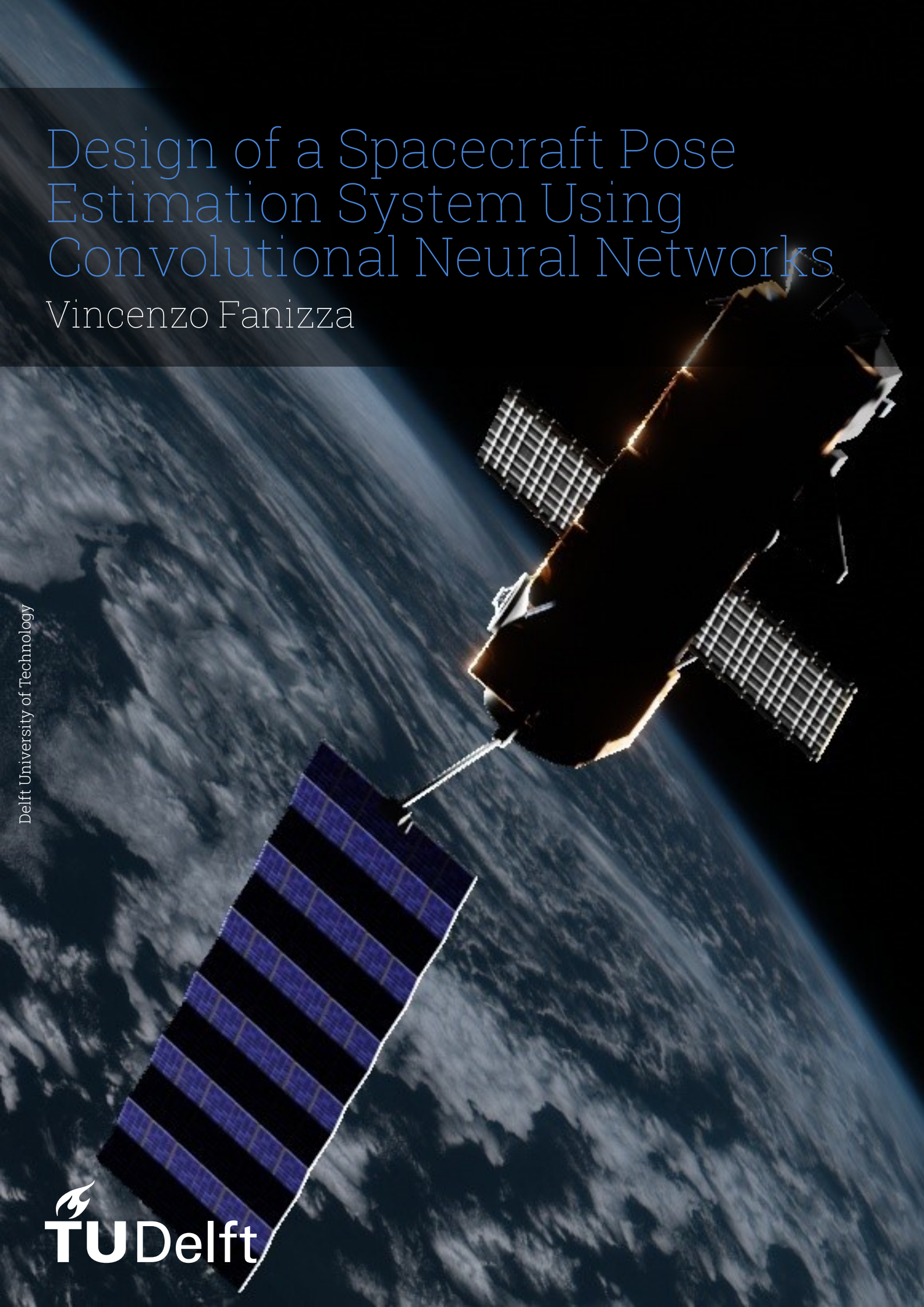
Design of a Spacecraft Pose Estimation System Using Convolutional Neural Networks

Vincenzo Fanizza



Delft University of Technology

TU Delft

Design of a Spacecraft Pose Estimation System Using Convolutional Neural Networks

by

Vincenzo Fanizza

Student Name	Student Number
Vincenzo Fanizza	5616484

In fulfilment with the requirements of the degree of

Master of Science
in Aerospace Engineering

at Delft University of Technology.
To be defended publicly on Tuesday November 28, 2023 at 14:00.

Project Duration:	15 March 2023 - 28 November 2023	
Faculty:	Faculty of Aerospace Engineering, Delft	
Master Track:	Space Flight	
Master Profile:	Space Exploration	
Thesis committee:	Dr. Ir. Dominic Dirkx	Chair, TU Delft
	Dr. Ir. Erik-Jan van Kampen	External Examiner, TU Delft
	Dr. Ir. Erwin Mooij	Supervisor, TU Delft

TU Delft

Preface

As I am about to finish my studies at TU Delft, this project represents the result of several months dedicated to make a contribution to the field of spacecraft relative navigation. Diving into such cutting-edge research has been an exciting and rewarding journey, where I have developed the belief that AI can truly transform space exploration through the development of innovative autonomous systems.

This could have not been possible alone. I owe a lot to the people who supported and guided me. Dr. Ir. Erwin Mooij, my supervisor, has been particularly helpful in guiding me through tough times and showing me the way forward.

The help from David and Tom at Ubotica Technologies was invaluable and allowed me to expand the scope of my research. Their support and sharing of expertise made a huge difference in the final work. I am also thankful to everyone else who contributed in various ways, no matter how small their help was.

My friends have been a great source of support from the start and, most importantly, I owe a huge debt of gratitude to my family. Their constant support, sacrifices, and belief in my abilities made this achievement possible. Their contribution means a lot more than just a thank you.

As this part of my life ends, I will take with me the lessons learned, experiences gained, and the meaningful connections made during my time here. While my time as a student at TU Delft has come to an end, I am thankful for the opportunities I have had and excited about using what I have learned, ready to explore new paths.

Vincenzo Fanizza
Delft, November 2023

Abstract

In the contemporary era, the space sector has experienced a remarkable resurgence, fueled by substantial government investments in a diverse array of space missions and programs. This renewed interest has led to substantial advancements in space technologies, but also to the need for sustainable operations for further growth. The quest for a self-sufficient space economy has raised critical questions regarding its viability and the timeline for its realization. One pivotal challenge pertains to Rendezvous and Proximity Operations (RPO), which intersect with vital facets crucial to the development of autonomous space systems on a large scale. These aspects encompass the management of space environmental hazards and the anticipated operational lifespan of satellites, constituting core considerations for this burgeoning space frontier.

To facilitate the scalability of RPO missions, it becomes imperative to streamline mission costs while simultaneously enhancing autonomy. Monocular vision-based navigation systems, harnessing the capabilities of a single monocular camera, are emerging as a compelling alternative to conventional sensors. This shift is primarily attributed to their advantages in terms of reduced mass, cost-effectiveness, lower power consumption, and reduced system complexity.

Furthermore, the contemporary landscape has witnessed the burgeoning role of Artificial Intelligence (AI) as a pivotal enabler for Autonomous Rendezvous and Docking (AR&D). A prominent illustration of this shift is discerned in the inaugural Satellite Pose Estimation Challenge (SPEC) in 2019. This competition unequivocally underscored the potential for Machine Learning (ML) to significantly enhance the pose estimation performance of uncooperative spacecraft. Notably, Convolutional Neural Networks (CNNs) have been explored in the context of spacecraft relative navigation.

Nevertheless, while CNN-based pipelines are rapidly gaining traction as a viable option for spacecraft relative navigation systems, substantial advancements remain imperative before deeming this technology space-ready. The nascent interest within the space community regarding ML applications signifies that optimal design choices and best development practices have yet to be comprehensively established. Preliminary findings, however, offer a glimpse into the potential of this approach to deliver precise and cost-effective solutions for spaceborne relative navigation.

This thesis presents the development of a pose estimation system for relative navigation around an uncooperative target. The system relies on images coming from a single optical camera, while adopting a multi-stage architecture involving an Object Detection (OD) network in conjunction with a Keypoint Detection (KD) network to extract essential features from images, followed by a pose reconstruction algorithm for determining the relative pose. To validate the effectiveness of this pipeline, a series of experiments is conducted.

First, a comprehensive investigation of how to bridge the performance gap between synthetic and testbed images is carried out using the SPEED+ dataset from Stanford University. The study also explores the benefits of training the system with photorealistic synthetic images. For this purpose, a novel ENVISAT dataset is generated, incorporating diverse illumination conditions for thorough pipeline testing. The research outcomes reveal that a thoughtfully designed synthetic augmentation pipeline reduces the pose estimation error by more than 50%. This result can be further enhanced by training the system on photorealistic images, encompassing challenging factors like adverse illumination conditions, which are difficult to replicate realistically using synthetic augmentations.

Second, the thesis engages in an architecture tradeoff analysis, comparing pipelines using different KD architectures. The tradeoff takes into consideration the pose estimation accuracy, overall number of parameters, and inference time on the target device. The results demonstrate that using an HRNet-w32 model yields the highest accuracy while still meeting the necessary inference time requirements.

The system developed in this thesis has the potential to outperform existing methods. Because of its lightweight design and being fully developed using synthetic data, it proves to be scalable, while its compliance with the limitations of space hardware demonstrates the feasibility of such a design for an actual space mission. This work is expected to boost further research and design an accurate and scalable relative navigation system, a key step towards the achievement of a sustainable space economy.

Contents

Nomenclature

Abbreviations

ADR	Active Debris Removal
ADRIOS	Active Debris Removal and In-Orbit Servicing
AI	Artificial Intelligence
AP	Average Precision
AP3P	Algebraic Perspective-3-Point
API	Application Programming Interface
AR&D	Autonomous Rendezvous and Docking
ARCSS	Autonomous Rendezvous and Capture Sensor System
AVGS	Advanced Video Guidance Sensor
CEPPnP	Covariance Efficient Procrustes Perspective-n-Point
CMX	Connection Matrix
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CoM	Centre of Mass
CPN	Cascaded Pyramid Network
CRP	Classical Rodrigues Parameter
CV	Computer Vision
DCM	Direction Cosine Matrix
DDR	Double Data Rate
DIAS	Data Integration and Analysis System
DL	Deep Learning
DNN	Deep Neural Network
DoF	Degrees of Freedom
EKF	Extended Kalman Filter

ELU	Exponential Linear Unit
EO	Earth Observation
EOL	End Of Life
EPnP	Efficient Perspective-n-Points
ESA	European Space Agency
FC	Fully Connected
FLOPs	Floating Point Operations
FOV	Field Of View
FPS	Frames Per Second
GC	Global Context
GEO	Geostationary Earth Orbit
GNC	Guidance Navigation and Control
GPU	Graphic Processing Unit
GUI	Graphic User Interface
HPC	High-Performance Computer
HRNet	High-Resolution Network
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection-over-Union
IP	Image Processing
IR	Intermediate Representation
KD	Keypoint Detection
LEO	Low Earth Orbit
LOS	Line-Of-Sight
LPN	Lightweight Pose Network
LVLH	Local-Vertical Local-Horizontal
mAP	mean Average Precision
ML	Machine Learning
MRP	Modified Rodrigues Parameter
NCS	Neural Compute Stick

NMS	Non-Max Suppression
NN	Neural Network
NST	Neural Style Transfer
OD	Object Detection
OE	Orbital Express
OMV	Orbital Maneuvering Vehicle
ONNX	Open Neural Network Exchange
OOS	On-Orbit Servicing
PnP	Perspective-n-Points
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Unit
RGB	Red-Green-Blue
RMSE	Root Mean Squared Error
RoI	Region of Interest
RPO	Rendezvous and Proximity Operations
SE	Squeeze-and-Excitation
SE	Squeeze-and-Exitation
SGD	Stochastic Gradient Descent
SHAVE	Streaming Hybrid Architecture Vector Engine
SPEC	Satellite Pose Estimation Challenge
SPEED	Spacecraft PosE Estimation Dataset
SPN	Spacecraft Pose Network
SQPnP	Sequential Quadratic Perspective-n-Points
SSD	Single Shot MultiBox Detector
SSH	Secure Shell
STS	Space Transportation System
SVD	Sharma-Ventura-D'Amico
TRL	Technology Readiness Level
TRON	Testbed for Rendezvous and Optical Navigation

VESPA	VEGA Secondary Payload Adapter
VGG	Visual Geometry Group
VPU	Vision Processing Unit
VVS	Virtual Visual Servoing
YOLO	You Only Look Once

Greek symbols

$\boldsymbol{\pi}$	Neural network parameters [-]
Φ	Euler principal angle [rad]
ϕ	Euler angle describing a rotation around the x axis of a reference frame [rad]
ψ	Euler angle describing a rotation around the z axis of a reference frame [rad]
θ	Euler angle describing a rotation around the y axis of a reference frame [rad]

Roman symbols

$+X$	Positive direction along the x axis [-]
$+Y$	Positive direction along the y axis [-]
$+Z$	Positive direction along the z axis [-]
$\hat{\mathbf{C}}_{T/C}$	Reconstructed relative rotation matrix [m]
$\hat{\mathbf{t}}_{T/C}$	Reconstructed relative position [m]
$\hat{\mathbf{x}}$	Unit vector along the x axis [-]
$\hat{\mathbf{y}}$	Unit vector along the y axis [-]
$\hat{\mathbf{z}}$	Unit vector along the z axis [-]
$\hat{e}$	Euler principal axis [-]
$\mathcal{L}$	Loss function [-]
$\mathbf{B}^{[l]}$	Bias matrix of the l-th layer of a neural network [-]
$\mathbf{C}_{B/A}$	Direction cosine matrix from frame A to frame B [-]
$\mathbf{H}$	Pixel heatmap [-]
$\mathbf{K}_C$	Camera intrinsic parameter matrix [-]
$\mathbf{P}_C$	Camera pose matrix [-]
$\mathbf{q}_A$	Quaternion describing an Euler eigenaxis rotation with respect to frame A [-]

$\mathbf{q}_v$	Quaternion vector part [-]
$\mathbf{r}$	Position vector [m]
$\mathbf{S}$	Pixel heatmap after exponential mapping [-]
$\mathbf{T}$	Arbitrary translation vector [-]
$\mathbf{t}$	Relative position vector [m]
$\mathbf{v}$	Arbitrary vector [-]
$\mathbf{W}^{[l]}$	Weight matrix of the l-th layer of a neural network [-]
$\mathbf{w}_m^{[l]}$	Weight vector belonging to the m-th neuron of the l-th layer of a neural network [-]
$\mathbf{Z}^{[l]}$	Output matrix of the l-th layer of a neural network [-]
A	Reference frame A [-]
a	Semi-major axis [m]
$b_m^{[l]}$	Bias belonging to the m-th neuron of the l-th layer of a neural network [-]
C_{ij}	Direction cosine matrix entry at row i and column j [-]
E_P	Pose score [-]
E_R	Rotation part of the pose score [°]
E_T	Translation part of the pose score [m]
f	Focal length [m]
h	Image height [pix]
h_a	Apogee altitude [m]
h_p	Perigee altitude [m]
J	Cost function [-]
u	Horizontal pixel coordinate [pix]
v	Vertical pixel coordinate [pix]
w	Image width [pix]
$z_m^{[l]}$	Output belonging to the m-th neuron of the l-th layer of a neural network [-]

Part I

Introduction and Background

1 Introduction

This chapter serves as an introduction to the thesis project. Section 1.1 outlines the problem this work collocates within. Following that, Section 1.2 delves into the key developments that have shaped and influenced this research. In Section 1.3, the research questions are formulated, whereas Section 1.4 concludes the chapter by providing an outline of the report's structure.

1.1. Problem and Relevance

The 21st century has seen a renewed interest in the space sector. Governments have sustained with increasing funding a variety of space missions and programs. This has led to large improvements in the advancement of space technologies and to achievements unfeasible until a few decades ago. Making a significant amount of resources available for the development of innovative space systems has also had the effect of giving more importance to private players. There are cases of both already-existing and newly-founded companies that have had the chance to grow exponentially in the last 20 years and have established themselves as leaders in the industry. Two examples are given by OHB[1] - founded in 1981 in Germany and now an international company also based in Sweden and Italy - and Planet[2] - founded only in 2010 and now one of the largest space data providers along with ESA and NASA. The undertaken strategy is clear: creating a sustainable space economy where companies can develop new technologies and make progress in the field in a self-sufficient manner.

Aiming for a self-sufficient space economy has raised many questions about its feasibility and the timeframe within which this can happen. In this regard, an important problem to tackle is that related to Rendezvous and Proximity Operations (RPO), as it is related to two factors that are of crucial importance for the development of autonomous space systems on a large scale, such as space environmental hazards and the expected operational lifetime of a satellite.

The first factor concerns the problem of space debris, which has received increasing attention given the increasing number of objects sent into space and the consequent increased risk of collision between two of them. Although the number of debris items currently orbiting the Earth still does not prevent access to space, some argue that the current debris population might be large enough to trigger a sequence of catastrophic cascade collisions (Liou, 2011; Bonnal et al., 2013). A possibility to reduce mission risks related to the space environment is given by Active Debris Removal (ADR), where an item of debris is manually removed by a spacecraft. The RemoveDEBRIS mission (Forshaw et al., 2016) has demonstrated the progress of debris-capture systems by capturing two CubeSats in a controlled space environment, while several missions are being designed to remove the first piece of debris from Earth orbit. An example of this is given by the Clearspace-1 mission (Biesbroek et al., 2021), which will attempt to remove the upper stage of the VESPA (VEGA Secondary Payload Adapter) upper stage launched in 2013.

Related to the second factor, operational lifetime stands as one of the primary limitations for today's missions, limiting both the amount of collectable data and accomplishable tasks. On-Orbit Servicing (OOS) aims at extending the operational lifetime of a spacecraft by having a servicer spacecraft

[1] `https://www.ohb.de/` (visited on 26/02/2023)
[2] `https://www.planet.com/` (visited on 26/02/2023)

performing tasks related to the repair and maintenance of the target. By doing so, the duration of a mission can be extended significantly and increase the value return on Earth. OOS mission concepts have been evolving since as early as the 1980s, when the Orbital Maneuvering Vehicle (OMV) (Stephenson, 1988) was employed to enhance on-orbit capabilities of the Space Shuttle. It was designed as a monolithic platform incorporating all the required technologies and tools to perform the different tasks. Later on, several Space Transportation System (STS) missions were launched in the 1990s, servicing the Hubble Space Telescope among the others (Tatsch et al., 2006). The progress achieved in the final stage of the century boosted the development of innovative OOS systems to improve their technological and financial feasibility. As of today, several technology demonstrators have already been performed, such as the Orbital Express (OE) mission, and others are planned in the upcoming future, as is the case for the OSAM-1 mission.

Researching applications to accelerate the development and scalability of RPO missions has therefore high relevance when looking at the bigger picture of creating a sustainable space economy. Forecasting a growing demand for this kind of system in the upcoming decades, they need to be developed quickly and be scalable to ensure their application on a large scale.

1.2. Research Trigger

To ensure that RPO missions can be performed on scale, it is important to minimise mission costs while increasing autonomy. Monocular vision-based navigation systems based on a single monocular camera are becoming a viable alternative to other types of sensors, such as lidar or stereo cameras, given their lower mass, cost, power consumption, and complexity. Furthermore, recent years have seen the rise of Artificial Intelligence (AI) as an enabling technology for Autonomous Rendezvous & Docking (AR&D). The first Satellite Pose Estimation Challenge (SPEC) (Kisantal et al., 2020) held in 2019 clearly showed how better pose estimation performance of uncooperative spacecraft can be achieved using Machine Learning (ML). In particular, after showcasing their potential in terrestrial applications, applications of Convolutional Neural Networks (CNNs) related to spacecraft relative navigation have been investigated (Sharma et al., 2018a). Compared to traditional Image Processing (IP) algorithms, CNN-based approaches have the advantage of being more robust against illumination conditions and image noise. Because of these benefits, AI-based systems represent one of the most active areas of research in the field of spacecraft relative navigation.

Despite the latest advancements, the development of CNN-based relative navigation systems is still far from seeing the first technology demonstration mission. The reason behind such a gap lies in the challenges related to integrating ML technology in space systems. According to Izzo et al. (2019), there are two major challenges related to the development of ML systems for spacecraft: the first one is the lack of large, publicly available datasets that can be used for model development, while the second one stems from the severity of on-board power constraints. Both aspects are in direct contrast compared to terrestrial applications of ML systems, for which large datasets containing everyday objects, such as ImageNet (Deng et al., 2009) and Common Objects in Context (COCO) (Lin et al., 2014), can be collected and computational power is virtually unlimited.

The lack of training data is a major problem when developing ML models for relative navigation. Most of the collected images are often proprietary, and in general, there are only a handful of datasets developed for spacecraft relative navigation and released to the community. To be effective for the selected mission profile, datasets need to be generated with a predefined target, meaning they are application-specific. This makes not only the data generation process slower but also the pipeline less prone to leverage models pretrained for other missions. In addition to this, even state-of-the-art datasets such as the Spacecraft PosE Estimation Dataset (SPEED) by Sharma et al. (2019) and its enhanced version, SPEED+ (Park et al., 2022c), experience the so-called *domain-gap* issue, where the performance drops significantly when testing on real images after training solely on synthetic ones. This results in a lack of robustness in the system. At the moment, the gap in performance between synthetic and testbed data is still significant and represents one of the greatest challenges to overcome before sending an AI-based relative navigation system into orbit.

A possible direction is that of using testbed data during training, however, as argued by Cassinis (2022), this type of resource is rarely available to research institutes. Even facilities previously used, such as the Robotic Testbed for Rendezvous and Optical Navigation (TRON) at Stanford University (Park et al., 2021), require several months to generate a few thousand images. Using a testbed facility to

develop a space-compliant relative navigation system requires investing extensive resources at early mission stages and strongly compromises its scalability. Alternatively, models could be trained on a combination of synthetic and real images. Park et al. (2022b), for instance, perform domain refinement of a pose estimation network after training only on synthetic images. The pipeline assumes that the model can be finetuned in orbit and that real images are available, allowing the model to be trained directly on the target data. This process, however, is severely constrained by the on-board power constraints, and the resources reserved for training may not be enough to improve performance significantly.

A complementary approach would be to fully rely on synthetic data, making them as realistic as possible. Currently, several spacecraft pose estimation pipelines make use of synthetic augmentations aimed at reproducing corruptions observed in real space imagery Park et al. (2022b), Cassinis (2022), and Park et al. (2023), whereas some research decided to include all the intended effects in a simulation environment within a rendering software (Proença et al., 2020). In either case, if synthetic images can be generated realistically enough, the performance gap is bound to disappear. Questions are still open, however, about the level of realism that can be reached by purely relying on synthetic data for training.

Concerning power limitations when running AI algorithms, more attention is being paid to limiting the computational complexity of ML models. A. G. Howard et al. (2017), for instance, introduced a family of architectures, MobileNets, designed to reduce the architecture size and complexity while keeping accuracy unaltered. As such a feature fits nicely with the limitations typically encountered by space systems, lightweight architectures have been used for applications in the space domain as in the work by Park et al. (2019) and Van der Heijden (2022). Because of the severity of power constraints for space systems, There is often a tradeoff to be made between the model inference speed and accuracy, which becomes even more crucial when AI applications have to be run *on-the-edge*, that is, with limited power available (Furano et al., 2020). This requires to take into account the limitations of the device of choice.

As of today, very little characterisation has been performed on space-representative hardware of ML models for spacecraft pose estimation. To the author's knowledge, only Cassinis (2022) has been able to test a model for spacecraft pose estimation on an edge device. Being able to accurately describe the behaviour of a model on the target device is crucial for the robustness of the final system, and a clear gap exists in the literature about the different steps required for model characterisation.

Although CNN-based pipelines are becoming a plausible alternative for spacecraft relative navigation systems, there is still significant improvement to be made before considering this technology ready for an actual space mission. This is due to the only recent interest of the space community for ML applications, meaning optimal design choices and best development practices still have to be fully determined. Preliminary results, however, have shown how this type of system can potentially lead to an accurate and cost-effective solution for spaceborne relative navigation.

1.3. Research Questions

Prior to this research, a literature study was performed about the current state-of-the-art in spacecraft relative navigation. The major outcome of the study consists of the following main research question, which has driven the entire research project:

How can an accurate and robust CNN-based relative navigation system be developed to perform RPO missions in a way that is scalable and optimised for space hardware?

The question focuses on researching methodologies that not only lead to systems with an adequate level of performance for space applications, but also preserve scalability and hardware-driven optimisation of such methodologies. As already discussed in Section 1.1, RPO systems need to be developed keeping in mind the high demand forecast in the upcoming decades for either on-orbit debris removal or servicing operations. To establish a more structured research framework, the following sub-questions were identified:

1. **How can the domain gap between synthetic and testbed images be bridged using state-of-the-art data augmentations?**

   As already mentioned, leveraging synthetic images is a key component for scalable CNN-based relative navigation systems. The aim of this question is to understand how much data augmentations can improve the system performance when training only on synthetic images.

2. **Can the performance be further improved by training the models on photorealistic synthetic images?**

   This question investigates the possible benefits of including image corruptions in a realistic rendering environment instead of applying each effect separately using data augmentations. The findings related to this question can ideally be generalised to any RPO mission, paving the way for a new standard in synthetic data generation and stimulating the generation of a larger number of datasets for spacecraft relative navigation.
3. **What are the pitfalls and bottlenecks of a model conversion pipeline from the training format to a format compatible with the target device?**

   The goal of this question is to provide insights about model conversion pipelines to test them on the target device. Similarly to the previous question, the findings can be used to establish guidelines and best practices in the joint selection of hardware and architecture for spacecraft relative navigation systems.
4. **What are the best practices when performing a tradeoff between a model's accuracy and inference speed on the target device?**

   By studying the existing literature, only preliminary studies were found about architecture tradeoffs for spacecraft relative navigation pipelines. This thesis aims to expand the current knowledge base by describing and reporting the main results of an architecture tradeoff between accuracy and inference speed on space-representative hardware.

1.4. Report Structure

This report is structured as follows: Chapter 2 presents the main literature findings, providing a comprehensive overview of the existing research in the field, while Chapter 3 defines the mathematical framework required to describe the spacecraft kinematics, laying the foundation for understanding the physical principles involved. Chapter 4 introduces the basics of spacecraft pose estimation, discussing the fundamental techniques and methodologies employed in this crucial aspect of the research, and Chapter 5 delves into the required machine learning concepts, offering a detailed explanation of the algorithms and methodologies used in the data analysis and modeling process. Chapter 6 provides an in-depth overview of the system architecture, the software tools utilized, and the verification steps undertaken to ensure the reliability of the research, while Chapters 7 and 8 respectively focus on the OD and KD architectures, delving into their intricacies and functionalities. Chapter 9 introduces the main datasets used for this research, elucidating their significance in the experimentation process. Chapter 10 presents the results of the experiments conducted with synthetic augmentations and photorealistic rendering, shedding light on the outcomes and implications of these experiments, with Chapter 11 offering a comprehensive comparison of different architectures, along with detailed insights into the steps of their implementation and verification on Myriad X. Lastly, chapter 12 summarises the conclusions drawn from this research and outlines potential directions for future research.

2

Mission Heritage

In this chapter, a comprehensive review of the heritage relevant to this thesis is presented. Section 2.1 focuses on the historical background of AI-based relative navigation systems for RPO missions. Building upon this heritage, Section 2.2 outlines the reference scenario and discusses the framework and assumptions considered in this work.

2.1. Relevant Work

As introduced in Section 1.1, in the context of generating a sustainable space economy, RPO missions play a critical role as it would lead to enormous benefits for in-space operations.

The current population of space debris orbiting the Earth may be already sufficient to trigger the so-called Kessler syndrome - a phenomenon consisting of the formation of a debris belt around the Earth, generated by uncontrolled cascade collisions between debris (Kessler et al., 1978) - jeopardising the safety of the space environment for future generations. Even without a catastrophic collision occurring in a short time, studies have shown how the situation may still become critical within a time between 50 and 200 years (Liou, 2011; Bonnal et al., 2013). To avoid this scenario, it is not only necessary to adopt proper End-Of-Life (EOL) management, but also to actively de-orbit several debris items every year. It follows that developing effective and large-scale ADR applications is of primary importance.

With the current cost of a space mission, the possibility of extending the lifetime of a satellite can bring dramatic advantages. The purpose of OOS is precisely that of performing assembly, repair, and maintenance on satellites in space, with the ultimate purpose of extending their operational lifetime. With the field being conceived as early as the 1970s, several technology demonstrators and missions have already been performed (Tatsch et al., 2006). Despite the majority of mission concepts being driven by the design of the robotic system performing the desired tasks (Tatsch et al., 2006; Ding et al., 2021), ensuring safety during the rendezvous and the proximity phase is key for the success of any OOS mission.

Autonomous navigation is foreseen to be crucial in the development of sustainable RPO missions, for which accurate knowledge about the target spacecraft state is required. In the case of an ADR task, the target is non-cooperative and there is only an approximate state knowledge provided by tracking stations on the ground. Furthermore, the target may be damaged due to previous collisions, adding the spacecraft geometry to the list of unknowns. To tackle these challenges, previous years have seen an increasing effort in developing autonomous relative navigation systems to increase on-orbit capabilities of RPO missions.

An essential step to perform relative navigation is pose estimation, for which stringent accuracy requirements are usually defined. Solutions adopted for space applications usually involve reframing state-of-the-art methods for Earth applications to the space problem of interest. Therefore, several insights on spacecraft pose estimation can be gained by well-established methods for terrestrial-related tasks, such as human pose estimation or camera viewpoint estimation.

Trying to tackle the challenge of performing relative navigation between spacecraft, the scientific community previously focused on techniques based on feature matching, where a large number of pose hypotheses is tested on each real image of the target and the one minimising a predefined distance

function is chosen for initialisation (Petit et al., 2012; Kanani et al., 2012). Sharma et al. (2018a), however, have shown how Neural Networks (NNs) can be deployed as a resource-efficient pose initialisation strategy by predicting an approximate relative pose subsequently refined by more elaborate pose solvers, boosting the interest in AI-based technologies.

2.1.1. Active Debris Removal

So far, there has not been one mission removing a real item of debris in space. Despite the increasing number of debris surrounding the Earth, Bonnal et al. (2013) outline how cleaning space from debris involves not only technological but also normative and economic challenges. Despite that, the achieved technological progress has allowed performing several in-space technology demonstrations.

The RemoveDEBRIS mission, launched in 2018, demonstrated several concepts to perform ADR (Forshaw et al., 2016). The system consisted of a microsatellite platform as chaser and two CubeSats as targets. The mission aimed to test different technologies for active removal, such as net and harpoon capture, as well as a vision-based navigation system consisting of a standard camera and a flash lidar. This was done by first ejecting one of the two CubeSats from the main platform so that it would acquire a small, out-of-plane relative velocity. The mission, however, only served as a proof of concepts, with the most valuable insights being gained about capture system design rather than the navigation system. The target, in fact, was tracked while drifting away using IP algorithms (Aglietti et al., 2020), whereas NNs have recently become the preferable choice for most vision-based tasks.

Nonetheless, proving that cost-effective technology can be used for ADR tasks, the RemoveDEBRIS mission enabled the planning of several other technology demonstrators. Following a similar concept, the ELSA-d mission by Astroscale wants to test new innovative technologies (Blackerby et al., 2018). Launched in 2021, the mission consists of a chaser and a semi-cooperative target performing ADR tasks of increasing complexity and risk. The chaser will have to capture a non-tumbling target first, then perform the same task with a tumbling target, and finally demonstrate the adopted relative navigation system. The capture is performed with a magnetic capture system, while a docking plate mounted on the target is tracked by the chaser to determine the relative attitude. It mounts a sensor suite consisting of day- and night-cameras, ranging devices, and an illuminator, which results in a significant system complexity. As argued by Cassinis (2022), the flexibility and little resources required by monocular cameras could result in a sensor suite only relying on this type of sensor, if coupled with a navigation algorithm robust enough.

Research is not only focusing on technology demonstrators but also on performing the removal of the first item of debris from space. The European Space Agency (ESA) has been very active in this regard, allocating funds for the e.Deorbit mission in collaboration with OHB (Wieser et al., 2015). The mission plans to remove the largest piece of debris currently in orbit around the Earth, *i.e.*, the ENVISAT spacecraft, with a mass of more than 7,000 kg and the largest dimension above 10 m. From the illustration provided in Figure 2.1[1], it can be seen how, besides its large size, the spacecraft structure is very asymmetric because of the presence of the on-board instrumentation, antennas, and the large solar panel. This results in a tumbling motion more unpredictable and the need for a more sophisticated capture system. Such a challenging goal drove the design of the mission Guidance Navigation & Control (GNC) system, which will include a relative navigation system to approach the target (Telaar et al., 2017). The system relies on different sensors depending on the mission phase. As shown in Figure 2.2, at the rendezvous entry gate the system employs a camera and then a lidar to obtain Line-Of-Sight (LOS) and range measurements, while a 3D lidar is used when closer than 100 m to the target for the full pose estimation. This approach, driven by the different observation conditions and the use of a robotic arm for capturing, results in a high system complexity which, combined with the heavy use of active sensors, makes the system costly and not scalable. The launch was initially planned for the early 2020s, but it was recently rescheduled to no earlier than 2025. One of the reasons for such a decision was the increasing effort put by the agency into the ADRIOS (Active Debris Removal and In-Orbit Servicing) program and its first mission, Clearspace-1 (Biesbroek et al., 2021). The mission has the goal of removing the ESA-owned VESPA upper stage launched in 2013, paving the way for future missions and opening a new market for ADR and OOS. The chaser will mount several active sensors, such as a radar and a lidar, which have a primary impact on the on-board power requirement.

In conclusion, the existing heritage in ADR missions outlines a growing interest and progress in the

[1] `https://www.esa.int/ESA_Multimedia/Images/2012/05/Envisat` (visited on 07/10/2023)

Figure 2.1: Illustration of the ENVISAT satellite.

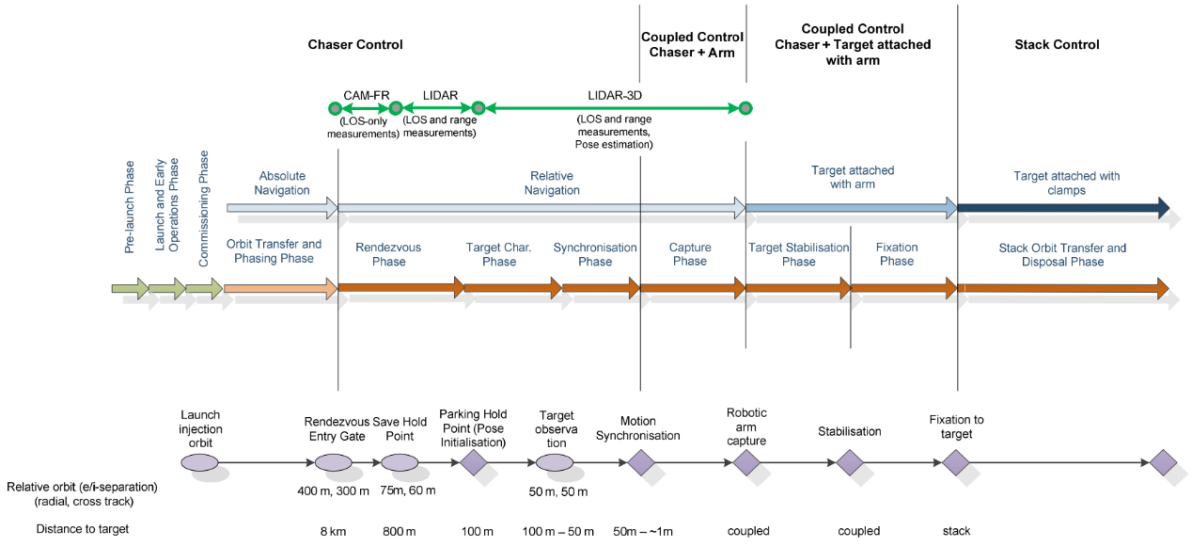


Figure 2.2: Overview of the different phases of the e.Deorbit mission (Telaar et al., 2017).

field. Technology demonstrations are regularly performed to showcase technological advancement and bring the creation of a sustainable market closer. In addition, the critical situation concerning the current space debris population has led public players, such as ESA, to fund future debris-removal missions. Currently, the main limitations concern the design of optimal capture systems and the complexity of the sensor suite, making the removal of the first debris still a few years away. Despite that, the intention of mitigating the debris problem as soon as possible is indubitable, with the interest in removing large and dangerous objects first, such as ENVISAT and VESPA, being a clear indication of that.

2.1.2. On-Orbit Servicing

As the concept of OOS goes back several decades, there have been multiple missions in the past proving OOS capabilities. Early system concepts consisted in designing a monolithic platform incorporating all the technologies required for the different tasks. This was the case for OMV (Stephenson, 1988), employed as a short-range system capable of moving payloads around the Space Shuttle in the 1980s. The system, however, lacked enough maturity and robustness in its design, making it an application far from profitable (Tatsch et al., 2006).

This led investors at the time to put little trust in robotic systems for spacecraft servicing, and the entire OOS niche saw little development in the subsequent years. The only OOS activities bringing tangible progress to the field were part of STS missions, servicing the Intelsat-VI spacecraft and the Hubble Space Telescope as well as the ETS-VII mission, where a chaser satellite with a robotic arm performed for the first time AR&D and OOS with a target (Tatsch et al., 2006). According to Pedersen et al. (2002), however, recent years have seen improvements in space robotic technologies, making OOS missions not only feasible according to modern space standards but also with the potential of being financially viable.

It is therefore not a surprise that OOS missions have been launched in the past 20 years not only to increase the Technology Readiness Level (TRL) of different architectures but also to try to create a profitable market. The DARPA OE mission, launched in 2007, pioneered in autonomous OOS technologies. The spacecraft Autonomous Rendezvous and Capture Sensor System (ARCSS) incorporated an advanced sensing system composed of two visible-light cameras, an infrared camera, a laser rangefinder, and the Advanced Video Guidance Sensor (AVGS). By combining measurements coming from the different sensors, ARCSS ensured accurate navigation throughout the mission (R. T. Howard et al., 2008). OE was able to demonstrate high-risk technologies, such as autonomous on-orbit refuelling and payload delivering (Whelan et al., 2000), attracting the interest of key industry players in the potential of autonomous systems for OOS, although yet not proving the scalability of the system.

Further demonstrations are planned for the upcoming future. NASA has invested in the OSAM-1 (previously Restore-L) mission aimed at servicing the Earth Observation (EO) satellite Landsat 7 (Reed et al., 2016). The technology demonstration will consist of first rendezvous and docking with the target, refuelling it and relocating it before undocking and re-entering the atmosphere. RPO manoeuvres are performed thanks to an autonomous, real-time relative navigation system, which relies on visual and infrared cameras as well as a lidar. The mission is part of a broader initiative by NASA, the OSAM program, aimed at improving OOS capabilities through robust and reliable autonomous systems like AI-based rendezvous and docking systems, showing how the latest advancements in the field have captured the interest of space professionals.

The overall picture of OOS missions is that of a field that, rejuvenated by the boosted interest in the space industry, is making significant progress and is slowly seeing the creation of a feasible market with the first commercial applications currently under development. Similarly to the role of ESA in ADR, NASA has adopted the approach of funding a program to dramatically increase OOS capabilities in the upcoming years. Although the journey remains long in this regard, cutting-edge technologies, such as AI-based systems, are being more often considered to enable the large-scale diffusion of OOS missions.

2.1.3. Pose Estimation

When talking about relative navigation in the context of an RPO mission, one refers to the determination of the chaser state with respect to the target. An essential step in such a process is the determination of the relative pose, *i.e.*, the relative position and attitude, between the chaser and the target. Spaceborne pose estimation pipelines often take inspiration from pipelines for terrestrial applications, for which more exhaustive research can be conducted.

In this regard, human pose estimation is a problem that has been studied for several years. It consists of estimating the pose of a human in a picture by identifying their joints, such as shoulders, knees, and ankles. Methods are evaluated against labeled images containing the ground truth joint locations and relative pose. An example of an annotated image is provided in Figure 2.3. Full labels typically include different types of annotations. Besides the ground-truth pose, the image also includes keypoint locations and bounding box labels, which leaves greater flexibility in developing the best algorithm. For instance, one could develop a pipeline determining the human pose by first identifying the joint areas, detecting their locations, and reconstructing the pose as a geometric problem.

Existing datasets for pose estimation represent humans in a variety of situations: from bicycling to

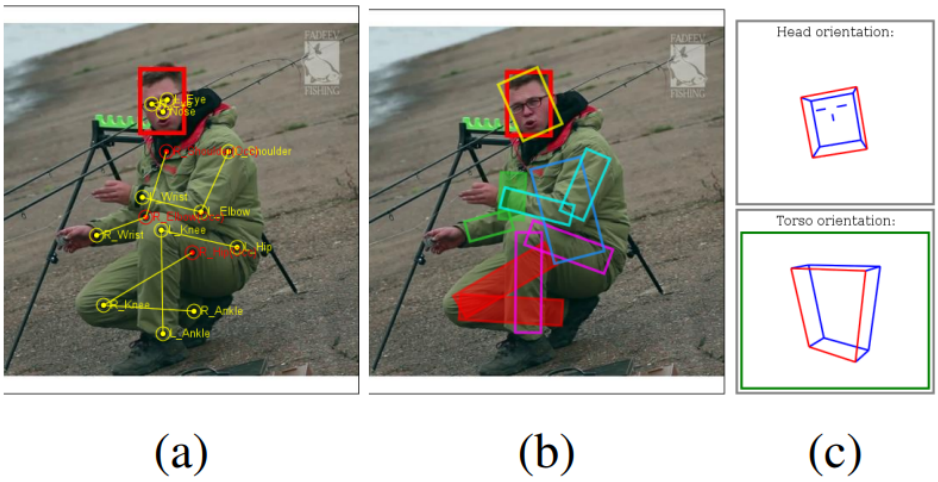


Figure 2.3: Annotations for an image in a dataset for human pose estimation (Andriluka et al., 2014). Labels typically include (a) joint locations, (b) bounding boxes, and (c) relative pose.

dancing and from sitting to standing (Figure 2.4). This is required given the countless scenarios in which human pose estimation algorithms can be applied. Robustness across different poses and observation conditions is therefore a consequence of the generic application domain they are conceived for. To exhaustively assess a pose estimation pipeline, there is a need for large amounts of images covering a sufficient number of cases. Andriluka et al. (2014) emphasise the importance of taking into account several factors when generating a dataset. Not only a high variance in terms of poses, clothes, and situations should be ensured, but also the many aspects affecting the complexity of the pose estimation task, *e.g.*, body pose, occlusions, truncations, and viewpoint location, should be taken into account when evaluating the performance. In the context of spacecraft pose estimation, a dataset may be allowed to account for less variety by representing only one target, but the error margin is drastically reduced when the reconstructed pose is the main measurement used by the navigation system.

While former state-of-the-art methods rely on techniques like pictorial structures (Pishchulin et al., 2013) and the flexible mixture of parts (Yang et al., 2012), Toshev et al. (2014) proved how Deep Learning (DL) is a promising approach for this task. The work formulates the pose estimation problem as a regression problem tackled by using a cascade of CNN regressors, where each network refines the pose solution found by the previous network. This method allowed to achieve results comparable with state-of-the-art methods, however, the arbitrary cascade length may result in excessive architectural complexity. Furthermore, later CNN models have been able to raise the performance bar.

CNNs have been able to set the current state of the art for pose estimation and similar tasks. Besides the above-mentioned work by Toshev et al. (2014), K. Sun et al. (2019) used CNNs to achieve state-of-the-art performance over two separate benchmark datasets for human pose estimation. The proposed architecture, HRNet, maintains a high output resolution across the different layers by connecting multi-resolution sub-networks in parallel and includes communication units that allow the exchange of information between sub-networks. Thanks to these features, the network can output accurate keypoint heatmaps that leverage the large amount of information retained thanks to the high-resolution approach. Sermanet et al. (2013) set a new benchmark using CNNs for object detection and obtained very competitive results for object recognition and localisation. Although not directly solving the pose estimation problem, these tasks may be performed within a pose estimation pipeline where a Region of Interest (RoI) containing the target has to be identified first.

The drawback of using a CNN-based approach is that the amount of data required increases significantly, as a large part of the data has to be used for training. With regards to camera pose estimation, Su et al. (2015) recommend the use of open-source 3D model repositories to generate large synthetic datasets, while Kendall et al. (2015) propose the use of such models in combination with large real datasets generated by, *e.g.*, videos. By leveraging larger datasets already available through transfer learning and the great learning capacity of CNNs, the two methods are able to outperform well-established ones, also showing better robustness in adverse observation conditions as in the work by Kendall et al. (2015).

Figure 2.4: Overview of different scenes captured in a dataset for human pose estimation (Andriluka et al., 2014).

The strong trend in developing CNN-based solutions to solve the pose estimation problem hints at the potential of this type of architecture. New benchmarks are periodically attained using innovative solutions and the existing literature shows that CNNs are adaptable to different tasks. A critical problem to the development of CNN models, however, is the size and representativeness of pose-estimation datasets. Large and accurate datasets are difficult to obtain, reason for which the strategy typically adopted involves the combined use of synthetic and real data together with transfer learning to boost the model performance. In conclusion, the literature indicates how CNNs are a solution providing high accuracy, even though the training of new models can be quite intensive in terms of resources.

2.1.4. Spacecraft Relative Navigation

Ensuring autonomous and accurate relative navigation between two spacecraft is a fundamental step towards the design of large-scale RPO missions. Traditionally, spacecraft pose estimators for relative navigation rely on IP algorithms focusing on certain image features, such as edges, corners, and lines (Petit et al., 2012; Kanani et al., 2012). Once these hand-engineered features are retrieved, the processed image is compared to a 3D model of the target to determine the relative pose. This task usually involves a pose initialisation step - where, from a set of predefined poses, the one more accurately matching the image is chosen as initial guess - and a pose solving step - where the initial pose is refined by minimisation of a distance function. The main disadvantages of IP-based approaches are a lack of robustness against adverse illumination conditions and the computational complexity caused by evaluating a large number of poses at the initialisation step (Sharma et al., 2018b).

Despite recent IP methods trying to solve the aforementioned issues, such as the Sharma-Ventura-D'Amico (SVD) method (Sharma et al., 2018b), CNNs are being adopted more often as feature extractors or pose initialisers for monocular systems. Sharma et al. (2018a) propose a pose determination method based on a CNN providing an initial pose guess for real-time on-board relative navigation. The author used the AlexNet architecture (Krizhevsky et al., 2017), pretrained on ImageNet and retrained on synthetic images of the PRISMA-Tango spacecraft (D'Amico et al., 2013) rendered using OpenGL. The results show how the proposed framework may be used to initialise more accurate pose solvers and speed up navigation loop convergence time, although more accurate methods have been proposed since then.

Sharma et al. (2019) leveraged this pipeline to generate SPEED, setting a new state of the art for synthetic datasets for spacecraft pose estimation. From the samples shown in Figure 2.5, it can be seen how the dataset represents the spacecraft with different orientations and at a variable relative distance,

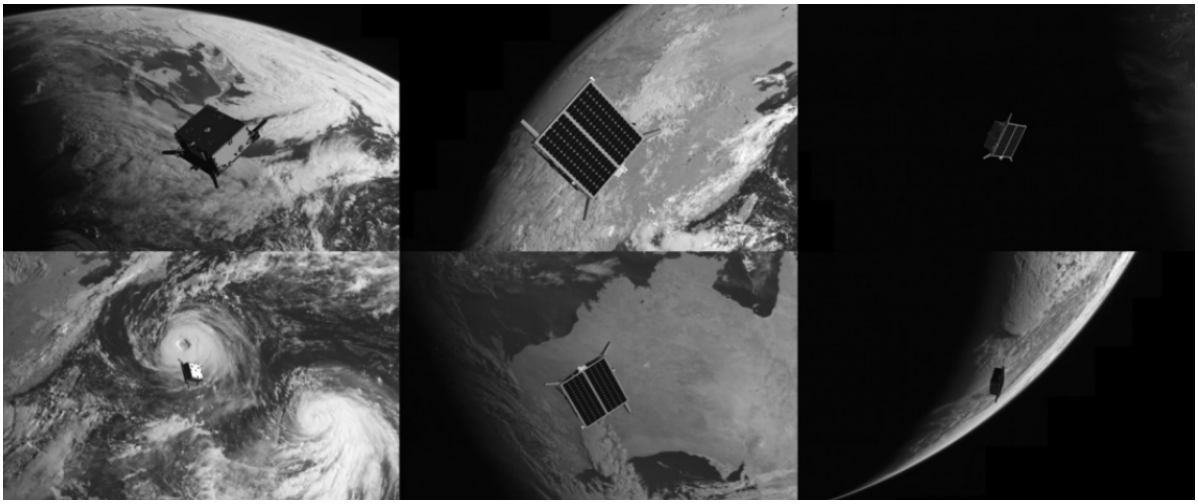


Figure 2.5: Samples from the SPEED dataset (Sharma et al., 2019).

besides including the Earth in the background to account for more challenging observation conditions. However, the realism of the illumination conditions is yet to be proven. The author also proposed a new framework, Spacecraft Pose Network (SPN). This optimised architecture consists of a CNN determining both the spacecraft attitude and the object bounding box, which are then combined to determine the relative position of the target through a Gauss-Newton algorithm. Despite splitting the pose estimation problem into tasks simpler to solve for the network, the method ended up struggling to reconstruct the relative position in synthetic images and the attitude in testbed ones.

The division of the pose estimation task in sub-problems is a direction further explored by Park et al. (2019) and B. Chen et al. (2019). Both works tackle the pose estimation problem by dividing it into Object Detection (OD) and Keypoint Detection (KD), and providing the estimated keypoint location to a pose solver. The two pipelines achieved a competitive score in the 2019 SPEC (Kisantal et al., 2020), while a similar system concept has been reproposed by the winners of the 2021 SPEC (Park et al., 2023), proving how an approach based on task simplification brings great benefits in terms of overall system accuracy. However, a major limitation of their work is the use of real images during training.

Furthermore, recent work by Cassinis (2022) integrates the pipeline within a relative navigation loop by combining a CNN-based feature extractor outputting keypoint heatmaps with a Covariance Efficient Procrustes Perspective-n-Point (CEPPnP) pose solver and an Extended Kalman Filter (EKF). By using heatmaps, it is possible to associate every pixel in the image to a pseudo-probability of that pixel containing the keypoints, including a statistical aspect in the detection process that enables their use to generate input measurements for a state estimation algorithm. The heatmaps shown in Figure 2.6 display how a model properly trained is able to generate Gaussian heatmaps with their center coinciding with the keypoint location. In the case of the pipeline designed by Barad (2020), the KD network detects 11 keypoints corresponding to strong visual features of the Tango spacecraft, such as antennas and edges. Although the method is still far from meeting in-space accuracy requirements, encoding the statistical aspect of feature detection in the model allows for a more reasonable estimate of the measurement uncertainty.

Similarly to its terrestrial counterpart, spacecraft pose estimation is relying always more heavily on CNN-based solutions. In particular, a promising pose estimation concept consists of splitting the pose estimation task into simpler sub-problems, namely OD and KD, followed by a pose reconstruction step, to maximise the network performance in each task. The latest developments in the field have also seen the integration of this architecture in a full relative navigation pipeline, proving how such a system concept may be used in the future for actual space missions. Although the current TRL of CNN-based spacecraft navigation systems is far from seeing their application in space soon, research in past years suggests how the concept has the potential to reach such a milestone.

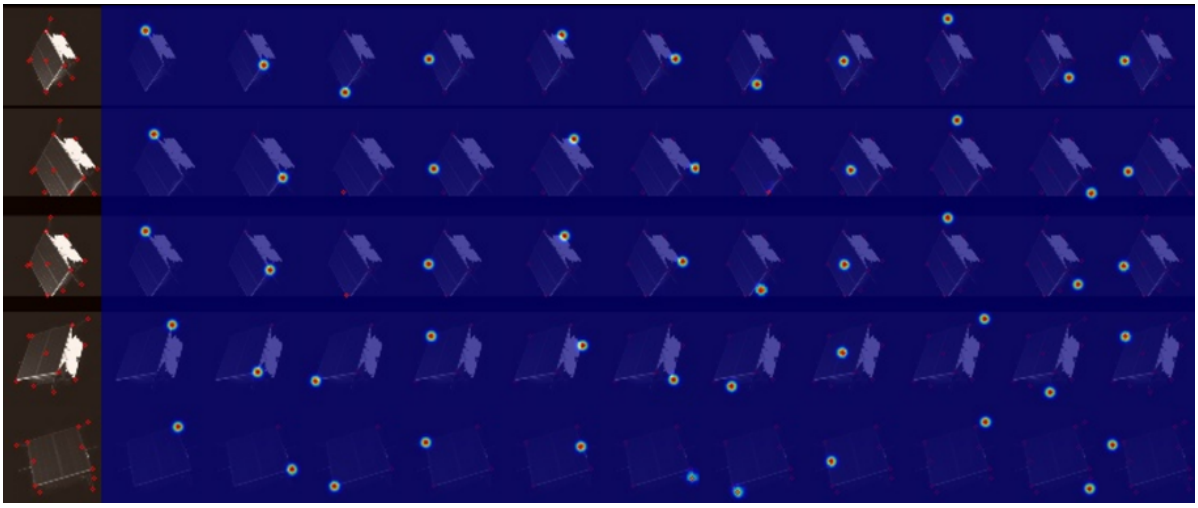


Figure 2.6: Examples of KD heatmaps (Barad, 2020).

2.1.5. Leveraged Research

Designing and developing an end-to-end relative navigation system, as discussed in the previous sections, can be a daunting challenge. The process involves crafting machine learning training and testing pipelines from the ground up and integrating various components, resulting in a high degree of complexity. The situation escalates even further when multiple ML models have to be trained and integrated within the pipeline, such as in the work by Park et al. (2019), where an OD and a KD model are used in sequence.

Most of the research mentioned in the previous paragraphs hosts code on version control platforms such as Github[2], making it freely accessible. The availability of open-source code is therefore crucial to reduce the software development time and improve existing methods. For this reason, this thesis strongly leverages both previous scientific research and available software about spacecraft relative navigation.

In particular, several theses, papers and repositories were used as a substantial part of the foundation of this thesis. Barad (2020) designed a pose estimation system for two separate targets: Tango and ENVISAT. As will be explained in Section 2.2, the same spacecraft are considered in this thesis, making the author's work extremely important for this thesis. Similarly, Van der Heijden (2022) developed a pipeline for pose estimation around the Bennu asteroid. Although concerning a different application, the system architecture can be applied to a variety of targets, such as spacecraft in orbit around the Earth, making it of high relevance. The pose estimation system developed in this thesis was inspired by both architectures, consisting of an OD and KD model for feature extraction followed by a pose reconstruction algorithm.

Another relevant source includes the work by K. Sun et al. (2019) was used as a baseline implementation of the training and testing pipelines for KD. The same repository was leveraged in later research, such as by B. Chen et al. (2019), winners of the 2019 SPEC, whose implementation of the relevant OD and KD pipelines was strongly leveraged in this thesis, and by Park et al. (2022b), who trained a model for relative pose estimation on SPEED+ leveraging heavy data augmentations. The pipeline achieved performance on testbed data comparable with the winners of the 2021 SPEC, who used the real images directly during training (Park et al., 2023). The code is fully available on Github[3] and was used for the implementation of the data augmentation pipeline as well as KD training and testing.

Given the similar application scenario, the code provided by Park et al. (2022c)[4] for training on SPEED+ was also found immensely helpful for the progression of this thesis, whereas the research performed by Deloo (2015) and Cassinis (2022) provided valuable insights into adapting the developed pipeline for the ENVISAT scenario.

[2] `https://github.com/` (visited on 07/10/2023)
[3] `https://github.com/tpark94/spnv2.git` (visited on 07/10/2023)
[4] `https://github.com/tpark94/speedplusbaseline.git` (visited on 07/10/2023)

Table 2.1: Semi-major axis a, perigee altitude h_p, and apogee altitude h_a for Tango (N2YO, 2022) and ENVISAT (ESA, 2022).

Spacecraft	a [km]	h_p [km]	h_a [km]
Tango	7086	726	787
ENVISAT	7144	780	820

2.2. Reference Scenario

To answer the main research question introduced in Section 1.3, it is required to design an end-to-end relative navigation system. This thesis, however, will only focus on developing a relative pose estimation system, which can later be integrated within a navigation loop using a state estimator. The reason for scoping the research is the lack of time to design and develop a navigation filter taking pose estimates as input measurements. It was chosen to leave this task for future work.

The general scenario used as a reference is the following. It consists of a chaser mounting a monocular camera as main relative navigation sensor, which is in close proximity to the target spacecraft. To remain conservative in the design, the target is assumed to be uncooperative, consistently with an ADR scenario. However, most of the design considerations, methodology, and findings also apply to OOS scenarios where the target is typically cooperative. The operation happens in Low Earth Orbit (LEO), with the chaser mission profile consisting of slowly approaching the target starting from an initial station-keeping point.

With regards to the target spacecraft, two different satellites are considered, namely the PRISMA-Tango and the ENVISAT spacecraft. Both targets follow polar orbits with an altitude roughly between 700 and 800 km (Table 2.1). The size and mass of the two spacecraft are widely different. ENVISAT is several meters wide (up to 25 m including the large solar panel, as shown in Figure 2.7) and has an estimated mass of above 7 tons, whereas Tango is a cube of 80×80×31 cm and weighs around 40 kg (Gill et al., 2007).

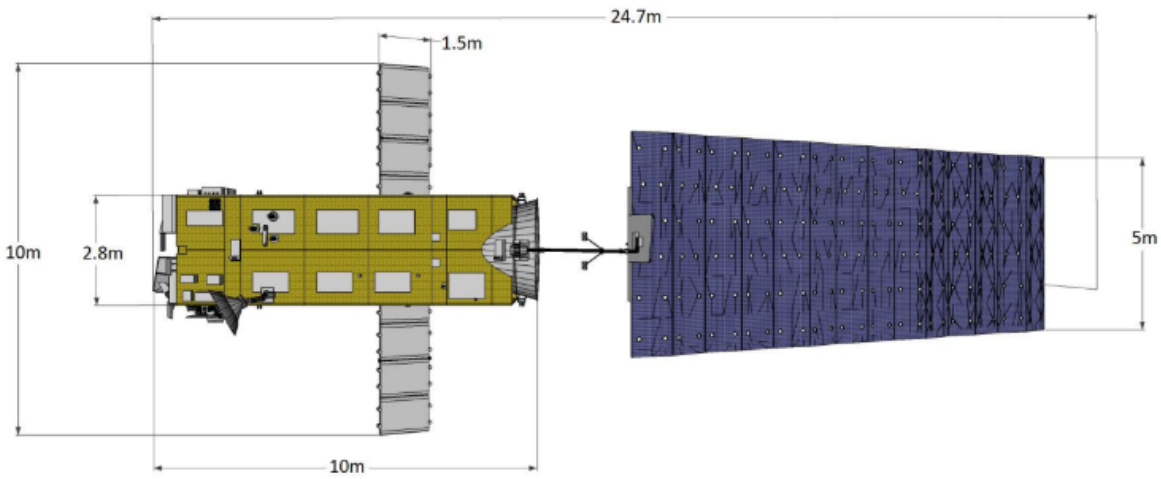
It was chosen to consider multiple targets to prove the generalisability of the methodology developed. Tango has been used to generate SPEED+, state-of-the-art dataset for spacecraft pose estimation. ENVISAT is the largest item of debris orbiting the Earth uncontrollably at the moment, therefore developing a system for this target scenario has high relevance for debris removal missions. As no existing open-access dataset could be found, a new one will be generated for the reference scenario of interest. Even if both targets have been used throughout this thesis, the corresponding pipelines are developed independently, meaning that the pipeline developed for Tango will not be tested on ENVISAT and vice-versa.

Another conservative assumption made involves the capture mechanism. It is chosen to consider the use of a robotic arm as it arguably poses the strictest accuracy requirements to the navigation system, besides being one of the optimal solutions for a target as large as ENVISAT (Telaar et al., 2017). Using a robotic arm requires accurate knowledge of both the chaser position and orientation relative to the target, whereas other designs, such as a harpoon or a net, may only require an approximate knowledge of the full relative state, as is the case for the relative attitude for a net capture. Note that this assumption can also be extended to the complementary scenario of performing OOS with the target, where the navigation system needs to be accurate enough to perform docking.

