AerospaceEurope Conference2021
WARSAW, POLAND | 23 -26 NOVEMBER 2021
CEAS Council of European Aerospace Societies

MACHINE LEARNING TECHNIQUES FOR DETECTION AND TRACKING OF SPACE OBJECTS IN OPTICAL TELESCOPE IMAGES

Jason Calvi[1], Alessandro Panico[1], Riccardo Cipollone[1], Andrea De Vittori[1] & Pierluigi Di Lizia[1]

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

Abstract

Performing real time and robust space object detection by means of optical acquisitions is a challenging task in space surveillance, especially when tracking involves uncatalogued objects. Conventionally, this problem is addressed by means of traditional detection or segmentation based techniques to identify the tracklets relative position in the sensor field of view. That said, the required computational time represents the main limitation for real-time applications. This paper proposes two innovative tools based on machine learning techniques, respectively a real-time object detector and a tracker. The first is a Convolutional Neural Network (CNN) based on YOLOv5 (You Only Look Once version 5) architecture. Notably, night sky acquisitions containing one or more tracklets are inputted to the network; the output is a text file containing the position and dimension information of the tracklet bounding box. The development of a machine learning application requires several steps, including dataset creation, pre-processing, training, and testing. To optimise network accuracy and execution time, the whole process is repeated for different datasets featuring both synthetic and real telescope acquisitions. The results obtained on the validation dataset exhibits an accuracy of 98%, and taking image pre-processing into account in the overall computational cost shows that the end-to-end process requires about 7 seconds to be executed. Consequently, the processing time is compatible with one of typical single telescope acquisition. As regards the second tool, it is devised to assess the object angular path by means of a linear regression on multiple detections. Since trajectory estimation depends on bounding boxes, the network accuracy plays a crucial role in object identification.To match the requirements of real-time application in spite of path estimation computational burden, this technique adopts a faster but less efficient image processing, followed by detection and finally a tracking script. The total time required in this case is about 1.5 seconds, and the accuracy drops to 91%. Both softwares tools are developed in Python programming language and executed on a 2017 machine with an i7-7700HQ CPU, 16Gb of RAM and a GTX 1050 graphics card with 4Gb of VRAM. The results achieved in pinpointing tracks prove that the detection network could represent a valid alternative to traditional techniques, and that machine learning can be a convenient addition to object tracking pipelines.

Keywords: Telescope Acquisitions, Real Time, Tracking, Detection, Machine Learning

1. Introduction

A space debris is defined as a man-made and non operational objects orbiting the Earth. In most cases, they are generated due to tank explosions and impacts between detritus and satellites. Even small objects can represent a threat because of their high velocity and high kinetic energy. In recent years the number of launches has decreased [1], but space pollution has not slowed down due to a higher mass lifted into orbit. Nowadays a launcher can carry several satellites and it is made up of several stages and detachable parts released during the orbit insertion phase of the payload. The increasing overpopulation of space objects could jeopardize the realization of future space missions, both in the short and long term.

The vast majority of space junk population is composed of small objects. Most are generated by the deterioration owing to the hostile space environment and by the thrusters waste products. The typical diameter of these particles is on the order of millimeters and sub-millimeters.
Space debris is mainly located in two orbital regimes [2]:

- Low Earth Orbit (LEO): Earth-centered orbit with altitude ranging from 160 to 2000 kilometers and rotation period spanning from about 90 minutes to 120 minutes. Approximately 55% of satellites belong to LEO for remote sensing missions.
- Geostationary Orbit (GEO): features an orbital period equal to Earth's rotational one. Around 35% of the satellites lie in o this region, making it the second most populous range. GEO orbits are typically used for telecommunications, defense, and meteorology.

Space Surveillance and Tracking (SST) covers a key role in monitoring and tracking objects in orbit. It defines the set of operations to be performed to keep space pollution under control and track population changes of man-made space debris. In this context data and measurements processing (radar and optical in particular) represents the core of these activities, providing information for catalogues update and maintenance. Optical telescopes measurements often represent the only available type of data for RSOs (Resident Space Objects) detection and tracking, an important asset to reach Space Situational Awareness.
Methods to perform object detection generally belong to two main categories: deterministic or neural network-based approaches [3]. The former usually requires the selection of targets features intrinsic to the problem at hand and also the development of an ad-hoc algorithm to detect them. An example of deterministic approach is represented edge detection implemented in the ASTRiDE Python package [4]: a custom image processing technique for finding the boundaries of any light source in astronomical images. As explained by Zou et al. [5], neural network-based object detection consists in locating and classifying targets of interested inside a shot. This, in turn, can be divided into two macro-categories: two-stage and one-stage detectors. In the first case the detection occurs in several stages, following a "coarse-to-fine" policy. The second one instead employs process attempts to complete the recognition in a single step using a single network. The most prominent two-stage detector is Faster R-CNN, while the most popular one-stage detectors are SSD (Single Shot Detection) and YOLO (You Only Look Once).
In this work, the latest version of YOLO is employed to relieve the computational burden of traditional tracklet detection pipelines while keeping comparable accuracy.

2. YOLO (You Only Look Once)

You Only Look Once (YOLO) is a real-time object detection framework, in which the image is only passed once through a Convolutional Neural Network (CNN). It is a family of compound-scaled object detection models that include simple functionality for Test Time Augmentation (TTA), model ensembling and hyperparameter evolution [6]. The models are characterised by high performance and lightweight in terms of processing time, making them ideal candidates for real time conditions and on-device deployment environments. The architecture of this network (from version 1 to version 4) is based on Darknet, a flexible research framework while its most recent version with several differences and improvements, including the implementation of the famous mosaic augmentation technique has been implemented with PyTorch, an open source machine learning framework easy to configure and use. Comparing this architecture to other object detection techniques:

- it sees the whole image during training and testing, then implicitly encodes contextual information about classes and their appearance;
- it learns generalizable representations of objects so that when trained on natural images and tested on works of art, the algorithm outperforms other high-level detection methods.

As the name says, YOLO "looks only once" at the image, meaning that it requires only one pass of forwarding propagation through the neural network to make predictions.

The task of an object detection-aimed neural network consists in creating features from input images and then to feeding them to a prediction system in order to obtain boxes around target objects while predicting their classes (see Fig. 1).

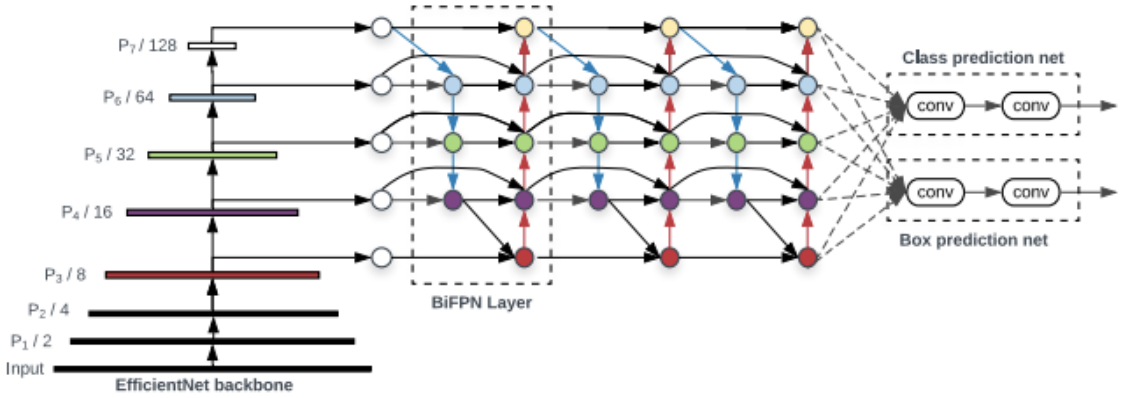


Figure 1 – Architecture of the layers of YOLO [6].

