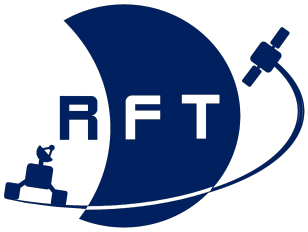


Technische Universität Berlin

Faculty V - Mechanical Engineering and Transport Systems

Bachelor Thesis

Vision-based Pose Estimation of a non-cooperative tumbling Object using Machine Learning

Author:

Dimitrij Schulz

November 27, 2023

Supervisor:

Prof. Dr.-Ing Enrico Stoll

Chair of Space Technology

Technische Universität Berlin

Assistant Supervisor:

Prof. Guillermo Gallego

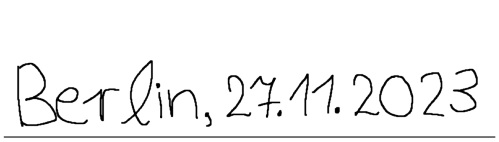
Chair of Electrical and Computer Engineering

Technische Universität Berlin

Schulz, Dimitrij:
Vision-based Pose Estimation of a non-cooperative tumbling Object using Machine Learning
Bachelor Thesis, Technische Universität Berlin, 2023.

Declaration

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

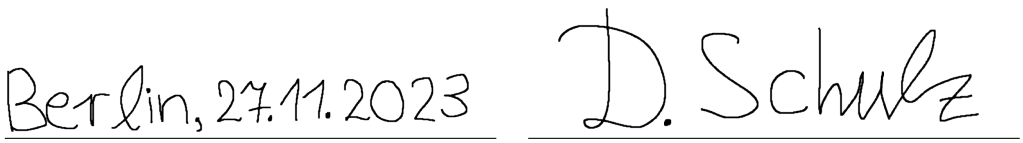


Place, Date

Signature

Agreement on utilization rights

According to § 60 of the AllgStuPO, students of the TU Berlin must complete a final thesis during their studies. The totality of the achievements associated with a final thesis are based not only on the written documentation and the commitment of the student, but also on the supervisory effort of the department, including the preparation of the assignment, the specific requirements, guidance of the students, help and advice in organizational, technical and scientific terms. Therefore, the student is the sole author of the written work, but not the sole author of the entirety of the services associated with a thesis.
By supervising the student, the Technische Universität Berlin, represented by the Chair of Astronautics, becomes co-author of the entirety of the achievements and the associated results according to § 8 section 1 of the German Copyright Act (Urheberrechtsgesetz).
For non-profit use in teaching and research, the Chair may use the results of the present work. As co-author, the chair is entitled to do so according to § 8 section 2 of the Copyright Act. If the work is written independently in such a way that there is no co-authorship, this right of use is granted by the student according to § 31 section 2 of the Copyright Act. This right of use is unrestricted and includes content of any kind (e.g. documentation, presentations, photos, videos, procedures, drafts, drawings, software including source code and the like) with naming of the author.
Due to the reference to ongoing research projects, publication without explicit permission of the department will not be permitted to the student. This is to be checked and approved by the department in each individual case.
Any commercial use of either site will only occur with the consent of all authors of the present work, with appropriate participation in the earnings.



Place, Date | Signature

Fakultät | Verkehrs- und Maschinensysteme
Institut für Luft- und Raumfahrttechnik

Fachgebiet Raumfahrttechnik

Fachgebietsleitung
Prof. Dr.-Ing. Enrico Stoll

Sekretariat F6 Raum F516
Marchstr. 12 - 14
10587 Berlin

Telefon +49 (0)30 314-21306
bettina.taube@tu-berlin.de

Markus Huwald

Telefon +49 (0)30 24277
m.huwald@tu-berlin.de

Task description for the bachelor's thesis
of Mr. Dimitrij Schulz
Matr.-Nr.: 366289

Subject:
Vision-based Pose Estimation of a non-cooperative tumbling Object using Machine Learning
Sichtbasierte Abschätzung der Pose eines nicht-kooperativen taumelnden Objekts mittels maschinellen Lernens

In the wake of ever-increasing amounts of space debris, the need for novel solutions to actively remove debris objects from orbit has become evident. Specifically in the lower and medium height orbital regions the debris population density poses a risk to the continued safe operation of space flight. Without active measures, a decrease of the debris population is no longer possible. The removal of large objects has been identified as a viable option to reduce the debris population and collision risk.

Active Debris Removal (ADR) however faces unique challenges uncommon to traditional close proximity operations. Debris objects are naturally non-cooperative, featuring no working communication or control systems and generally undergoing a tumbling motion. Relative navigation and target motion estimation must rely on sensors on-board of the active spacecraft, the so-called chaser, that do not require contact or communication with the target. This challenge is can only be fulfilled by vision-based sensors such as cameras or lidars. Determining the target motion state is vital to the subsequent tasks of motion planning, final approach, and capture. Such a system further must work autonomously as contact to the chaser typically cannot be always guaranteed.

