Real-time space object tracklet extraction from telescope survey images with machine learning

Andrea De Vittori (✉), **Riccardo Cipollone, Pierluigi Di Lizia, and Mauro Massari**

Department of Aerospace Science and Technology, Politecnico di Milano, Milano Via La Masa 34, 20156, Italy

ABSTRACT

In this study, a novel approach based on the U-Net deep neural network for image segmentation is leveraged for real-time extraction of tracklets from optical acquisitions. As in all machine learning (ML) applications, a series of steps is required for a working pipeline: dataset creation, preprocessing, training, testing, and post-processing to refine the trained network output. Online websites usually lack ready-to-use datasets; thus, an in-house application artificially generates 360 labeled images. Particularly, this software tool produces synthetic night-sky shots of transiting objects over a specified location and the corresponding labels: dual-tone pictures with black backgrounds and white tracklets. Second, both images and labels are downscaled in resolution and normalized to accelerate the training phase. To assess the network performance, a set of both synthetic and real images was inputted. After the preprocessing phase, real images were fine-tuned for vignette reduction and background brightness uniformity. Additionally, they are down-converted to eight bits. Once the network outputs labels, post-processing identifies the centroid right ascension and declination of the object. The average processing time per real image is less than 1.2 s; bright tracklets are easily detected with a mean centroid angular error of 0.25 deg in 75% of test cases with a 2 deg field-of-view telescope. These results prove that an ML-based method can be considered a valid choice when dealing with trail reconstruction, leading to acceptable accuracy for a fast image processing pipeline.

KEYWORDS

space surveillance and tracking (SST)
space debris
tracklet
telescope images
machine learning (ML)
U-Net

Research Article

Received: 8 October 2021
Accepted: 21 January 2022

1 Introduction

Space pollution is among the most significant concerns of space agencies worldwide. The increasing population of resident space objects is a threat to current and future space missions. In-orbit objects are identified by their type (according to the SATCAT classification) [1]: payload and rocket bodies, fragments and debris, or not identified. Recent evaluations have estimated the number of these objects to be [2]:

- approximately 34,000 with a size larger than 10 cm;
- approximately 900,000 with a size between 1 and 10 cm (based on statistical models);
- several tens of millions with a size between 1 mm and 1 cm (still based on statistical models);
- much more with a size less than 1 mm.

They are mainly located in the Low-Earth Orbit (LEO) (55% of all operational satellites) and in the Geostastationary Earth Orbit (GEO) (35%) [1]. Space debris are artificial objects (including fragments resulting from previous collisions or parts of multistage launchers) other than a space vehicle that is active or liable to be used in another way, being in orbit. Half of them are represented by entire objects (e.g., inactive satellites or upper stages of launchers), whereas the other half are composed of fragments of various shapes and sizes (resulting from explosions or collisions) or of objects lost during previous missions (e.g., coverage, strap). The increasing number of space objects jeopardizes operative satellites with more likely collisions. Consequently, reliable methods to track and survey space object populations are crucial to grant a trustworthy Space

✉ andrea.devittori@polimi.it

Nomenclature

CCD	Charged Coupled Device	SATCAT	Satellite Catalogue
CNN	Convolutional Neural Network	SSA	Space Situation Awareness
DNN	Deep Neural Network	SST	Space Surveillance & Tracking
FCN	Fully Connected Network	TLE	Two Line Elements
FITS	Flexible Image Transport System	$Dice_{\text{coeff}}$	Dice coefficient
FoV	Field of View	err	Centroid error (pixel)
GEO	Geosynchronous Orbit	f	Interpolating elevation function (rad)
ITG	Image Tracklet Generator	r_{p2a}	Pixel to angle ratio (pixel/rad)
LEO	Low-Earth Orbit	ϕ	Lagrangian basis function
PNG	Portable Network Graphics		

Situational Awareness (SSA) with ground-based (optical, radar, and laser) and space-based sensors.

Usually, debris optical follow-up is achieved using two different techniques: the staring (or survey) mode and the chasing (or tracking) mode. With the former, the telescope is pointed at the sky while moving at a sidereal rate. Thus, the stars appear as dots in the Field Of View (FoV), whereas the debris appears as streaks (usually called "tracklets"), resulting in a bright stripe of pixels on the image. Regarding the latter, as soon as the sensor spots an object, it starts chasing the target while acquiring the image: the object appears as a single dot, whereas stars slide through the field. The images collected by these sensors were then preprocessed with photometric techniques to enhance the features to be detected by astrometric reduction techniques. In recent years, several methods have been developed.

- *Thresholding* [3]: In computer vision, thresholding is a simple method for image segmentation. Adopting this approach, a grey-scale image is converted to a binary image. These two values are assigned to pixels whose intensities are below (0) or above (1) a specified threshold. In astronomical images (and in many other fields), thresholding is used to determine which regions (connected pixels) are considered as objects and which are perceived as the background. However, the definition of an appropriate cut-off is difficult because of several factors, such as noise, background variations, or diffuse edges of the objects. Any chosen threshold may cause some true objects to be overlooked (false negatives) and some spurious objects to be considered as real (false positives). Varying the threshold to the extremes minimizes one of these types of errors but maximizes the other. Hence, it is difficult to set the threshold to be as small as possible.
- *Edge detection*: A famous implemented algorithm for border detection is Automated Streak Detection for Astronomical Images (ASTRiDE) [4]. It is a Python package for streak detection in astronomical images using the "border" for each object, that is, "boundary-tracing" and their morphological parameters. The usual steps required by ASTRIDE are:
  (1) Removing background from the FITS input image. The background map was derived using the Photutils.
  (2) Deriving the contour map of the input image. The contour level was controlled by the threshold value. By tuning this parameter, the number of detected contours of bright objects may vary.
  (3) Streak determination is based on the morphology of each border. ASTRiDE removes outer sources using morphological parameters derived from each border: the shape factor, radius deviation, and area.