The YOLO models are the first object detectors to connect the procedure of predicting bounding boxes with class labels in an end-to-end differentiable network, as shown in figure [7]. The network is structured in three main components (see Fig. 2):

- Backbone: is a convolutional neural network that aggregates and forms image features at different granularities.
- Neck: is a series of layers to mix and combine image features to pass them forward to prediction.
- Head: is composed of consumers' features from the neck, and it takes the box and class prediction steps.

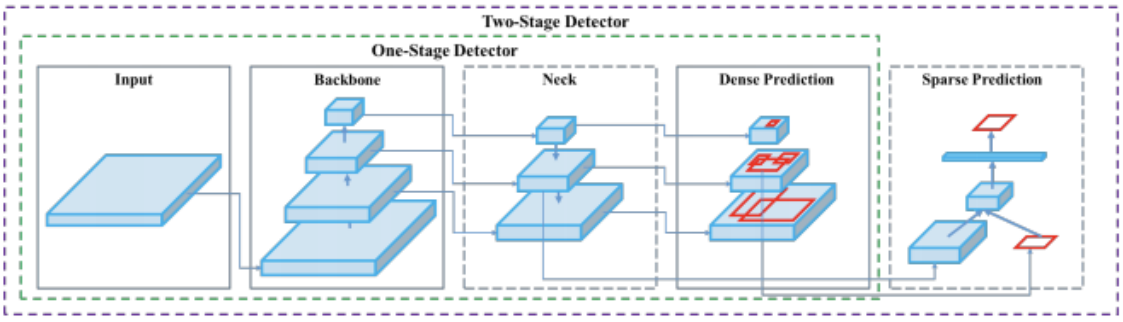


Figure 2 – Main components architecture of YOLO [6].

Many object detection algorithms, such as Faster R-CNN, MobileNet SSD, and YOLO, use mAP to evaluate their models and publishing their research. Also in this paper, this parameter is used as reference to measure the system performance and to compare different systems. For object detection tasks, Precision and Recall are calculated using the Intersection Over Union (IoU) value for a given IoU threshold. This parameter is calculated for each detected object and is defined as the overlap between the predicted bounding box and the ground truth bounding box (see Fig. 3) [8].
The typical value used for IoU threshold is 0.5, so:

- If the IoU value for a prediction is > 0.5, then the prediction is classified as True Positive (TP).
- If IoU is < 0.5, it is classified as False Positive (FP).

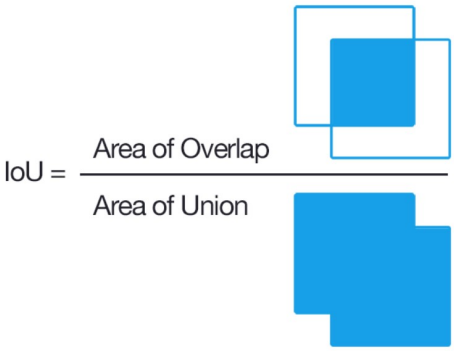


Figure 3 – Intersection over Union representation [9].

- If the element is not detected or if the IoU is > 0.5 but the classification is wrong, it is classified as False Negative (FN).
- Precision: the ability of a model to identify only the relevant objects;
- Recall: the ability of a model to find all the relevant cases.

By having TP, FP and FN formally defined, the precision and recall of detection for a given class across the test set are calculated as:

$$Precision = \frac{TP}{TP+FP}$$
$$Recall = \frac{TP}{TP+FN} \tag{1}$$

The AP is then calculated by taking the area under the Precision-Recall curve. The mAP is the average of the AP calculated for all the classes.

3. Datasets Generation

Gathering data is one of the most important stages of machine learning workflows. Specifically, thousands of synthetic night sky images with corresponding tracklets and YOLO labels are generated using TIG [10] software. It allows diverse dataset thanks to tacklets random length, thickness, position, and inclination, beside the addition background noise in synthetic telescope acquisitions. The dataset consists of about 4000 images: 70% of them are used for the training phase, 20% for the validation phase, while the remaining 10% are devoted to the testing phase.
On the other hand, real observations in FITS format are processed and employed in two different networks. In this case, manual labeling is necessary. The software inputs are the images to be processed, while the outputs are text files compatible with YOLO [6].

4. Image processing

The images produced by a generic telescope are not ready-to-use for the analysis; a series of transformations are needed to make the tasks at hand easier. Moreover, FITS images are not compatible with YOLO [6], due to the 16-bit architecture. Two different image elaboration processes based on two distinct libraries are leveraged to create the detector and the tracker datasets:

- RID dataset: employs the Scikit-Image Python library [11] (shortened Skimage). It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. In this case, once the *.fits* image is converted into a 4096x4096 matrix (corresponding to the telescope acquisition resolution) it is adjusted through the scikit-image exposure, util, filters, and transform modules. Finally, the image is saved in *.png*.

- LOT dataset: It is based on the Python library Matplotlib.pyplot (shortened PyPlot), In this case, once the fits file is converted into a matrix of 4096x4096 elements, it is resized keeping the same color range of values and an anti-aliasing filter is applied. Subsequently, thanks to the imshow visualization function of pyplot are applied a data range equalisation to emphasizes low light sources, and a Lanczos interpolation that consists of re-pixeling the image to make it more uniform and more grainy. Thus, the quality of the tracklets is improved and they appear smoother so easier to detect. Finally, the image is saved in *.png* format.

In short, Skimage transformations aim at obtaining the highest possible quality for improving the output prediction in the only detection case. Conversely, PyPlot processing converts images in a shorter time without penalizing quality too much in order to create an efficient and responsive tracker.

5. Real Images Detector

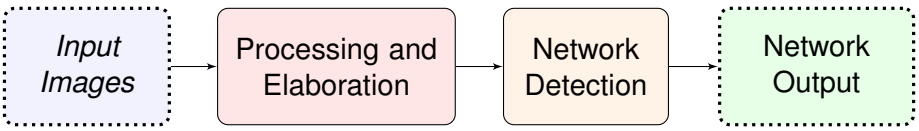


Figure 4 – Rid working principle

The structure of Real Images Detector (RID) can be divided into two parts (see Fig. 4): one dealing with the neural network design and the second focuses on real observations processing and conversion. RID acts as a filter to speed up the tracklet extraction process.
The input required by the script is a directory containing the *.fits* files to be processed.FIrstly, a folder is created within the input one where the processed images will be saved in *.png* format (YOLOv5 compatible).
Once all the images are converted, the detection phase begins by means of artificial intelligence, through the following steps:

- image check: YOLO checks the extension of the input files, if none of them is compatible, an error message will be printed on the screen and the process will stop.
- output folder creation: if a path is indicated in the options, the script generates the desired folder, otherwise it is automatically created with a custom name. It features the *.png* images.
- loading the model: if the computer is equipped with a GPU, the model will be loaded on it via PyTorch, otherwise the CPU will be employed.
- detection: the images are analysed and then the outputs are produced.
- results: a screen shows the detected objects, the inference times for each image, and the overall computational time.

