A Point-based rendering approach for on-board instance segmentation of non-cooperative resident space objects

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Spaziale

Author: **Mario Corradetti**

Student ID: 952839
Advisor: Prof. Pierluigi DI LIZIA
Co-advisors: Niccolò FARACO
Academic Year: 2021-2022

Abstract

According to the ESA's Space Environment Report dated May 19,2022 the increasing number of launches and permanence of spaces debris leads to a significant conjunction risk in the most populated orbit. To reduce this risk different techniques to mitigate the generation of debris or to enable their removal have been studied, but all of them are based on feedback with an on-ground station. The focus of this work is to adopt a machine learning model for instance segmentation that grants better accuracy and robustness than the state-of-the-art-model previously used, Mask R-CNN. Models were compared using different datasets that represent different possible scenarios during a real inspection mission. The simplicity in generating all the datasets is due to the new version of the JINS software that not only allows the user to obtain scenes very close to the reality through few choices to be made in the implemented GUI, but also to be able to describe the attitude of the chaser and target, an option that was not available in the previous version of the software. A fundamental aspect of the datasets used is that they not only contain the greatest variety of possible scenarios but that data augmentation techniques were used to increase the number of images the algorithm is trained on. After testing the models with various combinations of datasets and learning rates, their performance were evaluated on real images from the Intelsat 901 satellite.

Abstract in lingua italiana

Secondo il report del ESA sullo Space Environment del 19 Maggio 2022, l'aumento del numero di lanci e di debris che continuano a rimanere in orbita sta portando a un aumento del rischio di collisioni nelle orbite più popolate. Per ridurre il rischio diverse tecniche per mitigare la generazione di debris o per la loro rimozione sono state studiate, ma tutte quante sono basate su un feedback con una stazione a terra. Scopo di questo lavoro è quello di adottare un modello di machine learning per instance segmentation che permette di ottenere una migliore accuratezza e robustezza rispetto al modello stato dell'arte, Mask R-CNN. I modelli vengono confrontati utilizzando diversi dataset che rappresentano diversi scenari che possono verificarsi durante una missione di ispezione reale. La facilità nel generare tutti i dataset si deve alla nuova verisone del software JINS che non solo permette all'utente di ottenere scene molto simili alla realtà attraverso poche opzioni da scegliere nell'interfaccia grafica implementata, ma è anche possibile descrivere l'attitude di chaser e target, funzione che non era disponibile nella versione precedente del software. Aspetto fondamentale dei dataset utilizzati ò che non solo contengono una grande varietà di scenari possibili ma che tecniche di data augmentation sono state utilizzate per aumentare il numero di immagini utilizzate nell'addestramento. Dopo diversi test eseguiti sui modelli con diverse combinazioni di dataset e di learning rate, le loro performances sono state valutate su immagini reali del satellite Intelsat 901.

Contents

List of Figures

List of Tables

1 | Introduction

1.1. Space situational awareness

Since the 4th of October 1957, with the Sputnik1's launch, Earth orbit is exponentially getting cluttered with human-made objects also known as Resident Space Objects (RSOs). The space race resulted in an impressive number of launches, especially when it comes to military reconnaissance and telecommunication satellite. Today, commercial and scientific application dominate in space flights. All activities in spaces have left traces behind like intermediate stages of launchers, fragmented and decommissioned satellites. To have an overview of the global debris situation, the European Space Agency (ESA) has been publishing a Space Enviroment Report since 2017. According to ESA's latest update, dated May 19, 2022 [9] the number of objects in orbit exceeds 32000 and the majority of these are placed in Low Earth Orbit (LEO) and Geosynchronous Equatorial Orbit (GEO) orbits.

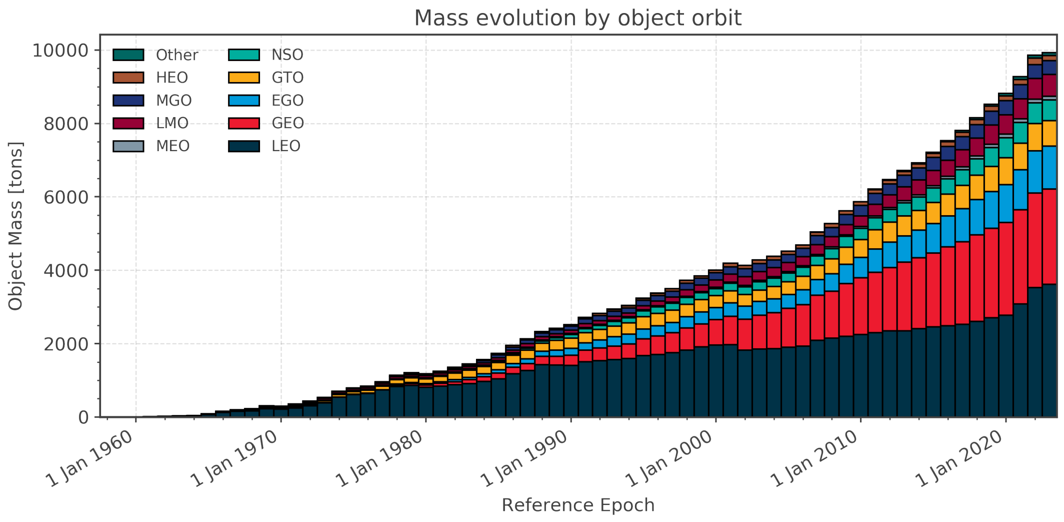


Figure 1.1: Evolution of number of objects in different orbit classes.

On average, over the last 20 years, 11.7 non-deliberates fragmentations occur every year. The increasing number of launches and in-orbit permanence of space debris leads to a significant conjunction risk in the most congested orbital regimes. Until the end of 2021, there have been 636 confirmed on-orbit fragmentation events. If no arrangements are made for the disposal of RSO, the number of collision will rise. Long term, this could lead to "Kessler Syndorme" [6], situation where the number of RSO in space is so high that collision between satellite and debris create a cascade effect, increasing the likelihood of further collision. The cloud of debris resulting from the uncontrollable chain of collision could have dramatic effects, such as making some classes of orbits inhospitable. The current situation pushes space agencies and companies to develop all kinds of solutions to remove debris or mitigate their generation.

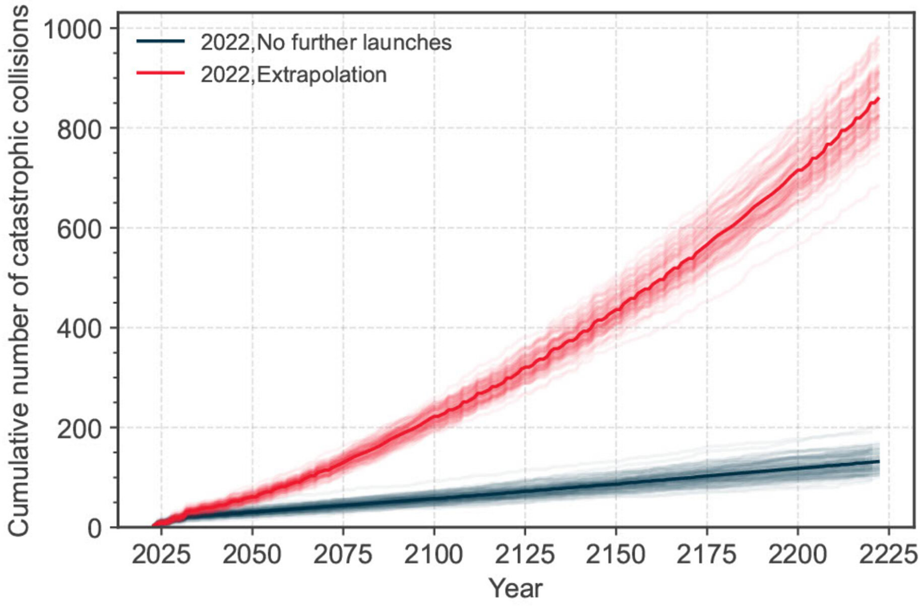


Figure 1.2: Number of cumulative collisions in LEO in the simulated scenarios of long-term evolution of the environment [10].

1.2. State of the art

To forestall such an accumulation buildup, space agencies have begun taking steps to mitigate the problem. Since 1979, NASA's orbital debris program has been trying to reduce the amount of orbital debris and designing equipment to track and remove debris already in space. Initially, techniques to avoid debris formation consisted of simply burning up all the fuel in a rocket stage to avoid explosions in orbit, or taking into account the orbital evolution of debris already present to perform avoidance manoeuvres. With the passing of time in addition to improved techniques for debris generation, several techniques for debris disposal have been developed and depends on the type of object.

Recently launched rocket bodies are either burned up using a "controlled re-entry" or placed on orbits that naturally decay within 25 years. In case of satellites that are close to the end of their operational life and still have control capabilities, their disposal occurs on a graveyard orbit. Problems arises when dealing with non-cooperative objects, and two major options have been studied to overcome this issue: track and propagate the trajectory of each object [35] or design mission for the active removal or disposal of the space debris [23]. The latter, recently, has gained a lot of attention from the space community. Some of the latest techniques use satellites equipped with harpoons [8] or robotic arms to capture debris techniques, lasers to change the shape of the orbit of the debris [37] or use expanding foam increasing the debris's surface to augment the drag [21]. Recently, debris recycling missions have been devised, such as ESA's OMAR mission. An overview on the active debris removal (ADR) techniques developed during the years is shown in Fig. 1.3.

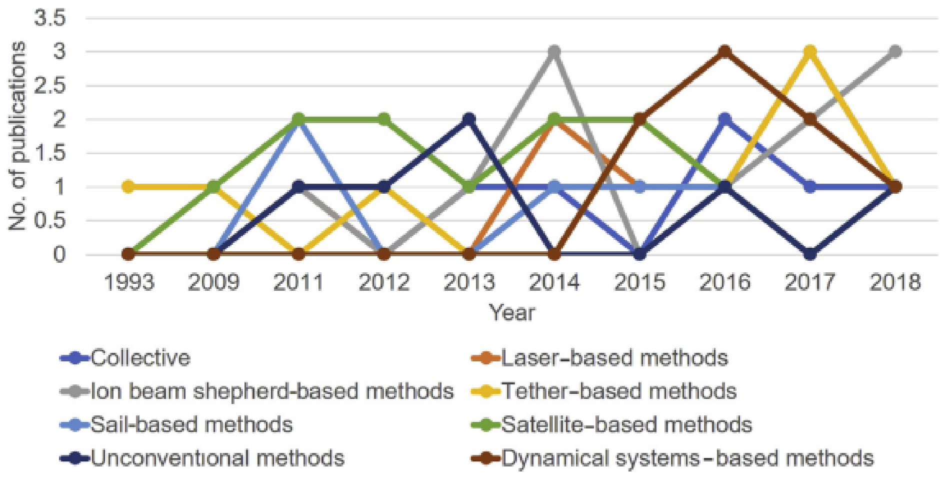


Figure 1.3: Recent trends on ADR methods.

Despite the high efficiency of the proposed techniques, almost none of them has been implemented or proven effective in the operational environment. Some require high power consumption or complex architecture, while others cannot yet be implemented due to technological limitations. All these different techniques have one common point, the knowledge of the attitude of the debris. This means that before the active removal an inspection operation of the target has to be performed. To this aim, many inspection trajectories have been studied [20] and their effectiveness has been confirmed. The selection of the inspection orbit is not a trivial task, due to the non-cooperative and uncontrolled nature of the debris, continuous changes in the trajectory are required to have a detailed description of the scene.

1.3. Proposed approach

Common techniques based on ground communication can not grant real-time decision making which is a fundamental aspect for the inspection, thus, in order to improve the performance of this kind of missions, performing on-board decision could lead to an optimal solution. Since the aim of the inspection mission is a visual inspection, the satellite is equipped with more cameras in order to better catch the scene.
In the proposed approach, the images taken from the cameras are fed inside an on-board computer that uses algorithms and machine learning models (ML) to perform features recognition or pose estimation of the target. An example of on-board control has been tested on the International Space Station (ISS) by means of two Astrobees. Scenes from the simulated experiment are shown in Fig. 1.4.

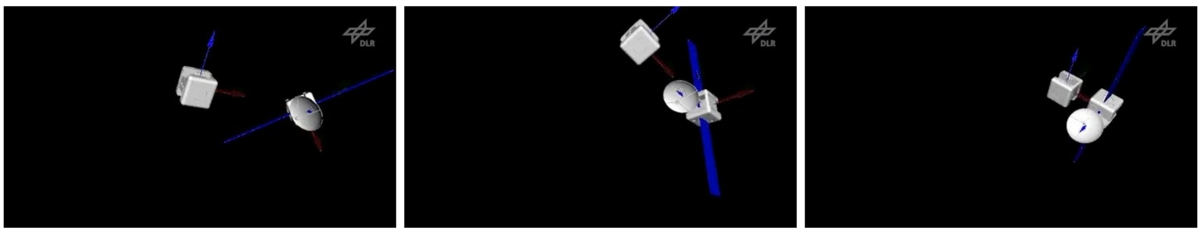


Figure 1.4: Scenes from a simulation of an inspection mission [7].

Computer vision algorithms not only perform analysis on the object but also give a score on how much the results are reliable. The information obtained from the images could be used to modify the inspection orbit to have better points of view of the observed RSO, moreover this could be employed to achieve autonomous navigation as demonstrated in [36]. The key point for this approach of autonomous guidance and navigation is the precision and accuracy of the feature detection algorithm. This topic was initially addressed by [24] and the current thesis is a continuation of the previous work.
Computer vision models require large datasets of labeled images in order for the algorithms to be able to efficiently detect the main features of the scene. Unfortunately, real satellite images currently available are not enough to generate an effective dataset, moreover, they all require manual labeling which is time-consuming. To overcome this problem an improved version of the software JINS[24] has been used. The initial version of JINS allows the scene of a possible inspection mission to be reproduced through the use of 3D CAD models. A `.json` file containing the annotations is generated along with the image of the scene seen from the chaser's point of view. The software has been improved in several aspects :

- User interface
- Scenes more faithfully reflect reality
- Possibility to implement attitude of the target and chaser

The new version of JINS will be illustrated and described in detail later on. The ML model is trained on the generated dataset, and it is then tested on an additional set of images, also known as validation dataset, to evaluate its performance. Before proceeding to the training phase, several ML models were selected and analyzed to see if their characteristics could fit in an inspection mission scenario. Among those selected, PointRend was used because it not only allows instance segmentation like the state-of-the-art model Mask R-CNN but also ensures higher accuracy values. The effectiveness of the trained models has been tested not only on synthetic images generated by JINS, but also on images coming from a real mission, specifically the ones taken during the first docking operation of the Mission Extension Vehicle-1 (MEV-1) to the Intelsat 901 (IS-901) spacecraft.

1.4. Structure of the thesis

This work is structured as follows. A general introduction to CNNs and their uses, along with the various models considered and the architecture chosen is explained in Chapter 2, while Chapter 3 provides an overview of how to describe relative motion between objects in a simplified way and some of the most commonly used inspection maneuvers. Chapter 4 shows how JINS reconstructs the inspection scene, generates the datasets, and the changes made to the software with respect to the previous version. The study of the performance of the chosen model and the comparison with Mask R-CNN are exposed in Chapter 5, analyzing its capabilities on both synthetically generated and real images. To conclude, Chapter 6 sets out conclusions and some ideas for the future development of the work.

2 | Convolutional Neural Network for image processing

2.1. Introduction to Convolutional Neural Network

" *I propose to consider the question: 'Can machines think?'* "

More than 70 years have passed since Alan Turing asked this question [1]. Since then. there has been an exponential increase in computational capabilities and consequently considerable advances in research fields, particularly in Machine Learning (ML). In 1959 Arthur Samuel, pioneer of artificial intelligence, defined the ML as "the field of study that gives computers the ability to learn without explicitly being programmed " [34]. The traditional way of programming needs to create codes with lots of detailed instructions, this is time-consuming and sometimes impossible, as for example in the case of recognizing different objects. While humans can do this tasks easily, it is hard to translate it for a computer. ML lets the computer learn to program itself through experience, and to do this it requires a lot of labelled data. The use of ML covers different fields in modern society, from banking security to advertisements and robotics. One of the main application that in the last decade has gained lot of attention is the computer vision. Computer vision is spreading all over the society from healthcare industry to agriculture and security. Its goal is to give computers the ability to extract high level information from digital images or videos, like objects or patterns, and to use this information for further analysis or decision making. Like any other ML model also computer vision's algorithms need to be trained on lots of data in order to be able to analyze and process images or video.

2.1.1. Convolutional Neural Network

Computer vision is a field of ML that enables computers to derive meaningful information from digital images or videos in the same way that humans do. Two technologies are required for computer vision algorithms to works: deep learning and Convolutional Neural Network. Deep learning is a subfield of ML where the model structure is composed by three or more layers. Differently from the standard machine learning, where important features of the scene are extracted manually to build a model used for object classification, with deep learning the features are extracted automatically, as shown in Fig. 2.1. In addition, deep learning performs "end-to-end learning," in which a model automatically learns how to process raw data and perform a task, such as classification.

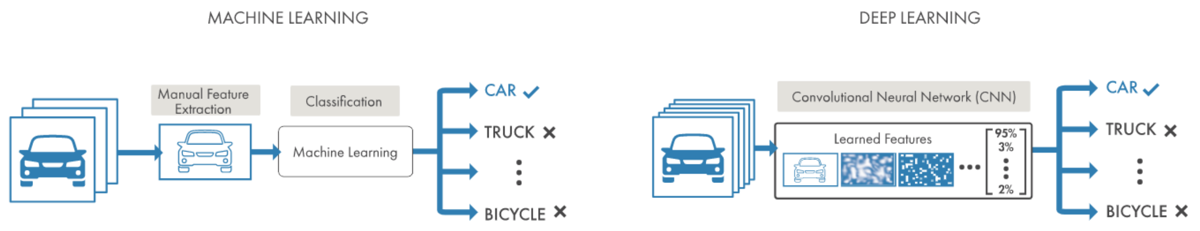


Figure 2.1: Comparison between a machine learning approach (left) and deep learning (right).

Convolutional Neural Network (CNN) is a specific architecture of Artificial Neural Network(ANN) proposed by Yann LeCun [19] and is primarily used in the field of pattern recognition within images. ANNs are composed of a high number of interconnected computational nodes, called neurons, in turn organized into independent layers that can be mainly divided into: input layer, hidden layer and output layer. All the work is done by the hidden layers which make decision based on the information coming from the previous layer. One of the largest limitations of traditional forms of ANN is that each neuron in a particular layer is fully connected to all the neurons in its adjoining layers, this means that a lot of computational effort is required. This problem is overcome by CNNs which use parameter sharing. All neurons in a particular layer share weights which makes the whole system less computationally intense. Goal of CNN is to transform the image in a way that it would be easier to process without losing features, leading to an accurate prediction.
There are four main components in a CNN :

- *Convolution layers*: locally extract features from the input image, so that smaller images are extracted from the original one. On the input image N filters, called kernels, are applied, producing N filtered images, known as feature map. Each filter is meant to highlight certain features of the image. The feature map shrinks every time a convolutional operation is performed. Pixels at the corners and edges are used much less than those in the middle so information coming from the edges is not preserved as the one from the middle. To avoid this problem, padding [16] is performed by adding layers of zeros at the border.

- *Rectified Linear Unit (ReLU) Activation Function*: the images are by their nature full of nonlinearities, the role of the ReLU layer is to introduce non-linearity in the neural model obtained by convolution. This layer is very useful in case there are multiple objects on the image, because boundaries between those objects might not be linear in nature. Without the presence of the ReLU the model becomes a traditional linear classifier so it could not be able to perform image classification or object detection. The ReLU acivation function is a simple piece-wise function represented as :

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

  There are lots of other activation functions such as the sigmoid function or the hyperbolic tangent function.

- *Pooling layer*: pooling operations use a 2D moving filter whose purpose is to decrease the size of feature maps while retaining only the most important information. Reducing the dimension of the feature map also reduces the trainable parameters leading to a faster training of the network. Different types of pooling exist and their use depends on what information have to be preserved. An example of pooling is the max-pooling, shown in Fig. 2.2, that selects the maximum element within the window of the feature map covered by the filter.

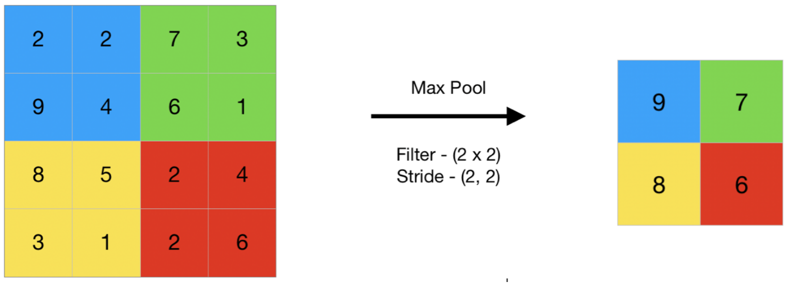


Figure 2.2: Max pooling example.

  Other types of pooling are min-pooling or average-pooling. Pooling thus allows the original image to be represented in a smaller size, enabling a reduction in the computing power required and the risk of overfitting.

- *Fully Connected Layers (FCL)*: after convolutions and pooling, the initial image, provided in matrix form, is transformed into a vector. As a final step, this vector is passed through a Multi-Layer Percetron (MLP) to obtain the output. MLP, or FCL, are those layers where all the neurons from one layer are connected to all the neurons of the next one. Along with Convolution layers, it is the second most time consuming layer.

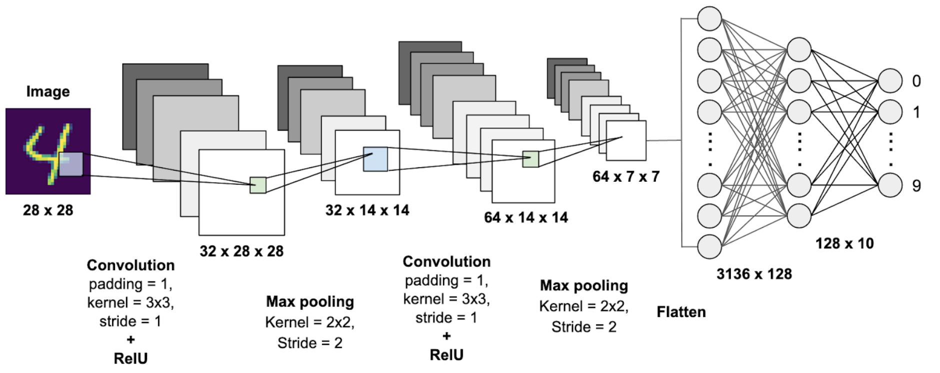


Figure 2.3: CNN architecture.

In order for the CNN to make predictions it needs to be trained. The techniques for training a network depend on the task it will have to perform. The main training methods are the supervised and the unsupervised approach. The main difference between the two methods is that the supervised requires labeled input and output, while the unsupervised works on unlabeled data. Supervised learning emphasizes the relationships between input and output through labels, which is why it is mainly used in computer vision. Unsupervised allows the recognition of patterns or trends within a sequence of data, which is why these kinds of models are mostly used for data analysis or stock predictions. The training process of a CNN can be divided into four steps:

- *Forward phase*: for the first iteration the network weights are initialized with random values such that the prediction on all classes has the same value.
- *Loss function*: it reduces all positive and negative aspects of the model to a single scalar value. To do this, it is necessary to select a loss function,L ,that will faithfully represent the model we want to obtain[15].
- *Backward phase*: the output is back-propagated through the network to evaluate the difference between the target and the computed output[14].
- *Weight update*: is estimated how much a given weight, w_i, contributed to the cost function, L, by evaluating $\frac{dL}{dw_i}$. The weights are updated using a simple relation:

$$w_{i,new} = w_{i,old} - \eta \frac{dL}{dw_{i,old}} \tag{2.1}$$

where η is the learning rate and is one of the most important hyperparameter.

2.2. State of the art of computer vision model

Before describing the different algorithms considered in this paper, it is appropriate to introduce the most common tasks for which computer vision algorithms are used. They are listed below and depicted in Fig. 2.4:

- *Image classification*: uses algorithms to identify which categories of objects are present within an image, among those that the network has been trained to recognize.
- *Object detection*: enables the identification of objects of certain defined classes. Object detection models receive an image as input and return bounding boxes, and their respective labels, on the detected objects.
- *Semantic segmentation*: allows each pixel in the image to be associated with a label. Semantic segmentation, also know as panoptic segmentation, can be seen as a classification of the image at the pixel level but it is not capable of distinguishing between two objects of the same class.
- *Instance segmentation*: enables the detection, segmentation, and classification of every individual object in an image. Can be thought as the combination of object detection and semantic segmentation.

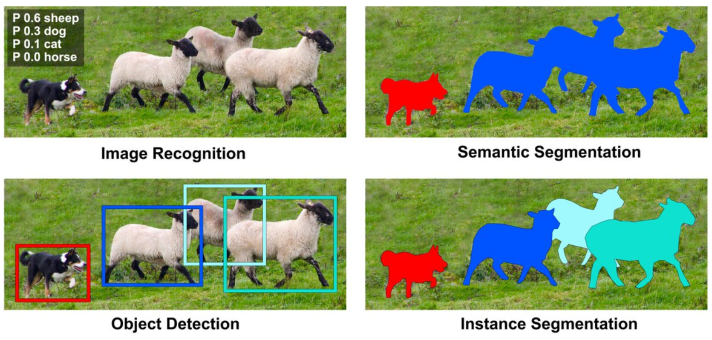


Figure 2.4: Tasks performed by computer vision algorithms.

Although image classification may seem like a trivial and less useful task when compared the other methods, it is the most important as it is the basis for all the other algorithms. Given an image, it is necessary to identify all objects, a difficult task since the number of objects in the scene is not known in advance. For this reason, a standard CNN cannot be used to solve this problem because the length of the output layer is variable. An ingenuous approach to solve this problem would be to take several regions of interest from the image and use a CNN to classify the presence of the object within that region. The issue with this kind of approach is that the objects might have different spatial locations within the scene and different aspect ratios. This leads to having to select a huge number of regions and would require a lot of computational power, for this reason ad hoc methods have been devised.

2.2.1. R-CNN, Fast R-CNN and Faster R-CNN

To overcome the problem of selecting a large number of region, in 2014, Ross Girshick et al. [32] proposed a fundamental concept for all modern object detection models: combine region proposals with CNNs. This method is called Region with CNN feature (R-CNN) and it is based on three main steps :

- The regions in the image where the objects can be found are determined with selective search algorithm
- Once the regions are determined, each one is given as input to a CNN model that acts as a features extractor
- Extracted features are fed into an Support Vector Machine (SVM)[33] to classify the presence of the object within the proposed candidate region.