Modern computer vision approaches rely on the use of machine learning. However, training and validating spaceborne models remain very challenging due to the limitations on large-scale datasets. This thesis aims at advancing the state-of-the-art vision-based estimation of a spacecraft state by numerically assessing the performance of ML models trained on synthetic data. Specifically, the target's pose or attitude is to be estimated from monocular imaging data. The resulting algorithms will be part of a larger AI-assisted motion state estimation system utilizing vision-based sensors and will be subsequently evaluated using Hardware-in-the-loop-simulation.

The following detailed tasks must be performed:

1. Literature research including the following topics:
    a. Pose estimation using artificial intelligence and machine learning methods and tools such as, but not limited to [1, 2]:
        - Perspective-n-Point.
        - Keypoint detection and matching.

        - YOLO, Pytorch.
        - Focus on feature-based approaches.
    b. Pose estimation using monocular vision-based sensors [3].
    c. Astrobee simulation environment and the Robot Operating System [4].
2. Review the RFSE 2021 submission to the Speed+ pose estimation challenge.
    a. Review the implemented concepts.
    b. Analyze issues with the implemented solution.
    c. Analyze failed attempts.
3. Select a suitable estimation method for deriving the attitude of a tumbling object using only image input from monocular cameras.
    a. Compare suitable algorithms found in literature:
        - Consider computational complexity, accuracy, and ease of implementation.
        - Consider the RFSE 2021 Speed+ submission as a baseline.
    b. Select an algorithm for implementation:
        - Reduce pose estimation to attitude estimation if applicable.
        - The algorithm should extend the RFSE 2021 Speed+ submission.
4. Implement the AI attitude estimation algorithm in the Astrobee simulation environment, the code must be implemented using either C++ or Python as the programming language. This also requires the implementation of automatic keypoint labelling in the Astrobee environment using the robotic operating system (ROS) and related libraries. It further must be ensured that the implementation of the estimation algorithm is compatible with Astrobee and the SPEED+ challenge.
    a. Choose suitable keypoints on the Astrobee and attach a new TF frame for each keypoint to the target Astrobee.
    b. Implement a TF frame conversion for each keypoint TF to the chaser Astrobee's camera frame.
    c. Implement the automatic keypoint labelling and attitude estimation algorithm in a standalone ROS package for Astrobee.
5. Verify the attitude estimation using the Astrobee simulator.
    a. Derive a test schedule to verify the functionality of the implemented algorithm.
    b. Define test metrics to evaluate the performance of the orientation estimation algorithm.
    c. Define suitable test scenarios with different target tumbles.
    d. Perform the verification campaign.
    e. Evaluate the performance of the attitude estimation.
6. Optionally, port the implemented package for the SPEED+ challenge.
    a. Train AI with SPEED+ data sets
    b. Compare results with RFSE 2021

In the beginning of the thesis, a definition and description of individual work packages (Work Breakdown Structure, Work Package Description) are to be compiled to a project schedule and visualized in a timeline. The work must be done according to the guidelines of the Chair of Space Technology and must be handed over in two copies (original and copy), or digitally.
The Chair of Space Technology supports the scientific publication of the results of student work with prior approval. However, the results of the work may only be carried out after consultation with the supervising institutions. This work may be provided to third parties only after consultation with the supervising institutions. The work remains the property of the supervising institutions.

References:

[1] T. H. Park, S. Sharma, S. D'Amico, *Towards Robust Learning-Based Pose Estimation of Noncooperative Spacecraft,* 2019
[2] E. Morando, A. Fatehally, A. Zachold, P. Wörner, J. Plambeck, M. Rahimi, A. Ohm, *Pose Estimation of Uncooperative Satellites Based on 2D Monocular Images Using Advanced Image Processing Algorithms Combiend with Machine Learning,* Student project, Chair of Space Technology TU Berlin, 2021
[3] S. Sharma, S. D'Amico, *Neural Network-Based Pose Estimation for Noncooperative Spacecraft Rendezvous,* IEEE Transactions on Aerospace and Electronic Systems, 2020
[4] K. Albee, M. Ekal, C. Oestreich, P. Roque, *A Brief Guide to Astrobee's Flight Software*. 2021

Duration:	20 weeks
Supervisors:	Markus Huwald, M.Sc.
	Dr.-Ing. Mohamed Khalil Ben-Larbi
Start:	10.07.2023
Submission:	27.11.2023

