AAS PREPRINT 21-648

RESIDENT SPACE OBJECTS DETECTION AND TRACKING BASED ON ARTIFICIAL INTELLIGENCE

Marco Mastrofini[*], Gilberto Goracci[†], Ivan Agostinelli[‡], Mohamed Salim Farissi[§], Fabio Curti[¶]

Resident Space Objects (RSOs) detection and tracking are challenging problems in the framework of Space Situational Awareness (SSA). The growing number of in orbit platforms and the incoming era of mega constellations is increasing both active and passive traffic in the near Earth segment. Recently, more and more researchers and companies have started investigating the problem. This, combined with the growing popularity of Artificial Intelligence (AI) applications, has led to interesting solutions. The present work will investigate an AI based approach for Image Processing, Objects Detection and Tracking oriented towards space optical sensors applications. It will show the architecture development and test of a Convolutional Neural Network (CNN) based algorithm: the image processing and object detection tasks are demanded to Neural Network (NN) modules while the tracking of objects inside the sensor's Field Of View (FOV) is formulated as an optimization problem. Dataset creation for the network training, algorithm design process and results with real images will be shown.

INTRODUCTION

In the framework of SSA and Space Traffic Management (STM), the problem of RSOs detection and tracking to maintain and update ground catalogues has become of greater interest. RSOs are made of active platforms and space debris: the latter constitutes the main important slice of RSOs population and represents a concrete threat to operating spacecrafts. Space debris can have different origins and sizes, varying from quite small elements to very large pieces,[1] they are totally uncontrolled and non-cooperative, with a behaviour and trajectory difficult to evaluate or predict, thereby increasing the risk of collisions and the creation of further debris (Kessler syndrome[2]). The planning of mega-constellations for space-based internet connectivity (SpaceX, OneWeb, Telesat and other companies) implicates the significant increasing number of on-orbit active satellites and potential collisions with fragments across different space regions. This leads the *big sky* assumption[3] (i.e., collision risk is acceptably small with relatively few orbiting objects) to vanish because of the higher probability of close approaches and collisions. For these reasons, the importance of detection and

[*]PhD student, School of Aerospace Engineering, Sapienza University of Rome, Via Salaria 851, 00138 Rome, IT, email: marco.mastrofini@uniroma1.it

[†]PhD student, School of Aerospace Engineering, Sapienza University of Rome, Via Salaria 851, 00138 Rome, IT, email: gilberto.goracci@alumni.uniroma2.eu

[‡]PhD student, School of Aerospace Engineering, Sapienza University of Rome, Via Salaria 851, 00138 Rome, IT, email: ivan.agostinelli@uniroma1.it

[§]Graduate student, School of Aerospace Engineering, Sapienza University of Rome, Via Salaria 851, 00138 Rome, IT, email: mohamedsalim.farissi@uniroma1.it

[¶]Associated Professor, School of Aerospace Engineering, Sapienza University of Rome, Via Salaria 851, 00138 Rome, IT, email: fabio.curti@uniroma1.it

tracking of RSOs has increased more and more in the recent years. In the beginning, this has been performed just through radar and optical ground stations with their consequent limitations: power consumption for radars and weather conditions for optical telescopes. However, the development and design of Space-Based Space Surveillance (SBSS) missions to increase RSOs monitoring coverage has led to the use of on-board optical sensors. Generally, they are telescopes that point specific space regions like GEO belt,[4] but the recent idea of using star trackers to perform Space Surveillance and Tracking (SST) tasks has been under study[5] and ready for the In-Orbit Validation (IOV) phase.[6] Despite the on-board applications, traditional filtering and image processing techniques are still being used[5] for RSOs detection. These techniques have the advantage of a long heritage due to their extensive application. On the contrary, their limitation is the need of complex filtering routines to perform easily human understandable tasks such as objects detection and background removal. For this reason the use of AI appears to be promising for space image data processing and space debris mitigation applications. In particular, the state-of-the-art shows great interest in AI applications to face space problems, to improve flexibility, autonomy and capability to handle failures and performance degradation while satisfying the reliability standards of a safety space system.

Recent works investigate the application of CNNs for RSOs detection. In[7] event based ground optical sensors are considered while in[8] point-like stars and streak-like stars scenarios are analysed with the consequent training of two different CNNs for object detection. A different approach based on semantic segmentation for dim small targets detection is exploited in,[9] achieving satisfying performances at the price of a complex network's training and architecture. The advantages shown by CNNs in previous works could be combined with traditional hardcoded modules into a simple architecture for on board RSOs detection applications through star sensors, adding also tracking capabilities to the detection task. Because of this, the development of a dedicated and reliable AI based software to perform on board RSOs detection and tracking could be crucial for space based SSA activities. A preliminar step in this direction has been taken in the present work.

The description, design, development and test of a AI based module for RSOs detection and tracking through optical sensors are provided. The developed software aims to have as few modules as possible to reduce the algorithm size, complexity and hardware implementation difficulties in order to provide a reliable product.

Contributes of this research are:

- Creation of a dataset of simulated star tracker images for training, validation and test of the You Only Look Once (YOLO) v3 net based detection block.
- Design and training of a CNN based RSOs detection module.
- Design of the Tracking module to collect the history of each RSOs during their motion in the FOV in terms of centroids coordinates.
- Test and Results with real images on several objects passages.

The work is organized as follows: a brief overview of CNN along with the Algorithm structure and modules descriptions and the proposed dataset are provided in the *Methodology* section. In the *Results* section, obtained results from the developed algorithm are presented. Real images have been considered and processed to validate both the developed modules and the capability to perform RSOs detection and tracking over multiple passages. Performance scores and indexes are provided in terms of precision, recall and F1. Future developments are reported in the *Conclusion* section.

METHODOLOGY

In this Section, the developed software functions for RSOs detection and tracking from star tracker raw images are described through a combination of Machine Learning (ML) and classical programming techniques:

- The detection function is achieved by stacking a U-Net[10] and a YOLO v3.[11,12] In this way the detection operation is helped by the image segmentation process performed by U-Net.[13]
- The tracking function is faced through a traditional hardcoded algorithm based on the minimization of a cost function.

The reason why object detection is performed on a mask rather than on the original raw image is due to the possibility of training the U-Net in segmenting accurately the weakest streaks. Indeed, these will result bright enough inside the segmented image to be easily detected by the YOLO v3. For the tracking task, a classical coding approach has been used in order to reduce the computational burden of the whole algorithm while achieving the desired RSOs tracking inside the FOV.