Despite the advantages introduced by the R-CNN architecture, shown in Fig. 2.5, over a standard CNN, it has several disadvantages. The main disadvantage lies in the fact that it takes about 47 seconds to perform the inference of a single image, this does not allow real-time application. Main reason of this high inference time is the search algorithm which is an algorithm that is not trained so it may select proposed areas that are not perfectly suitable. Also it must be considered that in the training phase a huge amount of memory is needed since for each image about 2000 regions are generated.

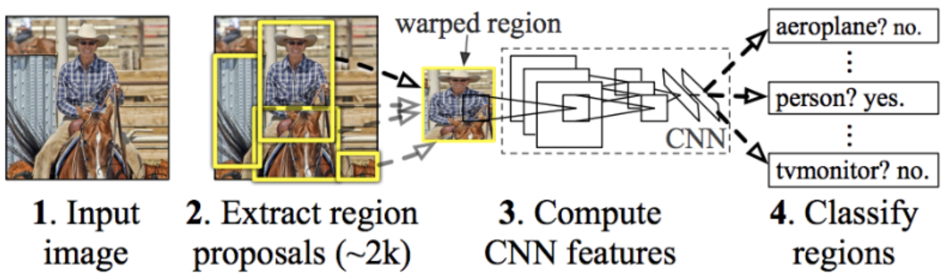


Figure 2.5: R-CNN architecture.

The same Girshick, realizing the large amount of time and memory required by R-CNN, tried to modify its architecture to achieve better performance. He thought that the reason behind the high inference time was that 2000 regions were generated for image and then each region was fed to a CNN. So he came up with the idea to use the CNN only once on the initial image and then apply the zone search algorithm. This new architecture, shown in Fig. 2.6, has been called Fast R-CNN [11], and is based on three steps :

- Proposed regions are generated by the selective search algorithm and, together with the input image, are fed into the CNN that generates convolutional feature map
- A Region of Interest (RoI) pooling layer is applied to each region that reshape them into a fixed size so that they can be fed into a fully connected layer
- The RoI feature vector is then passed into a Softmax classifier and a Bbox regression for classification of region proposal and determine the bounding box of the object detected

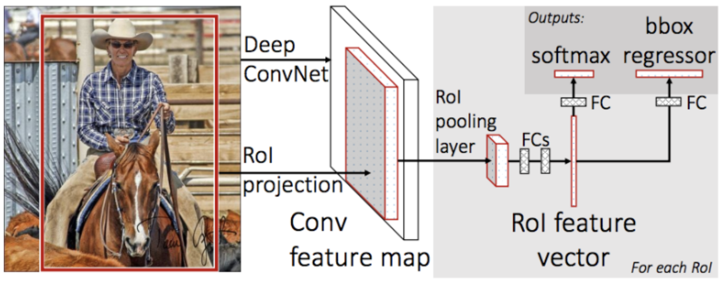


Figure 2.6: Fast R-CNN architecture

By changing the type of architecture, the inference time has been made much faster, from 47 s to 2 s. But still, it cannot be exploited for real-time applications. To reduce the inference time still further, the part devoted to selecting regions of interest needs to be modified. In 2015, Ross Girshick himself along with other colleagues found the solution to this problem by going on to develop Faster R-CNN[31]. Instead of using external selective search another algorithm that lets the network learn region proposals is used, called Region Proposal Network (RPN). Faster R-CNN consists of two networks, RPN and R-CNN, each of which is an independent network and can be trained either jointly or separately. It works with the following operational flow :

- The image is passed through CNN to generate the feature maps
- Feature maps are passed inside RPN which produces anchor boxes
- Since the anchor boxes have different shapes and sizes a pooling layer is applied to reshape them to a fixed size
- The reshaped proposed region are fed through a fully connected layer which has a

Softmax layer and a linear regression layer. Classification and representation of the bounding boxes is output.

Thanks to the described architecture, shown in Fig. 2.7, inference times reach 0.2 seconds allowing Faster R-CNN to be used for real-time applications. A disadvantage of Faster R-CNN is that the RPN is trained by extracting all anchors from a single image. Because all the boxes in a single image can be correlated (i.e., similar features), the network can take time to reach convergence.

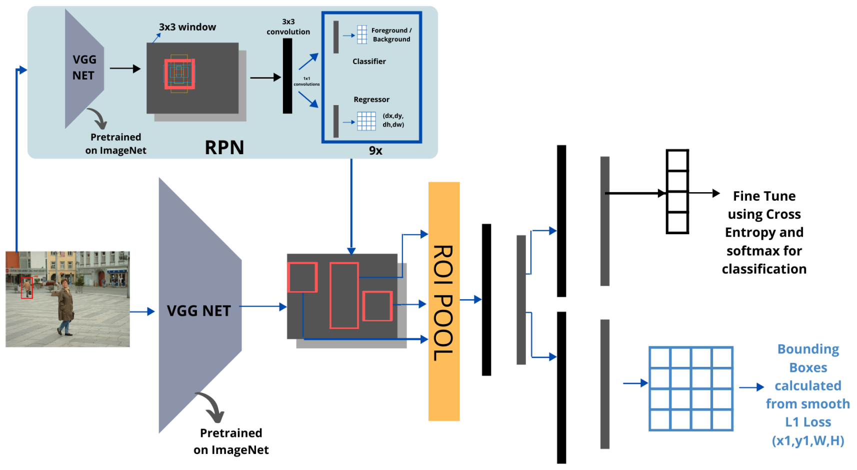


Figure 2.7: Faster R-CNN architecture.

2.2.2. You Only Look Once (YOLO)

Until 2014, R-CNN models were the most popular for object detection. Although the R-CNN family models were accurate, they were relatively slow because it was a multi-step process. In 2015 Joseph Redmon et al. came up with a new model called 'You Only Look Once' (YOLO)[30]. Unlike R-CNN methods, YOLO is able to perform object detection in only one step thus decreasing the inference time. The image is divided into a N $\times$ N grid of cells, each grid cell predicts a ceratain number of bounding boxes as well as C class probabilities. Each object in the image has a cell that is responsible for predicting that object. Nearby high value probability grids are grouped together and considered as a single object. Low value predictions are rejected using a technique called Non Max Suppression (NMS). The very simple architecture, shown in Fig. 2.8, allows the model to process frames at the rate of 45 fps to 150 fps depending on the dimension of the CNN.

Despite its speed of inference YOLO has the problem of not being able to distinguish group of small objects because for each grid cell only one object can be detected. To increase accuracy a new version, YOLO v2 [28], has been developed. Several improvements have

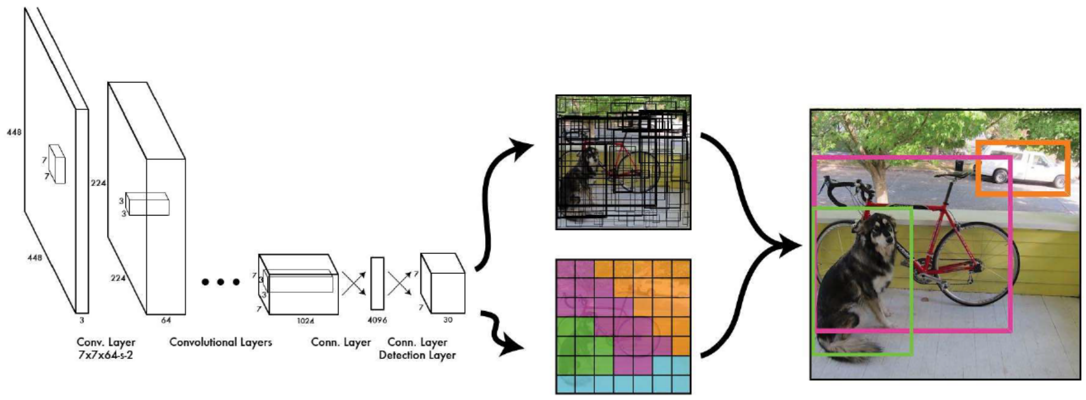


Figure 2.8: YOLOv1 architecture.

been made on the architecture but the most significant are the introduction of a batch normalization at each convolution layer, high resolution training of the classifier and also the problem where each grid cell could only be assigned one object is solved by using anchor boxes. These changes made YOLOv2 competitive in terms of accuracy with the models present at that time. As time went on, many algorithms were developed achieving higher and higher performances, among them YOLO also achieved improvements that led it to become YOLOv3 [29]. The architecture is inspired by other popular architectures such as ResNet and Feature Pyramid Network (FPN) allowing for higher accuracy values but working at slightly lower speeds than preceding versions. After the release of YOLOv3, Joseph Redmon took a break from computer vision research but that did not stop new versions of YOLO from being developed. The goal of all new versions is to increase the accuracy and to reduce the computational and memory cost required. In July 2022 YOLOv7 [38] was released, despite the more complex architecture than the first version it manages to achieve significantly higher accuracy.

2.2.3. Mask R-CNN

Faster R-CNN and YOLO are able to detect objects from the input image with such low inference times that they can be used for real-time applications. However, there is one task that cannot be performed by these models, which is to obtain information about the shape of the object. This feature is essential in the case of On-Orbit Services and can be achieved through instance segmentation models. In 2017 He et al., from the Facebook AI Research (FAIR) group, developed Mask R-CNN[13] which has become the state of the art model for instance segmentation and also the one used by Faraco et al. [24]. In Mask R-CNN a fully convolutional network is added on the top of the CNN features of a Faster R-CNN to generate a mask segmentation output. This is in parallel with the classification and the bounding box regressor network of the Faster R-CNN model. The mask generated is a binary mask in form of a matrix containing 1 when the pixel is part of the mask and 0 elsewhere. The architecture of Mask R-CNN, shown in Fig. 2.9, is composed by a backbone, a Region Proposal Network (RPN), a Region of Interest alignment layer (RoIAlign), a bounding-box object detection head and a mask generation head.

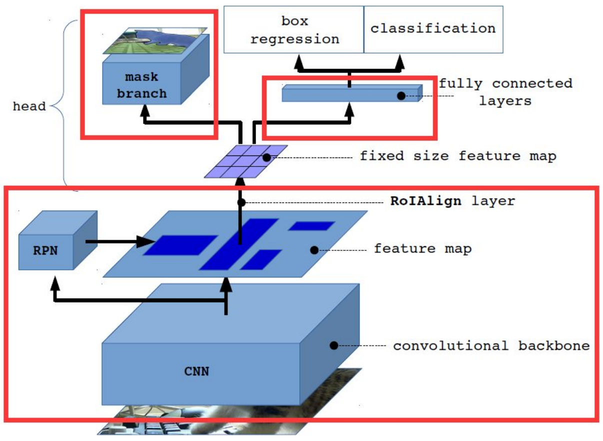


Figure 2.9: Mask R-CNN architecture.

The image is passed through a first CNN which is the main features extractor of the model. Common choices for backbone are ResNet with or without Feature Pyramid Networks (FPN). Once the feature map are obtained they are fed into the RPN that contains both classifier and regressor. The classifier has dimension $1\times1\times9\times2$, 9 due to the number of the anchor boxes used while 2 because is a binary classifier (1 is the object is present 0 otherwise). The regressor is responsible for the generation of the bounding box. Two are the inputs to the ROI layer, the regions of interest generated by the RPN and the feature maps from the first CNN. The main problem is that the input features have different scales and aspect ratio, this can be solved by applying a ROI pooling layer. In normal pooling there can be a mismatch between the input and output dimensions, for example considering a 5x7 feature that needs to be tuned into a 3x3 , an equal division of the map is not possible. For this reason a new version of ROI, called ROI align, has been introduced. After obtaining a map of the individual RoI features, the object category and a bounding-box are predicted. This part is a fully connected layer that maps the feature vectors to the final n classes and $4n$ coordinates of the bounding box. Simultaneously, the feature vector is fed into the mask branch, which consists of two convolution layers, to generate the mask for each ROI.

RoI Align

RoI Pooling, the technique used in Faster R-CNN, is prone to data loss and that is not allowed since, in order to perform instance segmentation,it is necessary to have as much information as possible. This happens when pooling is applied on the region of interest, especially max-pooling. In this method the stride is quantized. Pooling is used for downsampling a feature and to introduce minor distortion such as rotation, such that even if the object fed into the model is rotated it is still capable of recognizing it, i.e. pooling enables a model to become invariant to such distortion. As an example, consider a region of interest of dimension N×N that needs to be mapped to a space M×M where M<N. The required stride S is obtained as M/N. Assuming that C is not an integer number, RoI pooling quantize this value approximating it to the nearest integer. This approximation leads to loss of information and misalignment due to the fact that the stride can not cover entirely the input map. To address this problem RoI Align is used, where no quantization take place. Considering the previous case, with RoI Align, the stride used is C without approximation however this value is meaningless. Each cell is divided into a 2×2 bin, and each of these sub-cells is pulled through bi-linear interpolation leading to four values for cell and the final cell value is computed by either an average or the maximum over the four sub-values. By addressing the loss and misalignment problems of RoI Pooling, the new RoI Align leads to improved results, allowing us to preserve spatial pixel to pixel alignment for every region of interest and no information is lost as there is no quantization.

2.3. PointRend

Currently, convolutional neural network are the main techniques for image segmentation. CNNs operate with regular grid of features and therefore the computation is allocated uniformly across different spatial positions. Output resolution of a CNNs is a trade off between computational cost and the amount of detail it can capture. Different examples of resolution are shown in Fig. 2.10.

A 7×7 output is very efficient to compute but the mask has almost no details, on the other hand a high resolution output captures fine details. However the computational and memory costs might be too high and modern methods balance computation and quality by selecting some resolution in between. Most of the state-of-the-art instant segmentation techniques, such as Mask R-CNN, select 28×28 output resolution. Sometimes different areas of the output mask might require different levels of detail. A regular grid will unnecessarily oversample smooth areas while undersample object boundaries.

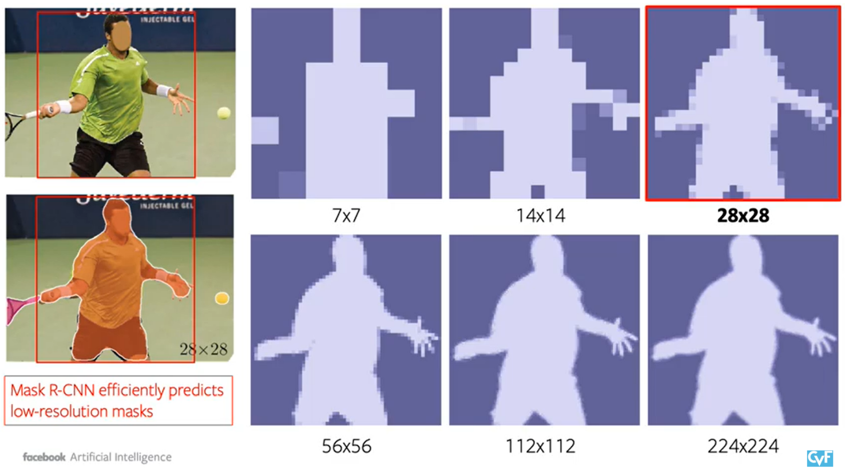

28×28
Mask R-CNN efficiently predicts low-resolution masks
facebook Artificial Intelligence
7x7
14x14
28x28
56x56
112x112
224x224

Figure 2.10: Mask resolution.

The result is overcomputation in the smooth regions and blurred contours. In other fields like 3D recognition and computer graphics more efficient representation than regular grids are commonly used. Inspired by them in 2020 the Facebook AI Research proposed PointRend[17]. The main idea is to view image segmentation as a rendering problem and to adapt classical ideas from computer graphics to efficiently obtain high quality label map. It is based on an implicit function approach, a technique used for rendering in computer graphics to predict a set of sample points at which to compute the label. The PointRend approach starts with a coarse prediction which can be computed efficiently, then it gradually upsamples it using bilinear interpolation, thus refining the prediction only for a subset of points. In few steps a high quality mask can be obtained by refining it only for a small set of points. The overall PointRend architecture and functioning, shown in Fig. 2.11, can be described in :

1. A CNN backbone produces a grid feature representation from an input image
2. For each bounding box detection a small efficient head makes a course prediction
3. Using coarse prediction, a set of points can be sampled, these will need to be refined later. For these points, corresponding features are extracted from the backbones and coarse prediction using bilinear interpolation.
4. For each sampled point an MLP is independently applied to make a refined prediction.

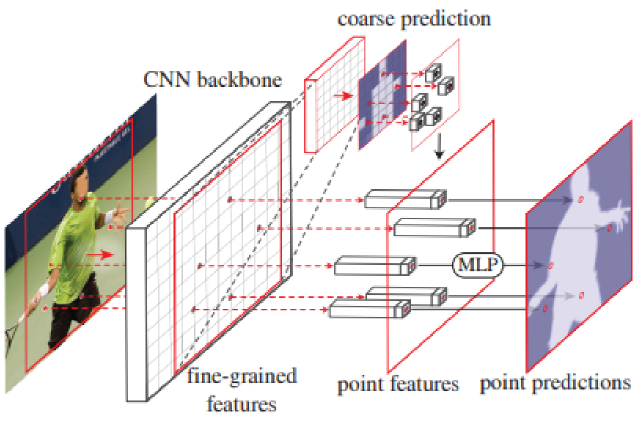


Figure 2.11: PointRend architecture.

2.3.1. Point selection

The core of PointRend is the selection of points in the image in a flexible and adaptive manner. Intuitively, these points should be denser near high-frequency region, such as object boundaries, analogous to the anti-aliasing problem in ray tracing. The way points are selected differs depending on whether one is in the inference or training phase.

Inference phase

The strategy adopted for selecting points during the inference is inspired by the adaptive subdivision, a technique used in computer graphics. The method allows to render high resolutions images efficiently by computing only at locations where there is a high chance that the value is significantly different from its neighbors, leading to a good detection of the edges. For each region of interest, the output mask is iteratively generated in a coarse-to-fine fashion. The low resolution level prediction is done using a standard segmentation head. In each iteration, PointRend performs an upsample of its previously predicted segmentation using bilinear interpolation, then the first N points, between the most uncertain ones, are selected on the low resolution grid. Then the point-wise feature representation, constructed combining the fine-grained and coarse prediction features, is computed for each point and their labels are predicted.
In order to obtain a fine segmentation rendering, a feature vector is extracted at each sampled point from CNN feature maps. A point is a real-value 2D coordinate, so bilinear interpolation is performed on the feature maps to compute the feature vector. Features can be extracted from a single feature map or from multiple feature maps and concatenated by a technique called Hypercolumn method [12]. The fine-grained features allow to

obtain better details but have two drawbacks. First, they do not contain region-specific information. Therefore, during instance segmentation, in the case where a point falls inside two different bounding boxes, additional information on the regions is needed for a good detection of objects' boundaries. Second, accordingly to the feature maps used for the high-resolution features, they may contain low-level information so feature sources with more information can be helpful. By design, the coarse resolution features provide more globalized information. Knowing these drawbacks, the low-resolution feature type is a coarse segmentation prediction from the network like the outputs made by the already existing architectures, 7×7 resolution mask head in case of Mask R-CNN.
Once the point-wise feature representations at each selected point are obtained, PointRend uses a MLP to perform point-wise segmentation predictions. From a computational point of view, if an output resolution of M×M pixels wants to be obtained starting from a resolution of M0×M0 the prediction point required are no more than $N = log_2(\frac{M}{M0})$, and this is much smaller than considering all the points in the grid, allowing PointRend to obtain high-resolution with less computational effort.

Training phase

During inference, the points are selected with a subdivision strategy. If the same technique were used for the training phase, sequential steps would be introduced, which during the backpropagation phase would cause problems such as vanishing or exploding gradient. Therefore a non-iterative method based on random sampling is adopted for the training phase. This method selects N points on the map used for training by focusing on regions with greater uncertainty, but still maintaining uniform coverage. The selection is based on three principles :

- *Over-generation*: instead of considering only N points, kN points (with k>1) are taken from a uniform distribution.
- *Importance sampling*: more attention is given to the points on the low resolution map that have uncertainties, kN point are interpolated and an estimate of the uncertainty is performed. Between the kN point, the most uncertain βN points are selected ($\beta \in [0, 1]$).
- *Coverage*: from a uniform distribution the remaining (1-β)N points are taken.

An example of the points selection with different values of the parameters k and β is shown in Fig. 2.12. The new architecture proposed by PointRend outperforms the standard Mask R-CNN with comparable computational cost. While Mask R-CNN outputs mask with resolution 28×28, PointRend with the same of memory and computational effort predicts

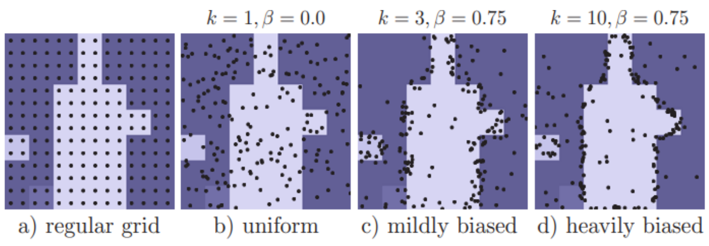


Figure 2.12: Point selection training phase.

mask of resolution 224×224. This increase in resolution allows to efficiently segment the contour of the objects and also sharp boundaries between objects, as it can be noticed in Fig. 2.13. In addition, PointRend can be applied as a plug-and-play on top of any CNN-based image segmentation method.

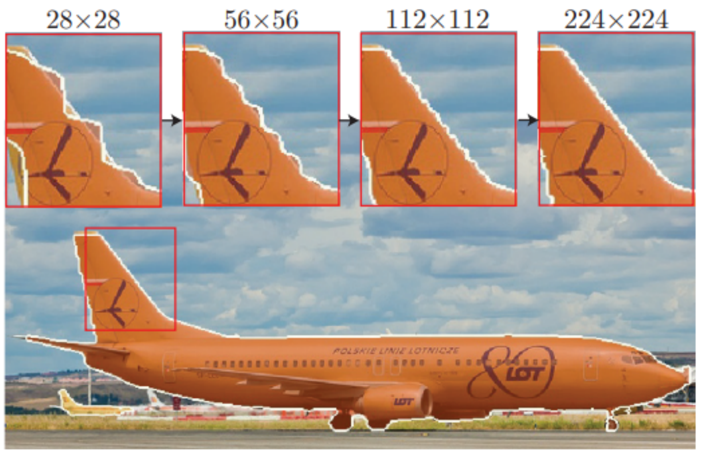


Figure 2.13: Point selection training phase.

An example of Pointrend's improvement over Mask R-CNN is shown in Fig. 2.14.

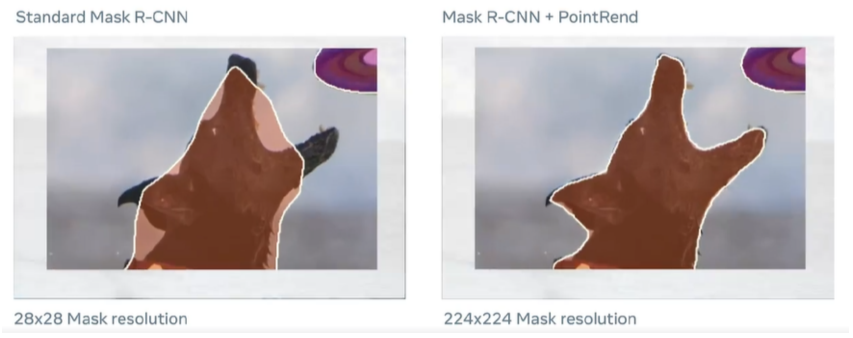


Figure 2.14: Comparison between Mask R-CNN with standard head (left) and Mask R-CNN with PointRend as head (right). Image credit

3 | Trajectory design for on-orbit satellite inspection

On January 21, 2022, United Launch Alliance (ULA) carried out an Atlas V launch of Geosynchronous Space Situational Awareness Program (GSSAP) satellites for a mission that injected the satellites into a near-geostationary orbit. GSSAP is a U.S. Space Force project whose purpose is to inspect spacecraft operating in geostationary orbit using small satellites. In order for the satellite, called chaser, to inspect the RSO, the target, there must be relative motion between the two. The relative motion between two objects is described by complex equations that, in general, do not have closed-form solutions. In the case of a real mission, not only the relative motion between the two objects is necessary, but the chaser must also keep the target in its field of view (FOV) through a correct attitude control. As a first analysis in this work only chaser-target relative motion will be considered, assuming that the target is always in the center of the chaser's FOV. A simple illustration of the scene both inertial, or Earth Centered Inertial (ECI), and relative frame is shown in Fig. 3.1 .

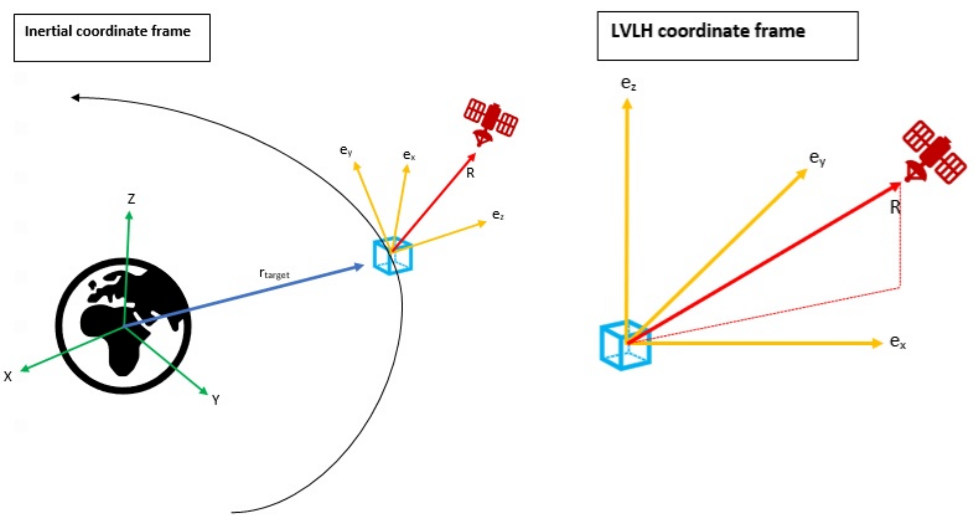


Figure 3.1: Representation of the reference frames considered.