The output generated by the YOLOv5 network is saved in the directory chosen before launching the script (if no directory is specified, the script will generate a folder in the input images folder):

- A copy of all images with bounding boxes printed around the objects, the class name, and confidence value.
- A folder storing all the text files enclosing the characteristics of the bounding boxes.
- A folder containing cropped tracklets images with an equivalent 1 deg FoV (382x382 pixels).

The original detection script *detect.py* automatically crops tracklets minimising the extraction time.

5.1 Training phase

Different versions of YOLOv5 are available, their mAP (Mean Average Precision) on the reference dataset proportionally increases with the model size, but the number of *FPS* (Frames Per Second) analyzed decreases at the same time. The one selected in this paper is medium-sized version of YOLOv5, in short YOLOv5m.
Both real (named Skimage) and synthetic observation-based models are trained with YOLOv5m as a one-class tracklet detector on Colab [12]: a free Jupyter notebook environment custom made for machine learning purposes that runs Python codes in the cloud. As regards real images, clouds and some light sources, taking on a variety of shapes including tapered and rectilinear ones, sometimes can be mistaken for tracklets. The risk in this case is that the network will not be able to distinguish the disturbances as background objects but will attribute them to the tracklet class. To overcome this problem, a new class named "Clouds" has been added to the "Tracklets" one (see Fig. 5).

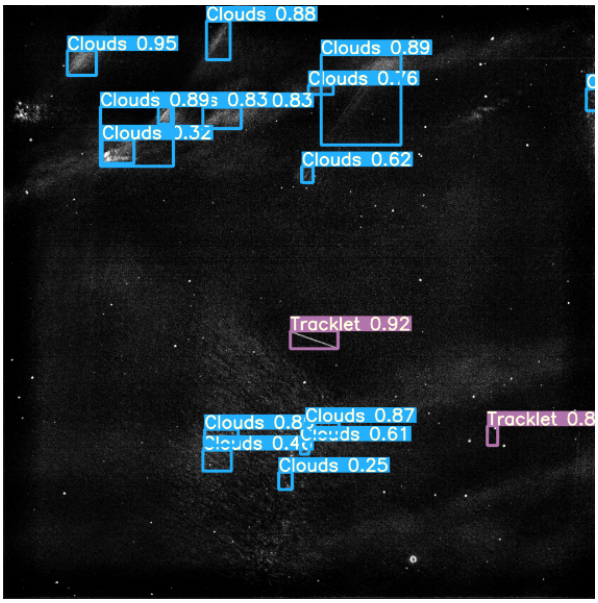


Figure 5 – Two classes network output.

The Synthetic network reaches the desired mean Average Precision (*mAP*) in just a few epochs, but if real images are inputted the performance drops considerably. The *mAP* value is about 1 in case of confidence value greater than 0.5 and 0.85 when the confidence lies between 0.5 and 0.95.
As regards the Skimage network, the *mAP* exceeds the value of 0.8 in case of confidence values greater than 0.5 and 0.6 when the confidence lies between 0.5 and 0.95. In this case, the *mAP* considers the two different classes, in particular, the "Tracklets" class achieves values greater than 0.9 for all the confidence value, while the "Clouds" 0.6 value.

5.2 Testing phase

The testing section processes the remaining 10% of the dataset to calculate the *mAP*, precision, and recall values for all classes.
The synthetic model is tested against synthetic images with the following performance:

Class	Precision	Recall	mAP@.5	mAP@.5 : .95
Tracklets	1	1	0.997	0.883

After the Skimage network training, the "Tracklets" class performs better than the "Clouds" one. The following table shows the output values of the test (in the table the row "All" represents the average of the two classes):
In order to obtain a complete overview, the model trained on real observations is compared with the synthetic one and the ASTRiDE [4] trace detection algorithm. They are assessed with regard to the

Class	Precision	Recall	mAP@.5	mAP@.5 : .95
All	0.782	0.81	0.805	0.596
Tracklet	0.968	0.975	0.988	0.792
Cloud	0.597	0.644	0.622	0.399

quality of their detection. Fig. 6 shows the number of real images correctly detected by the three schemes out of a total of 100 randomly selected.

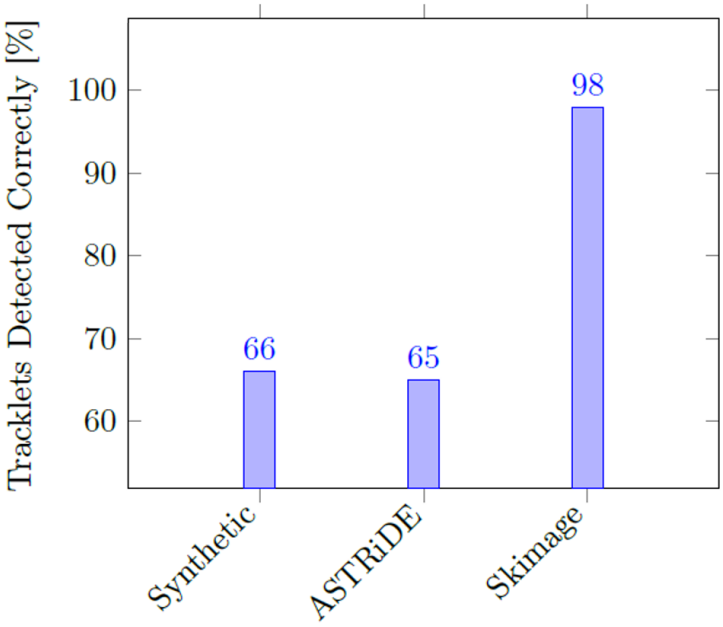


Figure 6 – Networks comparison bar chart.

It seems that the synthetic network and ASTRiDE can only detect sufficiently bright trails in cloudy sky weather conditions, whilst Skimage based network can detect hardly visible streaks too.
Moving on to computational burden, the total processing times can be splitted in three primary contributions:

1. image conversion (from *.fits* file to *.png* image);
2. loading the model on the graphic card via PyTorch (if GPU available);
3. detection.

The image conversion efficiency lies in the programmer's hands who must implement a a relatively fast algorithm and also the most suitable image transformations to reduce possible downtimes. Regarding instead the second and third one, they do not have any room of improvement unless upgrading the computer hardware. The timing calculation is executed both on Colab and on a local computer.