Convolutional Neural Networks

For the object detection module due to the capabilities of CNNs in image data processing and information extraction an AI approach is considered. Actually, lots of CNN models are available to be used such as Vgg-16 and Vgg-19,[14] Inception-v3[15] from Google, the U-Net, YOLO v3 and many others. They can handle many tasks as object detection and classification, semantic segmentation and pose problem involving both cooperative and non cooperative targets. They are made of stacked convolution and maxpooling layers and they can learn high and low level features of an image regardless of their position and orientation. CNNs have also the advantage of having the weights to be learned which are shared between the layers, minimizing the required memory storage.
A convolutional layer performs a convolution operation by sliding a small window (typically of a size of 3×3 pixels) over height and width of the image and over its channels (one or three depending on grayscale or RGB nature of the image respectively). The convolution operation produces a *patch feature map*, which is multiplied by a matrix of learned weights (*convolution kernel*). This operation leads to an image size reduction, which is handled by *padding*, namely the addition of border pixels in order to make the output size the same as the input. The network layers elements are activated through an *activation function*: *ReLU* (Rectified Linear Unit), *softmax* and *sigmoid* functions are typically used.
The pooling layer downsamples the image in order to optimize feature learning, taking a portion of the image (typically a 2×2 window) and substituting it with its maximum pixel value.
The performance of a CNN is given by its *loss* and *accuracy* parameters, computed during the training, validation and testing of the net. The aim of the process is to obtain similar accuracies in all of the three phases, to avoid *overfitting* and to generalize the network performance against never seen data. There are different techniques to overcome overfitting, typically L2 weight regularization,[16] batch normalization,[17] data augmentation[18] and dropout.[19]

Algorithm Design and Configuration

The algorithm structure is shown in Figure 1 with the main blocks. The incoming raw image from the star tracker arrives to the *Image Segmentation Module* where it is binarized. The output of this module is called *Mask* and it is a map of foreground and background pixels. The mask is

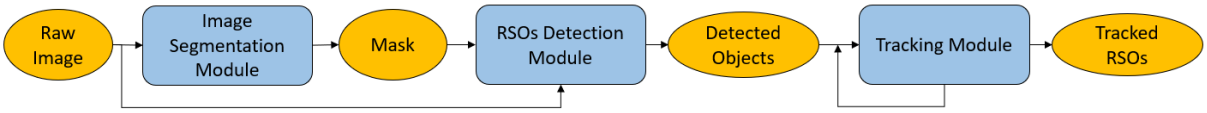

Figure 1. RSOs Tracking module algorithm structure.

used by the *RSOs Detection Module* to recognize the streaks in the night sky image and localize them into rectangular boxes. The convolution between the box-delimited regions in the *Mask* and the same sky portions in the *Raw image* provides rectangular areas in which every foreground pixel shows its original energy value. In this way, the computation of the streaks *centroids* and *extremal points* (see next subsections) is possible and, knowing the *exposure time*, the horizontal and vertical components of the streak velocity can be computed. At this point, the computed information relative to the detected objects is used by the *Tracking Module* to distinguish the objects inside the FOV and provide their time evolution in terms of image coordinates as the output of the algorithm.

In the framework of this algorithm, RSOs are considered as streaks over a background made of point-like stars under the implicit assumption of low angular velocity of the orbiting platform. This assumption is reasonable since most of the missions involving star trackers require a sidereal pointing.

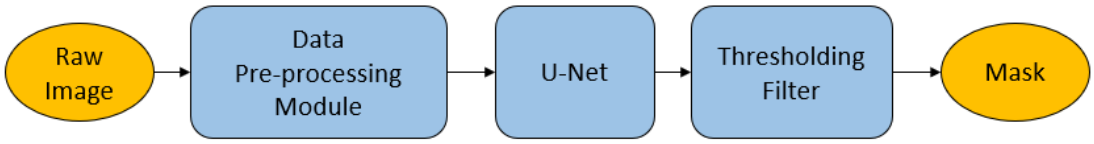

Figure 2. Image Segmentation Module algorithm structure.

Image Segmentation Module This module uses a U-Net to segment the image[13] and removes a large part of the noise background, leaving the brightest objects in the FOV. The *Raw image* is segmented through the U-Net and a suitable binarization level p_{min} in the *Thresholding filter* is applied to get the *Mask*. The module structure is reported in Figure 2. The U-Net used in the present work is reported in Figure 3. It has an initial number of filters equal to 128 and is applied on rectangular images of 960 $\times$ 640 size. The optimal value for p_{min} has been empirically found to be 0.45. It differs from the optimal value in[13] because the U-Net has been trained on a new dataset.

RSOs Detection Module In Figure 7 the RSOs Detection module is shown. The *Mask* is processed by a YOLO v3 model, classifying the streaks and localizing them through rectangular boxes. For every detected streak, the box top left pixel coordinates ($\mathbf{P}_{box}$), width and height and class probability are known. This set of information is then used to extract the relative rectangular area both on the *Mask* and the *Raw Image*. Through the convolution of these two arrays, the *Clustering Module* computes the streak centroid with a weighted energy based average (Equation (1)).

$$\mathbf{P}_{cluster,i} = \mathbf{P}_{box} + \sum_{j} \left(\frac{E_{px,j} \cdot \mathbf{p}_{px,j}}{E_{cluster,i}} \right) \tag{1}$$

with

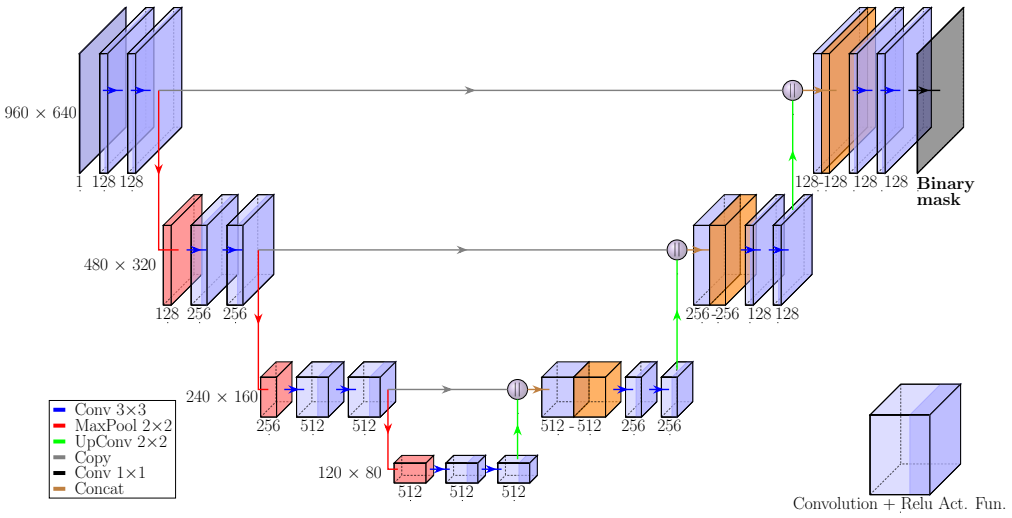

Figure 3. U-Net model with features dimensions and number of features.[20]

$$E_{cluster,i} = \sum_j E_{px,j} \tag{2}$$

Where $\mathbf{P}_{cluster,i}$ are the *absolute* coordinates of the i^{th} cluster, $\mathbf{p}_{px,j}$ are the *local* coordinates of the j^{th} pixel in the box and $E_{cluster,i}$ is the sum of the energies $E_{px,j}$ of the i^{th} cluster's pixels.