  However, this type of approach fails at a low signal-to-noise ratio, and because of brightness variations along the trail.
- *Template matching* [5]: It is a general-purpose object localization technique, which allows the identification of parts of an image that match, under some criteria of similarity, an arbitrarily chosen image template. The matching problem can be summarized as follows: based on the source image I and the reference image R, determine the offset (u, v) within the search region S such that the similarity between the shifted reference image R_{uv} and the corresponding sub-image of I is the maximum. Template matching

involves two critical points: similarity measurement and search strategy. A well-prepared template is a key to successful matching. A template is a typical model or representative instance of the target of interest in an image. For space objects, a template should be a rectangular region containing a trail. This type of method is time-consuming; thus, the pixels not belonging to the trail template can be set to 0 to relieve some computational burden and to automatically filter out surrounding targets. Simultaneously, the trail results were bright, unsaturated, and isolated, meaning that overlapping or contamination with other trails, targets, or cosmic rays was avoided. Despite providing an accurate streak extraction, an astrometric reduction is generally time-consuming and requires several intermediate steps. For object tracking in LEO, telescopes with an FoV of the order of 2°–3° are commonly employed. Combining this aspect with the usually high rotation rates of observed objects in this orbital regime, a faster model would be beneficial for quick tracklet reconstruction and follow-up observation scheduling, even at the expense of accuracy.

- *GEO-FPN for RSO detection* [6]: The proposed framework leverages the Feature Pyramid Network (FPN), a Convolutional Neural Network (CNN) for image segmentation, to automate GEO RSO sensing in the telescope images. The backbone used to detect low-level patterns from images is the pre-trained EfficientNet-B7 on ImageNet. A simple preprocessing is applied to images that are overexposed to scale the pixel values of the input image, consisting of thresholding driven by training data statistics. A custom deterministic post-processing method based on vector mathematics was developed to clean false detections. The employed dataset is not balanced regarding over-exposed images and crowded scenes, and the network may fail in the case of overlapping stars, blurring owing to atmospheric disturbances, and low signal-to-noise ratio trails. In addition, no information on the inference time exists, which is essential for real-time applications.

This study aims to explore novel approaches to the problem of space track reconstruction, as opposed to traditional methods. In particular, owing to the massive collection of information during space missions and operations, data-based techniques such as Machine Learning (ML) have become increasingly efficient in effectively solving complex and diverse problems. Supervised learning techniques such as recurrent neural networks are leveraged in guidance navigation and control systems to provide solutions to optimal control problems and the analysis of orbital data files; CNNs are employed to assist the spacecraft in the landing phase through visual processing of moon images [7]; in Ref. [8], a Deep Neural Network (DNN) determines the feasibility conditions associated with the fuel-optimal powered landing; in Ref. [9], an AI-based method is developed to detect, among different features, anomalies in space objects maneuver history. In this framework, U-Net (a CNN) focuses on the reconstruction of LEO object streaks. The advantage of this network is its architecture, which is specialized in the segmentation of objects in an image. In contrast to a basic CNN, it can output an image in which a reliable shape of the target is represented as a plain color silhouette over a uniform background. This assists in the characterization of the trail shape and the computation of their centroids. Adopting this neural network improves the performance by simply fine-tuning the architecture or employing custom-made training datasets. The orbital regime at hand is the toughest to track because of the target angular speeds involved, which are much higher than the others. Furthermore, most of the existing approaches based on astrometric reduction lack rapidity in terms of processing time. Two key points are therefore considered: the timing and accuracy of the proposed solution. In addition, to gradually match real case scenario standards, both synthetic night-sky images and real images from observation campaigns are employed for algorithm performance assessment.

The novelty lies in the application and fine-tuning of this neural network category to address the optical track reconstruction task, encouraging the exploitation of the ever-growing SST-related datasets. This alternative approach allows for a gain in processing time, given a suitable preprocessing, and to facilitate the upgrade. Multiple training sessions on progressively updated catalogs would lead to increased robustness and generalization capability. First, a synthetic dataset was generated using an in-house developed optical image simulator, allowing a preliminary analysis of the application.

2 Datasets and methods

The first step in an ML application is the dataset generation, aimed, for the specific case, at producing a set of tracklet and mask images. The major issue in dealing with neural networks is to obtain a sufficient and reliable amount of data for the learning phase. However, the retrieving of a significant pool of night-sky images with tracklets and the corresponding labels is difficult and cumbersome. Consequently, a significant effort was spent to build an Image Tracker Generator (ITG): a Python application capable of creating simulated images that faithfully mimic the real case scenario.

2.1 Synthetic images and mask generation

To start the generation process, a TLE file (a data format encoding a list of orbital elements) of different known LEO space objects enters an in-house software named SENSIT [10] that provides the relative azimuth az and elevation el at every instant of each passage over a selected ground station. This information, combined with simulated black and white night-sky images [11] (as shown in Fig. 1), is sufficient for ITG to generate two image folders: one for simulated tracklet images and the other for the mask counterparts. The selected pictures are in PNG format (8-bit color depth). In ITG, the input images are preprocessed through a greyscale conversion and resized to a resolution of (512, 512), which is considered a suitable compromise between loss of information owing to downsampling and improvement of the U-Net training and testing processing time. To simulate realistic sensor shots, a uniform noise pattern between two fixed boundaries was added. Subsequently, the object passages were extracted as arrays of local angular coordinates. For each image, an available transit is shortlisted and interpolated with quadratic Lagrangian basis functions ϕ (see Eq. (1) and Eq. (2)) [12]:

Fig. 1 Synthetic sky image (black pixels are the background and white dots are stars) [11].

$$\phi_j(az) = \begin{cases} 1 + 3\dfrac{az - az_j}{h_j} + 2\left(\dfrac{az - az_j}{h_j}\right)^2, \\ \qquad\qquad az_{j-1} \leqslant az < az_j \\ 1 - 3\dfrac{az - az_j}{h_{j+1}} + 2\left(\dfrac{az - az_j}{h_{j+1}}\right)^2, \\ \qquad\qquad az_j \leqslant az < az_{j+1} \\ 0, \qquad\qquad \text{otherwise} \end{cases} \tag{1}$$

where az is the azimuth angle, and $h_j = az_j - az_{j-1}$. The subscript j denotes the j-th element of a length N vector, and $2N$ is the angle array length. The ϕ basis associated with element midpoints is expressed as

$$\phi_{j-1/2}(az) = \begin{cases} 1 - 4\left(\dfrac{az - az_{j-1/2}}{h_j}\right)^2, \\ \qquad\qquad az_{j-1} \leqslant az < az_j \\ 0, \qquad\qquad \text{otherwise} \end{cases} \tag{2}$$

The final element trial function $f(az)$ mapping az to elevation el can be written as

$$\begin{aligned} f(az) &= \sum_{j=0}^{N} el_j \phi_j(az) + \sum_{j=1}^{N} el_{j-1/2} \phi_{j-1/2}(az) \\ &= \sum_{j=0}^{2N} el_{j/2} \phi_{j/2}(az) \end{aligned} \tag{3}$$

It is assumed that $f(az)$ represents the elevation as a function of azimuth and, whenever this is not possible, the elevation turns into the independent variable and azimuth is the dependent variable. Once f is defined, a curve arc is randomly chosen between two interpolating points, and the function is evaluated in a range of values between them. The angular coordinates can be converted to pixel coordinates by knowing the equivalent sensor FoV and image resolution.