Computer	Images Elaboration	Inference Time	Total Time
PC	$6.5s$	$0.25s$	$6.75s$
Colab	$7.1s$	$0.022s$	$7.122s$

It is quite clear that image processing is the most consuming activity, considering that the model is loaded on the GPU only once and requires about 6/7 seconds. Assuming 1000 images are processed

(the equivalent of about one night of acquisitions): 3.7% of the time is related to the detection, 0.1% for GPU model loading, and the remaining 96.2% for image conversion.

6. Linear Orbit Tracker

The tracker structure can be divided into three parts: the processing of *.fits* observations into *.png* images, the trained network, and the analysis of bounding boxes. In Fig. 7 a basic flowchart that underlines LOT inputs and outputs is represented. First, some directories need to be defined: the folder in which the telescope images are downloaded; the one devoted to the processed images; the output folder, which will contain subfolders divided according to the objects detected. The network is then uploaded to the *GPU* for a faster (almost instantaneous) response. This information combined with the actual sensor images is enough for LOT to identify and organize the detected objects into subfolders. LOT target orbital regimes are Low Earth Orbit (*LEO*) and Medium Earth Orbit (*MEO*). In this context, it is able to track and predict space objects position over time through consecutive observations. The algorithm perceives when the observed object leaves the telescope FoV, updating sensor pointing to maximise the number of observations.

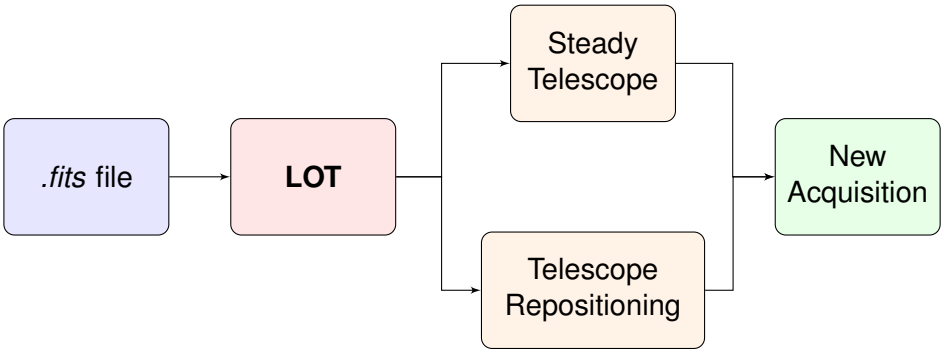


Figure 7 – Schematic representation of LOT I/O.

It is important to emphasise that this pipeline is not capable of reconstructing the tracklet direction in one shot but it only gives an estimate of its position and size in the acquisition. Fig. 8 shows the possible trajectories, calculated through statistical analysis.

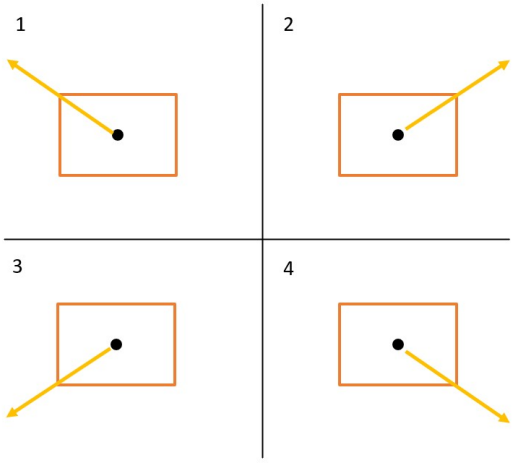


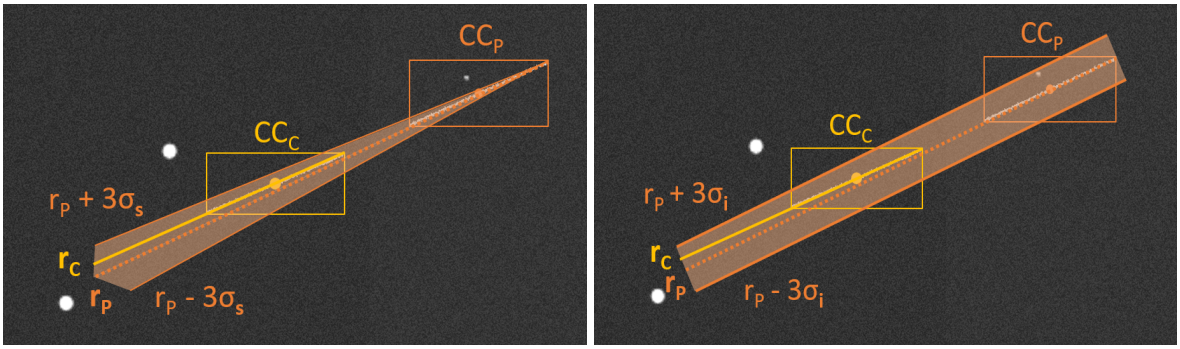
Figure 8 – Possible trajectories from a bounding box. The orange rectangles are the bounding boxes, the black points are the centroids, and the yellow arrows are the trajectories direction.

Once a target is identified, the script estimates four possible straight-line trajectories calculated from the output bounding box and saves the slope and intercept parameters in a text file.
During this process, the telescope continues to take images and when a tracklet is identified in two consecutive images, the script compares the new bounding box with the previous one by analysing

the possible trajectory and position of the centroids. This algorithm grants the possibility of acquiring more images of the same object throughout its passage in the sensor field-of-view.

6.1 Statistical analysis for trajectory prediction

In order to estimate whether two bounding boxes of two successive observations refer to the same object, the candidate trajectories and compared them through statistical analysis (see Fig. 9).



(a) (b)

Figure 9 – **(a)**: shows the slope empirical rule, **(b)**: presents the intercept empirical rule. Orange elements are referred to the previous observation, while yellow ones to the current. "CC_x" elements are related to the centroids, "r_x" to the straight lines, and "σ_x" to standard deviation.

Assuming that bodies move in a rectilinear motion in the images for short arcs, the trajectories are calculated from the vertices of bounding boxes through regression lines. Therefore, the purpose of the analysis for each new observation is to calculate the Root Mean Square (*RMS*) of the slope and intercept, and subsequently, to verify that the new object simultaneously has a slope and intercept that satisfy Empirical rule (also known as 68-95-99.7 rule).

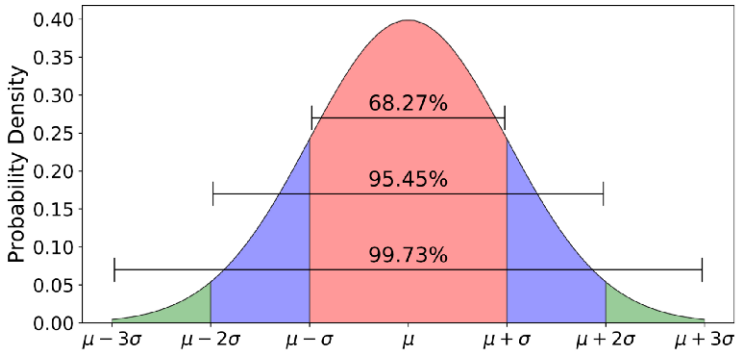


Figure 10 – Empirical rule: 68% of the data is within one standard deviation ($\pm\sigma$), 95% is within two standard deviation ($\pm 2\sigma$), 99.7% is within three standard deviations ($\pm 3\sigma$) [13].