3.1. Clohessy-Wiltshire equations

The reference frame used to describe the scene is the Local Vertical Local Horizontal (LVLH) coordinate frame. It is described by three versors $\{\hat{e}_x, \hat{e}_y, \hat{e}_z\}$ centered on the target. The $\hat{e}_z$ vector is directed along the radial direction from the Earth to the target. This axis can be linked to the difference in altitude between the chaser and the target, it is referred as the altitude component. The $\hat{e}_y$ axis is orthogonal to the target's orbital plane, it describes an out-of-plane motion and is known as cross-track component. The $\hat{e}_z$ is the one that completes the reference frame following the right hand rule, it represents how far the chaser is from the target and is therefore why is called downrange component. Each axis can be derived mathematically just by knowing the position and velocity vector of the target in inertial reference frame [5].

$$\text{Altitude} \qquad \hat{e}_z = \frac{\underline{r}_{target}}{|\underline{r}_{target}|} \tag{3.1a}$$

$$\text{Cross-Track} \qquad \hat{e}_y = \frac{\underline{\omega}_{target}}{|\underline{\omega}_{target}|} \tag{3.1b}$$

$$\text{Downrange} \qquad \hat{e}_x = \hat{e}_y \times \hat{e}_z \tag{3.1c}$$

where $\underline{\omega}_{target}$ is the target's angular velocity and can be computed as :

$$\underline{\omega}_{target} = \frac{\underline{r}_{target} \times \underline{v}_{target}}{|\underline{r}_{target}|^2} \tag{3.2}$$

The position vector of the chaser in the LVLH frame can be expressed as :

$$\underline{R} = x\hat{e}_x + y\hat{e}_y + z\hat{e}_z \tag{3.3}$$

The advantage in using the LVLH frame to describe relative chaser-target motion is that the dimensions of the relative orbit are easily represented and is more meaningful than its representation in inertial frame. The inertial position of the chaser in the inertial frame can be evaluated from the inertial position of the target and the relative position chaser-target :

$$\underline{r}_{chaser} = \underline{r}_{target} + \underline{R} = (r_{target} + z)\hat{e}_z + y\hat{e}_y + x\hat{e}_x \tag{3.4}$$

By making two simplifying assumptions on target and chaser, their relative motion can be described by a simplified set of equations. The first assumption is to consider the target on a circular orbit, this means that the angular rate is constant and depending on the radial distance. Second, the chaser and the target are close to each other such that :

$$|\underline{R}| \ll |\underline{r}_{target}| \tag{3.5}$$

Applying these two conditions a set of simplified equations, called the Clohessy-Wiltshire equations, can be used for describing the relative motion [3].

$$\text{Downrange :} \qquad a_x = \ddot{x} + 2\omega\dot{z} \tag{3.6a}$$

$$\text{Cross-Track :} \qquad a_y = \ddot{y} + \omega^2 y \tag{3.6b}$$

$$\text{Altitude :} \qquad a_z = \ddot{z} - 2\omega\dot{x} - 3\omega^2 z \tag{3.6c}$$

The terms a_x, a_y, a_z represent the external accelerations. Assuming spherical Earth, circular target orbit and neglect the disturbances a closed form solution can be determined :

$$\begin{aligned} x(t) &= \left(6z_0 + \frac{4\dot{x}_0}{\omega}\right)\sin(\omega t) + \frac{2\dot{z}_0}{\omega}\cos(\omega t) - (6\omega z_0 + 3\dot{x}_0)t + \left(x_0 - \frac{2\dot{z}_0}{\omega}\right) \\ y(t) &= y_0\cos(\omega t) + \frac{\dot{y}_0}{\omega}\sin(\omega t) \\ z(t) &= \frac{\dot{z}_0}{\omega}\sin(\omega t) - \left(3z_0 + \frac{2\dot{x}_0}{\omega}\right)\cos(\omega t) + \left(4z_0 + \frac{2\dot{x}_0}{\omega}\right) \end{aligned} \tag{3.7}$$

The assumption of neglecting external disturbances may be acceptable for a first approximation, but in the later stages of design they must be considered.

3.2. Repeating inspection orbit

The number of inspection orbits that can be achieved is infinite, one of the key parameters for choosing the orbit is the fuel consumption that is desired to be reduced as much as possible. A desirable situation would be the one where the relative motion between chaser and target is repeated over time and bounded. This type of orbit can be achieved without fuel consumption by observing the nature of Eq. (3.7), particularly out-of-plane, $y(t)$, and radial motion, $z(t)$. This is not valid for the cross-track component which includes a component that increases linearly over time:

$$(6\omega z_0 + 3\dot{x}_0)\,t \tag{3.8}$$

The condition of repetitive relative motion can be obtained naturally when both chaser and target have the same orbital period. Having the same period means that after an orbit they will return to the starting point while maintaining the relative position. From the definition of orbital period it can be observed that it is a function of the semi-major axis, a, of the orbit :

$$T = 2\pi\sqrt{\frac{a^3}{\mu}} \tag{3.9}$$

The semi-major axis also defined the specific orbital energy:

$$\epsilon = -\frac{\mu}{2a} = \frac{v^2}{2} - \frac{\mu}{r} \tag{3.10}$$

Therefore several inspection orbits can be generated from an initial situation where chaser and target are on the same orbit, but to maintain periodicity the maneuvers must not change the orbital energy of the chaser. Considering the scenario where both chaser and target are on the same orbit as in Fig. 3.2, performing a manoeuvre that changes only the direction of the velocity but not its magnitude causes the chaser to enter into a periodic relative motion with respect to the target.

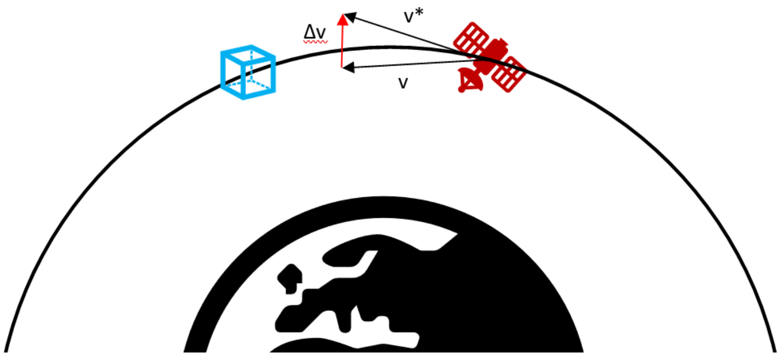


Figure 3.2: Repeating relative motion manoeuvre.

$A\Delta v$ that only has components along the radial or the cross-track direction, so perpendicular to the velocity vector, rotates the latter without changing its magnitude. If the manoeuvre presents a component along the downrange direction the velocity magnitude changes, hence a variation in the orbital energy is caused, so the chaser enters in a non-repeating inspection orbit.

Football orbits

Consider the initial scenario described in Fig. 3.2 where the chaser and the target are on the same initial circular orbit. If a Δv with only a radial component is applied, as a first approximation it can be said that only its direction has changed but not its magnitude. This causes the chaser to enter a relative periodic orbit with respect to the target. The approximation considered regarding the maneuver is acceptable because the impulse intensity required by an inspection mission is about three orders of magnitude less than the speed at which a satellite travels. This maneuver causes the chaser to move to an orbit with the same energy level but a different shape. The relative motion that is generated goes to describe an ellipse, which is why this type of inspection orbit is called a football orbit. Assuming the chaser ahead the target and the Δv directed along the positive direction of $\hat{e}_z$, the circular orbit become slightly elliptic, as shown in Fig. 3.3

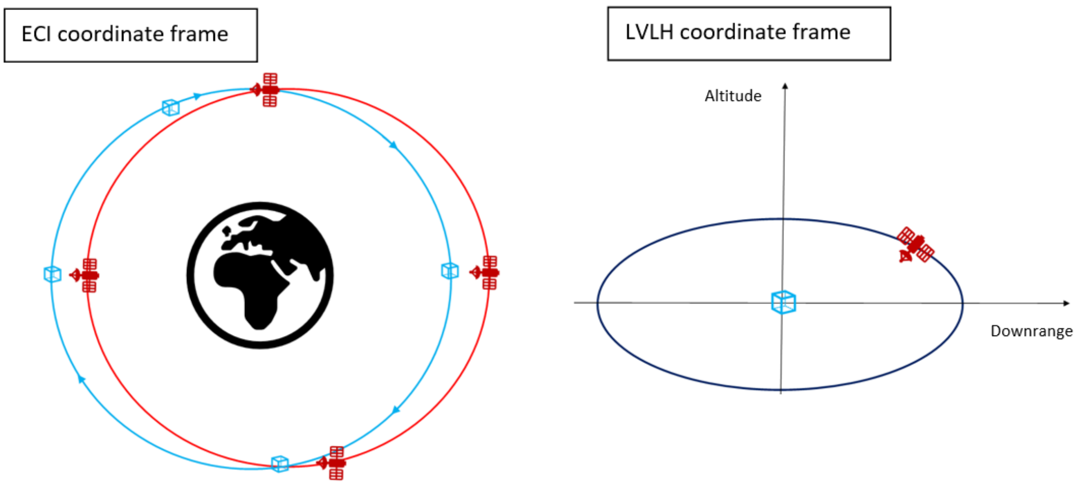


Figure 3.3: Football orbit in ECI and LVLH coordinate frame.

According to the vis-viva equation [5], an increase in the distance causes a decrease in the velocity.

$$v^2 = \mu\left(\frac{2}{r} - \frac{1}{a}\right) \tag{3.11}$$

The effect is to have an alternate upward and drift-back movement. As the chaser approaches the apogee it gradually slows down. Then, its radial distance starts decreasing causing the chaser to accelerate. Since the velocity change due to the impulse doesn't modify the semi-major axis of the chaser, its orbital period remains the same causing it to return to the same initial condition after a revolution. Considering the Eq. (3.7) a relationship between the semi-major axis and the Δv applied to enter the relative motion can be derived. Assume that the chaser is in front of the target of a certain distance x_0 in the downrange direction, it implies that the initial relative altitude is $z_0 = 0$. To enter into a football relative orbit the Δv must be only along the radial direction, which means that no downward component should be present, $\dot{x}_0 = 0$. Referring to the downrange component, $x(t)$, from Eq. (3.7) and imposing the previous condition:

$$x(t) = \frac{2\dot{z}_0}{\omega}\left[cos(\omega t) - 1\right] + x_0 \tag{3.12}$$

The chaser after a quarter of the orbital period reaches the maximum value of the relative altitude. Recalling that $\omega = \frac{2\pi}{T}$, the previous expression evaluated at $t = \frac{T}{4}$ becomes:

$$x\left(\frac{T}{4}\right) = -\frac{2\dot{z}_0}{\omega} + x_0 = 0 \tag{3.13}$$

Since the pulse is purely in the radial direction $\Delta v = \dot{z}_0$ the semi-major axis can be simply evaluated :

$$a_{Football} = 2\frac{\Delta v}{\omega} \tag{3.14}$$

Repeating the same procedure on the altitude component, the semi-minor axis of the football orbit can be determined:

$$b_{Football} = \frac{\Delta v}{\omega} \tag{3.15}$$

It can be easily noticed that the football orbit has a 2:1 ratio of the semi-major axis to the semi-minor axis.

Oscillating orbits

Consider again the initial scene illustrated in Fig. 3.2, but this time the Δv is applied only along the cross-track direction. A pulse in that direction will cause the chaser to change its inclination instead the eccentricity as before. This variation of inclination lets the chaser experience a bounded up and down relative motion with respect to the target, that is why this type of orbit is called oscillating orbit. An example is shown in Fig. 3.4.

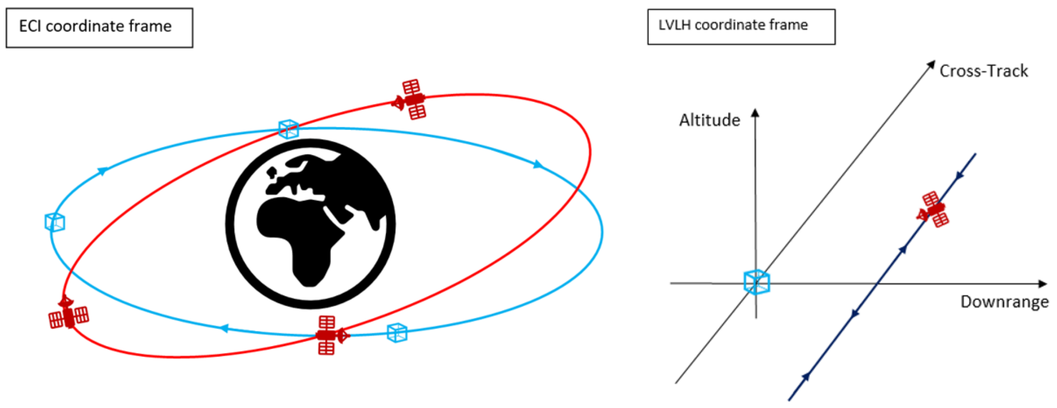


Figure 3.4: Oscillating orbit in ECI and LVLH coordinate frame.

It can be noticed that the equation that describes the cross-track component is decoupled from the other components , therefore only that component is considered from Eq. (3.7).

Initially both chaser and target are on the same orbit so $y_0 = 0$, and since the impulse is only on the cross-track direction $\Delta v = \dot{y}_0$ the equation that describes the cross-track motion becomes:

$$y(t) = \frac{\dot{y}_0}{\omega} sin(\omega t) \tag{3.16}$$

From Eq. (3.16) can be derived the amount of fuel required to travel Δy meters out-of-plane with respect to the target :

$$\Delta v = \omega \Delta y \tag{3.17}$$

Inclined Football orbits

A combination of the two orbits described above allows the generation of an additional periodic inspection orbit called inclined football orbit, shown in Fig. 3.5 . This can be easily achieved by applying an impulse with both radial and cross-track components. Unlike the derivation for the football orbit, the initial velocity component along the cross-track direction, $\dot{y}_0$, is nonzero. Starting from Eq. (3.7) they can be simplified in:

$$\begin{aligned} x(t) &= x_0 + \frac{2\dot{z}_0}{\omega}[cos(\omega t) - 1] \\ y(t) &= \frac{\dot{y}_0}{\omega} sin(\omega t) \\ z(t) &= \frac{\dot{z}_0}{\omega} sin(\omega t) \end{aligned} \tag{3.18}$$

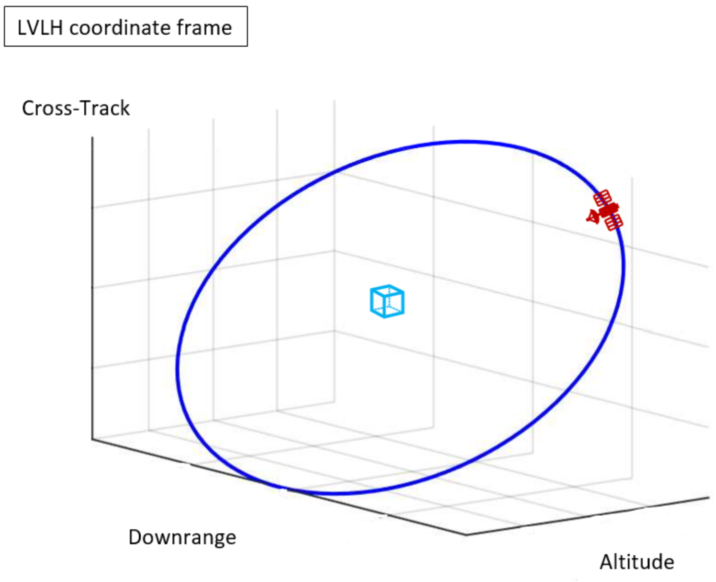


Figure 3.5: Inclined Football orbit in LVLH coordinate frame.

Rotating Football orbits

The methods previously described allow the target to be observed by having the chaser move only in one plane. Some missions may require observing the target from different points of view to obtain a greater amount of information. The football orbit is the most interesting repeating relative orbit for an inspection mission. By performing a plane change maneuver, it is possible to rotate the orbit, obtaining new views and thus greater knowledge of the target as illustrated in Fig. 3.6.

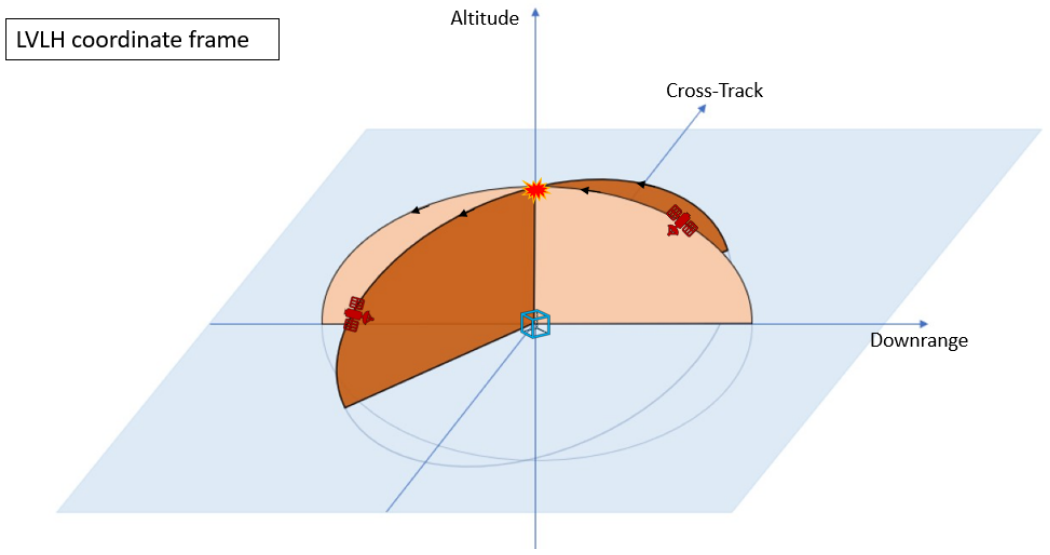


Figure 3.6: Rotating football orbit in LVLH coordinate frame.

This kind of approach allows the chaser to cover all the possible sides of the target, with a relatively simple manoeuvre. This condition occurs when the cross-track maneuver is performed at either the top or bottom of the football orbit. It can be noticed that the football orbit rotates around the altitude direction. Rotation can also occur around the downrange axis, but the passive collision avoidance capability is jeopardized[39]. For this type of inspection two types of impulses need to be performed: an entrance Δv that allows the chaser to enter into a football orbit and an out-of-plane maneuver to rotate the football.

4 | JINSv2: a synthetic images generator

As mentioned in chapter 2, Machine Learning models need data to be trained. Given the complexity of the task of recognizing objects, the dataset must not only contain as many examples as possible, but must also be robust to different conditions of the analyzed scene. An image classification model must be invariant to the presence of different variations; some of them defined by [2] are shown in Fig. 4.1:

- *Viewpoint variation*: an object can be oriented in many ways with respect to the camera.
- *Scale variation*: the same class presents objects with different scales from each other.
- *Deformation*: many objects of interest are not rigid bodies and can be deformed in extreme ways.
- *Occlusion*: the displayed scene may have occlusions making only certain areas of the object of interest visible.
- *Illumination conditions*: different lighting levels can make some parts of the scene difficult to be distinguished.
- *Background clutter*: the object of interest may have physical characteristics that make it difficult to identify.
- *Intra-class variation*: all objects in a class are not always associated with a single shape, as in our case the solar panels or the main body of the satellite

The number of images a dataset should contain is a topic that has always divided people in the computer vision field. Some people say that 100 images per class are sufficient, while others argue that the minimum number of images should exceed 1000 images per class. Indeed, the greater the number of images used during training, the higher the accuracy of the predictions will be, but this involves significant effort in generating the dataset as each image must be manually annotated. To avoid this time-consuming task several datasets

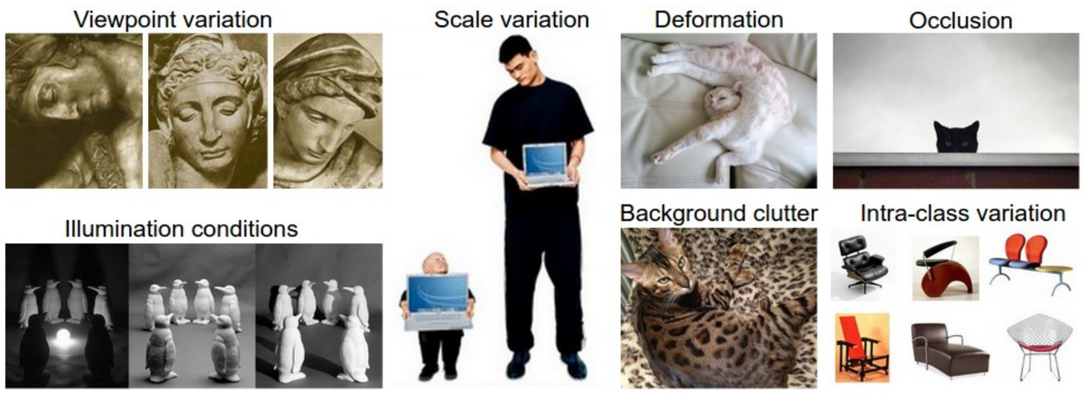
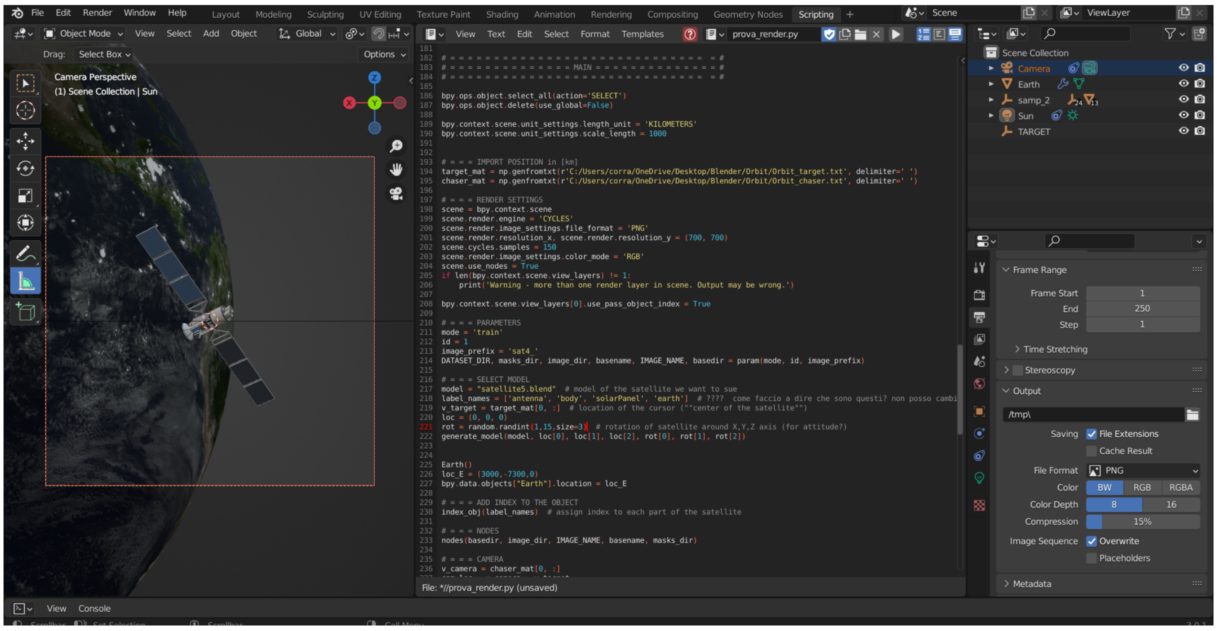


Figure 4.1: Different example of image variations. Image credit.

have been made public, they contain from 1200 to 24k annotated images. These types of datasets have scenes of objects that are used in everyday life (COCO, PASCAL VOC) or street scenes to train autonomous driving algorithms (Cityscape). In our case, there are no already annotated datasets for RSOs, so it needs to be generated from scratch. The only dataset concerning satellite components is the one proposed by Zhao et al. [42], it contains images taken from the web (both realistic and fictitious scenes) on which only the solar panels have been labeled.
Using this type of approach to construct the dataset, in addition of being time-consuming, does not allow to fulfill the points described above. To overcome these types of problems, Jins Is Not a Simulator (JINS) software was developed [24]. Purpose of the software is to automatically generate annotated images by using a computer graphics software. The process of generating data through programming is called Synthetic Data Generation, it allows to faithfully represent a variety of scenes and increasing the accuracy of the model since the annotations are more precise than the manual ones. The 3D elements required to reconstruct accurately an inspection scenario are: the target satellite model, the one of the Earth, a camera that captures the scene and a light to simulate the Sun.
The scene is built using special 3D modeling software called Blender[1]. Blender is a free open-source program provided by the Blender Foundation, it is embedded with many features that allow it to be on the same level as other paid programs used in computer graphic. It is mainly used for modeling and animation of 3D objects, but it is also used for special effects, video games, or realistic scene generation. Among all the functions offered the most interesting one is the capability to automate the scene generation through python scripting. To generate the scene, Blender's *Scritping* environment, shown in Fig. 4.2, can be used directly to run Python's code that allows to place each model at the desired location and assign property to the scenario like the light intensity, type, direction and

[1]Official website : `https://www.blender.org/`

to add constraints to the objects. The possibility to employ Blender's API for Python scripting not only enables the automatization of the image generation but also to code specialized tools or functions not present in Blender. A factor to take into account is the quality of the rendered image, which is greatly influenced by the rendering mode used. Blender provides two main rendering engines : *Eevee* and *Cycles*. *Eevee* is a real-time rendering engine, so the render is basically instantaneous and it is often used as a preview engine to check the lights and the materials in the scene while still modeling. Since it is a real-time rendering the quality is not that high. *Cycles* is a physically based rendering engine that tries to reproduce as faithfully as possible reflections and shadows by means of a ray-tracing model. For this work, a cycle-render engine was used as it is necessary to have images as close as possible to reality to validate the model in case of a real scenario. To get rendering speeds comparable to *Eevee*, it is possible to take advantage of the Graphical Processing Units (GPU) since their architecture is better suited for this kind of tasks. Another reason to use *Cycles* is that it is the only rendering engine that supports the *Compositor mode* needed for mask generations.



```
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = #
# = = = = = = = = = = = = = = MAIN = = = = = = = = = = = = = #
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = #

bpy.ops.object.select_all(action='SELECT')
bpy.ops.object.delete(use_global=False)

bpy.context.scene.unit_settings.length_unit = 'KILOMETERS'
bpy.context.scene.unit_settings.scale_length = 1000


# = = = IMPORT POSITION in [km]
target_mat = np.genfromtxt(r'C:/Users/corra/OneDrive/Desktop/Blender/Orbit/Orbit_target.txt', delimiter=' ')
chaser_mat = np.genfromtxt(r'C:/Users/corra/OneDrive/Desktop/Blender/Orbit/Orbit_chaser.txt', delimiter=' ')

# = = = RENDER SETTINGS
scene = bpy.context.scene
scene.render.engine = 'CYCLES'
scene.render.image_settings.file_format = 'PNG'
scene.render.resolution_x, scene.render.resolution_y = (700, 700)
scene.cycles.samples = 150
scene.render.image_settings.color_mode = 'RGB'
scene.use_nodes = True
if len(bpy.context.scene.view_layers) != 1:
    print('Warning - more than one render layer in scene. Output may be wrong.')

bpy.context.scene.view_layers[0].use_pass_object_index = True

# = = = PARAMETERS
mode = 'train'
id = 1
image_prefix = 'sat4_'
DATASET_DIR, masks_dir, image_dir, basename, IMAGE_NAME, basedir = param(mode, id, image_prefix)

# = = = SELECT MODEL
model = "satellite5.blend"  # model of the satellite we want to sue
label_names = ['antenna', 'body', 'solarPanel', 'earth']  # ????  come faccio a dire che sono questi? non posso cambi
v_target = target_mat[0, :]  # location of the cursor (""center of the satellite"")
loc = (0, 0, 0)
rot = random.randint(1,15,size=3)  # rotation of satellite around X,Y,Z axis (for attitude?)
generate_model(model, loc[0], loc[1], loc[2], rot[0], rot[1], rot[2])


Earth()
loc_E = (3000,-7300,0)
bpy.data.objects["Earth"].location = loc_E

# = = = ADD INDEX TO THE OBJECT
index_obj(label_names)  # assign index to each part of the satellite

# = = = NODES
nodes(basedir, image_dir, IMAGE_NAME, basename, masks_dir)

# = = = CAMERA
v_camera = chaser_mat[0, :]
```

File: *//prova_render.py (unsaved)

Frame Start 1
End 250
Step 1
/tmp\
Saving ✓ File Extensions
Cache Result
File Format PNG
Color BW RGB RGBA
Color Depth 8 16
Compression 15%
Image Sequence ✓ Overwrite
Placeholders

Figure 4.2: Example of Blender's GUI.

