Autonomous Navigation around Asteroids using Convolutional Neural Networks

Master Thesis

L.F.J. van der Heijden

Technische Universiteit Delft

TU Delft

Autonomous Navigation around Asteroids using Convolutional Neural Networks

Master Thesis

by

L.F.J. van der Heijden

in partial fulfilment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology
to be defended publicly on Thursday February 17, 2022 at 9:30 AM.

Student number:	4367286	
Project duration:	1 June 2021 - 17 February 2022	
Master Track:	Space Flight	
Master Profile:	Space Exploration	
Thesis committee:	Dr. ir. Erwin Mooij	Supervisor, TU Delft
	Dr. ir. Svenja Woicke	Supervisor, DLR
	Dr. Daphne Stam	TU Delft
	Dr. ir. Erik-Jan van Kampen	TU Delft

Cover image taken from the OSIRIS-REx mission available at
`https://www.asteroidmission.org/galleries/spacecraft-imagery/`

TU Delft

Preface

This work marks the end of an exciting few months as well as the end of my studies at TU Delft. It has been a pleasure to work on such a cutting-edge research field, combining machine learning with space exploration, which in my opinion would allow mankind to venture even further into space, finding answers to the most pressing questions. However, I was not alone in this endeavor and would therefore like to thank the people that helped and supported me along the way.

First of all, I would like to thank my supervisor, dr. ir. Erwin Mooij, for his enthusiasm, support, and interest in my work. I enjoyed our weekly discussions and your sharp insights, resulting in extra motivation to strive for the best possible result and getting the most out of this research. Next to this, I also want to thank dr. ir. Svenja Woicke of DLR for bi-weekly joining our discussions and providing valuable insights.

Finally, I would like to thank my family and friends for their unwavering support, offering the occasional sympathetic ear and distraction when necessary.

L.F.J. van der Heijden
Delft, January 2022

Abstract

Missions to small bodies are increasingly gaining interest as they might hold the secrets to our solar system's origin while some are also posing a threat to life on Earth. The small size and irregular shape result in complex dynamics complicating the close-proximity operations. The majority of past and current missions relied on humans-in-the-loop for a variety of purposes, however, communication with Earth is costly and due to the long round-trip time communication delays of up to 20 minutes can exist, excluding any required computation time on Earth. Moreover, for landing on these small bodies stringent accuracy requirements exist, whereas during these phases no communication with Earth is possible due to the involved time-scales. Therefore, close-proximity operations would benefit from accurate autonomous vision-based navigation. Currently used approaches either rely on detecting pre-defined landmarks on the target, detecting features and matching them to a database, or tracking craters or unknown features across images (relative navigation). However, these methods rely heavily on *a-priori* information, suffer from computationally intensive matching steps, or depend on the accuracy of the initial state estimate. Machine learning has seen great success in applications to terrestrial problems, and as of 2019 research into using Convolutional Neural Network (CNN)-based methods for pose estimation of uncooperative spacecraft has taken of through the Spacecraft Pose Estimation Challenge (SPEC). Pose estimation refers to estimating the distance and orientation of the target w.r.t. the camera, using a single 2D image. This work investigated the usage of a novel CNN-based pipeline that can be used to autonomously navigate accurately around asteroids.

A top-down CNN-based feature detector is developed, consisting of an object and keypoint detection network in sequence, which detects n pre-defined keypoints within the 2D image, which were designated on the target's 3D model using the 3D SIFT algorithm. These 2D detections alongside their 2D-3D correspondences are sent to an Efficient Perspective-n-Point (EPnP) solver that solves the Perspective-n-Points (PnP) problem. The CNN-based feature detector replaces traditional hand-engineered Image Processing (IP) algorithms, as it is more robust to illumination conditions and image noise. Furthermore, the use of a CNN facilitates an offline feature selection step and as such avoid the cumbersome and computationally intensive 2D-3D matching step of the detected 2D feature to their location on the 3D model, plaguing traditional approaches. The feature detector outputs heatmaps predictions around the pre-selected keypoints, allowing for the extraction of statistical information (covariance matrix) regarding the uncertainty of the detection. This enables the seamless integration of this developed pipeline within a navigation architecture. This pose estimation pipeline can be used to navigate around the asteroid up until it covers the full field of view of the camera, and it can be used to (re)-initialize the navigation filter for a relative navigation approach. The networks have been selected based on their applicability to embedded devices and this resulted in the use of the SSD-MobileNetV2-FPN-Lite for the object detection network and the Lightweight Pose Network (LPN) model for the keypoint detection network. This lightweight CNN-based pipeline has a fraction of the parameters and Floating Point Operations (FLOPs) compared to state-of-the-art deep-learning networks and pipelines.

These networks have been trained and evaluated on synthetic datasets created in this work, consisting of 32,352 images with a variety of poses for a distance of 4.5 km to 9 km from the asteroid, for different illumination conditions, asteroid orientations, and image corruptions that emulate real sensor artefacts. The pipeline could achieve a mean and median *line-of-sight* distance estimate of around 42 m and 30 m, respectively, at a confidence level of 90% for the large relative range, while satisfying the accuracy requirement of a maximum 10% knowledge error for 99.979% of the cases. The closer to the asteroid the more accurate the performance with a median accuracy of around 22 m from a distance of 4.5 km. Furthermore, the pipeline has been proven to be robust against illumination conditions, occlusions, textures, and image corruptions mimicking effects of real sensors and the space environment. Demonstrating the efficacy of this CNN-based approach for autonomous navigation around asteroids.

The results achieved in this work show that a CNN-based approach can achieve accurate results, while adhering to lightweight principles required for incorporation on a spacecraft processor. However, future research should focus on validating the performance of the synthetically trained CNN on-ground, using lab generated real images. Bridging the domain gap and achieving robustness of the synthetically trained CNN to real images is crucial in creating deep learning systems that can be deployed in safety-critical applications and fly on an actual spacecraft.

Contents

List of Abbreviations

ACT	Advanced Concepts Team
AFC	Asteroid Framing Camera
AIDA	Asteroid Impact & Deflection Assessment
AKAZE	Accelerated KAZE
API	Application Programming Interface
ARRM	Asteroid Redirect Robotic Mission
au	astronomical unit
BGD	Batch Gradient Descent
CCD	Charge-coupled device
CEPP*n*P	Covariant Efficient Procrustus Perspective-n-Point
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CoM	Center of Mass
CPN	Cascaded Pyramid Network
CPU	Central Processing Unit
CRP	Classical Rodrigues Parameter
CV	Computer Vision
DANN	Domain-Adversarial Neural Network
DART	Double Asteroid Redirection Test
DCM	Direction Cosine Matrix
DEM	Digital Elevation Map
DL	Deep Learning
DoF	Degree-of-freedom
DoG	Difference-of-Gaussian
DSN	Deep Space Network
DTM	Digital Terrain Map
EP*n*P	Efficient Perspective-n-Point
ESA	European Space Agency
ESTEC	European Space Research and Technology Center
FLOPs	Floating Point Operations
FOV	Field of View
FPN	Feature Pyramid Network
GC	Global Context
GNC	Guidance, Navigation, and Control
GPU	Graphical Processing Unit
HIL	Hardware-in-the-Loop
HRNet	High Resolution Network
IMU	Inertial Measurement Unit
IoU	Intersection over Union
IP	Image Processing
IQR	Interquartile range
JAXA	Japan Aerospace Exploration Agency
KD	Keypoint Detection
KLT	Kanade-Lucas-Tomasi
LHM	Lu-Hager-Mjolsness
lidar	Light detection and ranging
LPN	Lightweight Pose Network
MAD	Mean Average Deviation

MBGD	Mini-Batch Gradient Descent
ML	Machine Learning
MLP*n*P	Maximum Likelihood Perspective-n-Point
MRP	Modified Rodrigues Parameter
MSE	Mean Squared Error
Mult-Adds	Multiply-accumulate operation
NASA	National Aeronautics and Space Administration
nD	n Dimensional
NEAR	Near Earth Asteroid Rendezvous
NED	North-East-Down
NEO	Near-Earth object
NFT	Natural Feature Tracking
NMS	Non-Maximum Suppression
NN	Neural Network
NST	Neural Style Transfer
OD	Object Detection
OpenCV	Open Source Computer Vision Library
ORGL	Orbital Robotics & GNC Laboratory
OSIRIS-REx	Origins, Spectral Interpretation, Resource Identification, Security, Regolith Explorer
P*n*P	Perspective-n-Points
PANGU	Planet and Asteroid Natural Scene Generation Utility
PCL	Point Cloud Library
POSIT	Pose from Orthography and Scaling with Iteration
RANSAC	Random Sample Consensus
ReLu	Rectified Linear Unit
ResNet	Residual Network
RGB	Red, Green, and Blue
RMS	Root Mean Square
RMSE	Root Mean Square Error
RoI	Region of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SIFT	Scale-Invariant Feature Transform
SLAB	Space Rendezvous Laboratory
SLAM	Simultaneous Localization and Mapping
SNR	Signal-to-Noise
SPC	Stereophotoclinometry
SPEC	Spacecraft Pose Estimation Challenge
SPEED	Spacecraft Pose Estimation Dataset
SPN	Spacecraft Pose Network
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machine
TMT	Target Marker Tracking
TRN	Terrain Relative Navigation
TRON	Testbed for Rendezvous and Optical Navigation
YOLO	You Only Look Once

List of Symbols

Greek symbols

α	Bearing angle around Y_B	rad
α	Learning rate	-
β	Bearing angle around X_B	rad
β	Hyperparameter of the β-*Soft-Argmax* function	-
β_1	Hyperparameter controlling the momentum	-
β_2	Hyperparameter of RMS prop	-
ϵ	Variable to ensure numerical stability	-
λ	Regularization parameter	-
Φ	Principal rotation angle (Euler angle)	rad
ψ	Yaw angle	rad
σ	Standard deviation	-
θ	Pitch angle	rad
φ	Roll angle	rad

Latin symbols

a	Semi-major axis	m
$\mathbf{A}^{[l]}$	Activation matrix for layer l for all m training examples	-
$\mathbf{b}^{[l]}$	Bias matrix for layer l for all m training examples	-
$\mathbf{C}$	Unit-axis transformation matrix	-
$\mathbf{C}_{B,A}$	Transformation matrix from frame A to frame B	-
c_x	x location of the camera's principal point in the pixel reference frame	px
c_y	y location of the camera's principal point in the pixel reference frame	px
du	Horizontal pixel length	m/px
dv	Vertical pixel length	m/px
e	Eccentricity	-
E_{px}	Pixel error	px
E_t	Translational error	m
f	Focal length	m
f	Size of the convolutional filter	px
f_x	Scaled focal length in the x-direction	px
f_y	Scaled focal length in the y-direction	px
G	3D SIFT Gauss kernel function	-
h	Height of image	px
$\mathbf{H}_k$	Heatmap	-
i	Inclination	rad

J	Cost function	-
$\mathbf{K}$	Intrinsic camera parameters matrix	-
L	Loss function	-
l	l^{th} layer of the neural network	-
m	Number of training examples	-
N	Number	-
n	Order of refinement of the icosphere	-
n	Input size of the image $n \times n$	px
$n^{[l]}$	Number of neurons in the l^{th} layer of the neural network	-
n_c	Number of channels of the feature map	-
N_u	Number of horizontal pixels in the image	px
N_v	Number of vertical pixels in the image	px
P	3D point cloud	-
P	Probability	-
p	Padding value	-
$\mathbf{P}$	Pose matrix	-
p_c	Class probability	-
$\mathbf{q}$	Quaternion	-
R	Radius of the icosphere	-
$\mathbf{r}$	Position vector	m
s	Stride of the convolutional filter	px
s_x	Scaling factor: The size of one pixel in the x-direction	1/m
s_y	Scaling factor: The size of one pixel in the y-direction	1/m
$\mathbf{t}$	Translation vector	m
t_h	Log space representation of the offset of the height of the bounding box	-
t_w	Log space representation of the offset of the width of the bounding box	-
t_x	Representation of the offset of the x-coordinates of the center of the bounding box	-
t_y	Representation of the offset of the y-coordinates of the center of the bounding box	-
u_k	2D x-coordinate of the k-th keypoint in the pixel reference frame	px
v_k	2D y-coordinate of the k-th keypoint in the pixel reference frame	px
w	Width of image	px
$\mathbf{W}^{[l]}$	Weight matrix for layer l for all m training examples	-
$\mathbf{X}$	Input feature vector	-
X_i	X-axis in the i^{th} reference frame	-
x_i	Cartesian x coordinate in the ith reference frame	m
y	Ground-truth output	-
$\hat{y}$	Predicted output	-
Y_i	Y-axis in the i^{th} reference frame	-
y_i	Cartesian y coordinate in the ith reference frame	m
$Z^{[l]}$	Matrix containing the outputs of the neurons for layer l for all m training examples	-
Z_i	Z-axis in the i^{th} reference frame	-
z_i	Cartesian z coordinate in the ith reference frame	m

I

Introduction and background

1 Introduction

This chapter introduces the topic of this research project, where Section 1.1 identifies the framework around which this project revolves. Section 1.2 discusses the developments that have led to this research after which the research questions are formulated in Section 1.3. This chapter concludes with an overview of the structure of the report in Section 1.4.

1.1. Problem and relevance

The Origins, Spectral Interpretation, Resource Identification, Security, Regolith Explorer (OSIRIS-REx) mission is the most recent example of the burgeoning scientific interest towards small-bodies (asteroids). This increasing interest is caused not only because their composition might hold the secrets to our solar system's origins, but also from a planetary defense point of view as they may impact Earth and thereby threaten life on the planet. As of recently, it became clear that the asteroid Bennu has a 1-in-2700 change of impacting Earth in the late 22^{nd} century. Myriad of past, current, and future missions have been designed to find answers and solutions to these questions and problems. The recently launched Double Asteroid Redirection Test (DART) mission and the currently designed HERA mission to the binary asteroid system Didymos serve as the latest examples (Volpe et al., 2020).

However, the difficulty for asteroid exploration missions lies in the complex dynamics and that the exact properties of the asteroid are unknown before arriving. Moreover, small asteroids lack an accurate ephemeris and with current limitations in NASA's Deep Space Network (DSN) communication and tracking system, there is a 2-5 km uncertainty in the distance estimate between a spacecraft and a target (Schwartz et al., 2018). Currently, missions to small bodies rely on humans-in-the-loop for a variety of purposes, ranging from state estimation to the creation of a detailed shape model and determining the landmarks and landing sites on the target.

However, communication with Earth is costly, especially when images need to be send back to Earth. Moreover, due to the long round-trip time there can be communication delays up to 20 minutes, excluding required computation time on Earth. This results in painstakingly slow and careful progress throughout the mission phases and therefore *autonomous navigation* would be very beneficial, as it allows faster reactivity, which can maximize scientific return. Furthermore, it allows longer science operations, more accurate instrument pointing, and flying closer to the surface of the asteroid (Gil-Fernandez et al., 2019). Moreover, autonomous navigation can provide state estimates at a higher frequency than ground-based estimates.

Autonomous navigation is also a crucial technology for landing on asteroids and satisfying the stringent requirements existing due to the small size of the asteroids. During this phase, communication with Earth is impossible based on the landing-phase timescale with regards to the communication delay. The Guidance, Navigation, and Control (GNC) system traditionally fused the input from several sensors, such as radars, altimeters, and Inertial Measurement Units (IMUs). However, over the last decades vision-based navigation measurements have been found to improve the performance of autonomous navigation considerably (Flandin et al., 2010; Gil-Fernandez and Ortega-Hernando, 2018). The usage of some form of autonomous vision-based navigation during these close-proximity and touch-down operations has been used by past asteroid missions, such as OSIRIS-REx and Hayabusa2 (Lorenz et al., 2017; Ogawa et al., 2020)

Autonomous navigation would not only allow the spacecraft to adapt more rapidly to changing mission

requirements or objectives, but also increases the amount of scientific work it could perform in a given period of time. Vision-based navigation systems solely relying on a monocular camera are also becoming a more attractive alternative to active sensors, such as lidar or stereo cameras, as they have lower mass, hardware complexity, cost, and power consumption (Opromolla et al., 2017; Pasqualetto Cassinis et al., 2019; Sharma et al., 2018). The HERA mission proposed by the European Space Agency (ESA) will be the first to use deep-space CubeSats. Developing monocular vision-based autonomous navigation would be greatly beneficial for these deep-space CubeSats, which have stringent requirements on the available resources. These techniques would allow mankind to venture further into the solar system, visiting unknown environments, and safely and more efficiently perform close-proximity operations, while keeping the mission costs relatively low.

1.2. Research trigger

Advancements within machine learning in recent years have seen a widespread adaptation of these techniques to a variety of problems. These learning-based algorithms prove a good technique for automation as once they are trained they can be deployed and act autonomously. Izzo et al. (2019a) identified deep learning as the key technology for current and future research in field of spacecraft GNC. Machine learning techniques have already been applied to preliminary spacecraft design, mission operations, designing optimal interplanetary trajectories, and for guidance and control software of landers (Cheng et al., 2019; Furfaro et al., 2018; Izzo et al., 2019b).

Two major factors have limited the application of machine learning to spaceflight: 1) the unavailability of publicly available large-scale datasets required for training these deep learning models (Izzo et al., 2019a) and 2) the limited processing power available on-board spacecraft. As a result of the former, these models are therefore trained on synthetic imagery and their robustness to realistic space images is difficult to predict. The high risk associated with a wrong decision, detection, or calculation has limited their adaptation to safety-critical space applications.

However, there is a growing interest in applying machine learning to space-related problems and as of 2019, the research into applying Convolutional Neural Network (CNN)-based architectures to uncooperative spacecraft pose estimation has taken off through the Spacecraft Pose Estimation Challenge (SPEC), which was organized by the Space Rendezvous Laboratory (SLAB) at Stanford University in cooperation with ESA (Sharma and D'Amico, 2019). CNN-based approaches have several advantages, and have been proven to be more robust against illumination conditions and image noise compared to traditional Image Processing (IP) algorithms. This is crucial for the usage in a vision-based navigation system (Pasqualetto Cassinis et al., 2021a). Furthermore, as of 2021, the most recently announced SPEC 2021 tries to tackle the problem of validating the performance and demonstrate the CNN-based architecture's ability to fly in space through the creation of a dataset called Spacecraft Pose Estimation Dataset (SPEED)+, consisting of real camera images from a mock-up model using a robotic testbed (Park et al., 2021).

The radiation-filled space environment necessitates the use of radiation-hardened (rad-hard) computers for space missions. However, the state-of-the-art rad-hard computers typically have inferior performance compared to state-of-the-art terrestrial computers, i.e., the processing power is limited. This has made the application of deep learning to space-borne problems challenging, as these networks typically have millions of parameters requiring substantial memory size, and running these networks requires significant amount of processing power.

However, Howard et al. (2017) was the first to develop a lightweight network called MobileNet that could be used on mobile and embedded devices without compromising too much on accuracy. Currently, the trend within machine learning is to optimize the speed/accuracy trade-off and more research into the development of these lightweight networks is being performed. Furthermore, Manning et al. (2018) demonstrated that such lightweight networks can be used on the space-grade embedded system of a CubeSat and achieve good performance.

The aforementioned shows that machine learning architectures are a promising option for autonomous navigation of spacecraft and at the moment of writing, to the author's knowledge, no research regarding the usage of machine learning for autonomous navigation around asteroids exists.

1.3. Research questions

The purpose of this study is to investigate if and how a learning-based algorithm can achieve accurate relative navigation around an asteroid. This section discusses the research (sub-)questions that were formulated for this work. The research question that is to be answered during this work is:

How can accurate relative navigation be achieved in close-proximity operations around asteroids with limited a-priori information using learning-based autonomous algorithms?

The definition of relative navigation intended here is that the spacecraft navigates relative to the asteroid and thereby estimates the state of the spacecraft w.r.t. the asteroid. Furthermore, close-proximity operations refers to the fact that the spacecraft can observe the asteroid and the scientific mission phases commence. The *a-priori* knowledge refers to not relying on detailed information regarding the dynamics or other properties of the asteroid, such as the gravity field.

This research question was further divided into three subquestions:

1. **How can a representative, realistic dataset of synthetic images of the asteroid suitable for training and evaluating deep learning networks be created?**

   Due to the unavailability of large-scale datasets of asteroid images suitable for deep learning purposes, the networks need to be trained and evaluated on synthetic image datasets that have to be created within this work. Therefore, it should be researched how this can be performed in the best way.

2. **How can the pose estimation pipeline be made robust against a variety of factors, such as illumination conditions and image corruptions, representative of the real space environment?**

   This research subquestion is two-fold, where firstly the CNN-based pipeline needs to be designed such that it is able to work for a variety of scenarios representative of an actual space mission. This means that the pipeline should work for a variety of camera viewpoints, distances, and illumination conditions. Secondly, as the networks will be trained solely on synthetic images the CNNs need to demonstrate robustness to the *domain gap*. This *domain gap* refers to the gap in performance of the synthetically trained CNN to real images and is caused by real images having a different statistical distribution compared to synthetic images. Moreover, real images can contain noise and other image corruptions that are not or cannot be accurately modeled in the synthetic images. Bridging this domain gap and achieving robustness of the CNN trained on synthetic images is crucial in creating deep learning systems that can be deployed in safety-critical applications.

3. **How can the pose estimation pipeline be improved compared to comparable pipelines currently researched for satellites?**

   Current studies applying machine learning to the problem of pose estimation of uncooperative spacecraft rely on networks that have tens of millions of parameters and consequently require a lot of memory and computational effort, rendering them unsuitable for use on current spacecraft processors. Therefore, it would be interesting to explore if accurate navigation could be achieved, while also optimizing the pipeline for usage on embedded devices.

1.4. Report structure

This section discusses the structure of the report and is intended to give an overview of the discussed topics. The heritage of vision-based navigation around asteroids and the applicable research within machine learning is discussed in Chapter 2. Moreover, Chapter 2 discusses the mission and system requirements and outlines the scope of the research, diving into the assumptions and limitations. The theoretical basis to understand the subsequent algorithm development is discussed in Chapters 3 through 5. Chapter 3 discusses the reference frames and their conversions, coordinate systems, and kinematic descriptors used throughout this work. The pose estimation framework, outlining the camera model used for the dataset generation and the PnP problem, is discussed in Chapter 4. Furthermore, within Chapter 5 the basic building blocks of machine learning are discussed, which serve as a solid basis for subsequent discussions in later chapters. This chapter is less relevant for readers who are very familiar with CNNs and machine learning in general.

An overview of the different parts and software used in the pose estimation architecture is given in Chapter 6, where also the selection of the pose solver is discussed. The creation of the synthetic datasets used in this work is described in detail in Chapter 7. The selection, implementation, configuration, training, and evaluation procedures of the respective networks are discussed in Chapters 8 and 9, respectively. The verification of the different parts of the pipeline is addressed in Chapter 10. Chapter 11 discusses the results achieved by the CNN-based pose estimation pipeline on the datasets created in this work. Finally, Chapter 12 presents the answers to the research questions and the main conclusions of this work alongside recommendations for future research.

2

Heritage

This chapter defines the scope of the research performed in this work. In Section 2.1, the heritage of vision-based navigation methods with a focus on asteroid mission is discussed, as well as the applicable research into machine learning. Based on the heritage, the mission and system requirements for the algorithm developed in this work are devised in Section 2.2. The section is concluded with the scope of the research in Section 2.3, discussing the framework and the assumptions used in this work. This is intended to give the reader a good overview of the purpose of this work and what can be expected for the analysis performed in subsequent chapters.

2.1. Heritage

The development of autonomous vision-based navigation systems is a topical research subject not only for space-borne problems, but also terrestrial applications. As touched upon in Section 1.1, autonomous navigation is desired, because of the long round-trip time for communications, allowing for faster reactivity and longer science operations among other advantages. An overview of different vision-based navigation techniques are discussed in this section that have been applied to space missions after which their machine learning counterparts are discussed.

An asteroid mission can be categorized through its different mission phases. An initial global characterization phase (preliminary survey) determines the physical and dynamical characteristics of the asteroid, refining the mass, radius, and inertia using measurements as well as refining the rotational rate and spinning axis. Furthermore, a 3D shape model of the asteroid is created using a technique called Stereophotoclinometry (SPC), which uses high resolution images of the target. Subsequent phases focus on completing a wide range of scientific objectives, such as identifying sample sites for sample collection. A brief overview of the different approaches that have been designed and used to navigate around asteroids is given.

Navigation around asteroids

The Asteroid Redirect Robotic Mission (ARRM) was a proposed mission by the National Aeronautics and Space Administration (NASA) to a Near-Earth object (NEO) that was discontinued in 2017. The spacecraft was intended to navigate through the use of *landmark navigation*, which was first used extensively by the Dawn spacecraft to the asteroids Vesta and Ceres (Mastrodemos et al., 2011). *Landmark navigation* uses designated landmarks on the surface of the target for navigation, where landmarks do not have to be specific surface characteristics, such as craters or boulders, and as such, also does not depend on the presence of these features.

Mastrodemos et al. (2011) developed the *L-maps* landmark navigation approach for the Dawn spacecraft. The landmarks are defined as the center of a small Digital Terrain Map (DTM) of parts of the body, which are created using SPC, making it an intrinsic part of the characterization and mapping phase as aforementioned. The spacecraft is able to navigate by detecting and identifying the landmarks that are within the Field of View (FOV) of the camera and then use the known location of the corresponding landmarks within a navigation filter to estimate the position of the spacecraft w.r.t. the asteroid (Mastrodemos et al., 2011). The accuracy of this method depends on the accuracy of the created DTM and shape model, which in turn depend on the resolution with which the camera can model the surface. A similar approach was used by Razgus et al. (2017),

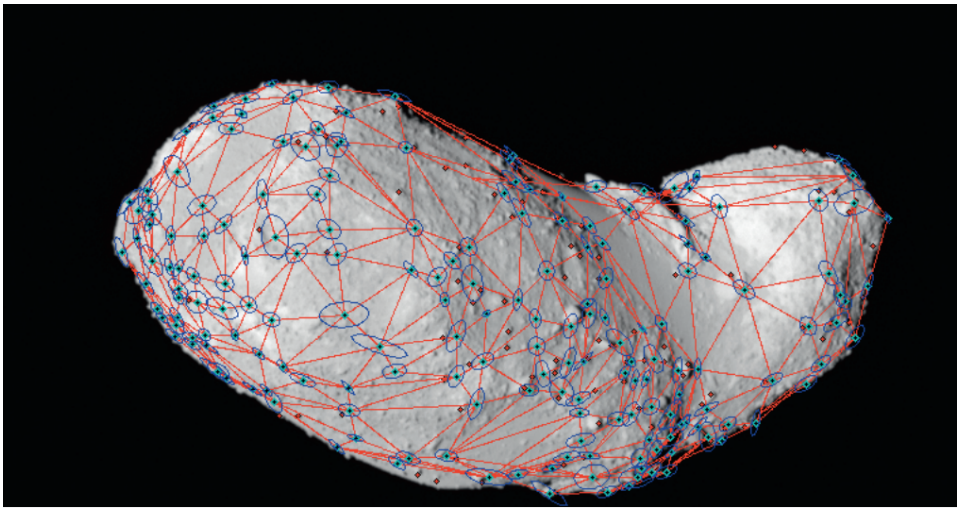


Figure 2.1: Converged solution of the landmark identification and localization approach, where the blue ellipses represent the 6σ matching radius for each landmark and the red diamonds represent the Harris corners detected in the real image (Rowell et al., 2015)

however, the landmarks were designated on the surface of the asteroid Itokawa by randomly placing points on the triangular faces of the polyhedron 3D model.

Rowell et al. (2015) developed another landmark based approach that uses an Image Processing (IP) algorithm to detect Harris corners on the asteroid surface as landmarks. A database of surface landmarks had to be created online by applying the Harris corner detector to all the (simulated) images. The corresponding 3D locations of those 2D Harris corners were determined using the available 3D shape model of the asteroid, which is created during the initial characterization phase. This algorithm was developed as part of an industrial assessment study for the Marco Polo-R mission intended to land a spacecraft on a small NEO and retrieve a sample. The designed vision-based navigation system consisted of two parts, namely feature detection and landmark recognition. The feature detection algorithm detected the Harris corners within the current navigation image, whereas the landmark recognition relied on an initial state estimate of the spacecraft to project the landmark positions from the database on the current navigation image. The projected landmark positions were compared with the extracted Harris corners within the current navigation image and positive matches are based on the proximity to one another. The pose is refined by aligning the identified landmarks with the detected Harris corners, where the pose refers to the distance and orientation of the camera w.r.t. the target. An example of a converged solution of the landmark navigation approach is shown in Figure 2.1, i.e., where the detected Harris corners have been aligned with the identified landmarks. This navigation system was used to initialize the navigation filter before the descent and landing phase, for which relative navigation methods (feature tracking) were used. This method relies on the asteroid to be entirely within the FOV. The downside of this approach is its reliance on an initial state estimate and the fact that enough landmarks need to be within the field of view of the camera to allow for accurate estimation.

The Hayabusa mission to Itokawa, designed by the Japan Aerospace Exploration Agency (JAXA), was launched in 2003 and the spacecraft was intended to navigate autonomously. A lidar was used to estimate the distance, and the direction to the asteroid was found using a monocular camera. During the final descent for sample collection, the system would navigate using Target Marker Tracking (TMT), where the target markers are spherical retroflective artificial landmarks. However, part of the navigation system failed resulting in the inability to point the lidar, and therefore a ground-based navigation system was developed, which relied on humans-in-the-loop. An image of the asteroid was sent back every 10 minutes with an one-way communication delay of 20 minutes, resulting in the inability to act fast (Hashimoto et al., 2010). This same concept of using artificial landmarks for the navigation was applied to the Hayabusa2 mission. These target markers are tracked on the unknown natural terrain under unknown illumination conditions through the use of a robust IP algorithm. This circumvents the reliance on natural terrain features and their respective difficulties in feature detection (Ogawa et al., 2020).

The HERA mission designed by ESA and scheduled to launch in 2024 is intended to investigate the effects of the DART mission. The HERA mission will be the first mission to operate fully autonomously around

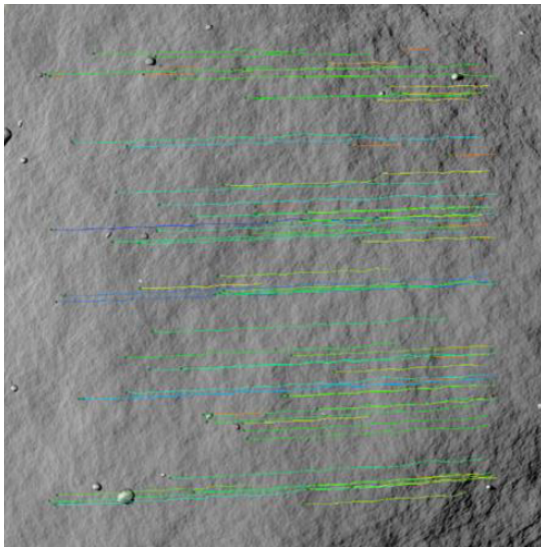


Figure 2.2: Extracted features that have been tracked in subsequent images, which is illustrated by the feature tracks indicating the movement of the features (Pellacani et al., 2019)

the asteroid, as it will fuse the data from several sensors and a monocular camera (Asteroid Framing Camera (AFC)) to simultaneously create a model of the asteroid and navigate relative to that model (Volpe et al., 2020), requiring the asteroid to be fully within the field of view of the camera during that phase.

The spacecraft uses two different techniques for state estimation based on the distance w.r.t. asteroid. Up until the asteroid fully covers the field of view of the camera, centroiding techniques are used for state estimation, whereas from closer distances *unknown feature tracking* will be used. *Unknown feature tracking* refers to autonomously detecting features and tracking them across images using the Kanade-Lucas-Tomasi (KLT) IP algorithm (Pellacani et al., 2019). These features do not represent terrain features, such as craters or boulders, but are simply pixels that stand out from their surroundings. These extracted features receive a feature descriptor (unique tag) that allows the tracking of the features through successive images. This allows the calculation of the displacement of the feature and as such the displacement of the spacecraft within that time period. Figure 2.2 demonstrates these feature tracks indicating the movement of the features.

Similar techniques have been researched for navigating around the Moon by Woicke (2019) and Magalhães Oliveira (2018). This method relies on the accuracy of the initial state estimate before the start of the feature-tracking, as it can only determine the relative displacement. A shortcoming of this approach is that the error is gradually accumulated, as the navigation filter integrates noisy sensor measurements. A solution to this would be to periodically re-initialize the relative navigation with a new measurement of the instantaneous position and orientation of the camera, i.e., pose initialization, using a method similar to the aforementioned landmark approach by Rowell et al. (2015). Furthermore, another limitation of the approach is that when the relative velocity is too slow, the features have not moved relative to the previous frame due to the velocity of the spacecraft, but merely due to the rotation of the asteroid. This can cause the relative tracking method to fail. The tracking of unknown features between frames is less challenging compared to recognizing known landmarks within navigation images, as for the former the viewing conditions do not change a lot between frames. Whereas, for landmark recognition algorithms, the viewing and illumination conditions, and the features' scale and orientation may change significantly between a first and later detection.

There are also some vision-based navigation techniques that rely on physical features, such as craters. This was researched for navigation around the Moon by Maass et al. (2020), but was also used to navigate around Eros in the Near Earth Asteroid Rendezvous (NEAR) mission (Cheng et al., 2002). The approach can differ between detecting a crater and matching this to an existing database or by simply tracking the crater across successive images similar to the *unknown feature tracking* approach. The downside of these approaches is that they rely on the presence of these features on the asteroid and that such a database of craters exists. Moreover, a computationally intensive matching step has to be performed.

The recent OSIRIS-REx mission, launched in 2019 and is expected to return back to Earth with a sample of the asteroid Bennu in 2023, also relied on physical features for navigation. Natural terrain features, such as boulders or craters unique to the surroundings and distinctly recognizable under the expected illumination

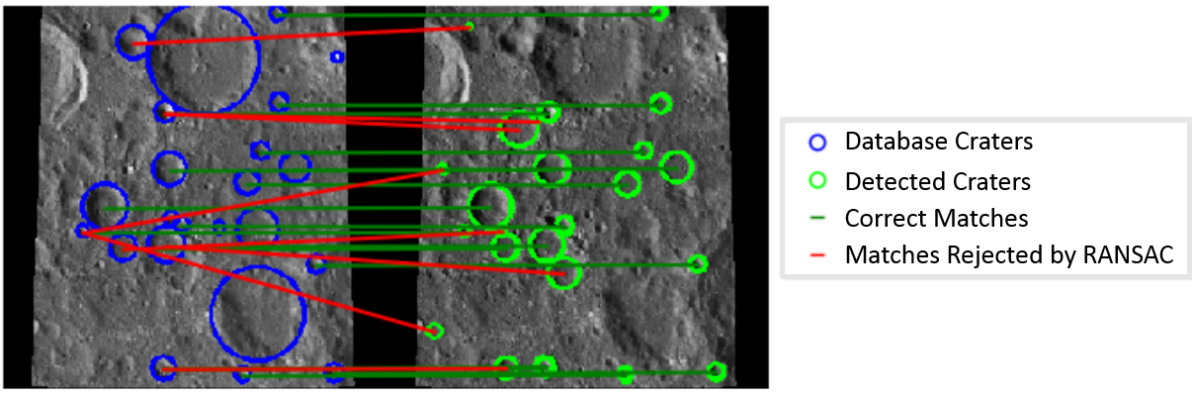


Figure 2.3: Demonstrating the crater matching process between the craters detected by the CNN-based system and the craters present in the database (Downes et al., 2020a)

conditions, were identified by experts on Earth (Lorenz et al., 2017). These manually selected features were stored in a feature-catalog. The spacecraft navigated using Natural Feature Tracking (NFT), which relied on initial estimates of the spacecraft pose, asteroid attitude, and the predicted location of the Sun to generate the expected appearance for the feature based on the Digital Elevation Map (DEM). This expected appearance is then matched to what the spacecraft actually captures through normalized cross correlation. The performance of this navigation method relies on accuracy and resolution of the DTMs.

Machine learning

There is a growing interest into applying machine learning methods to space-related problems due to its successes in solving terrestrial problems. The heritage of machine learning is discussed using approaches that somewhat align with the navigation approaches mentioned previously.

The *crater detection and matching* approach has been tackled using machine learning models by Downes et al. (2020a), Downes et al. (2020b), and Doppenberg (2021), which were applied to the Moon, for which an extensive crater database exists. They used CNNs to detect craters on the lunar surface and match them to known craters present in a database, from which the position of the spacecraft can be determined. This method, however, relies on an initial state estimate to reduce the search-space for the matching step. This crater detection and matching process is shown in Figure 2.3, where it can be seen that some database craters have not been detected and matched by the system. However, for asteroid missions this is not practical, as the surface of small asteroids might not be covered with craters nor does such a database exist. This approach is therefore not considered further.

The research into *unknown feature tracking* using machine learning is in its infancy state with a detailed overview of the different methods discussed in the survey of Chen et al. (2020). There are two major approaches that can be discerned, firstly the learning-based Simultaneous Localization and Mapping (SLAM) approaches that mimic the proposed method for the HERA mission by Volpe et al. (2020), i.e., simultaneously creating the 3D model and navigating w.r.t. that model. Secondly, methods, such as SuperPoint created by DeTone et al. (2018), that try to mimic hand-engineered feature extractors and trackers, such as Scale-Invariant Feature Transform (SIFT) and Accelerated KAZE (AKAZE). However, as stated by Chen et al. (2020), the existing models are not sophisticated enough to solve the respective problems, and the problem of localization and mapping is very complex. Furthermore, the amount of research into this field is limited and to the authors knowledge, at the moment of writing, no examples in literature exist related to space exploration. The applications are currently restricted to certain scenarios, such as self-driving cars, and adapting these networks to space-related problems is not trivial, especially within the framework of a MSc thesis. Therefore, the learning-based unknown feature tracking approach is not considered further in this work and the interested reader is referred to the work by Chen et al. (2020).

The *landmark based navigation* approaches discussed previously relied on the detection of *pre-defined* landmarks for which the 3D location was known and the 2D-3D correspondence could be established. Subsequently this was used to solve for the pose of the spacecraft w.r.t. the target. On a similar abstraction level, this problem within machine learning is referred to as keypoint detection with the goal to estimate the pose of the target. The pose refers to the distance and orientation of the camera w.r.t. the target. This is a challenging

computer vision problem and the majority of research revolves around human pose estimation, detecting joints, such as ankles, knees, and hips of person instances within images and subsequently estimate the pose of that person using these detected keypoints.

The seminal work by Sharma (2019) proposed a CNN-based *end-to-end* architecture to tackle pose estimation of uncooperative spacecraft. Furthermore, through the SPEC organized by Stanford University in cooperation with ESA, research into applying CNN-based architectures to uncooperative spacecraft pose estimation has increased, with the most recently announced SPEC 2021 as the newest stepping stone. Publicly available datasets have been created for these challenges and are referred to as SPEED and SPEED+, respectively (Park et al., 2021; Sharma and D'Amico, 2019).

The research can be divided into two major approaches: 1) *end-to-end* architectures (Alimo et al., 2020; Sharma and D'Amico, 2019; Shi et al., 2018), and 2) *feature-based* architectures (Barad, 2020; Chen et al., 2019; Park et al., 2019; Pasqualetto Cassinis et al., 2020).

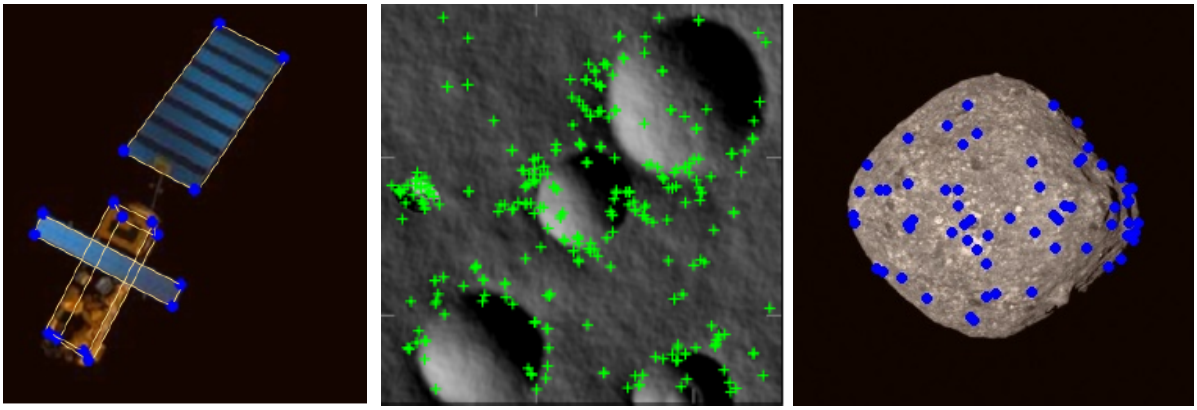
The former referring to approaches in which a CNN replaces the entire pipeline and directly outputs the pose from an input image, thereby learning the highly non-linear mapping function. Whereas, the latter refers to replacing a hand-engineered IP algorithm with a CNN-based feature detector that extracts n predefined features from the 2D image and these keypoints and their 2D-3D correspondence are then sent to a pose solver, which solves the PnP problem. The disadvantage of *end-to-end* architectures is that the learning problem is more complex and therefore more data is required to train the network. The advantage of decomposing the pose estimation problem into sub-tasks is that the complex problem is reduced to two simpler problems, with a lot of data available for each of those tasks. Furthermore, it was demonstrated by Shalev-Shwartz et al. (2017) that this decomposition results in better performance. This notion is reinforced by the fact that the *end-to-end* architectures have been outperformed by the *feature-based* architectures on the SPEC (Kisantal et al., 2020), with a four times smaller average position error and seven times smaller average orientation error.

The accuracy of the feature-based pipeline is determined by the selection of the feature detector. The majority of hand-engineered IP algorithms rely on the image gradient to detect rich textured features on highly visible parts of the target. These detected features are image-specific and this therefore does not allow for an offline feature selection step (D'Amico et al., 2014), thereby requiring an online image-to-model mapping step as was the case for the hand-engineered landmark navigation approach of Rowell et al. (2015). This computationally intensive matching step required for hand-engineered computer vision techniques, such as SIFT, makes them unsuitable for in-orbit applications (Sharma and D'Amico, 2016). One major advantage of using a CNN-based feature detector compared to hand-engineered IP algorithms is that the keypoints can be selected offline and as such their 2D-3D correspondence is known, thereby avoiding the cumbersome and computationally intensive matching of the detected 2D features to their location on the 3D model using methods such as Random Sample Consensus (RANSAC) (Park et al., 2019).

Furthermore, the keypoint-based CNNs output predicted detections for all the designated keypoints even when they are occluded or not directly visible (e.g., on the back of the target) (Zhao et al., 2018). This is because the network learns the inherent spatial relationship between the different keypoints. This is a major advantage compared to traditional methods, such as *landmark navigation*, which rely on the number of landmarks in view to make an accurate prediction. A CNN-based feature detector is also proven to be more robust against adverse illumination conditions and image noise compared to hand-engineered IP algorithms, which is crucial for the usage in a vision-based navigation systems for space-borne problems (Pasqualetto Cassinis et al., 2021a). Another advantage of using a CNN-based feature detection approach is that it does not rely on an initial pose estimate and can be used for pose initialization, i.e., lost-in-space scenarios.

As aforementioned, the research into keypoint detection for uncooperative spacecraft is heavily based on the problem of human pose estimation. The keypoints can have semantic meaning, referring to corners of the spacecraft or the joints of humans, or they can simply represent pixels that stand out with respect to their environment, which are referred to as interest points. Semantic keypoints can be manually selected and designated through algorithms on the available 3D (wireframe) model (Barad, 2020; Chen et al., 2019), whereas interest points can be designated on a 3D model using techniques, such as 3D SIFT (Zhao et al., 2018). The difference between semantic and interest points is illustrated in Figure 2.4.

The feature-based method can be subdivided further in a *top-down* or *bottom-up* approach, where the top-down approach refers to firstly detecting the different objects within the image after which the keypoints are detected on each object. The bottom-up approach refers to firstly detecting all the keypoints on the different objects of interest within an image after which the detections are associated with each respective object. The top-down approach is more robust against blurry or occluded images and it has shown increased perfor-



(a) Semantic keypoints designated on the spacecraft's 3D model projected to 2D (Barad, 2020)

(b) Interest points detected on an image of the Lunar surface (Woicke, 2019)

(c) Interest points designated on the asteroid's 3D model projected to 2D

Figure 2.4: Demonstrating the difference between semantic keypoints (a) and interest points (b,c)

mance over the bottom-up approach (Jin et al., 2017). The top-down approach is currently the state-of-the-art method used within keypoint detection for human pose estimation, terrestrial objects, and uncooperative spacecraft (Kisantal et al., 2020; Pasqualetto Cassinis et al., 2021a; Zhao et al., 2018).

The practical implementation of the top-down approach is the usage of an Object Detection (OD) network in front of the Keypoint Detection (KD) network. This OD network detects the object within the image and regresses the bounding box coordinates encompassing the object. This bounding box is used to crop the Region of Interest (RoI) of the original image. This RoI image is then resized to match the input size of the keypoint detection network. This top-down approach makes the CNN pipeline more robust to the scale of the object within the image, allowing for more accurate keypoint detections by the KD network (Pasqualetto Cassinis et al., 2021a).

Within keypoint detection two different approaches can be identified, the first approach directly outputs the feature coordinates from the image and the second approach outputs a confidence map/heatmap that provides pixel-wise pseudo-likelihoods that the keypoint location is found in that pixel. The direct coordinate approach was pioneered by Toshev and Szegedy (2014) and was used to locate the joints of a body. Subsequent research by Tompson et al. (2014) was focused on outputting heat maps for the keypoint location. The 2D pixel coordinates of the keypoint location correspond to the heatmap's peak intensity, where the shape and the intensity characterize the confidence of the detection (Pavlakos et al., 2017). Research by Tompson et al. (2014) and Szegedy et al. (2016) showed that the heatmap approach resulted in better performance compared to the direct regression approach. The heatmap approach is currently the state-of-the-art within keypoint detection architectures (Barad, 2020; Chen et al., 2019; Pasqualetto Cassinis et al., 2020; Sun et al., 2019; Zhang et al., 2019). Another advantage of using a heatmap approach is that it increases the explainability of the network, as the heatmap shows where the network focuses on for predictions in an image (Zhou et al., 2016).

Furthermore, Pasqualetto Cassinis et al. (2020) demonstrated that statistical information can be derived from the heatmap that allows to quantify the uncertainty of the detections through the form of a covariance matrix. This statistical information can then be used by a pose solver that is able to take this into account, such as the Covariant Efficient Procrustus Perspective-n-Point (CEPPnP) solver. Moreover, this can be used in a navigation filter, thereby illustrating another advantage of using this heatmap based approach as opposed to the direct regression approach.

These CNN-based feature extractors have to be trained on synthetic images due to the unavailability of datasets consisting of real images of the target that are suitable for training these models. Pasqualetto Cassinis et al. (2020) and Pasqualetto Cassinis et al. (2021b) used the Cinema4D rendering software to generate a synthetic image deep-learning dataset of the Envisat spacecraft. Black et al. (2021) used Blender to create a synthetic dataset of images of the Gygnus spacecraft for deep-learning purposes, demonstrating its suitability for deep-learning dataset generation. Moreover, Blender has been used in other works revolving space applications, such as feature-based landing on the Moon (Magalhães Oliveira, 2018), feature-based navigation around asteroids (Volpe et al., 2020), and uncooperative spacecraft pose estimation (Harvard et al., 2020).

However, neural networks cannot generalize well to out-of-distribution examples (real images) with respect to the training set (synthetic images), resulting in decreased performance of a synthetically trained CNN to real space images. This problem is referred to as the *domain* gap and arises from real images having different statistical distribution compared to synthetic images as well as containing noise or other image corruptions that are not or cannot be properly modeled in synthetic data. This poses challenges for the application of CNN-based methods to safety critical applications, such as spaceflight. Recent research by Park et al. (2021) and Pasqualetto Cassinis et al. (2021b) focuses on bridging this domain gap and validate the performance of these synthetically trained CNNs on lab-generated real images representative of the space environment. The usage of heatmaps instead of keypoint regression was also found by Park et al. (2021) to be more robust to the domain gap.

Conclusion

The methods designed and/or used to navigate around asteroids either navigated through the use of landmark navigation, detecting features and matching them to a database, or use hand-engineered IP algorithms to track unknown features or physical features (craters) across images (relative navigation). The aforementioned methods suffer from computationally intensive matching steps, rely on *a-priori* information, or depend on the accuracy of the initial state estimate.

Machine learning is a promising method that can resolve the issues plaguing traditional approaches. Based on the heritage, the most promising approach to develop a learning-based navigation algorithm is a top-down CNN-based feature detector, which replaces a hand-engineered IP algorithm and which will be researched in this work. This consists of an object and keypoint detection network in sequence, which detects n pre-defined keypoints within a 2D image that have been designated on the target's 3D model. The detections and their known 2D-3D correspondence are then used to estimate the distance to the asteroid by solving the PnP problem. Furthermore, the CNN-based feature detector outputs heatmap predictions around the pre-defined keypoints locations, which allows the quantification of the detection certainty through the use of heatmap-derived covariance matrices. This allows for an easy incorporation of the developed feature detection pipeline within a navigation architecture.

The usage of such a CNN-based pipeline allows the offline designation of keypoints on the 3D model, circumventing the use of a computationally intensive 2D-3D matching step, plaguing traditional approaches. Furthermore, these architectures guarantee the predictions of all designated keypoints (even on the back of the target) and are therefore not limited by the amount of landmarks/features within the field of view. Moreover, they have been proven to be more robust against against a variety of factors, such as image noise and illumination conditions, compared to hand-engineered IP algorithms. In addition, this CNN-based pipeline does not depend on an initial pose estimate and can provide an estimate of the instantaneous pose of the target, i.e., pose initialization.

2.2. Mission and system requirements

This section formulates the mission and system requirements based on the heritage discussed in Section 2.1

Mission requirements:

- **MR01**: The navigation system shall be used to navigate around an asteroid
- **MR02**: The navigation system shall be able to provide a distance estimate up until the asteroid covers the full field of view of the camera

These mission requirements are straightforward and summarize the main goal of the developed algorithm. **MR02** is discussed is in more detail in Section 2.3.

System requirements:

- **SR01**: The pose estimation pipeline shall rely on a monocular model-based feature detection approach
- **SR02**: The pose estimation pipeline shall use an object detection network in front of the CNN-based feature detector
- **SR03**: The feature detector shall output a confidence map/heatmap that provides pixel-wise pseudo-likelihoods that the keypoint location is found in that pixel

- **SR04**: The pose estimation pipeline shall have an optimal architecture focused on less memory usage and FLOPs that allows for future embedding in a spacecraft processing unit
- **SR05**: The pose estimation pipeline shall be trained on synthetic images
- **SR06**: The pose estimation pipeline shall have a relative line-of-sight distance to the center of mass of the target with a knowledge error lower than 10% of real distance with 99.73% probability at 90% confidence level.

The first three system requirements **SR01** through **SR03** are based on the heritage discussed in Section 2.1, demonstrating the highest performance was achieved using such an architecture. Furthermore, **SR03** allows the system to be easily incorporated into several different navigation architectures, as the navigation filter can use the statistical information derived from the detected features heatmap. The fourth system requirement **SR04** is mostly focused on designing an architecture that could be used on an actual spacecraft. As discussed in Section 1.1, state-of-the-art neural networks generally have millions of parameters and require substantial amount of computational power to produce outputs, whereas spacecraft processing power and memory is limited. Therefore, care was taken to select networks based on an accuracy/speed trade-off, allowing such a network to be incorporated on spacecraft hardware in the future, enhancing the contributions made by this work. The fifth requirement **SR05** is incorporated, as no deep-learning image datasets of asteroids exist and therefore the system should not rely on the availability of real images, but instead be trained on synthetic imagery rendered in a 3D rendering software. The last requirement **SR06** sets an accuracy requirement for the performance of the designed algorithm. This requirement is taken from a statement of work by ESA[1] related to this work, as to the best of the author's knowledge no comparable literature regarding the application of machine learning to navigate around asteroids exists.

2.3. Scope of the research

This section discusses the scope of the research that is performed in this work, specifying the target asteroid and the purpose of the designed algorithm alongside the limitations and assumptions.

Target asteroid

More than 700,000 asteroids have been discovered, however, for only a small fraction of them a 3D shape model exists. The majority of the shape models are created using lightcurve inversion techniques, but they provide only an approximation of the asteroid's shape, and small features, such as ridges or craters, cannot be detected. A handful of asteroids have been observed by radar or visited by spacecraft. Very precise models of these asteroids exists due to high-resolution photography and measurements. The most famous ones are *Eros*, *Itokawa*, *Vesta*, *Bennu*, and *Ryugu*.

The target asteroid chosen for this work is Bennu, as a detailed 3D model is publicly available, as well as a lot of literature surrounding the mission and the asteroid. Bennu is a sub-kilometer asteroid with a mean radius of 245 m, and it orbits the Sun at about the same distance as the Earth (1 astronomical unit (au)) (Lauretta et al., 2019). It has a spinning top shape with a pronounced equatorial bulge, similar to the Didymos asteroid, which is the target of the upcoming HERA mission designed by ESA. The Didymos asteroid has an estimated radius of 390 m, however, no shape model of the Didymos asteroid is publicly available[2]. Therefore, an asteroid that shares similarity with it (Bennu) would allow this work to contribute to the upcoming HERA mission, demonstrating that such a technique could work.

Assumptions and limitations

The scope of the research is discussed to clearly identify what will and what will not be researched in this work.

1. **Known 3D model is assumed**: The model-based approach requires the availability of a 3D model. This is normally created during initial mission phases and is subsequently used to create gravitational models and to navigate around the asteroid (Lorenz et al., 2017; Mastrodemos et al., 2011; Rowell et al., 2015).

[1] ESA, "Statement of Work: Artificial Intelligence for Terrain Relative Navigation in Unknown Environment (ATENA) - Reference: ESA-TECSAG-SOW-018784", 2020

[2] `https://solarsystem.nasa.gov/asteroids-comets-and-meteors/asteroids/didymos/in-depth/`, Date accessed: 31-01-2022

2. **Navigation up until the asteroid fills the FOV of the camera, no landing is analyzed**: The CNN-based pipeline could be used for two applications: 1) to navigate around the asteroid at distances in which the asteroid does not fully cover the FOV of the camera, and 2) to (re-)initialize an *unknown feature tracking* approach that can navigate from distances closer to the asteroid. As aforementioned, these feature tracking approaches rely on the accuracy of the initial estimate, as they can only perform relative tracking. By increasing the accuracy of the initial estimate, the overall accuracy of the navigation system can be increased. This accuracy and precision is required for potential landing on asteroids. However, the landing phase is not analyzed in this work. The assumption that the object should be fully within the field of view of the camera is common and was used by Rowell et al. (2015) for landmark navigation around asteroids. Furthermore, this is also common within the limited literature regarding uncooperative spacecraft using CNN-based approaches.

3. **Limited *a-priori* information**: This is inherent to the learning-based algorithm that will be developed in this work. Based on the detected features within an image and their 2D-3D correspondences, an estimate of the pose of the spacecraft w.r.t. target is calculated, i.e., pose initialization. No initial pose estimate is required for the algorithm to work as opposed to conventional approaches discussed in Section 2.1. Furthermore, no information regarding the expected scale of the asteroid within the image or the encountered illumination conditions is required. The network is trained to become robust against these aspects. Furthermore, other aspects, such as information regarding the gravitational field, is also not required.

4. **Distance estimation only**: The purpose of this work is to develop a machine learning algorithm that accurately determines the distance w.r.t. the asteroid as discussed in Section 1.3. The P*n*P problem, which will be elaborated upon in Section 4.2, is solved by pose solvers to output the pose. The pose refers to the distance as well as the orientation of the camera reference frame w.r.t. the target's reference frame. However, this is less relevant for an asteroid mission as other instruments, such as star-trackers, are used to accurately determine the orientation of the spacecraft. Therefore, the main focus of this work will be on the distance estimation. However, if desired, the orientation prediction can also be used and incorporated into a navigation filter, as the availability of more information could allow for better conversion of that filter.

5. **No incorporation into a navigation architecture**: This algorithm will consist of a CNN-based feature extractor pipeline, which can be incorporated into a navigation architecture, this could be a loosely or tightly coupled architecture. However, this work will not develop this navigation architecture due to time-limitations. Nonetheless, the statistical information that can be extracted from the heatmap outputted by the feature detector allows for the fusion of the detections with different measurements of other sensors within a navigation filter.

6. **Synthetic images only:** Due to the unavailability of large-scale datasets of asteroid images suitable for deep-learning purposes, the networks shall be trained and evaluated on synthetic image datasets that will be created in this work.

7. **Motion asteroid around the Sun is not modeled**: A fixed position of the asteroid w.r.t. the Sun is chosen, meaning that the Sun-asteroid vector is fixed. However, different asteroid orientations and camera positions result in a variety of illumination conditions. This does not compromise the validity of the conclusions drawn regarding the performance of the algorithm.

8. **An ideal camera is assumed**: The camera is perfectly calibrated and the intrinsic camera parameters are known. Moreover, no skewness and distortion are modeled, and the sensor is assumed to have square pixels.

9. **Focus is solely on the development of a navigation algorithm**: This work will purely focus on developing a novel vision-based navigation method, therefore ideal control is assumed, i.e., actual attitude is commanded attitude. Furthermore, it is assumed that the camera has no pointing errors.

10. **No variable pointing of the camera**: The camera direction is fixed w.r.t the spacecraft body frame (no variable pointing possible) meaning that if the camera turns, the whole spacecraft turns with it. Therefore, the camera frame can be used to describe the orientation of the spacecraft, removing the need of using a separate spacecraft body-fixed reference frame for the development of the navigation algorithm.

11. **The dynamics of the spacecraft and asteroid are not modeled**: The training of the machine learning algorithm does not rely on any dynamics, nor is it required for the generation of images. The networks are trained by generating images from a variety of poses, asteroid orientations, and illumination conditions. These asteroid orientations can be discretely changed and therefore the motion of the asteroid around its own axis is not modeled.

    Moreover, solving the PnP problem results in a distance and orientation estimate of the camera reference frame w.r.t. the target reference frame. This should then be transformed from the camera frame to another frame, such as the NED reference frame, to describe the motion of the spacecraft around the target. However, as the system will not be incorporated into a navigation architecture in this work, this removes the need to describe the dynamics of the problem of a spacecraft orbiting an asteroid. These reference frames and conversions are therefore not required.

12. **Navigation only in the illuminated portion of the orbit:** The vision-based navigation will use images from the visible spectrum, i.e., no infrared, and will therefore only be used during the (partially) illuminated portion of the orbit around Bennu

II

Theory

3

Reference frames

This chapter discusses the reference frames and kinematic descriptors used throughout this work. Section 3.1 introduces the reference frames and how they are defined. Section 3.2 discusses the kinematic descriptors used to describe the state of the spacecraft, diving into position as well as attitude. The chapter is concluded with the reference frame conversions in Section 3.3.

3.1. Reference frames

The translation and rotation of an object can only be described when a reference with respect to which the object is moving is defined. This is the reason reference frames are needed. A reference frame is defined as *A set of coordinate axes in terms of which position or movement may be specified* [1]. A reference frame is attached to a certain origin and its axes have a certain orientation. The reference frames used throughout this work are right-handed and are either pseudo-inertial or non-inertial. The following reference frames are used:

1. World reference frame (W)
2. Asteroid reference frame (A)
3. Blender camera reference frame (B)
4. Camera reference frame (C)
5. Pixel reference frame (P)
6. Satellite-fixed reference frame (S)

World reference frame - W

The world reference frame is the reference frame that is used in Blender to place the object, camera, and light source. This is a right-handed system where the axes are defined as follows, with the $+Z_W$-axis pointing up and the X and Y axes follow the right-handed triad.

Asteroid reference frame - A

The asteroid reference frame has its origin at the center of mass of the asteroid and the OX_AY_A-plane coincides with the equatorial plane of the asteroid. The 3D model of the asteroid is loaded into Blender with $+Z$ as up and $+Y$ as forward. This ensures that the axes of the asteroid correspond with the world axes at the initial orientation. This definition of the initial orientation, i.e., what is the front, is also used in the keypoint designation process. The axes are defined as follows:

- $+Z_A$-axis is pointing upwards
- $+Y_A$-axis is pointing forward
- $+X_A$-axis completes the right-handed system

Blender camera reference frame - B

The Blender camera reference frame is the camera frame that is used by Blender and it has its origin in the center of projection (Section 4.1). The axes are defined as follows:

[1] `https://www.thefreedictionary.com/Reference+frame+(physics)` Date accessed: 16-12-2020

- $+Z_B$-axis is pointing away from the image plane and is aligned with the *optical axis*
- $+Y_B$-axis is pointing up
- $+X_B$-axis completes the right handed frame

Camera reference frame - C
The camera reference frame has its origin in the center of projection and the axes are defined according to usual convention. The definitions are discussed in Section 4.1.

- $+Z_C$-axis is pointing towards the image plane and is aligned with the *optical axis*
- $+X_C$-axis is pointing towards the east
- $+Y_C$-axis completes the right handed frame

Pixel reference frame - P
The pixel reference frame is a 2D reference frame with its origin in the top left corner of the image plane, this is discussed in more detail in Section 4.1. The pixel reference frame is used to represent the pixel location in the image and the unit is the pixel index, counted row-wise and column-wise. The pixel reference frame is also often referred to as the image reference frame. The axes are defined as follows.

- $+X_P$-axis is parallel to X_C
- $+Y_P$-axis is parallel to Y_C

Furthermore, the following notation is also used, $X_P = u$ and $Y_P = v$

Body-fixed reference frame - S
The body-fixed reference frame is used to specify the orientation of the camera w.r.t. the body axes. As discussed in Section 2.3, there is no need to have a body-fixed reference frame for the development of the algorithm. However, this reference frame is needed to allow for the incorporation of the camera and the derived navigation system within an actual system containing gyroscopes and other sensors. The origin of the body-fixed reference frame is assumed to be at the center of mass of the spacecraft. A symmetric satellite is assumed and the axes are defined following a commonly used convention:

- $+X_S$-axis is in the symmetry plane of the spacecraft and is pointing forward
- $+Z_S$-axis is pointing downwards
- $+Y_S$-axis completes the right handed frame, i.e., pointing to the right and perpendicular to the $X_S Z_S$ symmetry plane.

The camera is assumed to be mounted on the bottom of the spacecraft in the center pointing along the $+Z_S$-axes and the X_C and Y_C axes are also aligned with the X_S and Y_S axes.

3.2. State representation

Kinematics describes the motion of points, objects, and bodies irrespective of the forces that caused it to move. There are two major types of kinematics, *translational kinematics* and *attitude or rotational kinematics*. The translational kinematics describe the 3-Degree-of-freedom (DoF) position and motion of point masses, whereas the attitude/rotational kinematics describe the 3-DoF rotation and motion of bodies. Many types of kinematic descriptors exist. The kinematic descriptors are called *state variables* and together they determine the state of the vehicle or point mass. Numerous state representation methods exists and only the ones that will be used are described in this section.

3.2.1. Coordinate systems

A coordinate system is necessary to represent the location of a point within a certain reference frame. The purpose of the algorithm is to determine the distance of the camera reference frame w.r.t. the target asteroid reference frame. The position will be expressed using Cartesian coordinates, as they are simple and easy to interpret and are useful for numerical simulations.

A vector is represented using three coordinates (x, y, z), which together make up the vector from the origin of the reference system to a point of interest. These scalar values (x, y, z) are multiplied with the unit vectors of the reference frame and added together. This can be represented mathematically using the following equation, where it can be expressed in any arbitrary frame F.

$$\mathbf{r} = \mathbf{r}^F = x\hat{X}_F + y\hat{Y}_F + z\hat{Z}_F \tag{3.1}$$

In vector form this is represented as follows:

$$\mathbf{r} = \mathbf{r}^F = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{3.2}$$

3.2.2. Attitude kinematics

The rotational parameters can be described in various ways, such as Direction Cosine Matrix (DCM), Euler angles, and attitude quaternions. There are several attitude descriptors that only need three variables to describe the rotation, such as Euler angles, Classical Rodrigues Parameter (CRP), and Modified Rodrigues Parameter (MRP). However, when only using three parameters to describe the attitude, singularities can occur and as such Euler angles are never more than 90° away from a singularity. CRPs and MRPs describe the attitude in such a way that in CRPs the singularity happens at ±180° and in MRPs the singularity happens at ±360°. This allows for better tracking of the attitude without having to worry about singularities. However, the main goal of this work is to develop a ML algorithm that can be incorporated in a navigation architecture to provide state estimates. Therefore, no detailed analysis is performed on which parameters are best suited to represent the attitude in a most optimal way. Furthermore, the spacecraft will not be rapidly rotating or tumbling, but will be pointing steadily towards the target. Therefore, there is no explicit need to use these more advanced attitude descriptors. Attitude quaternions are therefore used to describe the orientation of the spacecraft, as they have no singularities and are often used to describe the attitude of spacecraft (Pasqualetto Cassinis et al., 2020; Woicke, 2019).