Absolute coordinates are relative to the *image reference frame* and *local coordinates* to the *box reference frame* (Figure 4). The *Clustering Module* computes also the *Extremal Points* of the streaks, namely the coordinates of the upper and lower tips of the objects, as the points in which the horizontal coordinate takes its minimum and maximum value. If the streak goes from the lower-left corner of the box to the upper-right corner the minimum value corresponds to the lower tip and the maximum to the upper tip, viceversa if the streak goes from the upper-left corner of the box to the lower-right one.

Since more than one foreground pixel may have the same value of horizontal coordinate (x), the vertical coordinate (y) of the extremal points is computed as the mean of the y values of all the points sharing the same horizontal coordinate (Figure 5). Considering the absolute value of the extremal points coordinate differences, it is possible to divide them by the image *Exposure Time* and to obtain an estimate of the RSO velocity in the image plane. This velocity is used to predict the position of the centroid in the next image taking into account both the *exposure time* (T_{exp}) and the time between two subsequent camera acquisitions, hereafter referred to as *pause time* (T_p), see Figure 6.

RSOs Tracking Module The *Tracking Module* algorithm associates every detected object to one that has been previously classified or to a new one. The aim of the algorithm is to distinguish RSOs as they cross the FOV. Hereafter, the number of objects stored in memory will be referred to as n_{obj} while the number of detected objects (boxes) in the current analyzed image will be indicated with n_{box}.

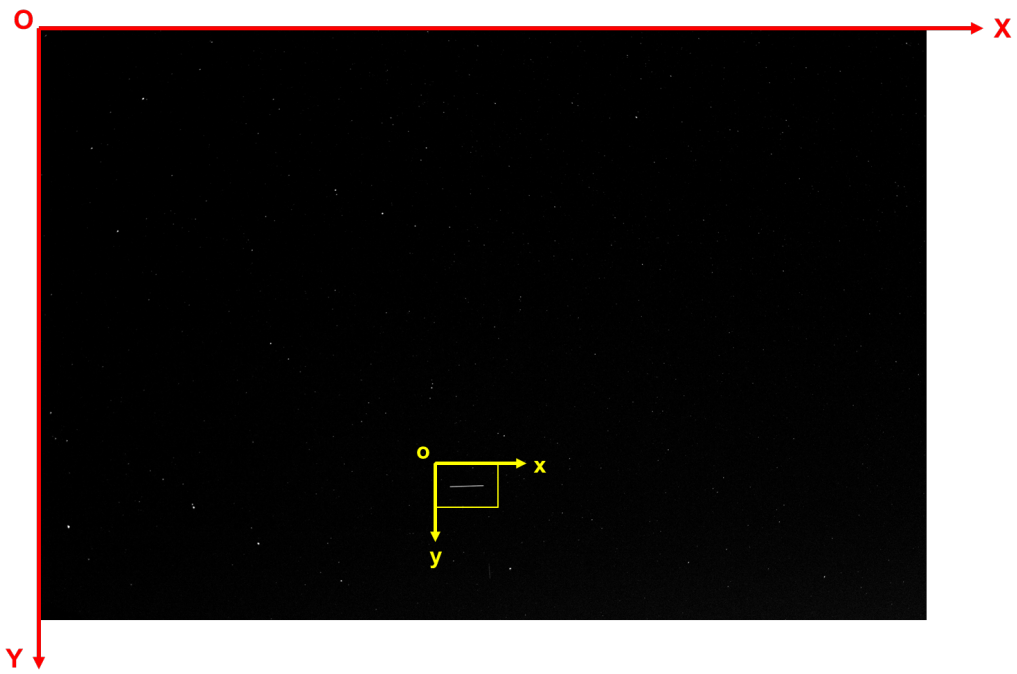

Figure 4. Absolute (red) and local (yellow) reference frame.

The tracking module is based on the minimization of a *Cost Function* J which is computed for each box-object pair. At each image the *Cost Function* takes into account the position and orientation of the box and compares them to the corresponding expected values determined from the previous image. The *Cost Function* for the i-th box with respect to the j-th stored object is given by:

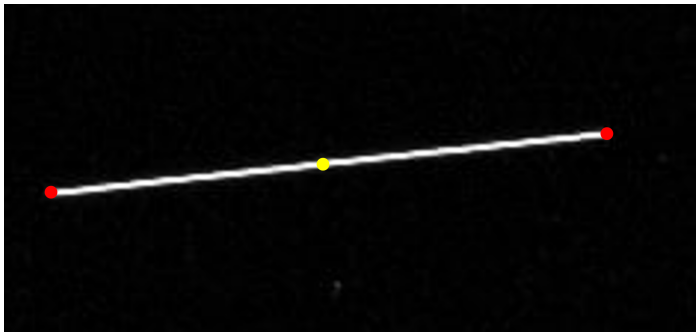

Figure 5. Extremal points (red) and cluster centroid (yellow).

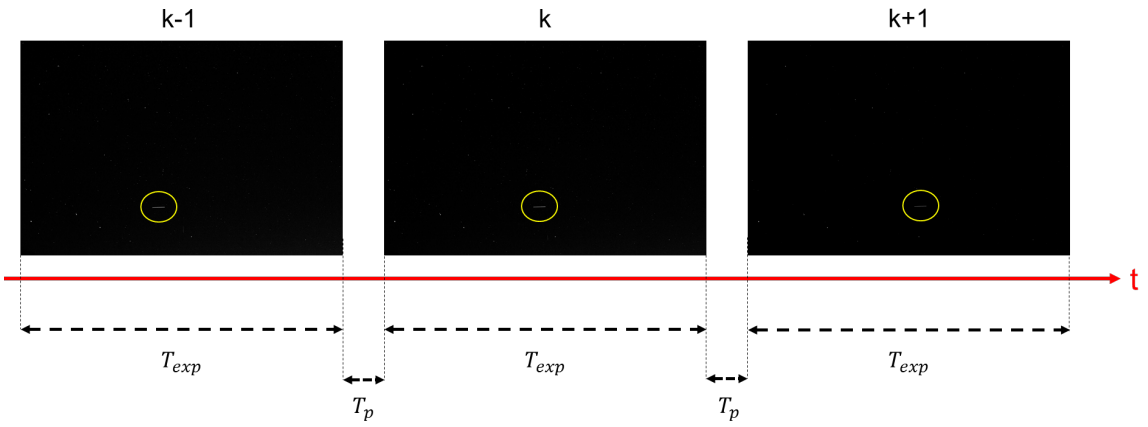

Figure 6. Acquisition Philosophy.

