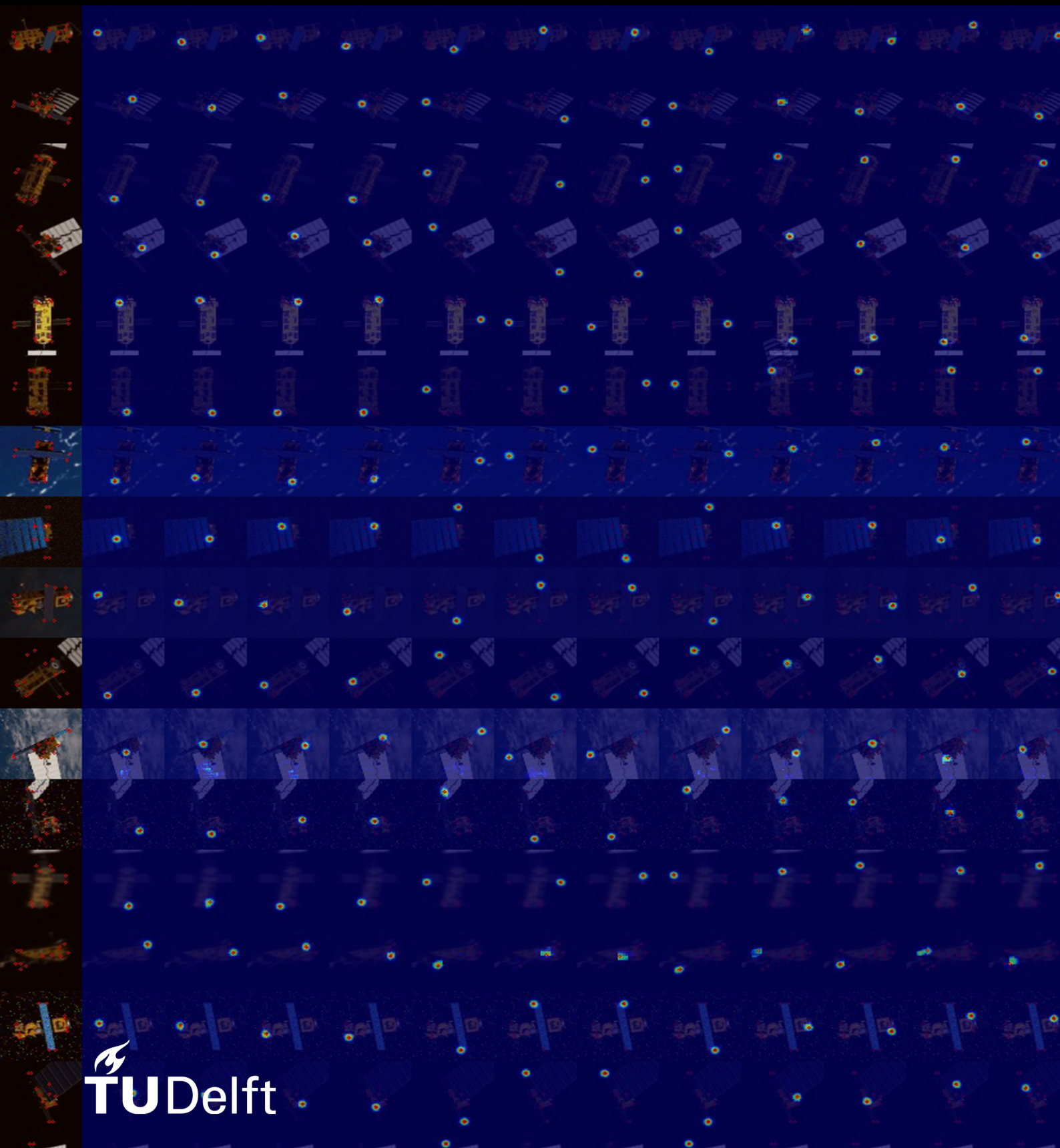
Robust Navigation Framework for Proximity Operations around Uncooperative Spacecraft

Kuldeep Rambhai Barad



TU Delft

Robust Navigation Framework for Proximity Operations around Uncooperative Spacecraft

A monocular vision-based navigation approach using deep learning

by

Kuldeep Rambhai Barad

to obtain the degree of Master of Science
at the Delft University of Technology,

to be defended publicly on Friday September 25, 2020 at 10:00 AM.

Student number:	4788443	
Project duration:	January 25, 2020 – September 24, 2020	
Thesis committee:	Prof. Dr. Eberhard Gill,	TU Delft, Committee chair
	Dr. Alessandra Menicucci,	TU Delft, Supervisor
	Ir. Lorenzo Pasqualetto Cassinis,	TU Delft, Co-supervisor
	Dr. Ir. Bart Root ,	TU Delft, External examiner

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Autonomous vision-based navigation is a crucial element for space applications involving a potentially uncooperative target, such as proximity operations for on-orbit servicing or active debris removal. Due to low mass and power characteristics, monocular vision sensors are an attractive choice for onboard vision-based navigation systems. This work focuses on the problem of utilizing images from a monocular vision sensor for estimation of the target's state relative to the servicer spacecraft. Of special interest is the underlying problem of estimating position and attitude (pose) from a single monocular image, given the knowledge of its 3D model. Motivated by the recent advancements in computer vision and machine learning, this work investigates a learning-based approach that has the potential to enable a new paradigm of robust and accurate onboard navigation systems.

A novel framework is proposed for pose initialization and tracking of an uncooperative spacecraft in close-proximity using monocular images and deep learning. An approach based on the use of Convolutional Neural Networks (CNN) is investigated for its scope in enabling reliable on-orbit operations. With a monocular camera as the sole navigation sensor, the underlying problem of relative pose estimation is tackled with deep learning in CNNs to provide robustness to illumination conditions, as opposed to conventional image processing approaches. The CNNs are trained on synthetic images generated from photorealistic renderings of the target spacecraft and integrated into a navigation loop. The emphasis is put on the robustness of such a CNN-based navigation loop, as CNN models are susceptible to learning implicit data distributions that generalize poorly to reality when trained on synthetic data. The central analysis in this work focuses on the European Space Agency's decommissioned Envisat spacecraft as the target, due to its potential debris generation risk. To that extent, a navigation framework is designed that uses two CNNs- a single-shot object detection network and a high-resolution keypoint detection network, to detect predefined surface keypoints on the target spacecraft. A heatmap representation is used for keypoint detection that provides contextual information per detection and allows indirect quantification of the observation uncertainty. The detected keypoint coordinates and the associated covariances are then used to solve the Perspective-n-Points (PnP) problem using a Maximum Likelihood PnP (MLPnP) solver. The MLPnP solver provides a pose estimate and the associated uncertainty, which is used by a loosely-coupled Multiplicative Extended Kalman Filter to track the state of the target spacecraft. The pose estimation pipeline in the first two stages is benchmarked and validated on the Spacecraft Pose Estimation Dataset (*SPEED*) from the Stanford Rendezvous Laboratory, containing images of the Tango spacecraft from the PRISMA mission. Subsequently, the framework is evaluated for the Envisat target case, using existing Envisat synthetic image datasets. The complete navigation loop is evaluated on a simulated perturbation-free trajectory of the Envisat spacecraft tumbling along the V-bar.

The proposed framework takes a step towards enabling real on-orbit operations by addressing critical challenges in learning-based methods for the navigation problem. The *SPEED* benchmark results for the pose estimation pipeline show comparable performance with the current state-of-the-art approaches, with a desirable balance in speed and accuracy of the CNNs. The CNN models trained on synthetic images and the resulting pose estimation pipeline also demonstrates robustness to previously unseen real images. Subsequent evaluations on the existing Envisat datasets reveal their inadequacy for training and evaluation of CNNs towards the real on-orbit operation. To tackle this, a new augmented image dataset of the Envisat spacecraft is introduced, which improves over the existing datasets by modeling Earth background and common corruptions in the images. The proposed dataset provides objective improvements for training deep neural networks towards robust and reliable on-orbit operations. Finally, a preliminary navigation analysis on a simplified V-bar scenario for Envisat, reveals that the proposed loosely-coupled estimation in the navigation loop provides an accurate navigation solution.

Preface

With a belief that all disciplines must reap the benefits of technology development, this study is aimed at embedding intelligence and vision for the spacecraft of the future. With the onset of machine learning, the transformation in the field of computer vision has been of consequential magnitude in the recent past. Today, deep learning is disrupting people's lives and machines' capabilities faster than ever. At such a time, it is natural to wonder if there is a way for all machines to learn and become better. With this question in mind, we look at spacecraft that have been some of the most complex engineering systems ever built. The potential for use of machine learning in spacecraft is enormous, in which it can help enable systems that meet increasingly challenging operational requirements of the prospective missions. This thesis is a culmination of skills and experience gained over the last two years at TU Delft. The possibility to conduct research that can potentially enable on-orbit servicing and active debris removal missions in the future and contribute to the body of research in the subject is the best way to utilize my skills and abilities at this stage.

Aside from the academic and technical learning experience, the duration of this thesis also involved a great amount of personal learning. During this time, the world faced some of the most harrowing problems in the history of mankind, including a pandemic that shook the very core of our society. Being faced with the dichotomy of being safe and lucky, yet a direct witness to the pain and suffering, greatly transformed my perspective on things. I have been very fortunate to receive the support, access, and the means to get through this period without any major disruption. For a person of modest origins, this is not a personal feat, but one made possible by some of the most incredible people, to whom I owe my gratitude.

First and foremost, I would like to thank Lorenzo, my daily supervisor and mentor, who has invested himself greatly, with patience and humility, in improving the quality and impact of this work. Lorenzo's expertise and experience were critical in directing this work to a successful conclusion. Reinforced by discussions and critical reasoning, his insight into the subject motivated me to make the best of my time and knowledge.

I would like to thank my faculty supervisor, Dr. Alessandra Menicucci, for the unwavering support during this work. I am also grateful to have worked under the supervision of Dr. Robert Fonod, for the guidance and support in the initial stages, when the idea of this thesis was framed. Further, I would like to extend my gratitude to Irene at ESA-ESTEC for providing me access to the GRALS testbed at the Orbital Robotics and Guidance, Navigation, and Control laboratory for the test visits.

I would like to thank Stefan for being a mentor and a friend who has been available to provide helpful insights on life in TU Delft, the industry, and beyond. I also owe a great amount of gratitude to Marco and Louise at Valispace for being the supportive and empathetic leaders who've invested a lot of time and effort in helping me make the best of myself outside academia. Thank you for letting me work on some very interesting projects in the digital systems engineering space and for allowing me to learn on the job, throughout the last year.

I would like to thank Mhatre for the help with editing this report and for the constant support and encouragement through the last two years. A special thanks to Iñigo for the frequent discussions by the canal, on various aspects of this thesis and beyond.

Finally and most importantly, I am grateful to my parents and my brother Arjun, for the constant support that allows me to fearlessly chase my passion.

Kuldeep Rambhai Barad
Delft, September 2020

List of Abbreviations

ADR	Active Debris Removal
AHI	Advanced Himawari Imager
AI	Artificial Intelligence
API	Application Programming Interface
ASTRO	Autonomous Space Transport Robotic Operations
AVGS	Advanced Video Guidance Sensor
CEPPnP	Covariant Efficient Procrustes Perspective-n-Points
CMOS	Complementary Metal–Oxide–Semiconductor
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPN	Cascaded Pyramid Network
CPU	Central Processing Unit
CSP	Cross-Stage Partial
CV	Computer Vision
CW	Clohessy-Wiltshire
DA	Data Augmentation identifier
DARPA	Defense Advanced Research Projects Agency
DART	Demonstration of Autonomous Rendezvous Technologies
DCM	Direction Cosine Matrix
EKF	Extended Kalman Filter
ENV-OD-n	Envisat Object Detection Model identifier
ENV-KP-n	Envisat Keypoint Detection Model identifier
EPnP	Efficient Perspective-n-Points
ESA	European Space Agency
ESTEC	European Space Research and Technology Centre
FCN	Fully Convolutional Network
FCOS	Fully Convolutional One-Stage network
FLOPs	Floating-point Operations
FOV	Field of View
FPGA	Field Programmable Gate Array

FPN	Feature Pyramid Network
GFLOPs	Giga Floating-point Operations
GNC	Guidance Navigation and Control
GPS	Global Positioning System
GPU	Graphical Processing Unit
HPE	Human Pose Estimation
HRNet	High Resolution Network
IoU	Intersection over Union
IP	Image Processing
IR	Infra-red
ISS	International Space Station
JSON	Javascript Object Notation
KD	Keypoint Detection
KRN	Keypoint Regression Network
LEO	Low-Earth Orbit
LHM	Lu-Hager-Mjolsness
LiDAR	Light Detection and Ranging
LVLH	Local Vertical Local Horizontal
MATLAB	Matrix Laboratory
MEKF	Multiplicative Extended Kalman Filter
MEV	Mission Extension Vehicle
MLE	Maximum Likelihood Estimation
MLPnP	Maximum Likelihood Perspective-n-Points
MRP	Modified Rodriguez Parameters
NASA	National Aeronautics and Space Administration
nD	n-dimensional
NMS	Non-maximum suppression
NST	Neural Style Transfer
OD	Object Detection
OE	Orbital Express
OOS	On-Orbit Servicing
OSAM	On-orbit Servicing, Assembly and Manufacturing demonstrator
OSIRIS-REx	Origins, Spectral Interpretation, Resource Identification, Security, Regolith Explorer
PAN	Path Aggregation Network

PANGU	Planet and Asteroid Natural Scene Generation Utility
PnP	Perspective-n-Points
POSIt	Pose from Orthography and Scaling with Iterations
PRISMA	PRecursore IperSpettrale della Missione Applicativa
RANSAC	Random Sample consensus
REG	Rendezvous Entry Gate
ReLU	Rectified Linear Unit
ResNet	Residual Network
RISC	Reduced Instruction Set Computer
R-NAV	Robust-Navigation framework
ROE	Relative Orbital Elements
RoI	Region of Interest
RPO	Rendezvous and Proximity Operations
RTN	Radial, Tangential and Normal
SAR	Synthetic Aperture Radar
SGD	Stochastic Gradient Descent
SLAB	Stanford Space Rendezvous Laboratory
SLAM	Simultaneous Localization and Mapping
SPEED	Spacecraft Pose Estimation Dataset
SPD-KP	SPEED Keypoint Detection Model identifier
SPD-OD	SPEED Object Detection Model identifier
SPN	Spacecraft Pose Network
SPP	Spatial Pyramid Pooling
SSD	(Single Shot Detector
SVD	Singular Value Decomposition
TPU	Tensor Processing Unit
VBN	Vision based Navigation
VGG	Visual Geometry Group
YOLO	You Only Look Once

Contents

List of Figures

List of Tables

I

Introduction and Overview

1 Introduction

Spacecraft Rendezvous and Proximity Operations (RPO) are elementary to the current and future technology infrastructures in space. Since the inception of the space age, RPO has been a critical domain of technology development in enabling ambitious space applications and exploration architectures like the Apollo program and the International Space Station (ISS) [1]. With increasing ease of access to orbit, new venues are opening up that demand challenging RPO capabilities [2]. This work aims to contribute to the future of RPO activities that will enable these capabilities in the newer space systems architectures.

1.1. Motivation

According to the latest annual space environment report [3], over 34000 resident space objects larger than ≈10 cm have been cataloged around Earth. The total population comprises mainly of the debris of various sizes from spacecraft and rocket bodies, with only around 2300 operational spacecraft. Events like the accidental collision of Kosmos-2251 and Iridium-33 and anti-satellite tests have persistently contributed to the debris population and continue to increase the total population. With an ever-increasing number of anthropogenic objects in orbit, the safety of the operational environment is threatened evermore, posing a grave threat to the current and future assets in orbit. Controlling the amount of space debris generation in the future is not adequate by itself, as current large inactive objects in the orbit continue to pose a high risk of fragmentation and cascaded collisions. According to an estimate [4], up to 15 large non-functional objects must be removed per year to maintain long-term stability of the orbital environment in the crowded orbital regimes. Consequently, cost-effective and feasible Active Debris Removal (ADR) solutions are required to improve the future space environment.

As a complement, there is also a dire need to add flexibility to the current and future space assets through On-Orbit Servicing (OOS). The cost of replacing indispensable space assets, due to the predominant expendable spacecraft philosophy, increases exponentially as older assets age. A feasible OOS infrastructure can

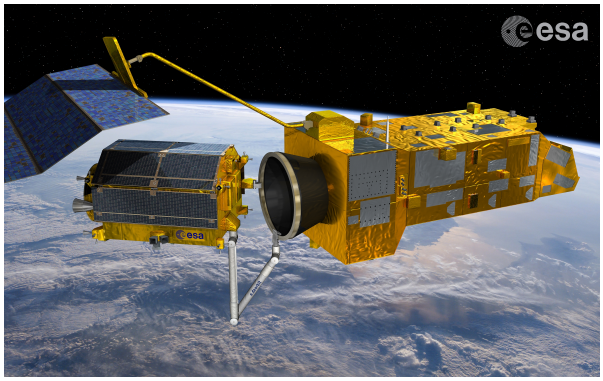


Figure 1.1: e.Deorbit [5] concept for Envisat capture and removal (Credits:ESA)

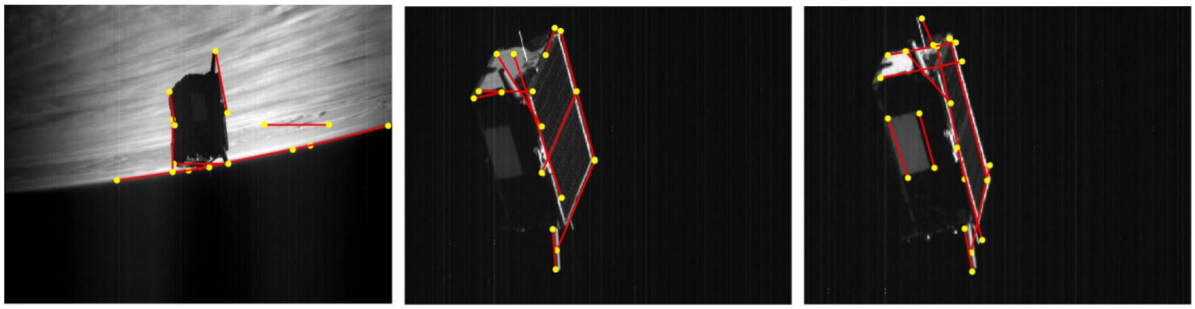


Figure 1.2: Example showing poor detections in images with adverse illumination and background texture using IP [9]

facilitate life extension and risk reduction for space systems improving the efficacy of the systems and systems of systems and transforming existing lifecycle paradigms. The need for such an infrastructure is being realized now more than ever. The feasibility and benefits of on-orbit servicing have been identified in the past [6, 7]. However, there are technical challenges to wide-scale applicability like hardware complexity, flexibility, and autonomy, especially if the target spacecraft is inactive and unable to cooperate during operations. These challenges must still be tackled to make a real impact.

The prime motivation for this work is to support technology development in these mission domains. The focus is put on the challenge of navigating around an uncooperative target body with minimal hardware complexity. By exploring the modern solutions that tackle technical challenges effectively and enable needed capabilities realistically, this work takes a step forward in transforming the way we access and utilize space.

1.2. Background

At the heart of ADR and OOS infrastructures are a challenging set of stringent system requirements for RPO. These requirements motivate the development of modern technology solutions, especially for a potentially uncooperative target. In particular, the development of reliable autonomous Guidance, Navigation, and Control (GNC) system is a key enabling technology. Autonomy is crucial and necessitated by the fact that the uncertainties involved in the state knowledge with ground-based control with time delay make it difficult to fulfill the stringent mission objectives. The systems that tackle these challenges are currently in concept realization and pre-demonstration phases with missions like the combined ADR/OOS demonstration mission-e.deorbit, visualized in Fig. 1.1. For such missions, accurate and robust estimation of the relative state of a potentially tumbling target in close-proximity is essential to allow safe rendezvous and capture.

For this problem, conventionally a range of sensors including Radar, Light Detection and Ranging (LiDAR), monocular cameras, and stereo cameras have been proposed in various configurations for the navigation systems. Inherently, the existing solutions impose significant hardware complexity and cost on the system and the system development. The development of such systems with intensive demands ultimately prohibits the cost-effective realization of ADR and OOS infrastructures that can be widely employed. Since the choice of the on-board sensor(s) is driven by the limited power, mass, and computation resources onboard the spacecraft along with the cost of hardware, monocular cameras have attracted considerable attention in recent works on uncooperative pose estimation [8–10]. However, the use of monocular cameras introduces known issues with high contrast, low signal-to-noise ratio [11]. Further, the monocular camera may not provide measurements during an eclipse or unfavorable illumination geometry. Therefore, the applicability, accuracy, robustness, and reliability of navigation systems based on monocular vision is an active topic for research [12]. This work is oriented towards a novel solution to overcome the associated problems and utilize a monocular vision sensor for autonomous navigation around an uncooperative spacecraft.

The problem of navigation concerns the accurate estimation of the dynamical state of a body with respect to a reference. For the RPO application considered here, it concerns the estimation of translation and rotational state of the target spacecraft relative to the servicer spacecraft. For vision-based navigation, the underlying problem is that of relative pose estimation from the images. The relative pose is defined as the position and attitude (pose) of the target's body axes with respect to the camera axes. Utilizing a monocular camera, relative pose estimation can be accomplished using either a single monocular image or a batch of such images. The conventional approaches for monocular pose estimation in close proximity use traditional

Image Processing (IP) methods that extract features like edges, corners, keypoints, or depth-maps from a single image. The extracted features are used to solve the Perspective-n-Points (PnP) [13] problem to estimate the pose using perspective projection between features in the image and those in the known 3D model. Such IP algorithms are unreliable due to the low solution availability and the lack of robustness to illumination and range [8, 10]. Fig. 1.2 shows one such example of poor detection of edges with the Sharma-Ventura-D'Amico [14] IP algorithm in the presence of adverse illumination and background texture. Further, to solve the PnP problem, computationally expensive routines such as Random Sample Consensus (RANSAC) [13] are required to match the detected features with the known features in the spacecraft model. Further, alternative approaches such as Simultaneous Localization and Mapping (SLAM) [15], that use more than one monocular image at a time, are also subject to favorable relative motion and a long initialization phase [16]. Therefore, the reliability of such systems has been brought into question and newer solutions are sought to reliably tackle the navigation challenges. Recent advances in Computer Vision (CV) for pose estimation in terrestrial applications have relied on the quickly evolving domain of machine learning to enable unprecedented efficacy across several applications. Using hierarchical function approximators called neural networks, machine learning techniques rely on pattern recognition in large amounts of data to create an accurate predictive mapping. For vision tasks, Convolutional Neural Networks (CNN) have shown unprecedented applicability and solved several persistent problems. Loosely inspired by the inner working of the human visual cortex, CNNs have pioneered visual data interpretation. Typically, a CNN is trained on representative visual data to generate a predictive model. This process is called machine learning or deep learning, in the case of modern CNNs.

For spacecraft pose estimation, although vision systems using CNNs overcome some of the challenges faced by IP systems [14, 17], development of reliable CNN models and the supporting vision pipeline are not trivial. The applicability of CNNs in the navigation loop is made challenging by the fact that unlike terrestrial applications, real images of the target spacecraft are usually not available at the scale required for deep learning. The CNNs must therefore be trained on virtually rendered images of the target spacecraft containing synthetic artefacts. The CNNs are then required to learn from the synthetic images and transfer the predictive performance to real images that may contain unmodelled image artefacts. Since CNNs generate a mapping using the synthetic training data, the trained CNN models can result in unpredictable behavior when used on real images that are out-of-distribution with respect to the training data [18]. This is a predominant problem in applied machine learning and poses a major hurdle for the real-life deployment of CNNs in safety-critical applications. Consequently, the robustness of CNNs and the learning-based approaches for spacecraft pose estimation must be investigated and tackled. Further, CNN models and neural networks at large, do not implicitly account for uncertainty when mapping an image to an output quantity. Therefore, taking measurement uncertainty in the pose estimation and ultimately state estimation, is an additional challenge for the robustness of the navigation loop during adverse operational conditions. This work is focused on designing a navigation framework that tackles the aforementioned problems to enable pose and state estimation with desirable accuracy and robustness. The framework encompasses all aspects of design, selection, configuration, and evaluation of a state estimation system that uses a learning-based approach for navigation around an uncooperative target spacecraft.

In this context, a pair of two spacecraft is considered throughout this report. The pair comprises of a target spacecraft which is assumed to be in an uncooperative non-nominal or inoperational state, and a servicer spacecraft that approaches the target in close proximity for a hypothetical capture, removal, or repair. The working range is assumed to be a 'close-proximity' approach, loosely defined as the relative distance of less than 200 meters between the two spacecraft when the target is distinctly large enough that its body features could be detected while maintaining a safe initial distance. Hence, neither the challenges in previous stages of proximity operations, nor the subsequent capture, docking, and removal stage is considered here. Additionally, an uncooperative target is defined as a known resident space object that has no means of active communication or presence of external fiducial markers to assist in relative pose estimation on-board the servicer spacecraft. Moreover, the focus will be on the problem of the navigation loop only, without considering closed-loop guidance or control. For the purpose of this work, a 3D model of the target spacecraft is also assumed to be known. The generalizability of the resulting system to multiple targets must be tackled in the subsequent research.

1.3. Research Overview

The research undertaken for this work is aimed at extending the analysis and understanding of learning-based pose estimation systems, currently available in the literature. A thorough review of the previous works was carried out during the literature study and critical knowledge gaps in the current research body were identified. These are as following:

- Analysis of multiple spacecraft/scenarios
- Investigation of CNN design, configuration, and memory/computation efficiency
- Analysis of robustness in CNNs
- Analysis on complete navigation loop end-to-end
- Use of contextual information from CNNs to quantify and preserve uncertainty

Since the research into learning-based approaches for navigation around an uncooperative target is in the early stages, the extensive analysis was deemed necessary for improving the state of research in learning-based solutions to spacecraft navigation around an uncooperative target.

Research Questions

The objective of this research is to investigate a navigation framework utilizing CNNs and deep learning to provide reliable pose estimation and state tracking for an uncooperative target. Building upon the past research, this work seeks to investigate aspects of a suitable navigation framework that provides accuracy, robustness, and applicability to real-life on-orbit navigation scenarios. Along that direction, the central question posed for this research can broadly be stated as following:

> **How can the performance of the current learning-based approaches for monocular vision-based navigation around an uncooperative spacecraft be improved to enable on-orbit operation?**

This is further broken down into more traceable research questions that were tackled directly in this work.

RQ-1: Which current learning-based pose estimation architecture is most suitable for improvements?

a. Which architecture is the most suitable for the navigation framework?

b. How can the components of the pose estimation architecture be selected, designed, and evaluated?

c. How can the learning-based pose estimation architecture be configured and implemented?

d. How does the resulting pose estimation pipeline compare with the state-of-the-art?

RQ-2: How do the proposed framework and implementation bridge the gap between synthetic training and real on-orbit operations for learning-based pose estimation?

a. How can CNNs be made robust to real images?

b. What aspects of network architecture and training are relevant for embedding robust learning?

c. How can robustness of the CNNs be assessed?

d. Does the performance of the devised framework transfer well to previously unseen images?

RQ-3: How can the given learning based pose estimation pipeline be integrated with a state estimator to enable state tracking?

a. Which state estimator architecture is suitable for the current navigation framework?

b. How can the measurement covariance be quantified in the navigation loop to ensure consistency in filtering?

c. How can the integrated navigation loop be verified for accuracy and robustness?

d. How does the designed navigation loop compare with other design alternatives?

In addition to these primary questions, one more optional question is targeted for the relevance and the added value to the quality of the proposed research:

RQ-4: How can newer datasets improve the performance of a learning-based architecture?

a. How should newer datasets be generated in comparison to the currently available datasets?

b. Which properties of the image dataset are important for improvements?

c. How can the improved dataset aid in making the navigation loop more suitable for real on-orbit operation?

d. Does the newer dataset provide improvement in training and testing of the learning-based pose estimation pipeline?

1.4. Thesis Outline

This report is divided into four major parts. The **Introduction and Overview**, provided the motivation for the work with a brief introduction to the relevant aspects of the navigation problem being addressed. **Chapter 2** lays down the foundation of the navigation problem, with a discussion on the relevant space missions and existing navigation systems. Further, a pre-requisites of monocular vision-based navigation are discussed with a description of pose estimation approaches and alternatives currently available in the literature. Following a critical analysis of identifying the limitations of the current monocular pose estimation systems, the use of machine learning for the pose estimation is motivated.

The **Navigation Framework Design** part of the report encapsulates the selection, design, and implementation of the concerned navigation system and its major components. In principle, this part of the report constitutes all the aspects of the proposed framework that were reasoned, developed, designed, implemented, or analyzed during this work. **Chapter 3** provides an overview of the framework, providing necessary high-level reasoning behind the architecture selection and design. Further, the framework design, analysis, and outcomes are compared with the relevant works in the literature to justify the novelty and contributions made by this work. Subsequent chapters describe the individual components of the navigation framework. **Chapter 4 and 5** present an in-depth discussion on the selection, configuration and implementation of the CNNs used in the pose estimation pipeline. The choice of design aspects in the selected CNNs is presented along with the evaluation metrics relevant for experimentation. Chapter 6 addresses the datasets used to train the CNN along with the aspects of the data pipeline necessary for implementation and experimentation. Two existing synthetic image datasets of the Envisat spacecraft as well as the Tango spacecraft from the PRISMA mission are employed, with a discussion on their features and limitations. Further, motivating the need for better datasets to enhance robustness, the generation of a new augmented dataset is elaborated. **Chapter 7** details the design of a maximum likelihood estimation based solver used for solving the PnP problem during pose estimation. With the necessary description of the underlying theory and the algorithm, the limitations of the methods are identified. Resulting from a detailed investigation of the limitations, a novel modification is identified for the algorithm that drastically improves its pose estimation performance. The performance improvement resulting from the proposed method is validated extensively through benchmarking and comparison with the original algorithm as well as other solvers. Finally, **Chapter 8** presents the design of a state estimator which provides estimates of the target's relative state, completing the navigation loop.

Following the detailed description of the navigation framework components, the whole framework is incrementally evaluated and composed into a complete navigation loop in the **Framework Evaluation** part of the report. In that, **Chapter 9** presents an extensive discussion on experiments to incrementally evaluate, benchmark, and assess the performance of the proposed framework components. For each relevant design decision, the effect on accuracy and robustness is analyzed. The final framework is compared to state-of-the-art approaches and applicability to a representative proximity operations scenario is demonstrated with a navigation analysis.

In **Closure**, **Chapter 10** traces the research questions to the developments and findings of this thesis to provide conclusions and recommendations for the future works.

2

Mission and Vision-based Navigation Overview

The problem of learning-based approach to vision-based navigation, under investigation in this work, involves multi-disciplinary aspects from various subject. This chapter provides a short overview of the prerequisite concepts required to support the subsequent work on the proposed navigation framework.

2.1. Rendezvous and Proximity Operations in Space Missions

Proximity operations have played a crucial role in the development of critical space infrastructure, dating back to the inception of the Space Age. RPO has been a pivotal technology in the advancement of human space exploration and robotic space applications. In this chapter, the evolution of RPO systems in the past space missions has been reviewed, followed by a discussion on vision-based navigation for RPO.

The roots of RPO in space missions can be traced back to the Gemini program in the early 1960s. A major part of the flight test objectives of the Gemini program was to demonstrate and build pre-cursor capability in RPO and docking for the Apollo program [19]. The twin spacecraft mission with Gemini VI and Gemini VII was the first-ever mission to execute rendezvous and proximity operations, followed by Gemini VIII which performed the first docking. The rendezvous and close proximity operations were controlled by the astronauts through visual monitoring, manual control, and decision making with significant ground support. This empowered the proximity operations and docking of the crew and command module in Apollo 9, where RPO was utilized for the first time to meet operational requirements. The maturity of the resulting RPO and docking capabilities played a critical role in the subsequent Apollo missions to the Moon. The RPO capabilities in the Gemini and Apollo programs required a high degree of manual control and cooperation with extensive ground intervention. Around the same time frame (1967), a pair of two uncrewed soviet spacecraft Kosmos-186 and Kosmos-188 executed the first-ever autonomous RPO and docking, using an active inter-spacecraft communication link. Subsequently, RPO capabilities including enabled the next cycle of missions for orbiting laboratories or space stations, from the Soyuz-11 flight to Salyut-1 to the current ISS missions. From a navigation point of view, the development of technology did not originate solely from earth-bound missions. A host of planetary and small-body exploration missions have provided heritage in autonomous vision-based navigation, from AutoNav in Deep Space-1 [20] to the very recent OSIRIS-REx [21].

In terms of OOS, the repair of Skylab-2's jammed solar panel was the first notable milestone, with the repair operations executed entirely by the crew with ground support [22]. The subsequent servicing milestones were achieved with the space shuttles in servicing of Intelsat VI with an upper stage and the on-orbit restoration of the Hubble Space Telescope [6]. The shuttle used a host of on-board GNC and robotic systems, including relative navigation that enabled semi-autonomous rendezvous and docking. However, the close-proximity and docking operations were carried out manually by astronauts using visual aids [23]. The continuous evolution and advancements in RPO created a push towards increasing autonomy in on-orbit operations. This enabled missions for robotic OOS, like DARPA's Orbital Express (OE) mission. The OE mission proved autonomous GNC for RPO from a relative range of 200 km up to docking as well as subsequent

servicing functions like fuel transfer, computer, and battery replacement [24]. The close-proximity navigation system used an Audio-Video Guidance Sensor (AVGS), which used laser-based tracking and a monocular CMOS sensor [25]. The simulated target spacecraft did not have communication link with the servicer but featured retro-reflectors to assist navigation using AVGS. The lessons learned from the partial failure in the OE mission operations [24], and the preceding failure of Demonstration of Autonomous Rendezvous Technology (DART) mission emphasized on the need for robustness. In terms of the latest advancements, the Northrop Grumman Mission Extension Vehicle (MEV-1) concept accomplished the first commercial servicing mission for extending the life of Intelsat 901 in early 2020. The MEV-1 servicer spacecraft for the first time demonstrated servicing on a target that was not designed for docking, extending its life by five years. However, despite the advancements, the trend up until this point does not involve an uncooperative spacecraft. Tackling the problem of working around an uncooperative is essential and necessary to achieve the expected goals for large-scale ADR and OOS for sustainable utilization of space.

In terms of ADR, among the missions that were not canceled, RemoveDEBRIS [26] recently demonstrated key technologies for ADR, including a vision-based navigation system, which utilized a combination of LIDAR, IR, and optical cameras. RemoveDEBRIS successfully demonstration simulated debris capture using a dummy CubeSat. A series of demonstration missions including e.Deorbit [5], Restore-L (now OSAM-1) [27] , DARPA Phoenix [28], ELSA-d (End of life Solution by Astroscale demonstrator) [29] and ESA-commissioned Clearspace-One that aim to tackle this problem, are planned for the future.

In consideration, the state of infrastructure for ADR and OOS is in early development or demonstration stages where spacecraft operate in a careful selected narrow operating environments. As the expected cost-serviceability and debris mitigation trends push the space industry towards the need for ADR and OOS, it is of significant importance to develop robust and reliable autonomous relative navigation systems for uncooperative targets.

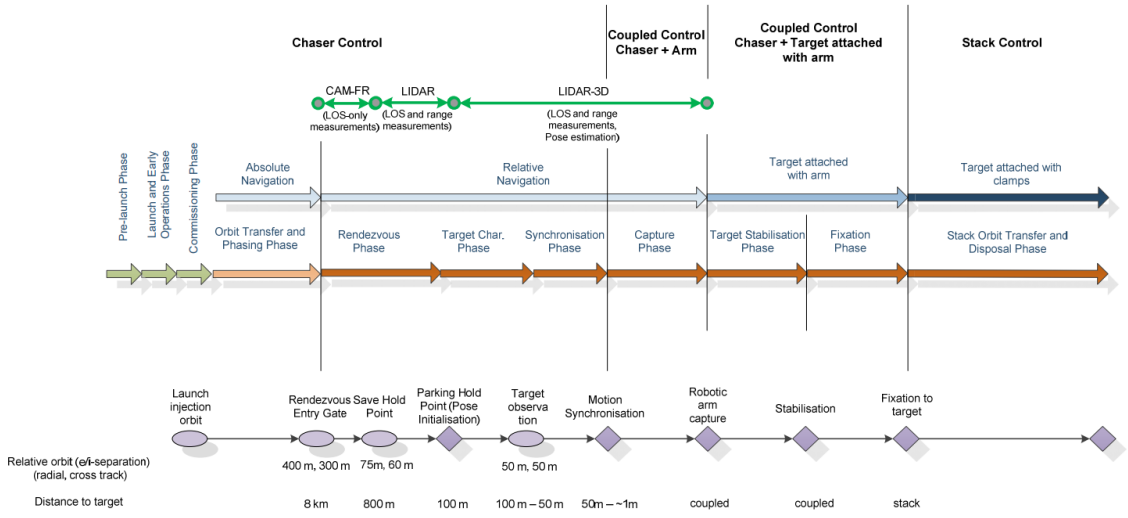


Figure 2.1: e.Deorbit mission and operation phases [30]

2.2. Mission Profile and GNC Overview

A generic mission profile for ADR/OOS mission involves the servicer spacecraft executing a rendezvous with the target spacecraft with absolute (radiometric) navigation and subsequently approaching and capturing the target using vision-based navigation. As a baseline, the operation plan from e.Deorbit specific to the Envisat target case is considered from Telaar et al. [30], as shown in Fig. 2.1. After the launch into an injection orbit, the servicer executes a series of checkouts and maneuvers preceding rendezvous with the target at a safe distance called the Rendezvous Entry Gate (REG). Until REG, the absolute state of a non-functional Envisat is determined with ground-based radar tracking systems. The state of the servicer spacecraft can be estimated on-board with GPS or through ground-based tracking. Once, the servicer spacecraft makes it to REG, the relative navigation is activated to get line-of-sight measurements for the target using a narrow-angle camera

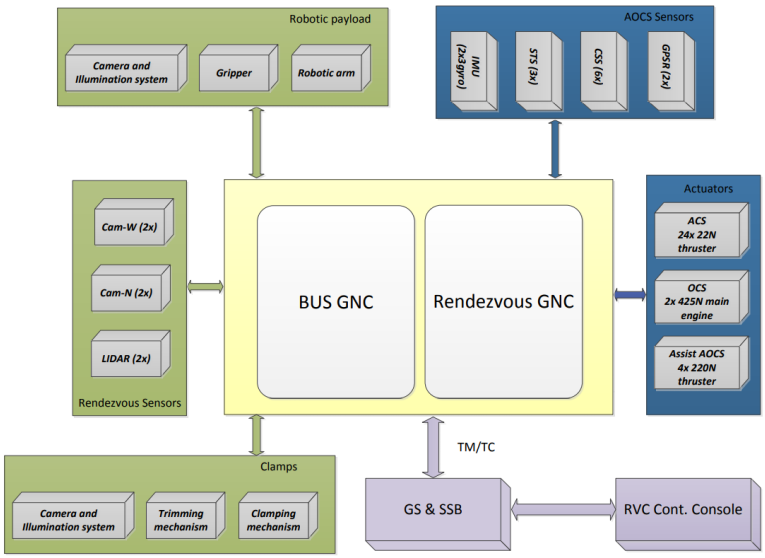


Figure 2.2: GNC architecture for e.Deorbit demonstration mission [30]. AOCS: Attitude and Orbit Control System, Cam-N/W: Camera-Narrow/Wide-angle, IMU: Inertial Measurement Unit, GPSR: Global Positioning System Receiver, STS: Star Trackers, CSS: Coarse Sun Sensors, ACS: Attitude Control System, OCS: Orbit Control System, TM/TC: Telemetry/Telecommand, GS & SSB: Ground Station and Single Sideband Stations

and/or LiDAR. The far-range navigation with a monocular camera can be accomplished using angles-only navigation [31, 32]. This is used until the park hold point, where the target's body scale in the field of view (FOV) of the vision system is sufficient for 6D pose initialization. This is the point of interest for this work, in terms of on-orbit operations. At this point, around 100 m relative to the target, the servicer spacecraft executes a hold and has to initialize the pose estimate. Note that the relative distance to the hold point is specific to Envisat and is subjective to the size and geometry of the target spacecraft. The servicer subsequently tracks the state of the target spacecraft along the V-bar. The V-bar is the direction along the passive target's velocity vector. Given an accurate navigation solution, the servicer spacecraft approaches the target along the V-bar upto around 30 m and executes the target inspection fly-around. Subsequently, maneuvers are executed for motion synchronization aimed at the launch interface ring of the Envisat spacecraft leading up to the robotic capture. The GNC system architecture for ADR or OOS mission comprises of a rendezvous GNC subsystem, in addition to the absolute GNC system used by the servicer spacecraft itself. Fig. 2.2 shows a typical GNC architecture from the e.Deorbit mission aimed at rendezvous and capture of the Envisat spacecraft. The absolute state of the servicer spacecraft is estimated using GPS. In the case of e.Deorbit concept, the narrow-angle camera was used during the far-range approach, while the LiDAR was used for the mid-range approach. Finally, at the park hold point (100 m range), LiDAR was used to initialize and track pose inside the navigation loop.

The operational phases in Fig. 2.1 and the architecture shown in Fig. 2.2 provide a representative baseline, for the Envisat target case. However, the proposed framework eliminates the LiDAR sensors from the rendezvous navigation hardware, by solely utilizing a monocular camera. The scope of this work is to design a monocular vision-based navigation framework that enables accurate pose initialization and state estimation from the hold point (assumed at 150 m). A conservative scenario used in e.Deorbit analyses [30] of involves the Envisat spacecraft tumbling along the V-bar at 5 deg/s . The resulting system supports navigation in subsequent operations leading up to the capture. However, the navigation analysis is restricted to the state tracking of the Envisat Target at the park hold point. The navigation analysis for the subsequent approach and capture phases needs to be investigated with guidance and control in the loop, which is beyond the scope here.

In terms of GNC performance requirements, there is no established benchmark for uncooperative proximity operations, and are highly specific to the overall system of systems used during the mission. However, drawing from the current knowledge of the missions, Table 2.1 shows a set of performance requirements expected from RPO systems outlined in Mitchell [33] in a generic sense and in Telaar et al. [30] specified for the Envisat target case. Of particular interest to this work is the park hold and pose initialization phase close

Parameter Performance (each axis)	**at 10 m** [33]	**at 100 m** [33]	**at 100m** [30]
Attitude Knowledge (deg)	0.3	0.3	5
Attitude Rate Knowledge (deg/s)	0.1	0.2	0.5
Relative Position Knowledge (m)	0.2	5	10
Relative Velocity Knowledge (m/s)	0.01	0.02	0.1

Table 2.1: Typical RPO performance requirements in close proximity [30, 34]

to 100 m relative to the Envisat target, in the baseline operations. A similar hold point is assumed for the initialization of close-proximity navigation, in this case at 150 m.

2.3. Monocular Vision-based Navigation

For primary navigation purposes, a monocular sensor was first used in the AVGS system for the DART and OE missions discussed earlier. Since then, the monocular vision-based navigation systems have also been proposed as a crucial technology for close-proximity operations in OOS, ADR, and other applications [35]. Monocular sensors are a promising option for spacecraft navigation systems as they have low mass and power characteristics along with a lower hardware complexity [8, 11]. In this section, a brief review of existing approaches using a single monocular camera for navigation around an uncooperative target in close-proximity is presented. The monocular sensor/camera from here on refers strictly for a visible spectrum monocular camera. However, no further distinction is made between multi-spectral and monochrome cameras.

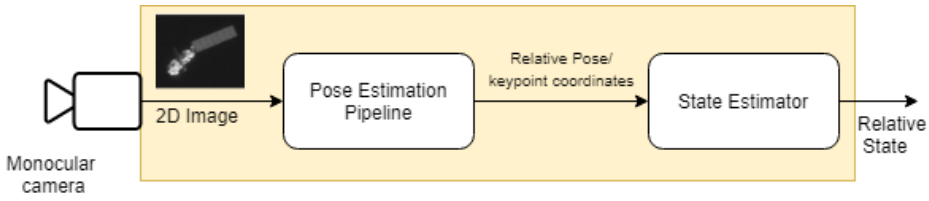


Figure 2.3: Overview of high-level blocks of a generic monocular vision-based navigation system, encapsulating various approaches

The problem of pose estimation for navigation can broadly be divided into pose initialization and pose/state tracking. The initialization process requires the pose estimator to estimate the pose with no prior knowledge. Following this, the pose and state may be tracked using previous pose and state estimates. A high-level abstraction of a monocular vision-based navigation system is shown in Fig. 2.3. The schematic shows a generic representation of pose estimation systems that use a single monocular image in the loop. The image acquired by the monocular camera is fed into the pose estimation pipeline to provide the pose estimate for initialization. Depending on the architecture, the pose estimation pipeline may encompass an IP subsystem and a pose solver. Depending on the type of the navigation loop and the design of the state estimator, the input to the state estimator for tracking may be high-level information such as predicted pose estimate or low-level information like predicted keypoint coordinates. The navigation filter subsequently estimates the full dynamical state, which usually comprises of the filtered relative pose along with relative velocity and angular rates.

The problem of pose estimation is a part of the vision-based navigation system that is not common to conventional GNC architectures. The task of processing an image as an array of numbers to synthesize higher-level information of the subject(s) in the image, falls under the interdisciplinary domain of Computer Vision (CV). The field of CV, born in conjunction with Artificial Intelligence (AI) as a research field around 1960, emphasizes on visual perception and providing computers with human-like ability to perceive [36]. It takes inspiration from the human visual system to enable machines to perceive 3D geometry from a 2D image. Depending on the level and type of information extracted, a problem is classified into one or more CV tasks. Most real-life visual data processing systems execute these CV tasks like image segmentation, object recognition, object pose estimation, and others. The problem solved by the pose estimation pipeline in Fig. 2.3 falls primarily under the purview of CV problems. Conventionally, Image Processing (IP) algorithms are a part of a CV task. IP methods are associated with processing a 2D image into a transformed image that reveals relevant visual information.

2.3.1. Review of Pose Estimation Pipeline Architectures

Depending on whether or not the estimator uses a known 3D model of the specific spacecraft, conventional pose estimation pipelines can be classified into model-dependent or model-agnostic type architectures. Model-agnostic architectures are those that do not require the knowledge of the object's 3D model and do not use model-specific information in the pose estimation loop. The classification presented here is not established in the literature, especially regarding the learning-based methods. The frequently adapted '*model-based*' and '*non-model based*' classification is ambiguous and excludes representation of many new approaches. The nomenclature and classification presented here also clearly divides the scope of application in space missions. For instance, an ADR or OOS mission aimed at a specific spacecraft can use model-dependent architecture. However, if the spacecraft needs to execute uncooperative close-proximity operations with multiple targets, the pose estimation pipeline needs to be model-agnostic. However, the literature is deficient in terms of such model-agnostic methods that combine the advantages of using a monocular vision-based system with the adaptiveness and applicability to non-specific targets. Hence, there is significant scope and promise for the future works to explore model-agnostic methods.

The preliminary classification [1] shown in Fig. 2.4 further breaks down into pose estimation methods used in respective architectures to a relevant level of detail (Note the learning-based methods in different parts of the two architectures). In the following discussion, various classes of pose estimation architectures used in the literature are reviewed briefly.

1. **Model-dependent feature-based pose estimation**

   Feature-based methods detect these notable features from the acquired 2D image and relate it to the feature locations on the known 3D model of the target spacecraft. From a CV perspective, the features on the target body include corners, edges, keypoints, and depth maps. The pose estimation process, in this case, comprises of two sequential processing tasks - feature detection and localization and solving the perspective projection to estimate the pose. First, the feature detection and localization subsystem processes the image and provides 2D feature coordinates in the image plane. This is then fed to the PnP solver which estimates the pose that projects the known 3D model to fit the detections.

   IP-based pipelines: Conventionally, IP subsystems have been employed to extract the features. The IP subsystems extract visible surface features such as corners, edges and keypoints by enhancing or transforming the pixel information in the original image. For feature extraction, IP systems utilize encoded representations called descriptors, which express the characteristics like color, texture or shape, relevant in identifying a pre-defined feature in the image. For pose initialization, the detected features need to be associated with the known features in the spacecraft 3D model. This data correspondence step results in a matching matrix corresponding features in the 2D image and the 3D model, which is then used in the PnP or pose solver. Feature correspondence is done using methods like RANSAC [13], Soft assign [37] and feature groups [14]. These correspondence finding algorithms typically test several correspondence hypotheses when the pose is being initialized from a lost-in-space state, making them extremely slow due to their computational complexity. This is a significant shortcoming of feature-based pose estimation methods that use IP subsystems for in-orbit operations. Further, IP systems are challenged by adverse illumination, background textures and slow data correspondence, while often lacking solution availability needed for application in spacecraft navigation [9, 10, 35]. The reader is referred to the works in Opromolla et al. [38], Sharma et al. [8] and Pasqualetto Cassinis et al. [12] for a thorough review of IP methods and algorithms.

2. **Model-dependent end-to-end pose estimation** End-to-end pipelines are usually represented as one functional block or subsystem, all encompassing, that take an input image and whose output is a pose estimate. This is in contrast to the feature-based methods, where two clear subsystems exist that are independent in terms of choice of implementation. Additionally, the blocks in a feature-based method interact with input data at different stages in the process. On the other hand, end-to-end methods can generally be considered as the methods having single point of input (image/model) in the model diagram. The latter clarification is necessary along with single functional block as the algorithms or methods can be broken down into smaller process blocks creating ambiguity. For clarity,

[1] Note: The new nomenclature is meant to organize and systemically present the knowledge from the vast body of literature to present the scope and applicability of learning-based methods to the problem in a clear manner.

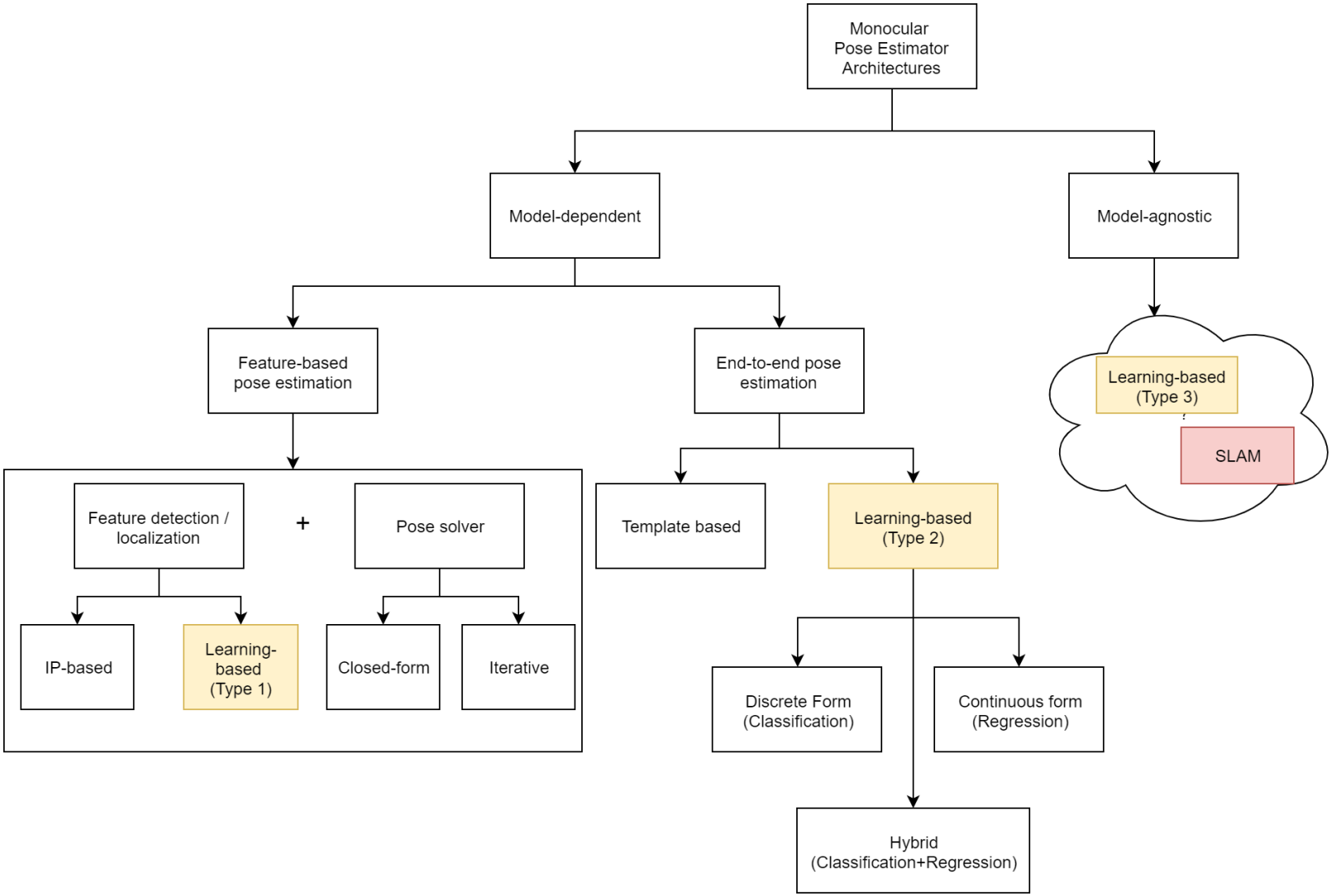


Figure 2.4: Classification of pose estimation architectures and elements

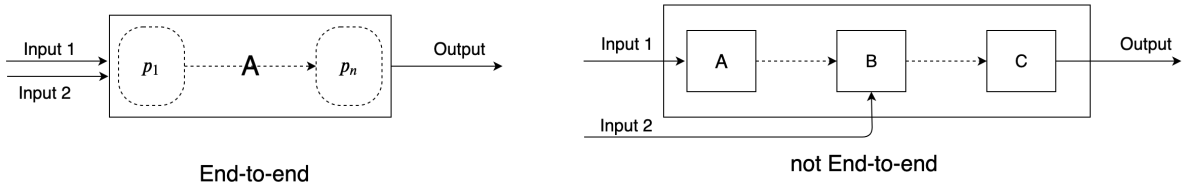


Figure 2.5: Distinction of End-to-end methods

this is shown in Fig. 2.5, where uppercase letters represent higher level subsystems and p_n represent internal sub-process without any further explicit definition. Among the conventional methods (non learning-based), Template-based methods are the only way to synthesize an end-to-end pose estimation pipeline.

Template-based Methods Template-based methods are based on template matching, which is a process used often in CV to match a section of the image with a template database generated offline. In the offline template generation, the complete six-dimensional pose space is sampled and a corresponding template is generated using a 3D model. In the online processing, the acquired image is matched to a template image using correlation techniques like the sum of absolute differences [39] and normalized cross-correlation [40]. Based on the degree of similarity in the image template, the best-correlated template image provides the corresponding pose assigned offline. While template matching has been used for other purposes, a monocular pose estimation application is only addressed in [41] for terrestrial CV problems. Opromolla et al. [42] used template matching for spacecraft uncooperative pose estimation, which used LIDAR as the primary navigation sensor. The main disadvantages of template-based methods include memory-inefficiency in storing template database and computationally for higher pose resolution and computation-inefficiency in intensive cross-correlation that ensues with such a database.

3. **Model-agnostic pipelines** The Model-agnostic architecture is an isolated category that is focused on the non-specificity (or non-availability) of the target spacecraft's 3D model. This needs to be emphasized as the area of significant importance in enabling the ideal ADR missions. Model-agnostic methods in CV have utilized stereo cameras [43] and optic flow sensors [44] for terrestrial applications. In that respect, model-agnostic systems using monocular vision have not been investigated for pose estimation around an uncooperative spacecraft. However, it is not certain how such systems/algorithms will be realized. One of the conceivable approaches uses learning-based methods to estimate pose by regressing coordinates of an enclosing box [45, 46]. However, the detailed investigation on the performance and robustness of such an approach is not in the scope of this work. Note that the SLAM approaches [15] which use a batch of monocular images are model-agnostic in nature. However, these approaches have limitations for real-time operations in orbit [16].

2.3.2. Pose Estimation and Pose Solvers

The most promising approaches relying on feature-based pipelines estimate the pose by solving the PnP problem. It is therefore relevant to review the PnP problem in detail along with pose solvers used to solve it.

Perspective-n-Points (PnP) problem

Perspective-n-Points problem (coined by Fischler and Bolles [13]) is the problem of determining the position and orientation of a camera relative to the object under observation, given camera intrinsic parameters and a set of n corresponding 3D body coordinates and their 2D projections. As highlighted earlier, this is at the core of accurate pose estimation in feature-based pipelines.

Given the knowledge of the target's 3D model points, the camera intrinsic parameters and 2D location of the feature in the image frame, the problem of determining pose due to perspective geometry is shown in Fig 2.6 and given as follows. Consider $\mathbf{r}_i^B = [x_i, y_i, z_i]^T$ n 3D points ($i = 1,2,3...n$) of interest on a known 3D model in the body frame ($\mathbf{B}$) and $\mathbf{p}_i = [u_i, v_i, 1]$, n corresponding image points on the image plane. The image plane is defined with respect to the Camera frame ($\mathbf{C}$) through the intrinsic parameters. The PnP problem is concerned with determining the translation vector $\mathbf{t}^C$, and attitude transformation matrix $\mathbf{R}^{BC}$ [2], to trans-

[2] also called the direction cosine matrix or combined rotation matrix interchangeably

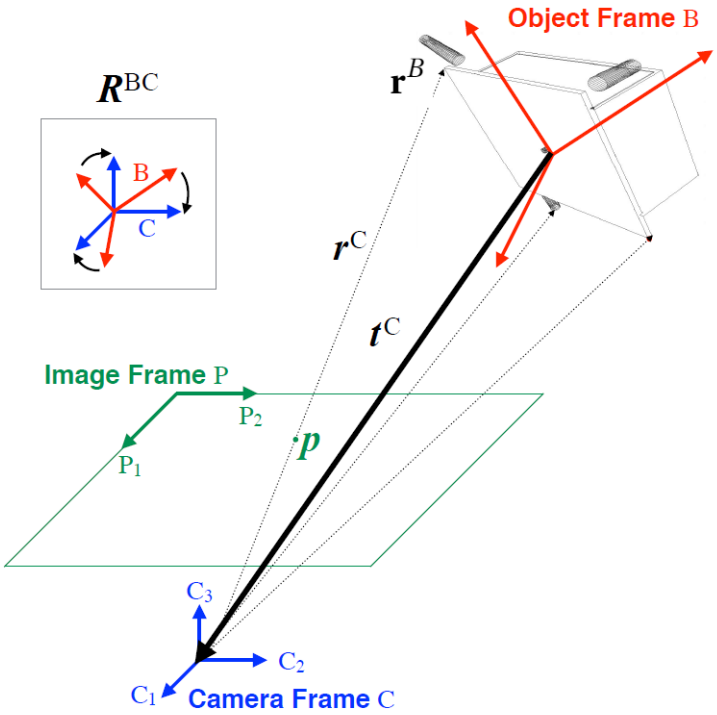


Figure 2.6: Geometric depiction of PnP Problem in spacecraft navigation [9]

form from frame **B** to frame **C**. A 3D model point coordinates $\mathbf{r}^C$ in the camera frame and its corresponding coordinates in the image frame **p** can then be expressed as:

$$\mathbf{r}^C = \mathbf{R}^{BC}\mathbf{r}^B + \mathbf{t}^C \tag{2.1}$$

$$\mathbf{p} = \left[\frac{x^C}{y^C}f_x + C_x\,,\ \frac{y^C}{z^C}f_y + C_y\right] \tag{2.2}$$

where f_x and f_y are focal lengths in the respective principle directions of the image frame, while C_x and C_y are locations of the principle points of the projected image.

To further condition the generic problem, it is assumed that the unit vector C_3 direction is along the boresight of the camera, and the unit vectors C_1 and C_2 are aligned with the image frame axes P_1 and P_2. In that case, the Eqs. 2.1 & 2.2 can be transformed using homogeneous coordinates and expanded as:

$$\begin{bmatrix} w_i u_i \\ w_i v_i \\ w_i \end{bmatrix} = [\mathbf{K}]\,[\mathbf{P}] \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{2.3}$$

where, **K** is 3x3 camera intrinsic matrix, w_i are scaling coefficients for homogeneous transformation, and **P** is the 4x3 pose matrix containing attitude transformation matrix and translation vector. Assuming zero distortion and square sensor pixels, Eqn. 2.3 can be expanded to

$$\begin{bmatrix} w_i u_i \\ w_i v_i \\ w_i \end{bmatrix} = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{c|c} \mathbf{R}^{BC} & \mathbf{t}^C \end{array}\right] \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{2.4}$$

where,

$$\mathbf{R}^{BC} = \begin{bmatrix} R_{11}^{BC} & R_{12}^{BC} & R_{13}^{BC} \\ R_{21}^{BC} & R_{22}^{BC} & R_{23}^{BC} \\ R_{31}^{BC} & R_{32}^{BC} & R_{33}^{BC} \end{bmatrix} \quad and \quad \mathbf{t^C} = \begin{bmatrix} t_1^C \\ t_2^C \\ t_3^C \end{bmatrix} \tag{2.5}$$

Pose Solvers

The PnP problem represented by Eq. 2.4 has 6 unknowns in pose matrix (**P**). **P** is defined by three independent parameters for relative attitude and 3 for relative position. Information of 2D image points provide u_i and v_i for each feature (i= 1,2, ... n) specifying two equations each. A collection of 3 (n=3) feature point locations (non-coplanar) would technically specify six equations to get six unknowns in **P**. However, there are eight general solutions for n=3, due to viewpoint ambiguity, of which four are in front of the image plane. The three concerned points on a rigid body can be placed in four distinct ways along the perspective lines, having same 2D point locations in the image frame.

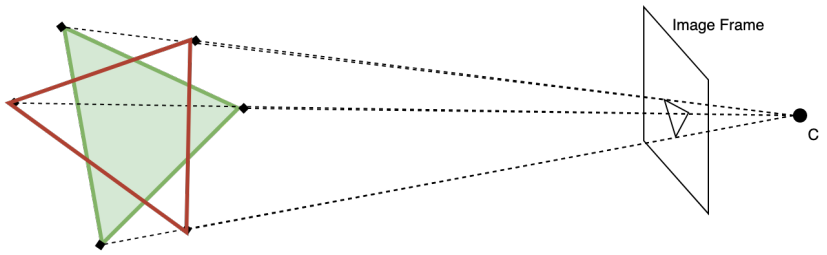


Figure 2.7: Ambiguity in P3P solutions

An example of this is shown in Fig 2.7, where two (out of possible four) orientations of the 3D object points result in the same 2D point locations in the image. Mathematically, this is due to the non-linear transformation from Euclidean space to homogeneous space. There are four such ambiguous configurations in P3P. Ideally, a unique solution is possible for a P6P problem (n=6), given six non-coplanar points on the target object. However, even with n=6 there are problems with object symmetry (which is often present in satellites) and near solutions due to noise. For instance, six identical solutions are possible for a cube in P6P problem. To overcome the limitations of direct solution approach, the PnP problem is often solved using dedicated solvers. Due to it's relevance to this work on learning-based methods, specifically to type 1, the PnP solvers are critical to understand.

Solvers for the PnP problem have been thoroughly dealt with, in the literature. Based on whether a solver employs an iterative minimization approach or multi-stage analytical approach, they can be classified as:

1. Analytical (Closed-form) Solvers: The Solvers utilize some or all point locations to solve linear forms of Eq 2.1& 2.2. These solvers provide a closed-form solution, thus not requiring an initial estimate of the pose. Often, these are computationally fast but offer comparatively lower accuracy. Some other methods, that account for non-linearities are computationally slow. For example, Efficient PnP (EPnP) [47] is the most common analytical solver used frequently in the literature.

2. Iterative Solvers: The solvers minimize an error-based objective function defined in the object or image space, solving Eqs. 2.1 & 2.2 taking non-linearities in to account. These solvers tend to be accurate but require an initial guess to start the iterative minimization and are can be computationally slower for a desirable performance. Iterative optimization does not necessarily provide a globally minimum solution for the objective function. POSIt [48], Coplanar POSIt [49] and Lu-Hager-Mjolsness (LHM) [50] methods are the some of the common iterative solvers in the literature.

Often modern solvers bridge the gap between analytical and iterative solvers in terms of computation and accuracy, by combining the two approaches. For instance, EPnP is most often used with Gauss-Newton iterative refinement to improve the solution accuracy. Unlike purely iterative solvers, the hybrid solvers do not need a priori estimate of the pose to initialize.

Among the state-of-the-art EPnP finds an efficient closed-form solution to a PnP (n>4) problem, where the complexity for n is linear ($\mathcal{O}(n)$). The PnP problem is reduced to that of estimating positions of four hypothetical control points, which are estimated as the weighted sum of n 3D points, forming a linear system of equations. EPnP's linear solution is often refined using Gauss-Newton refinement and has been employed for spacecraft pose estimation [17, 45, 46]. Newton-Raphson root-finding algorithm can be employed for iteratively solving the PnP problem [35], but has a very high computation complexity due to an expensive matrix inversion operation. LHM [50] method iteratively minimizes the reprojection error in the object space using a weak perspective assumptions. While LHM is an accurate, the computation time is found to be four times higher than the already slow Newton-Raphson method [8]. Robust PnP [51] proposes to robustly solve the PnP problem using a seventh-order polynomial to estimate locations of hypothetical stationary points (similar to EPnP control points) with lowest re-projection error. Other state-of-the-art methods include the Direct Least-Square (DLS) solver [52], Semi-Definite Program (SDP) [53], Optimal PnP (OPnP) solver [54] and Accurate and Scalable PnP (ASPnP) solver [55].

Most solvers discussed above take only the 2D image-point coordinates as input assuming equal detection accuracy across observed points and ignoring the observation uncertainty. Ferraz et al. [56] propose the Covariant Efficient Procrustes PnP (CEPPnP) solver which for the first time leverages observation uncertainty. By utilizing feature detection uncertainty in the form of covariance, CEPPnP essentially solves the EPnP linear system of stationary control points. The covariance information is transformed from image plane to the space in which the control points are defined and the system is solved using Maximum Likelihood minimization. Due to EPnP like formulation, CEPPnP also facilitates computationally efficient pose estimation. Pasqualetto Cassinis et al. [57] propose use of CEPPnP in a learning-based pose estimation pipeline to leverage covariance information to facilitate accurate pose initialization.

Urban et al. [58] propose a novel PnP solver that utilizes Maximum Likelihood Estimation (MLE) and takes observation uncertainty into account. The PnP problem is reformulated in the null-space of the normalized camera-frame keypoint coordinates, and using the minimal representation of uncertainty formulated by Forstner [59]. The maximum likelihood problem in MLPnP is formulated in terms of pose variables, unlike CEPPnP. This allows computation of uncertainty associated with estimated pose. The availability of pose estimation uncertainty as output can enable preservation uncertainty in the navigation loop improving robustness of the navigation solution. Besides, Urban et al. [58] demonstrate the speed and accuracy characteristics of the solver which is superior or comparable to all available solvers.

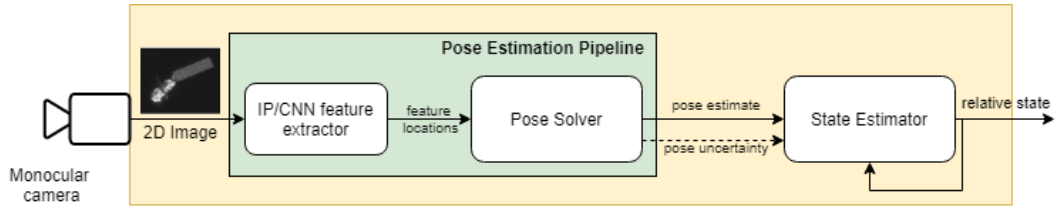
2.3.3. State Estimator

For vision-based navigation, a state estimator is used to track the relative state of the target, using the input from the pose estimation pipeline or the vision subsystem. The state estimator used the knowledge of dynamics model and the measurement model to estimate the relative state, which in most cases includes position, velocity, attitude and angular rates. The navigation filter is also used to provide state estimate at a frequency desired for the GNC loop, which may be higher than the operating frequency of the pose estimation pipeline [60]. Pasqualetto Cassinis et al. [12] review the various filters for translational and rotational state dynamics. Depending on the type of measurement model used, the state estimator architectures can be broadly divided into loosely and tightly-coupled type. These can be described as following:

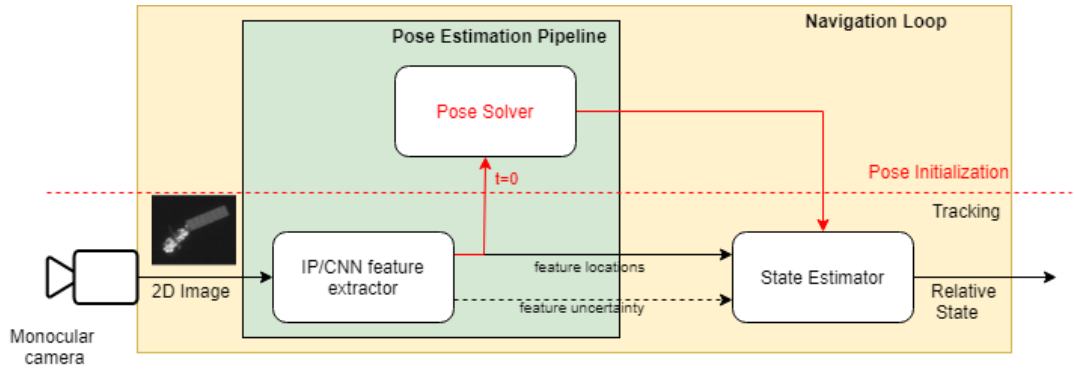
1. **Loosely-coupled**: A loosely-coupled estimator takes psuedomeasurements of the pose, estimated by the pose solver, as the measurment input as shown in Fig. 2.8(a). In a navigation loop, from a functional point-of-view, the state estimator is connected serially after the pose solver.