Direction Cosine Matrix: The DCM is a transformation matrix that is used to transform coordinates from reference frame $\mathbf{F}_A$ to another reference frame $\mathbf{F}_B$. The DCM is part of the 3D rotation group, $\mathbf{C}_{B,A} \in SO(3) := \{\mathbf{C} \in \mathbb{R}^{3\times 3}, \mathbf{C}\mathbf{C}^{\mathbf{T}} = I, \det(\mathbf{C} = 1)\}$. The transformation between the unit vectors of frame $\mathbf{F}_A$ and frame $\mathbf{F}_B$ is given by the dot product, as the dot product between two unit vectors is simply equal to the angle, $\cos\theta$, between the axes. The definition of the DCM matrix is shown below, where the notation for describing the transformation from reference frame $\mathbf{F}_A$ to another reference frame $\mathbf{F}_B$ is defined as $\mathbf{C}_{B,A}$ (Wie, 1998).

$$\mathbf{C}_{B,A} = \begin{bmatrix} b_1 \cdot a_1 & b_1 \cdot a_2 & b_1 \cdot a_3 \\ b_2 \cdot a_1 & b_2 \cdot a_2 & b_2 \cdot a_3 \\ b_3 \cdot a_1 & b_3 \cdot a_2 & b_3 \cdot a_3 \end{bmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \cdot \begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix} \tag{3.3}$$

Once the DCM is known, the transformation of vectors in the $\mathbf{F}_A$ frame to the $\mathbf{F}_B$ frame can be calculated using: (Wie, 1998).

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \mathbf{C}_{B,A} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \tag{3.4}$$

Euler Angles: Euler angles are the classical attitude angles and represent the roll angle φ, the pitch angle θ, and the yaw angle ψ. These three angles can represent the attitude of the spacecraft. Euler angles are based on sequential rotations of the original and intermediate reference frames. Any orientation of two frames in 3D space can be described with at most three successive rotations around the given coordinate axes. This therefore does not mean that three rotations are always necessary. This process is shown in Figure 3.1, where a rotation from the I frame to the B frame consists of three successive rotations around the respective axes. The Euler angles are defined as follows:

$$\begin{cases} -\pi \leq \varphi < \pi \\ -\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2} \\ -\pi \leq \psi < \pi \end{cases} \tag{3.5}$$

The DCM expressed in Euler angles is given in Equation (3.6). This DCM can be used to represent the orientation of one reference frame $\mathbf{F}_A$ to another $\mathbf{F}_B$ in Euler angles, where use has been made of the 3-2-1 rotation sequence (yaw, pitch, roll), which is commonly used in aerospace.

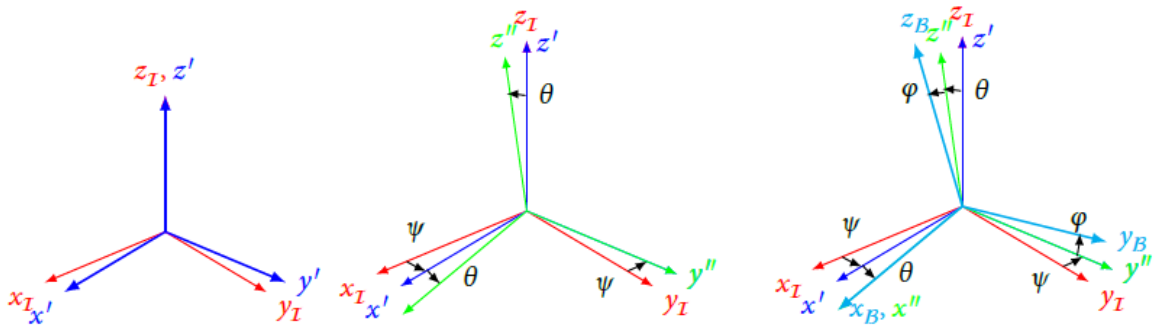


Figure 3.1: Euler angles: successive rotations about the Z_I axis, the y' axis and x'' axis (Gerth, 2014)

$$\mathbf{C}_{B,A} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\varphi\sin\theta\cos\psi - \cos\varphi\sin\psi & \sin\varphi\sin\theta\sin\psi + \cos\varphi\cos\psi & \sin\varphi\cos\theta \\ \cos\varphi\sin\theta\cos\psi + \sin\varphi\sin\psi & \cos\varphi\sin\theta\sin\psi - \sin\varphi\cos\psi & \cos\varphi\cos\theta \end{bmatrix} \tag{3.6}$$

Quaternions: Quaternions are a very popular set of attitude coordinates used in spaceflight as it results in a linear kinematic relation. The quaternions are based on Euler's eigenaxis theorem, which states that "A rigid body or coordinate reference frame can be brought from an arbitary initial orientation to an arbitrary final orientation by a single rigid rotation through a principal angle Φ about the principal axis $\hat{e}$, where the principal axis (Euler axis) is an axis that is fixed in both the initial and final orientation" (Schaub and Junkins, 2018). The Euler eigenaxis rotation is shown in Figure 3.2. Compared to Euler angles, which are a combination of rotations about different axes, the single rotation value Φ gives better insight in the magnitude of the rotation, as large Euler angle rotations could still result in a small attitude difference w.r.t. the initial orientation.

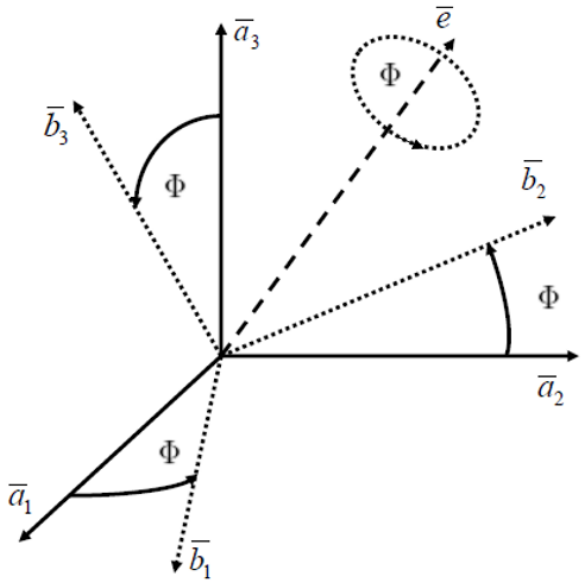


Figure 3.2: The Euler eigenaxis rotation where $\bar{e}$ represent the Euler eigenaxis, $\bar{a}$ the unit vectors of reference frame $\mathbf{F}_A$ and $\bar{b}$ the unit vectors of reference frame $\mathbf{F}_B$, Φ represent the principal rotation angle (Euler angle) (Schaub and Junkins, 2018)

Quaternions are a 4-dimensional hypercomplex number that consists of one real and three imaginary numbers. The quaternions are defined with respect to the principal rotation components (Euler axis components (e_1, e_2, e_3)) and the principal rotation angle (Euler angle, Φ). They are shown in Equation (3.7) and

depending on the convention q_0 can also be named q_4. $\bar{\mathbf{q}} = [q_1, q_2, q_3]$ is referred to as the *vector* part of the quaternion, whereas q_0 is the *scalar* part. Throughout this work the *scalar-first* notation (q_0) is used.

$$\begin{aligned} q_0 &= \cos(\Phi/2) \\ q_1 &= e_1 \sin(\Phi/2) \\ q_2 &= e_2 \sin(\Phi/2) \\ q_3 &= e_3 \sin(\Phi/2) \end{aligned} \tag{3.7}$$

The unit quaternions are constrained by the relationship that the L_2-norm is equal to 1, which is represented as follows: $\|\mathbf{q}\|_2 = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$. The advantage of quaternions is that there are no singularities present, since there are four coordinates to describe a 3-DoF attitude. However, quaternions have discontinuities at 180°, which constrains the application of continuous attitude tracking. However, this is not considered a problem for this work.

The quaternions describe a 3D unit sphere in 4D space, where a rotation from one attitude to another is a trajectory on the surface of this 3D sphere. Furthermore, the orientation described by (q_0,$\mathbf{q}$) is the same as the orientation described by (-q_0,-$\mathbf{q}$). A positive value of q_0 describes the short rotation and a negative value of q_0 describes the long rotation. The quaternion parameterization is therefore not unique.

When two reference frames, $\mathbf{F}_A$ and $\mathbf{F}_B$, are aligned, the quaternion representation would be (1,0,0,0), as the principal rotation angle (Φ) is then zero. Furthermore, a pure rotation about one axis (e.g., a_3 in Figure 3.2) would result in e_1 and e_2 being equal to zero, leaving only q_3 and q_0 as non-zero. For more detailed information about quaternions and their algebra the reader is referred to established literature on the topic, such as the book by Kuipers (2002).

The DCM describing the transformation from $\mathbf{F}_A$ to $\mathbf{F}_B$ can be obtained from the quaternion parameterisation using: (Wie, 1998).

$$\mathbf{C}_{B,A} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 + q_0 q_3) & 2(q_1 q_3 - q_0 q_2) \\ 2(q_1 q_2 - q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 + q_0 q_1) \\ 2(q_1 q_3 + q_0 q_2) & 2(q_2 q_3 - q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{3.8}$$

The inverse relationship, retrieving the unit quaternions representing the rotation from a DCM, can robustly be determined using Sheppard's algorithm. Firstly, the largest value of the relations given in Equation (3.9) is determined. This largest value is then used to calculate the remaining quaternions using the relations given in Equation (3.10).

$$\begin{aligned} q_0^2 &= \frac{1}{4}(1 + \text{trace}([\mathbf{C}])) & q_2^2 &= \frac{1}{4}(1 + 2C_{22} - \text{trace}([\mathbf{C}])) \\ q_1^2 &= \frac{1}{4}(1 + 2C_{11} - \text{trace}([\mathbf{C}])) & q_3^2 &= \frac{1}{4}(1 + 2C_{33} - \text{trace}([\mathbf{C}])) \end{aligned} \tag{3.9}$$

$$\begin{aligned} q_0 q_1 &= (C_{23} - C_{32})/4 & q_1 q_2 &= (C_{12} + C_{21})/4 \\ q_0 q_2 &= (C_{31} - C_{13})/4 & q_3 q_1 &= (C_{31} + C_{13})/4 \\ q_0 q_3 &= (C_{12} - C_{21})/4 & q_2 q_3 &= (C_{23} + C_{32})/4 \end{aligned} \tag{3.10}$$

3.3. Frame transformations

The transformation of one frame w.r.t. another generally consists of a translation and a rotation. The translation is necessary when the origins of the two reference frames do not coincide. Firstly, the rotation of reference frames is discussed after which translation will be covered.

Rotational transformation

This subsection discusses the transformations between the reference frames that are used to present the vector components in different reference frames. The transformations can also be combined to achieve the desired transformation matrices. The transformation matrix that transforms the vector components from frame $\mathbf{F}_A$ to frame $\mathbf{F}_B$ is given by $\mathbf{C}_{B,A}$. The convention used for the transformation matrices is that the reference frames are being rotated instead of the vector itself. This interpretation is often used in aerospace and the only implication is that the signs on the sin term would be reversed for the unit-axis transformation matrices. The following unit-axis transformation matrices are used for the z, y, x axes respectively.

$$\mathbf{C}_z(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

$$\mathbf{C}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.12}$$

$$\mathbf{C}_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi \\ 0 & -\sin\varphi & \cos\varphi \end{bmatrix} \tag{3.13}$$

The transformation matrix from the inertial frame **I** to the body frame **B**, as shown in Figure 3.1, is obtained using the 3-2-1 sequence $\varphi_z \rightarrow \varphi_y \rightarrow \varphi_x$ and is equal to the following.

$$\mathbf{C}_{B,I} = \mathbf{C}_x(\varphi)\mathbf{C}_y(\theta)\mathbf{C}_z(\psi) \tag{3.14}$$

Where $\mathbf{C}_{B,I}$ would be equal to Equation (3.6), the inverse transformation matrix $\mathbf{C}_{I,B}$ can be determined as follows:

$$\mathbf{C}_{I,B} = \mathbf{C}_{B,I}^{-1} = \mathbf{C}_{B,I}^{T} = \mathbf{C}_z(-\psi)\mathbf{C}_y(-\theta)\mathbf{C}_x(-\varphi) \tag{3.15}$$

Once the transformation matrix is known, vectors presented in one frame of reference $\mathbf{F}_A$ can be expressed in an other frame of reference $\mathbf{F}_B$ using the transformation matrix $\mathbf{C}_{B,A}$. This can mathematically be expressed as follows:

$$\mathbf{r}^B = \mathbf{C}_{B,A}\mathbf{r}^A \tag{3.16}$$

The inverse can then also easily be determined using:

$$\mathbf{r}^A = \mathbf{C}_{A,B}\mathbf{r}^B = \mathbf{C}_{B,A}^{T}\mathbf{r}^B \tag{3.17}$$

Translational transformation

As mentioned before, when the origin of one reference frame does not coincide with the origin of the other reference frame, a translation is necessary on top of the rotation. This is illustrated in Figure 3.3, where the location of point P can be expressed in either the A frame using $\mathbf{v}^A$ or in the B frame using $\mathbf{v}^B$. When point P is defined in the A frame, it can be transformed to the B frame using the following equation:

$$\mathbf{v}^B = \mathbf{T} + \mathbf{C}_{B,A}\mathbf{v}^A \tag{3.18}$$

Whereas, if point P is defined in the B frame, it can be transformed to the A frame using the following equation:

$$\mathbf{v}^A = \mathbf{C}_{A,B}\mathbf{v}^B - \mathbf{T} \tag{3.19}$$

3.3.1. Reference frame transformations

The reference frames and the transformations between them that will be used in this work are discussed below. These transformations are necessary to transform the determined position and orientation of one frame to the desired frame.

World reference frame (W) to asteroid reference frame (A)

The origin's of the two frames align and the axes align in the initial orientation. The transformation is therefore only a function of a rotation around the world's z-axis to arrive at the orientation of the asteroid.

$$\mathbf{C}_{A,W} = \mathbf{C}_z(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

World reference frame (W) to the Blender camera frame (B)

The orientation of the Blender camera (B) w.r.t. the world reference frame (W) is determined during the dataset generation, which is elaborated upon in Subsection 7.2.3. This orientation is parameterized as a quaternion, which can be transformed to the DCM using:

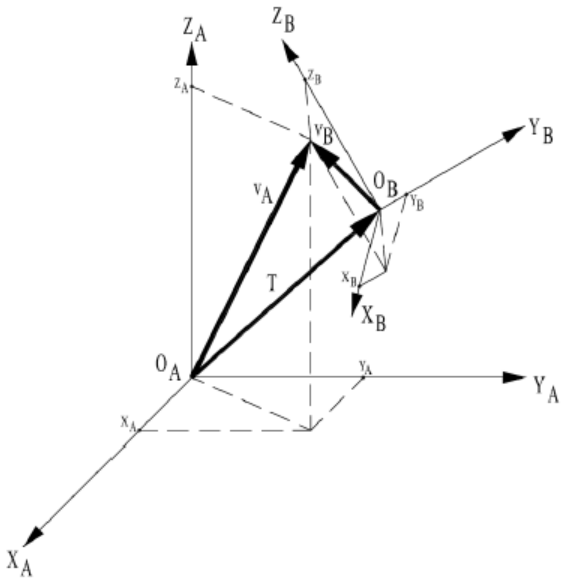


Figure 3.3: The expression of a point from frame A to frame B, composed of a translation and a rotation (Mooij, 2019)

$$\mathbf{C}_{B,W} = \begin{bmatrix} \left(q_0^2+q_1^2-q_2^2-q_3^2\right) & 2\left(q_1q_2+q_0q_3\right) & 2\left(q_1q_3-q_0q_2\right) \\ 2\left(q_1q_2-q_0q_3\right) & \left(q_0^2-q_1^2+q_2^2-q_3^2\right) & 2\left(q_2q_3+q_0q_1\right) \\ 2\left(q_1q_3+q_0q_2\right) & 2\left(q_2q_3-q_0q_1\right) & \left(q_0^2-q_1^2-q_2^2+q_3^2\right) \end{bmatrix} \tag{3.21}$$

Blender camera frame (B) to the camera reference frame (C)

The transformation between the Blender camera frame (B) and the camera reference frame (C) is fixed and is given below.

$$\mathbf{C}_{C,B} = \mathbf{C}_x(-\pi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(-\pi) & \sin(-\pi) \\ 0 & -\sin(-\pi) & \cos(-\pi) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{3.22}$$

Camera reference frame (C) to pixel reference frame (P)

A point in the camera reference frame can be expressed in the pixel reference frame by making use of the camera intrinsic parameters matrix $\mathbf{K}$, $\mathbf{r}^P = \mathbf{K}\mathbf{r}^C$, which is discussed in Section 4.1. The intrinsic camera parameters used in this work are shown below and can also be found in Table 7.2.

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4266.67 & 0 & 512 \\ 0 & 4266.67 & 512 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.23}$$

4

Pose estimation framework

This chapter discusses the pose estimation framework. As discussed in Chapter 2, a monocular vision-based navigation algorithm is developed in this work. A model-based monocular pose estimation pipeline uses a 2D image as the input and produces an estimate of the pose of the target w.r.t. the camera. The pose refers to the distance and orientation of the target w.r.t. the camera. Section 4.1 discusses the camera model that was used to generate the image dataset and develop the vision-based algorithm. Furthermore, Section 4.2 discusses the PnP problem, which uses detected features from an IP algorithm to output an estimate of the pose. Different dedicated pose solvers that are used to solve the PnP problem are discussed in Section 4.3.

4.1. Camera model

The camera is described using the pinhole camera model. This model is widely used in computer vision and is simple to implement. Furthermore, this is used by the 3D rendering software employed in this work. This simple model accounts for the focal length and pose of the camera, and is sufficient for the purpose of relating a 2D point in the image plane to a 3D point in the camera frame, which is required in this work. The virtual pinhole model is implemented, where the projection plane (image plane) is placed in front of the center of projection. This makes it easier to analyze and it does not require inverting the image. The following definitions are taken from Sturm (2014):

- The *focal length* f is the distance between the center of projection (optical center) and the image plane.
- The line passing through the center of projection and that is orthogonal to the image plane is called the *optical axis*
- The intersection point of the optical axis and the image plane is the *principal point*.

The virtual pinhole camera model and the aforementioned definitions are shown in Figure 4.1. The focal length influences the perspective projection, if the focal length f gets smaller, more points are projected on the image plane, i.e., FOV increases. The opposite effect occurs if the focal length increases. The perspective projection scales real world objects on the image plane in relation to the distance of the object to the camera. This is different from orthographic projection, where the size of the object within the image does not change with the distance of the object from the camera.

The 3D point **P** corresponds to the 2D point **p** in the image plane as shown in Figure 4.1. The coordinates of point **p** in the image plane can be calculated using the coordinates of point **P** in the camera frame (x^C, y^C, z^C) and the focal length f, by the use of equivalent triangles. These are the perspective projection equations and the coordinates of point **p** relative to the principal point are given below.

$$\mathbf{p} = \left(f\frac{x^C}{z^C}, f\frac{y^C}{z^C} \right) \tag{4.1}$$

The projection of a point **P** is not unique, as any point on the line OP has the same projection **p**. The focal length and the 3D coordinates are measured in meters or millimeters. However, the pixel coordinates (u, v) are measured in pixel distances. Therefore, the scale factors s_x and s_y are introduced to convert the coordinates to pixel distances. This results in the following, $f_x = s_x f$ and $f_y = s_y f$, where the scaling factors

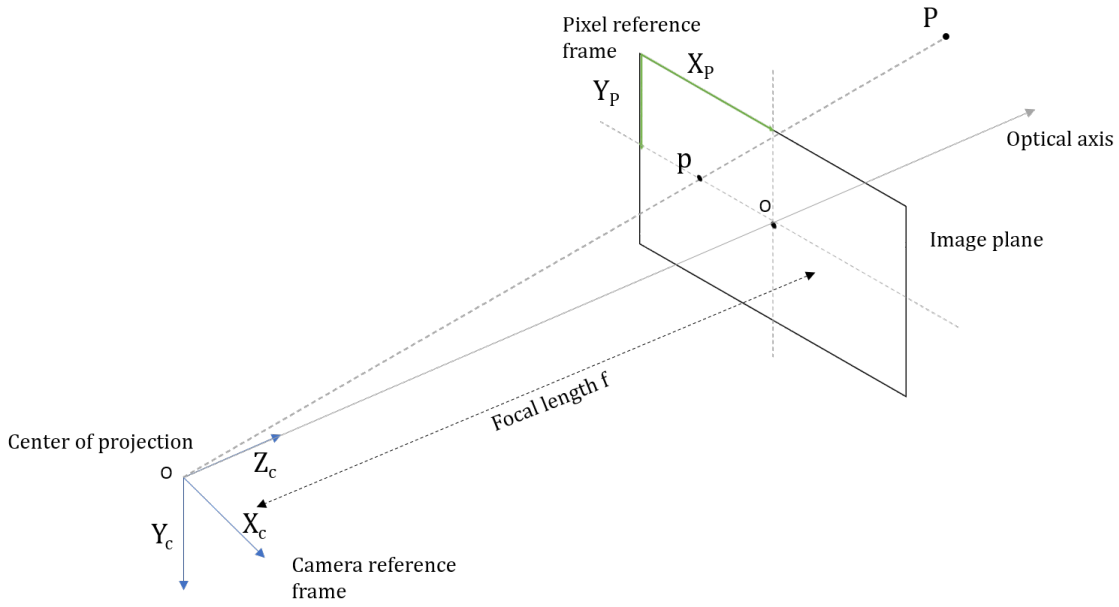


Figure 4.1: The virtual pinhole camera model containing the camera reference frame and the pixel reference frame and the other relevant definitions

are related to the dimensions of a pixel in geometric units, such as meters or millimeters. These scale factors are identical when dealing with square pixels. The point **P** can be expressed in pixel coordinates, relative to the principal point with the following matrix notation.

$$\mathbf{p} = \begin{bmatrix} \frac{f_x}{z^C} & 0 & 0 \\ 0 & \frac{f_y}{z^C} & 0 \end{bmatrix} \begin{pmatrix} x^C \\ y^C \\ z^C \end{pmatrix} \tag{4.2}$$

This method of transforming 3D coordinates to image coordinates is not practical, as the transformation matrix depends on the distance of the 3D point to the camera (z^C). Therefore, homogeneous coordinates are used to describe the transformation between the 3D points and the image coordinates with a unique matrix that is independent of the distance of the 3D point to the camera.

Normally, an object's position can be described in three-dimensional space using Euclidean geometry with three coordinates (x, y, z). However, to describe those coordinates in image coordinates, projective geometry is used, which has an extra dimension (w). This four-dimensional space is called the projective space and the coordinates in the projective space are called homogeneous coordinates. This extra dimension w, scales the x, y, z coordinates and therefore homogeneous coordinates can be viewed as scaled coordinates that remove the dependency of the projection coordinates on the distance of the 3D object w.r.t. camera.

The concept of homogeneous coordinates is best explained through an analogy in 2D. Imagine a projector at a distance (w) of 3 m away from a screen. There is a point present on that screen with the 2D coordinates w.r.t. principal point of (12,15). The projective coordinate vector (x, y, w) would then be equal to (12,15,3). When moving the projector closer to the image until $w = 1$, the projected image become three times smaller and the coordinates of the point need to be scaled by the same amount, resulting in the new projective coordinate vector $(\frac{12}{3}, \frac{15}{3}, \frac{3}{3}) = (4, 5, 1)$. The 2D coordinates of the point w.r.t. the principal point are (4,5). Therefore, homogeneous coordinates are coordinates that remove the dependency on the distance from the point to the camera, by setting the distance $w = 1$, and scaling the x, y coordinates accordingly. This concept is similarly applied to 3D. Rewriting Equation (4.2) to express **p** in homogeneous coordinates results in:

$$\mathbf{p} = \begin{bmatrix} f_x & \alpha & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x^C \\ y^C \\ z^C \\ 1 \end{pmatrix} \tag{4.3}$$

As discussed in Section 2.3, an ideal model is assumed, i.e., no skew factor ($\alpha = 0$), which can occur when

the optical axis is not perfectly perpendicular to the image plane. Furthermore, no distortion is taken into account.

Up until now the coordinates of **p** have been described w.r.t. the principal point. However, to express the coordinates of **p** in the pixel reference frame (P), $\mathbf{p}^P$, these coordinates need to be offsetted with the coordinates of the principal point (c_x, c_y) presented in the pixel reference frame. Therefore, the coordinates of point P can be presented in the pixel reference frame P as follows.

$$\mathbf{p}^P = \begin{pmatrix} u & v & 1 \end{pmatrix}^T = \left(c_x + f\frac{x^C}{z^C}, c_y + f\frac{y^C}{z^C} \right) \tag{4.4}$$

This can be written in the following matrix notation, using homogeneous coordinates measured in pixel coordinates.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x^C \\ y^C \\ z^C \\ 1 \end{pmatrix} \tag{4.5}$$

This transformation matrix between a point expressed in the camera frame and the pixel reference frame is referred to as the camera intrinsic parameter matrix **K**. The camera intrinsic parameter matrix is important in solving the P*n*P problem, which will be elaborated upon in Section 4.2. As addressed in Section 2.3, the camera intrinsic parameters are assumed to be known. However, when using an actual camera, these parameters are estimated through a calibration procedure.

4.2. Perspective-*n*-Points (P*n*P) problem

The goal of the Perspective-*n*-Points (P*n*P) problem is to determine the distance and orientation of the target with respect to the camera, given the camera's intrinsic parameters and the set of n 2D-3D correspondences between the 3D body coordinates and the 2D projections. The 3D model of the target needs to be available for the estimation. This approach is often used in computer vision problems and has also been applied to estimate the pose of uncooperative spacecraft by Barad (2020), Chen et al. (2019), Park et al. (2019), and Pasqualetto Cassinis et al. (2020).

The P*n*P problem can be mathematically described in the following way. Let $\mathbf{r}^A = \begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^T$, where i = 1,2, ... , n, where n represents the number of 3D landmarks/features/keypoints on the 3D model of the asteroid defined in the asteroid reference frame (A). Furthermore, let $\mathbf{p}_i^P = \begin{pmatrix} x_i & y_i & 1 \end{pmatrix}^T$, where i = 1,2, ... , n, represents the n corresponding 2D image points in the pixel reference frame (P). The pixel reference frame (P) is related to the camera reference frame through the camera's intrinsic parameters as discussed in Section 4.1.

The P*n*P problem consists of determining the position of the asteroid's center of mass $\mathbf{t}^C$ and the orientation of the asteroid with respect to the camera frame (C). This transformation matrix from the asteroid to the camera frame is given by $\mathbf{C}_{C,A}$. The P*n*P equations are given below and they relate the unknown pose to a detected 2D feature **p**, using the position of that feature w.r.t. camera frame, $\mathbf{r}^C$. The P*n*P problem is graphically shown in Figure 4.2.

$$\mathbf{r^C} = \begin{pmatrix} x^C & y^C & z^C \end{pmatrix}^T = \mathbf{t}^C + \mathbf{C}_{C,A}\mathbf{r}^A \tag{4.6}$$

$$\mathbf{p} = \left(\frac{x^C}{z^C} f_x + c_x, \frac{y^C}{z^C} f_y + c_y \right) \tag{4.7}$$

where f_x and f_y refer to the scaled focal lengths ($f_x = s_x f, f_y = s_y f$) in the respective principal directions and (c_x, c_y) refer to the coordinates of the principal point in the P frame as discussed in Section 4.1. A feature on the 3D model can be expressed in the camera frame with Equation (4.6), and the corresponding 2D points **p** in the pixel reference frame plane are obtained using Equation (4.7), relating 3D with 2D.

As was discussed in Section 3.1, $+Z_C$ is assumed to be aligned with the boresight of the camera (optical axis) and $+X_C$ and $+Y_C$ are aligned with the pixel reference frame axis X_P and Y_P. Using this, Equations (4.6) and (4.7) can be rewritten in homogeneous coordinates and matrix form resulting in Equation (4.8). The **K** matrix represents the internal camera matrix also referred to as the camera intrinsic parameters matrix as discussed in Section 4.1. The pose matrix $\mathbf{P} \in \mathbb{R}^{3\times4}$ contains the transformation matrix from the asteroid

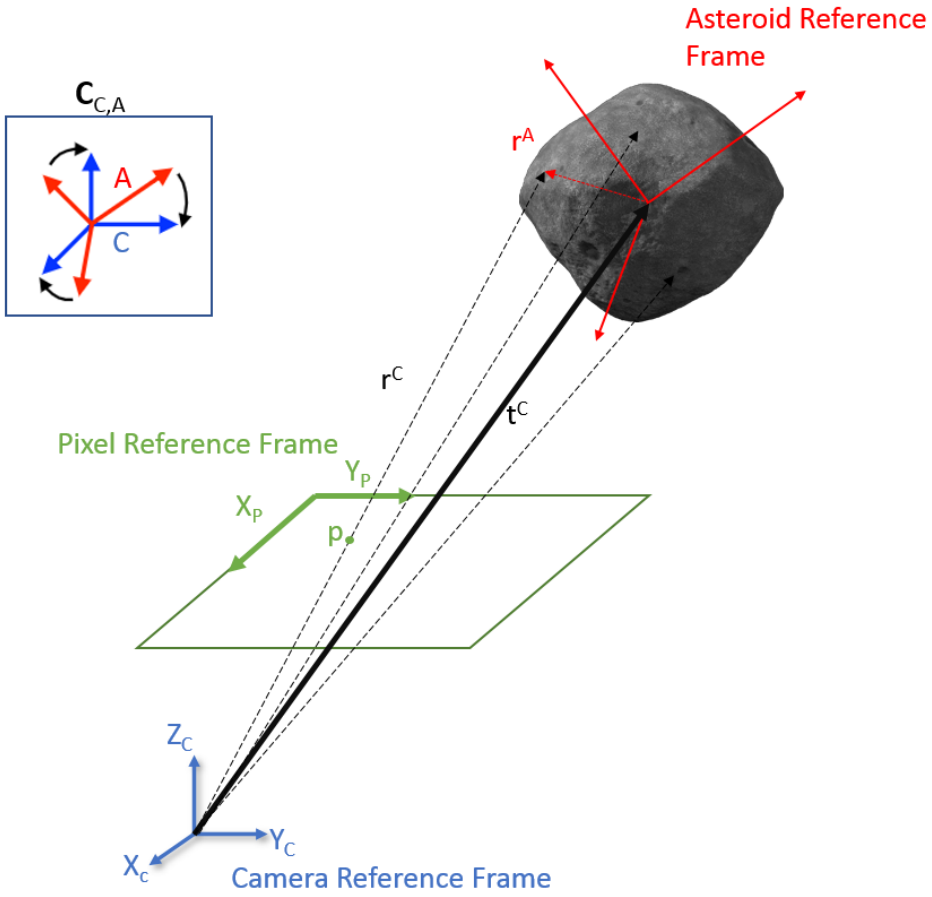


Figure 4.2: The geometric depiction of the PnP problem

reference frame (A) to the camera reference frame (C) $\mathbf{C}_{C,A}$ and the translation vector $\mathbf{t}^C$, which is the same as $\mathbf{r}^C_{A/C}$.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = [\mathbf{K}][\mathbf{P}] \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \tag{4.8}$$

Filling in the respective parameters for **K** and **P** results in:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{C}_{C,A} & | & \mathbf{r}^C_{A/C} \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \tag{4.9}$$

The PnP problem as shown in Equation (4.9) contains six unknowns, namely the three attitude descriptors and the three position coordinates. Equation (4.9) demonstrates that an important aspect of generating accurate pose estimates is the capability of the IP algorithm to extract features from the 2D image. These 2D feature coordinates (u_i, v_i) constrain the location of the 3D points as they have to lie on the perspective line through the 2D points. Therefore, each 2D feature detection will result in two equations indicating that for $n = 3$, the system is fully defined, i.e., six equations for six unknowns. However, the $P3P$ problem generally results in eight solutions of which four will be behind the image plane, leaving four actual solutions. These four solutions are caused by the viewpoint ambiguity as shown in Figure 4.3. The fixed rigid triangle, defined by the three 3D points, can be oriented in four different ways, such that the vertices all lie on the respective perspective lines, indicating that they have the same 2D point locations in the image frame.

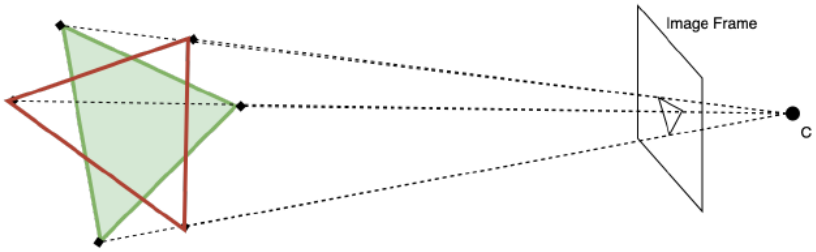


Figure 4.3: The viewpoint ambiguity that exists for $P3P$ where two of the four possible orientations are shown. The different orientations of the three 3D point locations have the same 2D locations in the pixel reference frame (P) (Barad, 2020)

This ambiguity is the result of the non-linear transformation between the Euclidian space and the homogeneous space. A unique solution is found when $n = 6$, however, with actual systems that are observing these points in an image, there could still be problems due to noisy detections of the feature location. This could result in solutions that are very close to the true solution in image coordinates, but very far from the actual solution in the orientation of the object, these are referred to as near solutions. Furthermore, there could still be problems with coplanarity and object symmetry. Generally, the more (accurate) detections that are used in the estimation ($n \geq 6$), the better the pose estimate. There is a lot of research into designing efficient and accurate methods (pose solvers) to solve the PnP problem. An overview of the different techniques and pose solvers is given in the subsequent Section 4.3.

4.3. Pose solvers

There are several PnP solvers and generally they can be divided into two different categories, namely based on whether they use an iterative or non-iterative approach:

- **Iterative approach:** This approach tries to minimize the error-based objective function that is defined in the image or object space, thereby solving Equations (4.6) and (4.7), while taking all the non-linear constraints into account. This can be solved in an iterative manner, which would require an initial estimate and tends to be computationally expensive to achieve a desirable result. These methods, however, have a high accuracy when they converge. However, for some methods it is not guaranteed that the obtained minimum for the error-based objective function is the global minimum and these iterative methods tend to get stuck in local minima. The most commonly used iterative solvers are Pose from Orthography and Scaling with Iteration (POSIT) (Dementhon and Davis, 1995), Coplanar POSIT (Oberkampf et al., 1996), Lu-Hager-Mjolsness (LHM) (Lu et al., 2000).
- **Non-iterative approach:** This approach solves the PnP problem by using linear forms of Equations (4.6) and (4.7). This results in a closed-form solution, thereby removing the need for an initial pose estimate. The use of linear forms greatly improves the computational speed, however, the accuracy of these methods are generally lower compared to the iterative approaches. The most commonly used *non-iterative* approach is the EPnP solver created by Lepetit et al. (2009).

The EPnP solver has been proven to be robust to noise on the predicted 2D keypoint location, however, as well as the other discussed solvers it does not take into account detection uncertainty. Ferraz et al. (2014) proposed the CEPPnP solver, which takes the detection uncertainty into account in the form of a covariance matrix. By quantifying the uncertainty belonging to a detection, the resulting solver knows which detections it needs to rely on more heavily and which it should discard for the determination of the pose. This quantification increases the accuracy of the resulting pose estimation.

Pasqualetto Cassinis et al. (2020) proposed a method that allows to quantify the detection uncertainty by deriving a covariance matrix from the heatmap outputted by the keypoint detection network. This could then subsequently be used by the CEPPnP solver. However, the implementation is not trivial as the network outputs 68 keypoints per image (Section 7.4), resulting in over 300,000 heatmaps. Therefore, due to time-limitations this method was not implemented in this work, but the basic concept of extracting the covariance matrix from the detection heatmap is discussed in Appendix A, as this is discussed further in the recommendations, Section 12.2.

5

Machine learning

This chapter discusses the theoretical background behind the machine learning concepts used within this work. Firstly, an introduction to deep learning is given in Section 5.1. Secondly, Section 5.2 discusses the different facets of implementing a neural network to build a basic intuition regarding the concepts. The Convolutional Neural Networks, which are the state-of-the-art architecture for computer vision, are discussed in Section 5.3. Section 5.4 discusses the building blocks underlying the current trend of optimizing the networks for usage on mobile and embedded devices. The chapter is concluded with general machine learning concepts in Section 5.5 that are vital for understanding the discussions in later chapters.

5.1. Deep Learning

Within the last decade Deep Learning (DL) has taken off through the increase of available data caused by the digitization of society. Traditional learning algorithms, such as logistic regression or Support Vector Machine (SVM), could not take advantage of the availability of more data and the performance of these methods plateaued. However, neural networks and CNNs were able to harness this data and outperform the traditional methods on challenging datasets, such as ImageNet, which consists of 14 million images containing a total of 20,000 categories (Deng et al., 2009).

Deep learning refers to networks that have three or more hidden layers. State-of-the-art deep learning networks, such as ResNet (He et al., 2016), can have up to 100 hidden layers. Deep learning is not limited to supervised learning, but can also be applied to unsupervised or reinforcement learning. However, the focus in this work will be on a specific branch of supervised deep learning, namely computer vision, in which convolutional neural networks are the state-of-the-art architecture.

Supervised learning is involved with finding the underlying function that maps x to y, using labeled training examples of x and y. This function is then used to predict y when presented with new data of x. Supervised learning can be divided into *classification* problems and *regression* problems. Classification means that the algorithm is for example trained to identify the type of animal present in the picture e.g., dog, cat, horse and the output is a finite, discrete set of values, it is either classified as 1,2 ... n, depending on the number of classes (animals). Regression means that the algorithm is, for example, trained to determine the price of a house based on its square footage, the output is a continuous number. These different problems within supervised learning require different algorithms.

There are two advantages of using deep representations. The first advantage is that the different layers within a neural network can extract different levels of abstraction in a hierarchical order. This process is graphically illustrated in Figure 5.1, where the first layers have a high resolution and detect more low-level features, such as edges or lines, that do not really have semantic meaning. These low-level features are added together to be able to learn more complex features, such as eyes, noses, and cheeks and eventually entire faces. Therefore, the features detected by the deeper layers of the network have more semantic meaning. The second advantage is that a small, but deeper network, which has more layers, but less hidden units, is computationally more efficient in representing the same function compared to a large and shallow network.

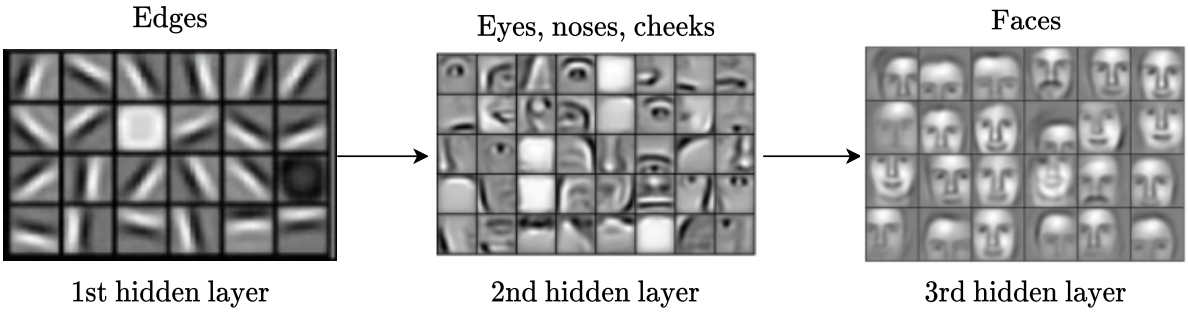


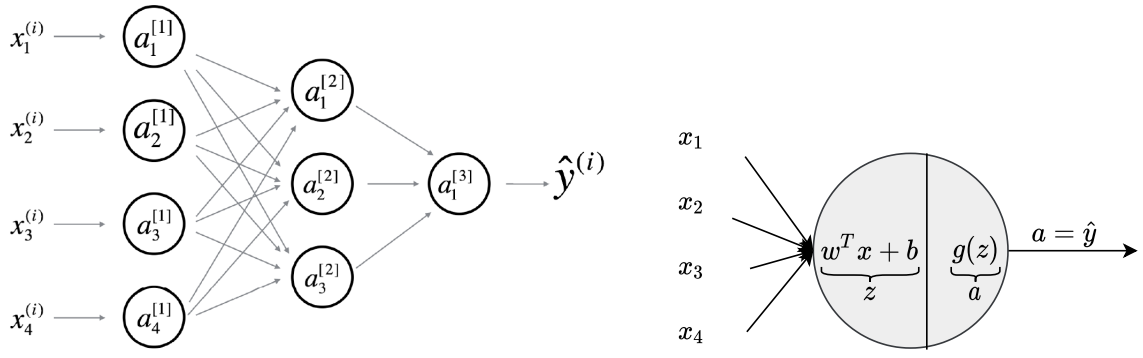
Figure 5.1: Graphically showing the hierarchical process of deep learning in which the first layers extract simple features, such as edges, whereas the later layers extract more complex and semantically strong features such as eyes, noses, and cheeks and eventually an entire face. Adapted from Lee et al. (2011).

5.2. Neural Networks (NN)

A neural network is used to estimate a function that maps the input x to the output y and consists of an input layer, one or more hidden layers, and an output layer. The input layer is a flattened (1D) array of the input, where for an image each input feature $x_1, x_2, ..., x_n$ corresponds to a pixel. The basic architecture of a (fully connected) neural network is graphically shown in Figure 5.2a. The size of an L-layer neural network, is expressed using the number of hidden layers and the output layer. Whenever a neural networks contains three or more hidden layers it is called a deep neural network. Each layer of the neural network consists of neurons/nodes/units and for a fully connected layer each unit in a given layer is connected to units from the previous layer, aside from the input layer. Neural networks are capable of estimating highly non-linear mappings from input to output. This capability was described by Cybenko (1989) and is called the universality theorem. This theorem states that a neural network consisting of at least one hidden layer, with a sufficient, but finite amount of units, is able to approximate any continuous non-linear function. The basic building blocks of a neural network can be coarsely divided into two sections, namely forward propagation and backward propagation, each of which will be discussed separately.

Forward propagation

The input x is transferred from the input layer to the output layer, which is called *forward propagation*. The output of the neural network is thereby computed by passing the input data through several hidden layers. Each node as shown in Figure 5.2b consists of tunable parameters w and b, weight and bias, respectively. Each node performs two calculations in succession as graphically illustrated in Figure 5.2b. Firstly, it calculates $z^{[l]}$ by multiplying the inputs of the neuron (x) with the weights and adding a bias as shown in Equation (5.2).



(a) The comprehensive network representation of a three-layer Neural Network

(b) Schematically showing the two computations that a node computes, firstly it computes z and then it computes a

Figure 5.2: General architecture of a neural network and a schematic view of what each node computes

Secondly, an *activation function* $g(z)$ is used to scale the output from the neuron (z) to a desired output range. These activation functions have to be non-linear, as the use of linear activation functions would result in the layers becoming linear and thereby rendering the network unable to estimate non-linear functions. The most commonly used activation function in deep learning is the Rectified Linear Unit (ReLu) function as it was shown by Krizhevsky et al. (2012) that this activation function accelerated convergence of the network by a factor of six. The ReLu activation function exists between $\begin{bmatrix}0 & \infty\end{bmatrix}$. The function gives output z if $z > 0$ and zero otherwise.

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \tag{5.1}$$

$$\begin{aligned} z &= w^T x + b \\ a &= g(z) \end{aligned} \tag{5.2}$$

The forward propagation is explained using Figure 5.2a, where the 1D input vector is given in Equation (5.1). Each neuron in layer l calculates Equation (5.2) based on the inputs to that neuron. For a single training example this would result in the following for the first two layers.

$$\begin{array}{cc} z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]} & z_1^{[2]} = w_1^{[2]T} a^{[1]} + b_1^{[2]} \\ a_1^{[1]} = g^{[1]}(z_1^{[1]}) & a_1^{[2]} = g^{[2]}(z_2^{[2]}) \\ \vdots & \vdots \\ z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]} & z_3^{[2]} = w_3^{[2]T} a^{[1]} + b_3^{[2]} \\ a_4^{[1]} = g^{[1]}(z_4^{[1]}) & a_3^{[2]} = g^{[2]}(z_3^{[2]}) \end{array} \tag{5.3}$$

The input to the second layer $l = 2$, is the activation of the first layer $l = 1$, meaning that for $l > 1$ the input is given by $a^{[l-1]}$. The parameters for each layer can be stacked together to form the following vectors for one training example i shown for layer $l = 1$.

$$z^{[1]} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{pmatrix} \qquad a^{[1]} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{pmatrix} \qquad b^{[1]} = \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{pmatrix} \qquad \mathbf{W}^{[1]} = \begin{bmatrix} \cdots w_1^{[1]T} \cdots \\ \cdots w_2^{[1]T} \cdots \\ \cdots w_3^{[1]T} \cdots \\ \cdots w_4^{[1]T} \cdots \end{bmatrix} \tag{5.4}$$

The general equation can be formulated as follows, where l refers to the current layer

$$\begin{aligned} z^{[l]} &= w^{[l]T} a^{[l-1]} + b^{[l]} \\ a^{[l]} &= g^{[l]}(z^{[l]}) \end{aligned} \tag{5.5}$$

However, this is for one training example only, for m training examples this would result in the following equation, which is the general form of forward propagation for a neural network for m training examples.

$$\begin{aligned} Z^{[l]} &= W^{[l]T} A^{[l-1]} + b^{[l]} \\ A^{[l]} &= g^{[l]}(Z^{[l]}) \end{aligned} \tag{5.6}$$

where the respective matrices are given by Equation (5.7). The superscript i denotes the i^{th} training example while superscript $[l]$ denotes the l^{th} layer. Furthermore, $n^{[l]}$ refers to the number of neurons in layer l, m refers to the total number of training examples and for the input layer $n^{[0]} = n_x$.

$$\mathbf{Z}^{[l]} = \begin{bmatrix} z_1^{[l](1)} & z_1^{[l](2)} & \cdots & z_1^{[l](m)} \\ z_2^{[l](1)} & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \end{bmatrix}, \mathbf{Z}^{[l]} \in \mathbb{R}^{n^{[l]} \times m} \quad \mathbf{A}^{[l]} = \begin{bmatrix} a_1^{[l](1)} & a_1^{[l](2)} & \cdots & a_1^{[l](m)} \\ a_2^{[l](1)} & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \end{bmatrix}, \mathbf{A}^{[l]} \in \mathbb{R}^{n^{[l]} \times m}$$

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & \vdots & & \vdots \\ x_3^{(1)} & \vdots & & \vdots \\ x_4^{(1)} & \vdots & & \vdots \end{bmatrix}, \mathbf{X} \in \mathbb{R}^{n_x \times m} \qquad \mathbf{W}^{[l]} = \begin{bmatrix} \cdots w_1^{[1]T} \cdots \\ \cdots w_2^{[1]T} \cdots \\ \cdots w_3^{[1]T} \cdots \\ \cdots w_4^{[1]T} \cdots \end{bmatrix}, \mathbf{W}^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}} \tag{5.7}$$

$$\mathbf{b}^{[l]} \begin{bmatrix} \cdots b_1^{[1]} \cdots \\ \cdots b_2^{[1]} \cdots \\ \cdots b_3^{[1]} \cdots \\ \cdots b_4^{[1]} \cdots \end{bmatrix}, \mathbf{b}^{[l]} \in \mathbb{R}^{n^{[l]} \times m}$$

In conclusion, the forward propagation consists of feeding the training examples through the network and multiplying it with the weight and biases for the respective layers to finally end up at the output layer. The network's weights and biases have to be initialized with small random values, as initializing with zero will result in the symmetry problem meaning that every hidden unit will calculate exactly the same function regardless of the duration of training, i.e., there is no point in having more than one hidden neuron. Depending on the task the output layer could be used to output a probability or a number of coordinates. Fortunately, when implementing neural networks these basic functions do not have to be programmed by hand and implementations of these functions can be found in open-source machine learning platforms, such as Keras, which is a part of TensorFlow[1], and PyTorch[2]. The forward propagation is determined by the network's architecture, which specifies the number of neurons and hidden layers.

Backpropagation

A cost function is required to be able to train the parameters w and b. For any given task, the goal of the network would be that for m training examples, the predicted output $\hat{y}^{(i)}$ for training example i closely resembles the actual (desired) output $y^{(i)}$, i.e., the goal when training a neural network is to minimize the error between the predicted $\hat{y}$ and the actual output y as listed below.

$$\text{Given } \left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}, \text{ want } \hat{y}^{(i)} \approx y^{(i)} \tag{5.8}$$

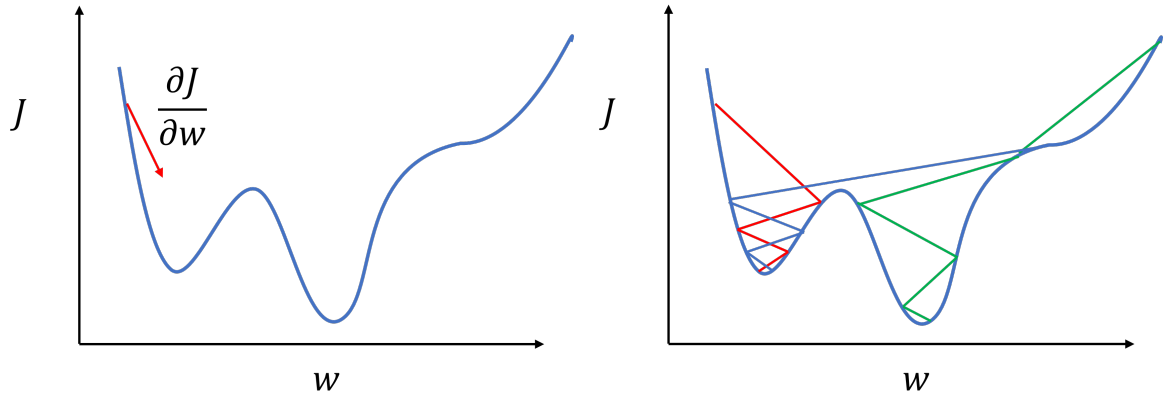
The loss function represents the accuracy of the prediction by comparing the predicted value with the true value on example i and therefore serves as a measurement of the performance of the network. The loss function depends on the problem and the selection of an inappropriate loss function will affect the performance of the algorithm.

The loss function ($\mathscr{L}\left(\hat{y}, y\right)$) evaluates how well the network performs on a single training example i, whereas the cost function evaluates how well the network performs on the entire dataset, i.e., m examples. The standard equation for a cost function is given by:

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathscr{L}(\hat{y}^{(i)}, y^{(i)}) \tag{5.9}$$

During training the neural network tries to find the weights and biases (w,b) that minimize the overall cost function $J(w,b)$. This is achieved using *gradient descent*, which iteratively moves in the direction of the steepest descent, which is defined by the negative of the gradient. However, there is no guarantee that the minimum obtained is the global minimum, as the majority of problems are not necessarily convex. The process of gradient descent is graphically illustrated in Figure 5.3 for a single dimension of w, however, in reality $w, b \in \mathbb{R}^n$. The gradient descent update law is given by:

[1] `https://keras.io/about/`, Date accessed: 13-1-2022
[2] `https://pytorch.org/`, Date accessed: 13-1-2022



(a) The gradient descent

(b) The effect of the learning rate and initialization point

Figure 5.3: The gradient descent method and the impact of the learning rate and initialization point on the performance

$$\begin{aligned} w &:= w - \alpha \frac{\partial J(w,b)}{\partial w} \\ b &:= b - \alpha \frac{\partial J(w,b)}{\partial b} \end{aligned} \tag{5.10}$$

This gradient descent update law depends on two parameters, namely the learning rate α and the gradient of the loss function w.r.t. the respective parameter w or b. This learning rate α is one of the most important *hyperparameters* (Section 5.5) and is dependent on the problem and has to be carefully selected. Having a small learning rate results in slow convergence to the local minimum, while larger learning rates will result in faster convergence, but it can also lead to unstable behavior, such as overshoot or erratic jumps in performance. The learning rate can also be set to decay over time, where the assumption is that the network is already close to the actual minimum and a smaller learning rate at that point would allow for better convergence to the minimum instead of oscillating around it. This is illustratively shown in Figure 5.3b, where the blue line represents a (too) large learning rate and the green line a good learning rate. Furthermore, the red line illustrates the effect of the initialization point, resulting in ending up in a local minimum.

The gradient of the cost function $\frac{\partial J(w,b)}{\partial w}, \frac{\partial J(w,b)}{\partial b}$ is calculated using *backpropagation*, which uses the chain rule recursively to compute this gradient in a computationally efficient manner. The gradient shows how sensitive the cost function is to each of the weights and biases of each layer. This allows the model to change those weights and biases that will cause the most efficient/effective decrease to the cost function. These weights and biases can then be updated to bring about this decrease of the cost function. Backpropagation starts from the output layer and works back towards the input layer, computing the dependencies of each layer on the output. The mathematics of backpropagation for neural networks are more involved than the forward propagation and when implementing neural networks in the open-source machine learning platforms TensorFlow (Keras) and PyTorch, only the forward propagation and cost function needs to be defined. This entails specifying the number of neurons and layers and then *backpropagation* is automatically computed.

More advanced optimization methods compared to (normal) gradient descent are gradient descent with momentum[3], Root Mean Square (RMS) prop[4] and Adam optimization (Kingma and Lei Ba, 2015). These optimization methods rely on the same basic concept of gradient descent and the advanced additions result in increased efficiency and better convergence. However, these optimization methods are not discussed in detail within this section, but will be discussed in Chapter 8 and 9.

5.3. Convolutional Neural Networks (CNN)

CNNs are a special form of neural networks and are used for computer vision. The concepts of the aforementioned forward propagation and backpropagation also apply. The challenge of using regular neural networks for computer vision problems is that the image size can be large, which results in an enormous input feature

[3] `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD`, Date accessed: 29-11-2021

[4] `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSprop`, Date accessed: 29-11-2021

vector x and consequently huge computational costs. As an example, a Red, Green, and Blue (RGB) image of 1000×1000 px would result in a feature vector of around 3 million parameters and given a first layer with a thousand hidden units, this would already result in having 3 billion parameters, adding more layers will only compound to this: $\mathbf{x} \in \mathbb{R}^{3M}, \mathbf{W}^{[1]} = (1000, 3M)$. The solution is to make use of convolutions, which transform the standard neural network into a CNN. CNNs are deep neural networks and have first been proposed to tackle the problem of recognizing handwritten zip codes in 1989 (LeCun et al., 1989). The standard architecture of the CNN was defined in the following years by LeCun and Bengio (1998). The seminal work of AlexNet (Krizhevsky et al., 2012) has sparked the renewed interest in the usage of CNNs for computer vision. AlexNet showed that CNNs were capable of greatly outperforming the state-of-the-art of image classifying algorithms on the ImageNet classification benchmark (Deng et al., 2009). Since then, CNNs have been used extensively for computer vision applications. A CNN consists of three types of layers:

1. Convolution layers
2. Pooling layers
3. Fully connected layers

The convolutional and pooling layers are used for feature extraction and are specific to CNNs. The last layers are fully connected similar to traditional neural networks and are used to solve the classification or regression problem using the extracted features. The convolutional and pooling layers are discussed in more detail in this section.

Convolution

The convolutional layers are used to detect features through the use of a convolution operation, where an input image is convolved with a filter/kernel. These filters contain weights and the convolutional operation in 2D is mathematically represented by Equation (5.11) (Goodfellow et al., 2016). The filter size f is almost always an odd number to avoid asymmetry. Moreover, the filter has a central position. In essence, at each location the dot product between the element of the filter (weight w) and the input element x it overlaps is determined. The total result is summed to arrive at the output value y for that location.

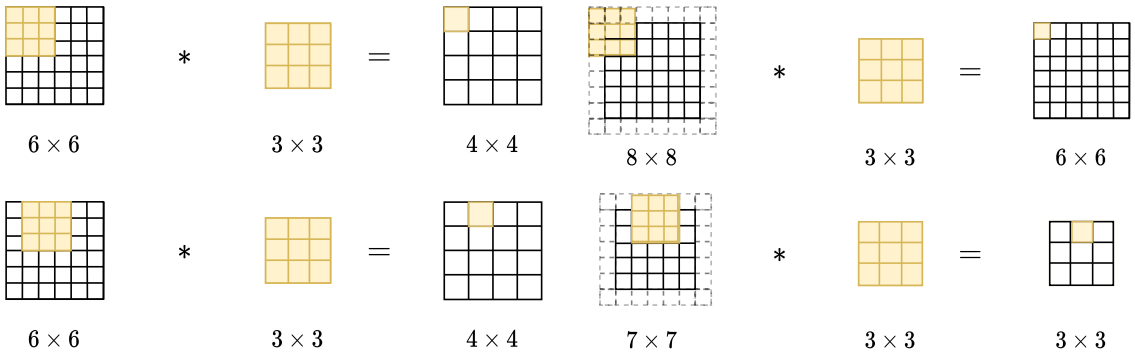
$$y_{i,j} = (x * w)_{i,j} = \sum_f \sum_f x_{f,f} \cdot k_{i-f,j-f} \tag{5.11}$$

where $*$ is the convolutional operator, i and j refer to the 'x' location (width) and 'y' location (height), respectively, of the center location of the kernel. The kernel size is represented by $f \times f$. The origin is in the top left corner of the filter and counting starts from one. The convolutional operation is graphically illustrated in Figure 5.4a. As can be observed in Figure 5.4a, when applying the convolutional operation the image shrinks, which can result in a very small image when a lot of layers are used within a network. Secondly, because the filter cannot be applied near the edges, valuable information from those regions is lost. Generally, when using a 3×3 filter, two pixels will be lost in the vertical and horizontal direction. This is because a pixel in the top-left corner has no top and left neighbors and therefore the filter cannot be applied there, as the filter should lie entirely within the image. The solution to this is padding, which adds p layers of pixels with value zero around the input image as shown in Figure 5.4b. There are two convolutions, *valid* and *same* convolutions, where *valid* convolutions refer to convolution without padding (Figure 5.4a) and *same* convolution uses padding, such that the output size of the convolutional operation is the same as the input size of the image (Figure 5.4b).

Furthermore, the stride s determines the step size that the filter takes as it 'slides' across the image. A stride s of 1 means that the filter moves one pixel to the right, which is the case in Figure 5.4a, whereas the bottom image of Figure 5.4b has a stride s of 2. The filter size f, the stride s, and the padding p are *hyperparameters* (Section 5.5) of the convolutional layer and influence the resulting output size as shown below, where n refers to the input size.

$$(n \times n) \quad * \quad (f \times f) \quad \rightarrow \quad \left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-f}{s}+1\right) \tag{5.12}$$

Each convolutional layer can have any number of randomly initialized filters. Different filters containing different weights emphasize different features, e.g., horizontal lines, vertical lines, edges, corners, and more abstract (unintuitive) features. Before the onset of CNNs the weights of these filters were hand-engineered to emphasize the desired features, such as vertical lines. However, the CNN learns the features itself and learns



(a) The convolutional operation using a filter size $f = 3$, stride $s = 1$

(b) The *same* convolutional operation using a filter size $f = 3$, padding $p = 1$ and stride $s = 1$ for the top image and $s = 2$ for the bottom image

Figure 5.4: Graphically illustrating the convolutional operation

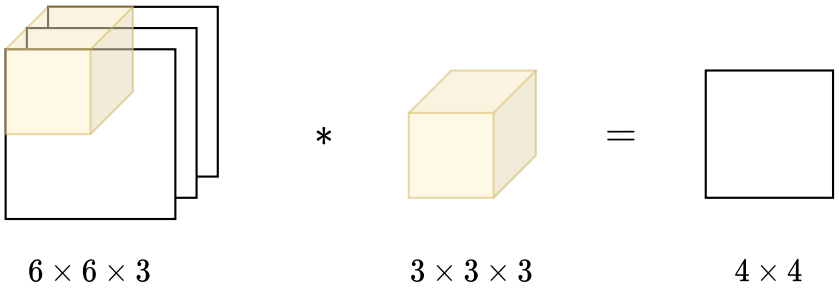


Figure 5.5: Graphically illustrating the 3D valid convolution operation for an example input size of $6 \times 6 \times 3$

the best filters with the corresponding weights (feature extraction) to properly represent the data and perform well on the designed task.

Convolutions over volumes

The input can have multiple channels and as such the filter also has to be 3D, meaning that the filter is a cuboid that slides across the height, width, and depth of the input feature maps. The 3D convolution operation is similar to the 2D convolution and is graphically shown in Figure 5.5, where the 3D convolution operation produces a 2D feature map. As previously mentioned, multiple convolutional filters are used to detect different features. The different 2D feature maps resulting from the 3D convolutional operations are stacked together, resulting in 3D feature maps as shown in Figure 5.8. Therefore, for a given convolutional layer l for an input image of $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$, the filter size will be $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$, and the resulting output size will be $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$, where $n_H^{[l-1]}$, $n_W^{[l-1]}$, $n_c^{[l-1]}$ refer to height, width, and number of channels of the input image to that convolutional layer l, respectively, and $n_H^{[l]}$, $n_W^{[l]}$, $n_c^{[l]}$ refer to the height and width of the output 2D feature map and to the number of filters used, respectively. The number of filters used in a convolutional layer is a *hyperparameter* (Section 5.5).

Advantages of convolutional layers

There are two advantages to the usage of convolutional layers:

- **Parameter sharing**: The different filters detect different features and thereby result in different feature maps. However, a filter that detects vertical edges that is useful in one part of the image is also useful in another part. This means that once the weights of the filter are determined, that same filter can be applied to every part of the image. The total number of parameters for six filters of 3×3, with each filter having nine weights and one bias term, would be only 60 parameters. Whereas, for traditional neural networks each pixel has a certain weight associated with it, resulting in a weight matrix containing

millions of parameters for even very small images (e.g., $32 \times 32 \times 3$)

- **Sparsity of connections**: Through the convolutional filter operation, the output value in the top-left corner of the output feature map only depends on the $f \times f$ pixels in the top-left corner of the original image as illustrated in Figure 5.4. It can be viewed as if only these 9 pixels are connected to the output. This aspect not only decreases the size and thus the memory requirements of the model, but is also reduces the computational effort required to determine the output.

Pooling layer

A pooling layer is used to decrease the size of the feature map and as such decreases the computational effort required to process the data while preserving the relevant information. The pooling operation works in a similar fashion to convolution as it slides a filter of size f, with stride s, over the feature maps. The filter contains a pooling function, which can be *max* or *average* pooling, with *max* pooling being the most commonly used function. The intuition behind it is that a large number within the feature map indicates the presence of a certain feature (e.g., vertical edge), therefore, if the feature is present in one of the quadrants it will be preserved through the use of *max* pooling. The size and stride of the pooling filter are *hyperparameters* and these are fixed and will not be learned through gradient descent. The concept of *max* pooling is graphically shown in Figure 5.6, using a filter size $f = 2$ and stride $s = 2$.

5.4. Lightweight networks

The majority of state-of-the-art deep learning networks typically have millions of parameters and require significant amounts of processing power. This places constraints on the required memory and computing power of the processor to be able to run these networks. However, recently the trend is moving towards designing so-called *lightweight networks*, which are networks that can be used on mobile and embedded devices. They are characterized by a substantial decrease in *computational cost* as well as *number of parameters* compared to traditional deep learning architectures. The building blocks of these lightweight networks will be discussed as they are the basis of the networks discussed in Chapters 8 and 9.

Depth-wise separable convolution

The seminal work by Howard et al. (2017) created the MobileNet architecture, which introduced the concept of *depth-wise* separable convolution, which replaces normal convolution thereby drastically reducing the number of parameters in the network, resulting in sped-up performance and less necessary memory. The concept of *convolution* was explained in Section 5.3 and the main concept is that the kernel/filter is applied over all the channels (Figure 5.7 (a)), where the channel refers to the input's depth. A RGB image has three channels, one channel corresponding to each color. The *depth-wise* separable convolution on the other hand consists of two steps, namely the *depth-wise* convolution and the *point-wise* convolution. *Depth-wise* convolution applies one filter per channel as graphically illustrated in Figure 5.7 (b) and *Point-wise* convolution applies a 1×1 filter over all the channels as graphically shown in Figure 5.7 (c). This means that the number of filters for *depth-wise* convolution is equal to the number of channels of the input n_c, whereas the number of filters of the *point-wise* convolution $n_c^{'}$ determines the output size. This is also illustrated in Figure 5.8. *Depth-wise* separable convolution reduces the computational effort by a factor of 8 to 9 compared to regular convolution, with only a minor decrease in accuracy (Howard et al., 2017). This is shown in Table 5.1, which shows that by using depth-wise separable convolution compared to standard convolution the number of parameters (memory usage) and FLOPs (computational effort) have been decreased by 86% and 88%, respectively, at the cost of only a 1.1% decrease in performance. The chosen network architectures in Chapters 8 and 9 use *depth-wise* separable convolution.