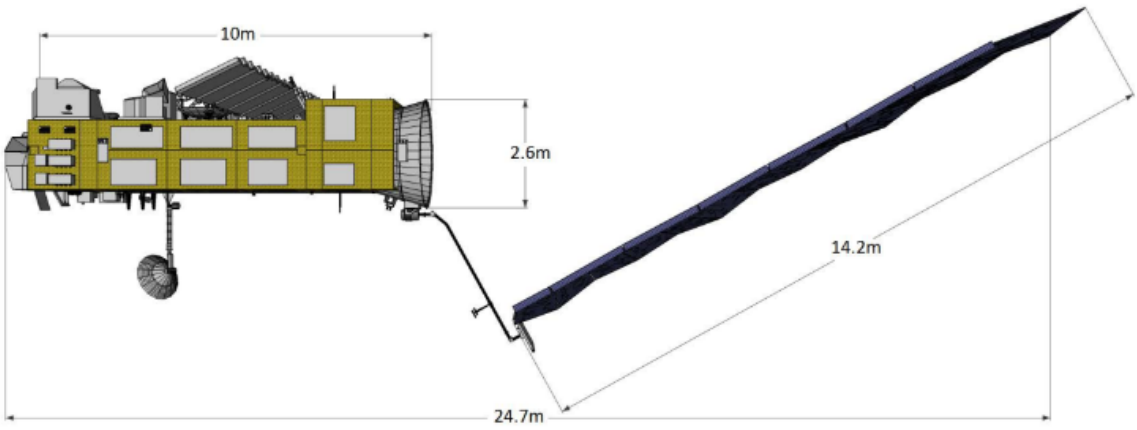
2.2.1. Assumptions and Limitations

In the context of the reference scenario just introduced, the following assumptions and limitations are made to drive the design of the relative pose estimation system:

1. **The target spacecraft is uncooperative:** Latest research has been focusing on this type of application. Several RPO missions planned for the upcoming years are focusing on uncooperative targets such as ENVISAT, making this scenario much more relevant for practical applications.
2. **The target has a known and fixed geometry:** It is assumed that there is a 3D model of the target spacecraft available and that the spacecraft itself behaves as a rigid body.
3. **The chaser is in close proximity of the target:** It is assumed that the chaser and the target are always close compared to their distance from the centre of the Earth.
4. **There is a minimum relative distance between the chaser and the target:** It is assumed that the pose estimation system will be used only when a certain separation between the two spacecraft



(a) Top view.



(b) Side view.

Figure 2.7: ENVISAT dimensions (Deloo, 2015).

exists. As a consequence of this assumption, the motion synchronisation and docking/capture phases are not considered.

5. **There is limited *a-priori* knowledge about the target state:** It is assumed that there is no knowledge about the target position and attitude relative to the chaser before the system is deployed. This implies that no information can be used to help reconstructing the relative pose.
6. **The chaser uses a robotic capture mechanism:** Assuming a robotic capture mechanism allows to account for the most conservative scenario in which accurate knowledge of both the relative distance and attitude is required.
7. **The chaser makes use of a multispectral camera as main navigation sensor:** Because of their simplicity, optical cameras are becoming increasingly popular when it comes to design a sensor suite for relative navigation.
8. **The camera intrinsic parameters are known:** The intrinsic parameters of the on-board monocular camera are known to some degree of accuracy. This allows generating images assuming certain sensor parameters. Possible uncertainties in the sensor modeling can be taken into account by applying image corruptions and performing a sensitivity analysis on corrupted data.
9. **The optical camera has square pixels and no skew factor:** This assumption allows to simplify the camera model introduced in Section 4.1 without losing generality in the reference scenario.
10. **The sensor nominal pointing direction is towards the target CoM:** Nominally, the chaser camera is assumed to point towards the Centre of Mass (CoM) of the target. Random pointing errors are

still allowed to appear in the images, however, they are assumed to be small enough to leave the target largely visible in the image.

11. **The sensor suite has only passive redundancy:** It is assumed that the pose estimation system requires only one image as input. In other words, the sensor suite contains sufficient redundancy to ensure safety of operations, but an additional sensor is switched on to collect images only when an already-active one has a failure. This allows to keep the system complexity to a minimum, *e.g.*, redundancy schemes are not considered, while being able to provide a valid system concept.
12. **Only synthetic images are used for ML model training:** It is assumed that only synthetic data can be used to train NNs. Whenever available, testbed data can only be used for validation and testing.

2.2.2. Mission and System Requirements

This section introduces the mission and system requirements. Again, they are derived from the heritage discussion in Section 2.1.

Mission Requirements

The following mission requirements were identified:

- **MR01:** The system shall be used to perform relative pose estimation around an uncooperative spacecraft with known geometry.
- **MR02:** The system shall reconstruct the pose relative to a target with unknown initial state.
- **MR03:** The system shall use a CNN-based feature extractor.
- **MR04:** The system shall be used neither when the target is too far nor in the docking phase.
- **MR05:** The relative navigation system shall be used for a target in a polar orbit with an apogee altitude between 700 and 800 km.

Each of the mission requirements above is formulated to capture part of the research scope. **MR01** identifies the main application of interest, namely to perform relative pose estimation with respect to a known, uncooperative spacecraft. **MR02** includes additional details about the considered scenario, *i.e.*, the initial state of the target is unknown. **MR03** is related to the purpose of the research of investigating the benefits of AI for spaceborne relative navigation. **MR04** specifies that the relative navigation system shall not be used when the target is too far away from the camera or for docking. Given that AI-based navigation is still in its early development, it is likely that the first technology demonstrators will test the system within an ideal distance range. Lastly, **MR05** assumes the type of target orbit for which the system should be used. This requirement is included to avoid scenarios where the target spacecraft is eclipsed for large amounts of time, making the use of a vision-based system extremely challenging.

System Requirements

The following system requirements were identified:

- **SR01:** The system shall reconstruct the target pose relative to the chaser on-board camera.
- **SR02:** The system shall only use data coming from the chaser.
- **SR03:** The system shall use an optical camera as main navigation sensor.
- **SR04:** The system shall use an OD network followed by a KD network for feature extraction.
- **SR05:** The OD and KD networks shall be trained only on synthetic images.
- **SR06:** The pose estimation system shall use a 3D model of the target for pose reconstruction.
- **SR07:** The pose estimation system shall have an operating range between 2.25 and 10 m for Tango and between 25 and 50 m for ENVISAT.
- **SR08:** The OD network shall have a mean and median Intersection-over-Union (IoU) of 80% or higher on testing data.
- **SR09:** The KD network shall have a Root Mean Squared Error (RMSE) on testing data equal to 10% of the heatmap resolution or lower after the integration with the OD network.
- **SR10:** The pose estimation system shall have a relative position error of 10% of the minimum operating distance or lower.
- **SR11:** The pose estimation system shall have an error of 5° or lower for the relative attitude.

- **SR12:** The relative navigation systems shall extract keypoint locations in less than 1.5 s.

The first two system requirements (**SR01**-**SR02**) are included to capture the application of interest: relative pose estimation around an uncooperative spacecraft. In particular, **SR01** specifies the convention used to define the relative pose, that is, the system will output the relative position and attitude of the target with respect to the chaser camera and not vice versa. **SR03** focuses on the type of sensor that the pose estimation system shall use. Despite lidar sensors being used in several RPO demonstrators (Wieser et al., 2015; Aglietti et al., 2020), recent work in AI-based relative navigation focused on the application of monocular cameras as main sensor (Sharma et al., 2018b).

System requirements **SR04** to **SR06** identify the key elements of the pose estimation pipeline. The selected pipeline architecture is taken from the work by Park et al. (2019) and Cassinis et al. (2020) and is explained in more detail in Chapter 6. These requirements are formulated to define a baseline architecture to potentially improve the current state of the art.

SR07 defines a system operational range, *i.e.*, **MR04** is its parent requirement, for both spacecraft as discussed in Section 2.2. The range chosen for Tango is the same used to generate the SPEED+ dataset, hence the system has to rely on images taken from that distance range. The range chosen for ENVISAT, instead, is selected to ensure that most of the target remains visible by the on-board camera. The specific details about the ENVISAT model and camera used can be found in Section 9.3.

Requirements **SR08** to **SR11** set the acceptable level of performance to be achieved by the system. Concerning the accuracy on the relative position (**SR10**), it was chosen to pose the requirement as a percentage error because of the different operating range of the system for the two targets, whereas the requirement on the relative attitude error (**SR11**) is directly expressed in degrees. This was done because no fundamental difference between the two scenarios was identified to force the use of a different error value. The numerical values are taken from Telaar et al. (2017) and refer to the approaching phase of the e.Deorbit mission. Note that the values used for the accuracy requirements may be quite conservative for a real mission, since e.Deorbit uses a clamping mechanism to rendezvous with ENVISAT (Wieser et al., 2015). In case other solutions are used, *i.e.*, a harpoon or a net, the requirements on the relative pose error can be expected to be relaxed. The required mean and median IoU specified in **SR08** is set after looking in the literature for reasonable OD performance, while the KD error is set using the same percentage threshold as for the relative position error.

Lastly, **SR12** poses a constraint on the maximum inference time for the OD and KD models. The selected value of 1.5 seconds was determined taking into account the work by Park et al. (2022a) and Cassinis (2022), who used a sampling rate of 1 and 2 seconds, respectively, for their navigation systems.

Part II

Theoretical Framework

3

Spacecraft Kinematics

This chapter introduces the mathematical framework used to describe the kinematics of a spacecraft. The relevant reference frames are defined in Section 3.1. Section 3.2 introduces the state variables used to parameterise the spacecraft position and attitude, whereas Section 3.3 provides the relevant frame transformations used in this work.

3.1. Reference Frames

The definition of a reference frame is required to describe the position and attitude of the chaser and target spacecraft. The fact that both are orbiting the Earth in a relatively short time allows the definition of non-inertial reference frames, however, the simplifications made in Section 2.2 enable the use of pseudo-inertial frames as well. Throughout this work, the following reference frames are used:

- **Target LVLH reference frame (LVLH):** The Local-Vertical Local-Horizontal (LVLH) reference frame is centred in the CoM of the target, while the orientation of its axes follows the convention illustrated in Figure 3.1:
  - $+X_{LVLH}$ points in the along-track direction.
  - $+Y_{LVLH}$ points in the across-track direction.
  - $+Y_{LVLH}$ points in the direction of the geocentric radius vector of the spacecraft, *i.e.*, towards the centre of the Earth.

  This frame can be used to define the position and attitude of the two spacecraft with respect to a common reference. Note that its axes perform a full rotation about $+Y_{LVLH}$ every orbit, meaning that LVLH is a non-inertial reference frame.
- **Target body-fixed reference frame (T):** Similarly to the LVLH frame, the target body-fixed reference frame T is centred in the CoM of the target spacecraft, with its axes $+X_T$, $+Y_T$, and $+Y_T$ identifying the principal spacecraft inertia axes. This frame is used to describe the target rotation. As the axes orientation changes while the target rotates, T falls into the class of non-inertial reference frames.
- **Pixel reference frame (P):** The pixel, or image, reference frame is a 2D reference frame defined for the on-board optical camera. It is introduced to associate a location with any pixel in an image. As shown in Figure 3.2, its origin is located in the top left corner of the image plane and the axes $+X_P$ and $+Y_P$ point rightward and downward, respectively. It is common to express them in pixels and, to set up a handier formulation of the pose estimation problem, associate them to a set of homogeneous coordinates (Section 4.1).
- **Camera reference frame (C):** The camera reference frame describes the position and orientation of the target with respect to the chaser on-board camera. As shown in Figure 3.2, its origin is located in the camera's centre of projection and its axes are oriented as follows:
  - $+Y_C$ is aligned with the optical axis and points towards the image plane.
  - $+Y_C$ corresponds to the $+Y_P$ axis of the pixel reference frame P.
  - $+X_C$ corresponds to the $+X_P$ axis of the pixel reference frame P.

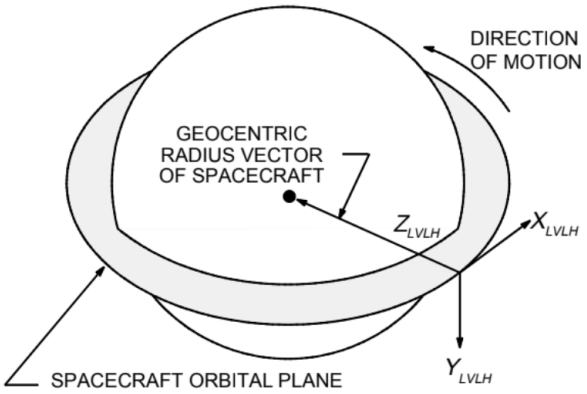


Figure 3.1: Illustration of the axes convention for a LVLH reference frame (Sauceda, 2001).

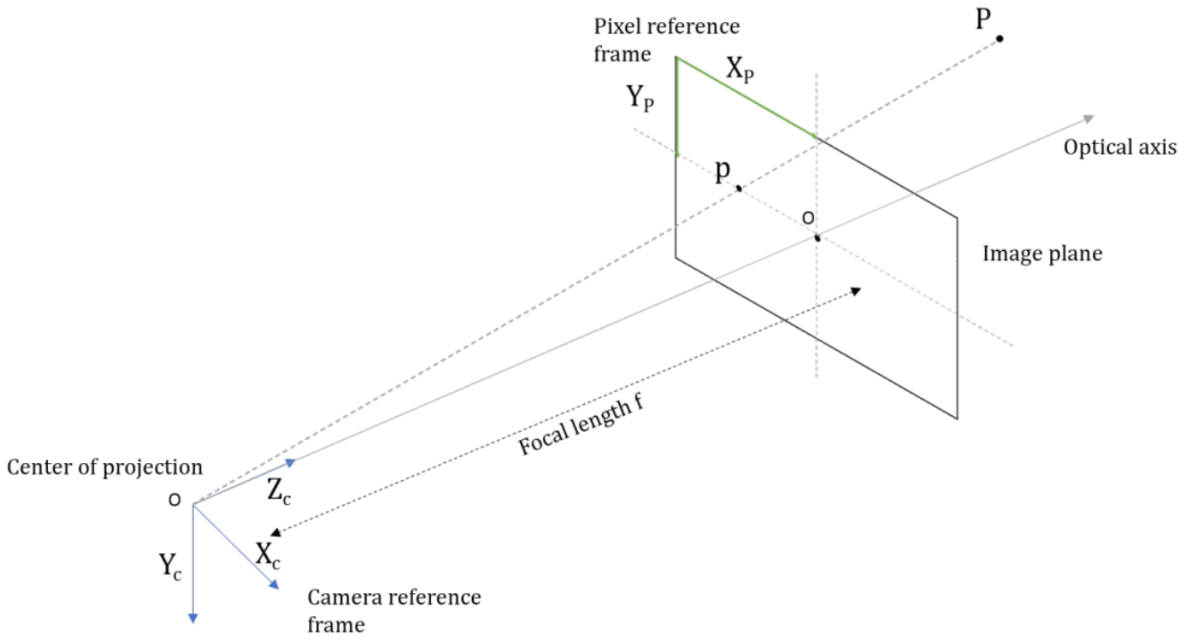


Figure 3.2: Illustration of the axes convention for the camera and pixel reference frames (Van der Heijden, 2022).

The pose estimation problem typically requires to determine the camera pose with respect to the target body-fixed frame. The C frame is therefore required to calculate the pose matrix solution to the pose estimation problem (Section 4.2).

- **World reference frame (W):** The world reference frame is the coordinate system used in Blender to locate the target, on-board camera, and light source. The default axes convention consists of having the $+Z_W$ axis pointing up, and the $+X_W$ and $+Y_W$ axes completing the right-handed set.
- **Blender camera reference frame (B):** The Blender camera reference frame is shown in Figure 3.3. It is the frame used by Blender to simulate a camera. The origin is located in the camera's centre of projection and the axes are defined as follows:
  - $+Y_B$ is aligned with the optical axis and points away from the image plane.
  - $+Y_B$ points in the upward direction of the image plane.
  - $+X_B$ completes the right-handed set of axes.

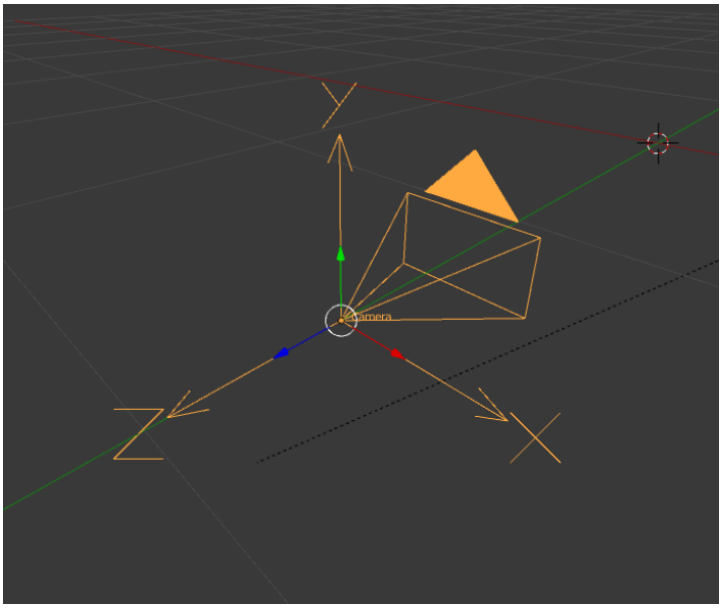


Figure 3.3: Illustration of the axes convention for the Blender camera frame.

3.2. State Variables

In the most general case, the state of a spacecraft describes both its translational and rotational motion. When one is interested in either of the two, simplified models as well as state variable formulations can be adopted. In the context of RPO missions, however, accurate knowledge of the full relative state is required to perform accurate manoeuvres. As this thesis only focuses on reconstructing the relative pose between the chaser and the target, the spacecraft state only consists of its position and attitude.

3.2.1. Coordinate Systems

There are several formulations in the literature used to parameterise the 3D position of an object in space. Besides the classical Cartesian coordinates, the Encke formulation is also used, as well as formulations derived from the planetary equations, such as Kepler elements and modified equinoctial elements, and the unified state model (Vittaldev et al., 2012). Given their lack of singularities and relatively easy interpretability, this thesis makes use of Cartesian coordinates. Given an arbitrary reference frame A, the Cartesian position of a generic point is expressed by the following:

$$\mathbf{r}_A = \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} = x_A \hat{\mathbf{x}}_A + y_A \hat{\mathbf{y}}_A + z_A \hat{\mathbf{z}}_A \tag{3.1}$$

3.2.2. Attitude Representations

Given the focus of this research on evaluating the performance of an AI-based relative pose estimation system, it is necessary to select the right attitude formulation to (i) establish a framework that enables handy computations of frame transformations, (ii) allow to interpret results intuitively, and (iii) formulate singularity-free attitude representations.

The rotation of an extended body can be parameterised in several ways. The most basic representation consists of defining a Direction Cosine Matrix (DCM) which describes the 3D rotation required to move from the initial to the target attitude. This formulation has the advantage of preserving generality, but the entries of a DCM are neither easy to determine nor intuitive. For this reason, DCM is usually expressed in terms of other attitude representations that can be determined more easily and interpreted more intuitively.

Unambiguous attitude representations of a rotating spacecraft require the use of at least three parameters. An example is given by Euler angles, which parameterise the spacecraft attitude as a rotation sequence about the three axes of a reference frame. Euler angles allow for an intuitive parameterisation of the attitude of a spacecraft, however, they present singularities every 90°. This is a

common issue for attitude representations based on solely three parameters. Because of this, alternative representations, such as Classical Rodrigues Parameter (CRP) and Modified Rodrigues Parameter (MRP), have been introduced to extend the singularity-free range: CRPs allow for a singularity for ±180°, while MRPs admit a singularity only at ±360°. The singularity could be fully removed by using the shadow parameter technique, however, this comes at the expense of a higher complexity of the attitude propagation algorithm. Representations using a larger number of parameters can also be found. The most common choice in this regard are quaternions, which rely on four parameters. The introduction of an additional parameter means that the attitude descriptors are not linearly independent. This aspect introduces redundancy in the attitude representation, *i.e.*, multiple sets of quaternions describe the same attitude in terms of Euler angles, but removes singularities completely. This is desirable in a case where the attitude cannot be limited to a certain range, which is plausibly the case for an uncooperative target as assumed in Section 2.2.1. Quaternions are therefore the desirable choice for attitude representation.

In conclusion, to respond to the needs of this work regarding the parameterisation of the attitude of a spacecraft, DCMs are chosen to formulate frame transformations, interpret results intuitively using Euler angles, and use quaternions for singularity-free attitude representations. Each of these attitude descriptors is introduced in the upcoming paragraphs.

Direction Cosine Matrix

A DCM is a transformation matrix that is used to transform a vector representation from a starting reference frame A to a new reference frame B. Given the unit-vector axes for each reference frame, $\{\hat{\mathbf{x}}_A, \hat{\mathbf{y}}_A, \hat{\mathbf{z}}_A\}$ for frame A and $\{\hat{\mathbf{x}}_B, \hat{\mathbf{y}}_B, \hat{\mathbf{z}}_B\}$ for frame B, the DCM from A to B is defined as follows (Wie, 1998):

$$\mathbf{C}_{B/A} = \begin{bmatrix} \hat{\mathbf{x}}_B \cdot \hat{\mathbf{x}}_A & \hat{\mathbf{x}}_B \cdot \hat{\mathbf{y}}_A & \hat{\mathbf{x}}_B \cdot \hat{\mathbf{z}}_A \\ \hat{\mathbf{y}}_B \cdot \hat{\mathbf{x}}_A & \hat{\mathbf{y}}_B \cdot \hat{\mathbf{y}}_A & \hat{\mathbf{y}}_B \cdot \hat{\mathbf{z}}_A \\ \hat{\mathbf{z}}_B \cdot \hat{\mathbf{x}}_A & \hat{\mathbf{z}}_B \cdot \hat{\mathbf{y}}_A & \hat{\mathbf{z}}_B \cdot \hat{\mathbf{z}}_A \end{bmatrix} = \begin{pmatrix} \hat{\mathbf{x}}_B \\ \hat{\mathbf{y}}_B \\ \hat{\mathbf{z}}_B \end{pmatrix} \cdot \begin{pmatrix} \hat{\mathbf{x}}_A & \hat{\mathbf{y}}_A & \hat{\mathbf{z}}_A \end{pmatrix} \tag{3.2}$$

By definition, DCMs have the property of being orthonormal ($\mathbf{C}_{B/A}^{-1}$=$\mathbf{C}_{B/A}^{T}$). They can be used to transform the components of a generic vector $\mathbf{v}$ from frame A to frame B as:

$$\begin{pmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{pmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \begin{pmatrix} v_{x,A} \\ v_{y,A} \\ v_{z,A} \end{pmatrix} \rightarrow \mathbf{v}_B = \mathbf{C}_{B/A}\mathbf{v}_A \tag{3.3}$$

Euler Angles

Euler angles are a well-established attitude representation method. Given a set of unit axes $\{\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}\}$, each Euler angle is defined as the rotation angle about its corresponding axis. The convention typically used consists of defining a roll (ϕ), pitch (θ), and yaw (ψ) angle corresponding to a rotation about the x, y, and z axis, respectively.

Given two reference frames A and B, the benefit of an Euler-angle attitude representation is that, whatever the orientation of frame B relative to A may be, it can be obtained as a sequence of at most three rotations, one about each unit axis. Note that the order in which the rotations are executed is important for the final mathematical expressions, and a sequence convention has to be defined beforehand. Figure 3.4 shows an illustration of an Euler-angle transformation assuming a 3→2→1 (z→y→x) rotation sequence.

Once a rotation sequence is defined, the entries of the DCM in Equation (3.2) can be expressed in terms of the Euler angles. With the 3→2→1 sequence, the following DCM is obtained:

$$\mathbf{C}_{B/A} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{3.4}$$

Quaternions

Quaternions are another common method for attitude representation. Their definition derives from Euler's eigenaxis theorem (Van der Heijden, 2022):

"A rigid body or coordinate reference frame can be brought from an arbitrary initial orientation to an arbitrary final orientation by a single rigid rotation through a principal angle Φ about the principal axis $\hat{\boldsymbol{e}}$, where the principal axis (Euler axis) is an axis that is fixed in both the initial and final orientation."

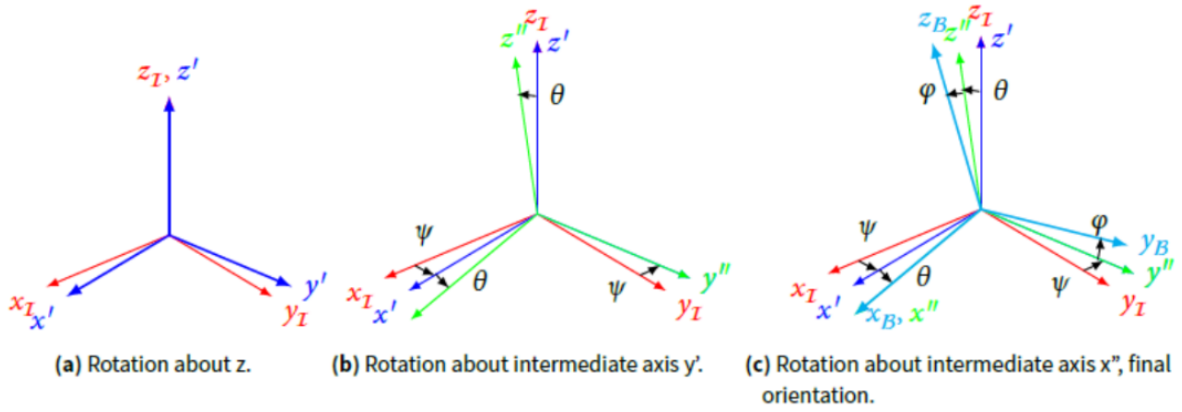


Figure 3.4: Illustration of an Euler-angle rotation sequence 3→2→1 (Gerth, 2014).

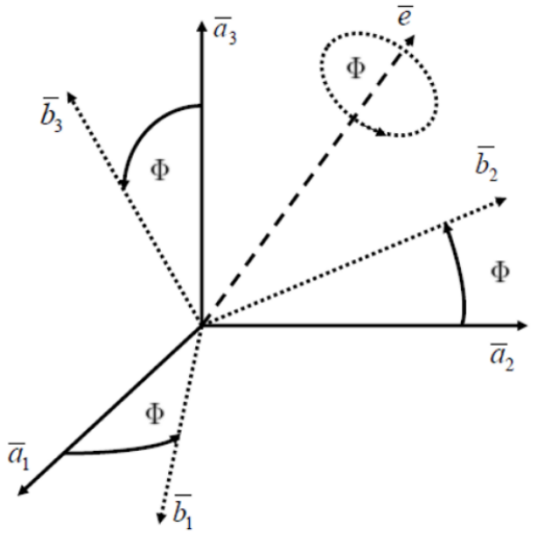


Figure 3.5: Euler eigenaxis rotation. $\bar{\mathbf{e}}$ represents the Euler principal axis and Φ represents the principal rotation angle. $\{\bar{\mathbf{a}}_1, \bar{\mathbf{a}}_2, \bar{\mathbf{a}}_3\}$ are the unit vectors of reference frame A and $\{\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2, \bar{\mathbf{b}}_3\}$ are the unit vectors of reference frame B (Van der Heijden, 2022).

A single-axis rotation can be therefore used to describe any change in the attitude of an extended body. As shown in Figure 3.5, all that is required is to define a principal axis $\hat{\mathbf{e}}$ and a principal rotation angle Φ. Quaternions can be defined as an alternative parameterisation of an Euler eigenaxis rotation. Given the Euler principal axis $\hat{\mathbf{e}}=(e_1, e_2, e_3)^T$ and principal angle Φ, a set of quaternions can be defined:

$$q_1 = e_1 \sin(\Phi/2) \tag{3.5}$$
$$q_2 = e_2 \sin(\Phi/2) \tag{3.6}$$
$$q_3 = e_3 \sin(\Phi/2) \tag{3.7}$$
$$q_4 = \cos(\Phi/2) \tag{3.8}$$

Since q_1, q_2, and q_3 are usually grouped in a vector $\mathbf{q}_v$, they are referred to as the *vector part* of the quaternion, while q_4 is usually referred to as the *scalar part* of the quaternion. It follows from Equations (3.5)-(3.8) that quaternions have a unitary L_2-norm:

$$\|\mathbf{q}\|_2 = \|(\mathbf{q}_v^T, q_4)^T\|_2 = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2} = 1 \tag{3.9}$$

where $\mathbf{q}$ is a short-hand notation to indicate the full set of quaternions. Quaternions have the advantage of describing an attitude with three Degrees of Freedom (DoF) using four parameters, which results in a

singularity-free representation (Wie, 1998):

$$\mathbf{C}_{B/A} = \begin{bmatrix} (q_4^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & (q_4^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & (q_4^2 - q_1^2 - q_2^2 + q_3^2) \end{bmatrix} \tag{3.10}$$

Inversely, given the DCM entries, quaternions can be derived using the method by Shepperd (1978). First, the maximum value from the following equations determines the first quaternion:

$$q_1 = \sqrt{\frac{1}{4}[1 + 2C_{11} - \text{tr}(\mathbf{C}_{B/A})]} \tag{3.11}$$

$$q_2 = \sqrt{\frac{1}{4}[1 + 2C_{22} - \text{tr}(\mathbf{C}_{B/A})]} \tag{3.12}$$

$$q_3 = \sqrt{\frac{1}{4}[1 + 2C_{33} - \text{tr}(\mathbf{C}_{B/A})]} \tag{3.13}$$

$$q_4 = \sqrt{\frac{1}{4}[1 + \text{tr}(\mathbf{C}_{B/A})]} \tag{3.14}$$

Second, the remaining three quaternions are calculated from the following relations:

$$q_1q_4 = (C_{23} - C_{32})/4 \tag{3.15}$$

$$q_2q_4 = (C_{31} - C_{13})/4 \tag{3.16}$$

$$q_3q_4 = (C_{12} - C_{21})/4 \tag{3.17}$$

$$q_1q_2 = (C_{12} + C_{21})/4 \tag{3.18}$$

$$q_3q_1 = (C_{31} + C_{13})/4 \tag{3.19}$$

$$q_2q_3 = (C_{23} + C_{32})/4 \tag{3.20}$$

This allows to have no singularities, but it leads to a non-unique attitude representation. For instance, from Equations (3.5)-(3.8) it can be seen how $\mathbf{q}$ and $-\mathbf{q}$ represent the same eigenaxis rotation. This can be solved by ensuring that the scalar part of the quaternion is always positive (Becker, 2017), although at the expense of a discontinuity for Φ=180°.

3.3. Frame Transformations

In some situations, it becomes preferable to express a set of equations with respect to an intermediate reference frame - where the formulation is simpler - before expressing them with respect to the final reference frame. This is where the concept of frame transformation comes into play: quantities measured in a certain reference frame can be transformed into another reference frame through a mathematical expression.

In general, a frame transformation consists of a rotational component, required when the directions identified by the axes of the two reference frames are not aligned, and a translational one, required when the origin of the two reference frames do not coincide. This section introduces the main frame transformations that will be used throughout the research.

3.3.1. Translational Transformations

When the origins of two reference frames do not coincide, a translational transformation becomes necessary to express quantities from one reference frame to the other. As shown in Figure 3.6, the fact that the origins of frame A and B are separated by a vector $\mathbf{T}$ causes the location of a point $\mathbf{P}$ with respect to frame A ($\mathbf{v}_A$) and B ($\mathbf{v}_B$) to be different. In this case, the transformation from reference frame A to B is given by the following:

$$\mathbf{v}_B = \mathbf{T} + \mathbf{C}_{B/A}\mathbf{v}_A \tag{3.21}$$

while the inverse transformation, from frame B to frame A, is:

$$\mathbf{v}_A = \mathbf{C}_{A/B}\mathbf{v}_B - \mathbf{T} \tag{3.22}$$

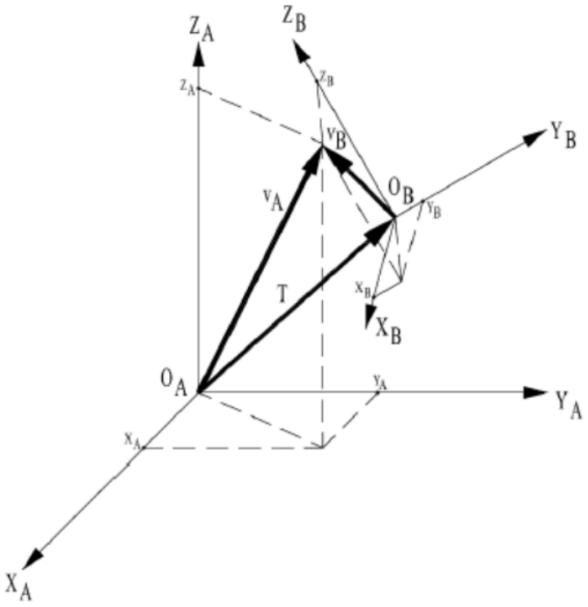


Figure 3.6: Coordinate representation of a point **P** with respect to frame A and B. In this case, the transformation from one frame to the other consists of a rotational and a translational component (Mooij, 2019).

Equations (3.21) and (3.22) are general and can be applied to any frame transformation. In particular, the vector **T** defines the component of a pose label associated with the relative position. The full relative pose of the chaser camera with respect to the target can therefore be determined together with Equation (3.38).

3.3.2. Rotational Transformations

A rotational transformation between two reference frames is required when there is a misalignment between corresponding axes. As already mentioned in Section 3.2.2, any rotation in a 3D space can be expressed as a sequence of at most three Euler-angle rotations. Therefore, the overall transformation matrix from frame A to B - denoted by $\mathbf{C}_{B/A}$ - can be expressed as a DCM representing a sequence of rotations around the unit axes. The unit-axis rotation matrices around the axes x, y, and z are the following:

$$\mathbf{C}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{3.23}$$

$$\mathbf{C}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.24}$$

$$\mathbf{C}_z(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.25}$$

In accordance with Wie (1998), it is selected to obtain the transformation matrix from frame A to B assuming a 3→2→1 rotation sequence. The full process can be visualised in Figure 3.4. Also note that each rotation is not applied with respect to the unit axes of the initial reference frame, but to the unit axes of the reference frame obtained by the previous rotation. The overall transformation matrix is given by the following:

$$\mathbf{C}_{B/A} = \mathbf{C}_x(\phi)\mathbf{C}_y(\theta)\mathbf{C}_z(\psi) \tag{3.26}$$

while the inverse transformation matrix, $\mathbf{C}_{A/B}$, follows from the orthonormality property of DCMs introduced in Section 3.2.2:

$$\mathbf{C}_{B/A}^{-1} = \mathbf{C}_{B/A}^{T} \rightarrow \mathbf{C}_{A/B} = \mathbf{C}_{B/A}^{-1} = \mathbf{C}_{B/A}^{T} \tag{3.27}$$

Finally, a vector represented with respect to frame A can be expressed with respect to frame B using Equation (3.3):

$$\mathbf{r}_B = \mathbf{C}_{B/A}\mathbf{r}_A \tag{3.28}$$

or, vice versa, a vector from frame B can be transformed to frame A:

$$\mathbf{r}_A = \mathbf{C}_{A/B}\mathbf{r}_B = \mathbf{C}_{B/A}^T\mathbf{r}_B \tag{3.29}$$

Relevant Transformation Matrices

The transformation matrices corresponding to the following frame transformations are used in this work:

- **World reference frame (W) to Target LVLH reference frame (LVLH):** This transformation matrix allows to move from the main reference frame in Blender to the orbit of the target spacecraft. For simplicity in the setup of the data generation pipeline, it is assumed that these two reference frames are co-rotating, meaning that the two sets of axes are always aligned:

$$\mathbf{C}_{LVLH/W} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{I}_{3\times 3} \tag{3.30}$$

- **World reference frame (W) to Target body-fixed reference frame (T):** This rotation matrix is used to rotate the target spacecraft with respect to Blender's main reference frame, which is a step required for the generation of pose labels. Since it was assumed that the W frame coincides with the LVLH frame, this rotation matrix describes the attitude of the target with respect to the LVLH frame. Therefore, the same DCM as in Equation (3.10) can be used:

$$\mathbf{C}_{T/W} = \begin{bmatrix} (q_4^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & (q_4^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & (q_4^2 - q_1^2 - q_2^2 + q_3^2) \end{bmatrix} \tag{3.31}$$

  where in this case the set of quaternions $\mathbf{q}$ parameterises the orientation of the target with respect to Blender's W frame.
- **World reference frame (W) to Blender camera reference frame (B):** To increase the variability of the synthetically generated data, it is also necessary to rotate the camera in Blender with respect to the main reference frame. Given a camera orientation in Blender parameterised by a set of quaternions $\mathbf{q}$ with respect to the world frame W, the following DCM can be used as transformation matrix:

$$\mathbf{C}_{B/W} = \begin{bmatrix} (q_4^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & (q_4^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & (q_4^2 - q_1^2 - q_2^2 + q_3^2) \end{bmatrix} \tag{3.32}$$

  which is formally identical to the transformation matrix in Equation (3.31).
- **Blender camera reference frame (B) to Camera reference frame (C):** A necessary step in the label generation pipeline is the frame conversion from the camera frame in Blender B to the on-board camera frame C. The relative orientation between these two frames is fixed and given by:

$$\mathbf{C}_{C/B} = \mathbf{C}_{x,B}(\pi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\pi & \sin\pi \\ 0 & -\sin\pi & \cos\pi \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{3.33}$$

  which can also be expressed by the quaternion:

$$\mathbf{q} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.34}$$

- **Camera reference frame (C) to Target body-fixed reference frame (T):** This transformation corresponds to the relative attitude between the chaser camera and the target. This transformation is therefore part of the relative pose label and can be expressed in terms of intermediate rotations C→B→W→T:

$$\mathbf{C}_{T/C} = \mathbf{C}_{T/W}\mathbf{C}_{W/B}\mathbf{C}_{B/C} \tag{3.35}$$

The first matrix in the right-hand part of the above equation represents the orientation of the target with respect to Blender's world reference frame, the second matrix represents the orientation of Blender's world reference frame relative to the Blender camera frame, and the third matrix represents the transformation from the on-board to the Blender camera frame. The first matrix is already known from previous transformations, while the second and the third ones require inverting transformation matrices previously introduced:

$$\mathbf{C}_{W/B} = \mathbf{C}_{B/W}^{-1} \tag{3.36}$$

$$\mathbf{C}_{B/C} = \mathbf{C}_{C/B}^{-1} \tag{3.37}$$

where $\mathbf{C}_{B/W}$ and $\mathbf{C}_{C/B}$ are the transformations described by Equations (3.32) and (3.33), respectively. Since pose estimation pipelines often use a quaternion-based attitude representation (Sharma et al., 2018a; Sharma et al., 2019), it becomes handy to derive the quaternion set $\mathbf{q}$ parameterising the full transformation from T to C. By parameterising the intermediate frame transformations with the following sets of quaternions:

$$\begin{aligned} C \rightarrow B: &\quad \mathbf{q}' \\ B \rightarrow W: &\quad \mathbf{q}'' \\ W \rightarrow T: &\quad \mathbf{q}''' \end{aligned}$$

and by observing that:

$$\begin{aligned} B \rightarrow C: &\quad \mathbf{q}'^{-1} \\ W \rightarrow B: &\quad \mathbf{q}''^{-1} \end{aligned}$$

the set of quaternions $\mathbf{q}$ parameterising the full rotation can be calculated as:

$$\mathbf{q} = \mathbf{q}''' \otimes (\mathbf{q}' \otimes \mathbf{q}'')^{-1} \tag{3.38}$$

where $\otimes$ represents the quaternion-product operation and $\mathbf{q}'''$ is fixed and given by Equation (3.34). The expression above allows associating an attitude label $\mathbf{q}$ to each image generated in Blender, when the camera and target orientation settings $\mathbf{q}'$ and $\mathbf{q}''$, respectively, are known.

- **Camera reference frame (S) to Pixel reference frame (P):** This transformation maps points in a 3D space into 2D points of an image taken by the on-board camera. This is done thanks to the camera intrinsic parameter matrix $\mathbf{K}_C$:

$$\mathbf{K}_C = \begin{bmatrix} f' & 0 & c_x & 0 \\ 0 & f' & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.39}$$

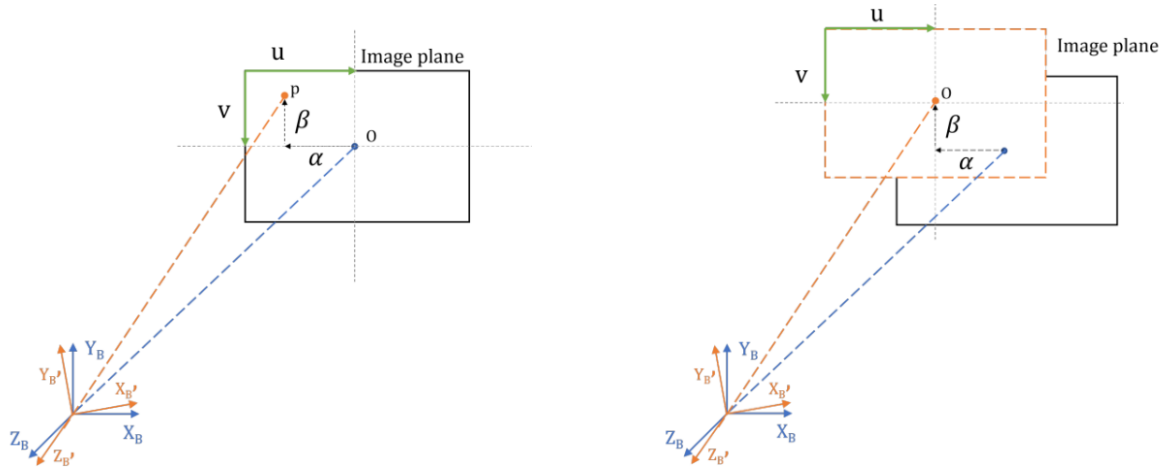
on which more details will be provided in Section 4.1.

Pointing Errors

In addition to the main frame transformation matrices, it is necessary to model pointing errors, always existing in space imaging systems and assumed to be present in Section 2.2.1. Every image in the SPEED+ datasets represents the Tango spacecraft not perfectly centred and the images of the new ENVISAT dataset will be generated with off-nominal pointing directions.

An illustration of the strategy adopted to model pointing errors is provided in Figure 3.7. When no pointing error is applied, point O is precisely located in the centre of the image with respect to frame B. By assuming the pointing error to consist of small rotations within the image plane, the resulting frame B' will see a new point, p, occupying the centre of the image (Figure 3.7a). The angles describing the pointing errors, α and β, can be used to describe the frame transformation from B to B':

$$\mathbf{C}_{B'/B} = \mathbf{C}_x(\beta)\mathbf{C}_y(\alpha) \tag{3.40}$$



(a) When a pointing error is applied, the optical axis identifies a new image centre p when intersecting the image plane.

(b) From the perspective of the new camera frame, point O is shifted to the bottom-right corner of the image.

Figure 3.7: Illustration of the effect of applying a pointing error to the nominal pointing direction (Van der Heijden, 2022).

Applied to the chaser camera, the equation above allows to find the off-nominal pointing direction, *i.e.*, the camera attitude with the pointing error applied. Note that, for the convention used, when α and β are both positive, the original image centre O is shifted to the bottom-right corner (Figure 3.7b).

4

Pose Estimation

This chapter sets up the relative pose estimation framework used in this work. The findings from the literature study summarised in Chapter 2 outlined the potential of an architecture using a CNN-based feature extractor followed by a pose solver to reconstruct the relative pose. This chapter discusses in more detail the considered monocular pose estimation pipeline. First, monocular cameras are introduced along with the pinhole camera model in Section 4.1. Second, the mathematical formulation of the PnP problem is provided in Section 4.2 and, third, different pose-solving algorithms are discussed in Section 4.3.

4.1. Monocular Cameras

As introduced in Chapter 1, monocular cameras have been the object of recent research in spacecraft relative navigation because of their simplicity. Cassinis et al. (2019), for instance, state how a system relying on a single monocular camera is flexible enough to provide an accurate navigation solution over the full distance range of an AR&D phase.

This thesis assumes the use of a multispectral camera as main navigation sensor, *i.e.*, the camera collects photons over a discrete set of bands. Sensitivity to a certain wavelength is achieved by using appropriate filters in the camera detector. The most common type of detector is sensitive to the Red-Green-Blue (RGB) bands and is the simplest detector from which a coloured image can be obtained. A common alternative to RGB cameras is greyscale cameras, which provide data in the form of monochromatic images. When generating synthetic data, it is a common operation in Computer Vision (CV) to convert RGB images collected with a first camera to grayscale to simulate a second camera. This can be achieved by performing the weighted sum of the values in the red, green, and blue bands of a pixel (Recommendation ITU-R BT.601-5, 2011):

$$p_{uv} = 0.299 r_{uv} + 0.587 g_{uv} + 0.114 b_{uv} \tag{4.1}$$

where p_{ij} is the greyscale pixel in the (u, v) location according to the image reference frame P, while r, g, and b represent the RGB pixel value of the red, green, and blue band, respectively.

In addition to a different data format, optical cameras are affected by a variety of corruptions and perturbations. Hendrycks et al. (2019) provide a list of common corruptions observed in images collected by optical cameras. The list includes detector noise, image blur, and illumination conditions and was leveraged by Van der Heijden (2022) to generate an augmented synthetic dataset of the Bennu asteroid. Similar corruptions will be introduced in Section 9.2.1 as well to reproduce the quality of raw satellite images more realistically.

4.1.1. Pinhole Camera Model

Because of its simplicity and adoption in other relative navigation pipelines (Sharma et al., 2018a; Cassinis et al., 2020; Van der Heijden, 2022), the pinhole camera model is chosen to model the chaser's on-board optical camera. The model was proposed by Sturm (2014) and it allows to map a 3D point in space to a 2D point in the image. It is commonly used within the field of computer vision and by several

rendering software, such as Blender. The following parameters have to be defined by the pinhole model (Van der Heijden, 2022):

- **Focal length**: is the distance between the centre of projection and the image plane.
- **Optical axis**: is the line passing through the centre of projection and that is orthogonal to the image plane.
- **Principal point**: is the intersection of the optical axis and the image plane.

The already-introduced Figure 3.2 provides an illustration of the pinhole camera model and relevant definitions. What the pinhole camera model does is mapping the 3D point **P** into the 2D point **p** contained in the image plane. Thanks to the focal length f and the simplified geometry of the model, the coordinates of point **P** in the camera frame $(x_C, y_C, z_C)^T$ can be transformed in coordinates in the pixel frame P:

$$\mathbf{p} = (x_P, y_P)^T = \left(f\frac{x_C}{z_C}, f\frac{y_C}{z_C} \right)^T \tag{4.2}$$

It can be seen from the equation above how point **p** is the projection of multiple points. Each point belonging to the OP line, indeed, has **p** as its projection in the image plane.

A common convention used in the pinhole camera model is to express the coordinates of point **p** in pixels. To do so, it is necessary to apply a scaling factor s to the focal length. As it depends on the pixel size, in the case of a square pixel shape s is the same along both image axes. The normalised focal length $f' = sf$ is therefore introduced to define the pixel coordinates $(u, v)^T$ of point **p** in the pixel reference frame:

$$\mathbf{p} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} \frac{f'}{z_C} & 0 & 0 \\ 0 & \frac{f'}{z_C} & 0 \end{bmatrix} \begin{pmatrix} x_C \\ y_C \\ z_C \end{pmatrix} \tag{4.3}$$

In addition, homogeneous coordinates are often used to describe the location of a point in the image plane. The advantage of this representation is that the mathematics behind 3D visual reconstruction problems is more easily manageable. Given the pixel and camera frame coordinates, their corresponding homogeneous coordinates are defined as $(u, v, 1)^T$ and $(x_C, y_C, z_C, 1)^T$, respectively. The homogeneous pixel coordinates of point **p** can be written as:

$$\mathbf{p} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} f' & \alpha & 0 & 0 \\ 0 & f' & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x_C \\ y_C \\ z_C \\ 1 \end{pmatrix} \tag{4.4}$$

where α is a skewing factor assumed to be zero in Section 2.2. Also note that the homogeneous coordinates have been further scaled by a constant, namely the feature depth, in the passage from Equation (4.3) to (4.4). The last transformation step from the camera to the pixel frame consists of taking into account the different origin of the pixel frame compared to the camera frame. As introduced in Section 3.1, frame P has its origin in the top left corner of the image, while the origin of frame C is located on the optical axis. This means that the coordinates of the principal point $(c_x, c_y)^T$ have to be taken into account when computing the pixel coordinates:

$$\mathbf{p} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} f' & 0 & c_x & 0 \\ 0 & f' & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x_C \\ y_C \\ z_C \\ 1 \end{pmatrix} \tag{4.5}$$

The resulting transformation matrix is usually denoted with $\mathbf{K}_C$ and is the camera intrinsic parameter matrix. Besides describing a coordinate transformation from the camera to the image plane, $\mathbf{K}_C$ is also used to solve the Perspective-n-Points (PnP) problem, which will be introduced in the next section.

4.2. Perspective-n-Points Problem

Performing pose estimation consists of determining the relative position and attitude between the chaser and the target spacecraft. To achieve this, one of the possible strategies involves splitting the

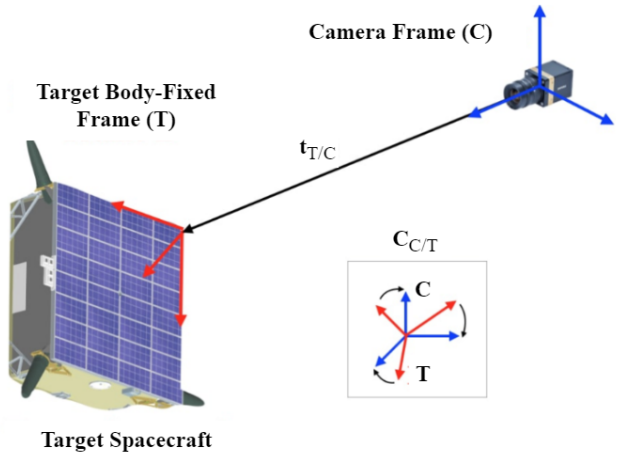


Figure 4.1: Visualisation of the PnP problem. The figure was adapted from Sharma et al. (2019).

problem in two sub-tasks: the first one concerns the extraction of keypoint locations from the input image, while the second one consists of determining the pose. The pose estimation pipeline considered in this research is functionally similar to the one proposed in other theses Barad (2020) and Van der Heijden (2022): a CNN-based feature extractor passes the keypoint coordinates to a pose solver, which uses the provided locations to reconstruct the relative pose. This section only covers the formulation and solution of the PnP problem, as the part of the pipeline involving keypoint extraction strongly relies on ML, whose concepts of interest will be introduced in Chapter 5.

A solution to the PnP problem provides the location of the principal point and orientation of the chaser on-board camera relative to the target body frame starting from a set of keypoint equations. This is usually done using a 3D model of the target, as recently done by Sharma et al. (2019), Park et al. (2019), and Cassinis et al. (2020) for spacecraft pose estimation.

Let $\mathbf{r}_{k,T} = (x_{k,T}, y_{k,T}, z_{k,T})^T$ be the location of the k-th keypoint represented in the target 3D model in the target body-fixed reference frame (T). Given the number of keypoints N, we have $i = 1, 2, ..., N$. Let $\mathbf{p}_k = (u_k, v_k, 1)^T$ be the homogeneous pixel coordinates of feature k in the pixel reference frame. The unknowns of the problem, shown in Figure 4.1, are the camera relative position and orientation, parameterised through a vector $\mathbf{t}_{T/C} \in \mathbb{R}^{3\times1}$ and a transformation matrix $\mathbf{C}_{C/T} \in \mathbb{R}^{3\times3}$, respectively. The former is the translation vector from the camera to the target frame, while the latter transforms the camera frame into the target frame. The relative position and orientation are related to the pixel coordinates of a detected feature through the position of the feature in the camera frame $\mathbf{r}_{k,C}$, leading to the following equations:

$$\mathbf{r}_{k,C} = \begin{pmatrix} x_{k,C} \\ y_{k,C} \\ z_{k,C} \end{pmatrix} = \mathbf{C}_{C/T}\mathbf{r}_{k,T} + \mathbf{t}_{T/C} \tag{4.6}$$

$$\mathbf{p}_k = \begin{pmatrix} u_k \\ v_k \\ 1 \end{pmatrix} = \mathbf{K}_C \begin{pmatrix} x_{k,C} \\ y_{k,C} \\ z_{k,C} \\ 1 \end{pmatrix} \tag{4.7}$$

where $\mathbf{K}_C$ is the camera intrinsic parameter matrix introduced in Equation (4.5). A more compact version of Equations (4.6) and (4.7) can be obtained by introducing the pose matrix $\mathbf{P}_C \in \mathbb{R}^{3\times4}$:

$$\begin{pmatrix} u_k \\ v_k \\ 1 \end{pmatrix} = \mathbf{K}_C\mathbf{P}_C \begin{pmatrix} x_{k,T} \\ y_{k,T} \\ z_{k,T} \\ 1 \end{pmatrix} \tag{4.8}$$

with

$$\mathbf{P}_C = \left[\mathbf{C}_{C/T}|\mathbf{t}_{T/C}\right] = \left[\mathbf{C}_{T/C}^{-1}|\mathbf{t}_{T/C}\right] \tag{4.9}$$

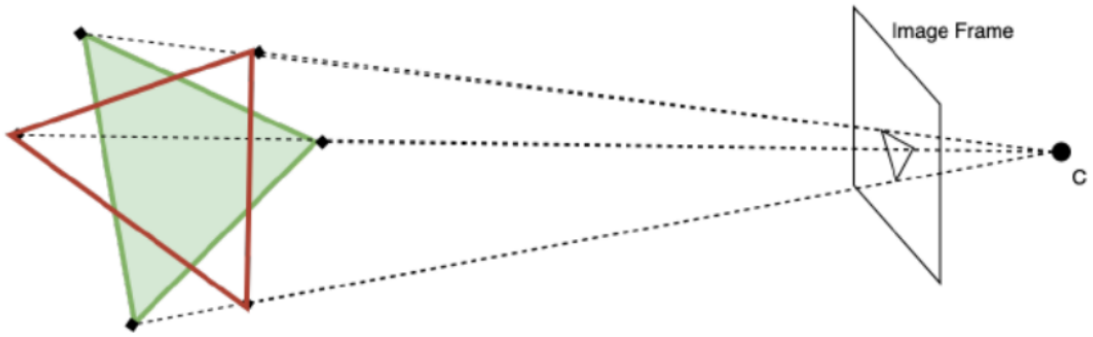


Figure 4.2: Illustration of the ambiguity in the solution of the PnP problem when only three keypoints are detected (Barad, 2020).

Equation (4.9) contains all the six unknowns of the PnP problem: the three attitude descriptors required to define the transformation matrix $\mathbf{C}_{T/C}$ and the three components of the translation vector $\mathbf{t}_{T/C}$. Together, they define the relative pose label. The coordinates of each detected keypoint in Equation (4.8) result in two scalar equations. Therefore, a minimum of $N = 3$ keypoints have to be detected. However, in this case as many as eight solutions are found. An intuitive explanation for the origin of this ambiguity can be given by looking at Figure 4.2. In the figure, the keypoints of the rigid triangle can be oriented in multiple ways so that each keypoint lies on the same projection line. By remaining on the same projection line, their coordinates in the image frame do not change. The number of solutions can be reduced by progressively adding keypoints, until a unique solution is found for $N = 6$. As Equation (4.8) suggests, however, the accuracy of the solution strongly depends on the accuracy of the predicted keypoint location (u_k, v_k) in the image. Therefore, in practice, a number of keypoints $N \geq 6$ is always used to compensate for the non-perfect accuracy of the keypoint detector.

4.3. Pose Solvers

Over the years, several methods for solving the PnP problem have been proposed. The existing literature can be distinguished in two main approaches: *iterative* and *non-iterative*. Iterative pose solvers determine the target pose by solving the PnP equations (4.6)-(4.7) in an iterative manner. The method has the advantage of achieving greater accuracy compared to its counterpart, but convergence is not always guaranteed. The reader is referred to the work by DeMenthon et al. (1995) and Oberkampf et al. (1996) for some examples of iterative pose-solving algorithms. Non-iterative pose solvers, on the other hand, improve the convergence speed of the algorithm by solving a linearised version of the PnP equations. This brings the problem to have a unique solution, although this is achieved at the expense of accuracy.

One of the most common non-iterative pose solvers is the Efficient PnP (EPnP) solver (Lepetit et al., 2009), which was also used by Park et al. (2019) and Van der Heijden (2022) in their spaceborne pose estimation pipelines. Despite the existing heritage, EPnP may not be the universally best choice for relative pose reconstruction. Sharma et al. (2018b), for instance, use SVD as pose initialization technique followed by a Newton-Raphson algorithm for pose refinement to improve upon computational efficiency. Similarly, Cassinis (2022) and Barad (2020) have found that taking into account the feature detection uncertainty using the CEPPnP solver works better for their pipelines. Even within the same domain of relative pose estimation in space, authors tend to adopt different techniques when it comes to pose reconstruction. This work follows up on this trend by not selecting a pose reconstruction algorithm *a-priori*, but rather empirically at the testing stage.

To compare the performance of different pose solvers efficiently and with minimal re-implementation effort, it was chosen to use the `solvePnP` functionality available in OpenCV[1]. The library not only implements an extensive Application Programming Interface (API) for pose estimation but also supports numerous pose solvers. Out of all the available methods, the following were deemed as viable options:

- **Iterative Levenberg-Marquardt optimisation:** Iterative method based on a Levenberg-Marquardt optimisation (Madsen et al., 2004), *i.e.*, a modified algorithm to solve the least-square problem

[1] `https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html` (visited on 29/09/2023)

by including a dumping coefficient. In this case, the function finds the pose minimising the reprojection error, *i.e.*, the sum of squared distances between the extracted keypoint coordinates and the projections of points from a 3D model. More details about how to reconstruct a 3D model for both spacecraft are provided in Section 9.1. The algorithm requires a minimum of 6 points for initialisation.

- **AP3P:** Gao et al. (2003) proposed an efficient Algebraic solution to the Perspective-3-Point problem (AP3P). The algorithm leverages the geometric constraints in place for the perspective problem to determine a set of trigonometric equations. The equations are then solved for the relative rotation matrix and subsequently for the camera position. The method is purely algebraic and requires exactly 3 points for initialisation.
- **EPnP:** As already mentioned, this algorithm was introduced by Lepetit et al. (2009). The central idea is to express the 3D point coordinates as a weighted sum of four virtual control points, allowing to reduce the algorithm complexity from exponential to linear without any loss in accuracy. A minimum of 4 points is required for initialisation.
- **SQPnP:** The Sequential Quadratic PnP (SQPnP) method, introduced by Terzakis et al. (2020), formulates the problem as a non-linear quadratic program, managing to consistently find the pose solution corresponding to the global minimum, regardless of possible coplanar keypoint arrangements. Because of its robustness, the method is able to find a solution with only 3 points.

Furthermore, similarly to the work by Sharma et al. (2018b), it was chosen to consider the inclusion of a pose refinement step. In case it is observed to yield an accuracy improvement, the refinement algorithm will be:

- Levenberg-Marquardt minimisation, previously introduced.
- A Virtual Visual Servoing (VVS) optimisation (Marchand et al., 2015). This method is commonly used for robotic controllers (Chaumette et al., 2006). However, the optimisation scheme can be adapted to other problems, such as pose refinement, by generalising the idea that control actions can be used to optimise the relative pose just like they bring the robotic arm to the target state. The algorithm requires at least 3 points.

A more extensive overview of the software used to implement an end-to-end pose estimation pipeline is provided in Section 6.2. As previously mentioned, the different PnP methods will be compared on the extracted keypoint locations to determine the optimal algorithm for the pipeline developed in this thesis.