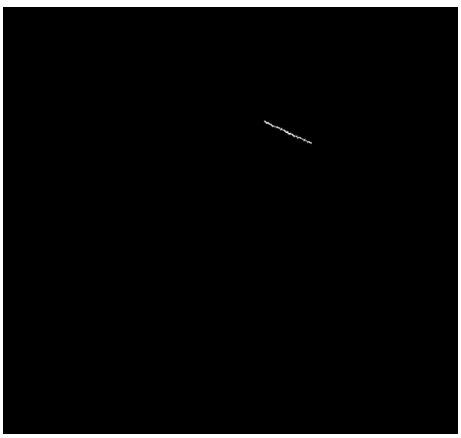
Usually, tracklets are characterized by an irregular shape; thus, to make them more realistic, the thickness is randomly changed and moves along the streak path. The only missing step is to set the fake trace in the image to realistic pixel intensity. The notion of randomness is employed, and thus, two quantities called **lb** and **ub** are defined as arrays of possible lower and upper bounds for tracklet coloring (an example of a resulting image is shown in Fig. 2(a)). Thus, the tracklet is shifted differently for each image to increase the variety of training and

testing sets. The same idea applies to the mask-creation process (see Fig. 2(b)): the background pixels were set to 0 (black), whereas the trace pixels were set to 255 (white). The aforementioned procedure aims to obtain a dataset composed of input information (images) and corresponding output problem solution (masks). They are both fed to U-Net to obtain an input–output relationship. To ease the learning phase, raw data are normalized with respect to 255, such that each pixel value shrinks between 0 and 1. The goal was to avoid gradient disappearance and divergence during training.

2.2 Training phase

The training process involves the minimization of a loss function that quantifies the distance between the modeled outputs and given labels (a pixel mask matching a given input in the case of segmentation). It is built over batches of images from the training set and adjustable weights, the latter being the variables of the problem. The optimal set is achieved by computing the gradients linking weights to the loss function to define a gradient descent-based optimization algorithm. In this case, the trained neural network is a custom version of U-Net, meant for segmentation of satellite images (further information is available in Section 3). Once convergence is reached, the weights fitting the entire training set are saved, and the model can be tested (Section 2.3).

(a)



(b)

Fig. 2 (a) A synthetic input image: the tracklet is added as a white pixel on the synthetic night sky. (b) A synthetic input mask: the night sky is completely black, and the white pixels are the target tracklets.

Before entering the network, the samples are shuffled to avoid the recognition of false patterns. The learning session was performed using 250 of 360 images. The samples were generated to ensure variability in the tracklet light intensity, size, and thickness, and to overcome overfitting during training. At each epoch, 70% of images were employed for the actual minimization and the remaining 30% for validation purposes (an unbiased evaluation of a model fit on the training distribution while tuning hyperparameters). At the end of this routine, the model was saved for testing. In Python, because of the developer community, it is likely to determine plug-and-play codes for the neural network. The same is true for Keras-unet [13], the one used for this work. It is a helper package with multiple U-Net implementations and utility tools that are helpful when working with image segmentation tasks.

2.3 Testing phase

Testing consists of the assessment of network performance: the obtained model predicts new cases (different from the training ones), the solution of which is known. If the results of the testing phase do not attest to a sufficient level of model goodness, the training is repeated. First, the code was tested on synthetic PNG images (110 of 360), and then on 80 real tracklet images. The adopted figure of merit for synthetic images is the Dice coefficient [14] (see Eq. (4)):

$$Dice_{\text{coeff}} = 2 \cdot \frac{|\boldsymbol{X} \cap \boldsymbol{Y}|}{|\boldsymbol{X}| + |\boldsymbol{Y}|} \tag{4}$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are the tracklet pixel position sets. $\boldsymbol{X}$ is the ground truth (reference tracklet), and $\boldsymbol{Y}$ is the one predicted by U-Net.

This formulation conveys a simple concept: the higher the intersection score between two sets, the greater the similarity between the two sets. This ratio can assume

values from 0 (the two sets have nothing in common) to 1 (the two sets are identical). However, a modified version is presented in Section 4 (Eq. (5)):

$$Dice^{*}_{\text{coeff}} = 1 - Dice_{\text{coeff}} \tag{5}$$

$Dice^{*}_{\text{coeff}}$ is adopted to establish how well the reference and produced tracklets overlap. To check the network precision, the centroid error was chosen as a figure of merit. The computation starts with the detection of white pixels across the output and the expected masks. Subsequently, an average of the corresponding x and y pixel coordinates was calculated to obtain the pixel grid position associated with the tracklet centroid. Next, the pixel-to-angle ratio is computed as follows:

$$r_{\text{p2a}} = \frac{FoV}{resolution} \tag{6}$$

where FoV is the characteristic dimension of the sensor field of view (in degree), whereas *resolution* is the pixel resolution of the square images used to train the network. Using r_{p2a}, the centroid error in degree is eventually defined as

$$err = r_{\text{p2a}}\sqrt{(x_{\text{c,pred}} - x_{\text{c,exp}})^2 + (y_{\text{c,pred}} - y_{\text{c,exp}})^2} \tag{7}$$

where $x_{\text{c,pred}}$ and $x_{\text{c,exp}}$ are the predicted and expected horizontal centroid pixel coordinates, respectively, and $y_{\text{c,pred}}$ and $y_{\text{c,exp}}$ represent the predicted and expected vertical coordinates, respectively.