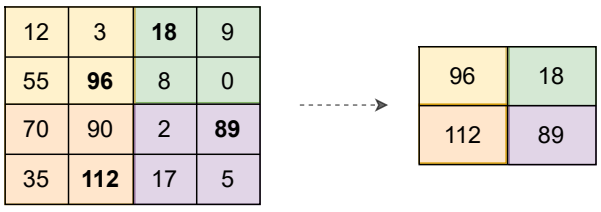


Figure 5.6: Graphically illustrating the concept of 2×2 max pooling

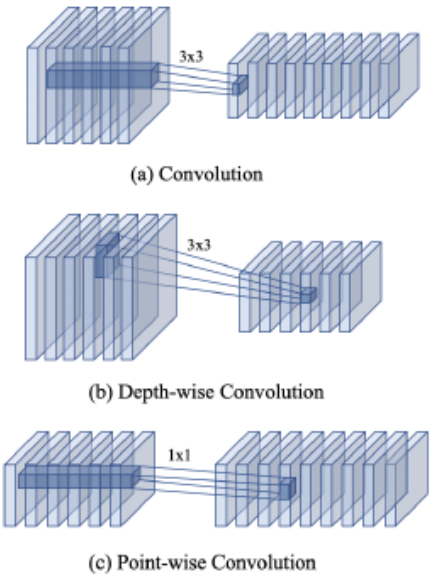


Figure 5.7: Figure graphically showing the difference between standard convolution and depth-wise and point-wise convolution (Park et al., 2019)

Table 5.1: Table showing the comparison between a MobileNet architecture that uses standard convolutions (Conv MobileNet) and depth-wise separable convolution (MobileNet) on the ImageNet benchmark (Howard et al., 2017)

Model	**ImageNet Accuracy** [%]	**Parameters** [Mn]	**FLOPs** [Bn]
Conv MobileNet	71.7	29.3	4.9
MobileNet	70.6	4.2	0.6

Bottleneck blocks

The MobileNet architecture has been improved upon, resulting in the MobileNet-V2 architecture proposed by Sandler et al. (2018), introducing two main changes compared to the initial architecture. The changes are an addition of a residual connection (skip-layer) inspired by ResNet (He et al., 2016) and the addition of an *expansion* layer before the *depth-wise* separable convolution. The name of the resulting lightweight block is the (lightweight) *bottleneck* block, which is graphically illustrated in Figure 5.8. The residual connection feeds information from the l^{th} layer in the neural network directly to the $l+2$ layer, changing the resulting output value of that layer to $a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$. The general notion is that the deeper the network the more complex the function it can represent, however, the training of these very deep networks is plagued by vanishing and exploding gradients. The residual connections proposed by He et al. (2016) offer a solution to this problem. The residual connection allows the layer to easily learn the identity function in comparison to a normal neural network thereby guaranteeing that the addition of layers will not hurt the performance and can potentially increase the performance.

The *expansion* layer uses *point-wise* convolution to increase the size of the representation by using more filters, whereas the aforementioned approach of *point-wise* convolution is now dubbed *projection* as it *projects* the output size back to smaller values. The advantage of adding the expansion layer is that the neural network can learn more complex functions, because of the increased size of representation, whereas the advantage of subsequently using projection is that you scale down the number of parameters that are transferred to the next block. Thereby reducing the amount of memory needed to store the parameters. This *bottleneck* block is considered a building block of lightweight networks and is used in a variety of other architectures as well as the LPN network proposed by Zhang et al. (2019), which is discussed in Chapter 9.

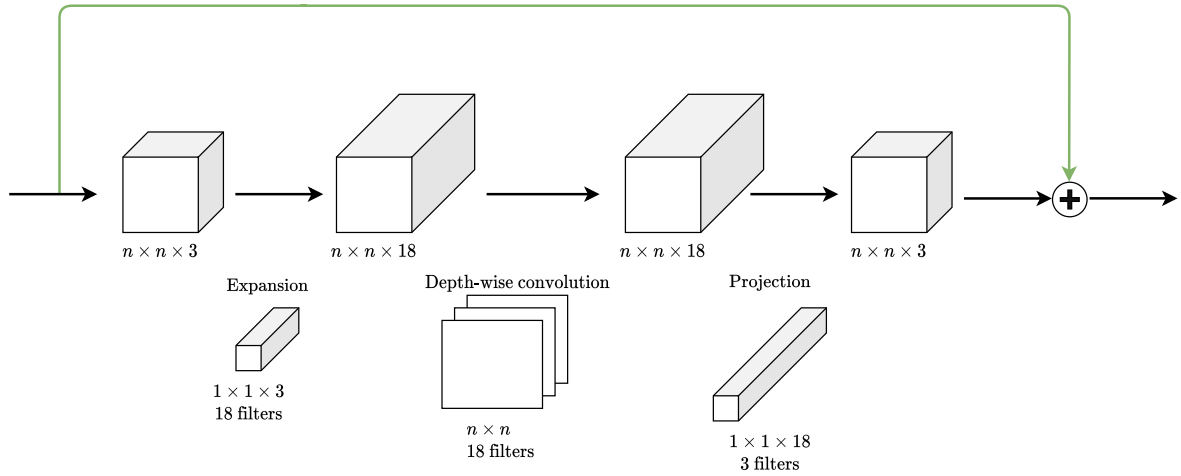


Figure 5.8: The lightweight bottleneck block, which consists of four distinct parts, namely the residual connection (green), the expansion layer followed by the depth-wise convolution and the projection layer

5.5. General machine learning concepts

This section briefly discusses several concepts and definitions that are often used when discussing machine learning models. This serves to better understand the discussion in Chapters 8 and 9 regarding the networks. The following concepts will be discussed:

- Hyperparameters
- Generalization
- Regularization
- Mini-batches
- Inference
- Transfer learning
- Network size and computational effort

Hyperparameters

Neural networks consists of parameters and hyperparameters, where the parameters are the learnable weights and biases (w, b) and the hyperparameters refer to parameters that influence those weights and biases, such as the learning rate α, the number of epochs (training steps), the number of layers, the number of filters per layer, and the mini-batch size. These hyperparameters are not learned by the network and have to be manually tuned to achieve the best possible performance of the network.

Generalization

As in line with non-machine learning models, the model is trained on certain data and then its performance is tested on unseen data to evaluate how well the trained model generalizes. During training the goal is to find a model that performs well on the training dataset (low bias), while also performing well on the validation dataset (low variance). This is relative to the best possible performance that a network can theoretically achieve, *Optimal Bayes error*. A high training error (high bias) indicates *underfitting*, which means that the function is too simple to explain the underlying patterns in the data, whereas *overfitting* means that the function is too complex and fits every data point in the training set very well. A large gap in performance of the model on the training set and the validation set (high variance) could indicate overfitting. This can be decreased by using more training examples and using regularization techniques. The concepts of overfitting and underfitting are strongly related to the *capacity* of the model, which refers to the ability to estimate complex functions. The more neurons and layers a neural network has, the higher the capacity.

Regularization and data augmentation

There are techniques that can used to reduce overfitting and increase the generalization performance, i.e., reduce the generalization gap (reduce the variance). The variance can be reduced by using more training data

or by applying regularization techniques. The most common regularization technique is $L2$-regularization and this is also used in the chosen object detection network (Chapter 8). This approach adds a penalty term that is scaled by the regularization parameter λ to the cost function $J(w,b)$ as shown below, where m is the number of training examples and w is the weight matrix. Because w is a matrix, $\left\| w^{[l]} \right\|_2^2$ represents the *Frobenius norm*.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathscr{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \left\| w^{[l]} \right\|_2^2 \tag{5.13}$$

The intuition is that by having a large value for λ, the weights will be forced to be small (≈ 0) as otherwise the value of the cost function increases. This results in the deactivation of a lot of units and thereby resulting in a simpler model, and as such the likelihood of overfitting decreases. However, using a too simple model could result in underfitting. The choice of λ is therefore crucial in determining the optimal model and preventing over/underfitting (Goodfellow et al., 2016). Another method of regularization that is commonly used is *dropout regularization*, which refers to randomly selecting certain neurons to be ignored during a forward or backward pass. For each training step a different random set of neurons is ignored to avoid the network to rely too much on certain weights. Furthermore, by ignoring certain nodes the network becomes smaller and this moves the network away from overfitting.

Another important tool in reducing the variance is *data augmentation*, which refers to expanding the dataset by creating modified copies of samples of the training set by flipping, random cropping, and rotating the original images. These types of augmentations are called *affine augmentation*, whereas random brightness or contrast changes applied to the images are referred to as *pixel-level* data augmentations. The larger dataset in itself helps to improve the generalization performance, as the model has more information it can learn from. Moreover, the model will also learn to become invariant to these induced brightness/contrast changes or scaling and rotations. Examples of augmentations are given in Figure 7.19.

Mini-batches

The regular gradient descent, also referred to as Batch Gradient Descent (BGD), uses the complete training set to compute the gradient of the loss function after which the weights and biases are updated and the process starts again. As the entire dataset is used to update the model, this is an accurate, but time-consuming method that takes a long time to converge and is not suitable for large datasets. The concept of using *mini-batches*, also known as Mini-Batch Gradient Descent (MBGD), refers to randomly splitting up the training set into smaller parts that are then used to compute the gradients of the loss function and consequently update the weights and biases. The advantage of using MBGD is that progress can be made without having to process the entire dataset, as it only processes n examples at a time. The process of the usage of mini-batches is shortly discussed below, for k *mini-batches*: For $k = 1, ..., k$

1. Forward propagation on $X^{\{k\}}$ for k mini-batches.
2. Compute the cost function $J(X^{\{k\}}, Y^{\{k\}})$
3. Backward propagation to compute the gradients $\mathrm{d}w, \mathrm{d}b$ and update the weights and biases

Running through all the *mini-batches* in the training set is called an *epoch* and networks are trained for multiple epochs. When the *mini-batch* size is equal to the number of training examples m, the method reduces to BGD. When the *mini-batch* size is equal to one, the method reduces to Stochastic Gradient Descent (SGD), meaning that every training example is its own mini-batch. However, in practice the *mini-batch* size lies somewhere between these two extremes and generally the *mini-batch* size is a power of 2 for efficiency reasons, e.g., $2^5 = 32$. A larger *mini-batch* size generally result in more accurate movement towards the minimum, albeit at a slower pace, as more data is processed before an update step is made. Selecting the ideal mini-batch size for a given problem can be based on literature, similar problems, or through trial and error. However, the upper limit of the mini-batch size is determined by the GPU memory.

The disadvantage of *mini-batches* is that they do not always exactly converge to the minimum and they can oscillate around it, however, this can be ameliorated by using a smaller learning rate α or decaying learning rate over time.

The practical implication of using mini-batches is that the cost function data is more noisy, because some mini-batches might be easier for the network to process than others, resulting in the value of the cost function being different between batches. However, the trend of decreasing costs over the epochs remains the same.

Inference
After a machine learning model has been trained, it can be used for *inference*. Inference refers to the situation in which the trained model is fed (new) data and creates actionable output. An example would be a trained cat/dog classification model that is fed an image containing a cat and that it then outputs a probability that the image contains a cat or a dog.

Transfer learning
The training of a CNN requires an extensive dataset of images and a lot of computational resources. However, initializing the network with random weights and biases results in starting the training with a model that has no knowledge whatsoever. The concept of transfer learning refers to the usage of the weights and biases of a model trained to a certain dataset, i.e., pre-trained model, to initialize the actual network. This concept relies on the fact that the earlier layers of the CNN detect low-level features, such as corners, edges, and lines, whereas the later layers detect semantically stronger higher-level features, such as a wheel or face. These low-level features are generic and relevant for any dataset and allow the network to leverage this knowledge for the new dataset. The concept of transfer learning is synonymous to how humans learn, where the knowledge gained by riding a bike can be used to ride a motorcycle.

The concept of transfer learning is almost always applied for computer vision problems (Barad, 2020; Sharma, 2019; Sun et al., 2019; Xiao et al., 2018). Two types of transfer learning can be discerned, namely *finetuning* and *feature extraction*. *Finetuning* refers to the process of using a pre-trained model's weights and biases to initialize the new model, and then retraining the entire network on the new task, i.e., updating all of the model's parameters. Whereas *feature extraction*, refers to the fact that the pre-trained model's weights and biases are used to initialize the new model and these values are then kept fixed and only the final layer's weight and biases are updated using the new dataset. By applying transfer learning the model has already learned some aspects and this approach generally produces favorable results in terms of overall accuracy and training time compared to a model that is trained from scratch, i.e., weights and biases randomly initialized[5]. Moreover, it allows complex networks to be adapted to the new purpose while using relatively small datasets.

Network size and computational effort
There are a myriad of different CNNs that use different numbers of convolutional and pooling layers. The performance of these networks on challenging datasets, such as COCO (Lin et al., 2014), are often published, however, the size and the computational efficiency of the network is equally important.

Comparing the execution time does not give a good indication as running the network on high performing GPU's or slow CPU's would give different results. Therefore, to accurately compare the efficiency of different networks two parameters are used, namely the *number of parameters* and *FLOPs*, also known as *Multiply-accumulate operation (Mult-Adds)*

- **Number of parameters:** This defines the size of the network and includes the parameters of the convolutional layers and fully connected layers, as pooling layers have no learnable parameters. The number of parameters can be viewed as a proxy for the amount of memory needed to run the model, i.e., the weights and biases need to be stored on the device. The state-of-the-art image classifying CNNs typically have between five to a hundred million parameters.
- **FLOPs** or **Mult-Adds**: This parameter represents the computational efficiency of the network. It represents the total amount of floating point operations that are necessary to process the image through all the layers of the network during a forward pass. The state-of-the-art CNNs typically have between one to a hundred billion FLOPs, therefore FLOPs are measured in billions. The lower the FLOPs, the more efficient the network.

[5] `https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html` Date accessed: 2-11-2021

III

Algorithm Design & Methodology

6

Algorithm's architecture overview

This chapter shows a top-level description of the pose estimation pipeline developed in this work. This serves as an overview of how all the different parts fit in and which software is used and developed for each respective part. The architecture and software overview are discussed in Sections 6.1 and 6.2, respectively.

6.1. Architecture overview

As discussed in Section 2.1, there is a variety of approaches that can and have been used to estimate the pose of objects or humans. The approach used in this work is a model-dependent feature-based learning approach, as this was identified as the most promising method in Section 2.1. This means that a CNN is used to detect pre-defined keypoints on the 3D model within a 2D image after which the 2D-3D correspondence can be used to estimate the distance to the asteroid by solving the PnP problem. The feature detecting CNN replaces traditional hand-engineered IP algorithms. This feature-based approach followed by a separate pose estimation step outperformed end-to-end CNN architectures in which a single CNN replaced the entire pipeline and thereby learning the complex non-linear relation between the 2D image and the pose directly (Sharma and D'Amico, 2019).

The advantage of using a CNN-based approach is that the keypoints can be selected offline and as such their 2D-3D correspondence is known, avoiding the cumbersome and computationally intensive matching of the detected 2D features to their location on the 3D model using methods such as RANSAC (Park et al., 2019). Furthermore, the keypoint-based CNNs output predicted detections for all the designated keypoints, even when they are occluded or not directly visible (e.g., on the back of the target) (Zhao et al., 2018). This is because it learns the inherent spatial relationship between the different keypoints. A CNN-based feature detector is also proven to be more robust against illumination conditions and image noise compared to traditional IP algorithms, which is crucial for the usage in a vision-based navigation system (Pasqualetto Cassinis et al., 2021a).

Figure 6.1 illustrates the pose estimation framework and the facets of the CNN-based feature detector, and how this would fit within a navigation framework. A top-down approach is used, which consists of an object detection network in front of the keypoint detection network, making the CNN pipeline more robust to the scale of the object within the image, allowing for accurate keypoint detections (Pasqualetto Cassinis et al., 2021a). Moreover, it is the state-of-the-art approach used within keypoint detection for human pose estimation, terrestrial objects, and uncooperative spacecraft (Kisantal et al., 2020; Pasqualetto Cassinis et al., 2021a; Zhao et al., 2018).

This OD network detects the object within the image and regresses the bounding box coordinates encompassing the object. This bounding box is used to crop the RoI of the original image. This RoI image is resized to match the input size of the keypoint detection network, which convolves the input image and outputs heatmap predictions around the pre-selected keypoints. The 2D pixel coordinates of the keypoint location correspond to the heatmap's peak intensity, where the shape and the intensity characterize the confidence of detecting the corresponding keypoint at that location (Pavlakos et al., 2017). This heatmap approach resulted in better performance compared to the direct regression approach and it allows for the extraction of statistical information (covariance matrix) regarding the detection uncertainty (Pasqualetto Cassinis et al., 2020). However, this step is not performed within this work due to time-limitations, but the method is described

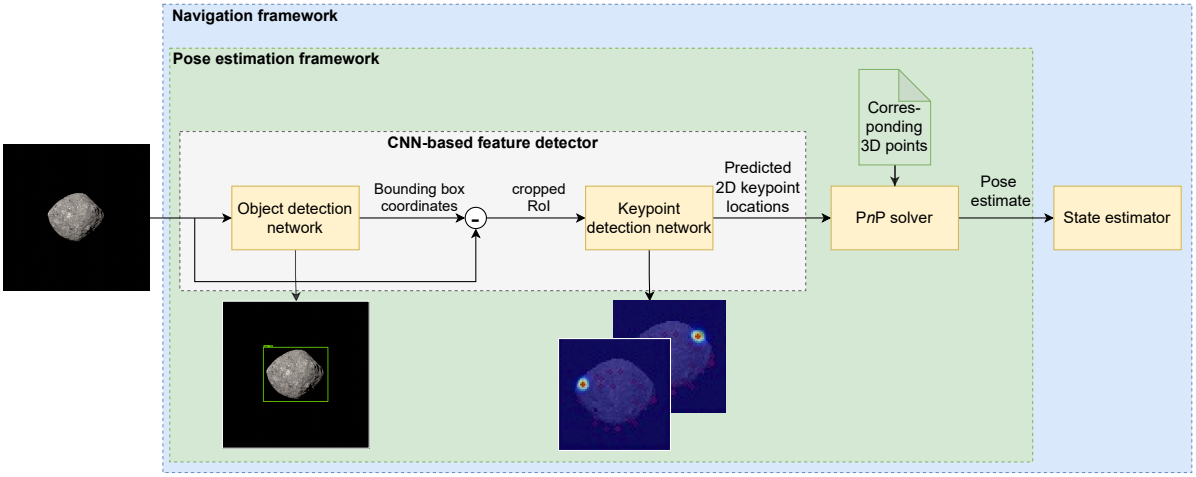


Figure 6.1: Schematically illustrating the different parts of the pose estimation framework that is designed within this work and how this fits within a larger navigation framework

in Appendix A. These networks have to be trained using large-scale datasets and due to the unavailability of datasets of actual space images of the target, synthetic images are to be used. The creation of this synthetic dataset is explained in detail in Chapter 7. The selection, implementation, configuration, training, and evaluation procedures of the respective networks are discussed in Chapters 8 and 9, respectively.

The detected 2D keypoints are then fed to the PnP solver alongside their corresponding 3D locations on the target. The EPnP solver developed by Lepetit et al. (2009) is selected for use in this work, which is available through the Open Source Computer Vision Library (OpenCV), allowing for ease of application. Therefore, no separate chapter is required detailing the implementation. The basic workings are explained and the choice is motivated within this section.

The EPnP solver is a non-iterative approach to solve the PnP problem. The main idea of the EPnP solver is to rewrite the 3D points as a weighted sum of four different non-coplanar virtual control points, c_j, where $j = 1,2,3,4$, presented in the asteroid reference frame (Figure 4.2). These 3D points, r_i^A, are expressed in barycentric coordinates, α_{ij}, of these virtual control points as shown below. The coordinates of these virtual control points are arbitrary.

$$r_i^A = \sum_{j=1}^{4} \alpha_{ij} c_j \tag{6.1}$$

This allows Equation (4.9) to be rewritten in terms of the vector $\hat{\mathbf{x}}$ containing the 12 unknown control point coordinates in the camera frame, resulting in a linear system $\mathbf{M}\hat{\mathbf{x}} = 0$ that is solved to find these unknowns. Where matrix $\mathbf{M} \in \mathbb{R}^{2n\times 12}$ is a known matrix for n 2D-3D correspondences.

This method proved a significant improvement with regards to computational complexity, $O(n)$, with little to no loss of accuracy while being less sensitive to noisy 2D feature locations compared to other non-iterative pose solvers. This makes it more suitable for application on an embedded device. Several other state-of-the-art PnP solvers have $O(n^5)$ or $O(n^8)$ (Lepetit et al., 2009). The EPnP solver is applicable to non-coplanar as well as coplanar 3D model points and it requires a minimum of four ($n \geq 4$) 2D detections with their corresponding 3D model points. However, under the influence of noisy 2D detections, $n \geq 6$, is desired. Furthermore, the EPnP solver achieves accuracies comparable to iterative approaches at a fraction of the computational cost and without requiring an initial estimate (Lepetit et al., 2009). The EPnP solver has also been used by Park et al. (2019) and Pasqualetto Cassinis et al. (2021a) for uncooperative spacecraft pose estimation and by Magalhães Oliveira (2018) for feature-based navigation around the Moon.

This CNN-based pose estimation pipeline can be incorporated into either a *tightly-coupled* or *loosely-coupled* approach, where the former refers to feeding the extracted keypoint directly to the navigation filter as measurements and the latter refers to firstly calculating the pose using a PnP solver after which these parameters are fed to the navigation filter as pseudomeasurements. The CNN-based approach guarantees detections of all pre-defined keypoints. Therefore, it overcomes challenges faced by traditional feature-tracking approaches for which the number of detections can deviate, resulting in highly-variable measurements sent

Table 6.1: An overview of the different parts of the algorithm architecture and the respective software used for each part

Part	Main functionality	Software
Dataset creation (Chapter 7)	1. Image rendering	Python, Blender (Python API), Google Colab (Python)
	2. Textured asteroid model creation	Blender
	3. Keypoint designation	Point Cloud Library (PCL) (C++)
	4. Dataset annotation	Python
	5. Corrupted images dataset creation	Python
Object detection network (Chapter 8)	Detecting the asteroid within the image	TensorFlow (Python), Google Colab (Python)
Keypoint detection network (Chapter 9)	Detecting the keypoints/features/landmarks on the asteroid surface	PyTorch (Python), Google Colab (Python)
EP*n*P	Output the pose estimate using the detected 2D points and their 2D-3D correspondences	OpenCV (Python)

to the filter. Moreover, a covariance matrix quantifying the uncertainty of the detections (measurements) derived from the heatmaps can be used to improve the filters robustness. These two aspects would allow for the incorporation of this pipeline in a *tightly-coupled* navigation architecture, which is normally preferred over a *loosely-coupled* architecture, as it allows for better convergence and does not suffer from colored noise.

6.2. Software overview

The different components and their main top-level functionality alongside the software/language that is used to perform that functionality are shown in Table 6.1. This serves as an overview and the respective parts and their design choices will be discussed in more detail in Chapters 7 through 9.

The dataset was created using Blender[1], which is a powerful open-source and free 3D modeling software, which can be used to create 3D models and render images from 3D scenes through a Python API. Furthermore, the 3D SIFT algorithm used to designate the keypoints on the asteroid is available in the open-source Point Cloud Library (PCL)[2]. This library contains a variety of algorithms suitable for point cloud processing and analysis. As aforementioned, the EP*n*P solver is available through OpenCV[3]. This library contains more than 2,500 optimized algorithms for computer vision applications and is extensively used in academia and industry, by renowned companies, such as Google, Intel, and Microsoft.

The best practice within the computer vision community is to use an open-source implementation of a network architecture that has been optimized for a given task, e.g., object detection or keypoint detection. The advantage of this is that such a network is often faster and better compared to implementing it from scratch based on some research paper. Furthermore, such a network has already been extensively verified. Another advantage is that new applications generally tend to have limited datasets and by using an existing architecture *transfer learning* can be applied, resulting in better performance regarding accuracy and computational effort as discussed in Section 5.5

The object and keypoint detection networks, which will be discussed in detail in Chapters 8 and 9, are implemented in TensorFlow[4] and PyTorch[5], respectively. These are open-source machine learning platforms built and maintained by Google Brain and Meta AI Research, respectively, which use a Python API. They are used by many around the world in academia and industry alike to create machine learning models.

The rendering of the images and the training and evaluation of the networks was performed using Google Colaboratory (Colab)[6], which allows to write and execute Python code in a browser and provides access to powerful GPUs. The NVIDIA Tesla P100-PCIE-16GB GPU was used and the data is transferred between Google Colab and the local machine through Google Drive.

[1] `https://www.blender.org/`, Date accessed: 17-01-2022
[2] `https://pointclouds.org/`, Date accessed: 7-10-2021
[3] `https://opencv.org/`, Date accessed: 7-10-2021
[4] `https://www.tensorflow.org/`, Date accessed: 22-01-2022
[5] `https://pytorch.org/`, Date accessed: 22-01-2022
[6] `https://colab.research.google.com/notebooks/welcome.ipynb?hl=en`, Date accessed: 21-01-2022

7

Dataset

This chapter discusses the dataset that has been generated within this work, going into detail regarding the design choices and the processes applied in generating this deep-learning dataset. The creation of a realistic asteroid dataset suitable for training machine learning models also allows for future training and testing of deep learning-based navigation systems. Firstly, in Section 7.1 a top-level overview of the entire dataset generation and annotation process is given. The model-agnostic image generation pipeline developed within this work will then be discussed in more detail in Section 7.2, touching on the rendering software used, the created textured asteroid model, the camera viewpoints, and the rendering process. The properties of the dataset created using the devised settings are summarized in Section 7.3. The annotation of the dataset, involving matching the ground-truth information to the images is discussed in Section 7.4. The problem of bridging the gap between synthetic and real images is addressed in Section 7.5 and a new dataset is introduced that aims at tackling this problem. Section 7.6 discusses the three different image sequences that are generated in addition to the two datasets. The section is concluded with a brief summary regarding the dataset API created in this work in Section 7.7, allowing for ease of processing the Bennu datasets and to allow for the generation of other datasets through the created pipeline in this work. Furthermore, Appendix B discusses two deep-learning datasets that exist for space-borne applications, namely the SPEED and Envisat datasets, which are related to uncooperative spacecraft pose estimation. This is intended to give an overview of what a deep-learning dataset suitable for space-borne applications entails.

7.1. Dataset generation overview

This section discusses the major steps of the dataset generation process and serves as a top-level overview, where detailed information is provided in each respective (sub)section. The top-level overview is given in Figure 7.1. As can be seen in Figure 7.1, the dataset generation can be subdivided into two major facets, namely the image generation pipeline, which creates the rendered images using the desired settings, and the annotation of the dataset. The model-agnostic image generation pipeline (Section 7.2) consists of three parts, namely the viewpoint sampling, the creation of a textured 3D model, and the rendering of the images itself. Once all the images of the dataset have been generated, the dataset can be annotated with the information required for training the neural networks. The different parts and their main purpose are discussed below:

1. **Creation of a textured 3D model** (Subsection 7.2.2): This creates the textured 3D model by using a 3D shape model of the asteroid Bennu and applying a realistic texture using Blender. This textured 3D model is required for the image generation. This process is independent of the other parts and the resulting textured 3D model is used within the *rendering* process.

2. **Viewpoint sampling** (Subsection 7.2.3): This creates the different camera poses, distance and orientation, w.r.t the target asteroid. This is used to place the camera within the 3D rendering software Blender and as such create a diverse dataset consisting of a variety of viewpoints. The output of this process consists of `.csv` files containing the camera pose w.r.t. the world reference frame (W), i.e., $[\mathbf{q}_{B/W}^{W} \mid \mathbf{r}_{B/W}^{W}]$. These camera poses are irrespective of the asteroid orientation.

3. **Rendering** (Subsection 7.2.4): The rendering step uses the camera poses determined in the *viewpoint sampling* part and places the camera within Blender. Furthermore, it places the textured 3D asteroid

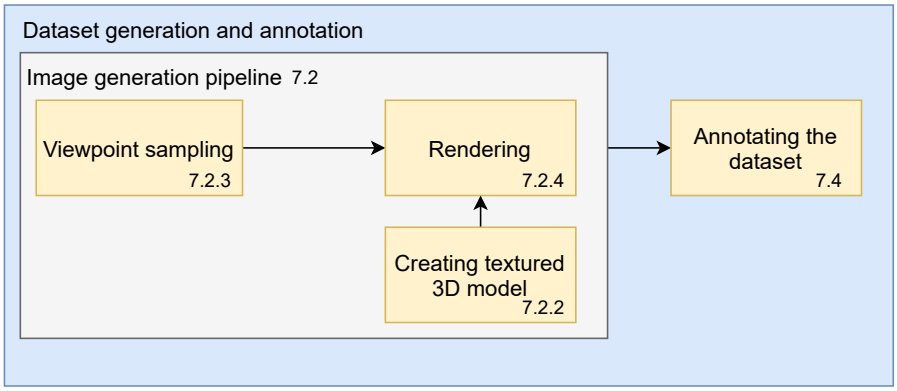


Figure 7.1: Top level overview of the major parts that make up the dataset generation and annotation process, where each (sub)section corresponding to each part is listed

model within the scene and renders the images using the camera poses, asteroid orientations, and render settings. The output of this process is the rendered images that together make up the dataset.

4. **Annotating the dataset** (Section 7.4): The annotation step revolves around matching the correct information with the correct image within the required format. This relies on the relevant information from the image generation pipeline. The output of this process consists of `.json` and `.csv` files that contain the bounding box, keypoints and camera poses corresponding to each image, which are used to train the neural networks.

7.2. Image generation pipeline

This section discusses the image generation pipeline that was created in this work. This image generation pipeline consists of various facets, as discussed in Section 7.1 and shown in Figure 7.1, which will be discussed separately in the respective subsections. These parts were implemented to create the realistic asteroid dataset that was used for the training and evaluation of the machine learning models. The implementation of this pipeline is agnostic to the target object and can therefore be used to generate deep-learning datasets of several targets different from the one used in this work.

7.2.1. Rendering software

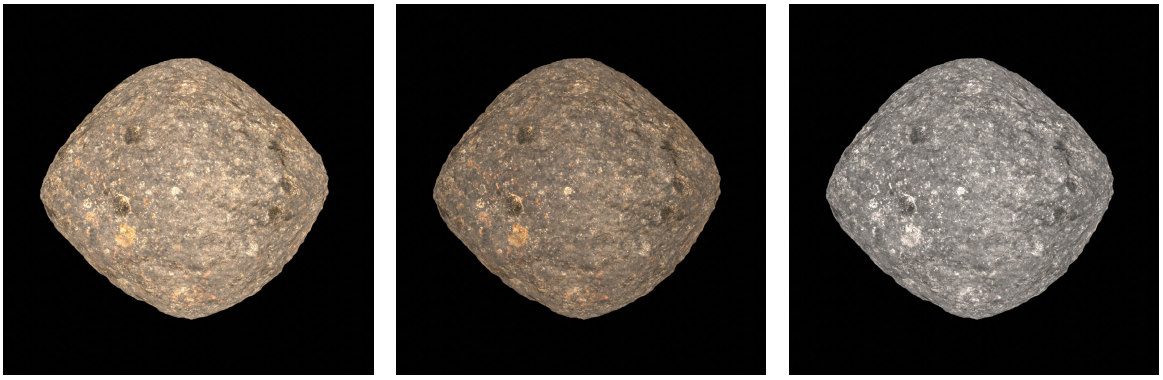
There are several rendering software that can be used for the creation of synthetic images, however, the majority of them are not open-source nor free. Specialized software, such as Planet and Asteroid Natural Scene Generation Utility (PANGU)[1] developed by ESA or SurRender developed by Airbus (Brochard et al., 2018), is used to model surfaces of planetary bodies, such as the Moon and Mars, and to model asteroids. However, these software are not open-source nor free and their efficacy in large-scale dataset generation for deep-learning purposes has not been investigated. Therefore, these are not considered further.

As discussed in Section 2.1, Cinema4D and Blender have been used by Pasqualetto Cassinis et al. (2020) and Black et al. (2021), respectively, to generate synthetic image deep-learning datasets of a target spacecraft, demonstrating the suitability of these rendering software for deep-learning dataset generation. However, the Cinema4D software is not free to use. Blender is open-source and free and has been used in other works revolving space applications, such as feature-based landing on the Moon (Magalhães Oliveira, 2018), feature-based navigation around asteroids (Volpe et al., 2020), and uncooperative spacecraft pose estimation (Harvard et al., 2020). Therefore, the Blender software [2] was selected for the generation of images within this work.

7.2.2. Target asteroid model

As discussed in Section 2.3, the target asteroid for this mission is Bennu, as the 3D model is publicly available and a lot of literature surrounding the object is present. There are several different 3D models available of

[1] `https://pangu.software`, Date accessed: 17-01-2022
[2] `https://www.blender.org/`, Date accessed: 17-01-2022



(a) Raw rocky texture

(b) After the application of noise texture, and contrast and brightness changes

(c) Final textured model after applying Hue saturation

Figure 7.2: Graphically illustrating the steps to arrive at the final textured asteroid model (c)

Bennu, as it is common for asteroid missions that the shape model becomes more refined while the mission progresses. For the OSIRIS-REx mission the initial shape model was based on radar observations and it did not contain the detailed surface topography. During the approach of the spacecraft towards the asteroid a preliminary shape model was created using a compilation of images taken by the PolyCam camera. This 3D model shows features as small as six meters. The highest resolution model was created after studying the asteroid for three months at a close range. This 3D model shows features smaller than one meter. During different mission phases, different 3D models are available. The most detailed shape model is used in this work, however, any other model would result in the same process. The most detailed publicly available shape model of Bennu has a 75-centimeter resolution (v20) [3], i.e., resolution of 75 cm per vertex. The dimensions of the 3D model are given in km. However, this 3D model does not have any texture applied, which is required for the image generation.

As of 26 June 2021, a normal albedo map of Bennu is publicly available[4]. A normal albedo map represents the innate reflectance of the surface and has zero phase angle, i.e., it is an image texture without any shadows or highlights. However, this albedo map only covers the region between $\pm$ 55° latitude. Moreover, the images that were used in generating this albedo map may contain noise and other artifacts. Because of the incompleteness of the albedo map it would be required to use the global basemap mosaic [5] for the higher latitudes. However, this global basemap was designed to highlight the surface morphology and has a phase angle of 30°. It therefore has shadows and highlights baked into the image. When such an image is used as the image texture, the object would then have two shadows, one caused by the synthetic light source and one that was already baked into the image. Therefore, when synthetically rendering images an albedo texture map is vital to avoid unrealistic synthetic images.

Consequently, to avoid unrealistic scenes, a texture was created and applied to the 3D model in Blender. The 3D model is imported to Blender using the +Y-forward and +Z-up convention, which determines the initial orientation of the asteroid, i.e., which side is illuminated. A rocky texture was downloaded from Poly-Haven [6] and adjustments to this standard texture, shown in Figure 7.2a, were made in Blender to mimic an actual surface of an asteroid. A noise texture is applied to randomize the rocky texture and mimic white and dark spots randomly spread over the surface. Furthermore, contrast and brightness changes were applied. These adjustments changed the rocky texture from Figure 7.2a to 7.2b. The final textured asteroid model can be seen in Figure 7.2c and was achieved by applying Hue saturation to the texture of Figure 7.2b, which changes the saturation and brightness [7]. The specific settings can be found in the `BennuTextured.blend` file that is made available on Github by the author [8].

[3] `https://www.asteroidmission.org/updated-bennu-shape-model-3d-files/`, Date accessed: 16-12-2020

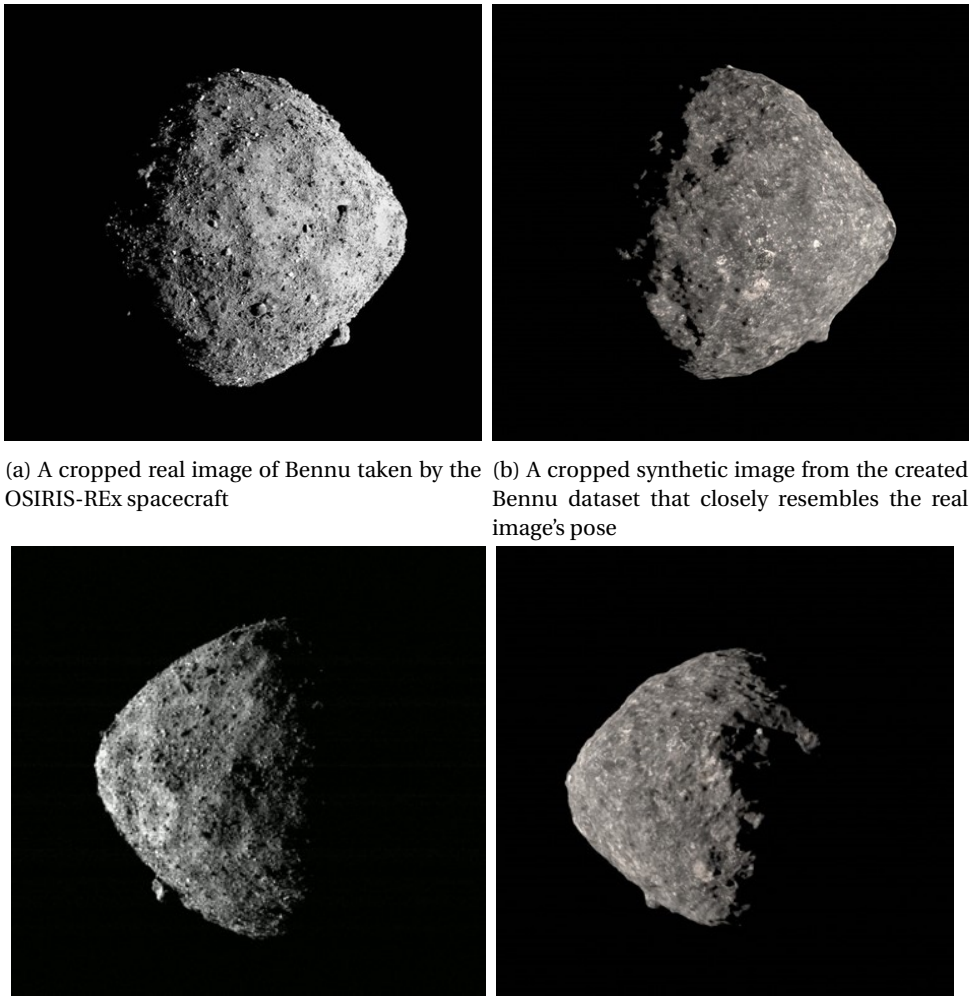
[4] `https://astrogeology.usgs.gov/search/map/Bennu/OSIRIS-REx/OCAMS/Bennu_global_ShapeV20_GndControl_ROLOphase_ALBEDO_8bit_v6`, Date accessed: 16-8-2021

[5] `https://www.asteroidmission.org/bennu_global_mosaic/`, Date accessed: 16-8-2021

[6] `https://polyhaven.com/textures/rock/natural`, Date accessed: 16-8-2021

[7] `https://docs.blender.org/manual/en/latest/compositing/types/color/hue_saturation.html`, Date accessed: 17-01-2022

[8] `https://github.com/lvanderheijden`



(a) A cropped real image of Bennu taken by the OSIRIS-REx spacecraft

(b) A cropped synthetic image from the created Bennu dataset that closely resembles the real image's pose

(c) A cropped real image of Bennu taken by the OSIRIS-REx spacecraft

(d) A cropped synthetic image from the created Bennu dataset that closely resembles the real image's pose

Figure 7.3: Comparison of real images taken by OSIRIS-REx and synthetic images generated within this work

The final textured asteroid model as seen in Figure 7.3b and 7.3d can be observed to differ from the actual texture of Bennu as seen in Figure 7.3a and 7.3c. However, this does not influence the machine learning algorithm too much, as it can be made invariant to texture and illumination conditions, which will be elaborated upon in Section 7.5. Moreover, the advantage of using a created texture is that it is not limited by the resolution of the available maps.

7.2.3. Viewpoint sampling

This subsection discusses the model-agnostic viewpoint sampling pipeline, which creates the different camera poses, distance and orientation, w.r.t. a target. Therefore, the discussion throughout this subsection is kept generic, whereas the specific settings used to generate the datasets created in this work are discussed in Section 7.3. These camera poses are used to place the camera within Blender. This is an important step within the image generation pipeline as shown in Figure 7.1, and is required before the images can be rendered.

The problem with real image datasets is that the images are often taken from canonical viewpoints, which then results in these viewpoints being oversampled in the dataset, resulting in viewpoint bias (Movshovitz-Attias et al., 2016). A canonical viewpoint refers to the seemingly 'best-view' to allow for object identification, whereas a non-canonical viewpoint represents the opposite. A canonical viewpoint for a car would be to view it in a 'three-quarter' fashion, whereas the non-canonical viewpoint would be to look at the bottom. However, the advantage of creating a synthetic dataset is that a uniform sampling of different viewpoints can

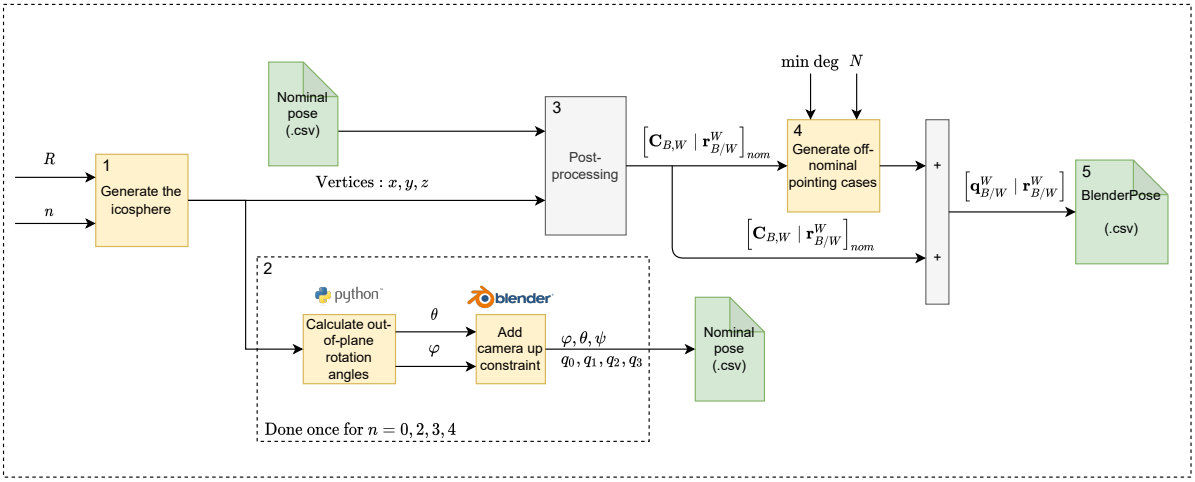


Figure 7.4: An overview of the different facets of the viewpoints generation pipeline

be guaranteed, thereby mitigating the problem of viewpoint bias.

The approach used in both the SPEED dataset as well as the Envisat dataset (Appendix B) is that the camera location is fixed and the target satellite is rotated and moved. The inverse of keeping the target satellite fixed and rotating and moving the camera is the exact same thing in relative geometry, but the only thing that changes are the illumination conditions, i.e., shadows and reflectance of the target. The camera fixed approach is relatively simple to apply and works well for free-tumbling objects that can take on a range of orientations.

However, the target in this work is an asteroid that rotates around its rotational axis in a predictable fashion. Furthermore, the location of the Sun w.r.t. the asteroid is fixed for a certain point in its orbit. As a result, the camera fixed approach used by the aforementioned datasets cannot be used, as this would result in unrealistic illumination conditions. Therefore, a new approach was devised to generate different camera viewpoints (attitude and distances) of the target while attaining realistic illumination conditions.

The viewpoints for the coverage of the target are created following a method proposed by Hinterstoisser et al. (2008) and used by Hinterstoisser et al. (2013) for the generation of templates that are used to estimate the pose of the object. This method uses an icosahedron, which is the largest convex regular polyhedron and has 12 vertices. The sampling is refined by recursively replacing each triangle with four almost equilateral triangles. These vertices are then projected onto a sphere to form an icosphere, also called an geodesic polyhedron. The target is placed at the origin of the icosphere and as such the vertex locations represent the camera position w.r.t. the target. The pipeline that is used to generate the different camera viewpoints (position and orientation) is shown in Figure 7.4. In what follows, the major steps of Figure 7.4 will be individually discussed. These steps are performed for each discrete orientation of the asteroid w.r.t. to the starting position, i.e., resulting in different sides being illuminated.

1. Generate the icosphere

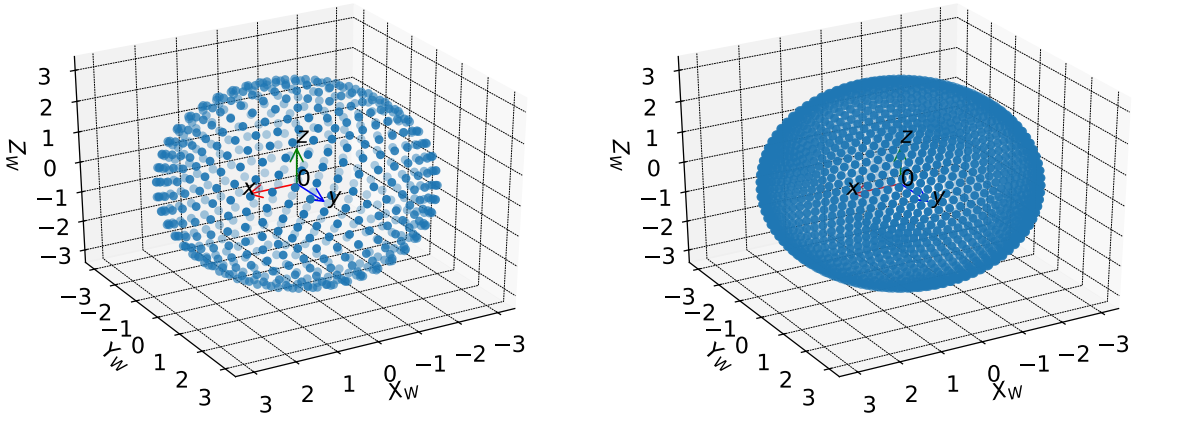
This step generates the icosphere, based on the following parameters:

- R: Radius of the icosphere
- n: Refinement order (how many vertices)

The radius of the icosphere specifies the distance of the vertices with respect to the target's origin (Center of Mass (CoM)). This is therefore synonomous with the relative distance between the camera and the target. A visual representation of the icosphere for two different orders is shown in Figure 7.5, where it can be seen that a higher order results in a more dense coverage.

2. Generate nominal pose datafiles

This process is performed once for each selected order of refinement n and is done to calculate the nominal pointing of the camera per vertex. This orientation is irrespective of the radius of the distance, since the vertex locations are simply multiplied by a constant when increasing the radius. The calculation of the nominal



(a) The icosphere for $n = 3$, representing an angle of $\approx 8°$ between vertices

(b) The icosphere for $n = 4$, representing an angle of $\approx 4°$ between vertices

Figure 7.5: Different icospheres for different orders of refinement n, using $R = 3$, where R can represent any dimension (e.g., m, km)

pointing of the camera consists of two steps: 1) the calculation of the out-of-plane rotation angles and 2) the addition of a camera up constraint as shown in Figure 7.4.

The vertices locations allow the calculation of the out-of-plane rotation angles, namely the roll φ and pitch θ. An object reference frame is centered in the world origin $\begin{pmatrix}0 & 0 & 0\end{pmatrix}^T$ and the axes are aligned with the world's axes, i.e., z pointing upwards, x pointing forward and y completing the right-handed system as can be observed in Figure 7.5. The out-of-plane rotation angles can be calculated by aligning the normal vector (z-vector) of the object reference frame with the normal vector (z-vector) of the reference frame going through the vertex. The coordinates of the vertex expressed in the vertex reference frame (V) are always given by $\begin{pmatrix}0 & 0 & 1\end{pmatrix}^T$, the vertex locations presented in the object reference frame can then be calculated using:

$$\begin{aligned}\mathbf{v}^O = \mathbf{C}_x(\varphi)\mathbf{C}_y(\theta) &= \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\varphi & \sin\varphi\\ 0 & -\sin\varphi & \cos\varphi\end{bmatrix}\begin{bmatrix}\cos\theta & 0 & -\sin\theta\\ 0 & 1 & 0\\ \sin\theta & 0 & \cos\theta\end{bmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\\ &= \begin{bmatrix}\cos\theta & 0 & -\sin\theta\\ \sin\varphi\sin\theta & \cos\varphi & \sin\varphi\cos\theta\\ \cos\varphi\sin\theta & -\sin\varphi & \cos\varphi\cos\theta\end{bmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\end{aligned} \tag{7.1}$$

This can be simplified to the following relationship:

$$\mathbf{v}^O = \begin{pmatrix}-\sin\theta\\ \sin\varphi\cos\theta\\ \cos\varphi\cos\theta\end{pmatrix} \tag{7.2}$$

Equation (7.2) can be rewritten into the following form and using the known vertex location $\mathbf{v}^O = \begin{pmatrix}x & y & z\end{pmatrix}^T$, the out-of-plane rotation angles θ and φ can be calculated using:

$$\begin{gathered}\theta = \arcsin(-x) \quad \sin\varphi = \frac{y}{\cos\theta} \quad \cos\varphi = \frac{z}{\cos\theta}\\ \varphi = \arctan 2(\sin\varphi, \cos\varphi)\end{gathered} \tag{7.3}$$

These angles represent the orientation of the vertex reference frame w.r.t. the object reference frame. However, the yaw angle ψ is still arbitrary and it is desired that the yaw angle for each vertex location is such that the camera is pointing up. This implies according to the Blender camera reference frame that the Y_B-axis is pointing up. This was guaranteed by implementing a tracking constraint in Blender. This tracking constraint makes sure that the camera is pointing up and that the optical axis of the camera, $-Z$-axis, is pointing towards the origin (center of mass) of the target asteroid reference frame A. Subsequently, the resulting final orientation was transformed to unit quaternions and exported as a `.csv` file. The procedure has been verified

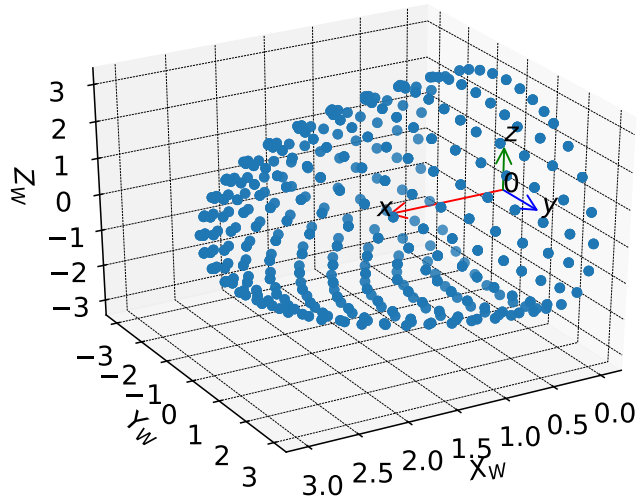


Figure 7.6: The post-processed icosphere, where all datapoints with $x^W < 0$ are removed, $n = 3$, using $R = 3$, where R can represent any dimension (e.g., m, km)

using known rotations and their resulting vertex locations and then using those vertex locations to retrieve the rotations. These nominal pose files for different orders of n could then be used in the dataset generation pipeline (Figure 7.4).

3. Post processing

This step refers to the removal of certain irrelevant data points of the icosphere. Based on the position of the Sun, only a certain part of the target is illuminated, whereas the other parts are in complete darkness. The only relevant camera positions are those that look at the (partially) illuminated part of the asteroid, which results in discarding about half of the sphere as can be observed in Figure 7.6.

4. Generate off-nominal pointing cases

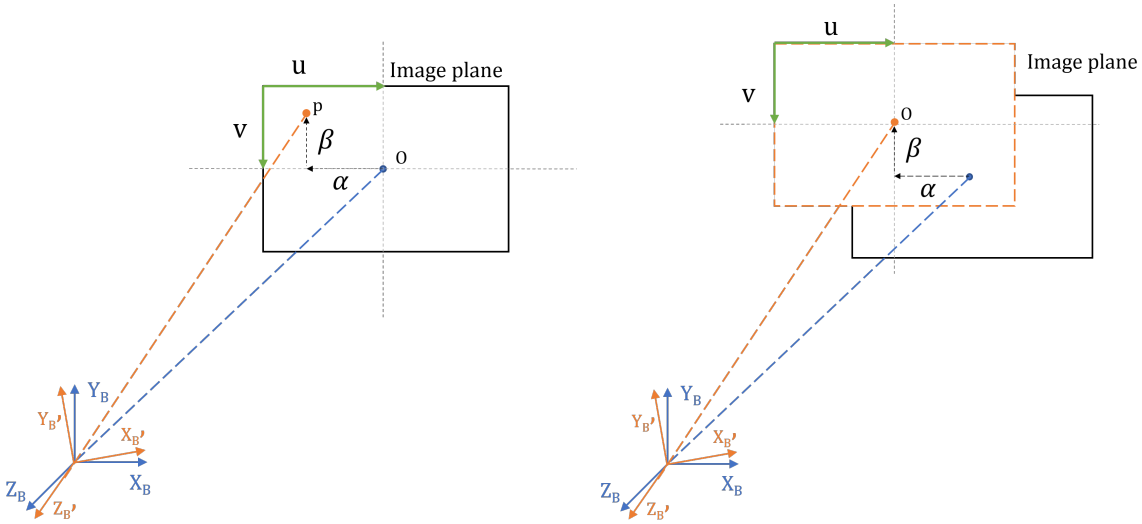
It is assumed that the origin of the target asteroid reference frame, A, lies on the perspective line from the origin of the Blender reference frame, B, towards the center of the asteroid in the pixel reference frame P (Figure 4.2), i.e., the 2D centroid (P frame) of the asteroid lies on the same perspective line as the actual 3D centroid (A frame) of the asteroid. Therefore, in the nominal pointing case the target (asteroid) is located in the center of the image plane, such that the coordinates of the center of mass in the image reference frame are $\mathbf{p}^P = (\frac{Nu}{2}, \frac{Nv}{2})$. However, the asteroid will not necessarily lie in the center of the image throughout the actual mission and the deviation from the middle of the image can be set using the azimuth and elevation angles (α, β), i.e., bearing angles. The camera's position is fixed and only the orientation of the camera w.r.t. the target is changed through the use of these bearing angles. This is visually illustrated in Figure 7.7, where the Blender camera frame is rotated over the Y and X-axes, respectively. A positive rotation of α and β results in the camera's $-Z$-axis (optical axis) pointing towards the top left corner of the 'old' image plane. This results in point p becoming the new principal point O and as such the asteroid has 'moved' to the lower right corner within the new image plane.

The values of α and β are randomly selected from the interval shown below. This interval allows the asteroid to be randomly spread over the image, but still results in the majority of the asteroid to be fully in the field of view.

$$\begin{cases} -5° < \alpha < 5° \\ -5° < \beta < 5° \end{cases} \tag{7.4}$$

The function is implemented in Python and takes the following arguments as input.

- **Minimal degree**: This parameter specifies the minimal off-nominal pointing angle for α and β. This was implemented to ensure that the off-nominal pointing case did not closely resemble the nominal



(a) This shows the image plane of the nominal pointing case where the asteroid is in the center of the image plane (O)

(b) This shows the situation in which the optical axis, $-Z$-axis, points towards the point p, making it the new optical point (O), moving the asteroid to the lower right corner

Figure 7.7: Graphically illustrating the off-nominal pointing implementation

pointing case. The practical result is that a square region, governed by the minimal degree, surrounding the center is off-limits.

- **N**: The number of off-nominal cases that are to be generated per vertex

The transformation matrix that rotates the nominal-pointing Blender camera frame B to the off-nominal pointing Blender camera frame B' is given by:

$$\begin{aligned}\mathbf{C}_{B',B} = \mathbf{C}_x(\beta)\mathbf{C}_y(\alpha) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & \sin\beta \\ 0 & -\sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \\ &= \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ \sin\alpha\sin\beta & \cos\beta & \cos\alpha\sin\beta \\ \sin\alpha\cos\beta & -\sin\beta & \cos\alpha\cos\beta \end{bmatrix}\end{aligned} \tag{7.5}$$

The position vector presented in B' reference frame can then be determined using the following, where $\|\mathbf{r}_{\mathrm{B/A}}\|_2$ is the 2-norm of the $\mathbf{r}_{\mathrm{B/A}}$ vector. This vector only has a z-component as in the nominal pointing case the $-Z$-axis (optical axis) points towards the center of mass of the target.

$$\mathbf{r}^{B'}_{\mathrm{B/A}} = \mathbf{C}_{B',B}\mathbf{r}^{B}_{B/A} = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ \sin\alpha\sin\beta & \cos\beta & \cos\alpha\sin\beta \\ \sin\alpha\cos\beta & -\sin\beta & \cos\alpha\cos\beta \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ \|\mathbf{r}_{\mathrm{B/A}}\|_2 \end{pmatrix} \tag{7.6}$$

The B' reference frame is the final orientation of the Blender camera frame when bearing angles are applied and the $'$ is only used to discern between the nominal pointing case and the off-nominal pointing case. This position vector can then converted into the camera reference frame for the labeling of the dataset, which is discussed in the annotations, Section 7.4.

5. Generate the Blender camera poses

For any given settings of the different parts, namely the radius of the icosphere R, the refinement order of the icosphere n, the minimal degree of the off-nominal pointing, and the number of off-nominal pointing cases per vertex N, the camera poses are calculated and the final results are exported as a `.csv` file. These poses are used in Blender to place the camera and represent the position and orientation of the Blender camera w.r.t the world frame, $[\mathbf{C}_{B,W} \mid \mathbf{r}^{W}_{B/W}]$. However, the transformation matrix is converted to unit quaternions,

$[\mathbf{q}^W_{B/W} \mid \mathbf{r}^W_{B/W}]$, to store within the `.csv` file. These camera poses are irrespective to the asteroid's orientation w.r.t. the world axes.

7.2.4. Rendering process

After establishing the desired dataset properties and creating the datafiles containing the camera poses, the images can be rendered in Blender. The rendering process is the next step in the image generation pipeline as illustrated in Figure 7.1. The rendering of images occurs in batches and is done through Blender's Python API. Each batch consists of 1348 images that represent all camera viewpoints for that distance, nominal and off-nominal pointing, for a certain asteroid orientation w.r.t. the world axes (Section 7.3). As discussed in Section 6.2, the images have been rendered through Google Colaboratory (Colab) using the NVIDIA Tesla P100-PCIE-16GB GPU. The rendering of a single batch takes about 2.5 hours using the specified GPU.

The rendering process for each distance and asteroid orientation (Equation (3.20)) combination is graphically illustrated in Figure 7.8. There are two main scripts involved in the rendering process, the `pipelineLinux.py` script sets the asteroid's orientation w.r.t. the world axes (ψ), places the cameras according to the datafiles ($[\mathbf{q}^W_{B/W} \mid \mathbf{r}^W_{B/W}]$), sets the illumination conditions, and renders the images according to the desired settings. The `BoundingBox.py` script outputs the coordinates of the tightly fit bounding box presented in the image reference frame (P) around the asteroid for every image. The following encoding (x_{min}, y_{min}, w, h) is used, where x_{min} and y_{min} refer to the coordinates of the top left corner of the bounding box and w and h refer to the width and height of the bounding box, respectively. This can be observed in Figure 7.14. The `BennuTextured.blend` file contains the textured 3D model of Bennu and creates the 3D environment, which is rendered.

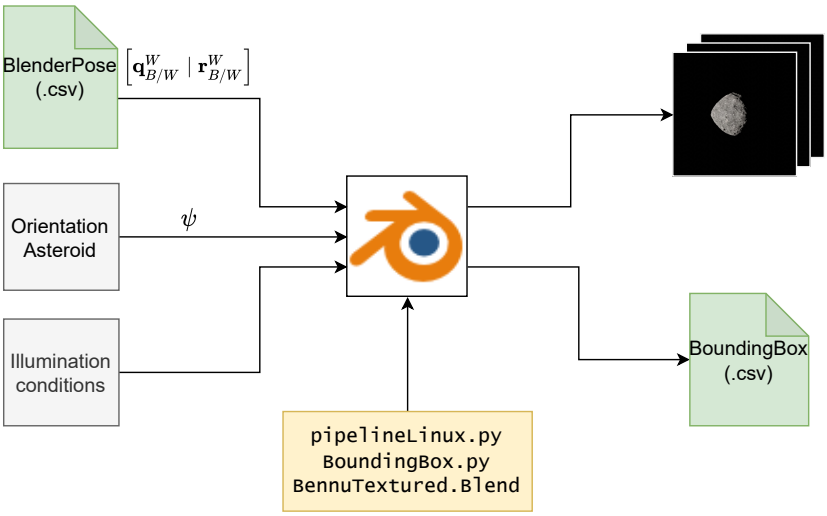


Figure 7.8: A visualization of the rendering process in Blender in which several inputs are given to the scripts that operate within Blender through the use of a Python API

7.3. Dataset properties

This section discusses the properties that were selected in generating the dataset, these settings are also summarized in Table 7.1.

Distances

The relative distance to the asteroid is a parameter that needs to be specified in the generation of the dataset. As was discussed in Section 2.3, the asteroid is assumed to be fully in the field of view of the camera. The distances can be selected arbitrarily within a certain range without influencing the resulting networks too much, as the networks will not solve the pose from the image directly (end-to-end). The problem is divided into sub-problems, i.e., first an object detection network and then a keypoint detection network. Therefore, the data does not have to be that complex. The object detection network will detect the image and eventually crop the image around the target, which will be fed to the keypoint detection networks.

Table 7.1: The settings used for the generation of the dataset

Parameter	Description	Value(s)
R	The radius of icosphere, i.e., relative distance between camera and target	[4.5;6;7.5;9] km
n	Refinement order of the icosphere, i.e., the number and spacing of the vertices	3
min deg	The minimal value for the bearing angles, α, β for the generation of off-nominal pointing cases	1.0°
N	The number of off-nominal pointing cases per vertex	3
ψ	The asteroid reference frame orientation w.r.t world axes	$0 + k \cdot 60$ for $k = 1, 2, ..., 5$

Therefore, based on the camera intrinsic parameters listed in Table 7.2 and based on the relevance for a potential mission the distances are discretized in the range [4.5;9] km with 1.5 km intervals. As aforementioned, the distance range and interval can be selected arbitrarily due to the usage of an object detection network. The 1.5 km interval allows to cover the large relative range without considerably increasing the size of the dataset. At 3 km the entire asteroid is still in the field of view of the camera, but the bearing angles change the position of the asteroid within the image, thereby cutting off a significant portions of the asteroid, rendering the images useless. For distances further away > 9 km, ground-based navigation or other techniques can safely navigate the spacecraft and feature-based navigation can become complicated as the asteroid only makes up around 15% of the pixels within the image for the camera settings used.

Camera viewpoints

For every selected distance R, the refinement order $n = 3$ was chosen to create a fine sampling of the vertices with an angle of $\approx 8°$ between vertices. Furthermore, for every vertex the nominal pointing case is automatically created. However, to expand the dataset and create more realistic conditions several off-nominal pointing cases were generated. The minimal offset for the bearing angles α and β was set at 1°, min deg = 1° and per vertex three randomly generated off-nominal cases were generated, $N = 3$. This resulted in a more complete dataset covering a variety of different views of the target, for a total of 1348 viewpoints per distance and asteroid orientation.

Asteroid orientations

As previously mentioned, the 3D model of Bennu is imported into Blender using the $+Z$ up and $+Y$ forward convention. This aligns the asteroid reference frame with the world axes for this initial orientation. The asteroid orientation w.r.t. the world axes determines which side of the asteroid is illuminated. To capture the asteroid from different sides and recreate a variety of relative geometries between the asteroid and the spacecraft, without considerably increasing the size of the dataset, the full circle of rotation is discretized into 60° intervals.

Illumination conditions

As was discussed in Section 2.3, the position of the asteroid w.r.t the Sun is considered fixed and therefore the asteroid-Sun vector is fixed. Furthermore, for simplicity the obliquity of the ecliptic is considered to be 0°. The Sun can be used as the light source in Blender for which the strength in W/m^2 and the angular diameter as seen from Earth need to be specified. The strength of the Sun, however, does not relate to the actual W/m^2 and is set at $10 W/m^2$ to create a realistic scene. Furthermore, the angular diameter is set at 0.5° to mimic the angular diameter of the Sun at 1 au. The Sun light source in Blender emits light of constant intensity in a single direction from infinitely far away. Therefore, the location of the Sun light source in Blender does not affect the rendered result, however, it is placed such that the phase angle is zero at the equator of the asteroid. The trained machine learning networks can be made invariant to illumination conditions as will be discussed in Section 7.5.

Table 7.2: The intrinsic camera parameters used in generating the synthetic images

Parameter	Description	Value
N_u	Number of horizontal pixels	1024
N_v	Number of vertical pixels	1024
f_x	Horizontal focal length	0.15 m
f_y	Vertical focal length	0.15 m
du	Horizontal pixel length	$2.34375 \cdot 10^{-7}$ m/px
dv	Vertical pixel length	$2.34375 \cdot 10^{-7}$ m/px
FOV	Field of View	13.686°

Camera settings

The images were rendered in Blender with the camera settings listed in Table 7.2. Navigation cameras often have an image size of 1024×1024 px (Rowell et al., 2015) and the AFC used for the Dawn mission and proposed for the HERA mission also uses this image size and focal length (Sierks et al., 2011)

General properties

Based on the aforementioned settings, which are listed in Table 7.1, for the four distances, 1348 camera viewpoints per distance and asteroid orientation, and six asteroid orientations, the size of the dataset ends up being 32,352 images. These images are randomly sampled and split into training, validation, and test sets in a 70/15/15 fashion, following conventions used for relatively small datasets (Pasqualetto Cassinis et al., 2021b). The test data is needed to achieve an unbiased estimate of the performance, as during training the model is fitted to the training data and evaluated on the validation data and as such also fitted to the validation data. The validation and test set need to come from the same distribution.

The size of the dataset lies in between the two reference datasets discussed in Appendix B. The dataset is relatively small for general deep-learning practices, however, this is possible because the networks that are used in this work are open-source implementations that have been pre-trained on similar problems, i.e., object detection and keypoint detection. Moreover, due to the separation of the pose estimation problem into three separate parts (OD, KD, and PnP) less data is required compared to training an end-to-end architecture.

7.4. Annotations

This section discusses the dataset annotations that are required to train and test the object and keypoint detection networks. Furthermore, this is used to evaluate the performance of the entire pose estimation pipeline, i.e., OD, KD, and PnP. The annotation process follows the image generation part as outlined in Figure 7.1. The required processing steps and resulting dataformats are discussed for three parts, namely the pose, the keypoints, and the bounding box. An overview of the entire process is graphically illustrated in Figure 7.9.