4.3.1. Keypoint Confidence Ranking

Despite the pose solver simply requiring a list of keypoint coordinates, it is important to properly characterise the KD confidence about the predicted location. The reason for this is that the difficulty in detecting keypoints from an optical image changes depending on a variety of factors. A keypoint may be occluded by other parts of the spacecraft, badly illuminated, or not visible at all. As a consequence, the keypoint locations alone do not carry any information about the network detection confidence, and it is necessary to rank each keypoint detection according to a predefined criterion.

This problem has already been tackled in multiple spacecraft pose estimation pipelines. Cassinis (2022) uses the heatmap shape and the intensity around the peak to characterize the detection confidence, whereas Van der Heijden (2022) directly uses the heatmap peak to rank keypoints. Both approaches are based on the observation that NNs tend to generate outputs with higher values the more confident they are about a certain prediction. For instance, OD models always output a probability score for the detected object to belong to a certain class. Consistently with this finding, this thesis uses a keypoint ranking method based on the heatmap softmax. An exponential map is applied to each pixel of the output heatmap according to the following:

$$\mathbf{S}_k(u,v) = \frac{\exp\left\{\beta[\mathbf{H}_k(u,v) - \mathbf{H}_k(u_{max}, v_{max})]\right\}}{\sum_{i=1}^{w}\sum_{j=1}^{h}\exp\left\{\beta[\mathbf{H}_k(i,j) - \mathbf{H}_k(u_{max}, v_{max})]\right\} + \epsilon} \tag{4.10}$$

where k indicates the keypoint index, (u, v) are the heatmap pixel coordinates, with the heatmap $\mathbf{H}$ having a resolution $w \times h$. (u_{max}, v_{max}) are the heatmap peak coordinates and the coefficients β and ϵ are added to ensure numerical stability. They are set to 10 and 10^{-12}, respectively, as these values were found

to guarantee stability during training. Note that, while β must be greater than 1 to remove numerical issues, ϵ is included to prevent dividing by zero, meaning it should be closer to the machine-level precision to avoid perturbing the result of Equation (4.10) excessively. This is the reason behind the difference of several orders of magnitudes between the two constants.

The maximum value of the resulting heatmap is then extracted and used as a confidence score for the prediction of the k-th keypoint:

$$S_{k,max} = \max_{\{u\}_1^w \{v\}_1^h} \mathbf{S}_k(u, v) \tag{4.11}$$

4.3.2. Pose Score

Once the relative pose is reconstructed using the extracted keypoint locations, the predicted pose is compared against the ground truth pose using the following metric:

$$E_P = \frac{E_T}{\|\mathbf{t}_{T/C}\|} + E_R \tag{4.12}$$

where E_P represents the *pose score* and $\mathbf{t}_{T/C}$ is the relative position label. E_T and E_R are the translation and rotation errors, respectively. They are defined as follows (Park et al., 2022b):

$$E_T = \|\hat{\mathbf{t}}_{T/C} - \mathbf{t}_{T/C}\| \tag{4.13}$$

$$E_R = \arccos\left[\frac{\mathrm{tr}(\mathbf{C}_{T/C}^T \hat{\mathbf{C}}_{T/C}) - 1}{2}\right] \tag{4.14}$$

where $\hat{\mathbf{t}}_{T/C}$ indicates the reconstructed relative position, and $\hat{\mathbf{C}}_{T/C}$ and $\mathbf{C}_{T/C}$ the reconstructed and ground truth relative attitude, respectively, parameterised as rotation matrices. The interpretation of the two terms is relatively simple. The translation term quantifies the distance between the predicted and true position vector, expressed in meters, whereas the attitude term represents a measure of the difference between the predicted and true rotation matrix. In fact, the larger the attitude error the larger the trace of the matrix resulting from the product $\mathbf{C}_{T/C}^T \hat{\mathbf{C}}_{T/C}$ is going to be. Equation (4.12) was introduced by Park et al. (2022c) and was used as main evaluation criterion for the 2021 SPEC.

Since the metric is also used to quantify the pose error over the testbed images generated by TRON, the metric in Equation (4.12), it is common to adjust the error taking into account the facility's calibration parameters:

$$E_T = \begin{cases} E_T, \frac{E_T}{\|\mathbf{t}_{T/C}\|} < 2.173 \cdot 10^{-3} \\ 0, \text{otherwise} \end{cases} \tag{4.15}$$

$$E_R = \begin{cases} E_R, E_R < 0.169^\circ \\ 0, \text{otherwise} \end{cases} \tag{4.16}$$

5

Machine Learning

This chapter introduces the machine learning concepts used in this research. Section 5.1 provides an introduction to deep learning, putting emphasis on the benefits of using CNNs for tasks such as object and keypoint detection, while the relevant ML concepts are introduced in Section 5.2 to select the training configuration maximising the performance of the final ML model.

5.1. Deep Learning

Recent years have seen the spread of neural networks in several fields. The reason for the increasing interest from the scientific community is their capability of handling large amounts of data and extracting meaningful patterns from the training set. Thanks to this feature, NNs have outperformed previous methods in tasks involving large datasets, such as object classification.

In particular, it was observed how networks consisting of multiple layers are able to learn more complex patterns and hence solve more complex problems. Such a finding led to the success of deep learning and Deep Neural Networks (DNNs), namely architectures including one or more hidden layers. The flexibility of deep networks lies in their property of being universal approximators (Cybenko, 1989): if a sufficient number of hidden neurons is used, a network can approximate any continuous non-linear function. This enabled deep networks to set the current state of the art in many pattern-recognition tasks.

Depending on the structure of the training data, DL can be separated into supervised, unsupervised, and reinforcement learning. Each class requires a different problem formulation and is based on different learning rules. The first class, supervised learning, consists of a problem where a NN is trained on a set of labeled data. The second class, unsupervised learning, requires the network to learn the labels by itself, therefore removing the need for a human operator. The third class, reinforcement learning, addresses the problem by defining a reward function the network is trained to maximise. Because their formulation results in a more complex problem, both the second and third classes generally reduce the accuracy of the final model. It was chosen to focus this work only on supervised learning, given the greater reachable accuracy by a model when trained in a supervised fashion and the availability of labeled data for spacecraft relative pose estimation.

A generic supervised-learning problem can be formulated mathematically as $\mathbf{y} = f(\mathbf{x})$, where $\mathbf{x}$ is the input, $\mathbf{y}$ the output, and f the function mapping the input to the corresponding output. Intuitively, it is clear that the better the network learns such a function, the more accurately it will perform the task. Depending on the type of the output $\mathbf{y}$, a task can be either a classification or a regression task. This has an impact on the network architecture itself: a classification network predicts an output from a discrete set, such as recognising a cat or a dog in an image, while a regression network outputs a number from a continuous set, such as predicting future stock prices. Pose estimation pipelines based on either classification or regression can be found in literature, without a significant difference in terms of performance.

As discussed in Section 2.1, the recent boost in researching AI-based solutions to spacecraft relative navigation has outlined the superiority of DL, specifically convolutional networks, over other methods. After taking an image as input, each layer in a CNN is responsible for learning relevant features helping

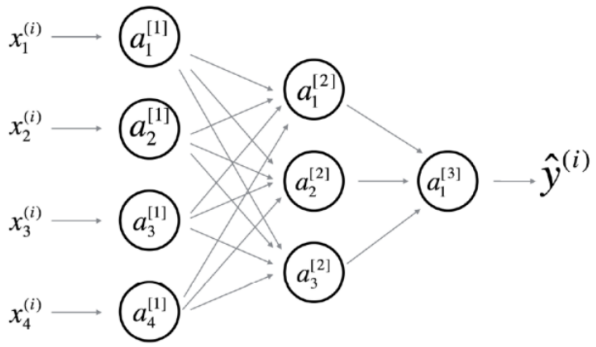


Figure 5.1: Illustration of a 3-layer, fully-connected architecture (Van der Heijden, 2022).

the model making the right prediction downstream. The deeper the layer, the more abstract the features it learns. Before diving into the building blocks of a CNN architecture, however, it is necessary to introduce other fundamental algorithms related to how NNs make predictions and learn, namely forward propagation and backpropagation.

5.1.1. Neural Networks

A neural network is a universal approximator able to reproduce the behaviour of a non-linear function $f : \mathbb{R}^{n\times 1} \rightarrow \mathbb{R}^{m\times 1}$ that maps an input $\mathbf{x} \in \mathbb{R}^{n\times 1}$ to an output $\mathbf{y} \in \mathbb{R}^{m\times 1}$. Resembling the structure of a human brain, the elementary building block of a neural network is referred to as *neuron*. In principle, NNs can contain an arbitrary number of neurons split over different layers. As shown in Figure 5.1, an architecture containing $L = 3$ layers consists of an input layer required to process the input features, followed by the remaining $L - 1$ layers. For vision-based tasks, the input layer processes a vector $\mathbf{x}$ that contains the pixel values of the input image. The input vector has a size depending on the number of pixels which, in the example shown, is $n = 4$. The i-th input pixel is connected to the corresponding neuron $a_i^{[1]}$ of the input layer, while the neurons $a_i^{[l]}$ in the subsequent layers map the output of the input layer into the final prediction $\hat{\mathbf{y}}$. Figure 5.1 represents a fully-connected architecture, where each neuron in a certain layer is connected to all neurons of the previous and following layer. Therefore, with the exception of the input layer, the output of each layer depends directly on the output of the previous layer and indirectly on the output of all the preceding layers. The deep interconnection between neurons plays a key role in both forward propagation and backpropagation.

Forward Propagation

The calculations performed by each neuron to transfer the input $\mathbf{x}$ to the subsequent layer define what is called forward propagation algorithm. Figure 5.2 illustrates the computation steps performed by a single neuron. First, the input vector $\mathbf{x}$=$(x_1, x_2, x_3, x_4)^T$ to the neuron a is used to compute z through the use of the neuron parameters: the weight vector $\mathbf{w}$ and the bias b. $\mathbf{w}$ is a vector of real numbers with size $n \times 1$ - where n is the number of input features - and b is a real number. Second, the obtained number is scaled by an activation function g to the desired output range, resulting in the output a (or $\hat{y}$). For a neuron belonging to any layer other than the output layer, a will be given as input to the subsequent layer. The overall procedure is given by the following equations:

$$z = \mathbf{w}^T\mathbf{x} + b \tag{5.1}$$

$$a = g(z) \tag{5.2}$$

Weights and biases represent the trainable parameters of a neural network. When referring to the trainable parameters contained in the entire architecture, it is common to indicate them with the notation $\boldsymbol{\pi}$. At the training stage, weights and biases of each neuron are tuned so that the output predicted by the network is as close as possible to the real output. Intuitively, the component of a weight vector w_i can be seen as a measure of the correlation between the neuron output a and the corresponding input x_i, while the bias b simply introduces an offset in the activation function. A common choice for the

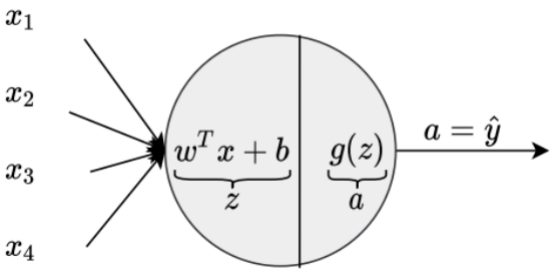


Figure 5.2: Computations performed by a single neuron (Van der Heijden, 2022).

activation function is the Rectified Linear Unit (ReLU) function:

$$g(z) = \begin{cases} 0 & z \leq 0 \\ z & z > 0 \end{cases} \tag{5.3}$$

which reduces the computational complexity and, as shown by Krizhevsky et al. (2017), reduces convergence time. However, other activation functions, such as the sigmoid and the Exponential Linear Unit (ELU) functions, can be found in literature and have been used in several architectures. A common feature of any activation function is the introduction of the non-linearity property between the input and output of a neuron. It is thanks to this feature that NNs are able to interpolate non-linear functions and perform complex tasks. Equations (5.1)-(5.2) can be extended to the forward propagation over an entire layer. By using as a reference the architecture in Figure 5.1, the output of the first layer can be written as:

$$z_1^{[1]} = \mathbf{w}_1^{[1]T}\mathbf{x} + b_1^{[1]} \tag{5.4}$$

$$a_1^{[1]} = g(z_1^{[1]}) \tag{5.5}$$

...

$$z_4^{[1]} = \mathbf{w}_4^{[1]T}\mathbf{x} + b_4^{[1]} \tag{5.6}$$

$$a_4^{[1]} = g(z_4^{[1]}) \tag{5.7}$$

where $\mathbf{w}_i^{[l]}$ and $b_i^{[l]}$ are the weight vector and bias associated to the i-th neuron of the l-th layer, respectively. By taking into account that the input to the second layer is the output of the first layer, *i.e.*, using the notation $\mathbf{a}^{[1]}=(a_1^{[1]}, a_2^{[1]}, a_3^{[1]}, a_4^{[1]})^T$ for the input to the second layer, the output of the second layer becomes:

$$z_1^{[2]} = \mathbf{w}_1^{[2]T}\mathbf{a}^{[1]} + b_1^{[2]} \tag{5.8}$$

$$a_1^{[2]} = g(z_1^{[2]}) \tag{5.9}$$

...

$$z_3^{[2]} = \mathbf{w}_3^{[2]T}\mathbf{a}^{[1]} + b_3^{[2]} \tag{5.10}$$

$$a_3^{[2]} = g(z_3^{[2]}) \tag{5.11}$$

Equations (5.4)-(5.11) can be formulated in a more general sense. Given the l-th layer of a neural network receiving n input features from layer $l-1$ and composed by m neurons, the forward propagation steps for a single data instance can be written as:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \tag{5.12}$$

$$\mathbf{a}^{[l]} = g(\mathbf{z}^{[l]}) \tag{5.13}$$

where the following notation has been used:

$$\mathbf{a}^{[l]} = \begin{pmatrix} a_1^{[l]} & a_2^{[l]} & a_3^{[l]} & \dots & a_n^{[l]} \end{pmatrix}^T \in \mathbb{R}^{n\times 1} \tag{5.14}$$

$$\mathbf{W}^{[l]} = \begin{bmatrix} \mathbf{w}_1^{[l]} & \mathbf{w}_2^{[l]} & \mathbf{w}_3^{[l]} & \dots & \mathbf{w}_m^{[l]} \end{bmatrix}^T \in \mathbb{R}^{m \times n} \tag{5.15}$$

$$\mathbf{b}^{[l]} = \begin{pmatrix} b_1^{[l]} & b_2^{[l]} & b_3^{[l]} & \dots & b_m^{[l]} \end{pmatrix}^T \in \mathbb{R}^{m \times 1} \tag{5.16}$$

$$\mathbf{z}^{[l]} = \begin{pmatrix} z_1^{[l]} & z_2^{[l]} & z_3^{[l]} & \dots & z_m^{[l]} \end{pmatrix}^T \in \mathbb{R}^{m \times 1} \tag{5.17}$$

In case of k data instances, Equations (5.12)-(5.13) become:

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]}\mathbf{A}^{[l-1]} + \mathbf{B}^{[l]} \tag{5.18}$$

$$\mathbf{A}^{[l]} = g(\mathbf{Z}^{[l]}) \tag{5.19}$$

with:

$$\mathbf{A}^{[l]} = \begin{bmatrix} a_1^{[l](1)} & \dots & a_1^{[l](k)} \\ a_2^{[l](1)} & \dots & a_2^{[l](k)} \\ a_3^{[l](1)} & \dots & a_3^{[l](k)} \\ \vdots & \ddots & \vdots \\ a_n^{[l](1)} & \dots & a_n^{[l](k)} \end{bmatrix} \in \mathbb{R}^{n \times k} \tag{5.20}$$

$$\mathbf{B}^{[l]} = \begin{bmatrix} b_1^{[l]} & \dots & b_1^{[l]} \\ b_2^{[l]} & \dots & b_2^{[l]} \\ b_3^{[l]} & \dots & b_3^{[l]} \\ \vdots & \ddots & \vdots \\ b_m^{[l]} & \dots & b_m^{[l]} \end{bmatrix} \in \mathbb{R}^{m \times k} \tag{5.21}$$

$$\mathbf{Z}^{[l]} = \begin{bmatrix} z_1^{[l](1)} & \dots & z_1^{[l](k)} \\ z_2^{[l](1)} & \dots & z_2^{[l](k)} \\ z_3^{[l](1)} & \dots & z_3^{[l](k)} \\ \vdots & \ddots & \vdots \\ z_m^{[l](1)} & \dots & z_m^{[l](k)} \end{bmatrix} \in \mathbb{R}^{m \times k} \tag{5.22}$$

In conclusion, the forward propagation algorithm represents the process by which a neural network makes predictions, *i.e.*, it generates an output for a given input. Assigned a set of weights and biases, the output of a model is fixed and can be computed analytically thanks to Equations (5.18)-(5.19). The physical meaning of the output depends on the specific task to be performed. For an OD network, it typically represents the location of the spacecraft in the image, along with a score indicating the network confidence about its prediction, while a KD model can provide the location of each feature of interest in a variety of forms, from direct pixel locations to heatmaps.

What has not been covered so far, however, is how the optimal weights and biases of each neuron are calculated. This is done by defining an optimisation problem, commonly called *network training*, where the network parameters are optimised through minimisation of a cost function. The algorithm is initialised by assigning the parameters small random values between 0 and 1, updating them at every iteration. The principles behind the optimisation algorithm are strictly related to the concept of backpropagation, which will be introduced hereafter.

Backpropagation

To have a network performing a certain task, it is necessary to tune the weights and biases of every neuron. This is done by solving an optimisation problem where the network parameters $\boldsymbol{\pi}$ are updated to minimise a cost function.

For a certain data instance $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}$, with $\mathbf{x}^{(i)} \in \mathbb{R}^{n \times 1}$ and $\mathbf{y}^{(i)} \in \mathbb{R}^{m \times 1}$, a measure of the network prediction accuracy can be obtained by computing a loss function $\mathcal{L}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$, a real-value function

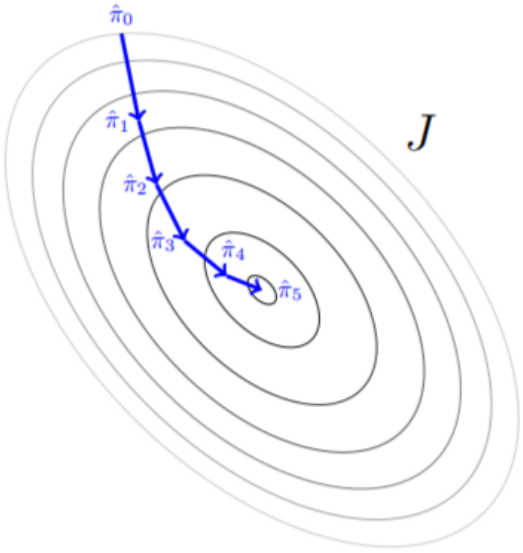


Figure 5.3: Gradient descent algorithm. Darker contour lines indicate a lower value of the cost function.

whose value decreases the closer the network prediction $\hat{\mathbf{y}}^{(i)}$ is to the true prediction $\mathbf{y}^{(i)}$. Typically, the cost function used during training is defined using the loss function calculated over multiple data instances. Given k data instances, a typical cost function has the following expression:

$$J(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) \tag{5.23}$$

with $\hat{\mathbf{Y}}$ and $\mathbf{Y}$ indicating the entire set of predictions and labels, respectively:

$$\hat{\mathbf{Y}} = \begin{bmatrix} \hat{\mathbf{y}}^{(1)} & \cdots & \hat{\mathbf{y}}^{(k)} \end{bmatrix} \in \mathbb{R}^{m \times k} \tag{5.24}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^{(1)} & \cdots & \mathbf{y}^{(k)} \end{bmatrix} \in \mathbb{R}^{m \times k} \tag{5.25}$$

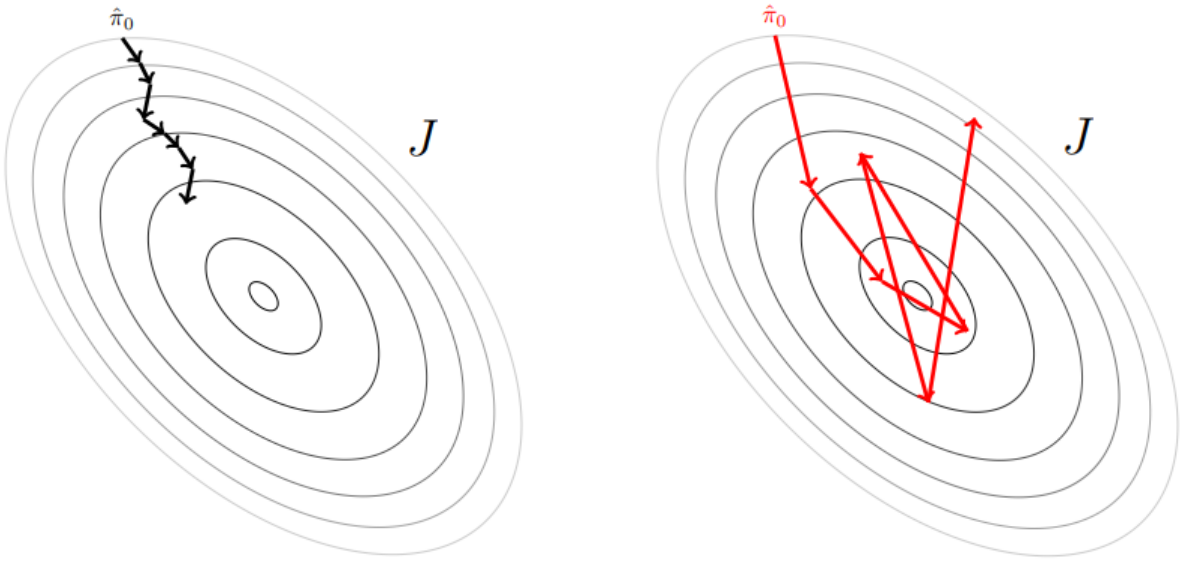
Once the cost function is defined, the optimisation problem can be formulated as:

$$\hat{\boldsymbol{\pi}} = \text{argmin}_{\boldsymbol{\pi}} J(\hat{y}, y) = \text{argmin}_{\boldsymbol{\pi}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \tag{5.26}$$

where $\hat{\boldsymbol{\pi}}$ represents the set of optimal network parameters. The optimisation problem is solved using the gradient descent method, which updates the parameters by moving along the direction opposite to the one identified by the gradient of the cost function. As visualised in Figure 5.3, this allows updating the parameters by moving towards the cost function minimum, however, as not all optimisation problems are convex, convergence cannot be always guaranteed. Furthermore, it is important to properly tune the algorithm parameters to avoid issues with numerical convergence. The update law for the gradient descent algorithm is given by:

$$\hat{\boldsymbol{\pi}}_{n+1} = \hat{\boldsymbol{\pi}}_n - \alpha \frac{\partial J}{\partial \boldsymbol{\pi}}\bigg|_{\hat{\boldsymbol{\pi}}_n} \tag{5.27}$$

It can be seen from the above equation that two parameters affect the gradient-descent learning rule: the initial choice of the parameters $\hat{\boldsymbol{\pi}}_0$ and the parameter α, which is called *learning rate*. As already mentioned, network parameters are usually initialised with small random values to avoid issues caused by a homogeneous initialisation to zero, such as vanishing gradient. The choice of an appropriate learning rate affects not only the algorithm convergence time but also the possibility of converging to a local minimum. As can be seen from Figure 5.4, a learning rate that is too small can increase substantially the number of iterations required by the algorithm to converge (Figure 5.4a), while a large



(a) Small learning rate.

(b) Large learning rate.

Figure 5.4: Effect of the learning rate on the convergence of the gradient descent algorithm. Darker contour lines indicate a lower value of the cost function.

learning rate may cause overshooting or even convergence to another, sub-optimal solution (Figure 5.4b). For this reason, the learning rate is typically tuned by trial and error or using more elaborate techniques, such as learning rate decay. In any case, the learning rate is one of the most important hyperparameters to tune before training can be started (see Section 5.2.1).

There is the issue, however, that the gradient in Equation (5.27) has to be calculated. Given that neural networks have millions of parameters, this step can be quite computationally expensive. The cost function gradient with respect to every network parameter is calculated using backpropagation, which relies on the recursive application of the chain rule. The computation starts from the output layer, tracking back the dependency on a certain parameter through all the intermediate layers and computing the gradient of each intermediate dependency.

Even though the cost function gradient can be computed analytically, it is good practice to rely on libraries that already implement backpropagation. The most common open-source ML libraries are Pytorch[1] and Tensorflow[2]. Thanks to their pre-implemented pipelines, network training can be started by providing basic hyperparameters, the network architecture, and the training data.

5.1.2. Convolutional Neural Networks

CNNs are a particular type of neural network whose architecture is inspired by biological visual systems. Compared to traditional NNs, CNNs are able to process large amounts of data as input, it is possible to encode image features directly in the architecture - making them more suited for image-related tasks - and tend to avoid overfitting (O'Shea et al., 2015).

CNNs were initially proposed for handwritten digit recognition (LeCun et al., 1989). Some years later, the first baseline for a CNN architecture was provided by LeCun et al. (1995), along with a broader discussion about CNN's potential applications. The architecture was deemed useful not only for image-based tasks but also for videos and speeches. More recently, an overview of the current progress in CNNs is outlined in the work by Gu et al. (2018). The interest shown in recent years toward this type of architecture was mainly driven by the work by Krizhevsky et al. (2017), which introduces the AlexNet baseline architecture. Its performance on the ImageNet dataset showcased the potential of CNNs when applied to visual data, and more recent work has been focusing on potential improvements to AlexNet to achieve better performance. ZFNet by Zeiler et al. (2014), VGGNet by Simonyan et al. (2014), GoogLeNet and LeNet bySzegedy et al. (2015) and He et al. (2016) are only some of the examples.

[1] `https://pytorch.org/` (visited on 27/12/2022)
[2] `https://www.tensorflow.org/` (visited on 27/12/2022)

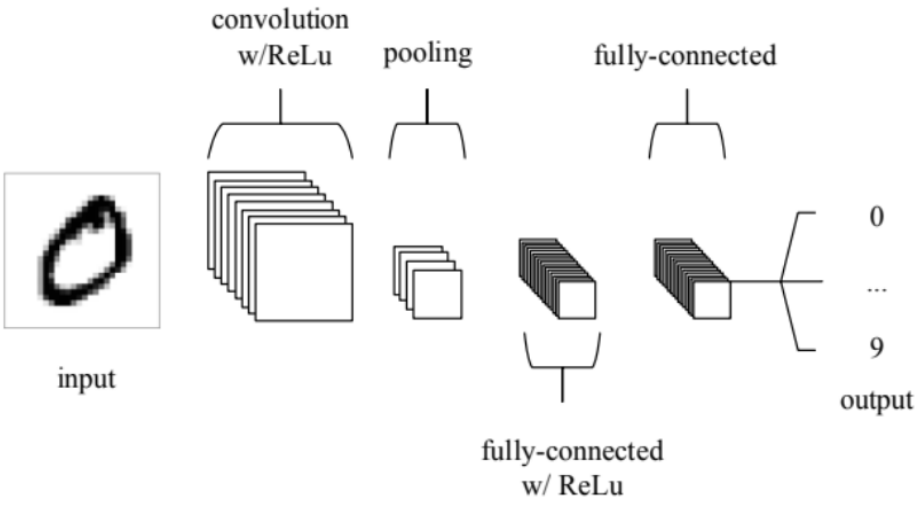


Figure 5.5: Example of a 5-layer, CNN architecture with ReLU activation (O'Shea et al., 2015).

The advantages of CNN architectures lie in the basic building blocks every CNN is built on. A 5-layer CNN architecture with ReLU activation function is shown in Figure 5.5. In the figure, the different elements of a baseline CNN architecture can be identified. The architecture relies on three main blocks: convolution, pooling, and fully-connected layers. It is thanks to the combination of these layers that CNNs are able to perform a variety of tasks with greater accuracy compared to other architectures. Each block will be described in the upcoming sections.

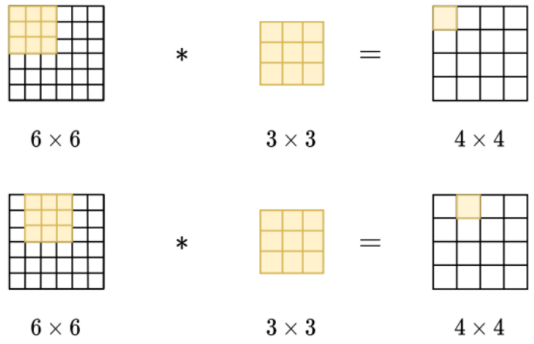
Convolution Layers

Convolution layers are a feature common to any CNN architecture. The concept behind them is that the network extracts information from the input image by sliding across it using a kernel function. The full image is therefore scanned sequentially by a set of kernels containing few parameters instead of having a layer with as many as millions of parameters.

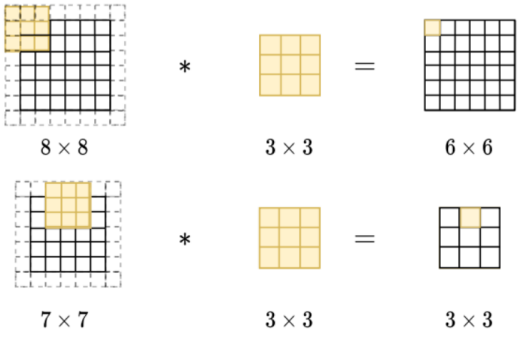
Because of the discrete nature of pixel data, continuous convolution cannot be applied, making it necessary to define a discrete convolution. Given an input image I and a kernel K with size $N_K \times N_K$, the convolution operation of K over I yields the function S, defined as follows (Goodfellow et al., 2016):

$$S(i,j) = (I * K)(i,j) \coloneqq \sum_{n=1}^{N_K} \sum_{m=1}^{N_K} I(i-n, j-m)K(n,m) \tag{5.28}$$

where i and j are the pixel coordinates and $*$ is used as short-hand notation for the convolution operation. A kernel can be seen as a filter. Kernels are commonly used in CV to extract specific features out of an input image, such as lines, edges, and high-contrast regions. The result of the convolution depends on the size of the kernel and the input image itself. Figure 5.6a shows the convolution operation applied to an image with size 6×6 and a kernel size of 3×3. It can be seen how each pixel in the output image is the result of the kernel sliding across the image and summarising the information contained in a set of pixels in a single value. What can also be seen is that the output image has a smaller size compared to the input image. This is a direct consequence of how the convolution operation is defined in Equation (5.28), as each element of the kernel has to fall within the input image to perform a step of the convolution. On the one hand, this helps CNNs to extract more meaningful features and achieve better generalisation. On the other hand, this results in a loss of information near the edges. This problem can be compensated for by introducing pixels around the edges of the image with a process called *padding*. This process is parameterised by a padding parameter p indicating the number of pixel rows and columns added to each edge of the image. Each of the introduces pixels has a value of zero, meaning that no information is added to the image. There is another convolution parameter, namely the striding parameter s. It defines the number of pixels the kernel is shifted along the image to perform the convolution.



(a) Illustration of the convolution operation for a kernel dimension $N_K = 3$ over an input image with size 6×6.



(b) Convolution operation over an image using padding $p = 1$ and a kernel size $N_K = 3$. The top figure represents a convolution over an input image with size $N_I = 6$ striding $s = 1$. The bottom figure represents a convolution with an input image size of $N_I = 5$ and stride $s = 2$.

Figure 5.6: Examples of a convolution operation (Van der Heijden, 2022).

The input-image size, the kernel size, the padding parameter, and the striding parameters are the convolution parameters for a typical convolution layer. They should be tuned like other hyperparameters - with an optimal value depending on the CNN architecture and the task of interest - as the result of the convolution can change significantly. Figure 5.6b shows how the size of the output image changes, using a padding parameter $p = 1$ and a kernel size $N_K = 3$, by varying the input image size and the striding parameter. With an input image size $N_I = 6$ and striding $s = 1$, the output image maintains its original size. However, an input image with $N_I = 5$ convolved with striding $s = 2$ results in a 3×3 output.

Figure 5.6 shows how the size of the output image of a convolution varies depending on the input image size N_I, the kernel size N, the padding parameter p, and the striding parameter s. In general, the size of the output of a convolution operation is given by the following formula:

$$\text{size}(S) = \left(\frac{N_I + 2p - N_K}{s} + 1\right) \times \left(\frac{N_I + 2p - N_K}{s} + 1\right) \tag{5.29}$$

A convolution layer can contain an arbitrary number of kernels or filters. These filters are randomly initialised and are updated during training, following the same learning rule as any other trainable network parameter. This feature is what enables CNNs to learn relevant features by themselves. Before the advent of CNNs, features were hand-engineered by including filters emphasising arbitrary features identified as relevant *a-priori*. This reduces not only the flexibility of the architecture but also its learning capabilities. By letting the network learn autonomously, the features do not have to be determined beforehand and the network is not constrained to rely on low-level features such as lines or edges, but it can rely on more abstract features as well. Because of the property of extracting features from the input image, the output of a convolution layer is also called *feature map*. Furthermore, the use of convolution layers brings two additional benefits: (i) it allows parameter sharing and (ii) it increases the sparsity of connections. The former enables the scanning of the input image with a reduced number of parameters and, together with pooling layers, makes CNNs shift-invariants. The latter makes the model more memory-efficient.

Pooling Layers

Pooling layers are another element common to all CNN architecture. Their scope is directly related to convolution layers, which is to downsample the resolution of the feature map. Because of this, a pooling layer is usually found after a convolution layer or, in the case of multiple convolution layers, between one convolution layer and the next one. Pooling layers can be mathematically described by a pooling operation, that is, a function that maps an input into a low-resolution representation. The general pooling operation can be expressed as in Gu et al. (2018):

$$y_{ijk}^{[l]} = \text{pool}(a_{mnk}^{[l]}), \forall (m, n) \in \mathcal{R}_{ij} \tag{5.30}$$

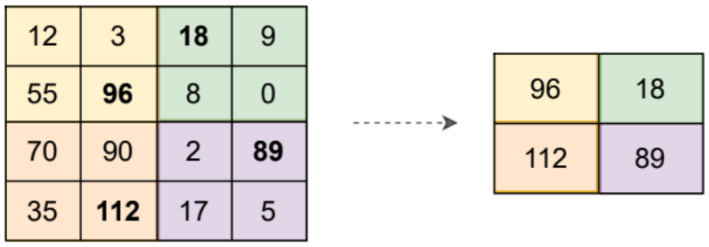


Figure 5.7: Example of max pooling operation from Van der Heijden (2022).

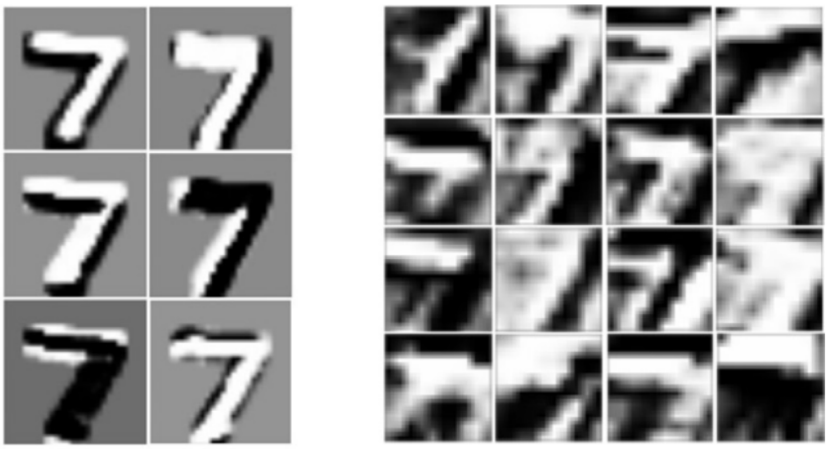


Figure 5.8: Sample output of the first (left) and third (right) convolution layers of the LeNet-5 network (Gu et al., 2018) for handwritten digit recognition. The first layer outputs six feature maps with resolution 28×28, while the third layer outputs 16 feature maps with resolution 10×10.

where $a^{[l]}_{mnk}$ is the output of the k-th filter of the l-th convolution layer (or feature map) in the location (m, n), while $y^{[l]}_{ijk}$ is the output of the corresponding pooling layer in the location (i, j). pool is the pooling function and $\mathcal{R}_{ij}$ is a local neighbourhood around the location (i, j). Typical pooling operations are max pooling and average pooling. An example of pooling operation is given in Figure 5.7, where a 4×4 feature map undergoes a max pooling operation resulting in a 2×2 output. In this case, the pooling function maps a region with size 2×2 to its maximum value. For instance, for the upper-left region of the feature map, the maximum is 96, which is the only one preserved after the pooling operation. Thanks to the combination of convolution and pooling layers, CNNs have the property of being shift-invariants, meaning that the learned features do not depend on their position in the image. This is one of the reasons why CNNs outperformed previously existing methods in tasks, such as digit recognition, where meaningful features do not depend on their own location. In particular, a cascade of multiple convolution and pooling layers enables the network to learn features with a higher degree of abstraction. Figure 5.8 shows the output of two convolution layers of the LeNet-5 architecture applied to digit recognition. It can be seen how the six filters of the first convolution layers learn features related to the digit's edges and shape (on the left), while the output of the 16 filters of the third convolution layer is related to more abstract features (on the right).

Fully Connected Layers

Fully Connected (FC) layers are typically placed after the sequence of convolution and pooling layers, just before the output layer. They cover the same role and have the same structure as hidden layers in typical NN architectures. Their aim is to fully leverage the features extracted by the previous layer to perform high-level reasoning and transform the feature map into the desired output. To achieve this, every neuron of the first FC layer is connected to every neuron of the last pooling layer, while every neuron of an intermediate FC layer is connected to all neurons from the previous and subsequent FC layer. Given such a dense interconnection, FC layers represent the region where the majority of parameters are in a CNN architecture. However, most of the tasks can be performed with just a few FC

layers.

5.2. Machine Learning Concepts

This section introduces several ML concepts taken into account for the development of the pose estimation pipeline. To begin with, hyperparameter tuning is used to find the optimal training configuration. Next, the concept of model generalisation is necessary to describe the model's capability to perform well on previously unseen data. Several techniques have been proposed over the years, with regularisation and data augmentation being undoubtedly the most successful for ML training. Lastly, since training a model from scratch can take a significant time and might lead to a suboptimal solution, a strategy based on transfer learning allows to leverage checkpoints trained on different tasks. Each of these concepts will be introduced in the upcoming sections.

5.2.1. Hyperparameter Tuning

Model hyperparameters are parameters, unlike weights and biases, that are not learned by the network. They are set prior to the training process and affect the outcome of the process. A hyperparameter can affect the network architecture, such as the aforementioned input resolution and padding parameter, the number of layers, and the number of neurons, or the training process itself. Examples of the latter are the learning rate, number of epochs, loss function, and so on. Before being trained, each network used in this thesis will go through a stage of hyperparameter tuning. First, the architecture of each model will be reviewed to ensure its compatibility with the task of interest. The specifics about architectural modifications for OD and KD models can be found in Sections 7.2 and 8.3, respectively. Second, the training settings will be finetuned before each training run. The reader is referred to Sections 7.3 and 8.4 for an overview of the settings used for OD and KD model training.

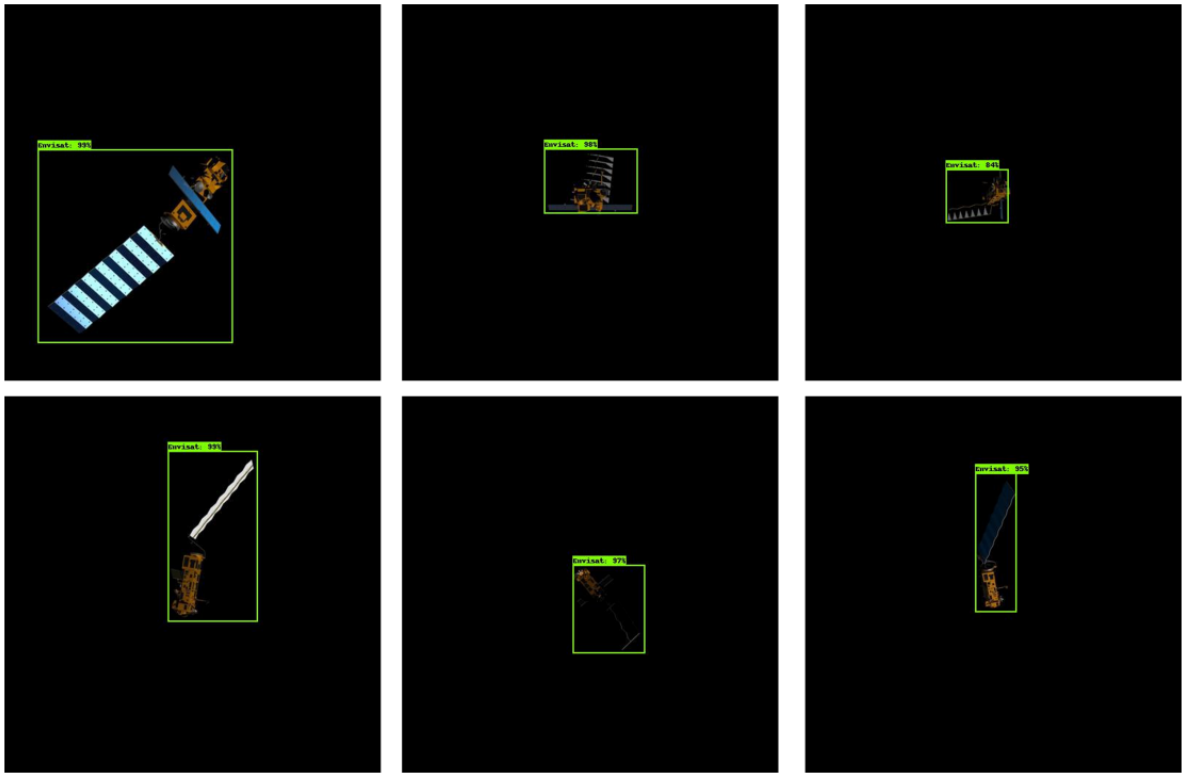
5.2.2. Generalisation

Generalization in NNs refers to their ability to make accurate predictions on unseen data. Since models are trained on a limited dataset, they aim to approximate the underlying distribution of a broader set of possibilities. For example, when performing handwritten digit recognition, it is impractical to include all possible handwriting variations in the training data. To evaluate the network's generalization, a validation and test set are collected independently from the training data. The validation set assesses the model's performance during training, while the test set provides a final performance evaluation after training.

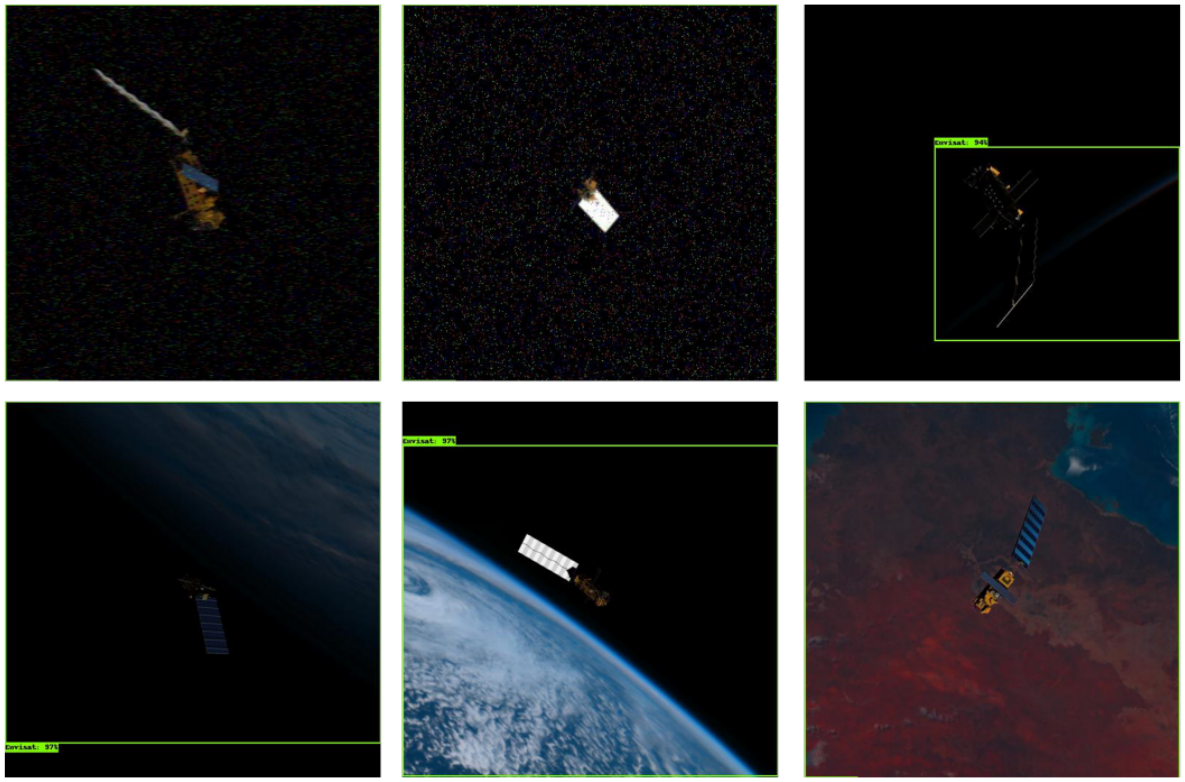
In principle, a model's generalization ability can be assessed by examining its training and validation loss. A low training loss indicates the model is learning to recognize patterns and has minimal bias, while a good validation loss suggests low variance between training and validation data. When a model performs well on both losses, it likely indicates effective learning. However, high values for either loss signal potential issues in the training process. High training loss suggests *underfitting*, where the model struggles to approximate the training data distribution, often due to a simple architecture with few parameters. Conversely, a decreasing training loss and increasing validation loss indicate *overfitting*, where the model learns training-specific features that do not generalize to the test data.

Concerning spacecraft pose estimation, the driving factor in generalisation is given by the domain-gap issue previously mentioned. Most datasets rely on synthetic images for training and real images for testing. Because of the wide differences between the two domains, CNNs struggle to properly generalise the features learned on synthetic images to testbed ones. Cassinis (2022), for instance, experienced the issue when testing the pipeline trained with synthetic ENVISAT data on testbed images with extreme illumination conditions. Similarly, Barad (2020) shows how some of the trained models would lack robustness over images including heavier corruptions. Figure 5.9 illustrates the issue observed by the author. Despite the OD network performing well on clean images (Figure 5.9a), its accuracy suffers significant degradation on corrupted ones (Figure 5.9b). In particular, adding image noise, challenging illumination conditions, or the Earth in the background causes the model to make poor predictions about the spacecraft location in the image or not detecting the spacecraft at all. This is the result of poor generalisation capabilities, caused by the model being too small to capture all the relevant features (underfitting) or by the training set not including such features in the first place (overfitting).

Preventing underfitting and overfitting is therefore a crucial part of ML training. The upcoming section discusses the different methods used in this thesis to avoid both behaviours.



(a) Accurate predictions.



(b) Poor predictions.

Figure 5.9: Illustration of the domain-gap issue on synthetic ENVISAT images (Barad, 2020).

5.2.3. Regularisation and Data Augmentation

As mentioned in the previous section, it is virtually impossible to include a fully comprehensive set of samples in the training set. While preparing a test set allows to test how well a model generalises,

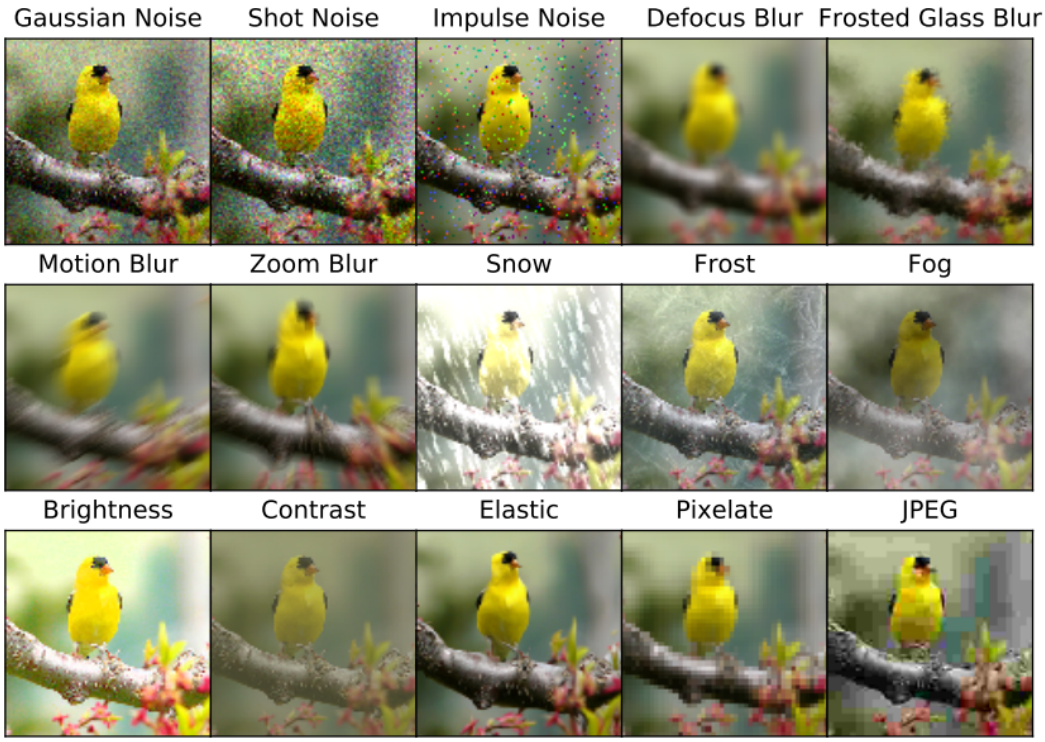


Figure 5.10: Illustration of the different augmentations listed by Hendrycks et al. (2019) as commonly applied during model training.

regularisation and data augmentation techniques force the model to generalise better during training.

Regularisation is mainly used to avoid overfitting. Because of this, the different methods proposed try to maximise the network performance on the test rather than the training set. The following techniques will be used during model training in this research:

- **Weight decay:** This technique consists of modifying a cost function by adding a penalty related to network weights. Given a cost function J, the regularised cost function is computed as:

$$J_r(\hat{\mathbf{Y}}, \mathbf{Y}) = J(\hat{\mathbf{Y}}, \mathbf{Y}) + \lambda \|\boldsymbol{\pi}\|_p \tag{5.31}$$

  where p determines the regularisation type, often indicated as L_p, and $\boldsymbol{\pi}$ indicates the trainable network parameters. The most common regularisation types based on weight penalties are L_1 and L_2. For $p = 1$, the cost function will lead to higher connection sparsity, *i.e.*, several weights will be very close to zero, and more selective feature selection in the final model. For $p = 2$, the cost function will lead to smaller weights throughout the network. In either case, the decay term prevents the weights in the network from growing too large and cause overfitting.
- **Earlystopping:** This technique simply consists of training a model for a large number of epochs to see when overfitting occurs. As already mentioned, overfitting usually occurs once the validation loss starts increasing while the training loss keeps decreasing. Earlystopping simply allows to save the model with the minimum validation loss. Because of its simplicity, it can be easily implemented in a training pipeline. This will ensure that, at the end of the training, it is possible to test the model achieving the best performance.

In general, multiple regularisation strategies can be applied at the same time. However, despite the vast number of regularisation methods proposed over the years, data augmentation remains the most effective strategy to improve a model's generalisation capability. It consists of increasing the size of the training dataset either by including new data instances or by creating new ones artificially. With regard to the creation of artificial data, augmentations often rely on image processing through certain filters, such as random brightness and contrast. Besides increasing the amount of data available to the network, training the network on data with features artificially included allows the model to be

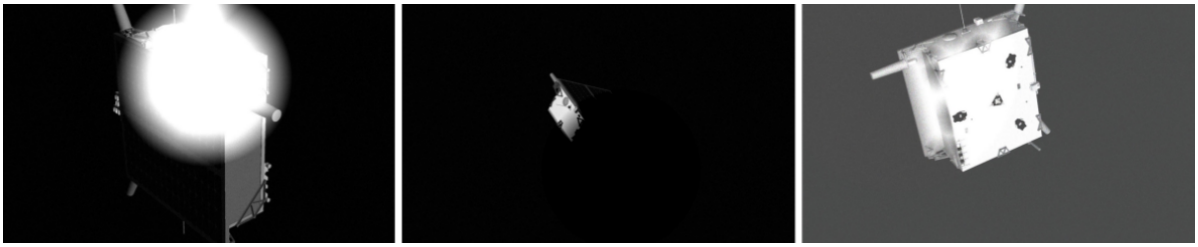


Figure 5.11: Light randomisation filters applied by Cassinis (2022) in his Tango pipeline.

more robust to the features themselves. Research has been performed on the augmentations to apply to improve the model performance the most. Hendrycks et al. (2019) elaborated a set of best practices to observe in every ML pipeline. The paper also provides a list of commonly-applied augmentations replicating several image corruptions, such as image noise, blur, and JPEG compression. An illustration of the different augmentations is provided in Figure 5.10.

Besides augmentations common to any ML application, it is necessary to account for features specific of the application of interest. For spacecraft pose estimation, the example of including the Earth in the background was already presented, which undoubtedly makes inference more difficult. Over time, more elaborate augmentations have been proposed to improve the realism of synthetic data and fill the gap with testbed images. Cassinis (2022) uses several augmentations to bridge the gap with SPEED+ testbed data. Figure 5.11 the different light filters applied by the author in the augmentation pipeline, with the goal of replicating adverse illumination conditions. By doing so, the pose score reduced by more than half, from 0.96 to 0.43. This highlights the importance of setting up a proper augmentation pipeline in filling the performance gap, reason for which one of the experiments in this thesis involves studying the impact of different synthetic augmentations. More details on the implementation of the augmentation pipeline for the OD and KD models can be found in Section 9.2.1.

5.2.4. Transfer Learning

One of the reasons why most regularisation techniques or augmentations are applied is that it is not possible to collect enough data for training. Another problem related to the lack of data for model development is that novel architectures contain millions of parameters, making the use of regularisations and augmentations extremely challenging. These techniques are applied to take the most out of the available data, but a benefit can be observed only if a sufficient amount of data has already been collected. In extreme cases where the dataset size has to be increased dramatically to attain satisfactory performance, each of the techniques previously mentioned tends to fail.

An approach commonly used to compensate for this is transfer learning. It aims at solving the problem of lack of data by leveraging similarities in feature representations across two task domains. Transfer learning makes use of an existing dataset or model, developed for a certain task referred to as *source domain*, to improve the performance of a different domain called *target domain*. The assumption is that, between the data distributions of the two domains, there are similarities that allow a model trained on the source domain to apply at least part of the features learned to the target domain. As a result of this assumption, models pretrained on large existing datasets, such as ImageNet and COCO, can be used to initialise the weights of similar models performing object and keypoint detection on spacecraft.

Transfer learning will be applied in this thesis in the form of *finetuning*, that is, a model trained for a different task is fully retrained on the dataset of the target domain. This approach is used to provide a good initial guess of the model parameters, enabling the model not to learn relevant features from scratch during training. The only instance of the application of transfer learning found in the field of spacecraft relative navigation is in the work by B. Chen et al. (2019), where the author finetuned an HRNet model pretrained on ImageNet. Even if an isolated example, the successful integration of transfer learning within a pose estimation pipeline proves the possibility of applying the concept on a larger scale. Similarly, this thesis will make use of pretrained checkpoints for OD and KD training whenever available.

Part III

System Design and Methodology

6

System Overview

This chapter provides an overview of the system architecture used in this work (Section 6.1), followed by an overview of the software used (Section 6.2) and of steps undertaken for software verification (Section 6.3).

6.1. Architecture

An overview of the pose estimation architecture adopted in this work is given in Figure 6.1. As already discussed in Chapter 2, research in recent years has proven the advantages of CNN-based architectures over previous pose estimation methods, such as IP algorithms. For this reason, the developed pose estimation pipeline contains an AI-based keypoint extraction block. This block takes as input a single image captured by the on-board monocular camera and predicts the 2D location of each keypoint in the image. This is achieved using two separate networks: an object detection and a keypoint detection one.

The object detection network regresses the coordinates of the spacecraft bounding box. This allows to crop the input image and resize it to a fixed resolution to give it as input to the keypoint detection network. By doing so, the feature-detection capabilities of the system become insensitive to the object scale in the input image.

The cropped and resized image is then given as input to the keypoint detection network, which regresses the coordinates of the spacecraft keypoints as heatmaps. The predicted keypoint location is then extracted by each heatmap using the soft-argmax function (see Section 8.2 for more details). The reason for splitting keypoint extraction into two simpler subtasks is dictated by the superior performance showed by this type of systems in spacecraft pose estimation (B. Chen et al., 2019; Park et al., 2019; Park et al., 2023).

The keypoint locations are then used by the pose solver to determine the relative pose. Along with the predicted image locations, the pose solver makes use of a spacecraft keypoint model which defines the 3D location of each keypoint in the target body-fixed frame T. As explained in Section 4.3, a pose solver will be used to determine the relative pose, followed by a pose refinement step. Both steps will use the most confident keypoint predictions, ranked according to the heatmap softmax values.

For the sake of completeness, the block diagram shown in Figure 6.1 also includes a state estimation step. This is done to show how the developed pose estimation pipeline collocates within a spacecraft relative navigation system. In fact, the estimated pose can be provided to a Kalman filter as input measurement to estimate the relative state between the chaser camera and the target, closing the navigation loop. Furthermore, the measurement uncertainty can be estimated following the methodology proposed by Cassinis (2022), using heatmaps to extract information about the network detection uncertainty.

6.2. Software

As outlined in Section 6.1, the pose estimation pipeline developed in this work consists of several steps. Furthermore, generating synthetic ENVISAT data and reconstructing the Tango and ENVISAT keypoint models are additional tasks required before running the main experiments. This feature of the system

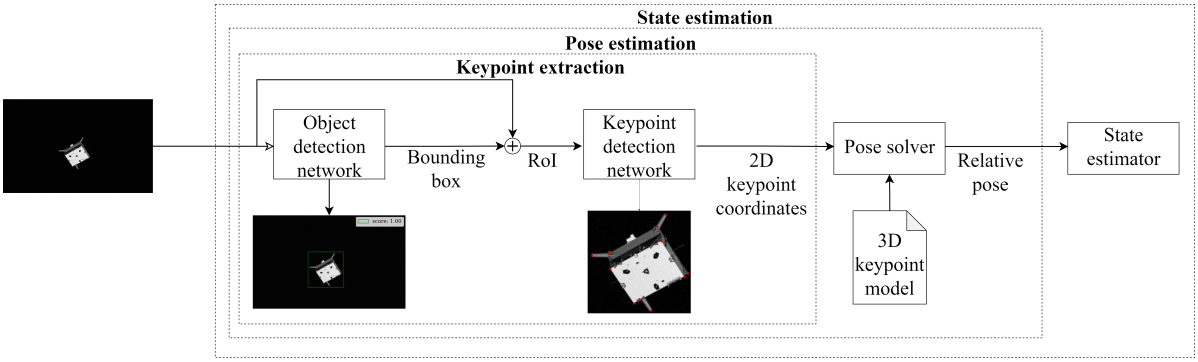


Figure 6.1: Overview of the pose estimation architecture adopted in this work and its integration within a spacecraft relative navigation system.

Table 6.1: Overview of the infrastructure and software used for the different pipeline functionalities.

ID	Functionality	Infrastructure	Software
1	Data generation	Laptop CPU	Python, Blender
2	Model reconstruction	Laptop CPU	MATLAB
3	Preprocessing	Laptop CPU, DelftBlue GPU node	Python, Pytorch
4	Object detection	DelftBlue GPU node	Python, MMDetection
5	Keypoint detection	DelftBlue GPU node	Python, Pytorch
6	Pose estimation	Laptop CPU	Python, OpenCV
7	Myriad X implementation	Myriad X, DelftBlue GPU node	Python, OpenVINO, ONNX

architecture led to the use of different infrastructures and software for the implementation of each functionality.

