POLITECNICO
MILANO 1863

DAER - Dipartimento di Scienze e Tecnologie Aerospaziali
MSc course in Space Engineering

INSTANCE SEGMENTATION FOR FEATURES RECOGNITION ON NON-COOPERATIVE RESIDENT SPACE OBJECTS

SUPERVISOR:
Prof. Pierluigi Di Lizia

CO-SUPERVISOR:
Mr. Michele Maestrini

CANDIDATE:
Niccolò Faraco

Academic Year 2019-2020

Alla mia famiglia,
salda àncora nelle acque tumultuose della vita,
ma anche vento fecondo
pronto a spingermi verso nuovi e inesplorati lidi.

Abstract

Active debris removal and unmanned on-orbit servicing missions have gained increasing interest in the last few years, as well as the possibility to perform them through the use of autonomous chasing spacecrafts. In this work, new resources are proposed to aid the design of such missions and the implementation of new control algorithms and autonomous guidance techniques for satellites devoted to the inspection of non-cooperative targets before any proximity operation is initiated. In particular, the use of state-of-the-art Convolutional Neural Networks (CNNs) performing object detection and image segmentation is proposed and its effectiveness in recognising features and parts of the target satellite is evaluated. To this aim, such Machine Learning (ML) algorithms have to be trained on representative datasets, that is, in this case, on a batch of carefully annotated images of tumbling satellites taken from other spacecrafts in relative orbit around them. No reliable dataset of this kind exists to date since something similar has only been achieved on images taken from the Internet. Here lies the raison d'être of this paper: to overcome this limitation a synthetic dataset has been created; i.e. CAD models of several existing satellites have been loaded on a 3D animation software and used to programmatically render images of the objects from different point of views and in different lighting conditions, together with the necessary ground truth labels and masks for each image. The results show how a relatively low number of iterations is sufficient for a CNN trained on such dataset to reach a mean Average Precision (mAP) value in line with the ones scored by the same algorithms on more common datasets, showing that using only artificially generated images to train the model does not compromise the learning process. A comparison between the performance obtained when training the neural network on images of a single satellite and the one obtained on a dataset containing pictures of several different spacecrafts is provided. To conclude, the method is tested on images from a real on-orbit servicing mission, namely the one targeted to the Intelsat 901 communication satellite.

Abstract (Italian version)

Le missioni per la rimozione attiva dei debris spaziali e quelle per il servicing in orbita hanno guadagnato, insieme alla possibilità di essere operate da satelliti a guida autonoma, crescente interesse negli ultimi anni. In questo lavoro viene proposto un nuovo approccio per aiutare nella progettazione di tale genere di missioni e l'implementazione di nuovi algoritmi di guida e controllo per satelliti votati all'ispezione di obiettivi non cooperanti prima dell'inizio delle manovre di prossimità. In particolare, si propone l'uso di Convolutional Neural Networks (CNNs) che eseguano object detection and image segmentation e se ne valuta l'efficacia nel riconoscere parti e caratteristiche del satellite obiettivo. A questo scopo, tali algoritmi di Machine Learning (ML) devono essere addestrati su dataset sufficientemente rappresentativi, ovvero, in questo caso, su una serie di immagini accuratamente etichettate di satelliti alla deriva scattate da altri satelliti in orbita relativa intorno ad essi. Ad oggi non esiste alcun dataset di questo tipo, qualcosa di simile è stato fatto solo su immagini prese da Internet. Qui risiede la raison d'être di questo lavoro: per superare questo problema, si sono creati dataset artificiali; nello specifico, modelli CAD di satelliti (esistenti e non) sono stati caricati su un programma di modellazione 3D e usati per la generazione programmatica di immagini del satellite obiettivo da differenti punti di vista e in differenti condizioni di illuminazione, simultaneamente alle maschere che definiscono la posizione delle parti in ogni immagine. I risultati mostrano come un numero relativamente basso di iterazioni di addestramento sia sufficiente per raggiungere valori di mean Average Precision (mAP) in linea con quelli ottenuti dagli stessi algoritmi su dataset più comuni, dimostrando come l'uso di immagini artificiali per l'addestramento del modello non comprometta il processo. Si sono inoltre comparate le performance ottenute addestrando la rete neurale su un singolo satellite con quelle ottenute su dataset che includono immagini di più satelliti diversi. Infine, il metodo è stato testato su immagini di una missione reale di on-orbit servicing, quella che ha avuto come obiettivo il satellite Intelsat 901.

Contents

List of Figures

List of Tables

Acronyms list

RSO	Resident Space Object
OOS	On-Orbit Servicing
LEO	Low Earth Orbit
GEO	Geostationary Earth Orbit
GNC	Guidance Navigation and Control
CAD	Computer Aided Design
AI	Artificial Intelligence
ML	Machine Learning
ANN	Artificial Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
RPN	Region Proposal Network
R-CNN	Region-based Convolutional Neural Network
AP	Average Precision
mAP	mean Average Precision
SVM	Linear Support Vector Machine
RoI	Region of Interest
FPN	Feature Pyramid Network
ReLU	Rectified Linear Unit
LVLH	Local-Vertical Local-Horizontal
JINS	Jins Is Not a Simulator
RNN	Recurrent Neural Network

Chapter 1

Introduction

Since the beginning of the space exploration era, the volume surrounding our planet has gradually become more and more populated by man-made Resident Space Objects (RSOs). Moreover, this trend has been exacerbated since the establishment of commercial exploitation of space. Larger and hence more dangerous RSOs include intermediate stages of launch vehicles, remains of past missions or fragmentations and decommissioned satellites. All-together these objects are known as *space debris*.
The incipient overcrowding of the most exploited orbits (think for example of LEO or GEO orbits) due to the presence of these objects constitutes a non-trivial challenge for the design of new missions, both manned and unmanned. This has led to the development of several different ideas to monitor the problem and possibly mitigate it, often based on the type of debris in question. When the RSO is a satellite at the end of its operational life but still preserving some control capabilities, the use of graveyard orbits or atmospheric reentry are the suggested strategies [1]. On the other hand when dealing with non-cooperative objects, the possible solutions include advanced on-Earth systems for accurate tracking of the objects [2], the implementation of models for the orbit propagation and the study and design of future missions for the active removal or disposal of the space debris [3].

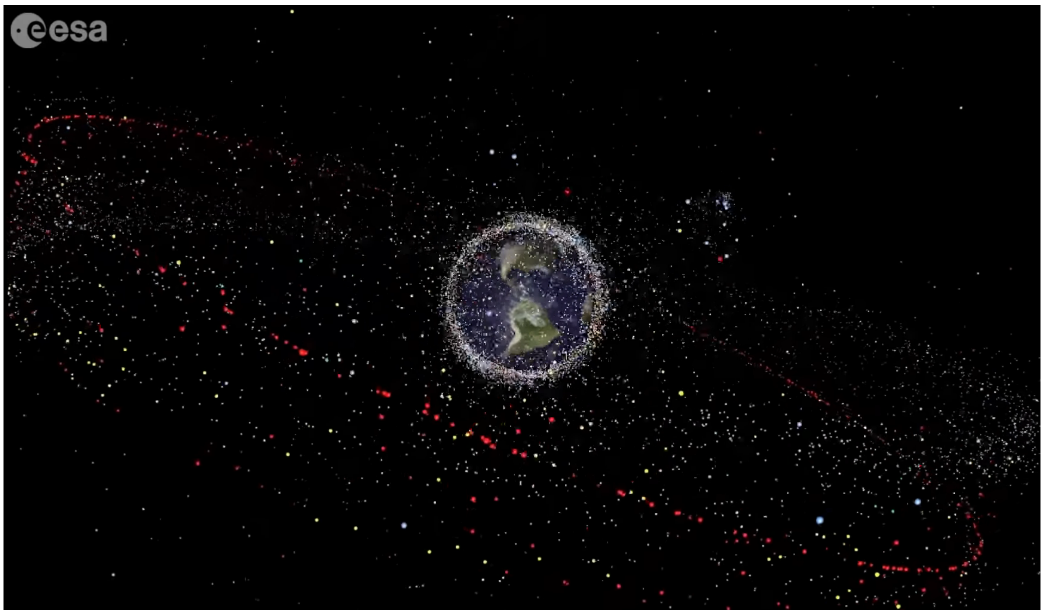


Figure 1.1: Representation of the space debris situation around our planet. Points represent objects larger than 10 cm. Data is from the US Space Surveillance Catalogue. Image credit: ESA

The latter, in particular, has increasingly gained the attention of the community and has been drawing big efforts since it is the only long term solution to the problem when atmospheric reentry disposal is not feasible. However, a very precise knowledge of the conditions of the satellite to be removed is needed to perform such missions. To this aim, missions specifically targeted to the inspection of the objective spacecraft come to play, e.g. the e.Inspector mission, which Politecnico di Milano has been recently entrusted by ESA.
This operations are carried out through a secondary satellite, the *chaser*, whose orbit is defined based on the one of the *target* RSO. The keplerian motion of the two bodies produces a relative orbit between target and chaser, which thus has the chance to inspect the RSO from different points of view. Many different types of relative orbits have been studied and their effectiveness addressed [4]. Which one to choose depends on the requisites of the mission.

As for any other ordinary mission to date, the satellite relies on a constant flow of information, back and forth from the spacecraft to the ground, in order to determine the current dynamics of the target body. Usually, the data transmitted from the space segment is elaborated on ground, and relevant commands are transmitted back on orbit to be executed.
The problem with this kind of approach lies in the fact that relative orbits for inspection missions have to be continuously modified based on whether they are still effective or not. In fact, the chaser spacecraft, along its orbit, will be able to observe only certain regions of the target, therefore it will be required to change its relative motion in order to reach the full coverage and gain more data about the RSO. It is clear how this increases a lot the controlling effort for the mission analysts and, therefore, how it constitutes a major limit for the state-of-the-art in inspection missions design. Moreover, the presence of humans in the loop may lead to the selection of suboptimal orbits and it is prone to possible errors.

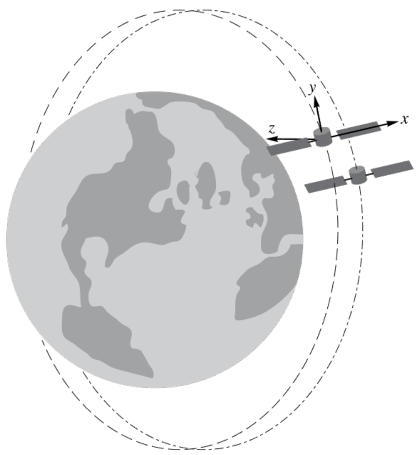


Figure 1.2: Satellites in relative orbit around each other.

A possible solution comes from the opportunity to introduce a certain degree of autonomy in the maintenance of this category of missions. Since the main aim of the mission is the visual inspection of the target satellite, the chaser is naturally equipped with one or more cameras which are used to shoot pictures to be sent to Earth for the subsequent analysis. The idea is to use the very same pictures in order to untie the GNC tasks from the ground segment: in fact, these images could be fed to the on-board computer and to machine learning algorithms devoted to object recognition, which are nowadays becoming more and more effective, in order to detect the prominent features of the target satellite. This kind of computer vision software also provides a measure of the reliability of the results that can, therefore, be used as a suggestion on which part of the target needs to be further examined. This score can be used as an input for another piece of software, namely an autonomous guidance algorithm (e.g. [5]), to define a new flight profile for the chaser, changing its relative orbit around the target so that it offers better points of view on the most uncertain features of the inspected RSO. Moreover, if properly tailored, also an autonomous navigation could be achieved with such approach as demonstrated in [6]. This obviously has to be repeated iteratively until full coverage of the target spacecraft is obtained with the prescribed accuracy.

At this point, it should be clear that the implementation of recognition algorithms and the assessment of their performances is of uttermost importance to enable the use of autonomous guidance and navigation algorithms for future inspection and proximity missions [7]. This is what this thesis is focused on.

Specific declinations of Region-based Convolutional Neural Networks (R-CNNs) have been chosen as the tool to tackle the problem presented in this work [8]. These algorithms, just like any other image recognition algorithm, have to be trained on sets of pictures for which the category of the objects depicted and their position on the pixel grid is provided. The key innovation of this work lies in the approach used to train such neural

networks or, more precisely, in the kind of images provided for the training. Up to date, in fact, something similar has only been done on sets of manually annotated images taken from the internet and on networks only able to recognise solar panels [9]. This has two main disadvantages. The first one deals with the scalability of the method, which is a relevant drawback since the algorithms' performance dramatically improves when trained on larger datasets, as also shown in Section 5.2.1: in fact, the images have to be manually labelled and there is no way to automate the process, which makes it labour intensive for a human operator to provide a large enough dataset. The second point is the inability to tailor the recognition on specific targets in the case of those missions that will have to inspect a single, specific and known a priori target.
The solution explored in this thesis is using 3D CAD models of various satellites in order to programmatically generate images with different points of view and lighting conditions. This has been achieved by means of an open-source computer graphic software, called Blender, accurately scripted in order to generate, together with the images, also the necessary information on the objects depicted and their position in the picture, which hereafter will be referred as *ground truth.* This approach proved to be an effective method for the training of the neural networks, also solving the problems outlined for the approaches previously used in the literature. Moreover, it granted good results not only on simulated images, but also when tested on images from a real mission, namely the Intelsat 901 life extension mission.

This work will be structured as follows. The working principles of the use of neural networks for image recognition along with the various implementations taken into account and the architecture of the chosen one are explained in Chapter 2, while in Chapter 3 an overview of relative orbital motion as a whole and common relative trajectories is provided. Chapter 4 introduces in more detail an original software, called JINS, which implements the algorithms and general framework for the generation of the synthetic datasets for the training and testing of the neural network. Various tests have been performed exploiting the generated dataset presented in this chapter, and their results are discussed in depth in Chapter 5. Also, the effectiveness of the trained neural network on real-life low quality images from the rendezvous of the life extension mission for the Intelsat 901 satellite is assessed. Finally, the conclusions and some ideas for the further development of the approach are drawn in Chapter 6.

Chapter 2

Neural networks for image recognition

2.1 Introduction to neural networks

In the last few decades, due to the exponential improvement of the computing capabilities of microprocessors in accordance to Moore's law [10] and, above-all, to the exploitation of graphics card architecture, Machine Learning (ML) has gained increasing popularity. ML is defined as the study of computer algorithms that are able to automatically improve their performance on a certain task without the need for someone to explicitly code their behaviour. It has enabled the tackling of many different problems where conventional algorithms were ineffective and it proved to be efficient in a variety of different fields, from robotics to data analysis, from computational statistics to optimisation problems. For a more accurate definition and different examples the reader is encouraged to dive in the extensive literature on the topic to have a glimpse of the variety and usefulness of this kind of methodology [11].
Among the various approaches used in ML (e.g. genetic algorithms, decision

trees, bayesian algorithms, etc.) Artificial Neural Networks (ANNs) and, in particular, CNNs have proved to be very effective in the field of computer vision, which is the ability for the machine to identify objects represented in a picture.
As the name suggests ANN are inspired by biological nervous systems: input data are transformed in the desired output through intermediate layers of *nodes*, containing semi-elaborated results, which are the analogous of the neurons of the brain and are connected among them through problem-specific *relations*. Actually, the numerical input of each one of these nodes is multiplied by a *weight* factor and added to a *bias* value. Weights and biases are indeed what is changed and refined during the training process. The underlying idea is in fact not to build a new neural network for each problem we want to solve, but rather using the same consolidated and high-performing ones for different tasks, allowing the neural network to adapt to the problem at hand by modifying the weights during the training. It is said, in fact, that in the neural network there is an onward and a backward flow of data: the onward one is simply the flow of the input data following the net through successive nodes, layers and algebraic manipulation, whereas the backward flow is represented by the modification of the weights based on the closeness of the predicted result with the ground truth. Through this method, the very same network can be used to identify objects of very different kinds, provided the algorithm is trained on a relevant dataset.

Since numerical inputs have been mentioned, one could think of images as an exotic entity to deal with. However, for a computer, an image is in fact nothing more than a three dimensional array of numbers, where the first two dimensions are the number of the pixels of the image and the third dimension depends on the number of channels used to describe the colours. For an RGB encoded image they are three, expressing the levels of red, green and blue in each pixel, while it is just one for a grey scale image. Thus, images are naturally suitable to being fed to a ML algorithm. After a broad introduction of the topic, a more in depth description of the machine learning techniques exploited in this work is given in the next section.

Deep and Convolutional ANNs

The harder the task to be solved, the more complex the networks need to be and the higher the number of layers of neurons necessary to manipulate the raw input information to deliver the desired outcome. When the number of intermediate layers is greater than one, we talk of Deep Neural Networks (DNNs) (Fig. 2.1) and we define any layer between the input and output of the network as an hidden layer. This is exactly what happens when dealing

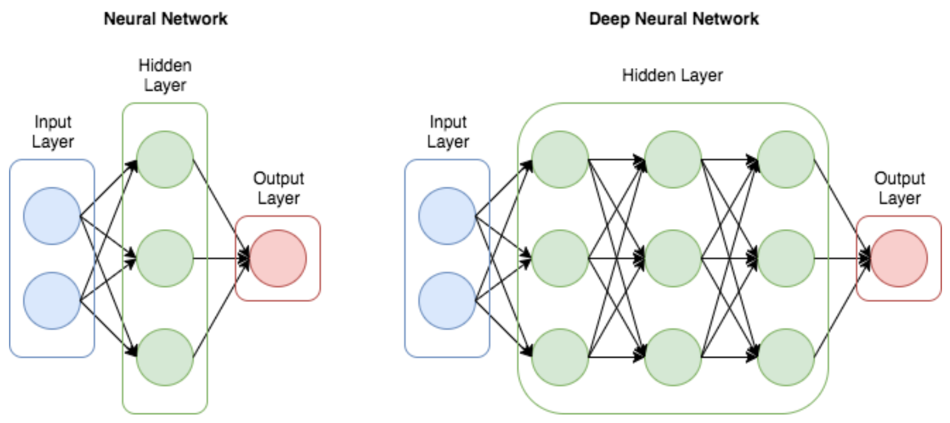


Figure 2.1: A very simple representation of a deep neural network. Blue nodes represent the input (the array of the image in the case of computer vision), the red one represent the output (the class of the object, its bounding box or its mask, as we will see later) and green nodes represent the hidden intermediate layers. The arrows represent the flow of the data through the relations. Image credit

with computer vision: images can be quite an heavy input for the network if they are high resolution and, although modern computers can easily deal with such an amount of data, recognising the objects in the picture still constitutes a very complex assignment. Furthermore, the machine needs to be able to distinguish a certain object not only by its features, but also independently on the dimensions or position it holds in the picture. One of the first approaches that was used to tackle this necessity was to include in the pipeline of the training algorithm some scripts that, starting from each of the images, generate new images with the object in different positions (Fig. 2.2). These practices are usually referred to as *data augmentation* techniques [12]. More data, however, makes the problem harder to solve and a bigger network is needed to learn more complicated patterns. Hence the use of DNNs in this field.

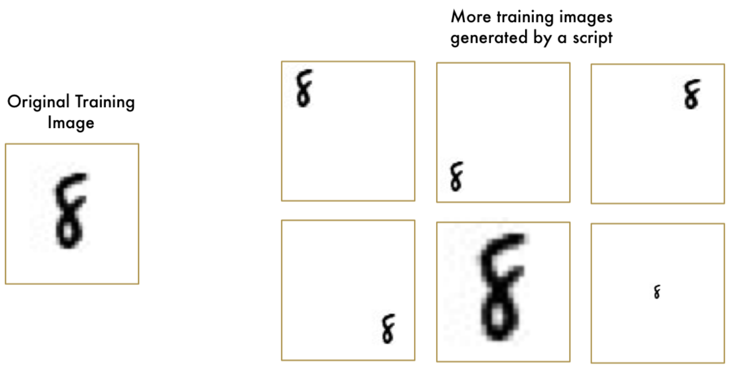


Figure 2.2: A possible representation of a technique to make the network able to recognise the object in every circumstance. This is actually an example of data augmentation, which will not be treated in depth in this work. Image credit

Although the idea has been around since the dawn of the artificial intelligence studies, dealing with such large neural networks only became feasible in the last decade, when the computing capabilities of graphics cards began to be exploited. The data in the neural network is in fact always represented in matrix form (both for the input and for the matrices containing the weights and the bias) and GPUs' architecture is explicitly designed to be efficient in matrix operation.
While the DNNs as explained up to now are a viable option, this still does not represent the state of the art in computer vision. They in fact require

massive amount of training data and need to be very complex because they still treat the very same object as a different one depending on the position in the image and the background it is against, thus failing if some different conditions are encountered in a picture. There is a smarter way to work the problem out: Convolutional Neural Networks (CNNs). The basic CNN is composed of five steps:

- *Extracting smaller tiles from the image*: using one of many possible techniques, smaller images are extracted from the original one, having care that the neighbouring ones overlap so that no information is lost on the continuity of the image.
- *Using a small ANN on each tile*: each of the smaller images of the previous step is fed to a small Neural Network, which is namely called *kernel*. The same weights are used on each tile, so that the simple NN highlights the more interesting ones, those where it is more probable to find a match for the objects that are being searched. In this way the number of weights to be trained is much less in CNNs than other ANN.
- *Saving the results in a new array*: The results for each tile are arranged to reproduce the same pattern the tiles have in the original image, so that the track from where the information comes from is not lost. Since the number of outputs of the small neural network is usually smaller than the number of pixels in each tile, a smaller array than the input image is obtained, which maps which part of the original image are the most interesting. This ends the convolutional part of the network.
- *Down-sampling* or *Pooling*: At this point, the array is usually still quite big, therefore various techniques can be used to reduce its size, for example *max pooling*. While those will not be explained in detail in this work, the important thing to mention for the explanation of this work is that all of them combine the outputs of a cluster of neurons of

the previous layer in a single neuron in the new layer, trying to preserve the most important bits of information from the original array.