This is a statistical rule which states that for a normal distribution, almost all observed data will fall within three standard deviations (denoted by σ) from the mean value (denoted by μ). In particular, the empirical rule predicts that 68% of observations falls within the first standard deviation ($\mu \pm \sigma$), 95% within the first two standard deviations ($\mu \pm 2\sigma$), and 99.7% within the first three standard deviations ($\mu \pm 3\sigma$) (see Fig. 10).

6.2 LOT structure

Fig. 11 shows the application of LOT with a workflow, beginning with the definition of directories and ending with three cases which will be described separately to help the reader.

6.2.1 Initial part

Once the inputs are defined and the network is loaded on the GPU, LOT is ready to be employed. It consists of an infinite while loop that encapsulates all the tracker processes. It requires a manual

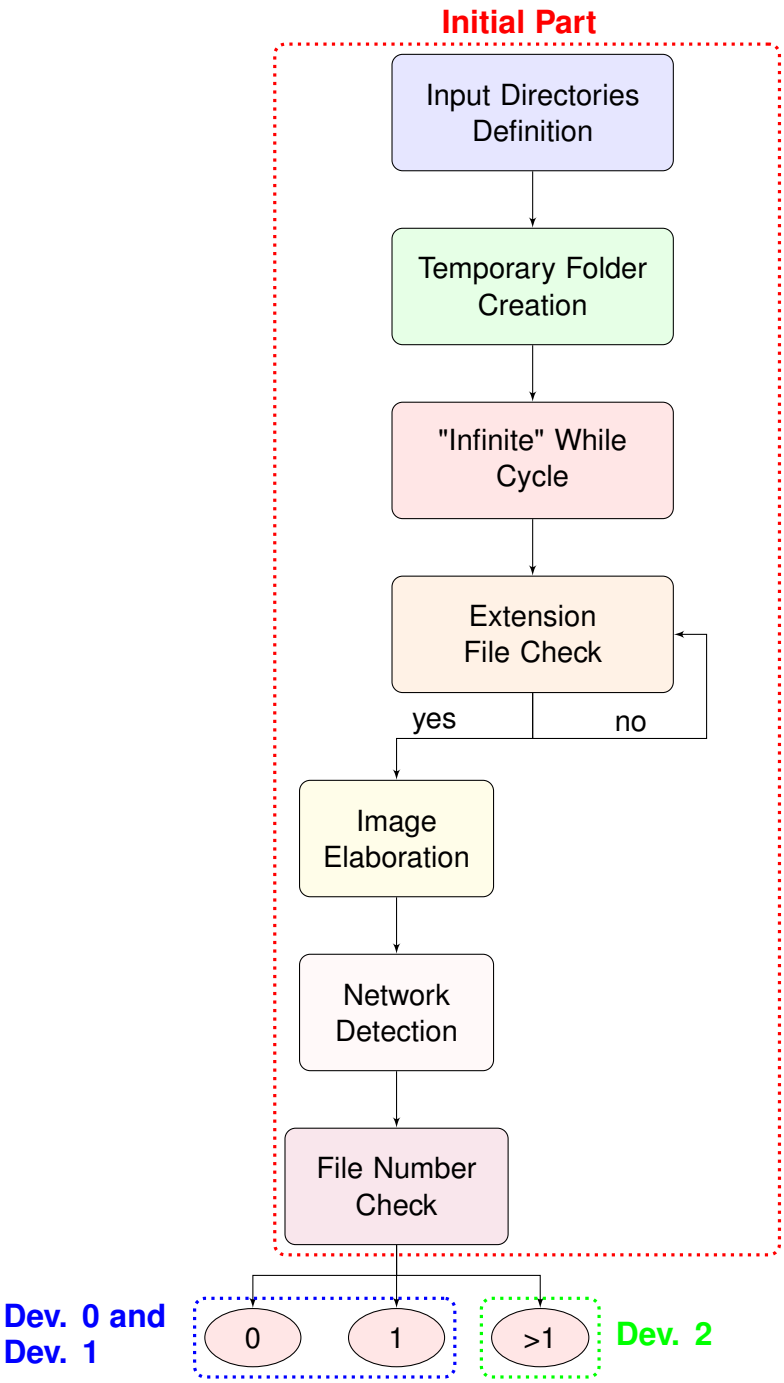


Figure 11 – Initial part of LOT architecture.

stop to allow the algorithm to run through the night without error or failure. In addition, the sleep time must be specified, i.e. after how many seconds the script will restart once it has finished. The time is the machine epsilon which is the minimum possible.
Within this loop, a for loop is added to check if there is a *.fits* file in the input folder (where the telescope observations are downloaded). If nothing is found the algorithm continues to search for a file with *.fits* extension. Conversely, if a .fits image is detected in the folder, it is processed, converted to *.png* format and eventually analysed by the network. Depending on how many objects are detected, the tracker now has three possible developments.

6.2.2 Developments

Dev_0 is triggered in case no target is detected neither in the current detection nor in previous processing. The script creates a folder named after the picture files, and moves them in it so that the auxiliary