Blender allows the user to insert his own CAD model of the satellite into a scene, a possibility that was leveraged in this work. The models used were not designed from scratch as that is not the purpose of this paper, but models made available on the web are employed. Their number is limited to 6, to have a fair comparison with [24], and are showed in Fig. 4.3:

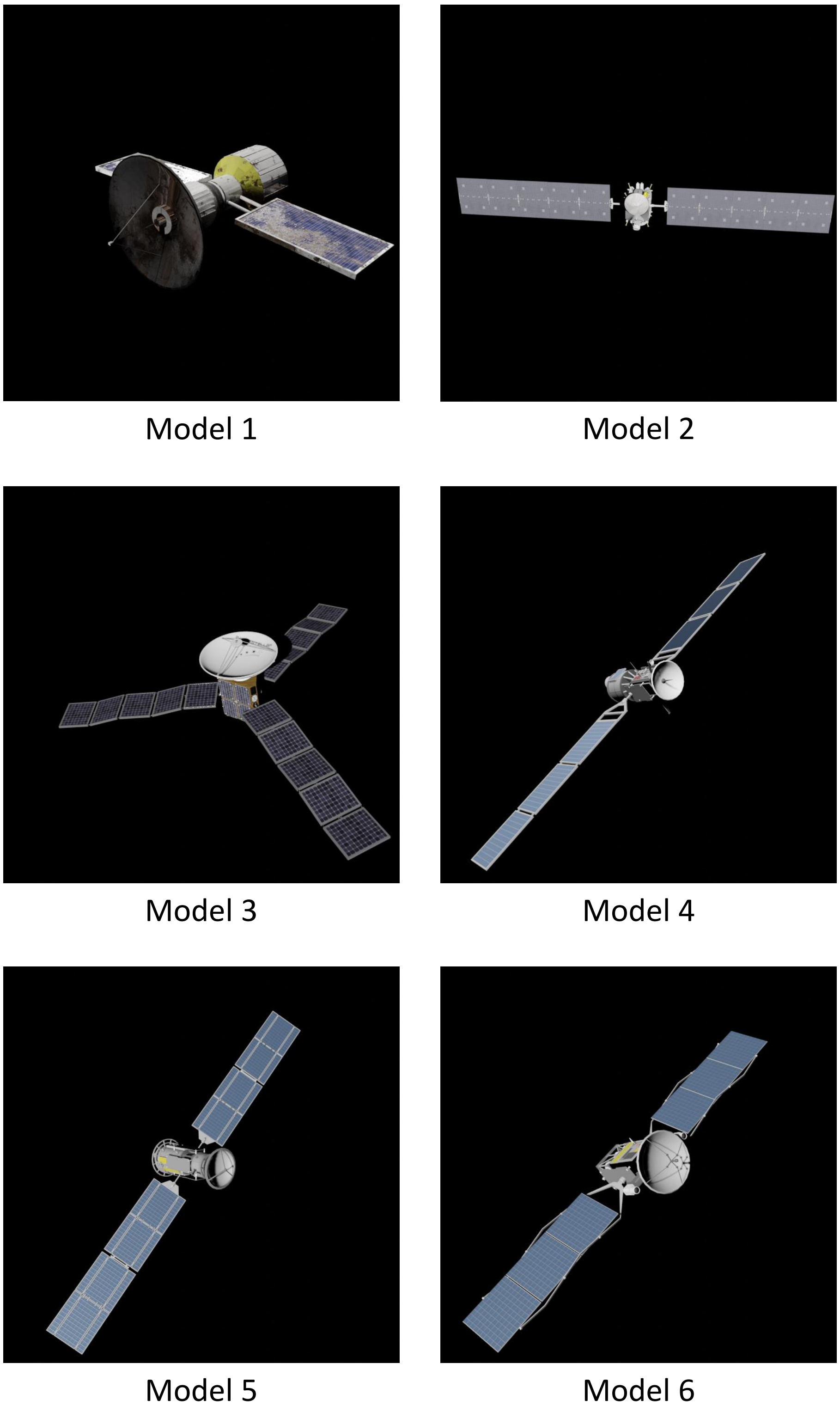


Figure 4.3: Spacecraft models used in the work.

The initial CAD model consists of many parts, but thanks to the *Join* tool provided by Blender it is possible to merge all the components into 3 macro-categories that are of most interest for an inspection mission. The class of the components can be assigned by creating a custom property identified by the word '*label*' and with the value set as a string of the corresponding class. The categories used for the study of RSOs are *body*, *antenna* and *solar panel*. An additional category, the *thrusters*, could have been considered, but does not appear to be of particular interest for an inspection mission. Label assignment can be done either manually or through a special plug-in[2], in our case few models and few classes are used so they were assigned by hand. As mentioned before, the advantage of using JINS is being able to make realistic images already annotated. Several parameters are set before proceeding to the generation of the images:

- Resolution in pixel of the images, 700×700
- Render engine : "*Cycles*"
- Label names : [*antenna* , *body* , *solarPanel*]
- Focal length of the camera and other Blender's parameters like the minimum and maximum distance captured by the camera.

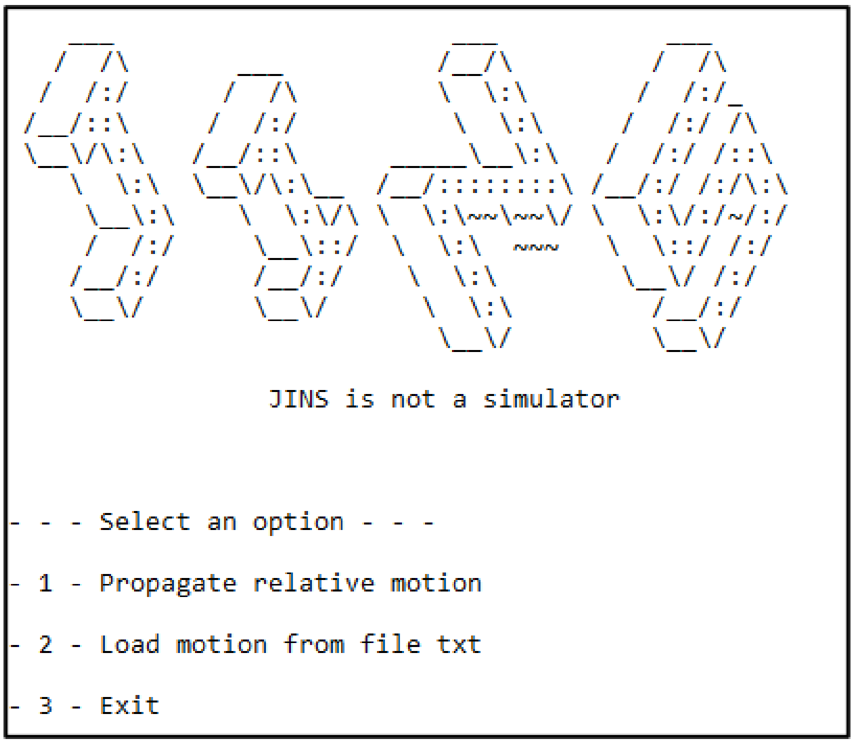
4.1. JINSv2

In the initial version of JINS, image generation was based on random lights and cameras generation, the presence of the Earth on the background was decided by the user, and the satellite's attitude was not considered. Also, to generate the scene using a different satellite model, one had to explicitly select it in the command line when the software was executed. Several features have been implemented in JINSv2 aimed at making the software user-friendly and capable of generating more realistic scenes.

4.1.1. User Interface

The first choice to make the software usable by others was to generate a simple but effective interface that would allow the user to be able to select which of JINSv2's functions to perform. Since the software is completely written in the Python[3] programming language, it was possible to use several libraries compatible with the Windows command line. An example of the initial user interface is shown in Fig. 4.4.

[2]Such as the one available at `https://github.com/csun/syntrain/tree/master/blender_scripts`
[3]Official website: `https://www.python.org/`

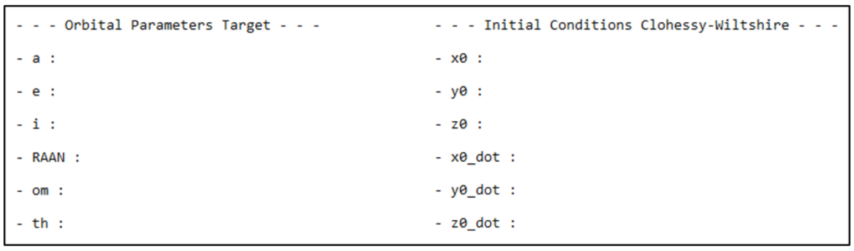


```
   ___                      ___           ___
  /  /\       ___          /__/\         /  /\
 /  /:/      /  /\         \  \:\       /  /:/_
/__/::\     /  /:/          \  \:\     /  /:/ /\
\__\/\:\   /__/::\      _____\__\:\   /  /:/ /::\
   \  \:\  \__\/\:\__  /__/::::::::\ /__/:/ /:/\:\
    \__\:\    \  \:\/\ \  \:\~~\~~\/ \  \:\/:/~/:/
    /  /:/     \__\::/  \  \:\  ~~~   \  \::/ /:/
   /__/:/      /__/:/    \  \:\        \__\/ /:/
   \__\/       \__\/      \  \:\         /__/:/
                           \__\/         \__\/

                  JINS is not a simulator



- - - Select an option - - -

- 1 - Propagate relative motion

- 2 - Load motion from file txt

- 3 - Exit
```

Figure 4.4: JINS: User Interface.

The user is led to choose between two main functions:

- **Propagate relative motion**: this option requires to provide to the software the Keplerian parameters of the target for its orbital propagation and the initial conditions of relative chaser-target motion. The two propagated orbits are used for the scene modeling in Blender. The window for data entry is shown in Fig. 4.5 :



```
- - - Orbital Parameters Target - - -          - - - Initial Conditions Clohessy-Wiltshire - - -

- a :                                          - x0 :

- e :                                          - y0 :

- i :                                          - z0 :

- RAAN :                                       - x0_dot :

- om :                                         - y0_dot :

- th :                                         - z0_dot :
```

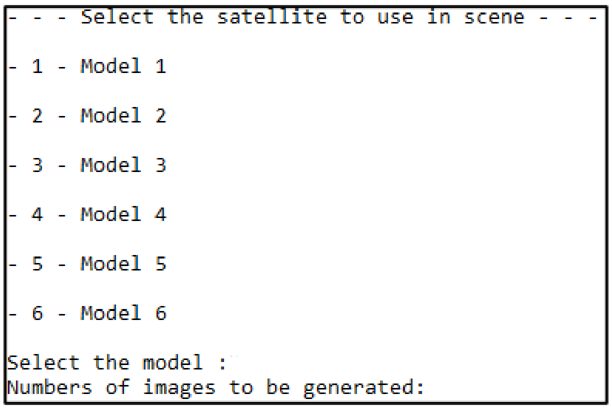
Figure 4.5: JINS : Option 1.

  No guidelines are provided for data entry because the user is assumed to have previous knowledge regarding Orbital Mechanics and the Clohessy-Wiltshire equations.

- **Load motion from file *.txt***: the user provides a *.txt* file containing a $N \times 6$ matrix, where N are the samples taken on both orbits at the same instant of time. The second dimension is 6 because 3 are the coordinates to describe the Earth-target vector in ECI frame and the other 3 for the target-chaser vector in LVLH frame.

- **Exit**: this option closes the software without performing any operation.

Once the coordinates of the target orbit and relative orbit are obtained, it is necessary to choose the satellite model to be represented in the scene. Unlike the previous version in which the model was chosen by selecting the *.blend* file on which to apply the image generation scripts, now the choice is made by the user through a selection menu, as in Fig. 4.6:



```
- - - Select the satellite to use in scene - - -

- 1 - Model 1

- 2 - Model 2

- 3 - Model 3

- 4 - Model 4

- 5 - Model 5

- 6 - Model 6

Select the model :
Numbers of images to be generated:
```

Figure 4.6: JINS : model selection.

Along with the model, the number of images to be rendered must be selected. The choice of the number of images to be generated may be important in cases where the sampling along the orbit is too dense and only a few images are needed, or in cases where the orbit is available but only a certain time interval is of interest.

4.1.2. Scene generation

One of the most obvious changes in JINSv2 from its previous version is scene generation. In JINS there was no real scene construction: the satellite model was placed in the center of the Blender scene, the cameras were randomly generated, and the Earth in the background was generated by overlaying the satellite scene with one containing only the Earth. The scene containing the Earth is generated from external Blender file. In order for the rendered scene to be as faithful as possible to an inspection mission a single Blender scene containing the target, the chaser and the Earth has been used. One of the most difficult aspects of generating a scene containing these objects is the difference in order of magnitude between the Earth and satellite model dimensions. The *Units* section of Blender allows the user to set some scene parameters to help in solving this type of problem. Since the values of the coordinates are much larger than the size of the satellite, it was decided to set the *Length scale* of the Blender scene equal to 1000 in order to contain the size of the scene. To faithfully represent the scenario of an inspection mission, the scene was composed as illustrated in Fig. 4.7.

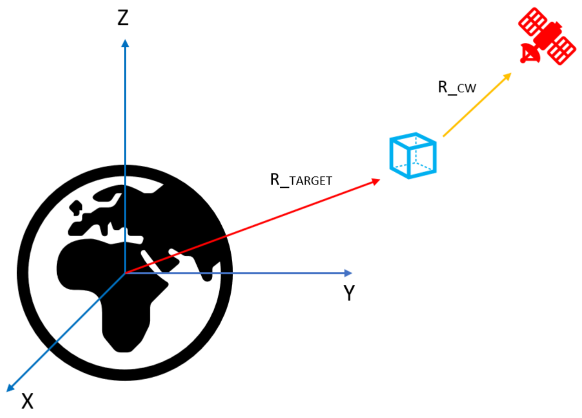


Figure 4.7: JINS: scheme scene.

A scene setup of this type requires knowledge of the rotation matrices between the various reference systems 'Blender frame - ECI frame' and 'ECI frame - LVLH frame' which can be easily determined. A problem with this configuration is the presence of small meshes, the satellite, far from the scene origin. Placing objects far from the origin causes precision errors in floating point numbers leading to warping and disruption of the meshes. This is due to the graphics capabilities of Blender very dependent on the GPU used to render the scene. An example of mesh problem is shown in Fig. 4.8.

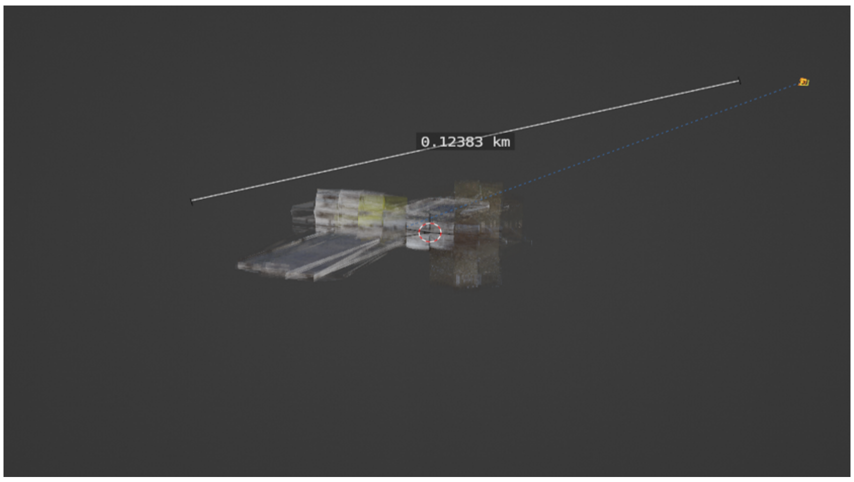


Figure 4.8: JINS: warping mesh problem.

To overcome this problem, the scene was constructed by placing the satellite model in the origin of the scene and using the LVLH coordinate frame. For simplicity it has been assumed that the Blender reference system matches the one of the Clohessy-Wiltshire equations thus avoiding the determination of the rotation matrix between the two reference frames, by doing so the Earth model is always located on the *Altitude* axis. The scheme of the new structure of scene is illustrated in Fig. 4.9.

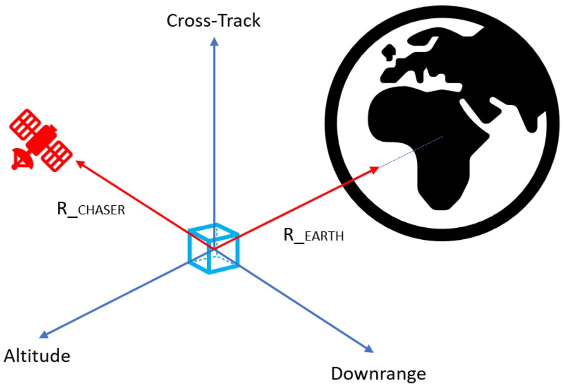


Figure 4.9: JINS: new scheme scene.

R_{chaser} has the same value as the coordinates obtained through the propagation of the Clohessy-Wiltshire equations, while R_{Earth} has only Altitude component equal to the norm of R_{target} in ECI coordinate frame. Once the scene settings, satellite model, and number of images to be rendered have been defined, it is possible to create a set of images that can be used to generate the final dataset. Images alone cannot be used by Machine Learning algorithms for training because they are not annotated.
Before proceeding to annotate the images, it is necessary to generate the masks, known as *ground truth*, of each component. Ground truths can be considered the exact solution of the Machine Learning model, in the case of computer vision they correspond to the segmentation that the model should achieve. Thanks to Blender's compositing mode, it is possible to generate masks for each element in the image. This mode, through the use of *nodes*, allows to do post-processing on the rendered scene such as overlaying different scenes, displaying only certain elements or applying filters. Similar to image generation, the creation of the masks can also be done either manually or automatically through scripting. Also in this case, Pyhton was used for mask generation in order to make the procedure automatable, since the number of nodes to be generated is dependent on the number of labeled elements contained in the scene. The compositing scene generated by JINSv2 is depicted in Fig. 4.10.

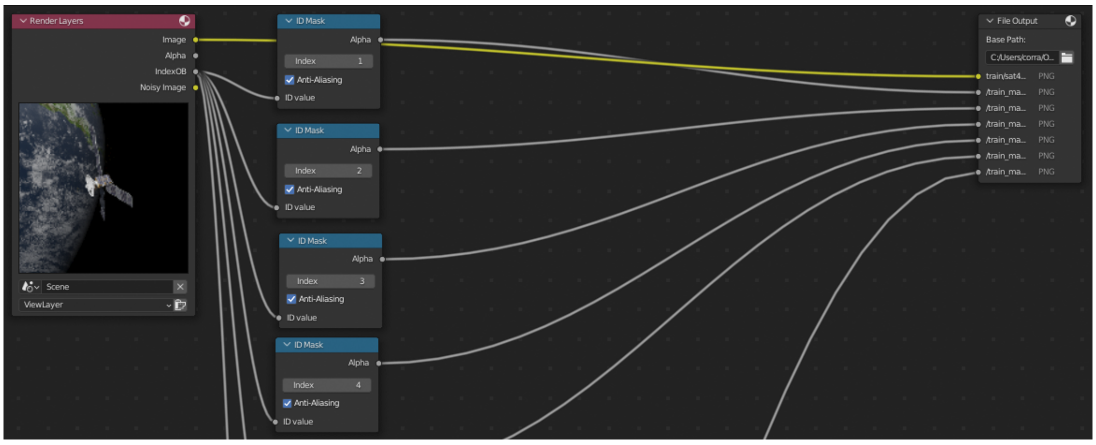


Figure 4.10: JINS: example of Compositing scene.

As it can be seen from the image, mask generation is based on the *ID Mask* node which allows to produce an alpha mask per object. The number of these nodes depends on the amount of mesh-type components in the scene that have the label equal to one among those given as input to JINSv2. The mask generated is white where the object is located and black where it is not. In case some parts of the object are transparent the mask assign to them grey values. To obtain masks as accurate as possible, the *Anti-Aliasing* filter was used to generate smooth edges. An example of rendered image with its masks is provided in Fig. 4.11.

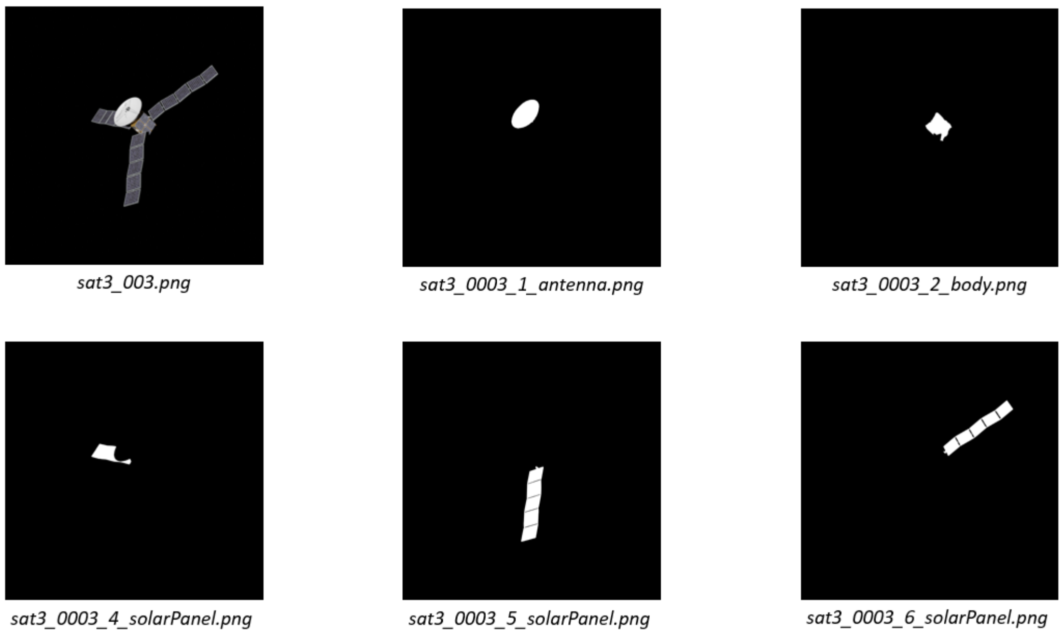

sat3_003.png
sat3_0003_1_antenna.png
sat3_0003_2_body.png
sat3_0003_4_solarPanel.png
sat3_0003_5_solarPanel.png
sat3_0003_6_solarPanel.png

Figure 4.11: JINS: example of generated masks.

To summarize how JINSv2 works its pipeline is reported in Fig. 4.12.

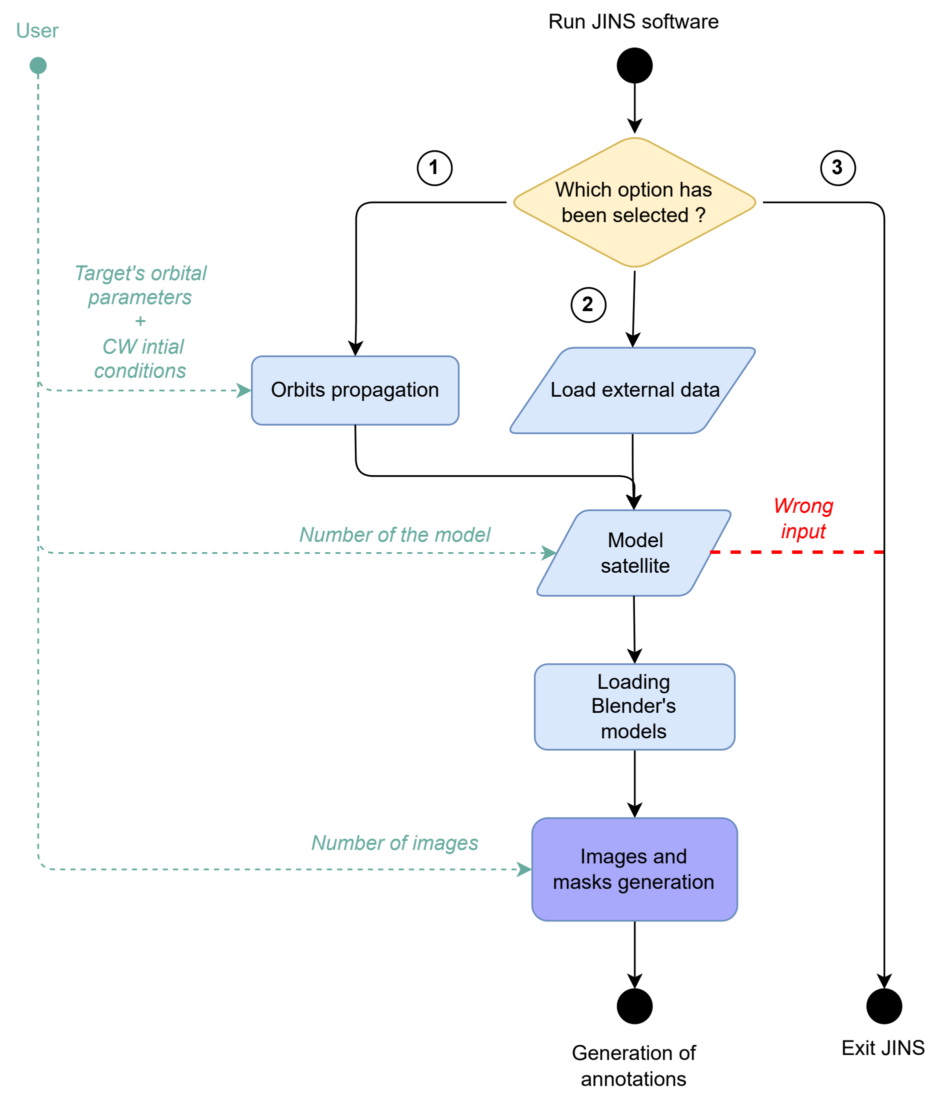


Figure 4.12: General flowchart of JINSv2 operation.