- *Prediction*: The pooled array is then naturally fed to another neural network. At this point, an ANN like the ones previously discussed can be used. Those are called *fully-connected ANN*, since every neuron is connected to all the other ones in the next layer, to distinguish them from the convolutional ones and can finally infer if the image is a match or not.

As already mentioned, this is only the basic structure of a convolutional neural network that could be used to efficiently solve, avoiding the bottlenecks explained before, the simplest task in computer vision, i.e. checking whether or not an image belongs to the category against which it is being tested. A visual representation is available in Fig. 2.3. In reality, much more complex networks are used to solve a variety of problems, usually involving many convolutional and pooling layers in different combinations or up-sampling layers too. Some of these architectures are also preceded by other networks with very specific tasks, such as Region Proposal Networks (RPNs), as it is shown in the next section.

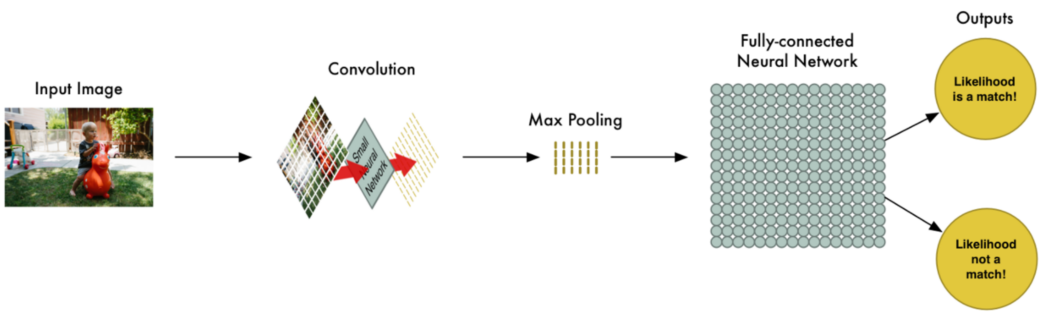


Figure 2.3: Basic scheme of a Convolutional Neural Network. Image credit

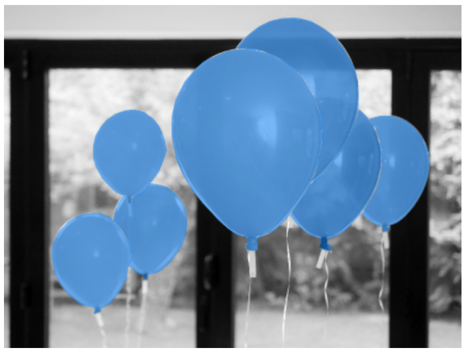
2.2 State of the art and proposed algorithms

Before going over the different algorithms that were considered for this work, it is appropriate to introduce the most common tasks they are asked to perform in computer vision. They are listed hereafter and shown in Fig. 2.4:

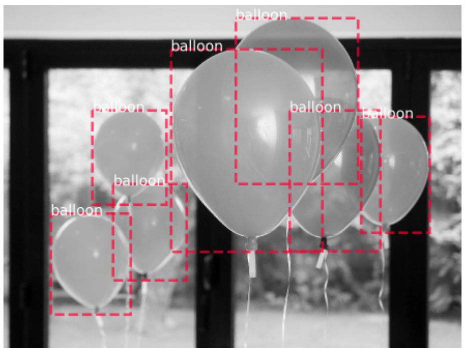
- *Image classification*: it is the task that has been used as an example up to now. An image classification algorithm just identifies which kind of object, if any, is primarily depicted in the image among the ones the network has been trained to identify.
- *Object detection*: the algorithm is capable to distinguish various objects in each image, drawing for each of them a bounding box.
- *Semantic segmentation*: rather than finding the envelope of each object as in the previous case, the algorithm is able to tell exactly which pixels belong to the object. However, this category of networks is not able to distinguish between objects of the same type, thus considering them as a single instance.
- *Instance segmentation*: it is an hybrid of object detection and semantic segmentation. It is, in fact, able to find the area of the picture covered by a certain object and distinguish among different instances of the same type.

It is important to notice that, although image classification may seem trivial, less versatile and therefore the less used of the introduced algorithms, it is of uttermost importance since it is the building block for all the other more complex tasks. In fact, this is not only true on a conceptual level, but also from a practical standpoint: the classification sub-networks, or classifiers, that will be mentioned later when illustrating the main algorithms that were taken into account for the present work, are often older, high performing networks designed at the dawn of the computer vision era with the specific and only purpose of image classification.

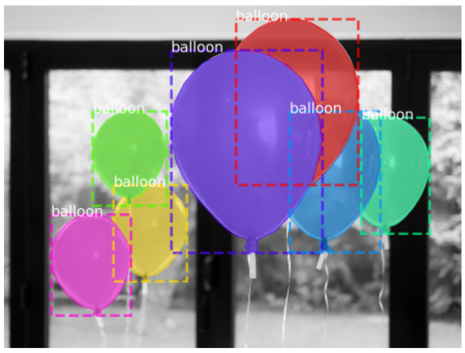
(a) ***Image classification*** *just says there is a balloon in the image.*



(b) ***Semantic segmentation*** *identifies all of the pixels belonging to the balloon class.*



(c) ***Object detection*** *identifies different instances of objects of the same class.*



(d) ***Instance segmentation*** *puts together the information of semantic segmentation and object detection.*

Figure 2.4: A graphical representation of the main kinds of approaches used in computer vision. Image credits

After the algorithm has been trained, it can be used to recognise the objects contained in the training dataset. To assess the performance of the algorithm, it is executed on sets of images which have not been used for the training. These sets are called *test datasets* and feature the very same kinds of entities (usually referred to as *classes* of objects) which are found in the training dataset.
Just like the training dataset, a testing dataset is also provided with a ground truth but in this case, rather than being used to monitor how the training is advancing and varying the weights accordingly, it is used to assess how close the prediction provided by the trained neural network is to the ground truth. What this last sentence actually means depends on the kind of prediction made: for image classification it just means checking if the object has been correctly identified or confused with some other category; in the case of objects detection, besides checking the class prediction, the evaluation accounts for how close the match of the predicted bounding box to the real one is; something similar is done in semantic or instance segmentation, with the obvious difference that the masks of the pixels recognised as part of the object are evaluated instead of the bounding boxes. What has just been conveyed in a qualitatively fashion, is quantitatively expressed as the Intersection over Union (IoU), which is the ratio between then intersection area of the predicted mask/box and the one highlighted in the ground truth over the union of the same two entities. This value is usually included, together with the class hypothesis, in the mathematical formulation (different for each algorithm) of the confidence on the prediction made. Among the many evaluation metrics used in machine learning, the most common and the ones which will be used in this work to compare the results are the Average Precision (AP) and the mAP. To understand them let us introduce some

other definitions first [13]:

$$\begin{aligned} Precision: \qquad & p = \frac{TP}{TP+FP} \\ Recall: \qquad & r = \frac{TP}{TP+FN} \end{aligned}$$

where TP stands for *true positives* and refers to the number of objects correctly identified, while FP are the *false positives*, i.e. instances identified as something they are not, and FN are the *false negatives*, which is the number of objects which are not identified. These quantities can be computed for each class of objects and lead to the definition of

$$Average\ Precision: \qquad AP = \int_0^1 p(r)\, dr$$

for each class. The definition is actually a little more complex than this and it involves the IoU and marking multiple thresholds on it, but this simplified explanation shall suffice for the aim of this work, for additional details see [13, 14]. The mean Average Precision (mAP) is then simply the average of the AP over the classes.

Once the general architecture of a CNN and the evaluation metrics have been introduced, it is possible to have a closer look at some specific algorithms. Object detection methods can be roughly subdivided in two macro categories: the ones based on region proposal, such as fast and faster R-CNN, and the ones based on a regression method, such as YOLO and SSD. In the next section, some of the most relevant examples of the two categories are analysed.

R-CNN, fast R-CNN and faster R-CNN

In 2014, Girshick et al. proposed an object detection algorithm that greatly improved the performances on the PASCAL VOC dataset (a dataset used as standard in the computer vision research field to assess the capabilities of new algorithms) while still being simpler than the best ones that were available at the time [8]. Being a combination of region proposal and CNNs, the method was called R-CNN and marked a revolution in the field, outperforming the algorithms that were mostly based on the sliding window technique [15]. The algorithm is based on three steps: first, selective search is used to identify the possible locations of the object in the image, then each of the candidate region is scaled to a fixed size and fed to a CNN which extracts the so-called *feature map* or *activation map*, the matrix that locates hidden features and patterns in the image. Finally, a Linear Support Vector Machine (SVM) is used to classify the windows and refine the bounding box to better match the real one.

While bringing to the table a remarkable improvement, R-CNN suffers a problem that cannot be overlooked, being the final objective building an algorithm capable of real-time detection: the selective search algorithm used for the region proposal is in fact really ineffective in using computing and storage resources, since it ends up repeating the same calculation multiple times [16]. Moreover, the classification algorithm is run on each of the proposed regions without taking care of sharing computation when dealing with overlapping regions.

This led to the development of the Fast R-CNN [17] algorithm, which can still be divided into three main steps: the image is first fed to a CNN to obtain a *convolution* feature map, then candidate regions are extracted by selective search and a second activation map is produced; finally, this latter activation map is passed through a single fully connected layer that classifies the object and corrects the bounding box. The improvement in terms of speed over the first R-CNN implementation lies in the fact that the selective search is not run over the whole image.

Nonetheless, the extraction of candidate regions still constituted a performance bottleneck since it accounted for the most part of the detection time, hence the Faster R-CNN algorithm [18] was proposed in 2016. Here, a fully convolutional Region Proposal Network (RPN) was first introduced making it possible to share it with the detecting network itself. In other words, the network is made up of two parts: the convolutional RPN, that generates region candidates, and the detector that was already used in the Fast R-CNN network, which classifies the object and performs border regression. Since the two parts share the convolutional layers, they could be considered as a single unified network and the region proposal is nearly cost-free. This greatly improves performances, reaching a detection rate of 5 frames per second without sacrificing accuracy. A simple visual representation of the network is shown in Fig. 2.5, please refer to the bibliography for a more precise explanation of the architecture layer by layer.

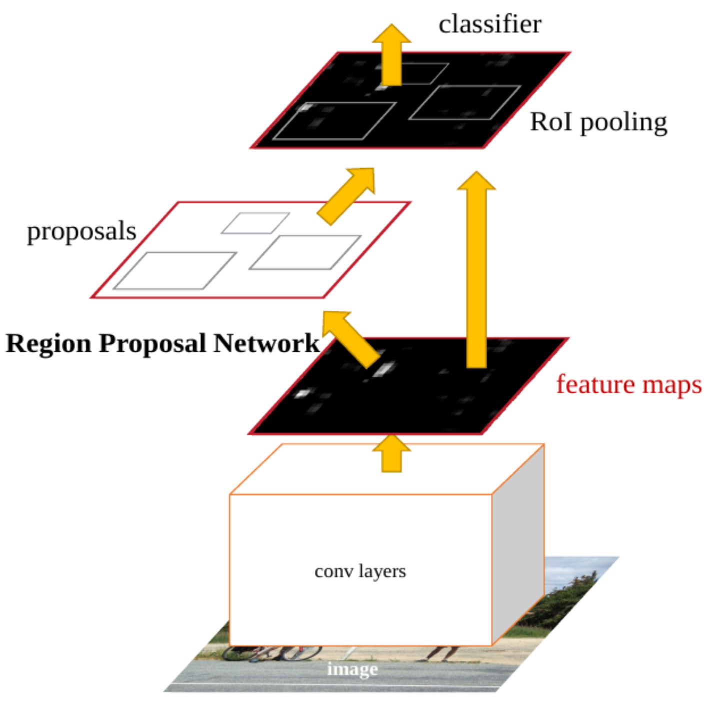


Figure 2.5: Scheme of the Faster R-CNN, a single, unified network for object detection. Notice how the convolutional layers are shared by both the Region Proposal Network and the classifier. [Image from the original paper]

You Only Look Once (YOLO)

Differently from the two-stages detection in the faster R-CNN algorithm, which, although being simpler than the previous implementation is still a quite complex network, YOLO features a one-stage detection algorithm. Said approach uses the whole graph as input to the network directly returning the position of the bounding boxes and the category of the objects without the need to first extract the Region of Interests (RoIs). Due to this peculiarity, it is obviously much faster than the solutions previously illustrated, since it was shown that the bottleneck in performance was caused by region proposal. Nonetheless, YOLO also retains high results in terms of accuracy [19].
The algorithm was first proposed by Redmon et al. in 2015 [20] and, just like what happened with R-CNN, it went through various revisions. The first iteration resizes the image to a fixed dimension, then subdivides it in a number of cells and feeds it to a single convolutional network, which understands if the centre of an object falls within a certain cell of the grid and computes the class-specific confidence on the object prediction for each box. Finally, the results are filtered, rejecting those with a lower confidence and adjusting the bounding box through regression.
While being really fast, the first version of YOLO, was not as accurate as the counterparts exploiting region proposal techniques. Nonetheless, by looking at the image as a whole, YOLO predicted way less false positives and was less prone to mistaking background patches as objects. Anyway, in order to improve overall accuracy, YOLOv2 [21] was published: the main differences that were introduced are multi-scale training (see Fig. 2.6), since the architecture of the network allowed to change the dimension of the input while training, the use of anchor boxes with direct location prediction and batch normalisation. While explaining the reasoning behind these decision and why they were possible would require a detailed explanation of the architecture of the network, which is out of the scope of the present work, the noticeable

result is that YOLOv2 was able to reach the precision of the opponents and being one of the best performing detection algorithm while still being the fastest and capable of real-time detection to a certain degree. Despite the many improvements, just like the first version, this struggled in recognising smaller objects.

However, with the release of SSD (see below) in 2016, it was outperformed in terms of accuracy, which led to the implementation of YOLOv3 [19]. This last iteration features a more complex underlying structure, thus trading off part of the speed of its predecessor for an improvement in accuracy. It anyhow still reaches 30 fps inference on images. This was obtained mostly using a better classifier and multi-scale prediction that is, just like what had already been done with the training, scaling the images to three different sizes throughout the pipeline at inference time. This approach proved useful in improving the detection of small objects too. The architecture for the network is shown in Fig. 2.6.

Single Shot Multibox Detector (SSD)

As already mentioned, the Single Shot Multibox Detector (SSD) is another single network object detection algorithm that established a big improvement in the image recognition scene when it was first proposed by Liu et al. in 2016 [22]. The algorithm, just like what happened with YOLO, is much simpler than the ones based on region proposal: the image and ground truth is fed to the network which extracts several feature maps of different dimensions; for each location in each activation map, a set of standard bounding boxes is proposed with different scales and aspect ratios (see Fig. 2.7) and, for each of them, both the offset from the ground truth box and the confidence on each object category is computed, producing adjustments to better mach the object shape.

This approach is retained at inference time, when the trained model is used

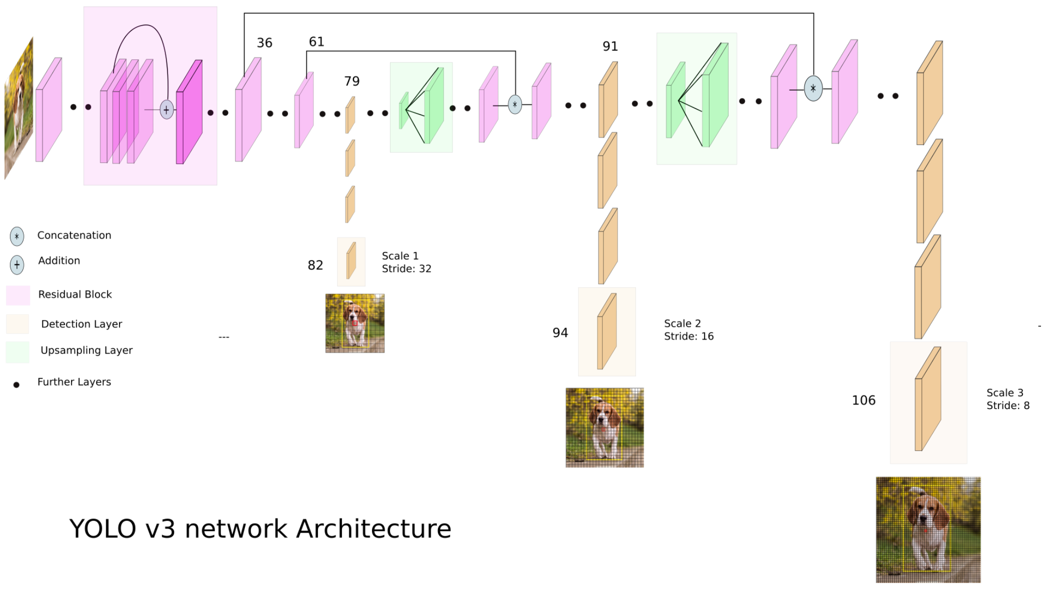


Figure 2.6: Architecture for the YOLOv3 algorithm. Notice how the image is scaled to three different dimensions throughout the pipeline due to the multi-scale training introduced in YOLOv2. Image credit

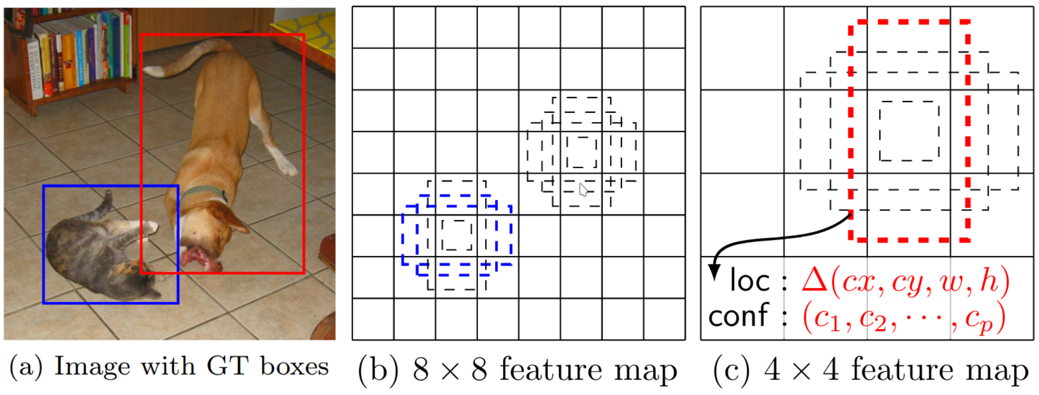

loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$
(a) Image with GT boxes
(b) 8×8 feature map
(c) 4×4 feature map

Figure 2.7: SSD is trained evaluating the deviation of default boxes proposed on differently sized feature maps from the ones expressed in the ground truth. [Image from the original paper]

to analyse new images: the network combines predictions from multiple feature maps with different resolutions to better handle objects of various sizes. Again, being a single ANN, SSD is really fast, making it a viable option for real-time video detection. Furthermore, as already mentioned, it outperformed the other algorithms of the same family in terms of accuracy at the time of publication, reaching performances similar to those of Faster R-CNN.

Region-based Fully Convolutional Networks (R-FCN)

As we have seen up to now, the main difference between region-proposal based algorithms and single-stage ones is that the first analyse the image on a per-region base, applying a subnetwork on each RoI without sharing computation, while the latter use a shared fully convolutional architecture independent on RoIs. The middle ground was explored by Dai et al., who proposed the R-FCN algorithm in 2016 [23].
It stems from some previous works proposing new and very efficient fully convolutional image classification networks: the idea was therefore to use them as classifiers of a fully convolutional object detection system too. However, this resulted in poor results, leading to a detection accuracy much worse than the classification one. The problem was partially solved by adding the RoI pooling layer from faster R-CNN in the middle of the detector. This highlighted that the problem was due to the need to increase translation invariance for image classification while respecting translation variance for object detection.
To obtain this result, Dai et al. implemented a set of position-sensitive score maps using a number of specialised convolutional layers. Conceptually it is like subdividing the image in a grid and inferring the position of the object by recognising his parts (e.g. a human face can be identified if an eye is found in a grid cell and the mouth in another one). The important thing that leads to better results in terms of speed with respect to Faster R-CNN

is that this maps are shared by the whole image and applied to the various RoIs proposed by the RCN, as shown in Fig. 2.8.

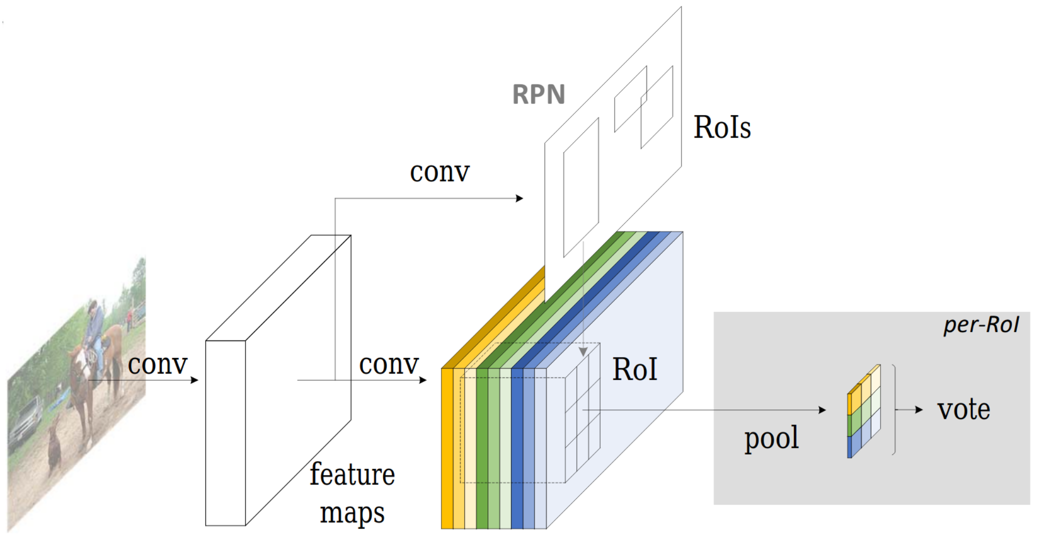


Figure 2.8: Overall basic scheme for the R-FCN algorithm. Each feature map, in different colours in the image, is specific to a certain region of the image. [Image from the original paper]

Mask R-CNN

All of the algorithms that have been shown up to now perform object detection. Indeed, instance segmentation could be a desirable feature in the field of On-Orbit Servicing (OOS) since it gives information on where the material actually is. In 2018, He et al., from the Facebook AI Research (FAIR) group, published Mask R-CNN [24], an instance segmentation algorithm that outperformed every other architecture devoted to the same task and that still represents the state-of-the-art in the field. Mask R-CNN is based on Faster R-CNN, just adding a branch for predicting an high-quality segmentation mask in parallel with the already existing pipeline for the bounding box recognition. This is different from what had been previously made in other works, such as DeepMask [25] and its evolution SharpMask [26] or Instance-sensitive FCNs [27], where the segmentation precedes the recognition. In this way, the computation overhead of Mask R-CNN with respect to Faster R-CNN is small and inference can be run at 5 fps, which is comparable to the speed of Faster R-CNN[1].