An overview of the used infrastructure and software is provided in Table 6.1. It can be seen how each functionality differs from the others for both the infrastructure and the software used. While the data generation, model reconstruction, and pose estimation jobs can be run on a laptop, running OD and KD models requires higher computational power. For this reason, all jobs involving training or inference of ML models are run on TU Delft's High-Performance Computer (HPC), DelftBlue[1]. The cluster provides access to nodes with NVIDIA Tesla V100S Graphic Processing Units (GPUs) for high-performance computing and allows running a job by simply logging in using the Secure Shell (SSH) protocol and submitting a shell script.

It can be highlighted from Table 6.1 how some functionalities make use of multiple running infrastructures. For instance, the preprocessing step involves generating the full dataset labels and applying style augmentation to the training images. While the former can be done on a laptop, the latter requires running inference with a style augmentor network, which requires a GPU. Similarly, the experiments concerning the Myriad X implementation are mostly run on the device itself, however, it is also necessary to run a job on DelftBlue to measure the inference time on GPU.

The synthetic ENVISAT dataset is generated using Blender[2], an open-source rendering software used for a variety of projects, including the generation of spaceborne imagery by Van der Heijden (2022) and Stolk (2022). It allows to set up a rendering environment where the camera and the target are simulated realistically, along with proper lighting and background conditions. Compared to professional rendering software specific for space applications, such as PANGU by ESA[3] and SurRender by Airbus (Brochard et al., 2018), Blender was chosen in account of its cost-free usage, extensive documentation, and possibility of using both a Graphic User Interface (GUI) and a Python API, which enables the implementation of routines generating thousands of images at a time.

Both the Tango and ENVISAT keypoint models are reconstructed using a pipeline implemented in MATLAB consisting of two main steps: the first one applies keypoint labels using MATLAB's Image

[1] `https://doc.dhpc.tudelft.nl/delftblue` (visited on 17/09/2023)
[2] `https://www.blender.org/` (visited on 17/09/2023)
[3] `https://pangu.software/` (visited on 17/09/2023)

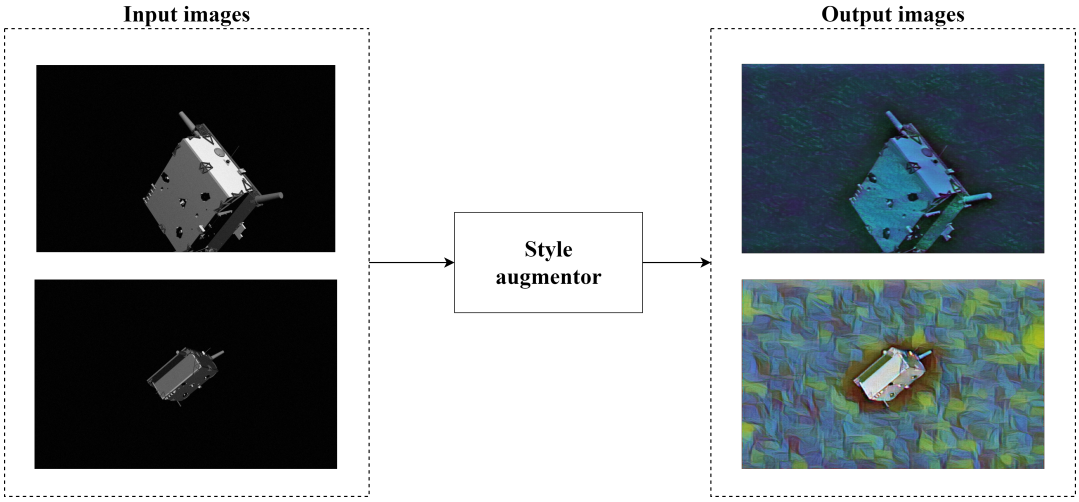


Figure 6.2: Style augmentation process.

Labeler[4], and the second one passes the generated labels to a script reconstructing the model. The labeling tool consists of an app that allows the user to manually label their data with minimal effort. For ENVISAT, this ended up being as simple as uploading an image to the app and selecting the pixel containing each keypoint.

Before running any job involving the pose estimation pipeline, it is also necessary to preprocess the datasets to generate the full labels and apply style augmentation. The full labels are generated using a Python script using the original pose labels and camera parameters to determine the keypoint and bounding box coordinates. Style augmentation is one of the augmentations considered to improve the network performance and will be explained in detail in Section 9.2.1. Unlike the other augmentations, it is applied by feeding the original image to a network changing its style, as shown by Figure 6.2. As this process requires significantly more time than the other augmentations, the best choice is to generate the augmented dataset offline and use it for training. To do this, an augmentation pipeline is implemented using the Pytorch-based style augmentor network by Jackson et al. (2019).

Given the complexity reached by model architectures, it is considered best practice to use existing model implementations instead of replicating the architecture from the original paper from scratch. For this reason, the following packages are used in the pose estimation pipeline:

- **MMDetection[5]:** This library, built on top of Pytorch, offers a high-level API to implement OD training and testing pipelines. It provides numerous architectures, directly implemented from the corresponding papers, as well as several pretrained checkpoints. Users can start training their ML models by simply providing a configuration file, which passes all the required model, dataset, and runtime settings. The high level of abstraction, combined with the extensive documentation and tutorials, made this the optimal choice for OD.
- **Pytorch[6]:** It is considered the best choice for the KD in this thesis. Even though the framework natively implements only a limited number of networks, its active community and widespread use in academia make it very easy to find open-source network implementations. In fact, most of the leveraged repositories mentioned in Section 2.1.5 use it as their ML framework of choice. Pytorch allows users to build and train NNs for a wide range of tasks. It provides a flexible computational API, making it easier to develop complex models, conduct research, and implement cutting-edge DL techniques.
- **OpenCV:** After the input image undergoes the keypoint extraction step, the relative pose is determined using OpenCV's `solvePnP` functionalities[7]. The library supports a broad range of algorithms for pose estimation, undergoes regular maintenance and updates, and is well-established in both industry and academia.

[4] `https://www.mathworks.com/help/vision/ug/get-started-with-the-image-labeler.html` (visited on 17/09/2023)
[5] `https://github.com/open-mmlab/mmdetection.git` (visited on 17/09/2023)
[6] `https://github.com/pytorch/pytorch.git` (visited on 17/09/2023)
[7] `https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html` (visited on (17/09/2023))

Lastly, the experiments on Myriad X are mostly run using OpenVINO[8], Intel's open-source toolkit to accelerate and optimize models for CV tasks. It allows users to deploy NNs on Intel's hardware, to achieve efficient and high-performance inference for different tasks, including OD and KD. The toolkit is used to run two experiments for each model on both Myriad X and a GPU: benchmarking the inference time and verifying the network accuracy on Myriad. The first experiment can be run using a benchmarking app already installed with the toolkit itself, whereas the second experiment requires implementing a custom script using OpenVINO Python API.

The toolkit, however, does not support model conversion directly from Pytorch and MMDetection, reason for which all models are converted to an intermediate format thanks to the Open Neural Network Exchange (ONNX) framework[9]. This open-source framework is designed to enable interoperability between various ML frameworks by providing a common format for representing DL models.

6.3. Software Verification

This section describes the verification performed to ensure the correct functioning of third-party software and the implemented code.

6.3.1. Unit Tests

As several functions have been implemented to set up an end-to-end relative pose estimation pipeline, unit tests have been performed throughout this research. The tests covered both functions and variables used throughout the pipelines. Overall, 19 functionalities were tested individually, covering the following areas:

- **Attitude conversion:** As already mentioned in Section 3.2.2, multiple attitude representations are used out of convenience. This made it necessary to extensively test the pipeline conversion capabilities, such as from quaternions to DCM and vice-versa.
- **Frame transformations:** Ensuring accurate transformations between two reference frames is a fundamental step in the label generation process. Hence the need to test this type of functionality in depth.
- **Label generation:** Section 2.1 outlined how pose estimation datasets may contain different types of labels. As the system performance are directly affected by the quality of the label quality, it was deemed crucial to ensure their correct generation. This group of tests involves functions, for instance, determining the location of keypoint and bounding box in the image, or the keypoint visibility label.
- **Performance evaluation:** As the different building blocks of the pipeline have to be tested individually before the final integration, it was necessary to test every function related to performance evaluation, such as computing the KD accuracy and the final pose score.

A detailed overview of the individual tests performed is provided in Appendix A.

6.3.2. Blender

Blender is one of the most commonly used open-source rendering software. Its rendering capabilities allow to generate thousands of images in a matter of minutes. It was also used by Van der Heijden (2022) to generate synthetic images of the Bennu asteroid for relative pose estimation, making it a desirable option given its widespread use and the minimal resources required.

As part of the experiments performed in this work involves generating synthetic images, it is important to verify Blender's image rendering and pose generation capabilities. The setup and outcome of the two tests are presented below.

Image Rendering

The goal of this test is to verify Blender's rendering capabilities. This is done by ensuring that, for a given scene and pose, Blender's Python API renders the image correctly. The pose labels and ground truth images are taken from Magalhaes Oliveira et al. (2019), while an overview of the pose parameters is provided in Table 6.2.

[8] `https://github.com/openvinotoolkit/openvino.git` (visited on 17/09/2023)
[9] `https://onnx.ai/` (visited on 14/10/2023)

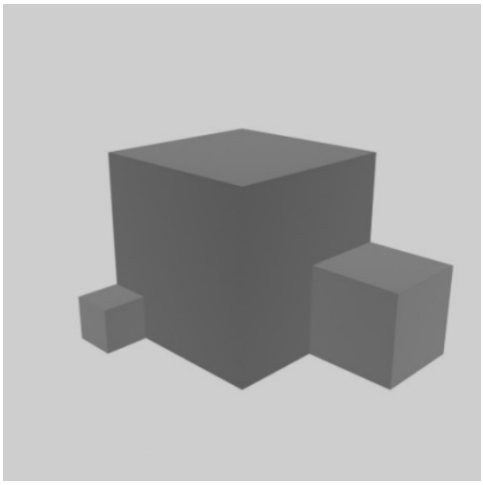


Figure 6.3: 3D view of the scene used for the image rendering test.

Table 6.2: Poses for the image rendering test.

Pose ID	x [m]	y [m]	z [m]	q_1 [-]	q_2 [-]	q_3 [-]	q_4 [-]
1	20	0	0	0.5	0.5	0.5	0.5
2	0	20	0	0	-0.707107	-0.707107	0
3	-20	0	0	0.5	-0.5	-0.5	0.5
4	0	-20	0	0.707107	0	0	0.707107
5	0	0	20	0	0	0.707107	0.707107
6	0	0	20	0	0	0	1
7	0	0	-20	0.707107	-0.707107	0	0
8	0	0	-20	0.382683	-0.923880	0	0
9	10	-20	0	0.707107	0	0	0.707107

- **Input:** Blender scene (shown in Figure 6.3) and input poses (listed in Table 6.2).
- **Expected output:** Rendered image with the object represented in a pose consistent with the ground truth image.
- **Passing criterion:** The object pose in the rendered image must coincide with the pose in the ground truth image through visual inspection. The object size is not required to be the same, as it also depends on the perspective and camera settings used.

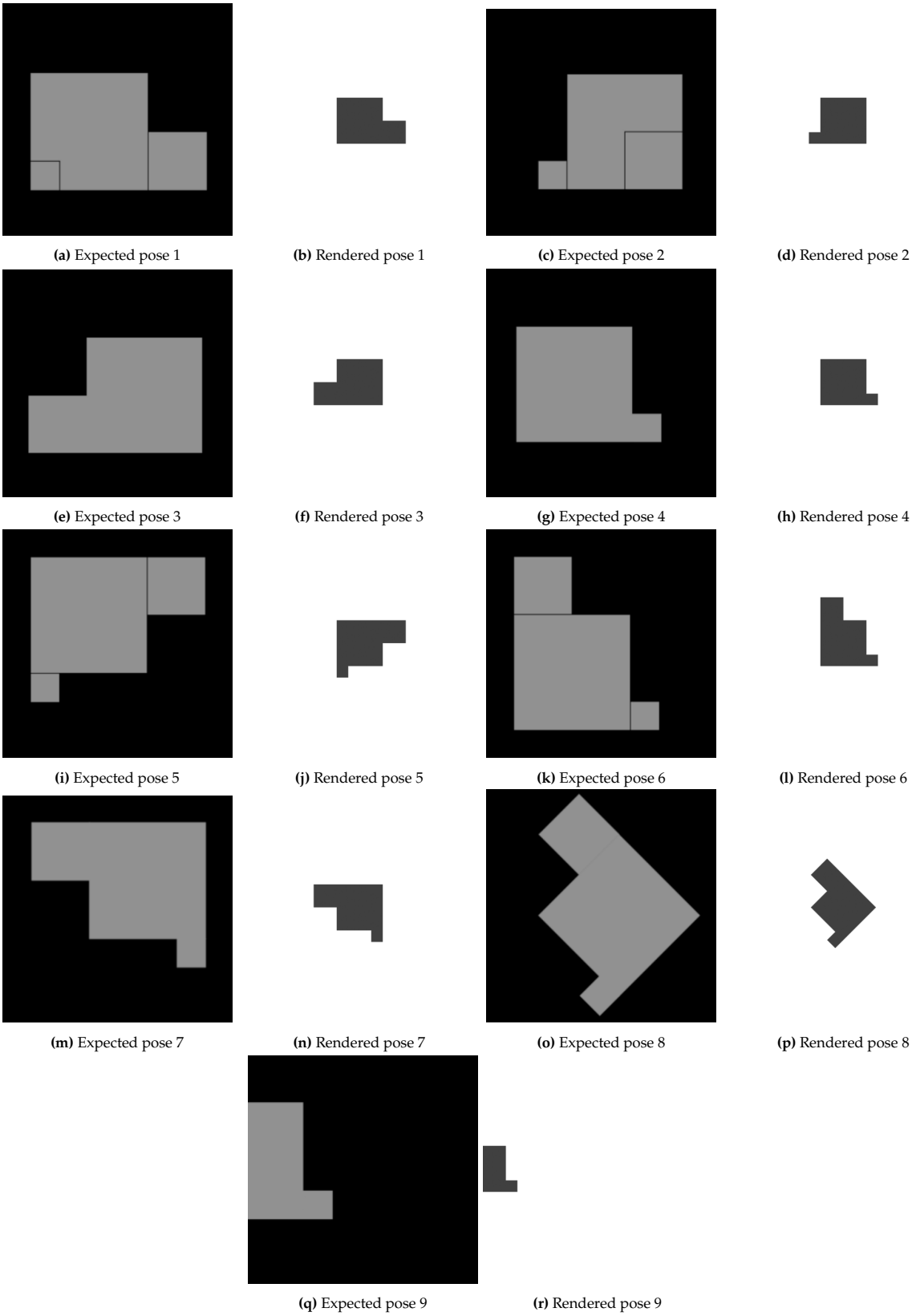
Figure 6.4 shows the rendered poses along with the ground truth images for each pose. It can be seen how object orientation with respect to the camera is consistent between the work by Magalhaes Oliveira et al. (2019) and the rendering pipeline set up in this work. As a result, the image rendering test is considered to be passed.

Pose Generation

The goal of this test is to verify the implementation of the mathematical framework introduced in Section 3.3 to generate relative pose labels. The object used in this test is a Blender's primitive cube. To verify the correct generation of pose labels, several images are rendered with the cube in different poses. The first image is rendered using the nominal pose, while the other poses are obtained using the relevant frame transformation and matrix multiplications.

The first part of the test consists of changing the nominal cube pose. The nominal pose, whose parameters are listed in Table 6.3, is used to generate the first image. After that, two additional images are generated with the cube rotated of +60° and -60°, respectively, around its vertical axis.

- **Input:** Blender scene (consisting of a primitive cube), nominal pose (see Table 6.3), and the additional rotation (0°, +60°, and -60°).
- **Expected output:** Rendered image with the cube in the desired pose.



(a) Expected pose 1 **(b)** Rendered pose 1 **(c)** Expected pose 2 **(d)** Rendered pose 2

(e) Expected pose 3 **(f)** Rendered pose 3 **(g)** Expected pose 4 **(h)** Rendered pose 4

(i) Expected pose 5 **(j)** Rendered pose 5 **(k)** Expected pose 6 **(l)** Rendered pose 6

(m) Expected pose 7 **(n)** Rendered pose 7 **(o)** Expected pose 8 **(p)** Rendered pose 8

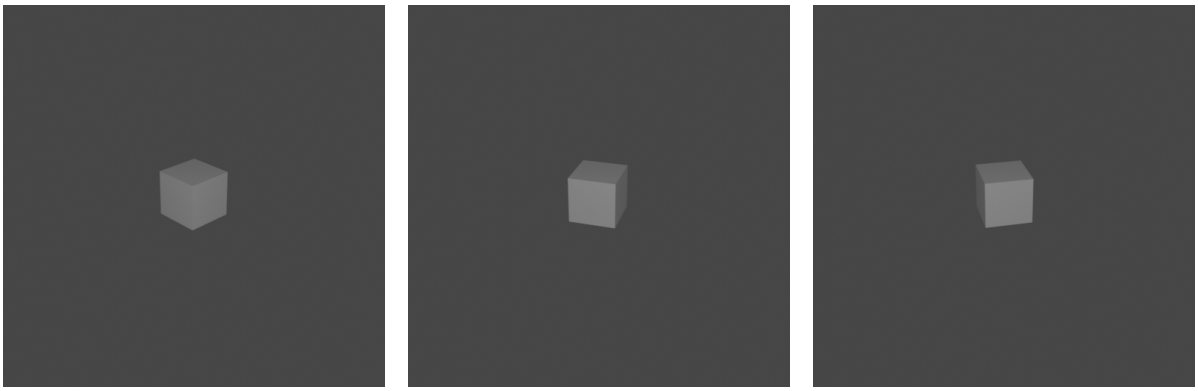
(q) Expected pose 9 **(r)** Rendered pose 9

Figure 6.4: Results for the image rendering verification test.

- **Passing criterion:** The cube in the rendered image must be in the right pose (verified through visual inspection).

Table 6.3: Nominal pose parameters for the pose generation test (Van der Heijden, 2022).

x [m]	y [m]	z [m]	q_1 [-]	q_2 [-]	q_3 [-]	q_4 [-]
7.359	-6.926	4.958	0.483536	0.208704	0.336872	0.780483



(a) Nominal pose (0° rotation about the cube's vertical axis).

(b) +60° rotation about the cube's vertical axis.

(c) -60° rotation about the cube's vertical axis.

Figure 6.5: Rendered images for the different poses of the pose generation test.

Table 6.4: Maximum and required average reprojection error for the Tango and ENVISAT 3D keypoint models.

Spacecraft	ϵ_{max} [pix]	ϵ_{req} [pix]
Tango	1.59	2.00
ENVISAT	1.04	2.00

Figure 6.5 shows the rendered images for the three different cubes. It can be seen how the observed cube rotation is consistent with the expected rotation about its vertical axis.

The second part of the test consists of applying pointing errors to the nominal pose using Equation (3.40). The same nominal pose as for the first part of the test is used. The in-plane rotation angles α and β are set to +4° and -4° covering all the four possible combinations.

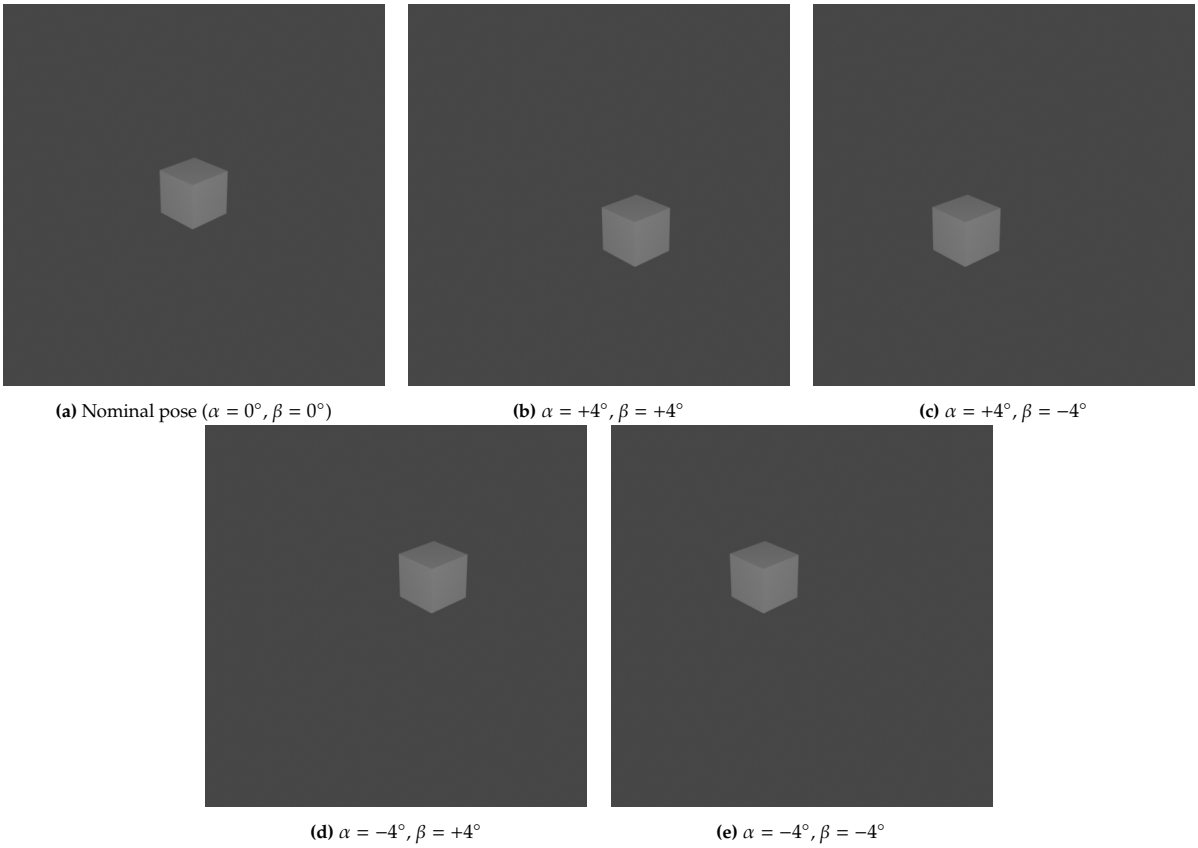
- **Input:** Blender scene (primitive cube), nominal pose (Table 6.3), and pointing errors, consisting of the different combinations between α and β.
- **Expected output:** Rendered image with the cube in the desired pose.
- **Passing criterion:** The cube in the rendered image must be in the right pose (verified through visual inspection).

The results for the different combinations of pointing errors are shown in Figure 6.6. As expected, when no pointing error is applied, the cube is perfectly centred in the image, whereas a positive α and β rotations shift the cube downward and rightward, respectively.

6.3.3. Model Reconstruction

Since no 3D keypoint model was found available for either the Tango or ENVISAT spacecraft, it was necessary to implement a model reconstruction pipeline, retrieving the 3D location of preselected keypoints from a set of images for which the pixel location of each keypoint is known. After reconstructing the models, the following test was performed to verify the correct functioning of the pipeline:

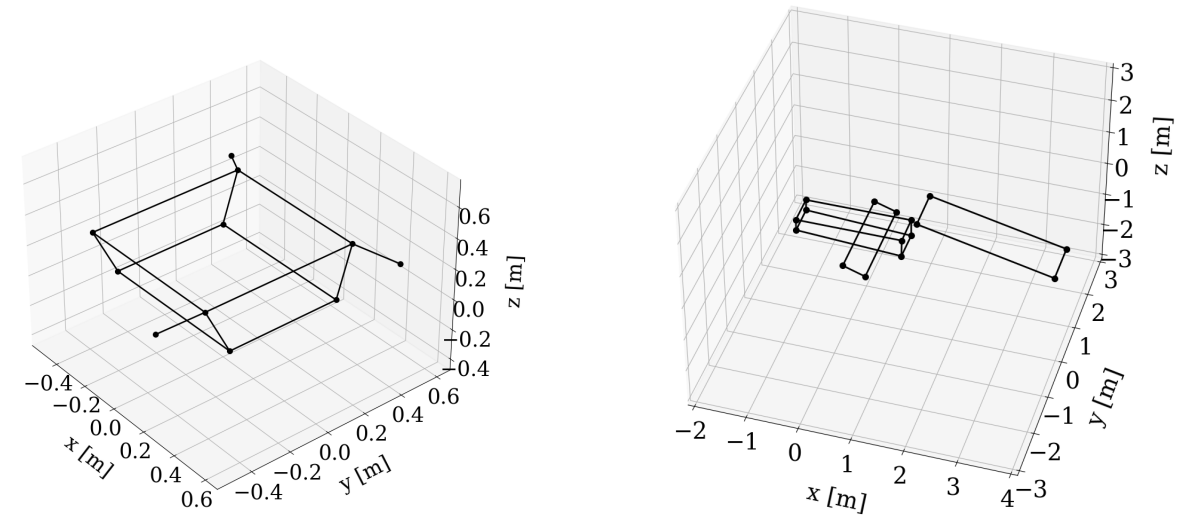
- **Input:** The pipeline requires a small set of manually labelled images to reconstruct the spacecraft model. The same images are then used to verify that the model was reconstructed correctly.
- **Expected output:** keypoint 3D coordinates in target body-fixed frame T.
- **Passing criterion:** The average reprojection error for each keypoint across the set of labelled images must be below 2 pixels.



(a) Nominal pose ($\alpha = 0°, \beta = 0°$) **(b)** $\alpha = +4°, \beta = +4°$ **(c)** $\alpha = +4°, \beta = -4°$

(d) $\alpha = -4°, \beta = +4°$ **(e)** $\alpha = -4°, \beta = -4°$

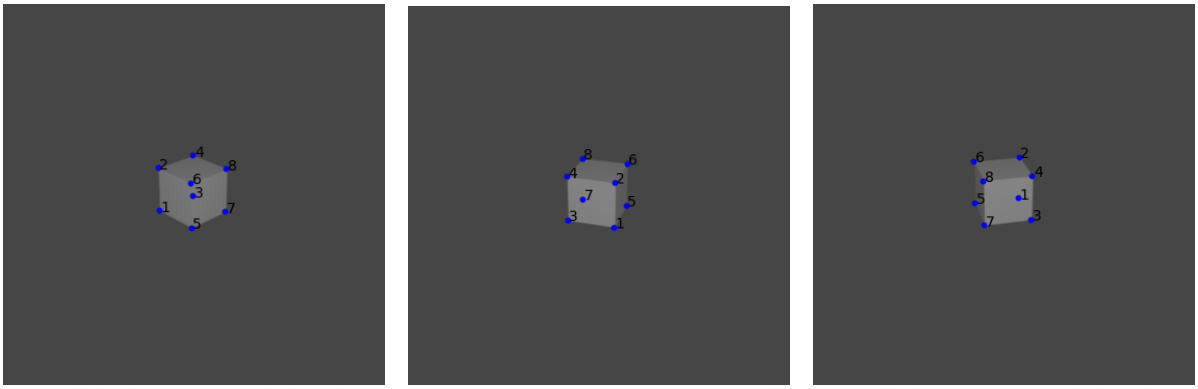
Figure 6.6: Results for the off-nominal pose generation test for the different combinations of in-plane rotation angles α and β.



(a) Tango **(b)** ENVISAT

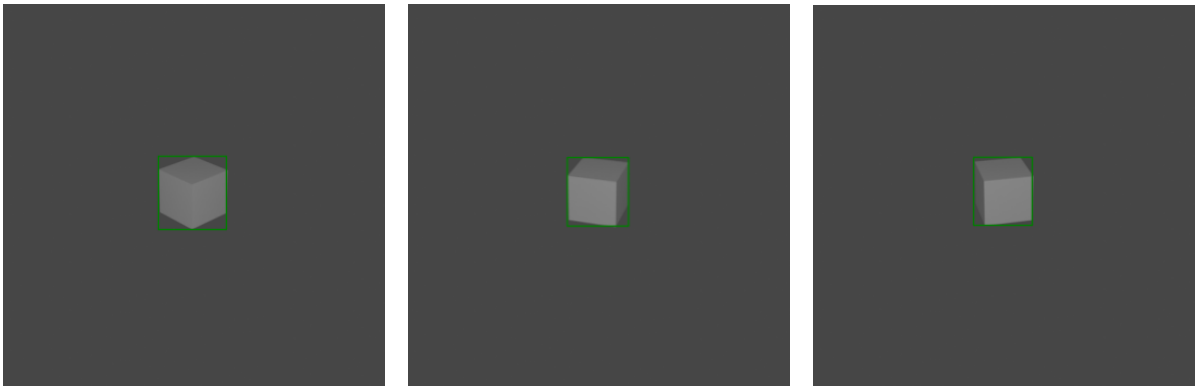
Figure 6.7: Wireframe models of the two targets. Notice how, despite Tango looking larger than ENVISAT, the axes scale is different.

Table 6.4 compares the maximum average reprojection error ϵ_{max} with the corresponding requirement. Tango and ENVISAT meet the passing requirement with a margin of 0.41 and 0.96 pixels, respectively. As an additional confirmation of the correct functioning of the reconstruction algorithm, Figure 6.7 shows both wireframe models. As can be seen from the figure, the two models resemble the original shape of their corresponding spacecraft.



(a) Nominal pose **(b)** +60° rotation **(c)** -60° rotation

Figure 6.8: Results for the keypoint label verification test.



(a) Nominal pose **(b)** +60° rotation **(c)** -60° rotation

Figure 6.9: Results for the tight bounding box label verification test.

6.3.4. Label Generation

An additional step in generating data to train an OD and KD model for spacecraft pose estimation is generating the full set of labels. While the generation of relative pose labels was already covered in Section 6.3.2, the following tests verify the correct generation of keypoint and bounding box labels.

Keypoint Labels

- **Input:** The Blender scene and the relative poses used in the pose generation test.
- **Expected output:** Rendered image with highlighted keypoint locations.
- **Passing criterion:** For each pose label, the drawn keypoint locations must visually coincide with the cube's edges.

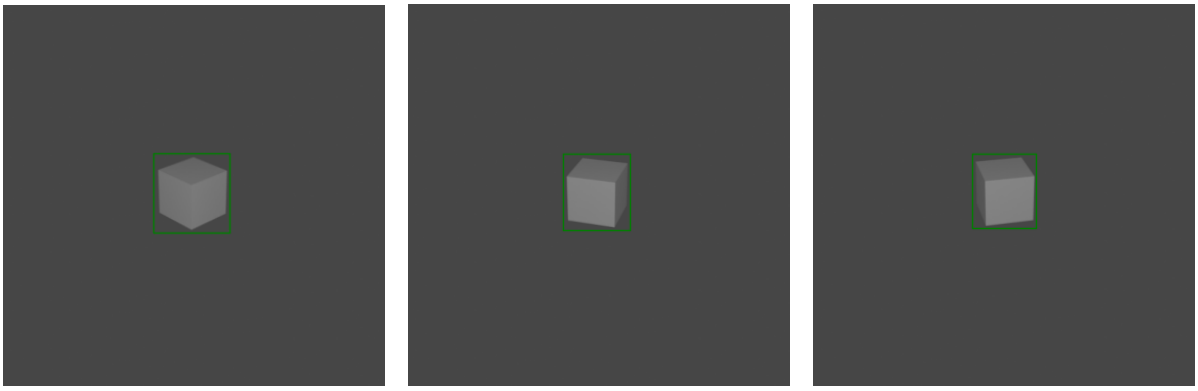
The keypoint labels are generated by the custom `project_kpts` function, which simply implements Equation (4.8) from the PnP problem. As shown in Figure 6.8, the keypoint labels are correctly applied for each pose.

Bounding Box Labels

A first test only concerns tight bounding boxes:

- **Input:** Blender scene and relative poses used in the pose generation test, along with the generated keypoint location labels.
- **Expected output:** Rendered image with the tight bounding box applied.
- **Passing criterion:** The cube must be contained inside the bounding box.

Figure 6.9 illustrates how each bounding box is correctly applied to the corresponding pose. A second test is also run on the relaxed bounding box labels, with a relaxation margin of 10%.



(a) Nominal pose **(b)** +60° rotation **(c)** -60° rotation

Figure 6.10: Results for the relaxed bounding box label verification test.

- **Input:** Blender scene and relative poses used in the pose generation test, along with the generated keypoint location labels.
- **Expected output:** Rendered image with the relaxed bounding box applied.
- **Passing criterion:** The cube must be contained with some margin inside the bounding box.

The results in Figure 6.10 show how the bounding box was correctly relaxed for each pose.

6.3.5. Machine Learning

All OD and KD models are implemented in MMDetection and Pytorch, respectively. The latter, maintained by Meta AI research is arguably the ML framework most commonly used for scientific research, whereas the former is a framework built and maintained by OpenMMLab and based on Pytorch itself. Both frameworks are open-source, with their source code stored on Github, and count numerous users every day. Given the extensive documentation and improved versions frequently released as well as that multiple OD and KD pipelines have been developed using the two frameworks, it is unlikely to find a bug in any of the functions used in this work. For this reason, it was deemed unnecessary to run tests on their correct functioning.

Object Detection

Concerning model implementation, there were only minor modifications applied to the original MMDetection architectures, mostly related to hyperparameters instead of the layer implementation. Furthermore, MMDetection has a very robust error-handling system that prevents the network from running in case any inconsistency in the architecture is found. Therefore, the correct implementation of the OD model can be verified by ensuring the training script runs and that the training converges to a reasonable accuracy on the validation set.

Keypoint Detection

A similar reasoning holds for KD. The main difference is that the KD models are not natively implemented in Pytorch. However, they come from the original paper implementations on Github. This also makes the presence of a bug very unlikely. Similarly to OD, only minor modifications to the original architectures were applied. Pytorch also has a robust error-handling system, meaning the correct implementation of the KD model can be verified by ensuring that the training job runs and converges to a reasonable accuracy on the validation set.

6.3.6. Pose Estimation

The pose solver is implemented using OpenCV. Similarly to the aforementioned libraries, it is regularly maintained and the probability of encountering a bug is quite small. The only test deemed necessary to verify the pose solver implementation consisted of using the generated keypoint location labels and the reconstructed 3D keypoint coordinates as input to the pose solver, verifying that the resulting pose corresponds to the relative pose label. The test is therefore structured as follows:

- **Input:** Keypoint labels and 3D keypoint coordinates.
- **Expected output:** Pose score statistics in `.json` format.
- **Passing criterion:** The average pose score must be zero.

The test was run for both the Tango and ENVISAT pose labels. For both scenarios, the pose score was observed to be exactly zero. Note that this is possible because the translation and rotation terms in Equation (4.12) are automatically set to zero when below the calibration threshold.

7

Object Detection

Deep neural networks find extensive use in a variety of applications, with object detection being among the most prevalent. In essence, OD can be framed as a combination of a classification and a localization task. In the past decade, significant advancements have been achieved in OD architectures, resulting in notable improvements in performance and an elevation in architectural complexity. Because of the task complexity and the generalisation capabilities required by such models, OD architectures now stand as some of the most intricate machine learning architectures.

This chapter dives into the aspects concerning the implementation of an OD network for spacecraft pose estimation. Section 7.1 introduces the current state-of-the-art in object detection and motivates the architecture choice. After that, Section 7.2 and Section 7.3 describe the model implementation and the default training settings used, respectively.

7.1. Architecture Selection

Typical OD architectures identify the object of interest within an image and provide its corresponding bounding box. The potential of CNNs for OD was initially showcased by Krizhevsky et al. (2017) at the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015), where a CNN-based architecture set a new state of the art on the ImageNet dataset. Ever since, CNNs have consistently set a new state-of-the-art in a variety of tasks, including spacecraft detection.

An OD architecture is composed of two main elements: a backbone and a detection head. A backbone is a network typically pretrained on large existing datasets, whereas a detection head learns to predict the coordinates of a bounding box and classify the object within it. A backbone commonly found in literature is ResNet, introduced by He et al. (2016) and based on a reformulation of the learning problem where the network learns a function mapping the residuals between the desired output and the input instead of the desired output itself. Besides setting a new OD benchmark on both ImageNet and COCO, the architecture has also heritage related to spacecraft pose estimation, being used by the winners of the 2021 SPEC (Park et al., 2023).

Following the trend of reducing the computational load of CNNs, A. G. Howard et al. (2017) designed the MobileNet family of architectures. Its lightweight property derives from replacing traditional convolution with depthwise separable convolution and makes it desirable for resource-constrained applications. This design philosophy results in a large reduction in computation and model size while maintaining satisfactory performance: MobileNets were able to reduce the model size from 138 to 4 million compared to the Visual Geometry Group (VGG) architecture by Simonyan et al. (2014), at the cost of a minimal accuracy reduction (from 71.5% to 70.6%) on ImageNet (A. G. Howard et al., 2017). MobileNets have been used repeatedly as a spacecraft detection backbone: Park et al. (2019) combined it with a You Only Look Once (YOLO) detection head, while Barad (2020) and Van der Heijden (2022) integrated it with a Single Shot MultiBox Detector (SSD).

More recently, Tan et al. (2019) proposed a new method to uniformly scale the network width, depth, and resolution. The author not only proves how such a careful balance, referred to as *compound scaling*, improves performance, but also introduces a new family of architectures, EfficientNets, built to maximise the benefits of compound scaling. EfficientNets are able to consistently outperform ResNet

Table 7.1: Comparison between the performance of different OD backbones on the ImageNet dataset (Tan et al., 2019).

Model	top-1% accuracy [%]	Number of parameters [Mn]	FLOPs [Bn]
EfficientNetB3	81.1	12	1.8
ResNet50	76.0	26	4.1
MobileNet	70.6	4.2	0.6

Table 7.2: Comparison between the performance of different detection heads on the Pascal VOC 2007 dataset (W. Liu et al., 2016).

Detection head	mAP [%]	FPS [-]
SSD512	76.8	19
SSD300	74.3	46
Faster R-CNN	73.2	7
YOLO	63.4	45

models with a comparable number of parameters, while their largest model, EfficientNetB7, achieves state-of-the-art performance on ImageNet despite being 8.4 times smaller and 6.1 times faster than the second best network, GPipe (Y. Huang et al., 2019). EfficientNet was deployed by Park et al. (2022b) in a multi-task network. Although not a conventional OD architecture, it relied on an EfficientNet backbone achieving promising results in spacecraft detection.

Detection heads can be divided into double-stage and single-stage detectors. Double-stage detection performs OD in two separate steps: first, the network predicts a set of possible bounding boxes, and second, each bounding box is assigned a pseudo-probability of containing the object. A commonly used double-stage detector is Region-based CNN (R-CNN) (Girshick et al., 2014), whose pipeline consists of several steps: potential bounding boxes are generated, a CNN is used to extract features and assign a score to each box and, finally, duplicates are removed through Non-Max Suppression (NMS). Each step has to be implemented and tuned independently, resulting in a longer development phase and inference time. For this reason, several modifications have been proposed to the original algorithm, such as Faster R-CNN (Ren et al., 2015), used by B. Chen et al. (2019) for spacecraft detection.

Single-stage detectors, on the other hand, perform bounding box regression and classification at the same time. YOLO is arguably the most well-known single-stage detection head. The architecture was proposed by Redmon et al. (2016) within a framework where the model outputs both the bounding box coordinates and class probability. This innovative approach was observed to bring several advantages related to detection speed and real-time applications. Another single-stage architecture, SSD, was proposed by W. Liu et al. (2016). The method is based on reducing the number of possible bounding boxes to a finite number of scales and aspect ratios, resulting in a lower complexity of the solution space, and achieves satisfactory performance on both the ImageNet and COCO datasets with reduced inference time. The two detection heads were used by Park et al. (2019) and Barad (2020) within the field of spacecraft detection.

The OD architecture used in this thesis combines an EfficientNet backbone with an SSD detection head. This choice was driven by the need for both high detection accuracy and low computational complexity. EfficientNet models have set a new standard in OD by achieving state-of-the-art performance with reduced Floating Point Operations (FLOPs), which is a measure of the network computational load.

Table 7.1 shows a comparison between the three backbone architectures considered. Although the data shown in the Table refer to image classification, they can still be considered representative of the model generalisation capabilities and complexity. The EfficientNetB3 architecture outperforms ResNet50 on ImageNet by increasing the top-1% accuracy from 76% to 81.1%, while also achieving about a 50% reduction in both the number of parameters and FLOPs. Furthermore, the accuracy difference of 11.1% compared to MobileNet achieved with only 8 million additional parameters makes it the desirable compromise between detection and runtime performance.

Table 7.2 compares several detection head architectures. The results in the table refer to an OD model with a VGG-16 backbone. SSD achieves a higher mean Average Precision (mAP), *i.e.*, the mean of the average precision scores for each class, on the Pascal VOC 2007 dataset than YOLO and Faster R-CNN. Furthermore, SSD300 and SSD512 achieve a larger number of Frames Per Second (FPS) than

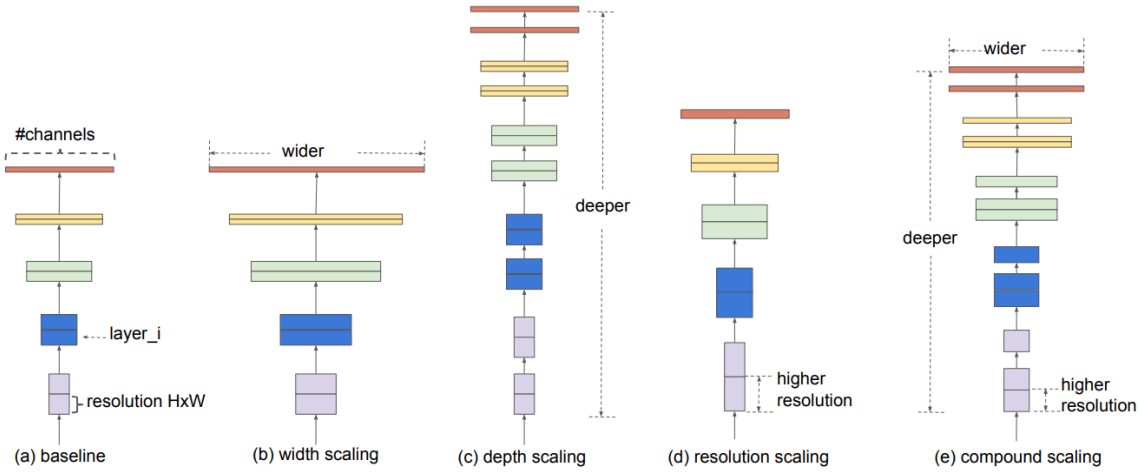


Figure 7.1: Illustration of compound scaling (Tan et al., 2019): (a) represents a baseline model, (b)-(d) show the conventional scaling strategies, where only one dimension between the network width, depth, and resolution is changed, (e) illustrates the compound scaling technique, where all three dimensions are scaled simultaneously.

YOLO and Faster R-CNN, respectively. Since SSD is a real-time detection framework, it is particularly suited for applications requiring minimal latency, such as on-orbit spacecraft detection. The SSD512 model is preferred over the SSD300 because of its greater accuracy.

Besides selecting a backbone and detection head, it has become common practice to include an additional element in OD architectures, a detection neck. This part of the network is placed in between the backbone and detection head and has the purpose of combining feature maps from different layers to allow the network to formulate a prediction based on features detected at multiple scales. This approach was proven to be more robust to image corruptions (Hendrycks et al., 2019). In its default MMDetection implementation, SSD is directly integrated with SSDNeck[1], which is kept to reduce the possibility of introducing a bug by integrating a different neck.

7.1.1. EfficientNetB3-SSD512

The model consists of a total of 10.8 million parameters, lower than the original implementation because, when used as a backbone, EfficientNet typically only uses part of the layers of the original implementation. This is because (i) the last few layers in an image classification network are task-specific and therefore only useful for classification and (ii) the feature map resolution becomes too small to extract any meaningful feature for accurate OD. This results in an overall reduction in the number of model parameters.

As already mentioned, EfficientNets have raised the standard for DL model performance while reducing the computational overhead. All EfficientNet architectures are built according to the design principle of compound scaling, that is, the network width, depth, and resolution are scaled simultaneously to maintain an optimal combination of the three dimensions. Figure 7.1 provides an illustration of the technique. Instead of modifying the architecture width, depth, and resolution separately, as shown in Figures (b)-(d), EfficientNet scales all three dimensions at the same time. This technique leads to several advantages: first, it ensures balance across the network dimensions, ensuring that resources are allocated efficiently and preventing over-provisioning in one dimension while neglecting others. Second, it prevents the network from becoming too shallow or too wide, which can lead to overfitting or underfitting.

The core of the baseline EfficientNet architecture, EfficientNetB0, consists of a sequence of seven stages using mobile inverted bottleneck convolution, introduced in the MobileNetV2 architecture (Sandler et al., 2018). An example of a bottleneck block is shown in Figure 7.2. It consists of four distinct parts: the residual connection (green), the expansion layer, depth-wise convolution, and the projection layer. On top of this configuration, (Tan et al., 2019) add a Squeeze-and-Excitation (SE) block (Hu et al., 2018). This block recalibrates channel-wise feature responses by modeling interdependencies

[1] `https://github.com/open-mmlab/mmdetection/blob/main/mmdet/models/necks/ssd_neck.py` (visited on 27/09/2023)

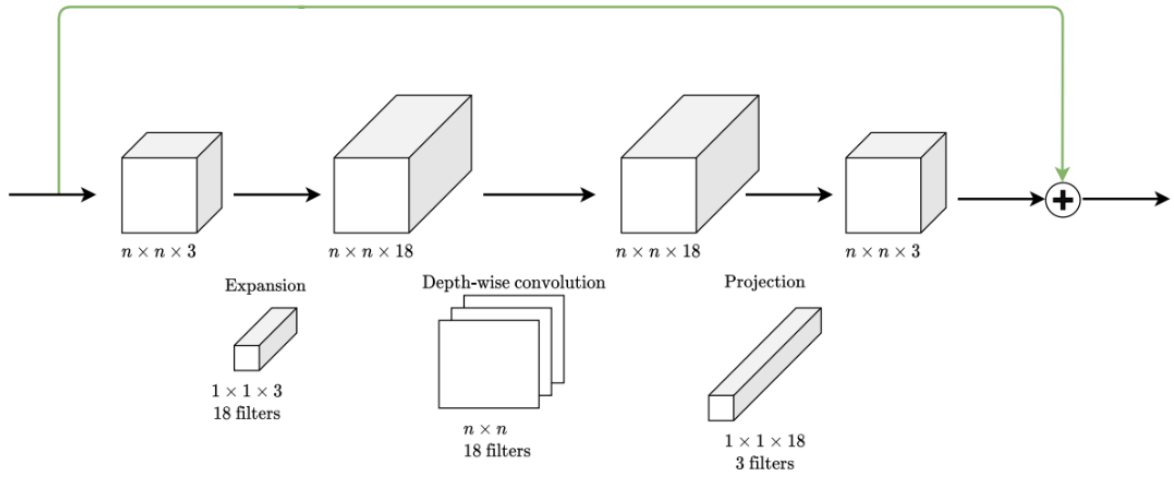


Figure 7.2: Illustration of a mobile bottleneck block (Van der Heijden, 2022).

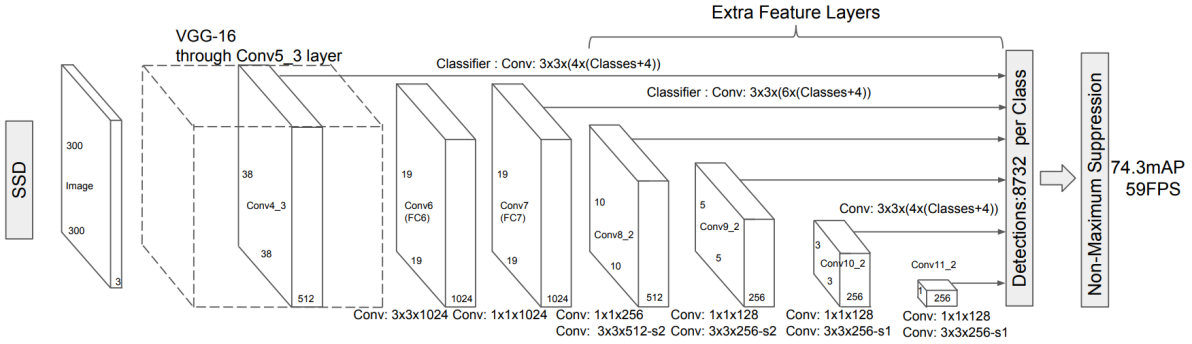


Figure 7.3: Original SSD architecture with a VGG-16 backbone (W. Liu et al., 2016).

among channels. It comprises two key operations: a global average pooling operation that captures channel-wise statistics and a set of fully connected layers that learn channel-wise scaling factors. The SE block enhances feature discrimination by focusing on informative channels and suppressing less relevant ones. The input stage is a 3×3 convolution layer, while the last stage includes a 1×1 convolution layer, followed by a pooling and an FC layer.

The original SSD implementation involves a VGG-16 backbone (W. Liu et al., 2016). An illustration of the architecture is provided in Figure 7.3. SSD simply adds multiple convolutional layers on top of the backbone architecture, gradually reducing the feature map resolution. For instance, the model proposed by W. Liu et al. (2016) shrinks the map resolution from 38×38 down to 1×1. The additional features extracted by the intermediate layers are used to generate multi-scale predictions, which allows the network to take into account representations of the image at different resolutions. SSD uses a set of predefined bounding boxes generated from a given list of scales and aspect ratios. The resulting boxes, called *anchor boxes*, are used by the network to scrutinise the image. At each resolution level, the different anchor boxes are slid through the image and assigned a confidence score for each class. Note that, as both the scale and aspect ratio are adimensional parameters, the size of an anchor box with respect to the input image changes with the feature map resolution. As shown in Figure 7.4, a set of anchor boxes applied to a feature map with 8×8 resolution (b) covers a smaller portion of the image than when applied to a feature map with lower resolution, such as 8×8 (c). Then, all the predictions are filtered using NMS to generate the final class and box location predictions. The difference between the two SSD versions from the original implementation, namely SSD300 and SSD512, lies in the resolution of the input when it is passed to the first layer of the detection head: 300×300 and 512×512 pixels, respectively.

Detection necks allow leveraging high-resolution layers from the backbone network for predictions

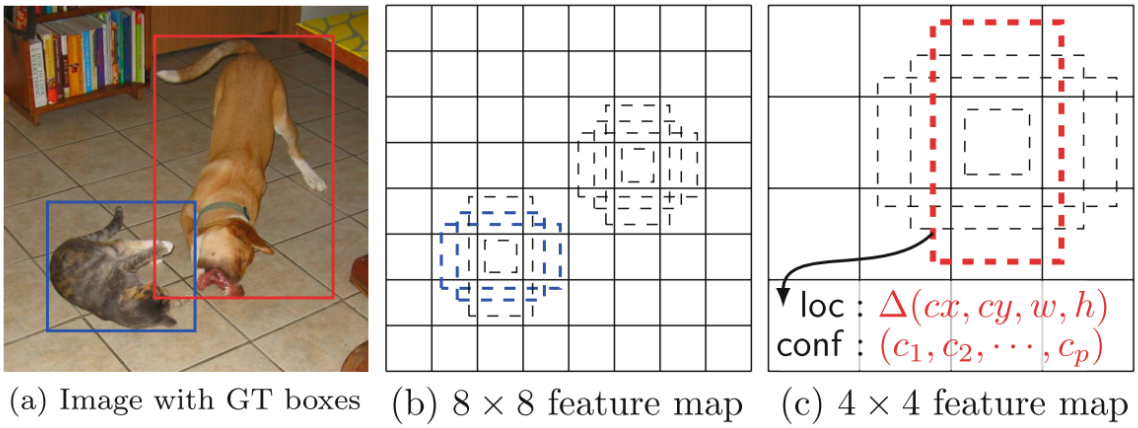


Figure 7.4: Comparison between an input image with label bounding boxes (a) and a set of anchor boxes for two different feature map resolutions (b)-(c) (W. Liu et al., 2016).

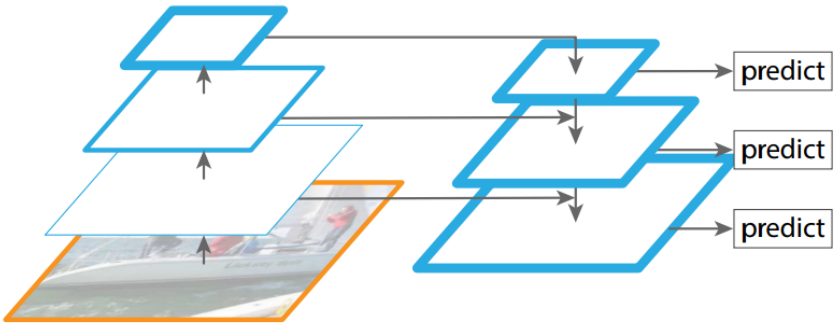


Figure 7.5: Illustration of the FPN neck architecture (Lin et al., 2017).

across all scales. For instance, the FPN neck, introduced by Lin et al. (2017), addresses this by merging low-resolution, yet semantically rich features with high-resolution, but semantically weaker ones, using a top-down architectural approach complemented by lateral connections, illustrated in Figure 7.5. In this setup, the top-down architecture enlarges the low-resolution semantically strong features to the necessary spatial dimensions, while the lateral connections incorporate feature maps of the same size from the bottom-up pathway. This strategy ensures the presence of semantically strong features across all feature map sizes, leading to more accurate feature localization. Relying solely on the top-down approach would result in semantically strong features but with imprecise locations due to the numerous down- and upsampling operations applied to these features. Similarly to this concept, the SSDNeck implemented in MMDetection adds several convolutional layers on top of the backbone network to ensure greater connections between maps at different levels and improve feature detection.

7.2. Implementation

The different parts of the models are available through MMDetection 3.0, regularly maintained and updated with new releases. The library allows the user to create a model by passing a Python configuration file. The file contains several dictionaries, one for each part of the model. Additional dataset and runtime settings can also be passed through the same configuration file.

MMDetection does not natively implement EfficientNet and SSD in the same architecture. The library integrates the backbone proposed by Tan et al. (2019) with an FPN neck and a different detection head[2], while SSD is integrated with a VGG-16 backbone, consistently with the original paper (W. Liu et al., 2016). This means that their architectures had to be adjusted and integrated, creating a new model configuration file. This led to removing the 4th backbone stage from the output for the SSD neck,

[2] `https://github.com/open-mmlab/mmdetection/blob/main/configs/efficientnet` (visited on 03/10/2023)

effectively using only the 5th and 6th stages. Furthermore, since the native SSD configuration assumes an input number of channels compatible with VGG-16, it was necessary to change the number of input and output channels of the SSD neck and head so that they would be consistent with EfficientNetB3's.

Once the new model configuration was created, some additional modifications were also applied to adapt the model to perform OD on spacecraft images. First of all, the number of classes was reduced from 80 for the COCO dataset to 1. Second, the model predictions were set so that no bounding box would fall outside of the image, with the input resolution being reduced from 896×896 to 640×640 to minimise the computational load. Third, it was chosen to reduce the number of boxes kept for NMS from 1,000 to 100, with only one being kept as the final output compared to the original 200. This was done because every image in both the SPEED+ and ENVISAT datasets contains one and only one spacecraft, while typical OD model configurations often assume the presence of multiple objects in the image. To determine the final bounding box coordinates, the model applies NMS to the set of kept boxes. As this process relies on both the confidence score and IoU, they have to be set in the model configuration. It was chosen to keep both values to default: 0.02 and 0.45, respectively. The selection can be made more strict by increasing the two parameters.

7.2.1. Input and Output

The network takes input images with size 640×640 pixels. Images are normalised using COCO statistics for consistency with the pretrained checkpoint (see Section 7.3). This step can be implemented directly in the model configuration by including a `data_preprocessor` field, which requires the channel-wise mean and standard deviation to be used for normalisation. All the other preprocessing steps - image and label loading, data augmentations, and resizing - are included in the dataset configuration through a *pipeline* field, which contains a list indicating the different input processing steps.

The network output consists of tuples including class scores and bounding box locations. As already mentioned, the model configuration was modified to prevent the final bounding box from falling outside of the image. Furthermore, during inference the number of bounding boxes is kept to 1. Both the label and predicted bounding boxes are in the format $(u_{min}, v_{min}, u_{max}, v_{max})$, which identifies a unique bounding box only using its upper-left (u_{min}, v_{min}) and bottom-right point (u_{max}, v_{max}). The coordinates are expressed in pixels with respect to the original image resolution. An illustration of a few sample predictions is shown in Figure 7.6.

7.3. Training Settings

Being a library built on top of Pytorch to facilitate setting up ML pipelines, MMDetection implements its own training and testing scripts, which are used to run training and inference, respectively. The training and testing jobs are run, as outlined in Section 6.2, using one of DelftBlue's NVIDIA Tesla V100S GPUs.

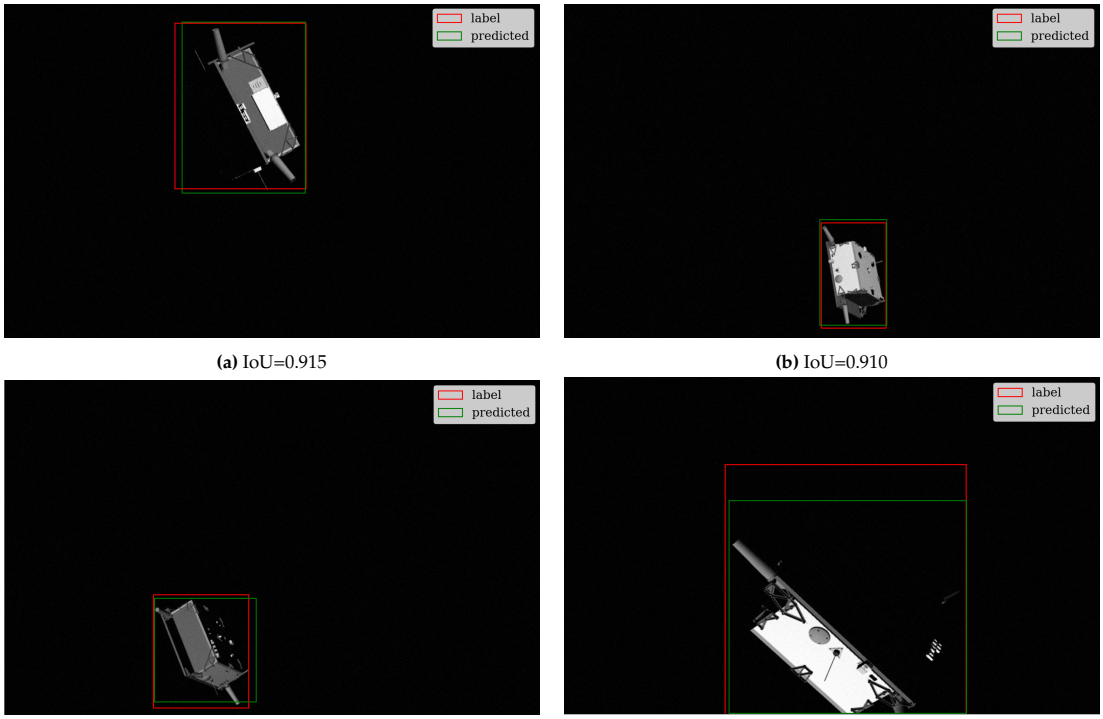
The network is trained independently on the two targets, *i.e.*, Tango and ENVISAT. As will be explained in Chapter 10, the Tango scenario allows to quantify the effect of synthetic augmentations, whereas the ENVISAT scenario is used to study the benefits of training on photorealistic images.

An overview of the training pipeline is provided by the diagram in Figure 7.7. Validation is performed at the end of every epoch, with different data about the model training, such as the training and validation loss, the evaluation metric, and the learning rate used, being saved using Tensorboard[3], a visualisation library for ML pipelines. Every 10 epochs, a model checkpoint is saved by MMDetection and, at the end of the training, the different log files and checkpoints remain stored in the working directory specified to MMDetection, as well as the best-performing model which is evaluated on the validation and testing data.

Training on a different dataset requires finetuning hyperparameters independently. The set of hyperparameters reported below are used as default and were found to work well with both datasets. The same set of hyperparameters was found to work well on the ENVISAT dataset as well, with the batch size being the only hyperparameter required to be modified. More details about the change in training settings are provided in Section 10.3.

- **Epochs:** The model is trained for 300 epochs. In addition to this, MMDetection provides an EfficientNetB3 checkpoint trained on COCO, used to initialise the backbone at the start of the training.

