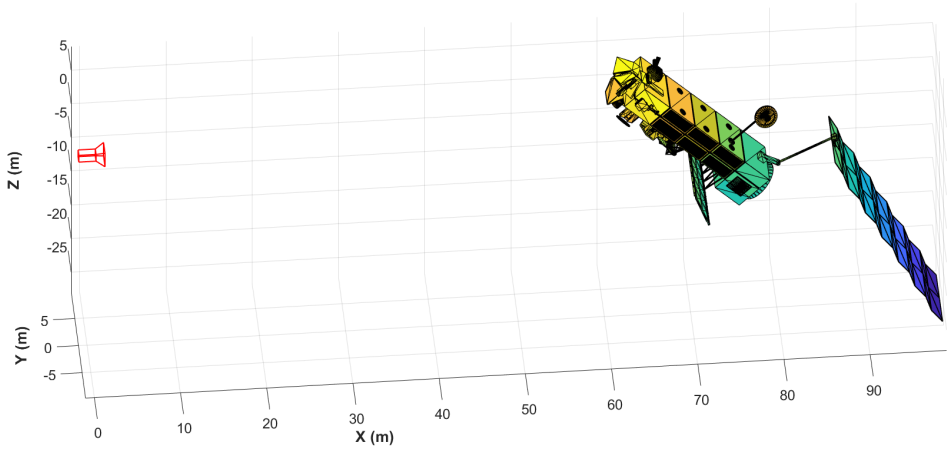


Figure 7. Positioning of the LiDAR (red) and the Satellite.

This ray tracing algorithm basically consist of calculate the intersection between each ray and 3D model of the satellite. As was mentioned before, the format used to capture the surface geometry of the satellite is STL, which is a 3D mesh composed of triangles. Due to this, the algorithm has two main parts, the first one is the identification of the tringles that are intersected by a ray, and the second one is the calculation of the intersection between the ray and the plane of the triangle.

The first part, that is the calculation of the triangles intersected by the ray, is carried out using a library developed by Tuszynski (Tuszynski, n.d.). This algorithm basically calculates which are the triangles intersected by a line (Figure 8), in this case each ray of the LiDAR, and returns their index. In some cases, the ray does not intersect the 3D model, generating no points in the simulated point cloud. As can be seen in the figure, in the mathematical model one ray can intersect more than one triangle, which cannot happen, since the ray would not reach beyond the first intersection. To define where the real intersection is made, the distance between the LiDAR and the intersections are calculated, defining the real intersection as the one with the shortest distance, which therefore is the first intersection between the ray and the object.

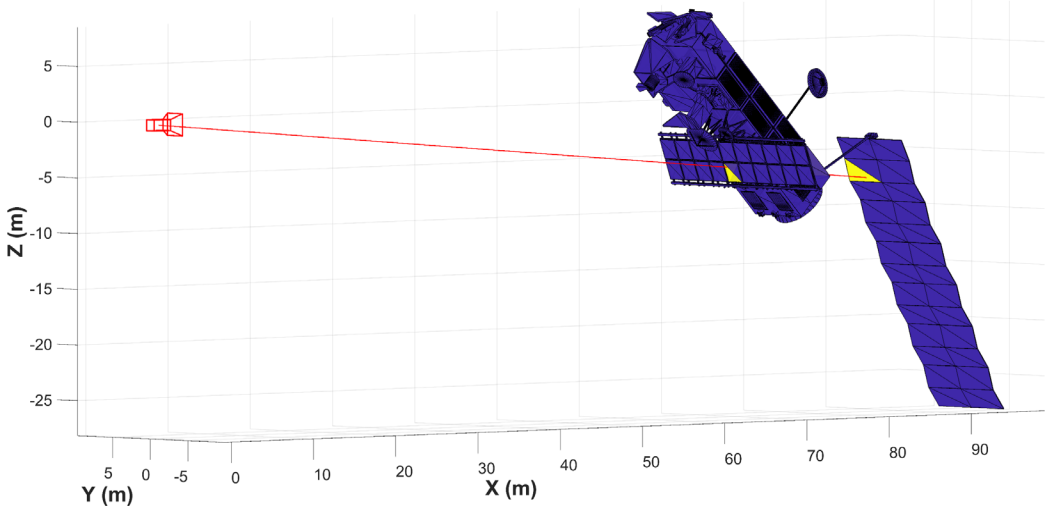


Figure 8. Ray tracing algorithm. Intersection of one ray with the STL model (red: LiDAR and ray; purple: triangles that are not intersected by the ray; yellow: triangles intersected by the ray).