Comparison

When it comes to deciding which of the proposed algorithm has to be chosen, it obviously depends on which is best suited for the problem at hand and, if one or more options are suitable, the less demanding in terms of computational resources has to be preferred. In fact, on-board computers on space platforms usually have really limited computational power and storage. Obviously, concerning object detection, there is no point in taking into account the original implementation of R-CNN and Fast R-CNN since they do not provide any benefit over Faster R-CNN. The choice is then between Faster R-CNN on one hand and YOLO/SSD on the other. As it was shown, Faster R-CNN grants higher accuracy at the cost of a lower inference speed (although, using VGG-16 as the pretrained model for region proposal, the algorithm is only six times slower than YOLO, rather than hundreds of times), while YOLO and SSD are much faster since they are a single network and do not have the RPN to act as a bottleneck, but they are also less accurate. For the actual OOS of functioning satellites, real-time detection could be mandatory and therefore SSD or YOLO should be preferred. This work, however, is focused on the preliminary inspection of non cooperative objects, before any proximity action or manoeuvre is performed, therefore there is no need for real time interpretation of the images. Observation legs could indeed last for hours [5] and, therefore, higher accuracy is preferred over speed.

[1]Indeed, as it will be shown in Chapter 5, the code is still being actively developed and, in the present work, it was able to run at a speed doubled with respect to the one of the original paper.

This could imply choosing Faster R-CNN, but there are some other factors that are worth taking into account. Using instance segmentation with an algorithm such as Mask R-CNN, in fact, could be a desirable option since, as already mentioned, it gives an information on where the material actually is in the picture and this could be useful for the following approaching phase. In addition, with some previous knowledge of the object under study, this could be used to determine the mass distribution of the object even if it was damaged and it is therefore no more in its nominal configuration, and this is obviously useful for the determination of the attitude dynamic of the target satellite. All of this information can be retrieved at almost no additional computational cost implementing Mask R-CNN instead of Faster R-CNN, as already shown. Therefore, this is the network that has been chosen for the present work. In addition, Mask R-CNN has the advantage that it can be easily generalised to perform other tasks, if provided with suitable training datasets. Among these, pose estimation would be very useful for dynamics determination.
A more in-depth review of the Mask R-CNN algorithm is provided in the next section.

2.3 Mask R-CNN

As already explained, Mark R-CNN is based on Faster R-CNN, adding an extra branch to the algorithm architecture to pixel wise segment each instance of the objects the network is trained to recognise, extracting them without any background.
Just like its predecessor, it is a two stage algorithm: the first stage is responsible for generating object proposals, while the second one classifies proposals to generate bounding boxes and masks. What is innovative with respect to other segmentation algorithms is that the three tasks of detection, classifica-

tion and segmentation are conducted in parallel, that is that the classification to a particular class does not depend on the mask predictions (like it happened in other solutions [25, 26]).
In addition to adding the third branch to generate object masks, other two fundamental differences are introduced with respect to faster R-CNN: the use of Feature Pyramid Networks (FPNs) and the replacement of *ROIPool* with *ROIAlign*.
FPN is the network used for the generation of the feature maps that are then exploited by the RPN to propose RoIs (remember that, differently from the original R-CNN, Faster R-CNN does not run the RPN on the whole raw image but rather on the feature maps). In Faster R-CNN RoIPooling is the layer which is responsible of generating a uniform square matrix for each of the extracted RoIs (which in theory could be differently shaped).
In the following subsection, a description of the fundamental features of Mask R-CNN is given.

Image pre-processing The input to the network is a $w \times h \times 3$ matrix, where w and h are the the dimension of the image (i.e. the number of pixels per side) and 3 is the number of colour channels for red, green and blue. Before being sent to the network, the image is:

1. transformed in a 'zero-mean' array by subtracting the 3×1 mean vector, that is the mean of pixel values across all images for each channel;

2. re-scaled, keeping the original aspect-ratio, so that either the shorter size is equal to a chosen parameter called *target size* or the longer one is equal to another parameter called *maximum size*;

3. padded with blank pixels so that both sides of the image are multiples of 32 (this is needed when using FPNs). The pixels are added only to the right and bottom side of the image so that the image does not change

position in the reference system which has its origin in the upper-left corner.

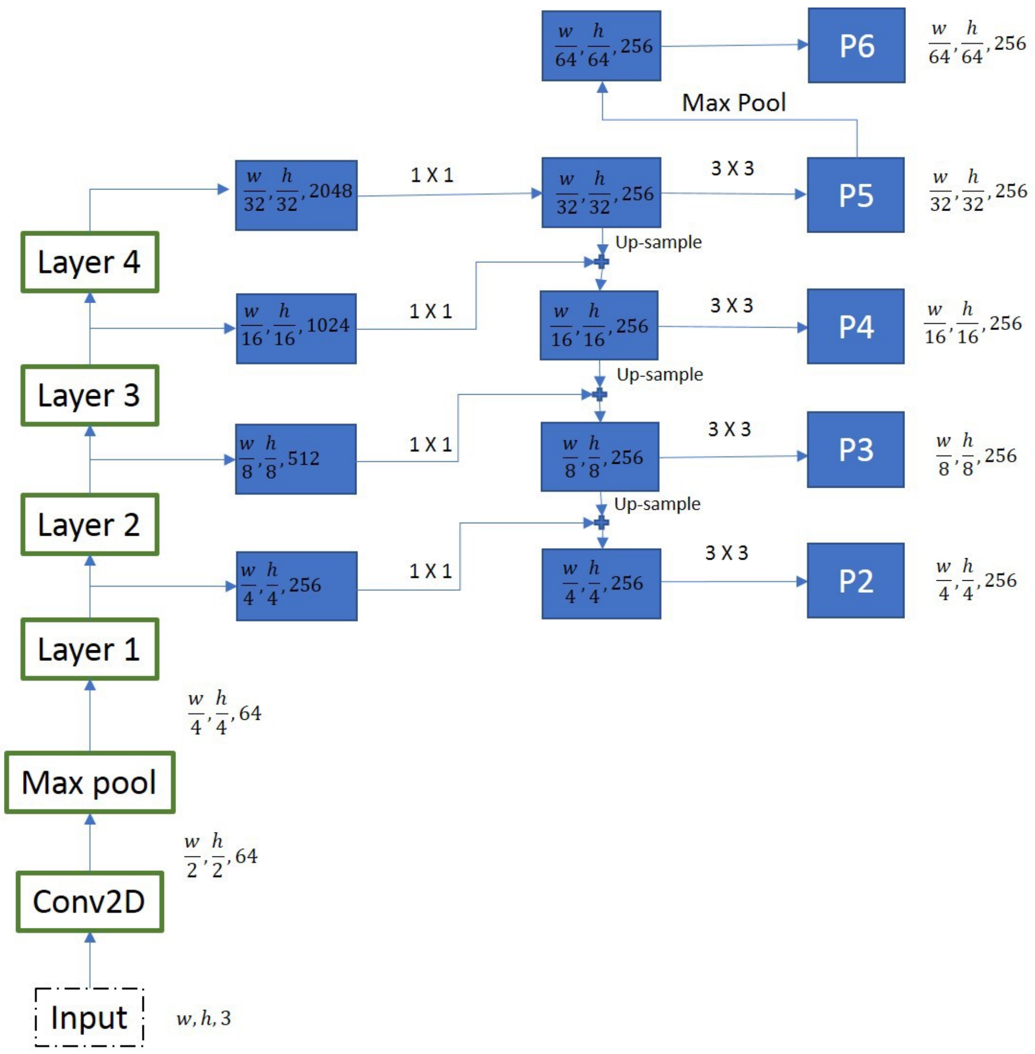

P6
Max Pool
P5
P4
P3
P2
1 X 1
3 X 3
Up-sample
Layer 4
Layer 3
Layer 2
Layer 1
Max pool
Conv2D
Input
$\frac{w}{64}, \frac{h}{64}, 256$
$\frac{w}{32}, \frac{h}{32}, 2048$
$\frac{w}{32}, \frac{h}{32}, 256$
$\frac{w}{16}, \frac{h}{16}, 1024$
$\frac{w}{16}, \frac{h}{16}, 256$
$\frac{w}{8}, \frac{h}{8}, 512$
$\frac{w}{8}, \frac{h}{8}, 256$
$\frac{w}{4}, \frac{h}{4}, 256$
$\frac{w}{4}, \frac{h}{4}, 64$
$\frac{w}{2}, \frac{h}{2}, 64$
$w, h, 3$

Figure 2.9: FPN feature maps extraction. Image credit

Feature Pyramid Network (FPN) backbone FPNs have shown to be effective backbones and to improve accuracy in many object detection algorithms [28]. In Mask R-CNN they are used to extract the different sized feature maps from which RPN extracts the Region of Interests.

As it can be seen in Fig. 2.9, after some convolutional and pooling layers, the input image goes through four layers, each doubling the number of intermediate feature maps channels (the third dimension) while halving their width and height up to the point they are $\frac{1}{32}$ of the original size, that is why the padding is needed in the pre-processing. Each of the feature maps then goes through a 1×1 convolution (that means that the first two dimensions are not changed) which brings the number of channels down to 256. In this way, a process called *top-bottom pathway* can be applied: starting from the smaller activation map, it is up-scaled to match the size of the map from the previous layer and added element-wise to it; all of the outputs from this process are subjected to a 3×3 convolution to create the final four maps (P2 through P5), while the last map P6 is obtained from P5 by max pooling (that is, for each group of four pixels of the original map, only the highest value is retained).

Region Proposal Networks (RPNs) Each of the maps obtained from the FPN, is passed through an activation layer, in particular a Rectified Linear Unit (ReLU), which is the most popular type of activation function since it enables a better training of deeper neural networks [29]. It is indeed a very simple function, shown in Fig. 2.10.

The output is then processed through a 3×3 convolutional layer and fed to two different branches, one dealing with classification score and the other with the bounding box regressor. At this point, a certain number (chosen a priori) of possible regions of interests with different aspect ratios is taken into account and evaluated. The important thing is that these do not have to be scaled on different sizes since the feature maps obtained with FPN are already suited to take care for objects of different sizes. Therefore, the number of channels is quite limited: it is just one times the number of aspect ratios (3 by default) for the classification and four times the number of aspect

$$f(x) = x^+ = \max(0, x)$$

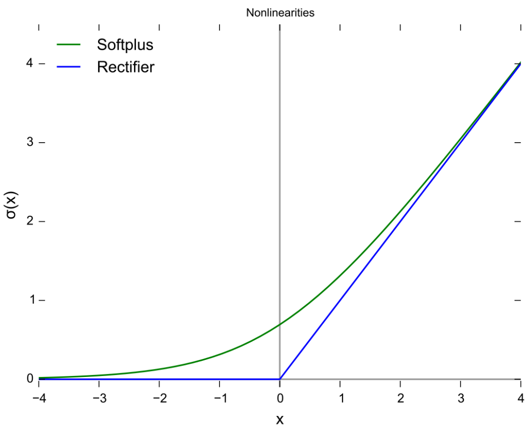


Figure 2.10: Rectifier function, in blue, and its variant Softplus (in green). Image credit

ratios for the bounding box regression (since the bounding boxes are defined by width, height and the two coordinates of the anchor point). Then the removal of invalid boxes and consolidation of the RoIs to be passed to the second stage of the algorithm proceeds just like in Faster R-CNN. Please refer to the relevant paper for more details [24].

Classification and bounding box regression The second stage of the algorithm is nearly identical to the original in Faster R-CNN. The only noticeable difference, as already mentioned, is the substitution of the RoI Pooling layer in favour of the new RoI Align. RoIPool is a standard operation in the field of deep neural networks which was first introduced in Fast R-CNN [17]. This operation has the objective of extracting a small square feature map for each RoI (which have in general different sizes) from the original activation map on which the FPN has been run, uniforming in this way the output so that the following part of the pipeline can be standardised.
The problem with RoIPooling is that it involves several quantization steps and numerical rounding in the evaluation of the values of the new map from those of the original one, introducing misalignment between the RoI and

the extracted features: while this is efficient and does not pose any problem when dealing with object detection since the classification and bounding box regression are robust to small translations, it is a major problem in instance segmentation where pixel to pixel correspondence is of uttermost importance in order to have good results.
RoIAlign avoids quantization and rounding and uses bilinear interpolation to compute the value of the new map in a certain point starting from the four closest points in the original activation map.

Masking In the bounding box prediction, after the FPN-RoI mapping, a reshape is performed followed by two fully connected layers. This is bypassed in mask identification since it would lose the spatial information needed for the mask generation. The extracted features are passed through the predictor, which involves a deconvolution operation giving an output with number of channels equal to the one of the classes. In fact, since the mask prediction is completely independent form the classification task, a mask for each class and each feature is proposed. Only at this point, there is the conjunction between the two branches and the prediction made in the box branch is used to select the correct channel in the mask head for each proposal. The mask itself is defined through a threshold parameter which can be tuned (default 0.5): if the value for the pixels is higher than the threshold, the object is assumed to be present in that pixel, else, absent.
A post-processing phase follows: the mask tensors are padded to avoid boundary effects in the up-sampling operation, the mask is resized according to the input image dimension and the bounding box coordinates are re-scaled according to the new mask and converted to the nearest integer in order to be correctly mapped into the pixel grid.
A complete overview of the architecture and the data flowing in it is provided in Fig. 2.11.

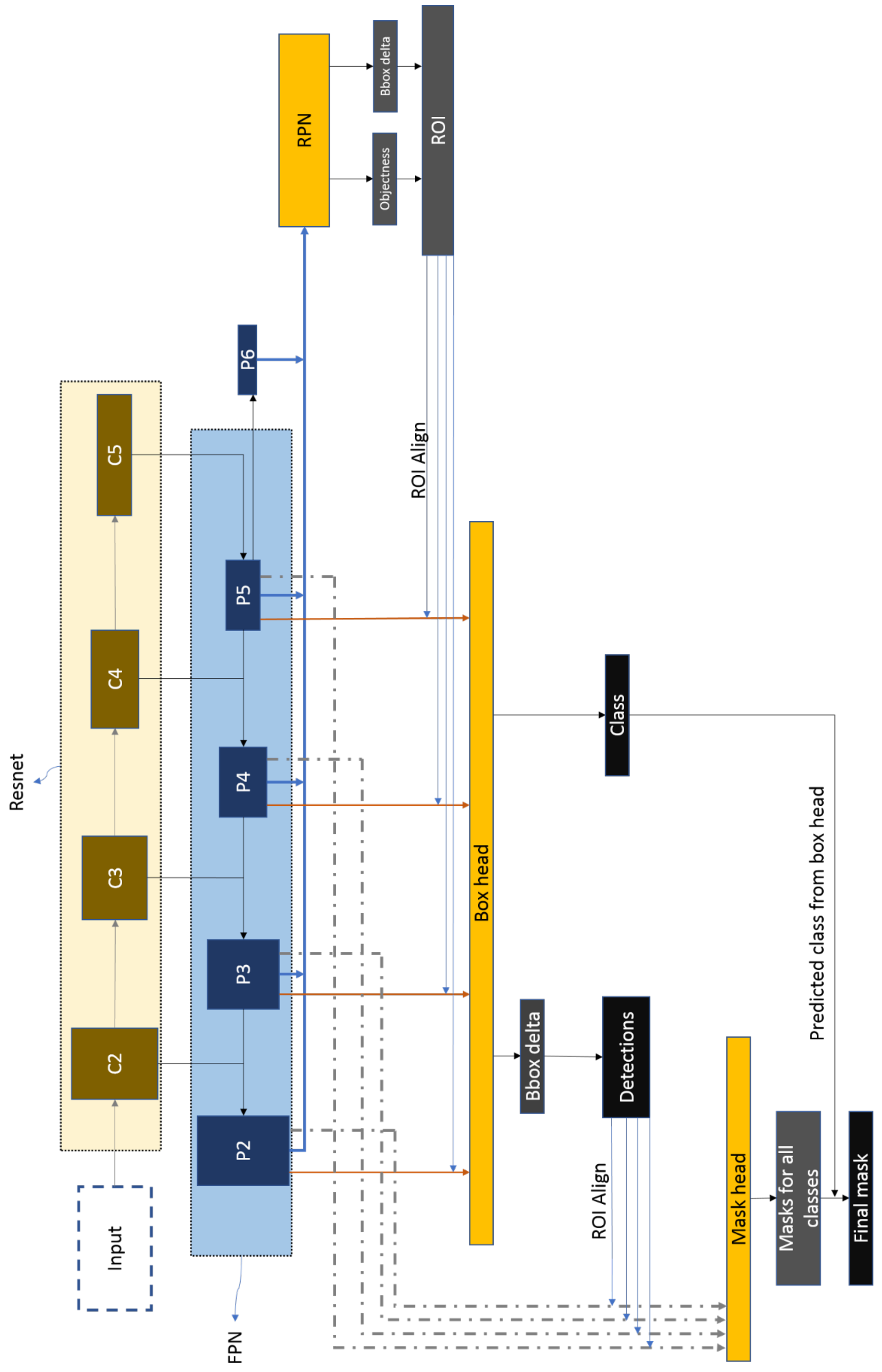


Figure 2.11: Simplified representation of the Mask R-CNN architecture. Image credit

Chapter 3

Relative orbital motion for inspection missions

Formulating the kinematics and dynamics of a spacecraft with respect to a point other than the main attractor is useful for a variety of space applications, for example rendezvous and docking, formation flying and even inspection missions, which are the focus of the present work.

In the latter case, in particular, a class of relative orbits is of interest, which are the ones governed by relative motion equations for which a closed form analytical solution exists. The peculiarity of these desired orbits is that the satellites are again in the same relative position after one orbital period of the target.

In this chapter, the theory behind relative motion and repeating relative orbits, which are the closed form solutions of the linearized formulation, is introduced, outlining the derivation process for the simplified case of a circular target orbit and small distance between the two satellites when compared to the radius of the orbits. Then, some of the most typical orbits of this type are briefly illustrated.

3.1 Repeating Relative Motion theory

While the relative motion equations are quite complex and do not have, in general, a closed form solution, it is interesting to quickly illustrate the general framework that is useful to introduce the repeating orbits and the derivation process.
The objective, taking as an example a simple system composed only by two spacecrafts, is to express the relative motion of the chaser with respect to the target. This formulation has two main advantages: it is very intuitive, since it decouples the relative motion from the orbit around the planet[1], and explicitly involves quantities that are of primary importance in a OOS mission, such as the distance between the spacecrafts.

Let us first introduce the reference frame used to describe the relative equations of motion. Among the various options that could be used, this work will follow the formulation proposed by H. Schaub [30], which uses the Hill coordinates frame [31].
Its origin is at the target satellite position[2] and its orientation is given by the vector triad $\{\hat{\mathbf{e}}_r, \hat{\mathbf{e}}_\theta, \hat{\mathbf{e}}_h\}$ shown in Fig. 3.1. The vector $\hat{\mathbf{e}}_r$ is in the direction of the position vector of the target spacecraft, while the $\hat{\mathbf{e}}_h$ is parallel to the orbit momentum vector, i.e. perpendicular to the orbital plane. The $\hat{\mathbf{e}}_\theta$ vector completes the right-hand system.

Mathematically, the definition of the $\mathcal{O}$ frame is as follow

$$\begin{aligned} \hat{\mathbf{e}}_r &= \frac{\mathbf{r}_t}{r_t} \\ \hat{\mathbf{e}}_\theta &= \hat{\mathbf{e}}_h \times \hat{\mathbf{e}}_r \\ \hat{\mathbf{e}}_h &= \frac{\mathbf{h}}{h} \end{aligned} \tag{3.1}$$

[1]The motion of the chaser in the inertial frame will in fact be the sum of the one of the target and the relative motion.

[2]Notice however that it is not necessary that the target position is occupied by a physical satellite, it can be simply an orbiting reference.

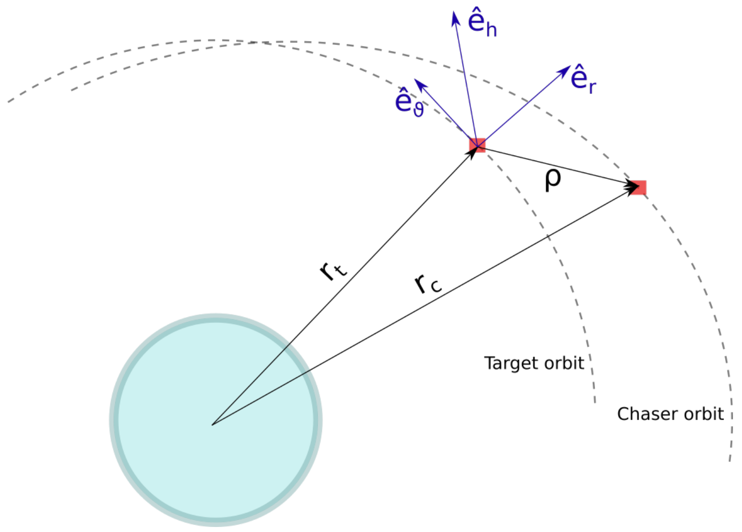


Figure 3.1: Hill's (LVLH) reference frame.

where $\mathbf{r}_t$ is the position vector of the target spacecraft and $\mathbf{h} = \mathbf{r}_t \times \dot{\mathbf{r}}_t$ is the angular momentum vector of the target spacecraft. Notice how, in the case of circular target orbit, the $\hat{\mathbf{e}}_r$ unit vector is in the direction of the radius of the orbit and the $\hat{\mathbf{e}}_\theta$ unit vector is parallel to the velocity of the spacecraft. This frame is also often referred to as the Local-Vertical Local-Horizontal (LVLH) frame.