Once the performances on the synthetic images have been assessed, it is worth evaluating how the trained network works on real images. Specifically, a set of 80 real 16-bit greyscale FITS images were provided by the Italian Air Force and taken with PdM-MITE [15], an optical telescope designed for SST by GMSpazio and Officina Stellare [16]. It is built with exclusive Riccardi-Honders flat field optical design with a diameter of 350 mm and a focal ratio f = 2:8. It is also equipped with two CCD sensors: one with a wide FoV used for surveillance tasks and the other with a narrow FoV dedicated to tracking specific objects. The latter (used as a reference benchmark for the optical part) is composed of an array of (4096, 4096) pixels in resolution, leading to a 16 Mpxl detector. FITS is an archiving format for images used in astrometric analysis. The FITS file has a header (a text file) aimed at concisely describing additional metadata (such as the tracklet centroid right ascension and declination). In selecting the test bench, a few aspects were considered:

- The tracklet should be visible in the original file. Sometimes, even if the FITS header accounts for the angular coordinates of the object, it is not necessarily easy to determine a match in the picture, even when changing the visualization settings.
- For the remaining pictures, where the tracklet is visible to the naked eye, only those that are sufficiently thick are picked. The resolution has to be decreased eight times, from (4096, 4096) to (512, 512), and by the force of circumstances, there is an information loss.

Real pictures suffer from vignetting (see Fig. 3). Vignetting is the effect in photography or optics wherein peripheral brightness of the photo appears reduced with respect to the center. Segmentation networks are susceptible to variations in brightness; to have consistent performance, all the input acquisitions should resemble the training images; otherwise, the detection might fail. To address this, a filter is specifically designed to work regardless of the telescope or sensor used during the observation campaign. Because the shots are on a 16-bit scale, they must be down-converted to 8-bit with a few steps. By switching between these two color depths, some of the information is lost. Two methods are employed to address this issue: compressing the initial color to a smaller scale or extracting a subset of the entire color range containing the features of interest (for tracklets, it is the base of the attainable spectrum). The first approach is conservative: if the image is too noisy, the output will not be entirely influenced; however, fainter tracklets may not be detected. Conversely, the second method is far more effective for dimmer tracklets; however, the noise is enhanced. The following steps were identical to synthetic PNG preprocessing.

Fig. 3 An example of real detection, taken from PdM-MITE sensor.

2.4 Post-processing phase

Post-processing aims to refine the label that originates from the model when fed with real images. Usually, the output masks not only show the trace itself but also undesired spots that have nothing to do with the tracklet. They must be filtered out; otherwise, centroid identification will be less accurate. It is possible for false positives to be valued as real traces, e.g, output mask with only sparse bright points: they do not lay down a real tracklet. If nothing is identified (for instance, because of an incorrect color depth filter), the application should retry label generation with other options. The centroid error computation is again applied as a benchmark for the algorithm performance in the real-case scenario.

3 Theory and calculation

The core of the algorithm is U-Net, a neural network belonging to the CNN family, aimed at semantic segmentation. A CNN [17] is a deep learning algorithm that can take in an input image, assign importance (through adjustable weights and biases) to various aspects/features of the image, and classify it accordingly. An example of an input can be a greyscale image consisting of a 2D matrix. The first two dimensions represent the number of pixels given by the resolution, whereas the third spans across its channels (a single one in greyscale). The input data structure information is preserved in a CNN, and the information that accompanies it. CNNs reduce images into feature maps without unwrapping them into vectors until their last layer, limiting structural feature loss that could be critical to obtaining reliable predictions. The architecture is usually organized into several layers (as shown in Fig. 4), combined to achieve effective feature decomposition and classification: convolutional, max pooling, and fully connected ones. In a convolutional layer, a filter is convolved with the input image (i.e., sliding over the image spatially, computing dot products), and a bias term is added to produce an activation map. Max pooling layers output a downsampled version of their input volumes through a max summary to reduce the data size and processing time. A fully connected layer consists of an FCN that computes a vector of scores, one for each target class. Typically, it serves as the final processing step for a CNN.

A widespread CNN application category is semantic segmentation [19]. It consists of the classification of each pixel in an image according to different labels, with no distinction between instances of the same class (for example, two cows together are perceived as one in Fig. 5).

In general, the method used to perform this task is based on a CNN aimed at generating a feature map to label every region of the image. Owing to its structure (a sequence of convolution and pooling layers), the resulting volume is characterized by low resolution. Consequently, the final model is usually complemented with a "symmetric" network in which the downsampled output can be interpolated back to the initial resolution such that a segmented version of the input is obtained.

This type of neural network can have diverse applications in SST frameworks. Whenever images are involved and a history of data analyzed with deterministic techniques is available, a structured dataset can be built, and a data-driven method can be used to address the problem. Meaningful examples can be the application developed in this study, providing a valid alternative to trail reconstruction, or a filter to delete disturbances and highlight particular features in a telescope image. In this case, the selected network was U-Net [20]. Its architecture can be broadly considered as an encoder

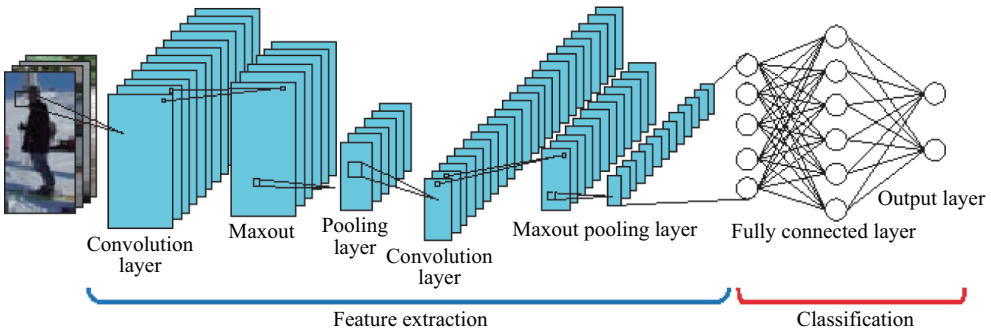


Fig. 4 An example of a simple convolutional neural network. Reproduced with permission from Ref. [18], © IEEE 2015.

network, followed by a decoder network (see Fig. 6):

- The encoder is the first half in the architecture diagram (see Fig. 6). It is usually a pre-trained classification network such as VGG/ResNet where convolution blocks are applied, followed by a maxpool downsampling to encode the input image into feature representations at multiple levels.
- The decoder is the second half of the architecture. The goal is to semantically project the discriminative features (lower resolution) learned by the encoder onto the pixel space (higher resolution) to obtain a dense classification. The decoder consists of upsampling and concatenation, followed by regular convolution operations.

The most commonly used loss function for image segmentation is the pixel-wise cross-entropy loss. This examines each pixel individually and compares the class predictions (depth-wise pixel vector) to a one-hot encoded target vector. More specifically, the last layer of U-Net performs classification using a Softmax activation

(a) (b)

Fig. 5 Comparison between an input image (a) and a segmented one (b). Reproduced with permission from Ref. [19].