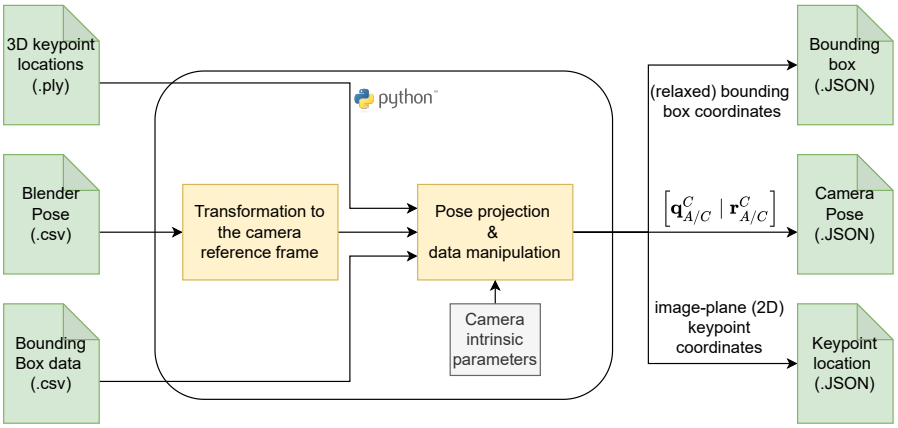


Figure 7.9: A visualization of the steps taken to create the annotated data required for labeling the dataset

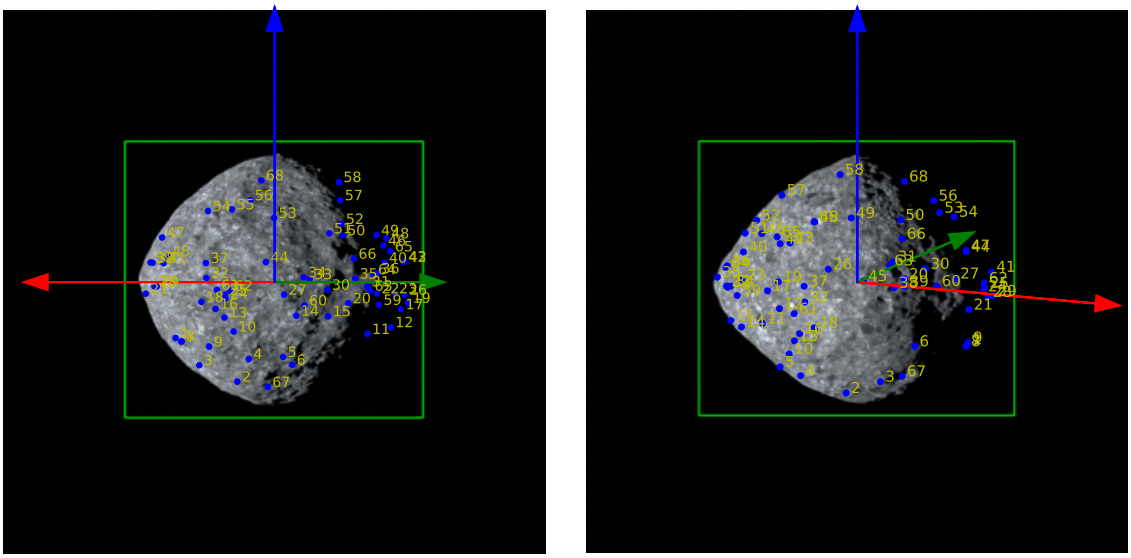


Figure 7.10: Visualizations of the different annotations, where the relaxed *bounding box* is given in green, the *keypoints* are shown with their corresponding number, and the axes represent the orientation of the asteroid reference frame w.r.t. the camera, where red, green, and blue refer to the X,Y, and Z-axis of the asteroid reference frame, respectively

A datafile is created that contains all the annotations to allow for consistent analysis. This is stored in a `.json` format, as this stores the data in an orderly fashion and it allows for easy retrieval of the data through the use of Python dictionaries. An example format is shown in Code Listing C.1. These different annotations are visualized in Figure 7.10.

7.4.1. Pose

During the dataset generation process the pose of the Blender camera (B) w.r.t the world axes (W) was used to generate the images, $\mathbf{C}_{B,W}$ and $\mathbf{r}_{B/W}^{W}$. However, for the purpose of training the networks and evaluating the performance of the subsequent PnP solver, the pose of the camera reference frame (C) w.r.t. the asteroid reference frame (A) is required. Therefore, $\mathbf{C}_{C,A}$ and $\mathbf{r}_{A/C}^{C}$ are required to set up the pose matrix $\mathbf{P} \in \mathbb{R}^{3\times4}$, $\left[\begin{array}{c|c} \mathbf{C}_{C,A} & \mathbf{r}_{A/C}^{C} \end{array}\right]$ as shown in Equation (4.9). The transformation matrix $\mathbf{C}_{C,A}$ can be established from the following:

- $\mathbf{C}_{A,W}$: The transformation matrix from the world reference frame to the asteroid reference frame
- $\mathbf{C}_{B,W}$: The transformation matrix from the world reference frame to the Blender camera frame
- $\mathbf{C}_{C,B}$: The transformation matrix from the Blender camera frame to the camera frame

The desired transformation matrix $\mathbf{C}_{C,A}$ can be calculated using the following, where $\mathbf{C}_{C,B}$ is constant and can be found in Equation (3.22). Furthermore, $\mathbf{C}_{A,W}$ and $\mathbf{C}_{B,W}$ can be found in Equations (3.20) and (3.21), respectively. These two depend on the asteroid orientation and the pose of the camera corresponding to that image.

$$\mathbf{C}_{C,A} = \mathbf{C}_{C,B}\mathbf{C}_{B,W}\mathbf{C}_{A,W}^{T} \tag{7.7}$$

The translational vector $\mathbf{r}_{A/C}^{C}$ can be constructed using the following:

- $\mathbf{r}_{B/W}^{W}$: The position vector from the origin of the world frame to the origin of the Blender camera frame presented in the world frame. However, only the orientation of the asteroid changes, i.e., the side that is illuminated, but the origin remains aligned with the world origin. Therefore, $\mathbf{r}_{B/W} = \mathbf{r}_{B/A}$
- $\mathbf{C}_{C,W} = \mathbf{C}_{C,B}\mathbf{C}_{B,W}$: The transformation matrix from the world frame to the camera frame.
- The origin of the Blender camera frame and the camera frame are the same and therefore $\mathbf{r}_{B/A} = \mathbf{r}_{C/A}$.

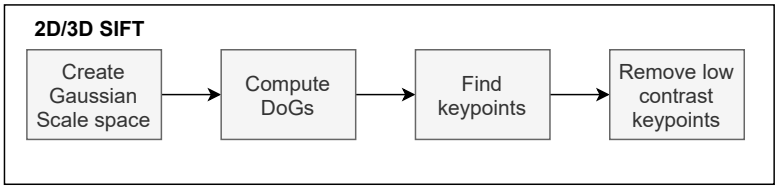


Figure 7.11: The steps of the 2D/3D SIFT algorithm

The final translational vector $\mathbf{r}^C_{A/C}$ can henceforth be calculated as follows:

$$\mathbf{r}^C_{A/C} = -\mathbf{r}^C_{C/A} = -\mathbf{C}_{C,W}\mathbf{r}^W_{C/A} \tag{7.8}$$

The pose matrix is then calculated using:

$$\mathbf{P} = \left[\begin{array}{c|c} \mathbf{C}_{C,A} & \mathbf{r}^C_{A/C} \end{array} \right] \tag{7.9}$$

This pose is used to label the images and an example file format can be found in Code Listing C.1, where $\mathbf{C}_{C,A}$ is transformed back to scalar-first unit quaternions using Sheppard's algorithm, Equations (3.9) and (3.10).

7.4.2. Keypoint designation and annotation

This subsection discusses the keypoint designation and annotation, where the designation refers to the fact that a set of points on the target are selected as the keypoints.

Keypoint designation

The keypoints (interest points) were designated on the asteroid's 3D model using the 3D SIFT algorithm, which is an adaptation of the 2D SIFT algorithm for 3D point clouds (Scovanner et al., 2007). This algorithm is implemented in C++[9] in the open-source PCL[10]. The algorithm estimates SIFT points based on the z-gradient of the 3D points. The designation of keypoints through the use of this algorithm has been successfully applied by Zhao et al. (2018) to estimate the pose of general terrestrial objects. The advantage of designating surface keypoints is that they are closely related with the models features.

By using a model-based approach, the minimum feature size on the asteroid surface is limited by the point cloud's resolution, which has a resolution of 75 cm. Because the feature is designated on the 3D model, it does not rely on texture information and as such on the resolution of the camera and available maps. The latter limits the landmark designation and extraction (Rowell et al., 2015) or *unknown feature tracking* approach (Pellacani et al., 2019), as it sets the minimum required scale for features on the surface to be resolved by the camera.

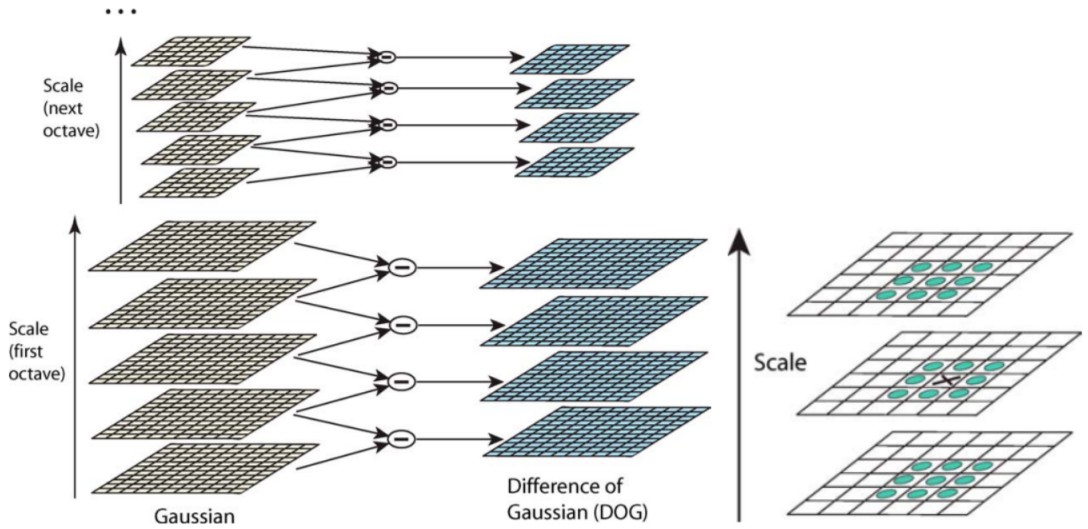
The 3D SIFT algorithm is an adaptation of the 2D SIFT algorithm and is used for point clouds. The concepts of the 2D SIFT algorithm apply and will be briefly discussed, as it is more intuitive, before diving into the adaptation to the 3D point cloud. The steps of the algorithms are shown in Figure 7.11, these steps will be discussed individually in what follows.

2D SIFT

The 2D SIFT algorithm has been created by Lowe (2004) and detects keypoints through the use of *scale-spaces*, where the scale refers to the amount of Gaussian blur added to an image. Moreover, the algorithm applies these different amount of blurs, *scales*, to different image sizes called *octaves*. The size of the image for the k^{th} octave decreases by half compared to the $k-1$ octave, where the image size of the first octave is equal to the input size. The algorithm generates progressively blurred out images (scales) for each different octave.

The blurring is applied to the image by using a convolution of the Gaussian operator on the image, where the standard deviation of the Gaussian distribution σ is a tunable parameter for the algorithm, which determines the amount of blur. The blur on the first image is equal to σ and subsequent images have a Gaussian blur of $k \cdot \sigma$, where k is a constant. The Difference-of-Gaussian (DoG) is calculated, which highlights certain parts in the images, which are useful for subsequent keypoint extraction. The process of calculating the DoG

[9] `https://github.com/sjtuytc/betapose`, Date accessed: 25-11-2021
[10] `https://pointclouds.org/`, Date accessed: 25-11-2021



(a) A schematic representation of the calculation of the DoG

(b) A schematic representation of the extraction of keypoints by finding the maxima/minima through the comparison of a pixel (marked by X) to the nearest pixels in the current scale and the scale above and below of the DoG

Figure 7.12: Graphically illustrating the basic building blocks of the 2D SIFT algorithm (Lowe, 2004)

is shown graphically in Figure 7.12a, where the DoG represents the difference between the different scales within an octave.

The keypoints are located by finding the maxima/minima within the DoG images, comparing not only with the nearest pixels in the current scale, but also in the scale above and below, graphically illustrated in Figure 7.12b. Therefore, the upper and lowermost scales are not used as they only have one neighbor. These keypoints are then refined by mathematically locating their subpixel location. The post-processing revolves around removing keypoints that have low contrast, meaning that some keypoints might lie along an edge or do not have enough contrast rendering them not useful as features. Therefore, a threshold is used.

3D SIFT

The scale-invariant 3D SIFT algorithm works in a similar way, however, instead of relying on the intensity of a pixel now the principal curvature of a point within the 3D point cloud is used. The Gaussian scale-space for the 3D point cloud is created by downsampling the point cloud (reducing the number of points) using voxelgrid filters of different sizes and by applying a Gaussian blur filter.

A voxelgrid can be conceptualized as a set of minute 3D boxes that are placed over the point cloud. Then all the points within each of these 3D boxes, *voxels*, will be replaced by their centroid, thereby reducing the size of the point cloud and replacing the intensity of the points with the weighted average of them all. The scale-space of the 3D SIFT is calculated using the following equation (Jiao et al., 2019), where $*$ represents the convolutional operator, $G(x,y,z,\sigma)$ represents the 3D-SIFT Gauss kernel function given in Equation (7.11), and $P(x,y,z)$ is the three-dimensional coordinate of the point cloud.

$$L(x,y,z,\sigma) = G(x,y,z,\sigma) * P(x,y,z) \tag{7.10}$$

$$G(x,y,z,\sigma) = \frac{1}{(\sqrt{2\pi}\sigma)^3} e^{-(x^2+y^2+z^2)/2\sigma^2} \tag{7.11}$$

Then the DoG for each different *scale* point cloud within an *octave* is created.

$$DOG\left(x,y,z,k^i\sigma\right) = P(x,y,z) * \left(G\left(x,y,z,k^{i+1}\sigma\right) - G\left(x,y,z,k^i\sigma\right)\right) \tag{7.12}$$

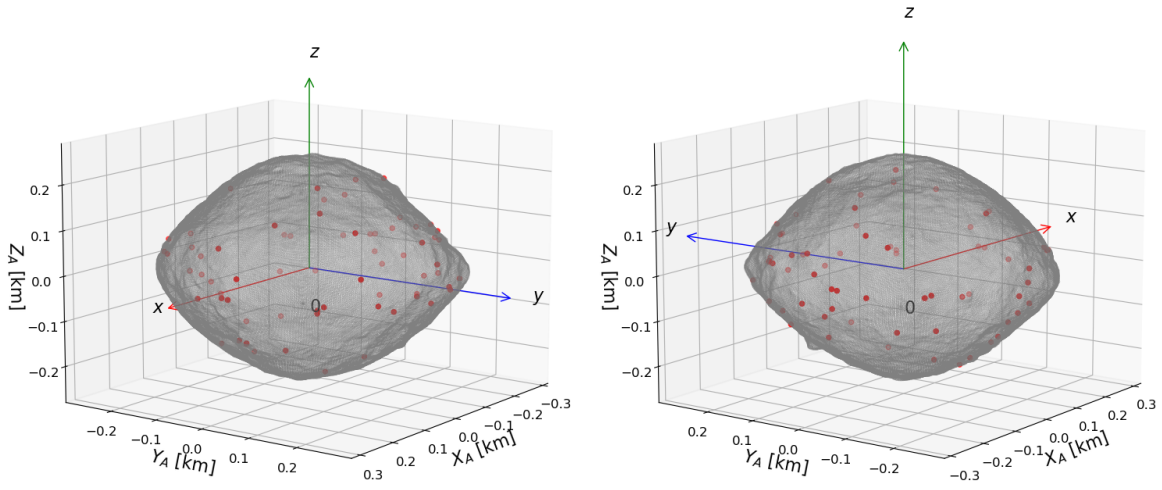


Figure 7.13: Distribution of the keypoints (red) on the asteroid for two different orientations, where points on the back are also visible for both orientations.

Similarly to the 2D SIFT algorithm, a point within the point cloud is designated as a keypoint, when it has the maximum/minimum of the DoG values among its k nearest points within the same DoG as well as the DoGs above and below. Finally, the keypoints in areas with low curvature values are rejected.

The 3D SIFT algorithm has been tuned to achieve the designation of 68 keypoints on the surface of the asteroid. This number was the result of the used settings with the main goal of having a good distribution of keypoints around the asteroid. The distribution can be seen in Figure 7.13 for two different asteroid orientations. The following settings of the algorithm were used to designate the keypoints on the 3D model.

- Standard deviation of the minimum scale: 0.001
- Number of octaves: 10
- Number of scales per octave: 5
- Minimum contrast: 0.00055

The designation of 68 keypoints also has the benefit that the network does not necessarily have to be able to predict all of them perfectly within every image, as for the subsequent pose estimation only a minimum of six points are required. The increase in training effort for 68 keypoints compared to a lower number is minimal.

Keypoint annotation

These keypoints need to be annotated such that it can be used by the keypoint detection network. These designated keypoints need to be transformed from the 3D model space to the 2D image space, to derive the ground-truth locations of these keypoints. The designated 3D features can be transferred to a 2D location based on the camera pose and intrinsic parameters. The theoretical accuracy that can be achieved by the keypoint detection network is 100%, as the conversion to the 2D locations already takes into account these effects, i.e., it represents the best achievable 2D location based on the 3D keypoints and camera parameters. Given the ground-truth pose data (Equation (7.9)) and the camera intrinsic parameters (Table 7.2), the PnP problem discussed in Section 4.2 is used to transform the 3D keypoints to 2D. The PnP equation is stated again for convenience below, where the camera intrinsic parameters have been filled in:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} 4266.67 & 0 & 512 \\ 0 & 4266.67 & 512 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{C}_{C,A} & | & \mathbf{r}_{A/C}^{C} \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \tag{7.13}$$

Equation (7.13) is solved for each keypoint and each image and the annotated keypoints are saved in the COCO format (`.json`), which is used to train the keypoint detection network. This file format is standard for

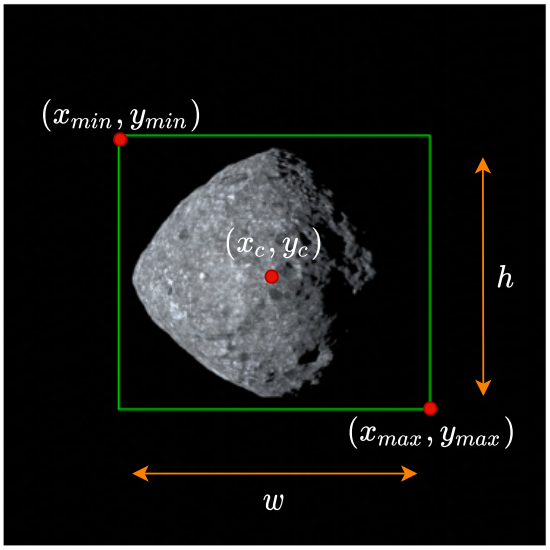


Figure 7.14: Visualization of the different bounding box encoding parameters

keypoint detection networks and therefore allows for easy adaptation. An example of this format is given in Code Listing C.2.

7.4.3. Bounding box

The bounding box resulting from Blender is a tightly fitted one, however, Chen et al. (2019) demonstrated that by relaxing the ground-truth bounding box coordinates by a small margin the object detection training was improved. Furthermore, the Envisat dataset relaxed the bounding box coordinates by approximately 10% of the original width and height. The bounding box coordinates from Blender are relaxed by 5% for the Bennu datasets and the values are stored in the `.csv` format for the training of the object detection network, with the following columns *filename*, *width*, *height*, *class* and $(x_{min}, y_{min}, x_{max}, y_{max})$, presented in the pixel reference frame (Section 3.1). The width and height refers to the image size and the class to *Bennu*, and the bounding box encoding has been converted to this representation, which can be observed in Figure 7.14. Furthermore, the ground-truth bounding box coordinates (x_{min}, y_{min}, w, h) are also used in the COCO format Code Listing C.2, where as before x_{min} and y_{min} refer to the coordinates of the top left corner of the bounding box and w and h refer to the width and height of the bounding box, respectively, as can be observed in Figure 7.14.

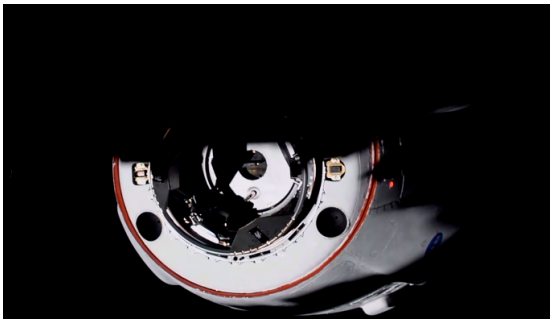
7.5. Bridging the domain gap from synthetic to real images

Before the CNN-based pipeline will be used in an actual space mission for vision-based navigation, its performance needs to be validated in a highly representative space environment prior to mission deployment. However, there does not exist any dataset consisting of real images of small-bodies with labeled poses that is large enough for training CNNs. Therefore, the neural networks are trained exclusively on synthetic images. Furthermore, target-specific images are not available prior to arriving at that small body.

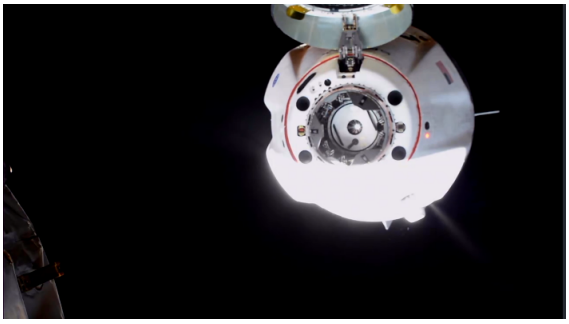
These synthetic images are easily mass-produced using 3D rendering software and annotated with the true poses, which has been performed in this work (Sections 7.2 to 7.4). However, CNNs trained solely on synthetic images are prone to the *domain gap*, as the CNNs might overfit to the features that are specific for the synthetic images. This *domain gap* refers to the gap in performance of the CNN on synthetic images compared to real images, and is caused by real images having a different statistical distribution compared to synthetic images. Moreover, real images can contain noise and other image corruptions that are not or cannot be accurately modeled in the synthetic images. Actual space imagery, as shown in Figure 7.15, in particular suffers from low signal-to-noise ratios, high contrast, and particular surface illumination conditions, i.e., over and underexposed (Sharma, 2019)

This *domain gap* stems from the fact that neural networks cannot generalize to out-of-distribution exam-

[11] `https://images.nasa.gov/`, Date accessed: 25-01-2022



(a) Partial occlusion caused by shadowing and lack of light diffusion due to the absence of an atmosphere. Source: NASA[11]



(b) Overexposure due to direct sunlight. Source: NASA[11]

Figure 7.15: Images representing difficult illumination conditions, such as partial occlusion or overexposure due to direct sunlight, causing high contrast.

ples with respect to the training data (synthetic images), resulting in deteriorated performance of the CNN to actual space imagery. Bridging this domain gap and achieving robustness of the CNN trained on synthetic images is crucial in creating deep learning systems that can be deployed in safety-critical applications. Artifacts from real images can be roughly divided into two parts, which are discussed below.

1. **Image corruptions**:
   The image corruptions are artifacts from real sensors and the actual space environment. These artifacts can arise from events, such as pluming and cosmic radiation, and due to miscalibration or other factors. Common corruptions among others include hot/dead pixels, defocusing, Gaussian and shot noise.
2. **Textures**:
   The textures in the real images can be different from the training examples, as the real sensors might capture different textures or the actual surface properties are unknown, which is the case for missions to small-bodies. Geirhos et al. (2018) have shown that contrary to former belief, CNNs rely more on the object's texture than on global shape, i.e., it is more biased towards texture than towards shape. This is counter-intuitive for humans who would for example classify a cat with elephant texture as a cat and not as an elephant. Therefore, this would result in inferior performance of the CNNs trained on local textures that deviate from the real images.

There are two distinctly different approaches that can be used to improve the network robustness to real images and thereby reducing the domain gap: 1) try and mimic the actual space environment as much as possible within the synthetic images and 2) include real images during training. However, in actual missions to small-bodies these are unavailable before arriving at the target and may not have accurate pose labels. Consequently, it should be considered that only synthetic images are available for training. Resulting in only being able to adapt the first approach.

Therefore, it is critical to mimic the actual space environment as much as possible within the synthetic images. However, it is extremely difficult to exactly replicate the target object's surface properties and illumination conditions that will be encountered throughout the mission. As discussed in Subsection 7.2.1, Pasqualetto Cassinis et al. (2020) and Pasqualetto Cassinis et al. (2021b) have created deep-learning datasets for the Envisat spacecraft using Cinema4D and Black et al. (2021) have used Blender to render images of the Cygnus spacecraft for training a CNN. However, Brochard et al. (2018) stated that these commercially available rendering software lack the realism required for advanced image processing techniques for space applications. Specialized software, such as ESA's PANGU and Airbus' SurRender (Brochard et al., 2018) are available, however, as aforementioned, licenses are required to use these sofware. Furthermore, their efficacy in generating realistic images for large-scale dataset generation for deep-learning purposes has not been investigated. Moreover, generating high photorealistic images through the use of such software might increase the complexity of the development process and limit wide-scale adaptation, as care should be taken to accurately model the sensor, surface properties (texture) of the target object, and specific optical properties. Next to this, this detailed information of the target small-body is often unavailable before hand. Therefore, it is deemed better to incorporate a training process that allows the CNN to close the domain gap without

relying on too much *a-priori* information. The major advantage of using such a training process is that it allows the network to become robust to a variety of augmentations, textures, and illumination conditions with minimum effort, i.e., the exact properties of the target do not have to be modeled accurately.

Data augmentation has been an important tool for increasing the generalization performance of neural networks. As discussed in Section 5.5, data augmentation is used to make the network invariant to the used transform, which could be pixel-level augmentations, such as brightness changes, or affine transformations, such as scale and rotation changes.

- **Corruption robustness:**
  This revolves around augmenting the dataset with image corruptions commonly found in the real images to emulate certain effects. As listed in Appendix B, the SPEED dataset added Gaussian blur and white noise to the images to mimic depth of field and shot noise. Hendrycks and Dietterich (2019) created a new dataset based on the ImageNet dataset (Deng et al., 2009) to benchmark network's robustness againts these common corruptions. They found that networks trained solely on uncorrupted images did not generalize well to real images containing these corruptions. Therefore, neural nets that are to be used for real-world applications should include images with these commonly found corruptions in their training sets. Moreover, it will not only increase the robustness of the trained CNNs against common image corruptions originating from real sensors or the space environment, but these corruptions can also make the network more robust against different textures. Pasqualetto Cassinis et al. (2021b) showed that a CNN trained on an augmented dataset consisting of *pixel-level* augmentations was more robust against variations in illumination conditions and textures.

Furthermore, currently there is a growing interest in using algorithms during training to bridge the domain gap (intra-class variation), these can be grouped as follows (Jackson et al., 2019; Park et al., 2021).

1. **Unsupervised domain adaptation**:
   This refers to using a Domain-Adversarial Neural Network (DANN), which combines a normal feature extractor with unsupervised domain adaptation in an end-to-end training process. Annotated training examples from the source domain (synthetic imagery) and not labeled examples from the target domain (real imagery) are used to train a domain-invariant feature extractor using adversarial training (Ganin et al., 2016).
2. **Domain randomization:**
   This refers to randomizing various parts of the input images where it is assumed that the target image (real image) is simply also a randomized version of the input image. This approach was applied to randomize the texture of the spacecraft object by Park et al. (2019) and Pasqualetto Cassinis et al. (2021b) in an effort to make it more robust to texture differences. This texture randomization process is based on Neural Style Transfer (NST) technique devised by Huang and Belongie (2017). NST refers to using the style of one image and the content of another, to generate a target image of the content in the style of the other. This technique has been proven to decrease the dependency of the network on textures (Geirhos et al., 2018). Furthermore, it forces the network to rely more on shape instead of texture (Geirhos et al., 2018). The underlying notion is that by using style augmentation, the network will learn to be invariant against low-level features, such as texture and illumination, and as such do not overfit to them. Therefore, the network will perform better on real images, which may have different properties than the synthetic training images (or different texture of the target body). Another positive effect is that the networks show robustness to image distortions, such as contrast changes and noise that were not present in the training set (Geirhos et al., 2018). An improvement to the work of Huang and Belongie (2017) is proposed by Jackson et al. (2019) and employed by Park et al. (2021) for the training of neural nets on the SPEED+ dataset.

Evaluating robustness

Once the networks are trained on the synthetic images the robustness to the real images, a different domain, needs to be evaluated to validate the performance of the model for an actual space mission. The following two methods can be used to do so and have been applied to validate the robustness of CNNs (Black et al., 2021; Park et al., 2021; Pasqualetto Cassinis et al., 2021b).

1. **The use of real images:**
   The usage of real images captured during previous missions, such as OSIRIS-REx, can be used to evaluate the robustness of the CNNs, however, the amount of available images, the diversity of poses, and

variability of environmental factors, such as illumination conditions is insufficient to be able to extensively evaluate the neural network's robustness. Moreover, these images are generally sensor-specific and often lack detailed pose annotation resulting in the inability to accurately evaluate the performance.

2. **On-ground Hardware-in-the-Loop (HIL) set-ups:**
   The alternative approach to validate the networks performance would be to recreate and simulate the space environment on-ground in laboratories, such as the Orbital Robotics & GNC Laboratory (ORGL) at the European Space Research and Technology Center (ESTEC) and the Testbed for Rendezvous and Optical Navigation (TRON) at DLR and SLAB. This approach has been applied by Park et al. (2021), Sharma and D'Amico (2019), and Pasqualetto Cassinis et al. (2021b) to generate realistic imagery of satellite mock-ups of the PRISMA and Envisat spacecraft, respectively, to train neural nets that tackle the problem of pose estimation of uncooperative satellites. Moreover, it was used to verify the image processing algorithm for the HERA mission by Volpe et al. (2020). However, these test-beds need to be carefully calibrated to allow for accurate ground-truth annotation (Park et al., 2021; Pasqualetto Cassinis et al., 2021b). These images were used to validate, on-ground, the robustness of the machine learning networks to the domain gap between "real" images and synthetic images.

Unfortunately, given the time-limit the creation of real images using a HIL-setup is not possible within this work. Based on the aforementioned, properly validating the performance of the network on a domain different from the training data (real images) is not possible. Therefore, more advanced techniques, such as unsupervised domain adaptation and domain randomization, to bridge the domain gap will not be required.

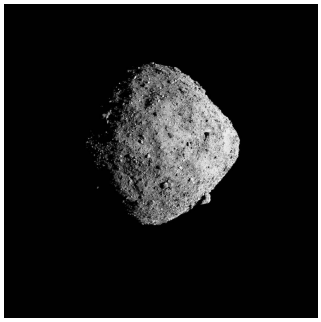
However, the network's robustness against image corruptions representative of real image artifacts can be evaluated through the use of corrupted synthetic validation and test datasets, which should not have the exact same distribution as the training set. This type of analysis has also been applied by Barad (2020) and would serve as an initial evaluation of the robustness, which should be expanded upon in future research to fully allow for validating the neural network to be able to fly in an actual mission. The details regarding the creation of this dataset is discussed in the subsequent Subsection 7.5.1.

7.5.1. Bennu+ dataset

Based on the discussion in the previous section, a dataset will be created that includes image augmentations, called *Bennu+*. This will not only increase the robustness of the trained CNNs against common image corruptions originating from real sensors or the space environment, but these corruptions can also make the network more robust against different textures and illumination conditions (Pasqualetto Cassinis et al., 2021b).

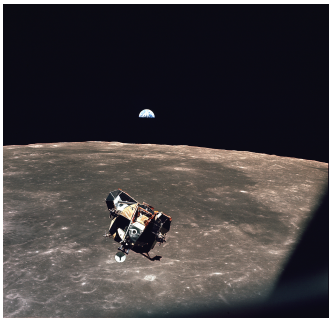
There are many different types of corruptions and the most common corruptions that can occur due to the space environment are related to radiation effects, image saturation due to overexposure or underexposure, blurring effects due to motion, and pluming effects. Furthermore, there are also corruptions that are inherent to real sensors, such as Gaussian and shot noise. The augmentations that are used in the creation of the corrupted dataset try to mimic these aforementioned effects. These corruptions are based on Hendrycks and Dietterich (2019) and are listed below and graphically shown in Figure 7.19.

- **Gaussian blur:** This augmentation mimics depth-of-field.
- **Motion blur:** Motion blur occurs when the camera moves quickly during the exposure time.
- **Defocus blur:** This augmentation mimics that the image is out of focus.
- **Zoom blur:** This augmentation mimics what a camera would capture when the spacecraft moves towards the target along the camera's optical axis with a significant velocity.
- **Spatter:** This augmentation emulates liquid condensation effects on the camera that can arise due to pluming events.
- **Gaussian noise:** Gaussian noise is statistical noise that follows a normal distribution and occurs during the creation of digital images. It can be caused by poor illumination and high internal temperatures in the sensor.
- **Impulse noise:** Impulse noise is also referred to as salt and pepper noise and can be caused by radiation. This results in some pixels being *dead pixels* (0) and some pixels being *hot pixels* (255). Thereby rendering the information of that pixel useless.
- **Shot noise:** Shot noise is inherent to any Charge-coupled device (CCD) camera, which is commonly used on spacecraft as well. This type of noise is associated with photons and the conversion of those photons into electrons within the camera.



(a) An image taken of Bennu by the MapCam of the OSIRIS-REx spacecraft. Source: NASA[12]

(b) An image taken of the Hubble space telescope. Source: NASA[13]



(c) An image taken during the Apollo mission. Source: NASA[14]

Figure 7.16: Real space images illustrating the lack of stars

- **Speckle noise:** This noise is similar to Gaussian noise, however, the noise is additive, meaning that the noise is larger when the original pixel intensity is larger.
- **Color jitter:** This augmentation randomly changes the brightness, saturation, and contrast of the images and can therefore simulate a variety of optical conditions, such as adverse illumination conditions (overexposure/underexposure).
- **Random erase**: This augmentation randomly selects a rectangular part of the image with a certain aspect ratio and sets those pixels to zero to reduce the network's reliance on certain features.

Background augmentation

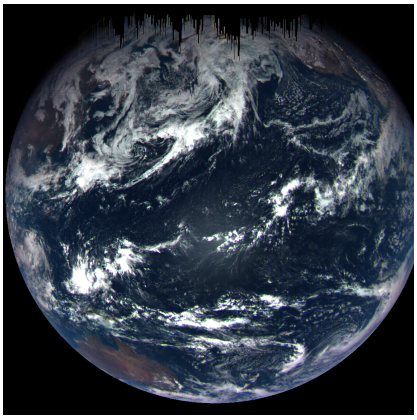
Furthermore, apart from these image corruptions, another augmentation could be the inclusion of a background. A commonly used background augmentation for satellites would be the Earth, however, this is not relevant for a mission to an asteroid. Another background augmentation that could come to mind would be distant stars. However, in actual space imagery of the asteroid Bennu, the Moon, and of a satellite as shown in Figure 7.16, no stars are present. This is because distant stars are extremely faint and due to the relatively short exposure time used by the cameras, they are not captured within the image. Longer exposure times allow to capture more light, resulting in the ability to detect faint stars. The brighter the object, the shorter the required exposure time, however, the cameras used in space missions are optimized for the objects that they encounter and want to study. The MapCam used to capture images of Bennu by OSIRIS-REx was designed to map the surface of the dark asteroid and was consequently overwhelmed when making an image of the bright Earth, resulting in the image artifacts seen in Figure 7.17a. Furthermore, the Hubble telescope wants to study distant stars and galaxies as shown in Figure 7.17b and has dedicated camera parameters for that purpose. Therefore, based on the camera properties used for asteroid missions a star-field background augmentation is not considered further. However, the star-field shown in Figure 7.17b could be used as an adversarial texture in future research.

Augmented dataset creation

The algorithms used to corrupt the images are taken from the software repository[15] created by Barad (2020), which is an adaptation of the repository[16] created by Hendrycks and Dietterich (2019). Furthermore, the procedure proposed by Barad (2020) is adapted, as this allows for a hierarchical application of the corruptions, resulting in realistic images.

Figure 7.18 illustrates the process of creating the augmented dataset. Firstly, a subset of images are randomly selected with a probability $P(C)$ to ensure that a minimal of x% of the images are clean. The remainder of the images will be sent to the augmented images pipeline. Firstly, an image gets applied random erase with a probability $P(RE)$ after which color jitter is applied to this image with a probability of $P(CJ)$. Following this, one of the blurs is randomly applied to this image with a probability of $P(B)$ after which one of the noise

[12] `https://www.asteroidmission.org/galleries/spacecraft-imagery/`, Date accessed: 4-11-2021
[13] `https://www.nasa.gov/mission_pages/hubble/multimedia/index.html`, Date accessed: 4-11-2021
[14] `https://moon.nasa.gov/resources/56/eagles-return/?category=images`, Date accessed: 4-11-2021
[15] `https://github.com/kuldeepbrd1/image-corruptions` Date accessed: 4-11-2021
[16] `https://github.com/hendrycks/robustness` Date accessed: 4-11-2021



(a) An image taken of Earth by the MapCam of the OSIRIS-REx spacecraft showing the image artefacts (top) due to the brightness of the Earth. Source: NASA[12]

(b) An image taken of distant stars and galaxies by the Hubble space telescope. Source: NASA [13]

Figure 7.17: Demonstrating the effect of the designed purpose of the camera on what it captures

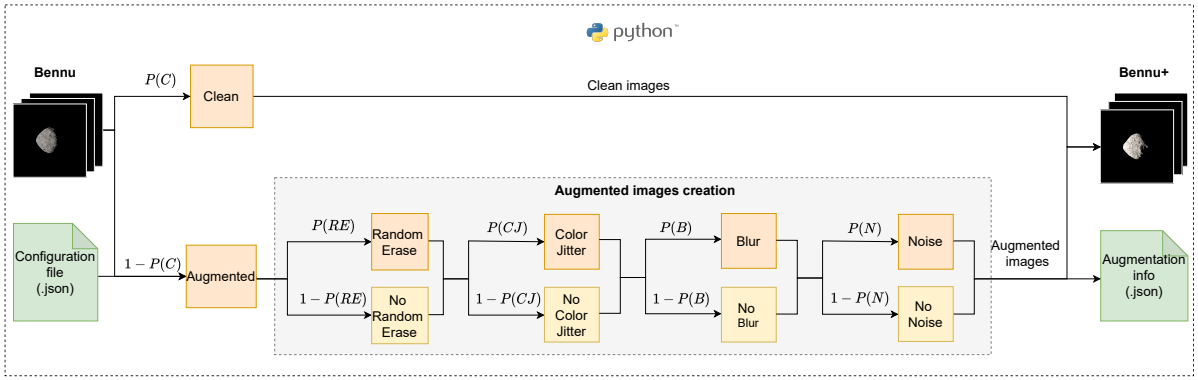


Figure 7.18: A schematic overview of the creation of the augmented dataset

augmentations is randomly applied with a probability of $P(N)$. The spatter augmentation is included in the blurs. This results in an image being able to have up to four augmentations applied, however, never more than one blur or noise effect. It could also happen that the image that is ran through the augmentation pipeline remains clean, i.e., no augmentations applied.

This ordering of the different augmentations and the clustering of the different blurs and noises is performed to create realistic augmented images. Firstly, random erase and color jitter are applied, as these are general augmentations that are intended to make the network invariant to either certain missing information (random erase) or illumination conditions (color jitter). The blurring and noise effects are real sensor artifacts and would influence the pixels regardless of the brightness/contrast change that can arise due to environmental effects (color jitter) or random erase.

Furthermore, as there are multiple corruptions of the same type, such as noise and blur, randomly assigning the corruptions could result in images having multiple different types of noise and blur, thereby creating images that are not representative of realistic corrupted imagery. The ordering and clustering of the different types of noises and blurring results in images in which the effects are fairly randomized, while having realistic and varying combinations of effects.

A corrupted training and validation set is created. The test set of the *Bennu-clean* dataset is used to evaluate the performance of the networks trained on the corrupted dataset *Bennu+* to clean images, i.e., without augmentations. The distribution of the number of corruptions applied to each image is listed in Table 7.3 and the total number of corruptions per type are shown in Table 7.4.

Table 7.3: Listing the distribution of the number of augmentations applied on the images within the *Bennu+* dataset

Number of augmentations	**Train+**		**Val+**		**Test-clean**
0 (clean)	8762	≈ 39%	1819	≈ 37%	4853
1	5614	≈ 25%	1205	≈ 25%	-
2	5767	≈ 25%	1296	≈ 27%	-
3	2271	≈ 10%	492	≈ 10%	-
4	232	≈ 1%	41	≈ 1%	-
Total	**22646**	**100%**	**4853**	**100%**	**4853**

Table 7.4: Listing the number of images per corruption type

Type of augmentation	**Train+**	**Val+**	**Test-clean**
None (clean)	8762	1819	4853
Random erase	1902	339	-
Color jitter	7133	1699	-
Gaussian blur	1683	368	-
Motion blur	1728	398	-
Zoom blur	1732	414	-
Defocus blur	1776	425	-
Spatter	1801	434	-
Gaussian noise	1806	359	-
Impulse noise	1761	346	-
Speckle noise	1727	350	-
Shot noise	1840	305	-

The probabilities P applied for each case in Figure 7.18 are specified through a configuration file and can be found in Appendix D. These were deemed sufficient in generating a proper corrupted dataset. The main idea is to have a training and validation set with a slightly different distribution of corruptions to evaluate robustness. Moreover, the augmentation applied to each image is unique to mimic real world corruptions, which also show variation of the corruption values even at fixed levels of intensity. This also ensures that not only the distribution of augmentations differs between the training and validation set, but also the corruption value.

The validation set consists of synthetic images with a certain corruption distribution created in this work. Therefore, tuning the probabilities of the augmentations for the training set, to improve performance on the created validation set, is purely subjective to the distribution of that validation set. This tuning of augmentations is therefore not performed. However, when a more standard validation set consisting of real images was to be used to evaluate robustness, then the distribution of augmentations of the synthetic training set can be researched to find the one resulting in the best performance on the validation set.

The severity levels used in the generation of these different types of corruptions have been based on values used by Barad (2020) and Hendrycks and Dietterich (2019) and are representative of real camera/sensor corruptions. These are detailed in Appendix D.

7.6. Trajectory generation

The performance of the algorithm can be tested on image sequences representative of three different trajectories. The most important orbital elements of these trajectories are listed in Table 7.5, where a refers to the semi-major axis, e to the eccentricity, and i to the inclination. These trajectories are selected to cover a variety of scenarios, namely retrograde and prograde orbits and a polar orbit. The retrograde and prograde orbits are w.r.t. the assumed rotation throughout this work (positive rotation around Z, with +Z up). Furthermore, one orbit is at a distance of 8 km. This is used to evaluate whether the network is able to interpolate on data not explicitly present in the dataset.

However, in this work the CNN-based feature extractor is not incorporated into a navigation architecture

Table 7.5: The three different trajectories and their most important aspects

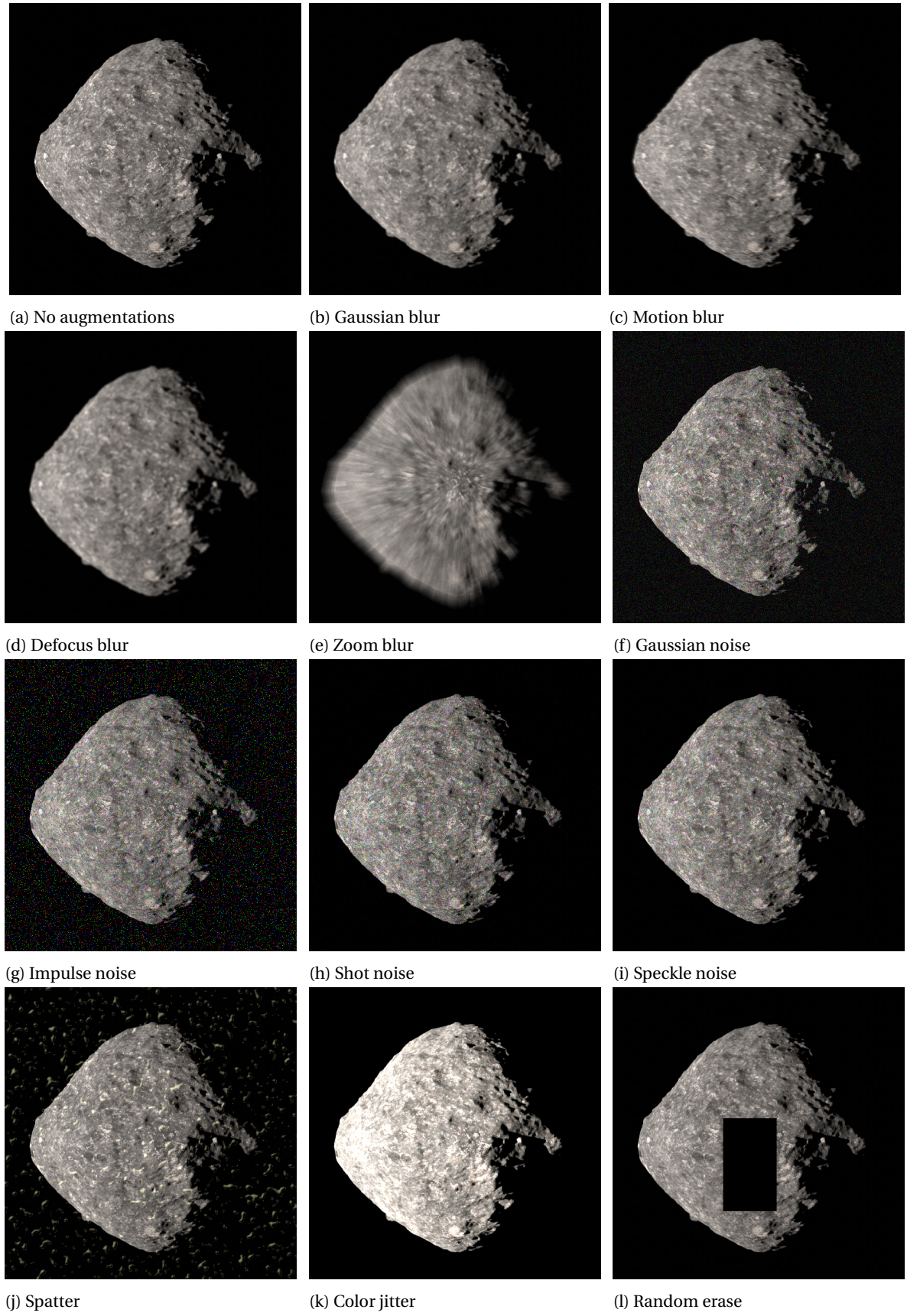
Scenario	**Orbit**	a [km]	e [-]	i [deg]	**CoM pointing**
1	Retrograde	6	0	0	Yes
2	Polar	7.5	0	88	Yes
3	Prograde	8	0	0	Yes

with a state estimator. Therefore, high realism taking into account the velocity of the spacecraft, the sampling time between images, and the rotation of the asteroid is not required and is left to future research. These image sequences are merely used to showcase the algorithms performance to sequences it could encounter in an actual orbit. These trajectories alongside their ground-truth data have been rendered using Blender and post-processed in the same way as the normal dataset. The scripts used to generate the trajectory will be made available in the software repository[17]. The image sequences do not contain corruptions.

7.7. Dataset API

The synthetic dataset has been created using Blender and the data has been processed and annotated using Python. The scripts used to generate the dataset are made available on Github[17] by the author . These scripts are generic/modular and can be used to generate synthetic datasets suitable for deep-learning purposes of other target bodies as well. The `datasetGeneration.py` script can be used to generate the datafiles, which are used to place the camera in Blender. The Blender scripts are generic and can be used to generate synthetic datasets of other target bodies as well, as long as a textured 3D model is given. Furthermore, an extensive utility file is created (`dataset_utils.py`) that allows a user among others to transfer the raw datafiles from Blender (pose, bounding box) into the required format, partition the dataset into training, validation, and test, annotate and visualize the data through the use of simple function calls. This utility file is not exclusive to the Bennu dataset, but can be applied to any dataset that uses the same file formatting. The `dataset_processing.py` script is used to process the raw-data originating from Blender into a fully annotated dataset with exported files in their desired formats (e.g., COCO) and file locations. An example of this file and the usage of functions is shown in Code Listing C.6.

[17] `https://github.com/lvanderheijden`



(a) No augmentations (b) Gaussian blur (c) Motion blur

(d) Defocus blur (e) Zoom blur (f) Gaussian noise

(g) Impulse noise (h) Shot noise (i) Speckle noise

(j) Spatter (k) Color jitter (l) Random erase

Figure 7.19: The different image corruptions that have been incorporated into the augmented *Bennu+* dataset. The severity for some augmentations have been increased for visibility

8

Object detection network

This chapter discusses the object detection network and the implementation. Firstly, an introduction to object detection is given in Section 8.1, followed by the selection of the architecture that will be used in this work in Section 8.2. The chosen network will then be discussed in more detail in Section 8.3. The implementation of the network is discussed in Section 8.4, after which the configuration is elaborated upon in Section 8.5, demonstrating the settings used to achieve the results on the datasets.

8.1. Object detection

The problem of object detection can be formulated as the detection and classification of objects within an image. The network regresses the coordinates of the bounding box surrounding the target and classifies the object present within that bounding box. The onset of deep learning has greatly improved the object detection pipelines, outperforming traditional hand-engineered methods. These hand-engineered methods had to be manually tweaked, which becomes cumbersome when dealing with all kinds of variations of the appearance of the object due to scale, illumination, occlusion, and noise. Furthermore, the classification of objects required much more information per class, resulting in extensive tweaking to achieve desirable performance, and these methods tend to fail on noisy and occluded data. The object detection field is heavily researched within machine learning and object detection networks are among the most complex machine learning models.

8.2. Architecture selection

As discussed in Section 6.2, it is best practice to use an open-source implementation of a network architecture that has been optimized for object detection. The most suitable network is selected based on the notion that the architecture should have *low memory usage* and *FLOPs* allowing for future embedding in the spacecraft processing unit as discussed in Section 2.3. This is to make the application of machine learning more realistic. The research regarding object detection applies to a wide range of objects, but also focuses on detecting person instances within images, however, the concept remains the same.

The selection of the CNN architecture determines the accuracy of the object detections. The state-of-the-art object detection networks can generally be divided into *two-stage* detectors and *single-stage* detectors. The *two-stage* detectors rely on *region proposals* to generate RoIs from feature maps after which these region proposals are used to classify the object within the RoI and regress the bounding box coordinates. These models achieve highly accurate detections, however, because of their two-stage approach they have a higher inference time, i.e., they are slower. Examples of two-stage detectors are Faster R-CNN (Ren et al., 2016) and Mask R-CNN (He et al., 2017). The *single-stage* detectors directly regress the bounding box coordinates and output the probability that an object belongs to a certain class. These networks are highly efficient and much faster than the *two-stage* detectors, however, they generally are less accurate. Examples of single-stage detectors are the You Only Look Once (YOLO) type algorithms (Redmon and Farhadi, 2018; Redmon et al., 2016) and Single Shot MultiBox Detector (SSD) (Liu et al., 2016).

Regardless of the approach, the object detection networks generally consist of two distinct parts, namely a *base network*, which is also referred to as the backbone, which is used for generic feature extraction as discussed in Section 5.3. These backbone networks are often used for classification problems and have been

trained on enormous datasets, such as ImageNet (Deng et al., 2009). Commonly used backbone networks are ResNets (He et al., 2016), VGG (Simonyan and Zisserman, 2014), Inception (Szegedy et al., 2017), and MobileNets (Howard et al., 2017; Sandler et al., 2018). The second part is the *detection head*, which is used to regress the bounding box coordinates and to classify the objects within the corresponding bounding box, which follows the aforementioned one or two-stage approaches. Some recent object detection networks also use an additional neck section, which serves as an intermediate layer between the backbone and the detection head. This layer combines feature maps from different layers (multiple scales), as different features can be found in different scales (Figure 8.5).

The selection of a certain backbone, neck, and detection head is not trivial and therefore fixed combination object detection networks are used that have been proven to work effectively. The majority of the networks aim to maximize the performance on a certain performance metric, such as the accuracy on certain challenging object detection datasets, such as Common Objects in Context (COCO) (Lin et al., 2014). However, for the actual application of machine learning to space missions not only the performance is important, but also the computational effort and the required memory as outlined in the requirements. Huang et al. (2017) performed an extensive speed/accuracy trade-off of different backbone and detection head combinations. Furthermore, several other combinations have been tested on challenging datasets and implemented within the TensorFlow Model Zoo[1].

As was previously mentioned, the backbone usually consists of state-of-the-art image classifying CNNs, such as ResNet (He et al., 2016) and VGG-16 (Simonyan and Zisserman, 2014). These networks consist of tens or even hundreds of millions of parameters and use normal convolutions, which affects the memory usage and execution time, limiting real-time applications on embedded systems. Howard et al. (2017) proposed MobileNets, which were specifically designed for mobile and embedded vision applications. They focused on less memory usage and lower latency by using less parameters (Section 5.4) while only resulting in a minor loss in the performance with respect to the much larger state-of-the-art CNNs (Sandler et al., 2018).
Soviany and Ionescu (2018) showed that for relatively easy images a single-shot detector, combining MobileNet with a SSD detector head, performs equally well compared to the more complex and slower Faster R-CNN. The definition of easy images refers to single objects, no challenging background, or heavily occluded objects in cluttered scenes. The datasets used within this work are considerably less challenging compared to the datasets the existing object detection architectures have been trained and evaluated on (COCO). These datasets contain a plethora of different object classes with images containing multiple objects and often in heavily occluded scenes. The Bennu datasets consist of a single class object on a black background (underlid starfields) with little to no occlusion. Therefore, the most important consideration is the efficiency of the network, as the reported accuracies are achieved on these much more challenging datasets. Based on the aforementioned, the one-shot detector, SSD-MobileNet combination is adapted within this work due to its computational efficiency. Furthermore, Park et al. (2019) and Barad (2020) used a single shot detector, YOLO and SSD, respectively, with a MobileNet architecture as their backbone within their uncooperative spacecraft pose detection pipelines.

This object detection architecture was improved upon by adding a Feature Pyramid Network (FPN) neck to the SSD detection head making it a multiscale network, which is more robust to image corruptions as proven by Hendrycks and Dietterich (2019). The *final architecture* therefore consists of the SSD detection head with the FPN neck and the MobileNetV2 backbone, where all the convolutional layers of the different parts have been replaced with depth-wise separable convolutions (Section 5.4) resulting in a highly optimized network architecture at a minimal loss in accuracy. This architecture is referred to as **SSD-FPN-MobileNetV2-Lite**, where the Lite refers to the depth-wise separable convolution layers. The basic network architecture is available through the TensorFlow Model Zoo[1] and is discussed in Section 8.3.

8.3. SSD-MobileNetV2-FPN-Lite

The SSD-MobileNetV2-FPN-Lite model consists of three distinct parts, the MobileNetV2 network as the generic feature detector, the FPN as the neck, and the SSD network as the detection head. As discussed before, each convolutional layer within the network has been replaced by depth-wise separable convolutional layers resulting in a substantial decrease in the number of parameters and FLOPs, therefore referring to it as Lite.

The exact number of parameters and FLOPs for this model are not reported, nor does the software through which it is available, TensorFlow Object Detection API (Section 8.4), at the moment of writing, enables the cal-

[1] `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`, Date accessed: 31-12-2021

Table 8.1: The number of parameters and FLOPs for the SSD-MobileNetV2-Lite model (Sandler et al., 2018)

Network	**Input size** ($w \times h$) **[px]**	**Parameters [Mn]**	**FLOPs [Bn]**
SSD-MobileNetV2-Lite	320×320	4.3	0.8

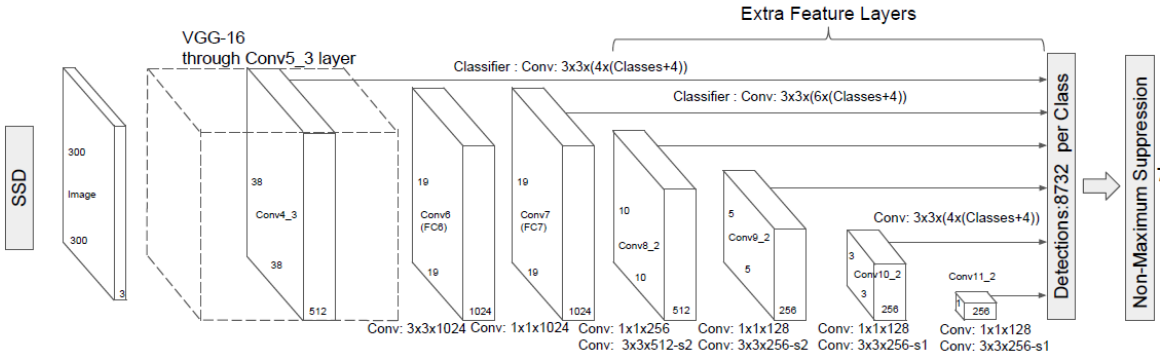


Figure 8.1: The architecture of the standard SSD detector with the original VGG-16 backbone (Liu et al., 2016)

culation of this. However, the parameters and FLOPs of a similar model, the SSD-MobileNetV2-Lite, with the same input size, are summarized in Table 8.1. This gives an indication of the relative size and computational efficiency. The addition of the FPN neck is not expected to increase these reported numbers too much, which is reinforced by comparing the reported inference times of 19 ms and 22 ms[1], respectively, for the model with and without the FPN neck.

The building blocks and general workings of MobileNetV2 have been explained in Section 5.4, the FPN and SSD are discussed in more detail in this section.

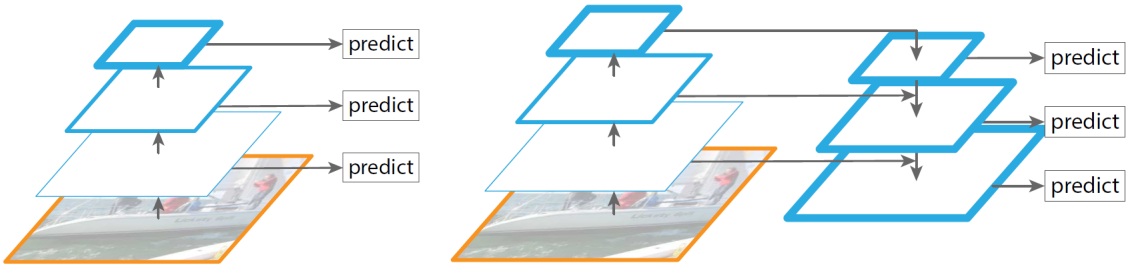
The original implementation of the SSD detector devised by Liu et al. (2016) adds multiple convolutional layers to the backbone network to detect objects as shown in Figure 8.1. These convolutional layers reduce the size of the image, i.e., decrease the resolution, from feature maps that have a size of 38×38 to feature maps of 3×3, thereby allowing for the detections of objects across multiple scales. The feature maps can be viewed as a grid that is placed over the image as shown in Figure 8.5, where the lower resolution feature maps are used to detect larger objects. The SSD network relies on the use of default boxes with different scales and aspect ratios called *anchor boxes*, which were first proposed by Ren et al. (2016) in Faster R-CNN. The aspect ratio of these anchor boxes are pre-selected and it is recommended to use between four and six different aspect ratios to be able to cover a variety of different object shapes. These anchor boxes are applied to each feature cell and their position relative to the feature cell is fixed, where the feature map resolution determines the scale of the anchor box as shown in Figure 8.5.

The network moves over every feature cell and outputs a probability score ($\mathbf{p}_c$) that the object's midpoint of a certain class is present within that feature cell, and it outputs the 4-vector encoded offset ($\mathbf{t}$) relative to the default anchor box (Equation (8.2)). Therefore, the total output for any given layer with a certain feature map size $m \times n$ is given below, where $\#A$ represent the number of default anchor boxes and c represents the number of classes.

$$m \times n \times \#A \times (c+4) \tag{8.1}$$

Whenever there are multiple layers (feature map sizes) used for detection this increases the output size accordingly. The feature maps of different layers are combined to form the final predictions in a manner as shown in Figure 8.2a.

The original implementation of the SSD does not use the low-level layers of the backbone network for detections. It only starts from one of the last few convolutional layers of the backbone and the added new layers to create the pyramidal feature hierarchy as shown in Figure 8.1. However, this misses the opportunity to reuse the higher resolution maps of the feature hierarchy, which were proven to be important for detecting small objects (Lin et al., 2017). The FPN neck is used to ameliorate this as it combines low-resolution semantically strong features with high resolution semantically weak features through a top-down architecture with lateral connections as shown in Figure 8.2b. The top-down architecture upsamples the low-resolution



(a) Pyramidal feature hierarchy (b) Feature Pyramid Network (FPN)

Figure 8.2: Different detection architectures where the thickness of the blue line represent semantically stronger features (Lin et al., 2017)

semantically strong features to the required spatial size, whereas the lateral connection adds the same size feature map from the bottom-up path. This results in having semantically strong features at all feature map sizes, and results in more precise locations of those features, as only using the top-down approach would result in semantically strong features, but the locations would be inaccurate, because of the several down- and upsamplings of those features. The semantic meaning of the features is based on the notion that low resolution layers have more semantic meaning, where low resolution refers to the layers towards the end of the network. The initial layers have a high resolution and as such detect more low-level features, such as edges or lines that do not really have semantic meaning. The number of convolutional layers to which the FPN process is applied can be adjusted.

The combination of SSD with FPN allows the network to combine the predictions of all the anchor boxes with the different aspect ratios from all the different locations of the different scale feature maps. This results in a diverse set of predictions, which are capable of detecting a variety of object sizes and shapes.

8.4. Implementation

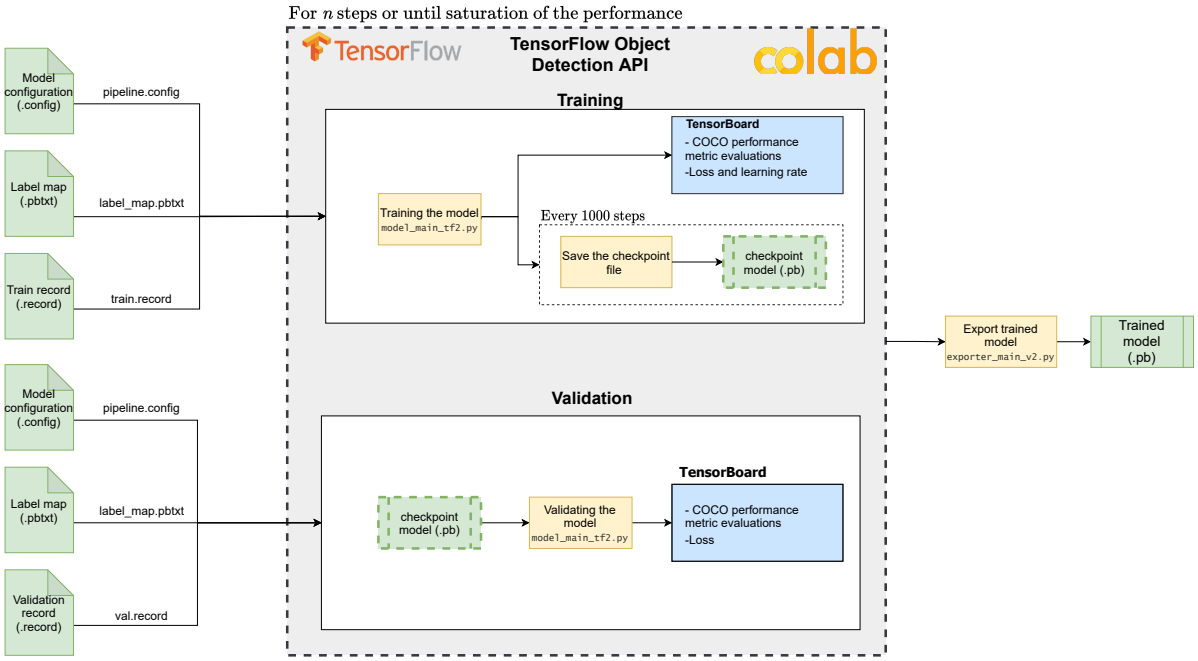
The SSD-MobileNetV2-Lite network is available on GitHub in the TensorFlow 2 Detection Model Zoo[2]. The network has been created using TensorFlow, which is an open-source machine learning platform built and maintained by Google Brain, which uses a Python API. The TensorFlow Object Detection API[3] created by Huang et al. (2017) is used, as it allows for a structured approach in adapting the object detection network to detecting asteroids. This API allows a user to more easily create and adapt object detection pipelines using existing networks, while also containing a variety of functions that are useful for the object detection training and evaluation process. This API is maintained and constantly updated. Furthermore, it allows the adaptation of certain parameters and settings of a given network, such as the input size, optimizer, learning rate, and mini-batch size through the use of a configuration file, which is in the `.config` format. Within this work, TensorFlow 2.5 has been used and the TensorFlow Object Detection API is compatable for any TensorFlow 2.x version. Detailed information regarding the installation of both can be found in the documentation[3].