Some considerations should be made about the general flowchart of JINSv2. The *Load external data* block has no direct output from the user since the path from which to load data was assigned directly in the script. If it is desired to change the path it is necessary to edit the script directly. This aspect can be kept in mind for later versions of the software by allowing the user to directly select the file of his choice. In the *Model satellite* block, the user is forced to choose from the six models proposed by the software, and in the case of wrong input JINSv2 closes. Also in this case, JINS can be refined by allowing the use of a custom model, in addition to the six available satellites. In order for the custom model to be used, it must meet certain specifications necessary for the software to work properly. The selected file must contain only the satellite model in *Mesh* format, the central part of the model must have "*Body*" as its name, and the component labels must be the same as those used by the functions for generating masks. For completeness, a schematic representation of the image and mask generation process is shown in Fig. 4.13.

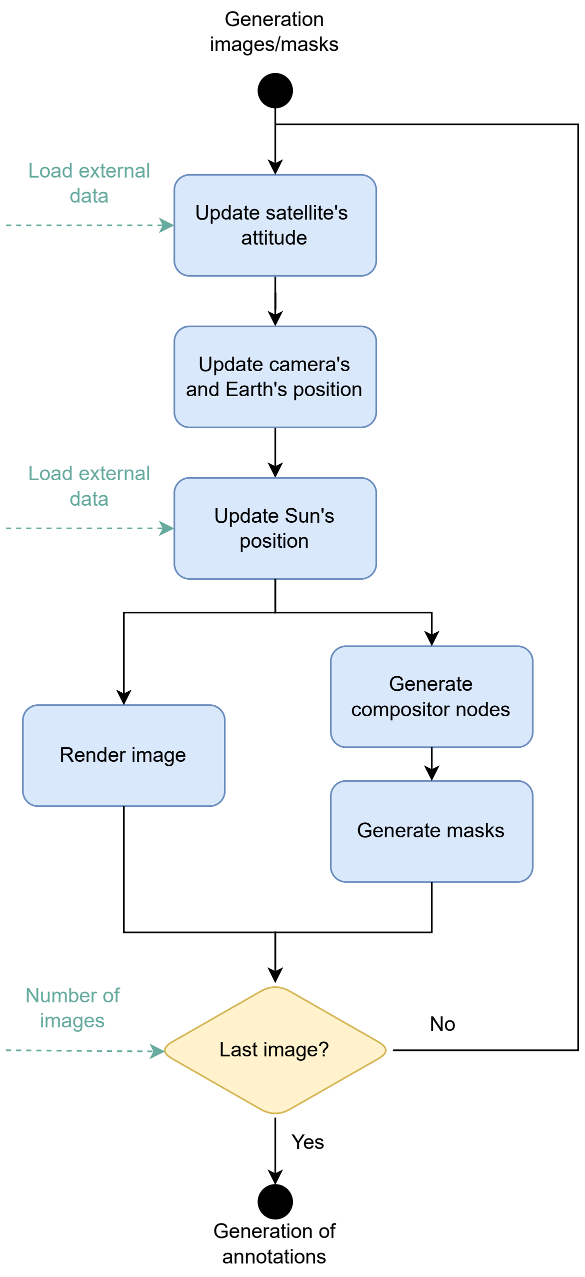


Figure 4.13: Images and masks flowchart of JINSv2.

Two points that have not been covered previously are present in the diagram, *Satellite's attitude* and *Sun's position*. To describe the satellite's attitude, it is possible to load an external file by editing the relevant variables in the script, since loading via a UI has not yet been implemented.

Blender allows to define rotation in three different ways:

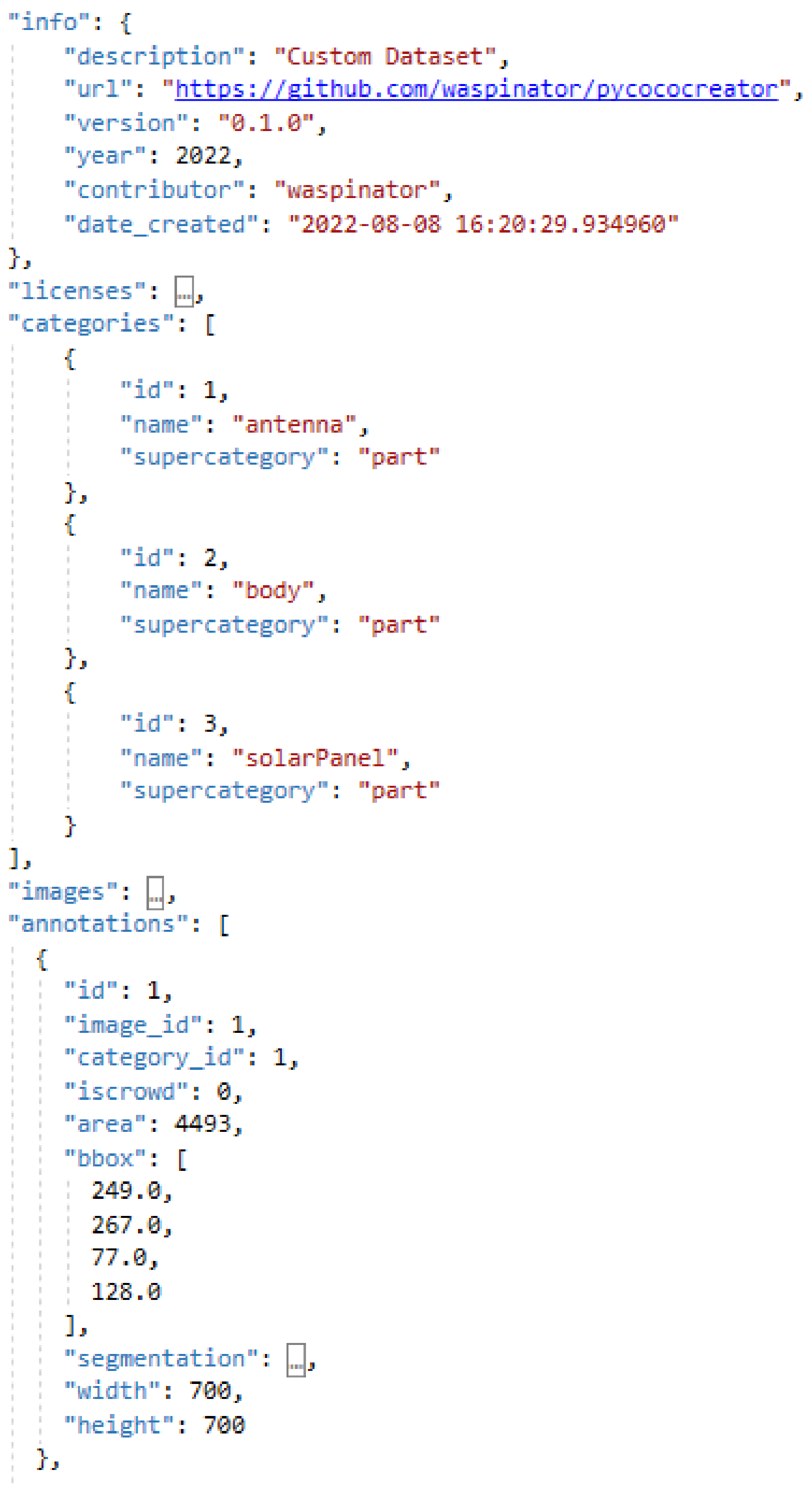
- *Euler modes*: used to perform rotation using a particular set of three axes, which have a hierarchical relationship between them.
- *Axis-angle mode*: let define and axis of the Blender coordinate frame and a rotation angle around that axis.
- *Quaternion mode*: the rotation is defined by four values; X,Y and Z define an axis and W an angle. The relation between them is important.

In this work it was decided to use *axis-angle mode* and assign random values to each of the three rotation axes. The choice is dictated by the fact that in the generation of a training dataset it is necessary to include the greatest amount of casualties, in this case viewpoint variation. Being able to implement the attitude may be useful for the creation of a dataset to verify the performance of the model in the case of a mission where the evolution of the debris is known. The same kind of observation is made for light generation in the scene. To generate light objects on Blender, it is possible to choose from several models that differ in how they propagate lights. To faithfully represent reality, an Sun-type model was chosen, which illuminates the scene with rays parallel to a certain direction.In particular, a constraint has been imposed which forces the lighting direction to be parallel to the Sun-Earth direction. The position of the Sun is generated by an external script by entering the date of the inspection in MJD2000 format. As in the case of the satellite's attitude, randomly generated dates were used to have different directions of illumination.

4.2. COCO format annotation file

Once the images and masks are generated it is possible to start creating the annotation file. There are several ways to annotate a dataset, the one adopted in this work is the COCO format, which stands for Common Object in Context. It is one of the most popular format of annotation for object detection, segmentation and captioning. The function that allows to generate annotations in COCO format is called *satellitetoCOCO.py*, and it is an external function not implemented within JINSv2. It needs the API[4] from COCO to work. The script cycles through all the files containing the rendered scenes, and based on their names the corresponding scene masks are associated. The output of the script is a *.json* file containing information about the different classes of components, for each image. The generated file is structured as shown in Fig. 4.14.

[4]COCO API is not directly available for Windows, a fork could be used to install it: `https://github.com/philferriere/cocoapi`. The Matlab and Python APIs are complete, the Lua API provides only basic functionality.



```
"info": {
    "description": "Custom Dataset",
    "url": "https://github.com/waspinator/pycococreator",
    "version": "0.1.0",
    "year": 2022,
    "contributor": "waspinator",
    "date_created": "2022-08-08 16:20:29.934960"
},
"licenses": […],
"categories": [
    {
        "id": 1,
        "name": "antenna",
        "supercategory": "part"
    },
    {
        "id": 2,
        "name": "body",
        "supercategory": "part"
    },
    {
        "id": 3,
        "name": "solarPanel",
        "supercategory": "part"
    }
],
"images": […],
"annotations": [
  {
    "id": 1,
    "image_id": 1,
    "category_id": 1,
    "iscrowd": 0,
    "area": 4493,
    "bbox": [
      249.0,
      267.0,
      77.0,
      128.0
    ],
    "segmentation": […],
    "width": 700,
    "height": 700
  },
```

Figure 4.14: Structure of the *.json* annotation file.

The *info* section contains high level information such as the dataset name, date of creation, and its version. The *categories* section lists all the classes contained within the dataset by associating each of them with an *id* and the *supercategory* to which it belongs. The *images* section contains informations of all the images such as name, dimensions, data and url in case it was taken from the web. The most important section, as well as the part needed by the algorithm for training, is the *annotations* section, that contains:

- *id*: each annotation has its own id.
- *image_ id*: denotes to which image the annotation corresponds.
- *category_ id*: denotes to which category the annotation corresponds.
- *iscrowd*: its value is 0 if the segmentation is for a single object or 1 in case of group of objects.
- *area*: area occupied by the segmentation measured in pixel.
- *bbox*: it gives information about the bounding box in the format [*top left x position*, *top left y position*, *width*, *height*].
- *segmentation*: is the most important part of the dataset and contains the list of coordinates (x,y) that define the contour of the object.
- *width*: width of the image in pixel.
- *height*: height of the image in pixel.

The segmentation is done using the function *approximate_ polygon* of the package *scikit-image*[5], library specific for image processing in Python. It approximates the binary mask generated by JINSv2 using the Douglas-Peucker algorithm [27]. The approximated polygon depends on the value of tolerance passed to the function, it indicates the maximum distance from the original points of the polygon to approximated polygonal chain. If tolerance is 0 the original polygon coordinates are returned. Another parameter that has to be set before proceeding with the creation of annotations is the *area threshold*, meaning that if the segmented area is less than a certain value for a certain component, that instance is not considered. To verify that the generated annotation exactly represents the indicated component, it is possible to use *BBox.py*. The function reads the information written in the *.json* annotation file and draws bounding boxes and segmentations of each component over the image, as shown in Fig. 4.15.

[5]API : https://scikit-image.org/docs/stable/api/skimage.measure.html

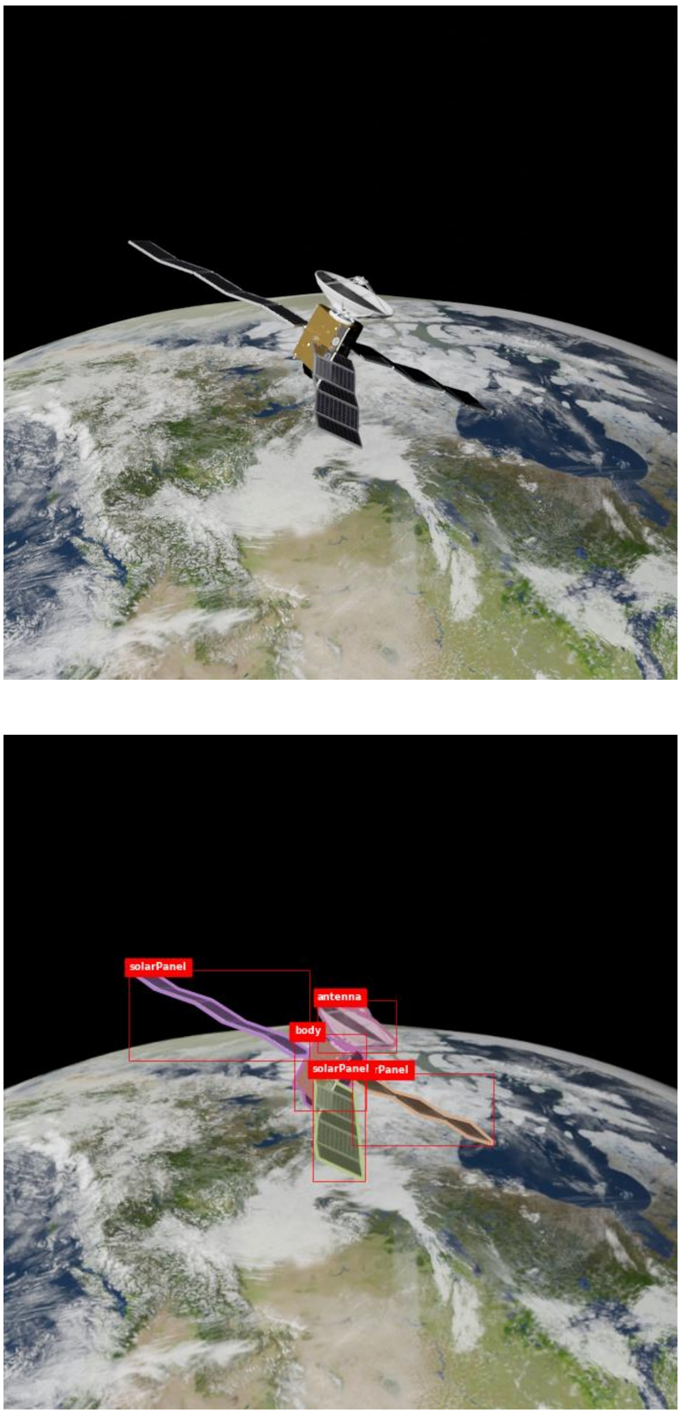


Figure 4.15: Rendered scene (top) and same scene with overlaid bounding boxes and segmentations (bottom).

5 | Results

The previous chapters discussed the architecture and pipeline of the JINSv2 software and the several Machine Learning models used in the field of computer vision. This chapter discusses the training of the model chosen in Chapter 2 and the various tests performed on it to assess its performance.

Before moving on to the actual results part some information about PointRend and the operating environment are given. Only one implementation of PointRend[17] is available online, which is that of *Detectron2*[1][41] provided by the Facebook AI Research (FAIR) group. Other than instance segmentation, PointRend can be applied also for semantic segmentation tasks by building on top of existing state-of-the-art models. Using Detectron2 not only is provided the head to generate the masks but also the model of Faster R-CNN for object detection. The only available base model provided by Detectron2 Model Zoo[40] is the *R50-FPN-3x*. It uses a ResNet backbone architecture with 50 layers, a Feature Pyramid Network (FPB) for the box prediction and trained with a 3x schedule. One of the parameters to be taken into account before the training of the model is the size of the image contained in the dataset. It is recommended to use images between 600 and 800 pixels to obtain good results, so a square images of size 700px have been used. Detectron2 can be built on different platforms, both local and online. The important thing is that they are equipped with GPUs for the training phase.

In this work Google Colaboratory has been used, it is a free Jupiter notebook[2] environment equipped with several GPUs. Using the free plan enables the use of standard GPUs for a limited daily period of time. If one wants to use higher-performance GPUs for a longer period of time, it is possible to take advantage of a premium subscription. To assess the performance of the model different types of training and testing were carried out. Since the training process is highly dependent on the GPU used, an attempt was made to use the same one in order to obtain trainings comparable to each other, of all those available, the TESLA T4 16 GB was used. Not only the same GPUs have been

[1]Detectron2 is Facebook AI Research's next generation library that provides state-of-the-art detection and segmentation algorithms. It supports a number of computer vision research projects and production applications in Facebook.Website : `https://github.com/facebookresearch/detectron2`

[2]Official website : `https://jupyter.org/`

used, but also different parameters that greatly affect the duration and accuracy of the training process. The parameters adjusted for training are the learning rate, the number of iteration, the number of workers and the images per batch. As explained in Chapter 2 the learning rate is the hyperparameter that controls the change of the weights in the model. During the training phase it was set to 0.003, while the number of iteration to 3000. The remaining two are parameters that manage the flow of data within the network. The number of workers, which is set to 2, tells to the dataloader how many subprocesses to use for data loading, while 2 images per batch are considered. The latter in particular is very important for the environment in which it is working, since very large images per batch can cause *out of memory* errors, it is always recommended to increase the number of iterations instead of batch size.
Before describing the various tests performed on the model, it is necessary to give an explanation of the metrics used to evaluate the results. Several evaluation methods exist in machine learning, the most common and the ones used in this work are the Average Precision (AP) and the mean Average Precision (mAP). To evaluate the AP, it is necessary to define some variables [26]:

$$\text{Precision :} \qquad p = \frac{TP}{TP + FP} \tag{5.1a}$$

$$\text{Recall :} \qquad r = \frac{TP}{TP + FN} \tag{5.1b}$$

where *TP* are the true positives which are the object correctly detected, FP are the false positives, i.e. the object is detected correcly but not its class, and the FN are the false negative which are the number of object not detected. The *precision* is an index of how accurate the prediction is, while the *recall* measures how well all the positives are found. From the above definitions it is possible to determine the AP :

$$AP = \int_0^1 p(r)dr \tag{5.2}$$

The AP is calculated for each class and then averaged to obtain the mAP, sometimes they are considered the same thing as in the case of the COCO challenge evaluation. The definition of AP does not depend only on these two parameters; sometimes it may depend on a third parameter called Intersection Of Union (IoU), depending on the assessment mode. This parameter is used to evaluate how well the predicted box overlaps the one of the ground truth. Considering the example shown in Fig. 5.1, the IoU can be calculated as:

$$IoU = \frac{\text{Area of intersection of the boxes}}{\text{Area of union}} \tag{5.3}$$

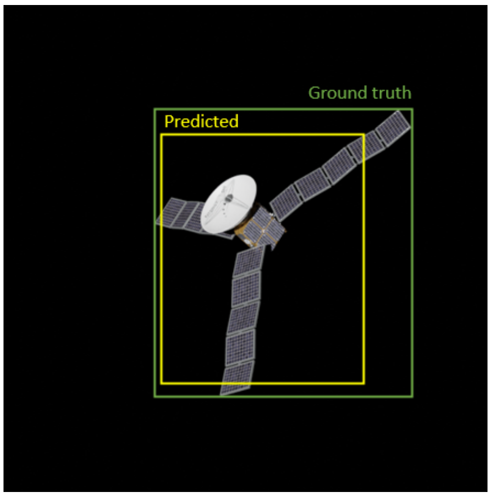


Figure 5.1: Example of box predicted.

The definition of IoU is very important because it allows to define which prediction is a true positive or a false positive. In most of the dataset if the IoU is ≥ 0.5 the detection is considered a true positive while if IoU < 0.5 then is a false positive. Since the dataset generated by JINSv2 is in COCO format, the COCO evaluation metrics [4] has been used. It consists in 12 metrics for characterize the performance of the object detector, only the ones relative to the average precision are considered:

- AP : AP at IoU = .50:.05:.95 (primary challenge metric)
- $AP^{IoU=.50}$: AP at IoU=.50 (PASCAL VOC metric)
- $AP^{IoU=.75}$: AP at IoU=.75 (strict metric)
- AP^{small} : AP for small objects : area$<32^2$
- AP^{medium} : AP for medium object : $32^2<$area$<96^2$
- AP^{large} : AP for big objects : area$>96^2$

5.1. Training and testing on a single model

The first test performed on PointRend was to train and test it on a dataset containing only one satellite model. The dataset used contains 360 images of satellite model 1, and the labels considered are *antenna*, *body* and *solarPanel*. To include a greater variety of images, scenes with Earth completely in the background, with Earth partially in the background, and without Earth in the background were considered, as reported in Fig. 5.2

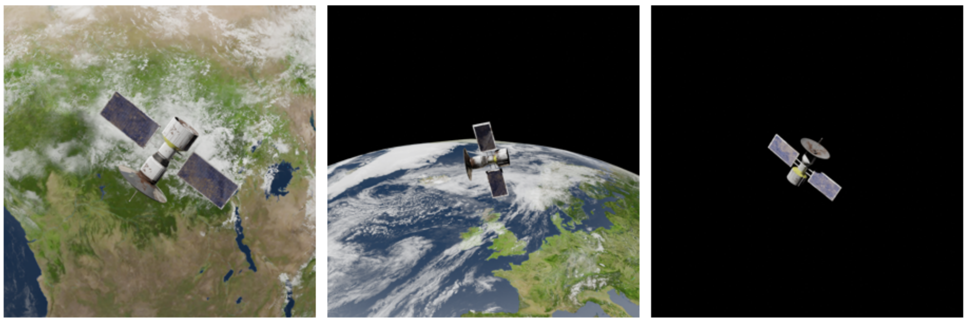


Figure 5.2: Example of images contained in the first dataset.

The time required for the training is about 40 minutes. The results are shown from Tables 5.1 to 5.4 which indicate the overall AP and the APs for each class for both object detection and segmentation.

AP	AP50	AP75	APs	APm	APl
86.212	97.910	95.515	79.273	90.984	92.329

Table 5.1: Evaluation results for bounding box

AP antenna	AP body	AP solarPanel
81.485	89.134	88.017

Table 5.2: Per-category bounding box AP

AP	AP50	AP75	APs	APm	APl
76.496	93.815	87.270	55.510	87.293	100

Table 5.3: Evaluation results for segmentation

AP antenna	AP body	AP solarPanel
72.213	78.854	78.420

Table 5.4: Per-category segmentation AP

An example of the inference is shown in Fig. 5.3.

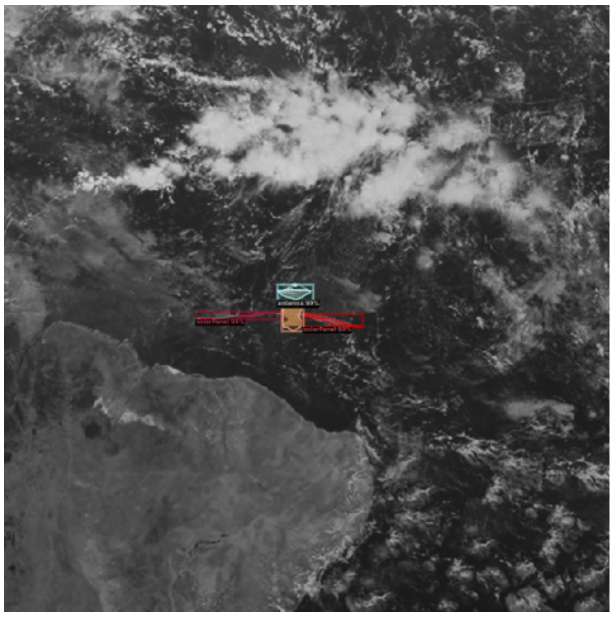


Figure 5.3: Inference of the model trained on only one satellite.

After inference, the image was converted to grayscale to make the performed segmentation more evident. Looking closely it is possible to see that the left solar panel has been identified correctly but its mask does not cover it entirely. Looking at the tables, it can be seen that the AP of segmentation for large objects is high. It was thought to improve the segmentation accuracy of small objects by going to consider large objects in the background as well. In the case under observation, if the model achieves high accuracy in identifying and segmenting the Earth in the background then the accuracy in recognizing small objects may increase. A new training was carried out, without changing neither the parameters nor the dataset but only a class "*Earth*" to be identified was added. The results of the new training are reported in Table 5.5 and an example is shown in Fig. 5.4.