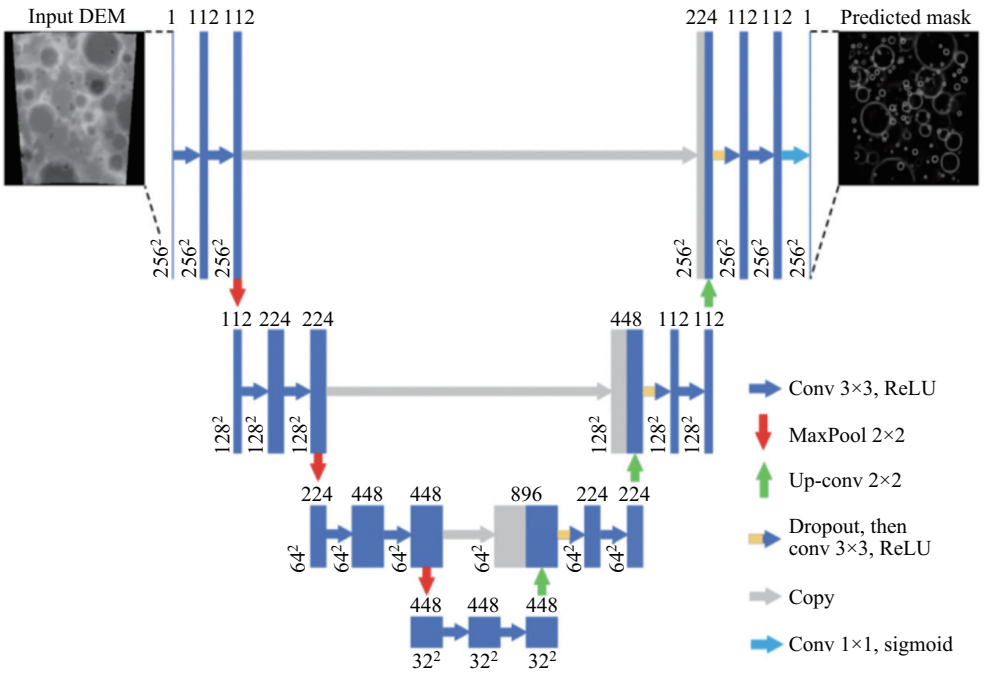


Fig. 6 Schematic representation of U-Net, with the encoder and decoder sections. Reproduced with permission from Ref. [21], © Elsevier Inc. 2018.

function p_k which generates class scores for each pixel [20]:

$$p_k(\boldsymbol{x}) = \frac{\mathrm{e}^{a_k(\boldsymbol{x})}}{\sum_{k'=1}^{K} \mathrm{e}^{a_{k'}(\boldsymbol{x})}} \tag{8}$$

where a_k denotes the k-th kernel channel comprising the final feature map element for pixel position $\boldsymbol{x}$, and K represents the total number of classes (a single one in this case). The resulting feature map is then compared with the ground-truth mask using a cross-entropy loss function type for each pixel $\boldsymbol{x}$:

$$L(\boldsymbol{x}) = -\sum_{k'=1}^{K} t_{k'}(\boldsymbol{x}) \log(p_{k'}(\boldsymbol{x})) \tag{9}$$

where the distance from the ground-truth pixel mask is considered for each class k' by comparing the true label $t_{k'}$ with $p_{k'}$.

The most important network entries are summarized in Table 1, highlighting the network structure, specifications of the minimization process, and input type.

Table 1 U-Net entries

Input	Value
Image size	(512, 512, 1)
Classes	1
Batch size	2
Epochs	2
Validation split	0.3
Number of layers	6
Input	**Value**
Filters	64
Up-conv-filters	96
Output activation	Sigmoid
Loss	Binary cross entropy
Optimizer	Adam
Metrics	Accuracy

4 Results

This section reports the results of the training and testing phases. The training loss function helps to determine the best possible values for the weights and biases that would provide the right class attribution for each input. Progressively reaching loss function convergence to a near-null value is the first requirement for the network to make predictions with unseen data (validation set and real case scenarios). In the present case, the final loss value was 0.0270 after 350 iterations (see Fig. 7). The total number of iterations is composed of two distinct epochs, each corresponding to an optimization process

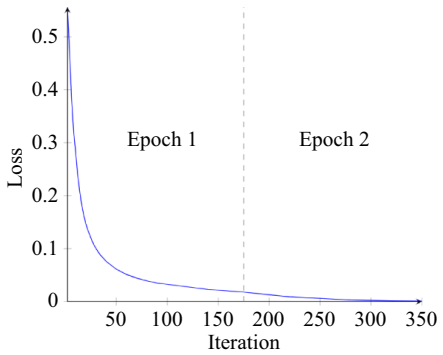


Fig. 7 Loss function trend during the training process.

window, where the weights are initialized using the ones coming from the last iteration of the previous epoch.

Testing consists of the assessment of model performance. U-Net is used to predict new cases (different from the training cases), the solution of which is known. Different types of tests were performed on synthetic and real images. Regarding the former, the relevant benchmarks are $Dice^*_{\mathrm{coeff}}$ and the centroid error distribution (see Figs. 8 and 10(a) and Table 2).

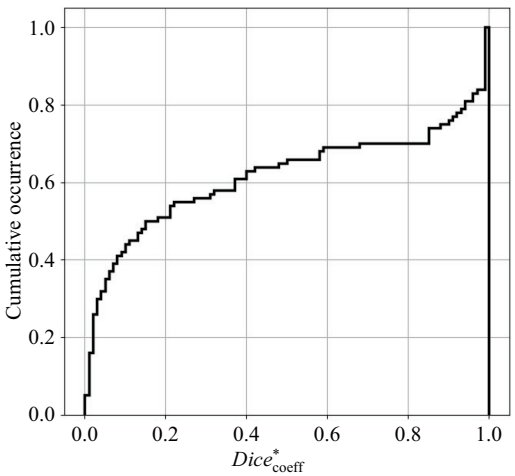


Fig. 8 Cumulative plot of the occurrence of Dice coefficient values obtained with synthetic images.