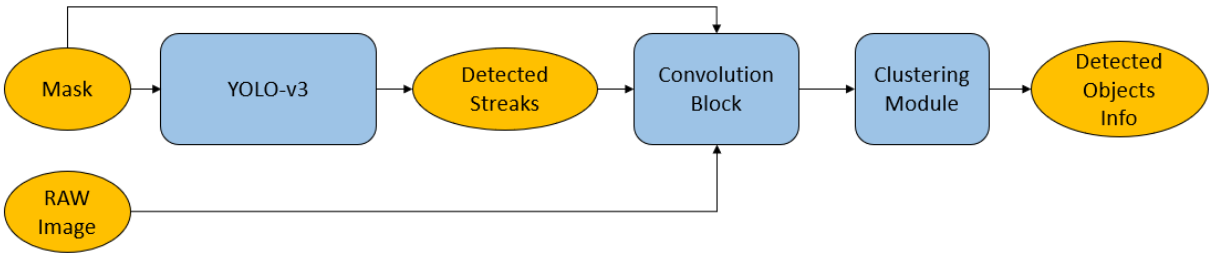

Figure 7. RSOs Detection Module algorithm structure.

$$
\begin{aligned}
J_{ij} = w_1 \left| |({x_i}^{new} - {x_j}^{latest})| - {v_{xj}}^{latest} T \cdot n_{skip} \right| + \\
+ w_2 \left| |({y_i}^{new} - {y_j}^{latest})| - {v_{yj}}^{latest} T \cdot n_{skip} \right| + \\
+ w_3 \left| ({m_i}^{new} - {m_j}^{latest}) \right|
\end{aligned}
\tag{3}
$$

where:

- x and y are the centroid coordinates;
- v_{xj} and v_{yj} are the horizontal and vertical components of the velocity of the j-th object, respectively;
- T is the time passed between two consecutive images, namely the sum of the exposure time T_{exp} and the pause time T_p (Figure 6);
- n_{skip} is the difference between the indices of the two images, it takes into account the fact that one object can be missed by the detection module for one or more images;
- m is the inclination angle of the object;

- w_1, w_2 and w_3 are the weights of the three different contributions to the total *Cost Function*. In this work they all have been set to 1.

The superscript *new* refers to the information contained in the new incoming image, while the superscript *latest* contains the latest available information related to the object.

The algorithm is divided according to four possible scenarios. Figure 8 shows an example of a real passage of two objects in the night sky in which each image represents a different classification scenario.

- $n_{obj} = 0$ and $n_{box} > 0$

  This scenario happens only for the first image containing one or more objects. Since no objects is stored in memory, the algorithm classifies every detected streak as a new object with its information on centroid position, velocity and orientation. No computation of the *Cost Function* is performed in this case (Figure 8a).

- $n_{obj} > 0$ and $n_{box} > n_{obj}$

  The number of streaks in the image is greater than the number of objects stored in memory, this means that one or more objects have to be created. In order to do this each old object is updated in the following way:

  1. The *Cost Function* J is computed and it will have more rows than columns ($n_{box} > n_{obj}$);
  2. The minimum values on each column determine the box corresponding to the object relative to that column;
  3. The boxes corresponding to the rows not associated with any column are classified as new objects.

  As an example, supposing $n_{box} = 2$ and $n_{obj} = 1$, the J matrix will look like this:

  $$J = \begin{pmatrix} 315.83 \\ 11.84 \end{pmatrix}, \tag{4}$$

  therefore, the stored object will be updated with the data from the second box in the image while the other box will be classified as a new object and stored in memory (Figure 8b).

- $n_{obj} > 0$ and $n_{box} = n_{obj}$

  The number of streaks in the image is equal to the number of objects stored in memory. The *Cost Function* is used to correctly update each object: in this case J is a squared $n_{box} \times n_{obj}$ matrix and the minimum values on each row determine which box corresponds to the object relative to that row (since J is a squared matrix the minimum can be found scanning column by column as well). As an example, supposing $n_{box} = n_{obj} = 2$, the J matrix will look like this:

  $$J = \begin{pmatrix} 286.53 & 3.74 \\ 4.78 & 370.73 \end{pmatrix}, \tag{5}$$

  therefore, object number 1 will be updated with the data from the second box in the image while object number 2 will be updated with the data from the first box (Figure 8c).

An exception occurs whenever $n_{box} = n_{obj}$ but one or more objects do not correspond to the stored ones (i.e.: one of the stored objects is not detected and at the same time a new object appears in the FOV and it is detected). The algorithm deals with this case by applying a threshold to the minimum value of the *Loss Function*: if the minimum value across a row is greater than a threshold value $\tilde{J}$ the corresponding box is associated to a new object, the relative information is stored in memory and n_{obj} is increased by one. At present time $\tilde{J}$ has been set to 100 but it can be changed according to the chosen weights w_1, w_2 and w_3.

- $n_{obj} > 0$ and $n_{box} < n_{obj}$

  The number of streaks in the image is lower than the number of objects stored in memory, this means that the algorithm failed to detect one or more objects or simply that one or more objects left the FOV. To determine which objects are missing the algorithm proceeds as it follows:

  1. The *Cost Function* J will be computed and it will have more columns than rows ($n_{box} < n_{obj}$);
  2. The minimum values on each row determine the object corresponding to the box relative to that row;
  3. The objects corresponding to the columns not associated with any row are classified as missing objects and therefore will not be updated.

  As an example, supposing $n_{box} = 1$ and $n_{obj} = 2$, the J matrix will look like this:

$$J = \begin{pmatrix} 266.10 & 5.07 \end{pmatrix}, \tag{6}$$

  therefore, the data from the unique box in the image will be used to update object number 2, while object number 1 will not be updated (Figure 8d).

Dataset Description

To train and validate the U-Net, a dataset of 100 real images taken in a night sky acquisition campaign at Campo Imperatore, Gran Sasso, Italy, has been used. The same set of images has been adopted as the baseline for the creation of 5000 segmented images for the training and validation of the YOLO v3 NN in the *Detection Module*. Each image consists of stars and streaks on a dark background and it is associated to a text file containing the information about the streaks in YOLO v3 format. To guarantee a proper training of the YOLO v3 the dataset has to be as uniform and complete as possible in terms of streak length, orientation, position and number of streaks per image. The data-augmentation process involved the creation of 100 background images removing the objects from the original set. Then, one of these backgrounds is randomly chosen and a random number of streaks is printed on the image, with length, orientation and position randomly chosen from uniform distributions (between 0.1% and 10% of the image diagonal length and between -90° and 90° for length and orientation, respectively). The printing process starts by selecting a top-left coordinate and a length for the object from which a rectangular box is created. The top-left and bottom-right points of the box are then united with a one pixel-thick or two pixels-thick white line according to the following probability split for a better resemblance to the real cases:

- 60% two pixels / 40% one pixel if the streak length is inferior to the 1% of the image diagonal length;