The relative position vector pointing from the target spacecraft (origin of the reference frame) to the chasing satellite can be expressed as

$$\begin{aligned} \boldsymbol{\rho} &= \{x, y, z\}^T \\ \boldsymbol{\rho} &= x\hat{\mathbf{e}}_r + y\hat{\mathbf{e}}_\theta + z\hat{\mathbf{e}}_h \end{aligned} \tag{3.2}$$

As already mentioned, the advantage of using Hill frame coordinates is that the physical relative orbit dimensions are immediately apparent from these coordinates. The (x, y) coordinates define the relative orbit motion in the target orbit plane. The z coordinate defines any motion out of the target orbit plane.

3.1.1 Clohessy–Wiltshire Equations

Let us now briefly outline the derivation process of the equations describing the chaser motion in such a reference system, which are known as *Clohessy–Wiltshire equations* [32].
The inertial position of the chasing spacecraft, $\mathbf{r}_c$, can be expressed as

$$\mathbf{r}_c = \mathbf{r}_t + \boldsymbol{\rho} = (r_t + x)\,\hat{\mathbf{e}}_r + y\hat{\mathbf{e}}_\theta + z\hat{\mathbf{e}}_h \tag{3.3}$$

where $\mathbf{r}_t$ is the inertial position of the target RSO, and the angular velocity of the rotating frame $\mathcal{O}$ relative to the inertial frame $\mathcal{N}$ as

$$\boldsymbol{\omega}_{\mathcal{O}/\mathcal{N}} = \dot{f}\hat{\mathbf{e}}_h \tag{3.4}$$

where f is the true anomaly of the target spacecraft, that is the one of the rotating frame.
Making the only assumption of Keplerian motion, i.e. neglecting the disturbing and external forces, being $\mathbf{h}$ constant, deriving two times the chaser position and using the orbit equation of motion

$$\ddot{\mathbf{r}}_t = -\frac{\mu}{r_t^3}\,\mathbf{r}_t \tag{3.5}$$

the equations describing the relative motion are found in the form

$$\begin{aligned} \ddot{x} - 2\dot{f}\left(\dot{y} - y\frac{\dot{r}_t}{r_t}\right) - x\dot{f}^2 - \frac{\mu}{r_t^2} &= -\frac{\mu}{r_c^3}\,(r_t + x) \\ \ddot{y} + 2\dot{f}\left(\dot{x} - x\frac{\dot{r}_t}{r_t}\right) - y\dot{f}^2 &= -\frac{\mu}{r_c^3}y \\ \ddot{z} &= -\frac{\mu}{r_c^3}z \end{aligned} \tag{3.6}$$

Being the Keplerian motion the only assumption introduced up to now, the equations in Eq. (3.6) hold true for arbitrarily large relative orbits and for any value of the eccentricity of the target satellite orbit.
Assuming, instead, that x, y and z are small compared to the target orbit radius r_t, some components can be neglected during the derivation process;

furthermore, assuming that the target orbit is circular, the rate of change of the true anomaly $\dot{f}$ becomes constant and equal to the mean angular motion. The equations simplify to the form known as *Clohessy–Wiltshire equations* [32]:

$$\begin{aligned} \ddot{x} &= 2n\dot{y} + 3n^2 x \\ \ddot{y} &= -2n\dot{x} \\ \ddot{z} &= -n^2 z \end{aligned} \tag{3.7}$$

where

$$n = \sqrt{\frac{\mu}{a_t^3}}$$

is the mean angular motion of the target spacecraft (being a_t the semi-major axis of the target satellite orbit). Notice how the out-of-plane component of the relative motion behaves as a simple mass-spring system.

Under these assumptions, in fact, the equations actually have a closed form solution [32]:

$$\begin{aligned} x(t) &= (4 - 3\cos(nt))\, x_0 + \frac{\sin(nt)}{n}\dot{x}_0 + \frac{2}{n}(1 - \cos(nt))\, \dot{y}_0 \\ y(t) &= 6(\sin(nt) - nt)\, x_0 + y_0 - \frac{2}{n}(1 - \cos(nt))\, \dot{x}_0 + \frac{4\sin(nt) - 3nt}{n}\dot{y}_0 \\ z(t) &= z_0 \cos(nt) + \frac{\dot{z}_0}{n}\sin(nt) \end{aligned} \tag{3.8}$$

where the quantities marked by the 0 subscript are the initial conditions on the position and velocity vector of the chasing spacecraft in the LVLH frame. The system can also be rewritten as

$$\begin{pmatrix} \delta\mathbf{r}(t) \\ \delta\mathbf{v}(t) \end{pmatrix} = \underbrace{\begin{bmatrix} \mathbf{M}(t) & \mathbf{N}(t) \\ \mathbf{S}(t) & \mathbf{T}(t) \end{bmatrix}}_{\mathbf{\Phi}(t)} \begin{pmatrix} \delta\mathbf{r}_0 \\ \delta\mathbf{v}_0 \end{pmatrix} \tag{3.9}$$

where the matrix labelled as $\mathbf{\Phi}(t)$ is time-varying and is referred to as the *state transition matrix* for the CW equations.

Let us now discuss the assumptions made. While neglecting the disturbing forces is something which has to be carefully taken care of during the trajectory modelling, assuming that no external controlling forces are applied to the spacecraft is actually not a restrictive hypothesis: in many applications, having the inspector's relative motion to naturally repeat is a desirable feature in order to minimise fuel consumption. The derived equations still hold true if the spacecraft state at the end of the impulsive manoeuvre operated to put it in the relative orbit is used as initial condition for the propagation of the relative orbit. Instead, the equations are not suitable for manoeuvres performed with continuous low thrust[3].
Finally, assuming that the distance between the spacecrafts is much smaller than the radius of the orbit of the target satellite is a reasonable assumption for an inspection mission, therefore the CW equations are naturally suited for the analysis of such scenarios.

3.1.2 Repeating relative orbits

A single fly-by near the target object is usually not enough to gather all of the needed information when dealing with the kind of inspection missions that is being used as a reference for this work. Moreover, it is completely useless when also having to perform rendezvous and docking or any other proximity operation. It is therefore necessary for the relative motion to be bounded.
As it can be seen in Eq. (3.8), the out-of-plane component $z(t)$ and the radial component $x(t)$ already satisfy this requirement for any initial condition. The same is not true for the along-track component $y(t)$ which includes a secular term

$$y_{sec} = -3t\dot{y}_0 - 6ntx_0 \tag{3.10}$$

[3]A different formulation of the equations, valid for continuous thrust, is actually possible. However, it does not have a simple analytical solution for any manoeuvre.

which will grow indefinitely as time passes. Therefore, in order to have a bounded relative motion and a repeating relative orbit, the condition

$$y_{sec} = 0$$

has to be imposed.
Substituting $\dot{y}_0 = -2nx_0$ into Eq. (3.8), the equations governing the relative unperturbed bounded motion of a chasing spacecraft around a close target moving on a circular orbit are

$$\begin{aligned} x(t) &= \cos(nt)x_0 + \frac{\sin(nt)}{n}\dot{x}_0 \\ y(t) &= -2\sin(nt)x_0 + y_0 - \frac{2}{n}\left(1 - \cos(nt)\right)\dot{x}_0 \\ z(t) &= z_0\cos(nt) + \frac{\dot{z}_0}{n}\sin(nt) \end{aligned} \tag{3.11}$$

From a practical standpoint, as it can be intuitively understood, the condition that grants repeating relative motion is that of matching orbital periods between the two spacecrafts. But the orbital period, P, is function of the semi-major axis, a, of the orbit as

$$P = 2\pi\sqrt{\frac{a^3}{\mu}}$$

However, the semi-major axis also defines the energy of the orbit

$$\epsilon = -\frac{\mu}{2a} = \frac{v^2}{2} - \frac{\mu}{r} \tag{3.12}$$

Therefore any two spacecrafts that have the same orbital energy are in a periodic relative motion around each other. Furthermore, any orbital manoeuvre that either changes the shape of the orbit or its plane without changing its energy causes the chaser to be on a repeating orbit around the target. Moreover, looking at the second identity in Eq. (3.12) another observation can be made. Consider the case in which the target and chaser spacecraft have the same velocity magnitude and radial distance from the planet, such as for example if they are on the same circular orbit: any manoeuvre that changes the velocity direction without affecting its magnitude (at least as a first approximation) puts the chasing spacecraft in a repeating relative orbit around the target. This can be considered true, for example, when providing a Δv in the out-of-plane or radial direction since, for an inspection mission, it is usually 3 orders of magnitude smaller than the typical spacecraft velocity in Low Earth Orbit (LEO). Therefore, the change in magnitude of the velocity vector is negligible with respect to the change in direction. This is not true when the thruster firing happens in the along-track direction.

3.2 Overview of possible relative trajectories

For the problem discussed in this work, high performance on the trajectory control such as the ones that can be obtained using continuous thrust would probably be useless and therefore trajectories exploiting the natural relative motion can be preferred. Among them are the football and oscillating orbits. The following sections briefly illustrate their characteristics and how they can be obtained. All other periodic natural relative motion patterns are a combination of these two.

3.2.1 Football orbits

Assume again, for simplicity, that the target and chaser spacecrafts are on the same circular orbit. As already mentioned, if a thruster is fired proving a Δv variation in the radial direction, it can be assumed that the energy of the orbit (and hence the semi-major axis) does not change, but the shape of the orbit changes, becoming slightly elliptic. This results in a relative motion of elliptic shape as shown in Fig. 3.2.

Before moving to the analysis of the equations, the motion can be intu-

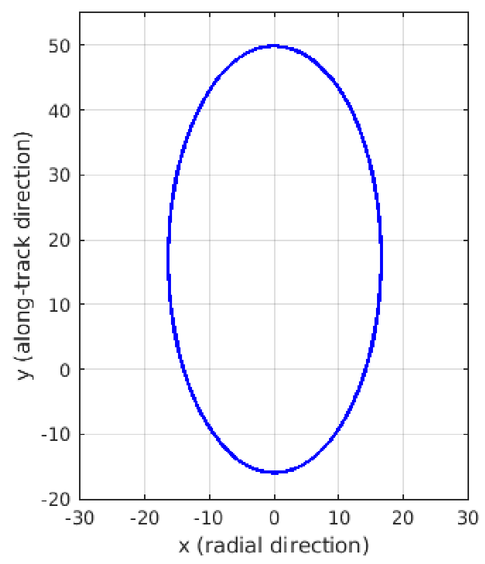


Figure 3.2: Football orbit representation in the LVLH reference frame. Values are in meters and just representative: obtained with 50 m initial along-track offset on a 700 km altitude target orbit, 1 m/s impulse in the radial direction.

itively understood by looking at the vis-viva equation

$$v^2 = \mu \left(\frac{2}{r} - \frac{1}{a} \right) \tag{3.13}$$

that is derived from Eq. (3.12). Assume that the chaser spacecraft precedes the target along the orbit and that the Δv firing is along the positive direction of the $\hat{\mathbf{e}}_r$ axis. The thruster burn that was just mentioned would cause the chaser spacecraft to increase its altitude, that means increasing r. This in turn would cause the satellite to decrease its velocity as shown in Eq. (3.13) and in the LVLH frame the chaser seems to rise upward and slowly drift back towards the target. Once the target reaches its apogee, however, it has a greater speed than the target satellite and therefore it gets closer to it again moving underneath it.

From a mathematical point of view, assuming that at the initial time the chaser spacecraft is some distance y_0 in front of the target in the along-track direction means that the initial relative altitude is $x_0 = 0$. Furthermore, assuming that the impulse manoeuvre does not have any along-track component means $\dot{y}_0$. Substituting these initial conditions in Eq. (3.11) yields

$$\begin{aligned} x(t) &= \frac{\sin(nt)}{n}\dot{x}_0 \\ y(t) &= y_0 + \frac{2}{n}\left(\cos(nt) - 1\right)\dot{x}_0 \end{aligned} \tag{3.14}$$

which is, as expected, the sum of a cyclic motion in the radial direction and an oscillatory motion along the tangential direction with an offset equal to the initial condition y_0, which results in the football-shaped relative motion shown in Fig. 3.2. Assuming that the initial orbit is the same for both the target and the chaser and that no Δv component is provided in the off-track direction ($\dot{z}_0 = 0$), it is also found that $z(t) = 0$ for any time, which means that the relative orbit belongs to the same plane of the spacecrafts orbits.
It can be demonstrated that the major and minor axis of the relative orbit ellipse are directly related to the magnitude of the velocity change and they are always one twice the other.

3.2.2 Oscillating orbits

The oscillating orbit is another fundamental type of relative orbital motion which results in an oscillatory trajectory of the chasing spacecraft along the direction perpendicular to the orbital plane of the target, while the other two components of the target-chaser vector remain constant, as shown in Fig. 3.3.

It results from a pure orbital plane change of the chaser, without modifying the shape of the orbit, obtained by a Δv in the cross-track direction z. Consider, as an example, the same situation illustrated for the football orbit where the chasing spacecraft is initially somewhere in front of the target satellite. This, again, translates in the initial conditions $x_0 = z_0 = 0$ and $y_0 \neq 0$. An impulse manoeuvre in the cross-track direction results in $\dot{x}_0 = \dot{y}_0 = 0$ and $\dot{z}_0 = \Delta v$.

Substituting these initial conditions in Eq. (3.11), it is found that

$$\begin{aligned} x(t) &= 0 \\ y(t) &= y_0 \\ z(t) &= \frac{\Delta v}{n}\sin(nt) \end{aligned} \tag{3.15}$$

which is exactly the mathematical representation of the motion described above. The maximum distance covered in the z direction is obviously

$$\Delta z = \frac{\Delta v}{n}$$

from which, if the mission requirements are expressed as a maximum distance to be travelled in the cross-track direction, the condition on the impulse magnitude $\Delta v = n\Delta z$ is found.

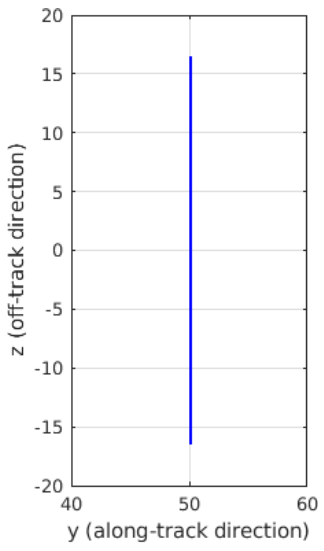


Figure 3.3: Oscillating orbit representation in the LVLH reference frame. Values are in meters and just representative: obtained with 50 m initial along-track offset on a 700 km altitude target orbit, 1 m/s impulse in the off-track direction.

3.2.3 Other relative orbits

As already mentioned, other types of relative repeating orbits can be obtained combining football and oscillating orbits. A typical example is the one of the *inclined football orbits*. These are obtained, starting from the same initial scenario already introduced in the previous sections, imposing an impulsive manoeuvre with Δv components both in the radial and off-track direction. The only difference with the initial condition for a football orbit like the one explained in Section 3.2.1 is that $\dot{z}_0 \neq 0$ in this case. This result in the following set of equations

$$\begin{aligned} x(t) &= \frac{\sin(nt)}{n}\dot{x}_0 \\ y(t) &= y_0 - \frac{2}{n}\left(1-\cos(nt)\right)\dot{x}_0 \\ z(t) &= \frac{\dot{z}_0}{n}\sin(nt) \end{aligned} \tag{3.16}$$

which, as it can be seen in Fig. 3.4, exhibits an oscillatory motion in the z direction too. Differently from what was showed in Section 3.2.1, the overall relative motion is no more co-planar to the the target inertial orbit in this case.

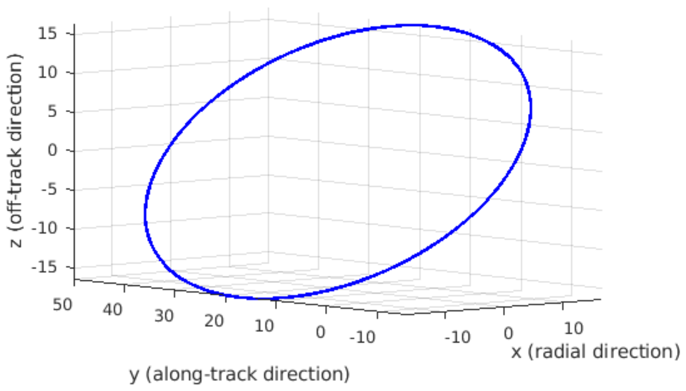


Figure 3.4: Inclined football orbit representation in the LVLH reference frame. Values are in meters and just representative: obtained with 50 m initial along-track offset on a 700 km altitude target orbit, 1 m/s impulse in both the radial and off-track direction.

While many other types of relative orbits are possible, this shall suffice as an example. It is important to notice, however, how such kind of relative repeating trajectories are often employed to design passively safe inspection orbits as in [33].

Chapter 4

JINS: a synthetic images generator

Before introducing the working principles of the Jins Is Not a Simulator (JINS) software, it is worth understanding why providing a large training dataset to a computer vision algorithm is so important. As it was shown in Chapter 2, even if, when referring to CNNs, we are talking about one of the branches of Artificial Intelligence (AI), other than the scientists writing the algorithm themselves, there is really nothing smart about what an ANN does: it simply takes an image as input, that is a 3D array of brightness values, and blindly manipulates those numbers through a series of operations that are dictated by the developer and that often do not have any physical meaning. The only thing that the network is instructed to do is to modify some parameters (the weights) so that the output at the end of the pipeline gets closer and closer to what the user states to be true. Therefore, the only way to make the network capable of correctly detecting the object with the highest possible confidence is giving it the chance to train on the widest range of conditions possible, hence the necessity of big training datasets. Furthermore, some conditions can constitute major challenges for the recognition and it is worth taking care of including them in the dataset. Among them, some are shown in Fig. 4.1, and here reported as defined in [34]:

- *viewpoint variation*: a single instance of an object can be oriented in many ways with respect to the camera;
- *scale variation*: visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image);
- *deformation*: many objects of interest are not rigid bodies and can be deformed in extreme ways;
- *occlusion*: the objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible;
- *illumination conditions*: the effects of illumination are drastic on the pixel level;
- *background clutter*: the objects of interest may blend into their environment, making them hard to identify;
- *intra-class variation*: the classes of interest can often be relatively broad, such as 'chair' or, in our case, 'solar panels' and 'antenna'. There are many different types of these objects, each with their own appearance.

Due to the complexity of the problem, it is therefore quite evident how extensive datasets are needed in order to build effective recognition systems. As already stated in the introduction, the objective of this work, in particular, is recognising the main elements of a non-cooperative artificial RSO. So, the question is: does such a kind of training dataset already exist? The short answer is no. Actually, Xu et al. built a similar dataset scraping the web for various types of satellites images (even unrealistic ones) and manually labelling each of the solar panels that appeared in the images [9]. Although

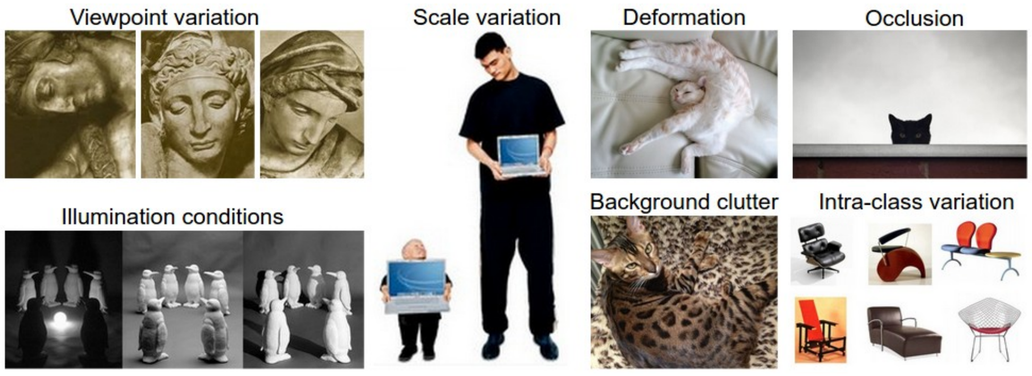


Figure 4.1: A visual representation of some of the most common challenges in computer vision. Image credit

fitting the requirement for their work, such a kind of approach is suboptimal if the objective is to build a system that is actually useful in real life operations for a number of reasons:

- it is not easily scalable: the labelling effort increases linearly with the complexity of the problem at hand and, as we have seen, the problem can get much more complex quite easily, e.g. having to deal with objects that are more elaborate and varied than solar panels would probably require a larger training dataset or dealing with more classes would require labelling more objects for each image;

- it is imprecise for instance segmentation: while labelling an object for object detection only requires the user to draw a bounding box, instance segmentation requires the drawing of polygon that is much more time consuming and prone to little offset errors that could overall cause a slight reduction in the performance;

- it is difficult to be tailored on a per mission basis: in the case of a mission towards a known target it offers no real advantage, since the dataset would have to be built in the exact same way, labelling images of the target satellite shot before the launch (if they are available, which may easily not be the case).

Ideally, an optimal solution to the problem would be the one that is flexible and requires no additional effort by the user independently of the dimension and complexity of the dataset to be built.
The idea proposed in the current work is to use CAD models of satellites and a 3D modelling program, such as the ones used by 3D artists in computer graphics, to recursively generate both images from different viewpoints, in different lighting conditions, with occlusion, etc. and the relative labelling at no extra cost or effort.