2. **Tightly-coupled**: A tightly-coupled estimator takes features such as keypoints, detected by the feature extractor, as the measurement inputs as shown in Fig. 2.8(b). To initialize the filter during pose initialization, this architecture needs to use the pose estimate from a pose solver. Once initialized, the flow bypasses the pose solver entirely and the features are directly fed into the state estimator. The measurement model implicitly relates the feature predictions to the pose variables with Jacobians of the perspective equations (Eqs. 2.1 & 2.2). In a navigation loop, from a functional point-of-view, the state estimator is connected serially after the feature extractor and the pose solver is utilized in parallel during pose initialization.

A loosely-coupled estimator is usually preferred for uncooperative and potentially tumbling targets, as fast relative dynamics can prevent effective feature tracking, when using IP methods. Modern pose estima-



(a) Loosely-coupled estimator



(b) Tightly-coupled estimator

Figure 2.8: Vision-based navigation system flow for various estimator architectures

tion pipelines tackle this by utilizing CNNs to provide a constant pre-defined set of features [57]. The computational complexity of the state estimation for loosely-coupled is lower as it processes six pose variables which are directly observable in terms of state, as opposed to higher number for feature coordinates used in a tightly-coupled estimator. This becomes especially crucial in estimation steps involving matrix inversion and other operations of high computational complexity.

The state estimator is usually a derivative of the Kalman Filter. Pasqualetto Cassinis et al. [12] provide an extensive review of estimators/filters used in contemporary work on navigation around uncooperative target. A common choice of filter involves an Extended Kalman Filter (EKF) that utilizes a linearized dynamic model to estimate the state, as it is more suitable for onboard operation. While EKF has been utilized in [61, 62] for navigation around an uncooperative target, it exhibits common issues in attitude state tracking when used with the common unit quaternion parameterization for attitude [3]. However, the use of quaternion representation in state tracking can cause covariance rank deficiency and overall filter instability due to the normalization constraint in the representation [60, 63]. A solution to this problem is to use a Multiplicative Extended Kalman Filter (MEKF) that uses dual representations in the filter with a reset step based on multiplicative error quaternion [64]. MEKF has recently been used in the works on uncooperative navigation for attitude estimation [60, 63]. The translation state is usually estimated using a conventional EKF.

From robustness perspective, it is necessary for an estimator to receive a representative covariance to be able to provide reliable state estimation during adverse operational conditions such camera flaring, target shadowing or other conditions that result in inaccurate detections. As pointed out by Pasqualetto Cassinis et al. [57], without an accurate representation of the uncertainty, the filter may exhibit divergence in state estimation. This is a concern for modern learning-based architectures, as discussed in the following section.

2.4. Review of Convolutional Neural Networks

To discuss the learning-based methods, it is relevant to first review the relevant details of its crucial element- the CNNs, which are a class of neural networks. Broadly, neural networks are computing models for Artificial Intelligence (AI) that rely on pattern recognition in the data and building up high-level abstract information by hierarchically breaking it down into lower-level information. The neural networks are loosely based on the concept of human neural system and feature analogous basic elements called neurons. Mathematically,

[3] See Appendix B for a review of attitude parameterizations

a neuron is a simple non-linear function that can be heirarchically composed into any arbitrary non-linear mapping by specifying the constant coefficients often called weights. The estimation of the right weights for each neuron, in order to fit the data is called learning or machine learning. Learning is executed by feeding the neural network with data and adjusting the weights to improve predictions on the data, in a process called training. Broadly, training is an optimization routine that minimizes an overall objective as a function of all the neuron weights, given the expected output, using a method like Stochastic Gradient Descent (SGD) or its derivatives. The cost function is also called loss in the common terminology and the trained network with fixed weight values is called a model or network model. For neural network models to be effective on a task, they need to be trained on large amounts of data, that enable effective learning of general and desirable patterns in that kind of data. The most important aspect of machine learning relates to the **no free lunch theorem** for machine learning [65]. The theorem state that no machine learning algorithm performs better than others when averaged over all data generating distributions. On making assumptions about the probability distribution of a specific real-world application, machine algorithms and models can be designed to perform extremely well on a specific distribution. Consequently, the algorithms and models developed on specific data distribution perform worse on other data generating distributions. This implies that machine learning algorithm cannot be universal or the absolute best learning algorithm. In simple terms- the neural network models developed with machine learning are only as good as the data.

The CNNs are deep neural networks, where 'deep' refers to the depth of representation hierarchy modeled in the network. Consequently, the learning in CNNs is referred to as deep learning. The success of CNNs was demonstrated only recently on ImageNet classification benchmark [66] by Krizhevsky et al. [67] in 2012. However, CNNs are not a recent concept. LeCun et al. [68] motivated CNNs for use in recognition of handwritten zip codes for US postal service and later formalized the CNN class of architecture that overcomes the limitations of traditional neural networks [69]. However, with trisection of advances in processing power, increase in image data and progress in deep learning, the performance of CNNs has improved significantly [70–72] that have enabled a new paradigm for computer vision applications. The following discussion is streamlined to only include the relevant design aspects of the CNNs. Readers can find the details of pre-requisite concepts in machine learning and deep learning in Appendix A.

Convolutional Neural Networks

An elementary component of learning-based pose estimation methods discussed in 2.3.1 are the CNNs. CNNs are specialized to process visual data in form of images and have a 3D structure unlike the regular neural networks. CNNs are named so because they use convolution operation in at least one layer of the network. A typical convolutional layer consists of the linear convolution operation and an activation function. In mathematics, convolution is an operation (denoted with $*$) that acts on two functions ($f(t)$ and $g(t)$) of a real-valued-argument such that the output ($s(t)$) after convolution is given as the weighted average with respect to infinitesimal time interval τ:

$$s(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau \tag{2.6}$$

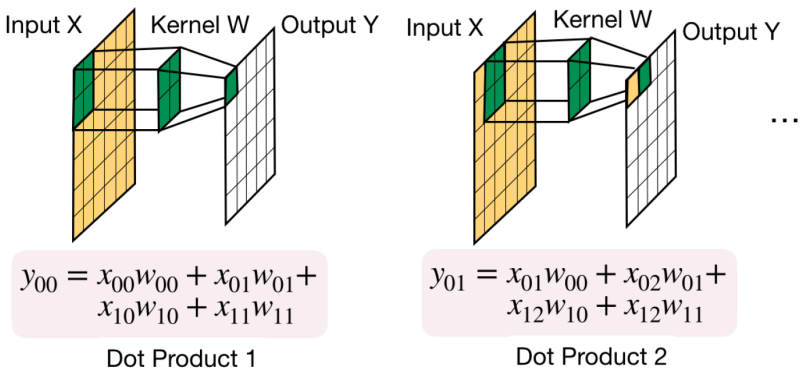


Figure 2.9: Visual representation of convolution operation on a single channel 2D image [16]

However, in visual data processing the convolution refers to sliding dot product or cross-correlation operation with a kernel as visualized in Fig. 2.9. For discrete pixel values of an input image, convolution is represented by $\mathbf{X}$ and a filter or kernel $\mathbf{W}$ [4] to provide an output feature map S, in discrete space described in pixels. These multidimensional arrays are called tensors. Note that the integral between $-\infty$ and $+\infty$ in the convolution operation, will be an infinite summation in discrete space. This is dealt with by assuming that the functions, or in this case, input image described by $\mathbf{X}$ and the kernel $\mathbf{W}$ are zero everywhere but in the finite set of pixels/points. The convolution operation is then described for a 2D image input (single channel) for instance as:

$$y_{i,j} = s_{i,j} = (\mathbf{X} * \mathbf{W})_{i,j} = \sum_m \sum_n x_{m,n}\, w_{i-m,j-n} \tag{2.7}$$

where, (m,n) are pixel indices in the 2D input image and (i,j) are indices of the corresponding pixel in the resulting feature map. Visual representation of how inner product of the input and kernel tensors form feature maps is shown in Fig. 2.9 for inner summation in n. The complete convolution operation can then be visualized as sliding a constant weight filter over the image and evaluating one pixel of the feature map at every step. Due to the difference in the size of the filter and the image, the convolution layer is able to detect local features in a region of the image, revealing a a higher level of information hierarchy from pixels. Note that the convolution layer linearly combines a local region of inputs with several weight filters. Generally, a convolution layer uses multiple weight filters on the image. Then, the output is a 3d volume of 2D feature maps, whose depth is equal to the number of weight filters used. The convolution layer is succeeded by an activation layer, which has neurons that transform the linear combination of inputs into a non-linear activation map. This also means the arrangement of neurons that produces that output volume is 3-dimensional of same size. A non-linear activation layer (with neurons) succeeds the convolution layer, which transforms this linearly transformed feature map into an activation map. An activation layer is composed of neurons. As the filter slides over the input image, each dot product is associated with the input to one neuron in the activation layer. The activation layer adds non-linearity in the information. Most often, modern neural networks use a Rectified Linear Unit (ReLU) function in the neurons (see Fig. A.6 in Appendix A). The intermediate information in form of multi-dimensional arrays (tensors) after each layer is called a feature map, as it represents relevant aspects of the image. As the information passes repeatedly through convolutional layers, higher level information and context is represented in the feature maps. Ultimately, the information is boiled down to the output variable.

A CNN development has two stages:

1. **Training** : Training is the stage when the example data is provided to the CNN for learning the weights. The network forward propagates the image input through the network and evaluates the loss. Based on the loss, an SGD like approach is used to back-propagate and adjust the weights to lower the loss value. This is done in mini-batches over thousands to millions of images. The forward and backward pass during training are very computationally intensive routines are generally done over Graphical Processing Unit (GPUs) rather than CPUs. Once training completes successfully, the weights are fixed and the CNN model is ready.

2. **Inference**: Once a trained model is available, it can be used with images to infer the desired output. At this stage, the image is fed to the network and propagated through all the fixed-weight layers in the network. In effect, the forward pass during the inference is a large scale cascade of simple operations like additions and multiplications.

The CNNs for vision tasks are most often trained in a supervised manner, wherein examples of images are fed to the network and the appropriate convolutional and activation weights are estimated to reduce the loss. The interesting aspect of training CNNs and deep neural networks is that the internal representations or intermediate variables that express the relationship between two feature maps are automatically generated. Therefore, given enough data CNNs can extracting abstract representations at an unintuitive levels of information in the images, to compose high level information. This is why the CNNs have been successful in tackling some of the challenges faced by IP systems, where a higher interpretability of the information is required. These include reasonable robustness to viewpoint variation, scale variation, deformation, occlusion, illumination conditions and background clutter among others. The CNNs for terrestrial use are trained on the order of a million images with large scale datasets like ImageNet [66] and Common Objects in Context

[4] kernel is of equal depth as the image, but not necessarily in height and width

(COCO). These datasets contain a broad range of images that test performance and robustness of CNN models on the aforementioned aspects in the image. Consequently, the benchmark comparison and the selection of CNN architectures are often made based on their performance on these benchmark datasets.

The network structure underlying modern CNNs can be extremely large with hundreds of layers. Consequently, the memory and computation efficiency are critical indicators for CNN deployment in real life. For this purpose a CNN model is usually judged and compared on two specifications:

1. **Parameter Count**: It is the total number of parameters or weights of coefficients that define the model. This represents the size of the model to be stored on a device. Typical large CNN architectures have on the order of fifty million parameters on an average. More optimized CNNs have between one and ten million parameters. Note that the parameters only provide a quantification of the model size and are not reflective of computational efficiency.

2. **Floating Point Operations**: FLOPs indicate the total number of floating point operations required to process the image through all the layers of the network during a forward pass. FLOPs represent the computational efficiency of the CNNs. FLOPs are quantified on the scale of billions (or 1 GFLOP). Modern state-of-the-art CNN architectures require as low as 0.5 Bn FLOPs and as high as 200 Bn FLOPs per forward pass.

From a robustness point of view, CNNs rely heavily on the training data to generate the model and consequently fail when they encounter an input outside of the training data distribution. The effectiveness of the trained model in practical applications can be seriously hampered due to limited training distribution. This problem is particularly important for spacecraft pose estimation using CNNs, as the models are trained only using synthetic rendered imagery. In order to improve robustness of the network to unmodelled optical artefacts, data augmentation techniques are most often used.

2.5. Learning-based Pose Estimation and Navigation

Recently, the work on uncooperative monocular pose estimation has leveraged the CNNs and deep learning with the initial work of Sharma et al. [9]. In general, learning based pose estimation pipelines use a CNN model trained to predict a specific type of output such as pre-defined keypoints or parameterized pose. As discussed in the previous section, the CNN models generate a complex non-linear mapping between the input image and the output information. In this case, the mapping is generated during training phase using a large database of representative images of the spacecraft. Since real space-borne images of the target spacecraft are not available at the scale required for deep learning, the training of such models relies on synthetically rendered images of the target spacecraft [9, 45, 57, 73].

Pose Estimation

The early pose estimation pipelines [9] employed CNNs for pose in an end-to-end manner (Learning-based architecture Type-2 in Fig. 2.4), wherein the CNN predicts a 6D pose vector from a monocular image, based on classification or regression problem formulation. Classification formulation discretizes the pose space and trains the network to map the image to a confidence score for each possible pose vector in the discretized space.On the other hand, the regression formulation aims to learn a non-linear mapping that directly regresses six values corresponding to each of the pose variables. Pose classification formulation has been utilized in terrestrial applications [74] as well as the initial efforts in leveraging deep learning for pose estimation around an uncooperative spacecraft [9]. On the other hand, regression formulation has been used most often used in terrestrial applications [75–77]. Another formulation to estimate pose uses a hybrid approach using classification and regression in a parallel branches.

More recently, the learning-based methods for pose estimation have used CNNs to extract keypoints [46, 57, 78]. The resulting (learning-based architecture Type-1) pipeline replaces the IP subsystem to overcome some of the limitations prevalent in IP, while getting the accuracy of feature-based pose estimation. Typically, two CNNs are used in a serial manner, followed by a pose solver. The first network performs object detection (OD) to identify and localize the spacecraft in a Region of Interest (RoI), while the second network performs keypoint detection (KD) within the RoI to identify and localize keypoints. Subsequently, the pose solver uses keypoint detections and the 3D model to estimate the pose. The primary advantage in using CNNs for feature extraction is that the KD network uniquely identifies the keypoints in an image. The output from

the CNN comprises of keypoints in a pre-defined order. Therefore, the 2D-3D point correspondence is not needed, eliminating the need for computationally expensive routines such as RANSAC. While CNNs provide robustness to illumination, the use of separate localization using the OD network further provides robustness to scale variation [45].

When Type-1 and Type-2 learning-based pose estimation methods are compared, Type-2 methods show superior accuracy performance [79]. This has also been noticed in a recent work [80], which notes that the end-to-end or absolute pose regression methods "... *tend to estimate the pose close to the training poses, in regions where little training data is available*". For this reason, these methods do not generalize very well unless an ideal distribution of training data is available. This is a significant shortcoming in the use of such end-to-end or absolute pose regression methods, due to training on synthetic images, lack of real images and smaller sizes of training datasets in general.

An essential component in the implementation and verification of learning-based methods is the image dataset for training and evaluation. Since training is a data driven process, the performance of CNN models depend highly on the size and distribution of the data provided to the network during the training. Most existing works on learning-based pose estimation use the open-source SPEED benchmarking dataset from SLAB introduced in [45] that contains close to 12000 labelled pose images of the Tango spacecraft of the PRISMA mission in addition to 3000 evaluation images. However, SPEED is the only established dataset available. Pasqualetto Cassinis et al [57] use a synthetic image dataset of the Envisat spacecraft to train a KD network. The focus of all the past works has been purely oriented towards a single dataset with a fixed distribution. Given the major relevance of data in deep learning algorithms, it is necessary that datasets of synthetic images be explored and more standard databases be established to identify knowledge gaps originating in the data. Kisantal et al [79] also motivate creation of newer datasets in the future that reflect the gap between syntheticity and reality for better training of learning-based methods.

Navigation

With a CNN-based pose estimation pipeline, the state estimator can be integrated in a loosely-coupled architecture or a tightly coupled architecture shown in Fig. 2.8. A loosely-coupled estimator can be used with Type-1 as well as Type-2 learning-based pose estimation architecture since the measurement input to the estimator is in terms of pose. On the other hand, the tightly-coupled architecture can only be used with Type-1 learning-based pose estimation architecture, as it utilizes keypoint information for tracking. This is not a concern, as most prevalent approaches currently use Type-1 architecture.

From a navigation point-of-view there is limited analysis on integrating a CNN-based pose estimation pipeline with a state estimator with analysis in [57, 61]. Further, uncertainty in the navigation loop is generally not taken into account and a constant measurement covariance is used in the filter instead. As discussed in Section 2.3.3, this can cause divergence in the state estimates during adverse visibility conditions. Pasqualetto Cassinis et al. [62] propose a solution to quantify feature detection covariance from CNN heatmaps and subsequently use CEPPnP [56] solver to produce better pose estimates and utilize a tightly-coupled EKF. There is a significant gap in the literature on the analysis of navigation using learning-base pose estimation pipeline. Investigation of navigation and analyses on various scenarios are a critical necessity for assessing the potential of learning-based systems for reliable on-orbit operations.

2.6. Conclusions

The problem of monocular pose estimation and vision-based navigation around uncooperative spacecraft has been thoroughly reviewed. Projecting the limitations and applicability constraints of the currently established methods for reliable on-orbit operations, the scope of learning-based methods has been introduced and motivated. The foundational aspects of CNNs and deep learning have also been reviewed to support the reasoning behind the subsequent analyses.

II

Navigation Framework Design

3

Framework Architecture Overview

Based on the existing approaches to pose estimation and navigation around an uncooperative target, a novel navigation framework is proposed. The framework encompasses the elements, methods, and processes that compose the navigation loop or support the development of individual components within the loop. In this chapter, a high-level description and overview of the designed framework are presented. Distinctions are made between the designed framework and the related works to broadly highlight the novelty and the scope of contribution.

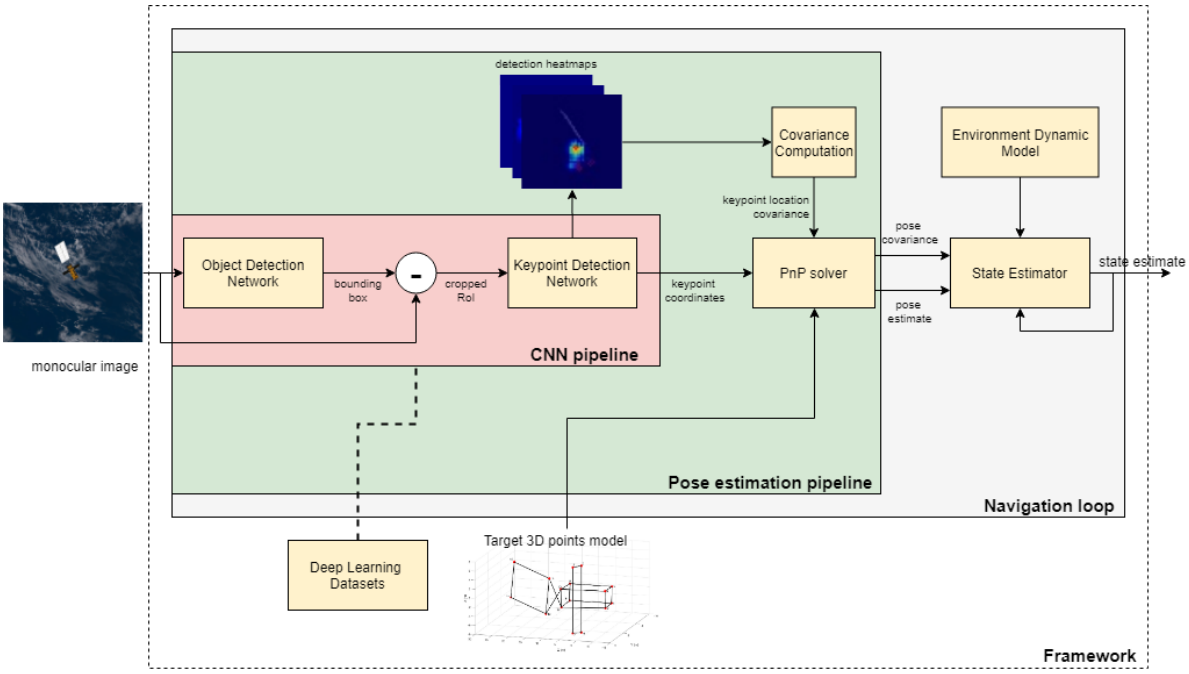


Figure 3.1: Schematic of the designed framework architecture

3.1. Architectural Overview

On the framework level, a model-dependent Type-1 Learning-based architecture (see Table 2.4) is utilized. Fig. 3.1 shows the overall framework architecture schematic providing the flow within the navigation loop and distinction between various parts of the framework. CNNs are used to extract keypoints from the input image in the CNN pipeline as opposed to traditional IP subsystems. Since the framework is model-dependent, it requires a 3D wireframe model of the Target spacecraft with coordinates of pre-defined points in the body frame. The CNNs are integrated with a pose solver to form the pose estimation pipeline. Unlike IP subsys-

tems, CNNs identify specific keypoints in the image providing default detection correspondence. Hence, the 2D-3D data correspondence step is not required before the pose solver. The KD network also take occlusion into account, enabling detection of hidden keypoints in the image. This provides a constant number of keypoint coordinates in a pre-defined order. To leverage uncertainty in the loop, KD covariance is computed from the detection heatmap outputs from the KD network. This is utilized by a covariant pose solver. The pose solver uses the predicted keypoint coordinates from the CNNs, corresponding 3D body-frame coordinates, and the keypoint covariances to estimate the 6D pose. A loosely-coupled estimator type is used, wherein the state estimator is integrated serially after the pose solver. The state estimator uses pseudo-measurements of pose provided by the pose solver and the covariance associated with the pose variables as the measurement covariance. The state estimator estimates the pose and the linear and angular rates at the desired frequency which may not be equal to the frequency at which the pose estimation pipeline can provide pose estimates. Parts of the framework that external to the navigation loop include the image datasets that are used to train or test the CNNs anas well as the 3D wireframe model of the spacecraft, which is utilized in the PnP solver.

CNN pipeline forms a crucial part of the navigation, tasked with extracting pre-defined keypoints from a single 2D image. A separate spacecraft localization step is utilized with an Object Detection (OD) network, which is then integrated into a Keypoint Detection (KD) network. The OD network provides a bounding box that localizes the spacecraft or a part of the spacecraft in the image. The bounding box is then used to crop an RoI of constant aspect ratio and resize it to match the input size of the KD network. Effective zoom availed by RoI cropping provides scale robustness to allow accurate KD for a wide range of relative separation, in which the spacecraft size in the image varies drastically. Further, the KD network is selected such that the detections are represented with heatmaps rather than deterministic keypoint coordinates. The choice of this specific CNN pipeline is inspired by a state-of-the-art top-down approach for the terrestrial Human Pose Estimation problem [81] and reinforced by comparative performance analysis of spacecraft pose estimation pipelines in Kisantal et al. [79]. The networks are trained using large-scale deep learning datasets containing synthetic images of a representative rendering of the specific spacecraft. This work investigates selection, configuration, training and evaluation of both the networks in Chapter 4 and 5.

Another salient aspect of the architecture is the quantification of uncertainty withing the loop, inspired by Pasqualetto Cassinis et al. [57]. Using heatmaps from the KD step, the observation uncertainty is indirectly quantified in the loop, unlike other approaches which assume constant accuracy of KD. This is used by the covariant pose solver to accurately and robustly estimate the pose. Finally, the pose solver is integrated with the state estimator in a loosely-coupled manner. The estimator takes in the covariance of the estimated pose as the measurement covariance for filtering. The novelty in this part of the pipeline is in utilizing MLPnP as the pose solver and enabling uncertainty preservation in a loosely-coupled state estimator. The MLPnP solver is selected as it is the only available solver that utilizes observation uncertainty and implicitly allows the computation of the pose uncertainty required for a loosely-coupled estimator, as discussed in Section 2.3.2. It may be possible to propagate the KD uncertainty to derive pose uncertainty while using other pose solvers in the pipeline. However, the task is non-trivial as intermediate representations of uncertainty must be derived for the space in which the solver frames the PnP problem, which may not be linear. A detailed investigation of the MLPnP solver and the uncertainty propagation in the solver is presented in Chapter 7 and the state estimator design based on an Extended Kalman Filter is described in Chapter 8.

The framework is evaluated for two spacecraft cases- Tango from the PRISMA mission and Envisat, using existing datasets. The image datasets of the Tango spacecraft in *SPEED* [82] are used for benchmarking and validation, while the Envisat image dataset from Pasqualetto Cassinis et al. [57] is used for evaluation. Subsequently, improvements are proposed to generate a new dataset for the Envisat target case. The new dataset adds real Earth background and various image corruptions in the synthetic images to enable robustness in the CNN pipeline to real conditions. The discussion on Datasets is detailed in Chapter 6. All the relevant elements of the framework are subsequently evaluated and tested for various design choices. Further, the design, analyses, and results are constantly and consistently with the existing works in the literature to highlight the contributions and distinctions from a global perspective,

3.2. Related Works

Sharma et al. [9] first proposed the use of CNNs for space-borne pose estimation. The initial approach was based on using AlexNet [67], a legacy CNN by repurposing the classification network to classify 6-vector discretized pose. Subsequently, Sharma and D'Amico [17, 45] proposed a novel pose architecture with a

Reference	OD	KD	Pose Solver	Uncertainty	State Estimator	Datasets
Sharma and Damico [17, 45]	✓	X	X	X	X	*SPEED*, PRISMA flight imagery
Park et al. [46]	✓	✓	✓	X	X	*SPEED* (+TR)
Chen et al. [78]	✓	✓	✓	X	X	*SPEED*
Pasqualetto Cassinis et al. [57]	X	✓	✓	✓	tightly-coupled	*Envisat-1*
R-NAV	✓	✓	✓	✓	**loosely-coupled**	***Envisat-1*, *SPEED*, *Envisat+IC***

Table 3.1: Comparison of architecture components with related works on learning-based approach to navigation around uncooperative target. (+TR): Texture Randomized

branched CNN using a combined discrete-continuous approach. The first branch detects a 2D bounding box that tightly contains the visible spacecraft in the image as a RoI. This RoI is elementary in significantly reducing the effect of background contrast. The second branch uses the RoI to classify the image as one of the attitude classes in discretized space, providing probability/confidence in all the attitude classes. The closest class (high confidence) is chosen and fed to the third branch. The third branch then regresses the pose estimate by assigning weights to each of the closest attitude classes found in branch 2. The relative attitude estimated in branch 3 along with 2D bounding box coordinates in branch 1 is used to estimate the distance using the Gauss-Newton algorithm. This is the state-of-the-art in end-to-end pose estimation using learning-based methods. Together, all the aforementioned approaches fall in to Type-2 learning-based methods in 2.4.Park et al. [46] use OD and key point regression, followed by EPnP solver (Section 2.3.2) to solve the PnP problem in an approach similar to the one proposed here. However, the key points are regressed directly with coordinate representation in CNN, in contrast to the heatmap representation used in this work. The pose estimation pipeline demonstrated high efficiency and improved robustness with a novel texture randomized training procedure. Chen et al. [78], also used OD to first localize the spacecraft before KD. HRNet [83] architecture is used to detect manually selected keypoints, emphasizing on the maintenance of high-resolution representation in the CNN to improve pose estimation performance. The pipeline also used PnP solver based on iterative solver Levenberg-Marquardt (LM) optimization to provide pose estimation accuracy. However, the design of the pipeline used an ensemble of 6 networks for both object and KD to augment accuracy performance, making it unsuitable for on-board deployment due to very high computation demand. In the aforementioned works, the analyses are limited to pose estimation pipeline and no investigation is conducted on the state estimator.

Pasqualetto et al. [57] use a KD network to evaluate pose estimation on synthetic images of the Envisat spacecraft. The aspect of quantifying uncertainty from keypoint heatmaps is proposed and used with a CEPPnP solver to estimate relative pose, taking observation uncertainty into account. In contrast with other works, a tightly-coupled MEKF is utilized to conduct a navigation analysis on a V-bar trajectory of the Envisat spacecraft. The work proposes aspects of robustness in the navigation loop to adverse operational conditions by availing uncertainty information to the pose solver and the state estimator. However, the pose estimation pipeline in the navigation loop is truncated and the OD step is not taken into account, with KD analysis done on ideal RoI cropping in the image. Further, the synthetic image dataset ($Envisat$) used in the work contains clean images of the spacecraft without artefacts, making it significantly less challenging and less representative for assessing suitability for on-orbit operations. Among other approaches, different from all the aforementioned works, Harvard et al. [61] re-purpose a classification CNN to predict the descriptor vector, which is utilized in the IP-based feature extractor with a pose solver and a non-linear filter.

Table 3.1 summarizes a high-level comparison from an architecture/framework perspective of the similar and relevant works. R-NAV identifies the framework/architecture designed in this work. Note that a broader analysis is targeted for the navigation loop, in comparison to other works. Further distinctions and contributions in comparison to the existing works are clarified in the subsequent discussions, as necessary.

4

Object Detection Network

Object detection is a computer vision task of (identifying and) localizing a specified object in an image. A detection in an image is most often characterized by a rectangular bounding box within the image that contains the object entirely. OD is used for numerous applications from face detection in smartphone camera applications to self-driving cars. While the range of applicability of OD is broad, for pose estimation using deep learning, the OD step forms a crucial element of the top-down approach [81] for the detection of keypoints. The bounding box determined by an OD model is used to crop the RoI from the original image. The KD network is then fed the cropped RoI, from which it detects the features. For monocular pose estimation in space, OD is a crucial step as RoI cropping allows robustness to scale, variation, and background texture. Fig. 4.1 shows the visual comparison of two images fed to the KD network, with and without RoI cropping. Notice that the spacecraft is visually more distinct and the background is less dominant for RoI cropping. Further, the OD network can support multiple detections, making it modular to support other on-board tasks, such as identifying and localizing a sub-component to grasp during servicing operations. Fig. 4.2 shows an example of multiple detections for various parts of the spacecraft using OD.

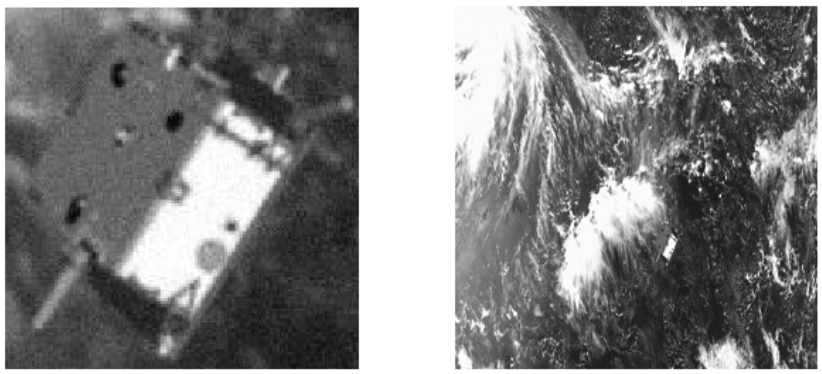


Figure 4.1: A sample image from *SPEED* dataset of Tango spacecraft at a range of 19.8m, resized to match the input size (256px x 256px) of the KD network with (a) RoI cropped with object detection, and (b) original image without RoI cropping

In this section, a brief overview of OD network architectures along with the aspects of training, testing, evaluation metrics, and the reasoning behind design selection are presented. The primary objective of this section is to present a summary of design selection for the localization of an uncooperative space-borne target in the field of view. After a review of the currently available networks, the most suitable and employable OD architecture is selected. The state-of-the-art OD architectures designed for terrestrial applications are selected directly as the task of detecting and localizing a target in a space-borne image is considered less challenging than in many terrestrial use-cases. This is because terrestrial images contain more objects, more textures, and colors, occlusions, and other artefacts, on which OD networks have already shown success [84, 85]. Relatively, a space-borne image only contains variation in illumination and possible Earth textures in the

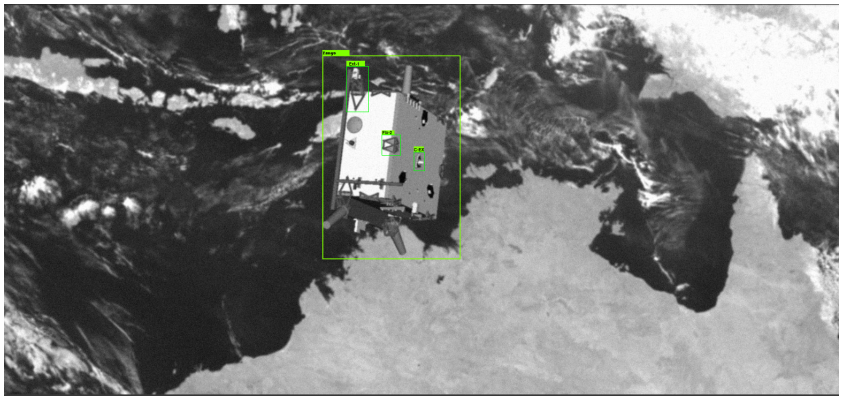


Figure 4.2: Multi-object detection possibility demonstrated on a sample image from the *SPEED* dataset

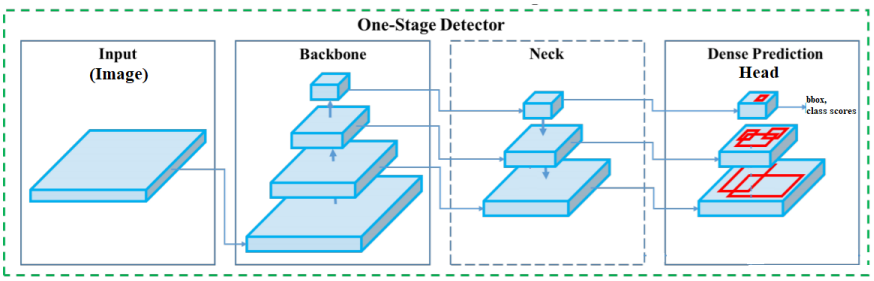
background. Following the selection of OD network architecture, the configuration and implementation to support successful training and testing of the network are outlined.

4.1. Network Architecture Selection

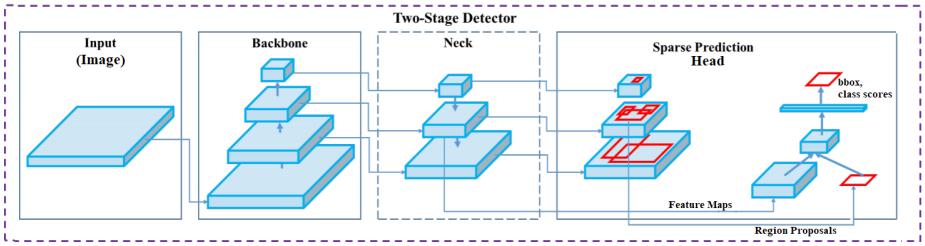
Modern object detectors rely on CNNs to achieve an unprecedented level of performance. The CNNs employed in these detection architectures solve two problems- regression of bounding box coordinates and classification of the object class. The object class refers to the name/identifier of the object, e.g: Envisat or Solar Panel. CNNs for OD are composed of two essential parts- a 'backbone', also called a base network or a feature extractor used for extracting generic hierarchical features in the image, and a 'head', also called detection head, used for regression of bounding box coordinates and classification of the object in the corresponding bounding box.

The backbone is usually a set of initial convolutional layers of a standard image classifier, often pre-trained on large datasets of natural images like ImageNet [66] or COCO [86]. The detection head is a custom network that can use the features detected by the backbone to predict bounding box coordinates and object class. The detection-head architectures can be classified into one-stage and two-stage architectures. A two-stage detection-head uses sparse prediction with 'region proposals' in the first stage. This stage proposes regions in the image that might contain the object(s). The second stage performs the classification that individually identifies the object(s) per proposal. A one-stage architecture, on the other hand, performs dense predictions in a single-stage unifying the localization and classification parts of the problem. Fig. 4.3 shows the composition of a modern OD network. Table 4.1 lists the common alternatives for each corresponding part shown in Fig. 4.3. Notice the distinction between a one-stage and two-stage detection. The input is usually an image, the blue volumes represent feature maps and the red boxes represent pixel region information. Notice the presence of an additional Neck section. Some modern detectors [87, 88] introduce this intermediate layer structure between the backbone and the detection head that aggregates feature maps from multiple detection stages (i.e. multiple scales).

It is evident that using the components shown in Fig. 4.3, the design, selection, and choice of the OD network for an application remains subjective. Given the rapidly evolving object detection sub-field, it is difficult to consistently compare one method to another. Most often, the works on detection architecture produce benchmark results against other methods that are subjective to the choice of backbone, neck, and detection head at the very least. The benchmark results are also meant for maximizing a metric, usually a parameterized score representing accuracy/precision, for the standard challenge datasets [66, 86]. When designing an object detector for a real-life embedded application, the choice of individual parts in an object detector cannot entirely be judged from the benchmark results in the original works. To tackle this, Huang et al. [90] use a constrained framework definition and evaluate the most common OD networks and underlying architectures on a fair basis. Based on the inferences outline in [90] as a baseline, the selection of detection head and backbone sub-networks is made.



(a) One-stage architecture



(b) Two-stage architecture

Figure 4.3: Composition of a CNN based OD network (adapted from [89])

Component	Alternatives
Input	Image, Patches and Image Pyramids
Backbone	VGG16 [70], ResNet-50 [71], Inception [91], CSPResNeXt [92], CSPDarkNet [92], MobileNet [93], EfficientNet [94]
Neck	FPN [95], PAN [96], SPP [97]
Head	**Single-stage dense prediction** : SSD [85], YOLO [98], RetinaNet[99], FCOS [88], CenterNet [87] **Two-stage sparse prediction** : Faster R-CNN [84], R-FCN [100], Mask R-CNN [101]

Table 4.1: Most common component alternatives for an object detector design

4.1.1. Detection Head

Two-stage architectures

The most prominent two-stage networks belong to the Regions with CNN (R-CNN) family of architectures [102]. The most recent evolution of R-CNN family is the 'Faster R-CNN' architecture [84] that achieves state-of-the-art performance in terms of detection accuracy [36]. Fig. 4.4 shows the generic schematic the R-CNN type detection heads. The feature map obtained from the backbone sub-network is fed to the Region Proposal Network (RPN), which predicts the box proposals. The box proposals are generated using default boxes of different scale, aspect ratios and at different location all over the image. These pre-defined default boxes used for generating the proposals are often called anchor boxes. A proposal consists of coordinates of the proposed bounding box and a corresponding score that represents the probability of presence of an object in that box. In the second stage, n proposals are selected with n highest scores, where n is specified by the designer. An RoI pooling converts the cropped feature map inputs of varying sizes to an output feature map of fixed size in a way that prevents loss of information [103]. Then, the remaining layers use these feature maps from the RoI pooling layer to classify the box into an object class or as the background (no object of interest). The reader is encouraged to review the details of the architecture and design from the original paper [84].

One-stage architectures

The architectures that use one-stage detection head are broadly called single shot OD architectures. In contrast to the branched structure of two-stage architecture, single shot detectors compute box and object class

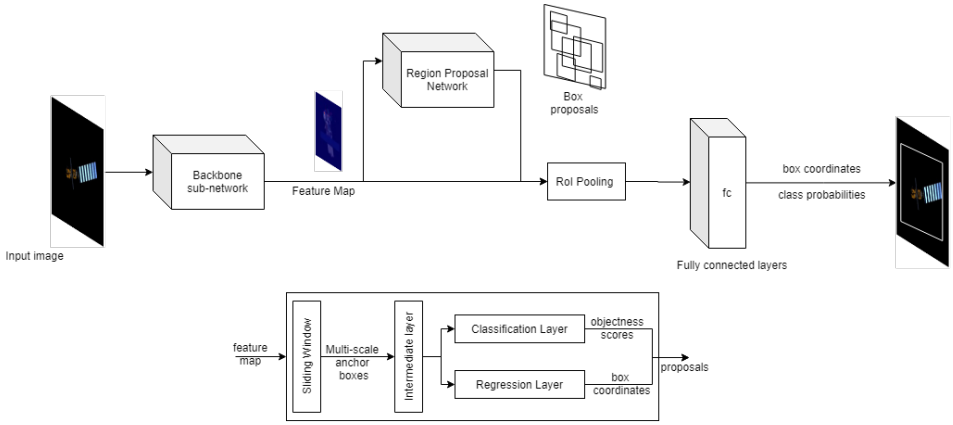


Figure 4.4: Two-stage detector architecture with Faster R-CNN detection head [84]

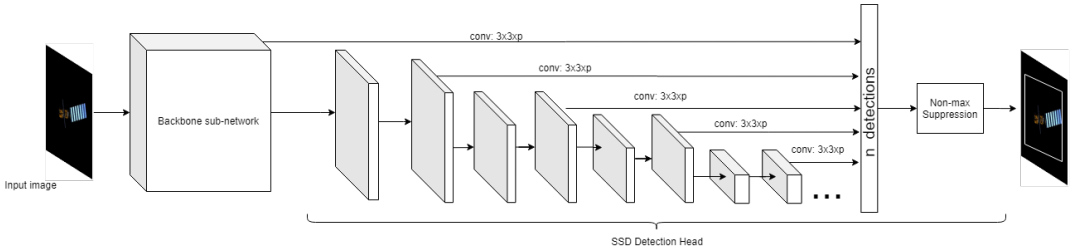


Figure 4.5: One-stage detector architecture with SSD detection head [85]

"in a single shot" as opposed to "per proposal". Currently, the most prominent one-stage object detector families are YOLO[98] and SSD [104]. SSD [1] uses additional auxilliary network of convolutional and fully connected layers, in decreasing size to extract feature maps at various scales. SSD uses anchor boxes similar to the Faster R-CNN detection head to produce default set of boxes at each layer. Fig. 4.5 shows the architecture of a SSD detection network. Detailed design of the convolutional layers and their sizing methodology can be found in the original paper [85]. Another popular type of single shot detection is YOLO (You Only Look Once). YOLO architecture has over times progressed closer to SSD detectors in terms of design and is not discussed in detail here. Choice of either detectors is nearly equivalent as both facilitate significantly faster inference with their optimal network design and sizing. The preference for SSD is justified multi-scale detection from feature maps at various resolution, as shown in Fig. 4.5, which tends to improve its performance [85].

Comparison

In a consistent comparative framework [90] for speed-accuracy trade-off between SSD (one-stage) and Faster R-CNN (two-stage), the following is observed:

1. **Inference Speed** : Meta-architectures with SSD detection heads on average need less computation than Faster R-CNN detection heads.

2. **Accuracy (mean average precision)**: Region proposal approach in Faster R-CNN is generally more accurate than one-shot approach in SSD.

3. **Performance Variation**: Faster R-CNN can be made faster by limiting the number of proposals, and SSD can be made more accurate with more anchor boxes and convolutional layers. However, due to architectural constraints, the respective speed and accuracy sacrifices are significant and the aforementioned observations (1 and 2) remain valid for optimal detector designs.

4. **Backbone Network**: For Faster R-CNN, there is a high correlation between classification accuracy of the backbone (top 1% on Imagenet [66]) and the detection performance. SSD on the other hand is less

[1] Single Shot Detector is both the category of single stage architectures and a specific model architecture. To avoid confusion, acronym SSD is used for the specific CNN architecture and the expanded form is used to refer to the category of one-stage architectures.

Convolution Operation	Parameters	FLOPs	ImageNet Accuracy
Standard convolution	29.3 Mn	4.9 Bn	71.7 %
Depth-wise separable convolutions	4.2 Mn	0.6 Bn	70.6 %

Table 4.2: Comparison of MobileNet network architecture with standard convolution and depth-wise separable convolution on the ImageNet benchmark

reliant on the classification accuracy of the base network. This is considered to be an advantage of convolutional layers with multi-scale aggregation in SSD.

5. **Object Size**: SSD performs very well in detecting large objects in the image, often even outperforming Faster R-CNN with a smaller (and thus faster) backbone sub-network. However, performance of SSD is significantly worse on smaller and dense objects than Faster R-CNN.

6. **Memory Usage**: The use of larger backbone networks adds larger memory requirement. While for the same backbone, SSD requires significantly less memory than Faster R-CNN. This is related to the number of parameters of the different sub-networks in a detector.

Since the computation and memory constraints for execution on-board a spacecraft remain a primary constraint for software models and algorithms, SSD is considered the most appropriate choice for the pose estimation pipeline in this work. For the pose estimation pipeline used in this work, the small inaccuracy in bounding box localization will not significantly reduce the performance of pose estimation. This is because the PnP solver in the pose estimation pipeline needs a minimum number (generally between 4-6) of keypoints or more to estimate the pose. Therefore, if the bounding box excludes some part of the spacecraft, the pose estimation is still expected work as desired, given reasonable accuracy of the KD network.

4.1.2. Backbone

The choice of the backbone is motivated by the desire to reduce the model size and therefore the memory usage and the execution time for object detection. Recall that the backbone sub-network for an object detector is composed of several layers of a standard classification network. Most traditional classifiers like VGG [72] , ResNet [71], Inception [91] use dense convolutions with tens of millions of parameters. Naturally, these models are unsuitable for practical real-life application on an embedded system. For this purpose, Howard et al. [93] proposed efficient mobile networks, called MobileNets that emphasize on lower latency and memory usage at a minor loss of performance. MobileNets factorize the convolution operation into a depth-wise convolution and a point-wise convolution operation on the feature maps, called a depth-wise separable convolution. For the most commonly used convolutional kernel of size 3x3, the depth-wise separable convolution reduces parameters by a factor of 8 or 9. Table 4.2 shows a comparison of MobileNet architecture that uses depth-wise separable convolutions to that of the same network using standard convolutions. The reduction in the number of model parameters (equivalent to memory usage) and floating point operations (equivalent to execution time) in using depth-wise separable convolutions is significant at a very minor performance loss. In the subsequent version MobileNetv2 [105] and MobileNetv3 the architecture is optimized for higher speed and accuracy. Howard et al. [106] design two optimal model architectures for Mobilenetv3: MobileNetv3-Large and MobileNetv3-Small. Table 4.3 shows the comparison various MobileNets with standard CNN architectures on ImageNet classification benchmark. The MobileNet model architectures are significantly faster incurring lower FLOPs per forward pass as well as incur significantly less memory usage due to less number of parameters.

The choice of MobileNets for object detection is unambiguous for an embedded computer. Huang et al. [90] show that MobileNet is consistently faster when used with SSD, in addition to performance results results provided in [93, 94, 105–107] for object detection with various benchmarks. Further, optimal scaling for custom application can be MobileNets are already available for deployment on smartphones an microcontrollers and have been investigated for use with FPGA [108] and RISC [109]. Further, MobileNetv2 has also utilized as a backbone for spacecraft pose estimation pipeline in Park et al. [46].

Given the benefit of depth-wise separable convolutions, Sandler et al. [105] optimize the SSD detection head by using the same depth-wise separable convolutions to decrease the network parameters and FLOPs. The modified detection head, proposed in is called SSDLite. The size and computational requirement of SSDLite are shown in Table 4.4, with comparison to original SSD. MobileNetv3-Large and MobileNetv3-Small

Network Architecture	Parameters	FLOPs	ImageNet accuracy
VGG-16	14.7 Mn	15.3 Bn	71.0 %
Inception V3	21.8 Mn	5.7 Bn	78.0 %
ResNet-50	26 Mn	4.1 Bn	76.0 %
Inception ResNet V2	54.4 Mn	13 Bn	80.4 %
MobilenetV1	4.2 Mn	0.5 Bn	71.1 %
MobileNetV2	3.4 Mn	0.3 Bn	72.0 %
MobileNetV3 Large	5.4Mn	0.2 Bn	75.2 %
MobileNetV3 Small	2.4Mn	0.06 Bn	67.4 %

Table 4.3: Performance comparison of state-of-the-art classification architectures on ImageNet benchmark

Detection Head	**Parameters**	**FLOPs**
SSD	14.8 Mn	1.25 Bn
SSDLite	2.1 Mn	0.35 Bn

Table 4.4: Size and computation reduction of SSD detection head with depth-wise separable convolutions

are selected as alternatives for the pose estimation pipeline. The choice of two network is motivated to perform a comparative analysis on the effect of size of the OD network for spacecraft pose estimation. The selected object detection networks and their computational properties are summarized in 4.5.

Configuration	**Backbone**	**Detection Head**	**Parameters**	**FLOPs**
1	MobileNetV3-Large	SSDLite	3.2Mn	0.5 Bn %
2	MobileNetV3-Small	SSDLite	2.4Mn	0.16 Bn %

Table 4.5: Summary of network size and computation requirement for the chosen object detectors

4.2. Configuration

Given a model architecture for object detection, it is necessary to configure the various higher level elements of the network that specify and tune the training and inference process for effective performance. The configuration closely follows the work in [85] with modest differences. The configuration is specified with a `.config` file [2] that can be interpreted by Tensorflow, discussed in Section 4.3.

4.2.1. Box Encoding

A bounding box in an image can intuitively be represented by 4 parameters. It is common to use one of - centroid (x_c, y_c, w_{box}, h_{box}), corner (x_{min}, y_{min}, x_{max}, y_{max}) or minmax (x_{min}, x_{max}, y_{min}, y_{max}) representations in accordance with parameters visualized in Fig. 4.7. While these representations are easy to interpret, the modern box detectors like SSD do not use such representations. Remember that the SSD/SSDLite network uses anchor boxes of various aspect ratios at each feature map cell to predict bounding boxes. The anchor boxes are a set of default boxes that are used at each feature map cell in a layer. During Training, a ground truth box is provided to the network and matched with a default box at a certain feature map location, scale and aspect ratio, based on a similarity metric (discussed below). Fig. 4.7 visualizes anchor boxes on a 8x8 map grid for an example image. The green anchor boxes indicate positive matches while red ones indicate negative matches. This is then used to learn the weights of convolution filters that predict the offsets (center, height and width) of the fixed anchor boxes at each feature map cell from the ground truth box. Consequently, these offsets are used to optimize the regression loss function. In order to facilitate this, it is helpful to encode the bounding box information with a representation that ensures stability and efficiency during training.

To this extent the Faster R-CNN box encoding, originally proposed in [102] and used in [84, 85], is utilized. Consider a centroid representation of the ground truth bounding box (x, y, w, h), with subscripts removed for

[2] A sample configuration file is provided in Appendix C

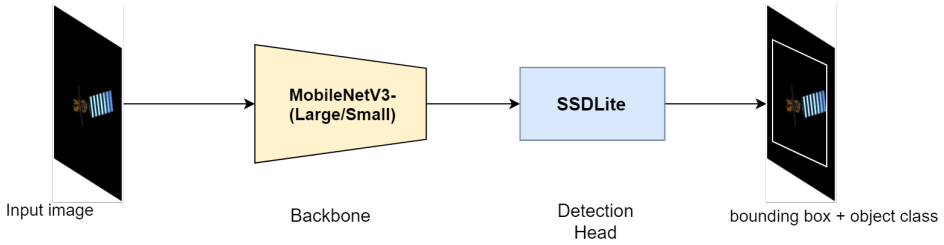


Figure 4.6: Object Detector: Overall Architecture

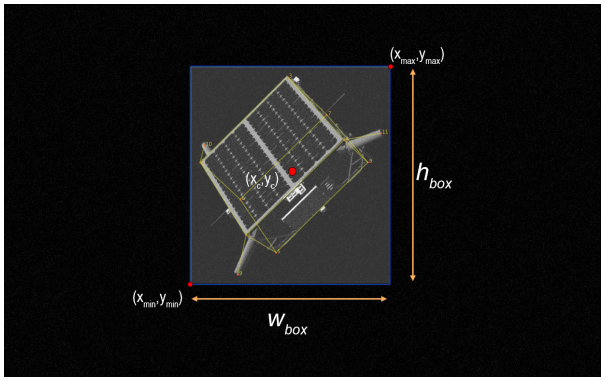


Figure 4.7: Visualization of bounding box around the (Tango) spacecraft and the representation parameters

clarity, and of an anchor box (x_a, y_a, w_a, h_a). The 4-vector encoded box representation, $\mathbf{t}_{box} = [t_x, t_y, t_w, t_h]$, is given as following:

$$\begin{aligned} t_x &= \frac{x - x_a}{w_a} \\ t_y &= \frac{y - x_a}{h_a} \\ t_w &= log\left(\frac{w}{w_a}\right) \\ t_h &= log\left(\frac{h}{h_a}\right) \end{aligned} \tag{4.1}$$

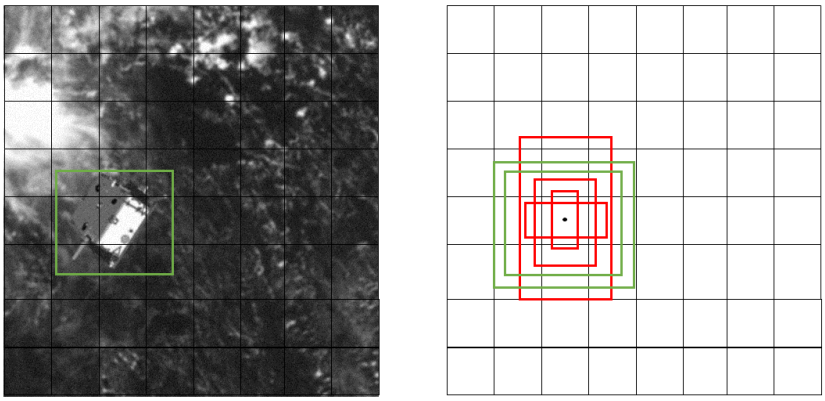


Figure 4.8: Visualization of arbitrary anchor boxes at a specific cell (right) and matching with the ground truth box (left) in a 8x8 feature map

4.2.2. Anchor boxes generation

In order to allow detection of the spacecraft effectively over larger range of separation, it is essential to have anchor boxes that can detect at respective sizes accurately in the feature maps. To that effect, the anchor boxes can be configured using the scale and aspect ratio parameters. The default boxes are used in 6 layers, i.e. over feature maps of 6 different scales between 0.2 and 0.95 times the size of the original image. For a each of the feature map cells at those scales, 5 boxes are generated of aspect ratios (1, 2, 3, 1/2, 1/3). Each aspect ratio relates to the size of the feature cell, which in turn relates to the original image, depending on the scale of the layer. The network learns to predict a 5-vector comprising of 4 localization parameters (Eq. 4.1) and an additional classification parameter (p_c), the class probability of the object inside the box. For instance, the object detection on SPEED and/or Envisat datasets learn to predict the probability of the pixel array inside the bounding box containing the Tango and Envisat spacecraft respectively.

4.2.3. Similarity and Matching

Given a encoded ground truth box ($\mathbf{t}_{box}$) and a predicted box ($\mathbf{t}^*_{box}$), the similarity is calculated using the Intersection-over-Union (IoU) metric. For an area of the ground truth box (A) and that of the predicted box (A^*), the IoU is given as:

$$\text{IoU} = \frac{A \cap A^*}{A \cup A^*} \tag{4.2}$$

The matching of the ground truth box to an anchor box is required to learn the weights of kernels that predict anchor box offsets. This is done using argmax matcher, where a detection is flagged positive if the similarity is greater than a certain threshold and negative otherwise. The similarity threshold for the matcher is set at 0.5 IoU.

4.2.4. Loss Functions

For the localization loss that penalizes the difference between predicted and true bounding box coordinates, a weighted smooth-L1 function is used which is more robust to outliers than the L2 loss [103]. The smooth-L1 (s$L1$) function is given as

$$\text{s}L1\,(x) = \begin{cases} 0.5\,x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{if } |x| \geq 1 \end{cases} \tag{4.3}$$

The localization loss based on smooth-L1 function is then defined as following:

$$L_{loc} = \sum_i \text{s}L1\,||\mathbf{t}_i - \mathbf{t}^*_i|| \tag{4.4}$$

where, $\mathbf{t}_i$ and $\mathbf{t}^*_i$ represent the 4-vector encoded bounding box representation of ground truth and predicted boxes respectively and i represents a box evaluation per anchor box. Other localization loss functions like L2 and weighted IoU losses were sensitive to diverge without excessive hyperparameter tuning.

For the classification loss that penalizes the wrong class detection for the object inside a bounding box, a sigmoid focal loss function [102] is used. Generally, the classification loss is taken into account, considering the more general use of object detector to localize and recognise specific parts of the spacecraft as shown in Fig. 4.2. However, in case of a single object, the classification is done between the object (spacecraft) and the background i.e. whether the part of the image inside contains the spacecraft or the background. The sigmoid focal loss is chosen as demonstrates significant improvement for one-stage detection. The weighted sigmoid focal loss is given in terms of a tunable focusing parameter (γ), the balance factor (α) and the class probabilities predicted by the detection head classifier layers (p_{ci}) as:

$$L_{cls} = \sum_i -\alpha\,(1 - p^*_{ci})^\gamma\,\log(p^*_{ci}) \tag{4.5}$$

where, i is the evaluation index per anchor box evaluation and p^* is defined as:

$$p^* = \begin{cases} p & \text{if true class} \\ 1 - p & \text{if false class} \end{cases} \tag{4.6}$$

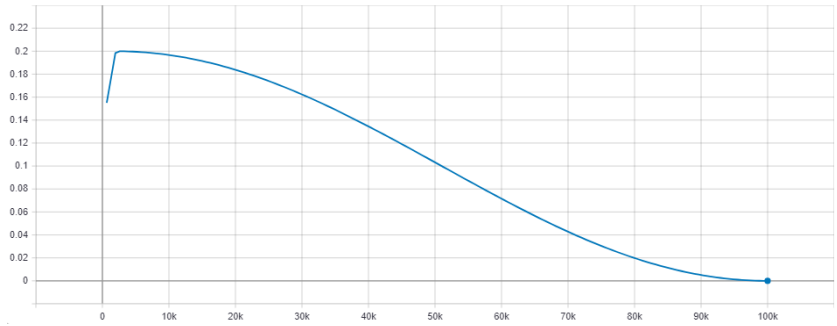


Figure 4.9: Cosine decay learning rate variation with warm-up

While α and γ are tunable parameters, an investigation on these parameters was considered out of scope and working values used on COCO dataset were adapted with $\alpha = 0.75$ and $\gamma = 0.2$.

The total loss (L) is computed as the weighted sum of classification and localization loss:

$$L = \frac{1}{N}(L_{loc} + w_{c/l} L_{cls}) \tag{4.7}$$

where, N is the number of matched anchor boxes and $w_{c/l}$ is the ratio of weights of classification loss to the localization loss. As a default, the weight ratio, $w_{c/l}$ is set to 1.

4.2.5. Training and Evaluation

The training is done with mini-batches of 64 images each from the training set. The weights or parameters of the network are updated using mini-batch gradient descent with a momentum term. The momentum term is a proportional term added to the update along the gradient, expressed in a general form in Eq. 4.8 for an update of weights ($\mathbf{w}$) of the network at a step k, to minimize the parameterized loss function (L). The update of weights corresponding to a mini-batch is called a step (or global step). The momentum term adds dependence of the weights update at the $(k-1)^{th}$ step on the weights update at the k^{th} step, scaled with a momentum parameter (p). Adding a momentum term has been shown to reduce the oscillations and achieve faster convergence in stochastic gradient descent in neural networks [110, 111].

$$\delta \mathbf{w}_k = -\alpha \nabla_w L(\mathbf{w}) + p\,\delta \mathbf{w}_{k-1} \tag{4.8}$$

Recall from Chapter 2, that α in Eq. 4.8 is the learning rate. While learning rate can be set to a constant value, the training is more effective with a variable learning rate that reduces over time. This facilitates better convergence by allowing larger weight updates at the beginning of the training and finer adjustments as it progresses further. A cosine decay is applied to the learning rate with a gradual linear warmup. The learning rate warmup is effective method to maintain stability of training during the initial steps when large weight changes are made. Proposed by Goyal et al. [112], the warmup uses a linear increase to model the initial learning steps until the peak after which, the learning rate follows a cosine decay. Fig. 4.9 shows a representative cosine decay of learning rate with 2000 step warm-up from 0.133 to the base learning rate of 0.2, followed by the cosine decay until 100,000 steps. A base learning rate of 1e-3 is found to be a good starting point for most training routines.

4.2.6. Data Augmentation

As highlighted in Section 2.4, data augmentation techniques are crucial to improve robustness of CNNs during learning. A set of data augmentation techniques is embedded in the training that is aimed at improving the generalization of the model to scenarios not seen in the original training set. For object detection network, the data augmentations included in the training are listed in Table 5.2. The data is classified into affine augmentations (**DA-0**) and pixel-level augmentations (**DA-1**). In addition to brightness and contrast variation, **DA-1** group also includes a random erase augmentation proposed by Zhong et al. [113] that improves box detection by enabling the network to identify an object, that may be occluded.

Type	Data augmentation	Value	Probability	Description
DA-0	Random scaling	0.5	0.6	Randomly scale the images with a random scale factor between [-0.5, 0.5].
	Random rotation (deg)	90	0.6	Randomly rotate the images by a random angle between [-90, 90] degrees in the image plane.
	Random flip	True	-	Randomly flip the images left-to-right.
DA-1	Random brightness	0.5	0.9	Randomly change the brightness by a factor between [-0.5, 0.5].
	Random contrast	0.5	0.9	Randomly change the brightness by a factor between [-0.5, 0.5].
	Random Erase [113]	True	0.3	Randomly erase (replace with black pixels) part of the images.

Table 4.6: Training data augmentations. **DA-0**: affine Augmentation, **DA-1**: pixel-level augmentations

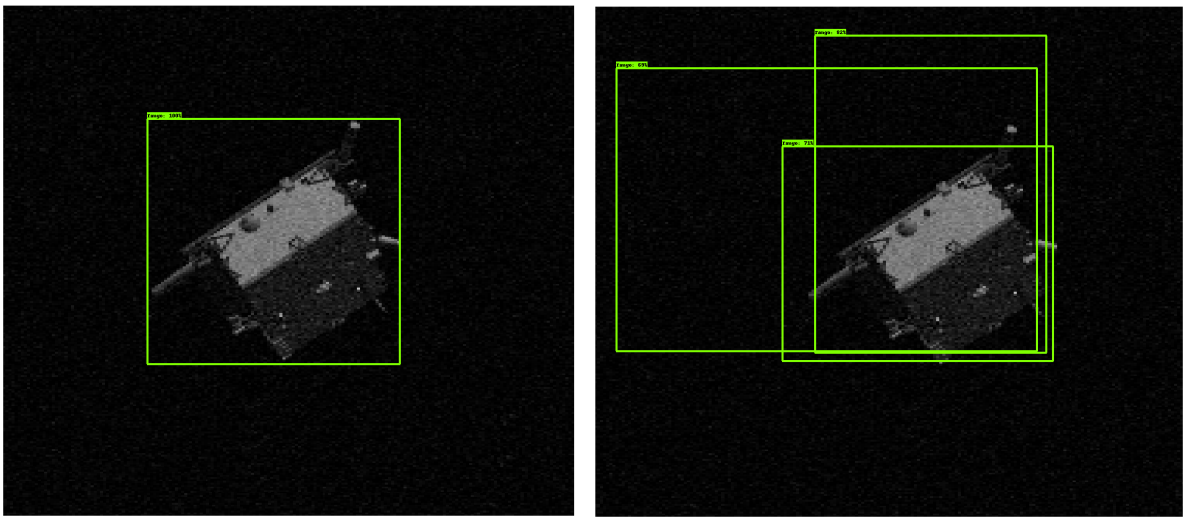


Figure 4.10: Ground truth box (left) and boxes inferred by the network (right) on a sample *SPEED* image of the Tango spacecraft

4.2.7. Inference

Once a trained model is ready, it can be put to the task of detecting the spacecraft in an image, that the network might not have seen before. In the absence of information about the ground truth boxes, the network needs to find a single bounding box per object. However, the network can produce several boxes that contain the spacecraft similar to the mis-detections visualized in Fig. 4.10. Since the network cannot use ground truth reference, it must be able to find one box per object. This is done using the Non-Maximum Suppression (NMS) algorithm. The NMS algorithm filters boxes of the same class (e.g. spacecraft) such that all the boxes that have a high IoU are assumed to be detecting the same object multiple times. Therefore, the box corresponding to the highest class probability and all the boxes that have IoUs greater than a threshold value are suppressed, ideally resulting in a single box detection per object. The IoU threshold for non-max suppression is fixed at 0.6.

4.2.8. Evaluation Metrics

To assess the accuracy of bounding box regression on an image, the IoU metric is used to indicate the similarity between the true bounding box and the predicted bounding box. The IoU is given by Eq. 4.2 and visualized as shown in Fig. 4.11. For multiple images, the statistical assessment of the performance is done with mean and median of the IoU over all the images.

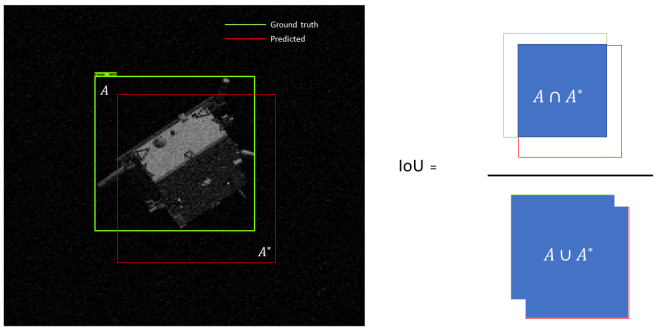


Figure 4.11: Visualization of IoU metric

4.2.9. Miscellaneous

Apart from the aforementioned elements and hyperparameters, there are several others that make up the full configuration of the detection network. These include network activation function, convolutional kernel sizes, regularization, weight initialization and batch normalization. However, these are critical parameters that relate to the sub-network architectures (SSDLite and MobileNetv3). Therefore, the optimal design values of those parameters are directly taken from [106].

4.3. Implementation

The object detection network is implemented in Tensorflow [114], an open-source machine learning platform built and maintained by Google Brain. Specifically, the Tensorflow object detection API [3] released with [90], is used. Tensorflow object detection API facilitates a consistent implementation and quick realization of object detection models in Python. Moreover, the models built and trained with Tensorflow APIs interface rapidly with multiple tools and across variety of hardware devices including most CPU, GPU and TPU architectures on phones and computers as well as microcontrollers and other embedded devices.

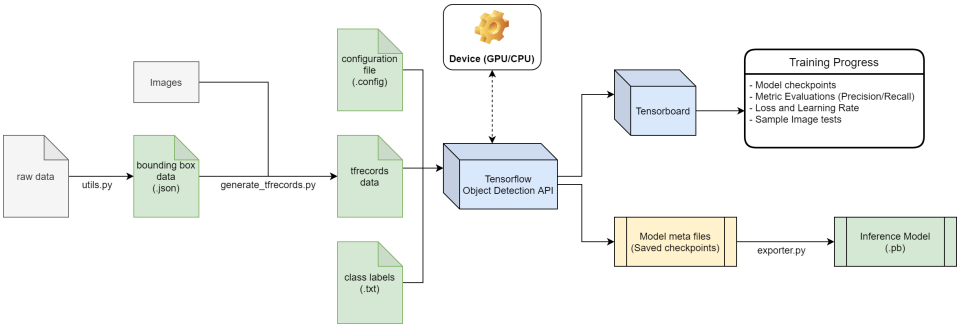
Fig. 4.12 shows the implementation schematic with the data pipeline established to carry out the training and evaluation. The network training is done on Nvidia Tesla P100-PCIE-16GB GPU made available through IPython notebook interface in Google Colaboratory (Colab) [4]. The data is synced between the local computer and the Colab server through Google Drive connection. A successful training results in a trained model along with meta-data and training logs. The trained model can then be used/deployed across machines with Tensorflow support to execute inference and batch evaluations. For this purpose, a Tensorflow environment is also maintained in a python environment on the local machine which is used with Nvidia P1000-Quadro-8GB GPU and Intel core i7-8750H CPU. The trained model is deployed on the local machine for small-scale inference, testing and benchmarking.

The modified code-base with the necessary files that adapts the interface to the Tensorflow Object Detection API for spacecraft pose estimation research in this work, are made available [5] by the author.