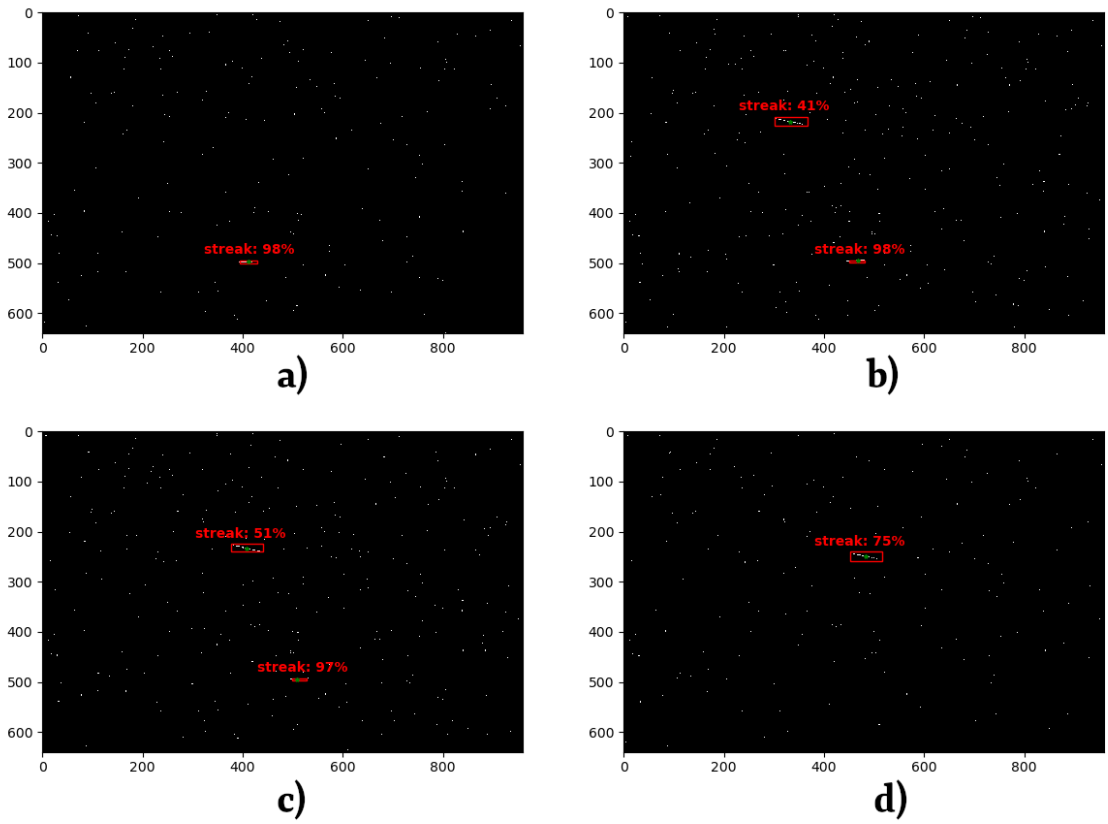

Figure 8. A real passage of two objects in the night sky. Each panel corresponds to a different classification scenario described in the *RSOs Tracking Module* subsection: a) case 1; b) case 2; c) case 3; d) case 4.

- 20% two pixels / 80% one pixel if the streak length is between 1% and 3% of the image diagonal length;
- 5% two pixels / 95% one pixel if the streak length is greater than the 3% of the image diagonal length.

Finally, the text file containing the information on the totality of objects in the image is compiled in YOLO v3 format. The box from which the information is collected is 0.9 times the size of the one used to draw the streak. The effects of the data-augmentation on the orientation, length and number of the streaks per image are shown in Figures 9, 10 and 11.

RESULTS

Network Training

This section describes the U-Net training steps. In particular, the network is configured with the values shown in Table 1.

U-Net training and validation have been performed in the Tensorflow Keras[21] framework considering 3 *epochs* and a dimension of the training batch equal to 3. This has been done for 128

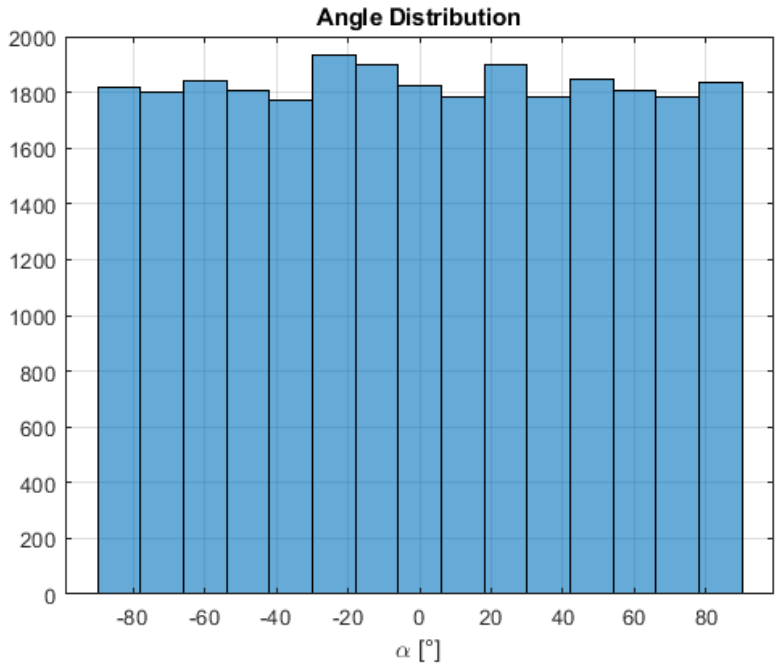

Figure 9. Streaks' angle distribution of the Dataset.

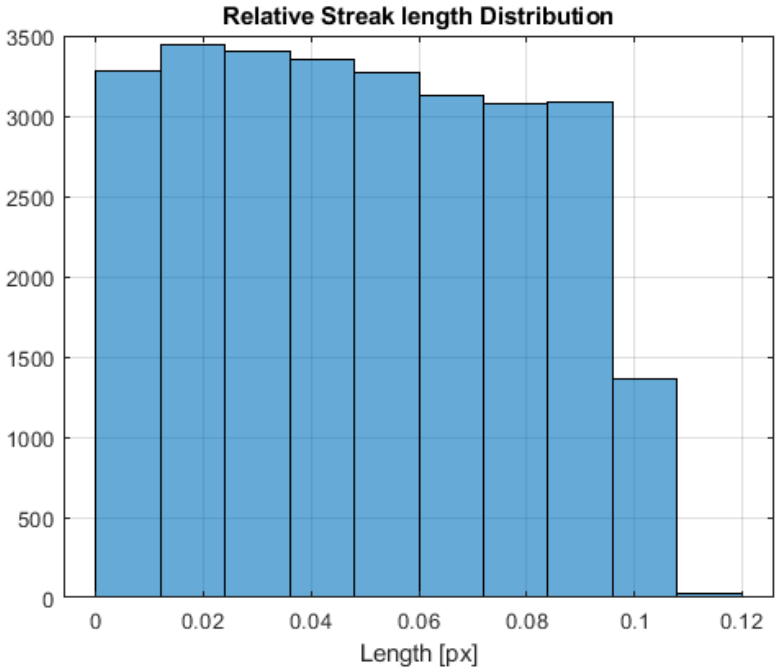

Figure 10. Streaks' relative length distribution of the Dataset.

filters. The trained network shows a reached accuracy higher than 99% and a loss lower than 0.01. Moreover, no overfitting showed up.