4.1 Blender scripting for images and masks generation

Blender[1] is a free and open-source[2] 3D computer graphics software toolset used for creating animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, and computer games. Blender's features include 3D modeling, UV unwrapping, texturing, raster graphics editing, rigging and skinning, fluid and smoke simulation, particle simulation, soft body simulation, sculpting, animating, match moving, rendering, motion graphics, video editing, and compositing.
Among these, for the current work, we are particularly interested in the possibility to import CAD models (`.stl` files) in the software, in the modelling capabilities in order to bring slight modification to the imported model for the upcoming operations, the possibility to apply textures for more realistic results, raster graphics and compositing for the production of the object masks, animation and, finally, the rendering ability itself to translate the

[1]Blender official site: `https://www.blender.org/`

[2]Blender is released under the GNU General Public License (GPL, or "free software").

image created in the software into an actual image.
Nonetheless, these are quite common features among this kind of software: the real reason why Blender was preferred to other solutions is its ability to be scripted using the Python™ language. In fact, being the Blender interface itself controlled via Python™, the software provides to the user an extensive API through which almost any of the operations that can be done via the point-and-click interface can also be done via a Python™ command. This may be useless when creating a single scene for a digital art project, but it is extremely useful when dealing with repetitive tasks as it gives the option to automate them. As it will be clear when illustrating the pipeline implemented in JINS, this is exactly where this approach resulted to be very convenient.

Process outline Before going deeper in the various steps involved in the generation of the dataset, it is useful for the reader to have a little overview of the process in order for him to be easier to follow along. Three main steps can be identified:

1. *Pre-processing phase.* Before being fed to the section of the software that actually takes care of rendering the images, the spacecraft model needs to be adjusted in order to be compliant with the requisites of the successive piece of software. This basically means appropriately subdividing the model into smaller parts corresponding to the features of interest and providing an *empty entity*[3] for the camera to point to.

2. *Images and masks generation.* The code that is responsible of the images generation is then run over the suitably adapted model. In short, it generates a user-defined number of light objects in the scene

[3]In Blender, an Empty is an object which is not rendered, i.e. an object that can be used for modelling purposes but that doesn't appear in the image when it is generated. In this case, a simple set of Cartesian axes has been used.

and activates them one at a time in a for-loop; then, for each of them, it generates a number of cameras and renders the image from each of these points of view, then moving to the next illumination conditions. It also takes care of generating the masks that specify which is the area of the image that is occupied by each of the objects. A conceptual scheme of the procedure is provided in Fig. 4.3.

3. *Images annotation.* Finally, a second script uses the generated images and masks to create, through the *COCO API*[4], an annotation file in COCO format. This is basically a `.json` file that encodes, together with some metadata for the dataset, an identification number for the images and each of the objects instances it infers from the masks, specifying the class of the object and the vertices of the polygon that encloses it. This is the file that the neural network uses to compare its prediction against the ground truth and then modify the weights (if in training mode) or evaluate performances (if in testing mode).

4.1.1 Scene creation and pre-processing

As already explained, *variety* in the dataset is a key feature in order to have a good performing algorithm, therefore a set of CAD models of satellites (both real and fictitious ones) was chosen. Due to the lack of time, there was not the possibility to build a larger models database and their number was then

[4]COCO (Common Objects in COntext) is one of the most common benchmarking datasets for new algorithms in the field of computer vision (it also sponsors an annual worldwide competition) and has, therefore, become one of the standards for the annotation of the images since many CNNs accept it as a ground truth description. The standard dataset and the API providing the tools for the annotation of custom dataset is provided on GitHub.

limited to six.[5] Nonetheless, this was enough to obtain good results and to highlight how increasing the number of examples leads to better results (see Section 5.2.1).

In this phase, the Blender program is used through its graphical interface as any other program meant to be used for digital art. Other than directly using it for the creation of the model or importing the work done in some other 3D computer graphics software, Blender gives the user the ability to load an external file in `.stl` format or any other of the most used output formats from Computer Aided Design programs. This is a relevant advantage since this kind of files are always produced during the design process of the satellite and could therefore be used by the manufacturer or provided to the end-of-life responsible company for the study of a tailored OOS or disposal mission. Once the CAD model is loaded and translated to a Blender one, the main interface of the program appears as in Fig. 4.2.

The model is then adapted to fulfil the requirements on which the following part of the pipeline relies. These are one time only edits.

The tools provided by Blender are used to subdivide the various parts of the object in case they are not distinct already: in other words, each occurrence of the objects which are among the classes we are looking to train the algorithm onto must be interpreted by the software as a distinct standalone object (as shown in the list in the upper right corner of Fig. 4.2).

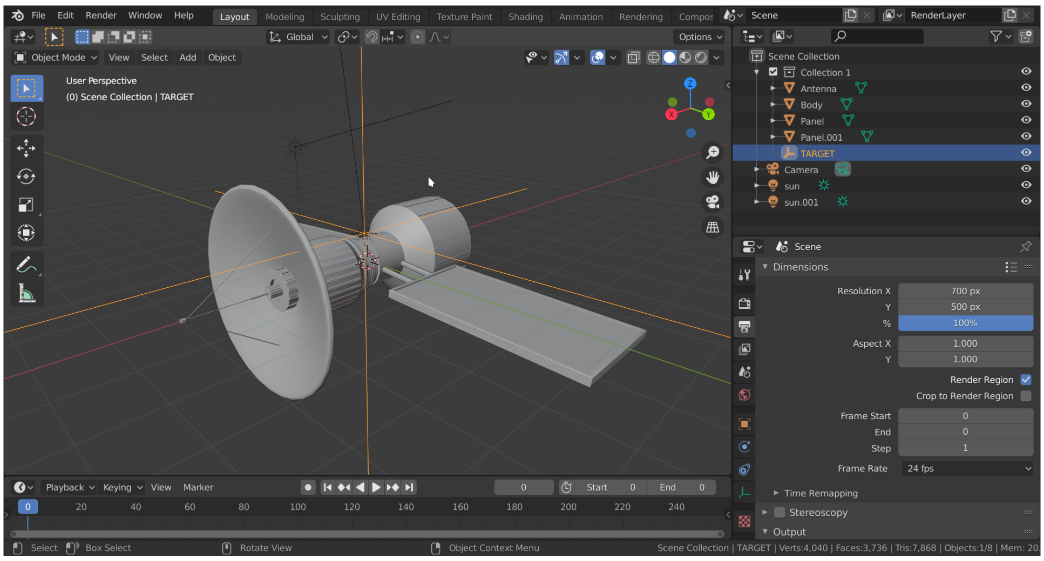
Then, a custom property has to be specified for each of the parts we're interested in: this has to be identified by the 'label' keyword and its value has to be set to a string corresponding to the class. This step can both be

[5]The resources used in the current work can be found at the following links:
`https://free3d.com/3d-model/small-satellite-308237.html`
`https://sketchfab.com/3d-models/satellite-f8bd72434281441db77e85ce830f89c1`
`https://nasa3d.arc.nasa.gov/models`
and `https://www.cgtrader.com/3d-models/space/spaceship/low-poly-satellite-parts` (last accessed: 03 June 2020)

done manually or using a custom written plug-in[6].

Lastly, as already mentioned, a Cartesian coordinate system is added to the scene. This is positioned in a random point close to the satellite but not necessarily coincident with any notable physical or geometrical property such as the centre of mass. This arbitrariness serves a very specific purpose: the cameras through which the scene will be rendered are generated by the main script and are made to point to the origin of the axes; this means that this point will always be exactly in the middle of the image. Being the satellite offset with respect to the coordinate system origin, it will appear slightly moved from the centre of the image and this is a desirable feature since it adds diversity to the dataset and forces the algorithm to learn to recognise the objects independently of their position in the image. An example can be seen in Fig. 4.2, where the object highlighted in orange is the target axes

[6]Such as the one available at `https://github.com/csun/syntrain/tree/master/blender_scripts`. Being the Blender interface managed in Python™, the program can be easily extended in functionalities and a large user community has grown around it.



Resolution X 700 px
Y 500 px
% 100%
Aspect X 1.000
Y 1.000
Render Region ✓
Crop to Render Region
Frame Start 0
End 0
Step 1
Frame Rate 24 fps

Start 0 End 0

Scene Collection | TARGET | Verts:4,040 | Faces:3,736 | Tris:7,868 | Objects:1/8 | Mem: 20

Figure 4.2: Interface of the Blender software.

system just described.

4.1.2 Image generation pipeline

Once the Blender scene is set up, the actual images can be encoded. This could be manually done (it would just require setting the illumination conditions for the scene, a camera object and starting the rendering of the image), but it would be extremely time consuming and inefficient. The dimension of a training dataset for an algorithm, in fact, depends on the complexity of the problem to be solved, the kind of objects that need to be identified and the number of classes, but it usually features at least some thousands of images. Needless to say, this would be an overwhelming work for a single person or even a group of people.
This is where the automation capabilities of Blender really helped and it is probably one of the most complex parts of JINS. A pipeline was generated to automate the rendering of the images and taking care that those are actually relevant, which means, for example, avoiding images that are mostly dark and in which the satellite is not clearly identifiable, since those would make the training process harder for the algorithm.
The overall conceptual scheme of the pipeline is reproduced in Fig. 4.3.

The file `parameters.py` lists all of the user-defined parameters so that they can be easily changed before running the main script and starting the renderings. Among them, the most important are:

- the image dimensions, i.e. its resolution, in pixels;
- the focal length of the camera and other parameters linked to that, like the minimum and maximum distance from the target and the maximum angle of the point of view from the lighting direction (Sun); those actually determine how big the satellite appears in the image;
- the list of the object categories to be detected;

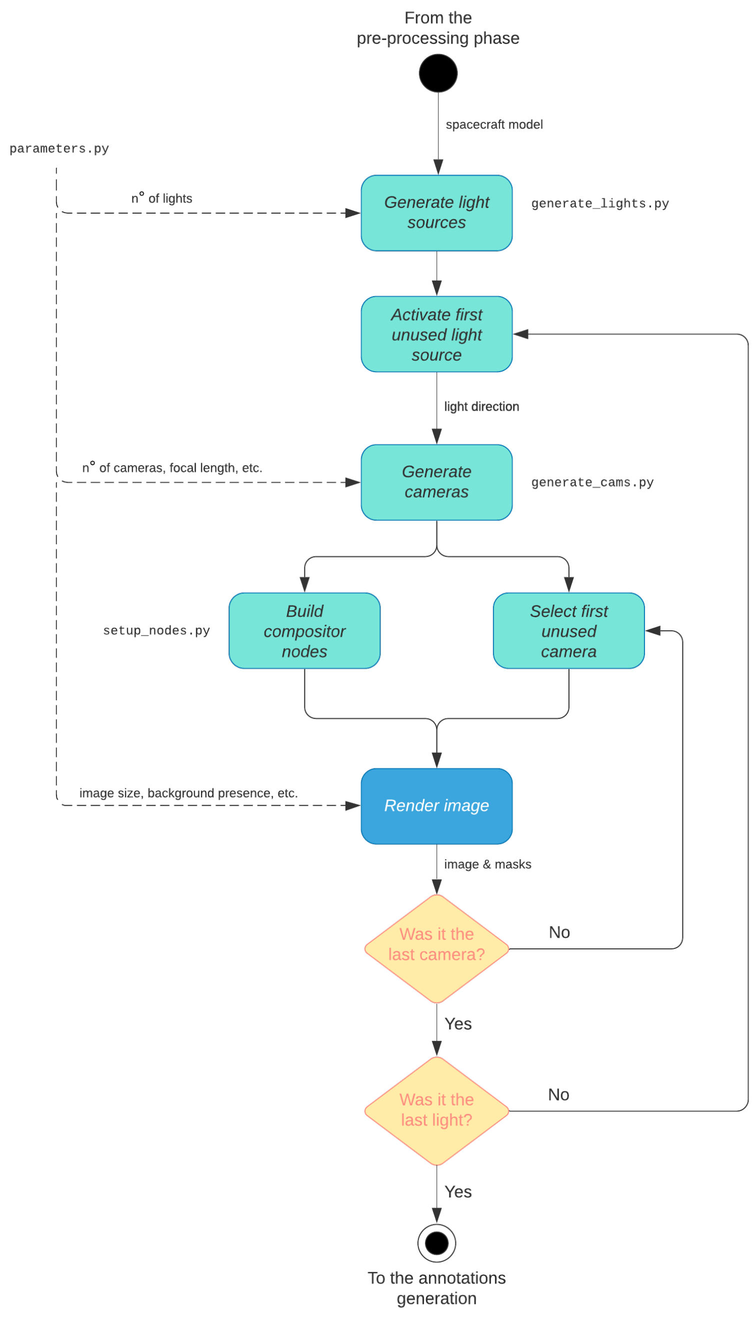


Figure 4.3: Scheme of the image generation pipeline in JINS as implemented in the `batch_render.py` *script. The text in monospace font close to the blocks indicates the function devoted to that particular operation.*

- the number of different lighting directions to be used and the number of different point of views (that is camera positions) for each of them. These determine the total number of images generated for the dataset:

$$number\ of\ images\ = number\ of\ lights\ \times\ number\ of\ cameras$$

- whether or not the Earth should be included in the background.

The file is structured so that these values can differ based on the fact that the generated images are used for the training or the testing dataset.
Other parameters that can easily be set from this file, rather than being hardcoded in the rest of the code, are the directory paths where the images should be saved based on which is the dataset they are part of and the prefixes to prepend to the image filename to distinguish them based on which of the satellite models is depicted in it.

These parameters are loaded by the `batch_render.py` file, which is the main script file and is run over the Blender instance with the desired model. Using auxiliary functions, it creates the different lighting sources in the scene, goes over them in a for-loop, activates them one at a time (so that the satellite is illuminated from a single direction) and for each of them generates the specified number of cameras in the scene. Then a nested for-loop cycles through the cameras, sets the compositor scheme to take care of the mask generation (more information below) and renders the image, moving then to the next light-camera couple and starting over. The process also takes care of creating an `info.json` file that gathers the information on the direction of the lighting source and the position and orientation of the camera for each of the images created, so that the single picture can be manually reproduced if needed.

Lighting conditions The function `generate_lights` creates the lights in the scene. Various types of light sources are available in Blender, like point sources or diffusing surfaces. The one of interest for this work is obviously the one that mimics the lighting from the Sun, which illuminates the scene with rays that are all parallel to a specified direction. The function generates a random unit 1×3 vector, spawns a Sun lighting object in the scene and then applies a quaternion rotation so that its direction matches the one of the vector just created. The light is then deactivated (just like switching off the source in the scene) by default. The process is repeated until a number of lighting sources equal to the one specified in the parameters file is present in the scene. Those are activated one at a time in the main loop, as already explained.

Camera generation In the nested for-loop, the direction of the currently active light is given as an input to the `generate_cameras` function. This unit vector is multiplied by a random number between the limits specified in the parameters file and rotated in a random direction of a random angle whose maximum value is specified in the same file too. Then, the new vector is added to the position vector of the target Cartesian axes, identifying a new point in the three dimensional volume of the Blender workspace. Using the built-in functions of the program, a new camera is added to the scene in that position and it is pointed towards the target Cartesian coordinate system.
This may seem a uselessly complicated procedure, but it actually grants some details whose importance was already mentioned: keeping the camera within a solid cone centered around the direction of the Sun and with a reasonable semi-aperture (a maximum angle of $35°$ was used) assures that the lighting conditions in the image are always reasonable (in fact, in a real application, there would be no reason to take pictures of the target satellite when it is not properly lit); limiting the range on the distance of the camera from the target, instead, grants that the satellite is clearly visible in the image, without being too small for its parts to be distinguishable or so large that

one of its components occupies the whole image.

Masks generation Blender also features an instrument called *compositor*. It can be used to apply various changes to the scene, modify parameters, filters and alike, manage and possibly completely alter the rendering process and therefore the output. In particular, this has been used in order to output, together with the rendered image, binary images specifying the area covered by a certain part of the spacecraft in the image, like the ones shown in Fig. 4.5. The Blender compositor is used creating a graph like the one shown in Fig. 4.4 and this can be built using Python™ too. This is what the `setup_nodes` function does. It firstly assigns a unique ID number to each of the parts that the spacecraft was previously subdivided into. The Cycles rendering engine[7] can be made aware of this unique information through an option called "object index pass" and it can therefore associate an index to each pixel of the image based on the one of the object that was first hit by the sampling rays in that pixel. A node called "ID mask" can then produce a mask image where all of the pixels that have a certain ID number are coloured in white while all of the others are black. The `setup_nodes` function creates one of these nodes for each of the IDs, i.e. one for each object, and an "output node" consequently saves both the coloured image and a series of binary images, each one showing only the visible part of a certain object, taking care of

[7]By default, Blender provides two different rendering engines, Eevee and Cycles. While the first is faster in the image elaboration, the second is slower but gives more accurate results since it uses path-tracing, that means it works by sending a number of light rays that act as photons from the camera out into the scene. If these rays hit an object, they will bounce based on the angle of impact, and continue bouncing until a light source is reached or until a maximum number of bounces defined by the user. Multiple rays are calculated and averaged out for each individual pixel. Furthermore, Cycles is the only rendering engine supporting *index pass*, which is a fundamental property in order to produce object masks. It is therefore the engine chosen for this work.

including the name of the object class in the filename (as it can be seen in Figs. 4.4 and 4.5), using the “label” custom property that was specified in the pre-processing phase.
The script is also able to generate compositor nodes so that body-mounted panels are identified both as solar panels and as part of the body.

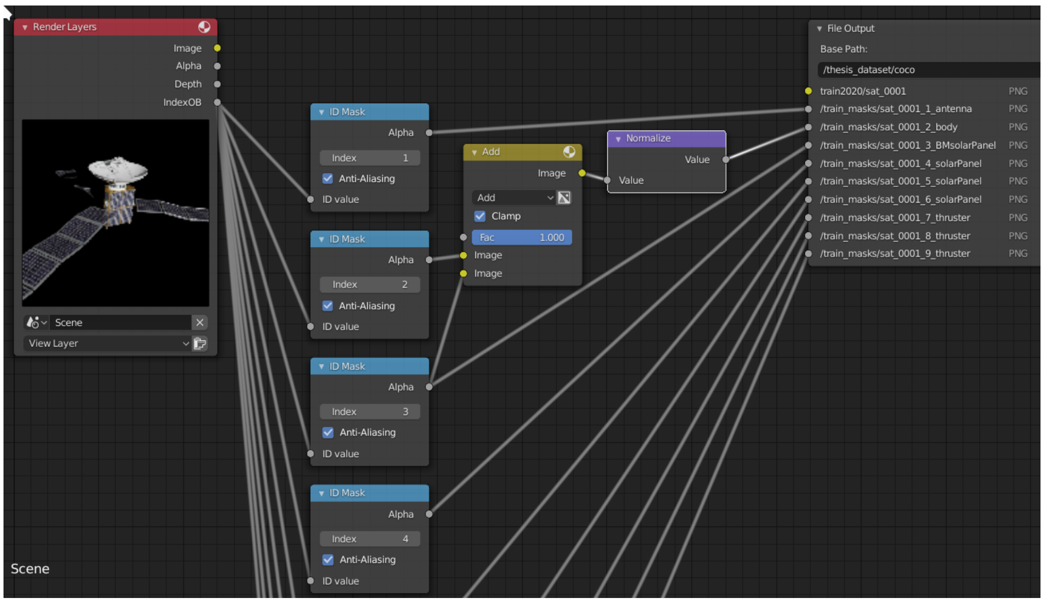


Figure 4.4: Example of a compositor graph generated by the `setup_nodes` *function.*

4.1.3 Earth background

As already mentioned, the JINS software is also capable of drawing the Earth in the background of the image. Unfortunately, due to limitations of the Earth model that was available and the rendering engine, the planet model could not be directly added to the scene to be automatically included in the rendering. Therefore a workaround was implemented: a series of pictures of the Earth in different light conditions and from different points of view were manually rendered; then, if it is specified to do so in the `parameters.py` file, the `setup_nodes` function is capable of randomly selecting one of

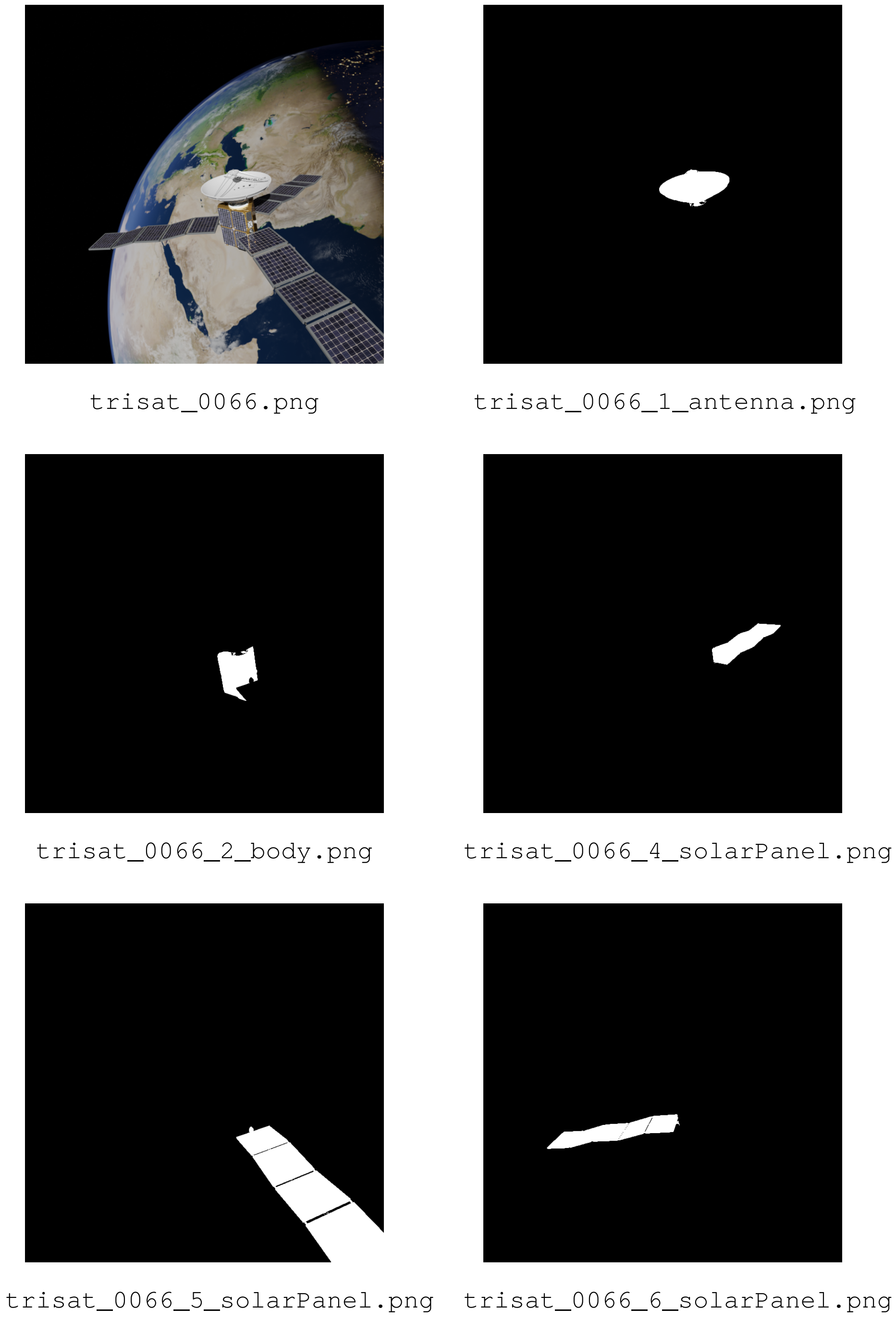

trisat_0066.png
trisat_0066_1_antenna.png
trisat_0066_2_body.png
trisat_0066_4_solarPanel.png
trisat_0066_5_solarPanel.png
trisat_0066_6_solarPanel.png

Figure 4.5: Example of a rendered image and the related masks identifying the different components. Notice how the class of the object is specified in the filename.

those pictures and using it as a background when rendering the model of the spacecraft.