[3] `https://www.tensorflow.org/tensorboard` (visited on 30/09/2023)



(a) IoU=0.915 **(b)** IoU=0.910

(c) IoU=0.843 **(d)** IoU=0.838

Figure 7.6: OD prediction samples with the corresponding ground truth bounding box.

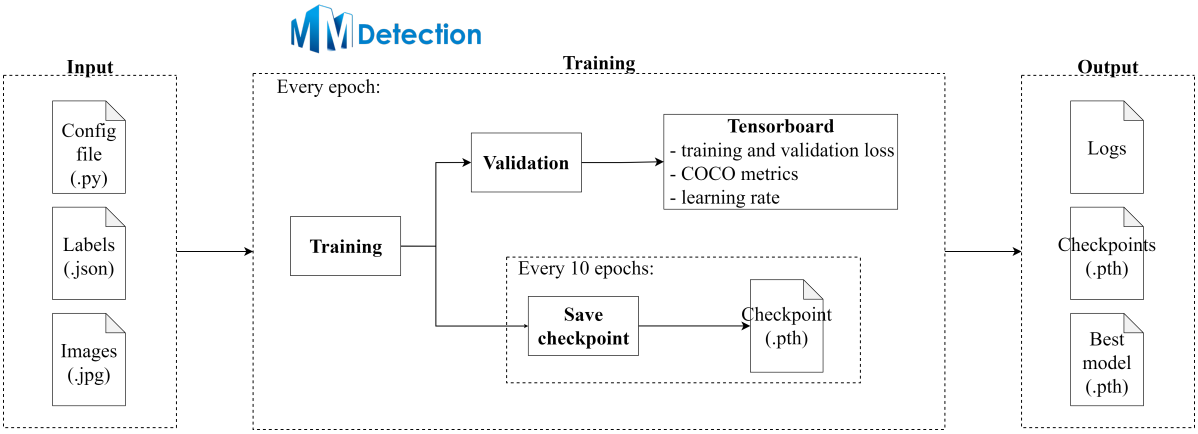


Figure 7.7: OD training pipeline.

- **Loss:** The OD loss consists of a classification and a localisation part:

$$\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{loc} \tag{7.1}$$

The classification loss is simply given by binary cross-entropy, setting the standard for image classification models:

$$\mathcal{L}_{cls} = -\hat{p} + (1 - \hat{p}) \log(\hat{p}) \tag{7.2}$$

where $\hat{p}$ is the probability of the detected object being a spacecraft. The localisation loss is given by the smooth-L1 function[4]:

$$\mathcal{L}_{loc} = \begin{cases} \frac{0.5}{\beta} \|\mathbf{y} - \hat{\mathbf{y}}\|^2, |\mathbf{y} - \hat{\mathbf{y}}| < \beta \\ |\mathbf{y} - \hat{\mathbf{y}}| - 0.5\beta, \text{otherwise} \end{cases} \tag{7.3}$$

[4] `https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html` (visited on 05/10/2023)

where $\mathbf{y}$ and $\hat{\mathbf{y}}$ indicate the predicted and ground truth bounding box coordinates, respectively, while β specifies the threshold at which the loss switch between one case and the other. For OD training, $\beta = 0.1$. The reduction from the default value of 1 in MMDetection was driven by the fact that the model had to classify only one class. This led to prioritising the localisation loss over the classification loss. By reducing the value of the parameter β, the localisation loss was observed to increase for early epochs, therefore having a greater influence on the training process.

- **Optimiser:** The training uses the Stochastic Gradient Descent (SGD) optimiser. It consists of a modification to the gradient descent algorithm introduced by Equation (5.27), by accounting for the noise contained in the training data. Similarly to the loss function being averaged over a number of samples, the gradient with respect to the network parameters is also averaged over the batch with the goal of reducing the impact of input noise in the optimisation process.

  The version of the optimiser used further modifies the update law from Equation (5.27) to account for the variation in parameters from one step to the other. The resulting parameter update step is given by:

$$\mathbf{v}_{n+1} \leftarrow \beta \mathbf{v}_n + (1-\beta) \frac{\partial J}{\partial \boldsymbol{\pi}}\bigg|_{\hat{\boldsymbol{\pi}}_n} \tag{7.4}$$

$$\hat{\boldsymbol{\pi}}_{n+1} \leftarrow \hat{\boldsymbol{\pi}}_n - \alpha \mathbf{v}_{n+1} \tag{7.5}$$

  It can be seen how the parameters are updated in two separate steps. The first one calculates the velocity term for the next step $n+1$, $\mathbf{v}_{n+1}$, given by the weighted average between the velocity at the previous step and the new gradient. The strength of each term depends on the momentum β, which can vary between 0 and 1. It was set to 0.9, allowing to take into account the gradient direction from previous steps and therefore providing a more robust learning law. The second one performs the weight update using the learning rate α.

  Finally, the weight decay introduced in Section 5.2.3 is also applied by modifying the cost function with the additional term in Equation (5.31), with the parameter λ is set to $4{\cdot}10^{-5}$.
- **Batch size:** The batch size is set to 16 for the Tango scenario and 8 for the ENVISAT one. The choice was driven by the different size of the two datasets, having a ratio of roughly 6:1. As this parameter defines the number of samples over which the loss and the gradient are averaged at each step, lowering the batch size allows to perform more steps in a single training epoch, enabling the model to be trained for a similar overall number of steps.
- **Learning-rate schedule:** The learning rate is set to an initial value of 10^{-3} and reduced by a factor of 0.8 after epochs 100, 175, 250, 280, and 295 to avoid overshooting. The same schedule was found to work well with both the Tango and ENVISAT datasets.
- **Metric:** The main metric used is IoU, given by the following expression:

$$\text{IoU} = \frac{A \cap B}{A \cup B} \tag{7.6}$$

  where A and B indicate the ground truth and predicted bounding box. IoU gives a measure of the similarity between the two boxes based on the overlapping area between the two. It can vary between 0 (no overlapping) and 1 (the two boxes coincide with each other). Because of its simplicity and interpretability, it is commonly used to evaluate OD models. A visualisation of the metric is provided in Figure 7.8. IoU is used as the main evaluation metric at both the validation and testing stages. During validation, it is used by the COCO detection metrics supported by MMDetection. This set of metrics calculates the IoU over a set of detections split according to the object size, *i.e.*, small, medium, and large, providing insights into how the model behaves for different object sizes. During testing, the best model is still tested on the COCO detection metrics, but all the predicted bounding boxes are stored in a `.json` file for further processing, such as computing the mean and median IoU over the entire dataset.

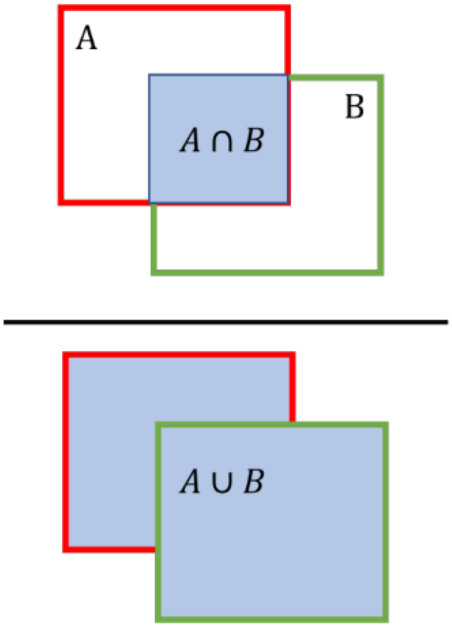


Figure 7.8: Intersection-Over-Union (Van der Heijden, 2022).

8

Keypoint Detection

As discussed in Chapter 2, recent work on spacecraft pose estimation pipelines have outlined the advantages of using keypoint detection to determine the keypoint image coordinates and pass them to a pose solver. This chapter describes the selection and implementation of a KD architecture. Section 8.1 elaborates on the architecture selection process, presenting two families of architectures that can potentially be integrated within a pose estimation system. Section 8.3 and Section 8.4 summarise the implementation details and training settings concerning the selected architectures, while Section 8.2 describes the methodology used to extract the keypoint locations from the output heatmaps.

8.1. Architecture Selection

Keypoint detection is a task where a feature extractor is required to determine the coordinates of preselected keypoints in the input image. The task has been initially applied to human pose estimation, where keypoints correspond to parts of the human body such as hands, joints, and shoulders. The work by Toshev et al. (2014) paved the way for solutions based on deep learning and ignited research on keypoint-based pose estimation. As already mentioned in Chapter 2, KD has been recently adopted within spacecraft pose estimation pipelines, with several architectures proven to be feasible alternatives to design such systems.

Similarly to OD, current state-of-the-art KD networks consist of convolutional architectures. With regards to spacecraft pose estimation, Cassinis et al. (2020) use the Hourglass architecture (Newell et al., 2016) in a relative navigation loop. The network consists of an encoder followed by a decoder, whereas pooling layers have the task of learning relevant features while upsampling layers leverage such features to introduce new information useful for the final prediction.

Novel KD models can also be obtained by modifying well-established architectures to suit them better for the task of interest. ResNet, already mentioned in the context of OD, has been leveraged extensively to develop new models. The Cascaded Pyramid Network (CPN) (Y. Chen et al., 2018), for instance, relies on ResNet-like sub-networks that downsample feature maps, followed by a refinement sub-network that creates a parallel upsampling. Another example is given by the SimpleBaseline architecture proposed by Xiao et al. (2018), whose primary novelty consists of the inclusion of deconvolution layers. The network was observed to outperform not only CPN but also the previous state-of-the-art Hourglass network.

Despite the efforts in developing new ResNet-based models, the state of the art in pose estimation remains the High-Resolution Network (HRNet) architecture (K. Sun et al., 2019), which leverages a high-feature representation across all its layers. The architecture presents also heritage in spacecraft pose estimation, as it was trained by B. Chen et al. (2019) and Cassinis et al. (2019) to perform spacecraft keypoint detection.

Given the severe constraints applied to space hardware, it is also important to consider lightweight KD architectures in the design process. Park et al. (2019) used MobileNet for the detection of spacecraft keypoints. Besides achieving a competitive score for the 2019 SPEC, the use of MobileNet allows taking advantage of the depthwise separable convolution to reduce the number of parameters. A similar computation-efficient approach was undertaken by Zhang et al. (2019) in Lightweight Pose Network (LPN), later used by Van der Heijden (2022) for asteroid landmark regression. Both MobileNet and

Table 8.1: Comparison between different KD architectures (Zhang et al., 2019) on COCO. The data refers to an input size of 256×192 pixels.

Model	AP [%]	Number of parameters [Mn]	FLOPs [Bn]
HRNet-w48	76.3	63.6	32.9
HRNet-w32	74.4	28.5	7.1
LPN152	71.0	7.4	1.8
LPN101	70.4	5.3	1.4
LPN50	69.1	2.9	1.0
CPN	68.6	27	6.2
Hourglass	67.1	25.1	19.5

LPN implement depthwise separable convolution, maintaining good performance while significantly reducing the architecture complexity.

Table 8.1 compares the different KD architectures on their Average Precision (AP) on the COCO validation set, their number of parameters and FLOPs. The two HRNet models, HRNet-w32 and HRNet-w48, outperform the other models on the COCO dataset, achieving a state-of-the-art average precision of 76.3% and 74.4%, respectively. The LPN models, however, are able to achieve greater precision than CPN and Hourglass by reducing both the number of parameters and FLOPs of up to one order of magnitude. For example, CPN reaches a 68.6% precision with 27 million parameters, while Hourglass achieves 67.1% precision with 25.1 million parameters and 19.5 billion FLOPs. LPN50 improves on their performance (69.1%) while significantly reducing the number of parameters and FLOPs to 2.9 million and 1.0 billion, respectively. The improvement in architecture complexity still holds when comparing LPN to HRNet models. Even the largest LPN network, LPN152, only counts 7.4 million parameters with 1.8 billion FLOPs, whereas HRNet-w32 displays a 3.9× increase in both the number of parameters and FLOPs, counting 28.5 million and 7.1 billion, respectively.

Overall, the comparison shows how a tradeoff can be made between the accuracy and size of different KD models. Although achieving state-of-the-art performance, HRNets require larger memory and computational power to run compared to LPNs. Because of the importance of both accuracy and power requirements in space missions, both families are considered further for a performance tradeoff. The upcoming sections describe the two architectures in greater detail.

8.1.1. High-Resolution Network

HRNet was introduced by K. Sun et al. (2019) to perform human pose estimation by predicting the 2D or 3D location of human body joints in images or videos. The key idea behind this architecture is to maintain a high-resolution representation throughout the entire network, rather than downsampling the feature maps early on as in many traditional CNN architectures. This allows HRNet to capture both fine-grained details and high-level semantic information. An overview of the architecture is shown in Figure 8.1.

The network is organized into stages, which consist of a group of parallel sub-networks across different resolution streams, while each resolution stream is formed by sets of serial sub-networks called branches. In each subsequent stage, a new branch with lower resolution is added to the network structure, allowing HRNet to avoid aggressive downsampling and capture information at multiple scales.

As layers contain information on multiple scales, the network adopts a fusion strategy to merge features from different branches (Figure 8.2), enabling the network to leverage both global and local information effectively. For a given resolution stream, the multi-scale fusion performs a strided 3×3 convolution to downsample the higher-resolution feature maps and a 1×1 convolution followed by a nearest-neighbour upsampling of the lower-resolution feature maps. In addition to multi-scale fusion, HRNet incorporates connections across parallel branches, facilitating the information flow between different scales. The final layer merges the information contained across all resolution streams in the higher-resolution branch to generate the output heatmaps.

The models initially considered were HRNet-w32 and HRNet-w48, where 32 and 48 indicate the maximum number of branches in the architecture. However, as can be seen from Table 8.1, HRNet-w48

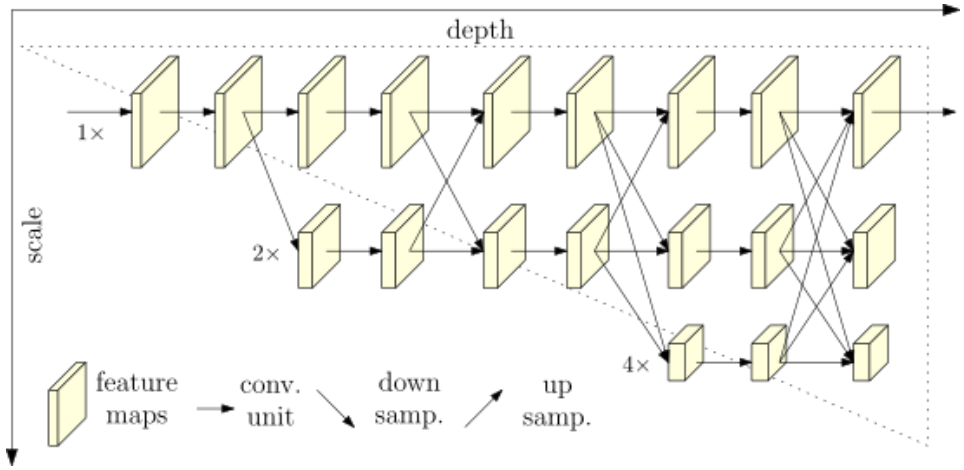


Figure 8.1: Illustration of the HRNet architecture (K. Sun et al., 2019).

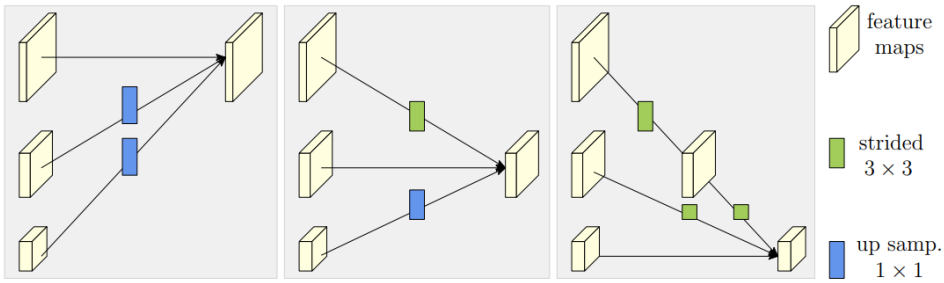


Figure 8.2: Illustration of the multi-resolution fusion in HRNet (K. Sun et al., 2019).

counts more than twice as many parameters as HRNet-w32 and a consequent 4.6× increase in FLOPs. For this reason, it was chosen to replace this architecture with a more lightweight one, HRNet-w18, which counts only 9.3 million parameters.

8.1.2. Lightweight Pose Network

The LPN architecture, introduced by Zhang et al. (2019), is illustrated in Figure 8.3. It consists of two main blocks: the first one downsamples the resolution using convolutional layers, whereas the second gradually upsamples the feature map to obtain the final heatmap representation. The feature extractor is based on the ResNet architecture, while deconvolutional layers are deployed to upscale the feature maps to the desired output size. This design choice has the purpose of maintaining simplicity while leveraging deep, low-resolution feature maps that contain rich semantic information.

The architecture simultaneously leverages the SimpleBaseline and HRNet architectures. The design follows the same principles as in Xiao et al. (2018) by using bottleneck blocks as the basic network components, redesigned to include depthwise convolution and reduce computational complexity compared to SimpleBaseline. Furthermore, such lightweight bottleneck block includes an additional Global Context (GC) block following the work by (Cao et al., 2019), which has been shown to improve performance by combining addition fusion as in X. Wang et al. (2018) maintaining a low memory footprint as in Hu et al. (2018).

However, as HRNet demonstrated the better performance attainable by maintaining a high-resolution representation throughout the network, LPN also removed the last convolutional layer and the first upsampling layer from the original SimpleBaseline implementation. This results in an architecture with a low memory footprint able to generate high-resolution heatmaps.

The models considered in this research are the same as in the paper by Zhang et al. (2019), namely LPN50, LPN101, and LPN152, which differ from each other for the SimpleBaseline architecture they leverage, based on ResNet 50, 101, and 152, respectively. It can be seen from Table 8.1 how LPNs conform more to the design principles of lightweight networks, achieving a lower model size and computational complexity than HRNets.

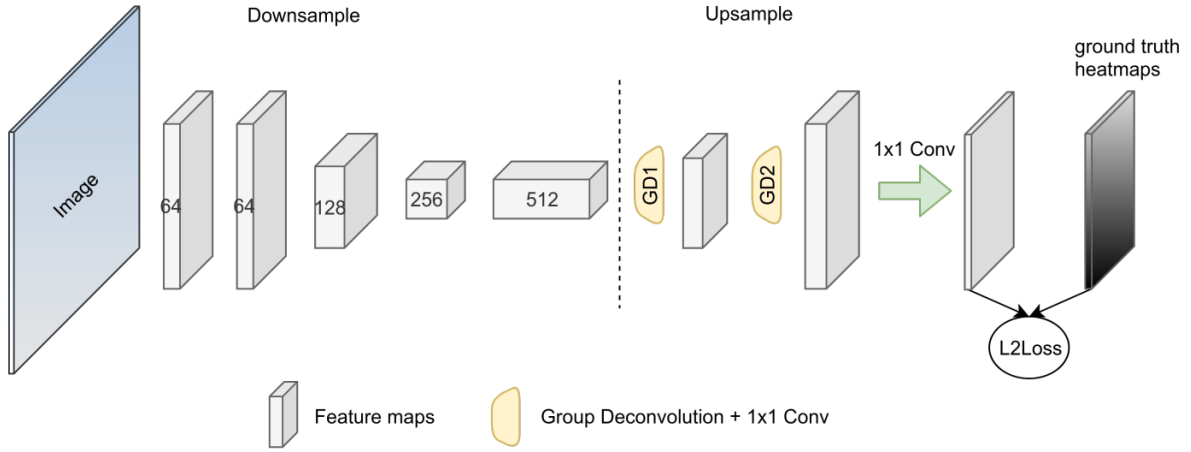


Figure 8.3: Illustration of the LPN architecture (Zhang et al., 2019).

8.2. Keypoint Location Extraction

The heatmap pixel values output by the KD network represent the pseudo-likelihood of a keypoint being located at a certain pixel location. This means that the network does not output the keypoint pixel coordinates directly, although all the relevant information are encoded in the heatmap itself. This makes it necessary to adopt a strategy to extract the keypoint location from the heatmap. To achieve this, some pipelines, such as in Cassinis et al. (2019) rely on the *argmax* algorithm, where a keypoint location is simply associated with its corresponding heatmap peak. However, this algorithm presents the drawback of not being differentiable, making it not applicable during training to extract keypoint locations, and lacking robustness, a consequence of relying on one single pixel value out of the entire heatmap. An alternative solution, deployed by X. Sun et al. (2018) and Van der Heijden (2022), involves the use of the *soft-argmax* algorithm, a modified version of argmax that makes it differentiable and applicable at the backpropagation step during training. Instead of using the original heatmap pixel values, the algorithm applies the same exponential map using Equation (4.10), extracting the keypoint location by calculating the weighted heatmap average:

$$\hat{u}_k = \frac{1}{w} \sum_{i=1}^{w} \sum_{j=1}^{h} \mathbf{S}_k(i,j) i \tag{8.1}$$

$$\hat{v}_k = \frac{1}{h} \sum_{i=1}^{w} \sum_{j=1}^{h} \mathbf{S}_k(i,j) j \tag{8.2}$$

The resulting keypoint location, $(\hat{u}_k, \hat{v}_k)$, is the one used by the other parts of the pipeline, such as calculating the loss in Equation (8.3) and reconstructing the relative pose.

8.3. Implementation

A Pytorch HRNet implementation is available on the official repository[1] of the paper by K. Sun et al. (2019), providing not only an implementation of the network from scratch, but also several pieces of code useful to implement a KD training and testing pipeline. The same repository was also leveraged by B. Chen et al. (2019) for their pose estimation pipeline, who applied the required modifications to make the code suitable for spacecraft pose estimation.

The original HRNet architecture consists of four stages, each containing one module. When passing from one stage to the next, the network is expanded by adding one branch. This baseline configuration is used for both the HRNet-w32 and HRNet-w18 models. The two models differ in the number of channels assigned to their branches. HRNet-w32 uses 32, 64, 128, and 256, whereas HRNet-w18 uses 18, 36, 72, and 144.

[1] `https://github.com/leoxiaobin/deep-high-resolution-net.pytorch.git` (visited on 12/07/2023)

Besides creating separate configurations for the two models, modifications were also done to the baseline architecture itself. Consistently with the work by B. Chen et al. (2019), the HRNet architecture used in this thesis contains a modified final layer, where the number of channels is reduced more gradually until the output layer. This led to the addition of two more convolution layers and corresponding ReLU activations. The number of output channels is dictated by the number of spacecraft keypoints to be detected, which is passed to the network through the `NUM_KEYPOINTS` configuration parameter. While the architecture by K. Sun et al. (2019) uses 17 keypoints for human pose estimation, 11 and 16 keypoints were selected for Tango and ENVISAT, respectively. More information about the keypoint selection process can be found in Section 9.1.

Furthermore, it was chosen to increase the input and output resolution to 256×256. The original input resolution, 256×192, was changed to increase the amount of information contained in the input. The output resolution was increased from 64×48 to reduce the loss in detection accuracy caused by upsampling.

Similarly to HRNet, the original LPN implementation can be found on Github[2]. The architecture used in this thesis leverages the original implementation, making several modifications necessary for the application of interest and to ensure the network runs on the target device. First, to allow for a fair comparison with HRNet architectures, the input image size is increased from 256×192 to 256×256. Second, the number of deconvolution layers is increased from 2 to 4 to generate heatmaps with the same resolution as the input. It is selected to assign each of the introduced layers 128 deconvolution kernels for a gradual transition to the final number of keypoints. Third, the above-mentioned GC block was replaced with the alternative Squeeze-and-Exitation (SE) block, reducing the block size to fit the network on Myriad X (see Section 11.3 for more details).

Because of its expected greater accuracy, HRNet-w32 is selected as the baseline KD architecture. Together with EfficientNetB3-SSD512, it constitutes the full pipeline for keypoint extraction that will be used for the experiments on synthetic augmentations and photorealistic rendering (Sections 10.2 and 10.3). The remaining KD architectures are only considered for the final architecture tradeoff in Chapter 11.

8.3.1. Input and Output

Before being given as input to the network, the input image and corresponding keypoint location label are preprocessed. First, the image is loaded and converted from greyscale to colored to be compatible with the network. Second, the labels are loaded: the ground truth bounding box is relaxed and used to crop the input image, which is then resized to the target input resolution (256×256), while the keypoint coordinates are transformed with respect to the pixel reference frame of the new image. Third, the visibility labels are generated by comparing the keypoint locations with the image size, making sure they fall within the image.

The network output consists of a batch of heatmaps, one for each keypoint, with a resolution of 256×256 pixels. The keypoint location is then extracted using the same procedure as in Section 8.2. Note that the output coordinates are provided relative to the heatmap reference frame. This convention is kept when testing the KD models to ensure comparability between bounding boxes with different size, but has to be dropped when performing pose estimation, since the pose solvers listed in Section 4.3 require the keypoint coordinates to be provided with respect to the original image reference frame.

8.4. Training Settings

As previously mentioned, the KD training and testing pipelines were implemented in Pytorch leveraging the work by K. Sun et al. (2019) and B. Chen et al. (2019). An overview of the training pipeline can be found in Figure 8.5. It is conceptually similar to the OD training pipeline shown in Figure 7.7, with the main difference being that a checkpoint is saved after every epoch instead of 10. This is done because MMDetection saves checkpoints as separate files, meaning storing a new checkpoint every epoch would result in a large memory footprint, while the pipeline implemented for KD allows overwriting the checkpoint file from the previous epoch. Besides that, other minor differences are also present, such as the fact that the configuration file is in `.yaml` format, and are due to the specifics of the implementation. As for OD, the best model is evaluated on the validation and testing data at the end of the training.

The hyperparameters listed below were found to work better on the HRNet models. They were

[2] `https://github.com/zhang943/lpn-pytorch.git` (visited on 12/07/2023)

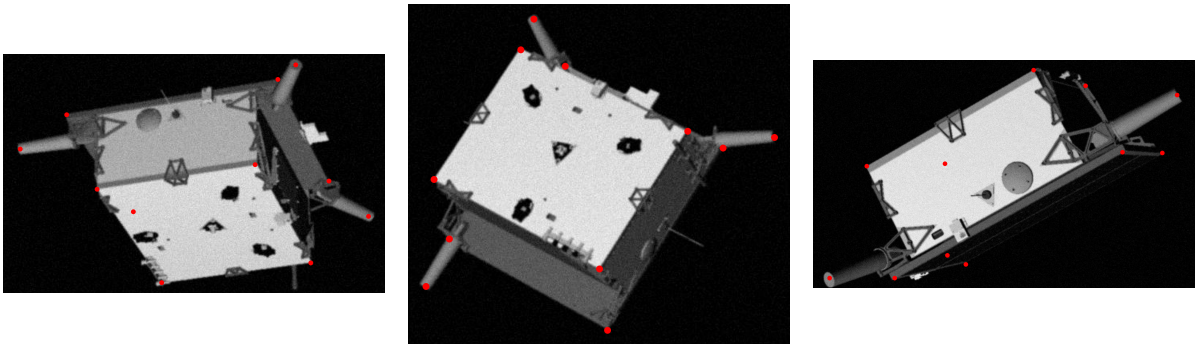


Figure 8.4: KD prediction samples.

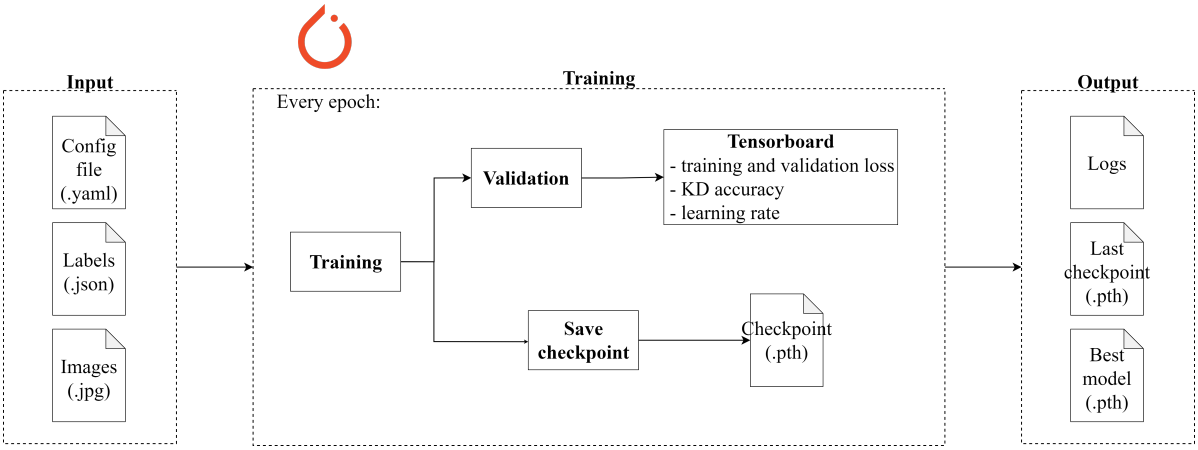


Figure 8.5: KD training pipeline.

also used as a starting point for LPNs, however, the models ended up not being trained after the poor inference time performance observed on Myriad X (see Chapter 11).

- **Epochs:** Like the OD model, the number of epochs was set to 300. An HRNet-w32 checkpoint is available through the official Github repository, while for HRNet-w18 a checkpoint is available through the repository for HRNet image classification[3]. The same chaekpoint was used by B. Chen et al. (2019) to initialise their OD model. HRNet-w32 is pretrained on COCO, HRNet-w18 on ImageNet. LPN checkpoints are available through the official Github repository. All models are pretrained on COCO.
- **Loss:** The loss consists of the average RMSE calculated over all visible keypoints N_{vis} in the image:

$$\mathcal{L} = \frac{1}{N_{vis}} \sum_{i=1}^{N_{vis}} \|\mathbf{p}_i - \hat{\mathbf{p}}_i\| \tag{8.3}$$

  where $\mathbf{p}_i$ and $\hat{\mathbf{p}}_i$ are the extracted and label keypoint pixel locations, respectively, normalised by the heatmap size. The keypoint locations are extracted using the procedure outlined in Section 8.2.
- **Optimiser:** Similarly to OD, SGD is also used for KD training. The optimiser also uses a momentum coefficient of 0.9 and a weight decay coefficient of $5{\cdot}10^{-5}$. This makes the update law the same as in Equation (7.5), with the additional term in Equation (5.31) to account for weight decay.
- **Batch size:** The batch size is set to 32 for the Tango scenario and 16 for the ENVISAT scenario. The rationale behind this decision is the same as for OD: reducing the batch size increases the overall number of training steps, compensating for the difference in size between the SPEED+ and ENVISAT datasets.

[3] `https://github.com/HRNet/HRNet-Image-Classification.git` (visited on 05/10/2023)

- **Learning-rate schedule:** The learning rate follows a one-cycle policy. This policy was initially introduced by Smith et al. (2019) and consists of changing the learning rate during training following a cyclical schedule. The rate is set to an initial value (10^{-3} in this case) and goes through a *warm-up* phase where it gradually increases until it reaches its maximum value. Next, the learning rate is gradually decreased up to its final value. As a result, the learning rate goes through an entire cycle during the training process. The learning rate can be changed following a linear or a cosine strategy. The former changes the learning rate with the same rate throughout the process, whereas the latter allows the learning rate to change more rapidly at the beginning and the end of the training. The scheduler used in this work sets the maximum learning rate to 10^{-2}, found to be the maximum value for which the training process remains stable, and the length of the warm-up phase to 5% of the training process. It uses a cosine annealing strategy to let the rate change with a variable speed, helping the model not to get stuck in a local minimum. The same strategy was adopted by Van der Heijden (2022), achieving good accuracy on the validation and test sets.

  Overall, this technique brings several advantages. First, the initial learning rate is smaller than for the rest of the training. This is particularly beneficial when using pretrained models, making the training more stable. Second, the increase in the learning rate allows the model to get out of possible local minima. Third, the reduction at the end of the training enables the model to find the best possible fit for the data.
- **Metric:** The metric used for validation and testing is a simple detection accuracy. A detection is considered correct if the predicted location is distant from the ground-truth location no more than 5% of the image size. Again, the predicted location is extracted using the *soft-argmax* algorithm introduced in Section 8.2.

9 Datasets

The scope of this chapter is to present the key components of the datasets used in this thesis. First, Section 9.1 provides an overview of the model reconstruction pipeline developed in this work. Second, Sections 9.2 and 9.3 introduce the reader to the SPEED+ and ENVISAT datasets, respectively.

9.1. Keypoint Model Reconstruction

As for neither Tango nor ENVISAT a keypoint model was available, it was necessary to develop a pipeline that would generate a 3D keypoint model for both spacecraft. The resulting workflow, illustrated in Figure 9.1, is presented in this section. It follows the same procedure as in B. Chen et al. (2019) and Park et al. (2019):

1. **Image selection:** While images of the Tango spacecraft could already be selected from the SPEED+ dataset, there were no images available of the ENVISAT textured model. To overcome this, a small number of scenes were rendered using the model along with the actual camera parameters. The poses were generated by placing the camera at a constant distance from the satellite and varying the satellite's Euler angles with a sampling of 90° from its initial orientation. These settings were chosen to ensure the scenes would capture the spacecraft from different angles and show every part of the spacecraft several times. Furthermore, to facilitate the manual labeling step (see step 2) all images were rendered as coloured while Blender's illumination settings were kept to default. A few samples of the synthetic ENVISAT images used for model reconstruction are shown in Figure 9.2. Overall, 10 and 13 images ended up being used for reconstructing Tango's and ENVISAT's models, respectively.
2. **Manual labeling:** For both spacecraft, labels were applied so that they would correspond to strong visual features, such as edges or antennas. This was done not only to minimise the error caused by applying labels manually, but also to ensure that the selected keypoints could be detected more easily by the network. This step made use of MATLAB's Image Labeler. The labels would then be saved to a `.mat` file to be used by the model reconstruction script.
3. **Model reconstruction:** After applying the labels, a MATLAB script was run to finally reconstruct the spacecraft keypoint model. This was done using the `triangulateMultiview` MATLAB routine, already used in the work by B. Chen et al. (2019) to reconstruct Tango's 3D model, which optimises the following objective:

$$\min_{\{\mathbf{x}\}_{i=1}^{N}} \sum_{i,j}^{M} \|\mathbf{z}_{i,j} - \boldsymbol{\phi}_j(\mathbf{x}_i)\|^2 \tag{9.1}$$

where N and M are the number of keypoints and images used, respectively, and $\mathbf{x}_i$ is the vector representing the coordinates of the i-th keypoint in the spacecraft 3D model. $\mathbf{z}_{i,j}$ indicates the image coordinates of the i-th keypoint in the j-th image, while ϕ_j is a function that maps the keypoint model coordinates to image coordinates depending on the relative pose between the camera and the target in image j. It is essentially given by the same expression as Equation (4.9).

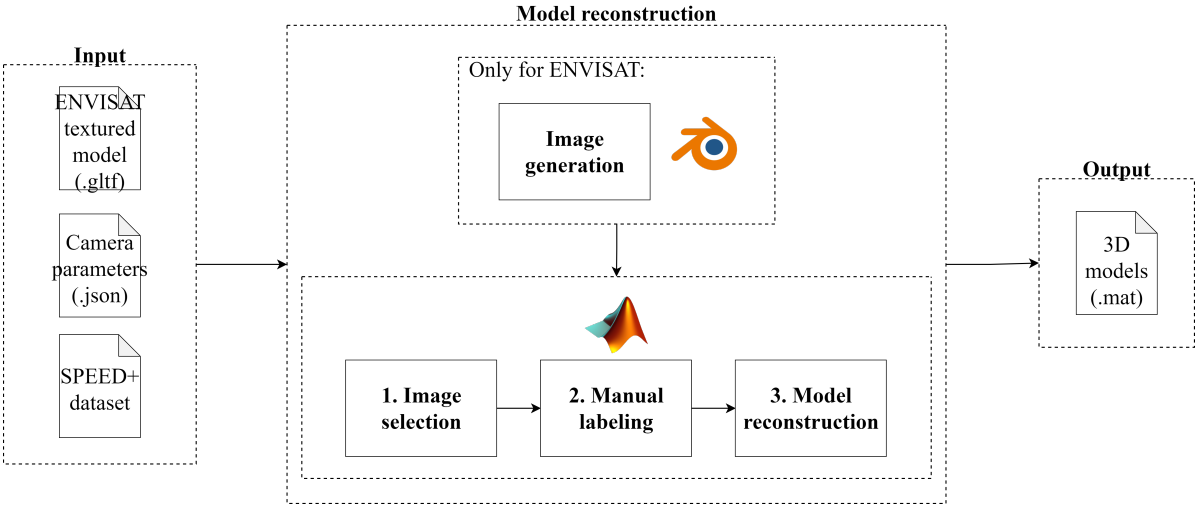


Figure 9.1: Overview of the model reconstruction pipeline.

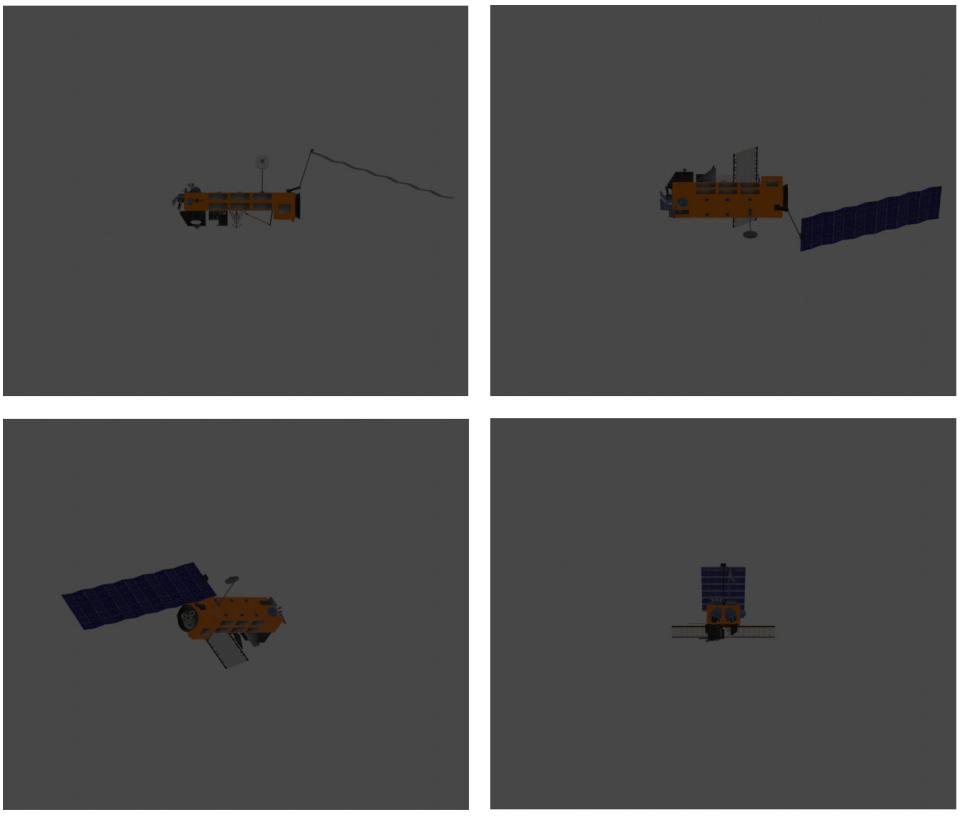
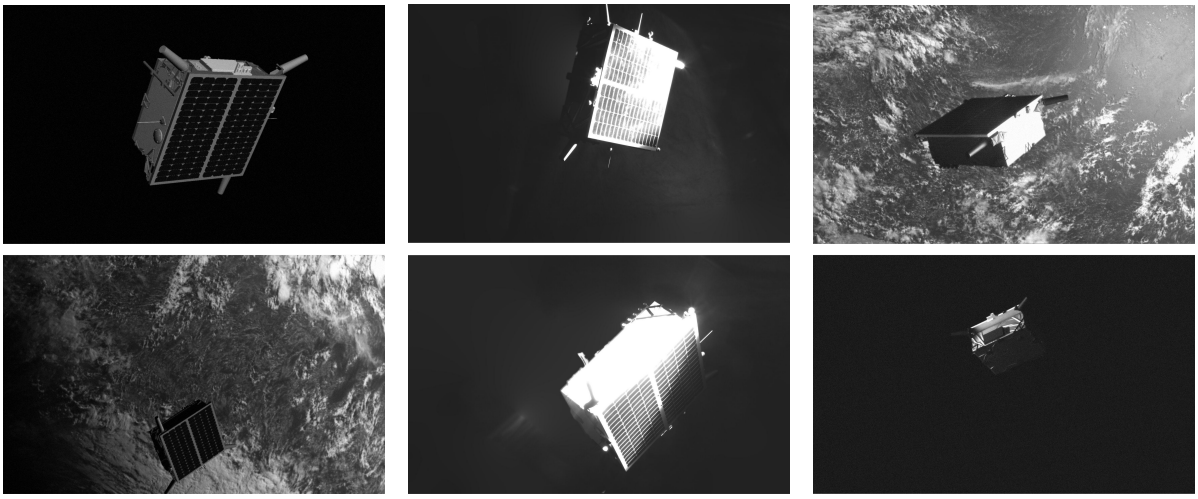


Figure 9.2: Samples of the ENVISAT images generated for model reconstruction.

The equation above leads to the solution minimising the average keypoint reprojection error across all images.

9.2. SPEED+

The SPEED+ dataset represents an evolution of its predecessor SPEED. The SPEED+ dataset includes a larger number of both synthetic and testbed images. The dataset counts 59,960 images of the Tango spacecraft rendered in OpenGL and more than 9,000 testbed images collected using the TRON facility. Besides increasing the dataset size, SPEED+ improves the accuracy of pose labels and ensures higher



(a) `synthetic` **(b)** `sunlamp` **(c)** `lightbox`

Figure 9.3: Examples of synthetic, `sunlamp`, and `lightbox` images from the SPEED+ dataset.

Table 9.1: Camera parameters used to generate the SPEED+ dataset (Park et al., 2022c) and the ENVISAT dataset introduced in this work.

Parameter	Value	
	SPEED+	ENVISAT
Number of horizontal pixels	1920	1024
Number of vertical pixels	1200	860
Focal length [mm]	17.513	39
Pixel size [μm]	5.86	8.25
Radial distortion parameters	[-0.2238, 0.5141, -0.1312]	[0, 0, 0]
Tangential distortion parameters ($\times 10^{-4}$)	[-6.650, -2.140]	[0, 0]

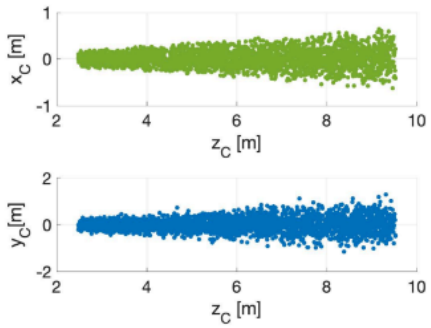
variability in illumination conditions. The testbed set is split into a `sunlamp` and a `lightbox` subset. The `sunlamp` subset contains 2,791 images reproducing the effect of direct sunlight illumination, while the `lightbox` set contains 6,740 images aimed at reproducing realistic albedo illumination. Sample images for each image domain can be seen in Figure 9.3. The dataset was created and made publicly available for the 2021 SPEC[1]. SPEED+ pose labels are continuous with millimetre position and sub-degree orientation accuracy at close range. The synthetic set contains images with a relative distance uniformly distributed between 2.25 m and 10 m and a broad range of relative orientations. The testbed set was instead created by using 10,000 different pose labels. As shown in Figures 9.4a and 9.5a, the in-plane relative position components, x_C and y_C, are uncorrelated to the line-of-sight component z_C, ensuring a uniform coverage of the different relative position labels within the given distance range. With the same scope of ensuring appropriate coverage of the different relative poses, the rotation angles around Tango's x and z axis, θ_x and θ_z, follow a uniform distribution between -180° and +180°, while θ_y follows a normal distribution between -90° and +90°, as shown in Figures 9.4b and 9.5b.

The synthetic images were post-processed taking into account TRON's calibrated camera parameters, which are listed in Table 9.1. Furthermore, the images from the synthetic and `lightbox` domains are randomly applied the Earth or a starfield in the background, along with random noise and blur. The `sunlamp` images, on the other hand, are only processed to remove excessive blooming from the images.

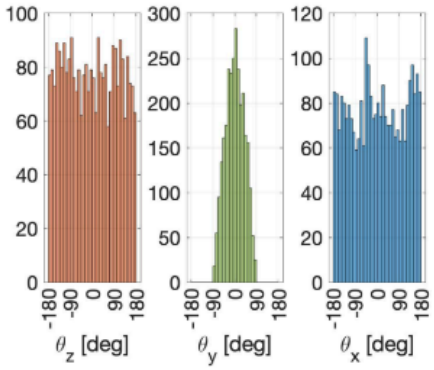
9.2.1. Synthetic Augmentations

The use of synthetic augmentations allows to include several effects and corruptions observed on real space imagery. Including such effects was proven to be necessary by Hendrycks et al. (2019), who showed how models trained only on uncorrupted images could not generalise properly once tested on real, corrupted data.

[1] `https://kelvins.esa.int/pose-estimation-2021/data/` (visited on 10/01/2023)

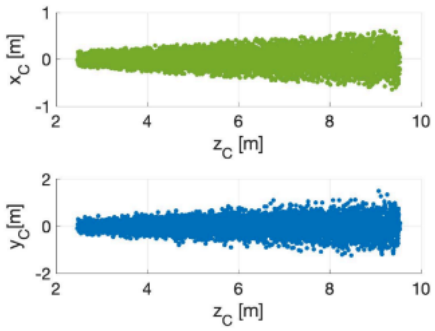


(a) Correlation between the line-of-sight position component z_C and the other position components.

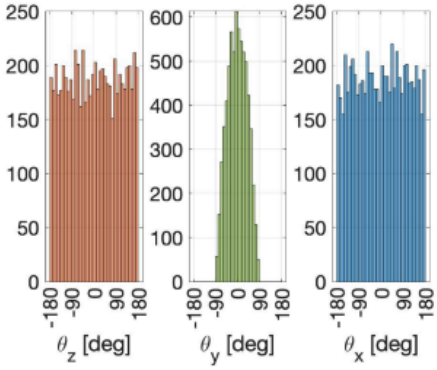


(b) Euler angles statistical distribution.

Figure 9.4: Pose label distribution for SPEED+'s `sunlamp` images (Park et al., 2022c).



(a) Correlation between the line-of-sight position component z_C and the other position components.



(b) Euler angles statistical distribution.

Figure 9.5: Pose label distribution for SPEED+'s `lightbox` images (Park et al., 2022c).

Within the field of spacecraft pose estimation, several augmentation pipelines have been used to increase the performance of models on testbed data. They include classic augmentations, such as Gaussian blur and noise, as well as domain-specific augmentations, specifically designed to replicate the quality degradation a space camera is subject to. Park et al. (2022b) set up an extensive augmentation pipeline that remarkably improved the model performance on testbed data. Additionally, Cassinis (2022) performed an analysis on the impact of domain-specific augmentations on the performance of CNN-based systems, gathering important insights on how to improve the model performance on adverse illumination conditions. Lastly, the winning team on the `lightbox` category of the 2021 SPEC (Park et al., 2023) implemented their own customed augmentations for spacecraft detection[2], namely random noise, haze, stars, and streaks. The existing research suggests how common augmentations, such as random noise, blur, and brightness and contrast, are used frequently, while novel augmentations, such as style augmentation and random Sun flare (Park et al., 2022b; Cassinis, 2022), that have been proven to enhance performance are used more rarely.

The analysis of synthetic augmentations based on the work by Hendrycks et al. (2019) and the latest progress in spacecraft pose estimation led to the selection of the following augmentation to be applied to SPEED+:

- **Blur:** The implementation in this thesis is the same as in Park et al. (2022b), where different types of blur are included. The Gaussian and median blur (Figures 9.6b and 9.6c) simulates effects related to the depth of field, while motion blur (Figure 9.6d) accounts for quick movements of the target relative to the camera when the picture is taken. Every time blur is applied to an image, one of the three augmentations is selected with equal probability.

[2] `https://github.com/mohsij/pose-estimation-augmentations.git` (visited on 12/09/2023)

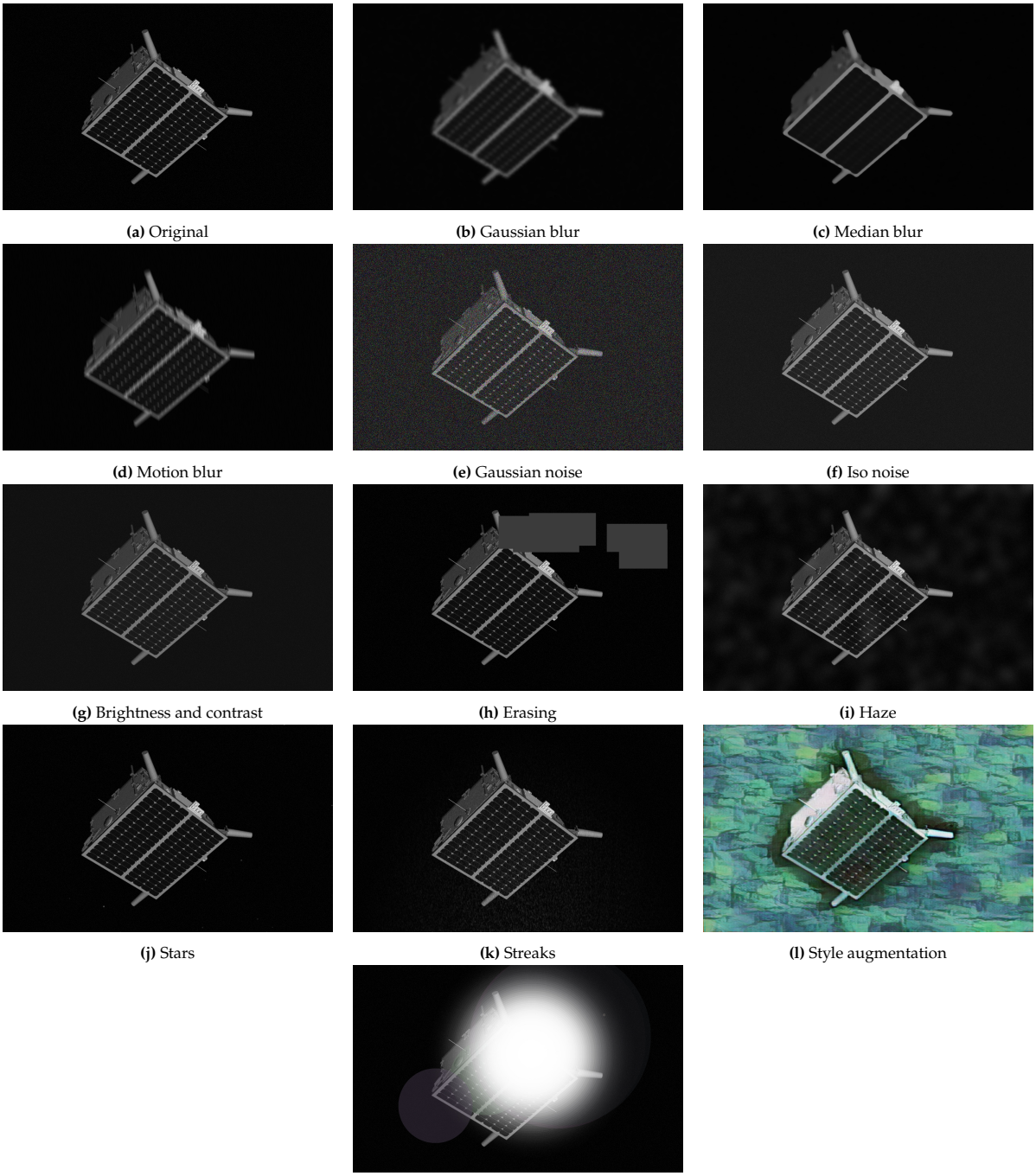
- **Noise:** The noise augmentation aims at simulating image corruptions due to non-ideal behaviour of the sensor or observation conditions. This work leverages the implementation of Park et al. (2022b). Isotropic noise (Figure 9.6f) consists of a random noise applied homogeneously to the entire image, whereas Gaussian noise (Figure 9.6e) is a statistical noise applied sampling from a normal distribution and is caused by a high sensor temperature or poor illumination conditions.
- **Brightness and contrast:** This augmentation, shown in Figure 9.6g, randomly changes the level of brightness and contrast in the image, capturing variable observation conditions, as well as under- and overexposure. It is implemented directly using the Albumentations library[3].
- **Erasing:** It simulates spacecraft occlusions due to adverse illumination (see Figure 9.6h). It was already applied in other pipelines (Barad, 2020; Van der Heijden, 2022; Park et al., 2022b). This work makes direct use of the implementation from Park et al. (2022b), who modified the Albumentation's one to only erase part of the spacecraft bounding box.
- **Haze:** Shown in Figure 9.6i, it simulates haze-like light observed in images with direct sunlight illumination (Park et al., 2023).
- **Stars:** Simulate the star field seen in `lightbox` images (Park et al., 2023). An example is provided in Figure 9.6j
- **Streaks:** Simulate light streaks observed in images with direct sunlight illumination (Park et al., 2023), as shown in Figure 9.6k
- **Style augmentation:** This augmentation randomises the style of the image. It does not simulate one effect in particular, but it was found to improve performance by forcing the network to learn the spacecraft shape rather than merely relying just on its texture (Geirhos et al., 2018; Park et al., 2019; Park et al., 2022b). The pipeline implementation is taken by Jackson et al. (2019) and is available on Github[4]. It consists of a pre-trained Neural Style Transfer (NST) architecture applying a random style embedding to the original image. The process can be described by the following equation:
$$z = \alpha \mathcal{N}(\mu, \sigma^2) + (1 - \alpha)\mathcal{P}(c) \tag{9.2}$$
where $\mathcal{N}$ represents the random distribution of the style embedding applied by the network, with μ and σ^2 being its mean and variance, $\mathcal{P}(c)$ is the style embedding of the original image, and α indicates the strength of the random sampling. In this work, α is set to 0.5 in accordance with Park et al. (2022b). A sample image augmented by the style transfer network is provided in Figure 9.6l.
- **Sun flare:** Simulates corruptions due to reflected sunlight (Figure 9.6m), such as adverse illumination and overexposure Park et al. (2022b) and Cassinis (2022). The implementation by Park et al. (2022b) is used in this work, based directly on Albumentation with the only difference being that the Sun flare is applied inside the spacecraft bounding box.

This list of augmentations replicates numerous effects witnessed in actual space imagery. Consequently, their application has the goal of bridging the gap between synthetic and testbed data by including each effect independently. To the author's knowledge, while it is possible to find examples of the application of each augmentation in the literature, no other analysis encompasses as many effects. The sole comparable instance is given by Cassinis (2022), which includes only a subset of the aforementioned list. A visualisation of the different augmentations is provided in Figure 9.6.

9.3. ENVISAT

The need for testing the robustness of the designed pose estimation system on a different target and investigating the effects of training on photorealistic synthetic data made it necessary to use a different dataset for the ENVISAT scenario. Although a synthetic ENVISAT dataset was already created by Cassinis (2022), such a dataset is neither open-source nor created using photorealistic rendering settings. Therefore, a choice was made to generate a novel synthetic ENVISAT dataset on which to train the OD and KD baseline models and test the pose estimation performance of the system. The different steps of the data generation pipeline will be described hereafter, providing an overview of the process and discussing the main design choices.

[3] `https://github.com/albumentations-team/albumentations.git` (visited on 19/09/2023)
[4] `https://github.com/philipjackson/style-augmentation` (visited on 13/09/2023)



(a) Original **(b)** Gaussian blur **(c)** Median blur

(d) Motion blur **(e)** Gaussian noise **(f)** Iso noise

(g) Brightness and contrast **(h)** Erasing **(i)** Haze

(j) Stars **(k)** Streaks **(l)** Style augmentation

(m) Sun flare

Figure 9.6: Overview of the different augmentations. Some corruptions have been amplified for illustrative purposes.

The dataset was generated assuming a Prosilica GC2450 monocular camera[5], like Cassinis (2022). The camera parameters used are shown in Table 9.1. The only difference compared to the real camera is the pixel resolution, which was reduced from 2448×2050 to 1024×860 to reduce the dataset memory footprint while minimising the loss of information due to excessive downsampling. The target resolution was chosen maintaining the same aspect ratio. Furthermore, since no testbed images were generated and Blender does not include it by default, there was no need to include camera calibration data, resulting in all distortion parameters being set to zero.

Prior to the generation of any data, an ENVISAT keypoint model was reconstructed using a textured

[5] `https://www.alliedvision.com/en/camera-selector/detail/prosilica-gc/2450` (visited on 31/07/2023)

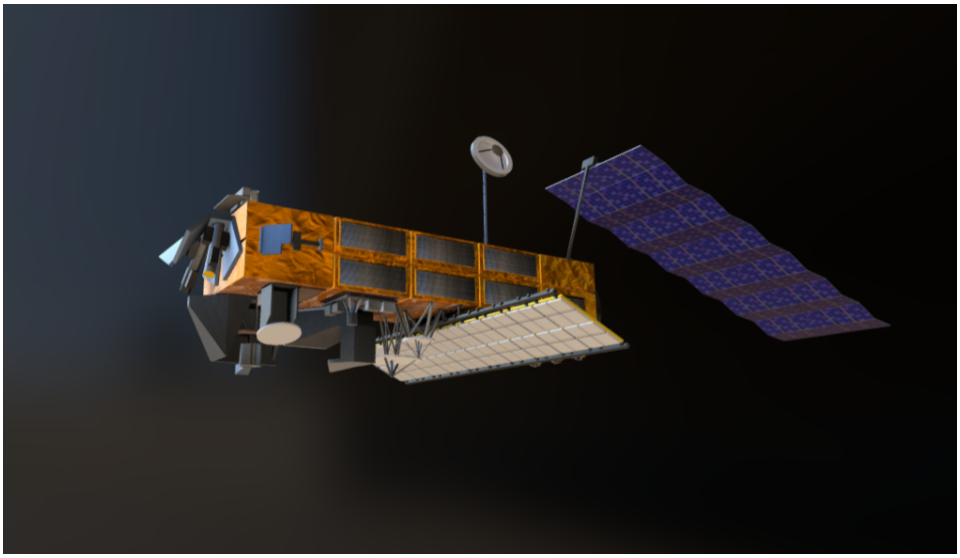


Figure 9.7: ENVISAT Sketchfab model.

model of the spacecraft. The model, shown in Figure 9.7, is available on Sketchfab[6] and is open-access. It includes every major geometrical feature of the satellite, such as the solar panel, the on-board instrumentation, and the antennas, as well as textures replicating the satellite surface material properties. The model file could be easily imported to Blender by downloading the file in `.gltf` format, which is directly supported. As discussed in Section 9.1, the model was used to generate a set of ENVISAT images with default illumination settings to reconstruct the spacecraft keypoint model.

Notice that, since most data about ENVISAT are proprietary, the open-source textured model is not as high-fidelity as the Airbus one used by Cassinis (2022) for the generation of his synthetic dataset. In particular, the two models were observed to differ from each other in terms of their overall size (the open-source model roughly being five times smaller than the Airbus one) and some details in the texture. From the perspective of the OD and KD models, this is not expected to affect the outcome of the experiments. In fact, adverse illumination conditions should be realistically simulated by a rendering software as long as the type of material, *e.g.*, metallic or non-metallic, does not change. However, before testing the pipeline on real ENVISAT images, the models should be trained on images including a more accurate model.

The block diagram in Figure 9.8 illustrates the different parts of the data generation process once the keypoint model is available. The data generation pipeline consisted of the following steps:

1. **Generating background images:** In this step, real Earth images are processed to generate background images that can be applied to part of the synthetic ENVISAT dataset.
2. **Rendering:** A script loads the textured model and camera parameters, creates a Blender scene with the camera and target pose with respect to the Blender's world frame, sets the illumination conditions and randomly samples the images to which the Earth in the background has to be applied. This is the stage where the pose labels are generated and stored.
3. **Applying background:** The images selected in the previous step, saved in a temporary directory, are processed to include the background images generated in step 1.
4. **Applying noise and blur:** Finally, all synthetic images are further processed to include random noise and blur.