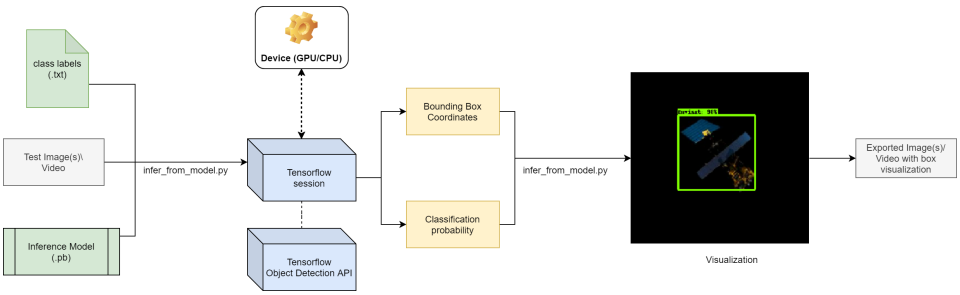
4.4. Conclusions

The selection, design review and configuration of an OD network has been presented. After rigorous comparison, the network architecture with MobileNetv3 backbone and SSDLite detection head is chosen. Two candidate networks are selected for evaluation in the pose estimation pipeline. Further, the configuration and implementation of network training, information encoding, data augmentations and evaluation metrics used in developing the object detection models are summarized. The selected OD networks: *SSDLite-MNetv3-large* and *SSDLite-MNetv3-small* are taken forward for the navigation framework testing and evaluation.

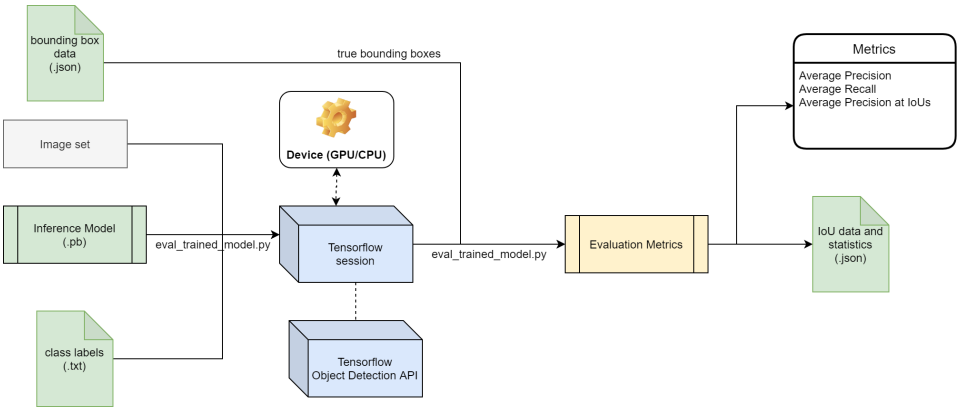
[3] https://github.com/tensorflow/models/tree/master/research/object_detection
[4] https://colab.research.google.com
[5] https://github.com/kuldeepbrd1/models



(a) Training



(b) Inference



(c) Batch Testing

Figure 4.12: OD implementation schematic and data pipeline/ (gray: third-party resource, blue: third-party software, yellow: intermediate implementation, green: processed/developed resources

5

Keypoint Detection Network

Keypoint detection is a computer vision task of localizing and identifying the points of interest or 'keypoints' in an image. Traditional keypoint detectors use descriptor-based image processing algorithms that lack robustness to variation of operational conditions as noted in Section 2.3.1. Recently, CNNs have become a popular alternative for challenging KD tasks like Human Pose Estimation (HPE) that have driven innovation in state-of-the-art KD methods. HPE concerns the detection of essential points (knees, elbows, shoulders, etc) on a generic human body from an RGB image. For HPE, the human pose is a representation of relative positions of the body keypoints (joints) with a simplistic skeleton (links) that finds its application in activity recognition, robot motion learning, sports injury prediction, and automated sign-language translation among others. The success of CNN models stems from their ability to generalize the detection of keypoints on humans of varying sizes and in presence of variations in color and other artefacts. This chapter motivates the adaptation of a state-of-the-art deep learning model and the underlying methods used in HPE, to tackle the problem of keypoints detection for robust spacecraft pose estimation.

The problem of KD in the selected pose estimation architecture is to predict the coordinates of k predefined keypoints on the spacecraft surface from a 2D image of size $w_k \times h_k$. In contrast, the pose in HPE problem refers to the relative positions of joints and links for a non-rigid human body with links that can move relative to each other. Despite the difference in the abstraction of pose and the rigidity, both problems boil down to detecting pre-defined keypoints. Figure 5.1 visualizes detected keypoints in humans and spacecraft use-cases. A quality of modern KD networks is that they also learn to identify keypoints that are not explicitly visible, by learning inherent spatial representation from the provided training examples [115]. This can be noticed in the occluded joints detected in Fig. 5.1. To estimate keypoint coordinates, the state-of-the-art KD networks use a confidence map representation rather than a direct keypoint coordinate vector. A confidence map provides pixel-wise pseudo-likelihood of the keypoint location being in that pixel. Such a map is also called a heatmap. An ideal heatmap is a distribution (e.g. Gaussian) around the true keypoint

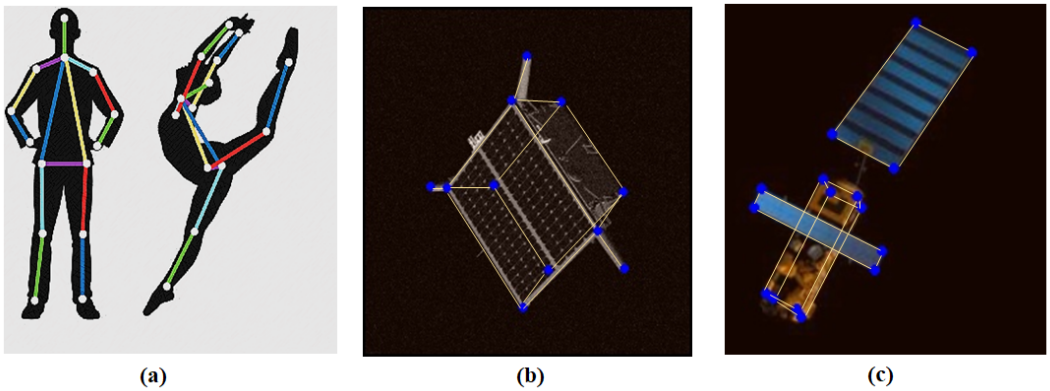


Figure 5.1: Visualization of Keypoint and Skeleton representation for (a) Humans (b) Tango spacecraft (c) Envisat spacecraft

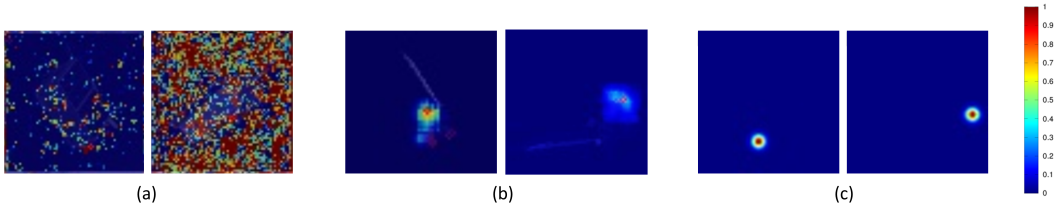
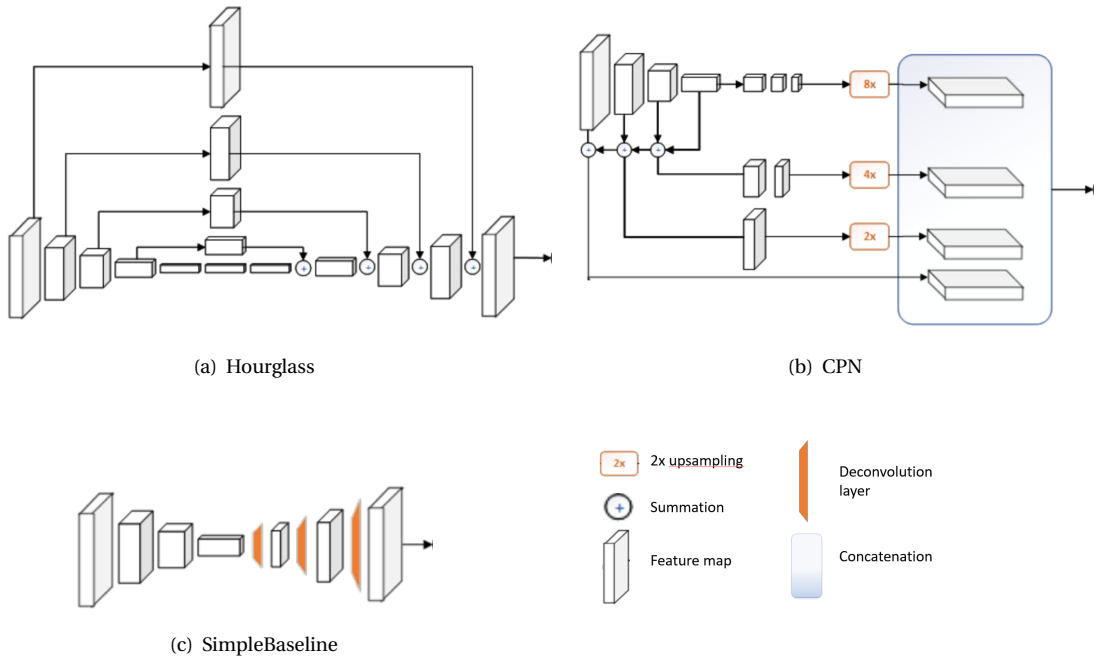


Figure 5.2: Visualization of keypoint prediction heatmaps (a) bad detections (b) imprecise detections (b) precise detections



(a) Hourglass

(b) CPN

(c) SimpleBaseline

Figure 5.3: Network architecture for hourglass like KD network architectures [121]

location. The task of the network is to effectively regress the per-pixel confidence value from the the input image, for each of the pre-defined keypoints. Fig. 5.2 shows examples of heatmap outputs from the KD network. The heatmap representation [116] with spatial likelihood information adds keypoint position ambiguity that improves generalization of keypoint predictions. Szegedy et al. [117] show that this spatial ambiguity in the heatmap representation improves generalization and adaptability of the CNN models, in contrast to the direct regression of keypoint coordinates. Further, using a heatmap representation means that the CNN learns an image-to-image mapping that is tied by location, which increases explainability of the CNN models by revealing where the CNN focuses on in an image. The keypoint coordinates can be extracted from the heatmap with simple decoding techniques [118]. To quantify uncertainty from heatmaps of the detections, the technique from [57] can be used.

5.1. Network Architecture Selection

Unlike object detection network, KD networks are not aggregated into commonly accepted architectures and components. The progress in KD (a.k.a human pose estimation) has undergone constant improvement and several architectures [101, 118–122] have been proposed with widely varying building blocks.

Hourglass [118], an early state-of-the-art architecture uses repeated down and up sampling in stages for KD. It uses pooling layers to down sample and skip-connections between up and down sampling halves to aggregate multi-scale feature maps. This is shown in Fig. 5.3. The hourglass architecture suffers from a number of limitations, that have been noted in the subsequent works and improved upon. The original stacked hourglass models (8-staged) is determined to be redundant and computationally inefficient for the corre-

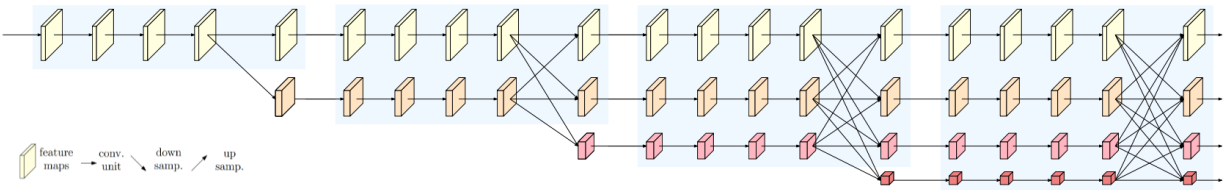


Figure 5.4: HRNet main body architecture with parallel 4 resolution branches and multi-resolution fusion across branches of feature maps

sponding performance [123], resulting in networks that have high computational (FLOPs) demand per forward pass. Further, Hourglass single-stage module is not optimally designed according to the established design practices for modern CNNs, and stacking above two stages results in significantly low performance gain [124]. Further, Models based on stacked (multi-stage) hourglass architecture are drastically outperformed by single-stage models on the more challenging COCO [86] benchmarking dataset. Further, the use of pooling layers in the architecture and repeated up and down sampling results in loss of information that limits the architecture's performance capability. Cascaded Pyramid Network (CPN) [119], the subsequent state-of-the-art method uses a similar U-shape high-to-low and low-to-high sampling, but is inspired by the best design practices from Feature Pyramid Network (FPN) [95]. CPN architecture is composed of a ResNet [71] inspired sub-network that downsamples the feature maps, followed by a refinement sub-network with bottleneck layers that creates a parallel upsampling to recover a pyramid of feature maps. The feature maps in the pyramid are concatenated to compute the final heatmap output. With these components, CPN adapts best practices to improve the efficiency and performance of the hourglass-like network structure for pose estimation. More recently, Xiao et al. [121] propose the use of deconvolution (transposed convolution [125]) layers and investigate a baseline design for effective pose estimation architectures. Following the baseline, the designed networks with ResNet backbone improve over both Hourglass and CPN. Despite the performance and computational efficiency, the network sizes are significantly high. The symbolic visualization of all the architectures is shown in Fig. 5.3.

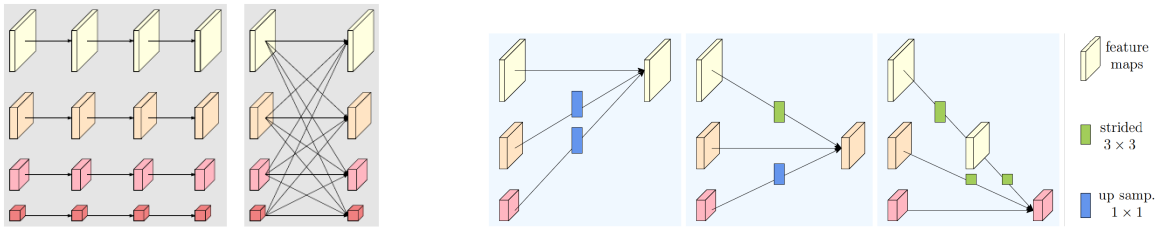
The main challenge associated with KD problem using heatmaps, is the need to recover the spatial resolution required for the output heatmap. Since most CNNs consistently reduce the resolution of the feature maps to infer patterns in the image at the lower levels, the resolution recovery is a crucial part of the KD performance. So far, the architectures discussed here decrease the resolution and subsequently recover resolution through upsampling. Sun et al. [122, 126] propose a radically new architecture that emphasizes on maintaining high resolution through out the network. The proposed High Resolution Network (HRNet) architecture remains[1] the state-of-the-art performance on COCO [86]. HRNet uses parallel sub-networks across multiple resolutions rather than commonly used multi-resolution serial stages, as shown in 5.4. Further, it uses persistent multi-scale fusion within the network to improve the high resolution maps from the lower resolution maps. This maintains a rich feature representation and improves the heatmap precision facilitating a more accurate KD. For a consistent comparison with other architectures, the results of COCO validation sets for a constant input size of 256 x 192, are reported in Table 5.1. Refer to Lin et al. [86] for the description of the average precision metric used for benchmarking. Notice that Hourglass is highly inefficient in computation with the highest FLOPs requirement per forward pass, while having the worst performance. CPN is efficient in terms of computation with comparable number of parameters, but remains only marginally better in performance than 8-stage Hourglass. SimpleBaselines has a high computational efficiency and performance but has a larger model size, therefore higher memory requirement. HRNet-W32, the smaller network with comparable parameters as CPN and Hourglass shows a large margin of performance gain while being computationally efficient in terms of FLOPs. As a result of its performance efficiency, HRNet architecture is expected to be more suitable when scaled down for embedded devices that have lower computational capabilities. Given the design and performance superiority of HRNet, it is selected as the KD network in this work.

Recall that the base networks in object detection were radically improved (parameters and FLOPs) after replacing the standard convolution layers with depth-wise separable convolutions, residual bottleneck layers and squeeze and excitation layers. The scope of optimizing the HRNet architecture with these elements further reinforces the potential of HRNet, as the parameter and FLOPs counts are expected to reduce signifi-

[1]The current state-of-the-art on COCO is DarkPose [127] that uses HRNet-W48 for detection with modifications being only in the keypoint decoding step.

Network Architecture	Parameters (Mn)	FLOPs (Bn)	Average Precision
Hourglass (8-stage)	25.1 Mn	19.5 Bn	67.1
CPN (ResNet-50)	27.0 Mn	6.2 Bn	68.6
SimpleBaseline (ResNet-50)	34 Mn	8.9 Bn	70.4
SimpleBaseline (ResNet-101)	53 Mn	12.4 Bn	71.4
HRNet-W32	28.5 Mn	7.1 Bn	74.4
HRNet-W48	63.6 Mn	32.9 Bn	76.3

Table 5.1: Performance comparison of state-of-the-art KD networks on COCO benchmark (validation set with image size: 256 x 192)



(a) (b)

Figure 5.5: HRNet Visualization: (a) blocks with parallel convolution (residual) units and multi-resolution fusion (b)representation of exchange units used for fusing multiple resolutions[126]

cantly. Since HRNet and most other KD architectures are very recent, this aspect of deployment optimization for mobile devices hasn't been explored yet.

5.1.1. HRNet

HRNet architecture features sub-networks aligned parallel to each other with respective resolution streams. Each parallel network operates on feature maps of the same resolution and feature maps are repeatedly fused with higher and lower resolution streams between the parallel subnetworks. These elements allow HRNet to maintain high spatial accuracy of the feature maps through out the network and result in heatmaps that allow a more accurate KD. The visualization of HRNet architecture is shown in Fig. 5.4.

The architecture is composed of sub-networks, both parallel and serial. A group of parallel sub-networks across resolution streams is called a stage and the group of serial sub-networks along a resolution stream, a branch. A subnetwork at s^{th} stage and processing feature maps at r^{th} resolution branch can be denoted by $\mathcal{N}_{sr}$. Then, the network structure of HRNet is constituted by adding one lower resolution sub-network, on the subsequent stage. For example, a 4-stage 4-resolution network structure is shown below. Note that Fig. 5.4 shows a similar representative structure, but it is visualized with feature maps.

$$\begin{array}{ccccccc} \mathcal{N}_{11} & \rightarrow & \mathcal{N}_{21} & \rightarrow & \mathcal{N}_{31} & \rightarrow & \mathcal{N}_{41} \\ & \hookrightarrow & \mathcal{N}_{22} & \rightarrow & \mathcal{N}_{32} & \rightarrow & \mathcal{N}_{42} \\ & & & \hookrightarrow & \mathcal{N}_{33} & \rightarrow & \mathcal{N}_{43} \\ & & & & & \hookrightarrow & \mathcal{N}_{44} \end{array}$$

Each stage consists of blocks (shaded blue in Fig. 5.4, each of which contain a group of parallel convolutions called residual units, and a multi-resolution fusion unit as shown in 5.5. Each parallel residual unit is composed of two sets, each comprising of a 3x3 convolution layer, a batch normalization layer and an activation (ReLU) layer. The multi resolution fusion is done by using: (a) a 3x3 convolution (+Batch Normalization) for the feature map at the same resolution, (b) a 2-strided 3x3 convolution (+Batch Normalization) for down-sampling the feature map at the higher resolution, and (c) a 1x1 convolution (+Batch Normalization) followed by a nearest-neighbour upsampling. Further, in each downward branch, the resolution is halved and the channels (convolution kernels) are doubled. Finally, the output heatmaps are the obtained from the highest resolution branch of the last stage at the end of the last block, using a final convolution layer.

The elements discussed above make up the main body, which processes the inputs and produces output maps of the same size. This size of the feature maps along the network is fixed to 1/4th of the input image for

optimal cost-performance. A small stem subnetwork is added at the beginning, which pre-process the image and feeds the feature maps of compatible size to the HRNet main body.

Sun et al. [122] use this architecture to construct two networks HRNet-W32 and HRNet-W48, where numbers denote the channels in the first stage. As a baseline, HRNet-W32 is used in this work, given it's smaller size and effective performance on SPEED dataset in [78]. It consists of a first stage with 4 ResNet-like residual bottlenecks, followed by second stage with one block, the third stage with 4 blocks and the fourth stage with 3 blocks. HRNet-W32 uses 32,64,128 and 256 channels from highest to lowest resolution parallel sub-networks. In contrast, HRNet-W48 uses 48, 96, 192 and 384 channels. While HRNet-W32 is selected as the baseline network, the parameter count and FLOPs are considered fairly high for an embedded deployment onboard a spacecraft. Therefore, new downsized networks HRNet-small and HRNet-smaller are introduced in this work for comparison. This is discussed further in the following section.

5.2. Configuration

Given the model architecture, the configuration needed to specify and tune the aspects of training and inference process is discussed. The configration closely follows the the one from the pipeline in Sun et al. [122] applied to the COCO dataset and some elements from Chen et al. [78] applied to the SPEED dataset. The configuration is specified with a `.yaml` file [2] that can be interpreted by PyTorch, the library in which HRNet architecture is available from [128].

5.2.1. Input, Output and Target Generation

The input to the KD network is the predicted ROI in the original image, cropped and resized to a fixed input size of the network. Since, input image size affects the computation requirement (FLOPs), the input size of the network is specified as 256 x 256, as a balance between performance and computation requirement. When the RoI is cropped and resized to the input size, the scaling parameters (center and scale) are preserved in a data structure outside the network.

For a network detecting k keypoints for an image input of size $w \times h$, the output of the network is k heatmaps of size $\frac{w}{4} \times \frac{h}{4}$. For the input size specified above, the output heatmaps are of size 64 x 64. This is a common practice adapted in KD [118, 119, 122, 126] for computational efficiency. The keypoint coordinates on the input image are predicted from the corresponding heatmap and re-projected back to the original image using the scaling parameters preserved during input generation.

As discussed earlier, the network uses heatmap representation to express detection. Consequently during training, the true keypoint coordinates cannot be provided directly to the network as a training example. Instead, the true keypoint coordinates are encoded into k target heatmaps with a Gaussian distribution of confidence around the true keypoint location. In this work, the target maps are generated with a standard deviation of 2 pixels.

5.2.2. Loss Function

During training, the network uses the target heatmaps and the predicted heatmaps to compute the loss. For the heatmap representation, a simple mean squared error (MSE) is generally used [117, 122] for per-pixel confidence values between the predicted and the target heatmap. Given k target heatmaps and the corresponding heatmaps predicted by the network, the loss function is given as:

$$L = \frac{1}{k}\sum_{h} L_h \quad ; \quad L_h = \frac{1}{m\,n}\sum_{i,j}\left(c_{ij} - c'_{ij}\right)^2 + \tag{5.1}$$

where, c_{ij} is the target confidence and c'_{ij} is the predicted confidence for a pixel at location (i, j) in the heatmap of size $m \times n$.

5.2.3. Training and Evaluation

The training is done with mini-batches of 32 images each from the training set. Evaluation is done on the validation set after each training epoch. The training epoch signifies one complete training iteration of all

[2] A sample configuration file is provided in Appendix C

Type	Data augmentation	Value	Probability	Description
DA-0	Random scaling	0.5	0.6	Randomly scale the images with a random scale factor between [-0.5, 0.5].
	Random rotation (deg)	90	0.6	Randomly rotate the images by a random angle between [-90, 90] degrees in the image plane.
	Random flip	True	0.5	Randomly flip the images left-to-right.
DA-1	Random brightness	0.5	0.9	Randomly change the brightness by a factor between [-0.5, 0.5].
	Random contrast	0.5	0.9	Randomly change the brightness by a factor between [-0.5, 0.5].

Table 5.2: Training data augmentations. **DA-0**: affine Augmentation, **DA-1**: pixel-level augmentations

the mini-batches. For every batch, the weights or parameters of the network are updated using the Adam optimizer [129]. Adam is an adaptive optimizer and a common choice of optimizer for large and deep network due to its computational and memory efficiency over other stochastic gradient descent methods. Adam associates first-order gradients of the loss function to adaptive moment vectors that scale the learning rate to individual weight parameter. The update rule for Adam algorithm can be written as:

$$\delta \mathbf{w}_k = -\alpha \frac{\hat{\mathbf{m}}_k}{\sqrt{\hat{\mathbf{v}}_k} + \epsilon} \tag{5.2}$$

, where α is the learning rate as usual. $\hat{\mathbf{m}}_k$ and $\hat{\mathbf{v}}_k$ are two bias corrected moment vectors that adapt the gradients $\mathbf{g}_k$ for the weight update, subject to constant hyper parameters β_1, β_2 and ϵ. The moments are defined as the following.

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1-\beta_1^k}; \qquad \mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\mathbf{g}_k$$
$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1-\beta_2^k}; \qquad \mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1-\beta_2)\mathbf{g}_k^2$$
$$\mathbf{g}_k = \nabla_{\mathbf{w}} L(\mathbf{w})$$

The values of hyperparameters are taken from [129]: $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 1e-8$. The learning rate (α) is The base learning rate is set to $\alpha = 1e-3$ and is scheduled to step down by a factor of 10 using a multi-step schedule at 100 and 150 epochs, as suggested in [122]. The network is trained upto 180 epochs unless the validation loss is found to saturate.

Generally, transfer learning [130] is used to initialize training with parameter weights from a network pretrained on the benchmark ImageNet [66] dataset. While utilization of transfer learning is helpful and has shown to provide a minor performance boost to HRNets [83, 122], pretrained models are available only for the standard HRNet-W32 network. For the custom down-scaled networks introduced in Section 5.2.5, pretraining on ImageNet is not in the scope of this work due to its intensive resource demand and less relevance to the research objectives. For HRNet-W32, the experiments in Section 9.2 are marked appropriately to reflect if a pre-trained network is used.

5.2.4. Data Augmentation

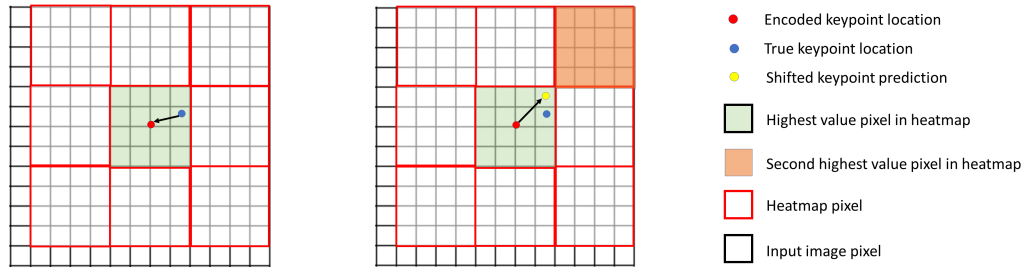
These augmentations are applied during training time and are applied randomly, similar to object detection, described in Section 4.2. The data augmentations used during training are listed in Table 5.2. For affine augmentations, the true keypoint coordinates for the training are simultaneously transformed using the same transformation during training time. The target heatmaps are then generated from transformed keypoint coordinates and fed to the network for training.

5.2.5. Network Scaling

As discussed earlier, the HRNet-W32 network is selected as the baseline network for this work. In addition, two scaled down versions of HRNet architecture are introduced. These are called HRNet-small and HRNet-

Network	Branches	Blocks per stage	Residual units per block	Channels	Parameters	FLOPs
HRNet-W32	4	[-, 1, 4, 3]	4	[32, 64, 128, 256]	28.5 Mn	9.5 Bn
HRNet-small	4	[-, 1, 1, 1]	2	[32, 64, 128, 256]	5.2 Mn	3.0 Bn
HRNet-smaller	4	[-, 1, 1, 1]	2	[16, 32, 64, 128]	1.6 Mn	1.8 Bn

Table 5.3: Comparison of HRNet-W32 network with scaled down versions HRNet-small and HRNet-smaller, introduced in this work



(a) Coordinates to heatmap encoding (b) Heatmap to coordinates decoding

Figure 5.6: Visualization of quantization effect in a heatmap representation with one-fourth input resolution. (a) shows the keypoint shift in the target heatmap as a result of encoding and (b) shows the sub-pixel shift used to compensate the keypoint prediction from the output heatmap

smaller, to avoid any more non-trivial acronyms. HRNet small is constructed by modifying HRNet to only have 1 residual units 2 blocks per stage. Further, HRNet smaller is conceived by halving the number of channels (convolution kernels) in each stage of HRNet-small. The scaled down versions are used evaluate their performance and robustness on spacecraft pose estimation datasets, which ted to be less challenging than the benchmark datasets like COCO [86]. The original codebase [3] allows seamless configuration of these new networks. Table 5.3 summarizes the new scaled networks with basline HRNet-W32 network from Sun et al. [122]. Refer to Section 5.1.1 for the definition of specified HRNet architectural elements.

5.2.6. Inference

One a model is trained the weights of the parameters are saved in a model file. The trained model can then be put to the task of KD. During inference, it processes the image through the network and provides a tensor with n heatmaps corresponding to the predefined keypoints. Recall that, the coordinates are encoded into heatmap representation in the network. So, the predicted keypoint coordinates are obtained by decoding the heatmap representation. Ideally, the output heatmaps are expected to have Gaussian distribution for confidence, as provided during the training. The keypoint coordinates can then directly be determined from the heatmap using the pixel location with highest value.

Recall that the heatmap representation in the network is kept at one-fourth of the size of the original image for speed-accuracy balance. In doing so, the encoding process quantizes the keypoint coordinates. When the keypoint location is projected back on the input image, the quantization results in an undesirable error in the keypoint coordinates. To compensate for this effect, a post-processing technique using sub-pixel shifting is used from Newell et al. [118]. The sub-pixel shift provides a considerable improvement to the overall performance as shown by Zhang et al. [127]. This is visualized in Fig. 5.6. A quarter-pixel shift is made from the center of the highest value pixel, in the direction of the second highest value pixel in the heatmap. The shift is expressed as following:

$$\mathbf{p}_s = \mathbf{p}_{01} + 0.25\frac{\mathbf{p}_{02} - \mathbf{p}_{01}}{||\mathbf{p}_{02} - \mathbf{p}_{01}||} \tag{5.3}$$

where $\mathbf{p}_s$, is a 2-dimensional coordinates of the shifted keypoint and $\mathbf{p}_{01}$ and $\mathbf{p}_{02}$ are the pixel coordinates with the highest and second highest confidence in the predict heatmap respectively.

At inference time, a dataset of results is stored with predicted heatmaps (confidence map scaled to 8-bit color resolution) , refined keypoints and confidence scores of each prediction.

[3] https://github.com/leoxiaobin/deep-high-resolution-net.pytorch

5.2.7. Evaluation Metrics

A model is evaluated only based on the statistics of the keypoint location error between the predicted and true keypoint coordinates.

$$E_{kp} = ||\mathbf{x}' - \mathbf{x}||$$

For human pose estimation, metrics like Percentage of Correct Keypoints [131] and Object Keypoint Similarity [86] are generally used. However, their adaptation, utility and relevance to spacecraft pose estimation were not investigated.

5.3. Implementation

The keypoint detection network is implemented in PyTorch [132], an open-source machine learning platform built and maintained by Facebook AI Research. PyTorch is used as the original API [4] for HRNet architecture is compatible with PyTorch. Since development of cross-platform compatibility is not in the scope of this work, the HRNet implementation was adapted for spacecraft pose estimation to use PyTorch. Fig. 4.12 shows the implementation schematic with the data pipeline established to carry out the training and evaluation. The network training is done on Nvidia Tesla P100-PCIE-16GB GPU made available through IPython notebook interface in Google Colaboratory (Colab) [5]. The data is synced between the local computer and the Colab server through Google Drive connection. A successful training results in a trained model along with meta-data and training logs. The trained model can then be used/deployed across machines with Tensorflow support to execute inference and batch evaluations. For this purpose, a Tensorflow environment is also maintained in a python environment on the local machine which is used with Nvidia P1000-Quadro-8GB GPU and Intel core i7-8750H CPU. The trained model is deployed on the local machine for small-scale inference, testing and benchmarking.

The modified code-base with the necessary files that adapts the interface to the Tensorflow Object Detection API for spacecraft pose estimation research in this work, are made available [6] by the author.

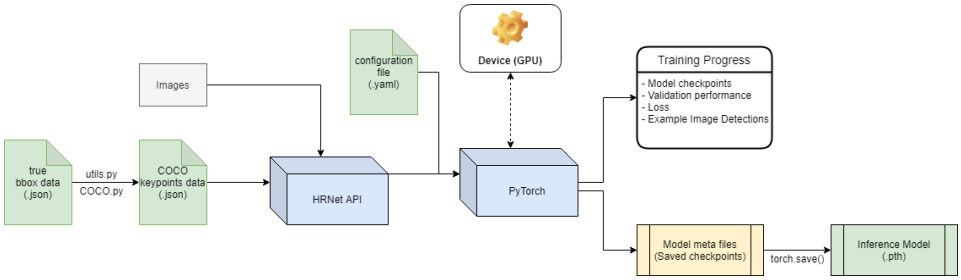
5.4. Detection Uncertainty

The CNNs in general do not account for uncertainty and do not allow direct quantification of uncertainty within their mathematical framework. However, the uncertainty in the predicted keypoints can be indirectly computed from the Heatmaps of the keypoint prediction, as proposed by Pasqualetto Cassinis et al. [57]. Since heatmaps represent per-pixel confidence of detection, it is used to quantify prediction uncertainty for the keypoint by approximating a distribution around the maximum confidence keypoint. In addition to generating uncertainty information for the navigation loop, the technique can also improve pose estimation accuracy with covariant pose solver when there are non-ideal heatmaps (like Fig. 5.2b). For details, the reader is referred to the original work [57]. The method is adapted directly to compute the image-plane covariance matrix for the predicted keypoints. First, a heatmap image is thresholded to focus on activated regions of the heatmap. Then, approximating a Guassian distribution a weighted covariance is derived around the maximum confidence point i.e. the point with highest pixel intensity. The weights for the pixel locations are empirically derived in terms of RGB intensities of the colorscale in the heatmap image that accurately captures the uncertainty. For m pixels, $\mathbf{p_j} = [u_j, v_j]$ above the threshold intensity the general expression for variance in Eq. 7.2 is given as:

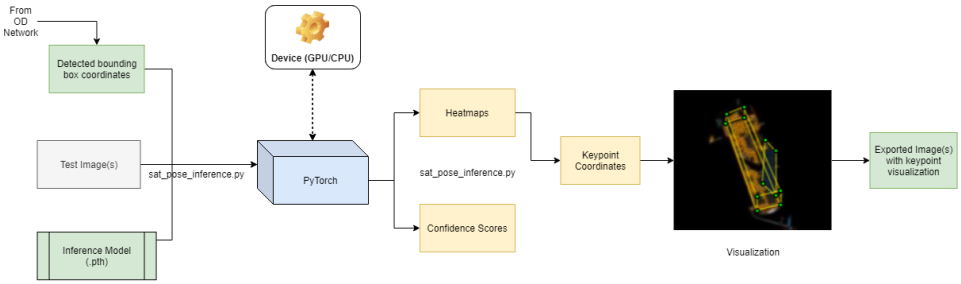
$$\sigma_{uv} = \frac{1}{m} \sum_{j}^{m} w_j (u_j - u^*)(v_j - v^*) \tag{5.4}$$

where, $\mathbf{p}^* = [u^*, v^*]$, is the predicted image-plane keypoint location estimated from the pixel with maximum confidence score in the heatmap.

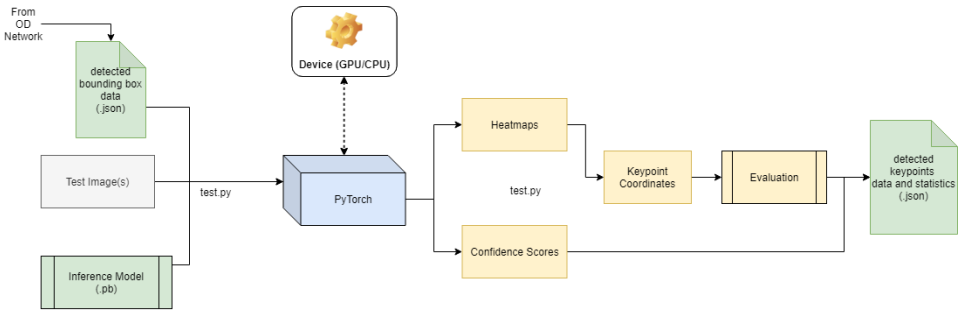
[4] https://github.com/leoxiaobin/deep-high-resolution-net.pytorch
[5] https://colab.research.google.com
[6] https://github.com/kuldeepbrd1/deep-high-resolution-net.pytorch



(a) Training



(b) Inference



(c) Batch Testing

Figure 5.7: OD implementation schematic and data pipeline/ (gray: third-party resource, blue: third-party software, yellow: intermediate implementation, green: processed/developed resources

5.5. Conclusions

The selection, design review and configuration of a KD network has been presented. After rigorous comparison, the HRNet network architecture is selected. Three candidate networks are selected for evaluation in the pose estimation pipeline. Further, the configuration and implementation of network training, data augmentations and evaluation metrics used in developing the detection models are summarized. The selected KD networks: HRNet-W32, HRNet-small and HRNet-smaller are taken forward for the navigation framework testing and evaluation.

6

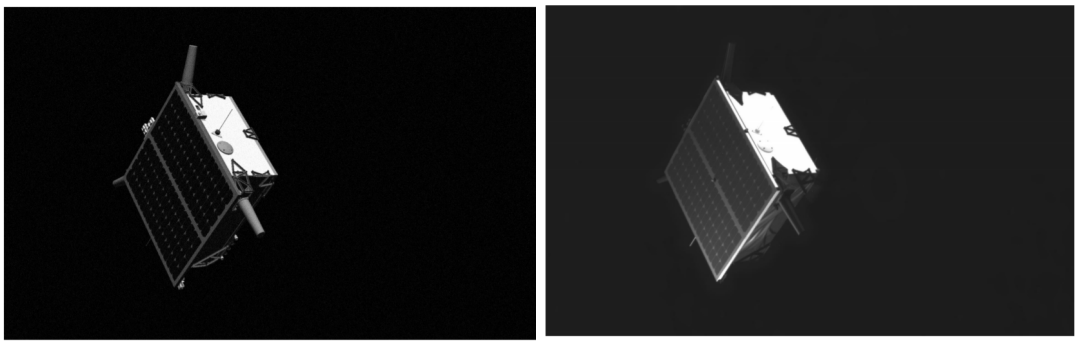
Datasets

The neural networks learn to predict outputs from patterns in the data provided during the training. A dataset is therefore central to a machine learning model's success in accomplishing a task with accuracy and robustness. In previous chapters, the selection, configuration and implementation of OD and KD networks have been described. CNNs use deep representation learning and require large datasets to be able to successfully predict the expected outputs. So far, terrestrial benchmark datasets like ImageNet[66] and COCO [86] have been used to make preliminary assessment and selection of the networks, techniques and configuration. While the terrestrial benchmarks serve as a standard reference, the datasets for spacecraft pose estimation present unique challenges of their own. Since it is not possible to acquire large-scale space-borne imagery, the datasets to be used for deep learning in spacecraft pose estimation problem need to rely on artificial rendering of the visual space environment. Unlike ideal renderings, the real image from a space-borne cameras during operation may contain optical artefacts and variations in the image, that cannot be modeled explicitly. This may form statistical outliers in the internal representations of a trained neural network model and cause failure. As noted earlier, the fundamental question and the primary hurdle to the utility of deep learning for spacecraft uncooperative navigation then, is to improve the robustness of synthetic training in the operational environment.

This chapter describes the datasets used for developing, verifying and evaluating the navigation framework and its components in detail. First, the predominant dataset- *SPEED* [82] from Stanford Space Rendezvous Laboratory has been described as a benchmark for comparison. Then, the Envisat dataset from [57] is taken as the first-order evaluation dataset for the Envisat target use-case concerned in this work. The discussion is extended to improving robustness of the deep learning models to bridge the gap between synthetic training and on-orbit operation. Further, a novel dataset is proposed for Envisat use-case that introduces key improvements to the existing synthetic dataset for the Envisat spacecraft. The focus remains on improving the existing synthetic datasets in a simple and efficient manner. The efficacy and exhaustive analysis on the aspects of photo-realism, surfaces, textures etc in computational rendering are not in the scope of this work. However, the goal is to take a step forward towards creating a standard Envisat dataset, which suitable for training and testing deep learning-based navigation systems going forward.

6.1. Benchmark Dataset: SPEED

Spacecraft Pose Estimation Dataset is the first large-scale dataset of photo-realistic spacecraft images, aimed at deep learning solutions to the close-proximity uncooperative navigation. *SPEED* consists of images of the Tango spacecraft from the PRISMA mission [133] with ground truth pose information available for training. Introduced by Sharma et al. [9], the dataset consists of synthetic and lab-generated images. The synthetic images are rendered using a custom simulator based on OpenGL library [134]. The method used by Sharma et al. [9] to generate synthetic images emulates the spacecraft surface illumination very close to that of the PRISMA in-flight images [79]. In addition to the synthetic images, the *SPEED* dataset also contains real images of a 1:1 representative spacecraft model, acquired using a camera similar to the PRISMA mission with a robotic test-bed. Fig. 6.1 shows an example of a synthetic and a real image for the same pose. Note the difference in contrast and brightness between the two images.



(a) OpenGL rendering (b) Lab-generated image

Figure 6.1: Visual comparison of real and synthetic images in *SPEED* [79]

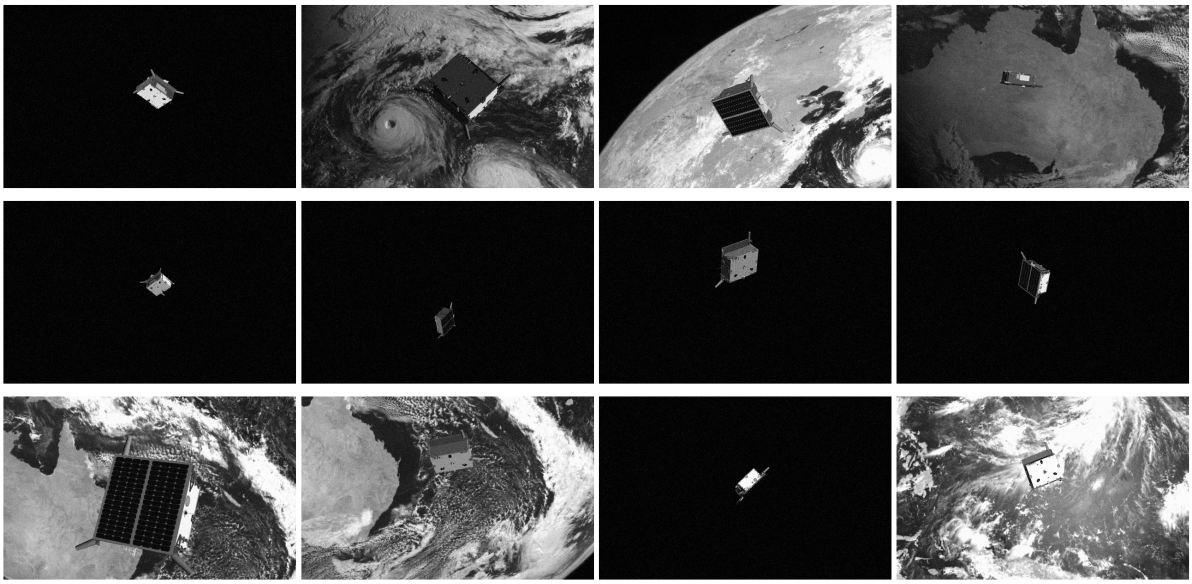


Figure 6.2: Selected synthetic images from *SPEED* [82]

A part of the synthetic images are augmented with Earth background, using the true-color full-disk image products from the Himawari-8 Earth observation satellite [135]. The Earth background is cropped from the high-resolution images such that the cropped image is representative of the field of view from PRISMA mission's orbit altitude. Then the spacecraft mask is overlaid on the cropped earth background. Also, the illumination for synthetic image rendering is configured to match the illumination in the Earth images acquired by Himawari-8. Finally, the synthetic images are corrupted with Gaussian noise and Gaussian blur. Fig. 6.1 shows a finite set of example images from the *SPEED* dataset.

6.1.1. Dataset Properties

SPEED is available in public domain from ESA's Kelvins portal[1] and Stanford Digital Repository [82]. The images are distributed into '*train*', '*test*', '*real*' and '*real-test*' sets. Note that the keyword '*real*' denotes lab-generated images. The properties of each image set is shown in Table 6.1 and the pinhole camera model used to generate the dataset images is described in Table 6.2.

The ground truth pose information is parameterized by a 3-vector for relative position of Tango's body frame origin with respect to the camera frame origin, and a 4-vector unit quaternion that represents the transformation from Tango's body frame to camera frame. In addition, the datafile in JSON format stores this information as a list of dictionaries. An example of the JSON data format is provided in Appendix C.

[1] https://kelvins.esa.int/satellite-pose-estimation-challenge/data/

Image set	Image count	Source	Pose labels	Augmentations
train	12000	OpenGL renders	✓	Earth, (Gaussian) blur + noise
test	2998	OpenGL renders	×	Earth, (Gaussian) blur + noise
real	5	TRON lab	✓	-
real-test	300	TRON lab	×	-

Table 6.1: *SPEED* image set distribution

Parameter	Description	Value
N_u	Number of horizontal pixels	1920
N_v	Number of vertical pixels	1200
f_x	Horizontal focal length	0.0176 m
f_y	Vertical focal length	0.0176 m
du	Horizontal pixel length	5.86×10^{-6} m
dv	Vertical pixel length	5.86×10^{-6} m

Table 6.2: Intrinsic camera parameters

The camera frame is such that the z-axis points in the boresight direction, while x-axis and y-axis form a right hand triad in the perpendicular plane. The body frame origin of the Tango spacecraft is at the center of the back panel, and the z-axis points in the direction perpendicular to the back panel in the direction of the solar panel. This is visualized in Fig. 6.3. The images are generated with relative distance uniformly distributed between 3m-50m, and the relative attitude is discretized and uniformly distributed in $SO(3)$ space. Figs. 6.4 & 6.5 show the distribution of pose across the synthetic image sets.

6.1.2. Pre-processing

The navigation framework in this work utilizes deep learning in the object detection and keypoint detection steps, as described in Section 3.1. The ground-truth data available in *SPEED* has to be used to train the networks for object and keypoint detection, as well as to benchmark the pose estimation pipeline. However, the object detection network needs to be trained with bounding box coodinates in the images, and the keypoint detection network needs to be trained with the keypoint coordinate data for each image. Since, these are not directly available in the *SPEED* dataset, the raw data must be pre-processed appropriately.

The ground-truth pose labels are only available for the *train* and the *real* image sets as described in Table 6.1. Therefore, only those data points have to be used for training and validation of the CNNs. To enable training and validation, the *train* image set is randomly split into two subsets- *train-1* and *val-1* image sets in 8:2 ratio, following the common practice in machine learning based on [136]. 9600 images are used for training the corresponding networks, and 2400 images are used for validation and evaluation the performance of the network. Further, the 5 images in the *real* image set are withheld completely from the networks, to

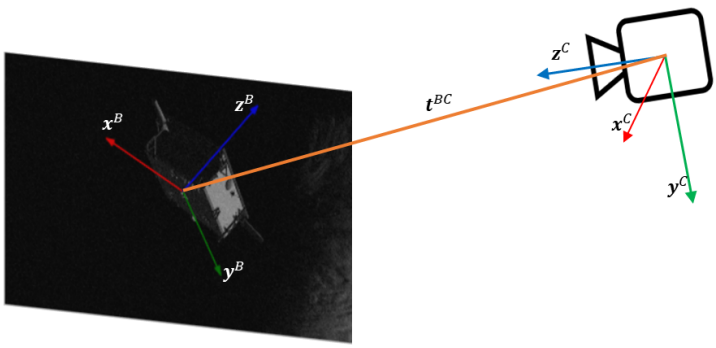


Figure 6.3: Coordinate frame definition for *SPEED* (adapted from [79])

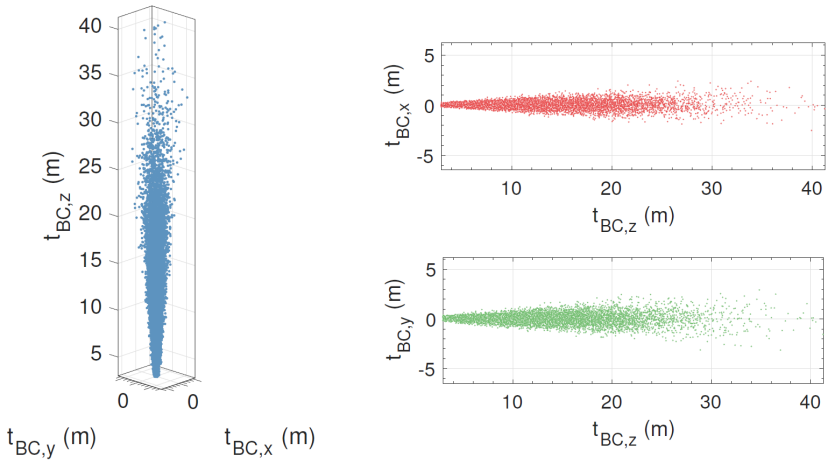


Figure 6.4: *SPEED* position distribution [16]

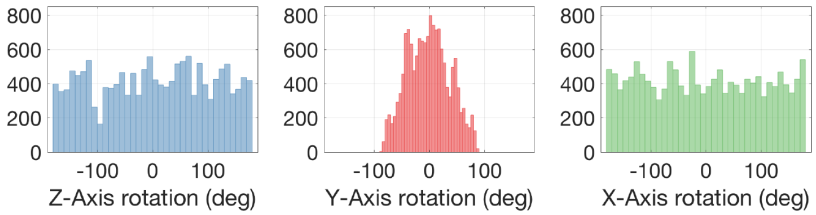


Figure 6.5: *SPEED* relative attitude distribution in terms of Euler angles [16]

assess the robustness of the synthetic training to previously unseen real images. This is done to emulate the gap between synthetic rendering environment and the reality, which is expected in real-life scenarios for the navigation framework proposed in this work.

3D keypoint model recovery

In order to retrieve the bounding box and keypoint coordinates for the images, a simplified wire-frame model is reconstructed of the Tango spacecraft. Recall the projection equation (Eqn. 2.1 & 2.2) that describes the relation between the 2D image-plane points, and 3D body points. In P*n*P solvers, 3D model points and 2D image points are used to estimate the pose. Conversely, 3D model points can be recovered using the ground truth pose information and 2D image points.

For this, 12 images showing well-illuminated spacecraft in varying poses are hand-picked. Visually iden-

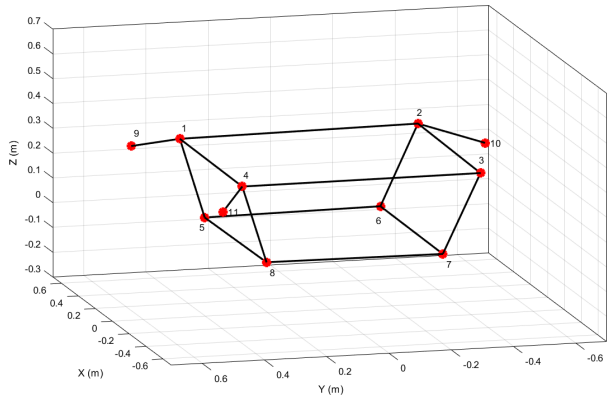


Figure 6.6: Reconstructed 3D wireframe model of the Tango spacecraft using multi-view reprojection

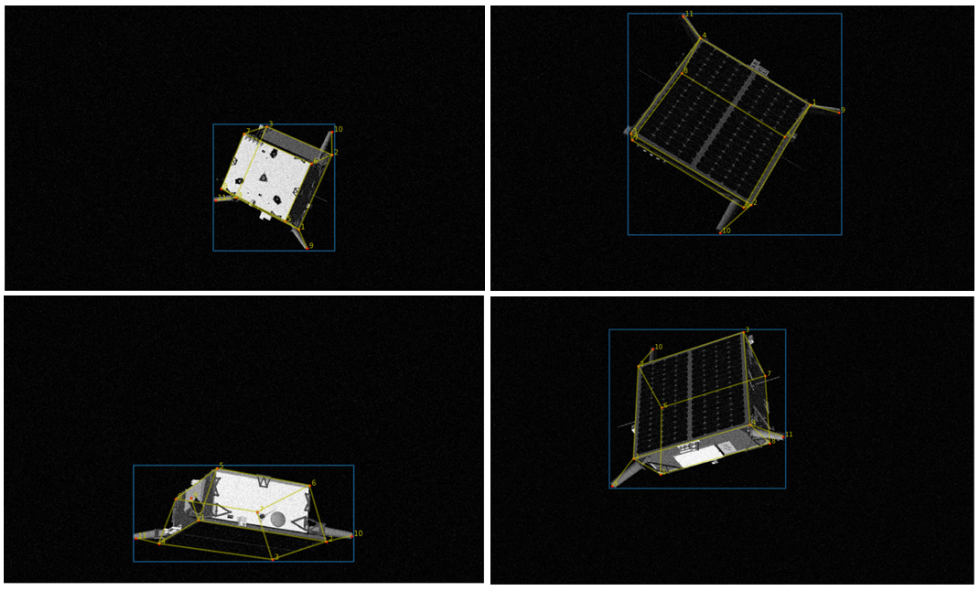
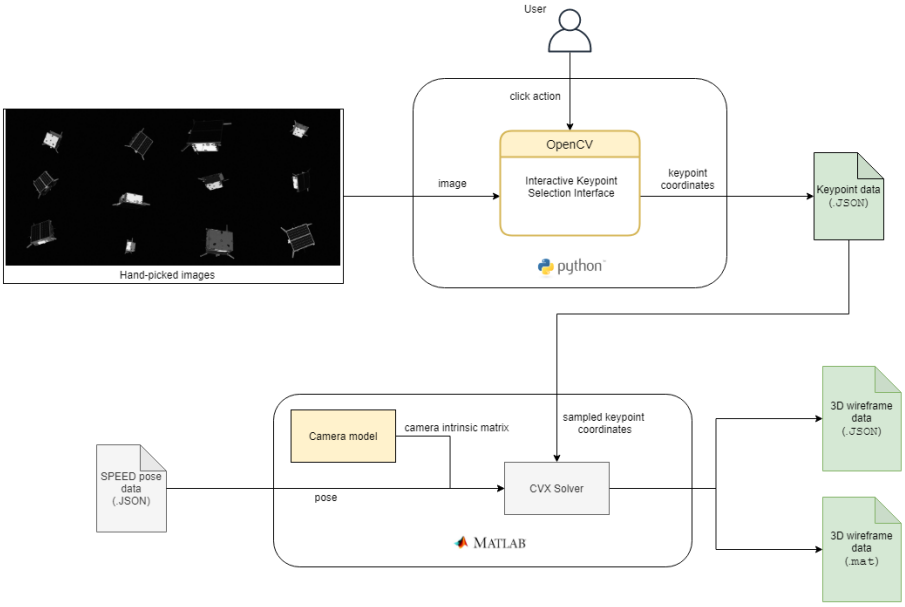


Figure 6.7: Examples of images from *SPEED* with reconstructed wireframe model overlay, showing keypoints (red), skeleton (yellow) and bounding box (blue)

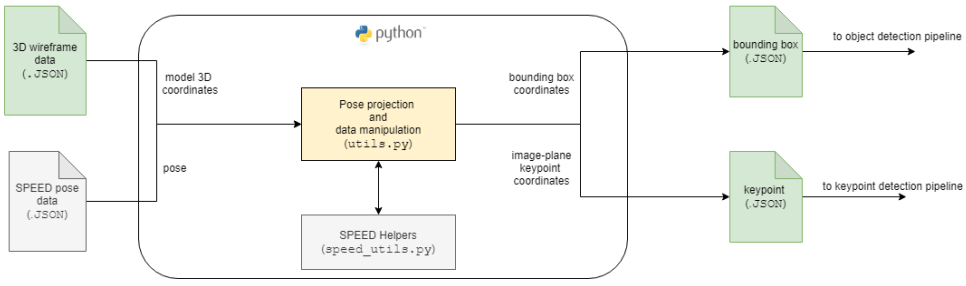
tifiable keypoints are chosen for the spacecraft to construct the wireframe model. For the Tango spacecraft in *SPEED*, four corners of the back panel, 4 corners of the solar panel and three tips of the antennae are chosen. In each of the hand-picked image, these points are manually located and the coordinates are noted. A simple tool is developed in Python that allows rapid user selection of points from an interactive window. The tool sequentially opens the images, takes click inputs to record individual keypoints and exports a JSON file for the 3D wireframe model containing the coordinates of the manually identified keypoints. Then, using the keypoint coordinates across images and the projection equations, an optimization problem for minimizing the re-projection error is given as following:

$$\mathbf{r}_i^B = \underset{\mathbf{r}_i, w_j}{\arg\min} \sum_{k}^{12} || \, w_j \, \mathbf{p}_i^h - \mathbf{K}\,\mathbf{P}\,\mathbf{r}_i^B \, || \tag{6.1}$$

The notation follows from Eqs. 2.1 & 2.2, where, for the i 3D body-frame keypoints and j sampled images, $\mathbf{p}_i^h$ are the homogeneous image-plane coordinates of the keypoints, $\mathbf{K}$ is the camera intrinsic matrix and $\mathbf{P}$ is the pose matrix composed of direction cosines and translation vector components. The unknowns of the problem, $\mathbf{r}_i^B$ represents the body frame coordinates of the i^{th} point and w_j is the scale factor between the homogeneous and image plane coordinates for the j^{th} image. The problem is that of a multi-view triangulation and the L2 reprojection error function is convex. This solved using the CVX solver [137, 138]. The average reprojection error for the solution is less than 10 pixels per keypoint per image, with respect to the manually selected keypoints. The Tango wireframe model from the resulting 3D point solution, is shown in Fig. 6.6. The wireframe model points are stored in `.JSON` and `.mat` formats for convenient use across platforms and tools. With the given body-frame coordinates of the wireframe model, the keypoint coordinates in the image-plane can be computed with projection equations using the ground-truth pose. Further, the four bounding box coordinates can be computed from the image-plane coordinates using maximum and minimum x and y coordinates across keypoints in an image. This provides the corner representation (bottom-left and top-right) of the bounding box, which can further be transformed into alternative representation as required. To improve object detection training, the ground truth bounding box coordinates are relaxed by a small margin as noted by Chen et al. [78]. The model is verified by projecting the points on random images from the synthetic image sets, and is found to fit very well to the spacecraft in the image. Fig. 6.7 shows a few examples of wireframe projections overlaid on *SPEED* images using the corresponding pose information. This is repeated for all of 12000 images in the dataset, to generate two compatible data files for OD and KD networks in the framework. The overall flow of the implementation is summarized in Fig. 6.8.



(a) Model reconstruction flow



(b) Data manipulation flow

Figure 6.8: Keypoint and bounding box data pre-processing flow for *SPEED*

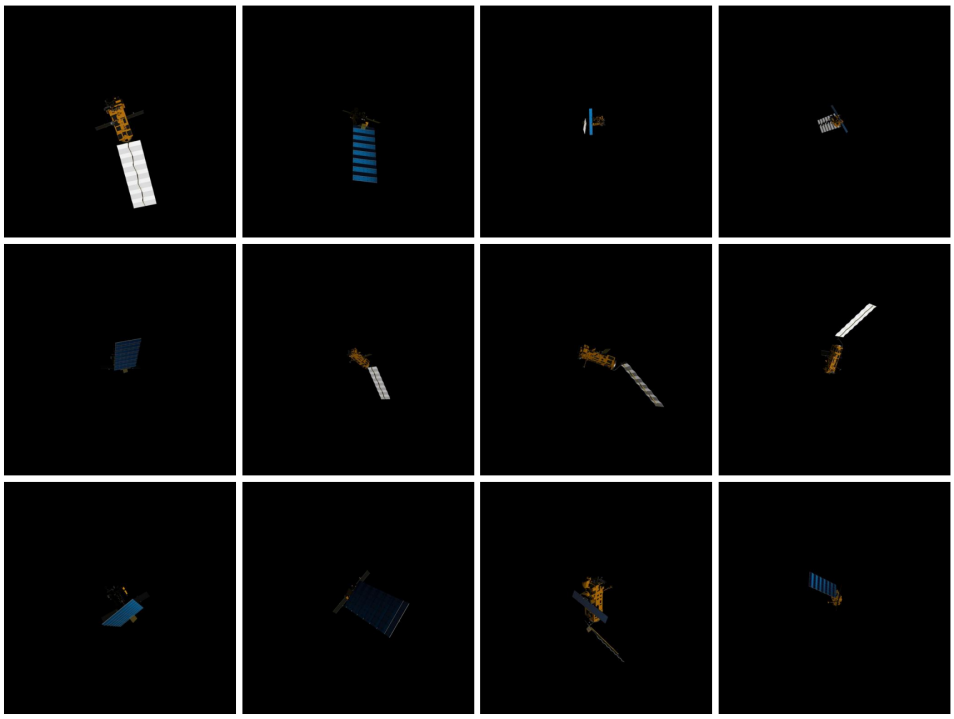


Figure 6.9: Examples of synthetic images from *Envisat-1* dataset

6.2. Evaluation Dataset: Envisat

For Envisat target use-case in this work, the dataset from Pasqualetto Cassisnis et al. [57] is used for a first-order evaluation of the navigation framework for Envisat target use-case. This is done to assess the existing Envisat dataset for pose estimation and draw comparisons with the benchmark dataset- *SPEED*. While the dataset contains synthetic spacecraft rendering without augmentations, it also provides image sequences that emulate close-proximity trajectories. This is important as it allows evaluation of the whole navigation loop along with the state estimator, which is currently not possible with *SPEED*. For clear identification, the dataset is called *Envisat-1* from here on. Fig. 6.9 shows a finite set of synthetic images from the *Envisat-1* dataset.

6.2.1. Dataset Properties

Envisat-1 is a dataset of relatively simple synthetic images of the Envisat spacecraft. The dataset is generated with Cinema4D [2], using an accurate texture model of ESA's Envisat spacecraft. The dataset and related resources are available internally [3]. The base dataset is generated by discretizing the relative distances between 90m-180m at 30 m intervals, and the attitude with three Euler angles at 10-degree intervals. Fig. 6.10 shows the camera poses relative to the stationary target, used for image generation. The resulting image set is then sampled randomly and split into image sets: *train*, *val* and *test* sets. Further, a trajectory image set, *trajectory-150m* is also available containing a 36 image sequence emulating a tumbling Envisat spacecraft with the servicer camera point along the V-bar at 150m separation. Table 6.3 describes basic distribution of image sets used in this work and Table 6.4 describes the camera model used to generate the images.

Wireframe 3D model

Unlike *SPEED*, the Envisat wireframe model is available and does not need to be reconstructed. The wireframe model consists of 16 points that include eight keypoints close to the corners of the main body, four corners of the SAR antenna and four corners of the extended solar panel. The keypoints are represented in the body frame defined at the assumed center of mass of the main body of the Envisat spacecraft. The X-axis

[2] https://www.maxon.net/products/cinema-4d/
[3] Courtesy of TU Delft SpE/ESA/Airbus

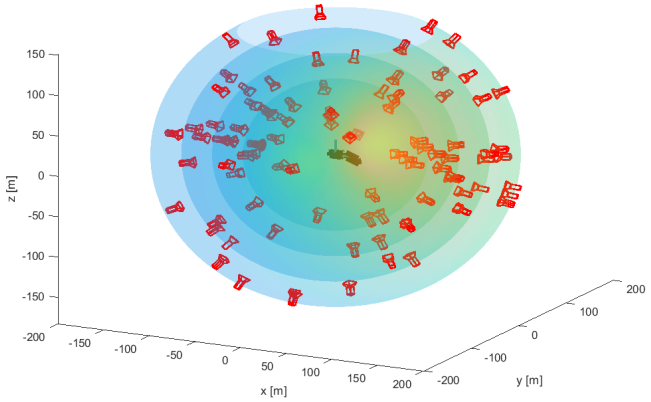
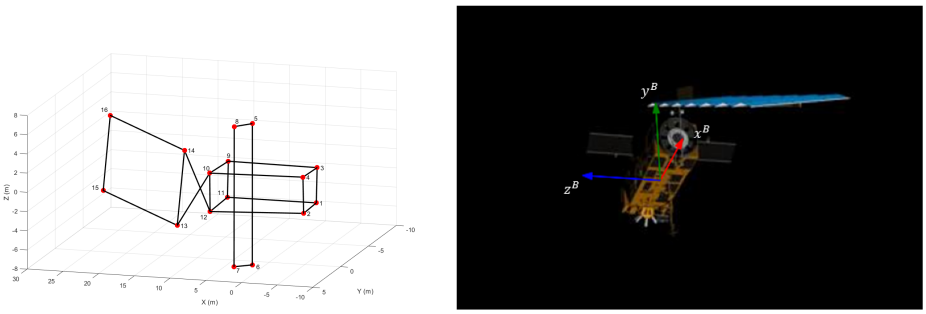


Figure 6.10: Camera pose distribution around Target for image generation [57]

Image set	Image count	Source	Pose labels	Augmentations
train	35000	Cinema4D	✓	None
val	9604	Cinema4D	✓	None
test	9608	Cinema4D	✓	None
trajectory-150m	36	Cinema4D	✓	None

Table 6.3: *Envisat-1* image set distribution

of the body frame is perpendicular to the launch interface in direction of the solar panel, the Y-axis is perpendicular towards the SAR antenna and the Z-axes forms the right-hand triad with the other two. The wireframe model and the visualization of the body frame in an image is shown in Fig. 6.11.



(a) Wireframe Model (b) Body-frame axes

Figure 6.11: Visualization of the wireframe 3D model definition and the body-centered reference frame

The COCO file format is used in the keypoint detection pipeline (see Fig. 5.7) and relates to the COCO benchmark [86] on which most modern CNNs are trained and evaluated. Having the data in COCO format allows rapid adaptation to most CNN architectures for spacecraft pose estimation, as it is compatible with most neural network software repositories with minimal configuration. Finally, the bounding box data is computed from the available keypoints. The bounding box is relaxed by around 10% of the original width and height and is stored in a `.JSON` format. Fig. 6.12 shows samples from *Envisat-1* dataset with wireframe model projections and bounding box visualization. Note that all the associated files can be found in the software repository [4].

[4] https://github.com/kuldeepbrd1/deep-high-resolution-net.pytorch

Parameter	Description	Value
N_u	Number of horizontal pixels	512
N_v	Number of vertical pixels	512
f_x	Horizontal focal length	0.00387 m
f_y	Vertical focal length	0.00387 m
du	Horizontal pixel size	5.5×10^{-6} m
dv	Vertical pixel size	5.5×10^{-6} m

Table 6.4: Intrinsic camera parameters for *Envisat-1* images

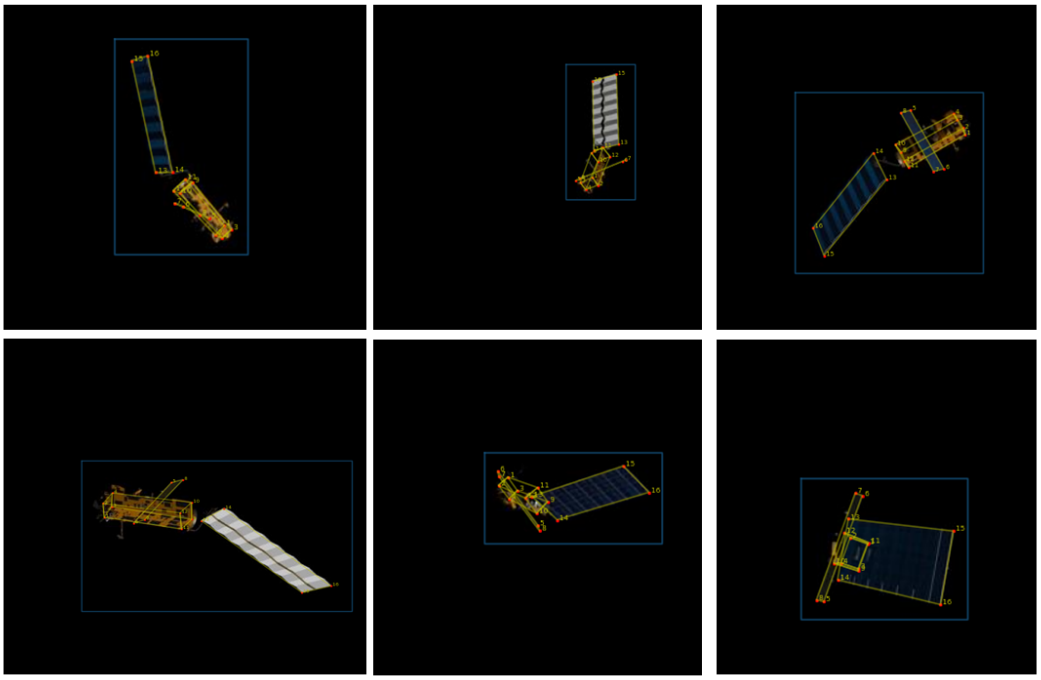


Figure 6.12: Examples of images from *Envisat-1* with 3D wireframe projection overlay, showing keypoints (red), skeleton (yellow) and bounding box (blue)

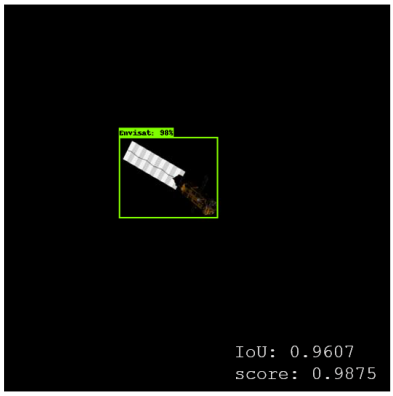
6.3. Robust Learning through Datasets

Deep learning with CNNs achieves great performance in several computer vision tasks. By learning abstract representations internally, CNNs are able to learn a non-linear mapping that provide detection from unstructured images with a varying artefacts and properties like illlumination, where traditional image processing techniques fail. However, machine learning models cannot generalize globally, and are only as good as the data used during the training. Consequently, the neural networks can behave in an unpredictable manner when the input provided is "out-of-distribution" with respect to the training data, i.e. artefacts not modeled in the training data that the neural network might encounter in real life. For instance, a network trained solely on clean images from *Envisat-1* dataset images, fails to detect the Envisat spacecraft in presence of motion blur and impulse noise as shown in Figs. 6.13(a), 6.13(b). The case of detection failure with Earth in the background background, shown in Figs. 6.13(c), 6.13(d) is more concerning as the network assigns a high confidence score for the detection when the box localization is highly inaccurate, as reflected by the IoU value. This imposes a significant risk, especially for spacecraft pose estimation problem where the networks are trained on synthetic images. Robustness is therefore an important aspect that must be investigated for navigation systems that will utilize deep learning. Two major factors are identified in this scope:

1. **Robustness in learning**: The aspects of training process including training dataset generation must be designed in a way that allows the network to learn internal representations that transfer well to real images. A network cannot generalize to artefacts in the images that are not present in the training set purely, and the training process and datasets must therefore be accountable for improving data

(a) Clean

(b) Augmented: Level-1 impulse noise and motion blur (see Appendix-D)



(c) Clean

(d) Augmented: Earth background

Figure 6.13: Detection results showing lack of robustness of an object detection network trained on clean images from *Envisat-1* dataset

distribution.