As discussed in Section 6.2, the network is trained and evaluated using Google Colaboratory (Colab) and the NVIDIA Tesla P100-PCIE-16GB GPU. The model architecture, which specifies the number of convolutional and pooling layers and other specifications is fixed. This architecture did not have to be adjusted and only the input to the network, such as the label map and the `.TFRecord` file, had to be created to fit the Bennu datasets (Section 8.5). Furthermore, the configuration of the network specifying aspects, such as the optimizer and learning rates, was adapted, which is discussed in detail in Section 8.5.

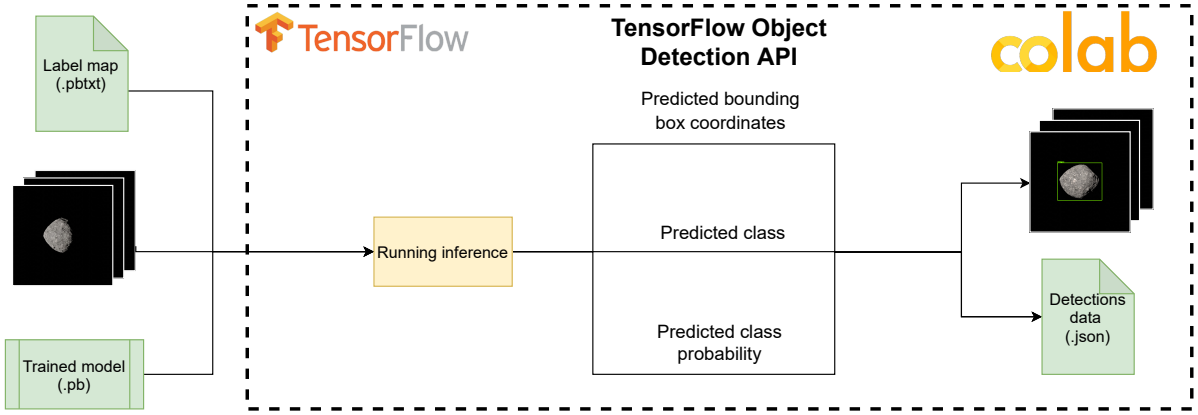
The training and validation have to be run in parallel, as the TensorFlow Object Detection API treats validation as an independent process. The dataset to use for each, can be specified in the configuration file (Code Listing C.4) under `train_input_reader` and `eval_input_reader`, respectively. As illustrated in Figure 8.3a, during training the model outputs a checkpoint file every 1000 steps of the current state of the model, which contains that current model's weights and biases. The validation pipeline will then wait for these checkpoint files and uses them one by one to validate the performance of the model on the validation dataset using the

[2] `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`, Date accessed: 11-10-21

[3] `https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html`, Date accessed: 25-11-2021



(a) The training process, demonstrating that training and validation are run in parallel



(b) Inference

Figure 8.3: Graphically illustrating the implementation of the training and inference process, showing the inputs and outputs

COCO dataset performance metrics. The progress of training and the performance of the model on both the training and validation set can be tracked through TensorBoard[4], which is a graphical interface with which performance metrics and other parameters can be easily visualized during training, validation, and testing. This process is repeated for n steps or until the performance has saturated and the best model is then exported as the final *trained* model. This final trained model is then used for inference (testing) using Google Colab or a local machine. Furthermore, the checkpoint files also allow the network to start (re)training from such a checkpoint.

Running the model in inference is an independent process and requires the trained model. This then makes predictions about the bounding box coordinates, class, and the confidence associated with that class, which are used for evaluation. This process is graphically shown in Figure 8.3b. Furthermore, these predictions are fed to the keypoint detection network. The object detection network is trained and optimized in isolation and eventually incorporated into the pose estimation pipeline.

[4] `https://www.tensorflow.org/tensorboard/get_started`, Date accessed: 25-11-2021

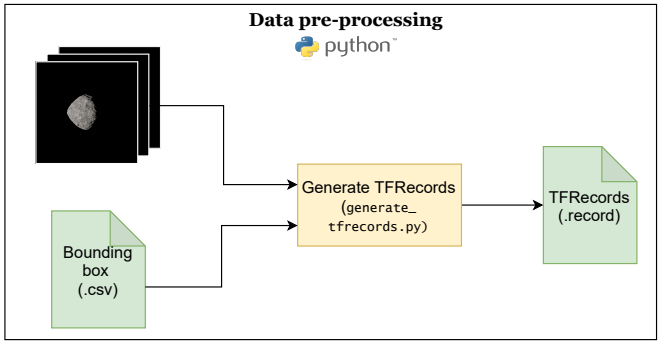


Figure 8.4: The pre-processing required for training the network

8.5. Configuration

The model architecture, which specifies the number of convolutional and pooling layers and other specifications, is fixed. However, certain hyperparameters need to be set, adapted, or tuned as well as training settings, such as selecting the loss function and optimizer. These parameters and settings specify the configuration of the network and the most important aspects will be discussed in more detail in this section. An example configuration file is given in Code Listing C.4.

Input and output

The input to the object detection network is an image rescaled to size 320 × 320 px, which is slightly larger than the input size of the subsequent keypoint detection network to minimize loss of information. The input size directly effects the computational demand of the network, therefore, the aforementioned image size is used, which is in accordance with the specification of the model (Liu et al., 2016). The ground-truth annotations as discussed in Subsection 7.4.3 are firstly transformed to a standard format containing the class and the bounding box encoding $(x_{min}, y_{min}, x_{max}, y_{max})$ (Figure 7.14). During pre-processing this file format is converted to `.TFrecord` as shown in Figure 8.4, which can be read by TensorFlow. Furthermore, a label map is created (`.pbtxt`), which simply links the class label to an integer value. Within this work only one object/class is detected, namely the asteroid. This label map is used by the TensorFlow Object Detection API for training and detection.

As discussed in Section 8.3, the network moves over every feature cell and outputs a probability score ($\mathbf{p}_c$) that the object's midpoint of a certain class is present within that feature cell, and outputs the 4-vector encoded offset ($\mathbf{t}$) of the ground-truth bounding box relative to a default anchor box. The feature maps of different layers are combined to form the final predictions. The final output of the network is transformed to the following format for the bounding box encoding $(x_{min}, y_{min}, x_{max}, y_{max})$ alongside a classification probability and the allocated class to the object within the bounding box.

Bounding box encoding

The 4-vector encoded offset ($\mathbf{t}$) is associated with the default anchor boxes and as discussed in Section 8.3, the default anchor boxes are applied to every feature cell within the feature map in a convolutional manner. The position of each anchor box relative to the corresponding feature cell is fixed, where the anchor boxes have a certain midpoint, aspect ratio, and scale. As aforementioned, at every feature cell, the network tries to learn and predict the offset of the ground-truth bounding box relative to the default anchor box. Thereby, learning a transformation that maps the default anchor box to the ground-truth box. The difference between the actual and predicted offset of the ground-truth bounding box and the allocated anchor box is minimized during training through the loss function. During training it needs to be established which default anchor boxes correspond to the ground-truth bounding box to train the network accordingly. However, the matching strategy of allocating a certain anchor box to the ground-truth box should be done carefully.

Whenever a default anchor box is selected that does not closely resemble the ground-truth box, the task of transforming this anchor box to the desired ground-truth box does not make sense and would lead to a highly inefficient learning problem. Therefore, care should be taken in allocating anchor boxes to the ground-truth box for optimization. Girshick et al. (2016) propose a matching strategy in which the anchor box with the highest Intersection over Union (IoU) (Figure 8.7) above a certain threshold (e.g., 0.5) is assigned

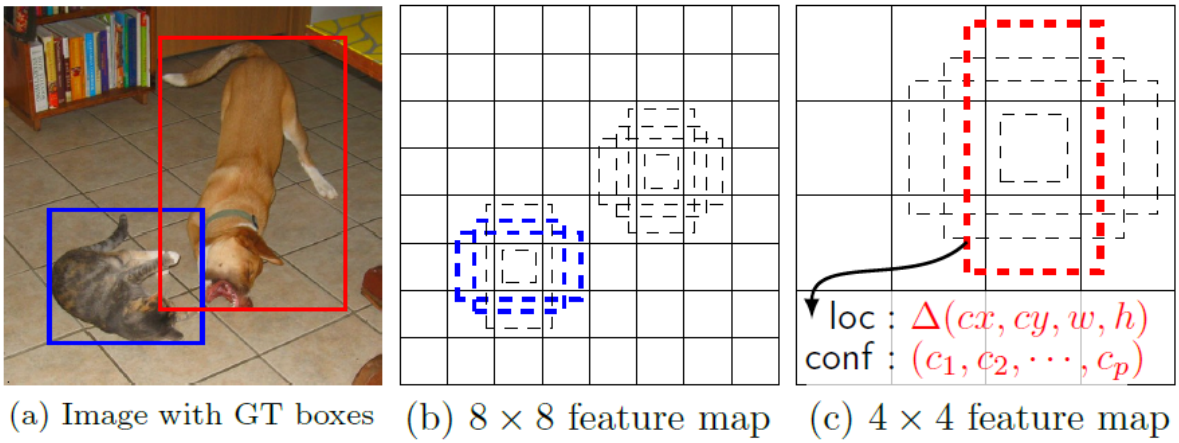


Figure 8.5: The visualization of the matching of default anchor boxes in a specific feature cell to the ground-truth bounding boxes on a 4×4 and 8×8 feature map (Liu et al., 2016)

to the ground-truth box. The rest of the anchor boxes are discarded. However, the learning problem can be simplified and better performance can be achieved by allowing multiple anchor boxes to be assigned to the ground-truth box, as long as they have an IoU value above a certain threshold (Liu et al., 2016). The network can then learn to predict the offsets and scores for multiple overlapping anchor boxes. Therefore, it is proposed to use this approach, which is in line with Liu et al. (2016). This concept of allocating anchor boxes to the ground-truth box is graphically illustrated in Figure 8.5, where the blue and red anchor boxes refer to the allocated boxes for the respective objects and the gray ones refer to negative matches. The majority of the anchor boxes are negative matches as can also be observed in Figure 8.5. During inference the network will output one final prediction using Non-Maximum Suppression (NMS), which is discussed in detail later.

The parameterization of the bounding-box offset used in R-CNN (Girshick et al., 2016) is proposed to improve the efficiency of the offset optimization. This parameterization results in a standard regularized least-squares problem, which can be efficiently solved. Furthermore, this parameterization of the offset results in a scale-invariant translation of the center of the anchor box and a log-space translation of the width and height of the anchor box. This parameterization is also applied by Liu et al. (2016) and Ren et al. (2016), and the 4-vector encoded box representation is given by:

$$t_x = \frac{x_c^G - x_c^A}{w^A} \quad t_y = \frac{y_c^G - y_c^A}{h^A} \quad t_w = \log\left(\frac{w^G}{w^A}\right) \quad t_h = \log\left(\frac{h^G}{h^A}\right) \tag{8.2}$$

The standard format for the ground-truth bounding boxes $(x_{min}, y_{min}, x_{max}, y_{max})$, which can be observed in Figure 7.14, is firstly transformed to a centroid representation (x_c^G, y_c^G, w^G, h^G) to align with the anchor box representation (x_c^A, y_c^A, w^A, h^A), where the superscripts G and A are added for clarity representing the ground-truth and anchor box respectively. The prediction of the offset by the network can be represented by replacing t and G with $\hat{t}$ and $\hat{G}$ respectively in Equation (8.2).

Anchor boxes

The dataset consists of images from a variety of different distances and viewpoints. Therefore, it is crucial to have anchor boxes that can accurately detect the target at the different relative sizes within the image. As was discussed in Section 8.3, the SSD detection head applies default anchor boxes over multiple scales, i.e., different feature map sizes. The anchor boxes have a certain aspect ratio and scale, where, when using the FPN neck, the scale is specified in relation to the current layer's stride length. The stride of the convolutional operation influences the resulting size of the feature map as was discussed in Section 5.3. The influence of the feature map resolution on the scale of the anchor box can be observed in Figure 8.5, where the 4×4 feature map has anchor boxes that are much larger than the ones in the 8×8 feature map.

The anchor boxes have a certain aspect ratio ($\frac{h}{w}$) and they can be properly selected by analyzing the distribution of aspect ratios of the bounding boxes present in the dataset. The width-to-height and height-to-width distribution is shown in Figure 8.6. Based on this six different aspect ratios are selected to cover the entirety of the dataset (0.7; 0.8; 0.9; 1.0; 1.1; 1.2). The anchor boxes can be adjusted within the `anchor_generator` part

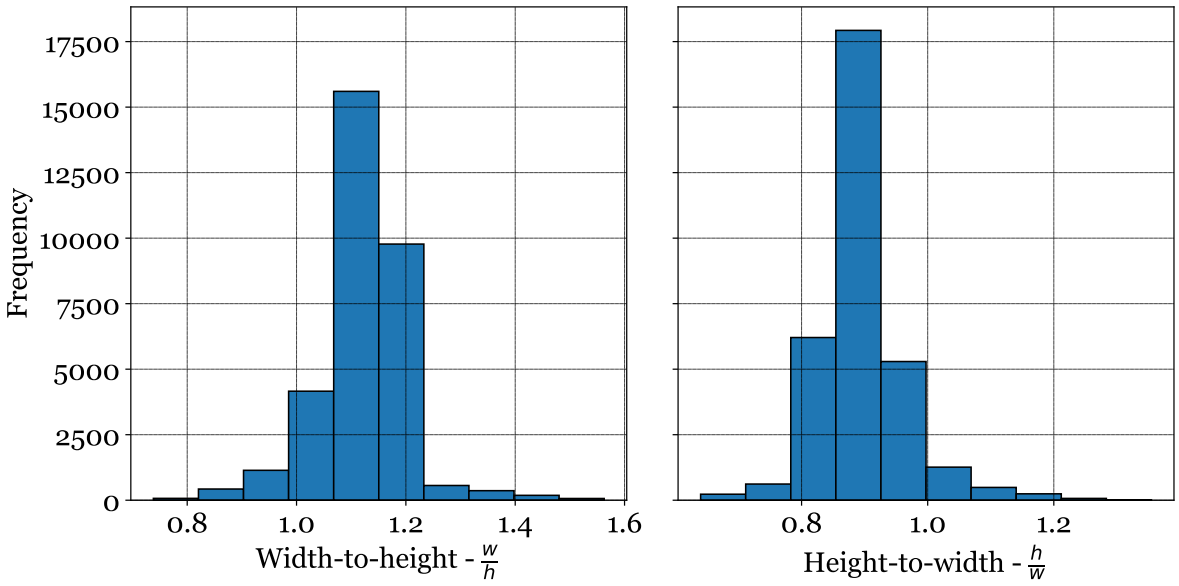


Figure 8.6: Distribution of the width-to-height and height-to-width ratios of the ground-truth bounding boxes enclosing the asteroid within the images throughout the entire dataset

of the configuration file Code Listing C.4. Adding more aspect ratios would not necessarily result in better performance, but would hurt the computational cost required. These anchor boxes are applied to four layers, with different scales (resolutions) of the feature map (e.g., 16×16, 8×8, 3×3), i.e., at four different pyramid levels. This process of anchor boxes at different resolutions can be observed in Figure 8.5.

Loss function

As previously mentioned, the object detection focuses on two tasks, namely classification and localization. Therefore, there are two separate loss functions that together make up the multi-task loss L, whose equation is given below, where N represents the number of matched anchor boxes and λ represents the weight of importance given to either subtask, which is set equal to 1 (equal importance).

$$L = \frac{1}{N}\left(L_{cls} + \lambda L_{loc}\right) \tag{8.3}$$

The localization loss is calculated using Equations (8.4) and (8.5). The smooth L_1 loss function is proposed, as this is better suited against outliers and removes the need of careful tuning of the learning rates to avoid exploding gradients (Girshick, 2015). As previously discussed, the network tries to predict the offset $\hat{\mathbf{t}} = \left(t_x, t_y, t_w, t_h\right)$ of the ground truth bounding box relative to the default anchor box. The actual offset ($\mathbf{t}$) is known and the loss functions aims to minimize this difference between the actual and predicted offset.

$$L_{\text{loc}}\left(\hat{\mathbf{t}}, \mathbf{t}\right) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}\left(\hat{t}_i - t_i\right) \tag{8.4}$$

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{8.5}$$

The classification loss is calculated using Equations (8.6) and (8.7) and is used to penalize the wrong class detection for a detected object. The object detection network normally deals with more than one class and then the classification loss is much more important. In this work, however, the detector only has to discern between the asteroid and the background, i.e., determining whether the cell within the feature map contains the object or not. The use of the *weighted sigmoid focal loss* is proposed, as this has been shown to greatly improve the performance of one-shot detectors. This has to do with the nature of one-shot detectors, as they can evaluate up to 10,000 proposed locations, whereas only a fraction contain the actual object and the majority is simply background. This results in the class imbalance problem, resulting in inefficient training, as the majority of those proposed locations are easily classified as background by the detector. Therefore, they

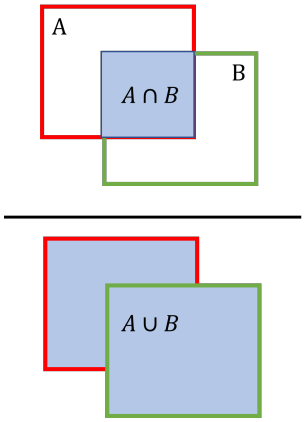


Figure 8.7: Visualization of the IoU performance metric

do not offer any useful information for learning. Furthermore, as the majority of the examples are easy negatives (detections with high probabilities), which have a near zero loss value, they can collectively dominate the loss function, which results in gradients and weights that detiorate the performance of the model. The *weighted sigmoid focal loss* function mitigates this by assigning more weight to hard or easily misclassified examples, and reducing the weight of easy negatives. Thereby reducing the contribution of the easy negative examples and increasing the importance of correcting the misclassified examples.

The balance factor α and the modulating parameter γ are tunable hyperparameters of the *weighted sigmoid focal loss* function. The following values were used $\alpha = 0.25$ and $\gamma = 2.0$, which are in line with the values found to achieve the highest performance (Lin et al., 2018). Furthermore, $\hat{p}_{c,i}$ represents the class probability predicted by the object detection network for the i^{th} default anchor box.

$$L_{cls} = \sum_{i}^{N} -\alpha \left(1 - \hat{p}_{c,i}\right)^{\gamma} \log\left(\hat{p}_{c,i}\right) \tag{8.6}$$

$$\hat{p} = \begin{cases} \hat{p} & \text{if true class} \\ 1 - \hat{p} & \text{if false class} \end{cases} \tag{8.7}$$

Similarity and evaluation metric

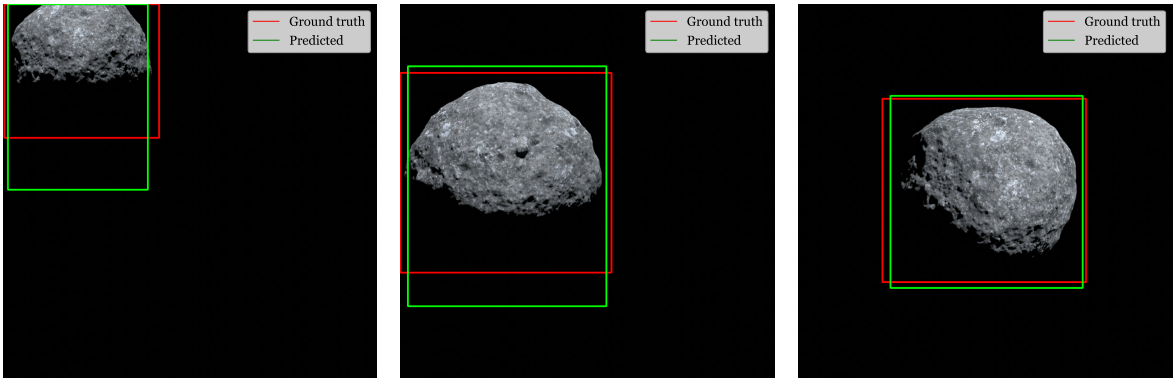
The performance metric that is used for object detection is the Intersection over Union (IoU), which assesses the accuracy of the bounding box detection within an image. The definition of this metric is given by:

$$\text{IoU} = \frac{\text{size of intersection}}{\text{size of union}} = \frac{A \cap B}{A \cup B} \tag{8.8}$$

It evaluates the similarity between the ground-truth bounding box (A) and the predicted bounding box (B). The detection is considered correct if the IoU has a value above a certain threshold, this threshold is set at 0.5 to follow the convention, i.e., IoU ≥ 0.5. However, this requirement could be made more stringent by increasing the value. The detected bounding box is considered perfect if the value of the IoU = 1. The performance over the entirety of the respective dataset is calculated by using the mean and median of the IoU over all the images. The IoU metric is graphically shown in Figure 8.7.

This metric is also used in determining the allocation of anchor boxes to the ground-truth box as mentioned when discussing the bounding box encoding. Three different example IoU detections are shown in Figure 8.8 and as can be observed, the predicted boxes naturally have a slightly different aspect ratio than the ground-truth bounding boxes, caused by the default anchor boxes. Detections with an IoU above 0.75 are considered good and this is also a strict metric for the challenging COCO dataset (Lin et al., 2014).

The detection with the lowest IoU in Figure 8.8 shows a predicted bounding box that is less accurate/tight than the ground-truth bounding box. However, it can be seen that it still accurately detects the asteroid within the image and only preserves a bit more of the background within the image. However, the purpose of the object detection network was to detect the object, crop the RoI, and then feed that cropped image to the keypoint detection network. Therefore, the lowest IoU detection can still be considered good.



(a) A detection with an IoU of 0.668 (b) A detection with an IoU of 0.792 (c) A detection with an IoU of 0.907

Figure 8.8: Examples of detections with different IoU values

Training

As discussed in Section 5.5 transfer learning is almost always applied for computer vision applications due to its many benefits. The *finetuning* approach is used in this case and the object detection model is initialized using the weights and biases of the model that has been trained on the COCO 2017 dataset, which contains more than 200,000 images of 80 different object categories (Lin et al., 2014). This dataset is more challenging than the *Bennu* asteroid datasets. The earlier layers of the network perform low-level feature extraction, such as brightness changes, curves, and edges, which apply to the space problem as well. Therefore, the model has already learned some aspects and this approach generally produces favorable results in terms of overall accuracy and training time compared to a model that is trained from scratch, i.e., weights and biases randomly initialized.

The training parameters can be adjusted within the `train_config` part of the configuration file as shown in Code Listing C.4. The model is trained using *mini-batches* consisting of 32 images from the training dataset, whereas the model is validated after completion of one *epoch* using the images in the validation dataset. This mini-batch size was the default and found to work well for the Bennu datasets. The weight and biases of the network are updated after each mini-batch by using the *momentum optimizer with a cosine learning rate decay*. The momentum optimizer is faster than using a standard gradient descent.

For *mini-batch* k, $\left(X^{\{k\}}, Y^{\{k\}}\right)$ and epoch t, the following equations are used to update the weights and biases. Where the small v and s refer to the value of $V_{dw}, V_{db}, S_{dw}, S_{db}$ of the previous *mini-batch*, i.e., $k-1$.

$$V_{dw} = (1-\beta)\mathrm{d}w \qquad V_{db} = (1-\beta)\mathrm{d}b \tag{8.9}$$

$$w := w - \alpha V_{dw} \qquad b := w - \alpha V_{db} \tag{8.10}$$

The learning rate α and the momentum term β are hyperparameters of the momentum optimizer and the following value $\beta = 0.9$, is the default value that has been found to result in the best performance. The learning rate α is often problem specific and the approach proposed by Loshchilov and Hutter (2017) of using a cosine learning rate decay with warmup is employed. This allows the learning rate to grow as well as decrease throughout the training process. This approach has several advantages, which are enumerated below.

- The initial learning rate is small, which results in the gradients being small at the starting phase of training. This behavior is desired as transfer learning is applied and therefore the weights and biases of the pre-trained model should not drastically change with the first training steps. The model is fine-tuned to the desired application using a custom dataset and it is not required to change the weights and biases of the layers of the feature extractor, as they have learned low-level features that are applicable to this problem as well
- The increase in the learning rate during the initial phase of training would help the model to avoid getting stuck in local minima
- The smoothly decreasing learning rate over time will result in stability during training and will allow the model to find the best possible fit to the data

The learning rate α starts with a warm-up rate of 0.0266 and this rate increases for a 1000 "warmup" steps until it reaches the base learning rate of 0.08, which then gradually decreases to zero using a cosine decay until it reaches 50,000 steps. However, the training can be stopped at any time when the performance is found to be saturating. The average time per step is about 0.6 s, therefore training can take up to $\approx$ 8 hours.

Data augmentation

The relevance and importance of dataset augmentation was discussed in Section 5.5. The data augmentations are applied during the training process to improve the generalization performance of the network. The generalization performance during training is evaluated on the validation set, which similar to the training set consists solely of clean images. This removes the need to include augmentations, such as brightness changes, as they do not occur in the validation or test sets. Moreover, for the Bennu+ dataset, these pixel-level augmentation are already incorporated. Therefore, only the following affine data augmentations are used, namely random scaling/cropping and random horizontal flip, where random cropping refers to randomly cropping a part of the image with a certain aspect ratio and area and then rescaling it back to the original image size. Data augmentations can be adapted, added, or removed within the `train_config` part of the configuration file, Code Listing C.4.

The random cropping is used to make the network more invariant against the scale of the target object within the image. The *Bennu* datasets consist of images of the asteroid taken from discrete distances and by including this augmentation the network will learn to detect the asteroid on distances that were not explicitly present in the original dataset. The random horizontal flip augmentation is commonly used within object detection networks and creates new unique images, which results in more data for the network to learn from.

The details regarding the used settings can be found in the configuration file in Code Listing C.4. These augmentations are applied through the Tensorflow Object Detection API, which automatically transforms the annotations to align with the newly created image. For detailed information regarding the implementation of these augmentations the reader is referred to the source-code, which is available on GitHub[5] [6].

Inference

The trained model is used in inference to detect the object within previously unseen images. The algorithm can predict multiple detections of the same object in the image, with bounding boxes of different shapes and sizes as shown in Figure 8.9, including misdetections. However, the final output should simply be one bounding box for which the algorithm is most certain that it contains the target. Non-Maximum Suppression (NMS) is used to achieve this and it works in the following way for a single class.

1. It looks at the probabilities associated with each detection p_c. The algorithm then selects the most confident detection (highest p_c) and uses this as its prediction.
2. The remaining detected bounding boxes with an IoU value above a certain threshold, IoU $\geq x$, are then suppressed. The threshold value is set at 0.6.

The idea is that bounding boxes of the same class (asteroid) that have a high IoU are detecting the same object multiple times. Therefore, the bounding box that has the highest class probability is selected and the other ones detecting the same object are discarded, resulting in the output of a single predicted bounding box. The output of the network is stored in a `.json` file format, which can then be fed to the subsequent keypoint detection network.

[5] `https://github.com/tensorflow/models/blob/master/research/object_detection/core/preprocessor.py`, Date accessed: 25-11-2021

[6] `https://github.com/tensorflow/models/blob/master/research/object_detection/protos/preprocessor.proto`, Date accessed: 25-11-2021

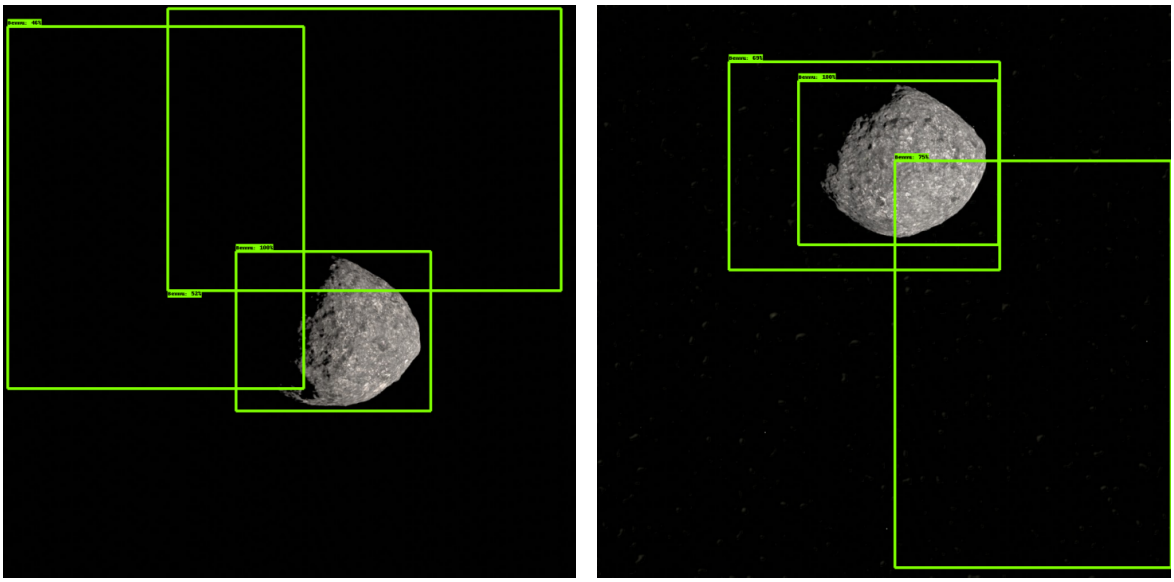


Figure 8.9: The predictions inferred by the object detection network trained on clean images only, on two different sample images from the *Bennu+* dataset

9

Keypoint detection network

This chapter discusses the keypoint detection network and the implementation. Firstly, an introduction to keypoint detection is given in Section 9.1, followed by the selection of the architecture that will be used in this work in Section 9.2. The chosen network will be discussed in more detail in Section 9.3. The implementation of the network is then discussed in Section 9.4 after which the configuration is elaborated upon in Section 9.5, demonstrating the settings used to achieve the results on the datasets.

9.1. Keypoint detection

The problem of keypoint estimation within machine learning generally can be described as predicting the coordinates of n pre-defined keypoints on the object from a 2D image of size $w \times h$. Within literature, keypoints are sometimes also referred to as features or landmarks. This is a challenging computer vision problem and the research revolving around keypoint detection is mostly linked to human pose estimation. The research regarding human pose estimation focuses on detecting the joints, such as the ankles, knees, and hips of person instances within images and subsequently estimate the pose of that person using these detected keypoints. The seminal work of Toshev and Szegedy (2014) with DeepPose resulted in the onset of deep learning for the problem of human pose estimation, as it showed much better performance compared to classical hand-engineered approaches. The majority of research on keypoint detection still revolves around human pose estimation, with increasingly more challenging datasets, such as COCO keypoint detection (Lin et al., 2014).

9.2. Architecture selection

As discussed in Section 6.2, it is best practice to use an open-source implementation of a network architecture that has been optimized for keypoint detection. Similar to the object detection network, the most suitable network is selected based on the notion that the architecture should have *low memory usage* and *FLOPs* allowing for future embedding in the spacecraft processing unit as discussed in Section 2.2. As aforementioned, the discussed networks are mostly applied to human pose estimation or general objects. However, the problem of detecting *pre-defined* keypoints is essentially the same throughout different applications and levels of abstraction. Therefore, these network architectures are deemed suitable for this work and will be adapted to the purpose.

The selection of the CNN architecture determines the accuracy of the keypoint detections. Newell et al. (2016) proposed an hourglass architecture that downsamples the input and subsequently upsamples the input to detect features at different resolutions (scales). Chen et al. (2018) proposed the Cascaded Pyramid Network (CPN), which uses a similar approach as the hourglass structure, but it concatenates the feature maps from different scales to arrive at the final heatmap output, similar to the FPN neck discussed in Section 8.3. Furthermore, Xiao et al. (2018) proposed a simple architecture based on the hourglass that replaces the upsampling operations by deconvolutional layers, which combines convolution and upsampling, thereby changing the way the high resolution feature maps are obtained. However, currently the state-of-the-art keypoint detection network on the COCO dataset is the High Resolution Network (HRNet) proposed by Sun et al. (2019). This network uses certain sub-networks in parallel that each have a different resolution, compared to the aforementioned networks that have different resolutions in series. The network then shares information between the different resolution feature maps (multi-scale) fusion to improve the heatmaps precision. The

Table 9.1: Comparison of the performance, size, and efficiency of the different commonly used networks for keypoint detection on the COCO validation set for an input size of 256×192 pixels ($h \times w$) (Zhang et al., 2019)

Network	Average precision [1]	Parameters [Mn]	FLOPs [Bn]
HRNet-W48	76.3	63.6	32.9
HRNet-W32	74.4	28.5	7.1
SimpleBaseline (ResNet-152)	72.0	68.6	15.7
SimpleBaseline (ResNet-101)	71.4	53	12.4
LPN (ResNet-152)	71.0	7.4	1.8
SimpleBaseline (Resnet-50)	70.4	34	8.9
LPN (ResNet-101)	70.4	5.3	1.4
LPN (ResNet-50)	69.1	2.9	1.0
CPN (ResNet-50)	68.6	27	6.2
Hourglass (8-stage)	67.1	25.1	19.5

HRNet-W32 network has been used for keypoint detection on uncooperative spacecraft by Chen et al. (2019), Barad (2020), and Pasqualetto Cassinis et al. (2021b).

However, these networks have a relatively large number of FLOPs and parameters, requiring a substantial amount of memory, making them unsuitable for resource-limited devices. Currently, research is being performed on the development of lightweight networks for keypoint detection, making them suitable for embedded devices. However, this research is relatively new and not as established as for the object detection networks. Bulat and Tzimiropoulos (2017) proposed the first lightweight design, however, the performance loss was significant. Zhang et al. (2019) proposed the LPN, which is based on the architecture proposed by Xiao et al. (2018). The LPN network replaced the convolutional layers with depthwise separable convolution (Section 5.4) resulting in a major reduction in the number of parameters and FLOPs, while achieving good performance. The discussed networks are compared on their accuracy, size, and speed, as shown in Table 9.1.

As can be observed in Table 9.1, the performance of LPN is comparable to other state-of-the-art networks on the challenging COCO dataset, but it is able to achieve this with only a fraction of the parameters and FLOPs. Therefore, from an accuracy-speed point of view, which is in line with the desired lightweight properties required for implementation on space hardware, the LPN (ResNet-101) network is selected.

9.3. Lightweight Pose Network

The LPN network is based on the simple network proposed by Xiao et al. (2018), SimpleBaseline. The network consists of two parts, it first decreases the resolution through the use of convolutional layers after which the resolution is recovered through upsampling to arrive at the final heatmap representation. This architecture is graphically illustrated in Figure 9.1. The LPN uses the commonly used ResNet-101 architecture as its feature extractor. The network then adds some deconvolutional layers at the end of the ResNet-101 architecture to upscale the feature maps to the desired output size. This structure is used for its simplicity and generates heatmaps from the deep, low-resolution feature maps, which have the most semantic meaning.

Most existing methods tend to use low-resolution feature maps to avoid increasing the computational effort too much, however, HRNet (Sun et al., 2019) demonstrated that high-resolution representations allow the CNN to produce more accurate and precise heatmaps. The LPN model therefore removes the last convolutional layer of the SimpleBaseline model and its corresponding upsampling layer, to be able to use a higher resolution feature map to make predictions while still adhering to the lightweight principles.

The improvements of LPN revolve around creating lightweight alternatives for commonly used building blocks of other networks. The lightweight bottleneck block was introduced when discussing lightweight networks in Section 5.4. The lightweight bottleneck block replaces the standard convolutional block with depthwise convolution, achieving a reduction in parameters and consequently computational cost of $\approx \frac{2}{17}$. Furthermore, a Global Context (GC) block is added to the lightweight bottleneck block to improve the capacity of the model as it was found to improve the performance of the network (Zhang et al., 2019).

[1] Definition can be found on: `https://cocodataset.org/#keypoints-eval`

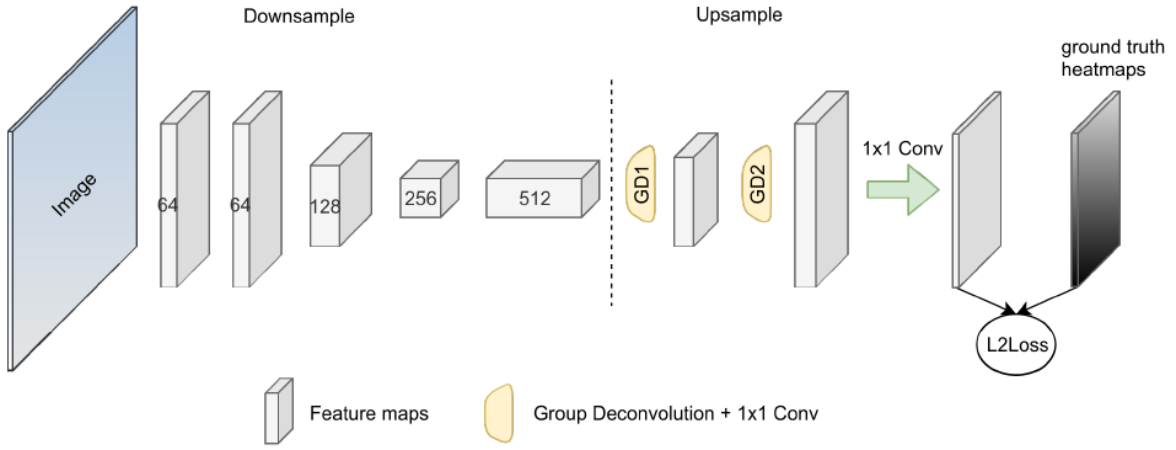


Figure 9.1: The architecture of the LPN model, which consist of several downsampling and deconvolutional layers (Zhang et al., 2019)

9.4. Implementation

The LPN network is available through GitHub[2] and is based on the repository of HRNet[3]. The HRNet repository has created a structured format of files and functions, allowing for a structured approach in adapting the network to keypoint detection on asteroids. Furthermore, it allows the adaptation of certain parameters and settings, such as the input size, optimizer, learning rate, and mini-batch size through the use of a configuration file, which is in `.yaml` format (Code Listing C.5). The LPN networks have been created using PyTorch, which is an open-source machine learning platform built and maintained by Meta AI Research, which uses a Python API. Within this work, PyTorch 1.9.0+cu111 and torchvision 0.10.0+cu111 have been used, however, detailed information regarding the installation and required folder formatting can be found in the HRNet repository[3].

As discussed in Section 6.2, the network is trained and evaluated using Google Colaboratory (Colab) and the NVIDIA Tesla P100-PCIE-16GB GPU. The model architecture, which specifies the number of convolutional and pooling layers and other specifications is fixed. This architecture did not have to be adjusted and only the input to the network, such as the COCO format had to be created for the Bennu datasets. Furthermore, some existing files had to be adapted to work for the Bennu datasets, where two major things had to be adjusted among other things:

- The configuration file, which specifies the number of keypoints, training settings, and data augmentations used among others, which will be discussed in more detail in Section 9.5.
- The `bennu_coco.py` file, which processes the annotated data used for training and evaluation.

The `bennu_coco.py` file processes the annotated dataset for training and evaluation purposes and had to be made suitable for the dataset format used in the creation of the Bennu dataset.

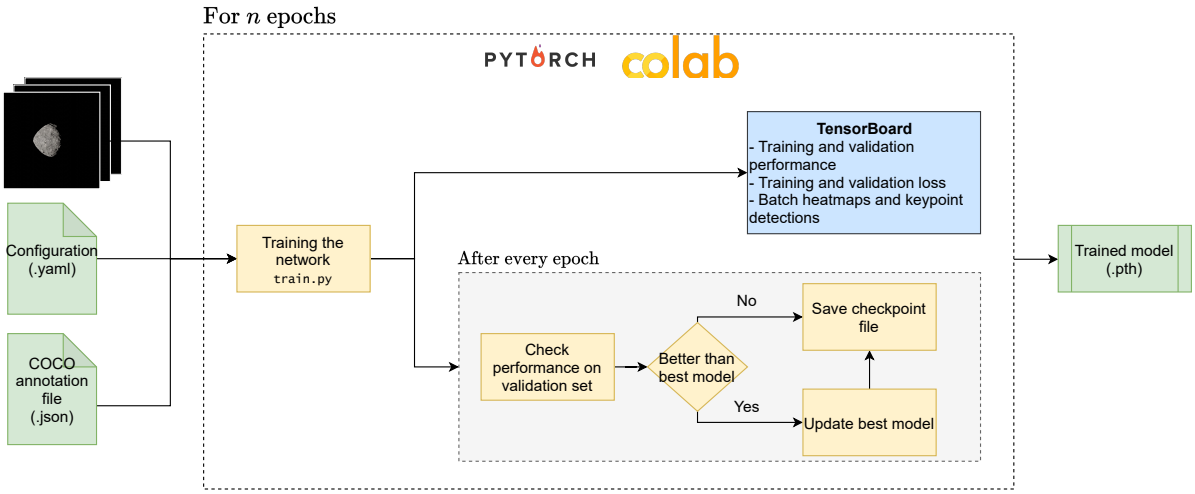
The training and validation of the network are performed simultaneously during training, and the respective dataset to use for each can be specified in the configuration file, Code Listing C.5. The network is trained through the use of the `train.py` script and the configuration file containing the desired settings. The training process is graphically illustrated in Figure 9.2a. During training, after each *epoch* the current model is evaluated on the validation dataset and if applicable the best model achieved up until that point is updated. Furthermore, the network outputs training logs and checkpoint files, which contain the current model's state alongside other relevant information, such as the best model's state and performance up until that *epoch*, and the value of the learning rate. These checkpoint files allows the network to start (re)training from such a checkpoint. Training can automatically resume from the last checkpoint by setting `AUTO_RESUME = true` in the configuration file (Code Listing C.5).

[2] `https://github.com/zhang943/lpn-pytorch`, Date accessed: 25-11-2021
[3] `https://github.com/leoxiaobin/deep-high-resolution-net.pytorch`, Date accessed: 25-11-2021

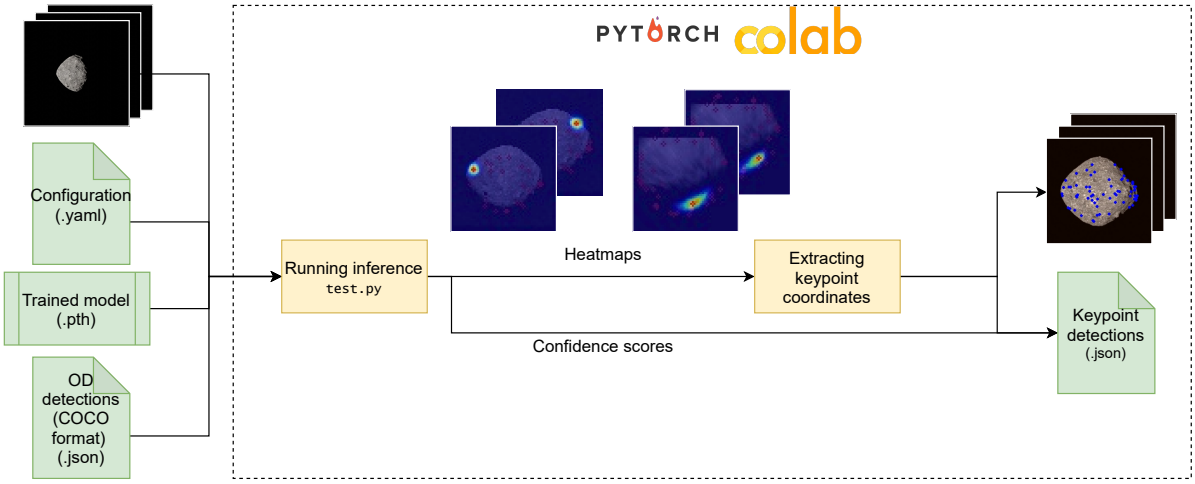
The trained model can subsequently be used for inference using Google Colab or on a local machine. This is an independent process for which the `test.py` script has to be used. This can either be used to simply test the keypoint detection network in isolation by using the ground-truth bounding box coordinates, or it can be used to evaluate the keypoint detection network as part of the entire pipeline, i.e., OD and KD. For the latter the detections by the OD network are fed to the keypoint detection network as will be discussed in Section 9.5. The inference process is graphically illustrated in Figure 9.2b. The network predicts heatmaps around the keypoint location, which are subsequently used to extract the keypoint coordinates that can be used for evaluation of the performance of the network.

9.5. Configuration

The model architecture, which specifies the number of convolutional and pooling layers and other specifications is fixed. However, certain hyperparameters need to be set, adapted, or tuned as well as training settings, such as selecting the loss function and optimizer. These parameters and settings specify the configuration of the network and this will be discussed in more detail in this section. An example configuration file is given in Code Listing C.5.



(a) The training process



(b) Inference

Figure 9.2: Graphically illustrating the implementation of the training and inference process, showing the inputs and outputs

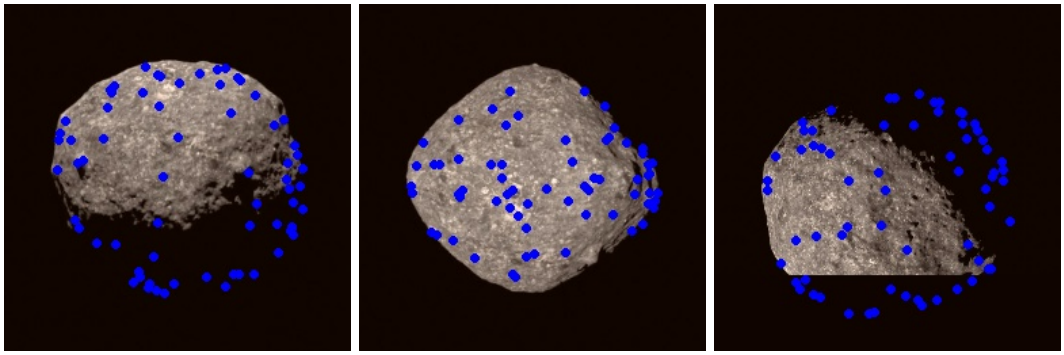


Figure 9.3: Visualization of the designated keypoints on the asteroid surface for three different camera poses, where points on the back of the asteroid are also plotted in the same plane. Furthermore, points can also lie outside of the image dimensions as shown in the most right figure

Input and output

The input to the keypoint detection network is the cropped RoI, detected by the object detection network. The aspect ratio of the input size of the network is used during the cropping to avoid deformations in the cropped images. This means that for a network input size of 256 × 256 px, i.e., aspect ratio of 1, a bounding box with a width and height of 500 px and 480 px, respectively, will be cropped to 500 × 500 px, around the center of the bounding box. This cropped RoI is then rescaled to the desired input size. During the rescaling, the scaling parameters, namely the center and scale, are preserved. The input size directly effects the computational demand of the network. During training, validation, and testing of the keypoint detection network the ground-truth bounding boxes are used to optimize the network in isolation. However, when evaluating the entire pipeline during inference, the cropped RoI is used as the input (Section 11.3), which would entail to setting `USE_GT_BBOX = False` and specifying the path of the `COCO_BBOX_FILE` in Code Listing C.5, which contains the object detections in the format given in Code Listing C.3. The required format can be created through a created utility script that can be found on GitHub[4]

The standard input size of the keypoint detection networks is 256×192 px ($h \times w$), this notation of $h \times w$ is the convention for representing the size of the image in PyTorch and the literature surrounding human pose estimation. The reason for this 4 : 3 aspect ratio of the input size is that the person instances (bounding box around the persons) generally have this aspect ratio, as the height is larger than the width when a person is standing up. Therefore, this input size was found to work the best for this type of problem, however, when adapting this network to the asteroid dataset another input size might produce better results. The aspect ratios of the ground-truth bounding boxes are predominantly between 0.8 and 1.0 as can be observed in Figure 8.6. Therefore, the following input sizes were tested to see the effect of the input size on the accuracy, namely 256×256 px and 256×192 px ($h \times w$). The results are discussed in Section 11.2. Larger input sizes, such as 384×384 would increase the number of FLOPs by 125%. Moreover, the GPU's memory places constraints on the maximum image size that can be used.

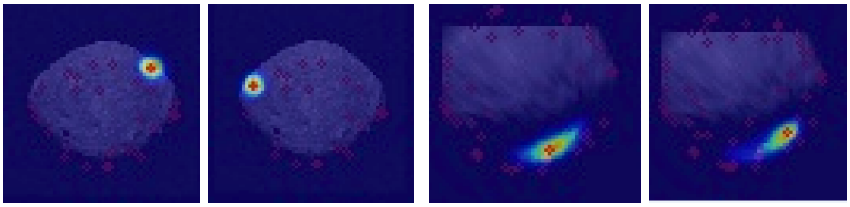
The ground-truth annotations as discussed in Subsection 7.4.2 are converted to the COCO format. This annotation format is shown in Code Listing C.2 and is commonly used for keypoint detection networks, and it therefore allows for easy adaptation. It contains the ground-truth keypoint locations and bounding boxes, where examples of the ground-truth keypoints are given in Figure 9.3.

As the network uses a heatmap representation of the keypoints, the ground-truth keypoint locations have to be converted to heatmaps. These ground-truth heatmaps are generated by the network using a Gaussian distribution around each keypoint as given below, where $\mathbf{H}_k \in \mathbb{R}^{w \times h}$ and (x_k, y_k) represent the value of the ground-truth heatmap and 2D coordinates of the k^{th} keypoint, respectively. The standard deviation σ is set to 2 px, following convention.

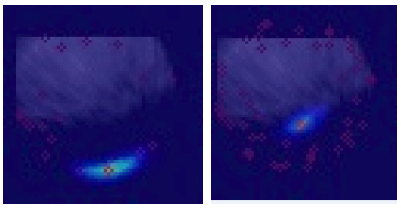
$$\mathbf{H}_k(x, y) = \exp\left(-\frac{(x - x_k)^2 + \left(y - y_k\right)^2}{2\sigma^2}\right) \tag{9.1}$$

The output of the network detecting k keypoints within an image, is k heatmaps per image. The heatmaps are kept at one fourth of the size of the input image, which is common practice for these keypoint detection

[4] `https://github.com/lvanderheijden`



(a) Confident detections (b) Less confident detections



(c) Bad detections

Figure 9.4: Visualization of different types of keypoint heatmap detections

networks and is done from a computational efficiency point of view (Chen et al., 2018; Newell et al., 2016; Zhang et al., 2019). Xiao et al. (2018) found that this ratio improved the performance compared to using even smaller heatmaps. Examples of different types of heatmap predictions by the network are given in Figure 9.4. Confident detections closely resemble Gaussian distributed heatmaps as observed in Figure 9.4a, whereas for less confident detections the heatmap is more smeared out (blob) as observed in Figure 9.4b. Figure 9.4c illustrates scenarios in which the network is unable to accurately and confidently predict the location of the keypoint.

The heatmap size for the specified input image sizes is 64×64 and 64×48 pixels, respectively. The keypoints extracted from the predicted heatmaps can be reprojected to the original image size, i.e., before rescaling of the RoI, using the aforementioned scaling parameters.

Loss function

The loss function that is used is the Mean Squared Error (MSE), which calculates the squared L2-norm between the confidence values of the ground-truth and predicted heatmap, respectively. This loss function is commonly used for keypoint detection. The loss function of the network for k heatmaps is given below, where c_{ij} and $\hat{c}_{ij}$ are the ground-truth and predicted confidence value, respectively, for a pixel at location (i, j) in a heatmap of size $w \times h$.

$$l_k\left(c_{ij}, \hat{c}_{ij}\right) = \frac{1}{wh} \sum_{i,j} \left(c_{ij} - \hat{c}_{ij}\right)^2 \quad ; \quad L_k = \{l_1, \ldots, l_k\}^T \tag{9.2}$$

$$L = \frac{1}{k} \sum_{k} L_k \tag{9.3}$$

Training

As discussed in Section 5.5, transfer learning is almost always applied for computer vision applications due to its many benefits. The *finetuning* approach is used in this case and the Bennu-LPN model is initialized using the LPN model designed for the problem of human pose estimation. This model was trained on the COCO train2017 dataset, consisting of 57,000 images and 150,000 person instances, labeled with 17 keypoints (joints). An adaptation to the model and configuration file (`FINAL_LAYER`) was required to allow for this initialization, as the model trained on COCO uses 17 keypoint, resulting in a dimension error for the last layer.

This dataset is distinctly different from the *Bennu* asteroid dataset, however, the earlier layer of the network perform low-level feature extraction, such as brightness changes, curves, and edges, which apply to the space problem as well. Therefore, the model has already learned some aspects and this approach generally produces favorable results in terms of overall accuracy and training time compared to a model that is trained from scratch, i.e., weights and biases randomly initialized.

The training parameters can be adjusted under the `train` part of the configuration file as shown in Code Listing C.5. The model is trained using *mini-batches* (Section 5.5) consisting of 64 images from the training dataset, whereas the model is validated after completion of one *epoch* using the images in the validation dataset. The default mini-batch size of 32 images, used by the LPN model on the COCO dataset, was found to result in subpar performance on the Bennu datasets, compared to a mini-batch size of 64 images. A mini-batch of 128 images was too large for the single GPU's memory for the input size of 256 × 256 px, although it showed promising results for the 256 × 192 px input size. Therefore, whenever more GPU's are available for training it could be worth exploring the 128 mini-batch size further.

These mini-batches are reshuffled after every epoch to avoid the network overfitting on certain mini-batches and to ensure that the network performs well on the entire dataset. The weights and biases of the network are updated after each mini-batch using the *Adam* optimizer (Kingma and Lei Ba, 2015). This is a commonly used optimizer for machine learning models due to its computational efficiency. This optimizer combines techniques from other stochastic gradient descent methods, such as *RMSprop* and *Gradient descent with momentum* (Goodfellow et al., 2016). The weights and biases of the nodes are updated after each *mini-batch*. For *mini-batch* k, $\left(X^{\{k\}}, Y^{\{k\}}\right)$ and epoch t, the following equations are used to update the weights and biases. Where the small v and s refer to the value of $V_{dw}, V_{db}, S_{dw}, S_{db}$ of the previous *mini-batch*, i.e., $k-1$.

$$\begin{aligned} V_{dw} &= \beta_1 v_{dw} + (1-\beta_1)\mathrm{d}w & S_{dw} &= \beta_2 s_{dw} + (1-\beta_2)\mathrm{d}w^2 \\ V_{db} &= \beta_1 v_{db} + (1-\beta_1)\mathrm{d}b & S_{db} &= \beta_2 s_{db} + (1-\beta_2)\mathrm{d}b^2 \\ V_{dw}^{corr} &= \frac{V_{dw}}{1-\beta_1^t} & S_{dw}^{corr} &= \frac{S_{dw}}{1-\beta_2^t} \\ V_{db}^{corr} &= \frac{V_{db}}{1-\beta_1^t} & S_{db}^{corr} &= \frac{S_{db}}{1-\beta_2^t} \end{aligned} \tag{9.4}$$

$$w := w - \alpha \frac{V_{dw}^{corr}}{\sqrt{S_{dw}^{corr}} + \epsilon} \qquad b := w - \alpha \frac{V_{dw}^{corr}}{\sqrt{S_{dw}^{corr}} + \epsilon} \tag{9.5}$$

The gradients $\mathrm{d}w$ and $\mathrm{d}b$ are calculated through backpropagation using the current *mini-batch*. The weights and biases are updated using Equation (9.5) and a new forward pass starts with the next mini-batch.

The hyperparameters of the *Adam* optimizer are the learning rate α and β_1, β_2, and ϵ. The following values $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ are default values that are often used (and untuned) and are also in line with what was used for training LPN (Zhang et al., 2019). The learning rate α is often problem specific and is scheduled to decay at different epochs using a multi-step schedule. The learning rate scheduling starts with a base learning rate of $1\mathrm{e}-3$ and is changed to $1\mathrm{e}-4$ at epoch 90 and to $1\mathrm{e}-5$ at epoch 120, which is then kept constant until the end epoch of 150. These learning rates are in line with the learning rates used by the LPN for the problem of human pose estimation (Zhang et al., 2019). This decreasing learning rate is used to slow down progress, based on the assumption that the network is close to the minimum, and as such avoid overshooting and allow convergence to the minimum. A larger learning rate of 0.01 was tested as well, however, this showed volatile behavior of the loss and consequently accuracy of the network, indicating that the learning rate was too large and the resulting update step of the weights and biases was too large (Figure 5.3). The cut-off points of the scheduled learning rate is not an as important *hyperparameter* as the learning rate α and the mini-batch size, therefore, these values were not tuned further.

A single epoch trained using the GPU specified in Section 6.2 takes around 13 min. Therefore, training the network on a single dataset for 150 epochs can take up to 32 hours. This applies to every experiment that is performed, e.g., different mini-batch size, input size, or learning rate, demonstrating that training neural networks is not trivial.

The original authors of the LPN model (Zhang et al., 2019) proposed the use of an *iterative training strategy*. The reasoning behind this was that the network might fall into suboptimal local minima during training and then it might be difficult to cross the ridge when the learning rate is small. Therefore, the learning rate has to be increased. The implementation of this approach is graphically illustrated in Figure 9.5. This means that in the first stage the network is trained for 150 epochs, after which the best model determined in stage 0 is used to initialize the model for stage 1, which then restarts the training from epoch 60 using the same *hyperparameter* settings, such as the scheduled learning rate.

This strategy was used to see whether this could result in an improvement of the models. When a model was found to benefit from the iterative approach, it is mentioned in Section 11.2.

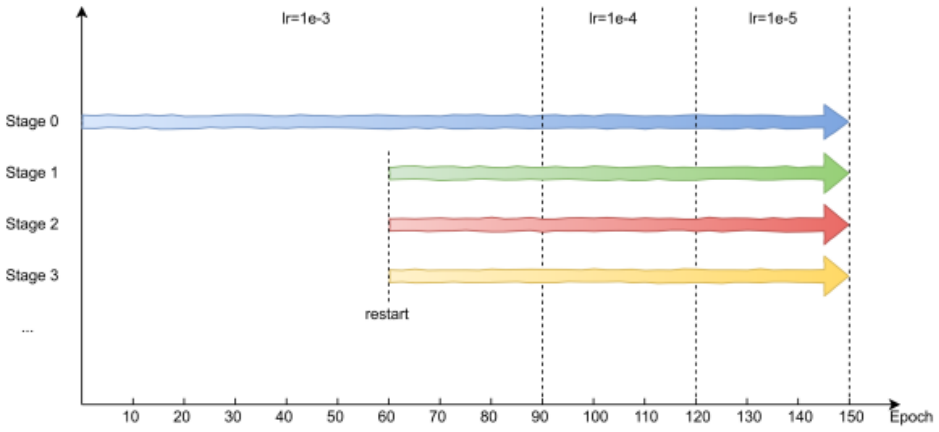


Figure 9.5: Illustrating the iterative training strategy (Zhang et al., 2019)

Data augmentation

The relevance and importance of dataset augmentation was discussed in Section 5.5. The data augmentations are applied during the training process to improve the generalization performance of the network. The generalization performance during training is evaluated on the validation set, which, similar to the training set, consists solely of clean images for the Bennu dataset. This removes the need to include augmentations, such as brightness changes, as they do not occur in the validation or test sets. Moreover, for the Bennu+ dataset, these pixel-level augmentation are already incorporated. Therefore, only the following *affine* data augmentations are used, namely random scaling, random horizontal flip, and random rotation.

The random scaling refers to multiplying the scale parameter as discussed in Section 9.5 with a scaling factor. The scaling factor is set to 0.3 meaning that the scale parameter is multiplied by a randomly selected value within the range $[0.7; 1.3]$. The random horizontal flip augmentation is commonly used within keypoint detection networks and creates new unique images, which results in more data for the network to learn from. The probability of this augmentation is set to 0.5. The random rotation randomly applies a rotation to the original image orientation and the rotation factor is set to 40°, meaning that image is randomly rotated by an angle within the range $[-80°, 80°]$. The probability of this augmentation is set to 0.6.

The annotations are automatically transformed to align with the newly created augmented image. For detailed information regarding the implementation of these augmentations the reader is referred to the source-code, which is available on GitHub[5]. Data augmentations can be adapted, added, or removed under the `dataset` part of the configuration file Code Listing C.5.

Keypoints extraction

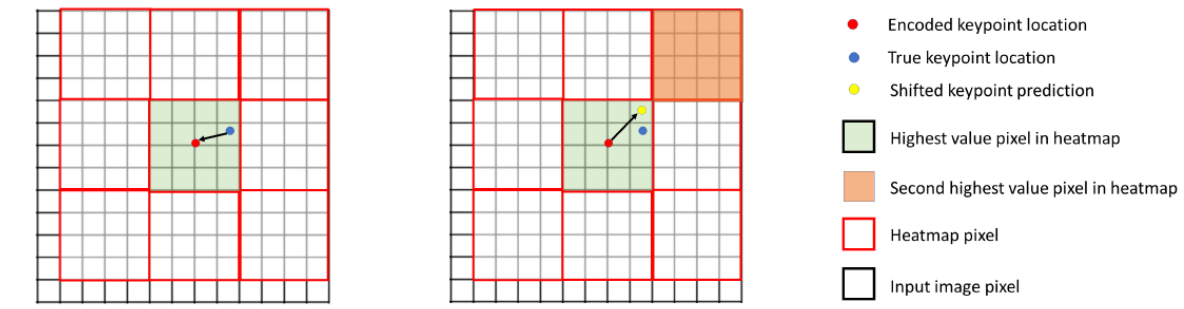
Most existing methods use the *argmax* algorithm to extract the keypoint locations from the heatmap after which they are transformed to the original image size, i.e., setting the keypoint position equal to the position of the highest value of the heatmap. However, the result of the algorithm is discrete, thereby limiting the accuracy of the predicted keypoints. An improved approach proposed by Sun et al. (2018) and Luvizon et al. (2019) uses the (spatial) *Soft-Argmax* algorithm. This *Soft-Argmax* function is differentiable and therefore it can be incorporated into the trainable network through backpropagation.

However, by normalizing the heatmaps to the domain $[0, 1]$ the majority of the values of the predicted heatmap image will be close to zero, which can affect the accuracy of the (spatial) *Soft-Argmax* algorithm:

$$S_k(x, y) = \frac{e^{H_k(x,y)}}{\sum_x \sum_y e^{H_k(x,y)}} \tag{9.6}$$

By having a large number of zeros in the heatmap, the probability of the maximum is reduced, as $e^0 = 1$, thereby affecting the accuracy of the results. Therefore, the usage of a different approach is proposed that tries to mitigate this, namely the β-*Soft-Argmax* algorithm, which is shown below (Zhang et al., 2019).

[5] `https://github.com/zhang943/lpn-pytorch/blob/master/lib/dataset/JointsDataset.py`, Date accessed: 25-11-2021



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

Figure 9.6: Demonstrating the effect of the keypoint encoding from a heatmap representation with one-fourth input resolution (Barad, 2020)

$$S_k(x,y) = \frac{e^{\beta H_k(x,y)}}{\sum_x \sum_y e^{\beta H_k(x,y)}}, \quad (\beta > 1) \tag{9.7}$$

The addition of the β parameter is used to suppress the impact of the heatmap values close to zero. The remainder of the procedure is similar to the *Soft-Argmax*, where the regressed location of the predicted keypoint is determined as follows:

$$(\hat{x}_k, \hat{y}_k) = (\sum_x \sum_y \mathbf{W}_x S_k, \sum_x \sum_y \mathbf{W}_y S_k)^T \tag{9.8}$$

where $\mathbf{W}_x$ and $\mathbf{W}_y$ are constant weight matrices defined as $\mathbf{W}_x = \frac{x}{w}$ and $\mathbf{W}_y = \frac{y}{h}$, where x and y represent the location and w and h represent the width and height of the image, respectively.

The keypoint coordinates are regressed using the β-*Soft-Argmax* algorithm where $\beta = 160$, which is in line with the value resulting in the best performance found by Zhang et al. (2019). However, as previously mentioned, the size of the heat map is one-fourth of the size of the original input image. Therefore, the predicted keypoint location needs to be projected back to the input image size.

Furthermore, for training purposes the encoding of the keypoint location also needs to be projected to the smaller image size. This is graphically illustrated in Figure 9.6a, where the true keypoint location represented by the blue dot is projected to encoded keypoint location represented by the red dot. The implication of this is that the network performs perfectly if it is able to estimate the location of the encoded keypoint. However, in reality the true keypoint location before downsampling was located at the blue dot in Figure 9.6a. Consequently, when projecting the predicted keypoint location from the heatmap size to the original image size, an undesirable error arises, which is caused by the downsampling operation.

Therefore, to reduce this error, the predicted keypoint is shifted by a quarter of a pixel from the center of the highest value pixel into the direction of the next highest value pixel. This is graphically shown in Figure 9.6b. This approach was first proposed by Newell et al. (2016) and has been shown to significantly improve the keypoint prediction by Zhang et al. (2020). This approach has also been applied in HRNet (Sun et al., 2019). This procedure is mathematically represented as follows, where $\mathbf{p}_s$ represents the coordinates (u, v) of the shifted keypoint, whereas $\mathbf{p}_1$ and $\mathbf{p}_2$ represent the pixel coordinates with the highest and second highest confidence in the predicted heatmap, respectively.

$$\mathbf{p}_s = \mathbf{p}_1 + 0.25 \frac{\mathbf{p}_2 - \mathbf{p}_1}{\|\mathbf{p}_2 - \mathbf{p}_1\|} \tag{9.9}$$

Evaluation metrics

The predicted keypoint's 2D location is given by $(\hat{u}, \hat{v})$, which is then compared to the ground-truth location (u, v). The prediction can have an error in both u and v and therefore the total error is calculated using the following (RMS error):

$$E_{px} = \sqrt{|\hat{u} - u|^2 + |\hat{v} - v|^2} \tag{9.10}$$

Equation (9.10) represents the pixel error for a single keypoint detection. However, the network outputs 68 detections for every image. Therefore, to get a sense of the general accuracy of the keypoint detection for any given image, the mean error of the n predictions is taken, where n is kept general, as not necessarily all 68 keypoints are required.