Table 2 $Dice^*_{\mathrm{coeff}}$ percentiles on synthetic input images referred to Fig. 8

Percentile	$Dice^*_{\mathrm{coeff}}$
25%	2.99×10^{-2}
50%	1.83×10^{-1}
75%	9.09×10^{-1}

$Dice^*_{\mathrm{coeff}}$ is very conservative when the establishment of good outcomes is concerned. When a tracklet is partially recognized and there is no other spurious illuminated

pixel, it might score close to 0.5, or even more. In the same scenario, the error associated with the centroid could still be acceptable (see Fig. 9). As shown in Fig. 8, 75% (the third quartile) of the analyzed input pictures have a $Dice^*_{\text{coeff}}$ value of 9.09×10^{-1}, which does not seem satisfactory; however, looking at the centroid error cumulative distribution, at the same percentage, the error is approximately 1.46×10^{-1} deg (see Table 3 for a detailed percentile description).

Moving to a real-case scenario (see Fig. 10(b) and Table 4 for a precise percentile description), the Dice coefficient cannot be evaluated because no exact output masks are available. It is only possible to compute the error between the estimated centroid and the exact centroid from the FITS metadata.

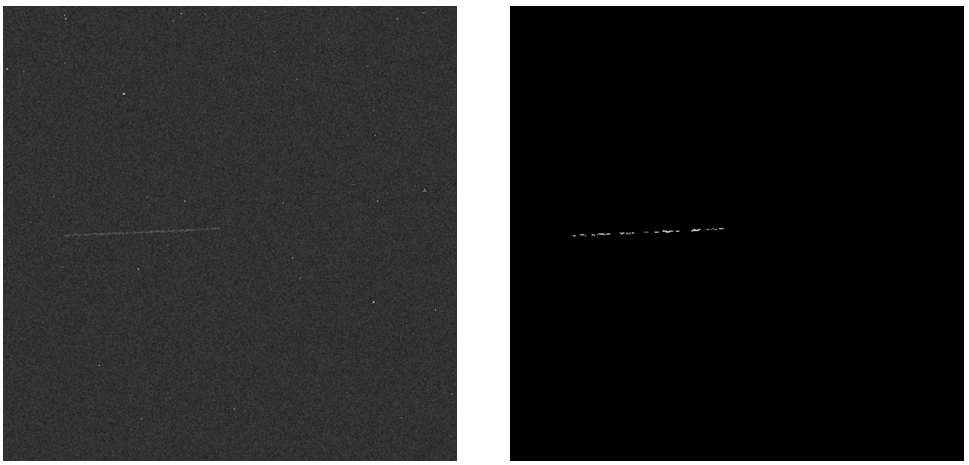
Table 3 Centroid error percentiles for the synthetic case linked to Fig. 10(a)

Percentile	Centroid error (deg)
25%	2.88×10^{-3}
50%	3.26×10^{-2}
75%	1.46×10^{-1}

Table 4 Centroid error percentiles for the real case linked to Fig. 10(b)

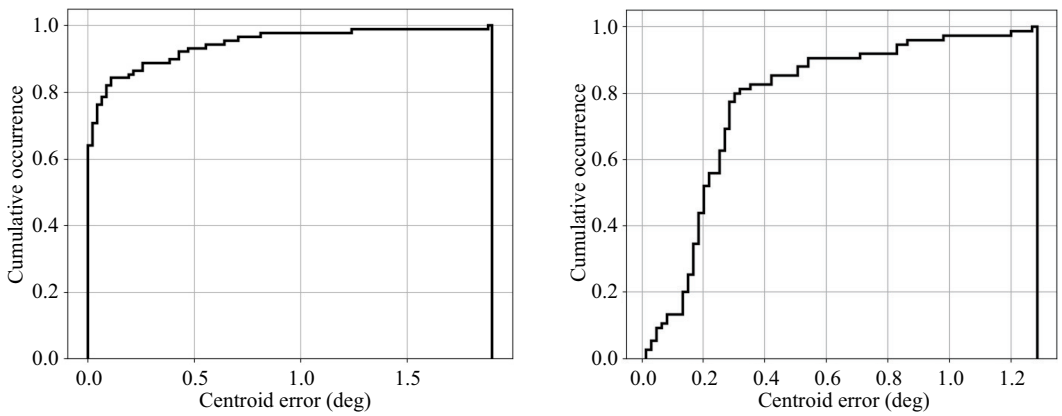
Percentile	Centroid error (deg)
25%	1.69×10^{-1}
50%	2.16×10^{-1}
75%	2.92×10^{-1}

In 75% of the cases (the third quartile), the error is below 2.92×10^{-1} deg (the estimated FoV is



(a) (b)

Fig. 9 (a) Synthetic tracklet; (b) equivalent prediction for $Dice^*_{\text{coeff}} \approx 0.5$.



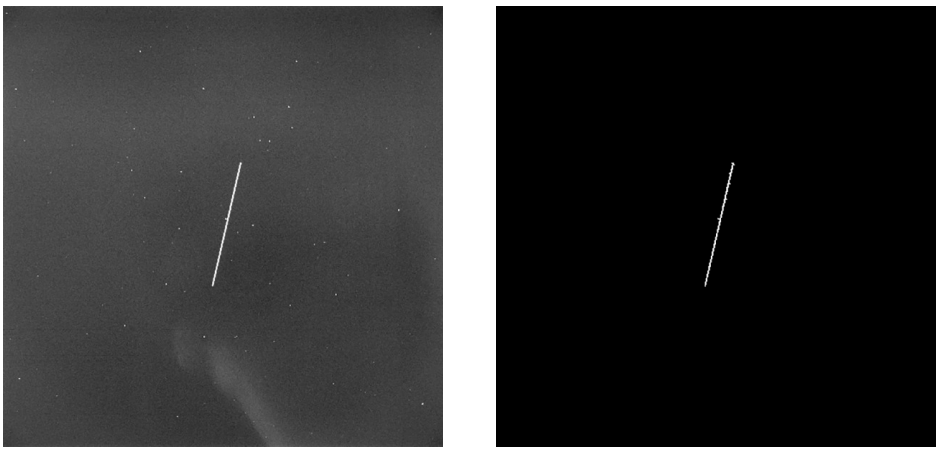
(a) (b)

Fig. 10 Cumulative plots of the occurrence of centroid error values obtained with synthetic (a) and real (b) images.

approximately 2°; thus, most of the time, the related error would be less than 14%). An example with a low estimation error is presented in Fig. 11. A high error value from 0.4 to 0.5 deg on the horizontal axis is associated with false positive detection (see Fig. 12). The outcome could vary owing to:

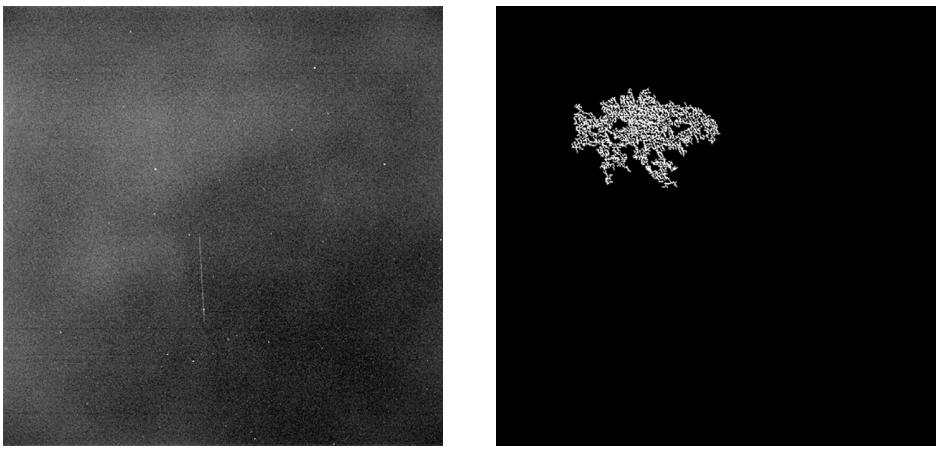
- The resolution reduction of the input pictures from 4096 to 512.
- Information loss owing to the transition from 16 to 8 bits.
- False positives may be perceived as true positives in wrong grid locations.
- The tracklet is identified together with outer sources, resulting in a poor estimation.

The implemented algorithm should balance both accuracy and processing time to enable immediate follow-up observations, possibly during the same transit of the object in the sensor field of regard. Therefore, the cumulative distribution showing the timing performance is shown in Fig. 13 and Table 5. In 75% of the cases, the total time (summation of preprocessing, testing, and post-processing) for a single prediction is approximately 1.22 s. The distribution is split into two regions: on the left, the prediction is made only once; on the right, it is performed twice, probably owing to an incorrect initial color depth filter choice in the first attempt, in which case the processing time doubles. The test bench is the Python version of the code, running on the CPU of a machine equipped with an AMD 3700X (3.6 GHz) and



(a) (b)

Fig. 11 (a) A bright trail (NORAD ID 31598); (b) the linked output mask.



(a) (b)

Fig. 12 (a) A cloudy scene (ID 33412); (b) the relative false-positive mask.

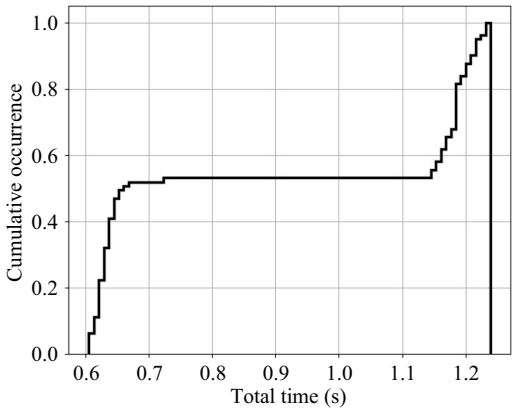


Fig. 13 Cumulation of the occurrence of a range of total processing time obtained with real images.

Table 5 Total processing time percentiles obtained with real FITS image processing

Percentile	Total time (s)
25%	6.56×10^{-1}
50%	6.92×10^{-1}
75%	1.18

16 GB of RAM.

Timing mostly depends on the computer hardware. However, U-Net grants sufficiently low processing time to predict the object's angular position such that the telescope pointing can be modified to follow its trajectory in real time. This is the driving parameter for the entire algorithm development, and the few seconds needed for the Python code to run on a desktop clarify that: if translated into operative language, processing time can reach even one half or a third of the obtained result and that target follow-ups can be performed. Regarding the centroid error values, they may seem large if the FoV of the sensor (a few degrees) is considered. However, because the downstream actions performed by the telescope do not require high precision (the sensor needs an area in which it is confident to a certain level to determine the object after the image acquisition), it is considered acceptable for the application. Precise astrometric reduction and orbit determination can be performed offline as image post-processing with traditional methods, with no waste of processing time during the tracking phase.

5 Conclusions

Based on the results, the network can generate reliable masks, performing better with synthetic input images than with real ones. This accuracy difference occurs because similar simulated image distributions were used for both the training and testing phases, and real images feature different disturbances such as clouds and stray light sources. The time taken for every photo to be processed was always under 1.22 s. The development of a compiled application, rather than an interpreted one, can undoubtedly relieve it from processing overhead. Furthermore, the Dice test run on synthetic images, despite not promising, shows that this figure of merit is sensitive to the small number of scattered pixels usually involved in the tracklet coloring. Consequently, even partial recognition of the trail leads to a meager result. The key findings are reported in Table 6.

Table 6 Total time for prediction, centroid error with simulated and real images, and Dice coefficient on synthetic masks

Quantity	Performance
Total time (75%)	1.18 s
ε synthetic case (75%)	0.15°
ε real case (75%)	0.29°
$Dice^{*}_{\text{coeff}}$ (75%)	0.91

In addition to showing significant achievements, some improvements can be implemented to refine them. Owing to limitations in the scenario generation variety, the trained model cannot handle clouds and similar disturbances. It would be better to start with a set of training images taken from real telescope shots and append an extracted streak from other observations in advance. This addition will make the network more robust against unexpected circumstances and also increase the success rate and cause the overall error to drop. A drastic change in network architecture can result in more efficient image processing, such as object detection or instance segmentation. In contrast to U-Net and traditional semantic segmentation, the aforementioned techniques succeed in the perception of targets belonging to the same class as different entities. A further step forward could be the integration of this tool as part of an autonomous space object tracking pipeline to be performed by optical telescopes equipped with a suitable processing unit.

Acknowledgements

The authors would like to acknowledge the crucial support of the Italian Air Force with the invaluable material provided. The numerous images from their telescope

observation campaign were key for an adequate algorithm testing phase.

Funding note

Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement.

Declaration of competing interest

The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] European Space Agency. ESA's Annual Space Environment Report. Technical Report 4.0. ESA Space Debris Office, Darmstadt, Germany, **2020**. Available at https://www.sdo.esoc.esa.int/environment_report/Space_Environment_Report_latest.pdf.

[2] Bennett, A. A., Schaub, H., Carpenter, R. Assessing debris strikes in spacecraft telemetry: Development and comparison of various techniques. *Acta Astronautica*, **2021**, 181: 516–529.