2. **Representative robustness evaluation**: The validation and test process, including the corresponding image sets must be designed in a way that allows fair evaluation of a trained model. For example, if a validation set contains images with exact same illumination angle or noise magnitude used during the training, the network's generalization cannot be judged fairly.

In order to improve robustness in training, data augmentation is the most widely used approach. In Chapters 4 and 5, simple data augmentations were included in the training process, that randomize image brightness, contrast, scale and rotation. These effectively improve robustness of the trained network to pixel-level variation and poses not sampled in the training set. For instance, *Envisat-1* dataset only contains images for four discrete values (90m, 120m, 150m and 180m) of relative range. However, using random rotation and scaling augmentation during training allows the network to learn and infer accurately even when the spacecraft is at ranges not explicitly available in the training set.

A critical factor for spacecraft pose estimation using deep learning is the degree of photorealism in the synthetic environment. Rendering environments like OpenGL used for *SPEED*, Cinema4D used for *Envisat-1* or Unreal Engine used for autonomous terrestrial robotics [139], are common for deep learning applications. However, Brochard et al. [140] note that most commercial graphics simulators lack the photometric accuracy that adequately represents a scenario in the space environment. For spacecraft vision-based applications, tools like ESA's PANGU [141, 142] and Airbus' SurRender [140] are also available. However, their efficacy to large-scale deep learning dataset generation has not been investigated. So, the problem of representing the virtual environment precisely, holds several underlying problems. Also, the burden of high photorealism can

increase the system development complexity and restrict wide-scale adaptation as every part including the sensor system must be accurately modeled in the virtual environment. An alternative is to adapt training process that transfers well to reality, as noted earlier.

Several approaches have been proposed to improve robustness of neural network models in such cases. For the relevance to this discussion, they can be divided into following:

1. **Adversarial Robustness**:

   Adversarial robustness is the major theme in computer vision approach that concerns with tackling failure on adversarial images. Adversarial images are clever alterations in the image information that can be used to 'confuse' the network by exploiting the internal representations learned by the network. Generally, the adversarial images are visually similar but contain adverse data patterns in the RGB space that fool the neural network [143, 144]. Methods to identify and counter adversarial attacks have been proposed [145–147], but remain an evolving subject as more crafty adversarial attacks are designed to push the neural networks to the limits.

2. **Texture/Shape Robustness**:

   Object texture in an image is a characteristic information that the convolution operations extract in the modern CNNs. The initial layers of a CNN extract lower-level abstract features in the input image, and subsequent layers extract higher-level abstract features as combinations of lower-level features [148]. In doing so, CNNs are found to give more importance to the texture in the images rather than global shapes, as noted by Geirhos et al. [149]. Therefore, CNNs performance can degrade when the local textures in the object are different from the ones in the training examples, despite the object shape. In general for natural objects and specifically for rigid bodies, shape is a distinct characteristic that must be captured by the CNNs for robust performance in the real world. This is expected for spacecraft pose estimation application as well, where the synthetic environment may not accurately model the textures that a sensor would produce on-board. Commonly, texture stylization or randomization [150] is used to tackle this. It has shown to eliminate spurious correlations related to local textures and emphasize on the object shape, improving texture/shape robustness [149].

3. **Corruption Robustness**:

   In real-life applications, the camera sensor and environment conditions can introduce visual corruptions in the image. These consists of several types of noise, blur, spatter etc., which are common to terrestrial images. Hendrycks and Dietterich [151] devise a new benchmark dataset wherein the natural images are corrupted with the common effects. The networks trained without corrupt data are found to lack robustness to real images containing such corruptions. Consequently, the synthetic datasets that train the networks for real-world applications, must also include the common corruptions expected in the operational environment.

For spacecraft pose estimation, the three categories of robustness mentioned above, must be addressed. For texture robustness, Park et al. [46] use texture randomization while training a keypoint regression network for spacecraft pose estimation. Using a custom dataset of the Tango spacecraft similar to *SPEED*, performance improvement is shown for real in-flight images from the PRISMA mission, using the Neural Style Transfer (NST) technique from Huang et al. [150]. However, commonly used texture randomization techniques like NST cause loss of edges and can create a bottleneck for CNN performance.

Adversarial robustness and corruption robustness remain unexplored for spacecraft pose estimation. The importance of adversarial robustness is brought into question by the fact that adversarial attacks most often do not manifest in reality and are rather designed to specifically attack specific architectures of neural networks. Therefore, adversarial robustness is excluded from the scope of this work and remains to be investigated further. On the other hand, corruption robustness is a necessity for images acquired from a space-borne sensor. The images taken in space often suffer from cosmic ray hits, plume fogging, defocusing, relative motion blur among others. As an example, the *SPEED* synthetic images are augmented with Gaussian noise and Gaussian blur, as with most common datasets. However, networks trained on a specific blur type, like Gaussian, do not ensure performance on other types of blurs, like zoom blur or motion blur [152]. Similarly for noise, Hosseini et al. [153] reveal limitations of networks to impulse noise. In general, it is found that fine-tuning on specific image corruptions do not ensure robustness to other types of corruptions [154]. The CNN training must therefore account for such corruptions during training.

Once augmentations are included in the training process, the second aspect comes into play i.e. the evaluation process. It is necessary that the evaluation of a trained network be able to qualitatively and quantitatively test the robustness. Qualitatively, the validation and test examples shall not exactly mimic the training examples, in terms of distribution. Quantitatively, the metrics used for evaluation must be able to capture the robustness aspect. For instance, *SPEED* provides real lab images for evaluation, which are very different from the training image set. Further, evaluation on *SPEED* are judged with a custom score metric that reflects the underlying problem and is able to demonstrate robustness. Kisantal et al. [79] show the network performance gap between synthetic and real image evaluations and therefore the robustness for various learning based pose estimation pipelines, using the SLAB/ESA score metric.

The discussion above establishes the reasoning and baseline for the newer datasets and how they must manifest certain aspects. This is used to take a step forward in improving the relatively simple *Envisat-1* dataset in the following section.

6.4. Improving Envisat Datasets

From earlier discussions, it is clear that the *Envisat-1* dataset lacks several features that are expected in spacecraft pose estimation dataset. In this section a new dataset is presented for the Envisat target use-case, that allows CNNs to improve robustness to common image corruptions. The goal is to improve existing dataset with minimal development rather than venture into computer graphics generation, photorealism and virtual environment analysis. The aspects of 3D graphics rendering is considered a dedicated research topic on it's own, and is not explored here. Instead, the creation of *Envisat+IC* (Envisat-1 + Image Corruptions) is presented, in line with the discussion presented in the earlier section. Another dataset *Envisat+TR* (Envisat-1 + Texture Randomization) was also compiled, but was not utilized in the main study due to the poor data quality. A brief description on failure and the reasons are outlined in Section E in Appendix E. Further, adversarial robustness is not taken into account for the reasons mentioned earlier.

Image Corruptions

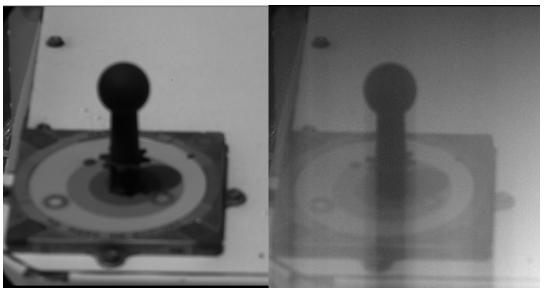
Real-life camera sensors most often suffer from corruptions in their operation environment. For space-borne cameras, the image corruptions often occur due to miscalibration, cosmic ray hits, motion-induced morphological effects and plume condensation among others. Bell et al. [155] show artefacts of shot noise, dead pixels, image brightening, compression artefacts for the Mars Science Laboratory rover cameras in pre-flight and post-flight imagery, as seen in Fig. 6.14(a) & 6.14(b). It is common to also spot image corruptions in the ISS broadcasts as seen in Fig. 6.14(c). Taking these into account, a new dataset is generate with subset of effects noted in [151] for increasing corruption robustness of networks in terrestrial images, are taken into account in creating the new dataset. These are shown in Fig. 6.15 and summarized below:

Noise

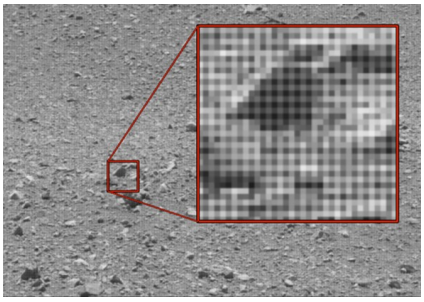
- **Gaussian Noise**: Statistical noise common to digital images and electronic devices, that follows a normal distribution and originates due to high internal temperature and poor illumination.
- **Shot Noise**: Statistical noise that follows Poisson distribution and occurs as a result of fluctuations in the number of photons within the exposure time.
- **Impulse Noise**: Impulse noise, also called salt and pepper noise, can arise due to digitization, bit transmission, or events like cosmic ray hits, that corrupt random pixel values emulating dead pixels (0) or hot pixels (255).
- **Speckle Noise**: Multiplicative noise appears in an image similar to Gaussian noise but with a gamma distribution, often due to high dynamic range of the sensor.

Blur

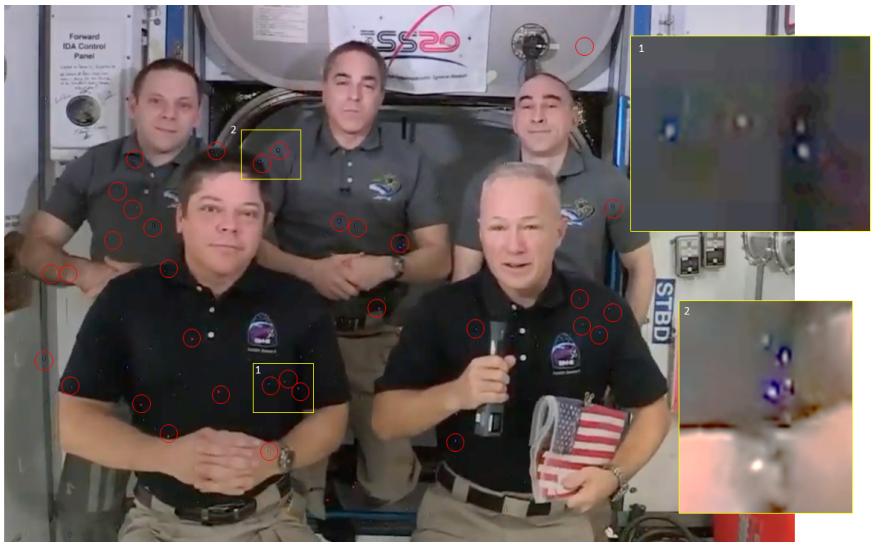
- **Gaussian Blur**: Gaussian blur can result from post-processing of raw image with a common low pass Gaussian filter.
- **Defocus Blur**: Defocus blur can result from detector misplacement along the optical axis away from the focus.



(a) Curiosity Mastcam: calibration target (left) with shutter smear and ghost image effect



(b) Curiosity Mastcam: High frequency noise and compression effect resulting in checkerboard pattern



(c) ISS Demo-1 crew broadcast

Figure 6.14: Downlinked images from (a & b) Curiosity [155] and (c) ISS [Credits: NASA TV 01-08-2020] with visually distinct image corruptions

- **Motion Blur**: Motion blur results from when the target object has a significant relative motion within the exposure time.
- **Zoom Blur**: Zoom blur results similar to the motion blur but due to motion along the camera boresight, producing a distinct concentric blur around the center.

Others

- **Spatter**: Spatter augmentation recreates liquid condensation effect on the camera that can occur due to a plume or venting.
- **Color Jitter**: This augmentation is randomizes brightness, contrast and saturation of the images to represent a broad range of optical conditions.
- **Random Erase**: Random erase augmentation randomly deletes rectangular part of the image to improve CNN detection robustness to occlusion and partial illlumination conditions [113]

The algorithm to corrupt images with above effects is adapted from the library [5] provided with [151], and is adapted for Envisat datasets. Color jitter and random erase augmentations are used directly using PyTorch [132], as they are natively available.

[5] https://github.com/hendrycks/robustness

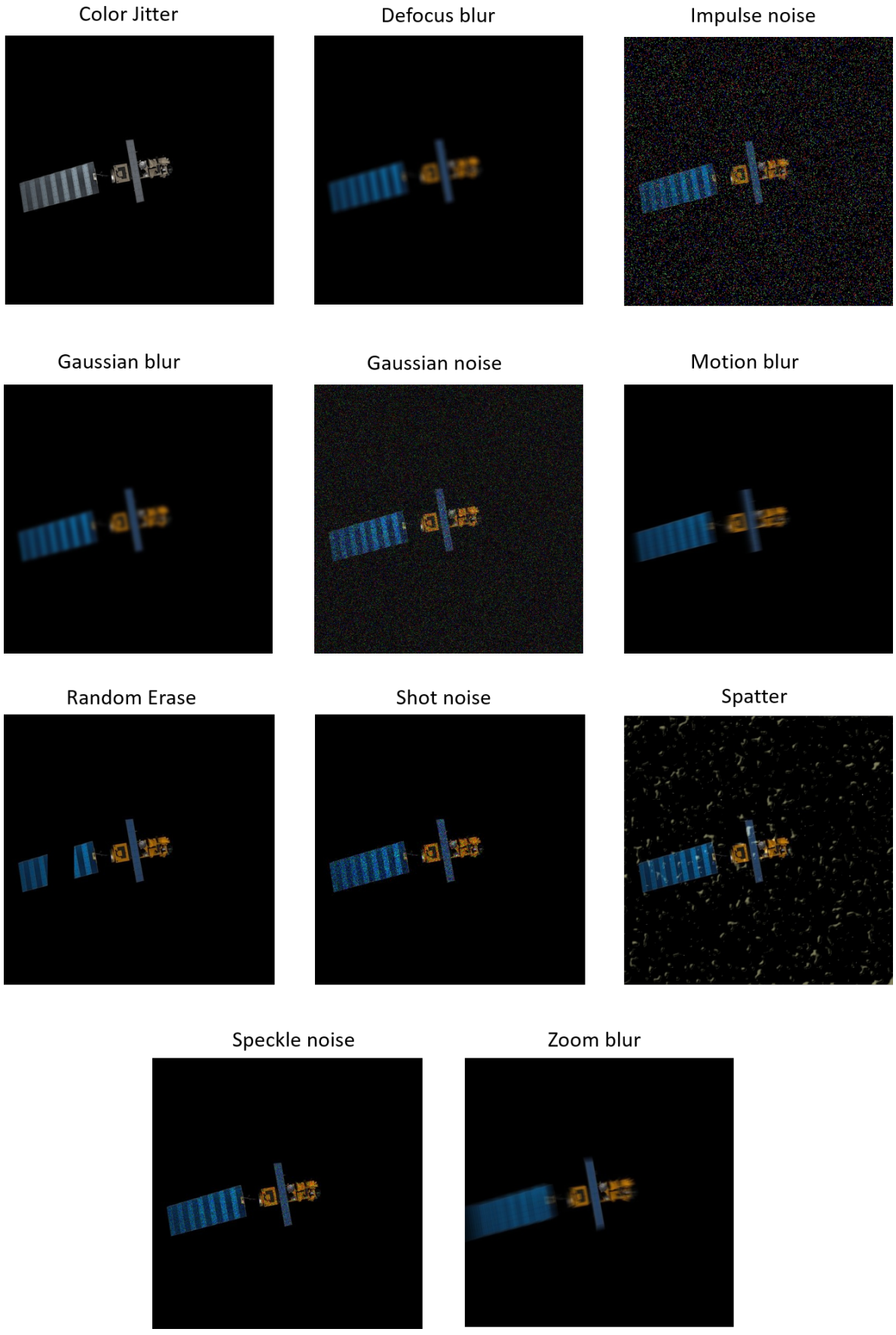


Figure 6.15: Image corruptions applied to a sample *Envisat-1* image

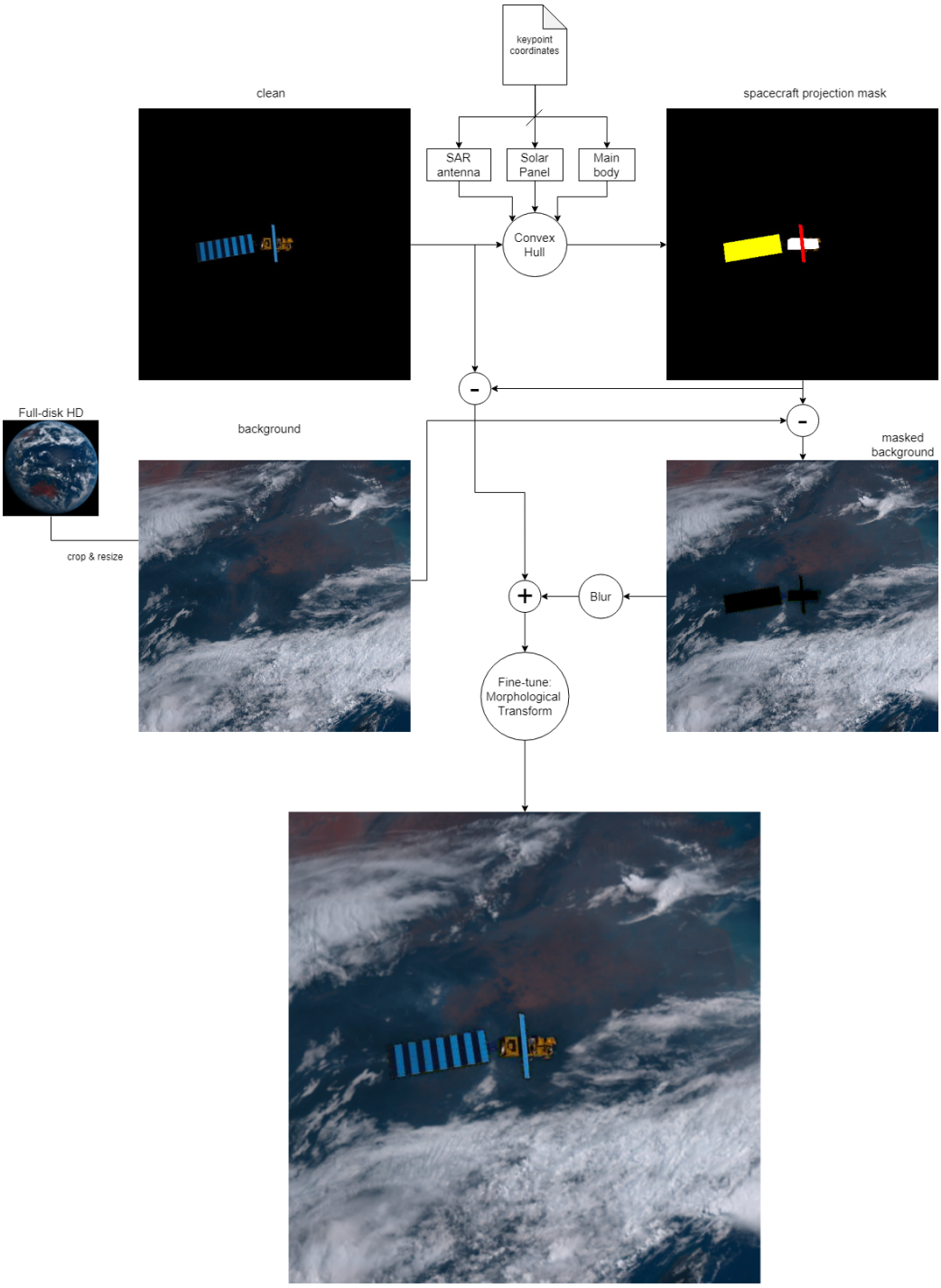


Figure 6.16: Visualization of Earth augmentation for *Envisat-1* images using projection mask

Earth background augmentation

One of the important visual characteristics of uncooperative proximity operation is the probable presence of Earth in the image background. Pose estimation systems must be able to operate reliably in presence of adverse textures from the Earth in the background. For learning-based methods, if the Earth is not rendered in the synthetic images, the networks may behave in an unpredictable manner as shown in Fig. 6.13(d). In order to improve robustness to the bright Earth, *Envisat-1* dataset is augmented with space-borne images of Earth captured by the Himawari-8 from its Advanced Himawari Imager (AHI) [135]. AHI provides high resolution images of the Earth's full disk at an interval of 10 minutes. The true color RGB composites of the Earth's disk are available through Himawari Cloud Archive [156]. For augmentation, the images acquired by AHI on 1st June for a 12 hour time-span are used. These 120 images represent a half cycle variation of Earth's illumination.

The Himawari images of 11000 px × 11000 px are randomly sampled and cropped, such that it is representative of the approximate field-of-view expected for a camera at Envisat's orbital altitude. This is done using first-order assumption of field-of-view of Himawari-8's AHI and computing the pixel area expected from the *Envisat-1* camera at around 772 km altitude. This corresponds to nearly a 1000 px × 1000 px area in the original 11000 px×11000 px full-disk image. The cropped area is then resized and fused with a clean image from the *Envisat-1* dataset. Note that for CNNs, the Earth background represents an adverse image texture that it must learn to oversee. Therefore, accurate computation of the exact FOV crop is not necessitated.

Figure 6.17: *Envisat-1* images augmented with Earth background

The algorithm flow for augmenting Earth background is shown in Fig. 6.16. The augmentation is done by first masking the spacecraft surface projections on the image plane. The keypoints considered in the Envisat's wireframe model are corners or extreme points on the spacecraft body. Since the coordinates of the keypoints are available, the area under the spacecraft's image projection can be masked. First, the Envisat keypoints are divided into three parts: the main body (8 keypoints), the SAR antenna (4 keypoints) and solar panel (4 keypoints). Each set of keypoint is used to create a non-intersecting polygon mask over the image that tightly and entirely engulfs the spacecraft projection. The outermost polygon mask is found using convex hull to get rid of internal points. The masked pixel locations are then removed from the cropped Earth image and fused with the *Envisat-1* image using bitwise operations. To represent difference in relative distance, Gaussian kernel is used to blur the background more than the foreground. Finally, morphological transformations like dilation and erosion are used to fine-tune the augmentation and remove adverse effects on the mask edges. The complete augmentation algorithm is implemented using OpenCV [157]. The resulting images after Earth background augmentation are shown in Fig. 6.17.

Number of augmentations	*train*	*val*	*val-challenge*	*val-clean*
0	11140	1518	1272	5000
1	10832	1436	1381	-
2	7330	1157	1152	-
3	4532	733	958	-
4	1166	157	238	-
total	**35000**	**5000**	**5000**	**5000**

Table 6.5: Distribution of images with respect to the number augmentations per image in the *Envisat+IC* image sets.

Envisat+IC

The augmentations described above are consolidated and scaled for large-scale augmentation. The augmentations are first assigned randomly for each image in the *Envisat-1* dataset. First, a fraction of the images are randomly assigned the Earth background or random erase augmentation. Then, half of all images are augmented with random color jitter, which randomizes brightness, contrast and saturation. Then, a fraction of all images are augmented with upto two augmentations from the rest of the augmentations i.e. noise, blur and spatter. Finally, subsets of all images are consolidated into a training set of 35000 images and validation sets (*val*, *val-clean*, *val-challenge*) of 5000 images each. In order to validate performance of networks trained on *Envisat+IC* images on the clean images, a *val-clean* set is also included as a validation set. The *val-clean* set contains 5000 clean images from *Envisat-1* i.e. without any augmentations. To stress test robustness on image corruptions, the *val-challenge* contains 5000 image with higher severity[6] of augmentations and higher number of images with augmentations. Together, these three datasets allow a fair assessment of robustness to unseen artefacts in the images. As per the discussion above, the augmentations are not mutually exclusive and an image may contain upto 4 augmentations. The distribution of number of augmentations per image are provided in Table. 6.5 and the distribution of specific augmentations are provided in Table 6.6 for the newly generate *Envisat+IC* dataset.

Augmentation	*train*	*val*	*val-challenge*	*val-clean*
None	11140	1518	1272	5000
Earth background	7000	1006	1271	-
Random erase	5250	256	257	-
Color jitter	10500	1795	2007	-
Spatter	2123	363	390	-
Gaussian blur	4151	715	383	-
Motion blur	2137	368	402	-
Zoom blur	2169	388	404	-
Defocus blur	2161	351	409	-
Gaussian noise	2072	335	472	-
Impulse noise	2072	312	519	-
Speckle noise	2048	341	515	-
Shot noise	2069	347	482	-

Table 6.6: Distribution of image count per augmentation in the *Envisat+IC* image sets.

[6] Explanation of severity levels magnitudes of quantities defining severity of each augmentation is summarized in Appendix D

6.5. Conclusions

In this chapter, the datasets used to train the CNN along with the aspects of the data pipeline necessary for implementation and experimentation are presented. Existing synthetic image datasets of the Envisat spacecraft as well as the Tango spacecraft from the PRISMA mission are reviewed, with a discussion on their features and limitations. Further, motivating the need for better datasets to enhance robustness, in particular to image corruptions, a new *Envisat+IC* dataset is generated. The specifics of the new dataset, distribution and generation are summarized.

7

Pose Solver

The pose solver solves the PnP problem to estimate the pose of the camera relative to the observed target, by associating a perspective projection of 2D points to the corresponding set of 3D points. For the pose estimation pipeline in the framework, the pose solver utilizes the predefined set of keypoints detected by the KD network. As highlighted in Sections 2.3.2 & 3.1, the MLPnP solver is selected to preserve uncertainty information with a loosely-coupled estimator. In this chapter, the MLPnP solver and the underlying algorithm are detailed. The analysis on limitations of the original MLPnP algorithm are identified and a novel modification is proposed to overcome the existing limitations. A preliminary analysis is presented to justify the reasoning, followed by extensive verification of the proposed modification.

7.1. Maximum Likelihood Perspective n Points (MLPnP) solver

MLPnP [58] is a non-iterative polynomial solver that frames the PnP problem as an MLE problem exploiting the minimal representation of uncertainty for 2D points developed in Forstner [59]. The MLE problem is framed in terms of bearing vectors that are obtained from spherical normalization of image points in the camera frame. Subsequently, by framing the PnP problem as that of estimating camera pose in a linear model represented by residuals of point projections in the tangent space of the bearing vectors. In the following description, the formulation of bearing vectors, tangent space and uncertainty propagation is elaborated. For extensive details on the derivation and proof, readers are encouraged to refer to the original papers by Urban et al [58] and Forstner [59].

7.1.1. Observations and Measurements

Consider a set of $\mathbf{p}_i$; $i = 1,2...n$ points observed on the image plane (ref Fig. 2.6), the measurements are expressed as following:

$$\mathbf{p}_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} \tag{7.1}$$

The uncertainty associated with the measurements are represented by the covariance matrix for the measurements:

$$\Sigma_{\mathbf{p}_i \mathbf{p}_i} = \begin{bmatrix} \sigma_u^2 & \sigma_{uv} \\ \sigma_{uv} & \sigma_v^2 \end{bmatrix} \tag{7.2}$$

where, diagonal elements are the variances of the 2D measurement in u and v axes respectively. For the pose estimation pipeline in this work, the measurements are the keypoint coordinates inferred by the keypoint detection network.

7.1.2. Uncertainty Propagation

Consider a linear mapping (f), i.e. the dependent variable $\mathbf{y}$ being a linear combination of an uncertain variables $\mathbf{x}$, given in matrix form as:

$$\mathbf{y} = f(\mathbf{x}) = \mathbf{A}\mathbf{x} \tag{7.3}$$

where, $\mathbf{A}$ is an arbitrary system matrix. Given the covariance ($\Sigma_{\mathbf{xx}}$) of the independent variable $\mathbf{x}$, the covariance matrix Σ_{yy} associated with $f(\mathbf{x})$, is computed as following:

$$\Sigma_{\mathbf{yy}} = \mathbf{A}\Sigma_{\mathbf{xx}}\mathbf{A}^T \tag{7.4}$$

For a non-linear function $g(\mathbf{x})$, a linearized mapping can be obtained using first-order Taylor expansion represented in matrix form:

$$\mathbf{y} = g(\mathbf{x}) = \mathbf{g_0} + \mathbf{J}\mathbf{x} \tag{7.5}$$

where, $\mathbf{J}$ is the Jacobian of the transformation (g). The uncertainty in Eq. 7.5 only propagates in the linear operation $\mathbf{Jx}$, since $\mathbf{g_0}$ is constant. Then, the covariance propagation in the linearized system, similar to Eqn. 7.4 is expressed as:

$$\Sigma_{\mathbf{yy}} = \mathbf{J}\Sigma_{\mathbf{xx}}\mathbf{J}^T \tag{7.6}$$

7.1.3. Bearing Vectors and Tangent Space

The 2D measurements in Eq. 7.1 can be transformed in the 3D coordinates of camera frame $\mathbf{C}$ as:

$$\mathbf{r}_i^C = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \mathbf{K}^{-1} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \tag{7.7}$$

where, $\mathbf{K}^{-1}\mathbf{p}_i$ is the forward projection function (parameterized in m than in px) for the perspective equations.

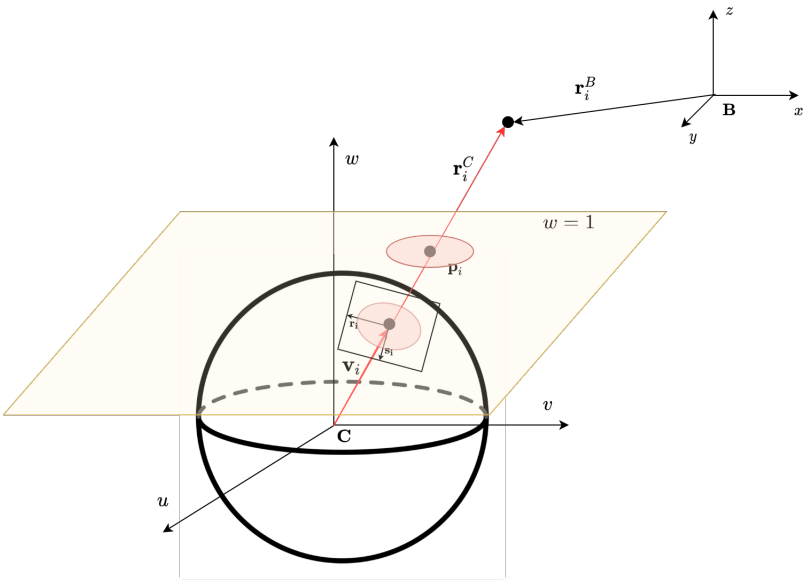


Figure 7.1: Visualization of projective geometry between the object body frame (**B**) and the camera frame (**C**), showing the projection plane and unit sphere, and the uncertainties associated with respective projections

The bearing vector $\mathbf{v_i}$ for the i^{th} point is then computed by spherical normalization of the vector in camera frame. Alternatively, it is a vector representing the point projection on a unit sphere around the origin of the camera frame as shown in Fig. 7.1.

$$\mathbf{v}_i = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \frac{\mathbf{r}_i^C}{|\mathbf{r}_\mathbf{i}^\mathbf{C}|} \tag{7.8}$$

The covariance propagated to the bearing vector space from the 2D image plane, is given as:

$$\Sigma_{\mathbf{vv}} = \mathbf{J} \begin{bmatrix} \Sigma_{\mathbf{pp}} & \mathbf{0}_{2x1} \\ \mathbf{0}_{1x2} & 0 \end{bmatrix} \mathbf{J}^T \tag{7.9}$$

where, $\mathbf{J}$ is the Jacobian of the transformation from image plane to bearing vectors is given in [59]:

$$\mathbf{J} = \frac{1}{||\mathbf{r}_i^C||} (\mathbf{I}_3 - \mathbf{v}\mathbf{v}^T) \tag{7.10}$$

From Eq. 7.9, it is clear that the covariance matrix associated with the bearing vector is rank deficient and therefore singular. Therefore, the MLE problem cannot be framed in terms of bearing vector residuals directly. For a valid MLE problem, it is necessary to obtain a minimal 2-dimensional representation of the associated uncertainty, in terms of a non-singular 2 x 2 covariance matrix. According to Forstner [59], this minimal representation is obtained by projecting the uncertainty in the null space of the bearing vector, which results in a 2D ellipse. The null space of the bearing vector $\mathbf{v_i}$ lies perpendicular to the bearing vector at the unit sphere i.e. its tangent space. The 2D coordinate system in the tangent space used to describe the uncertainty, is characterized in terms of two basis vectors $\mathbf{r_i}$ and $\mathbf{s_i}$:

$$null(\mathbf{v_i}^T) = [\mathbf{r_i} \quad \mathbf{s_i}] = \begin{bmatrix} r_1 & s_1 \\ r_2 & s_2 \\ r_3 & s_3 \end{bmatrix}_i = \mathbf{J}_{\mathbf{v}_r}(\mathbf{v_i}) \tag{7.11}$$

The null-space also represents the Jacobian $\mathbf{J}_{\mathbf{v}_r}$ of the transformation from tangent space to original bearing vector. Assuming orthonormal basis vectors $\mathbf{r}$ and $\mathbf{s}$, the transformation of bearing vector into the tangent space is given as

$$\mathbf{v}_r = \mathbf{J}_{\mathbf{v}_r}^T \mathbf{v} \tag{7.12}$$

The associated covariance matrix in the tangent space is a non-singular 2 x 2 matrix, given as:

$$\Sigma_{\mathbf{v}_r\mathbf{v}_r} = \mathbf{J}_{\mathbf{v}_r}^T \Sigma_{\mathbf{v}_r\mathbf{v}_r} \mathbf{J}_{\mathbf{v}_r} \tag{7.13}$$

Ideally, the transformation of the bearing vector (Eqn. 7.12) in its own tangent space is a zero vector. Therefore, $\mathbf{v}_r$ in Eq. 7.12 represents the residual in the tangent space. This allows pose estimation problem to be formulated in terms of residuals in the respective tangent spaces of the bearing vectors.

7.1.4. Projection Problem: Linear Formulation

The projection equation Eq. 2.4, can be transformed to represent observed points in the camera frame as:

$$\mathbf{r}_i^C = \lambda_i \mathbf{v}_i = \mathbf{R}^{BC} \mathbf{r}_i^B + \mathbf{t}^C \tag{7.14}$$

where, λ_i represents depth scaling of the i^{th} bearing vector.

Given a known set of 3D points in the body frame and a set of observed points in the image frame, Eq. 7.12 and Eq. 7.14 can be used to describe the residuals between projected and observed points in the tangent space as:

$$\begin{bmatrix} dr \\ ds \end{bmatrix} = \mathbf{J}_{\mathbf{v}_r}^T (\mathbf{R}^{BC} \mathbf{x}_i^B + \mathbf{t}^C) \tag{7.15}$$

With two equations per point, the total number of equations (as in Eq. 7.15) for n points are $2n$. When the relative pose is known, the residuals (dr and ds) in tangent space are zero. Considering zero residuals, n point projection problem is described by a homogeneous system of linear equations:

$$\mathbf{A}\mathbf{u} = \mathbf{0} \tag{7.16}$$

The column vector, $\mathbf{u}$, is a flattened vector with components of the pose matrix, such that:

$$\mathbf{u} = [R_{11}, R_{12}, R_{13}, R_{21}, R_{22}, R_{23}, R_{31}, R_{32}, R_{33}, t_1, t_2, t_3]^T \tag{7.17}$$

where, R_{ij}, $(i, j = 1, ..., 3)$ are elements of rotation transformation matrix $\mathbf{R_{BC}}$ and t_i are the elements of translation vector $\mathbf{t_C}$. Together, they completely describe the pose matrix from Eq. 2.4. While, the design matrix $\mathbf{A}$ derived from Eq. 7.15 is a $2n$ x 12 matrix, given as :

$$A = \begin{bmatrix} r_1 x_1^B & r_1 y_1^B & r_1 z_1^B & r_2 x_1^B & r_2 y_1^B & r_3 x_1^B & r_3 y_1^B & r_3 z_1^B & r_1 & r_2 & r_3 \\ s_1 x_1^B & s_1 y_1^B & s_1 z_1^B & s_2 x_1^B & s_2 y_1^B & s_3 x_1^B & s_3 y_1^B & s_3 z_1^B & s_1 & s_2 & s_3 \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ r_1 x_n^B & r_1 y_n^B & r_1 z_n^B & r_2 x_n^B & r_2 y_n^B & r_3 x_n^B & r_3 y_n^B & r_3 z_n^B & r_1 & r_2 & r_3 \\ s_1 x_n^B & s_1 y_n^B & s_1 z_n^B & s_2 x_n^B & s_2 y_n^B & s_3 x_n^B & s_3 y_n^B & s_3 z_n^B & s_1 & s_2 & s_3 \end{bmatrix} \tag{7.18}$$

where, r_i and s_i are the coordinates of the basis vectors in tangent space, while x^B, y^B, z^B are coordinate of the observed point in the object's body frame.

7.1.5. Linear Least Squares Solution

The solution to the linear system of equations is the one that minimizes the residuals in the tangent space. The normal equations for such a least squares problem, given the homogeneous system :

$$\mathbf{A}^T \mathbf{W} \mathbf{A} \mathbf{u} = \mathbf{N} \mathbf{u} = \mathbf{0} \tag{7.19}$$

If uncertainty is not taken into account, the weight matrix $\mathbf{W}$ is simply an identity matrix. If uncertainty of observations is taken into account, the weight matrix, $\mathbf{W}$, is composed of inverse of covariances as shown in Eq. 7.20. Note that MLPnP can be used with or without inclusion of keypoint covariance. The covariance information is always included in this work and the terms 'MLPnP with covariance' and 'MLPnP' is interchangeably used without further distinction.

$$\mathbf{W} = \mathbf{P} = \begin{bmatrix} [\Sigma_{\mathbf{v}_r \mathbf{v}_r}]_1^{-1} & . & . & . & \mathbf{0} \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ \mathbf{0} & . & . & . & [\Sigma_{\mathbf{v}_r \mathbf{v}_r}]_n^{-1} \end{bmatrix} \tag{7.20}$$

The solution to the given normal equations subject to $||\mathbf{u}|| = 1$, is obtained through Singular Value Decomposition (SVD) of matrix $\mathbf{N}$:

$$\mathbf{N} = \mathbf{U} \Sigma \mathbf{V}^T \tag{7.21}$$

The solution is the right singular vector in $\mathbf{V}$ associated with the smallest singular value in Σ. The elements in the estimated vector $\mathbf{u}$ are determined subject to the constraint $||\mathbf{u}|| = 1$, and are therefore scaled. Preliminary estimates of rotational transformation matrix and translation vector are:

$$\mathbf{R}' = \begin{bmatrix} R'_{11} & R'_{12} & R'_{13} \\ R'_{21} & R'_{22} & R'_{23} \\ R'_{31} & R'_{32} & R'_{33} \end{bmatrix} \quad ; \quad \mathbf{t}' = \begin{bmatrix} t'_1 \\ t'_2 \\ t'_3 \end{bmatrix} \tag{7.22}$$

In order to estimate the relative translation vector, the last three elements (see Eq. 7.17) of the solution vector need to be scaled appropriately. Urban et al. [58] in their original work use the orthonormality constraint for columns of the rotational transformation matrix- i.e. to scale all the 12 elements of $\mathbf{u}$ by a factor that makes the the columns of $\mathbf{R}'$ of unit norm. However, the nine elements of $\mathbf{R}'$ are independently estimated and thus the norm of each column vector is not the same. The scale factor ($\mathrm{s}_\mathbf{u}$) is then recovered from geometric mean of all the three columns of $\mathbf{R}'$ to estimate the camera relative position ($\mathbf{t}^C_{est}$):

$$\mathrm{s}_\mathbf{u} = \frac{1}{\sqrt[3]{||R'_{:1}||\,||R'_{:2}||\,||R'_{:3}||}} \tag{7.23}$$

$$\mathbf{t}^C_{est} = \mathrm{s}_\mathbf{u} . \mathbf{t}' \tag{7.24}$$

Finally, the rotation matrix ($\mathbf{R}^{\mathbf{BC}}_{\mathbf{est}}$) is estimated by recovering the nearest orthogonal matrix for the preliminary rotation matrix ($\mathbf{R}'$). The nearest orthogonal matrix is the one with the same right and left-singular vectors as in SVD of $\mathbf{R}'$, but with all singular values replaced with one- i.e. $\Sigma = \mathbf{I}_{3x3}$

$$\mathbf{R}^{BC}_{est} = \mathbf{U}_2 \mathbf{I} \mathbf{V}_2^T \quad s.t. \quad \mathbf{R}' = \mathbf{U}_2 \Sigma \mathbf{V}_2^T \tag{7.25}$$

This is equivalent to removing the scaling tranformation and preserving only rotation transformations in $\mathbf{R}'$ i.e. $\mathbf{U}$ and $\mathbf{V}$.

7.1.6. Non-linear Refinement

The linear estimate of relative state is further refined using Gauss-Newton optimization, similar to non-iterative solvers like EPnP [47]. Gauss-Newton algorithm is used to minimize the residuals in the tangent space given in Eq. 7.15. The pose vector in the Gauss-Newton optimization routine is reduced to a minimal representation. This is done by transforming the estimate of rotation matrix, $\mathbf{R}^{\mathbf{BC}}_{\mathbf{est}}$ to Rodriguez parameters [1]. The 6-vector pose representation is:

$$\mathbf{x}_{est} = [a_1 \; a_2 \; a_3 \; t_1 \; t_2 \; t_3]^T$$

with n tangent space residuals computed using Eq. 7.15:

$$E_{rs} = [dr_1 \; ds_1 \; .... \; dr_n \; ds_n]^T \tag{7.26}$$

The Gauss-Newton update to the pose vector ($\Delta\mathbf{x}$) of β is then given as:

$$\Delta\mathbf{x} = (\mathbf{J}^T_{x-rs} \, \mathbf{P} \, \mathbf{J}_{x-rs})^{-1} \, \mathbf{J}^T_{x-rs} \, \mathbf{P} \, E_{rs}$$

where, $\mathbf{J}_{x-rs}$ is the $2n \times 6$ Jacobian comprising of partial derivatives of pose vector with respect to the tangent space residuals (Eq. 7.15) and $\mathbf{P}$ is a $2n \times 2n$ matrix (Eq. 7.20). Urban et al. [58] suggest that linear solution is very close to the local minima, and therefore the optimization routine can find the solution in less that five iterations. This is favorable due to the computationally expensive inversion operation $(\mathbf{J}^T \, P \, \mathbf{J})^{-1}$ required for the Gauss-Newton refinement.

7.1.7. Pose Estimation Uncertainty

The MLE framework of the solver allows computation of uncertainty in the estimated pose. The uncertainty of the estimated pose is derived using uncertainty propagation for indirect observations in least squares formulation. The co-factor matrix for a normal equation (similar to Eq. 7.19), the cofactor-matrix ($\mathbf{Q}_{uu}$) that contains covariance elements of the vector $\mathbf{u}$ is the inverse of coefficients matrix, given as:

$$\mathbf{Q}_{uu} = N^{-1} \tag{7.27}$$

Then, the covariance can be computed as following:

$$\Sigma = \hat{\sigma}_0^2 \, N^{-1} \tag{7.28}$$

where, $\hat{\sigma}_0$ is the estimated variance factor for indirect observation. The reader is referred to [158] for extensive review of this aspect. Urban et al. [58] adapt this to MLPnP to compute the covariance of the estimated pose as:

$$\Sigma_{\mathbf{at}} = \sigma_0^2 \, \Sigma_{\mathbf{at}} \tag{7.29}$$

$$\Sigma_{\mathbf{at}} = (\mathbf{J}^T_{x-rs} \, \mathbf{P} \, \mathbf{J}_{x-rs})^{-1} \quad ; \quad \sigma_0^2 = \frac{d\mathbf{r}^T \, \mathbf{P} \, d\mathbf{r}}{2n-6} \tag{7.30}$$

where, $d\mathbf{r} = [dr_1, \, ds_1 \; ... \; dr_n, \, ds_n]$ is $2n \times 1$ sized vector of stacked tangent space residuals.

[1] NOTE: MLPnP toolbox from [58] uses classical Rodrigues parameters

7.1.8. Evaluation Metrics

Translation accuracy for a pose estimate is evaluated using translation error vector defined as:

$$\mathbf{E}_T = |\mathbf{t}^C_{est} - \mathbf{t}^C_{true}| \tag{7.31}$$

It can alternatively be used in terms of percentatage, which is simply: $100.||\mathbf{E}_T||$.

The Rotation accuracy for a pose estimate is evaluated as the axis-angle error around the Euler axis, defined as:

$$E_R = 2\cos^{-1}(q_{e,0}) \tag{7.32}$$

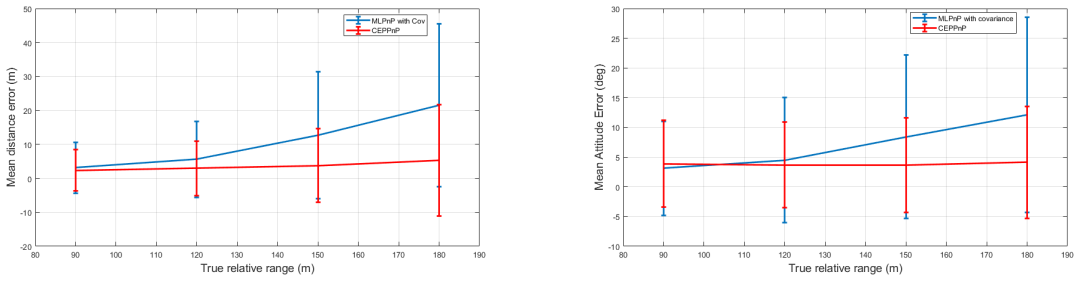
where, $q_{e,0}$ is the scalar element of the 4-vector error quaternion defined as

$$\mathbf{q}_e = \mathbf{q}_{true} \otimes \mathbf{q}^{-1}_{est}$$

using quaternion multiplication ($\otimes$) and quaternion inverse(q^{-1}). For convenience, the expression is described in quaternion parameterization. See Appendix B for details on attitude representations and transformations to alternative representations.

7.2. Preliminary Evaluation

Preliminary evaluation of the solver seeks to verify the performance promised in the original work [58] by employing it for uncooperative spacecraft pose estimation. To establish a reference for performance comparison, CEPPnP, the only other covariant solver is also included to estimate the pose for the same set of images. From the verification results in Urban et al [58], MLPnP is expected to have better performance than CEPPnP and other solvers, while being computationally efficient. However, on employing the MLPnP solver to estimate pose from network keypoint predictions of the SPEED and *Envisat-1* dataset, it exhibits a significant deterioration of performance in the pose estimate. The error magnitude is more pronounced in the *Envisat-1* dataset in which the target is imaged from 90m, 120m, 150m and 180m. It is observed that the MLPnP pose solution degrades in accuracy with increasing relative distance. In comparison to the CEPPnP solver, the pose estimates from the MLPnP solver tend to be significantly worse. Figs. 7.2(a) and 7.2(b) show the mean and standard deviation of relative attitude error in axis-angle representation and relative distance error for target for MLPnP and CEPPnP. The results are generated for 2000 random images from *Envisat-1* images using the keypoint (n = 16) predictions from a trained HRNet KD network, for a preliminary analysis.



(a) Relative Position Error

(b) Relative Attitude Error

Figure 7.2: Pose estimation statistics on *Envisat-1* using keypoint predictions from HRNet-W32 trained model

The pose estimation results are found to be incoherent with the results presented in the original work. Since *Envisat-1* is restricted in range and pose distribution, it is necessary to assess the solvers in detail using a broader regime of pose and noise conditions, such that the results are conclusive and generalizable. In order to assess and verify this in detail, the MATLAB toolbox [2] used in [58] was adapted and tested for higher relative distances on random set of 3d points. It was discovered that the default test set in Urban et al. [58] used random 3d points with distance along the boresight (z^C) between 4m and 8m, for which the solver shows superior performance. However, when the relative range in increased, performance of MLPnP solver

[2] MLPnP matlab toolbox: `https://github.com/urbste/MLPnP_matlab_toolbox/`

Test case	3D object points	σ (px)	x^C range (m)	y^C range (m)	z^C range (m)
default	random	1, 3, 5, 7	[-2, 2]	[-2, 2]	[4, 8]
A	random	1, 3, 5, 7	[-5, 5]	[-5, 5]	[10, 50]
B	random	1, 3, 5, 7	[-5, 5]	[-5, 5]	[50, 100]
C	random	1, 3, 5	[-5, 5]	[-5, 5]	[100, 200]

Table 7.1: MLPnP evaluation: Monte Carlo test case for random 3d object points

Test case	3D object points	σ (px)	t_x^C range (m)	t_y^C range (m)	t_z^C range (m)	samples
E1	Envisat model	3	[-20, 20]	[-20, 20]	[50, 200]	20000

Table 7.2: MLPnP evaluation: Monte Carlo test case for Envisat spacecraft model

degrades. Fig. 7.3 shows the performance (accuracy) of MLPnP solver in comparison with an exhaustive list of other solvers exactly as presented in [58]. The pose estimation tests are done for fixed number of points (n=20) and varying standard deviations (σ) for the pixel measurement error. Each sample is a random set of n points is generated in ranges of x,y and z axes in the camera frame. Each of these samples is associated to a true relative pose, such that center of the hypothetical object is the geometric center of randomly generated points and its orientation is characterized by three randomly generated Rodrigues parameters. This truly characterizes an object viewed by a camera in a generic sense, enabling conclusive analysis of solver performance. The randomly generated object points are then projected on to the image plane for a simple projective camera model, after which the Gaussian noise is added to each image keypoint. For each value of standard deviation for the Gaussian pixel noise, 500 such noisy samples are generated and fed to all the solvers. In order to assess the effect of relative range, the position of object points in the boresight direction is varied up to 200 m. In order to get a sense of degradation, the boresight ranges are divided into 3 cases- A,B and C as specified in Table. 7.1.

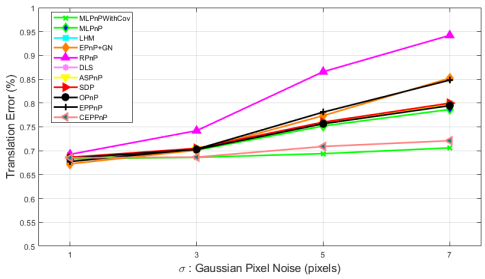
It is clear that with increasing relative distance, MLPnP severely degrades in performance in comparison to almost all other popular solvers. This conclusion remains invariant when the x and y ranges are varied. Therefore, the utility of MLPnP solver at higher relative distances to the target spacecraft is unsuitable for the navigation framework. To reinforce this inference, an evaluation of MLPnP solver is also conducted with Envisat's 3D model as the object and 16 keypoints characterized in the *Envisat-1* dataset. In contrast to random points (imitating random objects of an approximate size) in the previous evaluation, the keypoints in the body frame are fixed. k random pose vectors are generated from a normal distribution. Rotation is characterized by random 3-vector of Rodrigues parameters from a normal distribution in $SO(3)$. Translation is characterized by a 3-vector where each component is uniformly distributed in respective ranges as in Table 7.2. Using the random pose, model points are projected onto the image plane using the projective camera model of the *Envisat-1* dataset, and gaussian noise is added to the projected image points. A total of ten thousand random poses are used to evaluate the solver's performance for the Envisat spacecraft model. CEPPnP is used as a reference for comparison, since it is the only other covariant solver relevant to the navigation framework design. Therefore, other solvers are excluded, wherever not relevant.

Pose estimation results in Fig. 7.4 extend the results of Fig. 7.2 to uniformly distributed relative positions in the range of 50m-200m, in contrast to 3 discrete positions (90m, 120m, 150m) in *Envisat-1* image dataset. Steep increase in mean position error is especially drastic for MLPnP solver, compared to CEPPnP. Further, Fig.7.5 show how the pose estimation error varies with pixel location error at various relative separation. The sensitivity of MLPnP solver to pixel noise at higher relative range is clearly visible. The surface plots show that while MLPnP accuracies can be better or equivalent to CEPPnP at smaller relative distances, CEPPnP is clearly superior as it exhibits significantly smaller error in position and attitude at larger relative distances, even in the presence of pixel noise.

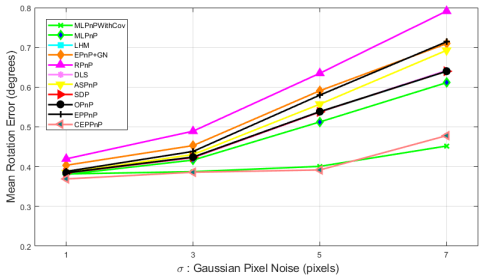
Preliminary evaluation provides clear indications of unsuitablity of MLPnP for the navigation framework.

7.3. Algorithm Modification: Scale Recovery

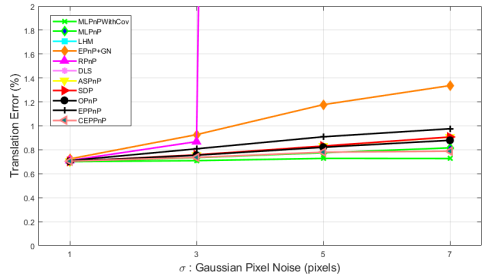
The dissection of the problem and debugging the MLPnP codebase reveals that the degrading performance of the solver results from a bad linear solution Eqs. 7.24 and 7.25. Specifically, the scale recovering step (Eq.



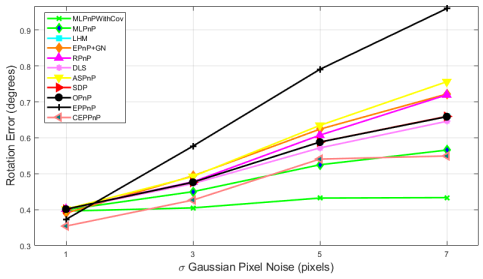
(a) Test Case-default: Mean Translation Error



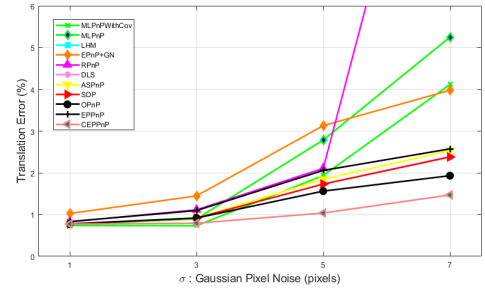
(b) Test Case-default: Mean Rotation Error



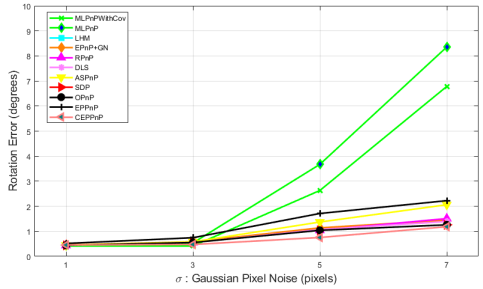
(c) Test Case-A: Mean Translation Error



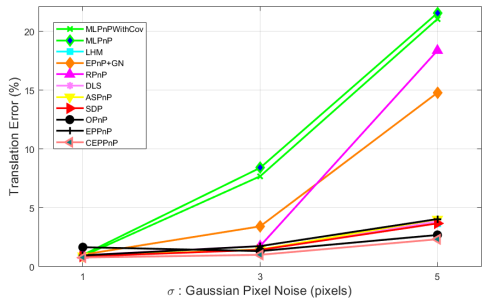
(d) Test Case-A: Mean Rotation Error



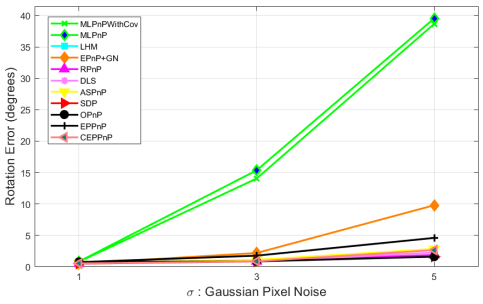
(e) Test Case-B: Mean Translation Error



(f) Test Case-B :Mean Rotation Error

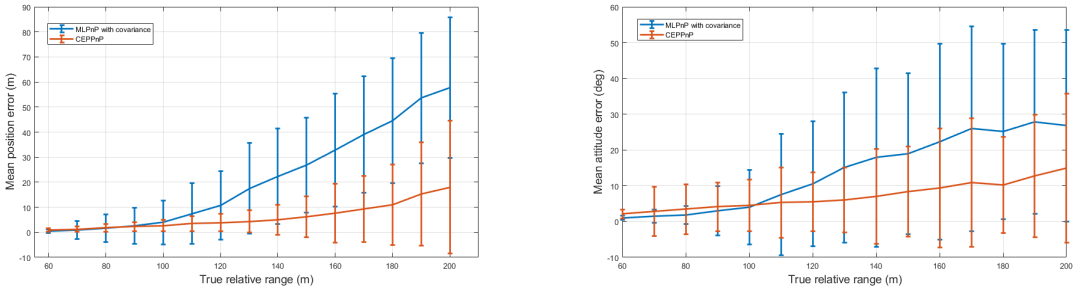


(g) Test Case-C: Mean Translation Error



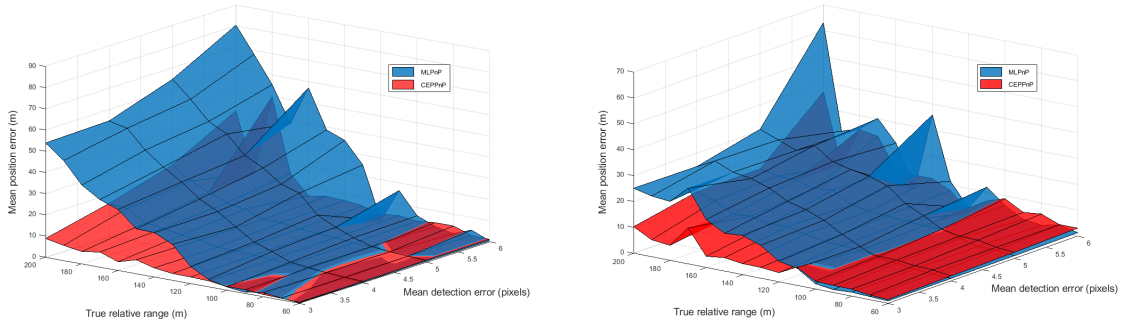
(h) Test Case-C :Mean Rotation Error

Figure 7.3: Pose Solver Performance Evaluation Results



(a) Relative Position Error (b) Relative Attitude Error

Figure 7.4: Pose estimation statistics on Test case E-1 for *Envisat-1* keypoint predictions



(a) Position Error (b) Attitude Error

Figure 7.5: Pose Estimate Test Case-E1: Error variation with relative range and pixel noise

7.24) from the preliminary rotational transformation matrix ($\mathbf{R}'$), using column orthogonality constraint. It is observed that the linear estimates (scaled) of translation vector ($\mathbf{t}'$), does points in the right direction, as noted in the original work [58]. However, the scale recovery step using the geometric mean of the column vector norms (Eq. 7.25) breaks down at higher relative distances. The geometric mean technique based on preliminary rotation matrix ($\mathbf{R}'$) for scale recovery is observed to be very crude. This is because, while a direction cosine matrix is a non-singular representation, it is subject to six constraints. Firstly, it is not clear why only three (column) constraints are exploited. Secondly, the independently determined elements of the preliminary rotation matrix do not guarantee column norms close to each other and therefore the method would be effective only for specific relative geometry. The method works well when all three column norms are close to the true value. On the other hand, when a small target is observed from a larger distance, the column norms can vary greatly from each other and drive the geometric mean of the three columns away from the true scale. As a result the bad scale recovery results in an inaccurate translation vector estimate. This in turn forces the Gauss-Newton optimization routine to find a pose solution with minimum residuals close to the linear solution. This result in further divergence of the attitude estimate.

In the preliminary analysis, the divergence between the true and estimated scale factor is assumed to be correlated to the nullspace values of bearing vectors - i.e. the basis vectors $\mathbf{r}_i$ and $\mathbf{s}_i$ (Eq. 7.12) corresponding to each keypoint. When the target object is far along the boresight direction, as is the case for the spacecraft in *Envisat-1* dataset, the tangent space is almost perpendicular to the Z-axis. In other words, the basis vectors lie mostly in X-Y plane with Z-components (r_3 and s_3) being negligible. In this case, the norm of the third column of $\mathbf{R}'$ in Eq. 7.22 ($R_{:3}$) diverges the most from the true scale factor. Therefore, it would be logical to discard the corresponding column norm. It is discovered that for such cases, discarding the corresponding columns result in better scale recovery. Alternatively, to generalize this for object not specifically along boresight, a correlated weighting scheme could be formulated which relates tangent space bearing vector elements with columns of $\mathbf{R}'$. Such a weighting scheme was found to be more nuanced and not generally applicable for all domain conditions in a generic pose estimation problem. However, constraining the 9 independently

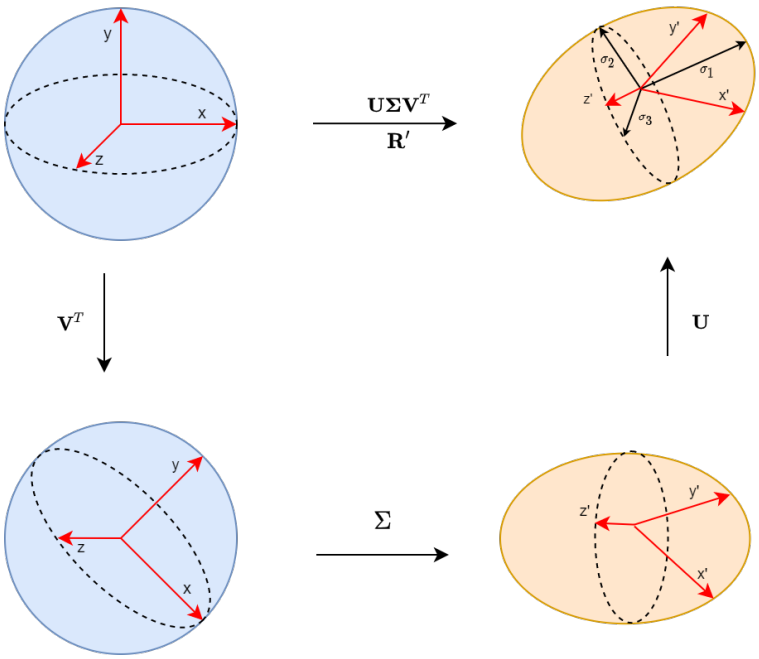


Figure 7.6: Illustration of SVD of composite scaling and rotation transformation on a unit sphere in $\mathbb{R}^3$

estimated elements of $\mathbf{R}'$ with unit-norm columns of other constraints of a DCM remains ineffective in the analyses. Additionally, the reasoning behind weighting based on tangent space bearing vector is not realized.

It is also noted that the approaches discussed thus far remain disconnected to the final step of recovering the nearest orthogonal transformation matrix for rotation. In principle, the appropriate "rescaling" of the 9 elements of $\mathbf{R}'$, occurs in Eq. 7.25 using SVD of $\mathbf{R}'$. As noted earlier, the nearest orthogonal matrix is obtained by removing the scaling effect in $\mathbf{R}'$ i.e replacing the singular values with unity ($\Sigma = \mathbf{I}$) in the decomposition. It is assumed that this step is relevant to a retrieving the scaling factor that is consistent with the reverse scaling with which the nearest orthogonal matrix computed in Eq. 7.25.

In effect, SVD decomposes a combined rotation and scaling transformation into orthogonal matrix of left-singular vectors ($\mathbf{U}$), a diagonal matrix with singular values (Σ) and an orthogonal matrix of right-singular vectors ($\mathbf{V}^T$). A geometric interpretation of SVD is illustrated in Fig. 7.6 for a given unit sphere in $\mathbb{R}^3$ undergoing a transformation $\mathbf{R}'$. The transformed ellipsoid is such that the semi-axis of the ellipsoid represent the singular values in the diagonal matrix Σ from SVD. The singular value matrix (Σ) is such that it contains singular values along the diagonal in the decreasing order.

$$\sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \quad ; \quad \sigma_1 \geq \sigma_2 \geq \sigma_3$$

This geometric interpretation allows articulation of singular values and their transformation in $\mathbb{R}^3$. Now, consider Eq. 7.25, where the left and right singular vector matrices are used to find the nearest orthogonal matrix. This results from replacing all singular values with unity. This is illustrated in Fig. 7.7 for an analogous unit sphere, where a transformation ($\mathbf{R}'$) that results into the ellipsoid, can be transformed into a unit sphere with orthogonal transformation, $\mathbf{UV}^T$, such that the basis vectors are closest in terms of their Frobenius norm. In effect, when the singular values are replaced with unity, the coordinate space is scaled by a linear combination of the singular values. This motivates the question - Can singular values be used to recover the scale factor for the translation vector?

Reconsider the 12-vector linear solution that determines the preliminary (unscaled) rotation matrix ($\mathbf{R}'$) and translation vector ($\mathbf{t}'$). Let's assume an arbitrary coordinate system exists in which the preliminary trans-

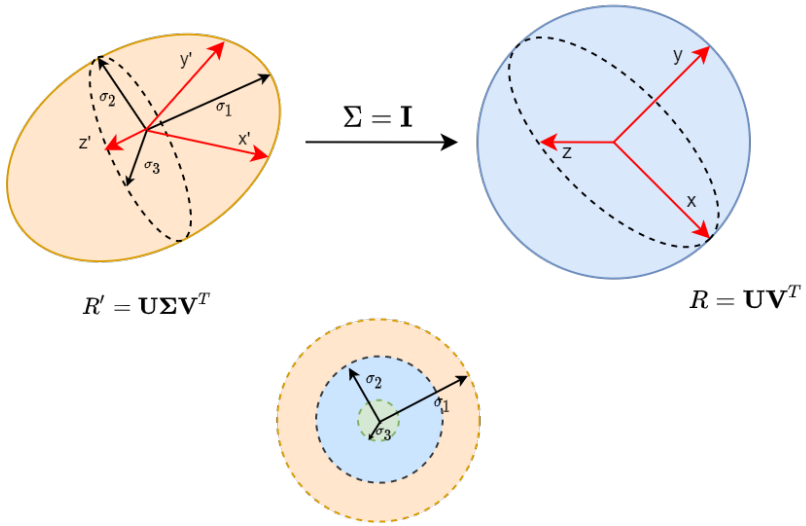


Figure 7.7: Illustration of nearest orthogonal matrix transformation composed of left and right singular vectors of $\mathbf{R}'$

lation vector ($\mathbf{t}'$) is defined that points in the right direction, as noted earlier. Then, the SVD operation that is used to derive the nearest orthogonal matrix, also contains representative scale information by which a vector in the arbitrary coordinate system are scaled with respect to the original frame (camera frame). Since, the arbitrary coordinate system that contains $\mathbf{R}'$ and $\mathbf{t}'$ is not defined, it is not possible to derive linear combinations of singular values that result in the exact 3D scaling. Instead, the proposed approach is to use the intermediate singular value (σ_2) from SVD of $\mathbf{R}'$, which is found to minimize the translational error. By obtaining a more accurate relative position estimate, the modification also results in improvement of relative attitude estimate during the non-linear refinement.

Therefore, the scale recovery modification to the algorithm is to eliminate the step in Eq. 7.24 that uses geometric mean of column norms and derive scaling factor from singular value decomposition of $\mathbf{R}'$.