The final dataset consists of 21,000 synthetic images divided into three different domains: `basic`, `realistic`, and `test`. The split between training, validation, and testing for each domain is reported in Table 9.2. Both the `basic` and `realistic` domains contain 10,000 images randomly divided into training and validation images with a ratio of roughly 90:10, while the `test` set includes 1,000 images.

The major difference between the `basic` and `realistic` domains is the illumination settings used at the rendering stage. The `basic` domain consists of images rendered with minimal changes to Blender's

[6] `https://sketchfab.com/3d-models/envisat-65b0ec49681a44f68dfc8bd4efe95839` (visited on 18/09/2023)

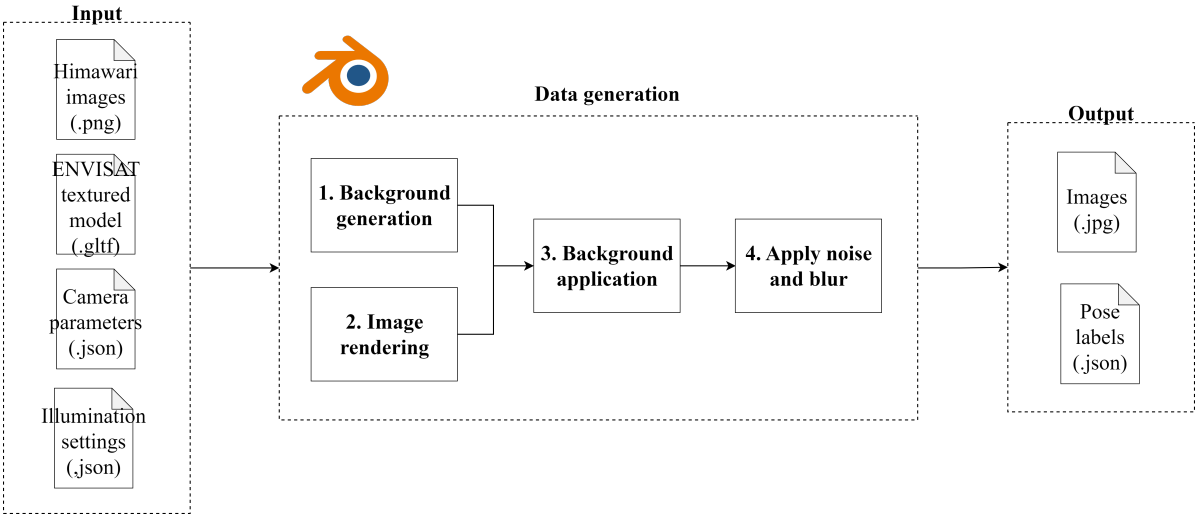


Figure 9.8: Overview of the ENVISAT dataset generation pipeline.

Table 9.2: Overview of the number of images, split between training, validation, and testing, for the ENVISAT dataset.

Split	Domain		
	`basic`	`realistic`	`test`
training	9,024	9,024	-
validation	976	976	-
testing	-	-	1,000
total	10,000	10,000	1,000

illumination settings, whereas the `realistic` images were generated by changing a higher number of options in Blender to obtain more realistic data. All the other dataset properties, such as the nominal pose and pointing error distributions, the background probability, and the blur and noise distributions are kept unchanged.

Furthermore, as the same sequence of random seeds is used, the two datasets have identical properties: from the relative pose labels to the number of images in the validation set, and from the images augmented with the Earth in the background to the level of noise and blur in every image. In other words, two images with the same filename belonging to the `basic` and `realistic` set represent the spacecraft in the same pose, with the same background conditions, and with the same level of noise and blur. The only difference between the two images is given by the illumination conditions. Some examples of corresponding images of the `basic` and `realistic` sets are given in Figure 9.9. By doing so, it is possible to remove differences in performance due to any factor other than the discrepancies in illumination conditions and quantify the impact of training a model on photorealistic data.

The `test` set was rendered independently with more extreme illumination settings and a higher background probability compared to the `basic` and `realistic` sets. This allows to test and analyse the generalisation capabilities of a models when trained on the `basic` set compared to when trained on the `realistic` set.

9.3.1. Data Generation

This section delves into the main design choices underpinning the dataset creation, as well as provides insights about the generated images and the resulting data distribution.

Background Application

The Earth images used to generate background patches are captured by the Himawari 8 meteorological satellite (Bessho et al., 2016) and are available through the Data Integration and Analysis System (DIAS) platform[7]. A total of 73 full-disk images collected within a timespan of 24 hours were downloaded to include a broad range of Earth illumination conditions.

[7] `https://diasjp.net` (visited on 18/09/2023)

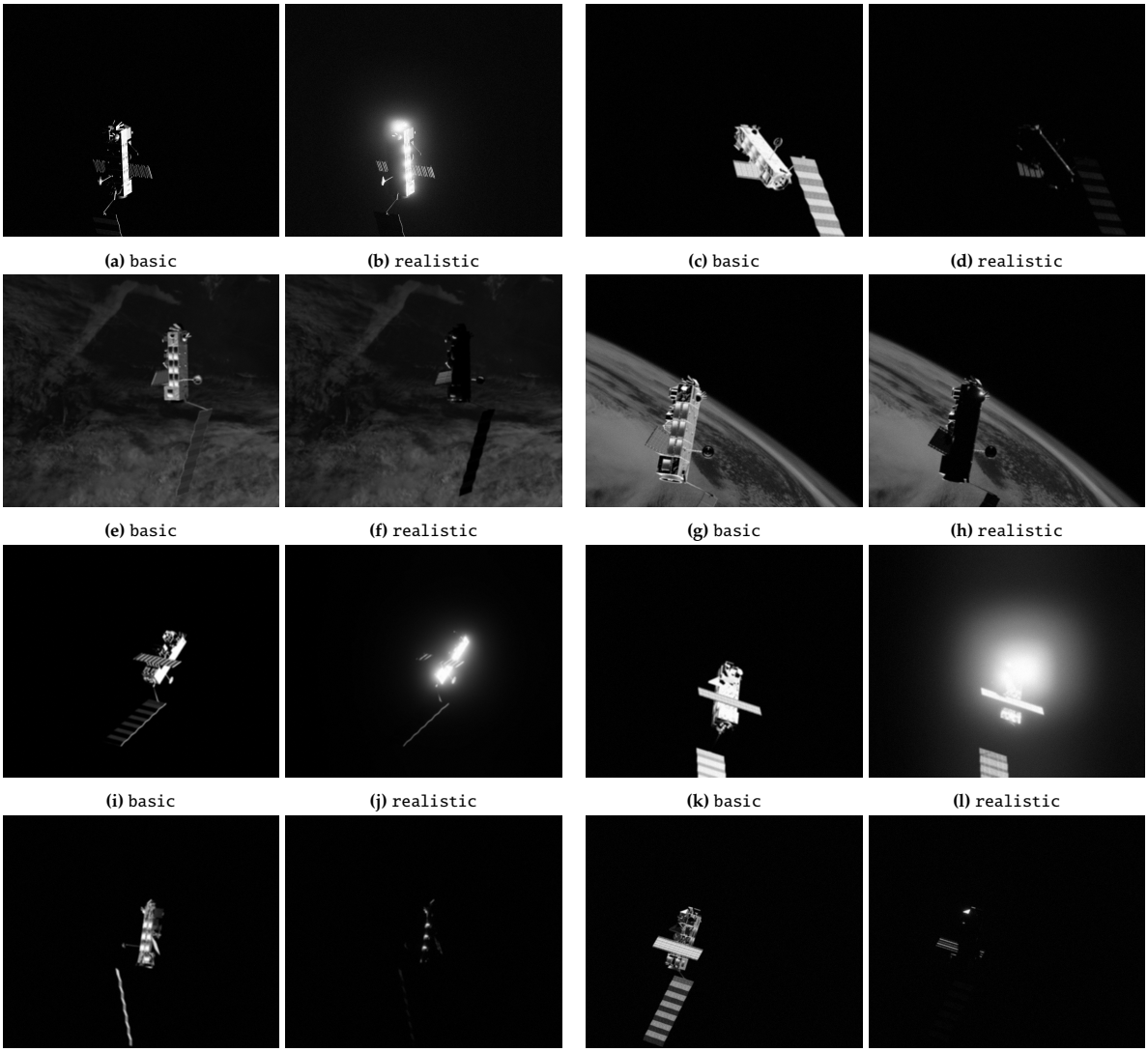
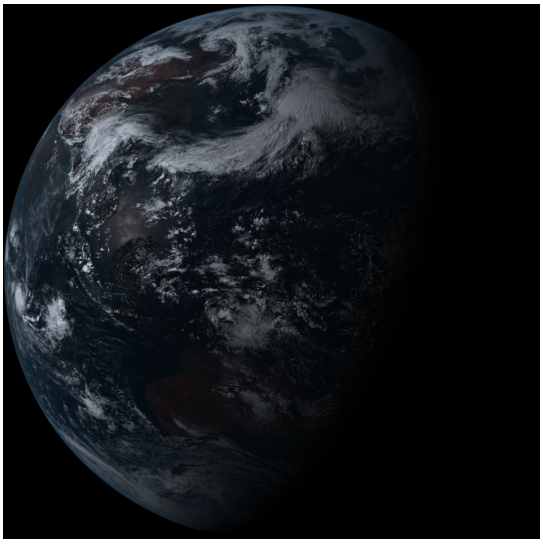


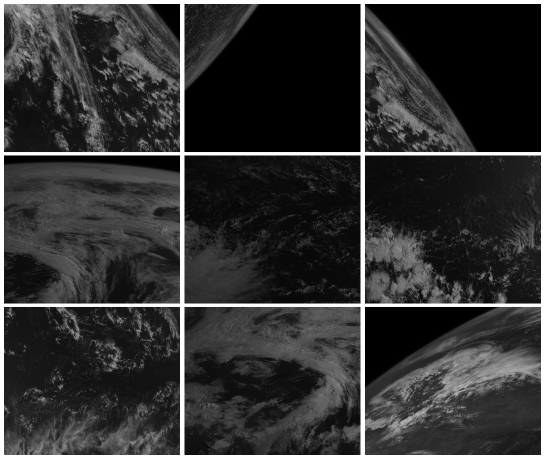
Figure 9.9: Comparison between corresponding images of the ENVISAT `basic` and `realistic` sets.

The background images were generated by randomly cropping the full-disk images to simulate observation conditions consistent with ENVISAT's orbital altitude and the camera used. After that, the images were resized to the target camera resolution. Figure 9.10 shows a comparison between one of the original full-disk images (Figure 9.10a) and a few examples from the generated backgrounds (Figure 9.10b). It can be seen how the original and processed images seem collected by two different imaging systems orbiting at different altitudes. In fact, Himawari 8 and ENVISAT orbit the Earth in Geostationary Earth Orbit (GEO) and LEO, respectively, the Himawari camera has different parameters than those listed in Table 9.1, and the generated images are stored as coloured, while they are converted to grayscale for the ENVISAT dataset using Equation (4.1).

The generated backgrounds were applied to the rendered ENVISAT images with a 50% probability to the `basic` and `realistic` set, and 60% probability to the `test` set. The motivation behind such a difference is to include more challenging observation conditions, given the issues discussed in Chapter 2 encountered by CNNs in dealing with Earth backgrounds. Whenever a rendered scene is selected to include the Earth, the background is set to transparent and the resulting image is saved in a temporary folder. To save an image with a transparent background, Blender adds an additional transparency channel which serves as a mask mapping every background pixel to zero. The mask is then used to



(a) Full-disk image.



(b) Generated backgrounds.

Figure 9.10: Comparison between an original full-disk image collected by Himawari 8 (left) and a few background images generated for the ENVISAT dataset (right).

merge the background and foreground together using the following equation:

$$I(i,j) = I_f(i,j) + [255 - M(i,j)]I_b(i,j) \tag{9.3}$$

where I_f, I_b, and I indicate the foreground, background, and final image, respectively. M is the transparency mask generated by Blender, and (i,j) are the pixel coordinates. Notice that no mask is applied to the foreground in Equation (9.3). This is done because the transparency mask does not consider the light reflected by ENVISAT as part of the foreground, which would result in the features due to adverse illumination being limited within the spacecraft edges, hence losing realism. Furthermore, whenever an image is sampled to have the Earth applied in the background, the default background intensity in Blender is set to zero. This makes all background pixel values irrelevant for the final image, and only the spacecraft and bright pixels have a visible impact on the image.

An overview of the background application process is provided in Figure 9.11.

Illumination Conditions

As mentioned before, the ENVISAT dataset has been generated to test the impact of the illumination settings used to generate training data on the model performance. The illumination conditions included in the ENVISAT dataset replicate the effect of direct sunlight illumination, the same simulated by SPEED+ `sunlamp` images. To quantify the effect of training a model on data simulating illumination conditions more accurately, it was deemed necessary to generate two datasets simulating adverse illumination conditions differently. For this reason, two separate domains, namely `basic` and `realistic`, were created using different illumination settings in Blender.

Blender allows to customise the illumination conditions in a variable level of detail depending on the application of interest. To allow a comparison between images rendered with basic illumination settings and images rendered with more `realistic` settings, it was chosen to change the light settings in Blender affecting the quality and realism of the rendered images the most. However, to avoid generating images that are completely unrealistic and too far from SPEED+'s synthetic set, some settings for the `basic` set were changed as well from Blender's default ones. Overall, the following settings were updated:

- **Illumination type:** This setting defines the type of light source simulated by Blender. The images for each domain were generated with a sunlight illumination, aimed at simulating a light source infinitely distant from the scene, such as the Sun.
- **Light properties:** Several light properties were changed from their default value in Blender. The energy was set to 1366 W/m^2, consistently with the Earth solar constant (Park et al., 2022c). The other settings changed were the diffuse and specular factors. These were kept to default for

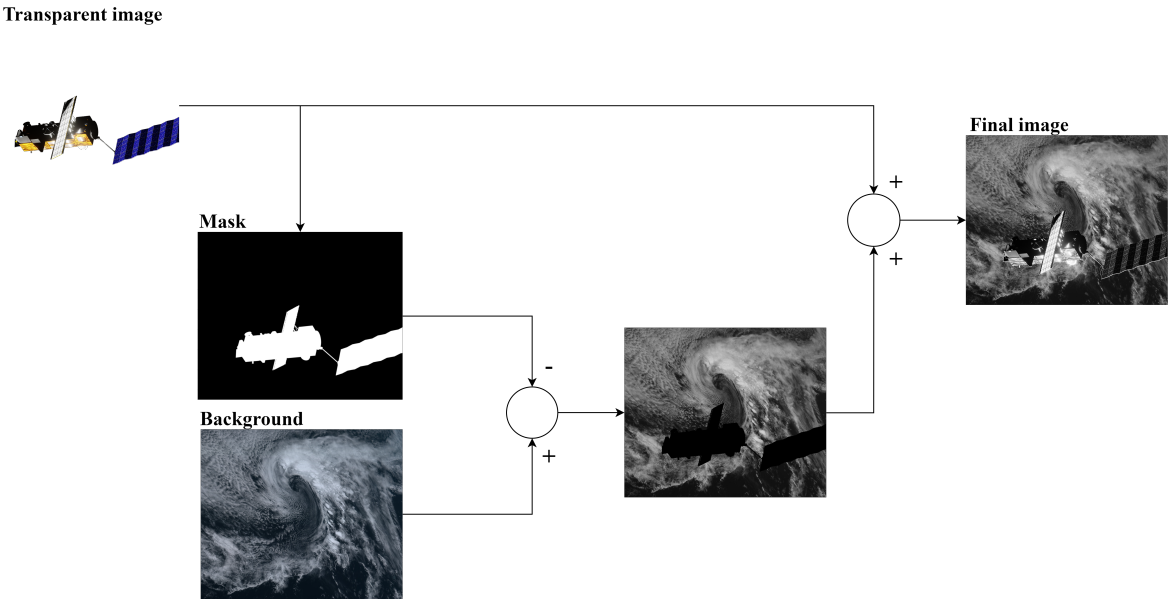


Figure 9.11: Overview of the background application process for the ENVISAT dataset.

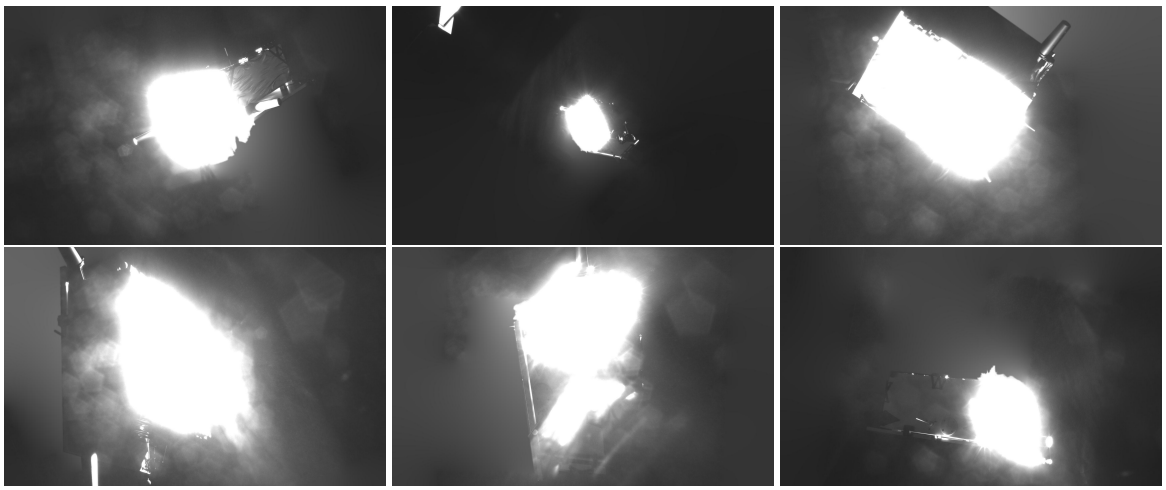


Figure 9.12: Images from the SPEED+ `sunlamp` set showing the blooming effect.

the `basic` images (0 and 1, respectively), and changed to 0.01 and 0.1, respectively, for both the `realistic` and `test` sets.

- **Shadows:** This setting is used to include realistic shadows in the images. It was set to false for the `basic` set - a choice influenced by the lack of realistic shadows in SPEED+'s synthetic images - and to true for the `realistic` and `test` sets.
- **Bloom:** This setting can be activated to include a blooming effect for adverse illumination settings. As this effect was observed in SPEED+'s `sunlamp` images (a few examples of which are reported in Figure 9.12) and was observed to have a strong impact on the realism of the rendering results, it has been included in the `realistic` and `test` sets and not in the `basic` set.
- **Contrast:** The low quality of raw space imagery often results in high-contrast pictures. While this effect is not included in the low-fidelity, `basic` images, the contrast for the `realistic` and `test` images is set to Blender's maximum level.
- **Exposure range:** Prior to running the data generation pipeline, Blender's exposure value was observed to significantly affect the rendering outcome. The resulting illumination in the rendered image is strongly dependent on this parameter. This is why it was selected to vary the exposure range across the three domains. The `basic` images were generated with the default value of 1.

Table 9.3: Overview of the Blender illumination settings used to generate each domain of the ENVISAT dataset.

Setting	Domain		
	`basic`	`realistic`	`test`
illumination type	sunlight	sunlight	sunlight
energy [W/m^2]	1366	1366	1366
diffuse factor	0	0.01	0.01
specular factor	1	0.1	0.1
shadows	False	True	True
bloom	False	True	True
contrast	"medium"	"very-high"	"very-high"
exposure range	$[1, 1]$	$[-2, 2]$	$[-3, 3]$
background colour range	$[0, 0]$	$[0.001, 0.003]$	$[0.001, 0.003]$

For the `realistic` images, the exposure was sampled from a uniform distribution between -2 and 2, whereas the `test` images were rendered with a random exposure uniformly distributed between -3 and 3. It was deemed necessary to diversify the exposure range between the `realistic` and `test` sets so that insights could be drawn about the network generalisation on unseen, more extreme illumination conditions. This was already discussed in Chapter 2 as one of the main reasons for performance degradation in CNN-based pipelines.

- **Background colour range:** One of the most noticeable differences observed between SPEED+'s synthetic and testbed data was the different level of contrast between the spacecraft and the background (Figure 9.3). The uniform background colour observed in the synthetic set of SPEED+ led to the use of a completely dark background in the `basic` set whereas, to increase the realism of the generated data, it was chosen to randomly sample the background intensity from a uniform distribution between 0.001 and 0.003 for the `realistic` and `test` images.

The values used for each setting in each domain of the ENVISAT dataset are summarised in Table 9.3. As shown in Figure 9.9, the different settings affect the features of the output images substantially. In some cases, the change in illumination settings leads to typical effects of overexposure, such as brighter areas of saturated pixels (see Figures 9.9i and 9.9k), or in other cases causes some parts of the spacecraft to be subject to underexposure. Sometimes, this effect is so considerable that radically changes the appearance of the image, with entire parts of the target perfectly visible becoming barely illuminated, such as in Figures 9.9c and 9.9e.

Nominal Poses

For each rendered image, the nominal pose label is generated by sampling the relative position and attitude from the following distributions:

- Since the nominal pose is defined with the chaser camera pointing towards the target's CoM, the nominal relative position label is given by the following vector in its most general form:

$$\mathbf{r}_{T/C}^{C} = \begin{pmatrix} 0 \\ 0 \\ z_{T/C}^{C} \end{pmatrix} \tag{9.4}$$

where, similarly to Park et al. (2022c), $z_{T/C}^{C}$ is sampled from a uniform distribution between 25 m and 50 m to ensure that the OD and KD networks would be trained on data covering the full operating range:

$$z_{T/C}^{C} \sim \mathcal{U}(25, 50) \text{ m} \tag{9.5}$$

- The nominal relative attitude label is generated following a similar reasoning. The three Euler angles, ϕ, θ, and ψ are sampled independently from a uniform distribution:

$$\phi \sim \mathcal{U}(0, 360)^\circ \tag{9.6}$$
$$\theta \sim \mathcal{U}(0, 180)^\circ \tag{9.7}$$
$$\psi \sim \mathcal{U}(0, 360)^\circ \tag{9.8}$$

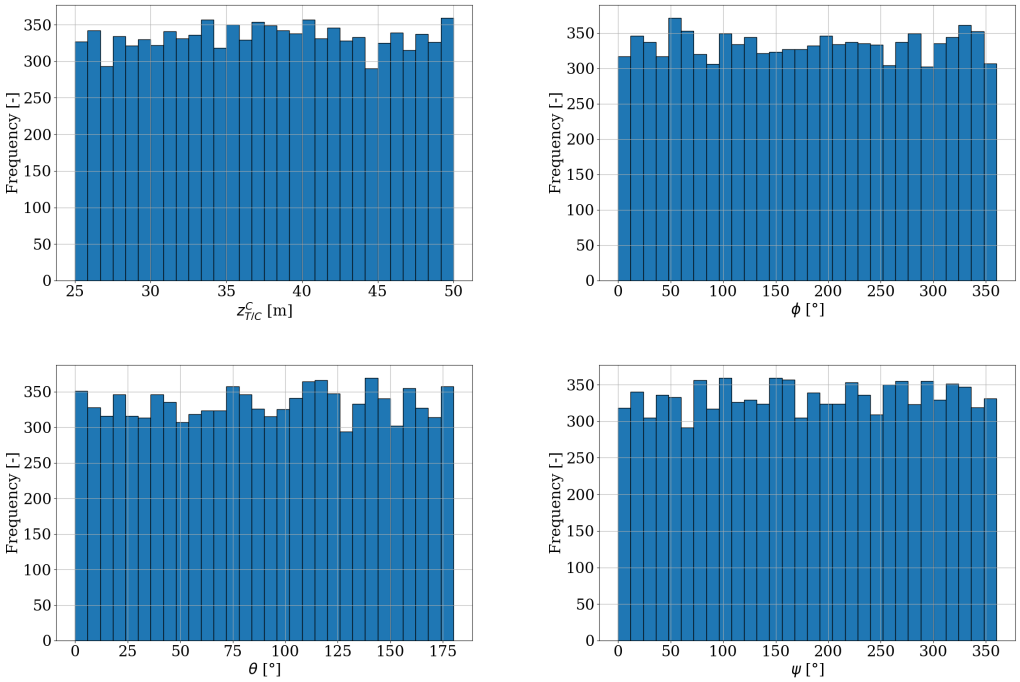


Figure 9.13: Nominal pose labels distribution for the ENVISAT `basic` and `realistic` sets.

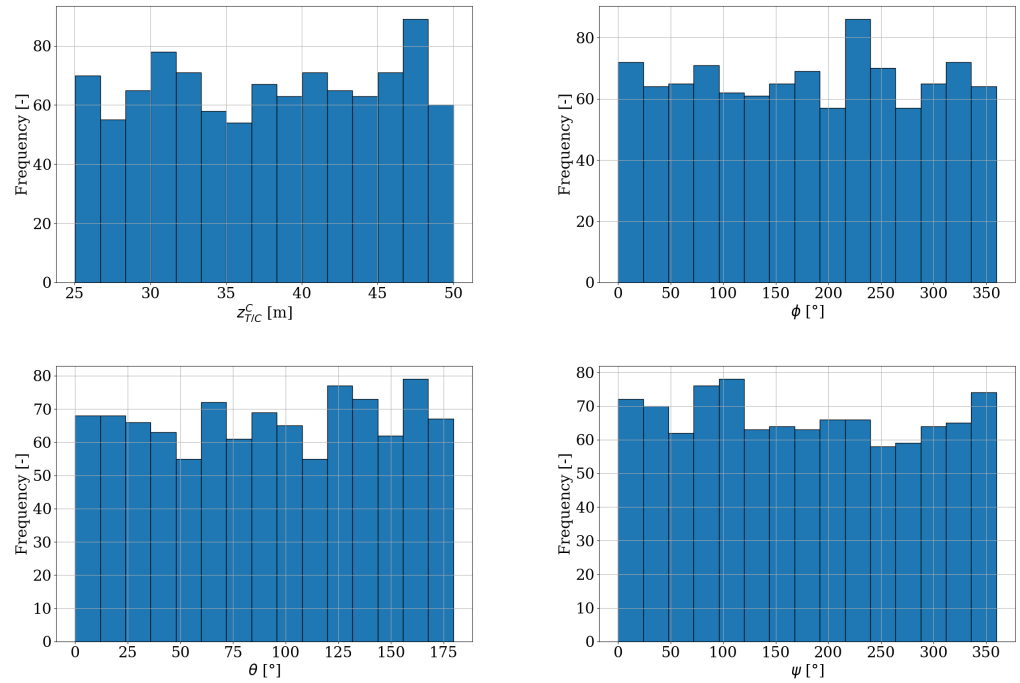


Figure 9.14: Nominal pose labels distribution for the ENVISAT `test` set.

The resulting pose label distributions for the different sets are shown in Figures 9.13 and 9.14. It can be seen how the resulting labels cover the range of possible relative distances and attitudes uniformly.

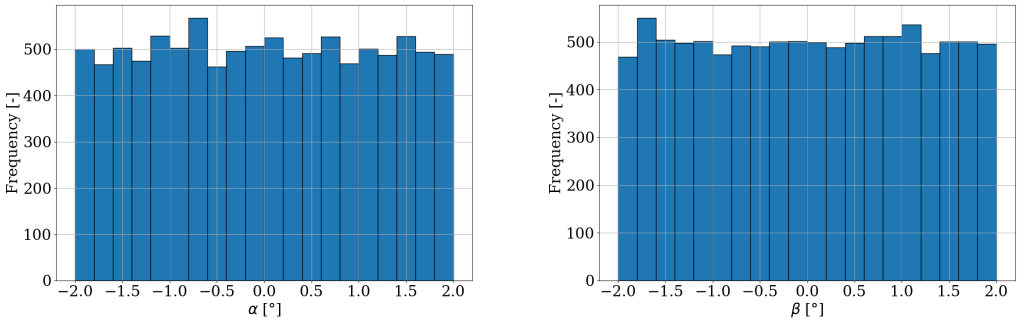


Figure 9.15: Pointing error distribution for the ENVISAT `basic` and `realistic` sets.

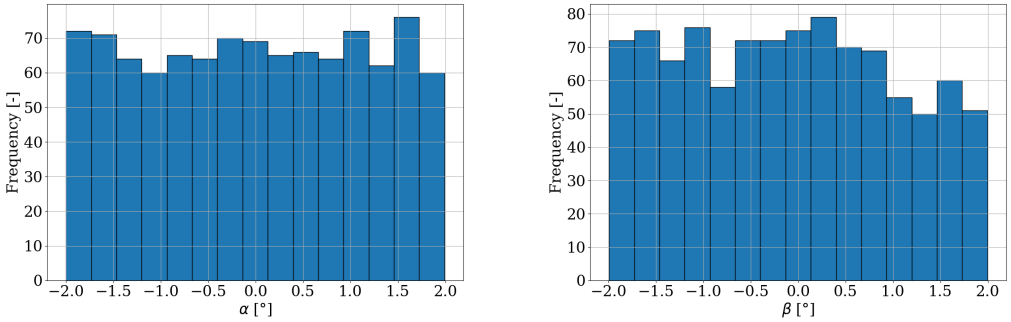


Figure 9.16: Pointing error distribution for the ENVISAT `test` set.

Pointing Errors

To include more challenging observation conditions with potentially non-visible keypoints, it was decided to apply a random pointing error to the nominal pose label. Such an error is applied using Equation (3.40), by assuming the camera is allowed to rotate along its x and y axes according to the following distributions:

$$\alpha \sim \mathcal{U}(-2, 2)^\circ \tag{9.9}$$

$$\beta \sim \mathcal{U}(-2, 2)^\circ \tag{9.10}$$

The extremes of the distributions above were chosen to include more challenging images where the satellite is not fully within the Field Of View (FOV), while making sure it was not possible to have less than six visible keypoints, which is the minimum number required by most pose solvers. Once again, the final distributions, shown in Figures 9.15 and 9.16, highlight a rather uniform coverage of the allowed pointing error range.

After combining the nominal pose label with the pointing error, the resulting relative position was saved as the relative position label $\mathbf{r}^C_{T/C}$, and the relative attitude converted to quaternions and saved as the relative attitude label $\mathbf{q}_{T/C}$.

Noise and Blur

The final step in the data generation pipeline consists of introducing random noise and blur in the images. This is done to better replicate the quality of actual raw space imagery, which is rarely subject to on-orbit processing to improve its quality, as Blender, by default, does not include any noise or blurring, resulting in images that lack any type of imperfection typical of realistic data. The augmentation parameters are included in Appendix D.

9.3.2. Cross-Validation and Testing

As shown in Table 9.2, both `basic` and `realistic` images are randomly split between training and validation, with about 10% of the generated images belonging to the corresponding validation set.

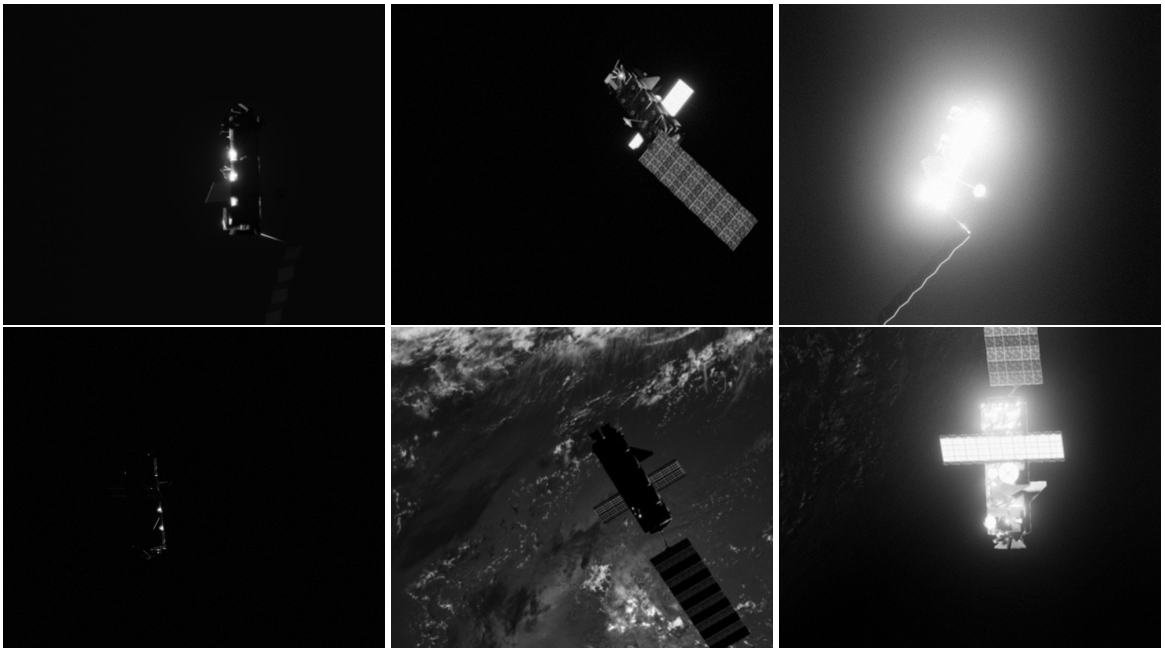


Figure 9.17: Images from the ENVISAT `test` set displaying different illumination conditions.

Generating a validation set for both domains was dictated not only by the need to perform online validation and verify that training would converge to an optimal solution, but also to investigate how well a model generalises on the other illumination conditions. The models trained on both the `basic` and the `realistic` sets will therefore be tested on the other validation images belonging to the other domain. The results of such a *cross-validation* step will be thoroughly examined in Section 10.3. After that, the models will be tested on the `test` images to study their behaviour over the variety of illumination conditions included in the `test` set. Some examples of the different illuminations the models will be tested on are shown in Figure 9.17. The dataset displays a great variance in terms of the illumination conditions, covering cases of both low and high exposure.

9.3.3. Sensitivity Analysis

Next to the `test` set, two additional sets are generated to perform a sensitivity analysis. Each of them contains 1,000 images generated with the same settings as the `test` set. The only difference is the exposure range used in Blender, set to [-3.2, 3.2] and [-3.4, 3.4].

9.4. Full Labels Generation

The generation of full labels consists of the steps illustrated in Figure 9.18. The labeling process is implemented with a Python script taking as input the relative pose labels. It applies the keypoint and bounding box location labels automatically by knowing the camera parameters and having a 3D model of the spacecraft available. The steps are here explained in detail:

1. **Pose label → keypoint labels:** Keypoints labels are generated by transforming the 3D coordinates of the target keypoints to the image frame. This can be done using Equation (4.9). The equation is implemented in the `project_kpts` function, which requires the nominal pose label and camera parameters. Both are provided to the Python script as `.json` files.
2. **Keypoint labels → bounding box label:** Once the keypoint pixel coordinates are retrieved, the tight bounding box label is generated by taking the minimum and maximum keypoint coordinates on the horizontal and vertical image axes.

Once the keypoint and bounding box labels are generated, they are merged together with the original pose label and saved in a new `.json` file containing the full set of labels, which can be used throughout the training and testing process.

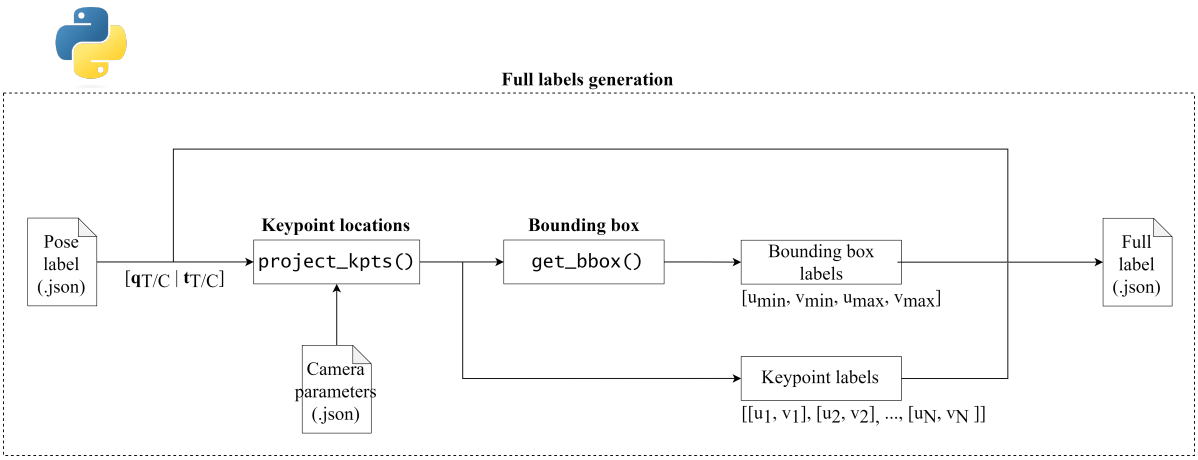


Figure 9.18: Overview of the label generation process.

Even if not included in the diagram in Figure 9.18, keypoint visibility labels are generated during training. A keypoint is labeled as visible or not by simply comparing the keypoint location with the image size. This is necessary, for instance, to only count the contribution of visible keypoints to the KD loss function in Equation (8.3). The other label processing steps performed during training include:

- Processing the bounding box label to take into account the image size. Any part of the bounding box falling outside of the images is removed.
- During training, the bounding box is relaxed by 10%. This was observed by B. Chen et al. (2019) and Van der Heijden (2022) to improve the OD performance.
- Keypoint locations are transformed to a new image frame, where the origin coincides with the pixel (u_{min}, v_{min}) from the bounding box label and the image resolution corresponds to the input size.

9.5. Summary

In the experimental phase of the study, two datasets are employed, namely SPEED+ and ENVISAT, each contributing unique characteristics to the investigation.

The SPEED+ dataset involves the Tango spacecraft and encompasses three different domains: `synthetic`, `sunlamp`, and `lightbox`. Notably, the synthetic set is generated with simple lighting settings, while the two testbed sets aim to replicate extremely adverse illumination conditions. Furthermore, the Earth in the background is only included in the `synthetic` and `lightbox` sets. The pose distribution is relatively uniform, and data augmentations will be applied to include an extensive list of effects observed in real space images into synthetic data.

On the other hand, the ENVISAT dataset, featuring the ENVISAT spacecraft, comprises three domains: `basic`, `realistic`, and `test`. They are characterised by increasingly more realistic and challenging illumination conditions, while all including the Earth in the background. The primary objective here is to assess whether training with photorealistic images enhances generalization to more adverse illumination conditions, simulating the transition from synthetic images for training and validation to real images for testing. Two additional sets have been generated for sensitivity analysis. Noteworthy is the plan to apply domain-specific augmentations exclusively when training on the `basic` set, since the `realistic` set already includes the most relevant ones because of the rendering settings used.

The next chapter will discuss the results of the development and testing of the baseline pose estimation pipeline on these two datasets.

Part IV

Experiments and Results

10

Domain-Gap Experiments

This chapter discusses the experimental setup and results concerning the analysis of domain-specific augmentations and photorealistic rendering.

10.1. Introduction

CNN models for terrestrial applications are trained on large datasets containing images from any source: online images, videos, and so on, subject to Earth illumination conditions and often highly processed (Deng et al., 2009; Lin et al., 2014). Given the hardware and power constraints in place for space missions, onboard preprocessing capabilities are severely limited. This results in images of poor quality compared to terrestrial data, and more in general in CNN models for on-Earth applications being trained on images with features represented very differently than space applications. Furthermore, none of the open-source large datasets typically used to train CNNs contain spacecraft images.

To train CNN models for spacecraft pose estimation, it is therefore necessary to generate data for training and testing. Such data needs to be representative of the specific mission, that is, the target spacecraft, camera parameters, background and illumination conditions have to be accurately simulated. Currently, the existing literature provides two distinct methods to generate training data, based on two different design philosophies: (i) collecting real images and using them for training and (ii) creating synthetic data mimicking the actual space environment as closely as possible. The first approach is based on the assumption that actual space data is available, however, this is almost never the case. To the author's knowledge, the only dataset used to develop a spacecraft pose estimation system is PRISMA25 (Sharma et al., 2018a), which is not publicly available and only contains 25 images of the Tango spacecraft. A proxy for this method is to generate images using a real camera and a scale model of the spacecraft in a testbed facility. However, this approach is not scalable as it results in significant mission costs as well as risks in case the accuracy of the testbed environment is not thoroughly validated. The approach based on purely synthetic data is the solution with minimal costs and early-stage mission risks, making it the most scalable in case of an increasing demand for RPO systems.

Although cost-effective, this approach requires accurate simulation of the imaging system and the operating conditions. Existing data generation pipelines make use of data augmentations to simulate image corruptions typical of the space environment. Some of the proposed augmentations were observed to significantly improve performance on testbed data (Park et al., 2019; Park et al., 2022b; Cassinis, 2022). However, as an extensive benchmark on the accuracy of synthetic augmentations was observed to be lacking in the literature, Section 10.2 compares the performance of pose estimation pipelines trained using different augmentation pipelines. Furthermore, rendering software is becoming increasingly more accurate, and even open-access ones, such as Blender, are arguably a competitive alternative to testbed facilities for generating representative data. As training pipelines relying on rendered images have already appeared, Section 10.3 provides a qualitative assessment of the benefit of including image corruptions directly in the images using photorealistic rendering. The conclusive remarks for this chapter are presented in Section 10.4.

Table 10.1: IoU performance across the SPEED+'s domains for different augmentation settings. The *baseline* augmentation configuration includes random brightness and contrast, noise, and blur. It is applied to every model along with the augmentation specified in the *model* column.

Model	synthetic		sunlamp		lightbox	
	mean [-]	median [-]	mean [-]	median [-]	mean [-]	median [-]
baseline	**0.971**	**0.979**	0.850	0.891	0.802	0.896
+ erasing	**0.971**	0.978	0.645	0.817	0.749	0.878
+ haze	0.970	0.978	0.844	0.890	0.779	0.884
+ stars	0.969	0.978	0.827	0.882	**0.886**	**0.929**
+ streaks	0.970	0.978	0.854	0.887	0.789	0.887
+ style aug.	0.967	0.977	0.856	0.897	0.766	0.885
+ Sun flare	0.968	0.975	**0.887**	**0.921**	0.855	0.922

10.2. Training with Synthetic Augmentations

The analysis on synthetic augmentations takes into account the augmentations introduced in Section 9.2.1. Following the work by Park et al. (2022b), each augmentation is applied independently with 50% probability, a value considered reasonable given the difficulty in determining the frequency at which each corruption occurs. The probability for each augmentation may be finetuned once more information about the mission is available.

The experiment requires training several models: one using only a baseline augmentation pipeline, and several others including one of the remaining augmentations each. Because of the extensive heritage in support of their application in any ML pipeline, blur, noise, and brightness and contrast are chosen as part of the baseline augmentation pipeline. Each of the remaining six augmentations (Figure 9.6h-9.6m) is added on top of the baseline pipeline to train an EfficientNetB3-SSD512 and an HRNet-w32 model on SPEED+. This results in six different models whose performance analysis compared to the baseline allows to design the optimal augmentation pipeline for the final model.

Lastly, as mentioned in Section 6.2, it was chosen to apply style augmentation by generating an augmented version of the dataset offline instead of running inference with the style augmentor network at each training step. Although no overfitting was observed during training, it should be noted that, by training the model on an identical dataset every epoch, the final generalisation capabilities could be subject to a slight degradation.

10.2.1. Impact on Object Detection

Table 10.1 illustrates the performance on SPEED+ of the EfficientNetB3-SSD512 models trained on the different augmentations. As can be seen from the table, there is no relevant performance difference on synthetic images, meaning that all the augmentations considered allow the network to generalise on the original data distribution. Nevertheless, it can be seen how the model trained on baseline augmentations performs slightly better than the other. This can be expected due to the fact that the baseline augmentation pipeline is the one modifying the synthetic data distribution the least, resulting in the training data following more closely the distribution of the validation data. Furthermore, the fact that all mean and median IoUs are close to 1 is an indication that each model has converged to a global optimum for the synthetic data distribution.

Despite all models having comparable performance on the synthetic validation set, significant differences appear when testing the networks on testbed data. Even if all models experience performance degradation, the erasing augmentation is the one leading to the worst detection accuracy on both `sunlamp` and `lightbox` images. Possibly, the network is misled by the artificial change in the object shape. Applying patches to the image modifies the spacecraft shape and texture in a way that is not representative of testbed data, leading the OD network to learn features that are not meaningful at all.

The haze, streaks, and style augmentation either do not affect performance significantly or produce slightly worse results. This can be attributed to the fact that these augmentations do not introduce features relevant for the network. The model does learn anything new compared to just applying the baseline augmentation pipeline.

The stars and Sun flare augmentations are the only ones leading to a relevant performance improvement in performance. Stars positively affect the model performance on `lightbox` images but not on `sunlamp` ones, whereas Sun flare improves the detection accuracy on both domains. This can be

Table 10.2: RMSE across the SPEED+'s domains for different augmentation settings. Again, the baseline augmentations are applied to every model along with the augmentation specified in the *model* column.

Model	synthetic		sunlamp		lightbox	
	mean [pix]	median [pix]	mean [pix]	median [pix]	mean [pix]	median [pix]
baseline	1.22	0.63	39.79	17.06	20.11	5.40
+ erasing	**1.09**	**0.62**	11.37	4.35	**7.05**	**2.68**
+ haze	1.25	0.64	39.30	16.60	22.55	6.35
+ stars	1.24	0.63	38.28	15.88	19.71	5.40
+ streaks	1.26	0.63	40.32	17.19	21.14	5.84
+ style aug.	1.44	0.75	38.06	13.92	19.45	5.20
+ Sun flare	1.25	0.70	**8.31**	**3.91**	7.37	2.93

attributed to the fact that the stars and Sun flare augmentations resemble testbed data the most. Also notice how the stars augmentation makes the model performance on `sunlamp` images slightly worse, which can be attributed to the fact that only `lightbox` images contain stars in the background.

Overall, the OD network seems to not respond to most of the augmentations tested, with the Sun flare augmentation being the only one leading to a significant improvement over both testbed domains. When applying synthetic augmentations to an OD network to bridge the gap with real data, the effects they try to simulate have to be carefully selected and the augmentations accurately implemented to replicate the testing data distribution. A positive impact on the model performance should not be given for granted.

10.2.2. Impact on Keypoint Detection

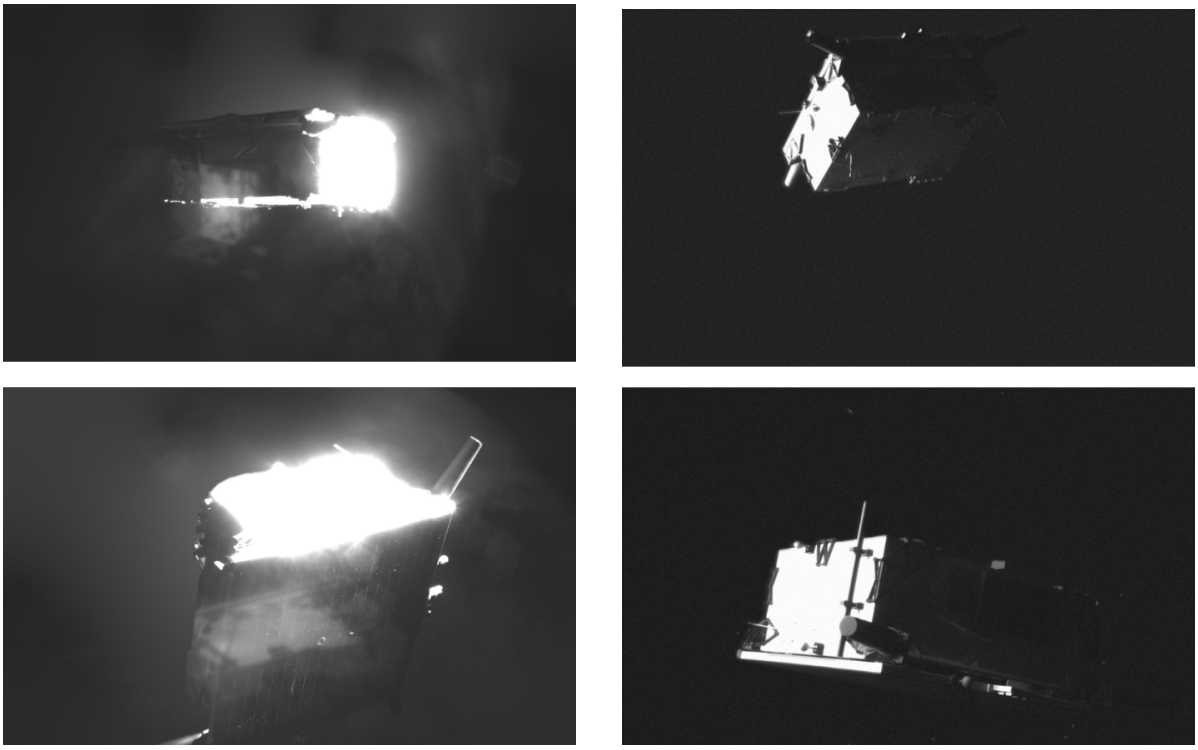
Table 10.2 shows the KD performance of each augmentation across each domain. It can be seen how the domain gap between the synthetic and testbed domains affects the KD performance of each network. The `sunlamp` domain is the most challenging. All models see the mean and median increase when tested on such a domain compared to the synthetic one. A similar although more limited degradation occurs with the `lightbox` domain as well.

The performance on the `synthetic` domain are similar across all augmentation settings. All models achieve an average RMSE close to one pixel, with a median well below the pixel-level accuracy. Again, this can be interpreted as the successful convergence of each model to a global minimum for the synthetic data distribution. The situation is different when considering testbed images: the model trained on the baseline augmentation settings only achieves an average RMSE of 39.79 pixels on `sunlamp` and 20.11 pixels on `lightbox`. A slight degradation in detection accuracy occurs when applying the streaks augmentation, whereas the haze augmentation leads to a minimal improvement over the `sunlamp` images and degrades performance over the `lightbox` ones. The stars and style augmentations lead to a small improvement in both testing domains, whereas the erasing and Sun flare augmentations remarkably improve KD accuracy. Erasing reduces the RMSE mean and median to 11.37 and 4.35 pixels on `sunlamp` images and to 7.05 and 2.68 pixels on `lightbox` images. Similarly, Sun flare shrinks the detection error to a mean of 8.31 and a median of 3.91 pixels on the `sunlamp` domain, and a mean of 7.37 and a median of 2.91 pixels on the `lightbox` domain.

Overall, the impact of each augmentation on the KD accuracy differs significantly, ranging from a slight degradation (see haze and streaks) to a reduction of 80% of the mean RMSE for the Sun flare augmentation. Similarly to the OD model, attention should be paid before including synthetic augmentations in the training pipeline, as they could lead to performance degradation if not tested in advance. Even if their aim is to replicate features observed in testbed images, the network might not be sensitive to changes in the training data distribution.

10.2.3. Augmentation Selection

The results summarised in Tables 10.1 and 10.2 offer some insights into the severity of the domain gap between synthetic and testbed data. First, the performance on synthetic data are hardly representative for testbed data. Second, high-intensity, direct sunlight illumination appears to be the most beneficial effect to simulate during training, as highlighted by the performance enhancement for both OD and KD.



(a) `sunlamp` **(b)** `lightbox`

Figure 10.1: Samples from the SPEED+ `sunlamp` and `lightbox` domains.

This is to be expected as both direct sunlight illumination leads to drastic changes in the images, such as higher contrast and the presence of saturated regions caused by the light reflected by the target. Figure 10.1 shows a few samples from the `sunlamp` set where the effect of extreme illumination is particularly evident. `lightbox` images, on the other hand, still display artefacts due to adverse lighting conditions, such as high contrast, but are arguably less extreme as can be seen from the figure.

Synthetic augmentations can be applied to improve the robustness of the neural network to corruptions and artefacts observed in real images. These augmentations are not necessarily among those commonly applied, such as in the list provided by Hendrycks et al. (2019), but aim at simulating effects that are specific to the target data distribution. For instance, the Sun flare augmentation replicates the saturation of entire image regions due to the reflection of high-intensity sunlight from the target, which is observed to benefit both the OD and KD models. This can be identified as the main cause why applying this augmentation leads to the best performance on the `sunlamp` domain. Similarly, erasing simulates the high contrast between parts of the satellite that are illuminated and shadowed by the Earth's albedo sunlight. This makes the model trained on random erasing the best-performing one in the `lightbox` category. The point also holds for the stars augmentations, even though with the due proportions. For a neural network, learning to make predictions regardless of the presence of stars in the background is a simpler task compared to detecting keypoints when entire parts of the satellite may not be visible.

Despite their aim to replicate effects observed in the target data distribution, the impact of some augmentations changes substantially between the OD and KD model, such as stars and erasing. This can be attributed to the distinction between the two tasks and the consequent difference in the relevant features learned by the network. A major difference between the two pipelines is that the inputs to the OD network are the full SPEED+ images, while the KD network takes as input SPEED+ images cropped around the bounding box and resized. Augmenting training images with stars in the background has a greater efficacy because (i) the augmentation affects a larger portion of the image and (ii) they are applied to the background of `lightbox` images, which is the domain on which stars has the greatest beneficial impact.

A similar reasoning holds for the erasing augmentation, which displays the greatest difference in

Table 10.3: Comparison between IoU performance on SPEED+ for the OD network before and after applying the full augmentation pipeline.

Model	synthetic		sunlamp		lightbox	
	mean [-]	median [-]	mean [-]	median [-]	mean [-]	median [-]
baseline	**0.971**	**0.979**	0.850	0.891	0.802	0.896
augmented	0.967	0.975	**0.866**	**0.915**	**0.888**	**0.930**

performance between the two tasks. Since it modifies the shape of the object, it can be expected to worsen the performance of an OD network rather than a KD one, where the focus is on the specific features to be detected. Considering augmentations, such as haze and streaks, that do not impact the spacecraft but the entire image, the benefit in KD accuracy will be less relevant compared to OD. For this reason, the haze and streaks augmentations improve the OD accuracy while not bringing any relevant impact to the KD accuracy. On the other hand, augmentations, such as Sun flare, replicating relevant effects that directly impact the visibility of the spacecraft, can be expected to have a beneficial impact on both OD and KD performance.

Based on the analysis just performed, a final augmentation pipeline is selected for the OD and KD models. Because they are the only augmentations leading to a significant improvement in performance, stars and Sun flare are selected for the OD model. With regards to the KD model, erasing and Sun flare lead to the most remarkable improvement in performance, whereas for the stars and style augmentations, the improvement is still clear for both testbed domains although less remarkable.

10.2.4. Final Performance Analysis

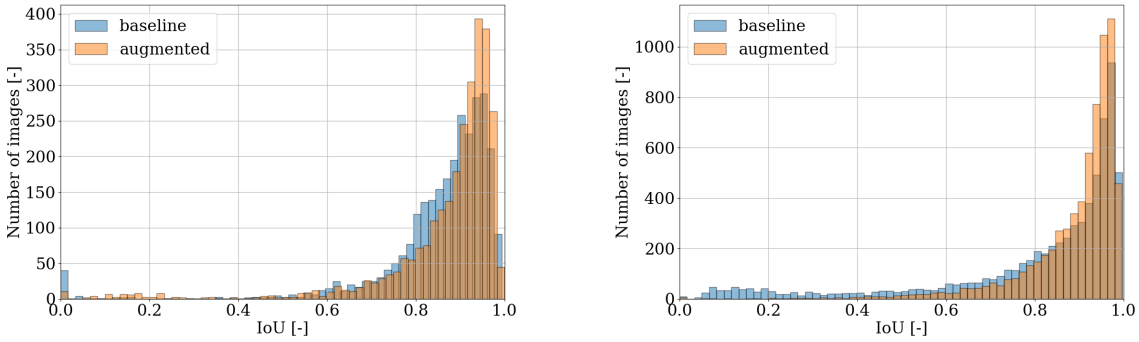
The final models are trained with the same settings as listed in Sections 7.3 and 8.4, with the only difference being the augmentation pipeline applied. For the OD model, the stars and Sun flare augmentations are applied on top of the baseline pipeline, whereas the KD models saw the erasing, stars, style, and Sun flare augmentations applied together with the baseline pipeline. The performance of the final augmented model is analysed in the upcoming sections.

Object Detection

Table 10.3 compares the performance of the baseline and augmented OD model. Consistently with the analysis on single augmentations, the baseline model performs slightly better than the augmented one on synthetic images, but struggles on testbed ones. This confirms the fact that SPEED+'s synthetic images are not representative of testbed images, making it necessary to introduce additional real-world effects in the data. In fact, applying the stars and Sun flare augmentations together improves the performance on both testbed domains compared to the sole baseline. On the one hand, the improvement can be considered remarkable for the `lightbox` domain, increasing by more than 8%. On the other hand, the improvement over the `sunlamp` domain is of only a few percentage points, it should be kept in mind that the baseline starts from a reasonably good accuracy, with the mean IoU being as high as 0.850. The model therefore ends up achieving better performance on `lightbox` images despite the more challenging background conditions, which can be seen as an indication of the model robustness to the Earth in the background.

The performance achieved on `lightbox` images are better than when applying the augmentations individually, achieving a mean and median IoU of 0.888 and 0.930, respectively. Instead, the performance achieved on the `sunlamp` domain are slightly inferior compared to the application of the only Sun flare augmentation. This can be expected by looking at the preliminary results in Table 10.1, where applying the stars augmentation resulted in a slight loss in performance. As a consequence, combining this effect with the Sun flare augmentation mitigated the overall improvement. Generally, the combined effect of the different augmentations on the OD model can be predicted *a-priori*. If an augmentation has a positive effect on performance alone, it will contribute positively to the final model performance, even when multiple augmentations are applied.

Looking at the model performance more in detail, Figure 10.2 shows the IoU distribution for the baseline and augmented models over the `sunlamp` and `lightbox` domains. The histograms show that applying the augmentation pipeline lets the network achieve a higher IoU on a larger number of images. On `sunlamp` images, the augmented model scores an IoU of 0.9 or above with a higher frequency than the baseline model, whereas the number of images with an IoU below 0.8 is significantly reduced on



(a) `sunlamp` **(b)** `lightbox`

Figure 10.2: IoU histogram comparison between the baseline and augmented models over the `sunlamp` and `lightbox` images.

`lightbox` images. Figure 10.2a also shows a peak for IoU=0, *i.e.*, images where the network does not detect the object at all, caused by the presence of the lamp in the background of several `sunlamp` images. This feature can be quite misleading for the model, as shown in Figure 10.3, which tends to confuse the lamp with Tango or ends up not regressing any bounding box at all. Since this effect is not included in SPEED+'s synthetic images, the model is not expected to perform well in this case. However, it can be seen how augmenting the dataset strongly mitigates this effect, reducing the total number of images from 45 to 10. To further improve the network robustness, whenever no bounding box is detected the full image is passed to the KD network.

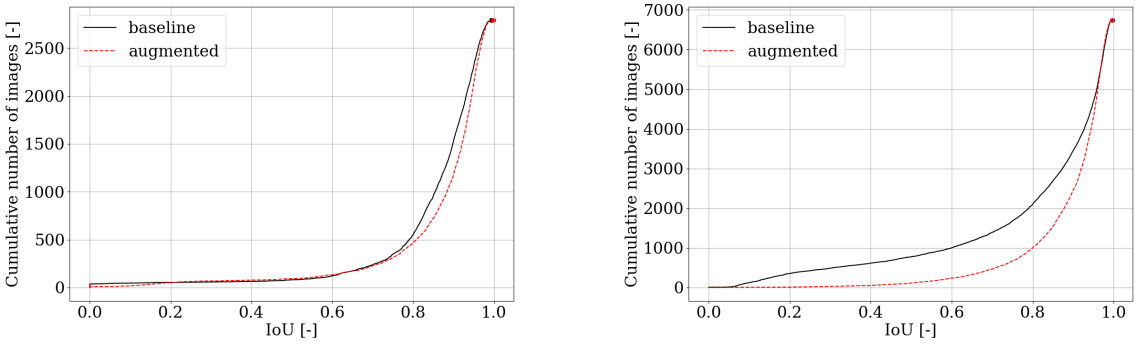
As an additional confirmation of the benefits brought by the augmentation pipeline, Figure 10.4 compares the cumulative IoU distribution of the baseline and augmented models. Although the model is able to detect a larger number of images with an IoU above 0.9 for the `sunlamp` domain, the benefit is more remarkable on `lightbox` images. The number of images with an IoU below 0.9 is largely reduced and, furthermore, the worst 1,000 predictions achieve an IoU of 0.8, against the 0.6 of the baseline model. The augmentation pipeline not only improves the overall model performance but also tends to improve thee predictions for individual images. Figure 10.5 shows the IoU correlation between the baseline and augmented models over the testbed data. The figure also includes the bisector to better identify the images on which the network improved. The augmented model improves IoU for over 60% of the images and displays enhanced robustness over images on which the baseline model achieves an IoU equal to zero. The improvement is even more noteworthy on `lightbox` images: despite the percentage of improved images being the same (64%), Figure 10.5b shows numerous data points clustering in the bottom-left corner of the image, corresponding to low IoU for the baseline model and high IoU for the augmented model. It can be concluded that the augmented model not only achieves better overall performance but also tends to improve individual predictions. The implemented augmentation pipeline improves the OD robustness over features that are meaningful for testbed data.