The mean and median error over all images are used to evaluate the performance of the keypoint detection network, i.e., the mean and median of the mean keypoint detection errors of all images for n keypoint detections per image (mean/median of means).

Furthermore, the order of the error of the keypoint detection network depends on the input size of the original image. The image size used in this work is 1024×1024 px, however, to allow for comparison of the performance of the network against different input sizes, the performance is normalized using the scaling parameters as discussed in Section 9.5 (Input and output). This *input-normalized* error provides a consistent reference of the keypoint detection network's performance across datasets.

10

Verification

This chapter discusses the verification process of the parts of the developed dataset and algorithms. The verification process is divided into three parts, namely the dataset generation in Section 10.1, which consists of the image generation as well as the annotation of the dataset, the machine learning algorithms (OD and KD) in Section 10.2, and the pose estimation pipeline evaluation in Section 10.3.

10.1. Dataset generation and annotation

The dataset generation and annotation can be divided into different parts:

1. Camera poses generation
2. Image rendering
3. Dataset annotation

Furthermore, underlying these different parts are several reference frame conversion and other utility functions, which have been grouped within a `Kinematics` class in Python. The functions that have been written to generate and annotate the datasets are grouped within their respective classes in the `dataset_utils.py` and `datasetGeneration.py` files[1]. These utility functions have all been verified and an overview of these low-level unit test are listed in Table 10.1.

The reference frames conversions, as shown in Table 10.1, have been verified using an input-output check, verifying the expected result. Furthermore, $\mathbf{C}^{-1} = \mathbf{C}^T$ has been checked and it was verified that the output $\mathbf{r}^B$, determined using $\mathbf{r}^B = \mathbf{C}_{B,A}\mathbf{r}^A$, results in the same input $\mathbf{r}^A$, determined using $\mathbf{r}^A = \mathbf{C}_{B,A}^{-1} r^B$, i.e., $\mathbf{C}\mathbf{C}^{-1} = I$. The orientation of the Blender camera w.r.t. the world axes, $\mathbf{q}_{B,W}^W$, is described using the singularity free unit quaternion parameterization and is converted into a DCM representing that rotation, $\mathbf{C}_{B,W}$. The only transformations that use Euler angles, which have a singularity at $\pm 90°$ pitch angle θ, to create the transformation matrix are $\mathbf{C}_{A,W}$, $\mathbf{C}_{B',B}$, and $\mathbf{C}_{C,B}$. The transformation matrix $\mathbf{C}_{A,W}$ is created using a single rotation (yaw, ψ) around the Z_W-axis (Equation (3.20)), $\mathbf{C}_{B',B}$ is created using maximum rotation angles of $\pm 5°$ (Equation (7.5)), and $\mathbf{C}_{C,B}$ consists of a fixed rotation roll angle (φ) of $-180°$ (Equation (3.22)). Based on the aforementioned, singularities do not occur and throughout this work DCMs are used to represent rotations and where required they are transformed back to unit quaternions.

10.1.1. Camera pose generation

As discussed in Subsection 7.2.3 and illustratively shown in Figure 7.4, the dataset generation pipeline consists of several facets. The nominal pose is generated through a combination of two factors, namely the generation of the camera positions through the use of the vertices of an *icosphere*, and the corresponding camera orientation is ensured through calculation of the out-of-plane rotation angles and a tracking constraint in Blender. This makes sure that the Blender camera's Y-axis is pointing up (aligned as much as possible with the world's Z-axis). The icosphere is generated using Python's library PyMesh[2]. This is a library that is used by many users throughout the world and therefore the chance of an error existing in the *icosphere* function is

[1] `https://github.com/lvanderheijden`

[2] `https://pymesh.readthedocs.io/en/latest/`, Date accessed: 7-10-2021

Table 10.1: Table listing the low-level unit tests that have been performed to verify the required functionality

Index	Part	Functionality	Method	Passed
1	RotfromVertices	Calculating the out-of-plane rotation angles (θ, φ) of a vertex (Subsection 7.2.3)	Input-output check	✓
2	Unit-axis transformation matrices	Calculating the unit-axis transformation matrices around the x, y, z axes respectively (Section 3.3)	Input-output check, $\mathbf{C}\mathbf{C}^T = \mathbf{C}\mathbf{C}^{-1} = I$	✓
3	world2asteroid - $\mathbf{C}_{A,W}$	Converting from the world reference frame (W) to the asteroid reference frame (A)	Input-output check, $\mathbf{C}\mathbf{C}^T = \mathbf{C}\mathbf{C}^{-1} = I$	✓
4	world2Bcam - $\mathbf{C}_{B,W}$	Converting from the world reference frame (W) to the Blender camera reference frame (B)	Input-output check, $\mathbf{C}\mathbf{C}^T = \mathbf{C}\mathbf{C}^{-1} = I$	✓
5	Bcam2camera - $\mathbf{C}_{C,B}$	Converting from the Blender camera reference frame (B) to the camera reference frame (C)	Input-output check, $\mathbf{C}\mathbf{C}^T = \mathbf{C}\mathbf{C}^{-1} = I$	✓
6	Bcam2CamBearing - $\mathbf{C}_{B',B}$	Converting from the Blender camera reference frame (B) to the intermediate off-nominal pointing Blender camera reference frame (B')	Input-output check, $\mathbf{C}\mathbf{C}^T = \mathbf{C}\mathbf{C}^{-1} = I$	✓
7	asteroid2camera - $\mathbf{C}_{C,A}$	Converting from the asteroid reference frame (A) to the camera reference frame (C) through multiplication of the intermediate conversions	Input-output check, $\mathbf{C}\mathbf{C}^T = \mathbf{C}\mathbf{C}^{-1} = I$	✓
8	dcm2quat	Converting a DCM to quaternions (Equation (3.10))	Input-output check	✓
9	quat2dcm	Converting quaternions to a DCM (Equation (3.8))	Input-output check	✓
10	quatMultiply	Multiplying two quaternions together	Input-output check	✓
11	quatConj	Creating the conjugate of a quaternions	Input-output check	✓
12	dcm2euler	Converting a DCM to Euler angles (3-2-1)	Input-output check	✓
13	compute_relaxed_bbox	Relaxing the 'raw' bounding box coordinates by 5%	Visual inspection/ calculation	✓
14	Retrieve the images	Retrieving the images from the respective folders and writing them to new directory	Visual inspection/ Analysis	✓
15	bbox_raw_data	Retrieving the bounding box coordinates and rewrite to standard format	Visual inspection/ Analysis	✓
16	Dataset partitioning	Randomly partitioning the dataset into train/val/test	Analysis	✓

close to zero. Based on that, it is assumed that this function has been verified. Furthermore, the calculation of out-of-plane rotation angles has also been verified and is listed in Table 10.1, as this is considered a low-level test. The camera-up constraint uses the *tracking constraint*[3] implemented in Blender. Along the same line as PyMesh, it is assumed that the function has been verified. The remaining verification of the pose generation revolves around the off-nominal pointing, as this is implemented from scratch. The verification procedure is discussed in more detail.

Off-nominal pointing

This discusses the unit-test performed to verify the functionality of the creation of off-nominal pointing cases. This consists of generating a quaternion parameterization representing the orientation of the Blender camera (B) w.r.t. the world frame (W).

Input:

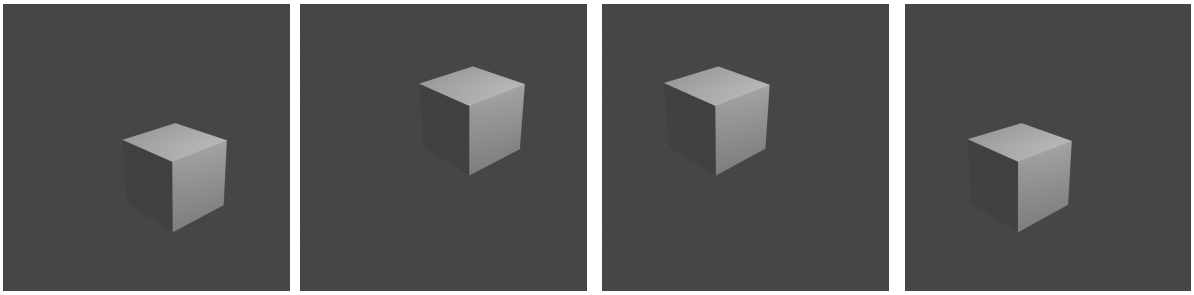
- The nominal pose of the Blender camera (B) w.r.t. world frame as shown in Table 10.3
- The number of off-nominal pointing cases $N = 4$
- The bearing angle combinations (α, β)

Expected output: A positive combination of the bearing angles would result in the object moving to the bottom right corner, in accordance with the definition shown in Figure 7.7 and the discussion on off-nominal pointing (part 4) in Subsection 7.2.3. The same notion applies to the other combinations

Test passed: When through visual inspection it can be observed that the actual results are equal to the expected results for the different cases

The nominal pose as shown in Table 10.3 was used to apply the selected bearing angles. The following combinations were tested $(\alpha, \beta) = (4.0°, 4.0°)$, $(\alpha, \beta) = (4.0°, -4.0°)$, $(\alpha, \beta) = (-4.0°, -4.0°)$ and $(\alpha, \beta) = (-4.0°, 4.0°)$ to verify the four different possible quadrants. Furthermore, the zero-test was performed, show-

[3] `https://docs.blender.org/manual/en/latest/animation/constraints/tracking/track_to.html#example`, Date accessed: 7-10-2021



(a) The off-nominal pointing pose using $(\alpha, \beta) = (4.0°, 4.0°)$

(b) The off-nominal pointing pose using $(\alpha, \beta) = (4.0°, -4.0°)$

(c) The off-nominal pointing pose using $(\alpha, \beta) = (-4.0°, -4.0°)$

(d) The off-nominal pointing pose using $(\alpha, \beta) = (-4.0°, 4.0°)$

Figure 10.1: The verification of the off-nominal pointing generation as discussed in Subsection 7.2.3 (part 4) for a variety of bearing angle combinations

casing that no off-nominal pointing was applied. As can be observed in Figure 10.1, the output is in line with the expected output. The same procedure was repeated for combinations of 8° and the cube had moved twice as far, thereby demonstrating the expected behaviour. Based on this, the generation of off-nominal viewing cases was deemed verified.

10.1.2. Image rendering

As discussed in Subsection 7.2.4, the images are rendered using Blender's Python API. The main script involved with rendering the images sets the object's orientation w.r.t. world axes, it places the cameras according to the pose data files in the expected orientation, it sets the illumination conditions, and renders the images according to the desired settings. This test is designed to verify whether the script can correctly perform these steps resulting in the desired rendered images. The functionality of the functions underlying the scripts (Blender functions) have been assumed to be verified as previously discussed.
Input:

- Three different cubes that form a distinctive object that appears different from different viewpoints. This is shown in Figure 10.2, where the first cube has four Blender units of size, the second cube two Blender units of size, and a third cube with one Blender unit of size.
- A data file consisting of the different camera poses to be used, which are listed in Table 10.2
- The desired illumination conditions, type, and strength, for the given experiment.

Expected output: The expected output images are taken from Magalhães Oliveira (2018), where the orientation and relative positioning of the cubes are the only things that are considered.
Test passed: When through visual inspection it can be observed that the actual output image is similar to the expected output for the different poses. The relative size difference of the objects between the expected and actual result is not important, as the focus lies on using the correct camera placement (orientation), i.e., perspective effects are ignored.

Table 10.2: Table listing the nine different poses used to verify the image rendering process in Blender

Pose number	x [m]	y [m]	z [m]	q_0	q_1	q_2	q_3
1	20	0	0	0.5	0.5	0.5	0.5
2	0	20	0	0.5	0.0	-0.707107	-0.707107
3	-20	0	0	0.5	0.5	-0.5	-0.5
4	0	-20	0	0.707107	0.707107	0.0	0.0
5	0	0	20	0.707107	0.0	0.0	0.707107
6	0	0	20	1.0	0.0	0.0	0.0
7	0	0	-20	0.0	0.707107	-0.707107	0.0
8	0	0	-20	0.0	0.382683	-0.923880	0.0
9	10	-20	0	0.707107	0.707107	0.0	0.0

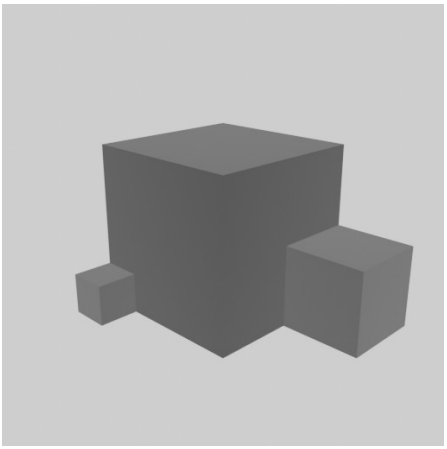


Figure 10.2: A view of the 3D scene used to verify the image rendering

Table 10.3: The Blender camera pose w.r.t the world frame that has been used for verification, i.e., $\mathbf{r}_{B/W}^{W}$ and $\mathbf{q}_{B/W}^{W}$

x [m]	y [m]	z [m]	q_0	q_1	q_2	q_3
7.3588914871	-6.9257907867	4.9583091736	0.7804827094	0.4835360348	0.2087036073	0.3368715942

The results can be seen in Figure 10.3 and it is observed that the output is in line with the expected output. Therefore, the generation of the images is deemed verified. As aforementioned, the differences in size of the cubes between the expected image and the actual result is irrelevant as well, as an orthographic projection is used.

10.1.3. Dataset annotation

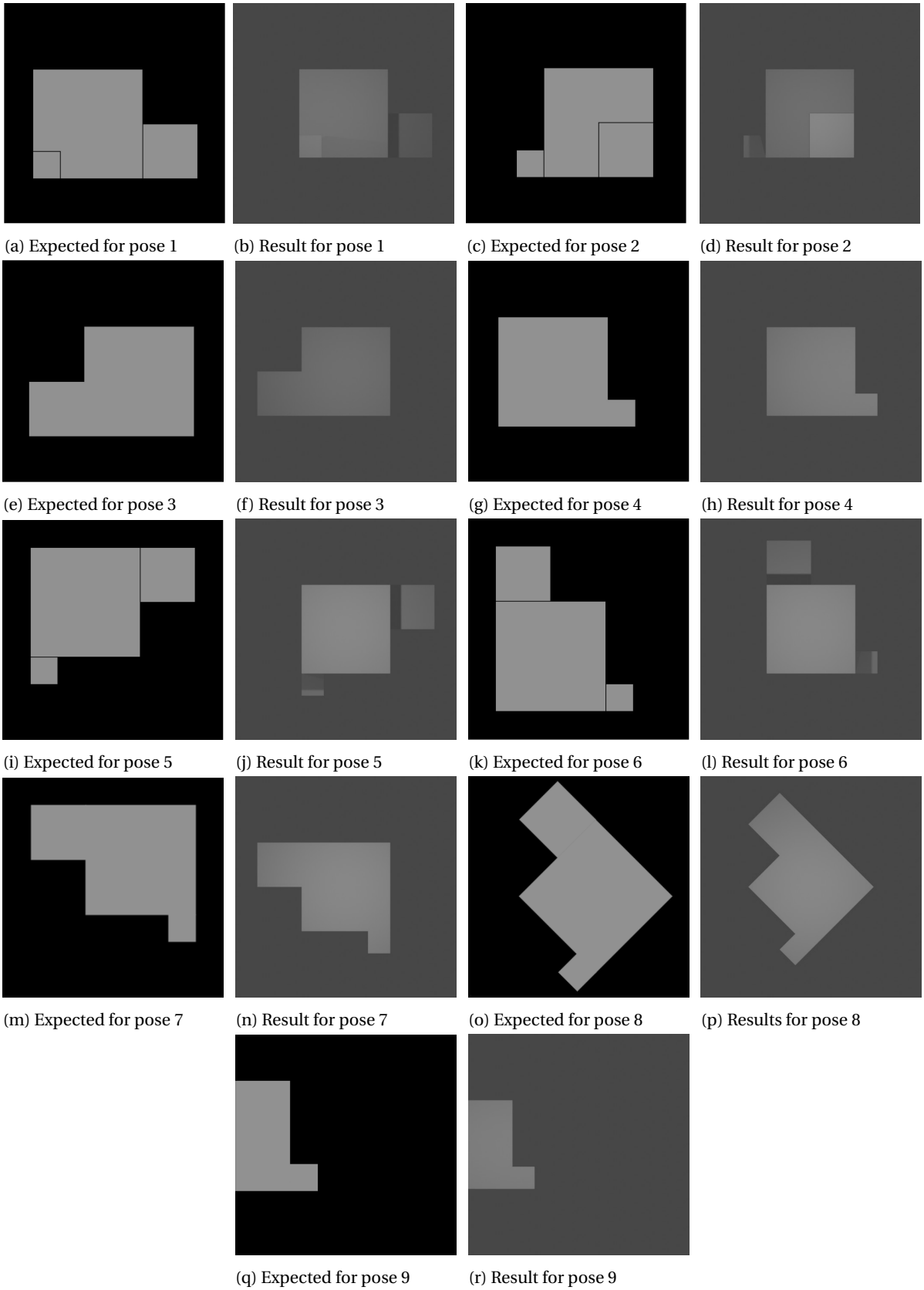
The images need to be annotated with their corresponding pose, bounding box, and keypoint coordinates to be able to train the machine learning networks. The low-level reference frame conversions and the kinematic functions used in this process have been verified and are listed in Table 10.1. Furthermore, the pose annotation simply consists of applying several reference frame conversion in a row to arrive at the desired pose presented in the camera reference frame, $\mathbf{C}_{C,A}$ and $\mathbf{r}_{A/C}^{C}$, as shown in Subsection 7.4.1. This is, therefore, also considered a low-level unit test. The emphasis is laid on the larger unit tests regarding the bounding box and keypoint annotation, which will be discussed in more detail. The keypoint annotation consists of two parts, namely the keypoint designation and annotation. As discussed in Subsection 7.4.2, the keypoints were designated on the asteroid's 3D model using the 3D SIFT algorithm. This algorithm is implemented in C++[4] in the open-source PCL[5]. This library is maintained and regularly updated and is used by many around the world for research projects or implementations in industry. Based on this, the chances of an error existing within the functions that are part of this library are slim and therefore, it is assumed that it is verified. The pose that is used for the verification of the bounding box and keypoint annotation is shown in Table 10.3.

Bounding box annotation

As discussed in Subsection 7.2.4, a script has been written, which interacts with Blender through its Python API. This script determines and outputs the coordinates of the tightly fitted bounding box presented in the image reference frame (P) around the target image for each image with the following encoding (x_{min}, y_{min}, w, h). Therefore, this test is devised to ensure that this is performed correctly and that these 'raw' coordinates can then be transformed to a different encoding and reshaped to annotate the images.
Input:

- The bounding box coordinates $\mathbf{x}^{P} = (x_{min}, y_{min}, w, h)$ presented in the pixel reference frame (P)
- The corresponding image

[4] `https://github.com/sjtuytc/betapose`, Date accessed: 7-10-2021
[5] `https://pointclouds.org/`, Date accessed: 7-10-2021



(a) Expected for pose 1 (b) Result for pose 1 (c) Expected for pose 2 (d) Result for pose 2

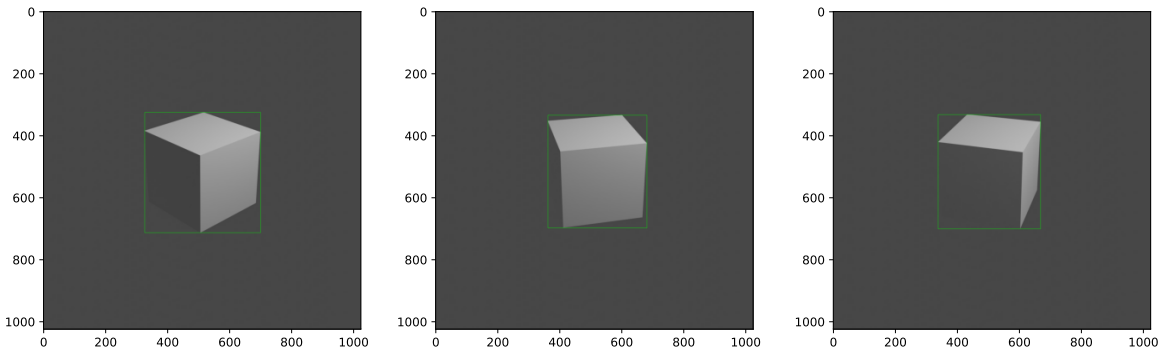
(e) Expected for pose 3 (f) Result for pose 3 (g) Expected for pose 4 (h) Result for pose 4

(i) Expected for pose 5 (j) Result for pose 5 (k) Expected for pose 6 (l) Result for pose 6

(m) Expected for pose 7 (n) Result for pose 7 (o) Expected for pose 8 (p) Results for pose 8

(q) Expected for pose 9 (r) Result for pose 9

Figure 10.3: The verification of the image rendering process where the expected result is compared to the actual output



(a) The tightly fitted bounding box using an object orientation of 0° w.r.t. world frame

(b) The tightly fitted bounding box using an object orientation of 60° w.r.t. world frame

(c) The tightly fitted bounding box using an object orientation of 120° w.r.t. world frame

Figure 10.4: The verification of the bounding box calculation using Blender for three different orientations of the target

Expected output: The expected output is an image in which the corresponding bounding box encompasses the target object present within the image.
Test passed: The test is passed when through visual inspection it can be seen that the bounding box correctly encompasses the target object.

As can be observed in Figure 10.4, the output is in line with the expected output for the three different orientations of the cube and therefore the generation of the bounding box coordinates is verified.

3D to 2D keypoint conversion

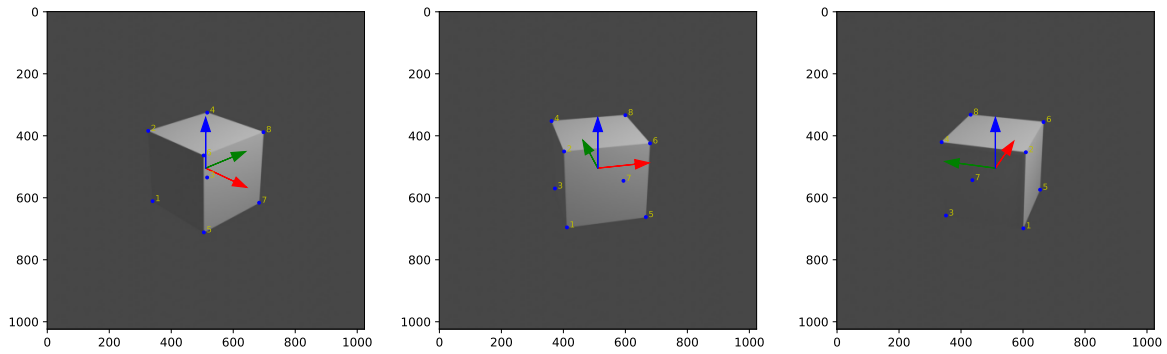
The keypoint annotation consists of several steps as outlined in Subsection 7.4.2. However, the reference frame transformations and kinematic functions required to attain $\mathbf{C}_{C,A}$ and $\mathbf{r}^{C}_{A/C}$ have been verified independently. Therefore, the remaining part of the chain consists of verifying the projection function, which projects the 3D keypoints to 2D through the use of the PnP equation (Section 4.2), which is re-stated below.

$$\begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{C}_{C,A} & | & \mathbf{r}^{C}_{A/C} \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \tag{10.1}$$

This test can also be viewed as a sub-system test as it consists of several reference frame conversions, the usage of kinematic functions, and the projection function, thereby testing their correct integration.
Input:

- The camera intrinsic parameter matrix $\mathbf{K}$
- The pose of the camera w.r.t. the object presented in the camera reference frame, i.e., $\mathbf{C}_{C,A}$ and $\mathbf{r}^{C}_{A/C}$.
- The 3D keypoint locations designated on the object presented in the object reference frame, i.e., $\mathbf{r}^{A}_{p/A}$

Expected output: The expected output is an image in which the keypoints are in the correct location and the object's axes are in the expected orientation w.r.t. the camera.
Test passed: The test is passed when through visual inspection it can be seen that the keypoints are on the expected location and that the object's axis are in line with their respective selected orientation.



(a) The 3D keypoints projected to 2D using an object orientation of 0° w.r.t. world frame

(b) The 3D keypoints projected to 2D using an object orientation of 60° w.r.t. world frame

(c) The 3D keypoints projected to 2D using an object orientation of 120° w.r.t. world frame

Figure 10.5: The verification of the 3D to 2D projection of the keypoints for three different orientations of the target. Thereby testing the entire pipeline consisting of kinematic function, reference frame transformations, and the projection function PnP

$$\mathbf{K} = \begin{bmatrix} 1422.22 & 0 & 512 \\ 0 & 1422.22 & 512 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{keypoints } (x, y, z) = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \tag{10.2}$$

An image of a cube, using the camera intrinsic matrix $\mathbf{K}$ shown in Equation (10.2), was generated in Blender. The pose of the Blender camera w.r.t. the world frame that has been used is shown in Equation (10.3), i.e., $\mathbf{r}_{B/W}^{W}$ and $\mathbf{q}_{B/W}^{W}$. The 3D keypoints are the vertices of the cube and are shown in Equation (10.2), the order matters and the first entry is the first keypoint and so on. The camera intrinsic matrix and the camera pose are used to convert each i^{th} 3D keypoint (x_i, y_i, z_i) to its 2D counterpart (u_i, v_i). The result is shown in Figure 10.5 and as can be observed the keypoints are in the expected location and aligned with the correct vertices. Furthermore, the object's axes are also in line with the expected orientations. Therefore, the function projecting 3D to 2D is assumed to be verified.

Desired formatting

The aforementioned tests have verified the basic building blocks underlying the dataset generation and annotation. However, for the training of the networks, several data formats were required as discussed in Sections 8.5 and 9.5, respectively. To train the object detection network the annotated data had to be converted to a `.TFRecord` file. A standard conversion script was provided by the TensorFlow Object Detection API, which was adjusted slightly. Furthermore, the annotations had to be converted to the COCO format (Code Listing C.2) to train the keypoint detection network. A script made available by Barad (2020) was adapted for the Bennu datasets and used to convert the annotations into the COCO format. The annotation process of linking the correct data with the correct image had been verified already. Therefore, a test was performed to merely verify that the annotation format was successfully created. Random entries were selected and through visual inspection of the bounding box and keypoints plotted on the image corresponding to the filename, it was concluded that the data was correctly converted. Furthermore, a test run was performed on the keypoint detection network using the annotated data format. The network shows the number of annotations it loads, as illustrated in Figure 10.6, which corresponded to the size of the respective dataset. Furthermore, the results outputted by the network were realistic, demonstrating that the annotation conversion was successful.

```
loading annotations into memory...
Done (t=2.34s)
creating index...
index created!
=> classes: ['__background__', 'Bennu']
=> num_images: 22646
=> load 22646 samples
loading annotations into memory...
Done (t=0.61s)
creating index...
index created!
=> classes: ['__background__', 'Bennu']
=> num_images: 4853
=> load 4853 samples
```

Figure 10.6: Example of the annotation print statement at the start of the training process of the keypoint detection network

10.2. Machine learning

The models that are used for object detection and keypoint detection are existing architectures that have been created using the open-source Python machine learning platforms TensorFlow and PyTorch, respectively. These open-source machine learning platform are built and maintained by Google and Meta AI Research. The source code of these architectures is made available on GitHub to foster a more inclusive research environment. These model architectures are made up of basic functions/building blocks, readily available in either TensorFlow or PyTorch. A plethora of machine learning networks that have been applied to real-world problems have been build using these machine learning platforms. Therefore, the functions that make up the model architecture, namely convolutional layers and pooling layers, can be assumed to have been extensively verified by the teams behind the creation of PyTorch and TensorFlow. Furthermore, these existing network architectures have been applied to different problems and their results have been published. Based on this, it is assumed that the selected network architectures and their building blocks are verified.

Object detection network

No changes were made to the network's architecture, and changes regarding the configuration were specified through a configuration file. The results outputted by the network are realistic and show good performance, indicating that the conversion to the `.TFRecord` file was successful. Therefore, it was concluded that the object detection network and its adaptations could be considered verified.

Keypoint detection network

In adapting the network to the asteroid dataset, several changes had to be made to the existing files. Two major things had to adapted, firstly the configuration file specifying the number of keypoints, the data, and configuration of the network. Secondly, the file that processes the annotated data used for training and evaluation (`bennu_coco.py`). Several functions within this file had to be adapted to make it suitable for the asteroid dataset. Care was taken in making these changes and the network has been trained and evaluated using the updated script. As previously discussed, the network shows the number of annotations it loads at the start of the training process as seen in Figure 10.6. The loaded number of annotations per dataset were equal to the actual number, demonstrating that the `bennu_coco.py` script had been successfully adapted. Furthermore, the results outputted by the network are realistic. The chances of getting strange results instead of realistic accurate results in the presence of an error caused by the adaptation of the `bennu_coco.py` script are much higher. Therefore, it was concluded that the keypoint detection network and its adaptations could be considered verified.

10.3. Pose estimation

The EP*n*P solver is available through OpenCV and it uses the same reference frame as the camera reference frame (C), so no further conversion was required. This library is maintained and regularly updated, and is used by many around the world for research projects or implementations in industry by companies, such as Google and Microsoft. Based on this, the chances of an error existing within the functions that are part of this library are slim. Therefore, it is assumed that it is verified. However, a test was performed to set a baseline of the performance of the network having perfect keypoint detections (ground-truth). The keypoints and

the camera intrinsic parameters matrix from Equation (10.2) are used and the pose of the image is given in Table 10.3. The error between the actual pose and the predicted pose was in the order of 10^{-8}, demonstrating that the EP*n*P works well and that with perfect measurements, a perfect pose estimate can be achieved.

IV

Results

11

Results and experiments

This chapter discusses the performance that was achieved by the developed algorithms and discusses the object detection, keypoint detection, and the pose estimation pipeline separately in Sections 11.1, 11.2, and 11.3, respectively. Furthermore, experiments that were performed to improve the results or allow for better understanding are also discussed in the respective sections. The best model was used for evaluation on the trajectories, which is discussed in Section 11.4. The chapter is concluded with a summary of the achieved results in Section 11.5.

11.1. Object detection

This section discusses the achieved performance and robustness of the object detection network. The network settings and training procedures that have been discussed in Section 8.5 were used to generate these results, unless specified otherwise. The performance is evaluated using the Intersection over Union (IoU) metric and the percentage of feasible detections (IoU > 0.75), Section 8.5. The IoU metric evaluates the similarity between the ground-truth bounding box and the predicted bounding box. The feasible detection metric is based on the COCO dataset, which evaluates object detection models, and refers to detections that can effectively be used by the subsequent keypoint detection network (Lin et al., 2014). The object detection network is the first part of the entire pose estimation pipeline, its performance is therefore crucial to allow for subsequent keypoint estimation and eventual pose estimation.

11.1.1. Accuracy assessment

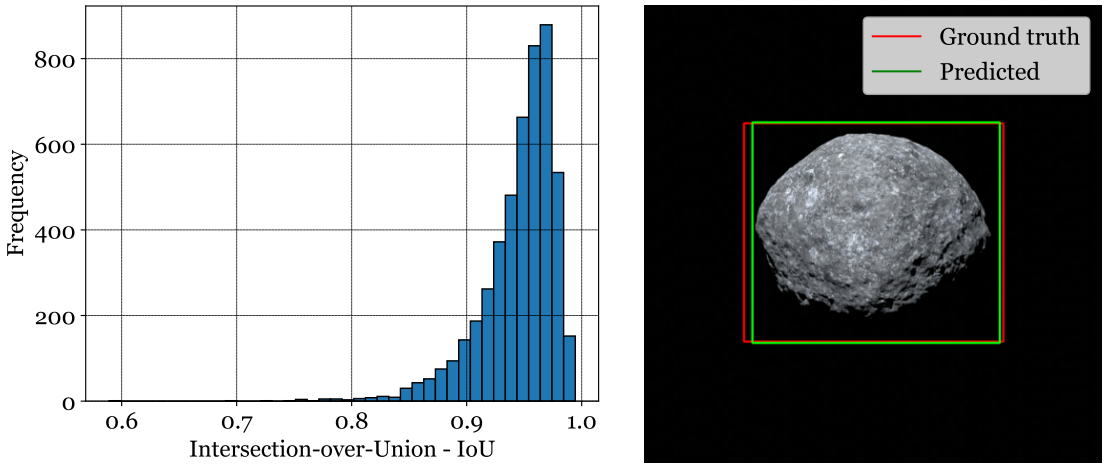
This subsection discusses the achieved performance of the networks on both the clean dataset Bennu and the augmented dataset Bennu+. As discussed in Section 8.5, the network uses default anchor boxes that have a certain aspect ratio. During inference the network uses these default anchor boxes to make a prediction, and the aspect ratio and scale of these anchor boxes will always be slightly different than the ground-truth bounding box, therefore the IoU values will never be exactly equal to 1.

Clean dataset: Bennu

The performance of the model trained and evaluated on the clean Bennu dataset (Bennu/Bennu) is shown in Table 11.1. The test set of the Bennu dataset consisting of 4853 images was used for the evaluation. The network has been trained for 50,000 steps and it could be observed that after the 7000th step the network

Table 11.1: Comparing the different inference results of the trained networks on the respective datasets

Network	Training/evaluation dataset	Mean IoU	Median IoU	IoU > 0.75
SSD-MobileNetV2-FPN-Lite	Bennu/Bennu	0.945	0.953	99.92%
SSD-MobileNetV2-FPN-Lite	Bennu/Bennu+	0.839	0.925	83.49 %
SSD-MobileNetV2-FPN-Lite	Bennu+/Bennu	0.959	0.962	100%
SSD-MobileNetV2-FPN-Lite	Bennu+/Bennu+	0.956	0.961	100%



(a) A histogram showing the distribution of the IoU values on the test dataset of the Bennu dataset

(b) A detection with an IoU of 0.94, demonstrating what the IoU values imply

Figure 11.1: Showing the distribution of the IoU values and an image with an IoU detection of 0.94 of the network trained and evaluated on the Bennu dataset (Bennu/Bennu)

showed signs of overfitting, as the training accuracy went up, while the validation accuracy went down. This checkpoint was therefore used as the final model.

The following can be observed from Table 11.1 and Figure 11.1a:

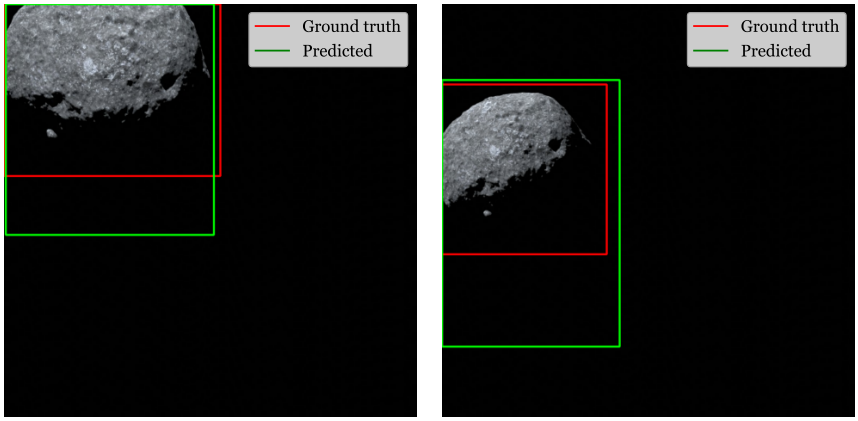
- The object detection network performs almost perfectly, with 99.92% of the cases having an IoU above 0.75 and the mean and median IoU are around 0.95. This means that only 4 out of the 4853 images may contain a bad detection. This good performance was expected, as this is a relatively simple object detection problem, namely the detection of a single bright object on a black background with no augmentations applied. Furthermore, the validation and test sets come from the same distribution as the training set and contain no augmentations. Therefore, the network was expected to easily recognize the patterns in the data and produce favorable results. Moreover, the majority of data points have an IoU above 0.9, with only a few outliers, demonstrating consistent performance of the network around the mean IoU.

The mean and median IoU of the predictions outputted by the network are around 0.94 and the meaning of this value is visually represented in Figure 11.1b, indicating that this detection perfectly detects the target within the image.

There are 4 images that have an IoU below 0.75 and they are shown in Figure 11.2. As can be seen, the asteroid is viewed from the top or bottom in these cases, resulting in only half of the asteroid being illuminated. Furthermore, a sub-optimal anchor box (aspect ratio > 1) is allocated for the prediction by the network, resulting in the incorporation of more background. However, the purpose of the object detection network is to detect the object within the image. Even though the detected bounding box does not perfectly match the ground-truth bounding box for the images shown in Figure 11.2, it still is able to detect the object and thereby achieving this purpose. The inclusion of more background within the image or slightly cropping a narrow strip of the asteroid will not influence the performance of the keypoint detection network, as the asteroid is almost fully visible allowing for the detection of the designated keypoints.

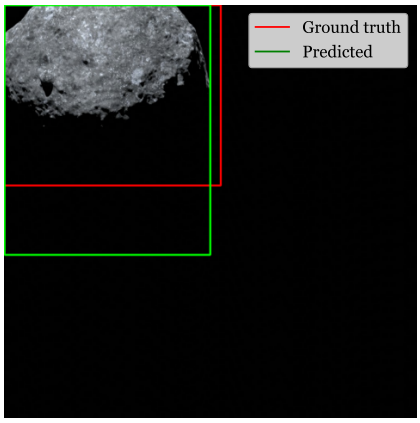
Augmented dataset: Bennu+

The Bennu+ dataset, as discussed in Subsection 7.5.1, consists of image corruptions and is used to emulate artifacts that may be present in real images. The performance of the model trained and evaluated on the Bennu+ dataset (Bennu+/Bennu+) is shown in Table 11.1. The val+ set of the Bennu+ dataset consisting of 4853 images was used for the evaluation. The network has been trained for 50,000 steps and it could be observed that after the 13,000th step the network showed signs of overfitting, as the training accuracy went up, while the validation accuracy went down. This checkpoint was therefore used as the final model. The longer

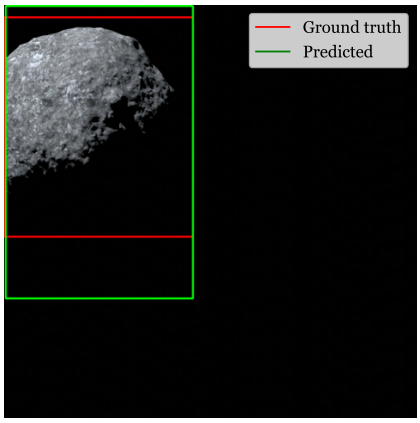


(a) IoU = 0.724

(b) IoU = 0.589

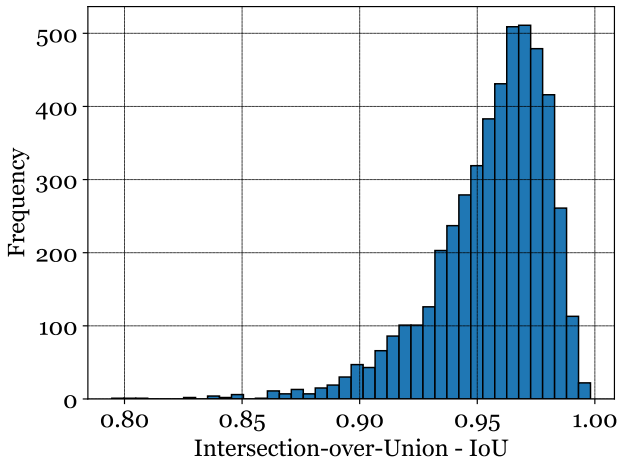


(c) IoU = 0.695

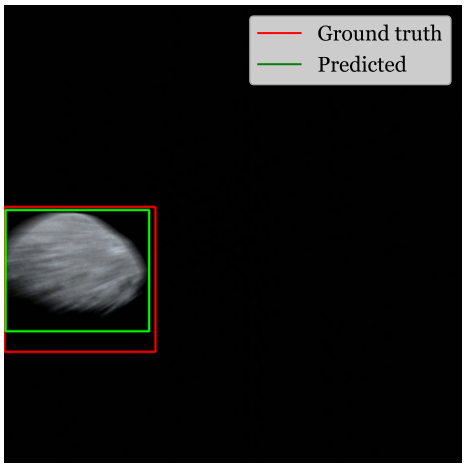


(d) IoU = 0.745

Figure 11.2: Detections of the asteroid by the object detection network trained and evaluated on the clean Bennu dataset (Bennu/Bennu) with an IoU < 0.75



(a) A histogram showing the distribution of the IoU values on the val+ dataset of the Bennu+ dataset



(b) The single detection with an IoU < 0.8

Figure 11.3: Showing the distribution of the IoU values and an image with an IoU detection of 0.79 of the network trained and evaluated on the Bennu+ dataset (Bennu+/Bennu+)

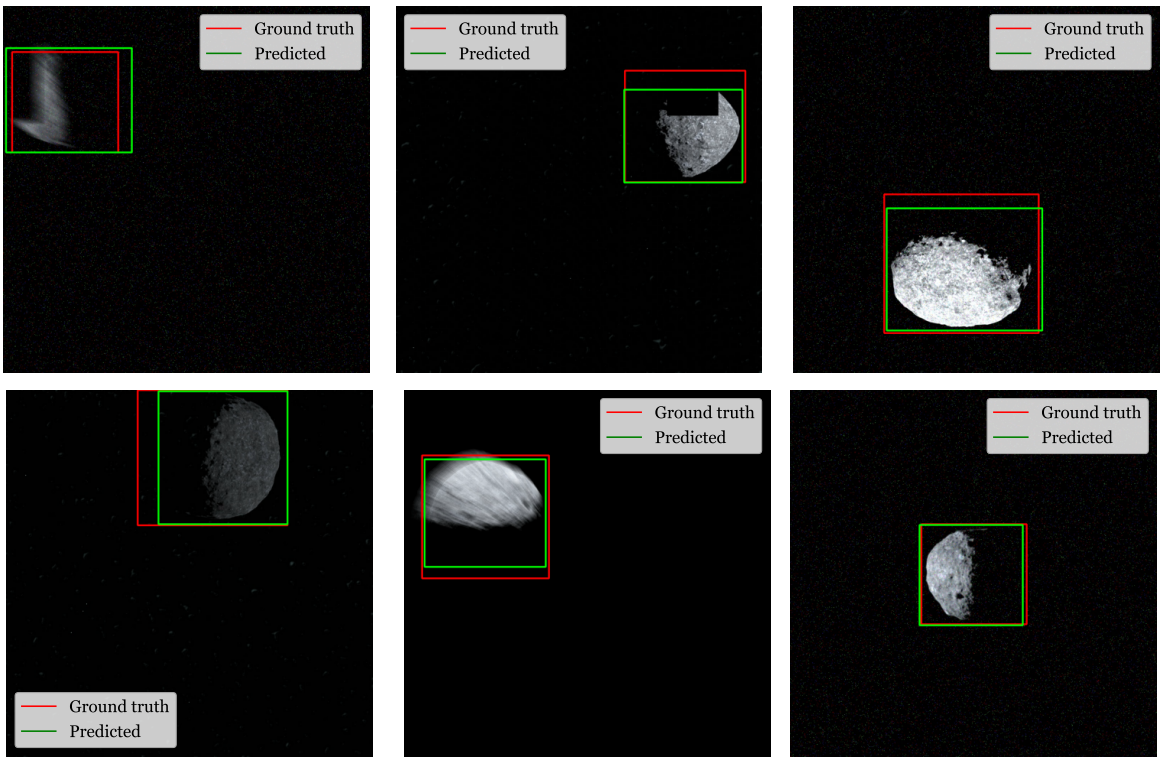


Figure 11.4: Detections of the asteroid by the object detection network trained and evaluated on the augmented dataset Bennu+ (Bennu+/Bennu+)

training duration compared to the clean dataset was expected, as the dataset is much more challenging and therefore finding the patterns underlying the data is more difficult.

The following can be observed from Table 11.1 and Figure 11.3a:

- The network performs perfectly with a 100% of the detections having an IoU above 0.75 and a mean and median performance around 0.96. This demonstrates that the network has learned to become invariant to the different image corruptions present in the Bennu+ dataset. Furthermore, the distribution illustrates that the majority of the detections have an IoU above 0.90, demonstrating that the network can consistently detect the asteroid within the image, with only a few outliers. Example detections are shown in Figure 11.4, where it can be seen that even with challenging augmentations the network is still able to properly detect the asteroid within the image.

11.1.2. Robustness assessment

As elaborated upon in Section 7.5, it is crucial that the networks are robust against image artifacts to allow for deep-learning systems that can be deployed on actual space missions. The robustness of the networks to unseen images from a different distribution is evaluated. This means that the network that was trained solely on clean images of the Bennu dataset will be applied to the Bennu+ validation set and the network that was trained on the Bennu+ dataset will be applied to the clean test dataset (Bennu). The results are shown in Table 11.1 for (Bennu/Bennu+) and (Bennu+/Bennu) respectively.

The following can be observed from Table 11.1 for Bennu/Bennu+:

- The network performs poorly compared to the network trained on the augmented dataset Bennu+, with only 83.49% (4052 images) resulting in feasible detections. These inaccurate detections will have a detrimental effect on the keypoint detection accuracy of the subsequent KD network, emphasizing the need for robustness. However, this drop in performance was expected as the network has been

trained solely on clean images and *affine* data augmentations, possibly resulting in the network to over rely on the pixel-level patterns present in the clean dataset. Examples of the failure cases are shown in Figure 11.5, where it can be observed that under the influence of augmentations the network is unable to accurately detect the asteroid within the image.

To get a sense of which augmentation effects the network the most, a new validation set was generated in which only one augmentation is applied per image. This set consists of 4853 images of the validation set of the Bennu dataset, and 70% of the images are randomly assigned one augmentation with the same severity levels as illustrated in Appendix D. The mean and median IoU of the network's performance to images containing the augmentation is shown in Table 11.2. The network shows robustness against a variety of image corruptions such as color jitter, speckle noise, random erase, shot noise, Gaussian blur, and motion blur. The network has not been trained on these corruptions, but is still able to accurately detect the asteroid within images containing these corruptions. This robustness could be inherent to the network architecture of the SSD detection head with the FPN neck, as Hendrycks and Dietterich (2019) proved that multi-scale networks are more robust to image corruptions. These effects are likely to exists in real images and therefore the network trained solely synthetic imagery would already bridge the domain gap rather well.

The performance of the network degrades when images contain defocus blur, Gaussian noise, spatter, zoom blur, and impulse noise, with the last two being the most detrimental to the networks performance. Spatter and impulse noise are especially relevant for space-applications as they can arise due to pluming effects or radiation. This further emphasizes the need to incorporate augmented images in the training process.

The following can be observed from Table 11.1 for Bennu+/Bennu:

- The network performs perfectly, with a 100% of the detections having an IoU above 0.75 and a mean and median performance around 0.96, outperforming the model trained solely on clean images. The network trained on the augmentations, has learned to become invariant to these corruptions and therefore rely less on the textures and more on the global shape. This could explain the better performance on the clean dataset.

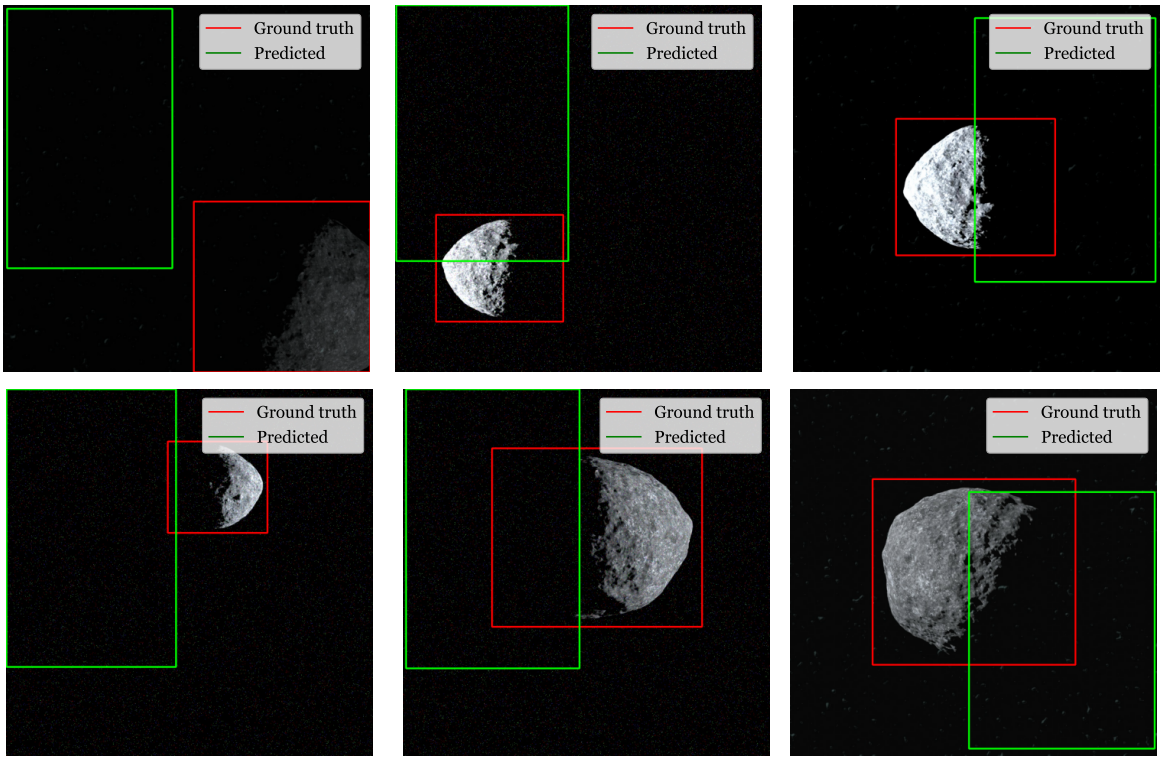


Figure 11.5: Detections of the asteroid by the object detection network trained and evaluated on the Bennu and Bennu+ datasets, respectively (Bennu/Bennu+)

Table 11.2: The performance of the model trained on Bennu on the single augmentation validations set

Augmentation	Number of images	Mean IoU	Median IoU	IoU > 0.75
None	1455	0.947	0.954	99.86%
Color jitter	315	0.946	0.955	100%
Speckle noise	300	0.946	0.955	100%
Random erase	308	0.943	0.953	100%
Shot noise	347	0.922	0.930	100%
Gaussian blur	304	0.917	0.928	99.34%
Motion blur	291	0.889	0.893	97.6%
Defocus blur	313	0.833	0.857	85.9%
Gaussian noise	281	0.801	0.845	79%
Spatter	321	0.772	0.846	77.26%
Zoom blur	293	0.690	0.689	27.65%
Impulse noise	325	0.506	0.622	40%

11.1.3. Robustness to real images

The performance of the object detection networks to five real images taken by the OSIRIS-REx spacecraft is evaluated[1]. However, as aforementioned, this number of images is not enough to perform a rigorous evaluation of the performance to real images and the so-called domain gap. Nonetheless, they do give some insights into the networks performance and therefore opening the door to future research. The networks are ran in inference and the performance can be observed in Figures 11.6 and 11.7.

The network trained solely on clean images and affine data augmentations was expected to perform slightly worse than the model trained on the Bennu+ dataset, due to the fact that real images might contain artifacts not present in the synthetic dataset. As can be seen when comparing Figure 11.6 and 11.7 this notion is true, however, both networks perform well. The real images allow for easy detections, as they do not contain challenging poses nor noticeable image corruptions, which explains why both networks perform (almost) equally well. Furthermore, it was found in Table 11.2 that the network trained solely on clean images is robust to a variety of image corruptions.

[1] `https://www.asteroidmission.org/galleries/spacecraft-imagery/`, Date accessed: 31-01-2022

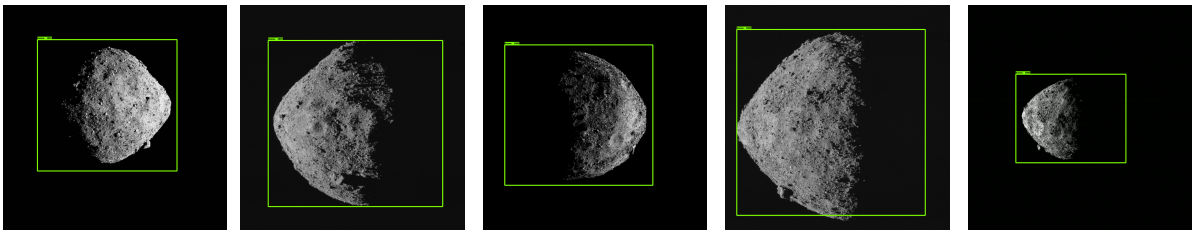


Figure 11.6: Performance of the network trained on the augmented dataset Bennu+ to real images taken by OSIRIS-REx

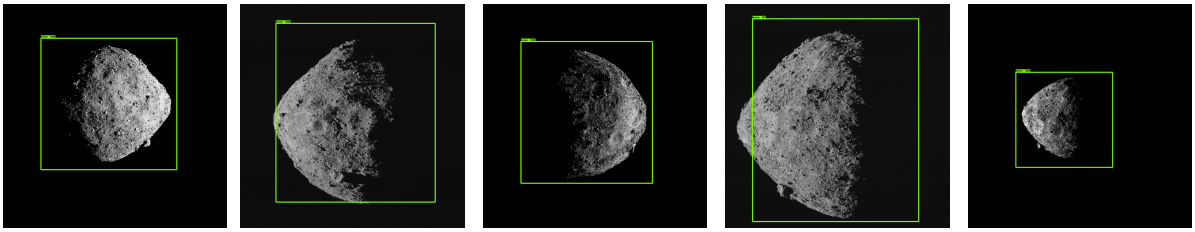


Figure 11.7: Performance of the network trained on the clean dataset Bennu to real images taken by OSIRIS-REx

Table 11.3: Comparison of the performance of the same network with different input sizes on the test set of the clean dataset Bennu. The errors are calculated using the predictions of the visible keypoints locations per image.

Network	Input size ($h \times w$) [px]	FLOPs [Bn]	Mean error [px]	Median error [px]	Normalized mean error [px]	Normalized median error [px]
LPN-101	256×192	1.451	3.98 ± 3.55	3.43	1.52 ± 1.08	1.40
	256×256	1.935	3.69 ± 1.84	3.19	1.43 ± 0.48	1.33

11.2. Keypoint detection

This section discusses the different experiments that were carried out and the performance and robustness achieved by the keypoint detection networks trained on the different datasets. The settings described in Section 9.5 were used to generate these results, unless specified otherwise. The results discussed in this section were obtained using the keypoint detection network in isolation, i.e., using the ground-truth bounding boxes as the RoI. The evaluation metric used throughout this section is given in Equation (9.10). Furthermore, the mean and median error used in the following discussions and reported in Tables 11.5 and 11.6 represent the error on the original image size, i.e., before rescaling of the RoI to the input size. The normalized error represents the input size agnostic error as discussed in the evaluation metrics part of Section 9.5.

The network predicts the location of all the 68 keypoints, however, some of these keypoint location are not within the image dimensions, which can occur for off-nominal pointing scenarios in which part of the asteroid is cut-off. The performance is listed for all the visible keypoints within the image, where visible refers to all the points present within the image dimension, also referring to points on the back of the asteroid. This can be observed in Figure 9.3. Firstly, the results of two experiments are discussed after which the achieved accuracy and robustness of the networks are addressed.

1. Effect of the input size

As discussed in Section 9.5, the input size has an effect on the FLOPs, and it is crucial to evaluate the effect of the input size to the network on the accuracy of the results. The following input sizes were tested to see the effect of the input size on the accuracy, namely 256×256 px and 256×192 px ($h \times w$).

The LPN network with the ResNet-101 backbone is used for the experiment and the networks use the same settings such as the data augmentations, learning rates, and optimizers as discussed in Section 9.5. The networks have been trained for 150 epochs and evaluated on the test set of the Bennu dataset. The best model obtained for each was taken for comparison and the results are shown in Table 11.3.

As can be observed in Table 11.3, the 256×256 px outperforms the 256×192 px input size. However, the increase in FLOPs is negligible as the network still has considerably less FLOPs than the state-of-the-art HRNet (Table 11.4). The reason for the square input size outperforming the 4 : 3 aspect ratio can be attributed to the bounding box aspect ratios of the Bennu dataset being predominantly between 0.8 and 1.0, as discussed in Section 9.5. This would allow the network to make more accurate predictions as the cropped and rescaled image is not elongated in any direction. This was therefore used for further experiments.

2. The effect of the capacity of the network

The LPN network used in this work has been selected based on its lightweight properties, allowing for future embedding on the resource-constrained spacecraft processor. However, this model has significantly less number of parameters (and consequently FLOPs) compared to the current state-of-the-art keypoint detection network HRNet used for keypoint detection on uncooperative spacecraft (Barad, 2020; Park et al., 2021; Pasqualetto Cassinis et al., 2021b) as shown in Table 11.4. The general notion is that the more parameters, the higher the capacity, i.e., the ability to estimate more complex non-linear functions. Therefore, the perfor-

Table 11.4: Comparison of the size and computational efficiency between LPN-101 and HRNet for the given input size

Network	Input size ($h \times w$) [px]	Parameters [Mn]	FLOPs [Bn]
LPN-101	256×256	5.37	1.94
HRNet	256×256	28.54	9.50

Table 11.5: The accuracy of the average keypoint detections by the networks for all visible keypoints per image with a maximum of 68 keypoints

Network	Training/evaluation dataset	Mean error [px]	Median error [px]	Normalized mean error [px]	Normalized median error [px]
LPN-101	Bennu/Bennu	3.69 ± 1.84	3.19	1.43 ± 0.48	1.33
	Bennu/Bennu+	59.0 ± 78.93	12.23	23.70 ± 30.35	5.02
	Bennu+/Bennu	3.30 ± 1.60	2.84	1.28 ± 0.40	1.18
	Bennu+/Bennu+	3.70 ± 5.71	3.00	1.44 ± 1.96	1.23
HRNet	Bennu/Bennu	10.79 ± 3.04	10.18	4.25 ± 0.40	4.20
	Bennu/Bennu+	82.49 ± 99.58	15.86	32.71 ± 37.47	5.21
	Bennu+/Bennu	12.80 ± 3.68	12.09	5.04 ± 0.52	4.96
	Bennu+/Bennu+	12.84 ± 3.73	12.01	5.08 ± 0.57	5.01

Table 11.6: The accuracy of the average keypoint detections by the networks for the 20 most confident keypoint detection per image

Network	Training/evaluation dataset	Mean error [px]	Median error [px]	Normalized mean error [px]	Normalized median error [px]
LPN-101	Bennu/Bennu	3.36 ± 1.66	2.92	1.31 ± 0.44	1.23
	Bennu/Bennu+	59.14 ± 86.66	10.15	23.77 ± 33.47	3.93
	Bennu+/Bennu	3.07 ± 1.44	2.66	1.19 ± 0.36	1.12
	Bennu+/Bennu+	3.44 ± 6.40	2.82	1.34 ± 2.13	1.17
HRNet-W32	Bennu/Bennu	3.98 ± 1.28	3.65	1.56 ± 0.24	1.54
	Bennu/Bennu+	71.82 ± 100.66	6.57	28.57 ± 38.22	2.50
	Bennu+/Bennu	4.82 ± 1.64	4.44	1.90 ± 0.33	1.85
	Bennu+/Bennu+	4.93 ± 1.72	4.51	1.95 ± 0.41	1.88

mance of the LPN is compared against this larger network to see whether the capacity of the network has an effect on the performance, not only on the clean images, but also on the corrupted dataset. Furthermore, the notion is that smaller models (simpler) are less prone to overfitting and therefore generalize better, i.e., more robust to unseen images.

The HRNet has been trained using the iterative training strategy (Section 9.5) and the same hyperparameter settings and training procedure as for the LPN model, discussed in Section 9.5. The results of HRNet are shown in Table 11.5 and as can be seen the network has much worse performance compared to the LPN model. There is a connection between the number of keypoints included in the analysis and the mean and median error, meaning that the more points are included the higher the mean and median error. This comes from the fact that the network may not able to accurately predict all the visible keypoints within an image, resulting in a higher average detection error. This effect is, however, more pronounced for HRNet, which sees a decrease of the mean and median error to 3.96 ± 1.26 px and 3.65 px, respectively, when the number of keypoints is decreased to the 20 most confident ones. Therefore, both networks are also compared on the 20 most confident detections, which is shown in Table 11.6.

However, comparing Tables 11.5 and 11.6 it can be inferred that HRNet is not able to predict all keypoints well and even more hyperparameter tuning could be required to achieve a model that is comparable to the performance achieved by LPN. However, training a neural network is not trivial and there is no guarantee whether the HRNet can achieve that comparable performance. Furthermore, this is not the main focus of this work. The main idea was to illustrate the effect of the capacity of a model on the accuracy that can be achieved on the Bennu datasets.

The LPN model is more consistent as it is able to output accurate predictions for almost all visible keypoints (Table 11.5). However, when comparing both the LPN and HRNet on the 20 most confident detections, it can be observed that the LPN network and HRNet achieve similar performance, where the LPN is able to achieve that with a fraction of the parameters and FLOPs, i.e., 81% less parameters and 80% less FLOPs. This demonstrates that the lightweight LPN model outperforms the current state-of-the-art model within keypoint detections on the Bennu datasets. This is immensely important for the application on embedded devices. The HRNet network is therefore not considered further.

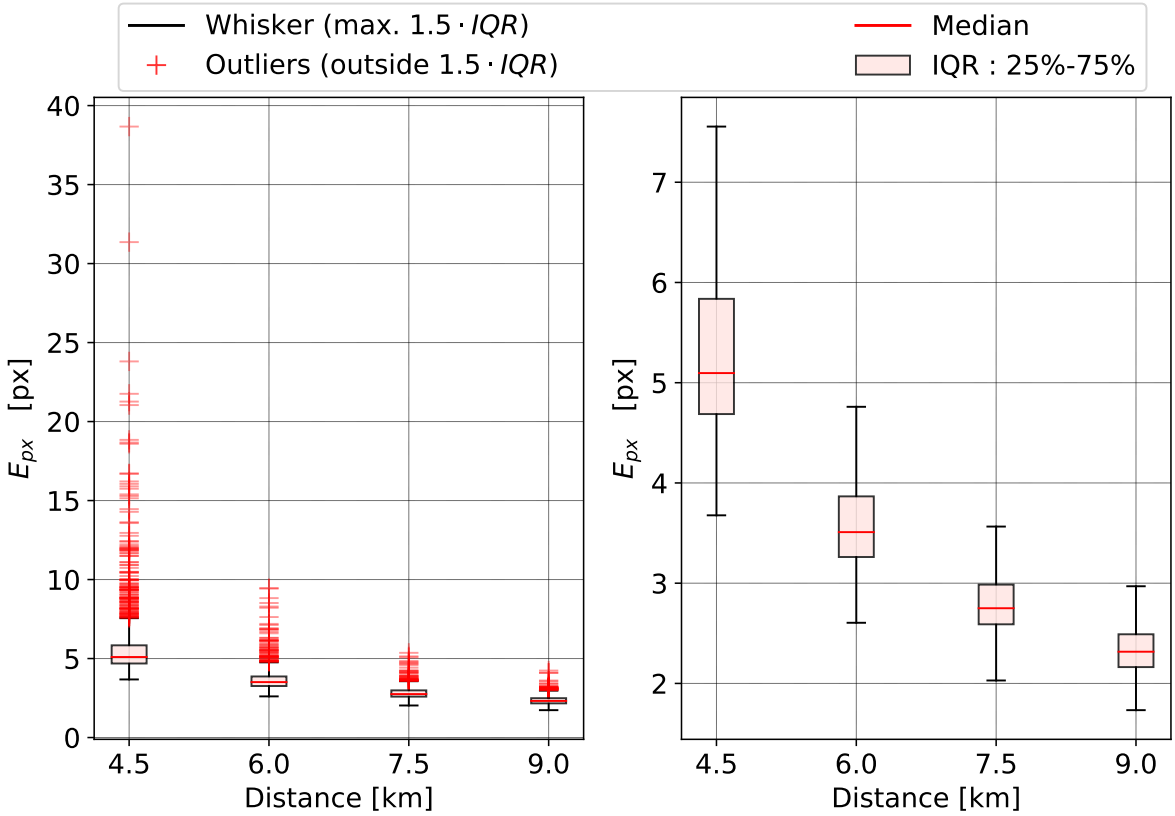


Figure 11.8: The performance of the LPN model trained and evaluated on the Bennu dataset (Bennu/Bennu) in terms of pixel error of the predicted location versus the ground-truth location (Equation (9.10)) for all visible keypoints within an image, shown per distance

11.2.1. Accuracy assessment

This subsection discusses the achieved performance of the networks on both the clean dataset Bennu and the augmented dataset Bennu+. The 256×256 px image size is used for all the networks.

Clean dataset: Bennu

The performance of the LPN-101 model trained and evaluated on the Bennu dataset (Bennu/Bennu) is shown in Table 11.5. The test set of the Bennu dataset consisting of 4853 images was used for the evaluation. The LPN model has been trained using the settings specified in Section 9.5. The iterative training strategy did not result in a better model.