Given $\mathbf{R}'$ and its SVD:

$$\mathbf{R}' = \mathbf{U}_2 \Sigma_2 \mathbf{V}_2^T \tag{7.33}$$

$$\Sigma_2 = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \quad ; \quad \sigma_1 \geq \sigma_2 \geq \sigma_3 \tag{7.34}$$

The rotational transformation matrix ($\mathbf{R}^{BC}$) is estimated as the closest orthogonal matrix (w.r.t Frobenius norm) to $\mathbf{R}'$, as in Eq. 7.25:

$$\mathbf{R}^{BC} = \mathbf{U}_2 \mathbf{V}_2^T \tag{7.35}$$

However, the scale factor that estimates the translation vector $\mathbf{t}^C$ from $\mathbf{t}'$ is the second singular value in Σ_2, such that:

$$\mathbf{t}^C = \frac{1}{\sigma_2} \mathbf{t}' \tag{7.36}$$

This pose estimate is then refined in the Gauss-Newton optimization routine, as earlier. Since, the modification to the algorithm only involves utilizing the singular values obtained from the SVD of $\mathbf{R}'$, there is no extra computational expense added to the algorithm. Therefore, the computational cost benefit of the algorithm remains consistent as stated in Urban et al [58]. In fact, eliminates the norm and geometric mean computations of Eq. 7.24, which reduces computation requirement, although very insignificantly.

7.4. Verification

The modification made to the MLPnP algorithm is tested extensively and verified. First, a very preliminary comparison is made to demonstrate effective scale recovery of the modified algorithm in selective cases as

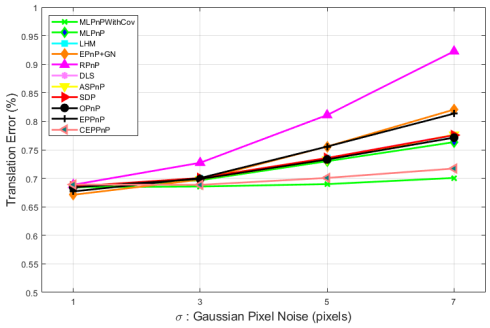
				MLPnP [58]		Modified MLPnP (Ours)	
No.	True Position ($\mathbf{t}^C$)	Pixel noise (σ_{px})	Expected scale factor (λ_s)	Estimated scale factor (λ_s)	Position Error ($\lvert\Delta\mathbf{t}^C\rvert$)	Expected scale factor (λ_s)	Position Error ($\lvert\Delta\mathbf{t}^C\rvert$)
	(m)	(pixels)			(m)		(m)
1	(0,0,120)	3	0.0083	0.0093	13.26	0.0086	4.31
2	(0,0,150)	3	0.0067	0.0101	51.45	0.0064	5.35
3	(0,0,180)	3	0.0056	0.0079	54.41	0.0052	11.51
4	(20,-20,90)	3	0.0106	0.0122	8.1	0.0107	0.85
5	(50,50,140)	3	0.0064	0.0120	73.32	0.0060	9.23
6	(60,-10,140)	3	0.0058	0.0080	46.62	0.0057	4.79

Table 7.3: Preliminary assessment and comparison of the scale recovery modification

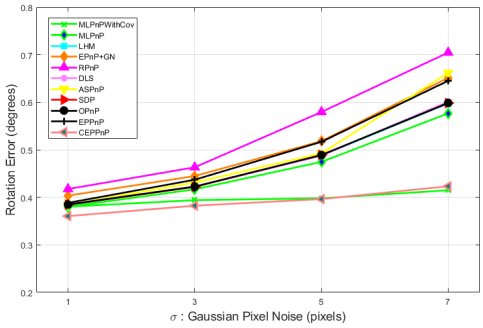
compared to the original algorithm. Table 7.3 shows the results for countable cases of envisat model at a considerable relative distance from the servicer. It is clear that the modified algorithm is more effective at scale recovery, which results in lower estimation error for relative position. It is observed that in singular cases of favorable geometry, the scale recovery step of the original algorithm performs well. However, in systematic comparative analysis, it remains largely inferior to the modified algorithm in scale recovery.

Given the preliminary quantitative indication of accurate scale recovery and lower translation error, the modification is further verified in the standard evaluation framework of MATLAB toolbox used in Urban et al [58]. The test cases in Table 7.1 are repeated for the MLPnP algorithm with scale recovery modification. The results for the modified MLPnP are shown in Fig. 7.8, which draw direct comparison to the results of the original MLPnP algorithm in Fig. 7.3. After modifications, the algorithm maintains the performance of pose estimation at smaller relative distances, while drastically improving the translation error at larger distances (Test cases B and C). The scale recovery modification also results in the improvement of the rotation error performance by close to 50%. However, the rotation error performance in test case C, with the largest relative distances, stands out in comparison to other solvers. Figs. 7.9 & 7.10 provide additional insight into test case C in terms of median of translation and rotation error in test case-C, before and after scale recovery modification. It is evident that the improvement in pose estimation is significant as median of modified MLPnP with covariance is at par with the best performing method (CEPPnP). From the given results, it can be concluded that while MLPnP produces a rotation estimate, the sensitive cases with higher noise can result in considerably high rotation error.

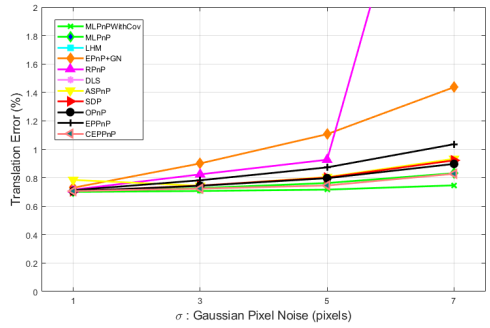
The results of the modified MLPnP algorithm show its effectiveness in overcoming sensitivity to large relative distances. More importantly, it provides verification of the modified algorithm and it's performance improvement in a truly generic sense for a pose estimation problem in the same framework as the original work [58]. To further validate it's performance specific to the spacecraft pose estimation problem, the modified algorithm is tested and compared with CEPPnP.



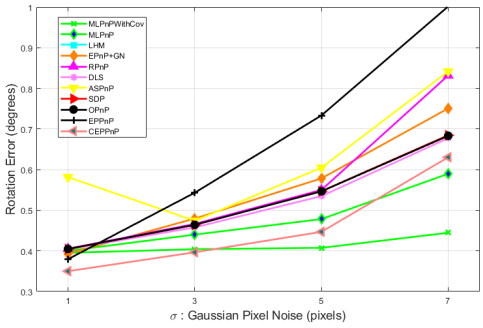
(a) Test Case-default: Mean Translation Error



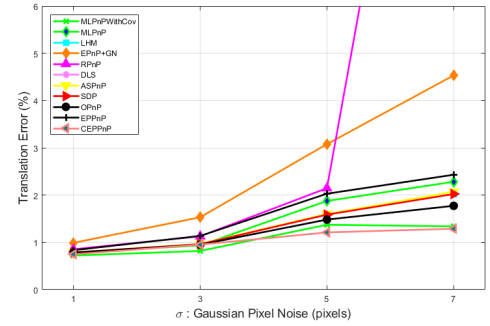
(b) Test Case-default: Mean Rotation Error



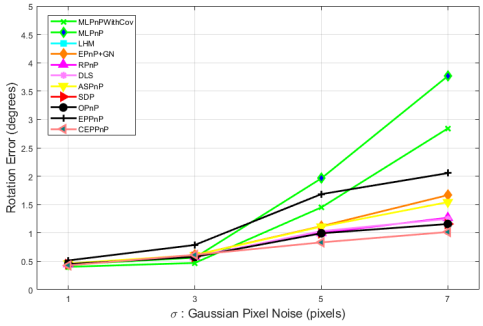
(c) Test Case-A: Mean Translation Error



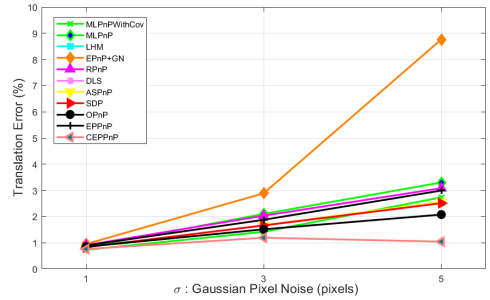
(d) Test Case-A: Mean Rotation Error



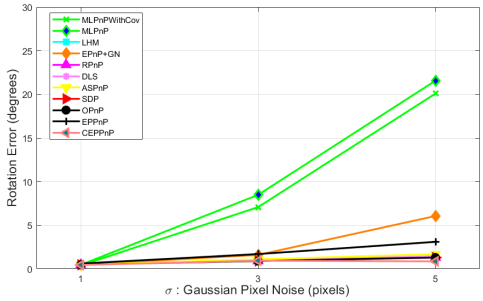
(e) Test Case-B: Mean Translation Error



(f) Test Case-B :Mean Rotation Error

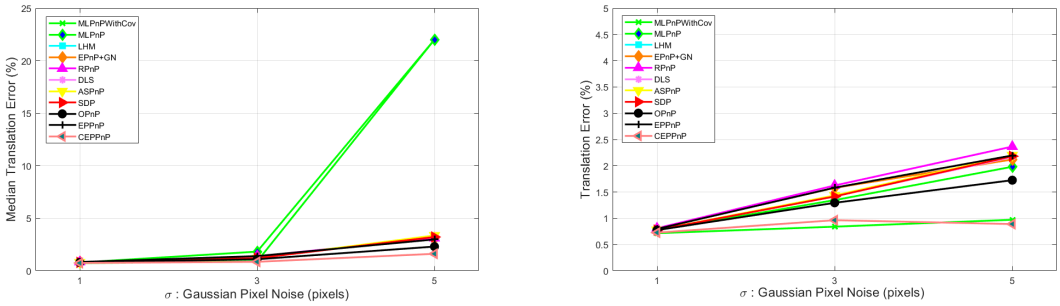


(g) Test Case-C: Mean Translation Error



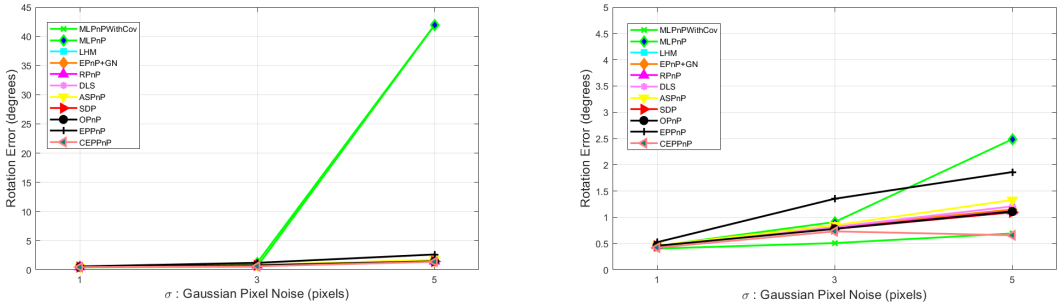
(h) Test Case-C :Mean Rotation Error

Figure 7.8: Pose Solver Performance Evaluation Results



(a) Original algorithm (b) Modified MLPnP (Ours)

Figure 7.9: Median Translation Error Performance on Test Case-C

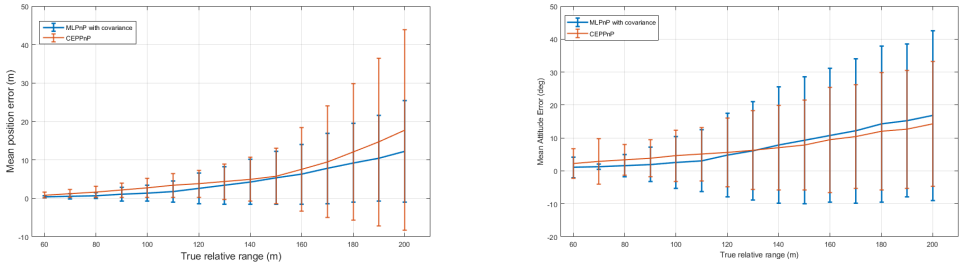


(a) Original algorithm (b) Modified algorithm (Ours)

Figure 7.10: Median Rotation Error Improvement on Test Case-C

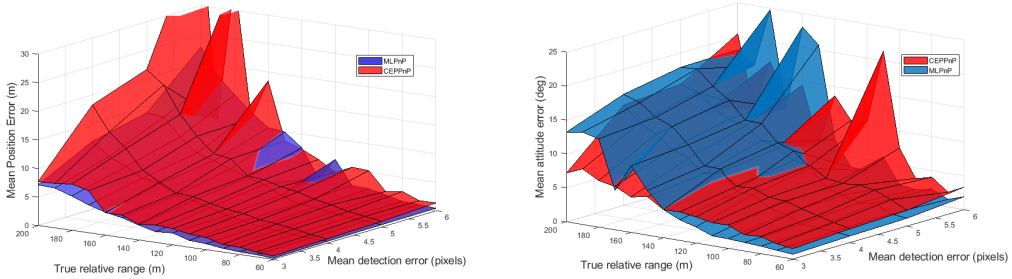
Comparison: CEPPnP vs modified MLPnP

The verification of MLPnP solver with the scale recovery modification is further reinforced by testing it for Envisat case under specific investigation in this work. The test case E1 in Table 7.2 is repeated with the modified algorithm. Compared to the results in Fig. 7.4 and 7.5, the modified algorithm demonstrates significant improvement as shown in Fig. 7.11. In contrast to CEPPnP, the modified MLPnP maintains translation error less than or almost 10% of the relative range upto 200m. In the rotation erro, the modified MLPnP degrades in performance beyond 120m, after which the mean error remains close to that of CEPPnP, but shows higher standard deviation in attitude estimate error. Simultaneous error variation with respect to true relative range and mean detection error in Fig. 7.12 provide further clarity on performance comparison between CEPPnP and the modified MLPnP solvers. While the modified MLPnP is consistently superior in translation, the rotation error performance is also superior to CEPPnP solver's below 120m. More importantly, the modified MLPnP proves to be more robust to increasing pixel noise when the same Monte Carlo analysis (test case E11 in Table 7.2) is done for a higher standard deviation of 5 pixel. The results in Figs. 7.13 and 7.14 show clear superiority of the modified MLPnP solver against CEPPnP. Given the exhaustive set of verification tests and supporting results, the modification made to the MLPnP solver is verified and the modified solver is deemed suitable for its applicability to the navigation framework concerned in this work, due to it's performance and robustness to separation range and pixel noise.



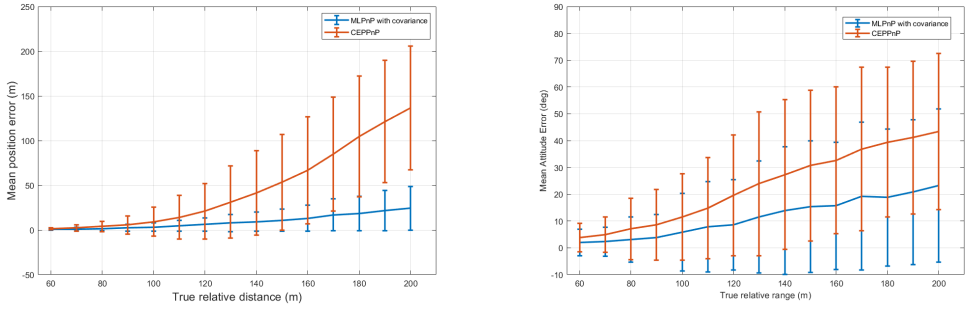
(a) Relative Position Error (b) Relative Attitude Error

Figure 7.11: Pose estimation performance comparison in presence of pixel noise (σ = 3 px)



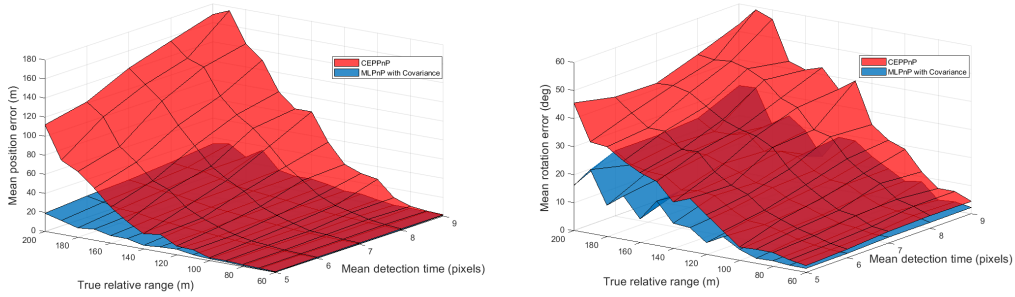
(a) Relative Position Error (b) Relative Attitude Error

Figure 7.12: Pose estimation comparison: Error variation with relative range and pixel noise (σ = 3 px)



(a) Relative Position Error (b) Relative Attitude Error

Figure 7.13: Pose estimation performance comparison in presence of pixel noise (σ = 5 px)



(a) Relative Position Error (b) Relative Attitude Error

Figure 7.14: Pose estimation comparison: Error variation with relative range and pixel noise (σ = 5 px)

7.5. Conclusions

The pose solver component of the navigation framework is presented in detail. A novel solver that employs maximum likelihood estimation to solve the PnP problem has been described. The merits of using the MLPnP solver are highlighted for robust pose estimation and preservation of uncertainty information in the process. More importantly, the shortcomings of the original algorithm are revealed and investigated in detail. Further, a modification is proposed to overcome the problems and improve its performance, followed by an extensive verification. The modified solver shows significant improvement over the original and is instrumental in enabling a loosely-coupled state estimation with pose uncertainty computed in the loop, by allowing the computation of covariance of the pose pseudo measurements in the navigation loop.

8

State Estimator

In order to support the on-board guidance and control functions, the vision-based navigation system needs to provide filtered state estimates. In the previous section, the MLPnP pose solver was described which estimates the relative pose of the target and the associated pose uncertainty from the keypoint predictions. The state estimator subsequently uses these pose estimates to provide filtered state estimates subject to the dynamics of relative motion. The estimated state vector consists of position, velocity, attitude and angular velocity components of the target relative to the servicer. The use of state estimator or filter ensures that the state estimates are available at a desired frequency in the GNC loop. This also ensures availability of the relative state information in between measurements, when the measurement frequency is lower due to the computation requirements of the pose estimation pipeline. Additionally, it can continue providing state estimates during short intervals when the target is not visible or out of view, due to operational constraints.

In this section, the Multiplicative Extended Kalman Filter design is reviewed that allows preliminary analysis on integration of a state estimator to the elements of pose estimation pipeline discussed in the previous sections. A loosely-coupled approach is adapted for the filter which uses the pose estimate and heatmap-derived pose uncertainty in it's measurement model. MEKF, an extension of EKF common to attitude estimation is utilized that adds an additional reset step for attitude parameterization within the filter. A loosely coupled MEKF uses linearized dynamics of relative motion in the prediction step and the estimated pose (and its covariance) in the correction step. In this chapter, a description of filter design is presented. A simple design with reduced dynamics is considered imperative for preliminary verification of the loosely-coupled approach that takes in the uncertainty from the MLPnP solver. Further, it is assumed that the target orbit is circular and the absolute position of the servicer spacecraft is known with certainty. These simplifications are motivated by other similar works [12, 60] on vision-based navigation for a preliminary analysis. Once verified, the fidelity of the relative dynamics, sensor model complexity and extensions to non-linear filters can be incrementally studied in the future.

8.1. Extended Kalman Filter Equations

For the relevance of estimation onboard a digital computer, a discrete time EKF is reviewed here. For further details, readers are referred to standard texts in optimal estimation like Simon [159] and Crassidis and Junkins [160].

For a linearizable dynamical system under observation, the system dynamics and a measurement model in discrete time can be written in discrete form as:

$$\mathbf{x}_k = \Phi_k \mathbf{x}_{k-1}, + \mathbf{w}_k \tag{8.1}$$

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k, + \boldsymbol{\nu}_k \tag{8.2}$$

where, $\mathbf{H}_k$ is a matrix expressing the measurement or observation model, $\Phi_\mathbf{k}$ is the state transition matrix $\mathbf{x}_k$ is the state vector and $\mathbf{z}_k$, the measurement vector at discrete time interval $k = 1, 2, \ldots n$. The process noise in state propagation, $\mathbf{w}_k$ and the measurement noise ν_k are assumed to be zero-mean Gaussian noise vectors

with covariance Q_k (dynamic) and R_k (measurement) respectively, such that:

$$\mathbf{w}_k \sim \mathcal{N}(0, Q_k) \qquad and \qquad \boldsymbol{\nu}_k \sim \mathcal{N}(0, R_k)$$

The filter is intialized with a random state vector $\hat{\mathbf{x}}_0^+$ such that:

$$\begin{aligned} \hat{\mathbf{x}}_0^+ &= E[\mathbf{x}_0] \\ \mathbf{P}_0^+ &= E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T] \end{aligned} \tag{8.3}$$

where, $\mathbf{P}$ represents the state covariance for the Gaussian distribution of the state estimate.

The filter loop consists of prediction (time update) and correction (measurement update) steps. The prediction step computes apriori estimates of the state ($\hat{\mathbf{x}}_k^-$) and the associated covariance ($\mathbf{P}_k^-$), given by prediction equations:

$$\begin{aligned} \hat{\mathbf{x}}_k^- &= \Phi_k \mathbf{x}_{k-1}^+ \\ \mathbf{P}_k^- &= \Phi_\mathbf{k} \mathbf{P}_{k-1}^+ \Phi_k + Q_k \end{aligned} \tag{8.4}$$

Subsequently, given the availability of measurements, the state estimate is updated with the Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^-, \mathbf{H}_k^T + R_k)^{-1} \tag{8.5}$$

to find the (sub-) optimal update to the apriori estimate using the measurement update equations:

$$\begin{aligned} \hat{\mathbf{x}}_k^+ &= \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^-) \\ \mathbf{P}_k^+ &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- \end{aligned} \tag{8.6}$$

It is necessary that the covariance matrices- $\mathbf{P}_k$, Q_k and R_k be symmetric and positive definite. Practically, truncation and other round-off errors can violate these conditions, leading to numerical instability in filtering. It is typical to use covariance update given in Eq. 8.7 to overcome these practical shortcomings.

$$\begin{aligned} \mathbf{P}_k &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^T + \mathbf{K}_k R_k \mathbf{K}_k^T \\ \mathbf{P}_k^+ &= \frac{1}{2}\left(\mathbf{P}_k + \mathbf{P}_k^T\right) \end{aligned} \tag{8.7}$$

8.2. Dynamics of Relative Motion

8.2.1. Translational Dynamics

Relative translation motion between the spacecraft can be described in the Local Vertical Local Horizontal (LVLH) frame of reference. LVLH frame is centered at the center of mass of the spacecraft in the reference orbit. For the problem at hand, the reference orbit is considered to be that of the servicer, since the absolute state of the servicer is assumed to be known. The X-axis of LVLH points along the radial vector (the local vertical), the Z-axis points perpendicular to the orbit plane and the Y-axis completes the right-hand triad (along the local horizontal).

For close-proximity scenario concerned in this work, the equations of motion can be linearized subject to:

$$\frac{||\mathbf{r}_T - \mathbf{r}_S||}{||\mathbf{r}_S||} << 1 \tag{8.8}$$

where, $\mathbf{r}_T$ and $\mathbf{r}_S$ are absolute positions of the target and servicer spacecraft respectively. Then, a system of first-order equations describing the relative motion between the servicer and target spacecraft is given as following ([161], Chapter 7):

$$\begin{aligned} 0 &= \delta\ddot{x} - \left(\frac{2\mu}{r^3} + \frac{h^2}{r^4}\right)\delta x - \left(\frac{2(\dot{\mathbf{r}}\cdot\mathbf{r})\,h}{r^4}\right)\delta y + 2\left(\frac{h}{r^2}\right)\delta\dot{y} \\ 0 &= \delta\ddot{y} - \left(\frac{\mu}{r^3} - \frac{h^2}{r^4}\right)\delta y - \left(\frac{2(\dot{\mathbf{r}}\cdot\mathbf{r})\,h}{r^4}\right)\delta x - 2\left(\frac{h}{r^2}\right)\delta\dot{x} \\ 0 &= \ddot{z} + \frac{\mu}{r^3} \end{aligned} \tag{8.9}$$

where, μ is the gravitational constant, h is the magnitude of orbital angular momentum and r denotes the absoluted distance of the servicer ($r = ||\mathbf{r}_S||$), with subscript removed for clarity and generality of expression. When the target spacecraft is assumed to be the reference, the system of equations apply for $r = ||\mathbf{r}_S||$.

The aforementioned equations of relative motion can be further simplified by assuming the reference orbit to be circular. In the case of a close-proximity rendezvous for debris removal, this is a reasonable assumption as highlighted in [57]. The resulting frame of reference is called the RTN (Radial, Tangential and Normal), similar to LVLH, in which the Y axis points in the direction of the instantaneous velocity. This is often ambiguously equated to the Clohessy-Wiltshire frame, which however in the original work specifies the X and Y axes in the opposite direction [162]. To eliminate the ambiguity of definitions, this coordinate system will be unambiguously generalized as the LVLH coordinate system defined earlier.

Assuming mean circular orbit of the servicer, the equations simplify to Clohessy-Wiltshire (CW) equations, given as:

$$\begin{aligned} 0 &= \delta\ddot{x} - 3n\delta x - 2n\delta\dot{y} \\ 0 &= \delta\ddot{y} + 2n\delta\dot{x} \\ 0 &= \delta\ddot{z} + n^2\delta z \end{aligned} \tag{8.10}$$

for $\ddot{\mathbf{r}} \cdot \mathbf{r} = 0$ and $h = \sqrt{\mu r}$ in Eq. 8.9. The coefficients are constant in terms of reference orbit's mean motion, $n = \sqrt{\mu / r^3}$. Therefore, the closed-form solution to the CW equations exists and is given as a function of time :

$$\begin{bmatrix} \delta\mathbf{r}(t) \\ \delta\mathbf{v}(t) \end{bmatrix} = \Phi_{rv}(t) \begin{bmatrix} \delta\mathbf{r}_0 \\ \delta\mathbf{v}_0 \end{bmatrix} \tag{8.11}$$

where, Φ_{rv} is the state transition matrix. The matrices Φ_{rv} is computed as:

$$\Phi_{rv}(t) = \begin{bmatrix} \Phi_{11} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix} \tag{8.12}$$

where,

$$\Phi_{11} = \begin{bmatrix} 4-3\cos nt & 0 & 0 \\ 6(\sin nt - nt) & 1 & 0 \\ 0 & 0 & \cos nt \end{bmatrix} \quad \Phi_{12} = \begin{bmatrix} \frac{1}{n}\sin nt & \frac{2}{n}(1-\cos nt) & 0 \\ \frac{2}{n}(\cos nt - 1) & \frac{1}{n}(4\sin nt - 3nt) & 0 \\ 0 & 0 & \frac{1}{n}\sin nt \end{bmatrix}$$

$$\Phi_{21} = \begin{bmatrix} 3n\sin nt & 0 & 0 \\ 6n(\cos nt - 1) & 0 & 0 \\ 0 & 0 & -n\sin nt \end{bmatrix} \quad \Phi_{22} = \begin{bmatrix} \cos nt & 2sinnt & 0 \\ -2sinnt & (4\cos nt - 3) & 0 \\ 0 & 0 & \cos nt \end{bmatrix} \tag{8.13}$$

The analytical solution to CW equation follows an in-plane cycloidal motion and a decoupled out-of-plane harmonic motion. Finally, the dynamic model used for filter can be written by adding process noise in Eq. 8.11:

$$\begin{bmatrix} \mathbf{r}(t) \\ \mathbf{v}(t) \end{bmatrix} = \Phi_{rv}(t) \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 \end{bmatrix} + \Gamma_{rv}\mathbf{w}_v \tag{8.14}$$

where, the δ notation is dropped and relative position and velocity is denoted simply as $\mathbf{r}$ and $\mathbf{v}$ respectively. $\Gamma_{rv}\mathbf{W}_v$ term is the process noise incorporating small disturbance forces, given by the dynamic process covariance (Q_k), and Γ_{rv} is specified as follows [63]:

$$\Gamma_{rv}(t) = \begin{bmatrix} \frac{1}{2m}\mathbf{I}_{3x3}\Delta t^2 \\ \\ \frac{1}{m}\mathbf{I}_{3x3}\Delta t \end{bmatrix}$$

It is relevant to note that the pose estimated by the pose solver described in Chapter 7, is represented in the camera frame of reference. Hence, the estimated position vector is transformed from the camera frame to LVLH and vice-versa by accounting for the separation of spacecraft's center of mass and the camera frame origin. The servicer's arbitrary body frame is assumed to be centered at the spacecraft's center of mass, coinciding with the origin of the LVLH frame. If that is not the case, an additional linear translation between body frame and LVLH must also be included.

While CW equations have been widely utilized in modelling relative orbital motion, they take neither perturbations nor ellipticity of orbits into account. Consequently, the non-linearities in the real-world diverge from the CW estimates, depending on the perturbations and the ellipticity. However, the simplified dynamics and the closed-form solution of CW equations make it suitable for preliminary analysis using first-order approximations for the relative motion. Given an unconventional navigation pipeline, it is considered crucial to first verify the system in a simplified yet representative scenario.

The CW equations can be extended to include the J_2 effect in linearized form as shown by Schweighart and Sedwick [163]. In a more general sense, specific perturbations and ellipticity of orbit can be taken into account by adding perturbative accelerations in Eq. 8.9. However, without a closed-form solution, the computational demand of such a cartesian representation increases significantly. Alternatively, relative orbital elements formulation from D'Amico [164] can be used to include first-order perturbations in the linear dynamics model. Relative Orbital Elements are the constants of CW equations under certain assumptions and allow convenient representation of common perturbations, while being computationally efficient. The utility of ROE based dynamics model and it's broader inclusion in the GNC subsystem has been demonstrated in-flight for autonomous formation flying of PRISMA mission [133]. For a comprehensive survey on dynamics modelling for relative orbital motion, the reader is referred to Sullivan et al. [165].

As discussed earlier, the central experiments in this work are simplified for verification of the proposed framework as a whole. A propagator based on relative orbital elements was implemented but considered redundant for the simplified navigation scenarios considered in the experiments. For completeness and future use, the equations in ROE representation are noted in Appendix F.

8.2.2. Rotational Dynamics

To model the relative rotation motion between the spacecraft, a constant angular velocity model is considered with small angular rate assumption. Therefore, internal or external torques affecting either of the spacecraft are not taken into account. The model described here, closely follows the model in Tweddle and Saenz-Otero [166].

Consider attitude parameterization using unit-quaternion representation:

$$\mathbf{q} = q_0 + q_1\,\hat{i} + q_2\,\hat{j} + q_3\,\hat{k} \quad s.t. \quad \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1 \tag{8.15}$$

where, q_0 is the scalar component of the 4x1 quaternion vector.

The Non-linear dynamics in continuous time can be written using Euler's equations as:

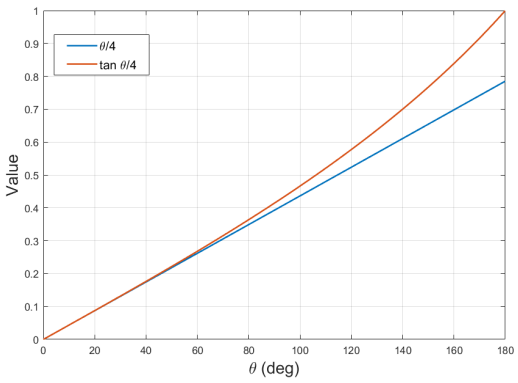
$$\dot{\mathbf{q}} = \frac{1}{2}\Omega(\omega)\,\mathbf{q} \tag{8.16}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}\left(-\boldsymbol{\omega}\times\mathbf{J}\boldsymbol{\omega} + \mathbf{w}_\omega\right) \approx \mathbf{J}^{-1}\,\mathbf{w}_\omega \tag{8.17}$$

where, $\mathbf{w}_\omega$ is the process noise parameter that incorporates small unmodeled torques acting on both spacecraft. Also,

$$\Omega(\boldsymbol{\omega}) = \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix} \qquad [\boldsymbol{\omega}\times] = \begin{bmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & \omega_1 & 0 \end{bmatrix}$$

Quaternion representation for attitude is convenient as the kinematic equations in quaternion form are (bi)linear and individual rotations represented by quaternions are non-singular. However, a 4-vector unit-quaternion is subject to normalization constraint given in Eq. 8.15. Additive nature of measurement update equations in the conventional EKF, do not ensure that the resulting quaternion has unit norm. More importantly, the covariance matrix associated with quaternion representation suffers from rank deficiency and one



r

Figure 8.1: MRP linearization trend for small angle assumption

of the eigenvalues of the state covariance matrix becomes zero. In numerical implementation, the eigenvalue becomes very close to zero but bears round-off errors which can result in the value being negative. This can lead to state covariance being non-positive definite. This effect compounds the numerical instability problem in the EKF and can lead to divergence of the filter estimate.

To eliminate the deterministic component of the quaternion vector, a 3-vector minimal notation similar to Modified Rodriguez Parameters (MRP) [1] is adapted. The 3-vector notation adapted in this work, is given as:

$$\mathbf{a}_p = \frac{4}{1+q_0}\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{8.18}$$

Note that $\mathbf{a}_p$ represents four times the MRP (σ) elements:

$$\begin{aligned} \mathbf{a}_p &= 4\sigma \\ \sigma = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \end{bmatrix} &= \mathbf{e}\tan\frac{\theta}{4} \end{aligned} \tag{8.19}$$

where, $\mathbf{e} = e_1\hat{i} + e_2\hat{j} + e_3\hat{k}$ represents the Euler axis and θ represents the angle of rotation around it, according to the Axis-angle representation summarized in Appendix B. For small angle rotations, notice that the MRPs linearize to represent one-fourth of the angles of rotation along the principal axes:

$$\mathbf{e}\tan\frac{\theta}{4} \approx \frac{\theta}{4}\mathbf{e} \tag{8.20}$$

Consequently, using a representation: $\mathbf{a}_p = 4\sigma$, provides direct physical interpretation of rotation angles about the principal axes. Given the tangent curve, the small angle linearization in Eq. 8.20, remains sufficiently close to the true value, even for sufficiently larger angles as shown in Fig. 8.1.

In addition to small angle assumption, if $\mathbf{a}_p$ is reset frequently, the kinematic equation is a linear function in state parameters can be obtained by linearizing around $\mathbf{a}_p = \mathbf{0}_{3x1}$:

$$\dot{\mathbf{a}}_p \approx \frac{1}{2}[\boldsymbol{\omega}\times]\mathbf{a}_p + \boldsymbol{\omega} \tag{8.21}$$

The system of equations in continuous-time are then given as :

$$\begin{bmatrix} \dot{\mathbf{a}}_p \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}[\boldsymbol{\omega}\times] & \mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}\begin{bmatrix} \mathbf{a}_p \\ \boldsymbol{\omega} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{3\times3} \\ J^{-1} \end{bmatrix}\mathbf{w}_\omega \tag{8.22}$$

[1] See Appendix B

To obtain a discretized form of Eq. 8.22, the state transition matrix ($\Phi_{a\omega}$) and the noise propagation matrix ($\Gamma_{a\omega}$) are computed using matrix exponential of an augmented matrix $\mathbf{A}'$ from Eq 8.22 as

$$\begin{aligned} \Phi_{\mathbf{D}} &= e^{\mathbf{A}'\delta t} \\ \mathbf{A}' &= \begin{bmatrix} \frac{1}{2}[\boldsymbol{\omega}\times] & \mathbf{I_{3\times3}} & \mathbf{0_{3\times3}} \\ \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{J_{-1}} \\ \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} \end{bmatrix} \end{aligned} \tag{8.23}$$

where the discrete time state transition ($\Phi_{a\omega}$) and noise propagation ($\Gamma_{a\omega}$) are submatrices of Φ_D, with Φ_{Dij} representing a 3x3 block submatrix in Φ_D :

$$\begin{aligned} \Phi_{a\omega} &= \begin{bmatrix} \Phi_{\mathbf{D11}} & \Phi_{\mathbf{D12}} \\ \Phi_{\mathbf{D21}} & \Phi_{\mathbf{D22}} \end{bmatrix} \\ \Gamma_{a\omega} &= \begin{bmatrix} \Phi_{\mathbf{D13}} \\ \Phi_{\mathbf{D23}} \end{bmatrix} \end{aligned} \tag{8.24}$$

8.3. Multiplicative Extended Kalman Filter Design

MEKF is an extension of EKF that utilizes two representations of attitude to overcome the rank deficiency and numerical instability issues involved in quaternion based EKF. Introduced by Lefferts et al. [64], the working principle of MEKF is based on a multiplicative error quaternion that allows the reset of the attitude state parameters at every filter loop. A multiplicative error quaternion [160] between two quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ is given by

$$\delta\mathbf{q}_{12} = \mathbf{q}_1 \otimes \mathbf{q}_2^{-1} \tag{8.25}$$

where, the operation $\otimes$ represents a quaternion multiplication and the -1 in the superscript represents a quaternion inverse.

The three parameter representation ($\mathbf{a}_p$), as defined in 8.18 is used for attitude representation within the filter loop, while the state tracking is done with quaternions. After every iteration, the update to the filter attitude parameters ($\mathbf{a}_p$) is used to update a reference quaternion with the multiplicative error quaternion. Subsequently, the filter attitude parameters are reset to zero for the next update. The MEKF formulation overcomes the rank deficiency issue in attitude covariance propagation that results from normalization constraint of the unit-quaternion representation for attitude. Since the reset step applies only to rotational state, the translation state estimation is equivalent to that of a traditional EKF. In the following, the state estimator design is described in detail, closely following the work of Tweddle and Saenz-Otero [63].

Consider the state vector:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} & \mathbf{v} & \mathbf{q} & \boldsymbol{\omega} \end{bmatrix}^T \tag{8.26}$$

where, $\mathbf{r}$ and $\mathbf{v}$ are relative position and velocity in the LVLH frame, while the $\mathbf{q}$ and $\boldsymbol{\omega}$ are the relative attitude and angular velocity in the body frame of the servicer.

At a time step k, the reference quaternion is set as the posterior estimate of quaternion vector from the previous step:

$$\mathbf{q}_{ref} = \mathbf{q}_{k-1}$$

The internal state in the filter loop is configured as:

$$\mathbf{x}' = \begin{bmatrix} \mathbf{r} & \mathbf{v} & \mathbf{a}_p & \boldsymbol{\omega} \end{bmatrix}^T \tag{8.27}$$

where, 3-vector $\mathbf{a}_p$ replaces the 4-vector $\mathbf{q}$. Every filter loop is initialized (reset) as $\mathbf{a}_p = \mathbf{0}_{3x1}$. The prediction step uses discrete-time combined dynamic model for the translation and rotation:

$$\hat{\mathbf{x}}_k^- = \Phi_k \hat{\mathbf{x}}_{k-1}^+ + \Gamma_k \mathbf{w}_k \tag{8.28}$$

$$\Phi_k = \begin{bmatrix} \Phi_{rv} & \mathbf{0}_{6x6} \\ \mathbf{0}_{6x6} & \Phi_{a\omega} \end{bmatrix}_k \quad ; \quad \Gamma_k = \begin{bmatrix} \Gamma_{rv} & \mathbf{0}_{6x3} \\ \mathbf{0}_{6x3} & \Gamma_{a\omega} \end{bmatrix}_k \tag{8.29}$$

with associated covariance prediction:

$$\mathbf{P}_k^- = \Phi_\mathbf{k} \mathbf{P}_{k-1}^+ \Phi_k + \mathrm{Q}_k$$

where,

$$\mathrm{Q}_k = \begin{bmatrix} \Gamma_{rv} & \mathbf{0}_{6x3} \\ \mathbf{0}_{6x3} & \Gamma_{a\omega} \end{bmatrix} \begin{bmatrix} \sigma_v^2 & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \sigma_\omega^2 \end{bmatrix} \begin{bmatrix} \Gamma_{rv} & \mathbf{0}_{6x3} \\ \mathbf{0}_{6x3} & \Gamma_{a\omega} \end{bmatrix}^T \tag{8.30}$$

the state transition sub-matrices for translation (Φ_{rv}) and rotation ($\Phi_{a\omega}$) in Eq. 8.29 are computed as decribed previously in Eqs. 8.13 & 8.24 respectively. The measurements are then used to update this apriori estimate.

8.3.1. Measurement Model

A loosely-coupled estimator is proposed for the navigation framework that utilizes the pose covariance computed by the MLPnP solver. As highlighted in Section 2.3.3 & 3.1, maintaining a representative measurement uncertainty in the filter loop is crucial for filter's robustness to adverse detection. The measurement model of a loosely-coupled estimator utilizes the psuedomeasurements of the pose obtained from the pose solver.

$$\rho_k = \begin{bmatrix} \mathbf{t}^C \\ \mathbf{q}^{BC} \end{bmatrix} \tag{8.31}$$

The translation vector must be transformed to the LVLH frame, while the quaternion representing relative attitude must be transformed to 3-vector $\mathbf{a}_p$ that represents the multiplicative error quaternion.

$$\mathbf{z}_k = \begin{bmatrix} \mathbf{r}_k \\ \mathbf{a}_k \end{bmatrix} \tag{8.32}$$

where, following from [57, 166]:

$$\mathbf{r}_k = \mathbf{R}_{LVLH}\, \mathbf{t}^C$$

$$\mathbf{a}_k = \frac{4}{1+\delta q_{z0}} \begin{bmatrix} q_{z1} \\ q_{z2} \\ q_{z3} \end{bmatrix} \quad ; \qquad \delta\mathbf{q}_z = \mathbf{q}^{BC} \otimes \mathbf{q}_{ref}^{-1}$$

$\mathbf{R}_{LVLH}$ is the transformation from camera frame to LVLH. For the close-proximity V-bar hold scenario considered in the navigation analysis, the camera boresight points in the along-track direction. Therefore, the transformation from camera frame to LVLH is given simply as following:

$$\mathbf{R}_{LVLH} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{8.33}$$

The measurement model for the considered loosely-coupled estimator can be written as:

$$\hat{\mathbf{z}}_k = \mathbf{H}\hat{\mathbf{x}}_k^- + \boldsymbol{\nu}_k \tag{8.34}$$

where, the measurement Jacobian ($\mathbf{H}$) is defined as:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{I}_{3x3} & \mathbf{0}_{3x3} \end{bmatrix} \tag{8.35}$$

Instead of using a constant measurement covariance, a commonly-adapted approach, the measurement covariance (R_k) used in the filter is the covariance ($\Sigma_\mathbf{at}$) computed by obtained from the MLPnP run for the corresponding image, as in Eq. 7.30.

$$\mathrm{R}_k = (\Sigma_\mathbf{at})_k$$

This ensures that measurement covariance is representative at all times to handle dynamic scenarios with bad visibility or poor feature detections for a loosely-coupled estimator

8.3.2. Update Equations

The updated a-posteriori state estimate $\hat{\mathbf{x}}_k^+$ is obtained from the a-priori estimate$\hat{\mathbf{x}}_k^-$ and the Kalman Gain $\mathbf{K}$ as following:

$$\mathbf{K} = \mathbf{P}_k \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k \mathbf{H}_k^T + \mathrm{R}_k)^{-1} \tag{8.36}$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}(\mathbf{z} - \mathbf{H}\mathbf{x}_k^-) \tag{8.37}$$

When the measurement frequency is not the same as filtering frequency, the apriori estimate, $\mathbf{x}_k^-$ is the propagated state at step k.

8.3.3. Reset Step

In the reset step, the reference quaternion $\mathbf{q}_{\text{ref}}$ is updated with the attitude error estimate $\hat{\mathbf{a}}_\mathbf{p}$ and the new attitude error is set to zero,

$$\hat{\mathbf{q}}_k = \delta\mathbf{q}(\hat{\mathbf{a}}) \otimes \mathbf{q}_{\text{ref}_k} \tag{8.38}$$

$$\hat{\mathbf{a}} = \mathbf{0}_{3\times 1} \tag{8.39}$$

$$\mathbf{q}_{\text{ref}_{k+1}} = \hat{\mathbf{q}}_k. \tag{8.40}$$

The obtained estimated quaternion set $\hat{\mathbf{q}}_k$ is then compared to the real quaternion set to assess the angle accuracy of the filter.

8.4. Conclusions

The design of an uncertainty-aware loosely-coupled estimator has been presented for preliminary analysis in the navigation framework. A resettable MEKF is adapted, which uses pose psuedo-measurements to track the translation and rotation states. The estimator is designed to be robust to variations in the optical optical observations of the monocular camera by using the dynamic measurement uncertainty computed in the loop by the pose solver. The estimator seamless integrates with the rest of the components of the navigation framework. Further, the design of an uncertainty-aware tightly-coupled estimator, a complementary approach to robust estimation, has also been described for simulation and comparison.

III

Framework Evaluation

9

Experiments, Simulations and Results

To assess the performance and robustness of the resulting navigation loop, the components of the navigation framework are tested incrementally. The designed experiments allow quantitative and qualitative assessment of design choices within the individual framework components. Components of the pose estimation pipeline are benchmarked and verified on the standard *SPEED*, allowing comparison with the state-of-the-art approaches. Subsequently, the efficacy of the neural networks to train on synthetic images and transfer effectively to the reality is tested on lab generated real images of the Tango spacecraft. The components are then evaluated on the Envisat target use-case concerned in this work, presenting a consistent comparison and contrast between the two datasets. The framework is further tested using the Envisat+IC dataset for corruption robustness to demonstrate improvement over the existing datasets. Finally, the ensuing navigation loop is evaluated on a V-bar approach scenario, emulating proximity operations around a tumbling Envisat spacecraft. Using a simplified dynamical environment, the performance of a loosely-coupled state estimator, with measurement uncertainty derived from CNN heatmaps is demonstrated. The results presented here extend the current analyses on vision-based navigation of uncooperative proximity operations and provide newer insights in developing learning-based pipelines that are more suitable for on-orbit operations.

9.1. Object Detection

The performance and the robustness of the OD networks are evaluated by consistently training the networks for 100000 global steps with a base learning rate of 0.001, as discussed in Section 4.2. The input size is fixed at 320 px × 320 px, which is common for real-time object detectors [106]. The input size for the OD network is slightly larger than the input window in the KD network to minimize loss of information in the pipeline while keeping FLOPs count manageable. The networks are evaluated in terms of the IoU metric and the percentage of feasible detections (IoU>0.75). The experiments assess the effect of network size and data augmentations on accuracy and robustness for *SPEED* and the Envisat datasets. The configuration of training and testing follows from Section 4.2, unless specified otherwise.

9.1.1. SPEED Benchmark

Accuracy Assessment

(a) Network size: The two selected networks: *SSDLite-MNetV3-large* and *SSDLite-MNetV3-small*, are trained and tested to evaluate the effect of network size on the performance. The networks are trained with the 9600 image subset from the training (*train-1*) set and the trained network models are subsequently evaluated on the 2400 images of the validation (*val-1*) set (ref Section 6.1.2). Recall that the original images of the *SPEED* dataset are of size 1920 px × 1200 px, which is resized to 320 px × 320 px at training time. A network training takes 1.25-2 seconds per global step with the GPU, in addition to miscellaneous overheads. Totally, the training routine requires between 24-48 hours. The training only includes affine augmentations i.e. random rotation, flip and scaling of the image, that enable pose generalization. The IoU>0.75 metric is used to quantify feasible detections i.e. the percentage of detections that can be effectively used by the KD network to detect sufficient keypoints for the pose solver, as highlighted in Section 4.2.8. To interpret the processing re-

quirement, the inference times are recorded on an Intel Core i7-8750 CPU 2.20 GHz and averaged through all inferences. This time is only an indicative measure to interpret and compare the effect of GFLOPs. Table 9.1 shows results of the evaluations on the validation set for the trained models. Note that for consistent referencing in the following discussions, the trained network models are assigned a unique model identifier.

The results show that the larger network in the SPD-OD-1 model outperforms the smaller network in SPD-OD-2 by 6% in terms of mean IoU. Also, SPD-OD-1 provides significantly higher feasible detections, as indicated by IoU>0.75 metric. The superior performance of the larger network model is provided at a marginally higher model size (parameters) and computation cost (GFLOPs). As an indication, the difference in CPU inference speed is not considered significant between the two networks. Since the OD network is the first part of the pose estimation pipeline, the performance of the pose estimation pipeline is highly dependent on its accuracy and robustness. Lower accuracy in OD can create a performance bottleneck by producing detections that cannot be used effectively in the KD network and the pose solver. For instance, the SPD-OD-2 network model can significantly deteriorate the detection of keypoints and pose solution performance in 275 out of 2400 images due to infeasible detections. In comparison, the SPD-OD-1 model presents only 76 images with potential deterioration. Given the performance superiority of the larger network at a marginally higher computation requirement, the larger network (*SSDLite-MNetv3-Large*) is selected for the subsequent experiments.

Identifier	Network	Parameters (Mn)	FLOPs (Bn)	Mean IoU	Median IoU	IoU>0.75	CPU inference speed (ms)
SPD-OD-1	*SSDLite-MNetV3-large*	3.2	0.5	**0.925**	**0.949**	**96.8**%	67
SPD-OD-2	*SSDLite-MNetV3-small*	**2.4**	**0.16**	0.864	0.902	88.5%	42

Table 9.1: Performance comparison of selected network sizes on *SPEED* validation set

(b) Pixel-level data augmentation: For this experiment, the selected network is trained with pixel-level data augmentations **DA-1** (random brightness and contrast). Note that this is in addition to the **DA-0** affine augmentations included in the previous training routines (see Table 4.2.6. Since the images are augmented in the loop, the overhead time for training increases as repeated calls to the augmentations library is made. The network training takes 4-8 seconds per global step and results in a total training time between 48-96 hours. Note that the implications of data augmentations only apply to the training and do not affect inference times or FLOPs. The evaluation results for the trained network model, SPD-OD-3 is presented in Table 9.2 with a comparison to SPD-OD-1.

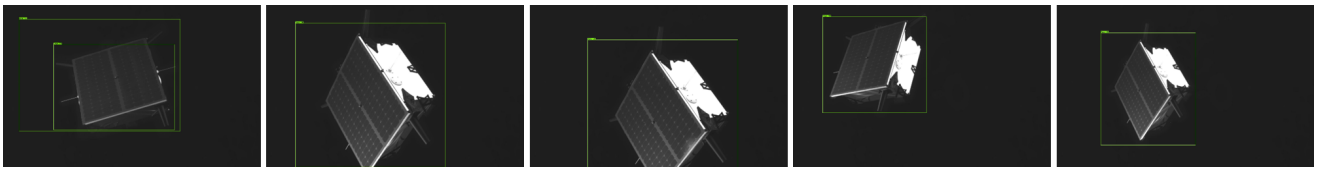
Identifier	Network	Data Augmentations	Mean IoU	Median IoU	IoU>0.75
SPD-OD-1	*SSDLite-MNetV3-large*	**DA-0**	0.925	0.949	96.8%
SPD-OD-3	*SSDLite-MNetV3-large*	**DA-0** + **DA-1**	0.921	0.944	96.2%

Table 9.2: Performance comparison of pixel-level data augmentations in training for *SSDLite-MNetV3-large* on *SPEED* validation set. Augmentations (1) **DA-0**: scaling, rotation and horizontal flip **DA-1**: erase (black patches), brightness and contrast

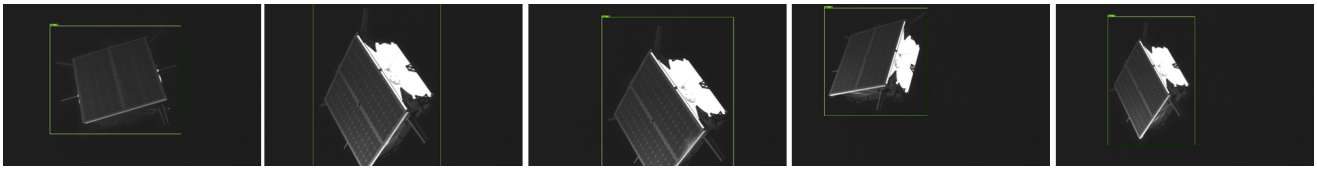
The network trained with additional pixel-level data augmentations performs almost equally well on the validation set. However, a 0.4% decrease is observed in the IoU. Ideally, data augmentations must improve the performance of a network on a dataset. But, having a broad range of data augmentations, which do not appear in the evaluation dataset can cause underfitting and therefore a decrease in overall performance in that specific dataset, as noted by Zheng et al. [167]. A possible solution to improving the performance with the pixel-level data augmentations, specifically on the validation set, is to carefully tune the data augmentations to reflect the variations seen in the validation set. However, this is not considered relevant in the scope of this work and for benchmarking. The relevance of data augmentations is realized when testing on images from a different distribution, as discussed below in robustness assessment.

Robustness Assessment

The previous experiments tested the accuracy of OD on the validation set, which contains synthetic images very similar to the training set, as they are sampled from the same superset. It is therefore essential to evaluate



(a) SPD-OD-1



(b) SPD-OD-3

Figure 9.1: Comparison of detection robustness of trained network models on unseen real images

the robustness of the network models trained on these synthetic images, to previously unseen images from a different source. A set of 5 lab-generated real images of the Tango spacecraft from the *real* set in *SPEED* are used to emulate the aforementioned gap between synthetic rendering environment and the reality. Note that the network has neither been trained on nor has encountered these five images before, to fairly evaluate the performance transfer of the trained models from synthetic to real images. The trained network models, SPD-OD-1 and SPD-OD-3, that differ in training time data augmentations are tested on the real images. Table 9.3 presents the results and compares the performance of the two network models. Additionally, Fig. 9.1 visualizes the box detections generated by both the network models.

Identifier	Data Augmentations	Mean IoU	Median IoU	IoU>0.75
SPD-OD-1	**DA-0**	0.776	0.757	80%
SPD-OD-3	**DA-0** + **DA-1**	**0.941**	**0.950**	**100.0**%

Table 9.3: Performance of trained network models on 5 images from *real* set in *SPEED*. Augmentations- **DA-0**: scaling, rotation and horizontal flip **DA-1**: erase (black patches), brightness and contrast

The SPD-OD-3 model trained with pixel-level data augmentations is objectively more robust and accurate to previously unseen real images than SPD-OD-1, which is trained only on affine augmentations. While SPD-OD-1 generates feasible detections for four images out of five, the accuracy of detections is significantly lower as indicated by the mean and median IoU. This reinforces the importance of pixel-level data augmentations for robustness. Fig 9.1 shows the visualizes the box detections of the two trained network models on the five real images. The SPD-OD-1 model's box detections are found to cut-off a part of the spacecraft containing one of the omnidirectional antennae. Besides, the SPD-OD-1 model produces a misdetection, as the network generates two boxes with low IoU that are not resolved with non-maximum suppression. On the other hand, the SPD-OD-3 model trained on pixel-level augmentations (**DA-1**), detects the spacecraft with a desirable accuracy in all the real images. The lack of robustness in SPD-OD-1 can be explained by the fact that since no pixel-level data augmentations were included during the training, the trained model overfits on pixel-level variation in the synthetic validation set as is. Therefore, it is unable to produce accurate results in real images that may not exhibit similar pixel-level variations.

The SPD-OD-3 model featuring the *SSDLite-MNetv3-large* network trained with pixel-level and affine data augmentations, is selected as the final OD model for benchmark comparison in the following section and further integration with the KD model in the pose estimation pipeline.

Benchmark Comparison

The selected network model (SPD-OD-3) is finally compared to the state-of-the-art OD results for *SPEED*, available in Sharma et al. [45], Park et al. [46] and Chen et al. [78]. The comparison of the OD networks and the results are presented in Table 9.4. Note the disparity in the image sub-sets on which results are available. This is because *test* set labels are not available to the public but are available to the original author of the dataset [45]. Notably, Chen et al. [78] use an ensemble of 6 networks, each of which are trained using 6-fold

cross-validation. This is done by dividing the data into 6 distinct training and validation sets of 5:1 ratio, such that each network is trained on one such pair of training and validation set. Park et al. [46] use a random validation subset consisting of 20% of the training images, which are held back from the training, similar to the approach adopted in this work. For distinction, the results from this work are identified with 'R-NAV'.

	Sharma and D'Amico [45]	Park et al. [46]	Chen et al. [78]	R-NAV (SPD-OD-3)
Mean IoU	0.858	0.919	**0.953**	0.921
Median IoU	0.891	0.936	**0.963**	0.944
Image set	*test*	validation subset	*train*	validation subset (*val-1*)
Detection Head	Faster R-CNN [84]	YOLO [98]	Faster R-CNN [84]	SSD [85]
Backbone	custom	MNetv2 [105]	HRNet-W18-C [128]	MNetv3-large [106]
Ensemble	No	No	6x	No
Input size (px)	N/A	412 × 412	N/A	320 × 320
Parameters (Mn)	N/A	5.53	(6x) 26.2*	**3.22**
FLOPs (Bn)	N/A	N/A	(6x) 159.1*	**0.5**

Table 9.4: Benchmark comparison of state-of-the-art results on *SPEED* dataset for OD. R-NAV identifies the framework designed in this work.(* : estimated for HRNet-W18-C [128])

The accuracy of the SPD-OD-3 network model is comparable with the state-of-the-art performance on *SPEED* while being significantly more efficient in terms of memory and computation requirements. It is relevant to note that the state-of-the-art performance in terms of IoU from Chen et al. [78] is achieved using an ensemble of 6 large networks (26.2 Mn parameters and 159.1 GFLOPs per network). This is neither feasible nor representative in terms of its suitability to onboard deployment. SPD-OD-3 outperforms the custom-designed network from Sharma and D'Amico [45] by around 6% in mean and median IoU. Finally, the OD network from Park et al [46] provides a basis for a fairer comparison, as the network performance is at similar levels and the network design is more feasible for on-board deployment. However, R-NAV OD outperforms by around 1% in the mean IoU, while being more memory efficient.

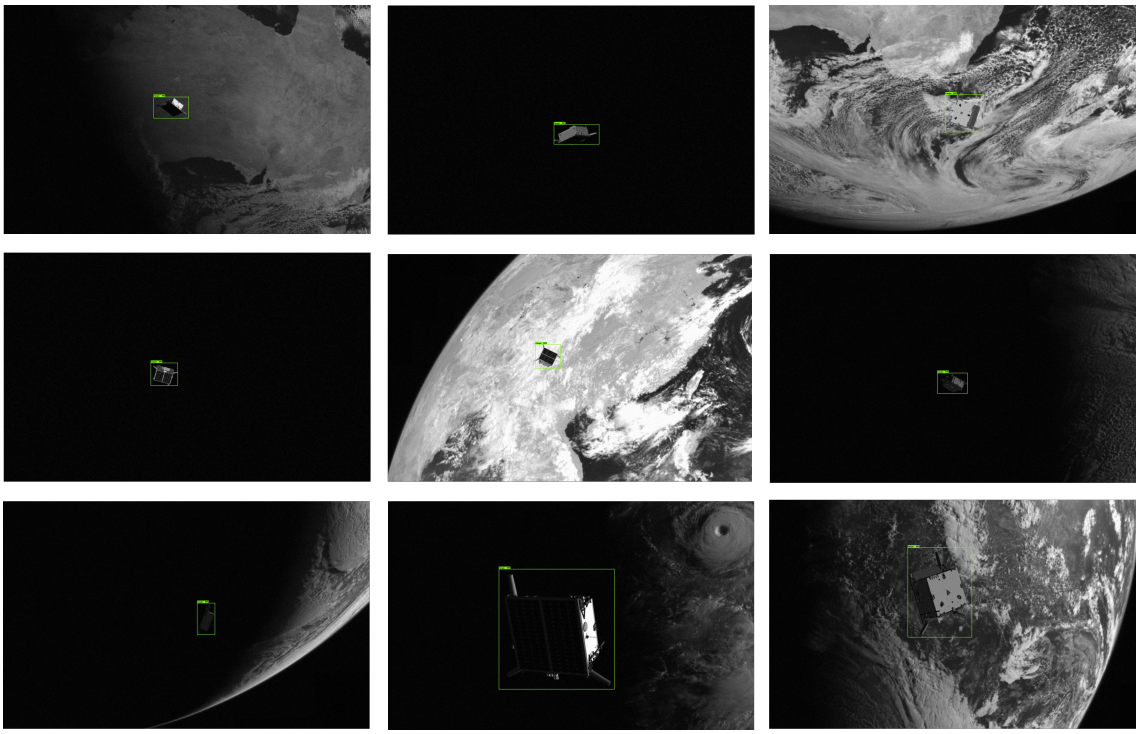
The selection of *SSDLite-MNetV3-large* network, configuration, and training has been further validated with a benchmark comparison on *SPEED*. The resulting network model (SPD-OD-3) demonstrates an appropriate performance vs cost trade-off compared to the existing state-of-the-art network choices while being robust to the gap between synthetic and real images. Fig. 9.2(a) shows accurate detections of the Tango spacecraft, while Fig. 9.2(b) shows poor detection, on the synthetic validation set in *SPEED* using the SPD-OD-3 model. Notice that bad detections are observed in the images where the spacecraft is at a higher relative distance, in poor illumination, and made visually ambiguous by the high contrast adverse features of the Earth.

9.1.2. Envisat: Evaluation and Robustness

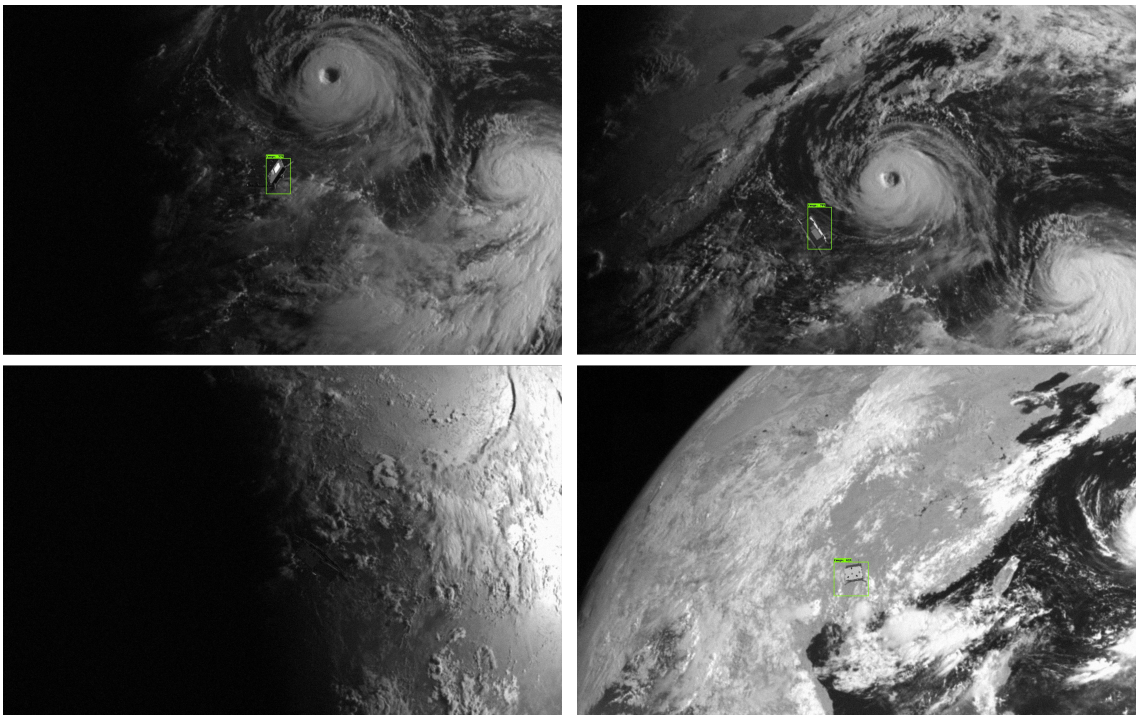
The experiments are repeated for Envisat image datasets, to evaluate OD performance for the Envisat target use-case in this work. First, the networks are trained and evaluated on the *Envisat-1* dataset from Pasqualetto Cassinis et al. [57]. The evaluation results and inferences are used to show the simplicity of the *Envisat-1* dataset, when compared with *SPEED*. Further, the networks are trained and tested with the *Envisat+IC* augmented dataset, introduced in this work. The network models trained on *Envisat-1* and *Envisat+IC* are tested on validation sets (*val, val-clean* and *val-challenge*), designed for robustness assessment.

Accuracy Assessment

(a) Envisat-1: The networks are trained on 35000 images of the *train* set and evaluated on 9600 images of the *val* set. Recall that the original images of the *Envisat-1* dataset are of size 512 px × 512 px, which are resized to 320 px × 320 px. The training normally takes 0.9 sec per global step and requires upto 36 hours. Note that this dataset uses 16 point 3D wireframe model (main body, SAR antenna and the solar panel) of the Envisat spacecraft similar to the original work [57]. First, to show the comparative simplicity of this dataset, the two



(a) Examples of accurate detections (IoU>0.9)



(b) Examples of inaccurate (IoU<0.5) or no detections

Figure 9.2: Visualization of images with detections from the *SSDLite-MNetv3-large* benchmark network

network sizes (*SSDLite-MNetv3-large* and *SSDLite-MNetv3-small*) are trained and evaluated. As earlier, only affine data augmentations are used during training. Table 9.5 shows the evaluation results on the validation set.

The trained models corresponding to the two network sizes show comparable performances with a difference of less than 1% in mean IoU. Also, the ENV1-OD-1 and ENV1-OD-2 models provide feasible detections for almost all validation set images. The ENV1-OD-1 model featuring the larger network only marginally outperforms the ENV1-OD-2 model with the smaller network. This is expected when the network can recognize the patterns in the data more easily, which is in contrast to the evaluations on *SPEED*. Further, the mapping between the input image and the detection generated by the trained model is expected to be simpler, since a smaller network with lower model capacity performs equally well. More importantly, the patterns used to generate the mapping between the input image and the detection apply almost always exactly to the validation set, implying the validation set is extremely similar to the training set. These inferences reinforce the premise set in Chapter 6 about the simplicity and the inherent inability of the *Envisat-1* dataset to sufficiently evaluate the CNNs.

Model	Network	Mean IoU	Median IoU	IoU>0.75
ENV1-OD-1	*SSDLite-MNetV3-large*	**0.959**	**0.967**	**99.9** %
ENV1-OD-2	*SSDLite-MNetV3-small*	0.938	0.949	99.7 %

Table 9.5: Performance evaluation of the selected OD networks on *Envisat-1* dataset

Model	Network	Data Augmentations	Mean IoU	Median IoU	IoU>0.75
ENV1-OD-1	*SSDLite-MNetV3-large*	**DA-0**	0.959	0.967	99.9 %
ENV1-OD-3	*SSDLite-MNetV3-large*	**DA-0** + **DA-1**+**DA-2**	0.883	0.902	93.1%

Table 9.6: Performance evaluation of pixel level data augmentations on *Envisat-1* dataset. **DA-0**: scaling, rotation and horizontal flip; **DA-1**: erase (black patches), brightness and contrast; **DA-2**: random RGB to gray

Next, the larger network is trained with additional pixel-level data augmentations (**DA-1**). Since the *Envisat-1* dataset consists of 3-channel color images as opposed to *SPEED*, an additional random RGB to gray augmentation is included with an expectation of making the trained model agnostic to camera channels. This is denoted as **DA-2**. Table 9.6 shows the performance of the resulting network model ENV1-OD-3, compared to ENV1-OD-1. When the network is trained with the aforementioned pixel-level augmentations, the resulting model exhibits unexpectedly poor validation performance with severe underfitting. In this case, the most plausible reason for the decrease in performance is the random RGB to gray augmentation. Hence, a trained network model agnostic to camera channels is not realized by adding RGB to gray augmentations, as it results in poor performance on the original images. Further investigation on this aspect is not in the scope of this work and must to be conducted in the future works. For subsequent experiments, only the **DA-0** and **DA-1** augmentation groups are used, where applicable.