[3] Masias, M., Freixenet, J., Lladó, X., Peracaula, M. A review of source detection approaches in astronomical images. *Monthly Notices of the Royal Astronomical Society*, **2012**, 422(2): 1674–1689.

[4] Kim, D.-W. ASTRiDE: Automated streak detection for astronomical images. **2016**. Available at https://github.com/dwkim78/ASTRiDE (accessed: 11.03.2021)

[5] Du, J., Hu, S., Chen, X., Guo, D. Improved space debris astrometry with template matching. In: Proceedings of the 1st NEO and Debris Detection Conference, **2019**.

[6] Abay, R., Gupta, K. GEO-FPN: A convolutional neural network for detecting GEO and near-GEO space objects from optical images. In: Proceedings of the 8th European Conference on Space Debris (virtual), **2021**.

[7] Izzo, D., Märtens, M., Pan, B. F. A survey on artificial intelligence trends in spacecraft guidance dynamics and control. *Astrodynamics*, **2019**, 3(4): 287–299.

[8] Song, Y., Miao, X. Y., Cheng, L., Gong, S. P. The feasibility criterion of fuel-optimal planetary landing using neural networks. *Aerospace Science and Technology*, **2021**, 116: 106860.

[9] Lane, B., Poole, M., Camp, M., Murray-Krezan, J. Using machine learning for advanced anomaly detection and classification. In: Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference, **2016**.

[10] Purpura, G., De Vittori, A., Cipollone, R., Di Lizia, P., Massari, M., Colombo, C., di Cecco, A., Salotti, L. SENSIT: A software suite for observation scheduling and performance assessment of SST sensor networks. In: Proceedings of the 72nd International Astronautical Congress, **2021**.

[11] In-The-Sky.org. Guides to the night sky. Available at https://in-the-sky.org/skymap.php (accessed: 11.03.2021)

[12] Burden, R. L., Faires, J. D. *Numerical Analysis*, 5th edn. Boston: PWS-Kent Publishing Company, **1993**.

[13] KZak. keras-unet 0.0.7. Available at https://pypi.org/project/keras-unet/ (accessed: 11.03.2021)

[14] Zou, K. H., Warfield, S. K., Bharatha, A., Tempany, C. M. C., Kaus, M. R., Haker, S. J., Wells, W. M. III, Jolesz, F. A., Kikinis, R. Statistical validation of image segmentation quality based on a spatial overlap index1: Scientific reports. *Academic Radiology*, **2004**, 11(2): 178–189.

[15] Del Genio, G. M., Paoli, J., Del Grande, E., Dolce, F. Italian air force radar and optical sensor experiments for the detection of space objects in LEO orbit. In: Proceedings of the 16th Advanced Maui Optical and Space Surveillance Technologies Conference, **2015**.

[16] Officina Stellare website. Available at https://www.officinastellare.com/ (accessed: 11.03.2021)

[17] Yamashita, R., Nishio, M., Do, R. K. G., Togashi, K. Convolutional neural networks: An overview and application in radiology. *Insights into Imaging*, **2018**, 9(4): 611–629.

[18] Fukui, H., Yamashita, T., Yamauchi, Y., Fujiyoshi, H., Murase, H. Pedestrian detection based on deep convolutional neural network with ensemble inference network. In: Proceedings of the IEEE Intelligent Vehicles Symposium, **2015**: 223–228.

[19] Li, F. F., Johnson, J., Yeung, S. Lecture 11: Detection and segmentation. Stanford University, **2018**. Available at http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture11.pdf.

[20] Ronneberger, O., Fischer, P., Brox, T. U-Net: Convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. Lecture Notes in Computer Science, Vol. 9351*. Navab, N., Hornegger, J., Wells, W., Frangi, A. Eds. Springer Cham, **2015**: 234–241.

[21] Silburt, A., Ali-Dib, M., Zhu, C. C., Jackson, A., Valencia, D., Kissin, Y., Tamayo, D., Menou, K. Lunar crater identification via deep learning. *Icarus*, **2019**, 317: 27–38.

Andrea De Vittori, born in 1995 in Milan, obtained his M.Sc. degree in space engineering and his B.Sc. degree in mechanical engineering at Politecnico di Milano University. His master thesis was related to the application of AI techniques for space debris track reconstruction of LEO objects from optical and bistatic radar observation campaigns. From June 2020 to January 2021, Andrea worked as a research fellow at the same institution. Currently, he is a Ph.D. student in space engineering, focusing on the development of methods to compute the probability of collision and collision avoidance maneuver design algorithms for both impulsive and low-thrust maneuvers. E-mail: andrea.devittori@polimi.it.

Riccardo Cipollone was born in 1995 in Teramo. He obtained an aerospace engineering B.Sc. degree and a space engineering M.Sc. degree with a thesis on machine learning-based techniques for optical and radar track reconstruction of LEO objects at Politecnico di Milano. After working as a research fellow at the same institution, he is currently a Ph.D. student in aerospace engineering. He is presently working on novel advanced correlation methods, spacecraft maneuver recognition, and optimal sensor tasking for space surveillance and tracking. E-mail: riccardo.cipollone@polimi.it.

Pierluigi Di Lizia is an assistant professor of aerospace mechanics and spacecraft guidance and navigation at the Department of Aerospace Science and Technology of Politecnico di Milano. His main research areas include space surveillance and tracking, space situational awareness, guidance navigation, and control of proximity operations. He has 17 years of experience in projects involving research agencies and private companies. He has been co-author of over 160 papers (41 peer-reviewed journals). He is an associate editor of the COSPAR journal *Advances in Space Research*. He is a member of the Italian delegations to the Inter-Agency Space Debris Coordination Committee (IADC) and the Space Mission Planning Advisory Group (SMPAG). He is an Italian member of the CapTech Simulation of the European Defense Agency. E-mail: pierluigi.dilizia@polimi.it.

Mauro Massari is an associate professor at the Department of Aerospace Science and Technology of Politecnico di Milano, where he lectures on payload design in the space engineering M.Sc. program. He obtained his M.Sc. degree in aerospace engineering in 2000 from Politecnico di Milano and his Ph.D. degree in aerospace engineering in 2005 from Politecnico di Milano. His research interests are focused on the fields of numerical astrodynamics, spacecraft guidance navigation and control, uncertainty propagation, space surveillance and tracking, and space robotics. E-mail: mauro.massari@polimi.it.