This is obviously a non-optimal solution as it could lead to images that are not physically accurate since the lighting direction of the background could be slightly different from the one of the satellite, but it is not an issue for the aim of this study. The reason why it is worth having such kind of images is that it makes the object segmentation task more difficult for the CNN (due to boundary effects) and therefore it is better to train the algorithm on them. In fact, as it will be shown in Section 5.3, performance issues arise when the algorithm is trained exclusively on images without the Earth in the background and tested instead on pictures where the Earth is present.

4.2 Dataset labelling in COCO format

Once the model tuning and pre-processing is done and the images and masks are generated, the `satellites_to_COCO` function takes care of the encoding of the `.json` file stating the class and the polygon that encloses each object.

It first identifies, based on the name of the image, all of the masks that are related to it. Then, using the API from COCO, it is able to identify the white polygon in each mask and, from that, also the bounding box. The class of the object is instead inferred from the filename of the mask image since it was determined by the label assigned to the object in the pre-processing phase. Through the API other properties can be inferred too, such as the measure of the area of the mask, which is useful in order to neglect objects that are too small in the image.

The output file counts three main dictionaries:

- the first lists the object classes on which the algorithm has to be trained on and assigns an ID number to each of them;

- the second one lists all the images in the dataset and assigns an ID number to them;
- the third is the list of the annotations, that means the list of all the instances that were found. Together with the vertices of the polygon, it reports the ID of the image where that instance is located and the ID of the kind of object it is. Other properties such as the bounding box of the mask and its area are specified here too.

This is the file that the algorithm uses to compare its predictions against the ground truth and its the only one that must be provided to the neural network other than the training and testing pictures. The network does not look at the mask images directly, but rather reads the information from the annotation file.
Finally, the function `check_BB` allows the user to visually verify that everything went fine in the image and annotation generation, drawing the polygon and bounding box as read from the annotation file over the original image, like in Fig. 4.6.

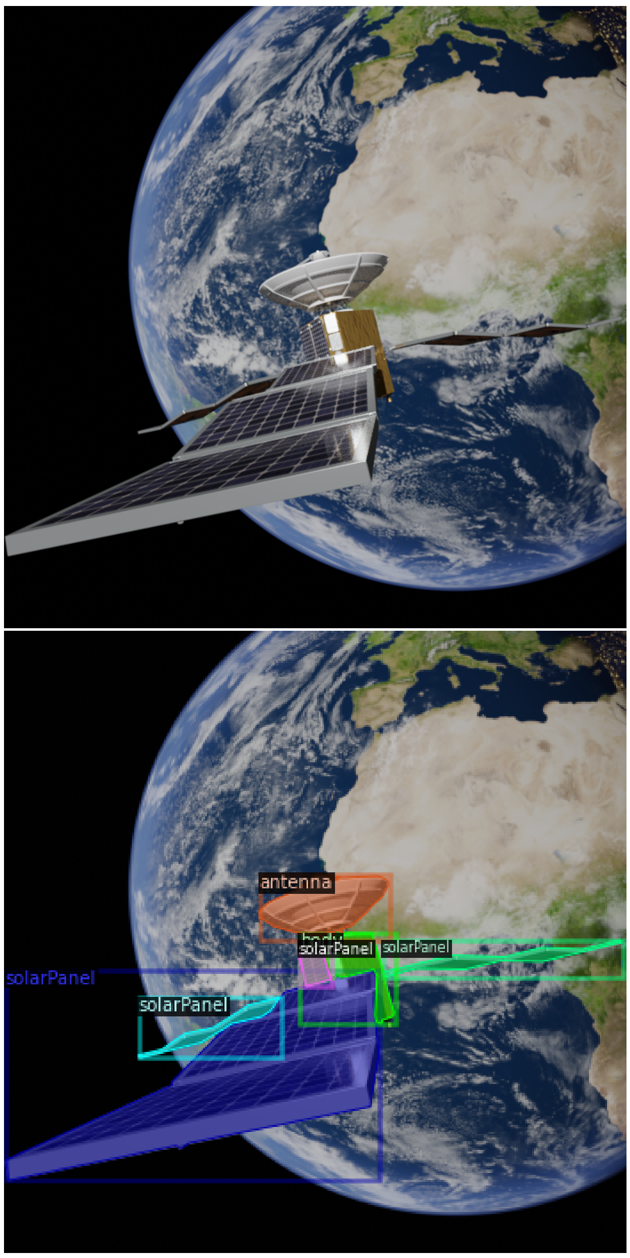


Figure 4.6: Final result of the dataset generation: the `check_BB` *function shows the image and the ground truth masks as they were identified by the* `satellites_to_COCO` *function.*

Chapter 5

Results

Let us start the chapter introducing some of the parameters that were used to perform all of the simulations, whose results are illustrated in the following sections, and some background information in order to understand them when needed.
As already explained in Chapter 2, the CNN algorithm on which the choice has fallen is the Mask R-CNN. Among the various implementations of the algorithm that are available online, the one from the original authors of the Mask R-CNN paper [24] has been preferred: `Detectron2` [35] from the Facebook AI Research (FAIR) group, of which the original authors are part of, is the last iteration of the code. Other than instance segmentation, it has been extended to perform other kinds of visual recognition tasks[1] and also provides the backbone implementation of the Faster R-CNN algorithm. By choosing Detectron2, it was therefore possible to test the effectiveness of the synthetic image generation approach both on the object detection task, through Faster R-CNN, and the instance segmentation task, through Mask R-CNN.

[1]For example it can perform pose estimation and panoptic segmentation depending on the base model the user specifies. More information on the project page `https://github.com/facebookresearch/detectron2`.

Among the different base models provided by Detectron2, the one that was chosen is the *R50-FPN-3x*, which uses a ResNet backbone architecture with 50 layers and a FPN for the regions proposal. It has been chosen because, as shown in the results on Model Zoo [36], although being one of the least resources intensive options both in terms of training time and memory usage, it still performs really well (although not has much as the equivalent with a deeper 101-layers ResNet backbone).

The number of classes the neural network was trained to recognise is four: antenna, main body, solar panels, engines. The 'thruster' class was initially taken into account too, but it always occurred that its instances were too little in the image to be even recognised by a person and had to be rejected when inferring the ground truth from the masks. Therefore, the 'thruster' class has been dropped, closer images would be needed in order to recognise its instances.

Two different couples of training and testing datasets were produced for each satellite model: one with the Earth in background and one without it. In either case the number of images is 800 for the training set (80 different lighting conditions and 10 different rendering points of view for each of them) and 200 for the testing set (20 lighting directions and 10 cameras each). These are fairly low numbers when compared to the ones usually found in the field and increasing them would certainly give better results, but rendering the images is a very time consuming task[2] and the available time for this study was not enough to build larger datasets. Nonetheless, being the number of classes quite small, the number of images that have been used is still enough to have good results, as shown hereinafter. `Detectron2` suggests image dimensions between 600 and 800 pixels to optimise training performances

[2] It can be done in a much faster way by rendering the image using the GPU instead of the CPU, which unfortunately was not possible in this work.

and timing. Square images with a 700 px size have been used to build the datasets used for this work. The models were trained using the Detectron2 API on Google Colaboratory, also known as Colab, a free cloud computing Google service specifically thought to ease the research in the ML field, since it gives the user the ability to train the models using powerful GPUs which, as explained in Chapter 2 greatly reduce the training time. For the current study, each of the model has been trained for 2500 iterations, which showed to be a good compromise between training time and performances. The training takes from 30 to 50 minutes, based on the capabilities of the GPU that Colab randomly assigns to the user at runtime.
The inference runs at around 10 FPS for each of the evaluated datasets.

Learning rate As already explained, at the end of each training iteration (or epoch as some algorithms prefer to measure the training progress), an ANN computes the *loss function*[3], which expresses how far the predictions are from the ground truth. Therefore, the training task is basically a minimisation problem in which the right set of weights has to be chosen in order to find the loss minimum.
The capability of the algorithm to solve the problem is greatly influenced by a number of user-defined parameters, called *hyper-parameters*, among which the *learning rate* is particularly important. It defines how the weights are modified from one iteration to the next following the relation

$$\vartheta'_n = \vartheta_n - \alpha \, \frac{\partial}{\partial \vartheta_n} J(\vartheta_n)$$

where ϑ'_n is the new weight, ϑ_n is the old one and α is the learning rate which multiplies the gradient of the loss function. As shown in Fig. 5.1, if the learning rate is to small, the gradient descent can be very slow, while, if

[3]Various different loss functions have established themselves as the standard in the field over the years, but it is also possible to define a custom one. Going deeper in the topic is out of the scope of the present work.

it is too large, gradient descent can overshoot the minimum of the function and the algorithm may not converge or even diverge.

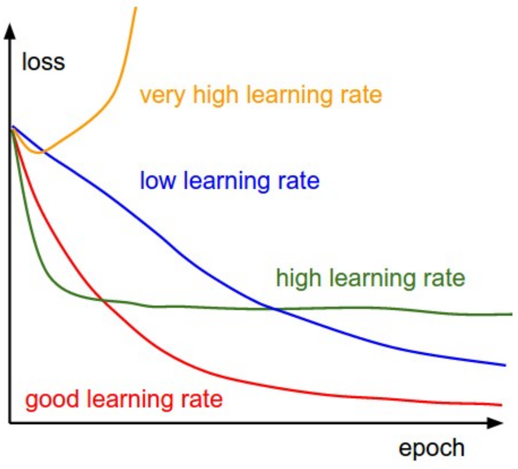


Figure 5.1: Effect of various learning rates on convergence. Image credit

Even if other possibilities exist, choosing a good learning rate is mostly a trial and error procedure since it is heavily dependent on the problem and its complexity. For the current work, a learning rate of 0.0025 showed to be a good compromise between training time and performance, in good synergy with the maximum number of iterations. Actually, this is just a starting value, since the algorithm is then capable to slightly adapt it to various needs as the training goes on.

As a final remark before the results are introduced, Fig. 5.2 shows the six models together with the names that will be used hereafter to refer to them. This is useful since some of the results find their reason to be in the geometry of the spacecraft itself.

5.1 Training on single model

The method used to identify the usefulness of the JINS approach was to subject it to increasingly difficult tasks, developing and tuning them based

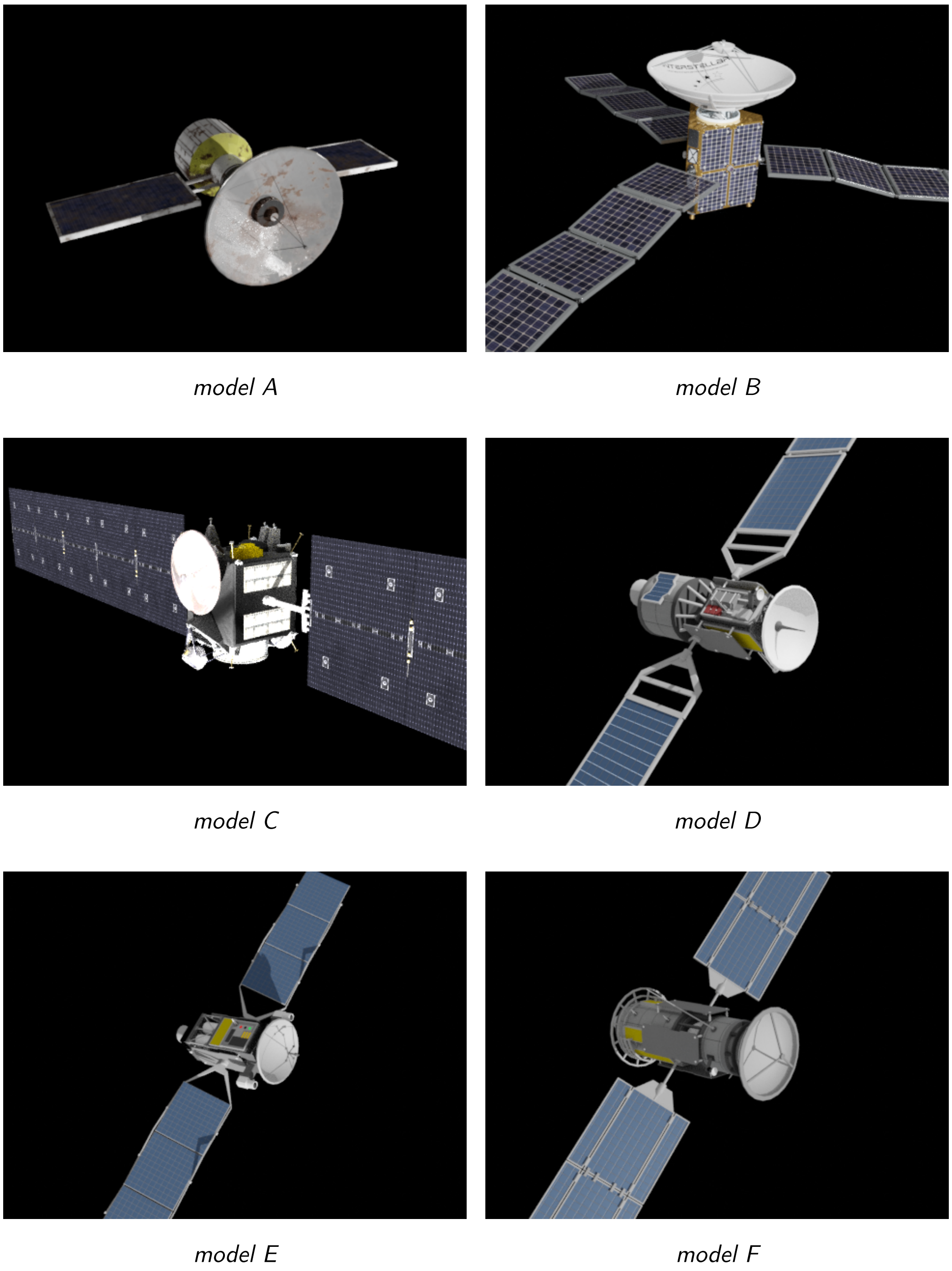


Figure 5.2: Representative images of the spacecraft models used in the present work. The names that will be used to refer to them are here reported.

on the results of the previous tests.
The first task was therefore testing the capabilities of the algorithm when trained and tested on a single model images without the Earth in background. As it can be seen from the results obtained on models A and B, reported in Tables 5.1 and 5.2, they are in line with the ones in the literature [18, 24], both considering the class specific average precision and the mean one. The same happens on the other models too, so it is redundant to report them.

Table 5.1: Results on a dataset including a single model (model A). The first line shows the metrics on the object detection task, while the second reports the one on the instance segmentation task. The metrics are the ones explained in Chapter 2.

	AP	AP-antenna	AP-body	AP-solarPanel	mAP
bbox	87.86	89.73	88.37	85.47	89.99
mask	89.28	88.53	89.10	90.22	86.39

Table 5.2: Results on a dataset including a single model (model B).

	AP	AP-antenna	AP-body	AP-solarPanel	mAP
bbox	89.35	89.35	88.95	89.73	89.34
mask	85.16	88.26	85.88	81.34	83.23

While it may seem a very trivial task to perform and therefore a useless result, it not only validates the idea of using image recognition with bodies like satellites, but also demonstrates how this can be useful in a real scenario. Think for example of an OOS mission towards a specific and known satellite: the technical documentation of the target spacecraft or pictures of the object shot before the launch could be used and the algorithm trained to recognise that RSO specifically, even on its most peculiar features if proper classes of

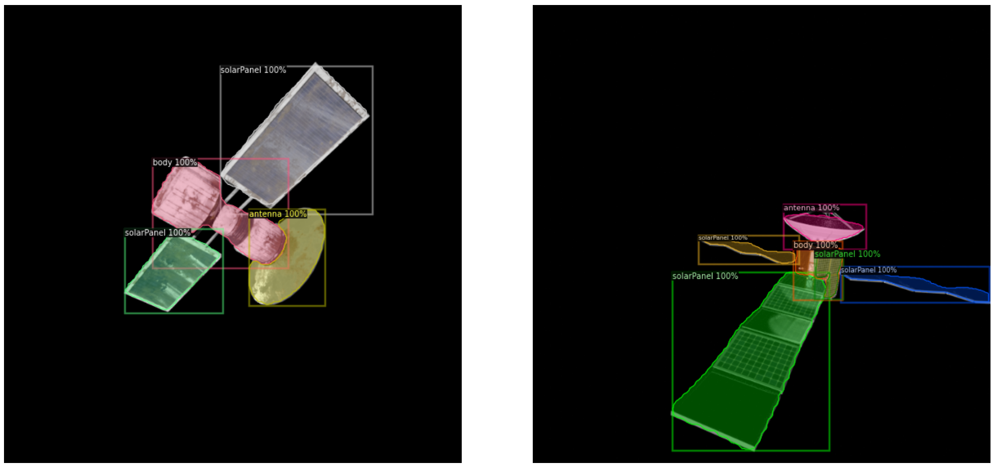


Figure 5.3: Results from the inference on the single satellite datasets.

objects were added besides the generic and more common ones that are being exploited in this study.

Figure 5.3 gives some examples of the results of the inference on images from the test sets (which, recall, are not included in the training set and, therefore, have never been seen before by the algorithm). It can been seen how the coloured masks overlap almost perfectly the actual objects, giving a visual proof of the high quality results previously shown in the tables.

5.2 Testing on 'unknown' satellites

The next step was trying to verify how the algorithm behaves if it is trained on a certain number of satellites and then tested on another satellite which is not part of the training set and therefore has never been seen by the neural network at the time of the inference. A training dataset was built with the images from models B and C and the test dataset only featured images of the satellite model A. The results are the one reported in Table 5.3:

The drop in performance is clearly visible when compared to the previous results. It is interesting to notice how the overall metrics of AP and mAP are

Table 5.3: Results obtained when training on models B and C and testing on model A.

	AP	AP-antenna	AP-body	AP-solarPanel	mAP
bbox	34.04	23.27	6.65	72.19	29.63
mask	34.56	22.34	2.48	78.79	27.27

anyway not very far from the analogous results reported in [24], even if they are greatly impaired by the very poor performance on the recognition of the body and the antenna. Looking at Fig. 5.2, this is clearly to be attributed to the geometry of the spacecraft model A, which has a cylindrical body that can be easily mistaken for the antenna from some points of view, in particular when seen from behind. Fig. 5.4 demonstrates this.

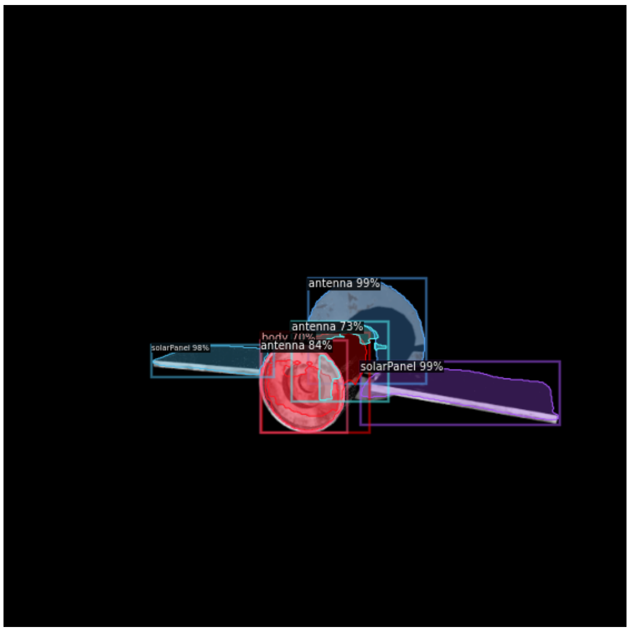


Figure 5.4: Example of the results when training on models B and C and testing on model A.