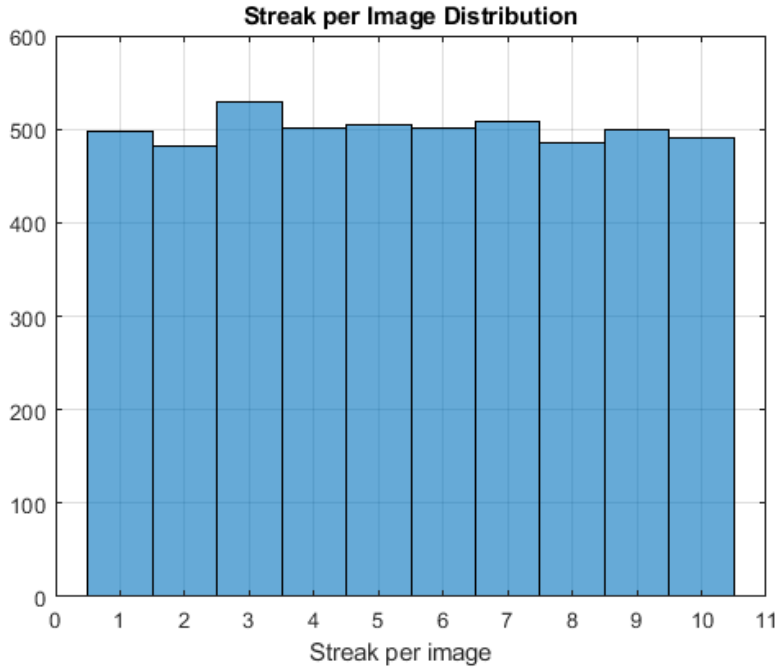

Figure 11. Streaks per image distribution of the Dataset.

Table 1. U-Net Configuration Parameters

Parameter	Value
Image size	960×640
Initial filters	128
Learning Rate	10^{-4}
Regularization Factor	10^{-5}
Dropout Rate	0.25
Kernel Size	3
Kernel initializer	$'he\_normal'$

YOLO v3 training and validation have been performed in the Darknet framework considering the previous dataset. The trained network shows a mean Average Precision (mAP) equal to 78.41% considering a IOU of 50%.

The training, validation and test have been performed on a workstation with the following specifications:

- CPU: AMD Ryzen Threadripper PRO 3975WX 32 Cores 3.50 GHz
- RAM: 128 GB
- GPU: Nvidia Quadro RTX 5000.

Test on a Real Passage

To test the *Detection* and *Tracking* modules a sequence of real images is considered . The whole passage is shown in Figure 12, where two objects are moving inside the FOV from top left region

Figure 12. ISS and Starlink Passage used for the test.

towards the bottom of the image: the ISS (brightest) and a Starlink. The sequence has been acquired with a reflex camera with the same philosophy described in Figure 6 with $T_{exp} = 5$ s and $T_p = 2$ s and then processed. The Object Detection and Tracking algorithm have been able to detect both the objects but not in every image as shown in Table 2 where centroids and streaks orientation of the detected objects are shown for every frame. For this test the weights of the *Cost Function* were set to 1.

Only the ISS has been clearly detected in almost every frame. The reason why the ISS is not detected in the first frame is to the sun-off/sun-on transition while the streak is too small and close to the image edge in the last two frames. The proximity to image border causes the YOLO v3 not to detect the objects. This aspect is being investigated. The Starlink indeed appears too weak to be detected in almost every frame and this is due to the segmentation performed by U-Net. This issue is being investigated as well and a possible solution could be increasing the raw image contrast with a suitable module before being processed by the U-Net.

Performance Indices Evaluation over Real Passages

A total of 31 real passages was collected during the Campo Imperatore Acquisition Campaign performed in the summer 2021. The algorithm has been tested on each image of every passages and an evaluation of Precision, Recall and F1 index is provided over the whole set. To introduce them, the definitions of True Positive (TP), False Positive (FP) and False Negative (FN) must be given:

- TP: A counter that increases of one unit every time a streak on the image is detected by the network.
- FP: A counter that increases of one unit every time the network detects a wrong object.

Table 2. Tracking Table

Frame	$x_{Obj1}\,[px]$	$y_{Obj1}\,[px]$	$\alpha_{Obj1}\,[deg]$	$x_{Obj2}\,[px]$	$y_{Obj2}\,[px]$	$\alpha_{Obj2}\,[deg]$
1	–	–	–	–	–	–
2	291.26	187.60	67.38	–	–	–
3	–	–	–	–	–	–
4	316.04	281.70	68.20	–	–	–
5	327.25	324.71	66.57	85.60	256.46	57.99
6	337.46	363.92	69.23	–	–	–
7	346.57	399.18	66.50	–	–	–
8	354.44	429.78	68.63	–	–	–
9	362.37	461.05	66.80	–	–	–
10	369.05	487.48	67.17	–	–	–
11	375.27	512.51	67.62	–	–	–
12	381.61	537.91	70.02	–	–	–
13	386.78	559.23	67.62	–	–	–
14	392.18	581.07	66.37	–	–	–
15	397.00	601.01	66.80	–	–	–
16	401.53	620.09	72.47	–	–	–
17	–	–	–	–	–	–
18	–	–	–	–	–	–

- FN: A counter that increases of one unit every time a streak on the image is not detected.

These quantities are then used to evaluate three performance indices defined as:

$$Precision = \frac{TP}{TP + FP} \times 100 \tag{7}$$

$$Recall = \frac{TP}{TP + FN} \times 100 \tag{8}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{9}$$

By module test over the passages collection the following numbers are obtained:

Table 3. Performance Indices

TP	FP	FN	Precision [%]	Recall [%]	F1 [%]
189	11	44	94.5	81.11	87.29

Table 3 demonstrates that the module shows good compatibility with real images. Night sky images acquired via ground-based equipment have been used for a preliminary evaluation of the present algorithm, but similar results are expected from tests on on-board acquired images. Nonetheless, the percentage of FN with respect to TP is not negligible (about 25 %). By visual inspection of real passages where the most of FN occurs, fragmented streaks are still difficult to be detected by the network. The presence of a not negligible number of FN is the cause of the mAP equal to 78.41%. This suggests to explore strategies to increase the detection capability against this kind of streaks. Results of these strategies are shown in the next subsection.

By a computational resources point of view Processing Time and Dynamic Memory have been measured: the former has shown a value lower than 3 seconds for every image processing. This Processing Time can be accelerated with a dedicated System on Chip (SoC) implementation that is under investigation. Dynamic Memory measurements have shown a RAM value lower than 60 MB for every image processing.

Increasing Performance Strategies

To increase the capability of the network for the detection of fragmented streaks, a new dataset of 5000 images has been generated at the same way of the previous described one, hereafter referred to as *Fragmented 5000*. This dataset differs from the *original* one just for a 25 % of probability in fragmented streaks generation. This percentage has been chosen because it is about the same percentage of FN with respect to TP. This value is also comparable with the FN to TP ratio of the mAP test. The previous trained YOLO v3 model has been trained again on *Fragmented 5000* and it has shown a mAP of 79.82% when tested on the *original* dataset. The increasing of mAP is related to a reduction of FN number and increment of TP as shown in Table 4. To increase again the performance, another dataset with 30000 images (*Fragmented 30000*) has been created in the same way as *Fragmented 5000*. Starting from the weights of the previous model, another Transfer Learning (TL) process has been performed by training again the net on this new dataset. The mAP of this last configuration on the *original* dataset is equal to 82.68% due to a further and significant

reduction of the FN percentage and the increasing of TP. Results of the mAP computation tests are shown in Table 4 for the original net and the two TL processes.