Once the first intersection between the ray and the 3D model is defined, the next step is to calculate the intersection between the ray and the triangle, that basically is the intersection between a line and a plane. The plane is defined by its parametric equations (Equation 2), using the three corners of the triangle and the line of the ray was defined in the previous section, as an origin and a direction (Equation 3).

$$\begin{cases} x = P_x + \gamma u_x + \mu v_x \\ y = P_y + \gamma u_y + \mu v_y \\ z = P_z + \gamma u_z + \mu v_z \end{cases} \quad \textbf{(2)}$$

where

$$\left.\begin{matrix} u = (u_x, u_y, u_z) \\ v = (v_x, v_y, v_z) \end{matrix}\right\} \text{Vectors parallel to the plane}$$

$$P = (P_x, P_y, P_z); \text{Point on the plane}$$

$$\gamma, \mu \in \mathbb{R}$$

$$\begin{cases} x = Q_x + \lambda v_x \\ y = Q_y + \lambda v_y \\ z = Q_z + \lambda v_z \end{cases} \quad (3)$$

where

$Q = (Q_x, Q_y, Q_z)$; Point on the line
$v = (v_x, v_y, v_z)$; Direct
$\lambda \in \mathbb{R}$

As was mentioned in Section 2.2 LiDAR modelling and positioning, these sensors have two different error sources, which are the angular error, used in that section for the ray definition, and the range error, that basically is the error of the sensor for distance measurement. Once the theoretical intersection between the ray and the model is calculated, the range error is introduced. This error usually has a normal distribution defined by its 1σ deviation, given by the manufacturer in the datasheet of the sensors. In order to generate this error, a normal distributed random number with the same distribution of the range error of the LiDAR is used (Figure 9).

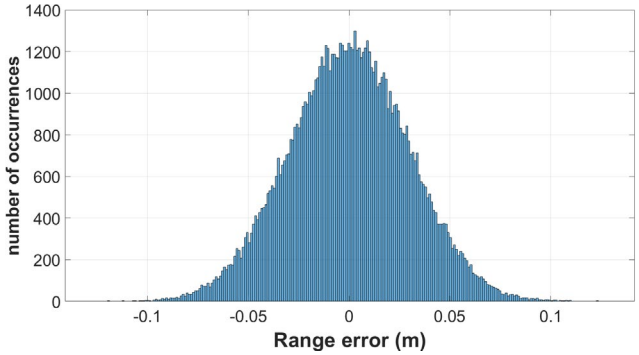


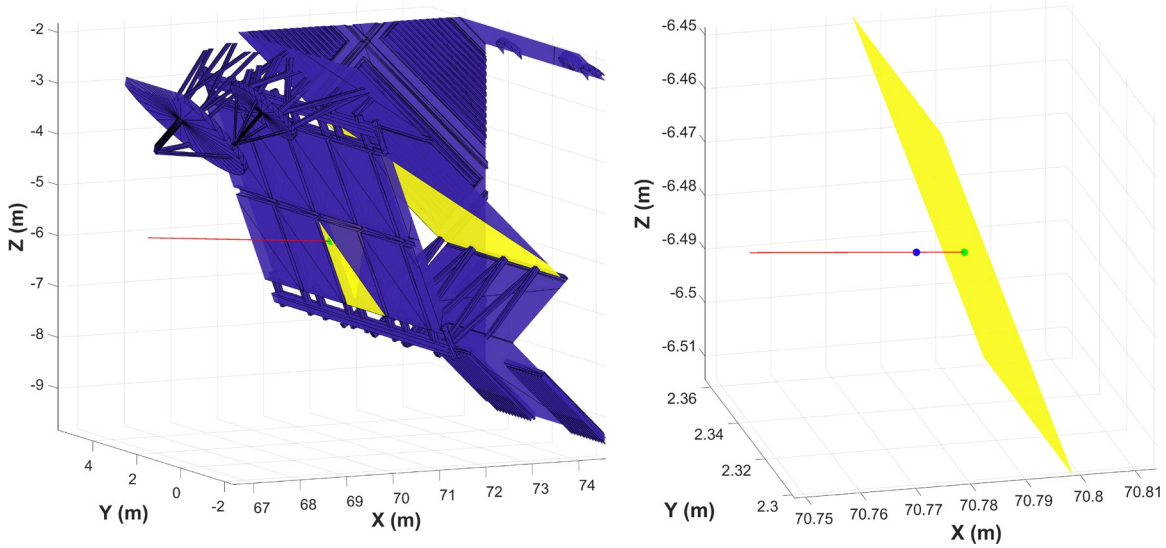
Figure 9. Normal distribution of the range error introduced.