The reason why, while being both very low, the metrics on the body recognition is much more affected than the one of the antenna has to be found

in the similarities between the test model and the training ones. Since the dataset used is still very small, it does not offer the variety whose importance was highlighted in Chapter 4 and therefore the algorithm is not robust to intra-class variations. The antenna is better recognised simply because the antenna of model A is more similar to the ones of models B and C than the body of model A is to the ones of the other two spaceraft.
The same reason justifies the great results on solar panels recognition, which are closer to the ones obtained when training on the same satellite. The solar panels have a pretty typical configuration in both colours and shapes; therefore the one of model A is quite similar to the one of the other two spacecraft and it cannot be easily mistaken for anything else in the picture.

5.2.1 Increasing the number of models

Once the cause at the origin of the problem with the results shown in the previous section is identified, the most reasonable way to try to improve the performance of the algorithm is extending the training dataset to make it more diverse. Therefore, the images from models D, E and F were added to the training dataset already containing model B and C, while still testing only on the spacecraft model A. The results are shown in Table 5.4. The results in the first column are the ones already presented in Table 5.3 and, compared with the ones of the extended training dataset, the difference is remarkable: while the results on the solar panels remain almost the same, a big improvement on the antenna and body recognition leads to an improvement on the general metrics AP and mAP ranging between 7% and 11%. This is a really appreciable result, which corroborates the hypothesis on the reason why poor results are obtained in Section 5.2 on the antenna and body recognition.
Moreover, it suggests that further increasing the size of the dataset and the variety of models included would lead to even better results. Unfortunately,

this is something that could not be verified in this work due to the limited resources.

Table 5.4: Results of the inference on a single model when training on all the other five. Dataset 1 reports the results in Table 5.3 for comparison;
Dataset 2 is tested on model A and trained on the other five models;
Dataset 3 is tested on model D and trained on the other five models.

	Dataset 1	Dataset 2	Dataset 3
	Object Detection		
AP	34.04	42.95	61.43
AP-antenna	23.27	38.91	72.30
AP-body	6.65	15.92	62.61
AP-engine	-[a]	-	42.06
AP-solarPanel	72.19	74.02	68.72
mAP	29.63	36.86	63.84
	Instance segmentation		
AP	34.56	45.41	55.90
AP-antenna	22.34	45.83	61.47
AP-body	2.48	9.01	58.16
AP-engine	-	-	43.26
AP-solarPanel	78.79	81.38	60.74
mAP	27.27	37.27	55.33

[a]The missing numbers are just because no engine is present in model A

However, to further validate the idea, another dataset was created, using as a testing set the model D and training the algorithms on the other five satellites. The results are shown in the third column of Table 5.4. They substantially improve with respect to the inference on model A and the reason it to be found in the geometry of the models: looking at Fig. 5.2, in fact, it is clearly visible how model D is more similar to the other satellites, in particular models E and F, than model A is; thus, being better represented in the training dataset, its parts are easier to be recognised by the algorithm.

As a side note, it is interesting to notice (and the same can be done also in the following sections in every result obtained when testing on model D) how the results on solar panel recognition actually worsen. This is due to the fact that model D also features some engines, whose geometry is very simple and can be sometimes mistaken for a part of the solar panels, thus decreasing the class performance. This is anyway a case-specific facet completely negligible with respect to the giant leap in performance on the antenna and body detection and segmentation.

5.3 Influence of the Earth in background

The results explained up to now were all obtained on datasets in which the images of the spacecrafts did not show the Earth in the background; it is therefore interesting to verify how the performance changes when the planet is present in the image.
Table 5.5 shows the results when the algorithm is run on a dataset with the same single satellite in both the training and test dataset when the images feature the planet in the background. The values for the same datasets but without the Earth (already shown in Section 5.1) are reported again for an easier comparison[4].

As it can be clearly seen looking at the first two columns, there is no

[4]Notice that these are two completely different dataset having different images rendered from scratch. The images of the second dataset are not just the ones of the first superimposed to pictures of the earth. The same consideration is valid for similar datasets that will be introduced later.

Table 5.5: Results when training and testing on a single model with and without the earth in background.

	model A no Earth	model A w/ Earth	model B no Earth	model B w/ Earth
		Object Detection		
AP	87.86	89.32	89.35	72.58
AP-antenna	89.72	90.29	89.35	88.29
AP-body	88.37	90.48	88.96	87.81
AP-solarPanel	85.47	87.20	89.73	87.37
mAP	89.99	89.15	89.34	87.38
		Instance segmentation		
AP	89.28	89.34	85.16	70.58
AP-antenna	88.53	89.68	88.26	86.56
AP-body	89.10	88.36	85.88	86.50
AP-solarPanel	90.22	89.97	81.34	80.44
mAP	86.39	87.53	83.23	82.58

appreciable difference between the results obtained on model A when the Earth appears in the picture and when it does not.
The same could be said about the results on model B, but for the AP value which instead shows a noticeable decrease. However, this is not due to the presence of the Earth in the background, but rather to the fact that some instances of objects of the type "thruster" are present in the dataset. They are however in a very small number and not significant enough for the algorithm to be able to properly train on them. This poses a false dilemma since it is somehow an ill-posed problem and the per-class thruster AP scores a bad result, which is reflected in the overall AP value. It is worth repeating, however, that this is not due to the presence of the earth in the images, as the

results on the other classes of objects and mAP demonstrate. It is therefore safe to conclude that the results on a single satellite are not influenced by the presence of the Earth in background.

It is interesting to verify if the same conclusion applies when dealing with datasets in which the testing is performed on a satellite model different from the other five used for the training. Therefore, a version of the datasets used in Table 5.4 has been made with the Earth in background. The results are shown in Table 5.6.

As it can be seen, differently from what happened with the single-model datasets, the difference on the performance of the algorithm based on the presence of the planet in the background is small but appreciable when testing the network on a spacecraft model which was not included. This is expected because the presence of the Earth makes the border of the objects more difficult to be recognised with respect to when they are against a black background, having some masks cover pieces of other parts of the spacecraft too (see Fig. 5.5). The variation is in any case modest: in particular, when testing on model D, the highest difference is registered on the engine recognition, which suffers, even if to a smaller extent, of the same problem of the thrusters since only half of the satellites models have clearly visible engines and they are obviously not always in view in the image; nothing similar happens on Dataset 1 (the one testing on model A), where the results are much closer to the case without the Earth, thus validating the reason here reported for the mismatch in Dataset 2.

Finally, another test was made to assess the algorithm behaviour when it is trained on a dataset without the Earth in background and tested on a set with the planet, or viceversa. The results are shown in Table 5.7. As expected, when training including the Earth and testing without it, the results are almost in line with the ones obtained when training and testing

Table 5.6: Results of the inference on a single model when training on all the other five, with and without the Earth in background.
Dataset 1 is tested on model A and trained on the other five models;
Dataset 2 is tested on model D and trained on the other five models.

	Dataset 1 no BG	Dataset 1 w/ BG	Dataset 2 no BG	Dataset 2 w/ BG
	Object Detection			
AP	42.95	41.68	61.43	57.86
AP-antenna	38.91	35.64	72.30	67.24
AP-body	15.92	14.29	62.61	65.40
AP-engine	-	-	42.06	32.55
AP-solarPanel	74.02	75.11	68.72	66.23
mAP	36.86	36.23	63.84	58.32
	Instance segmentation			
AP	45.41	44.03	55.90	48.86
AP-antenna	45.83	43.63	61.47	55.95
AP-body	9.01	7.21	58.16	51.11
AP-engine	-	-	43.26	30.86
AP-solarPanel	81.38	81.25	60.74	57.53
mAP	37.27	36.45	55.33	45.53

Table 5.7: Results obtained when including the Earth in the training dataset and not in the testing set, or vice versa.
Dataset 1 is tested on model A and trained on the other five models;
Dataset 2 is tested on model D and trained on the other five models.

	Train with BG Test without BG		Train without BG Test with BG	
	Dataset 1	Dataset 2	Dataset 1	Dataset 2
Object Detection				
AP	34.96	53.50	32.08	34.00
AP-antenna	17.35	66.76	34.54	47.42
AP-body	13.37	51.44	24.07	37.21
AP-engine	-	31.47	-	13.33
AP-solarPanel	74.16	64.33	37.64	38.04
mAP	31.91	55.18	30.14	41.15
Instance segmentation				
AP	36.89	48.82	35.51	27.79
AP-antenna	24.79	56.23	44.67	39.65
AP-body	8.96	47.11	16.85	28.75
AP-engine	-	35.89	-	3.55
AP-solarPanel	76.93	56.06	45.00	29.22
mAP	31.24	45.93	28.28	30.79

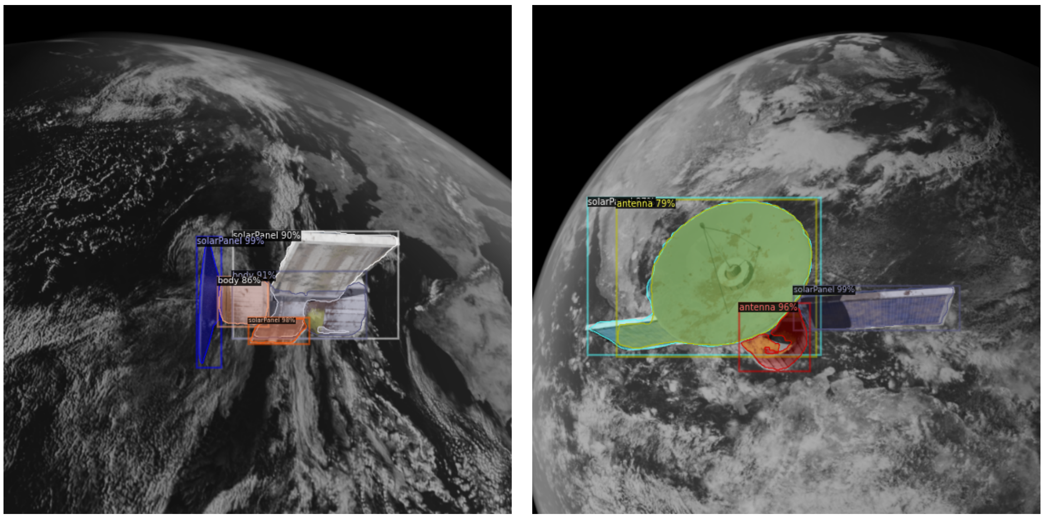


Figure 5.5: Images showing how the presence of the Earth in the background makes instance segmentation less precise.

including the planet. The same is not true when training without the Earth background and testing the network on images that include the planet. This is clearly to be expected and it is simply due to the fact that the algorithm has been tested on a dataset which is more complex than the one it has been trained on.

Finally, Table 5.8 reports the results obtained when training and testing on all the six satellites models and with the Earth in the background, con-

Table 5.8: Results obtained when training and testing on all the six models and with the Earth in the background.

	AP	AP-antenna	AP-body	AP-engine	AP-solarPanel	mAP
bbox	59.16	73.83	80.42	60.18	81.53	74.36
mask	56.58	71.12	70.08	58.35	77.37	70.93

firming once again when compared to the results in Section 5.1 that, even if a small decrease in the performance is noticeable, the addition of the Earth in the background does not represent a major challenge for the algorithm, not in object detection nor in instance segmentation. Its effect is secondary to the one due to the low number of engine instances present in the datasets.

5.4 Real life application

As it was highlighted from the very beginning, this work is targeted towards a practical standpoint. Therefore, once it is verified that it performs well even if it is not immune to the typical weak spots of a ANN, it is interesting to check how it would perform in a number of real life situations.

5.4.1 Video and conditions dependence

A short video was produced recreating in Blender a trajectory for the camera around the target that has the shape of a football orbit like the ones introduced in Chapter 3. The video was then fed to the trained algorithm to simulate a real-time recognition. Some of the frames from the video are shown in Fig. 5.6 and show how, even if recognition is sometime uncertain, there's always a leg on the orbit for each object during which, being the lighting conditions favourable, the object is recognised. While this may not be enough for a proximity operation, it would be a sufficient requirement for any preliminary inspection mission. This also give a suggestion for a follow-up research which is illustrated in Chapter 6.

Furthermore, quantitative results are reported in Table 5.9. It can be seen how these values are very high and this means that, although not being perfect, the prediction accuracy in the frames in which the satellite is not well illuminated is appreciable, even if the algorithm was trained only on well lit images. This is due to the fact that, thanks to the feature map extraction, the CNN is able to recognise patterns which are less prominent to the human

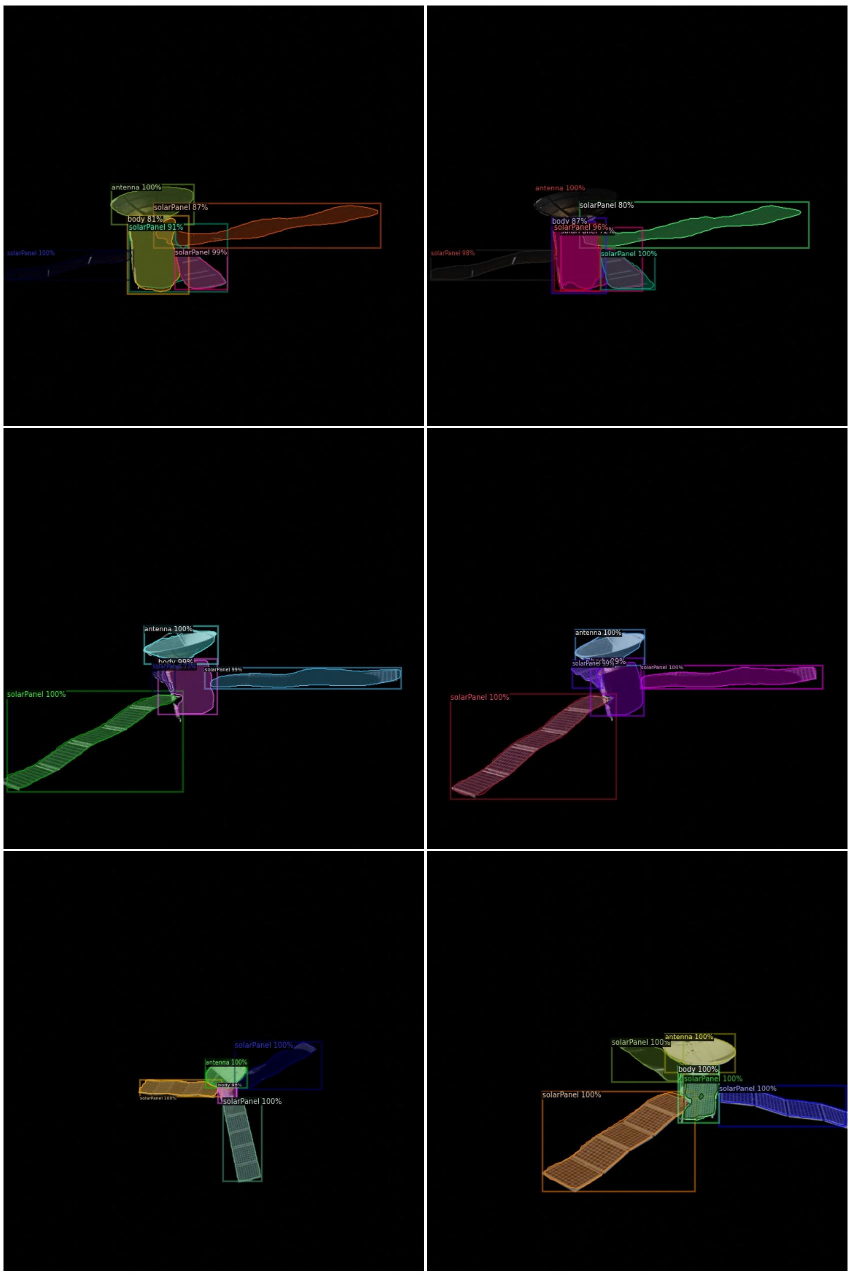


Figure 5.6: Frames from a video simulating a football orbit around the target spacecraft. While recognition is not always possible, like for example when the spacecraft is in shadow, there's always at least a leg of the orbit when it performs well. In the sequence it is clearly visible how inference results gradually improve as the chaser spacecraft approaches a better point of view.

mind.

Table 5.9: Results obtained when testing the algorithm on a video simulating a football orbit around the target spacecraft.

	AP	AP-antenna	AP-body	AP-solarPanel	mAP
bbox	83.25	93.44	75.00	81.30	84.91
mask	81.01	91.23	76.17	75.63	82.19

5.4.2 Gray scale images and pre-processing

In practice, most of the spacecrafts performing OOS would probably be equipped with cameras, maybe IR cameras for example, only able to acquire gray scale images. Trying to understand the feasibility of the JINS approach when dealing with such kind of images is therefore relevant to the current study.
An auxiliary script named `rgb2GS.py` was created: it uses the `pillow` Python™ package to create grayscale copies of the datasets given as input. Examples taken from the modified datasets are provided in Fig. 5.7.
When compared to the results in Table 5.6, the results in Table 5.10 demonstrate how the algorithm is robust to the change of color encoding. This validates further the possibility of using the approach in a real scenario.
Of even more interest are the results shown in Table 5.11. They are obtained training the neural network on colored images of model A, B, C, E and F and testing on grayscale images of model D. As it can be seen, there is no significant difference with the results reported in Table 5.10 for the full grayscale version of the same dataset. This is due to the intrinsic working principles of the CNNs, since the feature maps highlight patterns that are not the same which a human typically cares about such as colors.

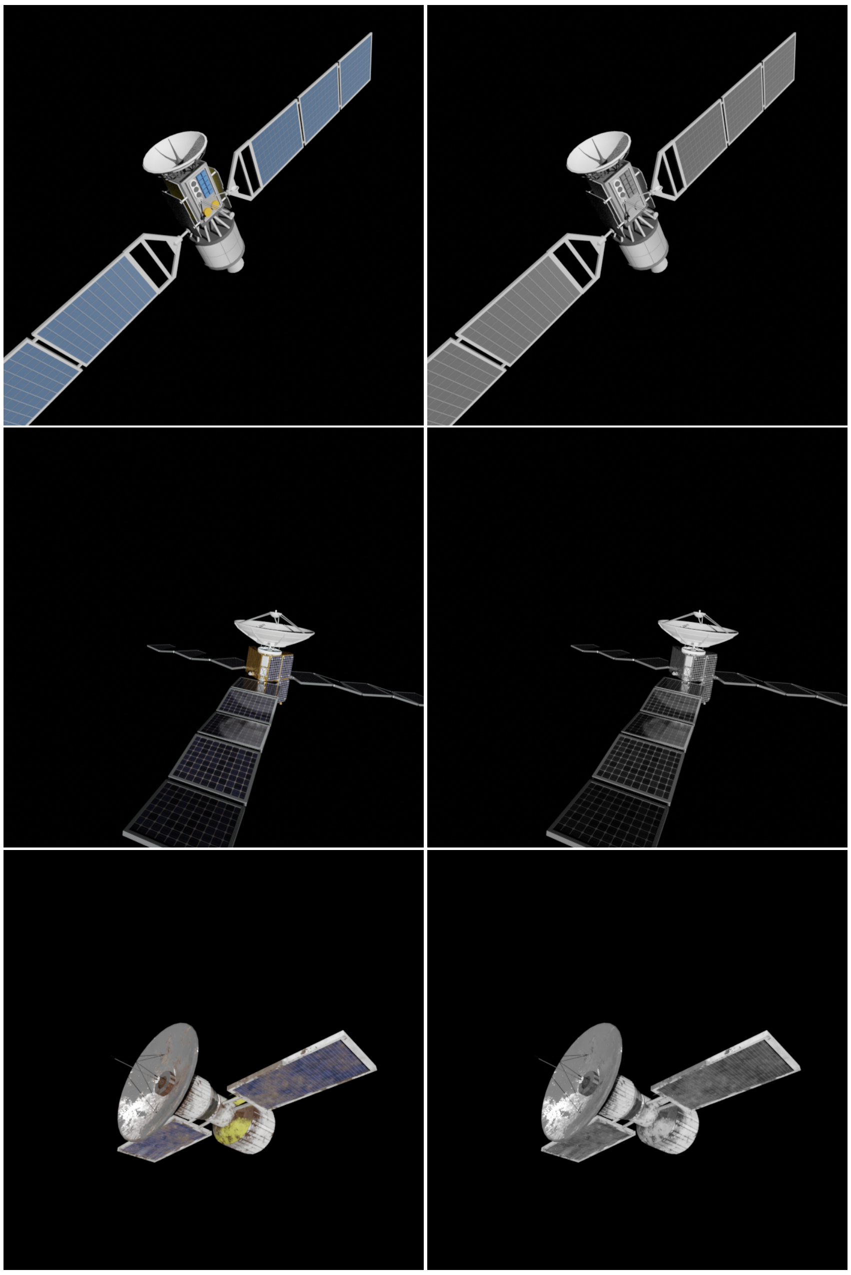


Figure 5.7: Examples of the output from the `rgb2GS.py` *script, which creates grayscale versions (on the right) of the images provided as input (on the left).*

Table 5.10: Results obtained when training and testing the algorithm on grayscale images.
Dataset 1 is tested on model A and trained on the other five models;
Dataset 2 is tested on model D and trained on the other five models.

	Dataset 1	Dataset 2
Object Detection		
AP	43.60	57.19
AP-antenna	23.83	70.23
AP-body	30.45	57.22
AP-engine	-	34.97
AP-solarPanel	76.52	66.35
mAP	39.23	58.94
Instance segmentation		
AP	48.36	53.12
AP-antenna	34.78	65.04
AP-body	27.56	53.53
AP-engine	-	37.36
AP-solarPanel	82.74	56.78
mAP	39.11	53.41

Table 5.11: Results obtained when testing the algorithm on grayscale images of model D and training on coloured images of the other five models.