Why not directly performing a training from scratch on *Fragmented 30000*? It has been done and the network has been tested on the *original* dataset showing a mAP lower than the 82.68% achieved with a double TL. Results are reported in Table 4. This confirms that the double TL approach is the most suitable strategy to reduce the FN and percentage among the ones that have been tested in this work. Moreover, this approach has led to a significant reduction of the number of FP.

Table 4. Models' Performance comparison

Model	mAP [%]	TP [%]	FP [%]	FN [%]
Original	78.41	72.4	8.5	19.1
First TL	79.82	74.4	8.5	17.1
Second TL	82.68	77.3	7.7	15.0
Scratch 30 k	77.96	71.9	8.3	19.8

The performance improvement with the double TL approach can be appreciated by applying the YOLO v3 with the improved capabilities on the previously tested real passages. In Table (5) the new objects centroids and orientation are shown and can be compared to the ones detected with the original training (see Table 2). It must be pointed out that the same detection threshold of 35% has been used in each test.

The improvement in results is due to the successful detection of object 1 and object 2 in *Frame* 3 and *Frame* 4, respectively. In *Frames* 1, 17 and 18 no streaks occur in the U-Net output and thus no streak is detected.

The same network has been tested on the overall 31 passages (Table 6).

By comparing Tables 3 and 6, a reduction of FP and FN is clearly visible as well as an improvement of TP. This increases the F1 index from 87.29 to 88.94, confirming a performance improvement.

Test with High-Fidelity Star Tracker Simulator Images

A sequence of 20 images from a High-Fidelity Star Tracker Simulator has been tested with the present algorithm (Figure 13). Five objects have been initialized into the FOV. This simulator has been developed and validated within the Automation Robotics & Control for Aerospace Laboratory (ARCALab) at the School of Aerospace Engineering, Sapienza University of Rome and has been used for the design of several previous SSA missions. Algorithm application with Star Tracker-simulated images using the best performance network shows the tracking of six objects in total (Table 7):

- Objects 2 and 3 appear in most of all the images due to their relative high distance from the observing platform.
- Objects 1 and 4 appear in a few images.
- Objects 5 and 6 appear in one image only.

Table 5. Tracking Table with the improved weights

Frame	$x_{Obj1}\,[px]$	$y_{Obj1}\,[px]$	$\alpha_{Obj1}\,[deg]$	$x_{Obj2}\,[px]$	$y_{Obj2}\,[px]$	$\alpha_{Obj2}\,[deg]$
1	–	–	–	–	–	–
2	291.27	187.61	65.46	–	–	–
3	303.94	235.66	68.96	–	–	–
4	316.04	281.70	66.80	61.91	215.12	57.99
5	327.34	325.06	67.01	83.71	253.10	52.43
6	337.46	363.92	64.29	–	–	–
7	346.38	398.45	65.56	–	–	–
8	354.54	430.25	67.38	–	–	–
9	362.37	461.05	65.56	–	–	–
10	369.12	487.77	66.04	–	–	–
11	375.38	512.84	64.80	–	–	–
12	381.61	537.91	66.04	–	–	–
13	386.76	558.83	63.43	–	–	–
14	391.93	580.02	64.98	–	–	–
15	396.81	600.47	70.35	–	–	–
16	401.54	620.10	75.07	–	–	–
17	–	–	–	–	–	–
18	–	–	–	–	–	–

Table 6. Performance Indices with the new strategy

TP	FP	FN	Precision [%]	Recall [%]	F1 [%]
201	13	37	93.93	84.45	88.94

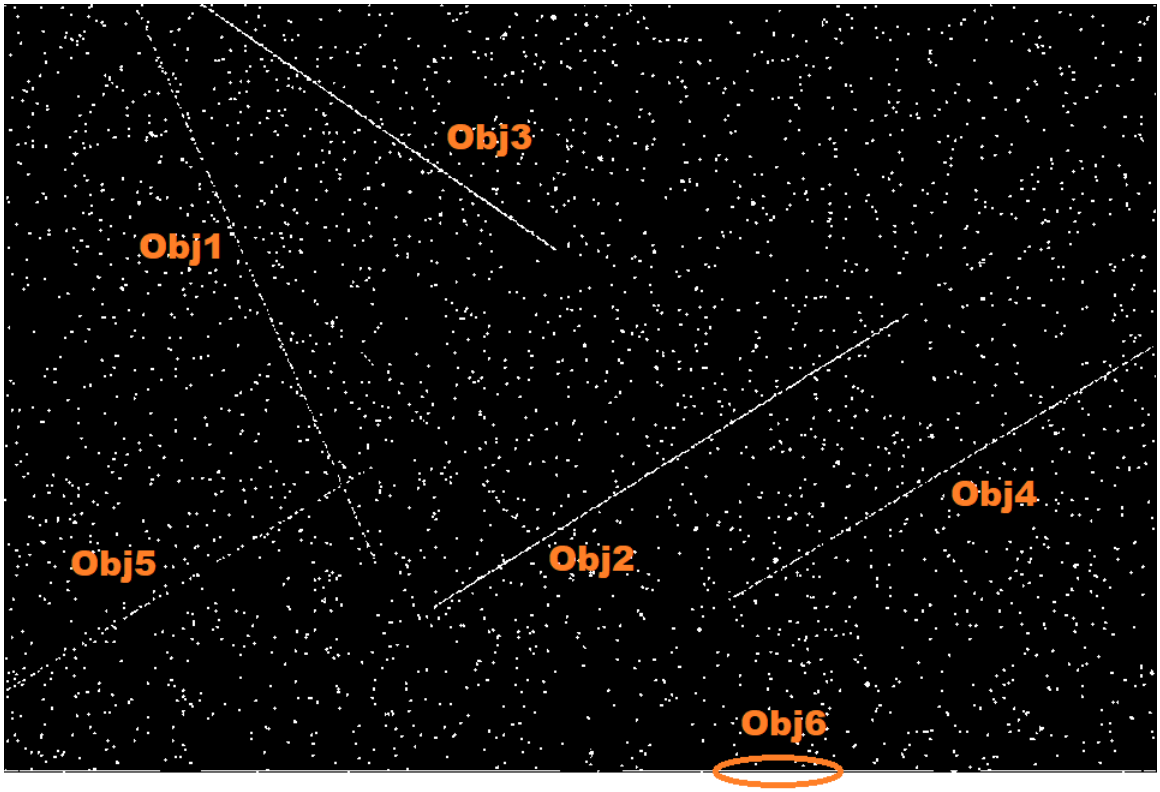

Figure 13. Merging of the whole test sequence with segmented images.