	AP	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
PointRend					
Bbox	86.580	78.632	83.106	84.952	99.630
Segm	80.260	69.600	75.860	75.747	99.831

Table 5.5: Training considering the *Earth* class

As expected, the accuracy of small component segmentation has increased. The left solar panel that was previously not fully segmented is now identified correctly and continuously. This type of test is not only used to verify the performance of the trained model but could

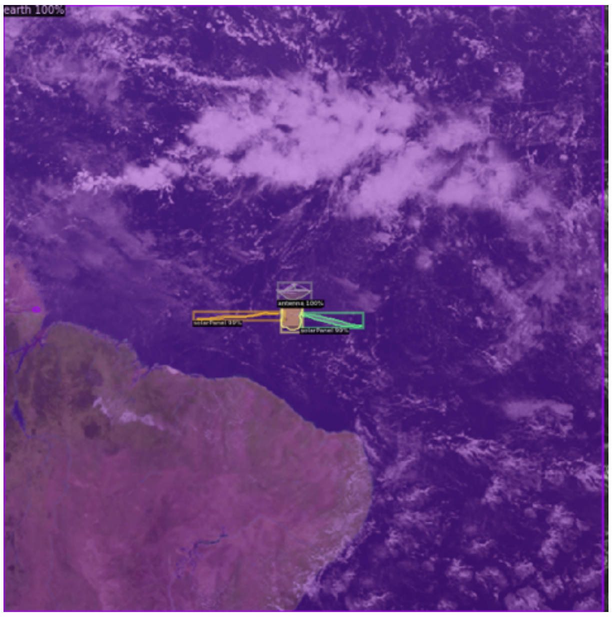


Figure 5.4: Inference of the model trained on only one satellite considering *Earth* class.

be useful in the case where during an inspection mission the RSO geometry is known and can be used to train the algorithm. Purpose of the work is not only to find a new model for instance segmentation but also to achieve better performance than the state-of-the-art Mask R-CNN model. For this reason, in parallel to PointRend , Mask R-CNN was trained using the same configurations and dataset. The results for Mask R-CNN are reported in Table 5.6.

	AP	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
Mask R-CNN					
Bbox	78.311	62.184	72.639	78.796	99.625
Segm	72.000	59.040	62.963	68.407	97.591

Table 5.6: Training considering the *Earth* class

The improvement using PointRend is significant, an increase of about 10 AP points for both the object detection and segmentation tasks. This increase might seem small in the eyes of a non-expert, but even a 2 point AP increase makes a new algorithm model state of the art.

5.2. Testing on out-of-dataset satellite

Once the performance of PointRend has been verified on one model, it is necessary to perform further tests to see whether the model performs properly under different conditions. The next step is to test the model on a satellite not contained in the dataset. This condition very much reflects reality since the geometry of the RSO to be observed is not always available. A dataset containing 5 models was used as a training dataset, while the remaining satellite model (Model 5) was used to generate a testing dataset. Specifically 4 types of datasets were generated for training of the model:

- No Earth in background
- Earth in background
- Earth in background in grayscale
- Earth in background with Poisson noise

All of these combinations of datasets are used to cover as many scenarios as possible and to test the robustness of the selected model.

5.2.1. Dataset with no Earth in background

The model was trained on scenes containing no Earth in the background. This was done to verify the performance of the model in the case where no disturbance element is present in the scene. The dataset used for the training contains 750 images of all the five models, the training require about 40 minutes for both PointRend and Mask R-CNN. After training, the model was tested on a testing dataset containing the same scene type as the training dataset but with satellite model number 5. The results are reported in Table 5.7.

	AP	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
PointRend					
Bbox	69.372	68.703	59.170	80.243	nan
Segm	64.602	55.652	67.880	70.270	nan
Mask R-CNN					
Bbox	66.091	64.893	53.911	79.468	nan
Segm	62.141	54.110	66.044	68.268	nan

Table 5.7: Train no Earth in the background - test no Earth

This type of test is not sufficient to test the performance of the model in case of an inspection mission. During a real inspection the chaser will have periods when the Earth is present in the background, so further testing was performed using a testing dataset that contains images of satellite 5 with the Earth in the background. This type of test is to see how the model behaves when it encounters a satellite it has never seen in a new scenario. The results for both PointRend and Mask R-CNN are reported in Table 5.8.

	AP	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
PointRend					
Bbox	27.586	25.753	30.519	54.072	0
Segm	23.231	20.100	23.522	49.3020	0
Mask R-CNN					
Bbox	29.006	26.947	37.071	52.005	0
Segm	24.196	16.021	29.571	51.193	0

Table 5.8: Train no Earth in the background - test with Earth

As expected the AP values obtained by testing the model with scenes containing the Earth are significantly lower than those of the model tested with scenes without Earth in the background. This difference in AP is due to the fact that the model was trained on a dataset containing no scene with the Earth in the background, so the presence of a scenario never seen during the training phase dramatically decreases the accuracy of the prediction. Unexpectedly, higher AP values have been obtained with Mask R-CNN using the testing dataset with the Earth in the background. This result may be caused by the intrinsic nature of PointRend, where more attention is paid to the edges of the object. The accuracy of the models trained without Earth in the background decreases when they perform the inference on a scene where the Earth is present. This is caused by the significant increase of details in the scene, which leads the models to show uncertainties along the boundaries of the objects and thus to perform incorrect predictions.

5.2.2. Dataset with Earth in background

Training PointRend without considering the Earth in the background makes the model sensitive to the details in the scene by decreasing its performance in comparison with the standard Mask R-CNN. A new dataset containing new scenes with the Earth in the background was generated to see if using a different scene topology improved model performance. The training dataset used contains 750 images of the same satellite models used in the previous test. To obtain a fair comparison between the two training the hyperparameters of the enviroment were not modified. In this case the training took about 53 minutes in contrast to the 40 minutes required for the previous test. This may be because the scenes are more complex than the previous ones and the model must also be able to identify the *Earth* class. The AP values for both PointRend and Mask R-CNN are reported in Table 5.9.

	AP	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
PointRend *No Earth in background*					
Bbox	68.675	69.219	54.059	82.747	nan
Segm	63.336	57.337	63.209	69.463	nan
Mask R-CNN *No Earth in background*					
Bbox	64.503	64.955	50.458	78.097	nan
Segm	60.107	51.423	63.607	66.290	nan
PointRend *Earth in background*					
Bbox	64.615	47.117	46.463	73.793	91.089
Segm	63.234	37.811	50.321	66.911	97.896
Mask R-CNN *Earth in background*					
Bbox	63.026	49.114	47.128	70.724	85.136
Segm	60.209	37.040	47.876	67.070	97.851

Table 5.9: AP values on models trained with Earth in the background

With this training PointRend shows higher AP values than Mask R-CNN in both test datasets. This may be due to the fact that since the model is trained on images full of details in the case of scenes with Earth in the background there are no more accuracy problems as in the previous case. In the case where the Earth is not present in the background there are less details than those encountered during the training phase so in this case the model has no problems during the inference.

Data augmentation

It is seen that by including images with a lot of details in the dataset, the model will not encounter problems during prediction. To faithfully represent an inspection mission a new dataset was generated by merging the two previously used datasets. As mentioned in previous chapters, machine learning models need a large number of images for training to achieve optimal results. To increase the number of images in the dataset the same images were taken using a second camera in Blender placed at a variable distance of ±10 meters from the actual position of the chaser, this is used to include the scale variation discussed in Chapter 4. To further increase the number of images in the dataset without resorting to generating additional images, *data augmentation techniques* were implemented. Data augmentation is a set of techniques used to artificially increase the number of images inside the dataset by adding slightly modified copies of an original image. There are endless ways to increase the number of data, the most used ones are shown in Fig. 5.5

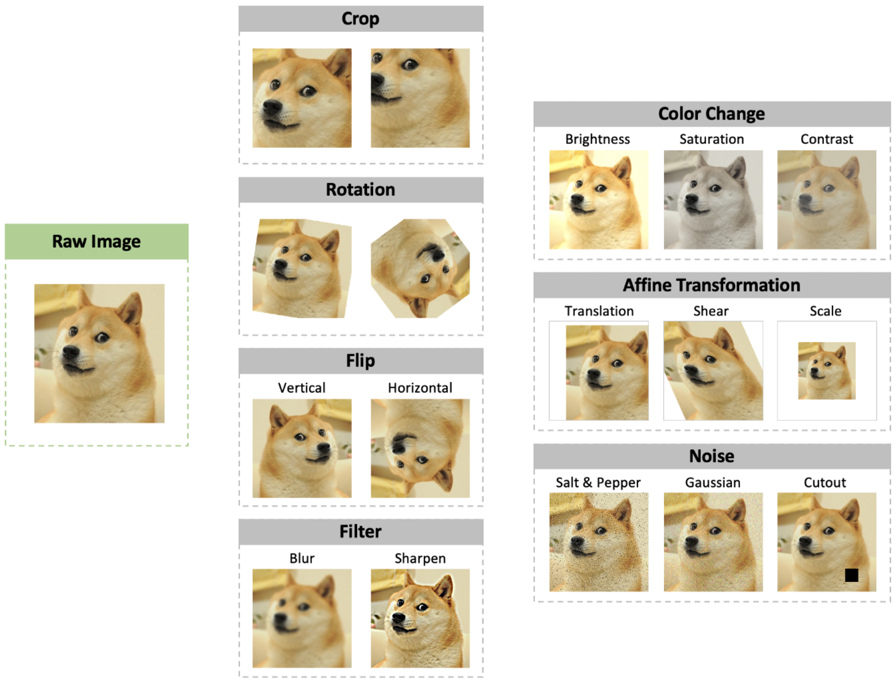


Figure 5.5: Example of data augmentation. Image credit

Among the several techniques shown above two in particular have been implemented, rotation and flip. A dedicated function called *Rotation.py* was written. It takes all the images in the dataset and performs three operations: flip and rotation of ±90°. Since these new images must also be annotated, the masks must also undergo these transformations in order to create the dataset correctly. The new dataset with the additional images obtained with data augmentation contains 9000 scenes. This large amount of images is very useful to the model during training but one thing has to be taken into account: when using Colab's free enviroment the time of GPU usage is limited, so one has to take into consideration that the more the images contained in the dataset the longer the time required for training with the same number of iterations. Always using 3000 iterations and the same hyperparameters, the training time increases significantly to 1 hour 43 minutes for PointRend while Mask R-CNN requires 1 hour and 30 minutes. Now the two methods require different training times differently from the previous trainings. This may be due to the slightly more complex architecture of PointRend, which requires more time due to all the images used for training. The trained model was tested as in the previous cases on both the dataset containing scenes without the Earth in the background and the dataset with the Earth in the background. The result of the inference are reported in Table 5.10 shown below.

	AP	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
PointRend *No Earth in background*					
Bbox	70.335	71.533	56.359	83.113	nan
Segm	65.121	58.887	65.207	71.582	nan
Mask R-CNN *No Earth in background*					
Bbox	66.092	65.462	51.831	80.897	nan
Segm	59.897	52.542	59.929	67.220	nan
PointRend *Earth in background*					
Bbox	77.284	69.907	57.872	85.898	100
Segm	71.284	56.531	67.201	71.112	100
Mask R-CNN *Earth in background*					
Bbox	65.393	49.951	45.100	75.898	100
Segm	64.284	40.515	53.907	69.720	98.847

Table 5.10: AP values on models trained with Earth in the background using data augmentation

As expected, due to the use of data augmentation and thus more images used in training the AP values for both PointRend and Mask R-CNN have increased. In the case of segmentation the difference in accuracy between the two methods was between 2 and 3 AP points, while now the difference has almost doubled to 6 AP points. The only unexpected result is to have obtained from Mask R-CNN a lower value of AP when data augmentation was used. The difference between the two trainings is about 1 AP, this leads to think that the problem is due to the training since it itself is an algorithm that uses a set of more or less randomly selected variables. One must take this random aspect of training into consideration when designing a real mission. It is recommended, whenever possible, to always carry out several trainings and choose the most accurate one. To show the difference in accuracy between the two methods, the inference was performed on a particular configuration where both solar panels are placed in profile, thus barely visible. The inference of both models are illustrated in Fig. 5.6.

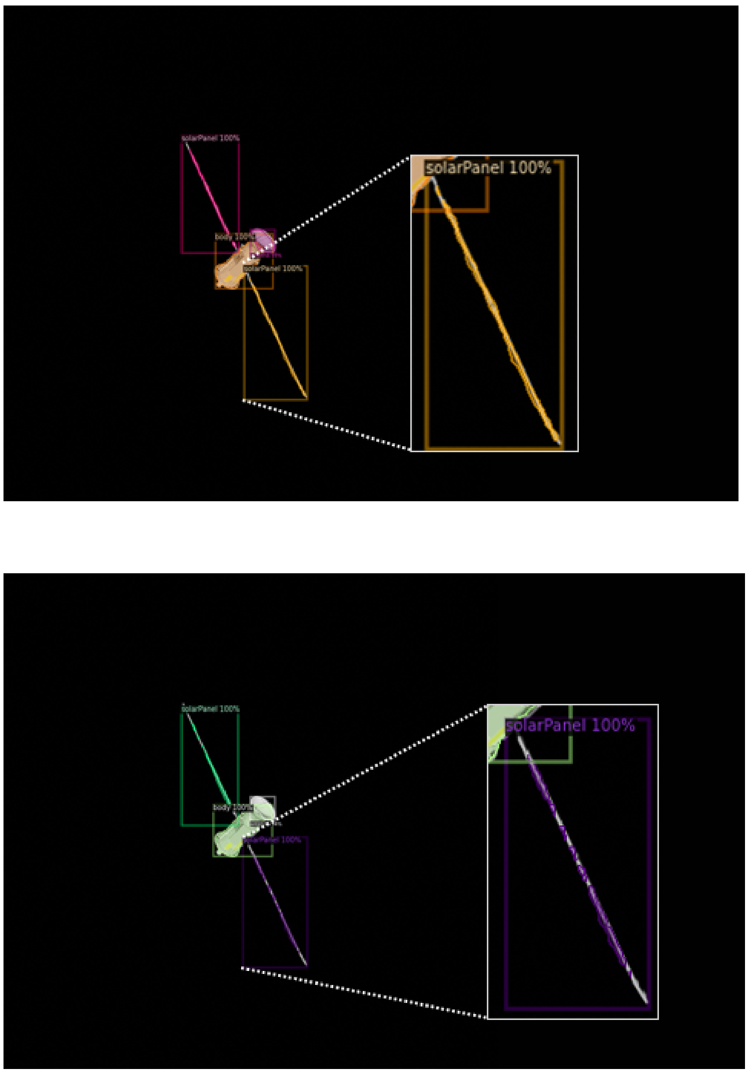


Figure 5.6: PointRend inference (top) and Mask R-CNN inference (bottom).

As confirmed by theory PointRend outputs masks with higher resolution than those provided by Mask R-CNN, increasing accuracy especially along object's edges. This increase in accuracy is very important for real missions because if the Machine Learning model recognizes the presence of an object but its mask is very discontinuous, the software that is going to use this data can interpret that result as an error in the prediction and thus neglects that area. To provide more clarity on how PointRend selects the points, a pattern of how the sampling points are arranged as the resolution of the mask increases is shown in Fig. 5.7.

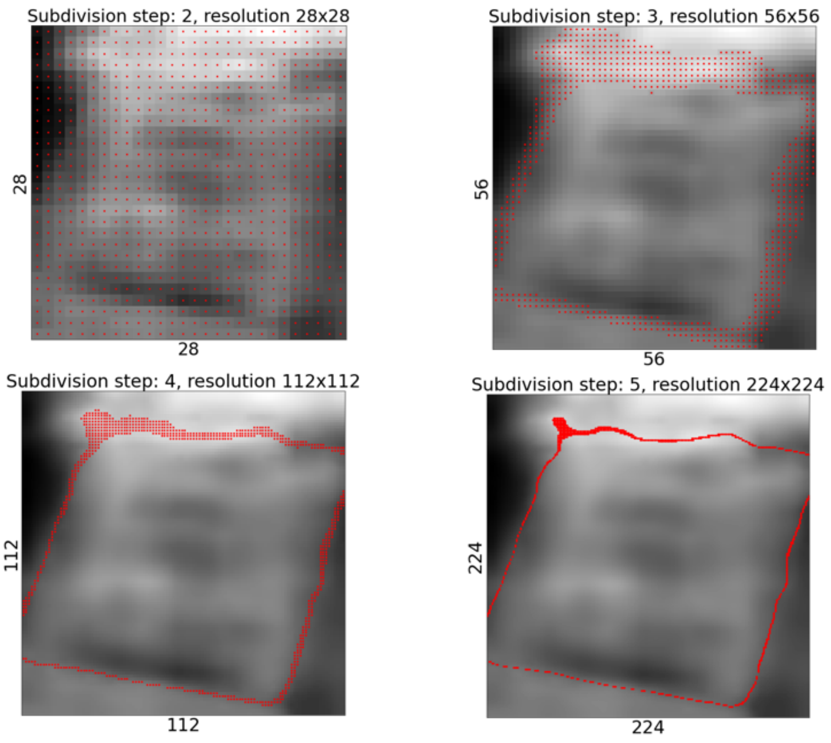

Subdivision step: 2, resolution 28x28
28
28
Subdivision step: 3, resolution 56x56
56
56
Subdivision step: 4, resolution 112x112
112
112
Subdivision step: 5, resolution 224x224
224
224

Figure 5.7: PointRend point selection example.

Results obtained from the trainings confirm that PointRend can also be used as a validated substitute for mask R-CNN for instance segmentation tasks by achieving higher accuracy in both object detection and segmentation.

5.2.3. Training with noisy images

So far the models have been trained with images generated directly from Blender without any processing done on them, these allows for sharp and high-resolution images. During a real mission, The images taken by the camera equipped on the chaser will never have the same level of resolution as those used so far for training and testing because there are disturbs such electrical noise and optical aberrations. To test the performance of the models in a scenario even closer to reality, noise was considered within the testing dataset. There are different types image noise but the most famous in the field of computer vision are *Gaussian noise* and *Poisson noise*. Gaussian noise is a noise of statistical nature that follows a Gaussian distribution. It is generated separately and independently and then added to the original image. The Poisson noise is a type of electronic noise that occurs when the energy carried by a finite number of particles , such as in our case the photons in the detector of the chaser optical device, is small enough to give rise to statistical variations detectable in a measurement. This means that Poisson noise is correlated with the intensity of each pixel while the Gaussian noise is independent from the original intensities of the pixels in the image. As described in [18] if the number of photons impacting the detector is very large, Poisson noise can be approximated to Gaussian noise. Assuming that the chaser does not take measurements when the Sun is near the field of view of the chamber to prevent the device from being damaged , the Poisson noise model is acceptable. An example of applying Poisson noise to an image generated by JINSv2 is shown in Fig. 5.8.

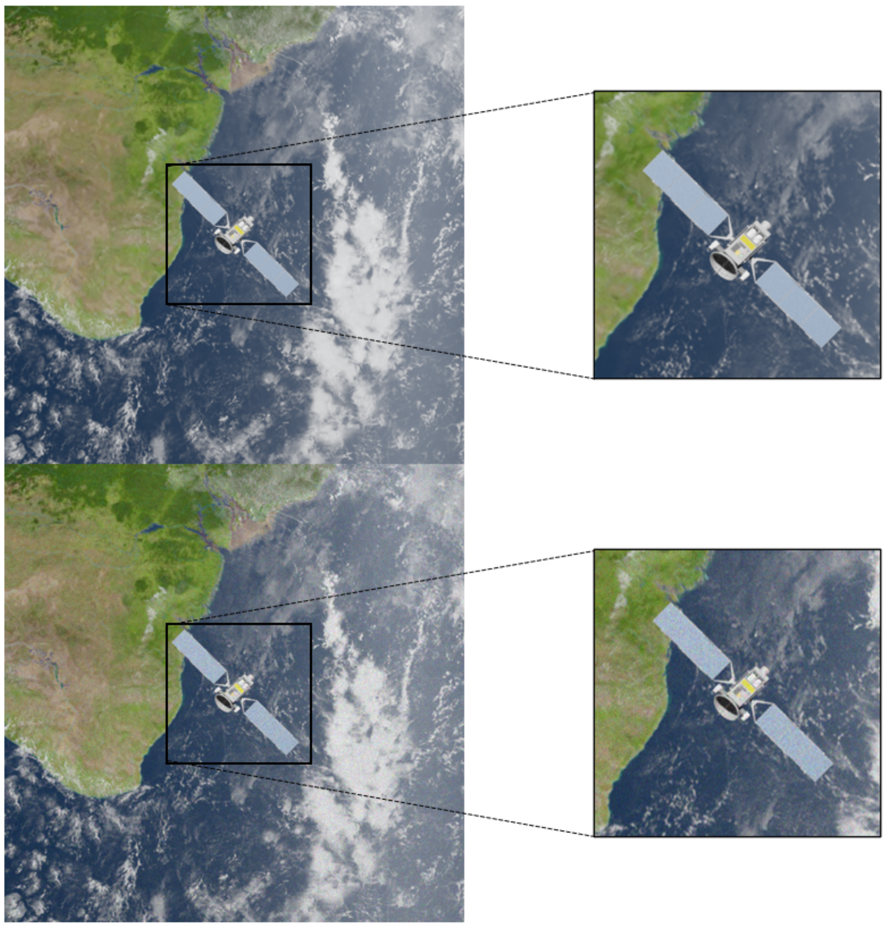


Figure 5.8: Original image (top) and image processed with Poisson noise (bottom).

The dataset containing the noise does not need to be generated all over again since the ground truth masks are not affected by it, so the annotations remain unchanged. Through a specially implemented function, *Noise.py*, it was possible to consider some Poisson noise on all the images already held. Training times are not affected by the presence of noise, remaining as in the previous case 1 hour 43 minutes for PointRend while 1 hour and 30 minutes fort Mask R-CNN. The results of the training are reported in Table 5.11

	AP	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
Bounding Box					
PointRend	69.393	37.951	48.100	71.898	98.625
Mask R-CNN	57.589	29.888	37.370	63.680	99.418
Segmentation					
PointRend	59.284	24.516	35.907	56.720	99.994
Mask R-CNN	50.306	22.679	33.064	47.557	97.925

Table 5.11: AP values on models tested on dataset with noise

As confirmed earlier PointRend achieves higher accuracy than Mask R-CNN even when tested on scenes containing noise. The overall AP decreased compared to training without noise, and simultaneously the difference in the segmentation AP between the two methods increased. Now the AP difference between the two methods reaches about 9 AP points while in previous cases it was between 6 and 7 AP points. This behavior is due to the inherent nature of the two methods. Mask R-CNN always returns an output with low resolution, so when noise is present in the image the generated mask will be more approximate since some corrupted pixels of the noise may be mistaken as the outline of the object to be identified. PointRend returns high-resolution masks, and in addition, due to its architecture, randomly chosen sampling points make the model less sensitive to noise so that object contour identification occurs more accurately than Mask R-CNN. An example of inference of both models on a scene when Poisson noise is considered is illustrated in Fig. 5.9.

At first glance, the two inferences appear to be similar despite the fact that the AP values obtained in Table 5.11 are very different from each other. However, if an enlargement is made, the differences between the two outputs can be seen. As described above Mask R-CNN returns a more approximate mask of the identified object. This can be seen in the *body* where the whole contour is approximated by going to consider only its central part. On the other hand PointRend by returning masks with a higher level of resolution

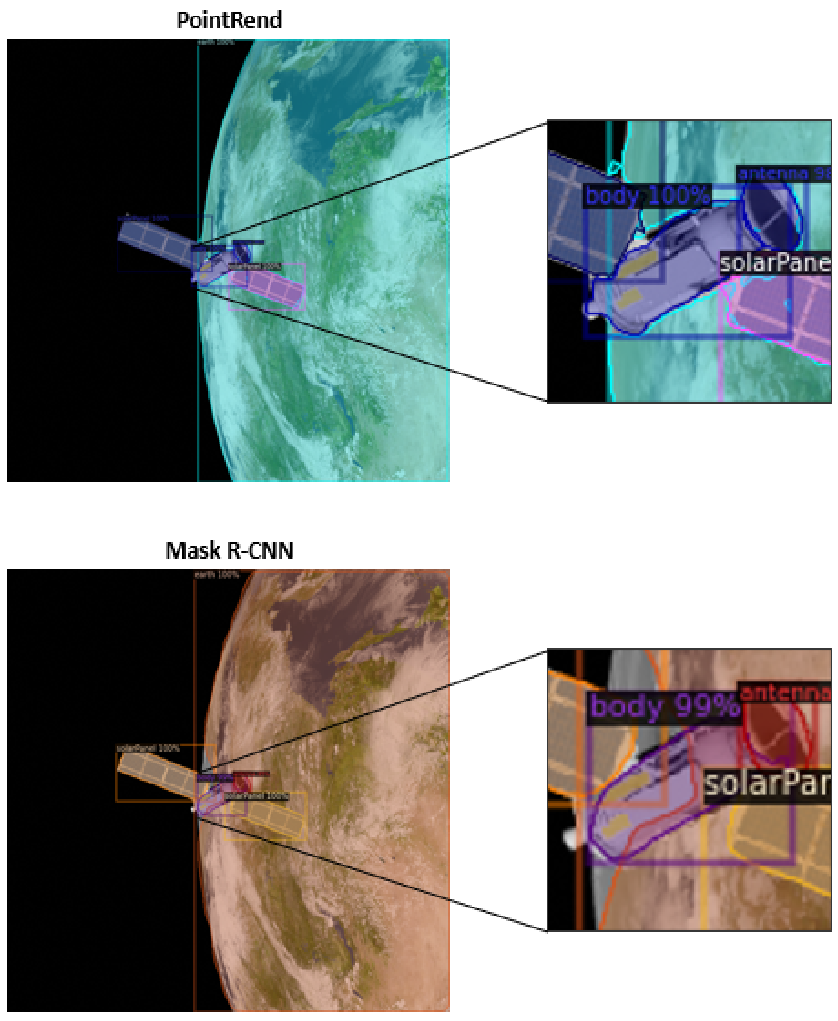


Figure 5.9: Inference on noisy images of models trained on dataset without noise.

succeeds in better describing the contour of the object. In fact in addition to distinguishing well the contours of the main part of the *body* it is also able to detect and segment the thruster on the bottom. Another evidence of the greater accuracy of PointRend is the segmentation of the Earth. Mask R-CNN succeeds in segmenting the Earth's contour well in areas far from the satellite, while in the vicinity of the satellite the accuracy drops dramatically by passing the mask contour inside the satellite and neglecting a part of the Earth between different parts of the satellite. PointRend succeeds in perfectly segmenting the contour of the Earth in areas far from the satellite while near the latter the contour of the segmentation mask follows its entire perimeter by going to consider all areas where even a part of the Earth is present.