	AP	AP-antenna	AP-body	AP-engine	AP-solarPanel	mAP
bbox	59.50	70.60	56.92	41.53	68.97	59.89
mask	54.30	65.19	51.18	42.78	58.05	54.16

5.4.3 Real images: Intelsat 901

As a last validation step, the algorithm trained on synthetically generated images datasets has been tested on images of a real world mission to verify if it is actually useful. A test dataset was created taking real images from the Intelsat 901 mission.
IS-901 is a telecommunication satellite launched in 2001 which was approaching its end of life due to the fuel consumption. In February 2020 another spacecraft, the Mission Extension Vehicle-1 (MEV-1) rendezvoused and docked with the Intelsat-901 to extend its operational lifespan by performing station-keeping for the aging satellite. This was the first mission of this kind and the chaser spacecraft took some photos while approaching the target. These are shown in Fig. 5.8 and were used to test the efficiency of the JINS approach and how the choice of the training dataset may impact the performance (since the images are very low quality, they actually present some shades and rings in the regions where they seem to be black). [5]

Since it was shown that good results are obtained on grayscale images even if the network is trained on coloured images, the training dataset involving all of the six satellites and the Earth background was first used evaluated. The results in Fig. 5.9 demonstrate an optimal performance since, in spite of the low quality of the images, the network is capable of recognising almost perfectly every part of the spacecraft.

The same has been done training the algorithm on a grayscale dataset without the Earth background. The results, shown in the left column of Fig. 5.10, are very poor and this is both because of the artefacts of the original image (halos, rings, shades, etc.) - which the grayscale training is not robust to since those are spurious variations in the gray scale themselves and cause border effects - and because the Earth closely resembles the antennas and

[5]The images were manually annotated using the *COCO Annotator* software [37].

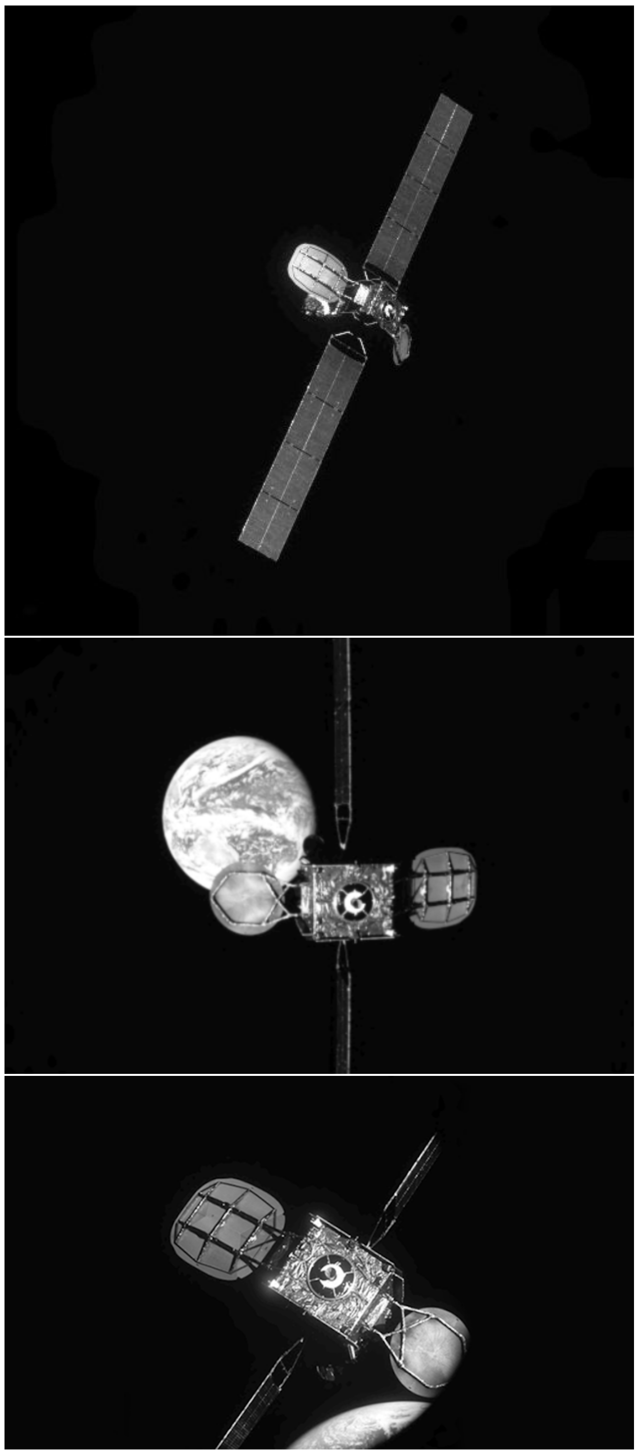


Figure 5.8: Images from the Intelsat-901 rendezvous.

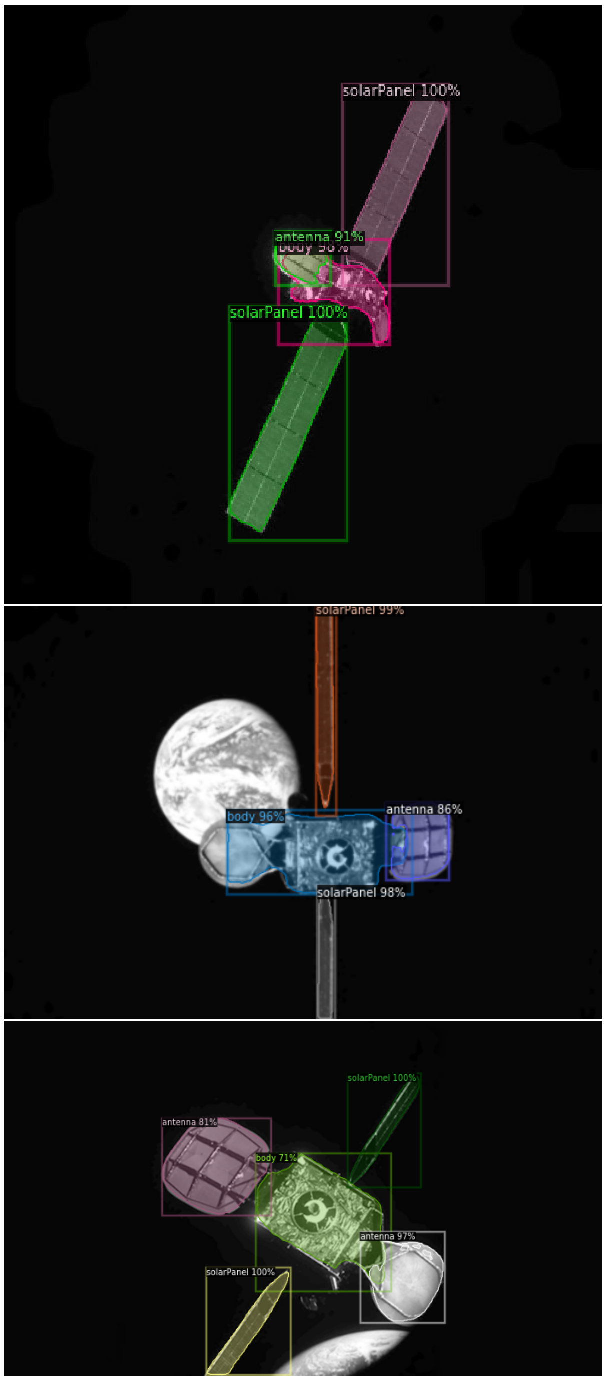


Figure 5.9: Results of the inference on the images from the Intelsat-901 rendezvous when the algorithm is trained on coloured images of all the satellites model with the Earth in background.

can easily be mistaken for one of them if the algorithm is not trained to recognise it.
Being highly probable that the problem was indeed the one just mentioned, the idea has been trying to improve the quality of the original pictures before the inference. The `threshold` function of the `cv2` Python™ package was used to get rid of the strange halos and shades. It can be used with an option that, setting a threshold for the grayscale value, leaves unaltered the pixels with an higher value than the threshold and sets to 0 (completely black) the value of the pixels below the threshold. In this way, the light gray shades that were present in the original images are substituted by completely black patches and this greatly increases the performance of the algorithm since it limits the boundary effects at the borders of the objects. However, the Earth is still mistaken for something else and not accurately rejected like it happens when it is included in the training dataset.

So, it has been shown that the synthetic dataset generation approach exploited in JINS is perfectly capable of being applied in real world scenarios, provided that a relevant dataset is used or that raw images are pre-processed to overcome potential difficulties.

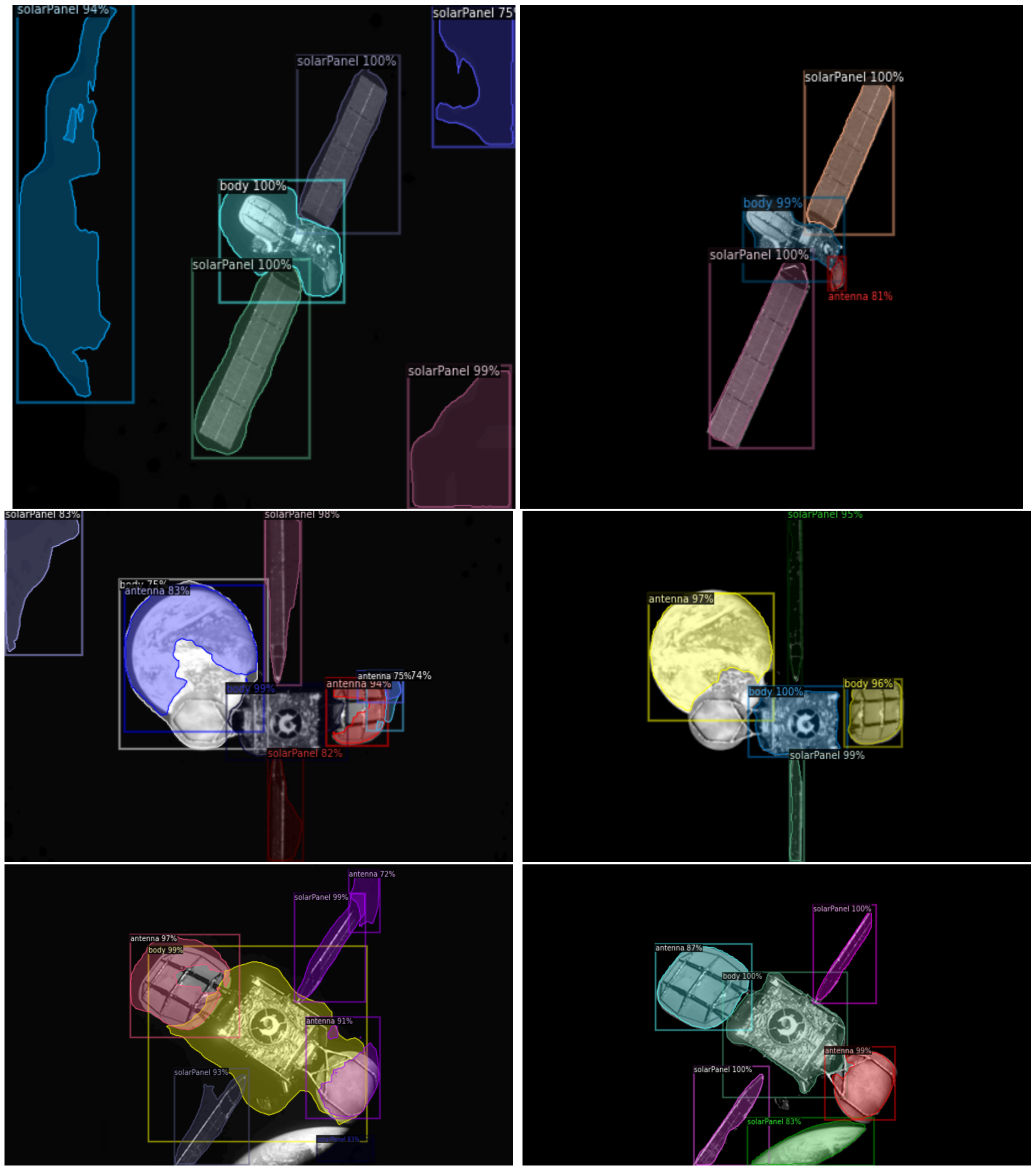


Figure 5.10: Results of the inference on the images from the Intelsat-901 rendezvous when the algorithm is trained on grayscale images of all the satellites model without the Earth in background. On the left, the results when the inference is run on the original images, on the right the ones after the threshold operation.

Chapter 6

Conclusions and further studies

How Convolutional Neural Networks and instance segmentation can be used to perform satellite features recognition in the framework of an inspection mission around a non-cooperative RSO has been demonstrated in the present work. In particular, after introducing some theoretical concepts about Artificial Neural Network and naturally repeating relative orbits, the possibility of generating artificial datasets to be used for the training of such algorithms has been evaluated together with its performance in real life applications. To this aim, a new software, called Jins Is Not a Simulator (JINS), has been developed, which takes care of generating an arbitrary amount of suitable training and testing images with minimum intervention by the user.

To author's knowledge, nothing similar has been done to-date in the field and the current study is proposed as a new research branch aimed at complementing the works on autonomous satellite navigation: for example, the recognition of typical satellite features, such as solar panels, could be exploited to improve the performance of autonomous navigation algorithms, providing a new instrument for the determination of the attitude of the target object or data useful for the planning of the ensuing manoeuvres.

Results have shown how performances are on par with those reported in the literature and, moreover, instance segmentation algorithms trained on

such synthetic datasets have also shown to be effective in recognising objects in images from the real mission Intelsat 901.
The convenience of the proposed approach lies in the ability to automate the dataset generation and the possibility to tailor the learning process on a specific satellite in case the mission is specifically targeted on it. The only limitations of the method have shown to be those intrinsic to the machine learning algorithms used (which are anyway being constantly improved in their baseline performances) and those linked to the necessity of choosing a proper training dataset or adequately pre-processing the real images if necessary in order to have good results on a real mission.

Finally, being such a new take on the subject, many possible advancements come to mind. Among them:

- improving the results by performing a better refinement of the training process (change learning rate, optimise hyperparameters etc);
- using data augmentation and including real images in the train dataset in order to improve performances;
- turning JINS in a full fledged simulator: using a better model for the Earth, implementing the ability to recreate the real dynamics of the spacecraft motion based on initial conditions and ephemerides, giving the option to create an image based on an input date, etc;
- using Recurrent Neural Networks (RNNs) instead of Convolutional Neural Networks (CNNs) when applying the method in real missions. During the mission, the images taken in sequence will reveal the evolution of the target position due to the relative motion of the chasing spacecraft and the tumbling attitude of the object. Therefore, an improvement in performance is expected by using a RNN since this kind of algorithms is capable of analysing the temporal sequence of images

rather than the single frame one by one, thus exploiting information and results from multiple previous images to refine the inference on the latest one;

- verifying if “bones” in Blender can be used to create ground truth that could be used for pose estimation;
- implementing the inferring algorithm on a Raspberry Pi board in order to demonstrate its low computational footprint.

Bibliography

[1] H. Klinkrad, P. Beltrami, S. Hauptmann, C. Martin, H. Sdunnus, H. Stokes, R. Walker, and J. Wilkinson. “The ESA Space Debris Mitigation Handbook 2002”. In: *Advances in Space Research* 34.5 (2004), pp. 1251–1259. ISSN: 0273-1177. URL: http://www.sciencedirect.com/science/article/pii/S0273117704001061.

[2] B. Weeden, P. Cefola, and J. Sankaran. “Global space situational awareness sensors”. In: *AMOS Conference*. 2010.

[3] K. Wormnes, R. Le Letty, L. Summerer, R. Schonenborg, O. Duboismatra, E. Luraschi, A. Cropp, H. Krag, and J. Delaval. “ESA Technologies for Space Debris Remediation”. In: *6th European Conference on Space Debris*. Ed. by L. Ouwehand. ESA, 2013.

[4] D. C. Woffinden. “On-orbit satellite inspection: navigation and Δv analysis”. MA thesis. Massachusetts Institute of Technology, Dept. of Aeronautics and Astronautics, 2004. URL: http://hdl.handle.net/1721.1/28862.

[5] F. Capolupo and P. Labourdette. “Receding-Horizon Trajectory Planning Algorithm for Passively Safe On-Orbit Inspection Missions”. In: *Journal of Guidance, Control, and Dynamics* 42.5 (2019), pp. 1023–1032. URL: https://doi.org/10.2514/1.G003736.

[6] S. Sharma. “Pose estimation of uncooperative spacecraft using monocular vision and deep learning”. MA thesis. Stanford University, 2019. URL: http://purl.stanford.edu/yx323yk4488.

[7] J. A. Starek, B. Açıkmeşe, I. A. Nesnas, and M. Pavone. “Spacecraft Autonomy Challenges for Next-Generation Space Missions”. In: *Advances in Control System Technology for Aerospace Applications*. Ed. by E. Feron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 1–48. ISBN: 978-3-662-47694-9. URL: https://doi.org/10.1007/978-3-662-47694-9_1.

[8] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Computing Research Repository (CoRR)* abs/1311.2524 (2014). URL: http://arxiv.org/abs/1311.2524.

[9] B. Xu, S. Wang, and L. Zhao. “Solar Panel Recognition of Non-cooperative Spacecraft Based on Deep Learnin”. In: *2019 3rd International Conference on Robotics and Automation Sciences (ICRAS)*. 2019, pp. 206–210.

[10] C. A. Mack. “Fifty Years of Moore’s Law”. In: *Institute of Electrical and Electronics Engineers (IEEE) Transactions on Semiconductor Manufacturing* 24.2 (2011), pp. 202–207.

[11] E. Alpaydin. *Introduction to Machine Learning*. 2nd. The MIT Press, 2010. ISBN: 026201243X.

[12] L. Perez and J. Wang. “The Effectiveness of Data Augmentation in Image Classification using Deep Learning”. In: *Computing Research Repository (CoRR)* abs/1712.04621 (2017). URL: http://arxiv.org/abs/1712.04621.

[13] D. Powers. “Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation”. In: *Mach. Learn. Technol.* 2 (Jan. 2008).

[14] Y. Liu. *The Confusing Metrics of AP and mAP for Object Detection and Instance Segmentation.* https://medium.com/@yanfengliux/the-confusing-metrics-of-ap-and-map-for-object-detection-3113ba0386ef. Oct. 2018.

[15] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks”. In: *arXiv e-prints* (Dec. 2013). URL: https://arxiv.org/abs/1312.6229.

[16] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. “Selective Search for Object Recognition”. In: *International Journal of Computer Vision* (2013). URL: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

[17] R. B. Girshick. “Fast R-CNN”. In: *Computing Research Repository (CoRR)* abs/1504.08083 (2015). URL: http://arxiv.org/abs/1504.08083.

[18] S. Ren, K. He, R. B. Girshick, and J. Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Computing Research Repository (CoRR)* abs/1506.01497 (2016). URL: http://arxiv.org/abs/1506.01497.

[19] J. Redmon and A. Farhadi. “YOLOv3: An Incremental Improvement”. In: *Computing Research Repository (CoRR)* abs/1804.02767 (2018). URL: http://arxiv.org/abs/1804.02767.

[20] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. In: *Computing Re-*

search Repository (CoRR) abs/1506.02640 (2015). URL: http://arxiv.org/abs/1506.02640.

[21] J. Redmon and A. Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *Computing Research Repository (CoRR)* abs/1612.08242 (2016). URL: http://arxiv.org/abs/1612.08242.

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. “SSD: Single Shot MultiBox Detector”. In: *Computing Research Repository (CoRR)* abs/1512.02325 (2016). URL: http://arxiv.org/abs/1512.02325.

[23] J. Dai, Y. Li, K. He, and J. Sun. “R-FCN: Object Detection via Region-based Fully Convolutional Networks”. In: *Computing Research Repository (CoRR)* abs/1605.06409 (2016). URL: http://arxiv.org/abs/1605.06409.

[24] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. “Mask R-CNN”. In: *Computing Research Repository (CoRR)* abs/1703.06870 (2018). URL: http://arxiv.org/abs/1703.06870.

[25] P. O. Pinheiro, R. Collobert, and P. Dollár. “Learning to Segment Object Candidates”. In: *Neural Information Processing Systems (NIPS)*. 2015.

[26] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. “Learning to Refine Object Segments”. In: *European Conference on Computer Vision (ECCV)*. 2016.

[27] J. Dai, K. He, Y. Li, S. Ren, and J. Sun. “Instance-Sensitive Fully Convolutional Networks”. In: *European Conference on Computer Vision (ECCV) 2016*. Springer International Publishing, 2016, pp. 534–549.

[28] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. “Feature Pyramid Networks for Object Detection”. In: *Computing Research Repository (CoRR)* abs/1612.03144 (2016). URL: http://arxiv.org/abs/1612.03144.

[29] P. Ramachandran, B. Zoph, and Q. V. Le. “Searching for Activation Functions”. In: *Computing Research Repository (CoRR)* abs/1710.05941 (2017). URL: http://arxiv.org/abs/1710.05941.

[30] J. J. H. Schaub. *Analytical Mechanics of Space Systems.* 2nd ed. AIAA Education Series. American Institute of Aeronautics and Astronautics (AIAA), 2009.

[31] G. W. Hill. “Researches in the Lunar Theory”. In: *American Journal of Mathematics* 1.1 (1878), pp. 5–26. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2369430.

[32] W. H. Clohessy and R. S. Wiltshire. “Terminal Guidance System for Satellite Rendezvous”. In: *Journal of the Aerospace Sciences* 27.9 (1960), pp. 653–658. URL: https://doi.org/10.2514/8.8704.

[33] B. W. Barbee, J. R. Carpenter, S. Heatwole, F. L. Markley, M. Moreau, B. J. Naasz, and J. Van Eepoel. “A Guidance and Navigation Strategy for Rendezvous and Proximity Operations with a Noncooperative Spacecraft in Geosynchronous Orbit”. In: *The Journal of the Astronautical Sciences* 58.3 (July 2011), pp. 389–408. ISSN: 2195-0571. URL: https://doi.org/10.1007/BF03321176.

[34] F.-F. Li, R. Krishna, and D. Xu. *Stanford course CS231n: Convolutional Neural Networks for Visual Recognition.* http://cs231n.stanford.edu/syllabus.html. 2020.

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

[36] W. Yuxin, K. Alexander, M. Francisco, L. Wan-Yen, and G. Ross. *Model Zoo results*. https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md. 2019.

[37] J. Brooks. *COCO Annotator*. https://github.com/jsbroks/coco-annotator/. 2019.