Notice that results are only provided for the *val* set of the *Envisat-1* dataset. This is done for consistency of using a validation set through out the experiments, while also decrease the amount of redundant data in the results. This is reasoned by the fact that the performance of the networks on *val* set is reflective of the performance on other subsets as well. This is verified by the consistent evaluation performance on the respective image sets as shown in Table 9.7. The detections produced by ENV-OD-1 for the images in the validation set are visualized in Fig. 9.3

Model	Image set	Mean IoU	Median IoU	IoU>0.75
ENV1-OD-1	*val*	0.9588	0.9670	99.9 %
	test	0.9592	0.9669	99.9%

Table 9.7: Performance generalization across *Envisat-1* image sets

(b) Envisat+IC: The *Envisat+IC* dataset adds Earth background and various blur and noise augmentations to the clean *Envisat-1* images. The preffered network (*SSDLite-MNetv3-large*) is trained on 35000 images of the *train* set. Recall that the solar panel is not taken into account and the detection is made for the main body and the SAR antenna (12 point 3D wireframe model). The resulting trained model is evaluated on the 5000

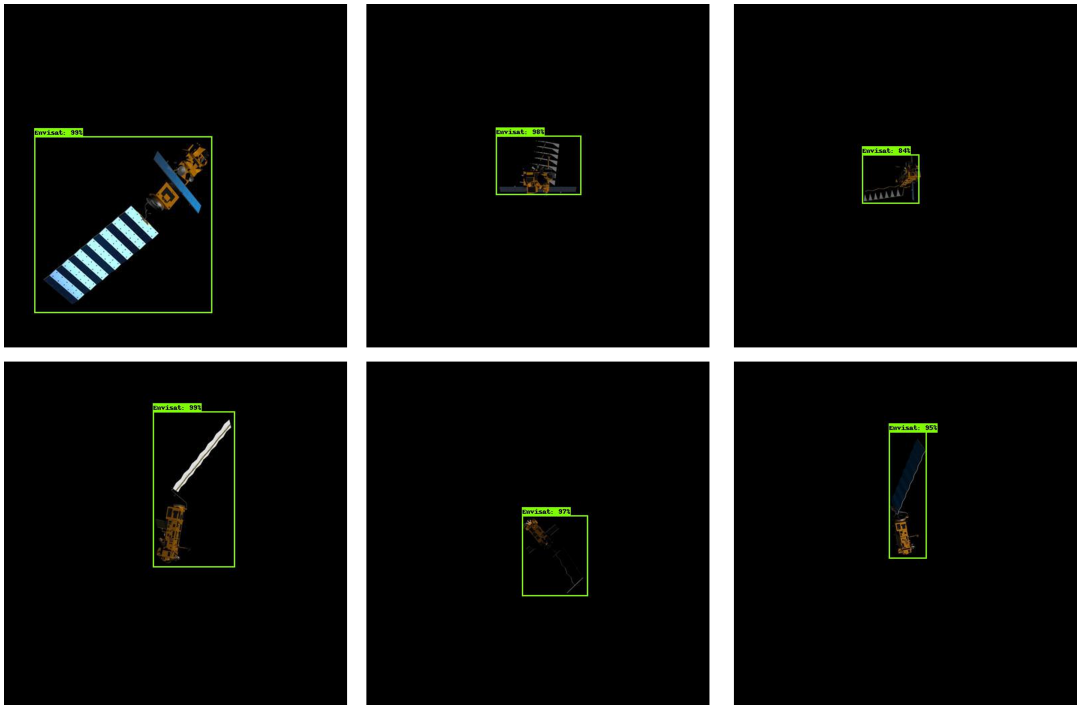


Figure 9.3: Sample detections produced by ENV1-OD-1 in *Envisat-1* validation set images

images of the *val* set described in Section 6.4. Only the default (**DA-0**) affine augmentations are included in the training, as the dataset already augments random brightness, contrast and saturation type pixel-level augmentations. The evaluation result is presented in Table 9.8.

Model	Network	Mean IoU	Median IoU	IoU>0.75
ENVIC-OD-5	*SSDLite-MNetV3-large*	0.903	0.919	96.9 %

Table 9.8: Performance evaluation of the OD network on *Envisat+IC* dataset

The ENVIC-OD-5 network model performs well on the corrupted images. The mean IoU is lower than that obtained for ENV1-OD-1 validation on *Envisat-1*. This is expected due as the *Envisat+IC* images are inherently more challenging with Earth background and the corruptions as compared to the *Envisat-1* images. But note that the decrease in accuracy also results from the fact that the network is now detecting smaller region (main body and SAR antenna) in comparison to the dataset with 16 point 3D model (main body, SAR antenna and solar panel) used for *Envisat-1*. Fig. 9.4 visualizes detections on a few images in the *Envisat+IC val* set.

Robustness Assessment

The accuracy of the trained network models was evaluated on the respective validation sets. Given desired accuracy performance, the trained models must be evaluated for their robustness to unseen images, to understand how well a model can detect the spacecraft in images with artefacts not present in the synthetic training set. To evaluate robustness, the best models trained respectively on *Envisat-1* and *Envisat-IC* are evaluated on *val, val-clean* and *val-challenge* image sets. These image sets are included in the new Envisat-IC dataset, as described in Section 6.4, to test and improve the robustness of deep neural networks for Envisat target case. The ENV1-OD-1 and ENVIC-OD-5 network models are selected as they are the best performing models on the respective datasets and only differ in the dataset on which they were trained. The detections are also evaluated for the respective 3D wireframe models i.e. ENV1-OD-1 is evaluated on 16 point model to detect a box around the whole spacecraft while the ENVIC-OD-5 model is evaluated on the 12 point model to detect a box excluding the solar panel. Table 9.9 presents the results of the robustness test.

The ENVIC-OD-5 model trained on *Envisat+IC* exhibits robustness on clean images of the *val-clean* set as well as on the corrupted images of *val-challenge* set which has corruption magnitudes higher from that

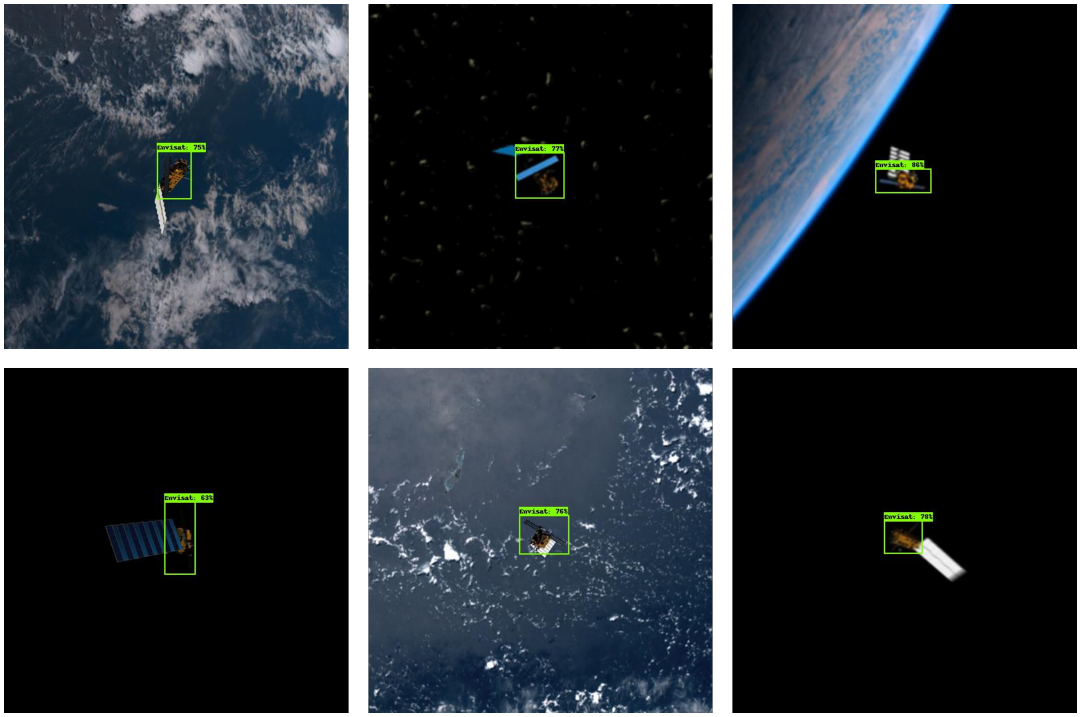


Figure 9.4: Sample detections produced by ENVIC-OD-5 in *Envisat-1* validation set images

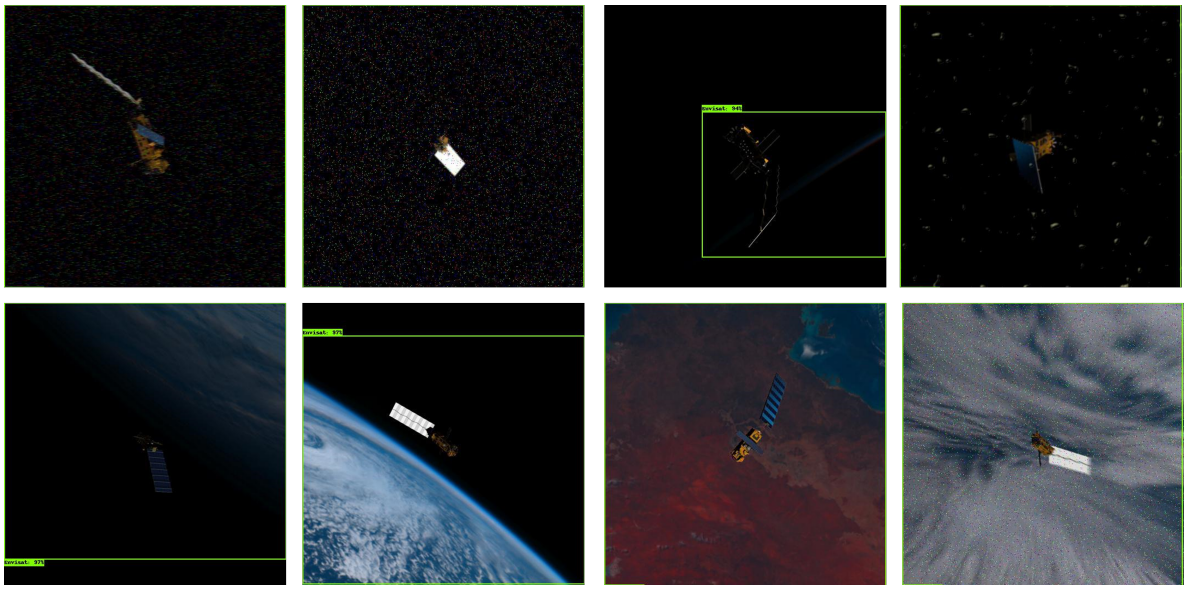
Model	Training dataset	Evaluation set	Mean IoU	Median IoU	IoU>0.75
ENV1-OD-1	*Envisat-1*	*val-clean*	**0.959**	**0.967**	**99.9** %
	Envisat-1	*val-IC*	0.699	0.940	63.7 %
	Envisat-1	*val-challenge*	0.622	0.924	63.7 %
ENVIC-OD-5	*Envisat-IC*	*val-clean*	0.924	0.933	99.7 %
	Envisat-IC	*val-IC*	**0.903**	**0.919**	**96.9** %
	Envisat-IC	*val-challenge*	**0.880**	**0.908**	**91.7**%

Table 9.9: Robustness evaluation of the selected OD networks on *Envisat-1* dataset

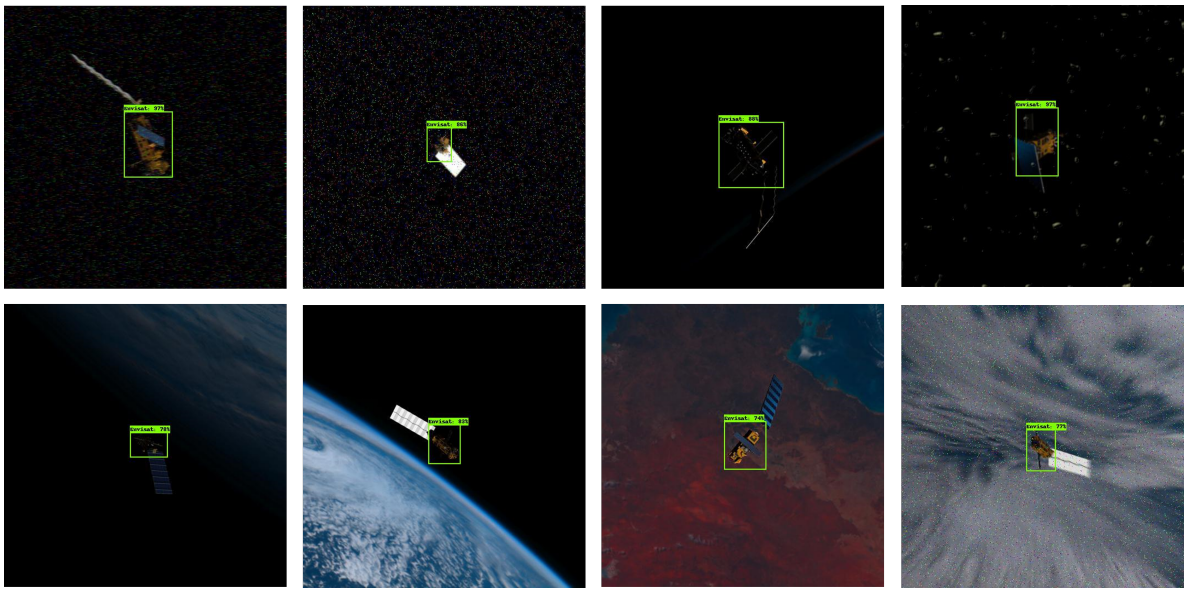
in the training. On the other hand, the ENV1-OD-1 model trained solely on clean images of *Envisat-1* shows poor performance on augmented images, further reinforcing the hypothesis that simple *Envisat-1* dataset is unsuitable for training deep neural networks that must generalize well to previously unseen images that do not exactly imitate the clean images. Fig. 9.5 shows example detections produced by ENV1-OD-1 and ENVIC-OD-5 models on augmented images from *val* and *val-challenge* image sets.

Note that the various validation sets used for the robustness assessment may contain multiple augmentations, as highlighted in Section 6.4. To further investigate the effect of each corruption to detection performance decrease of ENV1-OD-1, the model is tested on images each of which contains one corruption. Table 9.10 summarizes the variation of mean IoU per image augmentation/corruption for a total 5000 images, each of which contain only one augmentation.

The ENV-OD-1 model trained on *Envisat-1* shows robustness to several corruptions in the images like speckle noise, shot noise, Gaussian blur, etc. The accuracy of detection deteriorates significantly on images with color jitter augmentation which randomizes brightness, contrast, and saturation in the images. This shows that *Envisat-1* dataset lacks the basic variation in brightness, contrast, and saturation, which is very likely to persist in a real image. Further, the detections are severely degraded for images with Earth in the background, Gaussian noise, impulse noise, and spatter augmentations.



(a) ENV1-OD-1



(b) ENVIC-OD-5

Figure 9.5: Comparison of detections produced by models trained on *Envisat-1* and *Envisat+IC* datasets

Augmentation	Mean IoU
None	0.958
Speckle noise	0.961
Shot noise	0.957
Random erase	0.948
Gaussian blur	0.940
Defocus blur	0.931
Motion blur	0.899
Color jitter	0.830
Zoom blur	0.811
Earth background	0.427
Gaussian noise	0.072
Impulse noise	0.071
Spatter	0.063

Table 9.10: (Lack of) Robustness of an OD network trained on *Envisat-1* in terms of mean IoU across images containing corresponding augmentations

9.2. Keypoint Detection

The next component in the pose estimation pipeline comprises of the KD network described in Chapter 5. The KD networks (*HRNet*, *HRNet-small* and *HRNet-smaller*) are trained and the resulting network models are evaluated for accuracy and robustness on *SPEED* and the Envisat image datasets. Recall that the KD network uses bounding box information to crop an RoI in the image that contains the spacecraft. During training, the true bounding box coordinates are used to crop a corresponding portion of the image and resized to the fixed input size of the KD network model. To avoid deformation in the cropped image, the input aspect ratio is preserved during the cropping operation. As an example, for a KD model with an input aspect ratio of 1:1, a bounding box of width 150 px and height 120 px results in an RoI crop of 150 px × 150 px, around the center of the original bounding box. This RoI is resized to fit the input window of the KD network model.

The accuracy of KD is evaluated using the mean and standard deviation of the keypoint localization error in the original image. Since this error is subjective to the original image size, it is not entirely reflective of the KD network's performance. Recall that *SPEED* images are of size 1920 px × 1200 px and the Envisat images are of size 512 px × 512 px. To allow comparison of performance agnostic to the image size, the input-normalized localization error is computed. The input-normalized errors are evaluated for the network input size, and therefore provide a consistent reference of the KD performance across datasets. For fair evaluation and comparison of the trained network models, the experiments in this section use true bounding box information to crop the RoI, so that the OD performance is decoupled from the KD performance. As in the previous section, the networks are first benchmarked on *SPEED* and evaluated further for accuracy and robustness for the Envisat target use-case. The OD and the KD network models are further integrated and evaluated for pose estimation in Section 9.3.

9.2.1. SPEED Benchmark

Accuracy Assessment

Accuracy assessment is conducted to determine the effect of pixel-level data augmentations and the network size. Due to the larger memory and computation requirement of the HRNet architecture, other design aspects like input size and pre-training are also important. However, these are presented in Appendix E. For the experiments and results below, input size is fixed at 256 px x 256 px and it is specified if a network is pretrained, wherever applicable.

(a) Pixel-level data augmentations: As with the experiments on the OD network, the effect of adding pixel-level augmentations is assessed. The HRNet-W32 network is trained with additional **DA-1** augmentations that randomize the brightness, contrast, and saturation in the training images. Table 9.11 shows the evaluation results of the trained SPD-KP-3 network model, compared with the SPD-KP-2 network model trained earlier on affine augmentations only. Note that both the networks use ImageNet pre-trained weights (ref. pre-training in Appendix E) to initialize the training.

Identifier	Network	Augmentations	Error (px)	Input-normalized error (px)	Input-normalized median error (px)
SPD-KP-2	HRNet-W32	**DA-0**	9.18 ± 27.44	2.63 ± 6.53	1.06
SPD-KP-3	HRNet-W32	**DA-0 + DA-1**	**8.04 ± 23.72**	**2.36 ± 5.92**	**1.03**

Table 9.11: Performance comparison on *SPEED* for data augmentations with the standard HRNet-W32 network

The SPD-KP-3 network model with additional pixel-level augmentations improves the KD accuracy as well as precision. The SPD-KP-3 model restricts outlier errors as seen by the improvement in the mean and standard deviation, while also improving the median error. While the marginal performance boost justifies the inclusion of pixel-level data augmentations in training, the benefit of these augmentations is realized more with respect to robustness, as witnessed earlier for OD in Section 9.1.1. The effect of these augmentations on robustness of detection are presented in the following experiments.

(b) Network Size The standard HRNet-W32 network imposes significant memory and computation requirement for the KD step. In an attempt to utilize more efficient sized network, scaled down alternatives were designed within the HRNet architecture and introduced in this work (see Section 5.2.5). The new networks HRNet-small and HRNet-smaller are trained on the *SPEED* training images and compared with standard HRNet-W32. Each of the networks are trained without using ImageNet pre-trained weights for fair comparison, as ImageNet pre-training for the custom networks (HRNet-small and HRNet-smaller) is not possible in this work. Additionally, both **DA-0** and **DA-1** type augmentations are used during training. Table 9.12 presents the evaluation results on the validation set for the trained models of the aforementioned network sizes.

Identifier	Network	Parameters (Mn)	FLOPs (Bn)	Error (px)	Input-normalized error (px)	Input-normalized median error (px)
SPD-KP-4a	HRNet-W32	28.5	9.5	**10.69 ± 28.44**	**3.76 ± 14.03**	**1.06**
SPD-KP-5	HRNet-small	5.2	3.0	11.11 ± 33.09	3.69 ± 16.13	1.57
SPD-KP-6	HRNet-smaller	**1.6**	**1.8**	18.96 ± 36.90	6.44 ± 10.85	3.60

Table 9.12: Performance comparison on *SPEED* for HRNet network sizes

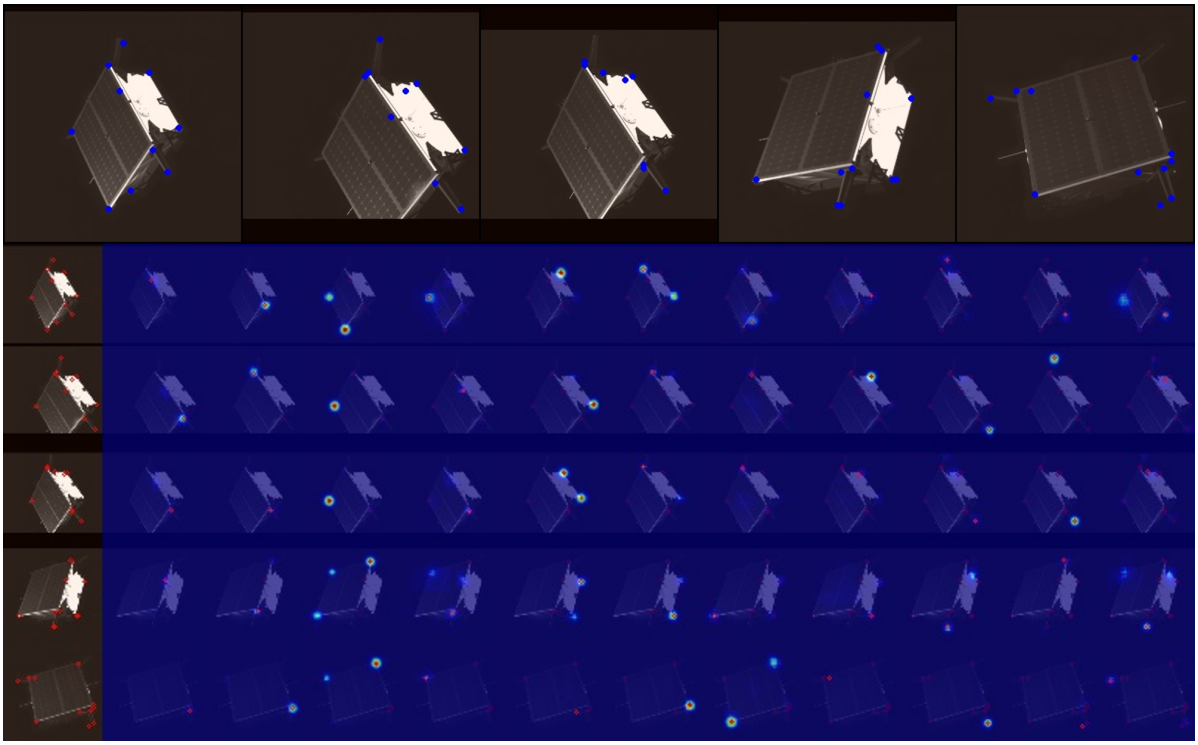
Among the two custom networks, the SPD-KP-6 model using HRNet-smaller shows significant degradation of accuracy in all respects. On the other hand, the SPD-KP-5 network model using the down-scaled *HRNet-small* network provides accuracy performance comparable to that of the SPD-KP-4a model, which uses the larger HRNet-W32 network. The accuracy difference is less than 1 px in mean error, while the input-normalized mean and median error are of sub-pixel magnitude. The minor accuracy loss is achieved with SPD-KP-5 while greatly increasing the memory and computation efficiency. The SPD-KP-5 uses less than a fifth of the total parameters and less than a third of the total FLOPs used by SPD-KP-4a.

In the experiments, the newly introduced *HRNet-small* provides an optimal speed vs accuracy tradeoff for keypoint detection. Given desirable accuracy, the *SPD-KP-5* model must be further assessed for robustness and compared to other candidate models. The SPD-KP-6 model using the other custom network, due to its performance loss, is not selected for further analysis.

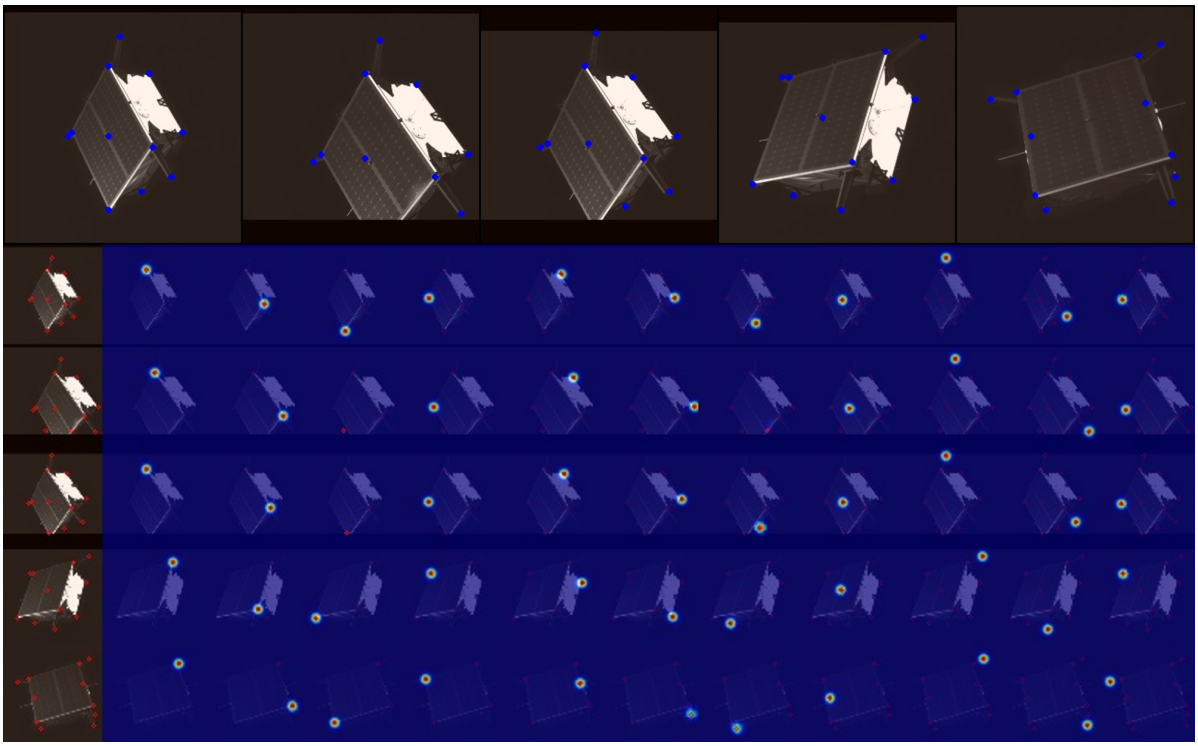
Robustness Assessment

Similar to the robustness assessment for OD, the candidate KD network models are tested on the five lab generated images of the *real* image set in *SPEED*. Note that the images are neither trained on nor encountered by the network models before. Table 9.13 shows the KD results for the trained SPD-KP-2, SPD-KP-3 and SPD-KP-5 network models, that provide superior performance in the synthetic validation set.

Between the SPD-KP-2 and SPD-KP-3 models that use the larger HRNet-W32 network, the importance of including pixel-level data augmentations is realized. The SPD-KP-2 model that was trained only on the affine augmentations, fails to transfer the KD performance to real images that contain artefacts not observed during training. The lack of robustness can be explained by the fact that not introducing pixel-level augmentations results in a network model that overfits on the pixel-level variation in the synthetic images, and fails on



(a) SPD-KP-2



(b) SPD-KP-3

Figure 9.6: Keypoint detection results from lab generated *SPEED real* set images visualizing keypoint and heatmap outputs

.

Identifier	Network	Augmentations	Error (px)	Input-normalized error (px)	Input-normalized median error (px)
SPD-KP-2	HRNet-W32	**DA-0**	411.75 ± 204.17	65.49 ± 27.11	27.10
SPD-KP-3	HRNet-W32	**DA-0 + DA-1**	**25.35 ± 14.38**	**4.07 ± 2.09**	**2.09**
SPD-KP-5	HRNet-small	**DA-0 + DA-1**	**53.72 ± 37.80**	**8.21 ± 5.05**	**7.85**

Table 9.13: Robustness assessment on lab generated *SPEED real* set

other images. On the other hand, the SPD-KP-3 model trained on additional pixel level data augmentations (**DA-1**) is able to provide keypoint detections for the previously unseen real images with lower accuracy loss, demonstrating better transfer to reality. Further, SPD-KP-3's generalization ability is evident from the input-normalized mean error in detection error, which is very close to the corresponding error magnitude on the synthetic validation set (Table 9.11). Fig. 9.6 shows detections and heatmap outputs from the KD network. The detection performance can be explained by visualizing the keypoint and the heatmaps. The keypoints detected in the images show spurious detection using SPD-KP-2, as seen in Fig. 9.6(a). The heatmaps from SPD-KP-3 model show a clean Gaussian distribution for most keypoint detections, demonstrating it's ability to focus on the right parts of the image despite texture variation, as it is trained with pixel level data augmentations. On the other hand, the heatmaps from SPD-KP-2 show that the network is unable to identify keypoints in most cases, as it encounters texture and pixel level variation different from that in the training set. In other cases, the network is confused by the image texture and shows multiple activation locations. This further reinforces the role of augmentations in making deep neural networks robust between synthetic and real environments.

Between the SPD-KP-3 and SPD-KP5 models that differ in terms of underlying network size, the accuracy difference increases for real images. The mean and input normalized errors are more than twice as large for the SPD-KP-5 that utilizes HRNet-small. While SPD-KP-5 provides detection accuracy closer to the larger SPD-KP-3 model on the synthetic images, the network's robustness is considerably lower. This can be explained by the fact that due to a smaller network (parameter count) the trained model approximates the non-linear mapping more coarsely than the larger network model, resulting in lower robustness. Note that the SPD-KP-5 model was not pre-trained, which may further degrade its robustness. However, notice that the smaller network although inaccurate performs significantly better than the larger SPD-KP-2 network model trained with only affine augmentations.

Given the accuracy and robustness results, the SPD-KP-3 provides the best accuracy and robustness in KD, while being extremely demanding in terms of memory and computation requirements. On the other hand, SPD-KP-4 provides comparable accuracy in the validation set with an optimal balance of memory and computation requirements, while providing lower robustness to unmodeled variations and artefacts in the real images. These two network models are taken as baseline models to benchmark the pose estimation performance with various state-of-the-art results on *SPEED* in Section 9.3.

Network Comparison

The selected network models are compared against the KD networks used in state-of-the-art pipelines [46, 78], which use a KD network. Note that Sharma et al. [45] do not use a KD network, but apply a hybrid classification-regression approach to estimate the pose elements directly after the OD network. Further, the isolated keypoint network evaluations are not provided in Chen et al. [78] and Park et al. [46]. Therefore, only the network specifications are compared at this stage, as summarized in Table 9.14

Note that Chen et al. [78] use an ensemble of 6 trained networks, with an input size of 768 px x 768 px, resulting in each of the network requiring 85.3 Bn FLOPs per forward pass. The use of HRNet is optimized to manage the model size and computation requirement compared to the pipeline adapted in Chen et al. [78].

The two baseline models, SPD-KP-5 and SPD-KP-3, are selected for further pose estimation analysis. The SPD-KP-3 model is comparable to the network used by Park et al [46], which is smaller and has similar number of parameters in the network. This allows comparison between the networks with moderate memory and computation requirements. While, the larger SPD-KP-3 model is used to compare with the state-of-the-art accuracy results produced by the ensemble of large networks used by Chen et al. [78]. For reference, the SPD-KP-3 model takes 200 ms on an average per inference on CPU while SPD-KP-5 takes approximately 90 ms.

	Park et al. [46]	R-NAV (SPD-KP-5)	Chen et al. [78]	R-NAV (SPD-KP-3)
Evaluation set	validation subset	validation subset	*train*	validation subset
Network	YOLO (custom)	HRNet-small (custom)	HRNet-W32 [122]	HRNet-W32 [122]
Input size (px)	224 × 224	256 × 256	768 × 768	256 × 256
Ensemble	No	No	6x	No
Parameters (Mn)	5.6 Mn	**5.2**	(6x) 28.5	28.5
FLOPs (Bn)	N/A	**3.0**	(6x) 85.3	9.5

Table 9.14: Comparison of state-of-the-art keypoint detection networks used on *SPEED* dataset. R-NAV identifies the framework designed in this work.

Together, the two baseline models can provide definitive inferences on the design choices and configuration adapted in this work by a comparison with the state-of-the-art with respect to speed and accuracy. In Section 9.3, these KD models are interfaced with the selected OD model (SPD-OD-3) for pose estimation pipeline evaluation.

9.2.2. Envisat: Evaluation and Robustness

The experiments are repeated for Envisat image datasets to evaluate KD performance for the Envisat target use-case in this work. First, the KD networks are trained and evaluated on the *Envisat-1* dataset from Pasqualetto Cassinis et al. []. The evaluation results are used to demonstrate simplicity of the original dataset, when compared with SPEED. Further, the networks are trained and tested with the*Envisat+IC* augmented dataset, introduced in this work. The network models trained on both the datasets are tested on *val*, *val-clean* and *val-challenge* image sets of the *Envisat+IC* dataset to assess the robustness of detection.

Accuracy Assessment

(a) Envisat-1: The networks are trained on 35000 images of the training set in *Envisat-1* to detect 16 points of the simplified 3D wireframe model of the Envisat spacecraft. Subsequently, the evaluation is done on the 9600 images of the validation set. The network training experiments on *Envisat-1* are streamlined, from the inference drawn for KD in *SPEED*. However, to draw a comparison between the *Envisat-1* and *SPEED* datasets, two experiments are conducted. First, the effect of pixel-level data augmentation is evaluated on a pre-trained HRNet-W32 network and is compared with the same network trained with only affine augmentations. Table 9.15 presents the comparison between the ENV1-KP-1 network trained with **DA-0** augmentations and the ENv1-KP-2 network trained with **DA-0 + DA-1** augmentations. Next, the three network sizes (HRNet, HRNet-small, and HRNet-smaller) are trained. The resulting network models are evaluated to determine the effect on network size on the accuracy of detections on *Envisat-1*. Table 9.16 presents the results and comparison between the resulting trained models.

Identifier	Network	Augmentations	Error (px)	Input-normalized mean error (px)	Input-normalized median error (px)
ENV1-KP-1	HRNet-W32	**DA-0**	0.94 ± 0.46	0.93 ± 0.14	0.92
ENC1-KP-2	HRNet-W32	**DA-0 + DA-1**	0.94 ± 0.46	0.94 ± 0.14	0.93

Table 9.15: Performance comparison of training data augmentations for networks trained on *Envisat-1*

In Table 9.15,it is observed that adding pixel-level data augmentations has negligible effect on the evaluation accuracy. This shows that the images in the dataset have little or no pixel-level variation between training and evaluation. As noted in the evaluation in Table 9.7, this conclusion is applicable to both validation as well as test sets in *Envisat-1*. Notice that the input-normalized mean and standard deviation of the error is significantly lower than of similar network models on *SPEED*. The mean error in the original image is similar to input normalized mean error because the ground truth RoI crop of a constant aspect ratio from the images is close to the input window i.e. 256 px × 256 px. Further, as shown in Table 9.12, the accuracy of trained models

Identifier	Network	Error (px)	Input-normalized error (px)	Input-normalized median error (px)
ENV1-KP-2	HRNet-W32	**0.94 ± 0.46**	**0.94 ± 0.14**	**0.93**
ENV1-KP-3	HRNet-small	0.98 ± 0.42	1.01 ± 0.23	0.97
ENV1-KP-4	HRNet-smaller	2.75 ± 8.38	2.87 ± 8.15	1.79

Table 9.16: Performance comparison on *Envisat-1* for input size variation for the standard HRNet-W32 network

of all three network sizes provide high accuracy on *Envisat-1*, with an especially low standard deviation.

The comparison of trained network models' performance on *Envisat-1* with those evaluated on *SPEED*, further demonstrate the simplicity of the *Envisat-1* dataset and the associated ease of keypoint detection.

(b) Envisat+IC: For the newly created *Envisat+IC*, the images are trained on 35000 augmented images in the training set and evaluated on 5000 augmented images of the validation set. The networks are trained to detect 12 points of the Envisat spacecraft (eliminating non-rigid solar panel keypoints). The network is trained with on affine augmentations. The pixel-level data augmentations are not included in the training as the augmentations in **DA-1** i.e. brightness, contrast, and saturation, are already randomized in the dataset for the same range of variation. Since the dataset models these and other augmentations implicitly, the need for pixel-level **DA-1** type augmentations are not relevant.

The *Envisat+IC* images include Earth in the background and image corruptions that are expected to introduce complexity in the images, affecting the effectiveness of the KD network. To evaluate this, the three network sizes are trained and evaluated on the *Envisat+IC* validation set. Table 9.17 shows the evaluation results of the corresponding trained network models.

Identifier	Network	Error (px)	Input-normalized error (px)	Input-normalized median error (px)
ENVIC-KP-5	HRNet-W32	**0.58 ± 0.59**	**1.10 ± 1.23**	**0.96**
ENVIC-KP-6	HRNet-small	1.65 ± 3.70	3.21 ± 7.29	1.86
ENVIC-KP-7	HRNet-smaller	2.53 ± 4.87	4.90 ± 9.47	3.09

Table 9.17: Performance comparison on *Envisat+IC* for input size variation for the standard HRNet-W32 network

It is evident that the Earth background and image corruptions introduce patterns in the data that are learned more accurately by the larger ENVIC-KP-5 using HRNet-W32 rather than the smaller networks. Further, the mean and standard deviation increase with decreasing network size. Notice that the mean error in the original image size is lower than the input normalized error. This results due to the fact that the network model is now detecting keypoints in a smaller RoI that contains the main body and the SAR antenna of the Envisat spacecraft. The cropped RoI is most often smaller than the input window. Therefore, the input-normalized mean and standard deviation are scaled down when projecting back to the original image rather than scaled up. This is why input-normalized errors are crucial in understanding the detection accuracy more objectively than the keypoint detection errors in the original image.

As earlier, the network models based on HRNet-W32 and HRNet-small are used as baseline for Envisat

Robustness Assessment

The network models trained on *Envisat-1* (ENV1-KP-2) and *Envisat+IC* (ENVIC-KP-5 and ENVIC-KP-6) that provide desirable accuracy on the respective validation sets are assessed for robustness and generalization. Each network model is evaluated on the *val-IC*, *val-clean* and *val-challenge* sets designed for robustness evaluation. Table 9.18 reports the corresponding results. Note that ENV1-KP-2 detects 16 points in an RoI containing the entire Envisat spacecraft. On the other hand, ENVIC-KP-5 and ENVIC-KP-6 detect 12 points from the RoI containing only the main body and SAR antenna.

The ENVIC-KP-5 using the standard HRNet-W32 is superior across all the image sets and shows high accuracy as well as robustness to variations in the image. In contrast, even though the larger HRNet-W32 network is trained with **DA-0 + DA-1** type augmentations on *Envisat-1* images, the ENV1-KP-2 model does not gen-

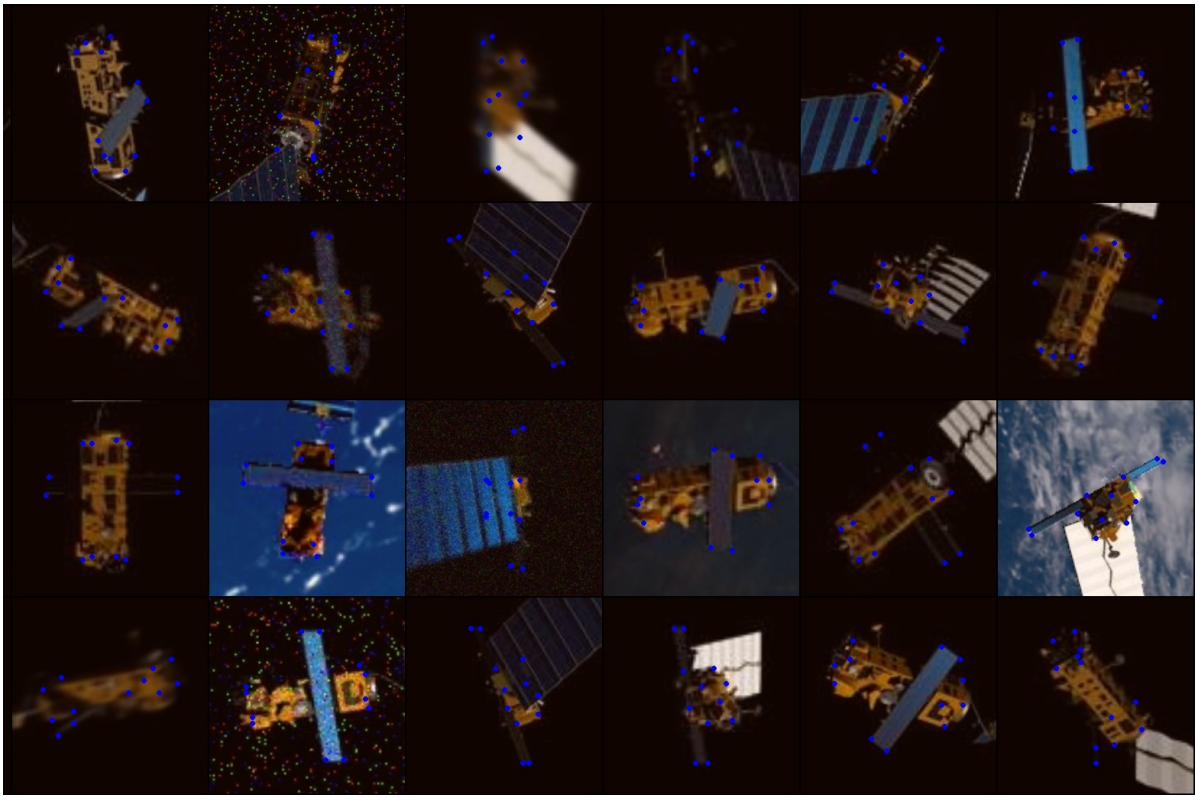


Figure 9.7: Visualization of keypoint detections within detected RoI from ENVIC-KP-5

eralize well to augmented images in the *val-IC* and *val-challenge* sets. The ENV1-KP-2 model's mean and standard deviation of the keypoint error are significantly higher with a nominal median error, indicating very poor accuracy and higher amount of outliers in the some of the images. This relates to the fact that CNNs can tackle some corruptions more than others as demonstrated in Table 9.10. Note that the input normalized error of the network models on clean images is very similar, while the mean error in the image is lower for ENVIC-KP-5. This is because the RoI for 12 point detection is magnified compared to that of the entire spacecraft (16 points). However, the magnification result may not always improve the overall accuracy, especially for lower resolution images that may lose local information during the RoI upsampling to the network input. The ENVIC-KP-6 model based on HRNet-small has accuracy decreases with increasing corruption magnitude from *val-clean* to *val-challenge* sets. However, it shows higher robustness compared to the large ENV1-KP-2 network model. Both the network models trained on *Envisat+IC* images show generalization capability across clean as well as highly-corrupted images. Fig. 9.7 shows the keypoint detection within the RoI as seen by the KD network in corrupted images using ENVIC-KP-5. Notice that the network also produces accurate detections in the presence of poor illumination, varying brightness/contrast conditions, and body occlusion. This suggests that the network can emphasize on global shape of the object and not rely heavily on the local texture cues, which is desirable [149].

The results of this robustness assessment further validates the *Envisat+IC* dataset, which exhibits desirable data distribution from which CNNs can improve generalization and robustness in learning. For further experiments and analyses, the network models trained on *Envisat-1* dataset are not included due to the aforementioned deficiencies. The pose estimation and state estimation analysis for Envisat target use-case is streamlined by considering only the models trained on *Envisat+IC* i.e. ENVIC-KP-5 which uses the larger HRNet-W32 and ENVIC-KP-6 model which uses HRNet-small. The choice of two network sizes is motivated by their desirable performance as well as the intention to provide accuracy vs speed trade-off in pose and state estimation.

Network	Image set	Error (px)	Input-normalized error (px)	Input-normalized median error (px)
ENV1-KP-2	*val-clean*	0.94 ± 0.46	0.93 ± 0.14	0.93
	val-IC	22.54 ± 41.96	22.09 ± 36.86	1.09
	val-challenge	28.03 ± 43.9	28.17 ± 39.10	1.21
ENVIC-KP-5	*val-clean*	**0.49 ± 0.15**	**0.90 ± 0.18**	**0.88**
	val-IC	**0.58 ± 0.59**	**1.10 ± 1.23**	**0.96**
	val-challenge	**0.82 ± 1.69**	**1.61 ± 3.70**	**1.02**
ENVIC-KP-6	*val-clean*	1.36 ± 3.66	2.64 ± 7.20	1.51
	val-IC	1.65 ± 3.70	3.21 ± 7.29	1.86
	val-challenge	2.59 ± 5.16	5.12 ± 10.20	2.23

Table 9.18: Robustness assessment of trained network models *Envisat+IC*

9.3. Pose Estimation

The experiments in this section are aimed at evaluation of the full pose estimation pipeline, with a loose[1] integration of OD model, KD model and the pose solver. The trained network models for OD and KD for *SPEED* as well as *Envisat+IC* are integrated with the MLPnP solver. The MLPnP solver is used with the modification proposed in this work to tackle the shortcomings of the original algorithm [58]. Recall that the modified MLPnP algorithm was validated and compared with CEPPnP, another state-of-the-art covariance-based PnP solver, demonstrating higher robustness to pixel noise.

In the following experiments, the bounding box coordinates estimated by the OD model are used an input to crop the RoI for KD. The coordinates of the keypoints detected by the KD model are fed to the pose solver. In addition, the heatmaps regressed by the KD model are used to quantify a image-plane covariances corresponding to the individual keypoints, as highlighted in Section 5.4. Note that unlike earlier, no ground truth information is provided to any part of the pipeline. The pose vector estimated by the pose solver is in the camera frame of reference. The pipelines are evaluated on translation and rotation error metrics defined defined in Eq. 7.31 and 7.32.

In order to reject outliers, a heuristic strategy is implemented and tuned to exclude low confidence detections during the solver routine, using confidence scores of the detections. Since the MLPnP solver only needs a minimum of 6 non co-planar points, it is possible to reject points from the wireframe model based on the confidence score values output by the KD network. In general, 8-9 points are provided for most images to allow enough redundant points for the solver to overcome viewpoint ambiguities. In the worst case, when multiple keypoint heatmaps are empty (no activation), the number of selected keypoints is lowered to the limit i.e. 6 points. This can occur when poor box detection crops out a large part of the spacecraft.

Finally, the 3D body-frame coordinates from the wireframe model, the detected image-plane keypoint coordinates and the image-plane covariances are used by the modified MLPnP routine to estimate the pose and the pose uncertainty.

9.3.1. SPEED Benchmark Comparison

Accuracy

For *SPEED*, two pose estimation pipeline configurations are used, named PE-A and PE-B. PE-A consists of SPD-OD-5 OD model, SPD-KP-5 model for KD and the modified MLPnP solver. PE-B consists of SPD-OD-5 model for OD, SPD-KP-3 model for KD and modified MLPnP solver. Note that the two pipelines only differ in the KD models, wherein PE-B uses a model based on HRNet-small network, PE-A uses a model based on the larger HRNet-W32 network. First, the two pipeline alternatives are benchmarked on the synthetic validation set for accuracy of pose estimation and compared to state-of-the-art pose estimation results [45, 46, 78]. Subsequently, the pipeline configurations are tested for robustness in pose estimation using the *real* set images in *SPEED*.

[1] Due to library and programming language constraints, the data between the components is interchanged through consistent data (JSON) files. Development of direct links between the components or the "tight" integration with cross-library compatibility is beyond the scope and relevance of this work.

Table 9.19 presents a detailed benchmark comparison of the pose estimation accuracy of the two pose estimation pipelines. The PE-A configuration that uses a larger network model has pose estimation accuracy comparable with that of the state-of-the-art [78], while demanding significantly lower memory and computation. Unlike the pipeline in Chen et al. [78] which uses an ensemble of 6 networks for each step, larger input sizes and a more demanding pose solver based on Lavenberg-Marquardt method, the PE-A pipeline is significantly more optimized. Consequently, the model size and the computation requirements for CNNs in PE-A are significantly lower, while providing comparable performance. The difference in mean rotation error is less than 1 degree and the difference in boresight range estimate is around 10 cm for a target range between 3 m and 30 m across 2400 validation images. On the other hand PE-A outperforms the other two pipelines from Park et al. [46] and Sharma and D'Amico [45], but demands modestly higher model size and computation. PE-A achieves a final SLAB/ESA score of 0.026, closely trailing the state-of-the-art.

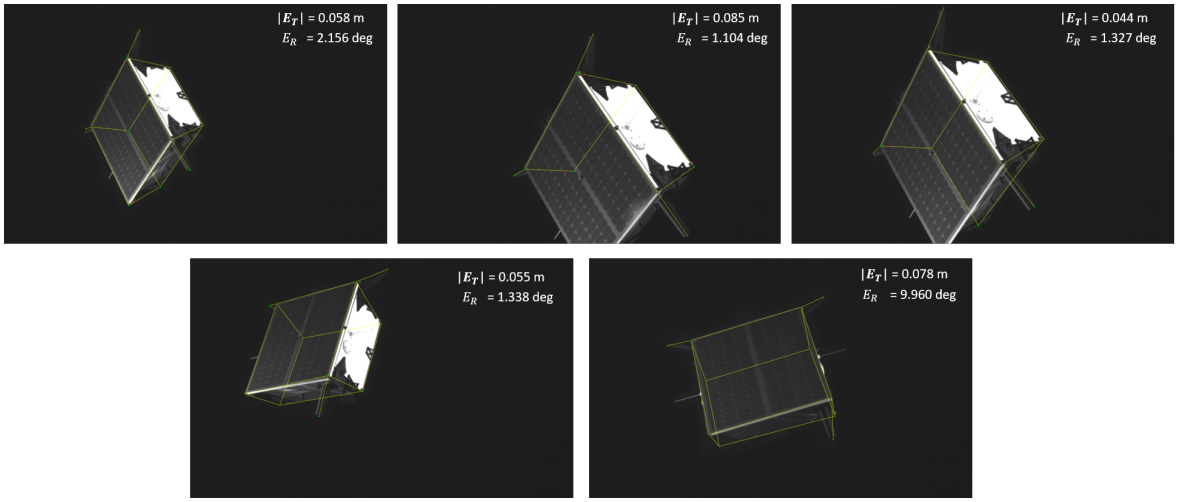
The PE-B configuration features a KD model based on custom HRNet-small introduced here. The configuration is comparable in terms model size and computation to the pipeline from Park et al. [46]. The PE-B configuration uses moderately lower number of parameters and expectedly lower FLOPs, while achieving a better SLAB/ESA score. In comparison, the PE-B configuration is significantly more accurate in estimating the attitude with around 2 degree difference in mean and median of E_R. The mean translation error is similar for PE-B than in [46], while the median translation error is lower. This suggests that the mean is skewed by certain images that exhibit poorer accuracy in the position estimate. The PE-B configuration achieves a SLAB/ESA score of 0.050, outperforming the Stanford Rendezvous Lab's pipeline in [46]. Note that the pipeline in [46] uses EPnP pose solver, which is similar to MLPnP in terms of computation. However, the EPnP's pose performance degrades significantly with increasing noise as well as relative range (see Section 7.2, 7.4 and Urban et al. [58]). This may explain a factor of performance loss in the corresponding results. Interestingly, that also means that for higher relative ranges the pose estimation pipeline in [46] may not provide sufficient accuracy despite the accuracy of CNN detections.

For the evaluation results presented here, a heuristic approach to keypoint selection/rejection (tweaking n of the PnP problem) plays a crucial role. This process essentially uses confidence scores provided by the KD network and rejects poor keypoint detections, resulting from adverse texture, occlusion or inaccurate box detections. This restricts their effect on the pose estimate. To realize the importance of the heuristic keypoint filtering, Table 9.20 presents a performance comparison of the PE-B pipeline with and without heuristic keypoint selection on the synthetic validation set. In absence of the heuristic strategy, the mean of translation and rotation error degrades significantly. More importantly, the mean error is heavily skewed by poor pose estimates in images with detection outliers, since the median errors are similar.

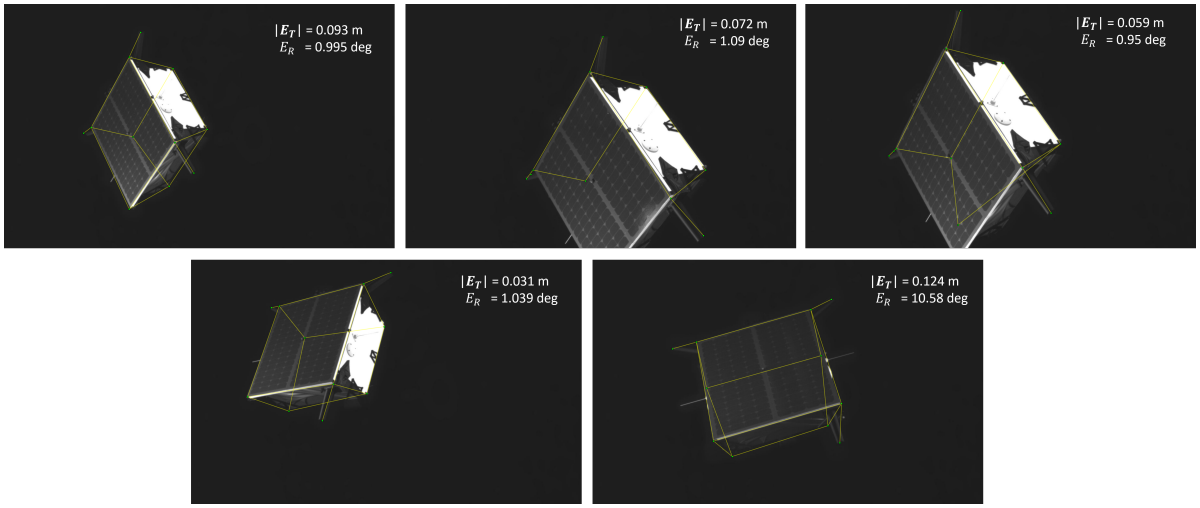
The results of the experiments demonstrate that the pose estimation pipeline is accurate. The performance of the two pipeline compare well with the state-of-the-art results.

	Sharma and D'Amico [45]	Park et al. [46]	R-NAV (PE-B)	Chen et al. [78]	R-NAV (PE-A)
	test set	validation subset	validation subset	*train* set	validation subset
SLAB/ESA Score	N/A	0.073	0.050	0.012	0.026
Mean $\mathbf{E}_T$ (m)	[0.055, 0.046, 0.780]	[0.010, 0.011, 0.210]	[0.010, 0.010, 0.248]	[0.004, 0.004, 0.035]	[0.010, 0.011, 0.141]
Median $\mathbf{E}_T$ (m)	[0. 024, 0.021, 0.496]	[0.007, 0.007, 0.124]	[0.003, 0.003, 0.034]	[0.003, 0.003, 0.013]	[0.002, 0.002, 0.030]
Mean E_R (deg)	8.425	3.097	1.158	0.727	0.807
Median E_R (deg)	7.069	2.568	0.773	0.521	0.598
Pose Solver	None	EPnP [47]	MLPnP [58]	SA-LMPE (custom)	MLPnP [58]
CNN parameters (OD+KD)	N/A	5.5 Mn + 5.6 Mn	3.2 Mn + 5.2 Mn	(6x) 26.2Mn + 28.5 Mn	3.2 Mn + 28.5 Mn
CNN FLOPs (OD+KD)	N/A	N/A	0.5 Bn + 3.0 Bn	(6x) 159.1 Bn + 85.3 Bn	0.5 Bn + 7.1 Bn
Network Identifiers	-	-	SPD-OD-3 + SPD-KP-5	-	SPD-OD-3 + SPD-KP-3

Table 9.19: Pose Estimation performance comparison with state-of-the-art results on *SPEED*



(a) PE-A



(b) PE-B

Figure 9.8: Pose estimation results on *real* image set in *SPEED* for the corresponding pose estimation pipeline configuration

Robustness

Given the accuracy results on the synthetic validation set, the PE-A and PE-B are evaluated for robustness on *SPEED real* set. Fig 9.8 shows the evaluation with errors annotated in the respective real images. The PE-A using HRNet-W32 achieves a SLAB/ESA score of 0.065, while the PE-B pipeline achieves a score of 0.071 for the 5 real images. Both networks perform almost equally well on the *real* set with subjective differences. Notice that the smaller network is made more robust by heuristic point selection, even though some of the keypoint detections have more error. Therefore one cannot be called superior than the other. Notice the poor attitude estimation accuracy in one of the images. This is likely a result of view-point ambiguity for the detected set of points.

A relevant inference that can be drawn from the robustness assessment is that both the networks show desirable transfer to previously unseen real images, since the SLAB/ESA scores are close to the corresponding scores on the synthetic images (Table 9.19), especially for the PE-B pipeline which requires considerably lower memory and computation in comparison to PE-A. However, given the relative distance of less than 5 m, both the pipelines have attitude estimation accuracy that does not necessarily meet the typical requirements, when compared with values in Table 2.1. Note that the robustness of the pipeline results from the use of pixel level augmentations such as randomized brightness and contrast. Consequently, the performance

Pipeline	Heuristic rejection	Mean $\|\mathbf{E}_T\|$ (m)	Median $\|\mathbf{E}_T\|$ (m)	Mean E_R (deg)	Median E_R (deg)	SLAB/ESA score
PE-B	Y	**0.249**	**0.035**	**1.158**	**0.773**	**0.050**
PE-B	N	1.779	0.040	3.730	0.994	0.379

Table 9.20: Comparison of performance of PE-B pipeline with and without Heuristic rejection

is limited by additional artefacts, that may be present in the real images. A possible remedy would be improve the *SPEED* dataset to include representative image corruptions, as done for the Envisat target case with *Envisat+IC*.

9.3.2. Envisat: Evaluation and Robustness

For *Envisat+IC* pose estimation analysis, two pipeline configurations that differ in the KD model size are used, as earlier. PE-C pipeline uses ENVIC-OD-5 as the OD model with ENVIC-KP-5 as the KD model (HRNet-W32) and MLP*n*P. PE-D pipeline uses ENVIC-OD-5 as the OD model with ENVIC-KP-6 KD model (HRNet-small) and MLP*n*P. The evaluation is done on the three validation sets of the *Envisat+IC* dataset, which together allow accuracy and robustness assessment. The pose estimation performance is evaluated for the 12 point wireframe model (excluding non-rigid solar panel) of the Envisat spacecraft using ENVIC-OD-5 for OD and ENVIC-KP-5 for KD. The performance metrics are summarized in Table 9.21.

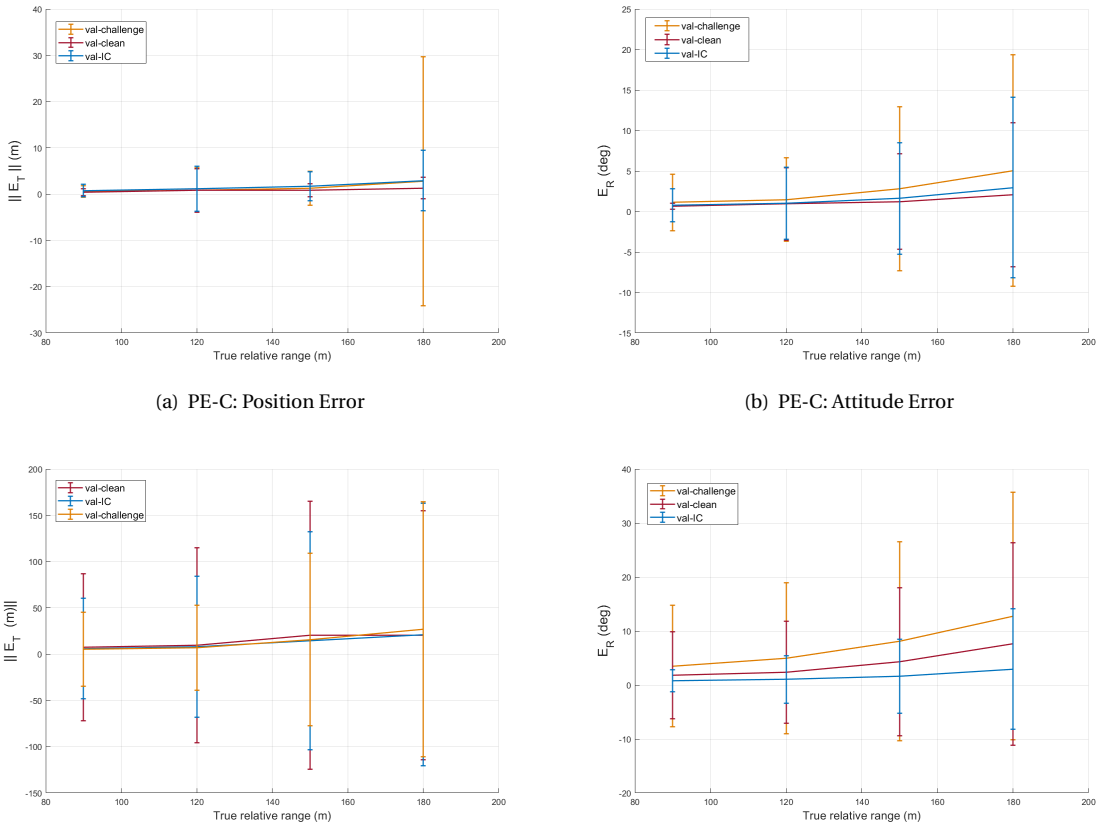
Pipeline	Test set	Mean E_R	Median E_R	Mean $\mathbf{E}_T$	Median $\mathbf{E}_T$
PE-C	*val-clean*	1.223	0.553	[0.0298, 0.030, 0.841]	[0.020, 0.021, 0.474]
	val-IC	1.609	0.630	[0.039, 0.038, 1.380]	[0.024, 0.025, 0.576]
	val-challenge	2.640	0.711	[0.049, 0.049, 1.658]	[0.027, 0.028, 0.675]
PE-D	*val-clean*	4.079	0.833	[0.465, 0.522, 14.602]	[0.035, 0.033, 0.676]
	val-IC	5.745	0.992	[0.410, 0.421, 12.553]	[0.047, 0.042, 1.038]
	val-challenge	8.348	1.188	[0.400, 0.377 , 13.091]	[0.059, 0.053, 1.328]

Table 9.21: Pose Estimation performance on *Envisat+IC* image sets

The use of larger (HRNet-W32) ENVIC-KP-5 model in PE-C shows clear advantage over the ENVIC-KP-6 model in PE-D that utilizes HRNet-small network. In addition to high accuracy, the PE-C pipeline also show desired robustness in pose estimation across clean images in *val-clean* and heavily corrupted images in *val-challenge* sets. For the PE-D pipeline, the mean errors in translation and rotation are significantly higher. However, the PE-D pipeline has median errors very close to that in PE-C. Therefore, the smaller KD model performs poorly on certain images, skewing the mean error to a high value.

Recall that the images contain Envisat spacecraft between 90 m and 180 m. The pose estimation performance for the 12 point main body wireframe with the PE-C pipeline for such a large relative range, is sufficiently high. For PE-D however, the pose estimation accuracy is poor. This can be further analysed by plotting mean and standard deviation of errors as a function of true relative range. Fig. 9.9 shows the error variation as a function of true relative range for both the pipeline. Notice that PE-D provide position accuracy of less than 5m and attitude accuracy of less 10 degrees upto 150 m, even for heavily corrupted images of the *val-challenge* set. At 180 m relative range, the pose estimation performance decreases significantly for *val-challenge* images, but performance on other images show moderate to desirable performance levels. This variation of error across the validation sets, show desired robustness and accuracy with PE-C pipeline. On the other hand, PE-D pipeline shows very low accuracy in pose estimation as well as poor robustness. Interestingly, the position error shows higher standard deviation for clean images than corrupted images. This undesired behaviour is expected to be produced by poor generalization in keypoint detection combined with high relative distance. At higher distance the error in keypoint localization contributes more to the pose estimation error. This is made more sensitive as the 12 point target (Envisat excluding solar panel) is present in a very small pixel region and at higher distances.

Fig. 9.10 visualizes pose estimation results on a collection of corrupted, augmented and clean images. The models trained on *Envisat-1* are not presented as they lack the keypoint accuracy in most cases for the MLPnP to solve the PnP problem viably. The models trained on Envisat-1 are therefore ignored for the subsequent analyses.



(a) PE-C: Position Error

(b) PE-C: Attitude Error

(c) PE-D: Position Error

(d) PE-D: Attitude Error

Figure 9.9: Pose Estimation performance on *Envisat+IC* image sets. *val* set contains similar image corruptions, *val-clean* contains clean images and *val-challenge* contains higher magnitude of corruptions than the training set

It is clear that as larger and deeper networks perform well when images contain challenging artefacts. This is expected as deeper networks are able to represent and learn more complex relations that exist between image artefacts and the output detection. The smaller networks tend to overfit on the given distribution reducing robustness despite desired accuracy, or underfit by not being able to learn complex relations in the data to provide sufficient accuracy. Therefore, the choice of network size should be made only after rigorous robustness assessment. Generalization of CNN performance forms should be at the core of learning-based navigation design.

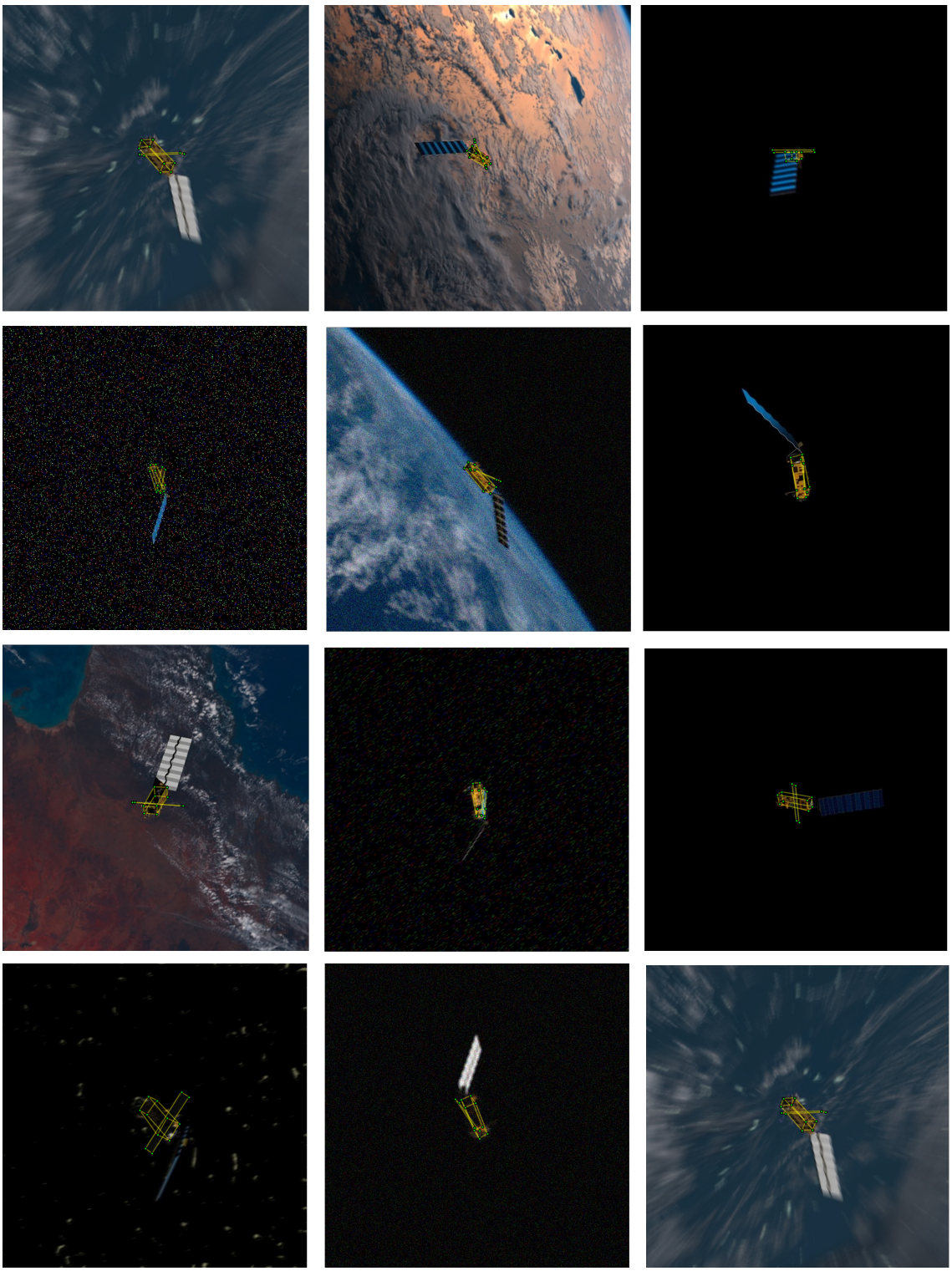


Figure 9.10: Visualization of Envisat main body projections from pose estimation results in *Envisat+IC*

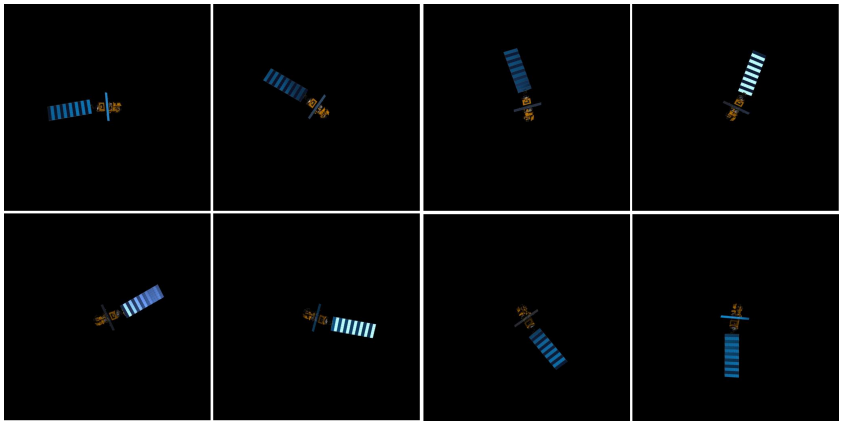


Figure 9.11: Images from *traj_150m* set emulating V-bar trajectory for the Envisat target. Images shown are a subset of total 36 images.