Despite the achieved improvement, the performance of the augmented model can still be improved. Table 10.3 shows how the IoU gap between synthetic and testbed, although reduced, is still significant and can as wide as 10% on the average IoU. Furthermore, as can be seen from Figure 10.5, the model still performs poorly on some samples from the `sunlamp` and `lightbox` sets, with numerous detections achieving an IoU below 0.2.

To investigate the reason behind the remaining performance gap, the number of visible keypoints was initially considered as the driving factor, assuming the network would lose accuracy when the target is only partially visible. Figure 10.6 shows how the IoU distribution changes over the number of visible spacecraft keypoints. Surprisingly, the majority of outliers occur when the spacecraft is fully within the FOV, while the network preserves its accuracy when fewer keypoints are visible. The network achieves an IoU superior to 0.8 when only 5 keypoints are visible on both the `sunlamp` and `lightbox` domains. Additionally, no clear trend is observed in the median IoU. The IoU seems to slightly increase in between 5 and 8 visible keypoints, remains constant, and finally decreases when all 11 keypoints are visible. The hypothesis of loss in accuracy with fewer visible keypoints is therefore inconsistent with the observed behaviour.

Another factor potentially explaining the presence of the remaining outliers is the distance between

Figure 10.3: Sample `sunlamp` predictions by the OD baseline model with IoU=0.

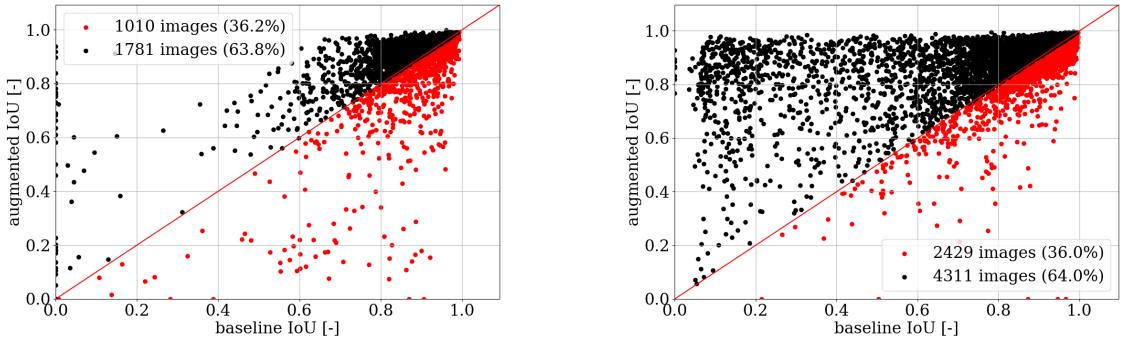


(a) `sunlamp` **(b)** `lightbox`

Figure 10.4: Comparison between cumulative IoU distributions for the baseline and augmented models over the `sunlamp` and `lightbox` images.

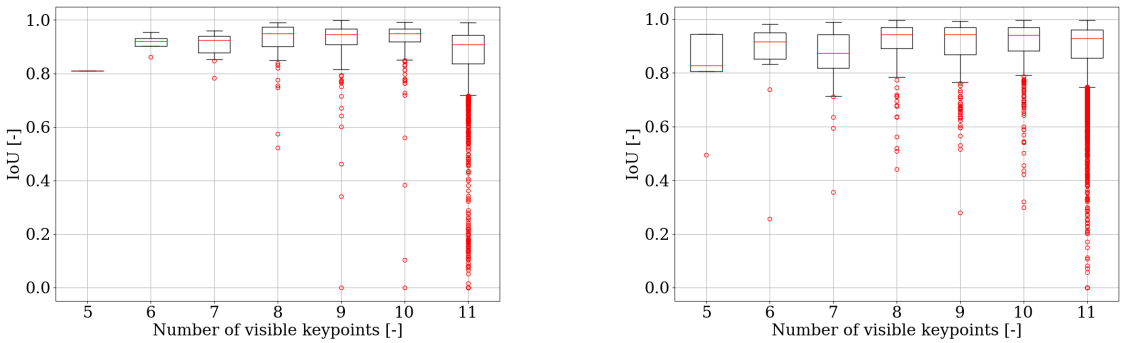
the camera and the target, as the network can be expected to struggle more in detecting the spacecraft when it is further away from the camera. Figure 10.7 shows how the IoU distribution changes with the relative distance. It can be seen from Figure 10.7a how there is a slight decrease in accuracy over `sunlamp` images when the relative distance increases, whereas Figure 10.7b shows that the IoU over `lightbox` images has very little correlation with the relative distance. In addition to this, the network achieves a very low IoU over a few samples from the `sunlamp` set with high relative distance. This can be again explained by the presence of the lamp in the background of some images confusing the network. When the relative distance is large, there is a higher chance of misleading the OD network by making the spacecraft and lamp appear to be the same size. Overall, however, the weak correlation found between the `sunlamp` IoU and the relative distance is not sufficient to explain the presence of all the outliers.

The last factor considered in the analysis is the variability in illumination conditions. This hypothesis was formulated after looking at the relevant literature, such as the survey by Cassinis et al. (2019), where adverse illumination is mentioned as the driving factor behind the domain-gap issue. The presence



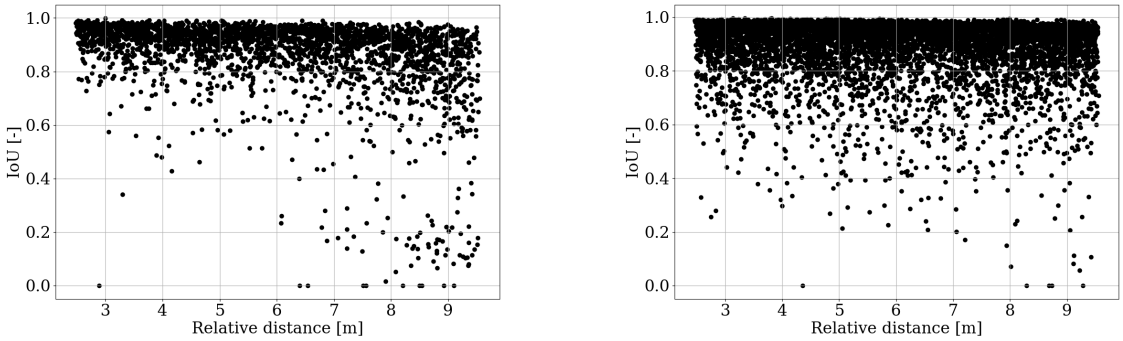
(a) `sunlamp` **(b)** `lightbox`

Figure 10.5: IoU correlation between the baseline and augmented models over the `sunlamp` and `lightbox` images.



(a) `sunlamp` **(b)** `lightbox`

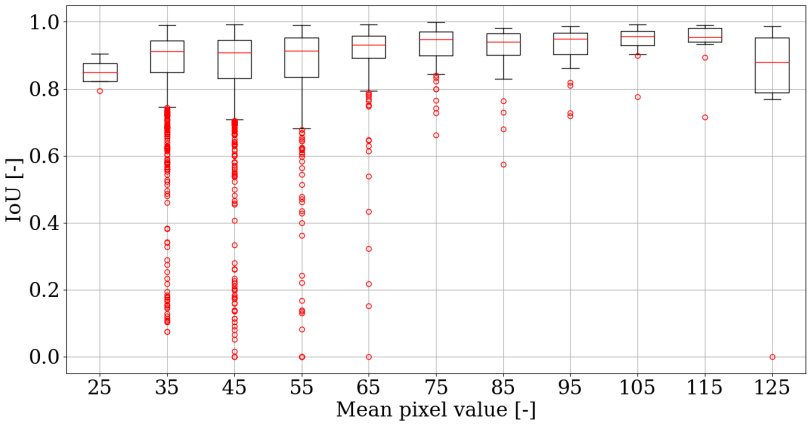
Figure 10.6: IoU distribution over the number of visible keypoints in an image for the augmented model.



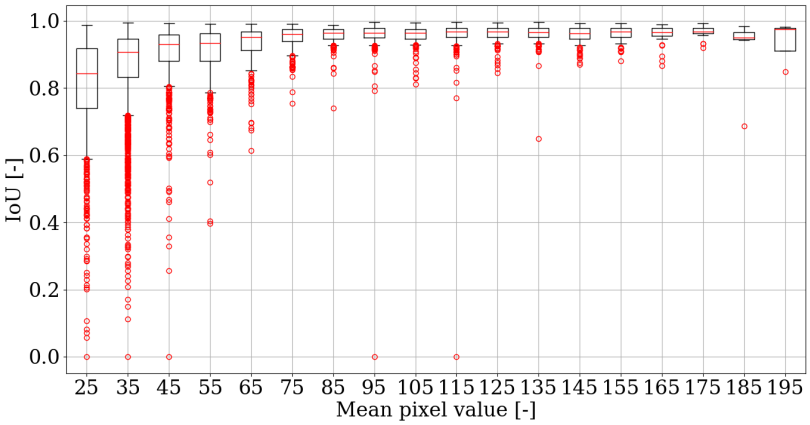
(a) `sunlamp` **(b)** `lightbox`

Figure 10.7: IoU distribution over the relative distance for the augmented model. The red line indicates the median, the boxplot extremes the 1st and 3rd quartiles, while the outliers are defined as the 10% least accurate predictions.

of outliers can therefore be explained by the network performing poorly on images with extreme illumination conditions. Figure 10.8 shows how the IoU distribution changes over different ranges of image mean pixel value. The mean pixel value is selected as the comparison metric because it is related to the overall brightness of the image. Images with a lower mean pixel value are typically correlated with low-exposure images and, similarly, images with a higher mean pixel value can be associated with high-exposure images. It is therefore considered a reliable indication of the global image illumination conditions.



(a) `sunlamp`

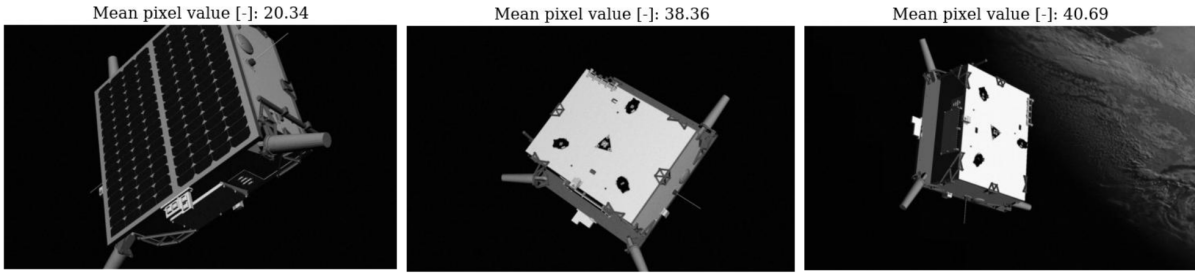


(b) `lightbox`

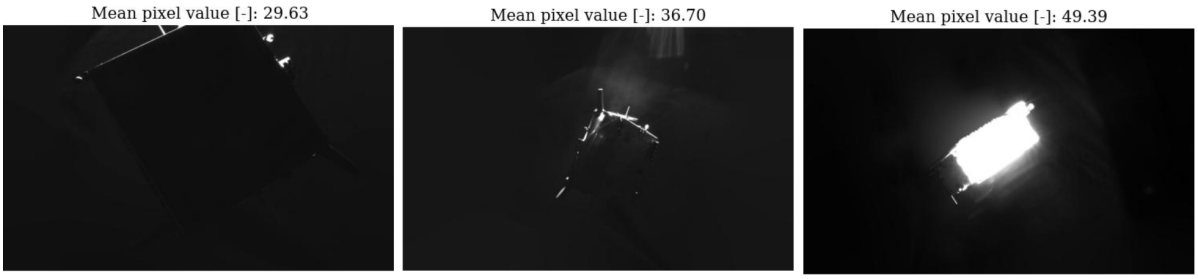
Figure 10.8: IoU distribution over different pixel mean values for the augmented model.

Images are split discretely into buckets with a width of 10. The values shown in the figure refer to the mean bucket value. It can be seen from Figure 10.8a how the majority of outliers over the `sunlamp` set occur for lower pixel values, specifically in the range between 30 and 70. The network accuracy increases for larger pixel values, but decreases again when the image is very bright (mean pixel value above 120). Similarly, the network struggles with low-exposure images belonging to the `lightbox` domain, with the majority of outliers having an average pixel value between 20 and 70. Also, the performance sees a slight degradation for images with a very high exposure. Unlike the performance on the `sunlamp` domain, however, the network sees a drastic increase in robustness for images with intermediate levels of exposure, with the IoU distribution being centred on high IoU values. The OD network therefore lacks robustness over images with extreme illumination conditions.

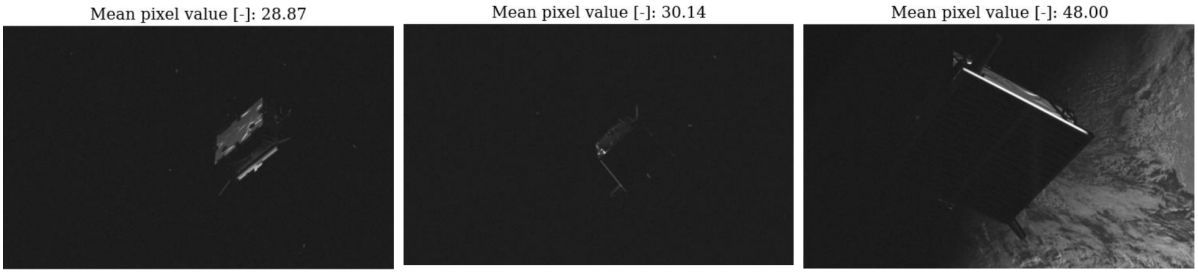
After the outlier analysis, it is surprising to see that, despite including synthetic augmentations replicating adverse illumination conditions, such as the Sun flare augmentation, the network seems to still be affected by adverse illumination conditions. This behaviour can be explained by looking at the illumination conditions across the synthetic and testbed sets. By looking at different samples from the SPEED+ dataset, it can be noted that the rendering settings used to generate the synthetic set are not consistent with the testing data. Figure 10.9 shows a few samples from each domain with comparable mean pixel values. What can be immediately seen is how differently the spacecraft is illuminated. While Tango's features are clearly visible in the synthetic set, a significant part of the spacecraft can be almost confused with the background in low-exposure images from the testbed sets. Furthermore, the overexposure achieved in some `sunlamp` images cannot be reached by the synthetic set because



(a) synthetic



(b) `sunlamp`



(c) `lightbox`

Figure 10.9: SPEED+ samples for each domain with different mean pixel values.

Table 10.4: RMSE performance on SPEED+ for the KD network before and after applying the full augmentation pipeline.

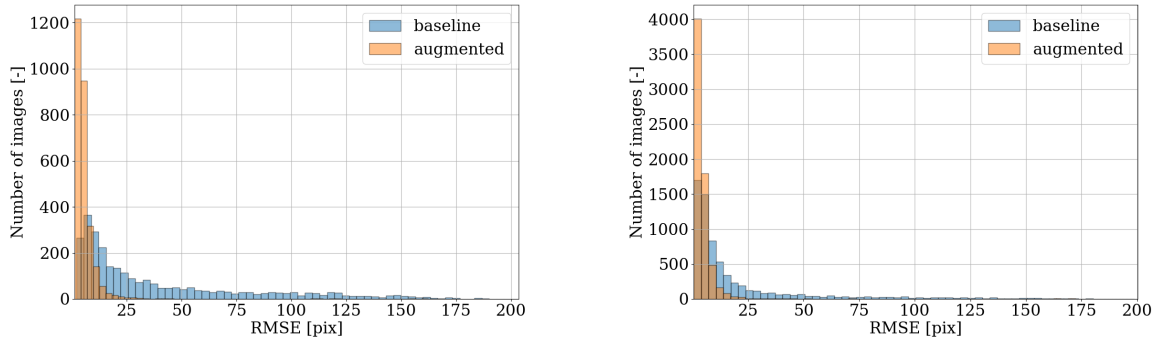
Model	`synthetic`		`sunlamp`		`lightbox`	
	mean [pix]	median [pix]	mean [pix]	median [pix]	mean [pix]	median [pix]
baseline	**1.22**	**0.63**	39.79	17.06	20.11	5.40
augmented	1.48	0.86	**6.69**	**3.51**	**6.03**	**2.69**

of the lack of realism in the rendering settings. As a result, it is reasonable to expect a performance gap between synthetic and testbed images. This explains the high performance difference experienced by the baseline model. Applying synthetic augmentations mitigates this effect, but they are unable to replicate the observation conditions with complete realism.

Keypoint Detection

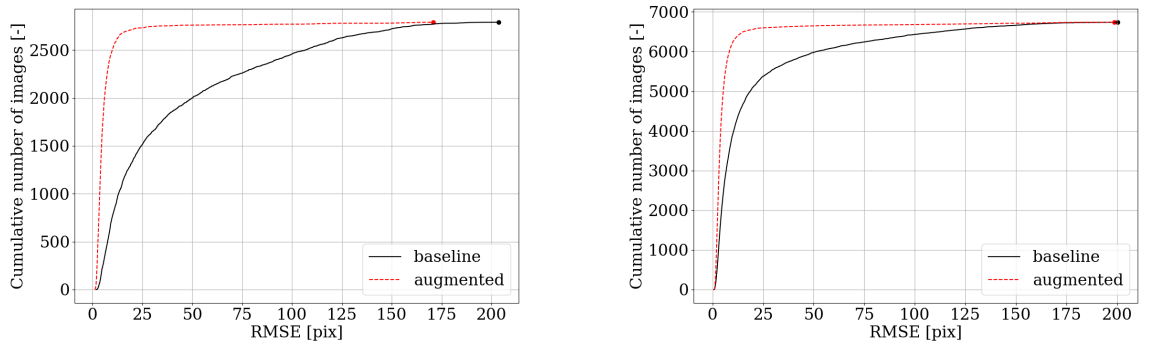
Table 10.4 shows the KD accuracy for the baseline and augmented model. As for OD, the baseline model performs better on synthetic but not on testbed data. The performance achieved on the validation set, however, can be seen as an indication that both models have converged on the synthetic data distribution.

The augmented model significantly reduces the KD error for both testbed domains. The average and median RMSE are reduced from 39.79 and 17.06 pixels to 6.69 and 3.51 pixels, respectively, for the `sunlamp` images. That is a decrease of above 83%. The performance on the `lightbox` images experience a similar improvement, improving the RMSE mean and median from 20.11 and 5.40 pixels to 6.03 and 2.69 pixels, which corresponds to a reduction of 70%. This is due to the simultaneous application of the augmentations selected in Section 10.2.3, which combine synergically and reduce the error further



(a) `sunlamp` **(b)** `lightbox`

Figure 10.10: RMSE histogram comparison between the baseline and augmented models over the `sunlamp` and `lightbox` images.



(a) `sunlamp` **(b)** `lightbox`

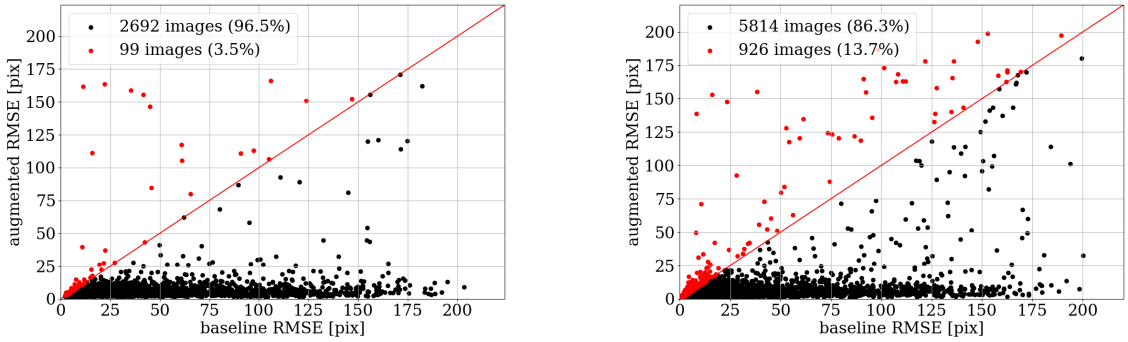
Figure 10.11: Comparison between cumulative error distributions for the baseline and augmented models over the `sunlamp` and `lightbox` images.

compared to the case where only one augmentation is applied. Compared to any of the models listed in Table 10.2, the augmented model in Table 10.4 achieves the best performance over both testbed domains. Similarly to OD, synthetic augmentations applied to a KD model combine positively for a further increase in performance.

The designed augmentation pipeline does not only reduce the error over the `sunlamp` and `lightbox` domains in absolute terms, but also reduces the difference in performance across the two domains. In fact, the model passes from having a `sunlamp` mean RMSE twice as large and a median RMSE more than three times as large as the corresponding `lightbox` error to a difference of less than a pixel. The final average and median RMSE error over `sunlamp` images is only 0.63 and 0.82 pixels larger than over `lightbox` images. The developed augmentation pipeline also makes the KD model robust to widely different illumination conditions, such as direct sunlight and albedo illumination.

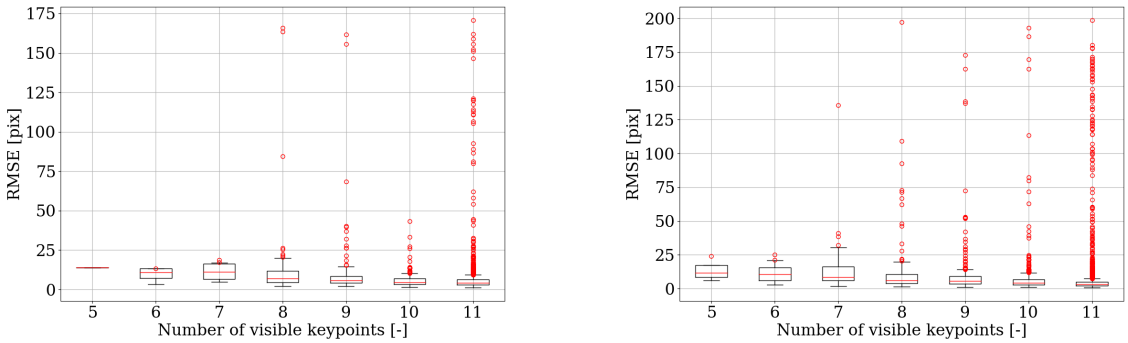
Figure 10.10 shows the RMSE distribution for the baseline and augmented models over the `sunlamp` and `lightbox` images. A distinct improvement can be observed on both domains, with the network achieving a RMSE below 25 pixels for most images, improving both the overall accuracy and robustness on the test sets. As shown in Figure 10.11, the cumulative error distribution is significantly improved for both testbed domains. The augmented model is able to detect 2,500 images with a RMSE of about 10 pixels, against the almost 110 for the baseline model. Similarly, the average RMSE corresponding to the first 6,000 predictions on the `lightbox` set is reduced from 50 to 10 pixels by the augmented model.

As can be expected by the large reduction observed in the mean and median RMSE, the testbed model greatly improves the detection accuracy on testbed data. Figure 10.12 shows the error correlation between the baseline and augmented models. Only 3.5% and 13.7% of the images from the `sunlamp` and `lightbox` sets, respectively, see the detection accuracy worsening after applying the synthetic augmentations.



(a) `sunlamp` **(b)** `lightbox`

Figure 10.12: Error correlation between the baseline and augmented models over the `sunlamp` and `lightbox` images.



(a) `sunlamp` **(b)** `lightbox`

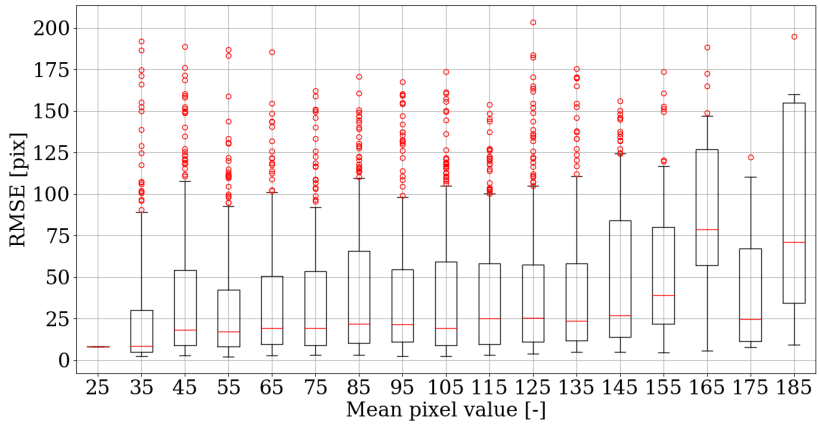
Figure 10.13: RMSE distribution over the number of visible keypoints in an image for the augmented model.

The drastic improvement observed in the KD performance further confirms the benefits of applying a customised augmentation pipeline to reduce the gap with testing data. More remarkably than for OD, the augmentation pipeline shrinks the performance difference between synthetic and testbed images. However, just like the OD network, the scatter plots in Figure 10.12 how the network still struggles on a subset of images, obtaining a RMSE as large as 200 pixels. To motivate the presence of these outliers, different factors were considered, such as the number of visible keypoints, relative distance, and the illumination conditions.

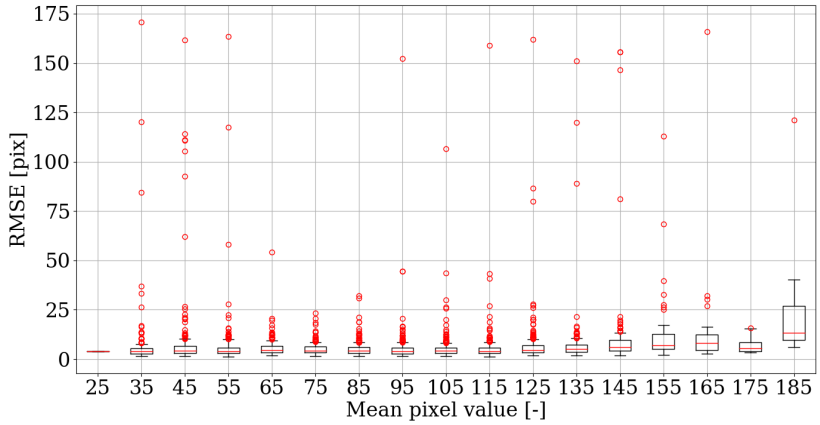
Figure 10.13 shows the RMSE distribution against the number of visible keypoints. The behaviour of the KD model is similar to OD, with most outliers corresponding to images where the entire spacecraft is visible. This can be expected as the KD is less likely to rely on the global shape and texture of the spacecraft. Instead, each keypoint is detected by leveraging local features. The impact of relative distance was also analysed, but no relevant trend was observed. This can be expected because the KD model performs inference on cropped images, making the size of the spacecraft in the input image always the same.

The final effect to be investigated is the image illumination. Again, the mean pixel value is used as comparison metric, splitting the images into buckets according to their average pixel value. In this case, however, the mean was calculated on the cropped image and not on the full-size one to make the analysis consistent with the input seen by the network.

Figure 10.14 and Figure 10.15 show the RMSE trend for different mean pixel values. The comparison between the baseline and augmented model clearly shows the benefits of applying the augmentation pipeline. The RMSE distribution over `sunlamp` images almost entirely collapses below the threshold of 25 pixels. Similarly, the median RMSE and the number of outliers for the different pixel value ranges are greatly reduced on the `lightbox` set. This further confirms the fact that synthetic augmentations can be used to fill the domain gap. The figures, however, also show how the model still struggles with



(a) baseline



(b) augmented

Figure 10.14: RMSE distribution over different pixel mean values for the `sunlamp` domain.

Table 10.5: RMSE performance on SPEED+ for the KD network when bounding box predictions are used.

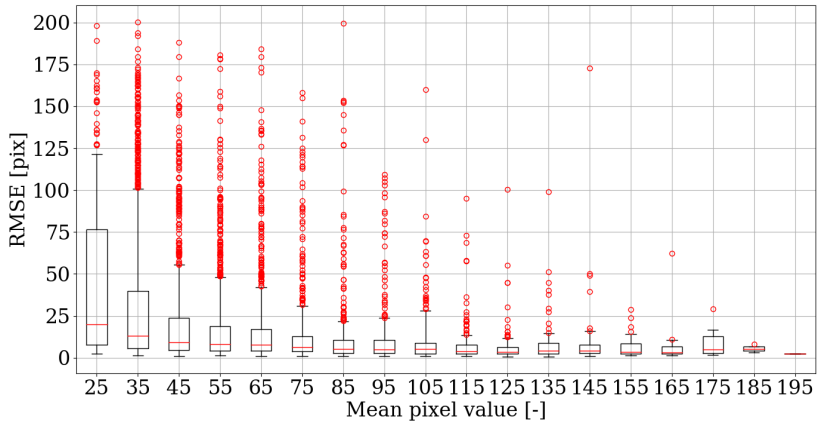
Model	`synthetic`		`sunlamp`		`lightbox`	
	mean [pix]	median [pix]	mean [pix]	median [pix]	mean [pix]	median [pix]
baseline	**1.92**	**0.97**	49.18	24.42	34.86	10.43
augmented	2.36	1.33	**16.99**	**6.17**	**15.27**	**4.55**

some samples from both sets. Although more robust, the augmented model displays a larger RMSE for high-exposure samples from the `sunlamp` set and low-exposure images belonging to the `lightbox` domain. As already discussed for the OD model, this is due to inconsistency in the illumination conditions between synthetic and testbed images, which cannot be fully compensated for by synthetic augmentations applied *a-posteriori*.

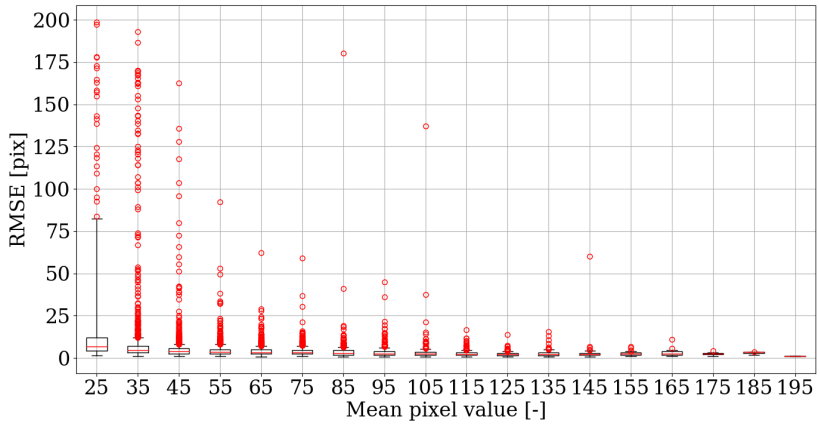
Pose Estimation

To fully implement an end-to-end pose estimation pipeline, it is necessary to first integrate the OD model with the KD one. This is done by using the predicted bounding box coordinates to crop the original image before giving it as input to the KD model. Second, the extracted keypoint locations are transformed from the heatmap to the original image frame and used by the pose solver. This section discusses the results concerning the two integration steps.

Table 10.5 shows the KD accuracy for the baseline and augmented pipelines when the predicted bounding boxes are used. The table confirms the results from the OD and KD analysis: the baseline



(a) baseline



(b) augmented

Figure 10.15: RMSE distribution over different pixel mean values for the `lightbox` domain.

pipeline performs better than the augmented one on synthetic data, but it is outperformed on both testbed domains. This is expected since the augmented OD and KD models achieve better performance on `sunlamp` and `lightbox` images than their baseline counterparts. The performance improvement is therefore maintained after the integration.

As expected, the KD performance degrades when using the predicted bounding boxes instead of the ground truth ones. Compared to Table 10.4, the `sunlamp` mean and median RMSE display an increase of about 10 and 2.5 pixels, respectively. Similarly, the mean and median RMSE for the `lightbox` domain increase by 9 and 2 pixels, respectively. This is because of the imperfect accuracy of the OD model. The drop in inaccuracy is due to several factors:

- The OD network achieves a low IoU, sometimes close to 0. This makes it impossible for the KD network to predict the keypoint locations, as they are cropped out of the original image. An example is given by the `sunlamp` images with the lamp in the background. As already explained, such a feature misleads the OD network for some images, despite the augmentation pipeline mitigating the effect.
- In some cases, the OD network does not identify any object. This forces the pipeline to give the full image as input to the KD network, which was not trained on images with this scale but rather on perfectly cropped bounding boxes.
- Keypoints are cropped out by the predicted bounding box. Inaccurate OD can lead to a visible keypoint to be cropped out of the image. This makes the detection of such keypoint more challenging for the network.

Table 10.6: Pose estimation performance for the baseline and augmented pipelines on SPEED+. The pipeline marked with a * corresponds to the one using ground truth bounding boxes.

Pipeline	synthetic			sunlamp			lightbox		
	E_T [m]	E_R [°]	E_P [-]	E_T [m]	E_R [°]	E_P [-]	E_T [m]	E_R [°]	E_P [-]
baseline	0.046	1.392	0.033	0.706	41.246	0.837	0.726	31.106	0.657
augmented	0.049	1.480	0.035	0.319	12.582	0.268	0.251	9.101	0.202
augmented*	**0.016**	**0.814**	**0.019**	**0.101**	**4.430**	**0.096**	**0.080**	**4.124**	**0.088**

What these three factors have in common is the fact that inaccurate OD predictions do not accurately replicate the ground truth bounding boxes the KD network was trained on. The OD accuracy is therefore expected to significantly affect the performance of the entire pipeline.

Despite the performance degradation, there is still a clear improvement when applying the augmentation pipeline. The `sunlamp` mean and median RMSE are reduced by 65% and 75%, respectively. The `lightbox` mean and median RMSE are both reduced approximately by 56%.

Table 10.6 shows the pose estimation performance for the baseline and augmented models. The SQPnP solver, in combination with a Levenberg-Marquardt optimisation for pose refinement, was observed to yield the best performance. The initial pose reconstruction is performed using the six most confident keypoints, discarding the last one to refine the pose. Once again, the trend observed for the OD and KD models is confirmed: the baseline pipeline performs slightly better on synthetic data, with a difference below 1 cm and 0.1° for the position and attitude error, respectively, resulting in a minimal difference between the pose scores (0.033 against 0.035). However, a large discrepancy in performance emerges when testing the pipelines on testbed data. When not applying the selected augmentations, the position error more than doubles for the `sunlamp` images (from 0.319 m to 0.706 m) and almost triplicates for the `sunlamp` images, increasing from 0.251 m to 0.726 m. The gap gets even wider when looking at the attitude error, with the error increasing by 3.3 and 3.4 times for the `sunlamp` and `lightbox` domains, respectively. This leads to an overall 3.1 times increase in the `sunlamp` pose score and 3.3 times increase in the `lightbox` pose score. The augmented pipeline is therefore able to reduce both the translation and rotation error, leading to an overall improvement in pose reconstruction.

The table also includes the performance of the pose estimation pipeline when using the ground truth bounding box locations instead of the OD predictions. It was chosen to include the results for this pipeline after observing the drop in accuracy of the KD model following its integration with the OD model. The results presented in the table allow to get a sense for the pipeline performance when only considering HRNet and the pose estimation block. In line with the expectations, using perfect bounding boxes leads to the best accuracy across all domains. It reaches a position accuracy of 16 cm and a sub-degree attitude error, with a pose score of 0.019. The performance on testbed data increases even more remarkably: the error shrinks of a factor 3 on `sunlamp` images and a factor 2 on `lightbox` images. This confirms the idea that the OD network is currently the driving performance factor. By improving the model robustness on testbed data, the performance of the augmented pipeline should converge to the ideal case where perfect boxes are used.

In conclusion, the designed augmentation pipeline consistently improves the performance at every step, from OD to pose reconstruction. It allows the system to become more robust to testbed data by replicating effects present in `sunlamp` and `lightbox` images. Unlike the models trained only using basic augmentations, the final models show to have learned features that are meaningful at the testing stage. Generally speaking, it is therefore of primary importance to ensure that the training data distribution matches as closely as possible the testing one. By having more accurate training data, the performance gap can be expected to be filled, as proved by the data in Table 10.6. In fact, both augmented pipelines show a drastic reduction in the gap between the synthetic and testbed pose scores.

What the analysis has shown, however, is that applying synthetic augmentations does not completely remove outliers during testing. This can be attributed to the fact that (i) the illumination settings used to render SPEED+'s synthetic set are not representative of the testbed illumination conditions and (ii) synthetic augmentations cannot reproduce certain effects as accurately as including them at the rendering stage. This means that, to achieve an accurate and robust pose estimation pipeline fully trained on synthetic data, a potential solution is that of including relevant effects in the rendering environment, overcoming the limitation encountered on the SPEED+ dataset.

Table 10.7: IoU performance on ENVISAT for the `basic` and `realistic` OD models.

Model	basic		realistic		test	
	mean [-]	median [-]	mean [-]	median [-]	mean [-]	median [-]
basic	**0.944**	**0.955**	0.914	0.938	0.906	0.934
realistic	0.934	0.945	**0.924**	**0.941**	**0.921**	**0.949**

Table 10.8: RMSE on ENVISAT for the `basic` and `realistic` KD models.

Model	basic		realistic		test	
	mean [pix]	median [pix]	mean [pix]	median [pix]	mean [pix]	median [pix]
basic	**3.46**	**2.52**	**8.48**	**4.48**	9.50	4.58
realistic	6.04	3.73	8.66	4.56	**8.86**	**4.63**

10.3. Training with Photorealistic Rendering

The alternative approach consists of training on photorealistic synthetic images. This allows the inclusion of image corruptions directly at the rendering stage, simulating the observation conditions more realistically. This section compares the performance of the EfficientNetB3-SSD512 and HRNet-w32 when training on the `basic` and `realistic` subsets of the ENVISAT dataset introduced in Section 9.3:

- When training on the `basic` set, the dataset is augmented using the augmentation pipeline described in Section 10.2.3. Therefore, the baseline pipeline, *i.e.*, blur, noise, and brightness and contrast, is applied to both the OD and KD models. The OD network is trained only including the stars and Sun flare augmentations, and the KD is trained including the erasing, stars, style, and Sun flare augmentations. Note that, just like the training on SPEED+, the augmentations are applied only during training and not during validation.
- When training on the `realistic` set, only the baseline set of augmentations is applied to the OD and KD networks. The reason for not including any additional augmentation is that their only purpose is to simulate effects that are not captured in the dataset. However, when rendering images in a `realistic` simulation environment, these effects are already included.

As mentioned in Sections 7.3 and 8.4, the hyperparameters used to train the various models are similar to the ones used to train on SPEED+, with the only modification of the batch size. The OD and KD models were trained on batches with a size of 8 and 16, respectively. This change was dictated by the smaller dataset size. While SPEED+'s synthetic set counts almost 60,000 images, the `basic` and `realistic` sets of the ENVISAT dataset contain only 10,000. Reducing the batch size allows the model to perform more optimisation steps during a training epoch and be trained for a number of steps closer to SPEED+'s.

10.3.1. Performance Analysis

The following sections discuss the performance over the different steps of the pose estimation pipeline for the models trained on the `basic` and `realistic` sets.

Object Detection

Table 10.7 shows the OD results for cross-validation, *i.e.*, over the `basic` and `realistic` validation sets, and testing. The `basic` and `realistic` models perform better on their own validation set. The results on the `test` set, however, display a wider difference in performance. The gap grows up to a difference of almost 2% on the mean and median IoU. This result is consistent with the findings from Section 10.2: synthetic augmentations have a smaller impact on the performance of the OD network. The model trained on the `realistic` set is able to generalise better on more extreme illumination conditions than the model trained on the `basic` set with synthetic augmentations applied.

Keypoint Detection

Table 10.8 shows the cross-validation and testing results for the KD models. The `basic` model performs better on both validation sets. The error over the `basic` validation set almost doubles for the `realistic` model. Such a significant difference can be expected considering the `basic` model was trained on

Table 10.9: RMSE on ENVISAT for the `basic` and `realistic` KD models when bounding box predictions are used.

Model	`basic`		`realistic`		`test`	
	mean [pix]	median [pix]	mean [pix]	median [pix]	mean [pix]	median [pix]
`basic`	**4.86**	**3.46**	11.24	**5.91**	14.60	6.20
`realistic`	7.57	4.78	**10.32**	5.92	**11.10**	**6.02**

Table 10.10: Pose estimation performance on ENVISAT for the `basic` and `realistic` pipelines.

Pipeline	`basic`			`realistic`			`test`		
	E_T [m]	E_R [°]	E_P [-]	E_T [m]	E_R [°]	E_P [-]	E_T [m]	E_R [°]	E_P [-]
`basic`	1.446	11.318	0.239	2.345	17.720	0.376	2.971	21.571	0.458
`realistic`	**1.104**	**8.720**	**0.182**	**1.669**	**11.969**	**0.255**	**1.540**	**13.212**	**0.272**

an augmented version of the `basic` set. Surprisingly, the `basic` model is also slightly better on the `realistic` validation set. This difference, however, is well below the pixel-level accuracy and may be due to the randomness of the training process.

What can also be noticed is that the error increases from the `basic` to the `realistic` set is larger for the `basic` model (it has about doubled), which could be a sign of the lack of robustness of the `basic` model to unseen illumination conditions. This trend is confirmed by the RMSE on the `test` set, where the `realistic` model outperforms the `basic` one.

Note that the error over the validation sets is larger compared to the validation error observed in Table 10.4 for the SPEED+ dataset. In general, this can be attributed to the smaller dataset size, which limits the generalisation capabilities of the models on the `test` set. Additionally, the larger error observed on the `realistic` validation set can be attributed to the more challenging illumination conditions.

Again, the performance of the `basic` KD model confirms the findings of the experiments on synthetic augmentations: by applying extensive synthetic augmentations simulating meaningful effects for the network, the performance gap with testing data can be largely reduced. In fact, the `basic` model achieves arguably comparable performance with the `realistic` one. However, the two models display a different level of robustness to changes in illumination conditions. The `basic` model, performing significantly better on the `basic` set, sees a larger increase in RMSE when assessed on the `realistic` and `test` sets, with the error on the latter being larger than the `realistic` model.

Pose Estimation

Table 10.9 shows the KD performance for the `basic` and `realistic` models using the bounding box predictions. The displayed performance is in accordance with the separate analysis of the OD and KD models. The `basic` model keeps performing better on its own validation set. This is to be expected since the behaviour was already observed for both the OD and KD models. The `realistic` model outperforms the `basic` one on both the `realistic` and the `test` sets. Again, this is to be expected since the `realistic` OD model was better than the `basic` one and there was a small difference between KD models. The difference in pixel, however, has grown especially for the `test` set where it grows up to 3.5 pixels for the mean RMSE.

Despite maintaining a KD accuracy comparable with the `realistic` model across all domains, the `basic` model is outperformed when integrating the OD and KD models together. Even though less pronounced compared to the Tango scenario, the pipeline experience again a drop in performance at the integration step, with the OD model limiting the KD performance. The `realistic` model, on the other hand, displays a less evident loss in performance. In fact, the performance difference between the `basic` and the `test` set is only 3.5 pixels on the mean RMSE, against the almost 10 pixels for the `basic` model. The `basic` model is unable to maintain consistent performance across the different sets.

Table 10.10 shows the pose estimation performance of the `basic` and `realistic` pipeline. Surprisingly, the `basic` model is now also outperformed on its validation set, with the `realistic` pipeline achieving a reduction in pose score as large as 0.05. This can be explained by the lack of robustness of the `basic` pipeline, with different outliers causing the pose score to grow significantly.

The difference grows even wider when evaluating the `realistic` and `test` sets, where it reaches 0.12 and 0.17, respectively. The `realistic` pipeline reduces both the translation and rotation errors,

Table 10.11: IoU performance on ENVISAT for the `basic` and `realistic` OD models over different exposure ranges.

Model	Exposure range					
	[-3, 3]		[-3.2, 3.2]		[-3,4, 3.4]	
	mean [-]	median [-]	mean [-]	median [-]	mean [-]	median [-]
`basic`	0.906	0.934	0.909	0.937	0.900	0.934
`realistic`	**0.921**	**0.949**	**0.920**	**0.939**	**0.916**	**0.938**

proving how including realistic illumination conditions in the rendering environment allows to achieve better performance.

Overall, the performed analysis shows how the `realistic` model has better generalisation capabilities than the `basic` model. The performance on the `test` set, which includes more extreme illumination conditions compared to training of both pipelines, hints at the importance of correctly simulating lighting when rendering synthetic images. Applying state-of-the-art augmentations during training allows to only partially compensate for the lack of realism in the training set.

Furthermore, the robustness displayed by the `realistic` pipeline is such to outperform the `basic` model on its own validation set. Note that this difference is not clearly visible by the performance of the OD and KD models alone, but only appears when integrating the different elements of the pipeline. Although achieving good performance individually, the `basic` pipeline suffers from a larger performance degradation at the pipeline integration stage. This further emphasises the superiority of the `realistic` pipeline which, despite achieving a larger KD error on the `basic` set, produces more consistent keypoint location predictions, eventually resulting in a significant improvement in the pose estimation performance.

10.3.2. Sensitivity Analysis

The ENVISAT dataset generated in this work contains three different sets of synthetic images: `basic`, `realistic`, and `test`. Their main difference consists of the different illumination settings used in Blender to simulate on-orbit observation conditions with a higher or lower degree of realism. Despite the `test` set being conceived to test the `basic` and `realistic` pipelines on previously unseen illumination conditions, lighting can change dramatically during proximity operations. This can be caused by a variety of factors, such as the target tumbling uncontrolled, the change in relative position with respect to the Sun, and so on. Since it can be difficult to accurately predict the illumination conditions the network will perform inference on in space, it is necessary to test the pipeline over a broader range of illumination conditions to predict the pipeline performance with higher certainty. For this reason, this section discusses the results and findings of the additional testing performed on the two pipelines.

As mentioned in Section 9.3, the exposure range was observed to be the parameter affecting the illumination conditions the most. For this reason, the two additional sets presented in Section 9.3.3 are used for the sensitivity analysis. They are generated with the same rendering settings as the `test` set, with the only difference being the exposure range used to render the dataset. The two sets are generated using an exposure range of [-3.2, 3.2] and [-3.4, 3.4], respectively. As for the nominal case, the analysis is performed at every step of the pipeline. The results of this analysis can be seen as an intermediate step before testing the pipelines on testbed data and formulating reasonable expectations about their performance.

Table 10.11 shows the OD performance of the `basic` and `realistic` models on the two additional sets. The performance on the `test` set is also reported for comparison. It can be seen how both models do not seem affected by the change in exposure range, with the mean and median IoU changing slightly across the different sets. This means that the OD model is sensitive to specific changes in illumination conditions. This was already clear at the stage of the analysis on the synthetic augmentations applied to SPEED+, where the performance of the OD model were observed to change less significantly compared to the KD model.

Table 10.12 shows the KD performance of the two models on the different sets. It can be seen how the `realistic` model achieves a better average RMSE across the three sets, while the `basic` model achieves a slightly better median RMSE on the `test` set and the first additional set, becoming slightly worse on the final set. Both models maintain the same level of performance across all three sets, confirming the hypothesis that the developed augmentation pipeline significantly reduces the performance gap in KD

Table 10.12: RMSE on ENVISAT for the `basic` and `realistic` KD models over different exposure ranges.

Model	Exposure range					
	[-3, 3]		[-3.2, 3.2]		[-3.4, 3.4]	
	mean [pix]	median [pix]	mean [pix]	median [pix]	mean [pix]	median [pix]
`basic`	9.50	**4.58**	9.43	**4.53**	9.73	4.82
`realistic`	**8.86**	4.63	**9.24**	4.58	**9.40**	**4.78**

Table 10.13: RMSE on ENVISAT for the `basic` and `realistic` KD models over different exposure ranges when bounding box predictions are used.

Model	Exposure range					
	[-3, 3]		[-3.2, 3.2]		[-3.4, 3.4]	
	mean [pix]	median [pix]	mean [pix]	median [pix]	mean [pix]	median [pix]
`basic`	14.60	6.20	13.59	6.12	15.25	6.47
`realistic`	**11.10**	**6.02**	**11.08**	**5.87**	**11.50**	**6.08**

Table 10.14: Pose estimation performance on ENVISAT for the `basic` and `realistic` pipelines over different exposure ranges.

Pipeline	Exposure range								
	[-3, 3]			[-3.2, 3.2]			[-3.4, 3.4]		
	E_T [m]	E_R [°]	E_P [-]	E_T [m]	E_R [°]	E_P [-]	E_T [m]	E_R [°]	E_P [-]
`basic`	2.971	21.571	0.458	2.610	20.317	0.427	2.814	20.900	0.444
`realistic`	**1.540**	**13.212**	**0.272**	**1.625**	**13.503**	**0.279**	**1.667**	**13.522**	**0.281**

and that the `realistic` model generalises well on unseen and more extreme illumination conditions.

Table 10.13 shows the KD performance of the `basic` and `realistic` models after the integration of the OD and KD models. Once again, the table shows how the better performance of the `realistic` OD and KD models individually results in better performance after integration compared to the `basic` model. The `realistic` model, in fact, never crosses the threshold of 12 pixels as average RMSE, whereas the `basic` model never goes below 13.5 pixels. What can also be observed is how the `realistic` model maintains an average RMSE roughly constant across all image sets, ranging from 11.08 to 11.50 pixels. The variance is similar for the `basic` model, although the error varies slightly more between 13.59 to 15.25 pixels.

The performance difference is amplified when performing pose estimation. Table 10.14 shows the performance of the pose reconstruction step on the different sets, proving how a difference in KD error of a few pixels translates into a broader gap in pose estimation error. Independently of the illumination settings used, the `realistic` model consistently performs better than the `realistic` one. For all three sets, the translation and rotation errors as well as the pose score do not change substantially.

At the end of the sensitivity analysis, the `realistic` pipeline confirms its superior performance over the `basic` one. Both models have been observed to behave well to changes in the rendering illumination conditions, with no drastic changes in performance. The findings from the nominal performance analysis also hold for the different tried illumination settings. Training on photorealistic images benefits the overall pipeline performance, mainly because of the better robustness achieved by the OD and KD network. Despite the performance of the KD networks using the predicted bounding boxes being only a few pixels apart, the outlier predictions made by the `basic` model result in a significantly worse pose score and consequently in a lower accuracy of the entire pipeline.

10.4. Conclusions

In conclusion, the analysis of the developed pose estimation system for spacecraft relative navigation, as evidenced by the results obtained in the Tango and ENVISAT scenarios, yields several valuable insights. Firstly, the incorporation of synthetic augmentations has a discernible positive impact on system performance, particularly in KD, although its influence on OD is less pronounced. However, the design of the augmentation pipeline is not without challenges, as some augmentations do not significantly enhance performance, and in some cases, even lead to performance degradation.

In terms of meeting accuracy requirements, the augmented pipeline successfully satisfies the criteria

for object and KD in the Tango scenario. Nevertheless, shortcomings persist in the accuracy of pose estimation when applied to testbed data, notably in the `sunlamp` images, where relative position error remains a concern. The pipeline also struggles to meet the relative attitude error requirement in these scenarios, revealing the need for further improvement.

A key revelation is that, when employing ground truth bounding boxes, the pipeline aligns with accuracy requirements. This emphasizes the pivotal role of the OD network's accuracy in determining the overall system performance, making it the primary limiting factor in the current pipeline's capabilities.

The analysis underscores the disparity between the synthetic SPEED+ images and the distribution of testbed data. This necessitates the application of synthetic augmentations to replicate missing real-world effects, such as extreme illumination conditions. However, even with state-of-the-art augmentations, there is room for improvement. The potential solution lies in the adoption of a more `realistic` rendering environment at the data generation stage, as highlighted in Section 10.3. The experiments confirm the value of transitioning to photorealistic rendering, which significantly enhances system performance by allowing better generalization to previously unseen illumination conditions, translating into marked improvements in pose estimation.

In summary, the research underscores the potential of pose estimation systems driven by CNNs and synthetic data. It suggests that with the rapid advancements in ML architectures, a slightly more accurate model could feasibly meet all the system's performance requirements, thereby paving the way for the development of scalable and robust relative navigation systems.

11

Architecture Tradeoff

This chapter covers the main outcomes of the architecture tradeoff as well as the implementation details to fully characterise the performance of the different models on Myriad X. Section 11.1 provides some context about the importance of performing such a tradeoff, while Section 11.2 describes the pipeline implemented to convert each model from the training to the deployment format. Sections 11.3 and 11.4 present and discuss the results of the experiments aimed at characterising every model's inference speed and accuracy. Section 11.5 summarises the main findings of the chapter.

11.1. Introduction

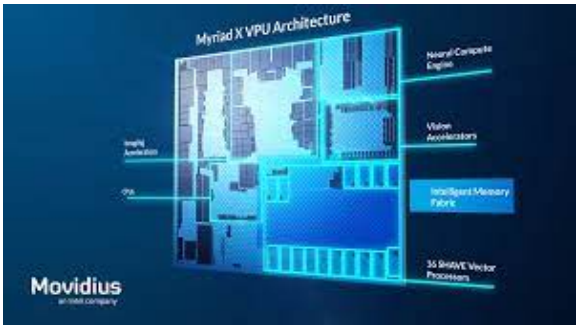
When designing a ML pipeline for spacecraft pose estimation, it is important to not only consider the model performance but also the device on which the model is going to be deployed. Especially when taking into account the severe on-board power constraints usually in place, ensuring the model runs properly on the hardware is a key aspect of the development process. This cannot be done by simply testing the model on the same device as for training, *e.g.*, a GPU, since it is unlikely that it will meet the requirement for on-orbit inference. Instead, space-embedded systems often rely on *edge* devices, such as Vision Processing Units (VPUs), which offer more power-efficient configurations.

There are several challenges related to making the model compatible with such edge devices. First, the model needs to be converted to a compatible format. However, it is unlikely that the performance will stay the same as right after training. Model conversion pipelines often rely on converting the data type of the model weights, and sometimes even modifications to the original architectures. A loss in precision can therefore be expected throughout the process. Second, the hostility of the space environment makes it necessary to test the hardware against the expected level of radiation to ensure its behaviour is compliant with space standards.

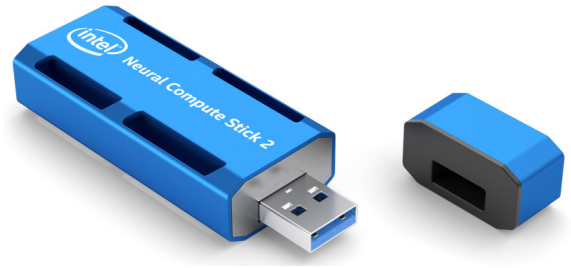
VPUs are optimised for CV tasks on the edge (Minaidis, 2022). They combine general-purpose cores with specialised ones and hardware accelerators, have low-power memory, and focus on low-latency data transport. Consistently with these principles, the latest VPU released by Intel, Myriad X, is based on a low-power design and a multicore architecture involving heterogeneous types of cores. Combined with a reduced area footprint and a wide range of peripherals interfaces, Myriad X is one of the most promising choices when it comes to select processors for on-board inference. An illustration of the VPU architecture can be found in Figure 11.1a.

Because of their design, VPUs outperform both CPUs and GPUs in performance per Watt Furano et al. (2020). This, however, comes at the expense of inference speed, as VPUs can be significantly slower than GPUs. Nevertheless, VPUs are considered an emerging alternative to run expensive algorithms on the edge, including for space applications. Cassinis (2022) managed to convert a KD model for spacecraft pose estimation to run inference on Myriad X, collecting valuable insights about the model inference time and accuracy performance. The author also proved how Myriad X is suited for ML model deployment, with the network accuracy experiencing only a mild reduction compared to inference on a GPU, likely caused by the model compression to FP16. In a more general sense, Myriad X has been qualified to operate in a space environment.

Another important aspect related to model deployment concerns architecture optimisation. Several



(a) Myriad X architecture (Cassinis, 2022).



(b) Intel Neural Compute Stick.

Figure 11.1: Illustrations of the Myriad X device.

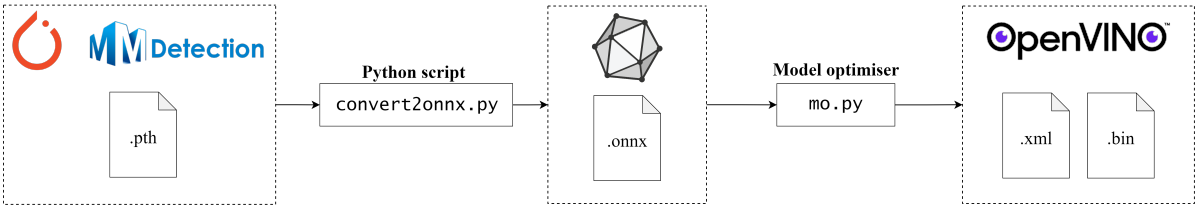


Figure 11.2: Myriad X model conversion pipeline.

architectures should be selected and considered for the final choice. The more information is taken into account, the more thorough the final decision is going to be. Typically, architecture tradeoffs take into account model accuracy and size. However, deployment on the target device should also be taken into account. The device considered for the experiments is the Myriad X VPU, provided by Ubotica Technologies[1]. The processor comes with a USB stick (see Figure 11.1b[2]), Intel's Neural Compute Stick (NCS), which can be easily plugged in a laptop.

11.2. Model Conversion

The model conversion from Pytorch and MMDetection to the Myriad-compatible OpenVINO Intermediate Representation (IR) format is illustrated in Figure 11.2 and consists of several steps. To begin with, the model trained on a GPU is converted from the native to the `.onnx` format. This step is required because OpenVINO does not support conversions directly from Pytorch's `.pth` format. In addition to this, the model parameters are also compressed to FP16, as Myriad X does not support FP32 precision. Then, the model is optimised using the model optimiser tool[3]. It performs a detailed analysis of the model's structure and makes the necessary adjustments to achieve optimal deployment. For example, it merges different layers when they can be replaced by a single operation to minimise the complexity of the computation graph. The tool outputs the optimised model in OpenVINO's IR format: a `.xml` file containing information on the model topology and a `.bin` file containing the model parameters. Converting a Pytorch model to ONNX can be achieved through the framework's dedicated API. Pytorch includes an `export` function performing direct model conversion. This strategy was adopted for both OD and KD models. The only difference between the two pipelines is that EfficientNetB3-SSD512 had to be loaded using MMDetection, whereas the KD models could be loaded directly using the implemented `build_model` function.

The final NMS operation for the OD model could not be exported, meaning the ONNX model output consisted of the unfiltered bounding boxes and classification scores. Although not a problem for measuring the inference time - given the overall number of layers, NMS can be expected to have a negligible impact on speed - the model cannot be used when verifying the post-conversion prediction

[1] `https://ubotica.com/` (visited on 11/10/2023)

[2] `https://www.intel.com/content/dam/develop/public/us/en/images/hardware/hardware-movidius-ncs-gen2-16x9.png` (visited on 16/09/2023)

[3] `https://docs.openvino.ai/2023.0/openvino_docs_model_optimization_guide.html`, (visited on 14/07/2023)

Table 11.1: Results of the inference time experiments for the different architectures.

	Model	Number of parameters [Mn]	Loading time on VPU [s]	Inference time [ms]		Energy consumption [J]	
				VPU	GPU	VPU	GPU
OD	EfficientNetB3-SSD512	10.8	41.4	830.4	33.4	2.1	10.0
KD	HRNet-w18	9.3	3.9	135.3	22.1	0.3	6.6
	HRNet-w32	28.5	4.4	157.9	22.7	0.4	6.8
	LPN50	2.9	34.8	251.9	-	0.6	-
	LPN101	5.3	52.3	287.6	-	0.7	-
	LPN152	7.4	70.4	290.9	-	0.7	-

accuracy. For this reason, the full inference pipeline was extended applying NMS to the model output.