To apply the range error, the theorical intersection and a unit vector parallel to the ray are used (Equation 4). Figure 10 shows both the theorical intersection and the intersection adding the range error.

$$P_r = P_t + e v_{dir} \quad (4)$$

where

P_r: Intersection point with the range error.
P_t: Theorical intersection.
e : Range error with a normal distribution.
v_{dir} : Unit vector parallel to the ray.



a) **b)**

Figure 10. Intersection between the ray and the 3D model. **a)** Broad view of the intersection. **b)** Close view of the intersection. (red: ray; yellow and purple: 3D model; green: theorical intersection; blue: intersection adding the range error).

3. RESULTS AND DISCUSSION

This work presents a realistic point cloud simulator developed for space in-orbit close range autonomous operations. It was tested using the 3D model of the Envisat satellite and the specifications of the Livox MID 70 LiDAR sensor. In order to simulate a point cloud, the satellite and LiDAR pose have to be defined, as well as the scanning time. With this information, the system positions both components. Once them are positioned, the rays of the LiDAR sensor are defined, using his scanning pattern and introducing an angular error. With these rays, a ray tracing algorithm is used to calculate the theorical intersection between these rays and the 3D model of the satellite. One ray can have multiple intersections with the 3D model, so intersection closer to the LiDAR section is selected as the theorical intersection. Then, the range error is introduced, using the direction of the ray. This process is repeated for all the rays of the scanning pattern. As is shown in Figure 11, it is fundamental to add the sensor errors in order to obtain a realistic result for point cloud simulation. As can be seen in the example without adding errors, the point cloud generated is homogeneous and the scanning pattern is clearly shown. Also, in the areas of the 3D model that are plane, the points of the simulated point cloud are perfectly fitted to the plane.