9.4. State Estimation

To assess the framework performance to realistic navigation scenarios, the pose estimation pipeline is connected with the state estimator i.e. the loosely-coupled MEKF designed in Chapter 8. An image sequence of a perturbation-free V-bar trajectory, available from *traj_150m* set in *Envisat-1*, is used to emulate a tumbling Envisat spacecraft. The V-bar trajectory assumes the servicer spacecraft at a steady hold in the along-track direction at 150 m relative to the Envisat spacecraft ($\mathbf{v} = 0$). The images simulate a constant rate roll of the Envisat spacecraft, with 36 images describing one full rotation. Fig. 9.11 shows the intermediate images in the set.

Given 36 images in the measurement sequence, two navigation scenarios are constructed with image measurement frequency of 1 Hz and 0.5 Hz respectively. Consequently, the roll rate of the Envisat spacecraft for the scenarios is $||\boldsymbol{\omega}|| = 10 deg/s$ and $||\boldsymbol{\omega}|| = 5 deg/s$ respectively, since the number of images is fixed. The internal propagation frequency for the estimator is fixed at 1 Hz. Table 9.22 describes the initial conditions for the scenario. The rotational state variable is described in the Envisat body frame and the translation state variables are described in the LVLH frame centered at the servicer. The initial attitude of the Envisat spacecraft is described in terms of Euler angles for clarity. Further, the servicer body axes are assumed to coincide with the camera frame axes, and the camera boresight points in the along-track direction observing the Envisat rotating at 5 deg/s along the same direction.

The simplicity of this dynamical scenario is necessary to provide first order navigation anlaysis of the new framework with the novel uncertainty-aware loosely coupled approach to state estimation. As described earlier, in the currently designed loosely-coupled state estimator, the MEKF uses psuedo-measurements of pose and the associated uncertainty estimated by the pose solver. By exploiting the measurement covariance estimated in the loop using heatmaps, the state estimator maintains representative uncertainty information to ensure robustness in measurement update step. This is essential to avoid bad navigation solution or filter divergence during high uncertainty intervals that may result due to detection inaccuracy, visibility constraints and adverse optical conditions during operation.

Scenario	dt_{meas} (s)	$\boldsymbol{\theta}_0$ (deg)	$\boldsymbol{\omega}_0$ (deg/s)	$\mathbf{r}_0$ (m)	$\mathbf{v}_0$ (m/s)
1	1	[-180, 30, -80]	[-5.00, -8.53, 1.50]	[0, 150, 0]	[0, 0, 0]
2	2	[-180, 30, -80]	[-2.50, -4.27 , 0.75]	[0, 150, 0]	[0, 0, 0]

Table 9.22: Envisat initial conditions of the V-bar simulation scenarios

Navigation performance

The MEKF is initialized with state vector as described in Table 9.23. Note that the filter state consists of relative position and velocity in the LVLH frame and the attitude and angular rates in the body frame of the servicer. The dynamic noise has standard deviation of 0.0001 m/s in velocity and 2 deg/s in angular velocity. The dynamic covariance is then derived using Eq. 8.30. Since the translation and rotation dynamics are ideal in the original scenario, the noise model in the state propagation adds representative noise to the state. The

scenario is simulated for a quarter-period of the Envisat orbit, wherein the image sequence is repeated until the final time. The simulation is carried out for two cases with PE-C and PE-D pose estimation pipelines, mentioned in the previous experiments. Fig. 9.12-9.15 show the convergence profiles and 3σ bounds of the filtered state for both cases in scenario 1 and 2 respectively.

Despite heavily perturbed initial state, the state estimate converges in all cases. The position estimate error stabilizes with a bias close to 0.2 m in along-track direction and 0.001 m in radial and cross-track direction. The attitude estimate error stabilizes with a bias close to 0.7 degrees about the Euler axis. In general, the steady state error for PE-C is marginally better than for PE-D, while the error with the same pipeline is marginally better for Scenario 1 rather than Scenario 2.

$\mathbf{a}_0$	$\boldsymbol{\omega}_0$ (deg/s)	$\mathbf{r}_0$ (m)	$\mathbf{v}_0$ (m/s)
[0.05, 0.05, 0.05]	[1.15, 1.15, 11.14]	[5, 210, 5]	[0.5, 0.5, 0.5]

Table 9.23: Perturbed initial filter state for the V-bar estimation scenario

To assess the robustness of the loosely-coupled estimator with the respective pose estimation pipelines, a Monte-Carlo simulation was performed by varying the initial state of the filter. Table 9.24 notes the standard deviations of the state variable used for the Monte Carlo routine. The large deviation in the initial state reflects poor knowledge of the initial state on initial image acquisition, as expected for an uncooperative proximity operations scenario. The Monte Carlo simulation results for the respective pose estimation pipelines and scenarios are reported in Table 9.25.

$\sigma_{\mathbf{r_0}}$ (m)	$\sigma_{\mathbf{v_0}}$ (m/s)	$\sigma_{\mathbf{a_0}}$	σ_{ω_0} (deg/s)
[10, 50, 10]	[0.5, 0.5, 0.5]	[0.05, 0.05, 0.05]	[2.9, 2.9, 2.9]

Table 9.24: Standard deviation of Monte Carlo variables

	Scenario	E_R (deg)	$\mathbf{E}_{T_1}$: Radial (m)	$\mathbf{E}_{T_2}$: Along-Track (m)	$\mathbf{E}_{T_3}$: Cross-track (m)
PE-C + MEKF	1	0.2842 ± 0.1780	0.0006 ± 0.0005	0.2021 ± 0.0369	0.0010 ± 0.0002
	2	0.4508 ± 0.1807	-0.0015 ± 0.0006	0.2233 ± 0.0684	0.0004 ± 0.0003
PE-D + MEKF	1	0.4989 ± 0.1780	-0.0008 ± 0.0006	0.2126 ± 0.0489	0.0004 ± 0.0002
	2	0.6129 ± 0.2430	-0.0016 ± 0.0006	0.2365 ± 0.0971	0.0013 ± 0.0003

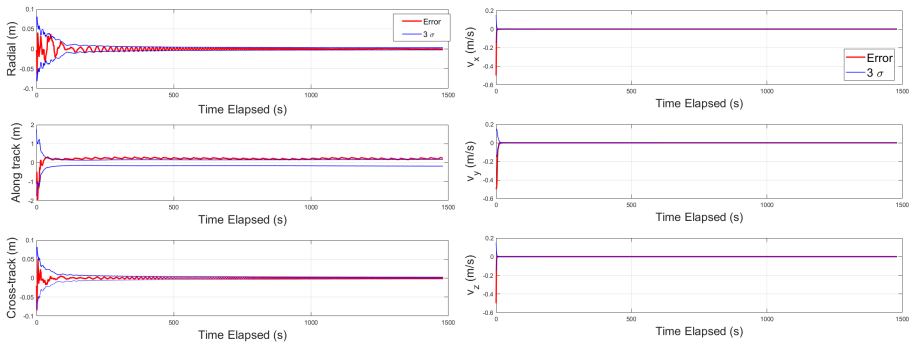
Table 9.25: Monte Carlo simulation results for V-bar hold scenario with loosely-coupled MEKF

Between the chosen pose estimation pipelines, the state estimation performance shows marginal difference considering the relative separation of 150 m. As such, the performance of HRNet-W32 and HRNet-small is comparable for the scenario represented by the image sequence. Note that PE-D performs well despite demonstrating poor performance and robustness during pose estimation evaluation in Section 9.3.2. This ascertains that the images emulating the V-bar trajectory are favourable images and do not necessarily provide sufficient fidelity to evaluate and compare pose estimation approaches or pipelines.

From the filter perspective, the loosely coupled filter using pose uncertainty derived by the MLPnP solver is effective at converging to high accuracy navigation solution, despite large deviation in the initial state. The loosely-coupled estimator introduced and analysed here provides a suitable alternative to the tightly-coupled alternative introduced in [57]. However, the loosely-coupled estimator is expected to be faster during execution as feature-wise representation in a tightly coupled estimator demands higher computation, especially of inverse matrix operations (Eq. 8.5). Further, it is not clear whether internal representation of perspective equations in the tightly-coupled filter provides better implict pose estimation than optimally design PnP solvers.

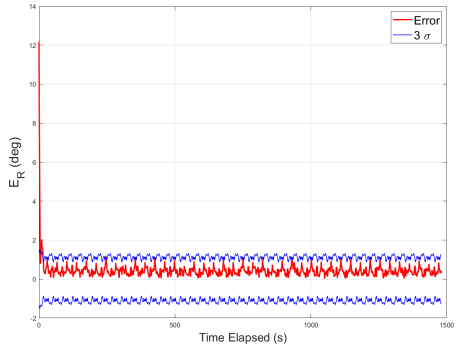
Finally, unlike typical loosely-coupled estimators that use a tuned and constant magnitude of the measurement covariance, the estimator introduced here uses contextual information from the CNN heatmaps to derive covariance in the loop. This makes state estimation inherently robust to poor detections during unfavourable optical and other operational conditions.

The Monte-Carlo simulation with random initial conditions on the given image sequence provides a limited insight on robustness, as the images are favourable for pose estimation. The simple nature of the scenario

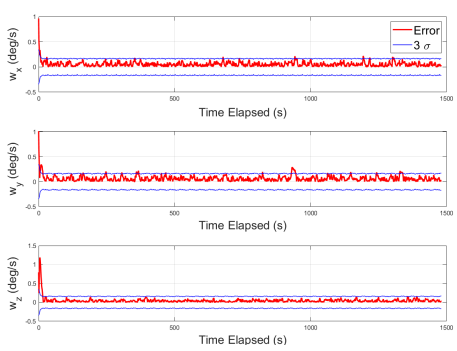


(a) Position

(b) Velocity

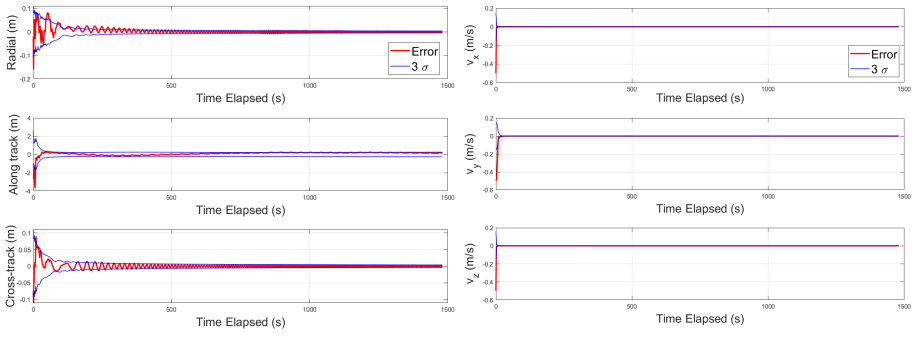


(c) Attitude



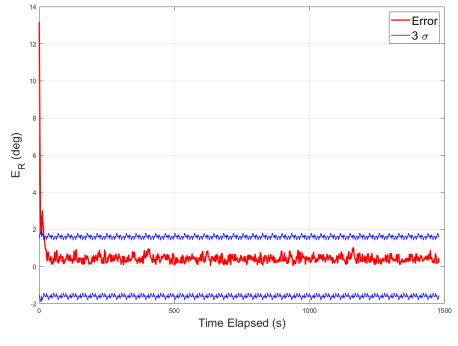
(d) Angular Velocity

Figure 9.12: Scenario 1: State estimation error for PE-C + MEKF

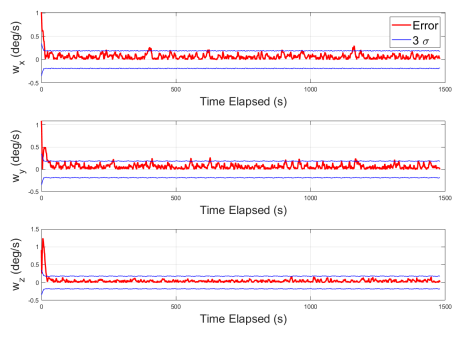


(a) Position

(b) Velocity

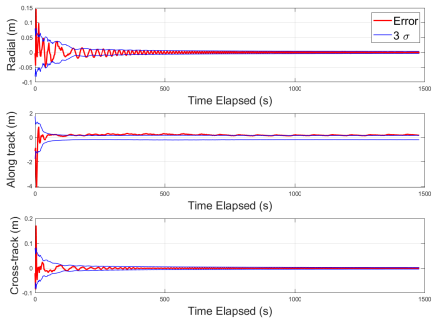


(c) Attitude

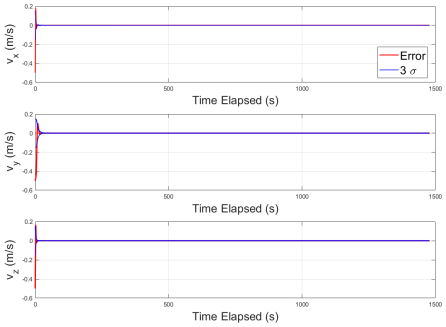


(d) Angular Velocity

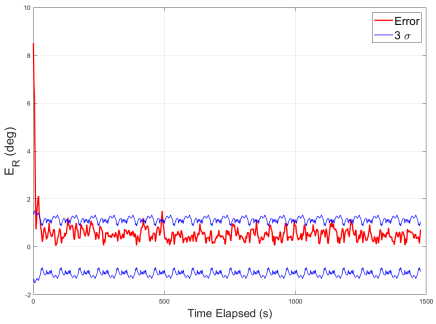
Figure 9.13: Scenario 1: State estimation error for PE-D + MEKF



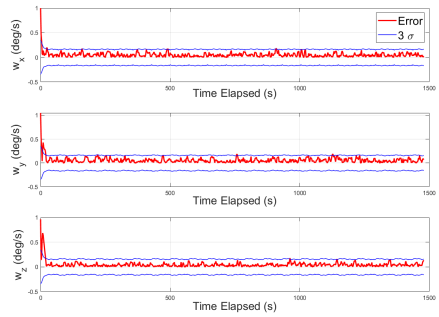
(a) Position



(b) Velocity

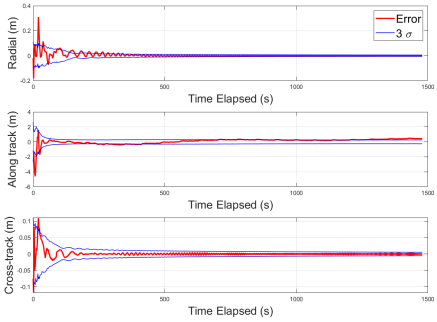


(c) Attitude

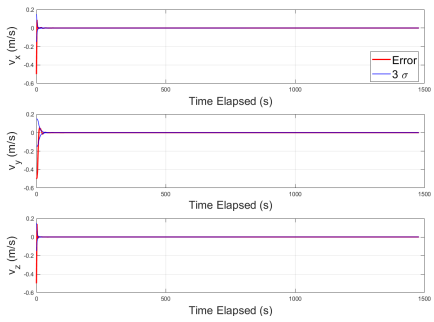


(d) Angular Velocity

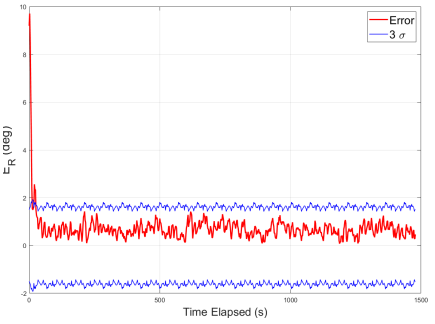
Figure 9.14: Scenario 2: State estimation error for PE-C + MEKF



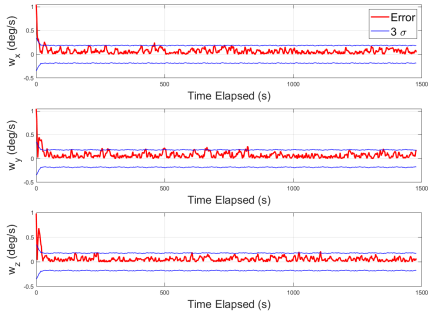
(a) Position



(b) Velocity



(c) Attitude



(d) Angular Velocity

Figure 9.15: Scenario 2: State estimation error for PE-D + MEKF

provides insufficient basis for conclusive comparison between design alternative. For the images in the used scenario, the heatmaps for all keypoints show near ideal Gaussian distribution indicating high accuracy of the prediction. This does fully demonstrate the utility of using contextual information from heatmaps, and the overall inclusion of uncertainty in the navigation loop. In order to provide definitive comparison between estimator design choices, a set of new trajectory scenarios must be generated that represent realistic conditions more accurately with image corruptions, challenging illumination conditions, continuous noise variation and other relevant artefacts. Since a conclusive investigation on this aspect is beyond the scope of this work, it must be conducted in the future. For this, a new set of images with desired fidelity that represent challenging dynamical scenarios, must be generated using the virtual rendering environment. Subsequently, an approach similar to the one adadpted for checking the robustness of the CNNs can be used to generate perturbations in the images that are designed specifically to stress-test the entire navigation loop for robustness. Subsequently, fidelity of simulation can be improved by including orbital perturbations in the trajectory sequence, representative on-board sensor models and uncertainty of servicer's absolute state can be taken into consideration.

9.5. Conclusions

In this chapter, the newly designed navigation framework has been evaluated through extensive experimentation. Incremental evaluation and validation of the navigation framework components and the underlying design choices provides key insights into the design, evaluation and testing of such learning-based frameworks. On a framework perspective, the selection configuration and implementation of the CNNs, pose solver and the state estimator is successfully realized. By testing the framework on two independent datasets of two spacecraft targets, the general applicability of the proposed framework is validated. The framework shows a desirable transfer to previously unseen real images and image artefacts, overcoming a major challenge in learning-based pose estimation and navigation. Further, the analysis is extended to include performance vs speed trade-off, which is crucial for embedded on-board application. Overall, the framework extends the current state-of-the-art approaches with superior accuracy and robustness in pose and state estimation.

From the perspective of using CNNs in the navigation loop, the experiments reveal that CNNs are able to overcome occlusion, illumination, background, texture and corruptions in the image to allow accurate pose estimation. However, the performance of the CNNs rely heavily on the training data, the training pipeline configuration, data augmentation, size and architecture. From accuracy point of view, larger and deeper network models perform better at the same task than a smaller and shallower models. More modern network architecture that use multi-scale information and higher internal resolution provide better accuracy and robustness for an optimal computation cost than more conventional architectures. Data augmentation is found to be a crucial link between accuracy and robustness, as shown repeatedly by the robustness assessment. Affine and pixel-level augmentations during training are crucial in improving robustness of the CNNs to previously unseen images. In particular, the pixel-level data augmentations with randomized brightness, contrast and saturation are critical to desirably transfer synthetic performance of the CNNs on to real images. Further, new alternatives are designed for the HRNet KD network, which are downscaled and optimized for inference speed with decreased FLOPs requirement.

With respect to the datasets, the experiments and analysis presented here provides a broader context than any existing work on similar navigation framework. The pose estimation pipeline is benchmarked on *SPEED* with individual validation of the object and KD network. The pose estimation pipeline is compared to the state-of-the-art results on the same dataset, demonstrating optimal speed vs accuracy performance over other similar pipeline. The robustness aspect is again emphasized in the discussion, as it is found that accuracy on synthetic image set does not reflect on robustness. The design decisions are reinforced at every step through robustness assessment using real images of the spacecraft acquired using a real camera in a representative environment. Further, the new HRNet-small network introduced in the network is found provide desirable accuracy and robustness, while having significantly lower model size and computation demand.

For Envisat use-case central to this work, the most important outcome from the experimentation is that the existing dataset being used to evaluate the use-case is not optimal for robustness. The *Envisat-1* images are found to be ideal and simple, allowing small and simple CNNs to achieve accuracies comparable to the larger and better networks. The experiments reveal that *Envisat-1* is unideal for training deep neural networks that must exhibit robustness to real images. The networks trained on the dataset overfit and perform poorly on images of the same dataset with perturbations. Consequently, the newly introduced *Envisat+IC*

is validated for fair evaluation of accuracy and robustness. The *Envisat+IC* dataset contains images with Earth background and real-life corruptions. Therefore, training on this dataset improves robustness to those artefacts expected in the real images. More importantly, to perform a fair robustness evaluation, the model trained on the dataset is tested on clean as well as highly corrupted images. The resulting model shows robustness to all the images, validating the ability of the new dataset to train the CNNs more effectively. In hindsight, the dataset is also the first ever spacecraft pose estimation dataset to take corruption robustness into account.

Following the CNN performance validation, the subsequent experiments are conducted to test the entire pose estimation pipeline, by integrating with the modified MLP*n*P solver. The novelty of the analysis lies in the using the MLP*n*P solver with the contextual information of the detections from the heatmaps to derive and preserve uncertainty in the loop. This is accomplished using posterior uncertainty estimation for the pose solution in MLP*n*P based on principles of maximum likelihood estimation. The results of the experiments demonstrate that the pose estimation pipeline is accurate. The performance of the two alternative pipelines compare well with the state-of-the-art results on **SPEED**. Especially, the new down-scaled network (HRNet-small) introduced in this work, provides an optimal speed vs accuracy performance to the pose estimation pipeline (PE-B) that is more suitable for on-board deployment in comparison with other pipelines. More importantly, the pose estimation pipelines in this framework show a desirable transfer of performance to real images. With fair evaluation on a standard dataset, the design choices made during the framework design have thus been validated with exhaustive analyses, evaluation and experimentation. For Envisat target use-case, the corrupted images in the *Envisat+IC* prove challenging for the HRNet-small network at large relative ranges, resulting in poor pose estimation performance. Although, the performance is expected to improve by fine-tuning the training process.

Finally, taking the entire navigation loop into consideration, the framework is evaluated for state estimation and tracking on a simulated V-bar trajectory scenario. The proposed loosely-coupled estimator is employed to track the state of the Envisat spacecraft in V-bar trajectory. The proposed estimator is desirable navigation performance in the first-order analysis. The robustness of the proposed estimator to large initial state perturbation is demonstrated with a Monte Carlo analysis. However, it is emphasized that the state estimation scenario, due to the inherent simplicity, only provides a basis for a preliminary investigation. Consequently, it does not allow effective assessment of the benefit of using uncertainty inclusion. The limitations of the scenario are outline and next steps for establishing more concrete navigation analysis are discussed.

IV

Closure

10

Conclusions and Recommendations

10.1. Conclusions

In this work, a novel monocular vision-based navigation framework has been proposed and investigated for proximity operations around an uncooperative spacecraft. At a higher level, this thesis presents the design and analysis of a navigation loop that reduces the sensor hardware complexity in the navigation system by utilizing CNNs with monocular images. The resulting framework takes a step towards enabling real on-orbit operations by addressing critical challenges such as robustness in CNNs, uncertainty preservation and size-accuracy trade-off. The outcomes of this thesis extend the current analyses and investigations in the research body for the new paradigm of learning-based navigation methods for uncooperative targets. The work on framework design provides a salient point of reference for future works by exploring and presenting the principles across multi-disciplinary domains of machine learning, computer vision, projective geometry and estimation theory relevant for understanding, comparing and developing better learning-based navigation systems. Further, this work also extends software resources for data-intensive deep learning aspects related to development, implementation, visualization and evaluation of a CNN-based navigation loop. The resources, tools and processes related to the Envisat target use-case will benefit the research on this topic being undertaken jointly at TU Delft SpE, ESA-ESTEC and Airbus. The more generic software resources are made publicly available [1] and are expected to benefit the future students and the wider community.

The analyses pertaining to the specific aspects of the navigation framework are traced back to the research questions and the final conclusions are classified into CNN-based pose estimation pipeline (**RQ-1**), integration of the navigation loop (**RQ-2**), syntheticity-reality gap (**RQ-3**) and datasets (**RQ-4**) for deep learning.

10.1.1. CNN-based pose estimation pipeline

The pose estimation pipeline is built according to a state-of-the-art architecture that utilizes an object detection network, a keypoint detection network and a PnP solver. The object detection network is based on the SSD architecture and the keypoint detection network is based on the HRNet architecture. The selection of these state-of-the-art network architectures is motivated with thorough assessment of the alternative network architectures. Subsequently, a detailed breakdown of network structure, configuration and implementation is presented. Identifying the computation demand of accurate keypoint detection using standard HRNet-W32 network, two down-scaled networks (HRNet-small and HRNet-smaller) are built upon the HRNet core architecture. The networks are trained on synthetic images of the PRISMA Tango spacecraft in *SPEED* and images of the Envisat spacecraft. The network size, data augmentation, input size and pre-training are selected as design variables to assess the variation of network performance during experimentation. The trained models for object detection and keypoint detection are evaluated for performance and robustness, and further integrated with the MLPnP solver. The challenge of quantifying measurement uncertainty is tackled by using heatmaps from the keypoint detection network to quantify the keypoint location covariance in the loop. This is utilized in the MLE framework of the MLPnP solver to provide an accurate pose solution and the associated pose uncertainty.

[1] https://github.com/kuldeepbrd1/Barad2020MSc under open-source MIT License

CNNs employed for object detection and keypoint detection demonstrate the ability to provide sufficient accuracy on synthetic images used for model validation. In general larger network size, that allow approximating deeper hierarchical mapping of the output, perform more accurately than smaller images. The margin of difference between the sizes increases as complexity of the artefacts in the image increases. On simpler images of Envisat from *Envisat-1* dataset, the choice network size is not very consequential, due to the overly simplistic scenario modeled across the images with little to no variation in image artefacts. On the other hand, the effect of network size on detection performance becomes clear on the more challenging *SPEED* images that contain blur, noise and earth in the background, along with challenging illumination conditions.

In terms of the pose solver, a modification to the MLPnP algorithm is proposed in this work to improve the accuracy of pose estimation at larger separation ranges. The reasoning behind the proposed modification is detailed and modified algorithm is benchmarked for higher separation ranges. The modified algorithm successfully overcomes the limitations of the original algorithm with superior accuracy. The suitability of the modified MLPnP is further reinforced by a Monte Carlo analysis specific Envisat target case in comparison with CEPPnP, the only other state-of-the-art covariant solver. The results show that the modified MLPnP provides superior translation accuracy and comparable rotation accuracy. However, the superiority of MLPnP is highlighted with increasing detection noise.

Finally, the integrated pose estimation pipeline is benchmarked on *SPEED* synthetic validation set. The results from the candidate pipelines finalized for evaluation provide accuracy on par with the the state-of-the-art results. Using the larger HRNet-W32 network for keypoint detection, the pipeline provides accuracy comparable to the the state-of-the-art while being significantly more optimized in terms of model size and computation requirement. Using the newly introduced HRNet-small network, the speed-accuracy trade-off is further fine-tuned while decreasing the model size and computation requirements by a factor of 5 and 3 respectively. The pipeline is more suitable for on-board deployment due to it's resource demand and the accuracy is considerably higher than a comparable state-of-the-art pipeline.

10.1.2. Syntheticity-Reality Gap

The proposed framework is designed for robustness in the pose estimation pipeline that allows to bridge the gap between synthetic images and the real images. The robustness is added implicitly by the optimal choice of CNNs and the architecture, while robustness is also improved explicitly using techniques like data augmentations. In terms of implicit robustness, utilizing object detection before keypoint detection provides robustness to scale and background texture. Further, use of multi-scale networks in the pipeline provides robustness to image resolution and the appropriate training configuration for the network improves robustness to overfitting. Of special interest is the explicit means of improving robustness for a given architecture. This is done by augmenting the data during training. For this, the role of affine and pixel-level data augmentations is investigated. In principle, the robustness assessment must be carried out on real images of the target after training on synthetic images. However, due to possible unavailability of space-borne images of the target, lab-generated images of the target in a realistic settings must be used. If the lab-generated images are not available, the preliminary robustness assessment can also be carried out using synthetic images with pixel-level variations and corruptions, not encountered in the training set.

The primary analysis on the robustness to real images is carried out using lab-generated real images of the Tango spacecraft in *SPEED*. The experiments on robustness provide conclusive evidence that data augmentations during training are crucial to allow generalization of detection. The networks show robustness when evaluated on previously unseen images of the spacecraft acquired with a real camera. Further, the SLAB/ESA score for real images shows marginal loss with respect to the synthetic images, demonstrating effective transfer to reality. The experiments reveal that tin order to perform well on real images, pixel-level data augmentation are critical. The robustness assessment analyses of the pose estimation pipeline provide preliminary validation of the underlying design approach adapted in the proposed framework. It is however emphasized that detailed investigation with more real images is necessary to provide conclusive inference on this aspect, due to limited availability of real images.

10.1.3. Datasets

The inadequacy of current Envisat datasets is demonstrated through extensive experimentation and the need for newer datasets is motivated. Two factors of improvements were considered: Earth background augmentation and the common image corruptions. A new dataset (*Envisat+IC*) is created that augments Envisat

images with the aforementioned artefacts. The Earth background is augmented using space-borne images of the Earth's full disk from the Himawari-8 satellite and the image corruptions include various models of blur, noise and other expected effects. To enable fair assessment of robustness, special validation sets are generated that allow analysis on generalization and robustness in learning, using clean images, corrupted images and highly corrupted images.

The resulting dataset is the first spacecraft pose estimation dataset to take image corruptions into account. Consequently, the network models trained on the new dataset present the first models that are robust to common corruptions in the image. The efficacy of training and validation on the new dataset is demonstrated through multi-faceted evaluations. The comparison between models trained on the existing dataset and the those trained on the new dataset shows that new dataset drastically improves robustness of the CNNs. Further, augmentations in the new dataset increase the complexity of the images and challenge the smaller CNNs that usually perform well on the existing datasets.

10.1.4. Integration of the Navigation Loop

In the designed framework a loosely coupled architecture is utilized, which takes psuedomeasurements of the pose as input to estimate the spacecraft state for the target. To quantify measurement uncertainty, the heatmap output from the keypoint detection model is utilized. With Gaussian approximation of the heatmap activation around the true keypoint, a score based weighting scheme is applied to derive covariance. The problem of uncertainty preservation in a loosely-coupled estimator is tackled by using the MLPnP solver. The MLPnP solver utilizes keypoint location uncertainty to estimate pose as well as the uncertainty associated with the estimate. This allows for an uncertainty aware loosely-coupled architecture that is introduced in this work. Consequently, the pose estimation pipeline with the trained CNN models and the MLPnP solver is integrated with MEKF, wherein measurement covariance is obtained from the MLPnP output.

The integrated navigation loop is evaluated on a simplified scenario for the Envisat target in a perturbation-free V-bar trajectory at 150 m along the boresight of the servicer spacecraft. The integrated navigation loop is tested with highly perturbed initial conditions and demonstrates a desirable convergence profile. In addition, the robustness of the integrated navigation loop to initial conditions is verified with a Monte Carlo analysis with random sampling of initial conditions. The integrated loop demonstrates high accuracy of steady state estimate across randomly sampled initial conditions. In general, the estimation error in the relative position is less than 0.3 m and that in the relative attitude is less than 1 deg along the Euler axis, at a separation of 150 m.

While the accuracy of the navigation solution is found desirable, it is noted that the simplicity of the scenario provides limited scope for valid comparison between alternative estimator designs. Due to the simplicity of the scenario and favourability of the images, the conclusive analysis justifying the use of covariance representation in the loop, as opposed to tuned and constant measurement covariance, is not realized.

10.2. Recommendations for Future Work

The work presents a limited analysis for enabling learning-based methods for vision-based navigation in proximity operations. Using the framework designed here as a baseline, the future works can further investigate several related aspects, as noted below.

CNN-based Pose Estimation pipeline

The CNNs in the pose estimation pipeline is developed and implemented in a discontinuous manner i.e. the CNNs within the pose estimation pipeline interface only through data files. Besides, the OD and KD networks are implemented on two different machine learning platforms, preventing a consistent implementation. In the future works, the development of software interfaces and incremental integration of pose estimation pipeline can be carried out to understand the way these novel CNN-based pipelines can be deployed on an embedded computer.

On the network level, the selected KD network based on HRNet architecture presents significantly higher computation demand. There is scope for improving the HRNet efficiency by replacing the convolution operations in the network with depth-wise separable convolution, which is expected to lower memory and computation demand significantly.

The pose estimation pipeline in the designed framework requires the target body's 3D model. For the viability and scalability of large-scale ADR efforts, the methods need to apply to multiple targets. To this extent, future works can explore ways in which this framework can be re-configured on-board to support multiple target captures. Alternatively, the same can be enabled by exploring newer neural network architectures that do not rely on model-specific features, while being accurate and robust.

Syntheticity-Reality Gap

The gap between syntheticity and reality is the most consequential aspect, that the future works must focus on. This work presents several key aspects that form a baseline for bridging the gap between synthetic images and real images. The *Envisat+IC* dataset proposed here could be tested with lab generated real images. Based on extensive image acquisition campaigns, the *Envisat+IC* dataset could be improved to exhibit more representative corruptions and the corresponding magnitudes, as observed with the real camera. Further, novel datasets can be generated that fuse synthetic and lab generated images, to improve performance on real images. The analyses on this front with the CNNs will improve the explainability of the CNN models.

While lab tests and using real images can provide qualitative inferences, the study of syntheticity and reality gap must also be assessed on a lower level, to ensure the inferences are valid. For this, future works could undertake image-level analyses in assessing which artefacts in the images are relevant, how to quantify the aspects that generate the gap, and how to employ techniques at the intersection of domain transfer [130] and meta-learning (learning to learn) [168, 169] in CNNs [170], to increase the robustness of CNN models to the relevant unseen images. The analyses on this front will improve the quantifiability of the gap.

The aforementioned steps to improve quantifiability and the explainability will provide the necessary elements to then tackle the challenges of validation and verification of the CNNs and the pose estimation pipeline. This remains a major challenge for neural networks, which unlike conventional approaches persist less deterministic behavior.

Datasets

This work presents a preliminary analysis of deep learning datasets for spacecraft pose estimation, by taking image corruptions into account. Future works can explore datasets with texture randomization and adversarial images that are used to make CNNs more robust.

More importantly, there is a need for image sequences that can simulate a continuous trajectory to enable a rigorous evaluation of the full navigation loop. Currently, only the *Envisat-1* dataset avails an image set simulating a relatively simple perturbation free trajectory. Future datasets must enable image sequences that include perturbations, image corruptions, temporal variation of illumination among others.

Integration of Navigation Loop

The preliminary analysis of a loosely-coupled state estimator to a CNN-based pose estimation pipeline was presented. However, it is highlighted that currently available image sequences representing trajectories do

not provide an adequate basis for comparison between various design alternatives. Given the generation of images with newer trajectories, a thorough analysis could be carried out in the future works to assess the importance of using covariance in the loop, by comparing the navigation loops that do not use heatmap-derived covariance. Consequently, more representative comparisons can be made between tightly and loosely-coupled estimators, using image sequences with perturbed trajectories and worst-case approach scenarios.

Bibliography

[1] D. C. Woffinden and D. K. Geller, "Navigating the road to autonomous orbital rendezvous," *Journal of Spacecraft and Rockets*, vol. 44, no. 4, pp. 898–909, 2007.

[2] M. Wilde, J. T. Harder, and E. Stoll, ""on-orbit servicing and active debris removal: Enabling a paradigm shift in spaceflight"," *Frontiers in Robotics and AI*, vol. 6, p. 136, 2019.

[3] ESA Space Debris Office, "ESA's Annual Space Environment Report," Tech. Rep. GEN-DB-LOG-00271-OPS-SD, European Space Operations Centre, July 2019.

[4] H. Klinkrad, "The Space Debris Environment and its Evolution," in *The 6th IAASS Conference, Montreal, Canada*, pp. 21–23, 2013.

[5] G. Hausmann, M. Wieser, R. Haarmann, A. Brito, J.-C. Meyer, S. Jäkel, *et al.*, "e.Deorbit mission: OHB debris removal concepts," in *Proceeding of the 13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2015), Noordwijk*, 2015.

[6] A. Tatsch, N. Fitz-Coy, and S. Gladun, "On-orbit servicing: A brief survey," in *Proceedings of the 2006 Performance Metrics for Intelligent Systems Workshop*, pp. 276–281, 2006.

[7] A. Long, M. Richards, and D. E. Hastings, "On-orbit servicing: a new value proposition for satellite design and operation," *Journal of Spacecraft and Rockets*, vol. 44, no. 4, pp. 964–976, 2007.

[8] S. Sharma *et al.*, "Comparative assessment of techniques for initial pose estimation using monocular vision," *Acta Astronautica*, vol. 123, pp. 435–445, 2016.

[9] S. Sharma, C. Beierle, and S. D'Amico, "Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks," in *2018 IEEE Aerospace Conference*, pp. 1–12, IEEE, 2018.

[10] V. Capuano, S. R. Alimo, A. Q. Ho, and S.-J. Chung, "Robust features extraction for on-board monocular-based spacecraft pose acquisition," in *AIAA Scitech 2019 Forum*, p. 2005, 2019.

[11] J. Ventura, *Autonomous Proximity Operations for Noncooperative Space Targets*. PhD thesis, Technische Universit "a t M ü nchen, 2016.

[12] L. P. Cassinis, R. Fonod, and E. Gill, "Review of the robustness and applicability of monocular pose estimation systems for relative navigation with an uncooperative spacecraft," *Progress in Aerospace Sciences*, 2019.

[13] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[14] S. Sharma, J. Ventura, and S. D'Amico, "Robust model-based monocular pose initialization for non-cooperative spacecraft rendezvous," *Journal of Spacecraft and Rockets*, vol. 55, no. 6, pp. 1414–1429, 2018.

[15] L. Schlenker, M. Moretto, D. Gaylor, and R. Linares, "Simultaneous Localization and Mapping for Satellite Rendezvous and Proximity Operations Using Random Finite Sets," 2019.

[16] S. Sharma, *Pose Estimation of Uncooperative Spacecraft using Monocular Vision and Deep Learning*. PhD thesis, Stanford University, 2019.

[17] S. Sharma and S. D'Amico, "Neural Network-Based Pose Estimation for Noncooperative Spacecraft Rendezvous," *IEEE Transactions on Aerospace and Electronic Systems*, 2020.

[18] J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. Depristo, J. Dillon, and B. Lakshminarayanan, "Likelihood ratios for out-of-distribution detection," in *Advances in Neural Information Processing Systems*, pp. 14707–14718, 2019.

[19] G. Lunney, "Summary of Gemini rendezvous experience," in *Simulation and Support Conference*, p. 272, 1967.

[20] B. Kennedy, J. E. Riedel, S. Bhaskaran, S. Desai, D. Han, T. McElrath, G. Null, M. Ryne, S. Synnott, M. Wang, *et al.*, "Deep Space 1 Navigation: Primary Mission," *Deep Space Communications and Navigation Systems*, 2004.

[21] J. M. Leonard, P. G. Antreasian, C. D. Jackman, B. Page, D. R. Wibben, and M. C. Moreau, "Orbit Determination Strategy and Simulation Performance for OSIRIS-REx Proximity Operations," 2017.

[22] W. D. Compton and C. D. Benson, "Living and working in space. A history of Skylab," 1983.

[23] D. Zimpfer, P. Kachmar, and S. Tuohy, "Autonomous rendezvous, capture and in-space assembly: past, present and future," in *1st Space exploration conference: continuing the voyage of discovery*, p. 2523, 2005.

[24] C. J. Dennehy and J. R. Carpenter, "A summary of the rendezvous, proximity operations, docking, and undocking (RPODU) lessons learned from the defense advanced research project agency (DARPA) orbital express (OE) demonstration system mission," 2011.

[25] R. T. Howard and T. C. Bryan, "DART AVGS flight results," in *Sensors and Systems for Space Applications*, vol. 6555, p. 65550L, International Society for Optics and Photonics, 2007.

[26] J. L. Forshaw, G. S. Aglietti, N. Navarathinam, H. Kadhem, T. Salmon, A. Pisseloup, E. Joffre, T. Chabot, I. Retat, R. Axthelm, *et al.*, "RemoveDEBRIS: An in-orbit active debris removal demonstration mission," *Acta Astronautica*, vol. 127, pp. 448–463, 2016.

[27] B. B. Reed, R. C. Smith, B. J. Naasz, J. F. Pellegrino, and C. E. Bacon, "The restore-L servicing mission," in *AIAA SPACE 2016*, p. 5478, 2016.

[28] S. Brook and B. David, "DARPA Phoenix Payload Orbital Delivery (POD) System: FedEx to GEO," *San Diego: AIAA*, 2013.

[29] C. Blackerby, A. Okamoto, K. Fujimoto, N. Okada, J. L. Forshaw, and J. Auburn, "ELSA-d: An In-Orbit End-of-Life Demonstration Mission," in *Proceedings of the 69th International Astronautical Congress. Bremen, Germany: International Astronautical Federation*, 2018.

[30] J. Telaar, I. Ahrns, S. Estable, W. Rackl, M. De Stefano, R. Lampariello, N. Santos, P. Serra, M. Canetri, F. Ankersen, *et al.*, "GNC architecture for the e. Deorbit mission," in *7th European Conference for Aeronautics and Space Sciences (EUCASS)*, 2017.

[31] G. Gaias, S. D'Amico, and J.-S. Ardaens, "Angles-only navigation to a noncooperative satellite using relative orbital elements," *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 2, pp. 439–451, 2014.

[32] S. D'Amico, J.-S. Ardaens, G. Gaias, H. Benninghoff, B. Schlepp, and J. Jørgensen, "Noncooperative rendezvous using angles-only optical navigation: system design and flight results," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 6, pp. 1576–1595, 2013.

[33] I. Mitchell, "Draper labroatory overview of rendez-vous and capture operations [R/OL]," 2011.

[34] T. M. Mitchell, "Does machine learning really work?," *AI Magazine*, vol. 18, p. 11, Sep. 1997.

[35] S. D'Amico, M. Benn, and J. L. Jørgensen, "Pose estimation of an uncooperative spacecraft from actual space imagery," in *5th International Conference on Spacecraft Formation Flying Missions and Technologies*, 2013.

[36] T. Huang, "Computer vision: Evolution and promise," 1996.

[37] S. Gold, C.-P. Lu, A. Rangarajan, S. Pappu, and E. Mjolsness, "New algorithms for 2D and 3D point matching: Pose estimation and correspondence," in *Advances in neural information processing systems*, pp. 957–964, 1995.

[38] R. Opromolla, G. Fasano, G. Rufino, and M. Grassi, "A review of cooperative and uncooperative spacecraft pose determination techniques for close-proximity operations," *Progress in Aerospace Sciences*, vol. 93, pp. 53–72, 2017.

[39] N. N. Dawoud, B. B. Samir, and J. Janier, "Fast template matching method based optimized sum of absolute difference algorithm for face localization," *International Journal of Computer Applications*, vol. 18, no. 8, pp. 0975–8887, 2011.

[40] K. Briechle and U. D. Hanebeck, "Template matching using fast normalized cross correlation," in *Optical Pattern Recognition XII*, vol. 4387, pp. 95–102, International Society for Optics and Photonics, 2001.

[41] C. Reinbacher, M. Ruther, and H. Bischof, "Pose estimation of known objects by efficient silhouette matching," in *2010 20th International Conference on Pattern Recognition*, pp. 1080–1083, IEEE, 2010.

[42] R. Opromolla, G. Fasano, G. Rufino, and M. Grassi, "A model-based 3D template matching technique for pose acquisition of an uncooperative space object," *Sensors*, vol. 15, no. 3, pp. 6360–6382, 2015.

[43] L.-P. Morency and R. Gupta, "Robust real-time egomotion from stereo images," in *Proceedings 2003 International Conference on Image Processing (Cat. No. 03CH37429)*, vol. 2, pp. II–719, IEEE, 2003.

[44] D. Sinclair, A. Blake, and D. Murray, "Robust estimation of egomotion from normal flow," *International Journal of Computer Vision*, vol. 13, no. 1, pp. 57–69, 1994.

[45] S. Sharma and S. D'Amico, "Pose Estimation for Non-Cooperative Rendezvous Using Neural Networks," *2019 AAS/AIAA Astrodynamics Specialist Conference (arXiv preprint arXiv:1906.09868)*, 2019.

[46] T. H. Park, S. Sharma, and S. D'Amico, "Towards Robust Learning-Based Pose Estimation of Noncooperative Spacecraft," *arXiv preprint arXiv:1909.00392*, 2019.

[47] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o (n) solution to the pnp problem," *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.

[48] D. F. Dementhon and L. S. Davis, "Model-based object pose in 25 lines of code," *International journal of computer vision*, vol. 15, no. 1-2, pp. 123–141, 1995.

[49] D. Oberkampf, D. F. DeMenthon, and L. S. Davis, "Iterative pose estimation using coplanar feature points," *Computer Vision and Image Understanding*, vol. 63, no. 3, pp. 495–511, 1996.

[50] C.-P. Lu, G. D. Hager, and E. Mjolsness, "Fast and globally convergent pose estimation from video images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 6, pp. 610–622, 2000.

[51] S. Li, C. Xu, and M. Xie, "A robust O (n) solution to the perspective-n-point problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 7, pp. 1444–1450, 2012.

[52] J. A. Hesch and S. I. Roumeliotis, "A Direct Least-Squares (DLS) method for PnP," in *2011 International Conference on Computer Vision*, pp. 383–390, IEEE, 2011.

[53] G. Schweighofer and A. Pinz, "Globally Optimal O (n) Solution to the PnP Problem for General Camera Models," in *BMVC*, pp. 1–10, 2008.

[54] Y. Zheng, Y. Kuang, S. Sugimoto, K. Astrom, and M. Okutomi, "Revisiting the PnP problem: A fast, general and optimal solution," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2344–2351, 2013.

[55] Y. Zheng, S. Sugimoto, and M. Okutomi, "Aspnp: An accurate and scalable solution to the perspective-n-point problem," *IEICE TRANSACTIONS on Information and Systems*, vol. 96, no. 7, pp. 1525–1535, 2013.

[56] L. Ferraz Colomina, X. Binefa, and F. Moreno-Noguer, "Leveraging feature uncertainty in the PnP problem," in *Proceedings of the BMVC 2014 British Machine Vision Conference*, pp. 1–13, 2014.

[57] L. Pasqualetto Cassinis, R. Fonod, E. Gill, I. Ahrns, and J. Gil Fernandez, "Cnn-based pose estimation system for close-proximity operations around uncooperative spacecraft," in *AIAA Scitech 2020 Forum*, p. 1457, 2020.

[58] S. Urban, J. Leitloff, and S. Hinz, "MLPnP-a real-time maximum likelihood solution to the perspective-n-point Problem," *arXiv preprint arXiv:1607.08112*, 2016.

[59] W. Förstner, "Minimal representations for uncertainty and estimation in projective spaces," in *Asian Conference on Computer Vision*, pp. 619–632, Springer, 2010.

[60] S. Sharma and S. D'Amico, "Reduced-dynamics pose estimation for non-cooperative spacecraft rendezvous using monocular vision," in *38th AAS Guidance and Control Conference, Breckenridge, Colorado*, 2017.

[61] A. Harvard, V. Capuano, E. Y. Shao, and S.-J. Chung, "Pose estimation of uncooperative spacecraft from monocular images using neural network based keypoints," 2020.

[62] L. Pasqualetto Cassinis, R. Fonod, E. Gill, I. Ahrns, and J. Fernandez, "Comparative Assessment of Image Processing Algorithms for the Pose Estimation of Uncooperative Spacecraft," in *Proceedings of the International Workshop on Satellite Constellations and Formation Flying*, 08 2019.

[63] B. E. Tweddle and A. Saenz-Otero, "Relative computer vision-based navigation for small inspection spacecraft," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 5, pp. 969–978, 2015.

[64] E. Lefferts, F. L. Markley, and M. D. Shuster, "Kalman filtering for spacecraft attitude estimation," *Journal of Guidance, Control, and Dynamics*, vol. 5, no. 5, pp. 417–429, 1982.

[65] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[66] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[67] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[68] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[69] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[70] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014.

[71] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2015.

[72] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," 2014.

[73] P. F. Proença and Y. Gao, "Deep Learning for Spacecraft Pose Estimation from Photorealistic Rendering," *arXiv preprint arXiv:1907.04298*, 2019.

[74] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, "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, 2015.

[75] S. Mahendran, H. Ali, and R. Vidal, "3D pose regression using convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2174–2182, 2017.

[76] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," *CoRR*, vol. abs/1711.00199, 2017.

[77] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, pp. 2938–2946, 2015.

[78] B. Chen, J. Cao, A. Parra, and T.-J. Chin, "Satellite Pose Estimation with Deep Landmark Regression and Nonlinear Pose Refinement," in *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

[79] M. Kisantal, S. Sharma, T. h. Park, D. Izzo, M. Martens, and S. D'Amico, "Satellite Pose Estimation Challenge: Dataset, Competition Design and Results," *arXiv preprint arXiv:1911.02050*, 2019.

[80] T. Sattler, Q. Zhou, M. Pollefeys, and L. Leal-Taixe, "Understanding the Limitations of CNN-based Absolute Camera Pose Regression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3302–3312, 2019.

[81] S. Jin, X. Ma, Z. Han, Y. Wu, W. Yang, W. Liu, C. Qian, and W. Ouyang, "Towards multi-person pose tracking: Bottom-up and top-down methods," in *ICCV PoseTrack Workshop*, vol. 2, p. 7, 2017.

[82] S. Sharma, T. H. Park, and S. D'Amico, "Spacecraft Pose Estimation Dataset (SPEED)." https://purl-stanford-edu.tudelft.idm.oclc.org/dz692fn7184.

[83] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, *et al.*, "Deep high-resolution representation learning for visual recognition," *arXiv preprint arXiv:1908.07919*, 2019.

[84] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[85] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[86] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[87] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Object detection with keypoint triplets," *arXiv preprint arXiv:1904.08189*, vol. 1, no. 2, p. 4, 2019.

[88] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 9627–9636, 2019.

[89] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv preprint arXiv:2004.10934*, 2020.

[90] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7310–7311, 2017.

[91] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[92] C.-Y. Wang, H.-Y. Mark Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "CSPNet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 390–391, 2020.

[93] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[94] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[95] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[96] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768, 2018.

[97] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

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

[99] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[100] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in neural information processing systems*, pp. 379–387, 2016.

[101] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[102] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "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, 2014.

[103] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[104] C. Liu and W. Hu, "Relative pose estimation for cylinder-shaped spacecraft using single image," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, no. 4, pp. 3036–3056, 2014.

[105] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[106] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

[107] Y. Xiong, H. Liu, S. Gupta, B. Akin, G. Bender, P.-J. Kindermans, M. Tan, V. Singh, and B. Chen, "MobileDets: Searching for Object Detection Architectures for Mobile Accelerators," *arXiv preprint arXiv:2004.14525*, 2020.

[108] H. Fan, S. Liu, M. Ferianc, H.-C. Ng, Z. Que, S. Liu, X. Niu, and W. Luk, "A real-time object detection accelerator with compressed SSDLite on FPGA," in *2018 International Conference on Field-Programmable Technology (FPT)*, pp. 14–21, IEEE, 2018.

[109] M. S. Louis, Z. Azad, L. Delshadtehrani, S. Gupta, P. Warden, V. J. Reddi, and A. Joshi, "Towards deep learning using tensorFlow lite on RISC-V," in *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2019.

[110] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[111] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[112] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," *ArXiv*, vol. abs/1706.02677, 2017.

[113] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random Erasing Data Augmentation.," in *AAAI*, pp. 13001–13008, 2020.

[114] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. Software available from tensorflow.org.

[115] T. Golda, T. Kalb, A. Schumann, and J. Beyerer, "Human pose estimation for real-world crowded scenarios," in *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–8, IEEE, 2019.

[116] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Advances in neural information processing systems*, pp. 1799–1807, 2014.

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

[118] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *European conference on computer vision*, pp. 483–499, Springer, 2016.

[119] Y. Chen, Z. Wang, Y. Peng, Z. Zhang, G. Yu, and J. Sun, "Cascaded pyramid network for multi-person pose estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7103–7112, 2018.

[120] W. Yang, S. Li, W. Ouyang, H. Li, and X. Wang, "Learning feature pyramids for human pose estimation," in *proceedings of the IEEE international conference on computer vision*, pp. 1281–1290, 2017.

[121] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 466–481, 2018.

[122] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep high-resolution representation learning for human pose estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5693–5703, 2019.

[123] F. Zhang, X. Zhu, and M. Ye, "Fast human pose estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3517–3526, 2019.

[124] W. Li, Z. Wang, B. Yin, Q. Peng, Y. Du, T. Xiao, G. Yu, H. Lu, Y. Wei, and J. Sun, "Rethinking on multi-stage networks for human pose estimation," *arXiv preprint arXiv:1901.00148*, 2019.

[125] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[126] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, *et al.*, "Deep high-resolution representation learning for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[127] F. Zhang, X. Zhu, H. Dai, M. Ye, and C. Zhu, "Distribution-aware coordinate representation for human pose estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7093–7102, 2020.

[128] K. Sun, Y. Zhao, B. Jiang, T. Cheng, B. Xiao, D. Liu, Y. Mu, X. Wang, W. Liu, and J. Wang, "High-Resolution Representations for Labeling Pixels and Regions," *arXiv preprint arXiv:1904.04514*, 2019.

[129] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*, 2015.

[130] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

[131] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "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, 2014.

[132] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[133] E. Gill, S. D'Amico, and O. Montenbruck, "Autonomous formation flying for the PRISMA mission," *Journal of Spacecraft and Rockets*, vol. 44, no. 3, pp. 671–681, 2007.

[134] C. Beierle and S. D'Amico, "Variable-Magnification Optical Stimulator for Training and Validation of Spaceborne Vision-Based Navigation," *Journal of Spacecraft and Rockets*, pp. 1–13, 2019.

[135] K. Bessho, K. Date, M. Hayashi, A. Ikeda, T. Imai, H. Inoue, Y. Kumagai, T. Miyakawa, H. Murata, T. Ohno, *et al.*, "An introduction to Himawari-8/9—Japan's new-generation geostationary meteorological satellites," *Journal of the Meteorological Society of Japan. Ser. II*, vol. 94, no. 2, pp. 151–183, 2016.

[136] I. Guyon, "A scaling law for the validation-set training-set size ratio," *AT&T Bell Laboratories*, vol. 1, no. 11, 1997.

[137] M. Grant and S. Boyd, "CVX: Matlab Software for Disciplined Convex Programming, version 2.1." http://cvxr.com/cvx, Mar. 2014.

[138] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control* (V. Blondel, S. Boyd, and H. Kimura, eds.), Lecture Notes in Control and Information Sciences, pp. 95–110, Springer-Verlag Limited, 2008. http://stanford.edu/~boyd/graph_dcp.html.

[139] P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez, "Unrealrox: an extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation," *arXiv preprint arXiv:1810.06936*, 2018.

[140] R. Brochard, J. Lebreton, C. Robin, K. Kanani, G. Jonniaux, A. Masson, N. Despré, and A. Berjaoui, "Scientific image rendering for space scenes with the SurRender software," *arXiv preprint arXiv:1810.01423*, 2018.

[141] S. Parkes, I. Martin, M. Dunstan, and D. Matthews, "Planet surface simulation with pangu," in *Space OPS 2004 Conference*, p. 389, 2004.

[142] O. Dubois-Matra, S. Parkes, and M. Dunstam, "Testing and Validation of Planetary Vision-based navigation systems with PANGU," in *21st International Symposium on Space Flight Dynamics (ISSFD)*, 2009.

[143] N. Carlini and D. A. Wagner, "Towards Evaluating the Robustness of Neural Networks," *CoRR*, vol. abs/1608.04644, 2016.

[144] M. M. Cisse, Y. Adi, N. Neverova, and J. Keshet, "Houdini: Fooling deep structured visual and speech recognition models with adversarial examples," in *Advances in neural information processing systems*, pp. 6977–6987, 2017.

[145] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Towards Proving the Adversarial Robustness of Deep Neural Networks," *Electronic Proceedings in Theoretical Computer Science*, vol. 257, p. 19–26, Sep 2017.

[146] D. Hendrycks and K. Gimpel, "Early methods for detecting adversarial images," *arXiv preprint arXiv:1608.00530*, 2016.

[147] A. S. Ross and F. Doshi-Velez, "Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing their Input Gradients," 2017.

[148] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[149] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness," *arXiv preprint arXiv:1811.12231*, 2018.

[150] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1501–1510, 2017.

[151] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

[152] I. Vasiljevic, A. Chakrabarti, and G. Shakhnarovich, "Examining the impact of blur on recognition by convolutional networks," *arXiv preprint arXiv:1611.05760*, 2016.

[153] H. Hosseini, B. Xiao, M. Jaiswal, and R. Poovendran, "On the limitation of convolutional neural networks in recognizing negative images," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 352–358, IEEE, 2017.

[154] R. Geirhos, C. R. Temme, J. Rauber, H. H. Schütt, M. Bethge, and F. A. Wichmann, "Generalisation in humans and deep neural networks," in *Advances in neural information processing systems*, pp. 7538–7550, 2018.

[155] J. F. Bell III, A. Godber, S. McNair, M. Caplinger, J. Maki, M. Lemmon, J. Van Beek, M. Malin, D. Wellington, K. Kinch, *et al.*, "The Mars Science Laboratory Curiosity rover Mastcam instruments: Preflight and in-flight calibration, validation, and data archiving," *Earth and Space Science*, vol. 4, no. 7, pp. 396–452, 2017.

[156] "Himawari real time images."

[157] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[158] E. M. Mikhail and F. E. Ackermann, *Observations and least squares*. Univ Pr of Amer, 1982.

[159] D. Simon, *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[160] J. L. Crassidis and J. L. Junkins, *Optimal estimation of dynamic systems*. CRC press, 2011.

[161] H. D. Curtis, *Orbital mechanics for engineering students*. Butterworth-Heinemann, 2013.

[162] W. Clohessy and R. Wiltshire, "Terminal guidance system for satellite rendezvous," *Journal of the Aerospace Sciences*, vol. 27, no. 9, pp. 653–658, 1960.

[163] S. A. Schweighart and R. J. Sedwick, "High-fidelity linearized J model for satellite formation flight," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 6, pp. 1073–1080, 2002.

[164] S. D'Amico, *Autonomous formation flying in low earth orbit*. PhD thesis, Delft University of Technology, 2010.

[165] J. Sullivan, S. Grimberg, and S. D'Amico, "Comprehensive survey and assessment of spacecraft relative motion dynamics models," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 8, pp. 1837–1859, 2017.

[166] B. E. Tweddle and A. Saenz-Otero, "Relative computer vision-based navigation for small inspection spacecraft," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 5, pp. 969–978, 2014.

[167] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, "Improving the robustness of deep neural networks via stability training," in *Proceedings of the ieee conference on computer vision and pattern recognition*, pp. 4480–4488, 2016.

[168] J. Schmidhuber, *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[169] G. E. Hinton and D. C. Plaut, "Using fast weights to deblur old memories," in *Proceedings of the ninth annual conference of the Cognitive Science Society*, pp. 177–186, 1987.

[170] F. Zhou, B. Wu, and Z. Li, "Deep meta-learning: Learning to learn in the concept space," *arXiv preprint arXiv:1802.03596*, 2018.

[171] A. L. Samuel, "Some studies in machine learning using the game of checkers. II—recent progress," in *Computer Games I*, pp. 366–400, Springer, 1988.

[172] P. Liang, "Machine Learning: Artificial Intelligence: Principles and Techniques."

[173] A. Amidi and S. Amidi, "Cheatsheet: Artificial Intelligence," Sep 2019.

[174] L. Rosasco, E. D. Vito, A. Caponnetto, M. Piana, and A. Verri, "Are loss functions all the same?," *Neural Computation*, vol. 16, no. 5, pp. 1063–1076, 2004.

[175] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[176] M. London and M. Häusser, "Dendritic computation," *Annu. Rev. Neurosci.*, vol. 28, pp. 503–532, 2005.

[177] N. Brunel, V. Hakim, and M. J. Richardson, "Single neuron dynamics and computation," *Current opinion in neurobiology*, vol. 25, pp. 149–155, 2014.

[178] A. Karpathy *et al.*, "Cs231n convolutional neural networks for visual recognition," *Neural networks*, vol. 1, 2016.

[179] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[180] M. A. Nielsen, *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, USA:, 2015.

[181] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in neural information processing systems*, pp. 6231–6239, 2017.

[182] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[183] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[184] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*, vol. 2. MIT press Cambridge, 1998.

[185] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[186] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic Routing Between Capsules," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 3856–3866, Curran Associates, Inc., 2017.

[187] G. Hinton, S. Sabour, and N. Frosst, "Matrix capsules with EM routing," 2018.

[188] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[189] M. D. Shuster *et al.*, "A survey of attitude representations," *Journal of Astronautical Sciences*, vol. 41, no. 4, pp. 439–517, 1993.

[190] A. W. Koenig, T. Guffanti, and S. D'Amico, "New state transition matrices for spacecraft relative motion in perturbed orbits," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 7, pp. 1749–1768, 2017.

Appendices

A

Review of Machine Learning Concepts

Owing to extensive research in the past few decades, the methods used for problem solving using AI are numerous. They can be classified into four categories based on working intelligence level. There are

1. State-based methods
2. Variable-based methods
3. Logic-based methods
4. Reflex-based methods

The working intelligence level of these algorithms is shown in Fig. A.1. Intelligence level here points to the abstraction of information and symbolic representation of the world. The main focus is put on machine learning, which is a reflex-based method. While, other methods are useful and often relate to machine learning, these are not considered relevant for the information here.

Machine Learning

Machine learning is the broad domain of algorithms based on the principle of learning. According to Mitchell [34],

> *A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if it's performances at tasks in **T**, as measured by **P**, improves with the experience **E**.*

This allows machine learning algorithms to solve problems that are difficult to solve using fixed (manually designed) programs. The term machine learning was coined by Arthur Samuel after he constructed an algorithm to teach the computer to learn checkers, which eventually surpassed his own abilities [171]. This aspect of experience-driven self improvement has been a fundamental part of the machine learning revolution that has followed.

Machine learning paradigm stems from the ideology of building up on simpler information at the lower levels to comprehend complex ideas. Lower level information is expressed using *features*, x_i (i=1 ...n), of an *example*, $\mathbf{x} \in \mathbb{R}$. Together, a collection of all such examples or data points make a dataset.

Most machine learning algorithms implement lower level intelligence and fall under the domain of reflex based models in AI. A model is said to be reflex-based, if it performs a fixed sequence of computations on a given input in a feed-forward manner. These models (or networks) have no feedback loop where output of the model is fed-back into itself as an input [1]

Tasks ***T*** include classification, regression, transcription, density estimation, translation et cetera. Performance measure ***P***, is often specific to the task undertaken in a machine learning problem. However, the most

[1] Not all machine learning algorithms are reflex based. e.g. Recurrent neural Networks

Figure A.1: Spectrum of working intelligence level in AI models [172]

common performances measures are accuracy and error rate. Experience ***E*** is usually associated with the kind of data/information that is available to the algorithm during the learning process. Experience forms the basis of the most elementary classification of learning algorithms in to:

1. **Supervised Learning**: The ability to learn the input-output mapping from a dataset of examples or input and their associated **label** or output. In terms of **E**, the supervised learning algorithms experience a dataset along with corresponding labels,

   Mathematically, supervised algorithms observe examples of a random vector $\mathbf{x}$ and the corresponding value $\mathbf{y}$ and learn to predict $\mathbf{y}$ from $\mathbf{x}$ usually by estimating the conditional probability $p(\mathbf{y}|\mathbf{x})$

2. **Unsupervised Learning**: Ability to learn from only a set of inputs and segregate data in some form based on commonalities in the inputs. In terms of **E**, the unsupervised learning algorithms experience only the dataset containing features.

   Mathematically, the problem of unsupervised learning is to observe input data for a random variable $\mathbf{x}$ and attempting to implicitly or explicitly learn the probability distribution $p(\mathbf{x})$ and the associated properties of it.

This broad binary division is not distinct and all-encompassing. However, from the machine learning application perspective, it provides a rough scope of the task involved.

A typical machine learning algorithm development involves training, validation and test phases. Training involves observing the available data and forming the model to minimize the error on the given dataset. Validation and test involves checking the performance of the trained model on the unobserved inputs, emulating performance in a real-world application. This '*training*', 'validation' and '*test*' terminology is adopted in the subsequent sections to explain the relevant concepts.

Supervised learning algorithms are of special interest to this work, as most deep learning methods based on images utilize supervised learning. In particular learning-based terrestrial and space-borne pose estimation applications almost always use supervised learning. Since supervised learning is the most likely type of learning to be adapted for the proposed project, the important aspects of supervised learning are discussed in detail here.

Supervised learning is the problem of estimating a function $h_{\mathbf{w}}(\mathbf{x})$, also called a hypothesis function, based on a set of training examples $\mathbf{x}_i \in \mathbb{X}$ and associated output $\mathbf{y}_i \in \mathbb{Y}$ for $i = 1, \ldots n$, subject to coefficients or weights $\mathbf{w}$. This dataset containing training examples is used to tune the weight parameters, to fit the data by minimizing some form of error, more formally called loss.

Type of Prediction

Based on the type of prediction, a problem in supervised learning is termed as a *regression* problem when the output space ($\mathbb{Y}$) is continuous or a *classification* problem if the output space is discrete.

Loss Function

A loss function, $\mathbf{L_i} : (z, y) \in \mathbb{R}$, quantifies the quality of prediction by taking predicted output value $z = h(\mathbf{x})$ [2] and true value y for an example $\mathbf{x}^i$. The common loss functions are shown in Fig. A.2

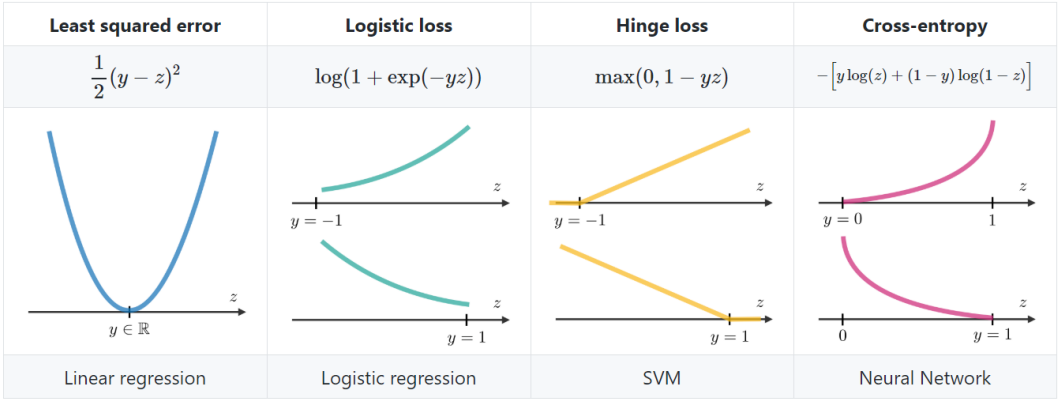


Figure A.2: Common loss functions used in various learning algorithms [173]

The choice of loss function in machine learning algorithms is subjective [174], but for deep learning algorithms the choice of loss function is most often based on convention.

A total loss function generally assesses the performance of the model over the whole dataset, expressed as:

$$J(\mathbf{w}) = \sum_{i=1}^{n} L_i(z, y) \tag{A.1}$$

and can often also include regularization term, for instance:

$$J(\mathbf{w}) = \sum_{i=1}^{n} L_i(z, y) + \Omega_R(\mathbf{w}) \tag{A.2}$$

Often in the literature, objective function is also called *total loss function* and described as a sum of *data loss* and *regularization loss*.

$$L(\mathbf{w}) = \sum^{n} L_i + L_R$$

Gradient Descent

During training, the model learns the weights ($\mathbf{w}$) of the model. During this process, weights are updated such that the cost function or the total loss function decreases towards the minimum. This is done using gradient descent (or ascent if dealt with maximum). Simplistically, the gradient descent update can be expressed as

$$\mathbf{w_1} = \mathbf{w_0} - \alpha \nabla J(\mathbf{w_0}) \tag{A.3}$$

The parameter α which specifies the magnitude for update of weights towards a lower gradient, is called the learning rate. Learning rate is an important hyperparameter, especially in neural networks. Smaller learning rate causes consistent progress towards minima but is slow. On the other hand, higher learning rate makes larger weight updates and lead to overshoot away from the minima in the direction of the local gradient. The discussion on computation of the gradient of the objective function ($\nabla J(\mathbf{w_0})$) is presented within the discussion on backpropagation and neural networks in A

The update of weights using gradient descent can further be done in two ways during the training. If weights are updated after each training example is observed, the approach is called Stochastic Gradient Descent (SGD). On the other hand, if the weights are updated on a the whole batch of training examples, it is called Batch Gradient Gescent (BGD). BGD shows great performance for convex models while SGD tends

[2] Note: It is assumed that the hypothesis function provides a scalar output y. This is done to describe loss functions with simplicity. It is often the case that output from the model is a vector ($\mathbf{y}$). In that case, loss functions are similar just adapted to the vector notation of the problem.

to find near-minimum solution with faster convergence. However, the typical deep learning models using large amount of training data (on the order of 10^6 examples) do not use either of those methods in their pure form. This is because BGD shows inefficiency in computing gradients for all examples at every step and SGD shows inefficiency as it cannot be implemented in vectorized form. Instead, the update is computed in mini-batches of the training data assuming correlation between the data and the mini-batch. The mini-batch gradient descent is usually also interchangeably called stochastic gradient descent in practical resources like libraries.

Note that the task of choosing weights to minimize the objective function can be accomplished using other optimization methods like Newton's method. Gradient descent is just one such method that happens to be widely used in machine learning.