The following can be observed from Table 11.5 and Figure 11.8:

- The mean error is around 3.7 px w.r.t the actual location, which is equal to about 0.36% of the total image size (1024×1024), demonstrating that the network achieves highly accurate predictions for the majority of the cases. Furthermore, the normalized mean and median error are around 1 px, which is about the same normalized performance as was achieved by HRNet-W32 trained and evaluated on the Envisat dataset by Barad (2020). Demonstrating that the keypoint detection is able to achieve good detection performance on the Bennu dataset.
- The performance of the network improves with distance, as the median error from 9 km is about 2.3 px whereas the median error from 4.5 km is around 5.1 px. This could be explained by the fact that from a shorter distance, the pixel area in which the keypoints can be found is larger, i.e., the spread in the distribution of the keypoints on the image plane is larger. Therefore, the keypoints locations are further apart and as such the predictions of the locations have to be more distinct. Whereas, for further distances the actual keypoint location can be found within a smaller range and the network's predictions have to be less precise on the pixel level to achieve the desired accuracy. Normally, an

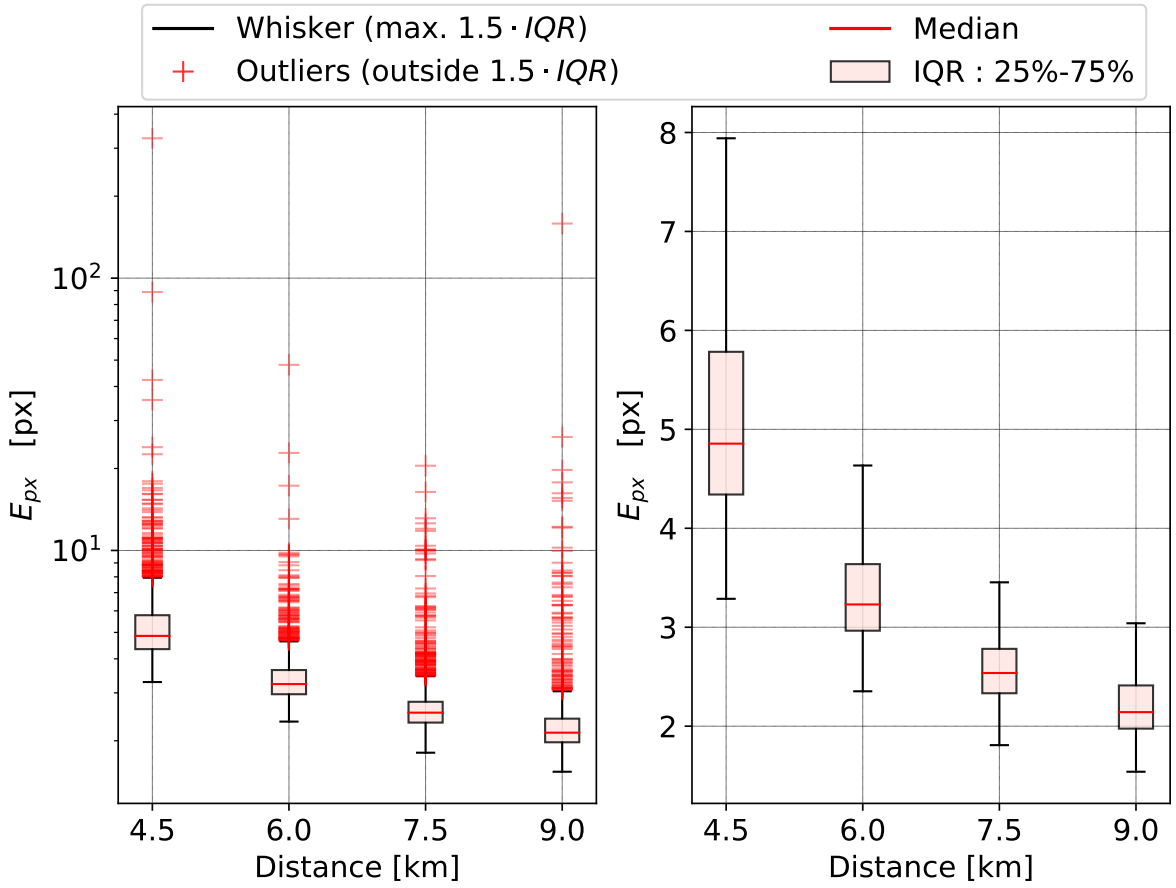


Figure 11.9: The performance of the LPN model trained and evaluated on the Bennu+ dataset (Bennu+/Bennu+) in terms of pixel error of the predicted location versus the ground-truth location (Equation 9.10) for all visible keypoints within an image, shown per distance

increase in distance would complicate keypoint detection, however, through the addition of an object detection network in front of the keypoint detection network this is mitigated.

- There are 275 outliers (5.67%) out of the total of 4853 images. Approximately half of these, are from a distance of 4.5 km to the asteroid. As elaborated upon in Subsection 7.2.3, the off-nominal pointing cases are generated using a fixed range of allowed angles. However, when closer to the object, the same angles cause the object to be cut off more as can be seen in Figure 11.20. This will not occur for distances further away as the maximum angle will still allow the object to be fully within the image, resulting in less challenging scenarios and therefore could result in better performance of the keypoint detection network.

Augmented dataset: Bennu+

The performance of the LPN-101 model trained and evaluated on the Bennu+ dataset (Bennu+/Bennu+) is shown in Table 11.5. The val+ set of the Bennu+ dataset consisting of 4853 images was used for the evaluation. An iterative training strategy as discussed in Section 9.5 was used and was successful in improving the performance of the network. The network was trained for one more stage using the best model of the previous stage for the initialization. The LPN model has been trained using the settings specified in Section 9.5.

The following can be observed from Table 11.5 and Figure 11.9:

- The mean and median detection error of around 3.7 px and 3.0 px are similar to the model solely trained and evaluated on clean image. This indicates that the network is able to accurately detect the keypoints within the images, even when they contain a variety of image corruptions. This tends to indicate that

the network has become invariant to these challenging illumination conditions, brightness/contrast changes, and other image corruptions and relies more on the global shape of the object and the inherent spatial relationship between the keypoints. This is desired behavior and would allow the network to generalize well to different textures and corruptions. Furthermore, the normalized mean and median error are around 1 px, which is about the same normalized performance as was achieved by HRNet trained and evaluated on the augmented Envisat+IC dataset by Barad (2020). Demonstrating that the keypoint detection is able to achieve good detection performance on the Bennu+ dataset.
- A similar trend of an improvement in the detections for larger distances can be observed as was discussed for the clean dataset.
- There are 403 outliers (8.3%) of the total of 4853 images. This increase was expected as the dataset is much more challenging than the Bennu dataset, consisting solely of clean images. Furthermore, as can be observed in Figure 11.9 there are a few very high errors, indicating that the keypoint detection network was unable to accurately detect all the visible keypoints within the image resulting in a high average keypoint detection error. This occurs on heavily augmented images, such as the top left image, shown in Figure 11.18, which is unlikely to happen in reality. These large outliers result in the increase of the standard deviation as shown in Table 11.5, which is heavily influenced by large deviations as it takes the square of them. Pasqualetto Cassinis et al. (2021b) uses the Mean Average Deviation (MAD) for this reason to indicate the spread of the data around the mean, which relies on the absolute difference w.r.t. the mean. However, Figure 11.9 also serves as an illustration of the spread of the data.

11.2.2. Robustness assessment

The robustness of the networks to unseen images from a different distribution is evaluated. This means that the network that was trained solely on clean images of the Bennu dataset will be evaluated on the val+ set of the Bennu+ dataset, and the network that was trained on the Bennu+ dataset will be evaluated on the test set of the clean Bennu dataset. The results are shown in Table 11.5.

The following can be observed from Table 11.5 for Bennu/Bennu+:

- The model trained solely on the clean images and with *affine* augmentations does not generalize well to the corrupted dataset consisting of a variety of image corruptions. This can be concluded from the large mean, median, and standard deviation of 59 px, 12 px, and 79 px, respectively, which points toward poor keypoint detection accuracy, with many outliers on which the keypoint detection fails completely. This can be attributed to the lack of corruptions in the training data that would make the network invariant to things, such as texture and brightness/contrast changes. Whereas the network trained solely on clean images has overfitted to the pixel-level patterns present in the clean dataset. This further emphasizes the importance of the training procedure and dataset in creating robust models.

The following can be observed from Table 11.5 and Figure 11.10 for Bennu+/Bennu:

- The model trained on the Bennu+ dataset generalizes well to the clean images and achieves even better performance on the clean images compared to the clean model, with a mean and median performance of 3.3 px and 2.8 px, respectively. This can be attributed to the fact that the network has become more invariant to a variety of factors such as texture, illumination conditions, and corruptions and therefore it relies less on the texture and more on the global shape. There are 264 outliers (5.44%) out of the total of 4853 images, which is a slight decrease compared to the clean model. Furthermore, the outliers are less high as can be observed when comparing Figures 11.10 and 11.8.

The model trained solely on clean images is not taken forward for the pose estimation pipeline due to poor robustness and generalization performance. Therefore, only the keypoint detection networks trained on the corrupted dataset Bennu+ are analyzed. Detailed error analysis is performed during the evaluation of the entire pipeline and is discussed in Section 11.3.

11.3. Pose estimation

This section discusses the evaluation of the performance of the entire pose estimation pipeline as shown in Figure 11.11, which consists of the CNN-based feature detection part and the EPnP solver.

The object detection and keypoint detection networks make up the CNN-based feature detection parts, which takes an image as the input and outputs the predicted 2D locations of the pre-defined keypoints. The

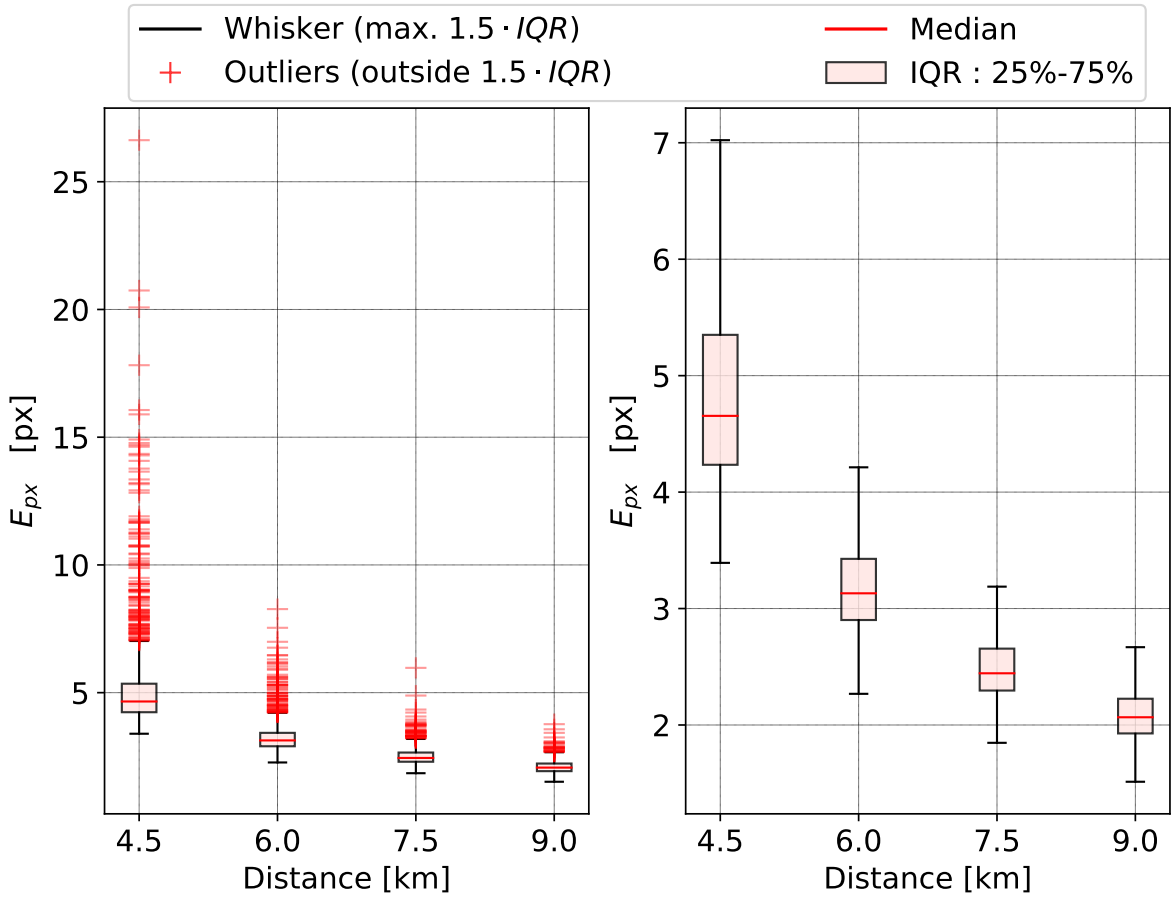


Figure 11.10: The performance of the LPN model trained and evaluated on the Bennu+ and Bennu dataset, respectively (Bennu+/Bennu), in terms of pixel error of the predicted location versus the ground-truth location (Equation 9.10) for all visible keypoints within an image, shown per distance

object and keypoint detection networks have been trained and evaluated independently as discussed in Sections 11.1 and 11.2, respectively. However, the input to the EP*n*P pose solver consists of the detected keypoints by the entire CNN-based pipeline (OD and KD together). Therefore, the pose estimation performance is evaluated using the entire pipeline meaning that the bounding box coordinates outputted by the object detection network are used to crop the RoI, which is fed to the keypoint detection network. The KD network then predicts the locations of the keypoints and feeds those points to the EP*n*P solver alongside the corresponding 3D points, based on which the distance of the asteroid w.r.t. camera is calculated.

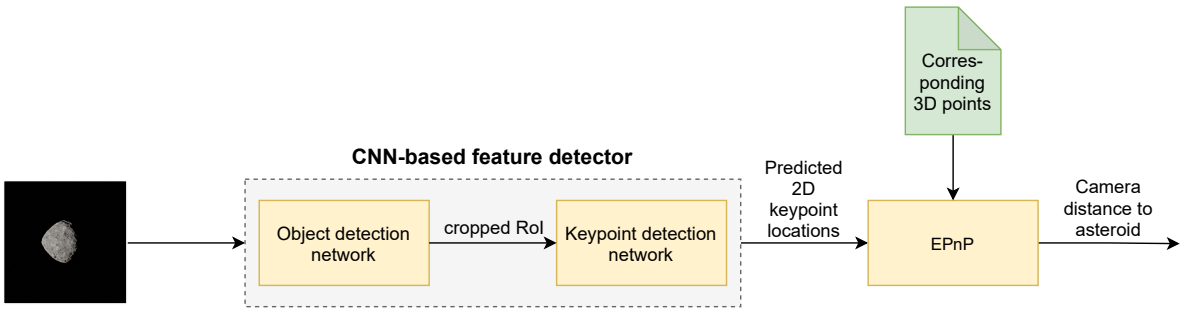


Figure 11.11: A schematic overview of the pose estimation pipeline consisting of the CNN-based feature detector and the P*n*P solver

The keypoint detection network outputs predictions for all 68 designated keypoints, however, as previously discussed, some of the keypoints locations are not within the image (Figure 9.3). The predictions for the visible keypoints within the image are sent to the EP*n*P solver. The EP*n*P solver only needs a minimum of 6 keypoints to be able to produce a unique solution. The performance of the pose solver, however, increases when more, accurate detections are available (> 6).

When using all the visible keypoints within the image, the performance was found to deteriorate slightly, especially on the more challenging images of the Bennu+ dataset. The overall standard deviation of the mean performance increased. This is caused by the keypoint detection network accurately detecting all the visible keypoints within some images, whereas for other images this does not occur, resulting in the usage of inaccurate detections by the pose solver leading to a poorer pose estimation. This effect of the number of keypoints on the overall accuracy of the keypoint detection can be observed when comparing Tables 11.5 and 11.6.

Heuristic approach

Therefore, two heuristic approaches have been implemented and tested to see which resulted in the best achievable performance of the EP*n*P solver. Both approaches are intended to mitigate the usage of bad detections that would result in inferior pose estimates.

1. **Threshold approach:** This approach adaptively feeds keypoints and their corresponding 3D points to the EP*n*P solver based on the confidence level of detection. This confidence level is equal to the peak value of the heatmap as discussed in Section 9.5. All the detections with a confidence above a certain threshold value are send to the pose solver, with a minimum desired number of keypoints of 8, which has shown to improve performance of the EP*n*P compared to the bare minimum required 6 points. Whenever only 7 or less keypoints detections surpass the threshold, the most confident keypoint is added until it reaches the desired 8 keypoints. However, when all the detections have a confidence value below the set lower bound, no new detections are added if 6 detections have already been found.

   A drawback of this method is that not every detection is as accurate and depending on the difficulty of the image, the heatmap detections do not have to be perfectly Gaussian but spread out blobs as shown in Figure 9.4b. Consequently, the peak values of these blob heatmaps will also be lower, resulting in the general confidence level for the keypoints to be lower as well, even though the accuracy of the actual location can be high. This could result in only a few detections surpassing the general threshold, whereas the solver would benefit from having more points of which the network is fairly sure as well. Furthermore, the cut-off point may be sensitive to the number of detections that will be used, as several detections may have approximately the same confidence value.

2. **Confident keypoints approach**: This approach uses the n most confident keypoint detections, where as before the confidence level is equal to the peak value of the heatmap as discussed in Section 9.5. This approach does not implement a 'general' threshold for all the images, but bases the selection of keypoints on the 'difficulty' of the image. This tries to tackle the disadvantage of the *threshold approach*, where the uncertainty in the detection influences the number of points that will be send to the pose solver. A drawback of this method is that in essence still a general number for all images is set, i.e., the n most confident keypoints.

These approaches could be further optimized or multiple other approaches could be devised, however, that is not the main focus of this work. Furthermore, as discussed in Section 4.3, the CEPP*n*P solver could possibly improve the performance of the pipeline. The usage of the CEPP*n*P pose solver deviates from the aforementioned approaches in the way that the CEPP*n*P allows the individual assessment of each keypoint detection per image, whereas the aforementioned approaches do set a general value for all images in their respective ways. However, as aforementioned due to time-constraints this is not analyzed in this work, but it will be elaborated upon in Section 12.2.

The best settings of the threshold and confident keypoints approach have been found by varying the threshold and the n most confident keypoints, respectively. The threshold has been varied between 0.70 and 0.95 in 0.01 increments, whereas the n number of most confident keypoint detections have been varied between 8 and 50, firstly in increments of five, after which the range was narrowed around the best performing candidates and increments of one were used. The *confident keypoints approach* outperformed the *threshold approach*. Comparing the *threshold approach* with the *confident keypoints approach* for the pipeline trained and evaluated on the Bennu+ dataset, the following became clear:

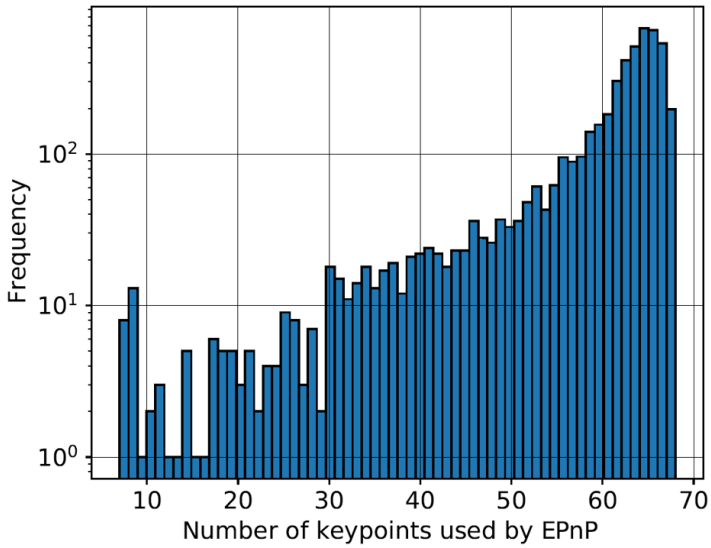


Figure 11.12: A histogram showing the distribution of the number of keypoints send to the EP*n*P solver, when applying the *threshold approach* on the model trained and evaluated on the Bennu+ dataset with a threshold value of 0.83

- By setting a general threshold, the number of keypoints used by the pose solver differ significantly as can be observed in Figure 11.12. Furthermore, the *threshold approach* resulted in 8 failure cases compared to zero failure cases for the *confident keypoints approach*. All these failure cases only have 7 or 8 keypoint detections because of the confidence levels regarding the detections. This demonstrates the difficulty of finding a correct general threshold value that works well on the entire dataset. It was found that for these images, about 20 keypoint detections that were fairly accurate had a confidence value just below the threshold. Because the minimum amount of keypoints were achieved and these confidence values were below the threshold, they were not included. However, challenging images that contain difficult poses and augmentations may benefit from the inclusion of more (fairly accurate) points.

Based on this, the *confident keypoints approach* was used for the final analysis.

11.3.1. Accuracy assessment

Only the performance of the pose estimation pipeline trained on the Bennu+ dataset is evaluated, as this showed the best generalization and robustness performance of the object and keypoint detection. The pipeline is evaluated by analyzing the translational error, which is calculated using the following, where $\mathbf{t}^C$ and $\hat{\mathbf{t}}^\mathbf{C}$ represent the ground-truth and predicted translational position vector, respectively (Figure 4.2).

$$E_t = \left| \hat{\mathbf{t}}^\mathbf{C} - \mathbf{t}^\mathbf{C} \right| \tag{11.1}$$

The results are shown in Table 11.7. The val+ set of the Bennu+ dataset and the test set of the Bennu dataset, both consisting of 4853 images, were used for evaluation. Furthermore, Figures 11.13 and 11.15 show the performance of the distance estimation as a function of the distance, to capture the trend of the decreasing distance estimation accuracy for increasing relative distance. The percentage error is shown alongside the absolute distance error to get more insights into the behavior of the pipeline, as the larger the distance, the higher the allowable margin (%).

The following can be observed from Table 11.7 and Figure 11.13 for Bennu+/Bennu:

- The pipeline trained on Bennu+ is able to achieve great performance on the clean dataset Bennu, satisfying the accuracy requirement (Section 2.3) in 100% of the cases. The pipeline is able to on average estimate the *line-of-sight* distance within 43 m and a median performance of 30 m. The median performance for the different distances lies between 21.5 m at 4.5 km and 42.6 m at 9 km.

Table 11.7: The results of the pose estimation pipelines using the *confident keypoints approach*, where for both pipelines the 39 most confident keypoints are used

Pipeline	Training/evaluation dataset	Mean $\mathbf{E}_t$ [m]	Mean $\|\mathbf{E}_t\|$ [m]	Median $\mathbf{E}_t$ [m]	Median $\|\mathbf{E}_t\|$ [m]	Failure cases [#/%]
SSD-LPN-EPnP	Bennu+/Bennu	(2.608 2.065 42.53)	42.82	(1.709 1.358 29.62)	29.86	0/0
SSD-LPN-EPnP	Bennu+/Bennu+	(3.077 2.501 47.96)	48.32	(1.889 1.469 31.09)	31.36	1/0.021

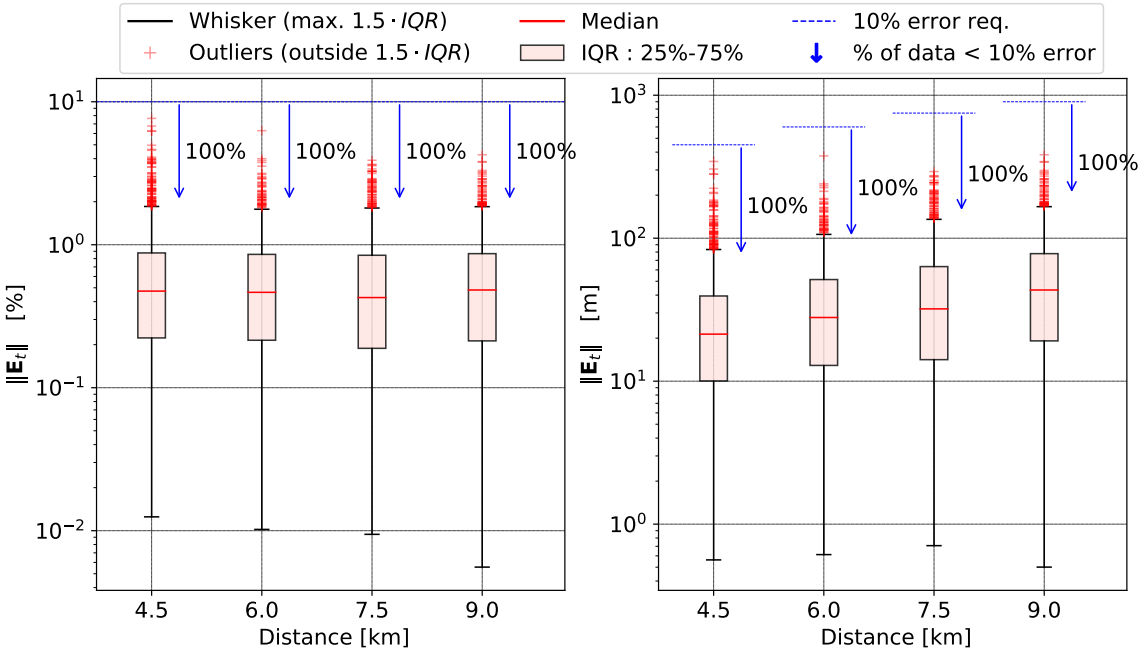


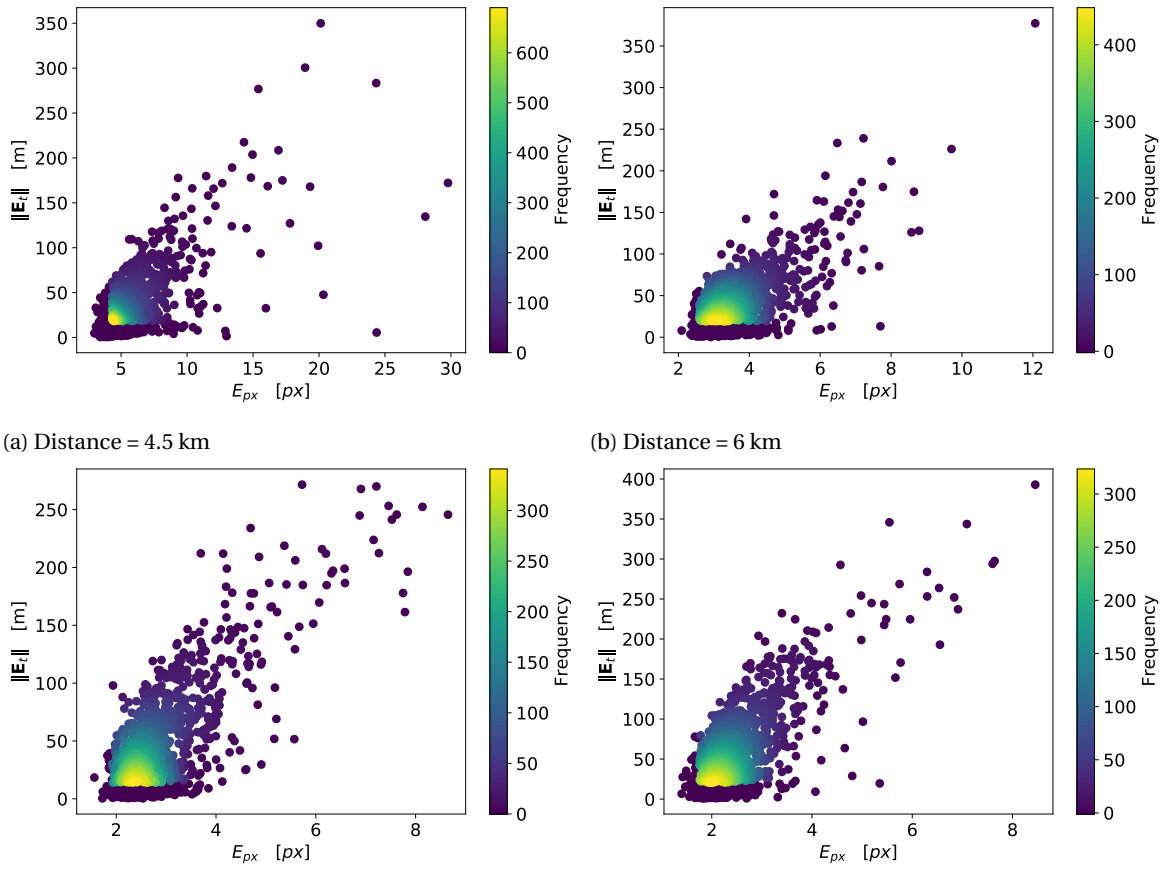
Figure 11.13: Performance of the SSD-LPN-EPnP pipeline that has been trained and evaluated on the Bennu+ and Bennu dataset, respectively, the outliers make up 5.23% (254 images) of the total of 4853 images

It can be observed that the performance decreases slightly with increasing distance. However, this is in accordance with the behavior of pose solvers such as the EPnP and CEPPnP (Sharma and D'Amico, 2016). The performance of these pose solvers deteriorate when the distance along the optical axis is increased and this has already been observed by Park et al. (2019) and Sharma and D'Amico (2017). Estimating the range from 2D imagery is challenging and this is caused by the nonlinear relation existing between z^C (Equation (4.4)) and the pixel location of the detected keypoints. This means that the pose solver is sensitive to pixel errors persisting in the detection when the relative distance increases, resulting in inaccurate pose estimates. This can also be observed in Figure 11.14, where the average detection error E_{px} for larger distances are generally more accurate, as was also observed in Figures 11.8 and 11.9. However, a relatively large pixel error has a much larger effect on the resulting pose error for these large relative distances. Furthermore, for all distances it can be observed that in general a lower keypoint prediction error results in a lower pose estimation error.

- Comparing the performance of the pipeline on the clean images of the Bennu dataset, with the performance obtained on the augmented val+ dataset, it can be seen that a lower mean and median translational error $\|\mathbf{E}_t\|$ of 42.82 m and 29.86 m can be obtained. The images present in the clean dataset do not contain augmentations and are therefore less challenging, resulting in improved performance of the overall pipeline.

The following can be observed from Table 11.7 and Figure 11.15 for Bennu+/Bennu+:

- The pipeline is able to achieve great performance, satisfying the accuracy requirement for 99.979% of the images present in val+. The pipeline is able to on average estimate the *line-of-sight* distance within 48.3 m and a median performance of 31.5 m. The median performance for the different distances lies between 22.5 m at 4.5 km and 42.6 m at 9 km. The mean and median error of 48.3 m and 31.5 m, respectively, are very accurate given the large relative range to the target (4.5 km to 9 km) and show that the



(a) Distance = 4.5 km

(b) Distance = 6 km

(c) Distance = 7.5 km

(d) Distance = 9 km

Figure 11.14: The relationship between the average keypoint detection error per image E_{px} and the resulting pose estimation error $\|\mathbf{E}_t\|$ evaluated using the test set of the Bennu dataset

CNN-based feature detection pipeline is able to achieve a desired performance. The percentage error w.r.t. the ground-truth distance shows similar performance for the different distances. However, the absolute distance error increases with relative distance. This trend follows the expected behavior of a pose solver for which the performance deteriorate when the distance along the optical axis is increased, as previously discussed.

Statistical certainty of the performance

The pipeline was shown to perform excellent, satisfying the accuracy requirement for all but one case. This accuracy represents the skill of the CNN-based pose pipeline, however, the statistical certainty regarding that expected accuracy also needs to be defined. This refers to determining the confidence intervals for that model skill, representing the likelihood that the accuracy of the system will fall in between those ranges when applied to new data, i.e., the accuracy of the pipeline is x ± y at 90% confidence level. This is performed through *bootstrapping*, which is a robust way to determine confidence intervals for machine learning models. *Bootstrapping* is a general method that allows to quantify statistics regardless of the underlying distribution of skill scores. The process of bootstrapping is briefly summarized below:

1. Randomly sample with replacement a population of 60-80% of the total dataset size n times, where for n, values between 1000 and 10,000 are recommended. For each population sample n, calculate the statistic of interest, which in this case is the mean and median $\|\mathbf{E}_t\|$ expressed in the absolute distance as well as the % difference w.r.t the ground-truth distance.
2. Calculate the confidence intervals around the median performance of all the samples, as no distribution is assumed. For a confidence level of 90% the lower bound is the 5^{th} percentile and the upper bound is the 95^{th} percentile.

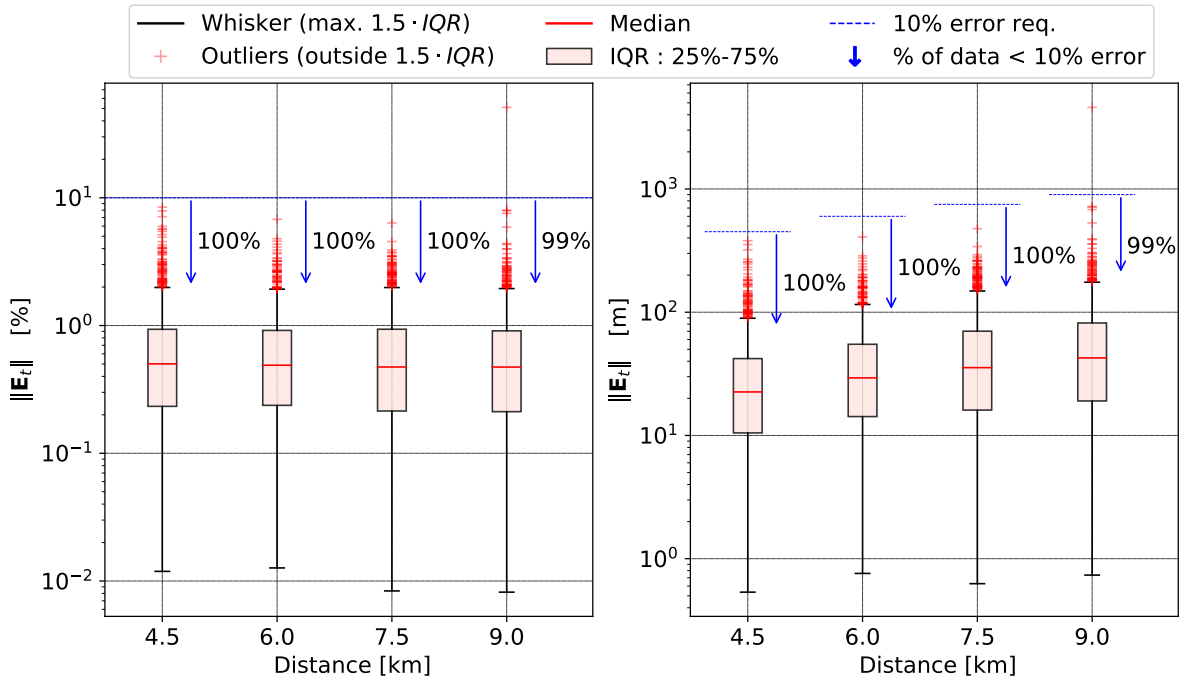


Figure 11.15: Performance of the SSD-LPN-EP*n*P pipeline that has been trained and evaluated on Bennu+, the outliers make up 5.96% (289 images) of the total of 4853 images

Table 11.8: The confidence intervals of the respective pipelines for a 90% confidence level, where the absolute error and the % error w.r.t. the ground-truth distance is given

Pipeline	**Training/evaluation dataset**	**Mean $\|\mathbf{E}_t\|$ [%]**	**Mean $\|\mathbf{E}_t\|$ [m]**	**Median $\|\mathbf{E}_t\|$ [%]**	**Median $\|\mathbf{E}_t\|$ [m]**
SSD-LPN-EP*n*P	Bennu+/Bennu	[0.644; 0.646]	[42.78; 42.85]	[0.461; 0.463]	[29.81; 29.88]
SSD-LPN-EP*n*P	Bennu+/Bennu+	[0.715; 0.717]	[48.10; 48.29]	[0.483; 0.485]	[31.32; 31.38]

This procedure was applied to both the pipeline evaluated on the clean dataset Bennu as well as on the augmented dataset Bennu+, and the results are shown in Table 11.8. Based on the accuracy requirement formulated in Section 2.2 and restated below, a confidence level of 90% was used. Furthermore, use was made of 1000 random samples consisting of 80% of the total population.

- **SR06**: The pose estimation pipeline shall have a relative line-of-sight distance to the center of mass of the target with a knowledge error lower than 10% of real distance with 99.73% probability at 90% confidence level.

As can be observed in Table 11.8, both pipelines achieve excellent performance ($\ll 10\%$) and the confidence intervals are small, meaning that the predicted performance is consistent. The pipeline is able to predict the distance on average within 43 m and 48 m with a 90% confidence level, respectively, for the clean and augmented datasets. Furthermore, the number of failure cases for the Bennu+/Bennu pipeline is zero and for the Bennu+/Bennu+ is one (0.021%). Therefore, it can be concluded that the designed pipeline satisfies the accuracy requirement with the required statistical certainty.

11.3.2. Outlier analysis

As can be observed in Figures 11.13 and 11.15, there are some outliers. The process of how the CNN makes it predictions is hard to understand, as it operates as a sort of black-box. However, these outliers were inspected further to understand the trends that underlie these predictions and to formulate hypothesis about what might explain this behavior. Detailed analysis on the pipeline trained and evaluated on Bennu+ is discussed first, as this is the most challenging dataset, after which the outlier trends of the pipeline evaluated on the clean images are discussed. The camera poses of the images of the clean and augmented dataset are the

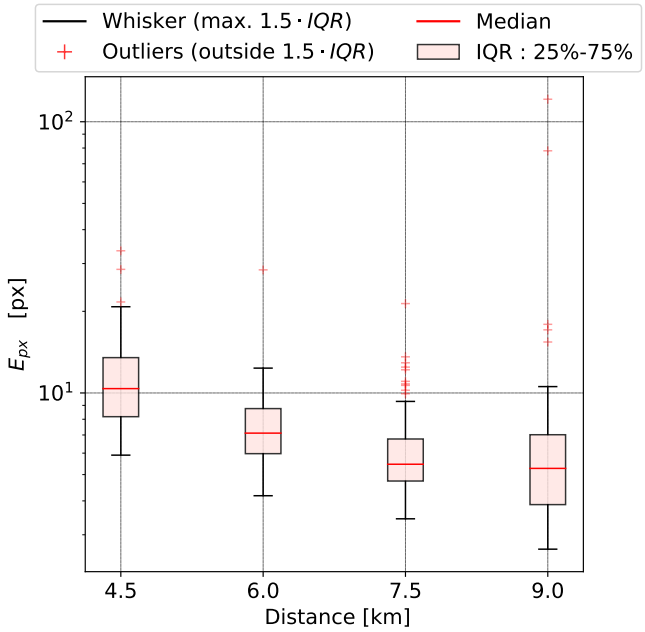


Figure 11.16: Boxplot of the average keypoint accuracy per image for the outlier images of the SSD-LPN-EP*n*P pipeline trained and evaluated on the Bennu+ dataset

same, the only difference are the image corruptions that have been added.

Augmented dataset: Bennu+

The outliers as shown in Figure 11.15 make up only 5.96% (289 images) of the total number of predictions. Their mean and median performance is 206.2 m and 180.3 m, respectively, and as observed in Figure 11.15, all but one satisfy the accuracy requirement (Section 2.2). Firstly, it was found that 79.6% (230 images) of the outliers were cases in which the camera was pointing off-nominally, meaning that the asteroid was not in the center of the image. This off-nominal pointing could result in part of the asteroid being cut off as can be seen in Figure 11.20, meaning that the network has less information, such as texture and global shape, to base its predictions on and as such resulting in less precise keypoint predictions. This hypothesis is also strengthened by the accuracy of the average keypoint predictions for the outliers, which is shown in Figure 11.16. The mean and median are 8.71 ± 8.94 px and 6.82 px, respectively, with a maximum outlier of ≈ 120 px error. This performance is above the mean and median of the entire dataset as listed in Table 11.7. This indicates that the keypoint detection network struggles with these images, which consequently results in sending inaccurate predictions to the EP*n*P solver, which leads to a less precise estimate of the distance. Furthermore, the median performance of the different distances are 10.4 px, 7.1 px, 5.5 px, and 5.3 px, respectively, and therefore much larger than the medians of the entire dataset as shown in Figure 11.9. Even though the accuracy of the keypoint detections increases with increasing distance, the relatively large pixel errors are especially detrimental to the eventual pose estimation for larger relative distances (7.5 and 9 km, Figure 11.14) as previously discussed.

To provide more insights into why the keypoint detection network would struggle with these images, the camera position w.r.t. the asteroid was analyzed and this is shown in Figure 11.17. The distribution of outliers per distance is pretty similar, 27.3% (79 images), 20.8% (60 images), 28.4% (82 images), and 23.5% (68 images) for the distance of 4.5 km, 6.0 km, 7.5 km, and 9.0 km, respectively. This demonstrates that the overall performance for each distance is similar and that the network struggles with some of the most challenging images for each distance. Figure 11.17 shows that the outliers are predominantly images taken from high inclinations (below and above the asteroid) and about 27.7% (80 images) of the outliers have the position $x^W = 0$, meaning that the camera is on the concentric circle of $x^W = 0$. This results in challenging images, as at most half of the asteroid is illuminated. Furthermore, combining this with off-nominal pointing results in images such as shown in Figure 11.20, again meaning that the network has little information to base it predictions on, complicating the keypoint predictions. Furthermore, it was found that the network struggles with the

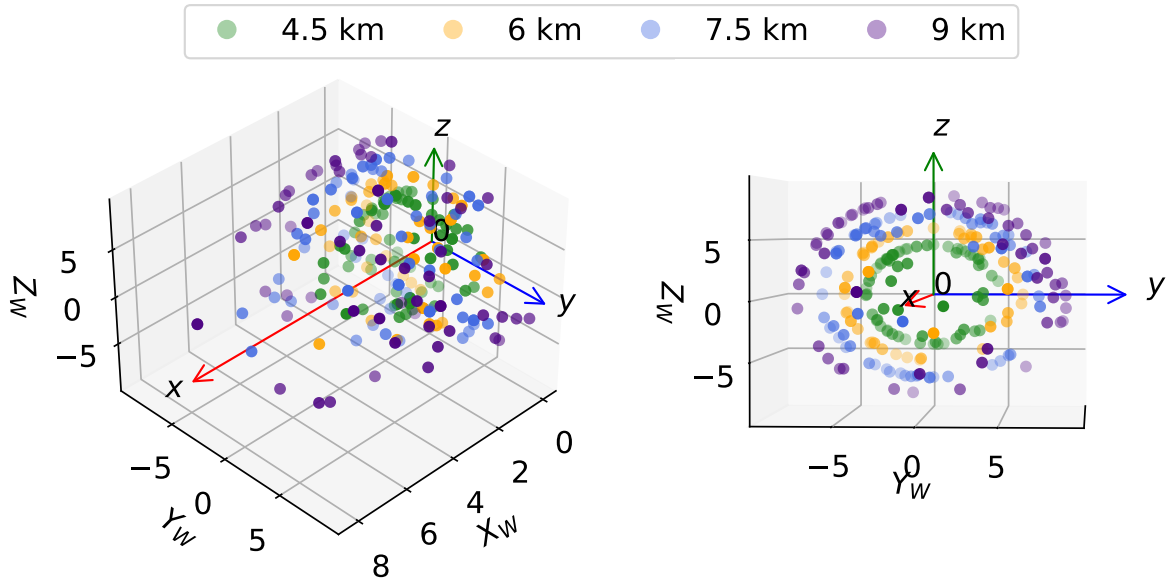


Figure 11.17: Distribution of the poses for the outlier images of the SSD-LPN-EP*n*P pipeline trained and evaluated on Bennu+

same camera orientation from a different distance as can be observed in Figure 11.20 (bottom two images from the left). Moreover, the same was observed for different asteroid orientations, which points to the challenging camera orientation (high inclination and off-nominal pointing) as the possible reason for the lower performance of the keypoint detection network.

Furthermore, images taken from this concentric circle around $x^W = 0$ may show similarities when viewing the asteroid from the top or bottom. The symmetry problem is a common problem for keypoint detection networks, as the keypoints that the network has to predict may appear the same as other keypoints, resulting in reduced detection accuracy (Zhao et al., 2018). Furthermore, the designated keypoints on the asteroid appear predominantly on the rim of the asteroid when viewed from high inclinations (including directly above and below) as can be observed in Figures 11.19a, 11.19b, and 11.19c, whereas from different views the keypoints are more spread over the asteroid as shown in Figure 11.19d. The combination of high inclination, off-nominal pointing, and challenging illumination conditions could result in the slightly worse performance of the network on these images.

Moreover, relevant for the pipeline evaluated on the augmented dataset Bennu+, 74% (214 images) of the outlier images are augmented with the corruptions discussed in Subsection 7.5.1. The distribution of the number of augmentations per image is: 71 images have one augmentation applied, 88 images have two augmentations, 48 images have three augmentations, and 7 images have 4 augmentations applied. Challenging combinations of augmentations can result in images shown in Figure 11.18, thereby resulting in images in which the asteroid is severely affected, complicating the keypoint detection. However, in reality it is highly unlikely that such combinations of effects will occur simultaneously.

Clean dataset: Bennu

Similar patterns that were found for the outliers of the Bennu+ dataset, were found for the outliers on the clean images of the Bennu dataset. The outliers only make up 5.23% (254 images) of the total number of predictions. Their mean and median performance is 165.82 ± 57.94 m and 164.69 m, and as observed in Figure 11.13 they all satisfy the accuracy requirement. The off-nominal pointing cases made up 77.2% (196 images) of the outliers, indicating that even in the absence of corruptions, the network struggles with images in which the global shape in not observable due to parts being cut-off (Figure 11.20). Furthermore, the mean and median keypoint detection errors were also larger compared to the overall mean, 7.15 ± 3.98 px and 6.21 px, respectively, and 24.8% (63 images) were present on the concentric circle around $x^W = 0$. Furthermore, similar poses as shown in Figure 11.17 were found for the clean images that the network struggles with.

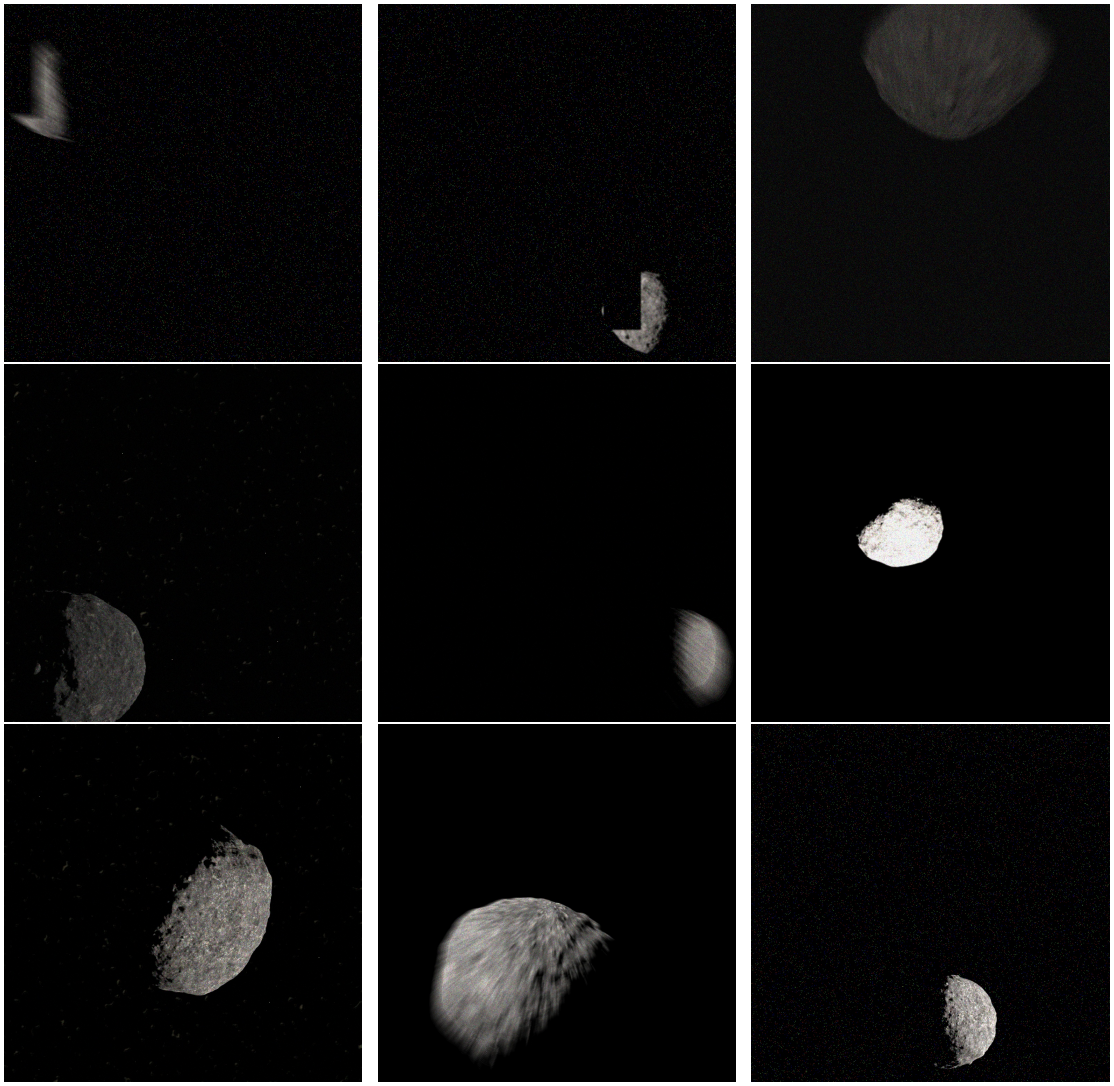


Figure 11.18: A selection of outlier images to show how augmentations can change the original image and thereby complicating the keypoint detection

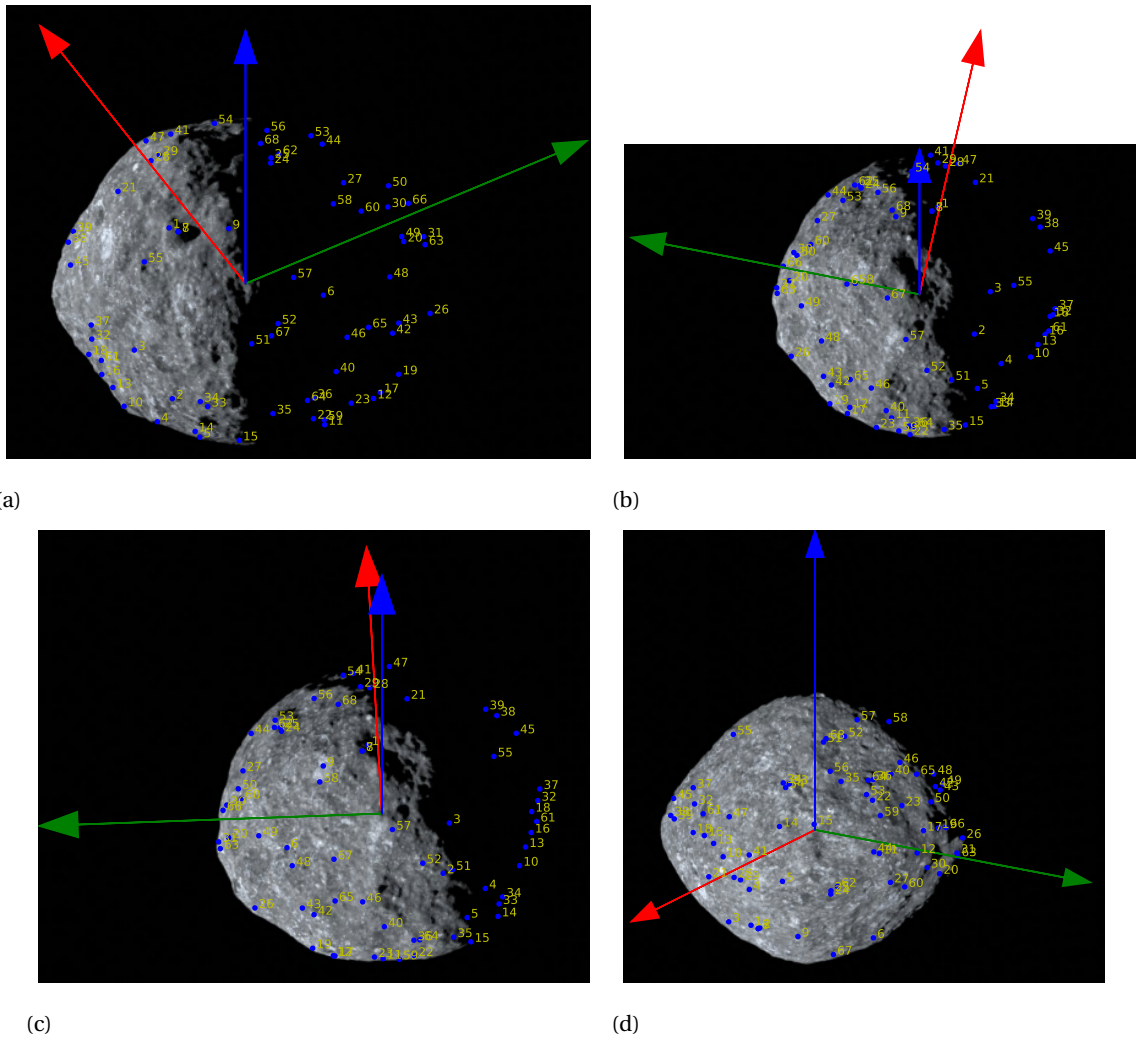
Conclusion

Based on the outlier analysis performed on both the Bennu+ and Bennu dataset, it points to the hypothesis that the network predominantly struggles with the challenging poses that have high inclinations and challenging illumination conditions in combination with off-nominal pointing cases. These challenges are compounded when augmentations are applied to the images. However, the pipeline still produces satisfactory results for these images, which satisfy the accuracy requirement and would still allow the spacecraft to navigate safely, even under the influence of these challenging inclinations, off-nominal pointing, illumination conditions, and image corruptions.

11.3.3. Possible improvements

As previously discussed, the network's detections are not always perfect and for challenging images the network can struggle with certain keypoints, which can influence the resulting pose estimation when these inaccurate detections are sent to the pose solver. However, the PnP solver could be made robust to this by combining the pose solver with RANSAC, which can reduce the effect of inaccurate detections (outliers) that are sent to the pose solver. The performance of the respective pipelines when using RANSAC alongside the EPnP solver is shown in Table 11.9

The following could be observed from Table 11.9 and analysis on the performance:



(a) (b)

(c) (d)

Figure 11.19: Visualization of the distribution of the keypoints for high inclination poses with $x^W \approx 0$ (a,b,c) alongside an example for a low inclination and far from $x^W = 0$ (d)

Table 11.9: The results of the pose estimation pipelines using the *confident keypoints approach* alongside RANSAC, where Bennu+ refers to the train+ and val+ dataset, respectively, and Bennu to the train and test dataset, respectively. The 39 most confident keypoints are sent to the pose solver

Pipeline	**Training/evaluation dataset**	**Mean $\mathbf{E}_t$ [m]**	**Mean $\|\mathbf{E}_t\|$ [m]**	**Median $\mathbf{E}_t$ [m]**	**Median $\|\mathbf{E}_t\|$ [m]**	**Failure cases [#/%]**
EP*n*P+RANSAC	Bennu+/Bennu	$(2.619 \quad 2.081 \quad 42.80)$	43.09	$(1.741 \quad 1.366 \quad 30.35)$	30.45	0/0
EP*n*P+RANSAC	Bennu+/Bennu+	$(3.005 \quad 2.425 \quad 47.13)$	47.49	$(1.915 \quad 1.490 \quad 31.53)$	31.69	0/0

- The performance for the respective pipelines is similar compared to the pipeline without RANSAC as shown in Table 11.7. Therefore, the inclusion of RANSAC does not influence the overall accuracy that much. This could be caused by the fact that the current outlier images are simply difficult and that the majority of the keypoint detections for those images are inaccurate, meaning that the usage of RANSAC will not be able to increase the accuracy of the detection. However, the usage of RANSAC removed the failure cases for the augmented pipeline (Bennu+/Bennu+), indicating the benefit of using the RANSAC method for those challenging images. Furthermore, it was found that the number of outliers for the augmented pipeline decreased by 5 images. Moreover, it was found that the spread of the outliers had decreased for some distances, meaning that the majority of the outliers moved closer to the whisker.

This shows that improvements to the distance estimates could be made by using a combination of dif-

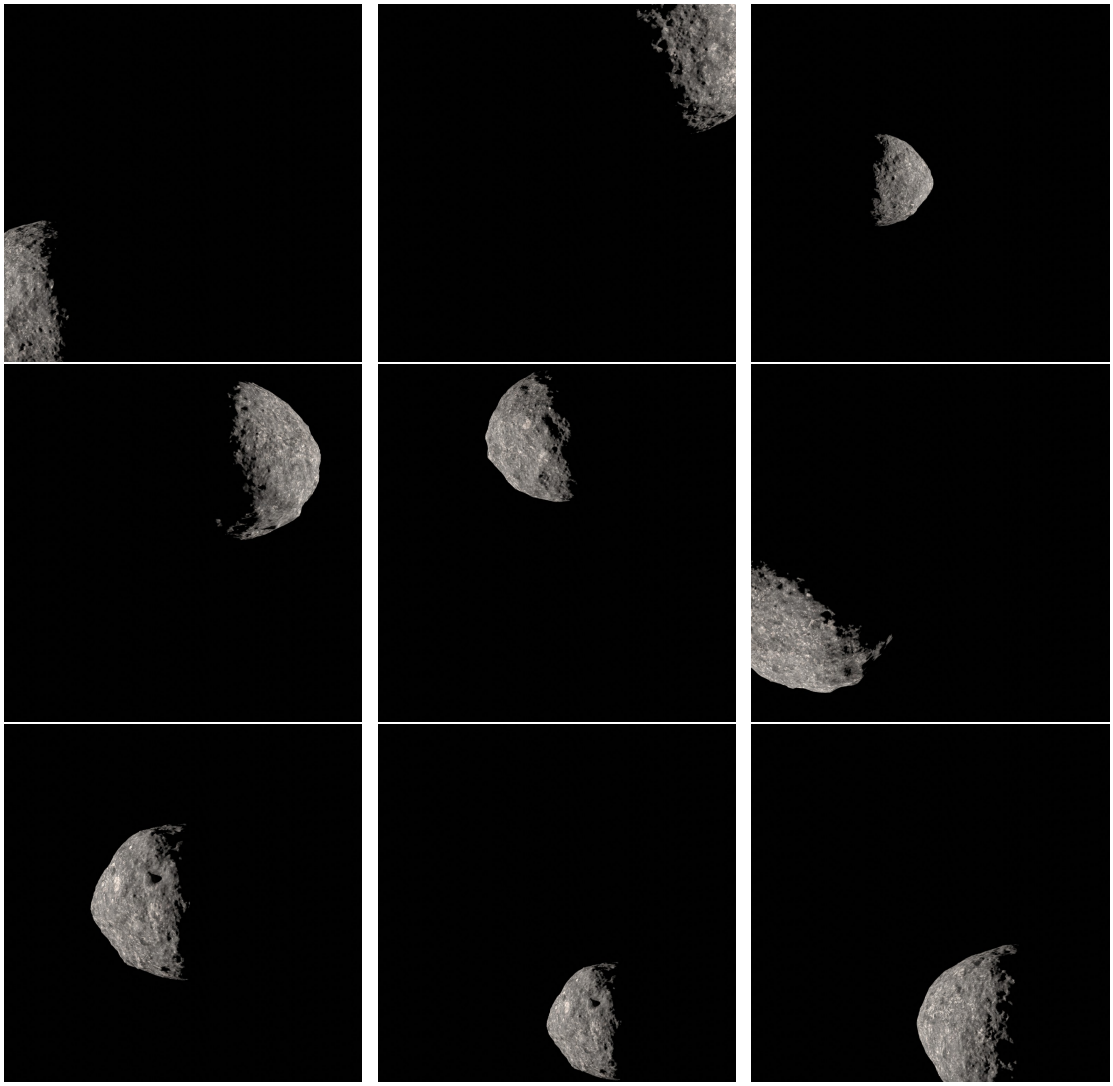


Figure 11.20: A selection of outlier images to showcase non-augmented images that the network struggles with due to the challenging illumination conditions and poses caused by off-nominal pointing, high inclinations and $x^W \approx 0$

ferent techniques. Furthermore, as previously discussed, the usage of a different pose solver, namely the CEPPnP could possibly improve the performance even more. Pasqualetto Cassinis et al. (2021b) demonstrated that the usage of the CEPPnP resulted in a more accurate mean performance and lower MAD, as the CEPPnP is less sensitive to increases within the distance along the optical axis and performs equally well on these different distances.

11.4. Trajectory simulations

The pipeline trained on the Bennu+ dataset is used to test its performance on the different trajectories. This pipeline not only showed very accurate performance on the clean images, but also on the corrupted images as shown in Table 11.7. The three different trajectories that have been generated were discussed in Section 7.6 and a montage of each trajectory is given in Figures 11.22 through 11.24. The results of the trained pipeline on these different trajectories are shown in Table 11.10 and Figure 11.21.

The following can be observed from Table 11.10 and Figure 11.21:

- The performance of the pipeline for all three trajectories is excellent, as it satisfies the accuracy requirement for all images (100%).

Table 11.10: The results of the pipeline on the different trajectories, where the 40 most confident detections have been sent to the EP*n*P pose solver

Trajectory	**Mean $\mathbf{E}_t$ [m]**	**Mean $\|\mathbf{E}_t\|$ [m]**	**Median $\mathbf{E}_t$ [m]**	**Median $\|\mathbf{E}_t\|$ [m]**	**Failure cases [#/%]**
1	$(1.996 \quad 0.612 \quad 23.35)$	23.55 ± 16.68	$(1.915 \quad 0.703 \quad 21.82)$	21.98	0/0
2	$(0.900 \quad 0.805 \quad 22.97)$	23.04 ± 15.70	$(0.883 \quad 0.734 \quad 18.30)$	18.47	0/0
3	$(2.000 \quad 0.710 \quad 42.47)$	42.67 ± 39.01	$(2.129 \quad 0.665 \quad 32.23)$	32.38	0/0

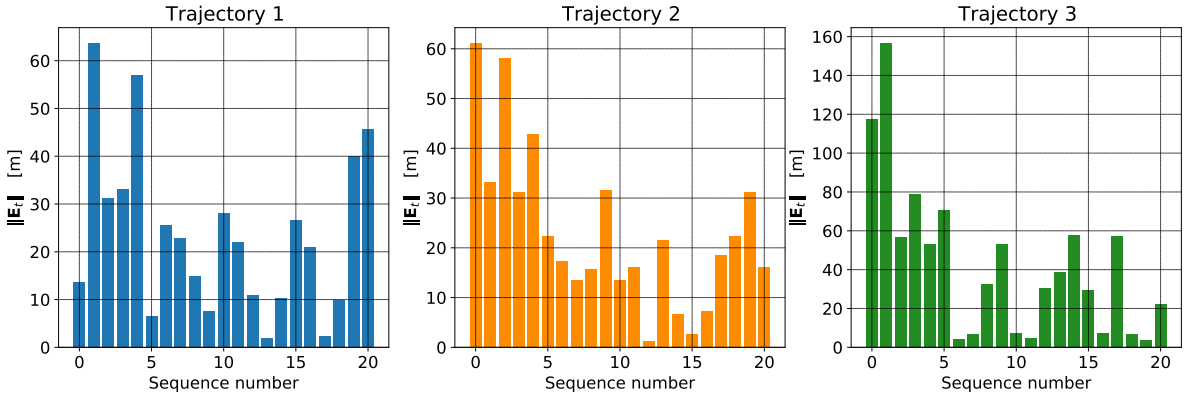


Figure 11.21: The distance estimation error displayed per image in each trajectory, for the different trajectories

- The pipeline is able to estimate the distance correctly within 0.3-0.5% of the actual distance, demonstrating that the CNN-based pipeline is able to very accurately determine the distance of the spacecraft w.r.t. the asteroid.
- The performance deviates between different images in the image sequence, however, it is always able to estimate the distance below 0.8-2.0% for the respective distances of 6 km to 8 km. Furthermore, when incorporating the CNN-based feature detector within a navigation architecture, the measurements are fed to the navigation filter alongside heatmap-derived covariances regarding the certainty of the feature detections. This reduces the effect of less accurate detections (varying measurements) on the state estimation.
- The performance of the pipeline for each trajectory is in line with the performance observed in Figure 11.13 for the respective distances, with a median between 20-30 m for a distance of 6-7.5 km and a median of around 45 m for a distance of 9 km. It was expected that the performance of the pipeline on the trajectory would fall around or below the median of Figure 11.13 for the respective distances, as the image sequences of the trajectory are not the most challenging images present within the dataset. The camera is *pointing nominally*, however, they do all have images in which only part of the asteroid is visible due to illumination conditions.
- The pipeline is able to generalize well to data it has not been explicitly trained on, which is demonstrated by the performance of the pipeline on the third trajectory, which has a distance of 8 km from the asteroid. The performance falls between the medians of 7.5-9 km, thereby illustrating that the network is able to interpolate properly. This allows the dataset to be relatively small and only contain a set of discrete distances while achieving accurate performance on the range of distances in between. This was the main reason of including an object detection network in the pipeline, as this makes the CNN-based feature extractor more robust against scaling of the target within the image.