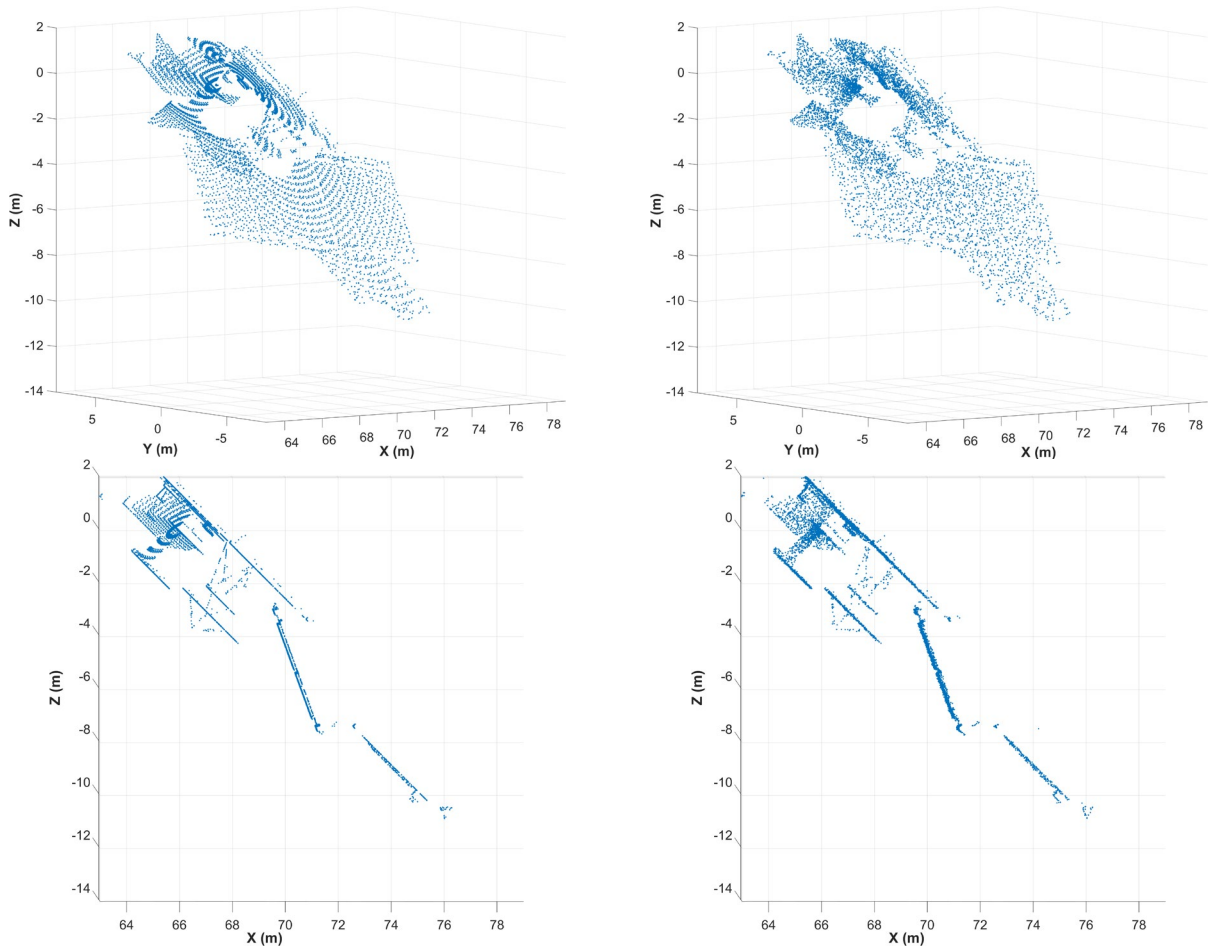


Figure 11. Point clouds generated. **a)** Point cloud without adding angular and range error. **b)** Point cloud adding both angular and range error.