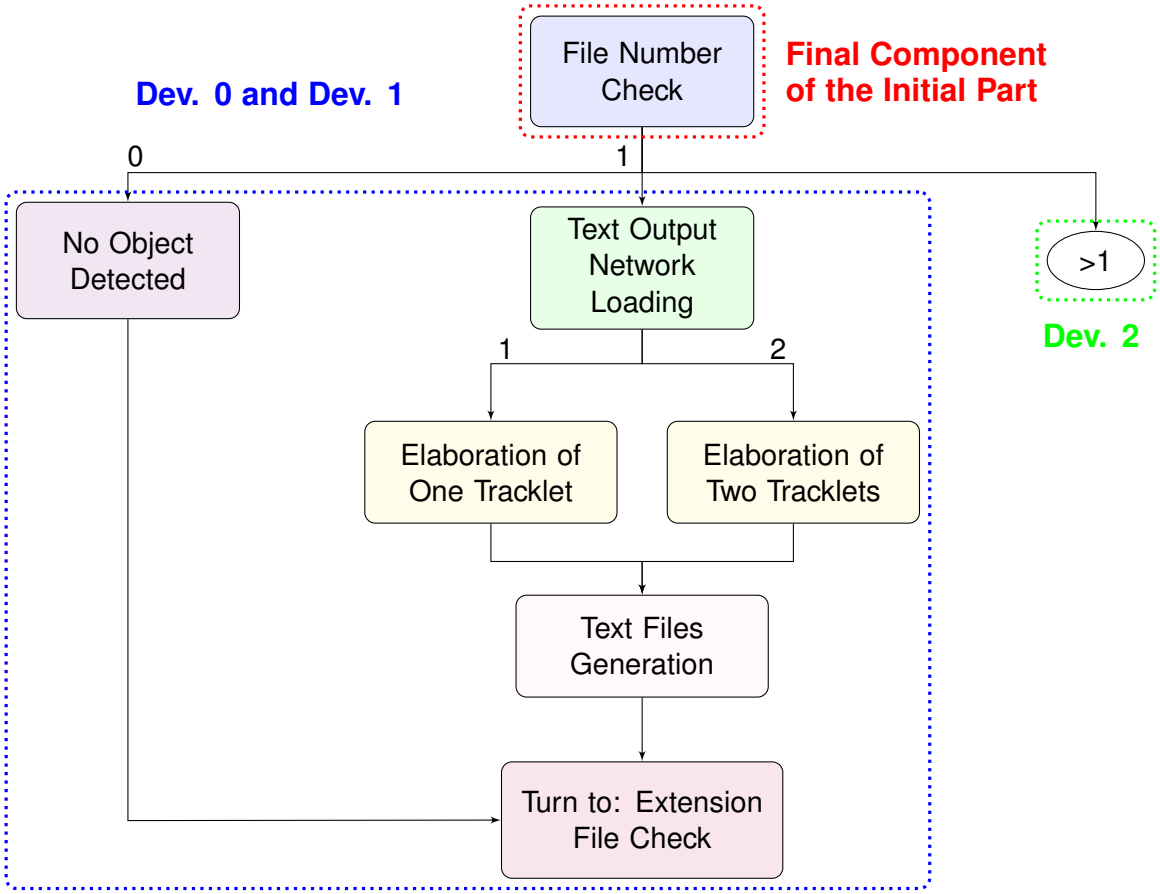


Figure 12 – LOT flowchart for the case with no objects detected, and the case of first object detection.

folder is empty and ready for future observations. After that, the algorithm resumes searching for *.fits* files in the input folder.
Dev_1 activates if the auxiliary text file folder contains one file. This means that a detection occured in the current image and no previous tracklets have been identified. The above mentioned text file is then processed to estimate possible body trajectories, saved in a new text file.
Dev_2 comes into play when the auxiliary text folder contains more than one text file. This means that previous detections have occurred.
In this case, the algorithm checks whether the current detection has identified any targets with two possible outcomes:

- The detection module has not spotted any track, so LOT keeps searching for tracklets in input images. The data from the previous observation remains available for comparison in case a new track is pinponted.
- A detection happened and the corresponding text file is generated. Using this data, LOT calculates the possible rectilinear trajectories of the object and the standard deviations of the lines passing through the opposite vertices of the bounding box. At this point, the tracker has all the information needed to check whether the current track and the previous one correspond to the same orbiting body.

  If they belong to the same object, the script saves the position of the centroids in a new text file and predicts the position of the next centroid. If the future position is outside the FoV, the telescope updates its position. From the third detection of the same body on, LOT calculates the deviation of the prediction from the actual position of the centroid and saves them.

  This procedure is also applied whenever two tracklets are present in the same previous image. If one of them matches the current tracklet, the procedure continues as described above.

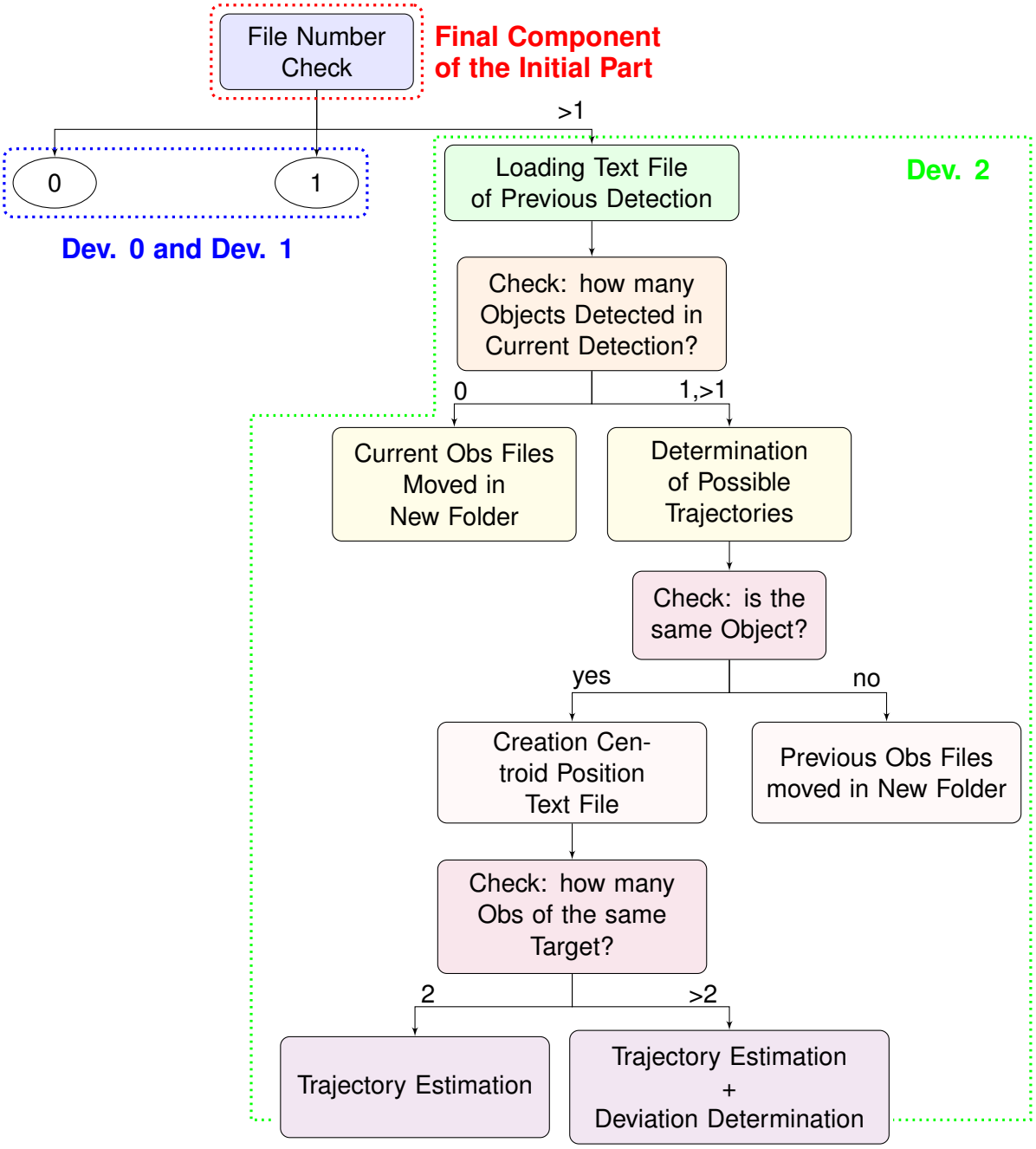


Figure 13 – Flowchart of the LOT main part.

The workflow difference lies in the case in which two bodies are detected in the current observation but only one object has been identified in the prior step. If so, no file is moved because the next observation could contain one or more saved objects. LOT is thus able to track up to two objects in the sensor FoV and chase the one with a higher detection probability.