Generalization, Capacity and Regularization

The central aspect of any machine learning problem is that of training the model, such that it subsequently performs well on inputs not directly observed during the training. This ability to perform well on previously unseen data is called generalization. While we use several mathematical optimization tools in machine learning, the essential difference between the two is that optimization seeks to minimize the error only on the observed data. Machine learning algorithms aim at further minimizing the **generalization error** or **test error**.

Usually, the interpretation that allows us to formulate the learning process to take generalization error in to account follows from a set of assumptions regarding the probability distribution of the training and test data. It is generally assumed that examples in each dataset are independent of each other, and that the training and test datasets are identically distributed, taken from the data generating distribution. Ideally, if a fixed model is available and training and test sets are then sampled from the associated probability distribution, the training and test error would be exactly same.

In machine learning problems however, we sample the dataset for training and then fix the model parameters using this set to minimize the training error. For this process, the test error is greater than or equal to the expected value of the training error. Thus the important aspects for developing or implementing an machine learning algorithm is its ability to:

1. minimize the training error
2. minimize the gap between training and test error

This relates to the principle problems in machine learning:

1. **Underfitting**: The inability of the model to achieve a sufficiently low training error.
2. **Overfitting**: The inability of the algorithm to achieve a sufficiently low gap between the training and the test error

Varying the capacity of a model is one way to control overfitting or underfitting. Model's capacity defines its ability to fit a wide variety of function [175]. Lower capacity means the model can fit a wide variety of functions for the given data, while higher capacity means the model fits the data more stringently. Generally, the capacity of the machine learning model should be appropriate for the true complexity of the task and the amount of data available for training. Lower than needed capacity can result in a simpler model that fails to sufficiently reduce training error, while higher than needed capacity can preserve information from the training set that does no propagate in the test set, resulting in increasing test error. An example of variation of capacity is shown in Fig. A.3.

Generally, the variation of performance with model capacity is such that the training error saturates at a minimum with increasing model capacity, while the test or the generalization error decreases up to the optimal capacity and then increases with higher model capacity. This is shown in Fig. A.4. Note that at optimal capacity, the model's training and test error are still not equal. This is because generally the data available might be stochastic (in terms of input-output mapping), contain noise or include other unsampled variables. Thus, even the ideal model, which knows the true data generating distribution, incurs error in predicting the test set results. This offset error of an ideal model is called the Bayes error. Thus, generalization error of any model is greater than or equal to the Bayes error.

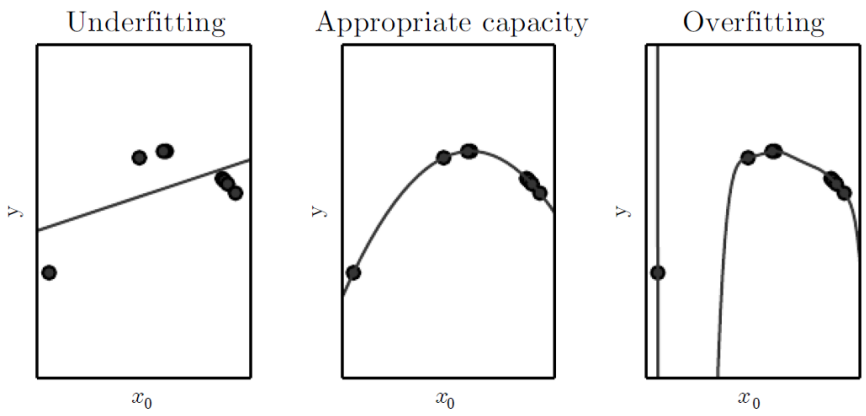


Figure A.3: An example of model fitting with varying capacity. Data is based on a quadratic function ($y = a_0x^2 + a_1x + a_2$) for randomly sampled independent variable (x). For limited training data, linear model with lower capacity (left) fails to fit the training data well. A nine degree polynomial (right) fits the training data well, but overfits the available data points as it cannot generalize well for test data from the given quadratic model. [175]

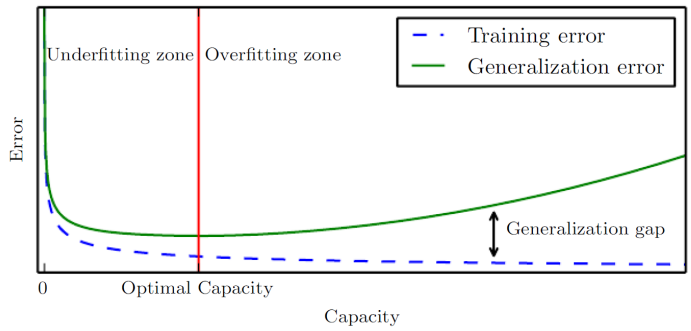


Figure A.4: Typical relationship between model capacity and errors in training and test [175]

It should be noted that it is possible for a model with optimal capacity to have significantly high gap between generalization and training error. In this case, this gap can be reduced by including more training examples. The expected generalization error can never increase as the number of training examples increases [175].

Regularization

According to the **no free lunch theorem** for machine learning [65], no machine learning algorithm performs better than others when averaged over all data generating distributions. On making assumptions about the probability distribution of a specific real-world application, machine algorithms can be designed to perform well on a specific distribution. Conversely, these perform worse on other data generating distributions. This implies that machine learning algorithm cannot be universal or the absolute best learning algorithm. Therefore, the machine learning algorithms are designed to perform well on a distribution specific to the real-world task that is considered and not others.

This task specific performance improvement is done by setting preferences. A general method of doing that is to set preferences for one function over another in the hypothesis space. This implies that while two or more functions are eligible, one of them is preferred. This is called regularization. Regularization is often added as a penalty to the cost/objective function. A common regularization method called weight decay adds a penalty in the cost/objective function given as:

$$J(\mathbf{w}) = J_0(\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w} \tag{A.4}$$

where, J_0 is the objective that quantifies the deviation between predicted data and the true data and λ is a chosen parameter that defines the strength of preference for smaller weights in the model. Smaller weights

ultimately affect model capacity (by avoiding selection of narrow space of function associate with unbalanced weights) improving generalization as shown in Fig. A.4 and thus also affect overfitting and underfitting.

For example, consider a simple linear objective function such that

$$\begin{aligned} J_0(\mathbf{w}) &= \mathbf{w}^T\mathbf{x} \qquad s.t. \\ \mathbf{x} &= [1,\ 1,\ 1,\ 1]^T \\ \mathbf{w}_1 &= [1,\ 0,\ ,0,\ 0]^T \\ \mathbf{w}_2 &= [0.25,\ 0.25,\ 0.25,\ 0.25]^T \end{aligned}$$

we have,

$$J_0(\mathbf{w}_1) = \mathbf{w}_1^T\mathbf{x} = \mathbf{w}_2^T\mathbf{x} = J_0(\mathbf{w}_2)$$

Using weight decay regularizer,

$$\Omega_R(\mathbf{w}) = \lambda\mathbf{w}^T\mathbf{w}$$

preference can be set for $\mathbf{w}_2$, even though both are eligible.

Weight decay is just one regularization method for the learning algorithms. More broadly, regularization is " *any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error*" [175]. Typical regularization techniques in relevant learning algorithms are discussed further in later sections.

Hyperparameters and Validation

In machine learning algorithms, certain parameters are often chosen manually either because they are hard to optimize or are unsuitable to learn in the training set. One such parameter is λ coefficient of the weight decay regularizer discussed earlier. Another could be the degree of polynomial used to represent a model. These hyperparameters are essential in controlling the behaviour of the learning algorithm. These are not learned during training, because for the training dataset the hyperparameters would choose the maximum possible model capacity, causing overfitting. Thus, hyperparameters are tested and learned on a separate **validation set**. Typically, validation set is formed by isolating 20 % of the training dataset, which is used, in a way, to train or select the hyperparameters. Consequently, only 80% of the original dataset is used for training the model.

Representation Learning and Neural Networks

Recall that the capacity of the model of a machine learning algorithm used on a data needs to be appropriate for the task and the data. In the case of linear models however, the capacity is restricted to lines. Two methods of overcoming this have been discussed so far in brief. One, by manually transforming an example $\mathbf{x}$ to $\phi(\mathbf{x})$ with nonlinear mapping ϕ to appropriate features in a higher dimensional space. Another way is use a generic infinite kernel method that implicitly learns the nonlinear mapping (ϕ)in a higher dimensional space that assures a fit on the training data, as discussed for SVM algorithms. These kernel tricks generally use an infinite dimensional mapping ϕ that provides high capacity in the model to smoothly fit the data. However, this generally leads to higher than appropriate capacity for the model, which often leads to poor generalization on the test set as expected from Fig. A.4. Both the approaches do not provide a scalable solution for difficult learning problems.

The domain of representation learning algorithms overcomes the pitfalls of the traditional learning algorithms, by appropriately learning the non-linear mapping for the training examples directly from the data. Given a non-linear model with the usual notation:

$$y = f(\mathbf{x};\mathbf{w}) \tag{A.5}$$

the representation learning algorithms form an additional intermediate mapping such that such that

$$\begin{aligned} \mathbf{z} &= \phi_1(\mathbf{x};\mathbf{w}_1) \\ \mathbf{y} &= \phi_2(\mathbf{z};\mathbf{w}_2) \end{aligned}$$

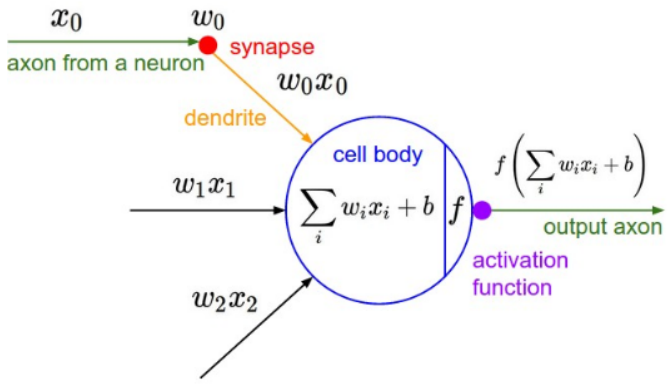


Figure A.5: A representation of an artificial neuron with mapping an arbitrary mapping f

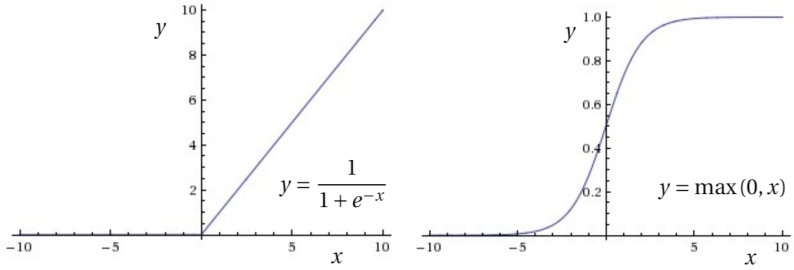


Figure A.6: Sigmoid (left) and ReLU (right) functions used as activation functions for artificial neurons

This forms the conceptual core of neural networks, which use this basic principle on a larger scale to learn complex[3] non-linear mapping between the input and the output by stacking many of these basic intermediate mappings to form a network.

Neural Networks

Neural networks were primarily associated with the goal of modelling the biological system that enable the human brain to comprehend complex ideas. However, the current state of artificial neural networks is only remotely linked to the biological analogy and has been shaped by engineering problems. However, the structure and flow of information is connected to the biological neural networks.

The commonly used model of the artificial neuron very coarsely imitates a biological neuron [176, 177], but is significantly simpler. In that, each neuron has dendrites (the input channels), the cell body (operations and the mapping function) and an axon (the output channel). Axons from multiple neurons, connect to the dendrites on one neuron. The outputs from a neuron that connects to another neuron by axon-dendrite connection, can be assumed to be weighted. This is represented in Fig. A.5. When the sum of the inputs is above a certain threshold, the neuron fires, sending a spike or a signal along the axon. The output is produced as a function of the weighted sum of all the inputs. This function is called the activation function. The central idea for neural network is based on activation or firing.

The choice of the activation function f has historically been the sigmoid function [4]. There are several other activation functions, among which a rectified linear unit or ReLU is the most common activation function in recent use of deep learning. ReLU and sigmoid functions are shown in Fig. A.6.

The neuron unit is then stacked together to form a network with a layer-wise arrangement called a neural network. Mathematically, the neural networks are neurons connected in an acyclic graph. The layer that takes the input or the example $\mathbf{x}$ is called the input layer. The input layer connects to intermediate layers

[3] Not related to complex numbers, but complexity as a characteristic

[4] activation functions are always non-linear as a cascade of linear units just form a linear mapping together, restricting the capacity only to linear functions

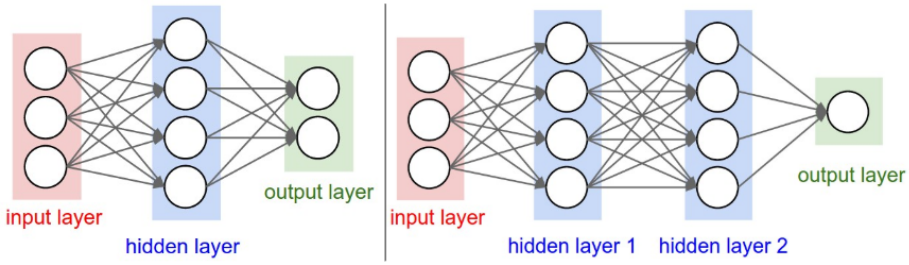


Figure A.7: 2 layer (left) and 3 layer (right) neural networks [178]

and finally form an output layer, which provides the final output $\mathbf{y}$ [5]. The inputs and outputs of each neuron are usually scalar, which means that the input layer contains the amount of neurons equal to the number of features in an example. Subsequently, the neurons in the intermediate layers indirectly operate in a fixed way on the input. This implies that input to a neural network is fixed in sized and pre-determined. As a result, a neural network processing images will not be able to deal with input images of different sizes.

The intermediate layers are called hidden layers. The hidden layers form the foundation of representation learning. The way each neuron in preceeding layer is connected to the hidden layer with the associated weights enable the network to represent a more complex non-linear mapping between input and output. Finally, the output layer provides the output parameters (scalar or vector) and does not use an activation function. Outputs generally linearly combine the outputs of the preceding layer and act as a container for a real value (regression) or a discrete value (classification). Neurons are connected from one layer to the next layer do not connect to neurons in any other layer or itself. When neurons of a hidden layer are connected to every other neuron in neighbouring layers, the hidden layer is called a fully connected layer. Fig. A.7 shows two exmaple neural networks with fully connected layers. Note the convention n-layer neural network, which specifies total number of layers excluding the input layer.

The striking feature of the neural networks in general is that they are powerful approximators. This is due to the cascaded and heirarchical non-linear functions defined by respective weights and biases at each connection. These weights and biases are the learning parameters. For instance, 2-layer neural network in Fig. A.7 (left) has 20 weight parameters defined at each of the 20 connections between the neuron and 6 bias parameters, one defined at each of the 6 neurons, making a sum total of 26 learnable parameters. Similarly, in the neural network shown in Fig. A.7(right) has 46 learnable parameters.

These learnable parameters allow neural networks to approximate highly non-linear mappings between input and output. In fact the universality theorem or the universal approximation theorem for sigmoidal function given by Cybenko[179], implies that a neural network made with sigmoid neurons (or any continuous, bounded and non-constant activation function) with atleast one hidden layer is a universal approximator. For a clear visual proof of this theorem, the reader is referred to [180]. This means that a two-layer neural network ($y = g(x)$) between a single input (x) and output (y) can be constructed to approximate virtually any function $f(x)$ with an arbitrary accuracy ϵ such that $|f(x) - g(x)| < \epsilon$. Practically, the size of the hidden layer in terms of the number of neurons increases exponentially depending on the non-linearity and the input space dimensions.

The modern neural networks, especially in deep learning, use ReLU instead of a sigmoid activation function. The reasons motivating the choice of ReLU over sigmoid function is presented later in the discussion on backpropagation. Intuitively, the visual proof in [180] can also be extended to ReLU function. However theoretically, notice that the ReLU function (Fig. A.6) is unbounded and violates the condition in the universality theorem. Lu et al. [181] recently showed that networks with ReLU functions can in fact universally approximate any continuous function with n-dimensional input space, with one hidden layer containing $n+1$ neurons. This is the reason why neural networks perform well on learning arbitrary non-linear mappings between input and output, where number of layers and the number of neurons in each layer specifies a family of functions. In practice the shallow neural networks used in machine learning with 3-layers are considered optimal and sufficient for all practical use, in contrast deep neural networks like CNN that contain around 10 or more layers and exponentially higher learnable parameters to extract lower level information

[5] generalized as a vector but can be scalar

from images.

A Neural network for practical purposes is defined by it's architecture- the number and type of the hidden layers as well the size of each of those layers. These aspects of the architecture define the capacity of the neural network i.e. the space of hypothesis functions increases. Larger neural networks will always perform better than smaller neural networks. However, arbitrarily choosing higher size and number layers can lead to the common problems with overfitting as discussed earlier. Finding the perfect combination of layers and their sizes is not practical or scalable to wide variety of practical applications. For this reason, regularization is the most preferred way to control the overfitting in these cases. Here, the user only decides a single hyperparameter λ from Eq. A.4, while the architecture of the network is more or less fixed to a larger network.

Notice that apart from the configuration the architecture, the user has no role in choosing the exact mapping. The neural network takes care of all the intermediate computations itself in training as well as the test. For this reason the neural networks are said to execute end-to-end learning. This end-to-end learning allows neural network to extract complex features from the data to fit a model that allows it to execute a machine learning task well. However, these complex features on which the neural network bases its decisions, usually make little sense to the user. For this reason, a neural network is often termed as a black box, which invents representations in the hidden layers that are not intuitive or understandable, thus preventing debugging or troubleshooting in a meaningful way.

Training, Learning and Backpropagation

The process of output computation from input in a neural network flows through layers in a feed-forward manner, also called forward propagation or forward pass. The forward pass results in a loss. The adjustment to the learnable parameters that fit the training examples that lowers the loss, is determined by backpropagation or backprop [182].

The training process consists of feeding a training example to the network and forward propagating the inputs until the output layer to determine a scalar cost $J(\mathbf{W})$, where $\mathbf{W}$ is a matrix containing weights and bias terms. The back propagation algorithm determines the gradients from the final cost by processing information in a backward flow. This gradient is then used in an SGD algorithm as discussed earlier. Note that backpropagation is only concerned with computing gradients and is not the same as learning. Recall that in order to make appropriate updates to the weights in the model that minimize the cost J, the gradient ∇J needs to be computed with respect to all the weights. This could be done by analytical or numerical gradient calculation, but for neural networks analytical gradient is very difficult and practically impossible to compute and the numerical gradient calculations are computationally expensive. Backpropagation overcomes these limitations by using simple application of chain rule for derivatives to compute the gradient in a computationally inexpensive way.

Backpropagation algorithms are described using computational graphs, which is an intuitive representation of a composite mathematical computation in terms of simpler operations defined as nodes. A computational graph representation is shown in A.8.

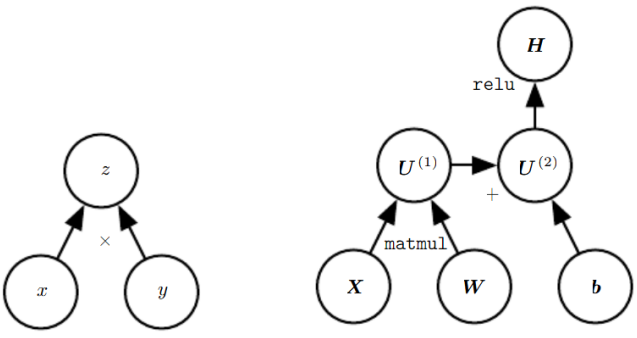


Figure A.8: Computational Graph representation for arbitrary functions $z = xy$ (left) and $H = \max(0, \mathbf{XW} + \mathbf{b})$

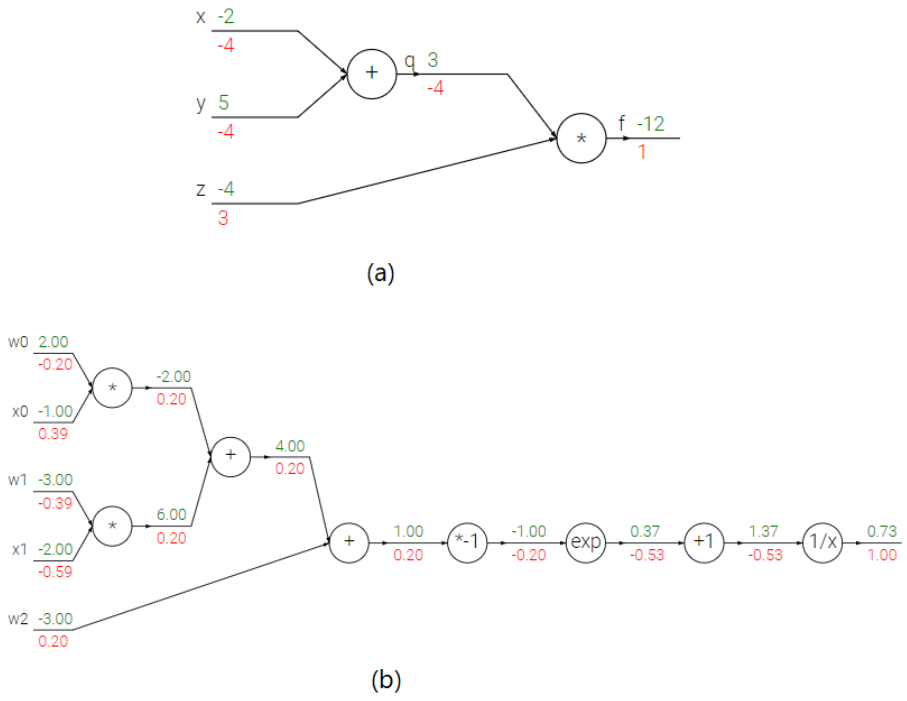


Figure A.9: Backpropagation represented in a computational graph

This is used in conjunction with the chain rule for derivative from calculus, which can be written as:

$$\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{A.6}$$

where, $\mathbf{y} = g(\mathbf{x})$, $z = f(\mathbf{y})$, and x_i and y_j are respective elements of vectors $\mathbf{x}$ and $\mathbf{y}$

During backprop, starting from the end node, the break down of cost function in to gradients is done at every preceding node. As an example, in Fig A.9 (a), the numbers in green are the function evalutions during forward propagation the numbers in red are gradients. During backward propagation we start from f where the gradient of before the node is $df/df = 1$. The value f is basically $f = qz$. Thus, gradient of f with respect to q, $df/dq = z$ (similarly for z). In the same way, $q = x + y$ and the local gradients $dq/dx = dq/dy = 1$. The gradient of f with respect to x is then a product of local gradient (dq/dx) multiplied by the preceding local gradient (df/dq). Fig. A.9 (b) shows the backprop done for a sigmoid neuron where $f(\mathbf{w},\mathbf{x}) = (1 + exp(-w_0x_0 - w_1x_1 - w_2)^{-1}$. In general, backprop breaks the gradient computation into very simple computations of computing gradient using local relations and only the preceding gradient. This concept is extended to vector/matrix operations as:

$$\nabla \mathbf{x} z = \mathbf{J}^T \nabla \mathbf{y} z \tag{A.7}$$

where, $\mathbf{J}$ is the jacobian matrix of g and the representation of variables is same as Eq. A.6. This is a very valuable tool, especially for deep learning where number of layers and the number of learnable parameters are very high. Hence, majority of deep learning libraries and platform use a similar approach for computing gradients.

Following backpropagation, the process of gradient descent progresses as discussed earlier. Combining these elements discussed in the previous sections, a complete process of training and testing a machine learning in general and a neural network in particular has been discussed. In section A, the shortcomings of shallow learning are outlined along with a review of relevant aspects of deep learning algorithms.

It should be noted that since backpropagation is about computing gradient, the performance of backpropagation in neural networks depend highly on the choice of activation functions. The traditional choice

of sigmoid activation function for the neural networks is non-zero centered exhibits near-zero gradients in saturate regions during activation. This causes weight updates during SGD to be very slow increasing computation for convergence. Krizhevsky et al [67] proposed the use of ReLU activation which accelerated SGD convergence by a factor of 6. It is also superior to other activation functions like tanh, as ReLU only involves thresholding matrices at zero, while tanh is a computationally expensive trigonometric computation. For this reason, almost all modern neural networks since 2012 prefer ReLU activation function over others.

Deep Learning

The domain of Deep Learning originates directly from the neural networks discussed in the previous section. Formally but not strictly, a neural network that contains more than one hidden layer between the input is called a deep neural network and is said to execute deep learning. Note that the neural networks in deep learning need not be feed-forward networks, like recurrent neural networks [183] and need not have the same learning procedure as described earlier, like in deep reinforcement learning [184]. Deep learning is a subset of machine learning and falls withing the larger domain of Artificial intelligence, as shown in Fig. A.10. The characteristic difference of deep learning from other techniques is summarized in A.11.

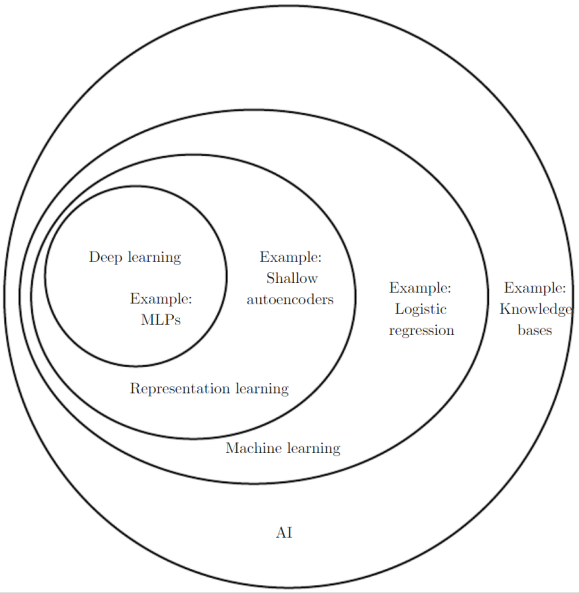


Figure A.10: Venn Diagram showing DL in AI domain [175]

Deep learning is powered by the concept of extracting abstract representations at an unintuitively low levels in unstructured data, to compose high level information. Here, unstructured data is meant to signify data with high dimensions such that no direct and fixed model from which patterns could be identified. Speech and images are the most prevalent unstructured data that are easily deciphered by humans, but difficult to be understood by the computers. Unstructured data represents challenges for understanding the inputs, like images, in the following ways as outlined in [178]. For an image describing the same object, for instance, it can have:

1. **Viewpoint variation**: A single instance of an object can be oriented in many ways with respect to the camera.
2. **Scale variation**: Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
3. **Deformation**: Many objects of interest are not rigid bodies and can be deformed in extreme ways.
4. **Occlusion**: The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
5. **Illumination conditions**: The effects of illumination are drastic on the pixel level.

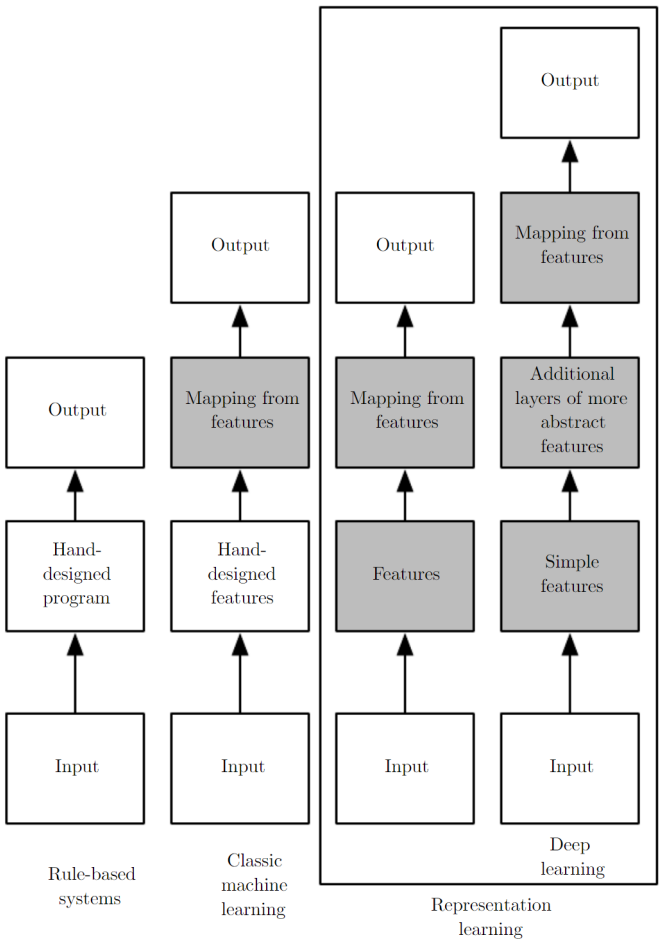


Figure A.11: Distinguishing features of different AI techniques [175]

6. **Background clutter**: The objects of interest may blend into their environment, making them hard to identify.

7. **Intra-class variation**:. The classes of interest can often be relatively broad, such as chair. There are many different types of these objects, each with their own appearance.

Traditional machine learning algorithms like linear classification/regression do not scale well for such inputs and other CV algorithms cannot tackle one or more of the aforementioned challenges. An exhaustive discussion on deep learning is neither practical for this literature study nor relevant. The reader is referred to the established texts for these topics by Goodfellow and Bengio [175] and Nielson [180]. This work will be mostly be focused on the study of a specific type of architecture in deep learning used especially for processing images, called the Convolutional Neural Network (CNN). CNNs are the central to recent progress in computer vision since they were first demonstrated with substantial performance increase on ImageNet classification benchmark [66] by Krizhevsky et al. [67] in 2012. However, CNNs are not a recent concept. LeCun et al. [68] motivated CNNs for use in recognition of handwritten zip codes for US postal service and later formalized the CNN class of architecture that overcomes the limitations of traditional neural networks by accounting for local receptive fields, shared weights and subsampling to ensure some degree of invariance to overcome challenges in visual imagery listed above [69].However, with trisection of advances in processing power, increase in image data and progress in deep learning, in 2012 when AlexNet [67] showed unparalleled error rate in image classification. The performance of CNNs has since improved with evolution of architectures [70–72] on the benchmark ImageNet classification beyond human performance. CNNs have thus empowered computer vision to address several crucial terrestrial applications in a very short time [185].

Finally, it is worth noting that CNNs are not the ideal solution to the high level information processing from images despite their pioneering performance. They are considered to have drawbacks in not being able to "understand" the real 3D/4D world information that is easily captured by the human brain. In latest research, Sabour et al. [186] propose a better heirarchical network (CapsNet) similar to the CNNs that captures dynamic information and is based on a basic unit made of a group of neurons called a capsule. The capsules among other things preserve probability of observation of the features and overcome translation ambiguity in contrast to CNNs. Hinton et al.[187] have shown better performance with CapsNet on smallNORB benchmark, but the overall establishment of capsules is still subject to extensive research.

B

Attitude Representations

This chapter describes the attitude representations used for describing the rotational kinematics and dynamics of a body, adapted from [188]. For a more comprehensive review of attitude representation, the reader is also referred to the survey in Shuster [189].

Rotation/Attitude Transformation Matrix (**R**) : A matrix which when multiplied with a vector, rotates it while keeping the length constant. For all such matrices defining the Special Orthogonal group with a 3x3 matrix ($\mathbf{R} \in SO(3)$):

$$|\mathbf{R}| = \pm 1 \quad \& \quad \mathbf{R}^{-1} = \mathbf{R}^{T} \tag{B.1}$$

where, the matrices can be represented in terms of their elements as:

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{B.2}$$

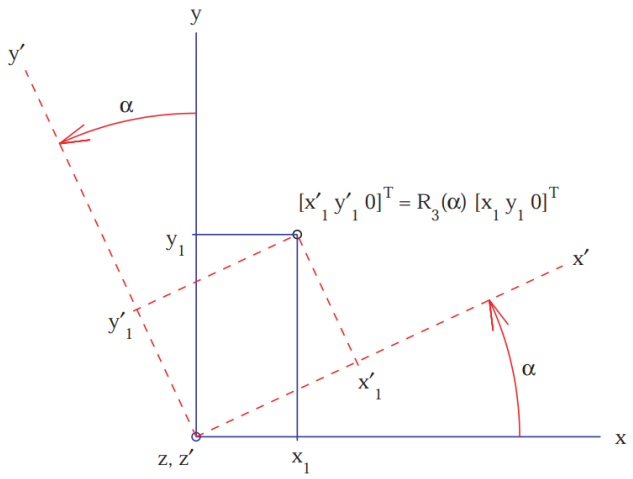


Figure B.1: Visualization of a generic coordinate rotation

B.0.1. Direction Cosine Matrix (DCM)

It is a matrix (C_{ij}) describing the transformation from a generic orthogonal reference frame to another frame through the rotations about its axes. It is then a 3×3 matrix with 9 elements with no singularities. However,

it contains 6 redundant elements of trigonometric representation that increase computational complexity in operation. For a generic rotation in 3D, the DCM can be represented as following (notation in Fig. B.1.

$$\mathbf{C} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{x',x}) & \cos(\theta_{x',y}) & \cos(\theta_{x',z}) \\ \cos(\theta_{y',x}) & \cos(\theta_{y',y}) & \cos(\theta_{y',z}) \\ \cos(\theta_{z',x}) & \cos(\theta_{z',y}) & \cos(\theta_{z',z}) \end{bmatrix} \tag{B.3}$$

B.0.2. Euler Angles

It is a set of three angles (θ_i) expressing the rotations over single axes of an orthogonal matrix A such that it can be expressed with respect to another orthogonal frame B. These rotations about the axes are performed in a sequential way, generating intermediate frames between A and B. For an arbitrary orthogonal axes system XYZ, the rotation matrix $\mathbf{R}$ resulting from a sequential rotation along X,Y and Z axes, is given as a multiplication of individual Euler angle rotations (θ_i) as following:

$$\mathbf{R} = \mathbf{R}_3(\theta_3)\,\mathbf{R}_2(\theta_2)\,\mathbf{R}_1(\theta_1) \tag{B.4}$$

where, the elementary rotation matrices $\mathbf{R}_i$ for an arbitrary rotation (α) is given as:

$$\begin{aligned} \mathbf{R}_1(\alpha) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \\ \mathbf{R}_2(\alpha) &= \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix} \\ \mathbf{R}_3(\alpha) &= \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{B.5}$$

The sequence of rotations is subjective to conventions, but the generic representation presented here remains valid. The main issue arising from this representation is the existence of singularity in rotations resulting from gimbal lock.

B.0.3. Axis-Angle Representation

It is a four parameter representation such that a three parameter unit vector ($\mathbf{e}$) describes the axis of rotation called Euler's eigen-axis or simply, Euler axis, while the fourth parameter (θ) is the rotation around that axis. This is visualized in Fig. B.2. The unit vector $\mathbf{e}$ represents the direction cosines and is subject to unit vector normalization constraint.

Euler's eigenaxis rotation states that by rotating a rigid body about an axis that is fixed to the body and stationary in an inertial reference frame, the rigid-body attitude can be changed from any given orientation to any other orientation. Such an axis of orientation, whose orientation relative to both inertial reference frame and the body remains unchanged throughout the motion, is called the Euler axis or eigenaxis.

B.0.4. Unit Quaternions

It is a set of four normalized elements (q_i) describing a single rotation about the Euler's eigenaxis of the motion. This attitude representation set is common for space applications as it renders a linear kinematic relationships, as well as the absence of rotation singularity in discrete rotations. A quaternion is represented in general form as:

$$\mathbf{q} = \begin{bmatrix} q_0 \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} q_0, q_1, q_2, q_3 \end{bmatrix}^T \tag{B.6}$$

where, $\mathbf{q}_v$ is the quaternion vector representing the direction of the eigen-axis, while q_0 is the scalae representing magnitude of rotation about it. A unit quaternion is subject to a normalization constraint given as:

$$\mathbf{q} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{B.7}$$

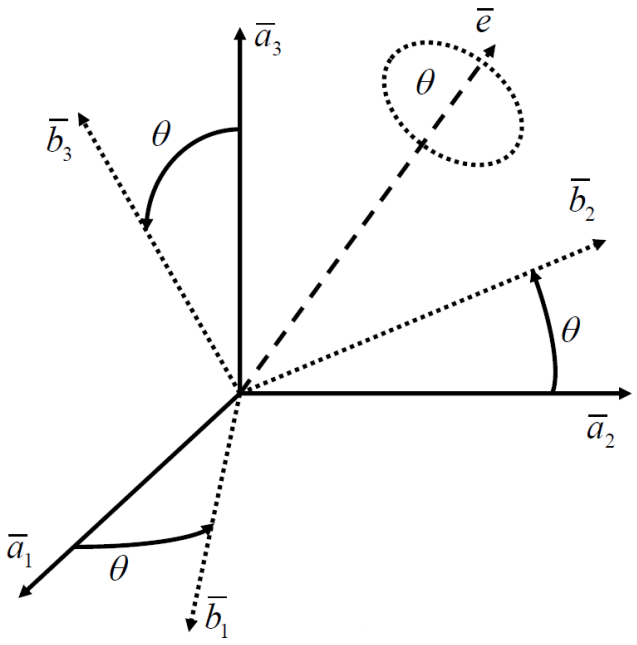


Figure B.2: Visualization of rotation in axis-angle parameterization

The relevant quaternion operations commonly utilized in spacecraft dynamics are the quaternion conjugate, quaternion inverse and quaternion multiplication.

The quaternion conjugate $\tilde{\mathbf{q}}$ and the quaternion inverse $\mathbf{q}^{-1}$ are given as:

$$\overline{\mathbf{q}} = \begin{bmatrix} q_0 \\ -\mathbf{q}_v \end{bmatrix} \quad ; \quad \mathbf{q}^{-1} = \frac{\mathbf{q}}{\|\mathbf{q}\|} \tag{B.8}$$

$$\begin{aligned} \mathbf{q} \otimes \mathbf{p} &= \mathbf{q}_m(\mathbf{q}, \mathbf{p}) \\ &= \begin{bmatrix} q_0 p_0 - \mathbf{q}_v^T \mathbf{p}_v \\ q_0 \mathbf{p}_v + p_0 \mathbf{q}_v - \mathbf{q}_v \quad \times \mathbf{p}_v \end{bmatrix} \end{aligned} \tag{B.9}$$

The quaternion vector can be derived to and from Euler angles as :

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\frac{\theta_1}{2}\cos\frac{\theta_2}{2}\cos\frac{\theta_3}{2} + \sin\frac{\theta_1}{2}\sin\frac{\theta_2}{2}\sin\frac{\theta_3}{2} \\ \sin\frac{\theta_1}{2}\cos\frac{\theta_2}{2}\cos\frac{\theta_3}{2} - \cos\frac{\theta_1}{2}\sin\frac{\theta_2}{2}\sin\frac{\theta_3}{2} \\ \cos\frac{\theta_1}{2}\sin\frac{\theta_2}{2}\cos\frac{\theta_3}{2} + \sin\frac{\theta_1}{2}\cos\frac{\theta_2}{2}\sin\frac{\theta_3}{2} \\ \cos\frac{\theta_1}{2}\cos\frac{\theta_2}{2}\sin\frac{\theta_3}{2} - \sin\frac{\theta_1}{2}\sin\frac{\theta_2}{2}\cos\frac{\theta_3}{2} \end{bmatrix} \tag{B.10}$$

and with Axis-angle representation as:

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \mathbf{e}\sin\frac{\theta}{2} \end{bmatrix} \tag{B.11}$$

While discrete rotations do not exhibit singularity with unit quaternions parameterization, it possesses discontinuity at 180 degrees, that causes problem during attitude tracking in state estimation.

B.0.5. Classical Rodrigues Parameters

It is a solution to solve the issues resulting from redundant four parameter representation in quaternion by deriving a three parameter representation(τ_i). It imposes the quaternion normalization constraint on the fourth element such that only three parameters are needed for attitude representation. The representation

generates a tracking discontinuity (singularity) at $\theta = \pm\pi$, allowing larger angles to be represented continuously in comparison to the quaternion representation. However, the kinematic equations are no longer linear as is the case for quaternions. The representation can be expressed in terms of Axis-angle representation as:

$$\boldsymbol{\tau} = \mathbf{e} \tan \frac{\theta}{2} \tag{B.12}$$

B.0.6. Modified Rodrigues Parameters

Similar to the Classical Rodrigues Parameters, it constraints the fourth term of the quaternion to express the attitude in terms of 3 parameters (a_i). Although it delays the singularity to $\theta = \pm 2\pi$, the kinematic equations are nonlinear.

$$\mathbf{a} = \mathbf{e} \tan \frac{\theta}{4} \tag{B.13}$$

C

Data Format and Configuration Files

SPEED Raw JSON Format

```
[
...
    {
        "filename": "img000240.jpg",
        "q_vbs2tango": [0.29082, 0.911185, 0.0209, -0.291082],
        "r_Vo2To_vbs_true": [-0.372055, -0.078814, 4.203779]
    }
...
]
```

Listing C.1: *SPEED* pose data example

Bounding Box JSON Format

```
[
...
    {"id": 5,
    "filename": "frame5.jpg",
    "bbox": [221.73316346862498, 312.669029233875, 200.99467887772502,
    298.010205766775],
    "features": [{"ID": 0, "Coordinates": [254.392077452, 247.485597363], "Visibility":
     1}, {"ID": 1, "Coordinates": [244.5518203905, 253.1792486995], "Visibility": 1}, {
    "ID": 2, "Coordinates": [262.569122631, 261.9557940015], "Visibility": 1}, {"ID":
    3, "Coordinates": [253.1282111505, 267.7248714325], "Visibility": 1}, {"ID": 4, "
    Coordinates": [264.379995976, 290.969447541], "Visibility": 1}, {"ID": 5, "
    Coordinates": [229.7143117105, 231.039419514], "Visibility": 1}, {"ID": 6, "
    Coordinates": [225.8666119125, 233.4276717185], "Visibility": 1}, {"ID": 7, "
    Coordinates": [260.6712322575, 293.60040909], "Visibility": 1}, {"ID": 8, "
    Coordinates": [263.433070378, 262.643677296], "Visibility": 1}, {"ID": 9, "
    Coordinates": [250.2814958095, 270.929975144], "Visibility": 1}, {"ID": 10, "
    Coordinates": [254.2100442415, 246.7730830405], "Visibility": 1}, {"ID": 11, "
    Coordinates": [240.465035016, 254.0293510745], "Visibility": 1}, {"ID": 12, "
    Coordinates": [235.5857128195, 246.955818306], "Visibility": 1}, {"ID": 13, "
    Coordinates": [259.505633217, 275.5097716965], "Visibility": 1}, {"ID": 14, "
    Coordinates": [280.7701570835, 205.4044755545], "Visibility": 1}, {"ID": 15, "
    Coordinates": [308.53558079, 238.3137828765], "Visibility": 1}],
    "pose": {
        "r": [0, 0, 150.0],
        "q": [0.6123724356957946, -0.35355339059327384, 0.6123724356957946,
    0.35355339059327384]
    },
    "illumination": []}
...
]
```

Listing C.2: Data Format used to train/test OD network

COCO JSON Format

```
{
    "info": {
        "description": "Envisat: Image Corruptions",
        "url": "https://tudelft.nl",
        "version": "1",
        "year": 2020,
        "contributor": "TU Delft Space Engineering",
        "date_created": "2019/06/01"
    },
    "licenses": {
        "id": 0,
        "url": "N/A",
        "name": "N/A"
    },
    "images": [
    ...
        {
            "license": 1,
            "file_name": "frame10.jpg",
            "coco_url": "",
            "height": 512,
            "width": 512,
            "date_captured": "",
            "url": "",
            "id": 10
        },
    ...
    ],
    "annotations": [
        {
            "segmentation": [],
            "num_keypoints": 12,
            "area": 15021.805948880521,
            "iscrowd": 0,
            "keypoints": [232.3673248675, 227.731243017, 2,
                234.2280647135, 241.584184091,  2,
                258.927872022,  221.6255002015, 2,
                261.113501245,  236.1497019495, 2,
                313.222118447,  259.987824034,  2,
                202.792812212, 284.452667031,   2,
                207.1060294585, 298.8305785945, 2,
                317.4950907715, 273.9966835845, 2,
                282.3744001285, 279.6892250175, 2,
                285.388164081, 299.2620538315,  2,
                252.464551336, 287.1199161695,  2,
                253.930608754, 306.1032908375,  2
            ],
            "image_id": 10,
            "bbox": [ 190.84200806027, 209.67469604976998, 138.60388686295997,
    108.37939893946003
            ],
            "category_id": 2,
            "id": 1
        },
        ...
    ],
    "categories":[
        {"supercategory":"Target",
         "id":2,
         "name":"Envisat",
         "keypoints":["1","2","3","4","5","6","7","8","9","10","11","12"],
         "skeleton":[[1,2],[2,4],[4,3],[3,1],[5,6],[6,7],[7,8],[8,5],
    [9,10],[10,12],[11,12],[11,9],[1,11],[2,12],[3,9],[4,10]]
         }
}
```

Listing C.3: COCO Data Format used to train/test KD network

```
from envisat_utils import Envisat, Camera, Projection

#Load Data
thiscam = Camera()
Envisat = ENVISAT_DS(camera_obj=thiscam, train_path= train_dir, val_path= val_dir,
    test_path= test_dir, **kwargs)

# Get pose for an image id
x_id = 420
gt = Envisat.gt_data['train'][x_id]
pose = gt['pose']

#Project body frame axes for an image
uv_pt= Projection.project_to_image(pose['q'],pose['r'], Envisat.Camera)

#Check wireframe projection simultaneously with ground-truth
check_projection_gt('train', img_name='frame1024.jpg', idx=None)

#Visualize ground truth bounding box and keypoints
Envisat.visualize('val', idx=rand_id, bbox=True, kpts=True )

# Get a list of of all image files and check if a filename exists
ex_file = 'frame4783.jpg'
is_file = True if ex_file in Envisat.filenames else False

# Write JSON data files for all image sets
Envisat.write2JSON(image_set='all',n_max =9000, filename='val_bbox')
...
```

Listing C.4: Easy accessibility for Envisat dataset through utilities file

Envisat Data API Example

The raw data files for *ENVISAT-1* contains relative cartesian position, relative attitude Euler angles, illumination angle and keypoint coordinates for each image in a `.txt` format. The data can be directly converted into required format by manipulating with a flow similar to the one for *SPEED* (Fig. 6.8(b)). However, an extensive python utility file is created for the current and future Envisat datasets, that allows increased accessibility and efficiency in development and prototyping using simple commands and function calls. Some functionalities include retrieval of a specific image, retrieval pose information, pose conversion, projection and visualization, data import and export, result analysis and visualization among many others. Code Listing C.4 shows a sample code with several high level actions being easily performed with calls to the classes and methods in the utilities file (`envisat_utils.py`).

Further, an additional utility file is created that transforms the Envisat data files into a COCO annotations file format, mentioned above.

Object Detection Configuration (`.config`) File

```
# SSDLite with Mobilenet v3 large feature extractor config for Envisat+IC
# 3.22M parameters, 1.02B FLOPs
# TPU-compatible

model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 1
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    encode_background_as_zeros: true
    anchor_generator {
      ssd_anchor_generator {
        num_layers: 6
        min_scale: 0.2
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        aspect_ratios: 3.0
        aspect_ratios: 0.3333
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    box_predictor {
      convolutional_box_predictor {
        min_depth: 0
        max_depth: 0
        num_layers_before_predictor: 0
        use_dropout: false
        dropout_keep_probability: 0.8
        kernel_size: 3
        use_depthwise: true
        box_code_size: 4
        apply_sigmoid_to_scores: false
        class_prediction_bias_init: -4.6
        conv_hyperparams {
          activation: RELU_6,
          regularizer {
            l2_regularizer {
              weight: 0.00004
            }
          }
```

```
        initializer {
          random_normal_initializer {
            stddev: 0.03
            mean: 0.0
          }
        }
        batch_norm {
          train: true,
          scale: true,
          center: true,
          decay: 0.97,
          epsilon: 0.001,
        }
      }
    }
  }
  feature_extractor {
    type: 'ssd_mobilenet_v3_large'
    min_depth: 16
    depth_multiplier: 1.0
    use_depthwise: true
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.03
          mean: 0.0
        }
      }
      batch_norm {
        train: true,
        scale: true,
        center: true,
        decay: 0.97,
        epsilon: 0.001,
      }
    }
    override_base_feature_extractor_hyperparams: true
  }
  loss {
    classification_loss {
      weighted_sigmoid_focal {
        alpha: 0.75,
        gamma: 2.0
      }
    }
    localization_loss {
      weighted_smooth_l1 {
        delta: 1.0
      }
    }
    classification_weight: 1.0
    localization_weight: 1.0
  }
  normalize_loss_by_num_matches: true
  normalize_loc_loss_by_codesize: true
  post_processing {
    batch_non_max_suppression {
      score_threshold: 1e-8
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 100
      use_static_shapes: true
    }
    score_converter: SIGMOID
  }
```

```
  }
}

train_config: {
  batch_size: 64
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 32
  num_steps: 100000
  fine_tune_checkpoint: "Detection_model/ssd_mobilenet_v3_large_coco_2020_01_14/model.
    ckpt"
  from_detection_checkpoint: true
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
  data_augmentation_options {
    random_rotation90 {
    }
  }
  data_augmentation_options {
    random_rgb_to_gray {
      probability: 0.4
    }
  }

  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: 0.001
          total_steps: 200000
          warmup_learning_rate: 0.0005
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "../../../Envisat/data/Envisat+IC/tfrecords/train.record"
  }
  label_map_path: "Detection_model/Envisat/label_map.pbtxt"
}

eval_config: {
  num_examples: 5000
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "../../../Envisat/data/Envisat+IC/tfrecords/val.record"
  }
  label_map_path: "Detection_model/Envisat/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Keypoint Detection Configuration (`.yaml`) File

```yaml
# HRNet-W32 configuration: ImageNet pre-training- 256x256
## Used with PyTorch/HRNet/ and HRNet_Envisat_1.ipynb notebook
# When using with Envisat/HRNet_regression, change dir structures accordingly

AUTO_RESUME: true
CUDNN:
  BENCHMARK: true
  DETERMINISTIC: false
  ENABLED: true
DATA_DIR: ''
GPUS: (0,)
OUTPUT_DIR: '../../Envisat/HRNet_regression/output/Envisat+IC'
LOG_DIR: '../../Envisat/HRNet_regression/Envisat+IC'
WORKERS: 24
PRINT_FREQ: 100

DATASET:
  COLOR_RGB: false
  DATASET: 'envisat_coco_envisat'
  DATA_FORMAT: jpg
  FLIP: true
  NUM_JOINTS_HALF_BODY: 12
  PROB_HALF_BODY: 0.3
  ROOT: '../../Envisat/data/Envisat+IC/'
  ROT_FACTOR: 45
  SCALE_FACTOR: 0.35
  TEST_SET: 'val'
  TRAIN_SET: 'train'
  IMG_PREFIX: 'frame'
MODEL:
  INIT_WEIGHTS: true
  NAME: pose_hrnet
  NUM_JOINTS: 12
  PRETRAINED: 'pretrained/hrnet_w32-36af842e.pth'
  TARGET_TYPE: gaussian
  IMAGE_SIZE:
  - 256
  - 256
  HEATMAP_SIZE:
  - 64
  - 64
  SIGMA: 2
  EXTRA:
    PRETRAINED_LAYERS:
    - 'conv1'
    - 'bn1'
    - 'conv2'
    - 'bn2'
    - 'layer1'
    - 'transition1'
    - 'stage2'
    - 'transition2'
    - 'stage3'
    - 'transition3'
    - 'stage4'
    FINAL_CONV_KERNEL: 1
    STAGE2:
      NUM_MODULES: 1
      NUM_BRANCHES: 2
      BLOCK: BASIC
      NUM_BLOCKS:
      - 4
      - 4
      NUM_CHANNELS:
      - 32
      - 64
      FUSE_METHOD: SUM
    STAGE3:
      NUM_MODULES: 4
```

```
      NUM_BRANCHES: 3
      BLOCK: BASIC
      NUM_BLOCKS:
      - 4
      - 4
      - 4
      NUM_CHANNELS:
      - 32
      - 64
      - 128
      FUSE_METHOD: SUM
    STAGE4:
      NUM_MODULES: 3
      NUM_BRANCHES: 4
      BLOCK: BASIC
      NUM_BLOCKS:
      - 4
      - 4
      - 4
      - 4
      NUM_CHANNELS:
      - 32
      - 64
      - 128
      - 256
      FUSE_METHOD: SUM
LOSS:
  USE_TARGET_WEIGHT: false
TRAIN:
  BATCH_SIZE_PER_GPU: 32
  SHUFFLE: true
  BEGIN_EPOCH: 0
  END_EPOCH: 200
  OPTIMIZER: adam
  LR: 0.001
  LR_FACTOR: 0.1
  LR_STEP:
  - 100
  - 150
  WD: 0.0001
  GAMMA1: 0.99
  GAMMA2: 0.0
  MOMENTUM: 0.9
  NESTEROV: false
TEST:
  BATCH_SIZE_PER_GPU: 32
  COCO_BBOX_FILE: '../../Envisat/data/Envisat+IC/val.json'
  BBOX_THRE: 1.0
  IMAGE_THRE: 0.0
  IN_VIS_THRE: 0.2
  MODEL_FILE: ''
  NMS_THRE: 0.9
  OKS_THRE: 0.9
  USE_GT_BBOX: true
  FLIP_TEST: false
  POST_PROCESS: true
  SHIFT_HEATMAP: false
DEBUG:
  DEBUG: true
  SAVE_BATCH_IMAGES_GT: false
  SAVE_BATCH_IMAGES_PRED: false
  SAVE_HEATMAPS_GT: false
  SAVE_HEATMAPS_PRED: true
  SAVE_HEATMAPS_TEST_ALL: true

```

D

Image Corruption Models: Specifications

The details of corruptions augmented in the *Envisat* and the respective magnitudes are presented here. For details on implementation and algorithms, the reader is encouraged to review the software repository [1].

- **Gaussian Noise**: Gaussian noise is added by perturbing the normalized pixel intensities in the image with pixel intensity drawn from a normal distribution. The severity is specified by the standard deviation of such a normal distribution. The **Level-1** standard deviation is set to **0.08** and the **Level-2** standard deviation is set to **0.12**.

```
    x = np.array(img) / 255.
    corrup_img =  np.clip(x + np.random.normal(size=x.shape, scale=severity), 0,
    1) * 255
```

- **Shot Noise**: Shot noise is added with the pixel intensity drawn from a Poisson distribution. The severity is used to specify the variance of the distribution, from which the the noise for that pixel would be generated. The **Level-1** severity factor is set to **60** and the **Level-2** severity factor is set to **25**.

```
    x = np.array(img) / 255.
    corrup_img =  np.clip(np.random.poisson(x*severity) / severity, 0, 1)*255
```

- **Impulse Noise**: Impulse noise is added by replacing a proportion of pixels in the image with hot pixels (normalized pixel intensity = 1). The severity specified the proportion of the total image pixels to be replaced. The **Level-1** severity factor is set to **0.015** and the **Level-2** severity factor is set to **0.06**

```
    x = np.array(img) / 255.
    corrup_img=np.clip(sk.util.random_noise( x, mode='s&p', amount=c),0, 1)*255
```

- **Speckle Noise**: Speckle noise is added by perturbing the image pixels by an amount obtained by multiplying pixel intensities with a random value drawn from a Gaussian distribution. The severity specifies the standard deviation of the Gaussian distribution. The **Level-1** severity factor is set to **0.15** and the **Level-2** severity factor is set to **0.2**

```
    x = np.array(img) / 255.
    corrup_img=(x + x * np.random.normal(size=x.shape, scale=severity), 0, 1)*255
```

[1] https://github.com/kuldeepbrd1/robustness

- **Gaussian Blur**: Gaussian Blur is added by using a convolution operation and modifying the pixel intensity value using a Gaussian kernel. The severity specifies the standard deviation of the Gaussian kernel, which is convolved. The **Level-1** severity is set to **1** and the **Level-2** severity is set to **2**

```
x = np.array(img) / 255.
corrup_img = skimage.filters.gaussian(x, sigma=severity, multichannel=True),
0, 1) * 255
```

- **Defocus Blur**: Defocus Blur is added by constructing a kernel that represents an aliasing disk with Gaussian blur. The image is then convolved with the disk kernel. The severity specifies a 2-tuple with the radius of the aliasing disk and the standard deviation of the gaussian blur used for the disk. The **Level-1** severity factor is set to **(3,0.1)** and the **Level-2** severity factor is set to **(4,0.5)**

```
x = np.array(img) / 255.
kernel = AugmentationHelpers.disk(radius=c[0], alias_blur=c[1])

channels = []
for d in range(3):
    channels.append(cv2.filter2D(x[:, :, d], -1, kernel))
channels = np.array(channels).transpose((1, 2, 0))  # 3x224x224 -> 224x224x3

corrup_img = np.clip(channels, 0, 1) * 255
```

- **Motion Blur**: Motion Blur is added using the Wand Library [2]. The severity specifies a 2-tuple with the radius of the aliasing disk and the standard deviation of the blur used with the disk. The **Level-1** severity factor is set to **(7,3)** and the **Level-2** severity factor is set to **(15,5)**
- **Zoom Blur**: Motion Blur is added by adding two overlays of the zooming on an image array. First overlay zooms in by a large zoom factor and the second overlay zooms out by a small zoom factor. The zoomed image pixel values after each zooming operation are determined by spline interpolation. The severity specifies a 2-tuple with the zoom factor for the two operations. The **Level-1** severity factor is set to **(1.11,0.01)** and the **Level-2** severity factor is set to **(1.16,0.02)**
- **Spatter**: Spatter is added by simulating liquid droplets. The description of magnitude is not trivial, and the reader is referred to the original code here [3].

[2] https://github.com/emcconville/wand
[3] https://github.com/hendrycks/robustness/blob/master/ImageNet-C/create_c/make_imagenet_c.py

E

Additional Results

SPEED: Keypoint Detection

(a) Input Size

HRNet, the selected keypoint detection network architecture is large in size (parameters) and demands significantly higher computation (FLOPs), unlike the object detection network. In this case, the input image size has significant effect on the computational speed or the FLOPs requirement per forward pass, which is essential to assess. For this, the standard HRNet-W32 network is trained on 9600 training images in *SPEED* and evaluated on the other 2400 images. This is done for two input image sizes: 256 px × 192 px and 256 px × 256. These are default sizes commonly used with HRNets [122]. While the number of parameters stay constant, the FLOPs requirement increases by about 2 Bn FLOPs for the larger size. Both networks are initialized with ImageNet pre-trained weights and only affine (**DA-0**) augmentations are included during the training. Table E.1 shows the evaluation results for the two input sizes.

Identifier	Network	Input size (px)	FLOPs (Bn)	Error (px)	Input-normalized error (px)	Input-normalized median error (px)
SPD-KP-1	HRNet-W32	256 × 192	**7.12**	14.46 ± 36.95	4.13 ± 9.27	1.34
SPD-KP-2	HRNet-W32	256 × 256	9.49	**9.18 ± 27.44**	**2.63 ± 6.53**	**1.06**

Table E.1: Performance comparison on *SPEED* for input size variation for the standard HRNet-W32 network

The SPD-KP-2 trained model outperforms SPD-KP-1 significantly in the input-normalized mean and median errors. Consequently, the mean and standard deviation of the localization error in the original image size is much higher for SPD-KP-1 due to scaling effect. While the FLOPs requirement for SPD-KP-1 is lower, the accuracy loss is considered significant. Notice that the higher input normalized standard deviation is a direct consequence of the complexity of SPEED images, which make detection non trivial due to illumination, earth textures and occlusion. it is emphasized that the mean and standard deviation of the error can be heavily skewed by misdetected keypoints or outliers. Therefore, the input-normalized median error provides additional insight into the keypoint detection accuracy that is affected less by the occasional outliers. For the more accurate SPD-OD-2 network model, the input normalized median error is close to 1 px per keypoint, which is considered optimal for keypoint detection.

For the desired accuracy performance in benchmark comparisons with the state-of-the-art results, 256 px × 256 px is selected as the input size for further experiments,

(b) Pre-training

The aspect of transfer learning in CNNs and potential benefits of pre-training were discussed in Section 5.2. Since pretrained weights for the standard HRNet-W32 network are available [1], this experiment is designed to assess the advantage of using pre-trained networks, if any.

[1] https://github.com/leoxiaobin/deep-high-resolution-net.pytorch

The HRNet-W32 network is trained on the training images from scratch with random weights intialization and not using the pre-trained weights. The resulting model, SPD-KP-4, is compared with the SPD-KP-2 model which differs only with respect to pre-training. Table E.2 shows the evaluation results of this experiment.

Identifier	Network	Pretrained	Error (px)	Input-normalized error (px)	Input-normalized median error (px)
SPD-KP-2	HRNet-W32	Y	**9.18 ± 27.44**	**2.63 ± 6.53**	1.06
SPD-KP-4	HRNet-W32	N	12.69 ± 38.44	4.76 ± 16.03	1.06

Table E.2: Performance comparison on *SPEED* for pre-training on standard HRNet-W32 network

The SPD-KP-4 network model trained from scratch shows a decrease in accuracy when compared with the SPD-KP-2 network pretrained on ImageNet. The input normalized median error are similar while the mean and standard deviation of the localization are higher for SPD-KP-4, implying higher number of outliers with worse detections than the pretrained model. In addition, a miscellaneous aspect is that the the pre-trained networks converge more quickly as the weights of the kernels in the initial convolution layers need to be varied by a lower magnitude during gradient descent, as opposed to random weights in a non pre-trained network.

Envisat+TR dataset

Object Detection Evaluation

Texture randomized images were introduced in the *Envisat-1* dataset, with an expectation to improve the texture robustness of the trained network, by learning representations that rely less on specific local textures. The resulting *Envisat+TR* dataset is used to assess the potential of texture randomization in improving CNN performance. The training images contain a 50% mix of clean images from *Envisat-1* and texture randomized images from an internally available dataset [2]. The trained models are evaluated on the 9600 images of the validation set from the *Envisat-1* dataset, but only considering the 12 points (main body and SAR antenna) as noted in Section E. Affine and pixel-level augmentations (**DA-0** +*DA-1*) are included in the training. The objective of the experiment was to first verify accuracy on clean images and subsequently the robustness on the corrupted images. Table E.3 shows the evaluation results on the clean images of the *Envisat-1* validation set. The accuracy expected from texture randomization is not witnessed with a network model training with *Envisat+TR* dataset, as it shows poor accuracy on the validation set comprising of clean images. Subsequent testing on corrupted images was not conducted as the trained model lacks the pre-requisite accuracy on clean images. The poor performance observed for the trained model is expected due to the size of the texture randomized images. The size of the texture randomized images (see Section E) is 256 px × 256 px i.e. half the size of the clean images in the dataset. During training, these images are upsampled to fit the input size of the object detection network and the network operates on a significantly lower resolution image. The network weights may therefore be unable to generalize well over clean as well as low-resolution texture randomized images adequately, resulting in underfitting.

Model	Network	Mean IoU	Median IoU	IoU>0.75
ENVTR-OD-4	*SSDLite-MNetV3-large*	0.809	0.830	77.5 %

Table E.3: Performance evaluation of the selected object detection networks on *Envisat-1* dataset

[2] Courtesy of TU Delft SpE/Airbus

F

Relative Orbital Elements

The translational state is defined in Radial, Tangential Normal (RTN) frame as $\delta x_{RTN} = [\delta x, \delta y, \delta z]$. Then, the quasi-nonsingular Relative Orbital Elements as defined in [32] as:

$$\delta\alpha = \begin{pmatrix} \delta a \\ \delta\lambda \\ \delta\mathbf{e} \\ \delta\mathbf{i} \end{pmatrix} = \begin{pmatrix} \delta a \\ \delta\lambda \\ \delta e_x \\ \delta e_y \\ \delta i_x \\ \delta i_y \end{pmatrix} = \begin{pmatrix} (a_t - a_s)/a_s \\ (M_t - M_s) + (\omega_t - \omega_s) + (\Omega_t - \Omega_s)\cos(i_s) \\ e_t\cos(\omega_t) - e_s\cos(\omega_s) \\ e_t\sin(\omega_t) - e_s\sin(\omega_s) \\ i_t - i_s \\ (\Omega_t - \Omega_s)\sin(i_s) \end{pmatrix} \tag{F.1}$$

where a, e, i Ω, ω and M are the keplerian elements, and the subscript t denotes the target spacecraft and subscript s denotes the servicer. The transformation matrix that maps the ROE's to RTN cartesian state is obtained from [31] and is recalled here for completeness:

$$\delta x_{RTN} = \mathbf{M} a \delta\alpha \tag{F.2}$$

$$\begin{pmatrix} \delta x \\ \delta y \\ \delta z \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\cos u & -\sin u & 0 & 0 \\ 0 & 1 & 2\sin u & -2\cos u & 0 & 0 \\ 0 & 0 & 0 & 0 & \sin u & -\cos u \end{pmatrix} \begin{pmatrix} a\delta a \\ a\delta\lambda \\ a\delta e_x \\ a\delta e_y \\ a\delta i_x \\ a\delta i_y \end{pmatrix} \tag{F.3}$$

Filter Dynamics

The dynamics model propagates the state vector as folowing:

$$\boldsymbol{x}_{ROE}(k+1) = \boldsymbol{\Phi}(k)\boldsymbol{x}(k) \tag{F.4}$$

where $\boldsymbol{x}_{ROE}$ is the state at epoch k, which is related to the state at the next epoch k+1 by the State Transition Matrix (STM), $\boldsymbol{\Phi}(k)$.

The STM will model the relative motion of the servicer and the target in an arbitrarily eccentric orbit perturbed by the J_2 effect and the differential drag. Since it is known that the atmospheric density can vary widely due to solar activity and other phenomena the differential drag model will be a density-model-free one. The relative motion of two satellites in a circular LEO orbit will be dominated by these two effects. In order to incorporate this model, however, the state vector needs to be augmented with three new variables that describe the mean time variations of some in-plane elements: $a\delta\dot{a}, a\delta\dot{e}_x, a\delta\dot{e}_y$ as mentioned in [190]. The dynamics model is then described as:

$$\begin{pmatrix} a\delta\boldsymbol{\alpha} \\ a\delta\dot{a} \\ a\delta\dot{e}_x \\ a\delta\dot{e}_y \end{pmatrix}(k+1) = \begin{bmatrix} \Phi_{HCW}(k)+\Phi_{J2}(k) & \tilde{\Phi}_{d-drag}(k) \\ O_{3x6} & I_{3x3} \end{bmatrix} \begin{pmatrix} a\delta\boldsymbol{\alpha} \\ a\delta\dot{a} \\ a\delta\dot{e}_x \\ a\delta\dot{e}_y \end{pmatrix}(k) \tag{F.5}$$

with:

$$\Phi_{HCW}(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -\frac{3}{2}ndt & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{F.6}$$

where n is the mean angular motion of the spacecraft and dt is the time step. Furthermore:

$$\Phi_{J2}(k) = ndt \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{21}{4}\gamma H(\eta+1) & 0 & 0 & 0 & \frac{-3}{2}\gamma\sin(2i)(3\eta+4) & 0 \\ 0 & 0 & 0 & -\varphi' & 0 & 0 \\ 0 & 0 & +\varphi' & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{21}{4}\gamma\sin(2i) & 0 & 0 & 0 & 3\gamma\sin^2(i) & 0 \end{bmatrix} \tag{F.7}$$

where $\gamma = \frac{R_E^2}{a^2\eta^4}$, $\eta = \sqrt{1-e^2}$ and $\varphi' = \frac{3}{2}\gamma(5\cos^2(i)-1)$ and $H = 3\cos^2(i)-1$

and

$$\tilde{\Phi}_{d-drag}(k) = \Phi_{d-drag}(k) + \begin{bmatrix} 0 & O_{1x2} \\ \frac{-21}{8}\gamma H(\eta+1)ndt^2 & O_{1x2} \\ O_{3x1} & O_{3x2} \\ \frac{21}{8}\gamma\sin(2i)ndt^2 & O_{1x2} \end{bmatrix} \tag{F.8}$$

with:

$$\Phi_{d-drag}(k) = \begin{bmatrix} dt & \frac{2}{n}\sin(u-u_0) & \frac{2}{n}(1-\cos(u-u_0)) \\ \frac{-3}{4}ndt^2 & -\frac{3}{n}(1-\cos(u-u_0)) & -3dt+\frac{3}{n}\sin(u-u_0) \\ \frac{\sin(u-u_0)}{n} & dt+\frac{\cos(u-u_0)\sin(u-u_0)}{n} & \frac{\sin^2(u-u_0)}{n} \\ \frac{(1-\cos(u-u_0))}{n} & \frac{\sin^2(u-u_0)}{n} & dt-\frac{\cos(u-u_0)\sin(u-u_0)}{n} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{F.9}$$

and where u is the mean argument of latitude $u = M+\omega$.

This model is valid as long as the assumption of the parameters incorporated $(a\delta\dot{a}, a\delta\dot{e}_x, a\delta\dot{e}_y)$ remain constant in time is acceptable [**?**].


TU Delft