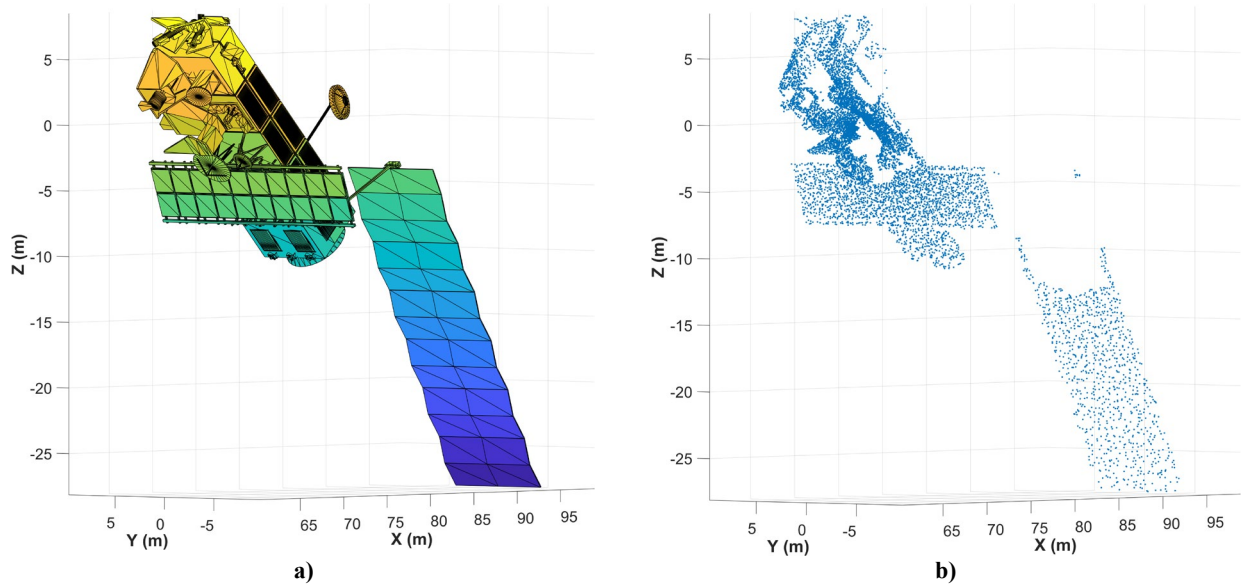


Figure 12. Point cloud simulation. **a)** 3D model of the satellite. **b)** Point cloud generated for the given 3D model.

All the algorithms of the presented point cloud simulator have been implemented in MATLAB 2021a. As was said before, the LiDAR sensor simulated in this work was the LIVOX MID-70, which is a solid-state sensor with a circular FOV of 70.4º, a point rate of 100.000 points/s, an angular precision (1σ) < 0.1º, a range error (1σ) ≤ 2cm and a detection range up to 130m (20% reflectivity). Figure 12 shows the point cloud obtained during the simulation.

4. CONCLUSSIONS

This paper presents a point cloud simulator designed for space in-orbit close range autonomous operations. This simulator uses the 3D model of the satellite or object to be simulated and the LiDAR specifications. With this information and the position and orientation of the satellite and the LiDAR, the simulated point cloud is generated. The objective of this algorithm is to generate realistic point clouds that can be used for both test point-cloud processing algorithm and for deep learning algorithm training.

As future work, the simulator will e improved, introducing the reflectivity and multiple returns effects. Also, the results will be compared with real data obtained in laboratory with a real LiDAR sensor, validating in this way the obtained results.

ACKNOWLEDGEMENTS

L.M.G.-d. was funded by the Recovery, Transformation, and Resilience Plan of the European Union–Next Generation EU (University of Vigo grant ref. 585507).

REFERENCES

ESA, n.d. Envisat [WWW Document]. URL https://www.esa.int/SPECIALS/Eduspace_ES/SEM306E3GXF_0.html (accessed 1.14.22).

He, Y., Liang, B., He, J., Li, S., 2017. Non-cooperative spacecraft pose tracking based on point cloud feature. Acta Astronaut. 139. https://doi.org/10.1016/j.actaastro.2017.06.021

Kaiser, C., Sjöberg, F., Delcura, J.M., Eilertsen, B., 2008. SMART-OLEV-An orbital life extension vehicle for servicing commercial spacecrafts in GEO. Acta Astronaut. 63. https://doi.org/10.1016/j.actaastro.2007.12.053

Liou, J.C., 2011. An active debris removal parametric study for LEO environment remediation. Adv. Sp. Res. 47. https://doi.org/10.1016/j.asr.2011.02.003

LIVOX, n.d. MID-70 [WWW Document]. URL https://www.livoxtech.com/mid-70 (accessed 3.16.22).

Opromolla, R., Fasano, G., Rufino, G., Grassi, M., 2017. Pose estimation for spacecraft relative navigation using model-based algorithms. IEEE Trans. Aerosp. Electron. Syst. 53. https://doi.org/10.1109/TAES.2017.2650785

Sharma, S., D'Amico, S., 2020. Neural Network-Based Pose Estimation for Noncooperative Spacecraft Rendezvous. IEEE Trans. Aerosp. Electron. Syst. 56. https://doi.org/10.1109/TAES.2020.2999148

Tuszynski, J., n.d. Triangle/Ray Intersection.

Winiwarter, L., Esmorís Pena, A.M., Weiser, H., Anders, K., Martínez Sánchez, J., Searle, M., Höfle, B., 2022. Virtual laser scanning with HELIOS++: A novel take on ray tracing-based simulation of topographic full-waveform 3D laser scanning. Remote Sens. Environ. 269. https://doi.org/10.1016/j.rse.2021.112772