6.3 Algorithm results

The performance of the algorithm was assessed on synthetic images using TIG [10] software as no adapt real images where available. It is important to specify the assumptions made about the timing of the telescope at PoliMi:

- downloading a shot from the telescope to the computer on which the tracker is mounted takes about 0.5 seconds. This is a fixed time that always elapses between observations;
- re-pointing the sensor after the orbiting body has left the FoV takes about 5 seconds. However, it is a variable interval directly proportional to angular distance. A worst-case scenario analysis

is performed.

The time elapsing between the current acquisition and the next one depends on the position of the object with respect to the sensor: it is equal to 0.5 seconds if LOT expects that the next tracklet will be in the FoV, while it is equal to 7 seconds if it expects that the telescope has to be re-pointed.

6.3.1 Training and testing results

After a 100-epoch network training, the mAP at 0.5 tends to 1 and the 0.5:0.95 mAP is roughly 0.8. The testing procedure is based on the remaining 10% of the dataset. The results are summed up in the following table:

Class	**Precision**	**Recall**	**mAP@.5**	**mAP@.5 : .95**
Tracklet	0.981	0.986	0.991	0.779

In order to have a complete overview of this model performance, it was compared to RID trained networks and ASTRiDE algorithm through the analysis of 100 *.fits* images. As can be seen in Fig. 14, LOT network correctly process about 91% images. LOT identifies more tracks than the synthetic model and ASTRiDE algorithm, but fewer than the Skimage detector network. LOT Detection quality drops due to the fast image processing involved.

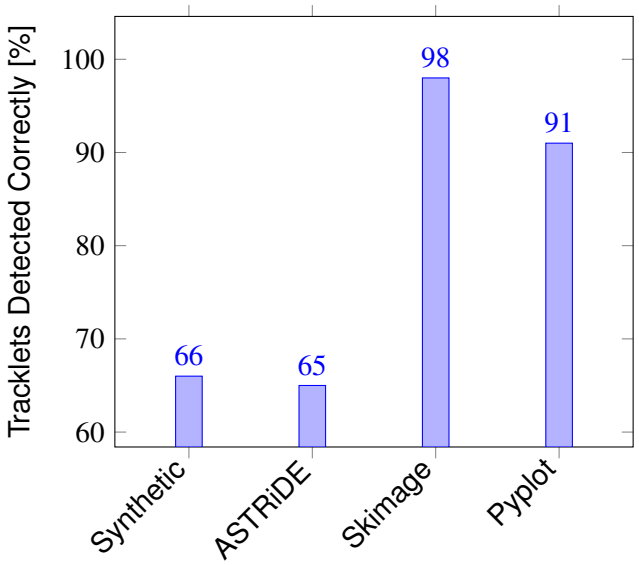
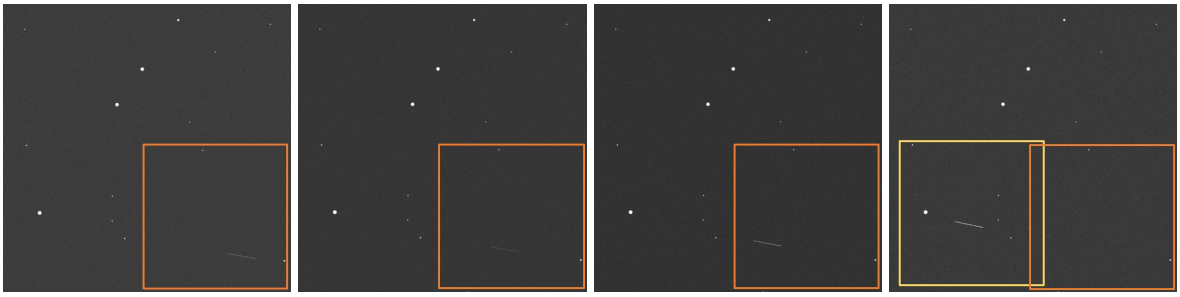


Figure 14 – Networks comparison bar chart.

6.3.2 LEO satellites test

Multiple LEO satellite passages, characterised by an elevation angle ranging from about $5°$ up to about $50°$, are analysed to check LOT performance. Generally, the exposure time to capture this kind of objects is roughly 3 seconds. Sometimes, when the elevation angle is very high, satellites travel about $0.5°$/s and this means that the tracklet is about as long as half an image. The excessive length of the trace could hinder detection if the training dataset does not contain enough samples of this kind. Therefore, the risk is that the estimated position of the object deviates too much from the real position causing the tracker to lose track of the object.
Fig. 15 shows the passage of the *Echo 1 Deb* satellite at an elevation of about $25°$, and the following table shows the LOT-processed values of the deviation of image (c), with respect to the prediction calculated using images (a) and (b), and the deviation after telescope repositioning of image (d), calculated from images (b) and (c).
The values show that the deviations computed using LOT are small enough to allow the tracking of the object.



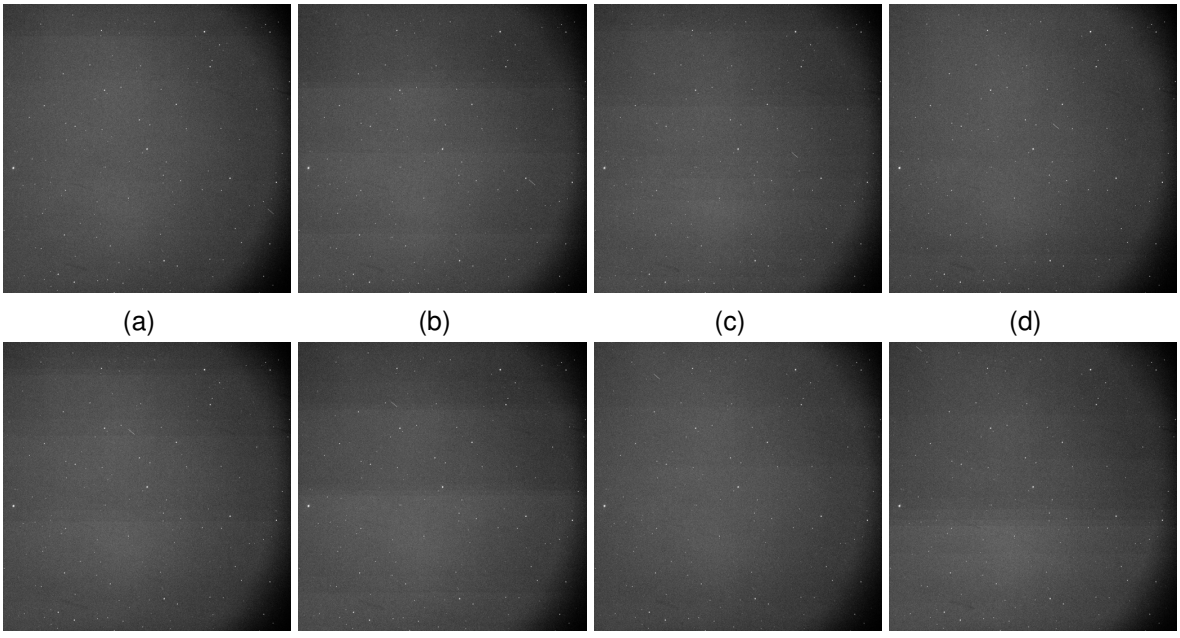
(a) (b) (c) (d)

Figure 15 – Echo 1 Deb passage, the rectangles represent 3° FoV, the orange ones show the first position of the telescope, while the yellow one is after the satellite has left the FoV and thus the position of the sensor has been moved.