Figure 11.22: Montage of trajectory 1 with a distance of 6 km to the asteroid using a subset of the images, where the first image is given in the lop left and the final image in the lower right

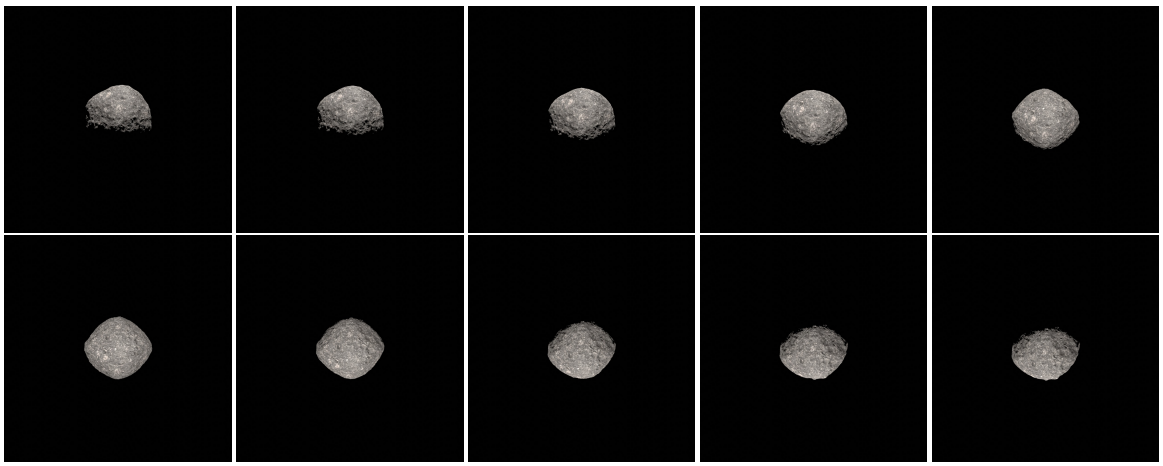


Figure 11.23: Montage of the polar orbit trajectory 2 with a distance of 7.5 km to the asteroid using a subset of the images, where the first image is given in the lop left and the final image in the lower right

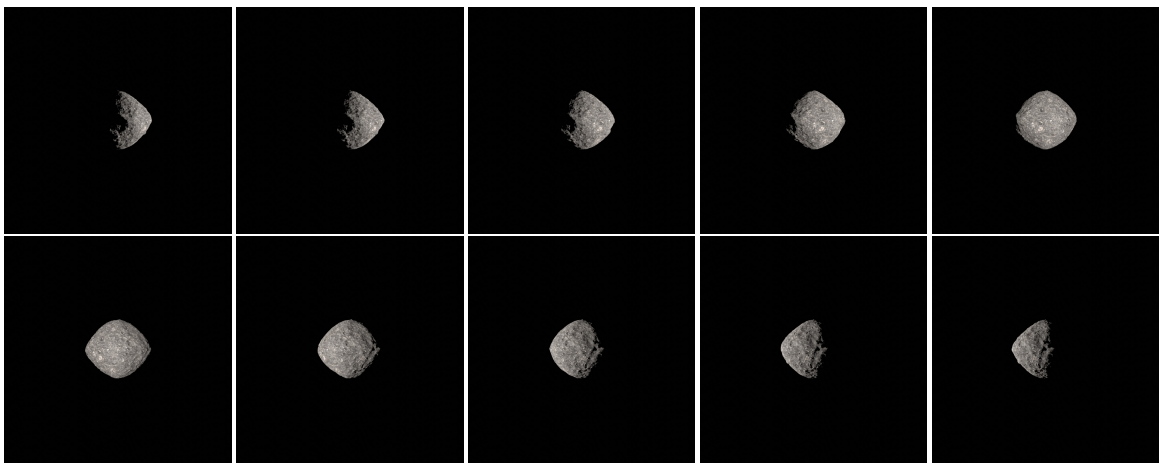


Figure 11.24: Montage of trajectory 3 with a distance of 8 km to the asteroid using a subset of the images, where the first image is given in the lop left and the final image in the lower right

11.5. Conclusions

This chapter thoroughly evaluated the performance that was achieved by the CNN-based pose estimation pipeline on the different datasets created within this work. The main take-aways are summarized below:

- The pipeline trained on the augmented dataset Bennu+ is able to achieve highly accurate performance on both the clean and augmented dataset with a mean *line-of-sight* distance estimation of around 42 m and 48 m, respectively, and a median distance estimation of around 30 m and 31 m, respectively, at a confidence level of 90%. This performance was achieved on the large relative range to the target of 4.5-9 km. The closer to the asteroid the more accurate the performance with a median error of around 22 m from a distance of 4.5 km. Furthermore, the developed system satisfied the accuracy requirement of $< 10\%$ knowledge error w.r.t. the ground-truth distance in 99.979% and 100% of the cases for the Bennu+ and Bennu dataset, respectively.

- This pipeline is able to accurately estimate the instantaneous position of the camera and does not depend on an initial pose estimate, making it suitable for *lost-in-space* scenarios. This CNN-based architecture can be used to accurately navigate at distances from 4.5 km to 9 km of the asteroid. Furthermore, as it provides an instantaneous position estimate, it can also be used to (re)-initialize an *unknown feature tracking* (relative navigation) approach that allows navigation from distances closer to the asteroid.

- The outliers have been extensively analyzed and the hypothesis is that the network can struggle with challenging poses that have high inclinations and challenging illumination conditions (at most half of asteroid illuminated) in combination with off-nominal pointing cases. These challenges are compounded when augmentations are applied to the images.

- The synthetically trained CNN-based feature extractor, consisting of the object and keypoint detection network, has proven to provide accurate distance estimates for a wide range of relative geometries between the camera and the asteroid. Moreover, it has proven to be robust against illumination conditions, occlusions, textures, and image corruptions. This is especially relevant for the application of the CNN to space applications, as the space environment is characterized by extreme contrast and low Signal-to-Noise ratio. Moreover, the pipeline has been shown to generalize well to data not explicitly present in the dataset as demonstrated by the 8 km trajectory. This emphasizes the advantage of the object detection network in the pipeline, as this makes the CNN-based feature extractor more robust against scaling of the target within the image.

- The employed training procedure and datasets are crucial in achieving this robustness and the results stress the importance of the data augmentations used in the Bennu+ dataset. The object and keypoint detection networks trained solely on clean images failed to generalize well to these corruptions, i.e., lacking the robustness required for safety-critical applications. Through the use of an augmented dataset, this robustness to illumination conditions, occlusions, textures, and image corruptions can be achieved with a minimal effort, as it is not required to model the exact surface textures, encountered illumination conditions, and sensors. Furthermore, this training procedure achieves the aforementioned highly accurate results without depending on the availability of accurate information regarding the target body's properties, which is often required for hand-engineered IP algorithms.

- The performance achieved by the CNN-based pipeline created in this work is similar to comparable pipelines created for the pose estimation of uncooperative spacecraft. Barad (2020) created a similar pipeline, consisting of an object and keypoint detection network in sequence, where the state-of-the-art HRNet-W32 (Table 11.4) has been used as the keypoint detection network. Furthermore, the Maximum Likelihood Perspective-n-Point (MLPnP) solver was used to incorporate uncertainty of the keypoint detection into the pose estimation process. This pipeline achieved a mean and median distance error estimate of 0.84 m and 0.47 m, respectively, on the clean dataset, and 1.38 m and 0.57 m, respectively, on the augmented Envisat dataset. This dataset consists of a relative range between the spacecraft and the target of 90-180 m. Furthermore, it achieved a mean distance error estimate of 0.14 m on a relative range of 3-30 m on the SPEED dataset. More detailed information regarding both datasets can be found in Appendix B. However, when adjusting for the differences in relative range between those datasets and the ones generated within this work, it can be concluded that the achieved performance is similar or better, i.e., $0.84/90 = 42/4500 = 0.0093$.

- The pipeline is able to achieve accurate results with the EPnP solver, however, the uncertainty associated with each individual keypoint detection is not taken into account. The performance of the pipeline is expected to increase even more when heatmap-derived covariance matrices associated with each keypoint detection are utilized. This can be achieved using the manner proposed by Pasqualetto Cassinis et al. (2020) and elaborated upon in Appendix A. By using the CEPPnP solver, which can take into account the detection uncertainty through the covariance matrix, the accuracy of the CNN-based pipeline could possibly be improved.
- Before this pipeline can be used on an actual spacecraft, the performance of the synthetically trained CNNs against real images needs to be validated, i.e., bridging the domain gap. However, due to the unavailability of large datasets consisting of actual space imagery of the target, the performance needs to be validated on-ground, using lab generated real images of a mock-up of the asteroid.
- The pipeline developed within in this work needs to be tested within a larger navigation framework, consisting of the developed CNN-based feature detector and a state estimator in sequence. The performance of navigation architecture, i.e., the CNN-based feature and state estimator, needs to be thoroughly evaluated using more advanced trajectories.
- The CNN-based pipeline developed in this work achieved great performance, while using only a fraction of the parameters and FLOPs of other state-of-the-art deep learning networks and pipelines, making it suitable for implementation on space hardware. However, before it can actually be used on hardware, the developed pipeline needs to be adapted/converted to allow for usage on embedded devices, which can be done using libraries such as TensorFlow Lite[2] or PyTorch[3]. This contains tools that apply optimizations, such as quantization, that are able to reduce the model size and latency with a minimal loss of accuracy. Furthermore, this allows the pipeline to be converted to a format that runs on C++ code, which is often the language of choice for production purposes.

[2] `https://www.tensorflow.org/lite/guide#1_generate_a_tensorflow_lite_model`, Date accessed: 25-01-2022
[3] `https://pytorch.org/tutorials/advanced/cpp_export.html`, Date accessed: 25-01-2022

V

Conclusion & Recommendations

12

Conclusions and recommendations

This chapter discusses the main conclusions of this work and recommendations for further research. Firstly, in Section 12.1, the main conclusions are discussed and the answers to the research question and respective subquestions are given. Secondly, in Section 12.2, the recommendations that allow for further development of the CNN-based feature detector pipeline are discussed to allow for robust navigation around asteroids using machine learning.

12.1. Conclusions

The research question that was formulated in Section 1.3 is restated:

How can accurate relative navigation be achieved in close-proximity operations around asteroids with limited a-priori information using learning-based autonomous algorithms?

This work developed a first-of-a-kind CNN-based pose estimation pipeline suitable for autonomous navigation around asteroids. The choice of a top-down feature based approach, consisting of an object detection and keypoint detection network in sequence, with a pose solver proved to produce accurate results. This means that a CNN is used to detect pre-defined keypoints on the 3D model within a 2D image after which the 2D-3D correspondence can be used to estimate the distance to the asteroid by solving the P*n*P problem. This method only relies on the availability of a 3D model of the target and can provide accurate *line-of-sight* distance estimates for a range of 4.5 km to 9 km from the asteroid. Furthermore, this pipeline is able to produce an estimate of the instantaneous position (pose initialization) and is as such useful for *lost-in-space* scenarios. This demonstrates the efficacy of using CNN-based architectures for navigation around small bodies and serves as a stepping stone for future research within this topic. The created and adapted software as well as the datasets used within this work will be made publicly available to foster research within this field, as it is expected to benefit the community as a whole.

The answers to the subquestions established in Section 1.3 are given below:

Research sub-question 1: *How can a representative, realistic dataset of synthetic images of the asteroid suitable for training and evaluating deep-learning networks be created?*

- Due to the unavailability of deep-learning datasets suitable for the training and evaluation of CNN-based systems, synthetic datasets had to be generated within this work. A model agnostic image generation and annotation pipeline was created using Blender and Python, which is suitable for generating deep learning datasets. This demonstrated the efficacy of Blender for large-scale dataset generation and this pipeline can be used for other targets as well. The generation of a first-of-a-kind publicly available dataset for vision-based navigation around asteroids, suitable for training and testing deep learning networks, was realized in this work. This dataset consists of 32,252 images containing a variety of camera viewpoints and distances, and realistic illumination conditions to ensure that the pipeline works for a variety of scenarios representative of an actual space mission. Moreover, image sequences corresponding to three different trajectories have been created to be able to demonstrate the efficacy of the developed algorithm to scenarios it could encountered in-orbit.

Research sub-question 2: *How can the pose estimation pipeline be made robust against a variety of factors, such as illumination conditions and image corruptions, representative of the real space environment?*

The pose estimation pipeline has been proven to produce accurate results for a variety of different camera viewpoints and distances and has shown to be robust against illumination conditions, occlusions, textures, and image corruptions. Due to the unavailability of a dataset consisting of real images of the target properly validating the performance of the synthetically trained CNN to real images was not possible within this work. However, the network's robustness against image corruptions representative of real image artifacts was researched and evaluated. The robustness of the synthetically trained CNNs has been achieved in two ways.

- **Robustness through the selection of the architecture**: The usage of an object detection network in the pipeline, making the CNN-based feature extractor more robust against scaling of the target within the image. Moreover, by using the multi-scale SSD-MobileNetV2-FPN-Lite object detection network the pipeline is more robust against image corruptions as was proven by Hendrycks and Dietterich (2019). Furthermore, by selecting a keypoint detection network that outputs heatmap predictions, the pipeline is more robust to the domain gap as proven by Park et al. (2021).
- **Robustness through training procedure and datasets**: The training procedure and datasets were found to be crucial in achieving robustness against illumination conditions, occlusions, textures and image corruptions. An augmented dataset Bennu+, consisting of a variety of image corruptions, has been generated within this work and used to train the pipeline. By employing this training strategy of using augmented datasets, this robustness is achieved with minimal effort without the need to model the exact surface textures, encountered illumination conditions, and sensors. The approach proposed in this work achieves highly accurate results without relying on accurate information regarding the target body's properties as required for hand-engineered IP algorithms. This demonstrates that the training procedure plays an important part in bridging the *domain gap* persisting between synthetic and real images.

Research sub-question 3: *How can the pose estimation pipeline be improved compared to comparable pipelines currently researched for satellites?*

- **Lightweight networks can achieve accurate performance at a fraction of the required memory and computational costs**: A major improvement of the developed CNN-based pipeline compared to other pipelines, which were designed for pose estimation of uncooperative spacecraft, is the usage of lightweight networks. The CNN-based pipeline has only a fraction of the parameters and FLOPs compared to other state-of-the-art deep-learning networks and pipelines. This is a crucial contribution of this work, showcasing that accurate and robust performance can be achieved using lightweight networks suitable for embedded devices. Furthermore, it was shown that the lightweight LPN model outperformed the current state-of-the-art keypoint detection network HRNet on the Bennu datasets, while having 81% less parameters and 80% less FLOPs.

12.2. Recommendations

This section discusses the recommendations for further research. The recommendations are split into four areas, namely pose solver, bridging the domain gap between synthetic and real images, navigation architecture, and the application of the developed pipeline. This also serves as a suggested order of importance for future research.

The following recommendation revolves around possibly increasing the performance of the pose estimation using the detected keypoints.

- **CEPP*n*P solver**: The pose estimation pipeline currently uses the EP*n*P solver alongside a heuristic approach that feeds the n most confident detections to the pose solver. However, the performance of the resulting pose estimate could possibly be improved by using the CEPP*n*P pose solver, which as discussed in Appendix A, can take into account the uncertainty of each detection through a covariance matrix that can be derived from the heatmap associated with the detection. This approach and pose solver allows the individual assessment of each keypoint per image, whereas the currently used heuristic approach sets a general value for all images, i.e., the n most confident keypoint detections. This is especially relevant for challenging images in which the keypoint detections are less accurate.

Pasqualetto Cassinis et al. (2021b) demonstrated that the distance estimation was improved using the CEPP*n*P solver compared to the EP*n*P solver, i.e., more accurate mean and lower MAD. Furthermore, the CEPP*n*P solver is less sensitive to the increase in distance along the optical axis and produces similar performance for all distances.

The following recommendations are with regards to validating the performance against real images and adjusting the training/pipeline to bridge the domain gap.

- **On-ground validation (HIL)**: The CNN-based feature detector has been trained on synthetic imagery and a robustness evaluation to image corruptions has been performed. However, to properly address the domain gap that exists between synthetic and real images and validate the performance of the CNNs to real images, on-ground validation is required. As addressed in Section 7.5, using actual space imagery of the target is not possible due to the unavailability of images with varying conditions (poses and illumination). Therefore, the robustness of the synthetically trained CNNs against real images has to be tested using real camera images generated in a laboratory environment using a mock-up model of the target asteroid, mimicking space-like illumination conditions.
- **Adjusting the training procedure to increase robustness**: This recommendation is part of the *on-ground validation* and refers to the usage of algorithms during training to bridge the domain gap as discussed in Section 7.5. The unsupervised domain adaptation as well as domain randomization can be employed to evaluate whether this improves the performance of the synthetically trained CNNs against the lab-generated images. The usage of these algorithms was not required within this work as no dataset of real camera images was available.

The following recommendations are related to testing the CNN-based feature detector within a navigation architecture:

- **Incorporation in a navigation architecture**: The CNN-based feature detector has been developed and tested within this work alongside a pose solver and was shown to produce accurate estimates of the *line-of-sight* distance to the asteroid. However, by incorporating it in a navigation architecture, the full state of the spacecraft including the relative translational and rotational velocities can be estimated. This navigation architecture can be a *tightly* or *loosely* coupled approach as the CNN-based feature detector is suitable for both due to the heatmaps that are outputted by the network for the predictions, allowing for the creation of a covariance matrix representing the uncertainty of detection.
- **More advanced trajectories**: The trajectories used in this work are simple image sequences without consideration of the velocity of the spacecraft, sampling time between images, and the rotation of the asteroid. Moreover, the camera is pointing towards the center of mass of the asteroid and no corruptions are added to the image. More challenging image sequences are required to properly evaluate the entire navigation architecture, i.e., the CNN-based feature detector and state estimator.

The following recommendation refers to the incorporation of the developed algorithm within a larger navigation framework.

- **Incorporation into an entire navigation scheme**: As previously addressed in Section 2.3, the CNN-based pipeline can be used to (re-)initialize an *unknown feature tracking* approach that can navigate from distances closer to the asteroid. The developed pipeline within this work could be one link in the chain of an entire navigation scheme that allows for accurate autonomous navigation around asteroids. Future works could explore this and allow for seamless transitions between the different navigation approaches while achieving desired performances.

Bibliography

Alimo, R., Jeong, D., and Man, K., "Explainable Non-Cooperative Spacecraft Pose Estimation using Convolutional Neural Networks," *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics, Reston, Virginia, 2020.

Barad, K. R., *Robust Navigation Framework for Proximity Operations around Uncooperative Spacecraft (MSc Thesis)*, Delft University of Technology, 2020.

Black, K., Shankar, S., Fonseka, D., Deutsch, J., Dhir, A., and Akella, M. R., "Real-Time, Flight-Ready, Non-Cooperative Spacecraft Pose Estimation Using Monocular Imagery," *arXiv preprint arXiv:2101.09553*, 2021.

Brochard, R., Lebreton, J., Robin, C., Kanani, K., Jonniaux, G., Masson, A., Despré, N., and Berjaoui, A., "Scientific image rendering for space scenes with the SurRender software," *Conference: 69th International Astronautical Congress (IAC)*, 2018.

Bulat, A., and Tzimiropoulos, G., "Binarized Convolutional Landmark Localizers for Human Pose Estimation and Face Alignment with Limited Resources," *2017 IEEE International Conference on Computer Vision (ICCV)*, Institute of Electrical and Electronics Engineers (IEEE), Venice, Italy, 2017, pp. 3726–3734.

Chen, Y., Wang, Z., Peng, Y., Zhang, Z., Yu, G., and Sun, J., "Cascaded Pyramid Network for Multi-person Pose Estimation," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), 2018, pp. 1703–1712.

Chen, B., Parra, , Cao, J., Parra, A., and Chin, T.-J., "Satellite Pose Estimation with Deep Landmark Regression and Nonlinear Pose Refinement," *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Seoul, South Korea, 2019, pp. 2816–2824.

Chen, C., Wang, B., Lu, C. X., Trigoni, N., and Markham, A., "A Survey on Deep Learning for Localization and Mapping: Towards the Age of Spatial Machine Intelligence," *arXiv preprint arXiv:2006.12567*, 2020.

Cheng, Y., Johnson, A., Matthies, L., and Olson, C., "Optical Landmark Detection for Spacecraft Navigation," *Proceedings of the 13th Annual AAS/AIAA Space Flight Mechanics Meeting*, 2002.

Cheng, L., Wang, Z., Song, Y., and Jiang, F., "Real-Time Optimal Control for Irregular Asteroid Landings Using Deep Neural Networks," *arXiv preprint arXiv:1901.02210*, 2019.

Cybenko, G., "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, Vol. 2, No. 4, 1989, pp. 303–314.

D'Amico, S., Benn, M., and Jørgensen, J. L., "Pose Estimation of an Uncooperative Spacecraft from Actual Space Imagery," *International Journal of Space Science and Engineering*, Vol. 2, No. 2, 2014, pp. 171–189.

Dementhon, D. F., and Davis, L. S., "Model-based object pose in 25 lines of code," *International Journal of Computer Vision*, Vol. 15, No. 1-2, 1995, pp. 123–141.

Deng, J., Dong, W., Socher, R., Li, L.-J., Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2009, pp. 248–255.

DeTone, D., Malisiewicz, T., and Rabinovich, A., "SuperPoint: Self-Supervised Interest Point Detection and Description," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, The Institute of Electrical and Electronics Engineers (IEEE), Salt Lake City, UT, 2018, pp. 337–33712.

Doppenberg, W., *Autonomous Lunar Orbit Navigation With Ellipse R-CNN (MSc thesis)*, Delft University of Technology, Delft, 2021.

Downes, L., Steiner, T. J., and How, J. P., "Deep Learning Crater Detection for Lunar Terrain Relative Navigation," *AIAA Scitech 2020 Forum*, AIAA, Orlando, FL, 2020a.

Downes, L. M., Steiner, T. J., and How, J. P., "Lunar Terrain Relative Navigation Using a Convolutional Neural Network for Visual Crater Detection," *2020 American Control Conference (ACC)*, Institute of Electrical and Electronics Engineers (IEEE), 2020b, pp. 4448–4453.

Ferraz, L., Binefa, X., and Moreno-Noguer, F., "Leveraging Feature Uncertainty in the PnP Problem," *Proceedings of the British Machine Vision Conference*, British Machine Vision Association, Nottingham, UK, 2014, pp. 1–13.

Flandin, G., Polle, B., Lheritier, J., and Vidal, P., "Vision based navigation for autonomous space exploration," *2010 NASA/ESA Conference on Adaptive Hardware and Systems*, 2010, pp. 9–16.

Furfaro, R., Bloise, I., Orlandelli, M., Lizia, P. D., Topputo, F., and Linares, R., "Deep Learning for Autonomous Lunar Landing," *AAS/AIAA Astrodynamics Specialist Conference*, 2018.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V., "Domain-Adversarial Training of Neural Networks," *The Journal of Machine Learning Research*, Vol. 17, No. 1, 2016, p. 2096–2030.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W., "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness," *arXiv preprint arXiv:1811.12231*, 2018.

Gerth, I., *Convex Optimization For Constrained and Unified Landing Guidance (MSc Thesis)*, Delft University of Technology, Delft, the Netherlands, 2014.

Gil-Fernandez, J., and Ortega-Hernando, G., "Autonomous vision-based navigation for proximity operations around binary asteroids," *CEAS Space Journal*, Vol. 10, No. 2, 2018, pp. 287–294.

Gil-Fernandez, J., Casasco, M., Carnelli, I., Martino, P., and Küppers, M., "HERA autonomous Guidance, Navigation and Control experiments: enabling better asteroid science & future missions," 2019.

Girshick, R., Donahue, J., Darrell, T., and Malik, J., "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 38, No. 1, 2016, pp. 142–158.

Girshick, R., "Fast R-CNN," *arXiv preprint arXiv:1504.08083*, 2015.

Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, MIT Press, 2016.

Harvard, A., Capuano, V., Shao, E. Y., and Chung, S.-J., "Spacecraft Pose Estimation from Monocular Images Using Neural Network Based Keypoints and Visibility Maps," *AIAA Scitech 2020 Forum*, AIAA, 2020.

Hashimoto, T., Kubota, T., Kawaguchi, J., Uo, M., Shirakawa, K., Kominato, T., and Morita, H., "Vision-based guidance, navigation, and control of Hayabusa spacecraft - Lessons learned from real operation -," *IFAC Proceedings Volumes*, Vol. 43, No. 15, 2010, pp. 259–264.

He, K., Zhang, X., Ren, S., and Sun, J., "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

He, K., Gkioxari, G., Dollar, P., and Girshick, R., "Mask R-CNN," *Proceedings of the IEEE International Conference on Computer Vision*, Vol. 2017-October, Institute of Electrical and Electronics Engineers Inc., 2017, pp. 2980–2988.

Hendrycks, D., and Dietterich, T. G., "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

Hinterstoisser, S., Benhimane, S., Lepetit, V., Fua, P., and Navab, N., "Simultaneous Recognition and Homography Extraction of Local Patches with a Simple Linear Classifier," British Machine Vision Association, 2008, pp. 10.1–10.10.

Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N., "Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes," , 2013.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

Huang, X., and Belongie, S. J., "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization," *arXiv preprint arXiv:1703.06868*, 2017.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., and Murphy, K., "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Honolulu, HI, 2017, pp. 3296–3297.

Izzo, D., Märtens, M., and Pan, B., "A Survey on Artificial Intelligence Trends in Spacecraft Guidance Dynamics and Control," *Astrodynamics*, Vol. 3, No. 4, 2019a, pp. 287–299.

Izzo, D., Sprague, C. I., and Tailor, D. V., "Machine Learning and Evolutionary Techniques in Interplanetary Trajectory Design," *Modeling and Optimization in Space Engineering*, Springer Optimization and Its Applications, Vol. 144, edited by G. Fasano and J. Pintér, Springer, 2019b, pp. 191–210.

Jackson, P. T., Atapour-Abarghouei, A., Bonner, S., Breckon, T. P., and Obara, B., "Style Augmentation: Data Augmentation via Style Randomization," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019, pp. 83–92.

Jiao, Z., Liu, R., Yi, P., and Zhou, D., "A Point Cloud Registration Algorithm Based on 3D-SIFT," 2019, pp. 24–31.

Jin, S., Ma, X., Han, Z., Wu, Y., Yang, W., Liu, W., Qian, C., and Ouyang, W., "Towards Multi-Person Pose Tracking : Bottom-up and Top-down Methods," *ICCV PoseTrack Workshop*, Vol. 2, 2017, p. 7.

Kingma, D. P., and Lei Ba, J., "Adam: A Method For Stochastic Optimization," *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

Kisantal, M., Sharma, S., Park, T. H., Izzo, D., Martens, M., and D Amico, S., "Satellite Pose Estimation Challenge: Dataset, Competition Design and Results," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 56, No. 5, 2020, pp. 4083–4098.

Krizhevsky, A., Sutskever, I., and Hinton, G. E., "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems*, Vol. 25, No. 2, 2012, pp. 1097–1105.

Kuipers, J. B., *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*, Princeton University Press, 2002.

Lauretta, D. S., DellaGiustina, D. N., Bennett, C. A., Golish, D. R., Becker, K. J., Balram-Knutson, S. S., Barnouin, O. S., Becker, T. L., Bottke, W. F., Boynton, W. V., Campins, H., Clark, B. E., Connolly, H. C., Drouet d'Aubigny, C. Y., Dworkin, J. P., Emery, J. P., Enos, H. L., Hamilton, V. E., Hergenrother, C. W., Howell, E. S., Izawa, M. R. M., Kaplan, H. H., Nolan, M. C., Rizk, B., Roper, H. L., Scheeres, D. J., Smith, P. H., Walsh, K. J., and Wolner, C. W. V., "The unexpected surface of asteroid (101955) Bennu," *Nature*, Vol. 568, No. 7750, 2019, pp. 55–60.

LeCun, Y., and Bengio, Y., *Convolutional Networks for Images, Speech, and Time Series*, MIT Press, Cambridge, MA, USA, 1998, p. 255–258.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D., "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, Vol. 1, No. 4, 1989, pp. 541–551.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y., "Unsupervised learning of hierarchical representations with convolutional deep belief networks," *Communications of the ACM*, Vol. 54, No. 10, 2011, pp. 95–103.

Lepetit, V., Moreno-Noguer, F., and Fua, P., "EPnP: An Accurate O(n) Solution to the PnP Problem," *International Journal of Computer Vision*, Vol. 81, No. 2, 2009, pp. 155–166.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L., "Microsoft COCO: Common Objects in Context," *European Conference on Computer Vision (ECCV)*, Vol. 8693, Springer, 2014, pp. 740–755.

Lin, Y., Tsung, Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S., "Feature Pyramid Networks for Object Detection," *arXiv preprint arXiv:1612.03144*, 2017.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P., "Focal Loss for Dense Object Detection," *arXiv preprint arXiv:1708.02002*, 2018.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C., "SSD: Single Shot MultiBox Detector," *arXiv preprint arXiv:1512.02325*, 2016.

Lorenz, D. A., Olds, R., May, A., Mario, C., Perry, M. E., Palmer, E. E., and Daly, M., "Lessons learned from OSIRIS-REx autonomous navigation using natural feature tracking," *2017 IEEE Aerospace Conference*, Institute of Electrical and Electronics Engineers (IEEE), Big Sky, MT, 2017, pp. 1–12.

Loshchilov, I., and Hutter, F., "SGDR: Stochastic Gradient Descent with Warm Restarts," *arXiv preprint arXiv:1608.03983*, 2017.

Lowe, D. G., "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, Vol. 60, No. 2, 2004, pp. 91–110.

Lu, C.-P., Hager, G., and Mjolsness, E., "Fast and globally convergent pose estimation from video images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 6, 2000, pp. 610–622.

Luvizon, D. C., Tabia, H., and Picard, D., "Human pose regression by combining indirect part detection and contextual information," *Computers & Graphics*, Vol. 85, 2019, pp. 15–22.

Maass, B., Woicke, S., Oliveira, W. M., Razgus, B., and Krüger, H., "Crater Navigation System for Autonomous Precision Landing on the Moon," *Journal of Guidance, Control, and Dynamics*, Vol. 43, No. 8, 2020, pp. 1414–1431.

Magalhães Oliveira, A. B., *Feature-based Optical Navigation for Lunar Landings (MSc thesis)*, Delft University of Technology, Delft, 2018.

Manning, J., Langerman, D., Ramesh, B., Gretok, E., Wilson, C., George, A., Mackinnon, J., and Crum, G., "Machine-Learning Space Applications on SmallSat Platforms with TensorFlow," *Small Satellite Conference*, 2018.

Mastrodemos, N., Rush, B., Vaughan, A., and Owen, W., "Optical Navigation For The Dawn Mission At Vesta," *Advances in the Astronautical Sciences*, Vol. 140, 2011, pp. 1739–1754.

Mooij, E., *Re-entry Systems: Lecture Notes (2019-2020)*, Delft University of Technology, 2019.

Movshovitz-Attias, Y., Kanade, T., and Sheikh, Y., "How useful is photo-realistic rendering for visual learning?" *arXiv*, Vol. arXiv:1603.08152, 2016.

Newell, A., Yang, K., and Deng, J., "Stacked Hourglass Networks for Human Pose Estimation," *Computer Vision - ECCV 2016. Lecture Notes in Computer Science*, Vol. 9912, edited by B. Leibe, J. Matas, N. Sebe, and M. Welling, Springer, 2016, pp. 483–499.

Oberkampf, D., DeMenthon, D. F., and Davis, L. S., "Iterative Pose Estimation Using Coplanar Feature Points," *Computer Vision and Image Understanding*, Vol. 63, No. 3, 1996, pp. 495–511.

Ogawa, N., Terui, F., Mimasu, Y., Yoshikawa, K., Ono, G., Yasuda, S., Matsushima, K., Masuda, T., Hihara, H., Sano, J., Matsuhisa, T., Danno, S., Yamada, M., Yokota, Y., Takei, Y., Saiki, T., and Tsuda, Y., "Image-based autonomous navigation of Hayabusa2 using artificial landmarks: The design and brief in-flight results of the first landing on asteroid Ryugu," *Astrodynamics*, Vol. 4, No. 2, 2020, pp. 89–103.

Opromolla, R., Fasano, G., Rufino, G., and Grassi, M., "A review of cooperative and uncooperative spacecraft pose determination techniques for close-proximity operations," *Progress in Aerospace Sciences*, Vol. 93, 2017, pp. 53–72.

Park, T. H., Sharma, S., and D'Amico, S., "Towards Robust Learning-Based Pose Estimation of Noncooperative Spacecraft," *2019 AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society (AAS), 2019.

Park, T. H., Märtens, M., Lecuyer, G., Izzo, D., and D'Amico, S., "SPEED+: Next Generation Dataset for Spacecraft Pose Estimation across Domain Gap," *arXiv preprint arXiv:2110.03101*, 2021.

Pasqualetto Cassinis, L., Fonod, R., and Gill, E., "Review of the robustness and applicability of monocular pose estimation systems for relative navigation with an uncooperative spacecraft," *Progress in Aerospace Sciences*, Vol. 110, 2019, p. 100548.

Pasqualetto Cassinis, L., Fonod, R., Gill, E., Ahrns, I., and Gil Fernandez, J., "CNN-Based Pose Estimation System for Close-Proximity Operations Around Uncooperative Spacecraft," *AIAA Scitech 2020 Forum*, AIAA, Orlando, FL, 2020.

Pasqualetto Cassinis, L., Fonod, R., Gill, E., Ahrns, I., and Gil-Fernández, J., "Evaluation of tightly- and loosely-coupled approaches in CNN-based pose estimation systems for uncooperative spacecraft," *Acta Astronautica*, Vol. 182, 2021a, pp. 189–202.

Pasqualetto Cassinis, L., Menicucci, A., Gill, E., Ahrns, I., and Gil-Fernández, J., "On-Ground Validation of a CNN-based Monocular Pose Estimation System for Uncooperative Spacecraft," *8th European Conference on Space Debris*, 2021b.

Pavlakos, G., Zhou, X., Chan, A., Derpanis, K. G., and Daniilidis, K., "6-DoF Object Pose from Semantic Keypoints," *arXiv preprint arXiv:1703.04670*, 2017.

Pellacani, A., Graziano, M., Fittock, M., Gil, J., and Carnelli, I., "HERA vision based GNC and autonomy," *8th European Conference for Aeronautics and Space Sciences (EUCASS)*, Madrid, Spain, 2019.

Razgus, B., Mooij, E., and Choukroun, D., "Relative Navigation in Asteroid Missions Using Dual Quaternion Filtering," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 9, 2017, pp. 2151–2166.

Redmon, J., and Farhadi, A., "YOLOv3: An Incremental Improvement," *arXiv preprint arXiv:1804.02767*, 2018.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A., "You Only Look Once: Unified, Real-Time Object Detection," *arXiv preprint arXiv:1506.02640*, 2016.

Ren, S., He, K., Girshick, R., and Sun, J., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *arXiv preprint arXiv:1506.01497*, 2016.

Rowell, N., Dunstan, M. N., Parkes, S. M., Gil-Fernández, J., Huertas, I., and Salehi, S., "Autonomous visual recognition of known surface landmarks for optical navigation around asteroids," *The Aeronautical Journal*, Vol. 119, No. 1220, 2015, pp. 1193–1222.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

Schaub, H., and Junkins, J. L., *Analytical Mechanics of Space Systems, Fourth Edition*, American Institute of Aeronautics and Astronautics, Inc., Washington, DC, 2018.

Schwartz, S., Nallapu, R. T., Gankidi, P., Dektor, G., and Thangavelautham, J., "Navigating to small-bodies using small satellites," *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, IEEE, 2018, pp. 1277–1285.

Scovanner, P., Ali, S., and Shah, M., "A 3-Dimensional Sift Descriptor and Its Application to Action Recognition," *Proceedings of the 15th ACM International Conference on Multimedia*, Association for Computing Machinery, New York, NY, USA, 2007, p. 357–360.

Shalev-Shwartz, S., Shamir, O., and Shammah, S., "Failures of Gradient-Based Deep Learning," *Proceedings of the 34th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 70, PMLR, Sydney, Australia, 2017, pp. 3067–3075.

Sharma, S., and D'Amico, S., "Comparative assessment of techniques for initial pose estimation using monocular vision," *Acta Astronautica*, Vol. 123, 2016, pp. 435–445.

Sharma, S., and D'Amico, S., "Reduced-Dynamics Pose Estimation for Non-Cooperative Spacecraft Rendezvous using Monocular Vision," *40th Annual AAS Guidance and Control Conference*, Breckenridge, CO, 2017.

Sharma, S., and D'Amico, S., "Pose Estimation for Non-Cooperative Spacecraft Rendezvous Using Neural Networks," *29th AAS/AIAA Space Flight Mechanics Meeting*, Ka'anapali, Maui, HI, 2019.

Sharma, S., Ventura, J., and D'Amico, S., "Robust Model-Based Monocular Pose Initialization for Noncooperative Spacecraft Rendezvous," *Journal of Spacecraft and Rockets*, Vol. 55, No. 6, 2018, pp. 1414–1429.

Sharma, S., *Pose Estimation of Uncooperative Spacecraft Using Monocular Vision and Deep Learning (PhD Thesis)*, Stanford University, Palo Alto, CA, 2019.

Shi, J.-F., Ulrich, S., and Ruel, S., "CubeSat Simulation and Detection using Monocular Camera Images and Convolutional Neural Networks," *2018 AIAA Guidance, Navigation, and Control Conference*, AIAA, Grapevine, TX, 2018.

Sierks, H., Keller, H. U., Jaumann, R., Michalik, H., Behnke, T., Bubenhagen, F., Büttner, I., Carsenty, U., Christensen, U., Enge, R., Fiethe, B., Gutiérrez Marqués, P., Hartwig, H., Krüger, H., Kühne, W., Maue, T., Mottola, S., Nathues, A., Reiche, K. U., Richards, M. L., Roatsch, T., Schröder, S. E., Szemerey, I., and Tschentscher, M., "The Dawn framing camera," *Space Science Reviews*, Vol. 163, No. 1-4, 2011, pp. 263–327.

Simonyan, K., and Zisserman, A., "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

Soviany, P., and Ionescu, R. T., "Optimizing the Trade-off between Single-Stage and Two-Stage Object Detectors using Image Difficulty Prediction," 2018.

Sturm, P., "Pinhole Camera Model," *Computer Vision: A Reference Guide*, edited by K. Ikeuchi, Springer US, Boston, MA, 2014, pp. 610–613.

Sun, X., Xiao, B., Wei, F., Liang, S., and Wei, Y., "Integral human pose regression," *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 529–545.

Sun, K., Zhao, Y., Jiang, B., Cheng, T., Xiao, B., Liu, D., Mu, Y., Wang, X., Liu, W., and Wang, J., "High-Resolution Representations for Labeling Pixels and Regions," *arXiv preprint arXiv:1904.04514*, 2019.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z., "Rethinking the Inception Architecture for Computer Vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Institute of Electrical and Electronics Engineers (IEEE), Las Vegas, NV, 2016, pp. 2818–2826.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A., "Inception-v, Inception - ResNet and the Impact of Residual Connections on Learning," *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI Press, San Francisco, CA, 2017, pp. 4278–4284.

Tompson, J., Jain, A., LeCun, Y., and Bregler, C., "Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation," *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, MIT Press, Cambridge, MA, USA, 2014, pp. 1799–1807.

Toshev, A., and Szegedy, C., "DeepPose: Human pose estimation via deep neural networks," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), Columbus, OH, 2014, pp. 1653–1660.

Volpe, R., Sabatini, M., Palmerini, G. B., and Mora, D., "Testing and Validation of an Image-Based, Pose and Shape Reconstruction Algorithm for Didymos Mission," *Aerotecnica Missili & Spazio*, Vol. 99, No. 1, 2020, pp. 17–32.

Wie, B., *Space Vehicle Dynamics and Control*, AIAA Education Series, 1998.

Woicke, S., *Hazard Relative Navigation: Towards Safe Autonomous Planetary Landings in Unknown Hazardous Terrain (PhD Thesis)*, Delft University of Technology, Delft, the Netherlands, 2019.

Xiao, B., Wu, H., and Wei, Y., "Simple Baselines for Human Pose Estimation and Tracking," *Computer Vision - ECCV 2018*, 2018, pp. 472–487.

Zhang, Z., Tang, J., and Wu, G., "Simple and Lightweight Human Pose Estimation," *arXiv preprint arXiv:1911.10346*, 2019.

Zhang, F., Zhu, X., Dai, H., Ye, M., and Zhu, C., "Distribution-Aware Coordinate Representation for Human Pose Estimation," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 7091–7100.

Zhao, Z., Peng, G., Wang, H., Fang, H.-S., Li, C., and Lu, C., "Estimating 6D Pose From Localizing Designated Surface Keypoints," *arXiv preprint arXiv:1812.01387*, 2018.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A., "Learning Deep Features for Discriminative Localization," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Institute of Electrical and Electronics Engineers (IEEE), Las Vegas, NV, 2016, pp. 2921–2929.

A

Heatmap-derived covariance matrix

This appendix serves as a basic explanation of the covariance computation from the heatmap, quantifying the uncertainty of the detection. This can be used as a guideline for future research. The covariance computation from the heatmap consists of several steps as outlined by Pasqualetto Cassinis et al. (2020), which are enumerated below.

1. Thresholding
2. Weight allocation
3. Covariance computation

This process is performed for each keypoint (heatmap) present within an image and for all images. The entire process is graphically illustrated in Figure A.1, where each step is discussed below.

Thresholding is a type of image segmentation, which can be used to select certain regions of an image deemed relevant while ignoring other parts. This procedure is applied to the heatmap image outputted by the keypoint detection network to extract the heatmap's pixels. The *weight allocation* procedure refers to giving each heatmap pixel a normalized weight w_i based on the gray intensity at its location. This gives more weight to pixels that are close to the confidence peak and therefore brighter, and gives less weight to pixels that are far from the peak and therefore fainter. The weighted *covariance can be computed* using Equation (A.1), where $(\hat{u}, \hat{v})$ represents the predicted keypoint location estimated using the highest confidence value instead of the mean, which is used normally. This is so that a distribution around the peak can be found instead of around the heatmap's mean, which becomes even more relevant when dealing with asymmetric heatmaps where the peak does not coincide with the mean. Furthermore, w_i refers to the normalized weight based on the gray intensity at that location.

$$\mathrm{cov}(x, y) = \frac{1}{m} \sum_{i=1}^{m} w_i \, (u_i - \hat{u}) \, (v_i - \hat{v}) \tag{A.1}$$

The resulting covariance matrix is shown below, where the covariance matrix can only represent Gaussian (normal) distributions. The limitation of this is that asymmetric heatmaps as shown in Figure A.1 (far right) might overestimate the dispersion of the heatmap over some directions.

$$\mathbf{C}_i = \begin{pmatrix} \mathrm{cov}(x, x) & \mathrm{cov}(x, y) \\ \mathrm{cov}(y, x) & \mathrm{cov}(y, y) \end{pmatrix} \tag{A.2}$$

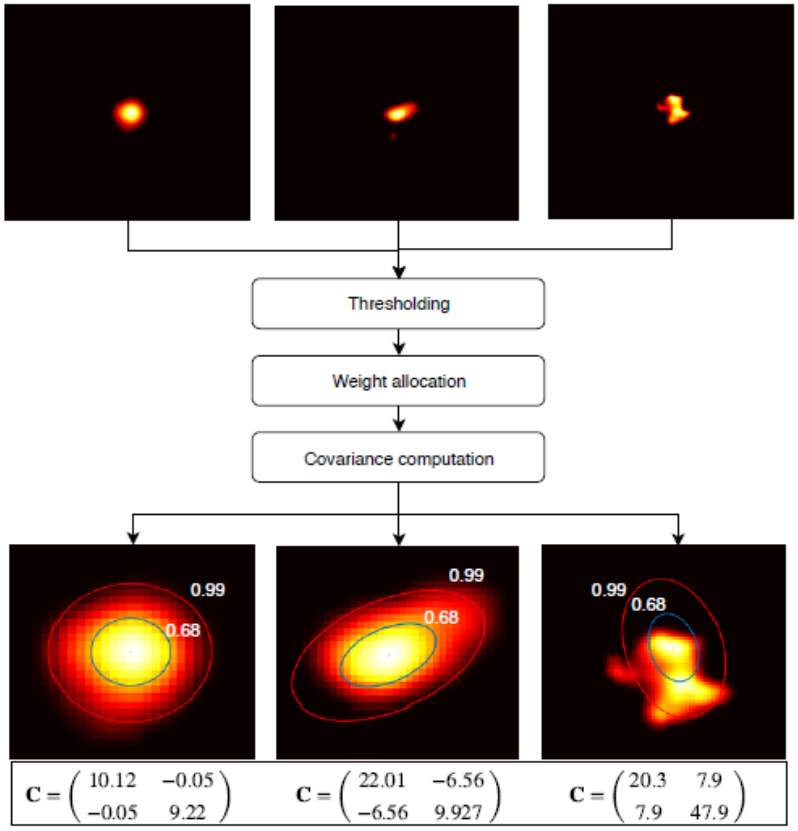


Figure A.1: Illustrating the process used to derive the covariance matrices from the heatmaps outputted by the keypoint detection network. The ellipses are determined using the computed covariances and the confidence intervals $\sigma = 0.68$ and $3\sigma = 0.99$ (Pasqualetto Cassinis et al., 2020)

B

Comparable deep-learning datasets

This appendix discusses the two datasets that have been used as a reference in generating the asteroid dataset. The SPEED dataset is created by Sharma and D'Amico (2019) at Stanford University and the Envisat dataset is created by Pasqualetto Cassinis et al. (2020) at the TU Delft. These deep-learning datasets are used for space-borne applications and are related to uncooperative spacecraft pose estimation. The SPEED dataset consists mainly of synthetic images and has a small number of real images that were generated in the TRON facility of SLAB using a realistic satellite mockup. The Tango spacecraft was used as the target object and photo-realistic images of this spacecraft with different poses were generated using OpenGL. Furthermore, random Earth images were used as the background for half of the images, where the illumination conditions were created to best match those background images. The Envisat dataset was generated in Cinema4D and consists of synthetic images of the Envisat spacecraft and of an image sequence that simulates close-proximity trajectories. An accurate texture model of ESA's Envisat was used. The properties of the datasets will be discussed, where the focus will be on the size, the viewpoints, the illumination conditions, and the applied data augmentation, which serve as a reference for the to-be created dataset.

- **Size**: The SPEED dataset consists of 15,000 synthetic images, which were randomly split 80/20 into training and test data. The Envisat dataset consists of 54,000 synthetic images, which were randomly split ≈ 60/20/20 into training, validation, and test data.
- **Viewpoints**: The SPEED dataset separately selects the attitude and the position. The relative attitudes are created using a unit quaternion parameterization of uniformly random rotations in the SO(3) space. The relative positions are determined by separately selecting the relative distance and the bearing angles. The bearing angles control the location of the spacecraft within the image (off-center). The relative distances are uniformly distributed between 3 m - 50 m.

  The Envisat dataset also separately selects the attitude and the position. The relative attitudes were created by discretizing the yaw, pitch, and roll (ψ, θ, φ) angles of the target w.r.t. the camera using 10° increments. The relative distance is discretized in the interval 90 m - 180 m using 30 m increments.
- **Illumination conditions**: The SPEED dataset uses azimuth and elevation angles of the Sun that specifically match the illumination conditions in the background images of the Earth. The Envisat dataset uses constant azimuth and elevation angles of 30° to create favorable as well as adverse illumination conditions.
- **Data augmentation**: The SPEED dataset applies Gaussian blurring and white noise to each image to emulate depth of field and shot noise, whereas the Envisat dataset has no data augmentations applied.

C

Dataset annotations format

Code Listing C.1: Dataset annotation file (`.json`)

```
[
   ...
   {
        "id": 63,
        "filename": "0_4.5_0095.jpg",
        "bbox": [227.05, 247.35, 797.95, 767.65],
        "keypoints": [
            {"ID": 0, "Coordinates": [311.8229960874263, 616.867447376349],
            "Visibility": 1},
            {"ID": 1, "Coordinates": [450.75387910520016, 699.4627574947973],
            "Visibility": 1},
            {"ID": 2, "Coordinates": [376.48776316752185, 668.2631389556271],
            "Visibility": 1},

                                        ...

            {"ID": 65, "Coordinates": [635.7146406522888, 467.74024244163127],
            "Visibility": 1},
            {"ID": 66, "Coordinates": [487.5215207126556, 709.8767655527533],
            "Visibility": 1
            }, {"ID": 67, "Coordinates": [475.99706669811314, 319.9303895058466],
            "Visibility": 1}
        ],
        "pose": {"r": [0.0, 0.0, 4.499999999999936],
            "q": [0.1522561833159995, -0.1522561833159995,-0.6905200737581759,
                0.6905201930581703]
        }
    },
    ...
]
```

Code Listing C.2: COCO format (`.json`)

```
{
    "info": {
        "description": "Bennu-1",
        "url": "https://tudelft.nl",
        "version": "1.0",
        "year": 2021,
        "contributor": "Lars van der Heijden",
        "date_created": "2021/08/01"
    },
    "licenses": {
        "id": 0,
        "url": "N/A",
        "name": "N/A"
    },
```

```
    "images": [
        {
            "license": 1,
            "file_name": "0_4.5_0009.jpg",
            "coco_url": "",
            "height": 1024,
            "width": 1024,
            "date_captured": "",
            "flickr_url": "",
            "id": 450009
        },
        {
            "license": 1,
            "file_name": "0_4.5_0013.jpg",
            "coco_url": "",
            "height": 1024,
            "width": 1024,
            "date_captured": "",
            "flickr_url": "",
            "id": 450013
        },
        ...
    ],
    "annotations": [
        {
            "segmentation": [],
            "num_keypoints": 68,
            "area": 282965.76,
            "iscrowd": 0,
            "keypoints": [
                308.6702660056609, 646.7661919182307, 2,
                457.48837130483, 630.2760834909972, 2,
                383.3427462194653, 610.6900328801723, 2,
                                ...
                619.5316958892159, 592.2540945871782, 2,
                481.7909390894226, 724.4413819715895, 2,
                469.1477189840984, 386.11767517215304,2
            ],
            "image_id": 450009,
            "bbox": [224.9, 261.6, 574.2, 492.79999999999995],
            "category_id": 1,
            "id": 1
        },
        {
            "segmentation": [],
            "num_keypoints": 68,
            "area": 298516.68000000005,
            "iscrowd": 0,
            "keypoints": [402.5270717868648, 675.321560029599, 2,
                417.42037021822216, 679.8817894317076, 2,
                364.3848113607935, 669.3804610892698, 2,
                                ...
                736.3252685426577, 508.0718045702962, 2,
                543.5773480292133, 725.5522610683839, 2,
                530.9154183571504, 356.07794462766645, 2
            ],
            "image_id": 450013,
            "bbox": [223.3, 255.9,587.4000000000001, 508.20000000000005],
            "category_id": 1,
            "id": 2
        },
        ...
    ],
    "categories": [
        {
            "supercategory": "Asteroid",
            "id": 1,
            "name": "Bennu",
            "keypoints": ["1","2","3","4","5","6","7","8","9","10","11","12","13","14",
                "15","16","17","18","19","20","21","22","23","24","25","26","27","28","
                29","30","31","32","33","34","35","36","37","38","39","40","41","42","
```

```
                43","44","45","46","47","48","49","50","51","52","53","54","55","56","
                57","58","59","60","61","62","63","64","65","66","67","68"]
        }
    ]
}
```

Code Listing C.3: Detections by OD network COCO format (`.json`)

```
[
    {
        "bbox": [
            306.3857421875,
            321.36151123046875,
            565.94970703125,
            523.72119140625
        ],
        "category_id": 1,
        "image_id": 450876,
        "score": 0.9999995231628418
    },


        ...


    {
        "bbox": [
            464.2561340332031,
            67.12730407714844,
            559.7438659667969,
            496.20960998535156
        ],
        "category_id": 1,
        "image_id": 2451095,
        "score": 0.9999922513961792
    }
]
```

Code Listing C.4: Example of the object detection configuration file (`.config`)

```
""" The configuration file for the SSD-MobileNetV2-FPNLite object network used to train
     the network on the Bennu+ dataset"""
model {
  ssd {
    num_classes: 1
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
```

```
        scale: true
        epsilon: 0.001
      }
    }
    use_depthwise: true
    override_base_feature_extractor_hyperparams: true
    fpn {
      min_level: 3
      max_level: 7
      additional_layer_depth: 128
    }
  }
  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 {
    }
  }
  box_predictor {
    weight_shared_convolutional_box_predictor {
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
      }
      depth: 128
      num_layers_before_predictor: 4
      kernel_size: 3
      class_prediction_bias_init: -4.6
      share_prediction_tower: true
      use_depthwise: true
    }
  }
  anchor_generator {
    multiscale_anchor_generator {
      min_level: 3
      max_level: 7
      anchor_scale: 4.0
      aspect_ratios: 1.0
      aspect_ratios: 1.1
      aspect_ratios: 1.2
```

```
        aspect_ratios: 0.7
        aspect_ratios: 0.8
        aspect_ratios: 0.9
        scales_per_octave: 2
      }
    }
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-08
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
        use_static_shapes: false
      }
      score_converter: SIGMOID
    }
    normalize_loss_by_num_matches: true
    loss {
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 32
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.08
          total_steps: 50000
          warmup_learning_rate: 0.026666
          warmup_steps: 1000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "/content/gdrive/My Drive/TensorFlow/workspace/training_demo/
      pre-trained-models/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt
      -0"
```

```
  num_steps: 50000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "/content/gdrive/My Drive/TensorFlow/workspace/training_demo/
      annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "/content/gdrive/My Drive/TensorFlow/workspace/training_demo/
        annotations/train+.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1
}
eval_input_reader {
  label_map_path: "/content/gdrive/My Drive/TensorFlow/workspace/training_demo/
      annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "/content/gdrive/My Drive/TensorFlow/workspace/training_demo/
        annotations/val+.record"
  }
}
```

Code Listing C.5: Example of the keypoint detection configuration file (`.yaml`)

```
AUTO_RESUME: true
CUDNN:
  BENCHMARK: true
  DETERMINISTIC: false
  ENABLED: true
DATA_DIR: ''
GPUS: (0,)
OUTPUT_DIR: '/content/gdrive/MyDrive/PyTorch/lpn-pytorch/output/Bennu+256x256Stage1'
LOG_DIR: '/content/gdrive/MyDrive/PyTorch/lpn-pytorch/log/Bennu+256x256Stage1'
WORKERS: 24
PRINT_FREQ: 100

DATASET:
  COLOR_RGB: false
  DATASET: 'bennu_coco'
  ROOT: '/content/gdrive/MyDrive/PyTorch/lpn-pytorch/data'
  TEST_SET: 'val'
  TRAIN_SET: 'train'
  FLIP: true
  ROT_FACTOR: 40
  SCALE_FACTOR: 0.3
MODEL:
  NAME: 'lpn'
  INIT_WEIGHTS: true
  PRETRAINED: '/content/gdrive/MyDrive/PyTorch/lpn-pytorch/output/Bennu+/bennu_coco/lpn
      /lpn101_256x256_Bennu+/model_best.pth'  #Using the best model obtained in
      previous stage to initialize the networks weights and biases
  IMAGE_SIZE:
  - 256
  - 256
  HEATMAP_SIZE:
  - 64
  - 64
  SIGMA: 2
  NUM_JOINTS: 68
  TARGET_TYPE: 'gaussian'
  EXTRA:
```

```
    FINAL_LAYER:
    - 'final_layer'
    ATTENTION: 'GC'
    FINAL_CONV_KERNEL: 1
    DECONV_WITH_BIAS: false
    NUM_DECONV_LAYERS: 2
    NUM_DECONV_FILTERS:
    - 256
    - 256
    NUM_DECONV_KERNELS:
    - 4
    - 4
    NUM_LAYERS: 101
LOSS:
  USE_TARGET_WEIGHT: true
TRAIN:
  BATCH_SIZE_PER_GPU: 64
  SHUFFLE: true
  BEGIN_EPOCH: 60
  END_EPOCH: 150
  OPTIMIZER: 'adam'
  LR: 0.001
  LR_FACTOR: 0.1
  LR_STEP:
  - 90
  - 120
  WD: 0.0001
  GAMMA1: 0.99
  GAMMA2: 0.0
  MOMENTUM: 0.9
  NESTEROV: false
TEST:
  BATCH_SIZE_PER_GPU: 1
  COCO_BBOX_FILE: '/content/gdrive/MyDrive/coco/od+_val+Detections.json'
  BBOX_THRE: 1.0
  IMAGE_THRE: 0.0
  IN_VIS_THRE: 0.2
  MODEL_FILE: '/content/gdrive/MyDrive/PyTorch/lpn-pytorch/output/Bennu+256x256Stage1/
      bennu_coco/lpn/lpn101_256x256-Bennu+-stage1/model_best.pth'
  NMS_THRE: 1.0
  OKS_THRE: 0.9
  FLIP_TEST: false
  POST_PROCESS: true
  SHIFT_HEATMAP: true
  USE_GT_BBOX: false
  SOFT_ARGMAX: true
DEBUG:
  DEBUG: true
  SAVE_BATCH_IMAGES_GT: false
  SAVE_BATCH_IMAGES_PRED: false
  SAVE_HEATMAPS_GT: false
  SAVE_HEATMAPS_PRED: false
  SAVE_HEATMAPS_TEST_ALL: false
```

Code Listing C.6: Example of the `dataset_processing.py` script used to process the raw data and create a fully annotated dataset

```
import sys
sys.path.insert(0, 'D:/MscSpaceflight/Thesis/thesisCode/datasetCreation/python/utils')
from dataset_utils import *
sys.path.insert(1, 'D:/MscSpaceflight/Thesis/thesisCode/machineLearning/
    keypointDetection/utils')
from convert2COCO import COCOAnnotation

### Process the raw dataset and create the annotations required for the subsequent ML
    networks and PnP solver

# Specify the paths for the raw data [images, files]
source_blender_img = r'D:\MscSpaceflight\Thesis\thesisCode\datasetCreation\images\
    Rendering\Bennu'
```

```
dest_blender_img = r'D:\MscSpaceflight\Thesis\thesisCode\datasetCreation\images\datasetBennu'
bb_root = r'D:\MscSpaceflight\Thesis\thesisCode\datasetCreation\dataFiles\boundingBox'

# The paths for the destination of the processed dataset
source_img = r'D:\MscSpaceflight\Thesis\thesisCode\datasetCreation\images\datasetBennu'
dest_img = r'D:\MscSpaceflight\Thesis\dataset_ML_total'

# The path for the pose used for the generation of the images
blenderPose =  r'D:\MscSpaceflight\Thesis\thesisCode\datasetCreation\dataFiles\blenderPose\used'

### Create the dataset class instance
data = processingDataset(source_blender_img,dest_blender_img, source_img, dest_img, bb_root, blenderPose)

### Partition the dataset
data.partition_dataset(train_ratio = 0.70, val_ratio = 0.15, val = True)

### Annotate the bounding box in the OD format
data.annotateBBox('train', relaxed_bb = True, margin = 5)
data.annotateBBox('val', relaxed_bb = True, margin = 5)
data.annotateBBox('test', relaxed_bb = True, margin = 5)

### Import the designated 3D keypoints on Bennu
path = r'D:\MscSpaceflight\Thesis\thesisCode\datasetCreation\dataFiles\keypoints\keypointsBennu.ply'
points_3D = plyUtil.read_ply(path)
points_3D = points_3D['points']
points_3D.drop(points_3D.tail(1).index,inplace=True)
kp_coordinates = points_3D.values.reshape(-1,3)

### Create and write the annotated json file
train_json, kp_list = data.annotatedJSONfile('train',kp_coordinates)
val_json, kp_list_val = data.annotatedJSONfile('val', kp_coordinates)
test_json, kp_list_test = data.annotatedJSONfile('test', kp_coordinates)

### Write to file
data.write2JSON('train',train_json)
data.write2JSON('val', val_json)
data.write2JSON('test', test_json)

### Create and write the annoted COCO format file

### Define the basic arguments
dataset_name= 'Bennu-1'
n_kps = 68
base_args = {
        "dataset":dataset_name,
        "size":(1024,1024),
        "url": "",
        "n_keypoints":n_kps,
        "skeleton": []
        }

### The training arguments
train_args = base_args
train_args["image_path"] = r'D:\MscSpaceflight\Thesis\dataset_ML_total\train'
train_args['datafile'] = r'D:\MscSpaceflight\Thesis\dataset_ML_total\train\train.json'
train_json_path = r'D:\MscSpaceflight\Thesis\dataset_ML_total\train\COCO_train.json'

train_COCO = COCOAnnotation(**train_args)
train_COCO.write2File(train_json_path)

### The validation arguments
val_args = base_args
val_args["image_path"] = r'D:\MscSpaceflight\Thesis\dataset_ML_total\val'
val_args['datafile'] = r'D:\MscSpaceflight\Thesis\dataset_ML_total\val\val.json'
val_json_path = r'D:\MscSpaceflight\Thesis\dataset_ML_total\val\COCO_val.json'

val_COCO = COCOAnnotation(**val_args)
```

```
val_COCO.write2File(val_json_path)

### The test arguments
test_args = base_args
test_args["image_path"] = r'D:\MscSpaceflight\Thesis\dataset_ML_total\test'
test_args['datafile'] = r'D:\MscSpaceflight\Thesis\dataset_ML_total\test\test.json'
test_json_path = r'D:\MscSpaceflight\Thesis\dataset_ML_total\test\COCO_test.json'

test_COCO = COCOAnnotation(**test_args)
test_COCO.write2File(test_json_path)

### Visualize the keypoints, bounding box and pose for an image
image_set = 'val'
img_name = '5_4.5_0527.jpg'
image = data.visualize_gt_kps(image_set,img_name=img_name)
```

D

Image corruptions specifications

This appendix specifies the settings used in the generation of the augmented dataset *Bennu+*. The ImageNet-C dataset created by Hendrycks and Dietterich (2019), is a benchmark dataset that was created to evaluate the robustness of neural nets. The dataset has applied 15 different corruptions types ranging from noise, blur, weather, and digital effects to mimic real camera/sensor and environment artifacts. The algorithms used to generate this dataset can be found in the Github repository[1] and have been used in this work. The approach proposed by Barad (2020) and the adapted software repository[2] from the original (Hendrycks and Dietterich, 2019) were used in the generation of the dataset. This repository uses algorithms from open-source libraries, such as Python's *skimage*, *OpenCV*, *scipy*, *Wand*, and PyTorch. Because these libraries are maintained and used by many users for different applications, it is assumed that the functions can be considered verified. The severity levels used in the generation of these corruptions are based on the values used by Hendrycks and Dietterich (2019) and Barad (2020) and are representative of real camera/sensor corruptions. Furthermore, the augmentation applied to each image is unique to mimic real world corruptions, which also show variation of the corruption values even at fixed levels of intensity.

- **Gaussian blur:** Gaussian blur is applied to the image by using a convolution operation whereby the pixels are multiplied with a Gaussian kernel. The severity specifies the standard deviation of the Gaussian kernel and is set to 1.
- **Motion blur:** Motion blur is applied to the image through the use of the Wand Library[3]. The severity specifies a tuple containing the radius of the aliasing disk and the standard deviation of the blur used for the disk, and is set to $(7,3)$.
- **Defocus blur:** Defocus blur is applied to the image by using a convolution operation whereby the pixels are multiplied by a created kernel that emulates an aliasing disk with Gaussian blur. The severity specifies a tuple containing the radius of the aliasing disk and the standard deviation of the blur used for the disk, and is set to $(3;0.1)$.
- **Zoom blur:** Zoom blur is applied to the image by using scipy's zoom function, which uses two zooming overlays. Firstly, the image is zoomed in by a large factor and secondly the image is zoomed out by a small zoom factor. The pixel values of the zoomed image are determined using a first order spline interpolation. The severity specifies a tuple containing the zoom factor for the two operations and is set to $(1.11;0.01)$
- **Spatter:** Spatter is applied to the image by simulating liquid droplets, however, the implementation is quite extensive and not trivial. Therefore, the reader is referred to the original implementation[4].
- **Gaussian noise:** Gaussian noise is applied to the image by adding pixel intensities drawn from a normal distribution to the normalized original pixel intensities. The severity specifies the standard deviation of the normal distribution and is set to 0.08.
- **Impulse noise:** Impulse noise is applied to the image by replacing normalized random pixels with 0 (dead pixels) or 1 (hot pixels). The severity specifies the proportion of the total pixels that are to be replaced and is set at 0.015.

[1] `https://github.com/hendrycks/robustness` Date accessed: 4-11-2021
[2] `https://github.com/kuldeepbrd1/image-corruptions` Date accessed: 4-11-2021
[3] `https://github.com/emcconville/wand`, Date accessed: 3-12-2021
[4] `https://github.com/hendrycks/robustness` Date accessed: 4-11-2021

- **Shot noise:** Shot noise is applied to the image by adding pixel intensities drawn from a Poisson distribution to the normalized original pixel intensities. The severity specifies the variance of the distribution from which the noise for the pixel is generated and is set at 60.
- **Speckle noise:** Speckle noise is applied to the image by multiplying pixel intensities drawn from a normal distribution (Gaussian) with the normalized original pixel intensities, and adding it to the normalized original pixel intensities. The severity specifies the standard deviation of the normal distribution and is set at 0.15.
- **Color jitter:** Color jitter is a data augmentation that is available through PyTorch. The severity specifies the range from which the brightness, contrast, and saturation values are uniformly chosen and is set at 0.5 for all parameters resulting in the following range $[0.5, 1.5]$.
- **Random erase**: Random erase is a data augmentation that is available through PyTorch. The severity specifies the scale of the erased area w.r.t. the input image and is set at 0.02, i.e., 2% of the input size.

Probabilities

The probabilities used in the generation of the training and validation set of *Bennu*+ as shown in Figure 7.18 are listed in Table D.1.

Table D.1: The probabilities used in the generation of the augmented images for the training and validation set

Probability	**Training**	**Validation**
P(C)	0.30	0.30
P(RE)	0.12	0.10
P(CJ)	0.45	0.50
P(B)	0.55	0.60
P(N)	0.45	0.40