As mentioned earlier the images that the chaser acquires are affected by noise. It was thought that training the model directly on images with noise could increase performance during a real mission. The use of noise within the dataset is sometimes seen as a form of data augmentation, but in this case the entire dataset contains images with noise. The training hyperparameters are identical to those used earlier while the dataset is composed of the same images as the previous ones to which Poisson noise is applied. The performances of the models were tested on all three types of datasets used in previous trainings, and the results are reported in Table 5.12 and Table 5.13.

PointRend	**AP**	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
No Earth in background					
Bbox	68.131	66.906	55.058	82.431	nan
Segm	63.890	57.180	63.478	71.013	nan
Earth in background					
Bbox	69.078	48.425	50.628	77.258	100
Segm	66.852	40.608	55.765	71.036	100
Earth + noise					
Bbox	70.997	47.241	56.289	80.458	100
Segm	65.385	38.409	52.015	71.114	100

Table 5.12: PointRend trained on noisy dataset

Mask R-CNN	**AP**	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
No Earth in background					
Bbox	66.157	67.608	52.058	78.805	nan
Segm	59.660	54.062	59.543	65.375	nan
Earth in background					
Bbox	66.909	45.953	49.565	72.119	100
Segm	65.653	39.359	50.829	72.770	99.654
Earth + noise					
Bbox	67.711	42.126	50.989	77.832	99.896
Segm	62.477	33.829	47.044	70.791	98.246

Table 5.13: Mask R-CNN trained on noisy dataset

Both models tested on dataset without noise return lower AP values than when trained and tested on dataset without noise, as expected. A noticeable improvement occurs on the testing dataset with noise: AP values increase by almost 10 points. An example of inference on a scene with noise is shown in Fig. 5.10.

As already found in the previous test Mask R-CNN has problems in segmenting the Earth contour when overlapped with the satellite while PointRend does not suffer from this problem. Unexpected is the fact that despite higher AP levels in the object detection

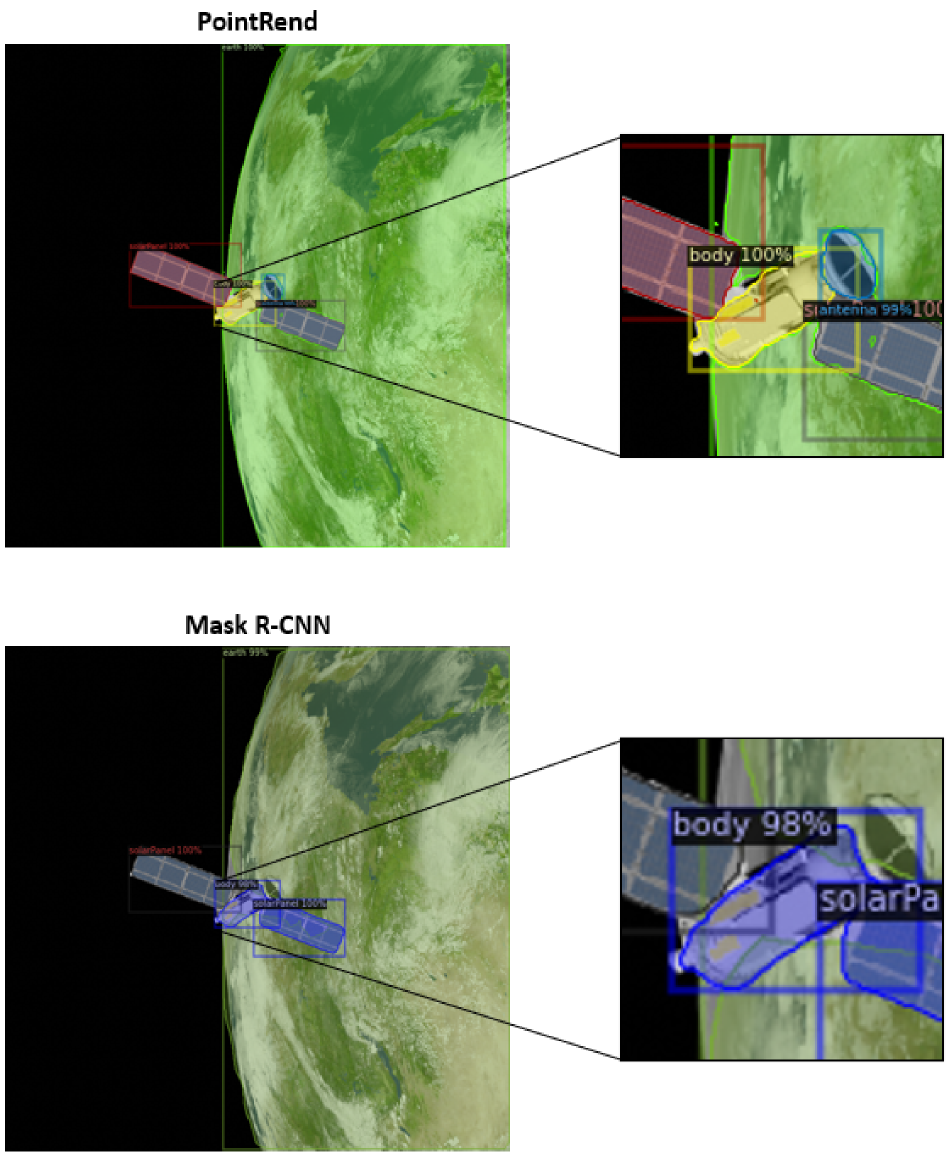


Figure 5.10: Inference on noisy images of models trained on dataset with noise.

task, Mask R-CNN fails to recognize the antenna that in the previous test was successfully identified. This may be due to the sensitivity of the model to noise or to the aleatory nature of the inference, as different tests on the same image can produce sometimes very different results. The segmentation precision of PointRend is increased, all edges are detected and segmented faithfully, and even the boundaries between different components have higher accuracy. This type of test demonstrated the utility of using images with noise in the training phase; the performance of the model during a real mission increased compared with when noiseless images were used in training. Using only images with noise during a training would mean assuming that the chaser camera always has that intensity of noise, in case it is different there could be a drop in accuracy. The best strategy to make the model more robust is to use the dataset without noise as the base dataset and then apply Poisson and Gaussian noise with different intensities as a form of data augmentation. This makes the model more generic and usable with cameras having different specifications and therefore return the scene with different accuracies.

5.2.4. Dataset in grayscale

Many of the camera-equipped satellites acquire grayscale images, one of the main reasons is because they require little memory to save and therefore are easier to process or send to a ground station. As in the case with noise, it was not necessary to regenerate all the images from scratch but only to convert them since the masks are independent of the color scale used. A dedicated function has been implemented, *Grayscale.py*, which allows to create a copy of a dataset and convert it to grayscale through the use of the Python package *cv2*. An example of the output returned by the function is shown in Fig. 5.11.

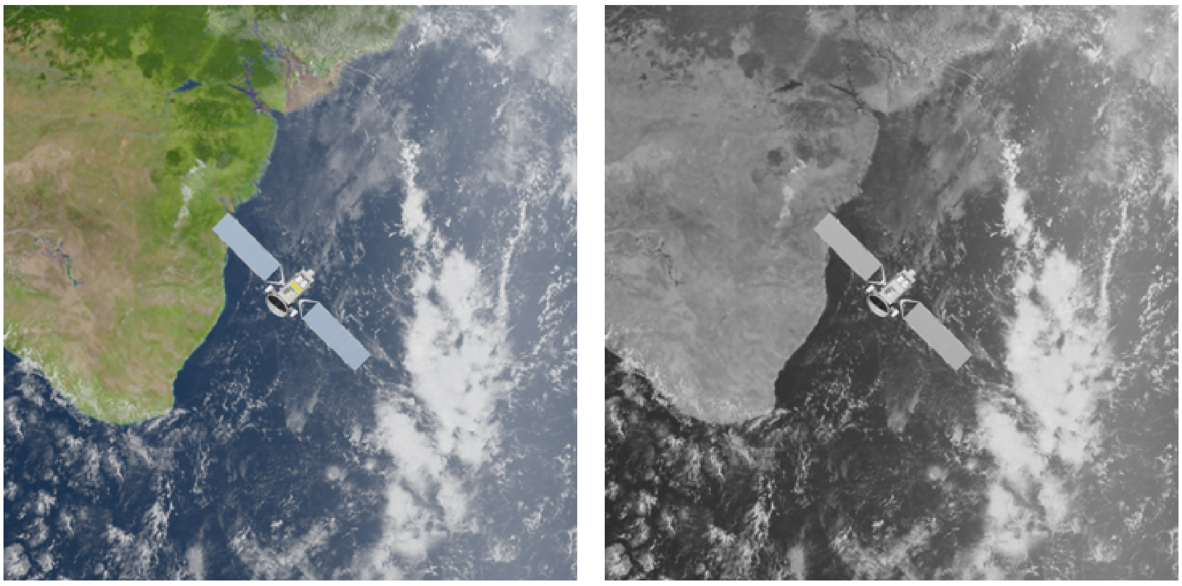


Figure 5.11: Example of grayscale conversion.

Using grayscale images the training time was slightly decreased to 1 hour 28 minutes for PointRend and 1 hour 12 minutes for Mask R-CNN. This is due to the nature of the images, since they have only one channel fewer operations are required to be performed on them. The results of the trainings are reported in Table 5.14 and Table 5.15.

Despite being trained with grayscale images PointRend has even higher accuracy values than Mask RCNN. This occurs since scenes without Earth in the background do not have a large amount of details to be recognized and therefore the format of the images used

PointRend	**AP**	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
No Earth in background					
Bbox	71.279	72.279	59.228	82.174	nan
Segm	64.132	58.925	64.760	68.711	nan
Earth in background					
Bbox	70.477	56.551	50.196	76.892	98.269
Segm	67.811	44.236	56.092	72.535	98.380
Earth + noise					
Bbox	58.017	31.068	42.134	72.821	86.046
Segm	50.148	19.621	36.379	62.665	81.927

Table 5.14: PointRend trained on grayscale dataset

Mask R-CNN	**AP**	**APantenna**	**APbody**	**APsolarPanel**	**APEarth**
No Earth in background					
Bbox	66.802	69.095	50.588	80.725	nan
Segm	59.685	55.201	58.980	64.873	nan
Earth in background					
Bbox	67.716	49.909	46.854	74.576	99.526
Segm	66.004	41.720	52.587	72.795	98.013
Earth + noise					
Bbox	61.059	30.870	42.561	71.960	98.847
Segm	56.076	24.205	37.507	65.132	97.459

Table 5.15: Mask R-CNN trained on grayscale dataset

in training does not affect the accuracy of the inference on colored images used as tests. Unexpected are the results obtained in the tests presenting the Earth in the background with and without noise. In the first case the two models have very similar accuracy values except for the Body and Antenna classes where PointRend has higher precision. On the other hand in the noise case the performance of PointRend is lower than that of Mask R-CNN unlike the previous case where the models were trained on colored datasets. These results confirm PointRend's greater sensitivity to scene details. Although many CNNs automatically convert RGB images to grayscale after a few layers, several models also use information from colors to recognize objects. Since both models are designed to take RGB or RGBA images as input [22], the lack of colors in the training dataset leads to reduced performance when tested on colored images. To evaluate the performance of PointRend one more test was performed. More specifically, both the model trained on color images and the grayscale model were tested on grayscale images without noise, to see if indeed training with color images produces better results. The results of the testing are shown in Table 5.16.

	AP	APantenna	APbody	APsolarPanel	APEarth
Trained RGB					
Segm	66.798	42.212	54.973	70.201	100
Trained Grayscale					
Segm	68.493	43.858	58.402	72.964	98.749

Table 5.16: PointRend models tested on grayscale dataset

Comparing the results obtained with those in Table 5.10 and Table 5.14, it can be seen that no obvious improvement is obtained by training in grayscale. The only test where higher values are obtained is when the model is trained and tested on grayscale images. Despite this, the improvement is about a 1 point AP and this does not motivate one to transform the whole dataset since using the color dataset gives higher accuracy for all other case studies. As confirmed by [25], RGB images produce improved results than grayscale images, returning clearer and noise-free images for better human or software interpretation. The use of grayscale images can be used as a form of data augmentation to train the model that will be used in the real inspection mission, or to generate a dataset if it is certain that the chaser only acquires images in grayscale.

5.3. Performances change due to variation of learning rate

So far the Machine Learning models have been trained by only changing the type of dataset and showing that PointRend produces more accurate segmentations than Mask R-CNN in most of the tests performed. A machine learning model in addition to making use of the training dataset employs two types of variables that make the trained model unique, *model parameter* and *model hyperparameter*. Model parameters are variables internal to the model and are determined from the data during training. An example are the weights that are updated at each iteration. Hyperparameters are parameters whose values control the learning process and determine the values of model parameters,and they are assigned externally by the user. There are many types of hyperparameters that can be assigned before the training phase, the most important are the learning rate, optimization algorithm, number iterations, batch size and the drop-out rate. Often the optimal values of the hyperparameters that guarantee the optimal solution are not known. To obtain the best combination of parameters, there are ad hoc techniques to determine these values. This hyperparameter selection procedure is known as Hyperparameter Tuning. The optimal hyperparameters are not unique but depend on the model and also on the dataset that is used for training. The main methods for hyperparameter selection are three, namely:

- *Manual search*: hyperparameters are selected based on experience, so it requires user supervision, and the performance of the model are verified. This process is repeated with another set of values until the optimal accuracy is achieved or the model has reached the desired error.
- *Random search*: Instead of performing multiple rounds of manual search, it would be better to provide the model with multiple values of the hyperparameters at once and let the model decide which one is the best. The model randomly creates its own combinations and tries to fit the dataset and check the accuracy.
- *Grid search*: it is based on the same principle as random search except here all combinations are tried and not chosen by the model. This is the most efficient method, as the chance of missing the optimal solution is very low.

The last two methods for choosing hyperparameters are quite expensive both computationally and in terms of time. Since Colab's free plan allows training for a very limited daily time, a manual approach was used for choosing hyperparameters. As mentioned earlier one of the most important parameters for training is the learning rate. In all previous trainings it has been kept fixed with a value of 0.003. It is intended to see how the accuracy of PointRend changes as the value of the learning rate varies. The selected values are: [0.0025, 0.004, 0.025, 0.0001]. The first two were chosen to see how the model behaves using a learning rate close to the initial one, and the last two values in the case where the learning rate was varied by one order of magnitude. The dataset used for training is the one containing colored scenes with and without the Earth in the background, on which data augmentation was performed. This is because, as mentioned above, it covers a greater variety of case histories and allows for higher accuracy values than the other datasets used. Using different values of learning rate also changed the training times required. Higher learning rates require shorter training times while for low values of learning rate the training times increase. Considering $J(\theta)$ the loss function describing the performances of the model and θ the set of weights used in an iteration it is possible to observe the importance of the learning rate, as shown in Fig. 5.12.

To maintain consistency with previous trainings all other parameters were kept constant as well as the GPU model used. The training times required by PointRend vary between 1 hour and 20 minutes for the learning rate of 0.004 and 1 hour and 35 minutes for that

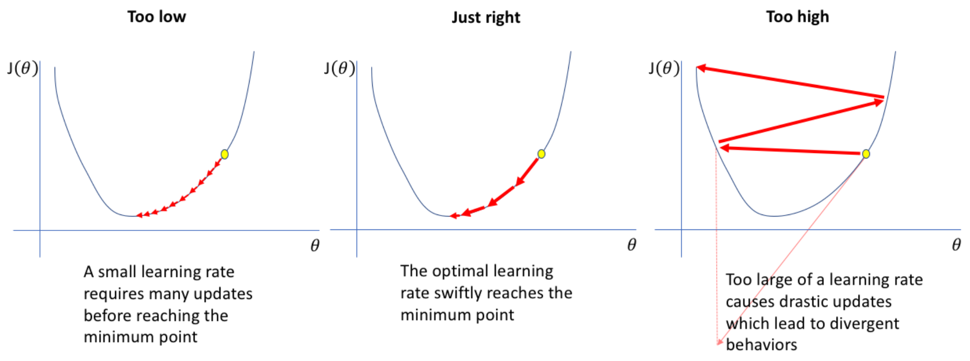

Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Figure 5.12: How learning rate affects training. Image credit

of 0.0001. The results are reported from Tables 5.17 to 5.19.

	AP	**AP antenna**	**AP body**	**AP solarPanel**	**AP Earth**
Lr = 0.004					
Bbox	68.418	69.320	55.186	80.749	nan
Segm	64.534	57.534	65.167	70.913	nan
Lr = 0.0025					
Bbox	69.356	66.089	60.460	81.520	nan
Segm	62.875	55.810	63.700	69.115	nan
Lr = 0.0001					
Bbox	66.745	67.210	51.803	81.222	nan
Segm	57.095	52.374	56.371	62.432	nan

Table 5.17: PointRend tested without Earth in background

	AP	**AP antenna**	**AP body**	**AP solarPanel**	**AP Earth**
Lr = 0.004					
Bbox	69.511	54.406	49.258	74.379	100
Segm	67.417	47.318	52.690	69.661	100
Lr = 0.0025					
Bbox	69.019	51.391	49.306	75.716	99.662
Segm	67.081	44.350	56.433	67.543	100
Lr = 0.0001					
Bbox	68.210	53.320	45.213	74.306	100
Segm	61.070	37.443	43.318	63.521	100

Table 5.18: PointRend tested with Earth in background

	AP	AP antenna	AP body	AP solarPanel	AP Earth
Lr = 0.004					
Bbox	51.924	23.154	32.596	60.330	91.618
Segm	45.774	16.646	28.127	41.175	97.147
Lr = 0.0025					
Bbox	50.119	23.755	28.277	54.457	93.986
Segm	42.661	14.433	25.132	36.160	94.918
Lr = 0.0001					
Bbox	60.114	30.391	41.257	71.141	97.668
Segm	49.370	18.896	30.099	52.351	96.133

Table 5.19: PointRend tested with Earth in background and noise

The tables are missing AP values related to training with $Lr = 0.025$. Such a high value leads the model to diverge as described in Fig. 5.12. In fact after a hundred iterations the model returns an error warning : "*FloatingPointError: Predicted boxes or scores contain Inf/NaN. Training has diverged.*". This value could be used as an upper bound for future hyperparameter tuning of the model. Looking at the accuracy values described in the table, it is possible to confirm that the learning rate is a very important parameter for model training purposes. In the first two tests carried out it can be observed that the highest accuracy values are obtained for higher learning rate values, this may be due to the fact that since the scenes on which the inference is carried out are similar to those used for training a higher learning rate succeeds in apprehending the general features of the scene faster than a lower learning rate which learns slower by focusing more on details. In contrast, in the test in which noisy images were considered, the highest accuracy values were obtained from the lowest learning rate. This can be explained by the fact that, since the scenes used in the inference are not contained in the training dataset, the lowest learning rate will be less sensitive to noise and to scenes that has never seen as it has focused much more on details than on the generic feature. This aspect should be taken into account for training the model in the case of a real mission. A more widely used approach in training by the latest developed machine learning models is to adopt a variable learning rate. Initially the value used is higher so that the model can better detect generic features, while toward the later iterations the value of the learning rate is decreased so that details are better recognized. A further remark should be made on AP values obtained, particularly for those concerning solar panels. It can be seen that the AP value of the solar panels for both detection and segmentation is always higher than the one of the other components of the RSO. This is because all the solar panels in the models used by JINS to create the datasets have the same shape, so there is no much difference between those used during training and those during inference. To increase the robustness of the model it would be necessary to use in the training phase not only variety of scenes but also variety of component shapes. An example of inference for each learning rate value is shown in Fig. 5.13.

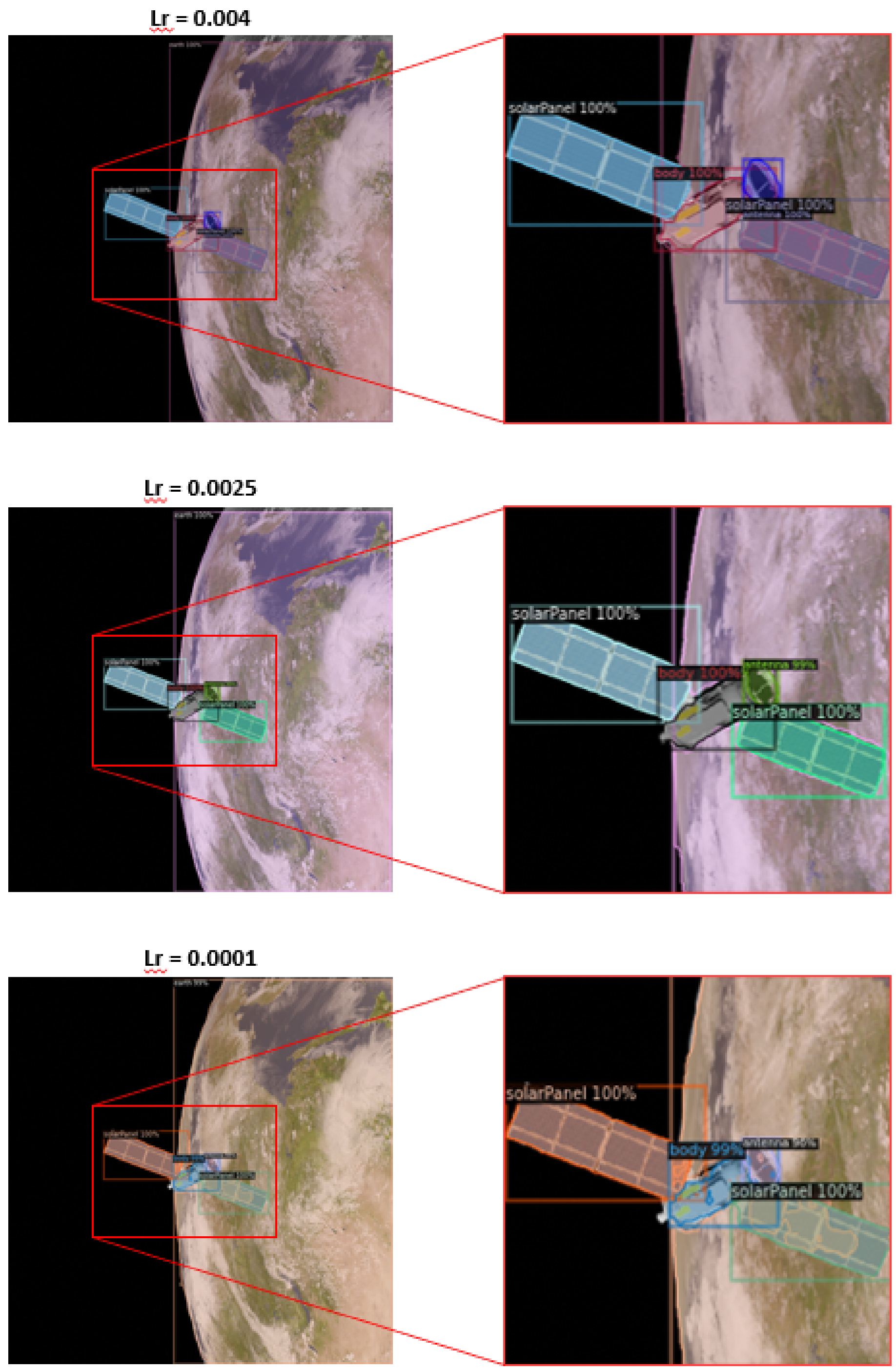


Figure 5.13: Inference models trained with different learning rate.

5.4. Inference on real images

As a last test, models should be tested on real images. This is to test whether models trained on synthetically generated images can be used in real missions. Similarly to [24], images captured from an inspection mission were picked. More specifically, the mission that began in 2019 aimed to extend the service life of the satellite Intelsat 901 (IS-901) by means of another satellite, the Mission Extension Vehicle-1 (MEV-1). In 2020 before MEV-1 performed the docking, some shots were taken. These will be used by PointRend and Mask R-CNN to test the validity of the models.
The images of the Instelsat 901 satellite used for the inference are shown in Fig. 5.14.

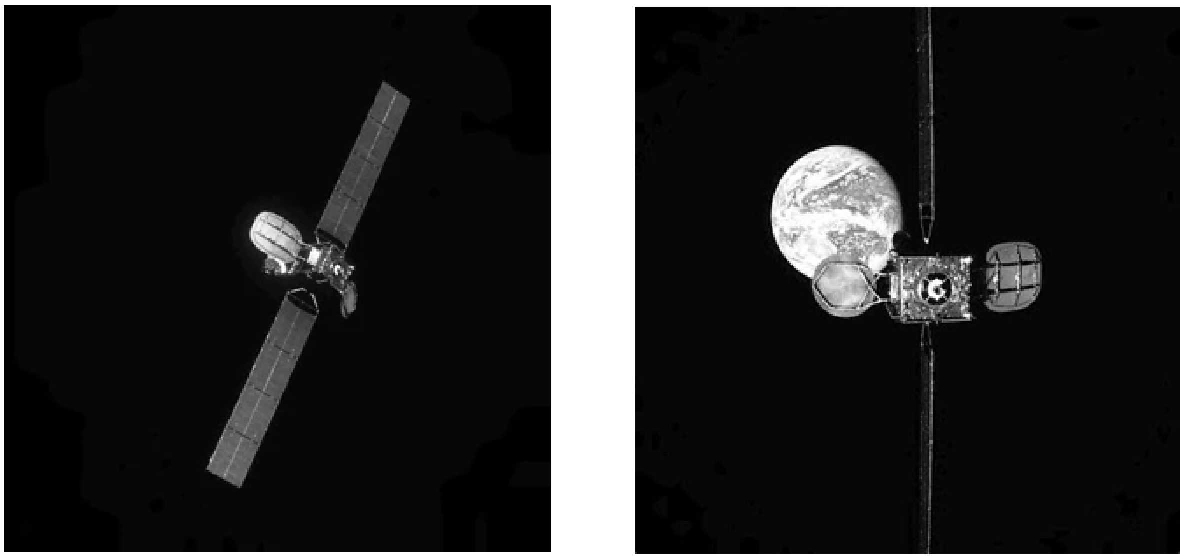


Figure 5.14: Intelsat 901 images.

As can be seen from the figure, the images that will be used are in grayscale and with a much lower resolution than the ones generated by JINS since they come from a real scene. Also an important factor is that they are taken from the web so they will have a lower quality than the original ones. The models used for inference are those trained previously on colored datasets with Earth in the background, datasets with noise, and grayscale datasets. In [24] the images are fed through a function that performs a threshold on the image to eliminate background noise, in this work it was decided not to preprocess the images because it is intended to verify the performance of the models even in the presence of noise. The inference of the models are given below, Fig. 5.15 for PointRend and Fig. 5.16 for Mask R-CNN, respectively.