Deviation (pixels)	Images	Δt (s)
2.4927	$(a)(b) \rightarrow (c)$	2
1.7381	$(b)(c) \rightarrow (d)$	7

6.3.3 MEO satellites test

The same procedure as the LEO case is followed. Multiple MEO satellite passages and *Galileo FM2* satellites are analysed. The core difference is that these bodies, located in higher altitude orbits, travel at lower angular speeds, so the exposure time is about 30 seconds per image.
The *Galileo FM2* test bench is a series of real observations made by the Pulsar2 telescope at PoliMi. The telescope is used to track the object with an exposure of about 3 seconds and an interval between one acquisition and the next of about 17 seconds. Figure 16 shows the passage tracklets and following table shows the deviation values and the times between one shot and the next. The values never exceed 2 pixels of 512x512 images .



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 16 – Galileo FM2 passage, each synthetic image covers a FoV of 3°.

The algorithm most critical cases are the ones involving vertical tracklets, but even in this instance tracking is performed without any failure.

Deviation (pixels)	Images	Δt (s)
1.885	$(a)(b) \rightarrow (c)$	17
1.527	$(b)(c) \rightarrow (d)$	16.136
0.6897	$(c)(d) \rightarrow (e)$	16.867
0.6798	$(d)(e) \rightarrow (f)$	19.001
0.9443	$(e)(f) \rightarrow (g)$	16.332
0.9118	$(f)(g) \rightarrow (h)$	16.436

6.3.4 Algorithm timing performances

LOT consists of several processes that occur sequentially. They can be divided into two groups:

1. hardware-dependent, e.g. data acquisition speed and the repointing one
2. software-dependent, e.g. image conversion and tracker architecture

By the way, in a real observation campaign few adjustments should be done to improve performance:

- avoid moving the real *.fits* image from the input folder to not be detected again, because usually .fits files weight a few megabytes (Mb).
- use the algorithm directly on the local telescope embedded computing system. In this way, the data transmission rate between detector and computer would improve.

It is quite clear from the following table that the observation processing, detection, and trajectory estimation phases take about 1.5 seconds, and represent about 75% of the time without telescope movement, and about 30% with telescope movement.

LOT + steady telescopes (s)	LOT + telescope respositioning (s)
2	7

Unfortunately, it is impossible to compare the results with a similar tracker due to the current unavailability of a similar one running on the same machine.

6.4 Conclusions and future developments

As the results show (see the following table), RID is able to quickly and accurately analyse real observations. In almost all cases the detection is successful and corresponding processing time is about 0.25 seconds even on a low-level 2017 GPU. The critical point of RID lies in the rather slow processing of the observations run on processors only. Surely this step would improve if the algorithm ran on a recent computer with updated hardware.

Quantity	Performance
Images Detected	98%
Conversion Time	$6.5s$
Inference Time	$0.25s$

A fundamental advantage with respect to conventional methods, is that, once data are collected in the correct way, the algorithm can be re-trained, upgrading the training set with further information, and making the network more and more robust to new cases.

One of the most challenging tasks in object detection is to estimate the direction of the target from a single observation. A future development could start from this problem to find a solution involving both the telescope and the detection software. This could lead to a model that not only identifies and locates an object, but also determines its direction quickly.
Moving on to LOT, it never fails to track the object without a priori knowledge on its orbit. The calculated deviations are orders of magnitude less than the FoV extension. The algorithm also succeeds in vertical cases, for fast satellites, and even in the case of real MEO satellite Galileo FM2 observations.

Quality	**Performance**
Network	91%
Elaboration image	0.9s
Detection	0.1s
Tracker	0.5s

As previously mentioned, the processing phase of the observations takes few seconds allowing the software to track objects with straight trajectories assumption. The conversion process used limits the quality of the image conversion and therefore the number of tracklets detected. It could be improved as for the RID case by simply upgrading computer hardware or by enabling the GPU for these instances.
To have a working and reliable pipeline the tracker should be tested on real-time observation campaigns. Therefore, all the criticalities related to practical implementation may be better defined. A possible future development is to implement LOT with a script that calculates the orbital parameters every time it identifies and tracks a new object. In this way, the orbital estimate can be used to command new pointing angles for the telescope, instead of relying on local linearizations as in the current work. Another feasible evolution of the software consists in extending the tracking procedure to GEO objects and estimate the trajectory through artificial intelligence to increase the efficiency of the process.

7. Contact Author Email Address

Jason Calvi: *jason.calvi@mail.polimi.it*
Alessandro Panico: *alessandro.panico@polimi.it*
Riccardo Cipollone: *riccardo.cipollone@polimi.it*
Andrea De Vittori: *andrea.devittori@polimi.it*
Pierluigi Di Lizia: *pierluigi.dilizia@polimi.it*

8. Copyright Statement

References

[1] E. Kyle. Space launch report: Orbital launch summary by year. (accessed: 01.03.2021). https://www.spacelaunchreport.com/logyear.html.

[2] T.G. Roberts. Popular orbits 101. (accessed: 01.03.2021). https://aerospace.csis.org/aerospace101/popular-orbits-101/.

[3] J. Brownlee. A gentle introduction to computer vision. (accessed: 01.03.2021). https://machinelearningmastery.com/what-is-computer-vision/.

[4] J.M. Juan Zornoza J. Sanz Subirana and M. Hernández-Pajares. Automated streak detection for astronomical images. (accessed: 01.03.2021). https://github.com/dwkim78/ASTRiDE.

[5] Y. Guo J. Ye Z. Zou, Z. Shi. Object detection in 20 years: A survey. *Online Article*, 2019. DOI:https://arxiv.org/pdf/1905.05055.pdf.

[6] G. Jocher. yolov5. (accessed: 01.03.2021). https://github.com/ultralytics/yolov5.

[7] J. Solawetz. Yolov5 new version - improvements and evaluation. (accessed: 01.03.2021). https://blog.roboflow.com/yolov5-improvements-and-evaluation/.

[8] A. Cabras. Design and implementation of an efficient orbital debris detection in astronomical images using Deep Learning. Master's thesis, Università di Pisa, Italy, 2020.

[9] A. Rosebrock. Intersection over union (iou) for object detection. (accessed: 01.03.2021). https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[10] A. De Vittori R. Cipollone. Machine Learning Techniques for Optical and Multibeam Radar Track Reconstruction of LEO Objects. Master's thesis, Politecnico di Milano, School of industrial and information engineering department of aerospace science and technology, Italy, 2020.

[11] scikit-image image processing in python. (accessed: 01.03.2021). https://scikit-image.org/.

[12] Google colaboratory. (accessed: 01.03.2021). https://research.google.com/colaboratory/faq.html.

[13] M. Galarnik. Explaining the 68-95-99.7 rule for a normal distribution. (accessed: 01.03.2021). https://towardsdatascience.com/understanding-the-68-95-99-7-rule-for-a-normal-distribution-b7b7cbf760c2.