An HRNet architecture was already converted and run on Myriad X by Cassinis (2022). The conversion pipeline implemented in this work confirms the author's findings, as all KD models were converted without particular issues.

After converting all models to ONNX, every architecture was refined using OpenVINO's model optimiser. The tool can be directly used from the command line by passing the relevant settings as input arguments. At this stage, the EfficientNet and HRNets models did not experience any bottleneck, while for the LPN models not all merging operations were successful. In particular, a few operations in the deconvolution layers could not be simplified by OpenVINO. This is expected to have an impact on the network inference time on Myriad, as the scope of the optimisation is to minimise the inference time.

11.3. Inference Time Experiments

The experiments on inference speed are performed using OpenVINO's benchmarking tool[4]. Similarly to the model optimiser, it can be used directly from the terminal. The scope of the experiments is to characterise the inference speed of each model on both a VPU and a GPU. This allows not only to study the pipeline performance on the target device, but also to quantify the energy savings by using a VPU instead of a GPU.

All models were tested on Myriad using randomly generated inputs. The input size was the same as mentioned in the implementation section: $3 \times 640 \times 640$ for EfficientNetB3-SSD512 and $3 \times 256 \times 256$ for all KD models. The models were tested on 500 inference requests. Furthermore, to reproduce a representative scenario for on-orbit inference where only one image at a time is captured and used as measurement, parallelisation is removed, meaning Myriad only processes one request at a time. This is done by setting the corresponding input argument, `nreqs`, equal to 1.

For testing on GPU, a Python script was written leveraging OpenVINO's Python API. The script loads the model to the device and iterates over the first 500 images of SPEED+'s `synthetic` validation set. For each image, the inference time is measured and saved to an output file. Once inference is performed over all images, the average inference time is computed.

Table 11.1 lists the results of the inference time experiments for the different architectures. They are split into OD and KD models for clarity.

HRNet-w18 takes the least amount of time to load and run inference on Myriad X. HRNet-w32 has a slightly higher loading time and takes roughly 20 ms to run a single inference, as can be expected by its larger size. Surprisingly, EfficientNetB3-SSD512 is significantly slower than the two HRNet models in terms of loading and inference time on the VPU, despite having a number of parameters comparable with HRNet-w18. Compared to it, the OD model takes up to 10 times longer to be loaded to Myriad X and one inference takes 6 times as long. A possible explanation can be found in how the OD algorithm works. The model generates thousands of bounding boxes and performs classification on each of them. Therefore, the number of operations performed for a single inference request can be expected to be larger compared to a KD model, where a single forward pass is performed for every inference request. Inference time on GPU scales much better with the number of parameters compared to the VPU. Despite

[4] `https://docs.openvino.ai/2023.0/openvino_inference_engine_tools_benchmark_tool_README.html`, (visited on 14/07/2023)

the model size being more than tripled, HRNet-w32 is able to perform inference with a minimal time increase compared to HRNet-w18, while the OD model takes only 50% more time to run.

The table also reports the energy consumption for a single inference on a VPU and a GPU. This metric is included to give an idea of the energy resources required to run on-orbit inference, and therefore provides useful insights to select the right architecture not only in terms of performance but also of its energy footprint. The worst-case power required by the two devices is set to 300 W for NVIDIA Tesla, value taken by the company datasheet[5], and 2.5 W for Myriad X, consistently with what assumed by (Minaidis, 2022). Given the different application scenarios for which the two devices are conceived, with the GPU being used in an HPC cluster and the VPU designed to run in systems with limited power, it is not surprising to find a difference of two orders of magnitude between their power consumption.

Because of the 120× increase in power consumption, running every model on the GPU ends up requiring more power compared to a VPU, with EfficientNetB3-SSD512 requiring as much as 10 J for every inference. Using a VPU allows to save more power and energy, regardless of the architecture considered. The most remarkable power saving occurs for KD models, where Myriad X is able to reduce the energy consumption by 22 times for HRNet-w18 and 17 times for HRNet-w32.

The performance of LPN models is hardly comparable with the other models. Despite being the most lightweight models, they surprisingly take more time to be loaded to Myriad and perform inference more slowly. Although the model is one tenth as large as HRNet-w32, LPN50 takes about 8 times more to load to the VPU and displays an increase in inference time by 60%. The larger models, LPN101 and LPN152, take even more to load and run inference. Such poor performance is likely caused by the modifications to the original architecture described in Section 8.1 to ensure LPN models have the same input and output resolution as the HRNet models.

The default LPN50 architecture, however, could only run on Myriad X with an input size of 220×220 pixels or lower. The reason for the observed behaviour is the presence of a softmax layer in the lightweight bottleneck block whose size increases with the input resolution. When using the same input size as HRNet (256×256), the layer reaches a shape of 1×4096, which ended up being too large to fit on Myriad as an individual layer. The adopted solution, as introduced in Section 8.1, was to replace the GC with the SE block in lightweight bottleneck layers, removing softmax from the architecture. In addition to this, to increase the output resolution it was chosen to add two additional deconvolution layers. Starting from an architecture already poorly optimised, in combination with non-optimal support from OpenVINO to deconvolution operations, resulted in a poor overall optimisation. Because of the poor performance exhibited, it was therefore selected to discard LPNs from the final tradeoff after the performance on the VPU.

When integrated with the OD model, both HRNets are able to meet the inference time requirement of 1.5 s. HRNet-w18 would achieve an overall inference time of 0.966 s, against the 0.988 s achieved by using HRNet-w32. The two KD models also achieve a similar worst-case scenario energy consumption, making the hardware limitations not a primary constraint in the architecture selection process.

11.3.1. Copy Operation

A behaviour observed for all models is the presence of *copy* operations in the network forward pass. This operation does not correspond to any layer in particular but rather is triggered when the memory required by a certain operation does not fit on Myriad's Connection Matrix (CMX) memory and has therefore to be assigned to the Double Data Rate (DDR) memory. How this process is implemented is related to the architecture of the Myriad processor itself. As shown in Figure 11.3, the LEON processor, in charge of the VPU memory management, is connected to both the CMX and DDR memory. CMX communicates directly with the neural compute engine, hardware accelerators, and Streaming Hybrid Architecture Vector Engine (SHAVE) processors, making it a more efficient way to store data that will be soon used for computations. DDR, on the other hand, is more peripheral and therefore transfers data less efficiently. Efficiency, however, comes at the expense of storage space, since the CMX memory can only store up to 2.5 MB of data, whereas the space in the DDR memory goes up to 512 MB. This means that operations performed on large data may not fit on Myriad's fast-access memory, slowing the computation process.

[5] `https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf` (visited on 10/10/2023)

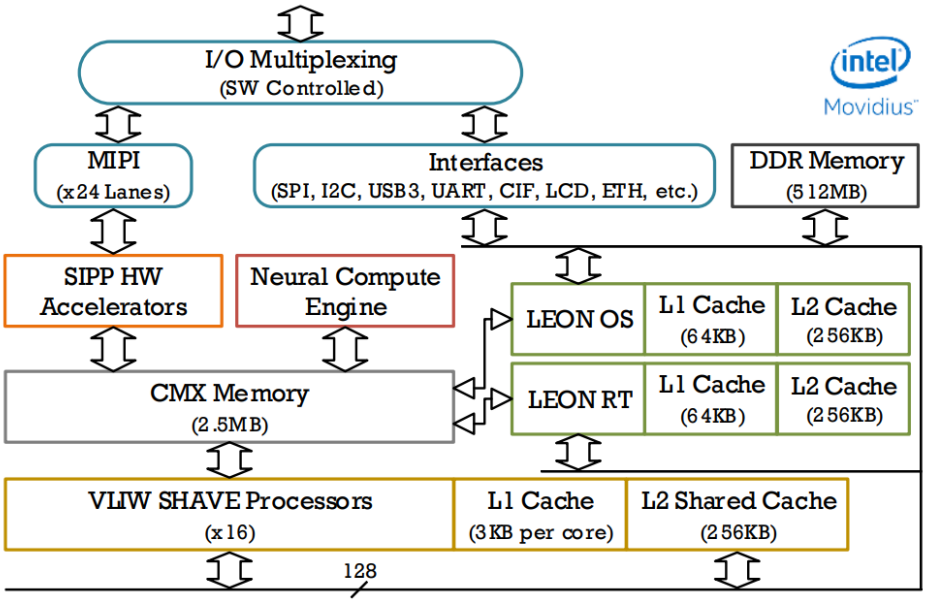


Figure 11.3: Illustration of the Myriad X architecture (Petrongonas et al., 2021).

Table 11.2: Pose estimation performance on SPEED+ for the different KD models. The pipeline marked with a * corresponds to the one using ground truth bounding boxes. Some examples from the literature are also included.

Model	`synthetic`			`sunlamp`			`lightbox`		
	E_T [m]	E_R [°]	E_P [-]	E_T [m]	E_R [°]	E_P [-]	E_T [m]	E_R [°]	E_P [-]
HRNet-w18	0.061	2.040	0.048	0.372	18.147	0.375	0.301	12.977	0.280
HRNet-w32	0.049	1.480	0.035	0.319	12.582	0.268	0.251	9.101	0.202
HRNet-w32*	**0.016**	**0.814**	**0.019**	**0.101**	**4.430**	**0.096**	**0.080**	**4.124**	**0.088**
SPEED+ baseline	0.050	1.520	0.040	0.850	47.750	0.980	0.970	34.710	0.770
SPNv2 ($\phi = 3$)	0.054	0.987	0.027	**0.225**	11.065	0.230	**0.175**	**6.479**	**0.142**
SPNv2 ($\phi = 6$)	**0.031**	**0.885**	**0.021**	0.230	**10.369**	**0.219**	0.216	7.984	0.173

11.4. Accuracy Comparison

After the experiments on inference time, two pipelines were tested on SPEED+, one using an HRNet-w18 model and one using an HRNet-w32 model for KD. No LPN model was considered because of their poor inference time performance. While the latter was already trained to perform the domain-gap experiments discussed in Section 10.2, the former required to train an HRNet-w18 model. The network was trained using the training settings listed in Section 8.4. Once the training was over, both pipelines were tested on the SPEED+ validation and test sets.

The results, reported in Table 11.2, show how both pipelines demonstrated superior performance on `lightbox` images when compared to `sunlamp` images, suggesting that the features learned by the KD models were not significantly influenced by the architecture choice. However, it is evident that the use of HRNet-w18 as the KD model led to a decrease in accuracy due to its smaller size, resulting in consistently poorer pipeline performance compared to the HRNet-w32 model. The larger HRNet-w32 model exhibited reduced translation and rotation errors, ultimately leading to a lower pose score across both domains.

Regrettably, neither the HRNet-w18 nor the HRNet-w32 pipelines met the accuracy requirements outlined in Section 2.2 on testbed data:

- **SR10:** The pose estimation system shall have a relative position error of 10% of the minimum operating distance or lower.
- **SR11:** The pose estimation system shall have an error of 5° or lower for the relative attitude.

SR10 translates in a maximum allowed position error of 0.225 m for the Tango scenario, while the best-performing pipeline, HRNet-w32, only achieves a relative position error of 0.319 m and 0.251 m on

`sunlamp` and `lightbox`, respectively. Concerning **SR11**, neither model attains an accuracy on the relative orientation below 9°. Removing the OD error from the pipeline significantly improved performance and enabled compliance with accuracy requirements. This highlights the importance of enhancing the OD model's performance to achieve suitable results for deployment in a space environment.

Overall, given that both pipelines satisfied the inference time requirements, opting for HRNet-w32 over HRNet-w18 appeared to be the more logical choice. Although neither pipeline achieved the required accuracy, HRNet-w32 managed to reduce translation error by over 50 cm in both testbed domains and reduced the rotation error by as much as 5.5° on `sunlamp` images. By looking at Table 11.1, it can be seen how the final pipeline counts around 39.3 million parameters. This result is comparable with other pipelines from the literature, such as the one by Park et al. (2022b) where, depending on the specific model considered, the model size varies between 12 and 52.5 million. Furthermore, this is not expected to be a constraining factor since both the EfficientNet and HRNet models could be loaded on Myriad without experiencing any memory issue.

For comparison, the table also includes results from the literature. The SPEED+ baseline refers to the best-performing model in the work by Park et al. (2022c), which employed a HigherHRNet for KD followed by an EPnP solver. Then SPNv2 models were introduced by Park et al. (2022b). It makes use of an EfficientNet backbone, with the parameter ϕ in the table referring to the compound scaling parameters of the architecture introduced in Section 7.1.1. In comparison to the pipelines developed in this research, several observations can be made. All the pipelines outperform the SPEED+ baseline. Notably, the HRNet-w18 pipeline reduced the pose score achieved by the SPEED+ baseline by up to two-thirds on `sunlamp` and `lightbox` data. This underlines the limited generalization capabilities of the pipeline. The state-of-the-art SPNv2 achieved better performance than the two pipelines developed in this research. However, it is essential to acknowledge that the performance gap may not be as significant as indicated in Table 11.2, as the author utilized a subset of testbed images for domain refinement, slightly enhancing performance. Notably, the HRNet-w32 pipeline, employing perfect bounding boxes, outperformed SPNv2 in every metric, outlining the potential of future improvements for the developed system.

Originally, the accuracy comparison also included an accuracy verification on Myriad X. The experimental setup is the same as for the GPU, with the different models being tested on the first 500 images of the SPEED+ validation set. However, the performance were observed to vary significantly when running the model on the VPU. A possible cause is the model conversion to FP16, which may have led to an excessive loss of information and could have been avoided by implementing a quantisation-aware training strategy. By doing so, the model parameters are forbidden from going below a certain value, making the conversion process safer. The results of the experiment are reported in Appendix C. Despite such a performance drop, the insights gathered throughout the tradeoff still hold. The inference time is affected by the architecture and the data type of its parameters, regardless of the accuracy achieved, while the performance achieved on a GPU can still be used for comparison purposes.

11.5. Conclusions

Concerning architecture tradeoffs for spacecraft pose estimation systems, the experiments performed in this chapter make it evident that a robust decision-making process extends beyond just the model size and computational efficiency. There are nuanced factors that play a pivotal role in achieving optimal performance.

First, it was demonstrated that within a given family of architectures, smaller models generally exhibit faster inference times. This aligns with the conventional wisdom that reducing the model's complexity and size results in improved speed. However, the absolute performance strongly depends on how well the architecture aligns with the specific requirements and constraints of the target device. Smaller models may not always be the best choice, as their specific implementation may include operations not fully supported by the edge device of choice.

Moreover, the experiments highlighted the importance of avoiding architectures with large or input-dependent layers. These layers impose limitations on the input size and might affect the performance of the entire system. Similarly, model operations that are not adequately supported by the hardware or the conversion pipeline can substantially impede overall performance. This emphasizes the necessity of aligning the model's operations with the capabilities of the hardware to ensure efficient execution. The analysis also revealed that the heritage of an architecture can offer valuable insights into its compatibility

with the target device and the complexity of the conversion process, such as is the case for HRNet-w32.

Regarding the OD model, no actual tradeoff was performed as only one architecture was considered. Nevertheless, characterising its performance on Myriad X was essential for determining the overall pipeline inference time, providing valuable insights into the system's behaviour.

Even though LPN models were initially excluded after the inference time experiments, a KD model still had to be chosen between the HRNet-w18 and HRNet-w32. As anticipated, the smaller HRNet-w18 model exhibited a faster inference time, but at the cost of pose estimation performance. The overall inference time requirement did not emerge as a limiting factor for HRNet models, allowing to prioritize model accuracy. This made HRNet-w32 the optimal choice for the final system architecture, as it complied with the inference time requirement while coming close to achieving acceptable accuracy. As discussed in Section 10.2, enhancing OD accuracy remains the primary driver for the entire pipeline performance. Ideally, employing an HRNet-w32 model in conjunction with a more accurate OD model enables the pipeline to not only meet but potentially exceed the accuracy requirements.

Part V

Concluding Remarks

12

Conclusions and Future Work

This chapter marks the conclusion of this thesis, summarising the primary findings of the study and suggesting areas for future research. Section 12.1 delves into the key conclusions, addressing the main research question and related sub-questions, while Section 12.2 provides recommendations aimed at enhancing the developed CNN-based pose estimation pipeline and its applicability in a real space mission.

12.1. Conclusions

The following research question was identified in Section 1.3 and served as primary guidance throughout this thesis:

How can an accurate and robust CNN-based relative navigation system be developed to perform RPO missions in a way that is scalable and optimised for space hardware?

During this thesis, a system reconstructing the relative pose of the on-board camera with respect to an uncooperative target spacecraft was designed and developed. It relies on images coming from an optical camera containing the target. To reconstruct the pose, it makes use of an OD and a KD network to extract the keypoint pixel locations, which are then passed to a pose solver with the purpose of determining the distance and orientation between the camera and the target.

It is not the first time that this high-level architecture has been designed. Indeed, several examples can be found in the literature based on the same principle of feature extraction through an OD and a KD model followed by a pose reconstruction step. To the author's knowledge, however, this is the first pipeline developed using the following elements:

- An OD model consisting of an EfficientNetB3 backbone with an SSD512 detection head.
- An HRNet-w32 model for KD.
- A SQPnP pose solver, followed by a Levenberg-Marquardt optimisation for pose refinement.

The system was developed and tested on two different scenarios. One assuming the Tango spacecraft and the other assuming ENVISAT as targets. Although the operating conditions were assumed to be fairly similar, *i.e.*, the two spacecraft follow similar orbits and the operating range for the two cases is comparable, this required making some adjustments to the system. Despite that, the pipeline proved to work on both scenarios achieving similar performance. The OD and KD models were trained without any additional bottleneck when moving from one target to another and the pose solver smoothly adapted to the different 3D spacecraft model and number of keypoints. This can be seen as proof of the general applicability of this system outside of the scenario it was originally conceived for, hence making its design more scalable to perform proximity operations at scale.

To help answering the main research question, the following sub-questions were also identified in Section 1.3. They are reported along with the corresponding insights gained during the work:

1. **How can the domain gap between synthetic and testbed images be bridged using state-of-the-art data augmentations?**

At the beginning of this thesis, a significant gap was identified between the performance on synthetic and testbed images of existing methods. The literature attributes this behaviour to challenging background and illumination conditions.

Attempting to bridge such a domain gap, the methodology found to be the most promising is to apply heavy data augmentations to the training data with the aim of replicating specific features observed in real images, such as detector noise or pixel saturation due to extreme illumination.

Several augmentations from the current literature were considered since they were specifically designed to include realistic corruptions. Then, an analysis of the impact of the different augmentations on OD and KD was performed, showing how including realistic effects in the training data significantly improves performance. This allowed to design the optimal OD and KD augmentation pipeline and achieve an error reduction in keypoint extraction by up to 75%. This was attained with an augmentation pipeline as simple as applying every augmentation independently with equal probability, leaving room for improvement through finetuning of the augmentation parameters.

In particular, augmentations related to adverse illumination conditions, such as the Sun flare one, were observed to be particularly effective on both OD and KD, while augmentations related to the image background benefited OD the most, with the inclusion of random stars being the clearest example. In general, KD was affected more than OD by synthetic augmentations, a behaviour that can be attributed to the already good performance of the baseline OD network.

It was also observed how not all the considered augmentations seemed to be effective in bridging the performance gap, but in some cases led to a performance degradation. This emphasises the necessity to carefully design a data augmentation pipeline before training the final model, as the type of features NNs are sensitive to can hardly be determined prior to training.

2. **Can the performance be further improved by training the models on photorealistic synthetic images?**

   Despite the promising results, the methodology based on synthetic augmentations was identified to have a bottleneck related to the maximum achievable realism. For this reason, an alternative methodology was developed based on rendering photorealistic images, including all the relevant effects thanks to the realism of the rendering environment instead of applying them individually.

   This led to the generation of a novel synthetic ENVISAT dataset, conceived to compare the performance of two identical pose estimation architectures, with the only difference being the approach used to include real-world corruptions. The dataset consists of 21,000 images in total, split between three domains differing from each other for the illumination settings used during the rendering in Blender. The `basic` set (10,000 images) is generated using simple lighting, while the `realistic` (10,000 images) and the `test` (1,000 images) sets were generated including more challenging illumination.

   The results proved how photorealistic rendering further improves the pose estimation performance. Such a result is hardly surprising, as the level of realism achieved by rendering software nowadays allows to generate high-fidelity synthetic data. The outcome is corroborated further by the sensitivity study performed, where the two pipelines were tested on two additional sets of synthetic images, rendered with even more extreme illumination settings than the `test` set.

   This finding is foreseen to be of primary importance for the generation of a new state-of-the-art dataset for spacecraft pose estimation. Synthetic data should replicate more closely real images, allowing models to easily generalise on real-world scenarios and improve their robustness.

3. **What are the pitfalls and bottlenecks of a model conversion pipeline from the training format to a format compatible with the target device?**

   Numerous architectures were also converted to the OpenVINO IR format to perform experiments on the Myriad X VPU, gathering several important findings.

   First, it was observed how a conversion pipeline can involve several steps. Since OpenVINO does not support model optimisation directly from `.pth` format, it was necessary to convert the models to the `.onnx` format first. This type of inconsistency adds unnecessary complexity to the pipeline and should therefore be limited. Taking the pipeline in Chapter 11 as an example, an alternative

solution could be to train the model in a framework supported by OpenVINO, *e.g.*, Tensorflow, or use a device where the model can be deployed in `.onnx` format.

Second, multiple conversions are expected to cause a drop in performance. In particular, attention should be paid to which data types are supported by the target device. In the case of Myriad X, the hardware only supports data in half-precision format. This means that the model parameters require to be converted to a lower-memory representation to ensure successful deployment, which may cause a loss in accuracy. A potential solution is that of using quantisation-aware training, where the model weights are forbidden from going below a certain threshold to avoid a catastrophic loss of information during the conversion process.

4. **What are the best practices when performing a tradeoff between a model's accuracy and inference speed on the target device?**

   With their performance on the target device fully characterised, the different architectures were subject to a final tradeoff, which showcased the importance of taking into account the deployment on the target device in the selection process.

   Surprisingly, the assumption that smaller models are faster turned out to be inaccurate. Instead, the tradeoff showed how it is desirable to test each model on the target device prior to training to spot potential incompatibilities. Because of the memory limitations of edge devices such as the Myriad X VPU, a more suited approach would be to avoid models containing large or input-dependent layers. In other words, one should not only try to minimise the model size and complexity but also suit it as much as possible to the target device.

   Although an important requirement, the overall pipeline inference time was observed not to be a driving factor for the choice of the final architecture. Even a family of architectures as poorly optimised as LPNs managed to meet the inference time requirement when combined with EfficienNetB3-SSD512. The accuracy requirements, on the other hand, turned out to be a crucial factor to consider. The pose estimation pipeline only achieves a satisfactory level of performance when using ground-truth bounding boxes instead of the OD network. This emphasises the importance of maximising the OD performance prior to the other steps in the pipeline.

The system developed in this research achieves pose estimation performance comparable with other methods from the literature, with yet room for improvement to establish a new benchmark in spacecraft pose estimation. This is done by entirely training the OD and KD models on synthetic data, which empowers researchers to develop new pipelines with minimal resources required. By carefully implementing a realistic rendering environment, synthetic images are expected to fill the gap with testbed ones, enabling on-ground validation of relative navigation systems.

The system was also proven to comply with the limitations of space hardware. The validation process allowed the gathering of precious information for future work related to compatibility with the target hardware and architecture selection. Combined with a sensor suite as simple as a single vision-based camera, the system architecture is concluded to be feasible for space applications as well as cost-effective, and it can be used as a starting point to employ relative navigation systems at scale.

Using this work as a baseline, future research may improve the system and get closer to the goal of autonomous and scalable relative navigation for proximity operations.

12.2. Future Work

This thesis does not intend to exhaustively resolve the issues associated with developing a CNN-based relative navigation system. Instead, it provides insights into several gaps that are expected to ignite further research. This factor, combined with the numerous assumptions and limitations made to complete the work within a reasonable timeframe, led to the identification of several points relevant for future research:

- **OD performance:** The performance of the OD network were identified to be the major limiting factor for the pipeline. Future research should focus on developing a more accurate OD model, allowing the pipeline to meet the accuracy requirements.
- **Try a different feature extraction design:** Despite the system developed in this thesis using a feature extractor composed by an OD and a KD network, different architectures have also achieved interesting results. An example is given by Park et al. (2022b), who use a multi-task head network

to perform multiple tasks, such as OD, KD, and pose regression. Training CNNs on multiple tasks usually improves model's generalisation and may be the reason behind the observed performance gap.

- **Integration with relative navigation:** The developed pose estimation pipeline should be integrated within a relative navigation loop to assess the performance of the relative navigation system.
- **Recurrent neural networks:** Recurrent neural networks can be used to improve the quality of predictions by taking into account the temporal dimension. The output from previous epochs, in fact, is used by the network to make predictions at the current epoch. This is not only helpful for integration within the relative navigation loop but also for making the network more robust by taking into account the estimated pose at previous epochs.
- **Use of a more realistic ENVISAT model:** The use of an open-source textured model clearly poses limitations to the accuracy of the simulated scenario: material properties might differ slightly, surface features could be different, and the difference in size between the model and the actual satellite complicates the system applicability to the real-worlds ENVISAT scenario.
- **Use ENVISAT testbed data:** Future work should consider testing the pose estimation pipeline trained on photorealistic images on testbed data. This will allow to progress further in validating the developed methodology for on-orbit inference.
- **Generate a large-scale, photorealistic dataset for spacecraft pose estimation:** The novelty behind the ENVISAT dataset is the generation of photorealistic synthetic images. This thesis proved how this approach improves the pose estimation performance while ensuring system scalability, however, a large-scale dataset generated using the same principle is still missing. Future work should fill this gap by generating a larger dataset containing both synthetic and testbed images, accelerating the development of the first on-orbit demonstration for a CNN-based relative navigation system.
- **Accuracy verification on Myriad X:** It remains an open question why a high-performance degradation was experienced when testing the models on Myriad X. A possible explanation is the model conversion to half-precision representation, which could be solved by performing quantisation-aware training.
- **Use a higher number of keypoints:** The 3D Tango and ENVISAT models contain 11 and 16 keypoints, respectively. Despite being enough to properly define the PnP problem, using a larger set of keypoints is expected to make the pose estimation more robust and therefore improve performance.
- **Ensure full consistency across different targets:** The pipeline developed in this thesis was easily reconfigured when moving from one scenario to the other. Leveraging this promising result, future research may focus on a pose estimation pipeline requiring little to no reconfiguration effort. This may be achieved by using a sufficiently large number of keypoints or adopting a target-agnostic approach.

References

Aglietti, G. S., Taylor, B., Fellowes, S., Salmon, T., Retat, I., Hall, A., Chabot, T., Pisseloup, A., Cox, C., Mafficini, A., et al. (2020). "The active space debris removal mission RemoveDebris. Part 2: In orbit operations". In: *Acta Astronautica* 168, pp. 310–322.

Andriluka, M., Pishchulin, L., Gehler, P., and Schiele, B. (2014). "2d human pose estimation: New benchmark and state of the art analysis". In: *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pp. 3686–3693.

Barad, K. (2020). "Robust navigation framework for proximity operations around uncooperative Spacecraft: A monocular vision-based navigation approach using deep learning". MA thesis. Delft University of Technology.

Becker, R. (2017). *Dealing with the quaternion antipodal problem for advecting fields*. Tech. rep. US Army Research Laboratory US Army Research Laboratory United States.

Bessho, K., Date, K., Hayashi, M., Ikeda, A., Imai, T., Inoue, H., Kumagai, Y., Miyakawa, T., Murata, H., Ohno, T., et al. (2016). "An introduction to Himawari-8/9—Japan's new-generation geostationary meteorological satellites". In: *Journal of the Meteorological Society of Japan. Ser. II* 94.2, pp. 151–183.

Biesbroek, R., Aziz, S., Wolahan, A., Cipolla, S.-f., Richard-Noca, M., and Piguet, L. (2021). "The clearspace-1 mission: Esa and clearspace team up to remove debris". In: *Proc. 8th Eur. Conf. Sp. Debris*, pp. 1–3.

Blackerby, C., Okamoto, A., Fujimoto, K., Okada, N., Forshaw, J. L., and Auburn, J. (2018). "ELSA-d: An In-Orbit End-of-Life Demonstration Mission". In: *Proceedings of the 69th International Astronautical Congress, Bremen, Germany: International Astronautical Federation*.

Bonnal, C., Ruault, J.-M., and Desjean, M.-C. (2013). "Active debris removal: Recent progress and current trends". In: *Acta Astronautica* 85, pp. 51–60.

Brochard, R., Lebreton, J., Robin, C., Kanani, K., Jonniaux, G., Masson, A., Despré, N., and Berjaoui, A. (2018). "Scientific image rendering for space scenes with the SurRender software". In: *arXiv preprint arXiv:1810.01423*.

Cao, Y., Xu, J., Lin, S., Wei, F., and Hu, H. (2019). "Gcnet: Non-local networks meet squeeze-excitation networks and beyond". In: *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pp. 0–0.

Cassinis, L. P. (2022). "Monocular vision-based pose estimation of uncooperative spacecraft". PhD thesis. Delft University of Technology.

Cassinis, L. P., Fonod, R., and Gill, E. (2019). "Review of the robustness and applicability of monocular pose estimation systems for relative navigation with an uncooperative spacecraft". In: *Progress in Aerospace Sciences* 110, p. 100548.

Cassinis, L. P., Fonod, R., Gill, E., Ahrns, I., and Gil Fernandez, J. (2020). "CNN-based pose estimation system for close-proximity operations around uncooperative spacecraft". In: *AIAA Scitech 2020 Forum*, p. 1457.

Chaumette, F. and Hutchinson, S. (2006). "Visual servo control. I. Basic approaches". In: *IEEE Robotics & Automation Magazine* 13.4, pp. 82–90.

Chen, B., Cao, J., Parra, A., and Chin, T.-J. (2019). "Satellite pose estimation with deep landmark regression and nonlinear pose refinement". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*.

Chen, Y., Wang, Z., Peng, Y., Zhang, Z., Yu, G., and Sun, J. (2018). "Cascaded pyramid network for multi-person pose estimation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7103–7112.

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

D'Amico, S., Bodin, P., Delpech, M., and Noteborn, R. (2013). "Prisma". In: *Distributed Space Missions for Earth System Monitoring*. Springer, pp. 599–637.

Deloo, J. (2015). "Analysis of the rendezvous phase of e.Deorbit: guidance, communication and illumination". MA thesis. Delft University of Technology.

DeMenthon, D. F. and Davis, L. S. (1995). "Model-based object pose in 25 lines of code". In: *International journal of computer vision* 15, pp. 123–141.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Ding, X., Wang, Y., Wang, Y., and Xu, K. (2021). "A review of structures, verification, and calibration technologies of space robotic systems for on-orbit servicing". In: *Science China Technological Sciences* 64.3, pp. 462–480.

ESA (2022). *Envisat Overview*. URL: `https://earth.esa.int/eogateway/missions/envisat/description` (visited on 12/09/2022).

Forshaw, J. L., Aglietti, G. S., Navarathinam, N., Kadhem, H., Salmon, T., Pisseloup, A., Joffre, E., Chabot, T., Retat, I., Axthelm, R., et al. (2016). "RemoveDEBRIS: An in-orbit active debris removal demonstration mission". In: *Acta Astronautica* 127, pp. 448–463.

Furano, G., Meoni, G., Dunne, A., Moloney, D., Ferlet-Cavrois, V., Tavoularis, A., Byrne, J., Buckley, L., Psarakis, M., Voss, K.-O., et al. (2020). "Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities". In: *IEEE Aerospace and Electronic Systems Magazine* 35.12, pp. 44–56.

Gao, X.-S., Hou, X.-R., Tang, J., and Cheng, H.-F. (2003). "Complete solution classification for the perspective-three-point problem". In: *IEEE transactions on pattern analysis and machine intelligence* 25.8, pp. 930–943.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. (2018). "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness". In: *arXiv preprint arXiv:1811.12231*.

Gerth, I. (2014). "Convex optimization for constrained and unified landing guidance". PhD thesis. Delft University of Technology.

Gill, E., D'Amico, S., and Montenbruck, O. (2007). "Autonomous formation flying for the PRISMA mission". In: *Journal of Spacecraft and Rockets* 44.3, pp. 671–681.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. (2018). "Recent advances in convolutional neural networks". In: *Pattern recognition* 77, pp. 354–377.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hendrycks, D. and Dietterich, T. (2019). "Benchmarking neural network robustness to common corruptions and perturbations". In: *arXiv preprint arXiv:1903.12261*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861*.

Howard, R. T., Heaton, A. F., Pinson, R. M., and Carrington, C. K. (2008). "Orbital express advanced video guidance sensor". In: *2008 IEEE Aerospace Conference*. IEEE, pp. 1–10.

Hu, J., Shen, L., and Sun, G. (2018). "Squeeze-and-excitation networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. (2019). "Gpipe: Efficient training of giant neural networks using pipeline parallelism". In: *Advances in neural information processing systems* 32.

Izzo, D., Märtens, M., and Pan, B. (2019). "A survey on artificial intelligence trends in spacecraft guidance dynamics and control". In: *Astrodynamics* 3, pp. 287–299.

Jackson, P. T., Abarghouei, A. A., Bonner, S., Breckon, T. P., and Obara, B. (2019). "Style augmentation: data augmentation via style randomization." In: *CVPR workshops*. Vol. 6, pp. 10–11.

Kanani, K., Petit, A., Marchand, E., Chabot, T., and Gerber, B. (2012). "Vision based navigation for debris removal missions". In: *63rd International Astronautical Congress*.

Kendall, A., Grimes, M., and Cipolla, R. (2015). "Posenet: A convolutional network for real-time 6-dof camera relocalization". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2938–2946.

Kessler, D. J. and Cour-Palais, B. G. (1978). "Collision frequency of artificial satellites: The creation of a debris belt". In: *Journal of Geophysical Research: Space Physics* 83.A6, pp. 2637–2646.

Kisantal, M., Sharma, S., Park, T. H., Izzo, D., Märtens, M., and D'Amico, S. (2020). "Satellite pose estimation challenge: Dataset, competition design, and results". In: *IEEE Transactions on Aerospace and Electronic Systems* 56.5, pp. 4083–4098.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6, pp. 84–90.

LeCun, Y., Bengio, Y., et al. (1995). "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10, p. 1995.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.

Lepetit, V., Moreno-Noguer, F., and Fua, P. (2009). "EPnP: An accurate O(n) solution to the PnP problem". In: *International journal of computer vision* 81, pp. 155–166.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017). "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer, pp. 740–755.

Liou, J.-C. (2011). "An active debris removal parametric study for LEO environment remediation". In: *Advances in space research* 47.11, pp. 1865–1876.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer, pp. 21–37.

Madsen, K., Nielsen, H. B., and Tingleff, O. (2004). "Methods for non-linear least squares problems". In.

Magalhaes Oliveira, A. B., Krüger, H., and Theil, S. (2019). "Feature-based optical navigation for Lunar landings". In.

Marchand, E., Uchiyama, H., and Spindler, F. (2015). "Pose estimation for augmented reality: a hands-on survey". In: *IEEE transactions on visualization and computer graphics* 22.12, pp. 2633–2651.

Minaidis, P. (2022). "Embedded development of aI-based computer vision: Acceleration on intel myriad X VPU". In.

Mooij, E. (2019). *Re-entry Systems Lecture Notes*. Aerospace Faculty, Delft University of Technology.

N2YO (2022). *PRISMA (TANGO) Satellite details*. URL: `https://www.n2yo.com/satellite/?s=36827#results` (visited on 12/09/2022).

Newell, A., Yang, K., and Deng, J. (2016). "Stacked hourglass networks for human pose estimation". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*. Springer, pp. 483–499.

O'Shea, K. and Nash, R. (2015). "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458*.

Oberkampf, D., DeMenthon, D. F., and Davis, L. S. (1996). "Iterative pose estimation using coplanar feature points". In: *Computer Vision and Image Understanding* 63.3, pp. 495–511.

Park, T. H., Bosse, J., and D'Amico, S. (2021). "Robotic testbed for rendezvous and optical navigation: Multi-source calibration and machine learning use cases". In: *arXiv preprint arXiv:2108.05529*.

Park, T. H. and D'Amico, S. (2022a). "Adaptive neural network-based unscented Kalman filter for spacecraft pose tracking at rendezvous". In: *arXiv preprint arXiv:2206.03796*.

Park, T. H. and D'Amico, S. (2022b). "Robust multi-task learning and online refinement for spacecraft pose estimation across domain gap". In: *arXiv preprint arXiv:2203.04275*.

Park, T. H., Märtens, M., Jawaid, M., Wang, Z., Chen, B., Chin, T.-J., Izzo, D., and D'Amico, S. (2023). "Satellite pose estimation competition 2021: Results and analyses". In: *Acta Astronautica*.

Park, T. H., Märtens, M., Lecuyer, G., Izzo, D., and D'Amico, S. (2022c). "SPEED+: Next-generation dataset for spacecraft pose estimation across domain gap". In: *2022 IEEE Aerospace Conference (AERO)*. IEEE, pp. 1–15.

Park, T. H., Sharma, S., and D'Amico, S. (2019). "Towards robust learning-based pose estimation of noncooperative spacecraft". In: *arXiv preprint arXiv:1909.00392*.

Pedersen, L., Kortenkamp, D., Wettergreen, D., Nourbakhsh, I., and Smith, T. (2002). "NASA EXploration Team (NEXT) space robotics technology assessment report". In: *Moffett Field, CA: NASA*.

Petit, A., Marchand, E., and Kanani, K. (2012). "Vision-based detection and tracking for space navigation in a rendezvous context". In: *Int. Symp. on Artificial Intelligence, Robotics and Automation in Space, i-SAIRAS*.

Petrongonas, E., Leon, V., Lentaris, G., and Soudris, D. (2021). "ParalOS: A Scheduling & Memory Management Framework for Heterogeneous VPUs". In: *2021 24th Euromicro Conference on Digital System Design (DSD)*. IEEE, pp. 221–228.

Pishchulin, L., Andriluka, M., Gehler, P., and Schiele, B. (2013). "Strong appearance and expressive spatial models for human pose estimation". In: *Proceedings of the IEEE international conference on Computer Vision*, pp. 3487–3494.

Proença, P. F. and Gao, Y. (2020). "Deep learning for spacecraft pose estimation from photorealistic rendering". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6007–6013.

Recommendation ITU-R BT.601-5 (2011). "Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios". In: *Int. Radio Consultative Committee Int. Telecommun. Union, Switzerland, CCIR Rep*, pp. 624–4.

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

Reed, B. B., Smith, R. C., Naasz, B. J., Pellegrino, J. F., and Bacon, C. E. (2016). "The restore-L servicing mission". In: *AIAA space 2016*, p. 5478.

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

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3, pp. 211–252.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520.

Sauceda, F. (2001). "Space station reference coordinate systems". In: *International Space Station, Rept. TR-SSP-30219*.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013). "Overfeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint arXiv:1312.6229*.

Sharma, S., Beierle, C., and D'Amico, S. (2018a). "Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks". In: *2018 IEEE Aerospace Conference*. IEEE, pp. 1–12.

Sharma, S. and D'Amico, S. (2019). "Pose estimation for non-cooperative rendezvous using neural networks". In: *arXiv preprint arXiv:1906.09868*.

Sharma, S., Ventura, J., and D'Amico, S. (2018b). "Robust model-based monocular pose initialization for noncooperative spacecraft rendezvous". In: *Journal of Spacecraft and Rockets* 55.6, pp. 1414–1429.

Shepperd, S. W. (1978). "Quaternion from rotation matrix". In: *Journal of guidance and control* 1.3, pp. 223–224.

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

Smith, L. N. and Topin, N. (2019). "Super-convergence: Very fast training of neural networks using large learning rates". In: *Artificial intelligence and machine learning for multi-domain operations applications*. Vol. 11006. SPIE, pp. 369–386.

Stephenson, A. G. (1988). "Space station-the orbital maneuvering vehicle". In: *Aerospace America* 26, pp. 24–26.

Stolk, T. (2022). "Visual-lidar feature detection for relative pose estimation of an unknown spacecraft". MA thesis. Delft University of Technology.

Sturm, P. (2014). "Pinhole camera model". In: *Computer Vision: A Reference Guide*, pp. 610–613.

Su, H., Qi, C. R., Li, Y., and Guibas, L. J. (2015). "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2686–2694.

Sun, K., Xiao, B., Liu, D., and Wang, J. (2019). "Deep high-resolution representation learning for human pose estimation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5693–5703.

Sun, X., Xiao, B., Wei, F., Liang, S., and Wei, Y. (2018). "Integral human pose regression". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 529–545.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Tan, M. and Le, Q. (2019). "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR, pp. 6105–6114.

Tatsch, A., Fitz-Coy, N., and Gladun, S. (2006). "On-orbit servicing: A brief survey". In: *Proceedings of the IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR'06)*, pp. 276–281.

Telaar, J., Ahrns, I., Estable, S., Rackl, W., De Stefano, M., Lampariello, R., Santos, N., Serra, P., Canetri, M., Ankersen, F., et al. (2017). "GNC architecture for the e. Deorbit mission". In: *7th European Conference for Aeronautics and Space Sciences (EUCASS)*. ESA Publications Division, ESTEC Noordwijk, The Netherlands, pp. 1–15.

Terzakis, G. and Lourakis, M. (2020). "A consistently fast and globally optimal solution to the perspective-n-point problem". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer, pp. 478–494.

Toshev, A. and Szegedy, C. (2014). "Deeppose: Human pose estimation via deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1653–1660.

Van der Heijden, L. (2022). "Autonomous navigation around asteroids using convolutional neural networks". In.

Vittaldev, V., Mooij, E., and Naeije, M. (2012). "Unified State Model theory and application in Astrodynamics". In: *Celestial Mechanics and Dynamical Astronomy* 112.3, pp. 253–282.

Wang, X., Girshick, R., Gupta, A., and He, K. (2018). "Non-local neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7794–7803.

Whelan, D. A., Adler, E. A., Wilson III, S. B., and Roesler Jr, G. M. (2000). "DARPA Orbital Express program: Effecting a revolution in space-based systems". In: *Small Payloads in Space*. Vol. 4136. SPIE, pp. 48–56.

Wie, B. (1998). *Space vehicle dynamics and control*. Aiaa.

Wieser, M., Richard, H., Hausmann, G., Meyer, J.-C., Jaekel, S., Lavagna, M., and Biesbroek, R. (2015). "E. Deorbit mission: OHB debris removal concepts". In.

Xiao, B., Wu, H., and Wei, Y. (2018). "Simple baselines for human pose estimation and tracking". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 466–481.

Yang, Y. and Ramanan, D. (2012). "Articulated human detection with flexible mixtures of parts". In: *IEEE transactions on pattern analysis and machine intelligence* 35.12, pp. 2878–2890.

Zeiler, M. D. and Fergus, R. (2014). "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer, pp. 818–833.

Zhang, Z., Tang, J., and Wu, G. (2019). "Simple and lightweight human pose estimation". In: *arXiv preprint arXiv:1911.10346*.

Appendices

A

Unit Tests

OD is the only step in the pipeline where all functionalities are already implemented. This is the case thanks to MMDetection, which allows to make use of a preimplemented training and testing pipeline.

For the other steps of the pipeline, however, several functionalities had to be adapted from other projects or implemented from scratch. This made it necessary to perform a list of unit tests, which are reported below.

The functionality type refer to how the functionality itself has been implemented, *i.e.*, either as a function or a variable. In the first case, the function can be imported in every script it is needed and used, while in the second case the functionalities defines a variable that can be used in a specific script. Most frame transformations have been implemented this way because of the high number of possible frame transformations and the fact that one frame transformation may be obtained with a different sequence of operations depending on the data available.

- `eul2dcm:`
  - **Type:** Function.
  - **Description:** Convert the set of Euler angles $(\phi, \theta, \psi)^T$ to quaternions.
  - **Method:** Verify input-output pairs for basic cases.
- `quat2dcm:`
  - **Type:** Function.
  - **Description:** Convert quaternions to DCM.
  - **Method:** Verify input-output pairs for basic cases.
- `dcm2quat:`
  - **Type:** Function.
  - **Description:** Convert DCM to quaternions.
  - **Method:** Verify input-output pairs for basic cases.
- `project_kpts:`
  - **Type:** Function.
  - **Description:** Project keypoints from a 3D model to an image.
  - **Method:** Visually inspect that the projected keypoints correspond to the corresponding features.
- `get_bbox:`
  - **Type:** Function.
  - **Description:** Get tight bounding box coordinates in the format $(x_{min}, y_{min}, x_{max}, y_{max})$.
  - **Method:** Visually inspect that the bounding box is correctly applied to the image.
- `relax_bbox:`

  - **Type:** Function.
  - **Description:** Relax bounding box coordinates.
  - **Method:** Visually inspect that the bounding bx is correctly relaxed.

- `prepare_bbox:`
  - **Type:** Function.
  - **Description:** Make the bounding box fully contained within the image and transforms the coordinates to integers.
  - **Method:** Input-output check.

- `prepare_kpts:`
  - **Type:** Function.
  - **Description:** Transforms the keypoint coordinates with respect to the bounding box starting point.
  - **Method:** Input-output check.

- `get_affine_transform:`
  - **Type:** Function.
  - **Description:** Calculate keypoint transformation function from the original image frame to the cropped and resized image frame.
  - **Method:** Verify input-output pairs for basic cases.

- `transform_kpts:`
  - **Type:** Function.
  - **Description:** Apply the keypoint affine transform.
  - **Method:** Input-output check.

- `get_soft_argmax:`
  - **Type:** Function.
  - **Description:** Calculate soft-argmax from a heatmap.
  - **Method:** Verify that the soft-argmax coordinates correspond to the ground-truth.

- `generate_vis_labels:`
  - **Type:** Function.
  - **Description:** Generate visibility labels for each keypoint.
  - **Method:** Input-output check.

- `calc_dists:`
  - **Type:** Function.
  - **Description:** Calculate distance between the predicted and ground-truth keypoint coordinates.
  - **Method:** Input-output check.

- `dist_acc:`
  - **Type:** Function.
  - **Description:** Calculate the accuracy on the predicted keypoint coordinates.
  - **Method:** Input-output check.

- `pose_score:`
  - **Type:** Function.
  - **Description:** Calculate pose score.
  - **Method:** Verify that the computed pose score is reasonable for a set of inputs and that, when the ground truth pose is passed as input, the output is zero.

- `dcm_error:`

  - **Type:** Variable.
  - **Description:** Calculate the pointing error DCM $\mathbf{C}_{B'/B}$ (3.40).
  - **Method:** Input-output check for basic cases.

- `dcm_B2C:`

  - **Type:** Variable.
  - **Description:** DCM describing the frame transformation from the on-board camera to the Blender camera frame ($\mathbf{C}_{C/B}$).
  - **Method:** Since the orientation between the two frames is constant, it is enough to verify that the output DCM coincides with what is expected.

- `dcm_W2C:`

  - **Type:** Variable.
  - **Description:** DCM describing the frame transformation from the Blender world to the on-board camera reference frame ($\mathbf{C}_{C/W}$).
  - **Method:** Input-output check for basic cases.

- `dcm_C2T:`

  - **Type:** Variable.
  - **Description:** DCM describing the frame transformation from the on-board camera to the target reference frame ($\mathbf{C}_{T/C}$).
  - **Method:** Input-output check for basic cases.

B

MMDetection Configuration

A complete MMDetection training configuration file is created by defining the model, dataset, and runtime settings. A model configuration file contains all the parameters required to create the model. For the EfficientNetB3-SSD512, the configuration file looks like the following:

```
# ------------------------------------------------------------------------ #
# Model settings
# ------------------------------------------------------------------------ #
checkpoint = 'https://download.openmmlab.com/mmdetection/v2.0/efficientnet/
    retinanet_effb3_fpn_crop896_8x4_1x_coco/
    retinanet_effb3_fpn_crop896_8x4_1x_coco_20220322_234806-615a0dda.pth'
input_size = 512

# Define data preprocessing
data_preprocessor = dict(
    type='DetDataPreprocessor',
    mean=[123.675, 116.28, 103.53],
    std=[58.395, 57.12, 57.375],
    bgr_to_rgb=False,
    pad_size_divisor=1
)

# Model settings
model = dict(
    type='SingleStageDetector',
    data_preprocessor=data_preprocessor,
    backbone=dict(
        type='EfficientNet',
        arch='b3',
        drop_path_rate=0.2,
        out_indices=(4, 5),
        frozen_stages=0,
        norm_cfg=dict(type='SyncBN', requires_grad=True, eps=1e-3, momentum=0.01),
        norm_eval=False,
        init_cfg=dict(type='Pretrained', prefix='backbone', checkpoint=checkpoint)),
    neck=dict(
        type='SSDNeck',
        in_channels=(136, 384),
        out_channels=(136, 384, 512, 256, 256, 256, 256),
        level_strides=(2, 2, 2, 2, 1),
        level_paddings=(1, 1, 1, 1, 1),
        l2_norm_scale=None,
        use_depthwise=True,
        norm_cfg=dict(type='BN', eps=0.001, momentum=0.03),
        act_cfg=dict(type='ReLU6'),
        init_cfg=dict(type='TruncNormal', layer='Conv2d', std=0.03)),
    bbox_head=dict(
        type='SSDHead',
        in_channels=(136, 384, 512, 256, 256, 256, 256),
        num_classes=1,
        use_depthwise=True,
```

```
        norm_cfg=dict(type='BN', eps=0.001, momentum=0.03),
        act_cfg=dict(type='ReLU6'),
        init_cfg=dict(type='Normal', layer='Conv2d', std=0.001),
        anchor_generator=dict(
            type='SSDAnchorGenerator',
            scale_major=False,
            input_size=input_size,
            basesize_ratio_range=(0.1, 0.9),
            strides=[8, 16, 32, 64, 128, 256, 512],
            ratios=[[2], [2, 3], [2, 3], [2, 3], [2, 3], [2], [2]]),
        bbox_coder=dict(
            type='DeltaXYWHBBoxCoder',
            target_means=[.0, .0, .0, .0],
            target_stds=[0.1, 0.1, 0.2, 0.2])),
    # Model training and testing settings
    train_cfg=dict(
        assigner=dict(
            type='MaxIoUAssigner',
            pos_iou_thr=0.5,
            neg_iou_thr=0.5,
            min_pos_iou=0.,
            ignore_iof_thr=-1,
            gt_max_assign_all=False),
        sampler=dict(type='PseudoSampler'),
        smoothl1_beta=0.1,
        allowed_border=-1,
        pos_weight=-1,
        neg_pos_ratio=3,
        debug=False),
    test_cfg=dict(
        nms_pre=100,
        nms=dict(type='nms', iou_threshold=0.45),
        min_bbox_size=0,
        score_thr=0.02,
        max_per_img=1)
)
```

The configuration includes several fields required to define the model architecture, such as the `type` argument, indicating it is a single-stage detector, the `data_preprocessor` argument, specifying the input preprocessing steps, and the different part of the model, namely the `backbone`, `neck`, and `bbox_head`. MMDetection also allows to specify different settings related to the model training and testing settings, such as the `smooth_l1_beta` parameter to set the coefficient β from Equation (7.3) for training and the `max_per_img` field specifying the number of detections to be kept after NMS at the inference stage. Concerning the dataset settings, when training on SPEED+ the file contains the following:

```
data_root = '../datasets/speedplusv2'
dataset_type = 'SpeedPlusv2'
image_size = (640, 640)

# Baseline configuration
aug_cfg=dict(
    p=0.5,
    brightness_and_contrast=True,
    blur=True,
    noise=True,
    erasing=False,
    haze=False,
    stars=False,
    streaks=False,
    sun_flare=False
)

# Training
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='Augmentations', cfg=aug_cfg, stage='train'),
    dict(type='Resize', scale=image_size, keep_ratio=False),
    dict(type='PackDetInputs', meta_keys=('img_path', 'img_id', 'img_shape', 'filename',
                                          'width', 'height', 'instances', 'scale_factor', '
```

```
                                            ori_shape'))
]
train_dataloader = dict(
    dataset=dict(
        type=dataset_type,
        data_root=data_root + '/synthetic/images',
        ann_file=data_root + '/synthetic/train.json',
        pipeline=train_pipeline,
        filter_cfg=dict(filter_empty_gt=True)
    ),
    batch_size=16,
    shuffle=True,
    num_workers=16,
    pin_memory=True,
    drop_last=True,
    persistent_workers=False,
    sampler=None,
    batch_sampler=None
)

# Validation
valid_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='Augmentations', cfg=aug_cfg, stage='validation'),
    dict(type='Resize', scale=image_size, keep_ratio=False),
    dict(type='PackDetInputs', meta_keys=('img_path', 'img_id', 'img_shape', 'filename',
                                          'width', 'height', 'instances', 'scale_factor', '
                                              ori_shape'))
]
val_dataloader = dict(
    dataset=dict(
        type=dataset_type,
        data_root=data_root + '/synthetic/images',
        ann_file=data_root + '/synthetic/validation.json',
        pipeline=valid_pipeline,
        filter_cfg=dict(filter_empty_gt=True)
    ),
    batch_size=1,
    shuffle=False,
    num_workers=16,
    pin_memory=True,
    drop_last=True,
    persistent_workers = False,
    sampler=None,
    batch_sampler=None
)
val_evaluator = dict(
    type='CocoMetric',
    metric='bbox',
    format_only=False,
    backend_args=None
)
```

The dataset configuration sets the training and validation data processing pipeline, including data loading, augmentations, and packing. Note that the `Augmentations` processing is used to apply the synthetic augmentations introduced in Section 9.2.1, whose configuration is specified through the `aug_cfg` dictionary. They are not natively implemented in MMDetection. This file is also used to specify the data loading strategy through the `train_dataloader` and `val_dataloader` parameters. For example, in the configuration provided the training batch size is set to 16 and is reduced to 1 for validation. Lastly, the runtime configuration file includes the complementary training settings, such as the number of epochs, optimiser, and learning rate policy, as well as visualisation settings:

```
# ------------------------------------------------------------------------ #
# General settings
# ------------------------------------------------------------------------ #
log_level = 'INFO'
env_cfg = dict(
    cudnn_benchmark=True,
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0),
```

```
    dist_cfg=dict(backend='nccl'),
)

load_from = None
resume = False

default_scope = 'mmdet'

default_hooks = dict(
    timer=dict(type='IterTimerHook'),
    logger=dict(type='LoggerHook', interval=50),
    param_scheduler=dict(type='ParamSchedulerHook'),
    checkpoint=dict(type='CheckpointHook', interval=1),
    sampler_seed=dict(type='DistSamplerSeedHook'),
    visualization=dict(type='DetVisualizationHook'))

# ---------------------------------------------------------------------------- #
# Runtime settings
# ---------------------------------------------------------------------------- #
total_epochs = 300
dist_params = dict(backend='nccl')

train_cfg = dict(type='EpochBasedTrainLoop', max_epochs=total_epochs, val_interval=1)
val_cfg = dict(type='ValLoop')

# ---------------------------------------------------------------------------- #
# Optimizer settings
# ---------------------------------------------------------------------------- #
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer=dict(type='SGD', lr=1.0e-3, momentum=0.9, weight_decay=4.0e-5)
)

# ---------------------------------------------------------------------------- #
# Set learning rate policy
# ---------------------------------------------------------------------------- #
param_scheduler = [
    dict(
        type='MultiStepLR',
        begin=0,
        end=total_epochs,
        by_epoch=True,
        milestones=[100, 175, 250, 280, 295],
        gamma=0.8
    )
]
default_hooks = dict(
    logger=dict(
        type='LoggerHook',
        interval=1000,
    ),
    checkpoint=dict(
        type='CheckpointHook',
        interval=10,
        save_best='auto'
    )
)

# ---------------------------------------------------------------------------- #
# Visualization settings
# ---------------------------------------------------------------------------- #
log_processor = dict(type='LogProcessor', window_size=50, by_epoch=True)

vis_backends = [
    dict(type='LocalVisBackend'),
    dict(type='TensorboardVisBackend'),
]
visualizer = dict(
    type='DetLocalVisualizer',
    vis_backends=vis_backends,
    name='visualizer')
```

As the code above is used to train several models, it was deemed more convenient to create a default model, dataset, and runtime configuration file. The files can then be imported in another configuration file through the `_base_` argument:

```
_base_ = ['../_base_/models/efficientnetb3-ssd512.py',
      '../_base_/datasets/speedplusv2.py',
      '../_base_/default_runtime.py']
```

and modified, for example by specifying a different augmentation configuration. In the example above `../_base_/models/efficientnetb3-ssd512.py`, `../_base_/datasets/speedplusv2.py`, and `../_base_/default_runtime.py` indicate the relative path to the mode, dataset, and runtime configuration files, respectively.

C

Myriad Accuracy Verification

The OD and KD models did not maintain the level of performance observed on GPU when tested on VPU. Tables C.1 and C.2 show the OD and KD accuracy, respectively, on the first 500 images of the SPEED+ validation set. It can be seen from the table how the IoU experiences a slight decrease when moving to a VPU, while the RMSE increases significantly for both HRNet-w18 and HRNet-w32. As already mentioned in Section 11.4, this behaviour is attributed to the conversion from FP32 to FP16, not accounted for during training.

Table C.1: IoU comparison between VPU and GPU on the first 500 images of the SPEED+ validation set.

Model	`synthetic`			
	mean [-]		median [-]	
	VPU	GPU	VPU	GPU
EficientNetB3-SSD512	0.955	0.974	0.961	0.979

Table C.2: RMSE comparison between VPU and GPU on the first 500 images of the SPEED+ validation set.

Model	`synthetic`			
	mean [px]		median [px]	
	VPU	GPU	VPU	GPU
HRNet-w18	60.07	1.61	52.49	0.96
HRNet-w32	19.59	1.27	9.41	0.74

D

Augmentation Parameters

Data augmentation was applied using the Albumentations library:

```
import albumentations as A
```

Every augmentation was applied with a probability of 50%, *i.e.*, `self.cfg.AUGMENTATIONS.P`=0.5. The following settings were used:

- **Blur:** Blur is applied by randomly selecting one out of three possible effects: gaussian, median, and motion blur using the following syntax:

```
    A.OneOf(
        [
            A.MotionBlur(blur_limit=(3, 9)),
            A.MedianBlur(blur_limit=(3, 7)),
            A.GlassBlur(sigma=0.5, max_delta=2)
        ], p=self.cfg.AUGMENTATIONS.P)
```

  where `MotionBlur`, `MedianBlur`, and `GlassBlur` are the Albumentations functions corresponding to the augmentations just mentioned.
- **Noise:**

```
    A.OneOf(
        [
            A.GaussNoise(var_limit=40**2),
            A.ISONoise(color_shift=(0.1, 0.5), intensity=(0.5, 1.0))
        ], p=self.cfg.AUGMENTATIONS.P)
```

  `GaussNoise`, `ISONoise`.
- **Brightness and contrast:**

```
    A.RandomBrightnessContrast(
        brightness_limit=0.2,
        contrast_limit=0.2,
        p=self.cfg.AUGMENTATIONS.P)
```

- **Erasing:**

```
    CoarseDropout(
        max_holes=5,
        min_holes=1,
        max_ratio=0.5,
        min_ratio=0.2, p=self.cfg.AUGMENTATIONS.P)
```

- **Haze:**

```
    RandomHaze(
        mean_min=0.05,
        mean_max=0.15,
        std_min=0.03,
        std_max=0.05,
        p=self.cfg.AUGMENTATIONS.P)
```

- **Stars:**

```
    RandomStreaks(
        mean_min=0.05,
        mean_max=0.15,
        std_min=0.03,
        std_max=0.05,
        p=self.cfg.AUGMENTATIONS.P)
```

- **Streaks:**

```
    RandomStars(
        mean_min=0.05,
        mean_max=0.15,
        std_min=0.03,
        std_max=0.05,
        p=self.cfg.AUGMENTATIONS.P)
```

- **Style augmentation:** α=0.5.
- **Sun flare:**

```
    RandomSunFlare(
        num_flare_circles_lower=1,
        num_flare_circles_upper=10,
        p=self.cfg.AUGMENTATIONS.P)
```