Object 6 is a false object due to a bad segmentation of the image performed by the U-Net. Object 5 is detected just in one image due to its heavily fragmented streak. The exception mentioned in the second scenario of the *RSOs Tracking Module* Subsection happens in the passage from the sixth to the seventh image (Figure 14) in which five boxes are present with 5 objects stored in memory but one of the objects is not detected and a new one appears (Object 6). The algorithm correctly classifies that box as a new object and then proceeds with the tracking.

CONCLUSIONS

In the end, the present work provides the description, design, development and testing of a Convolutional Neural Network-based algorithm for Image Processing, Object Detection and Tracking oriented to space optical sensors applications. Details and characteristics of every sub module are presented together with the optimization problem for the Tracking module. An *ad hoc* dataset for Objects Detection sub-module has been created and presented with characterising histograms through a streaks simulator in night sky images. Details of U-Net and YOLO v3 training have been provided in terms of accuracy. Module tests over multiple real passages have been carried out showing encouraging preliminary performances of the algorithm in terms of Precision, Recall and F1 indices and a strategy to increase performances in terms of FN reduction is shown. In the future, some aspects will be investigated:

- Heuristic optimization of the *Cost Function* weights through extensive tests with the use of a

Table 7. High-Fidelity Star Tracker Simulator tracking results.

$Frame$	Obj_1	Obj_2	Obj_3	Obj_4	Obj_5	Obj_6
1	x	x	x	x	-	-
2	x	x	-	-	-	-
3	x	x	x	x	x	-
4	x	x	x	-	-	-
5	x	x	x	x	-	-
6	x	x	x	x	-	-
7	x	x	x	x	-	x
8	-	x	x	-	-	-
9	-	x	x	-	-	-
10	-	x	x	-	-	-
11	-	x	x	-	-	-
12	-	x	x	-	-	-
13	-	x	x	-	-	-
14	-	x	x	-	-	-
15	-	x	x	-	-	-
16	-	x	x	-	-	-
17	-	x	x	-	-	-
18	-	x	-	-	-	-
19	-	x	-	-	-	-
20	-	x	-	-	-	-

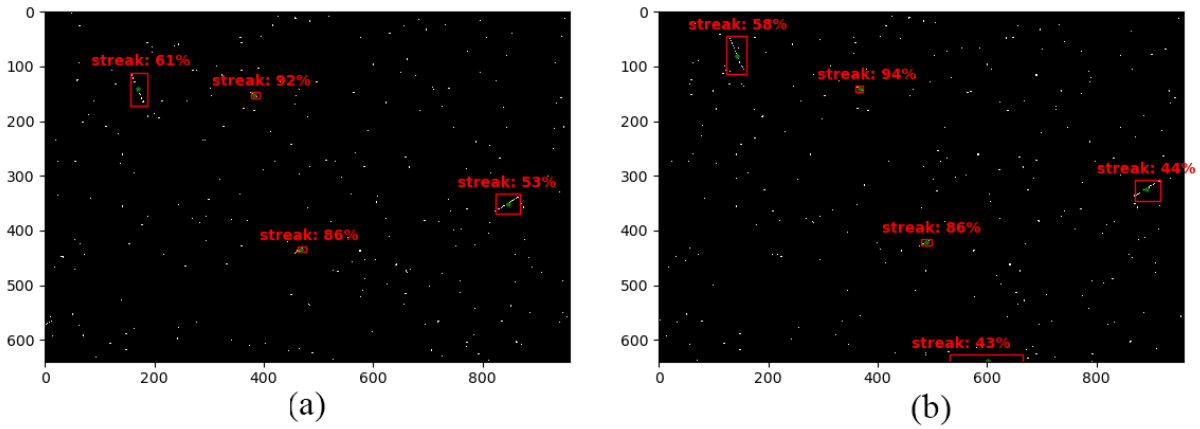

Figure 14. Algorithm inference over frames 6 (a) and 7 (b).

High Fidelity Star Tracker Image Simulator.

- Feasibility study of an improved extremal point estimation strategy by using AI.
- Analysis of a possible integration of the developed architecture on SoC for a potential on-board application.
- Further tests with real images from future acquisition campaigns.

REFERENCES

[1] "Space Debris and Human Spacecraft," `https://www.nasa.gov/mission_pages/station/news/orbital_debris.html`. Accessed: 2022-01-10.

[2] D. J. Kessler, N. L. Johnson, J. Liou, and M. Matney, "The kessler syndrome: implications to future space operations," *Advances in the Astronautical Sciences*, Vol. 137, No. 8, 2010, p. 2010.

[3] D. L. Oltrogge and S. Alfano, "The technical challenges of better space situational awareness and space traffic management," *Journal of Space Safety Engineering*, Vol. 6, No. 2, 2019, pp. 72–79.

[4] J. Utzmann and A. Wagner, "SBSS Demonstrator: A Space-Based Telescope for Space Surveillance and Tracking,"

[5] D. Spiller, E. Magionami, V. Schiattarella, F. Curti, C. Facchinetti, L. Ansalone, and A. Tuozzi, "On-orbit recognition of resident space objects by using star trackers," *Acta Astronautica*, Vol. 177, 2020, pp. 478–496.

[6] M. S. Farissi, I. Agostinelli, M. Mastrofini, F. Curti, C. Marzo, C. Facchinetti, and L. Ansalone, "Hardware Implementation Of The SPOT Payload For Orbiting Objects Detection Using Star Sensors," *72th International Astronautical Congress (IAC), 25-29 October 2021*, 2021.

[7] N. Salvatore and J. Fletcher, "Learned Event-Based Visual Perception for Improved Space Object Detection," *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 2888–2897.

[8] A. Bobrovsky, M. Galeeva, A. Morozov, V. Pavlov, and A. Tsytsulin, "Automatic detection of objects on star sky images by using the convolutional neural network," *Journal of Physics: Conference Series*, Vol. 1236, IOP Publishing, 2019, p. 012066.

[9] D. Xue, J. Sun, Y. Hu, Y. Zheng, Y. Zhu, and Y. Zhang, "Dim small target detection based on convolutinal neural network in star image," *Multimedia Tools and Applications*, Vol. 79, No. 7, 2020, pp. 4681–4698.

[10] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[12] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[13] M. Mastrofini, F. Latorre, I. Agostinelli, and F. Curti, "A CONVOLUTIONAL NEURAL NETWORK APPROACH TO STAR SENSORS IMAGE PROCESSING ALGORITHMS,"

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[16] C. Cortes, M. Mohri, and A. Rostamizadeh, "L2 Regularization for Learning Kernels," *CoRR*, Vol. abs/1205.2653, 2012.

[17] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," *arXiv preprint arXiv:1805.11604*, 2018.

[18] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," *2018 international interdisciplinary PhD workshop (IIPhDW)*, IEEE, 2018, pp. 117–122.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, Vol. 15, No. 1, 2014, pp. 1929–1958.

[20] H. Iqbal, "HarisIqbal88/PlotNeuralNet v1.0.0," Dec. 2018, 10.5281/zenodo.2526396.

[21] F. Chollet, *Deep Learning with Python*. Manning, Nov. 2017.