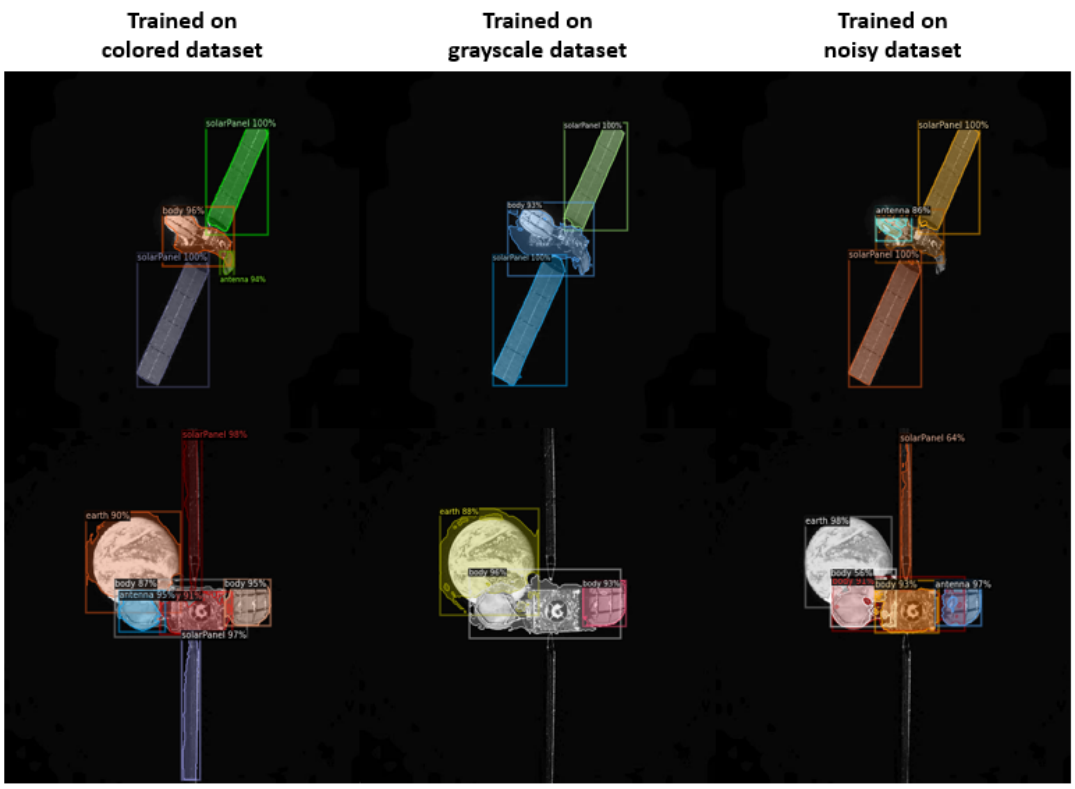


Figure 5.15: Intelsat 901 inferences from PointRend.

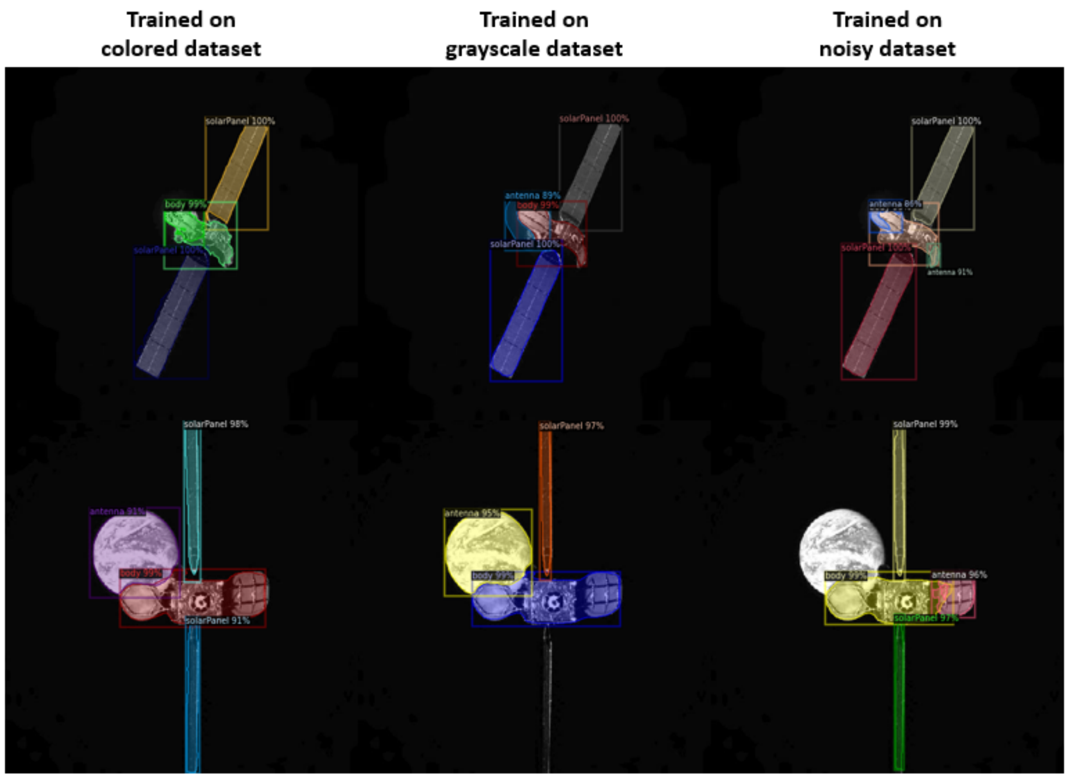


Figure 5.16: Intelsat 901 inferences from Mask R-CNN.

As can be seen from the above images, the models manage to recognize the satellite components quite well despite having a completely different structure from those contained in the training dataset. Not only the satellite model but also the scene type is entirely new to the trained models since within the training datasets the Earth in the background is never seen in its complete entirety. Depending on the type of dataset used to train the models very different results are obtained. Some observations can be made by looking at the inference results. When trained on color datasets, excellent results are obtained as all components are recognized. The greater accuracy of PointRend can be seen in the greater amount of details and components recognized. It can be noted that sometimes satellite antennas are mistaken for *body*. This is due to the fact that in the training dataset the antennas have never been seen in similar position and shape. Consequently, they are mistaken for other components. Very important is the fact that PointRend is able to identify the Earth in the background with the correct label, unlike Mask R-CNN, which assigns to the Earth the *antenna* label. This aspect is critical in the case of a real mission because if the chaser were to perform operations on the antenna and the algorithm identified the antenna in an incorrect position problems would arise.
Unexpected are the results obtained with the models trained on grayscale datasets. The performances of the models are clearly lower than the previous test despite the fact that the images on which inferences are made are in grayscale. Also it can be noticed from the inaccurate antenna and body masks that both models are more sensitive to background noise. PointRend being very sensitive to details the noise does not allow the identification of low-illuminated solar panels that were previously identified correctly. Instead, the expected result is the increased robustness of models trained on noisy images to background noise. This can be easily noticed by comparing the segmentations of the Earth made by PointRend. Despite the fact that the model trained on the dataset with noise should have higher accuracy on real images, the presence of few details in the scene and the fact that it is in grayscale makes it perform worse than the model trained on noise-free colored dataset. So the scene can be approximated to a simple RGB scene without Earth in the background given its scale. This assumption validates the fact that the model trained on colored scenes performs better than the model trained on noisy images, and that is confirmed by the values in Table 5.10 and Table 5.12.
The result of Mask RCNN to which attention should be paid is the one obtained from the model trained on a noisy dataset. This time the Earth in the background is not identified, and it is not possible to know from one photo whether the model is unable to recognize it all the time or it failed to recognize it only on this occasion due to its location or the randomness of the inference. The first hypothesis does not cause negative consequences on the model since the position of the Earth is not of primary importance for an inspection

mission, while in case the second hypothesis is true it would be necessary to monitor the returned output so that as in previous trainings the model does not identify the Earth as an antenna and lead to arising of problems. An additional test was carried out by using the trained models with different learning rates. The results of the infernces are reported in Fig. 5.17.

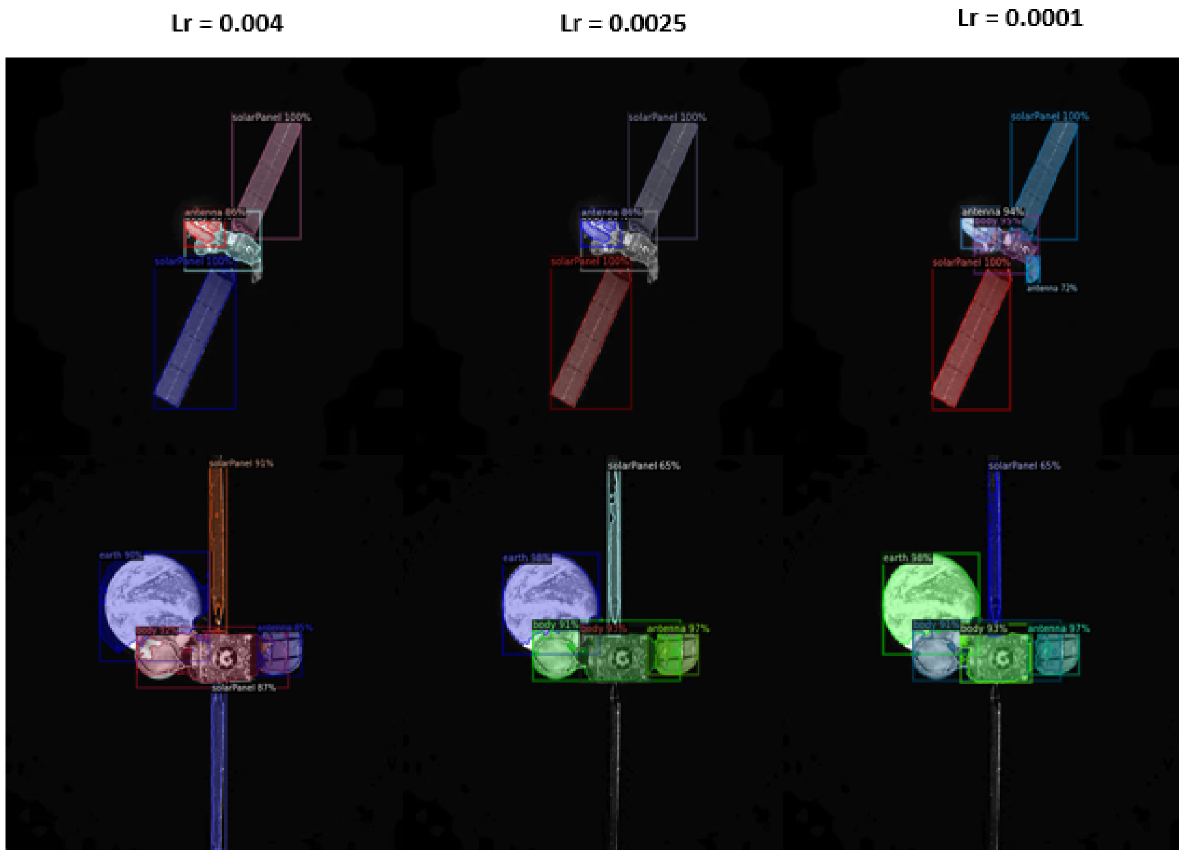


Figure 5.17: Intelsat 901 inference of models trained with different learning rate.

As expected when looking at the results of the same type of test performed on synthetically generated images, the results differ visibly depending on the learning rate value used. The highest learning rate value, 0.004, succeeds in identifying all components of the satellite but is not very accurate. This can be seen from the fact that it recognizes the body and the left solar panels as *body* and from the low accuracy of the Earth mask. On the other hand decreasing the learning rate the accuracy of the mask along the edges increases. In the case where the learning rate is 0.0001 all the identified components are segmented perfectly. Fundamental difference between the inference with high learning rate and that with a low value is the correct identification of the lower solar panel. The high learning rate succeeds in correctly identifying the solar panel. This is because by

learning better the generic features it does not need much detail to identify a class. While the higher learning rate fails to identify it because being the solar panel low illuminated it does not have much details, a fundamental factor for the proper functioning of the model trained with low learning rate.
An aspect that stands out immediately after this training is the difference between the results obtained by performing inference on synthetically generated images and real images. In tests performed on images generated by JINS, the models always succeed in identifying the objects contained in the scene, and their segmentation accuracy varied depending on the training performed. In this last test with real images the models also begin to fail in detection, this could be due to the fact that the Intelsat 901 satellite does not have a similar structure to those contained in the training dataset, and the same is true for the type of scene ever seen in training phase. To test whether the problem of failed detections was actually related to the model of the satellite or to the quality of the image used, it was planned to synthetically generate scenes similar to those of the real images using satellite model 5, the one not contained in the training dataset. The results of both PointRend and Mask R-CNN are illustrated in Fig. 5.18 and Fig. 5.19, respectively.

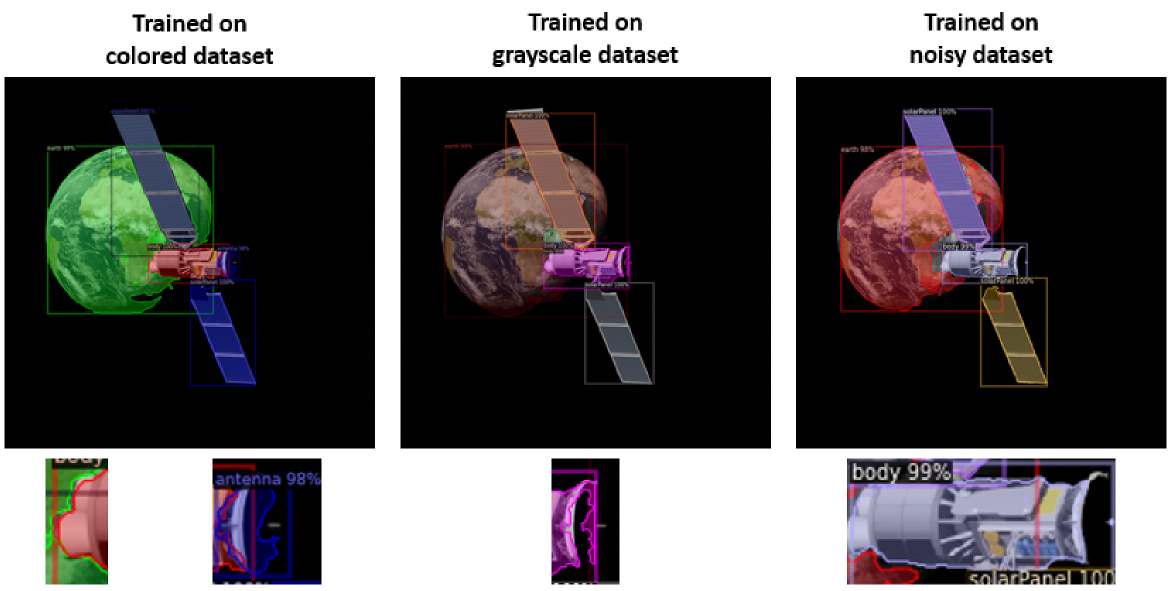


Figure 5.18: Reproduction of the Intelsat 901 scene analyzed by PointRend

As seen in previous tests, the model trained on the color dataset without noise provides higher accuracy than the other trainings. PointRend continues to return a more accurate mask than Mask R-CNN, but there is one issue that is common to almost all inferences, the missed detection of the antenna. This problem may be due to the fact that the antenna is almost completely in shadow and only a small portion is visible. The same situation

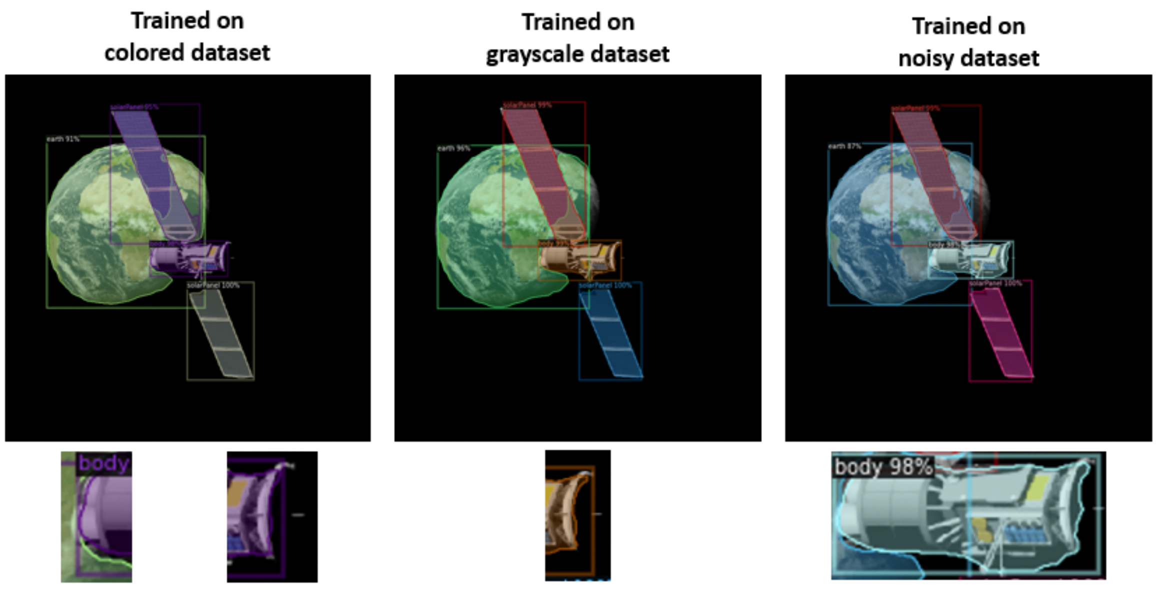


Figure 5.19: Reproduction of the Intelsat 901 scene analyzed by Mask R-CNN

occurs for IS-901 images where the lower solar panel is completely in shadow and it is only recognized by PointRend with the model trained on dataset without noise. All these tests performed to compare PointRend and Mask R-CNN served to show that, for the same hyperparameter and training dataset PointRend turns out to be a more robust and accurate model than the standard Mask R-CNN. Its perfromance makes it ideal for use during an inspection mission by ensuring higher accuracies. In addition, the new version of JINS software makes it easier to generate consistently real and accurate images.

6 | Conclusion and further studies

The use of neural networks to perform detection and segmentation tasks during an inspection mission around a non-cooperative RSO has already been demonstrated in previous work [24] leading to the choice of Mask R-CNN as the state-of-the-art model. The aim of this work was to identify a better machine learning model that uses a different architecture to perform mask generation and compare its performance with the state-of-the-art model, Mask R-CNN.

To simulate various conditions that may be encountered during a real mission, different types of tests were performed on the two models to check their performances. Despite the fact that Mask R-CNN manages to obtain good results when tested under standard conditions, PointRend provides much greater accuracy along component edges and not only under nominal conditions but also when tested on noisy or grayscale images. The results obtained show that PointRend is an excellent candidate for object detection and segmentation tasks, providing higher performance than the state-of-the-art model with the same inference speed and required computational power. Since the model has high accuracy, the data provided by the inference are very reliable and can be used to identify the attitude of the target RSO, which is fundamental in the case of docking, or used by algorithms for autonomous navigation in the proximity of other satellites. As seen in Chapter 5 a crucial part for obtaining good results is the choice of the training dataset and the setting of the hyperparameters. Using different types of scenes within the training dataset caused the accuracy of the model to increase. Nevertheless the major contribution came from the data augmentation techniques used to expand the dataset. This allowed, at the expense of a small increase in the training times, the models to be more robust to new types of scenes and also more accurate.

The ease with which it was possible to generate so many datasets for training is thanks to the new version of the JINS software. The software now has several functions such as propagating the relative orbit that one wants to represent, the satellite model can be directly chosen by the user via a menu and no longer by running several scripts, and most importantly it is possible to implement both chaser and target attitude. This last function is very important if one wants to represent even more realistically the inspection

mission considered. In this way it is possible to generate a video on which one can run the inference to assess the performance of the model during the mission. To test the models not only on JINS-generated scenes, real images of the Intelsat 901 satellite were used. Unlike previous work where these images were preprocessed to eliminate the background noise, now they are provided to the models without eliminating the noise to compare the performance of the models with noisy images. It was shown that PointRend has higher accuracy even on real images. Moreover eliminating the preprocessing step resulted in reducing the number of operations required to obtain information from the scene.
The branch of computer vision is a new and constantly evolving environment. As the use of machine learning in orbit receives more and more attention, such dynamic environments make it possible to formulate several ideas for future development of this work. Among them:

- Implement on a Raspberry Pi board to verify the low computational cost and the performances on real images knowing the parameters of the camera.
- Verify if the data obtained from the instance segmentation of the scene can be used for avoidance manoeuvres.
- Use the instance segmentation model as a base to estimate the pose of the satellite.
- Replace the standard backbone of the model with a vision transformer specifically the *SWIN transformer* developed by Microsoft. The use of the vision transformer is an approach to replace convolutions entirely with a transformer model.
- Implement additional data augmentation techniques and embed them directly within the JINS software.
- Perform hyperparameter tuning to see the maximum performances of the model and compare with the ones obtained using a variable learning rate.
- Implement models based on the YOLO family of methods, such as YOLACT, which not only enables very high inference rates but also instance segmentation. This allows to have greater temporal consistency between segmentations in the case where the inspection mission takes place around a satellite rotating at high speed.

Bibliography

[1] T. Alan. Computing machinery and intelligence. *Mind*, 1950.

[2] S. C. class. Cs231n convolutional neural networks for visual recognition, 2022. URL `https://cs231n.github.io/classification/`.

[3] W. H. Clohessy and R. S. Wiltshire. Terminal guidance system for satellite rendezvous. *Journal of the Aerospace Sciences*, 27(9):653–658, 1960.

[4] COCO. Coco evaluation metrics, 2022. URL `https://cocodataset.org/#stuff-eval`.

[5] H. Curtis. *Orbital Mechanics for Engineering Students*. Elsevier, 2014.

[6] D.J.Kessler and B.G.Cour-Palais. Collision frequency of artificial satellites: The creation of a debris belt. *Journal of Geophysical Research*, 1978.

[7] DLR. Mini robots practise grasping space debris, 2022. URL `https://www.dlr.de/content/en/articles/news/2022/01/20220322_mini-robots-practise-grasping-space-debris.html`.

[8] S. Dmitry and A. Vladimir. Space debris removal with harpoon assistance: Choice of parameters and optimization. *Journal of Guidance, Control, and Dynamics*, 2020.

[9] ESA. Esa's annual space environment report. Technical report, ESA, 2022.

[10] ESA. Space environment statistics, 2022. URL `https://sdup.esoc.esa.int/discosweb/statistics/`.

[11] R. Girshick. Fast r-cnn. *Proceedings of the IEEE international conference on computer vision*, 2015.

[12] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization, 2014.

[13] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, 2017. URL `http://arxiv.org/abs/1703.06870`.

[14] R. Hecht-Nielsen. Theory of the backpropagation neural network. *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989.

[15] R. R. . R. M. II. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Bradford Books, 1999.

[16] T. O. Keiron and N. Ryan. An introduction to convolutional neural networks. *Neural and Evolutionary Computing*, 2015.

[17] A. Kirillov, Y. Wu, K. He, and R. Girshick. Pointrend: Image segmentation as rendering. *ArXiv:1912.08193*, 2019.

[18] M. Konnij and J. Welsh. High-level numerical simulations of noise in ccd and cmos photosensors: review and tutorial. Technical report, Faculty of Engineering and Built Environment, University of Newcastle, Australia, 2014.

[19] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 1995.

[20] J. Luo, W.-w. Tang, W.-y. Zhou, and Y. Jianping. Observation trajectory design and control of on-orbit inspection satellite. *SPIE International Conference on Space information Technology*, 2005.

[21] M.Andrenucci, P.Pergolaand, and A.Ruggiero. Expanding foam application for active debris removal. *ESA*, 2011.

[22] matterport. Training with rgb-d or grayscale images, 2018. URL `https://github.com/matterport/Mask_RCNN/wiki#training-with-rgb-d-or-grayscale-images`.

[23] NASA. Technical report on space debris. Technical report, NASA, 1999.

[24] F. Niccolò. Instance segmentation for features recognition on non-cooperative resident space objects. Master's thesis, Politecnico di Milano, 2020.

[25] K. Padmavathi and K. Thangadurai. Implementation of rgb and grayscale images in plant leaves disease detection – comparative study. *Computer Graphics and Image Processing*, 9(6):1–6, 2016. doi: 10.17485/ijst/2016/v9i6/77739.

[26] Powers, David, and Ailab. Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *J. Mach. Learn. Technol*, 2:2229–3981, 01 2011. doi: 10.9735/2229-3981.

[27] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972. ISSN

0146-664X. doi: https://doi.org/10.1016/S0146-664X(72)80017-0. URL `https://www.sciencedirect.com/science/article/pii/S0146664X72800170`.

[28] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, 2016. URL `http://arxiv.org/abs/1612.08242`.

[29] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, 2018. URL `http://arxiv.org/abs/1804.02767`.

[30] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, 2015. URL `http://arxiv.org/abs/1506.02640`.

[31] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[32] G. Ross, D. Jeff, D. Trevor, and M. Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. *UC Berkeley*, 2014.

[33] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1 edition, 1995. ISBN 0-13-461099-7.

[34] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 1959.

[35] T. Sgobba and F. Allahdadi. *Safety Design for Space Operations*, chapter 8, pages 411–602. Elsevier, 2013.

[36] S. Sharma. Pose estimation of uncooperative spacecraft using monocular vision and deep learning. Master's thesis, Stanford University, 2019.

[37] S. Shuangyan, J. Xing, and H. Chang. Cleaning space debris with a space-based laser system. *Chinese Journal of Aeronautics*, 2014.

[38] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022. URL `https://arxiv.org/abs/2207.02696`.

[39] D. C. Woffinden. On-orbit satellite inspection : Navigation and av analysis. Master's thesis, Arizona State University, 2002.

[40] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2 model zoo. `https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md`, 2019.

[41] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[42] B. Xu, S. Wang, and L. Zhao. Solar panel recognition of non-cooperative spacecraft based on deep learnin. *2019 3rd International Conference on Robotics and Automation Sciences (ICRAS)*, pages 206–210, 2019.