Zusammenfassung

Die zunehmende Präsenz von Weltraumschrott stellt eine erhebliche Bedrohung für Raummissionen und insbesondere für die Sicherheit von Raumflügen dar. Dieser Schrott umfasst außer Betrieb gesetzte Satelliten, Raketenstufen und Fragmente, die aus Weltraumkollisionen resultieren, von denen viele unkontrolliert taumeln. Die genaue Schätzung der Position und Lage dieser nicht-kooperativen taumelnden Objekte ist unerlässlich für deren Manipulation und Entfernung. Diese Bachelorarbeitarbeit zielt darauf ab, diese Herausforderung durch die Entwicklung eines sichtbasierten Position und Lage-Schätzungssystems zu bewältigen.
Die Ziele dieser Studie umfassen eine gründliche Untersuchung der vorhandenen Literatur und der modernsten Techniken zur sichtbasierten Position und Lage-Schätzung und zum maschinellen Lernen, die Sammlung eines vielfältigen Datensatzes eines nicht-kooperativen taumelnden Objektes, des Astrobee-Roboters, mit Ground-Truth-Pose-Informationen, das Design und die Implementierung einer deep learning (DL)-Architektur für die Position und Lage-Schätzung, das Training und die Evaluierung des maschinellen Lernmodells sowie die Validierung durch Experimente und den Vergleich mit bestehenden Methoden.
Die Forschung mündet in einen hybriden Ansatz, der DL-basierte Objekterkennung mit einer geometrischen Methode kombiniert und die Bedeutung eines präzisen Datensatzes für eine robuste Objekterkennung verdeutlicht. Trotz einiger Einschränkungen, wie der Verwendung synthetisch generierter Daten und Ungenauigkeiten bei den Kameraparametern, legt diese Arbeit den Grundstein für weitere Verfeinerungen und vielseitige Anwendungen in der Raumrobotik und auf terrestrischen Szenarien, in denen genaue Position und Lage-Schätzung und Objekterkennung von entscheidender Bedeutung sind.
Diese Arbeit zeigt nicht nur die Machbarkeit der genauen Position und Lage-Schätzung für nicht-kooperative taumelnde Objekte innerhalb des Robot Operating System (ROS) auf, sondern bietet auch Potenzial zur Weiterentwicklung der Fähigkeiten von Robotersystemen in komplexen und dynamischen Umgebungen mit Auswirkungen auf Raumfahrtmissionen, einschließlich der aktiven Beseitigung von Weltraumschrott.

Abstract

The growing presence of space debris poses a significant threat to space missions, and particularly to the safety of space flights. This debris includes defunct satellites, rocket stages, and fragments resulting from space collisions, many of which tumble uncontrollably. Accurate pose estimation of these non-cooperative tumbling objects is essential for their manipulation and removal. This research aims to address this challenge by developing a vision-based pose estimation system.
The objectives of this bachelor thesis include a thorough investigation of the existing literature and state-of-the-art (SOTA) techniques in vision-based pose estimation and machine learning (ML), the collection of a diverse dataset of a non-cooperative tumbling object, the Astrobee robot, with ground truth pose information, the design and implementation of a DL architecture for pose estimation, training and evaluation of the ML model, and validation through experiments and comparative analysis with existing methods.
The research culminates in a hybrid approach that combines DL-based object detection with a geometric method, demonstrating the importance of a precise dataset for robust object detection. Despite some limitations, such as the use of synthetically generated data and inaccuracies in camera parameters, this work lays the foundation for further refinements and versatile applications in space robotics and terrestrial scenarios where accurate pose estimation and object detection are vital.
This work not only showcases the feasibility of accurate pose estimation for non-cooperative tumbling objects within the ROS but also holds promise for advancing the capabilities of robotic systems in complex and dynamic environments, with implications for space missions, including Active Debris Removal (ADR).

Contents

List of Figures

List of Tables

List of Acronyms

ADR	Active Debris Removal
AI	Artificial Intelligence
CLAHE	Contrast Limited Adaptive Histogram Equalization
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CV	Computer Vision
DG	Domain Gap
DL	Deep Learning
DLT	Direct Linear Transformation
DNN	Deep Neural Network
DOF	Degrees of Freedom
EPnP	Efficient PnP
Fast R-CNN	Fast Region-based Convolutional Network
FOV	Field-of-View
HRNet	High-Resolution Net
i.i.d.	Independent and Identically Distributed
IoU	Intersection over Union
ISS	International Space Station
JEM	Japanese Experiment Module
JPEG	Joint Photographic Experts Group
KRN	Keypoint Regression Network
Mask R-CNN	Mask Region-Based Convolutional Neural Network
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
NASA	National Aeronautics and Space Administration
NN	Neural Network
ODN	Object Detection Network
PnP	Perspective-n-Point
RANSAC	RANdom SAmple Consensus
ReLU	Rectified Linear Unit

RMSE	Root Mean Square Error
RoI	Region of Interest
ROS	Robot Operating System
SGD	Stochastic Gradient Descent
SOTA	State-Of-The-Art
SSD	Single Shot MultiBox Detector
TF	Transform Frames
YOLO	You Only Look Once
YOLOv8	You Only Look Once Version 8

List of Symbols

$a \cdot b$	Multiplication (dot product).
$\sum$	The symbol (sigma) denotes a sum of multiple terms.
$\hat{y}_i$	Predicted output value for the i^{th} observation.
y_i	Ground true value for the i^{th} observation.
n	Sample size.
σ	Activation function of a neuron.
ω_i	Connection weight for the i^{th} input value.
b	Bias parameter in neural network.
x_i	i^{th} input value.
X	Instances domain (usually a set).
Y	Labels domain (usually a set).
L	Loss function.
D	A distribution over some set (usually over X).

1. Introduction

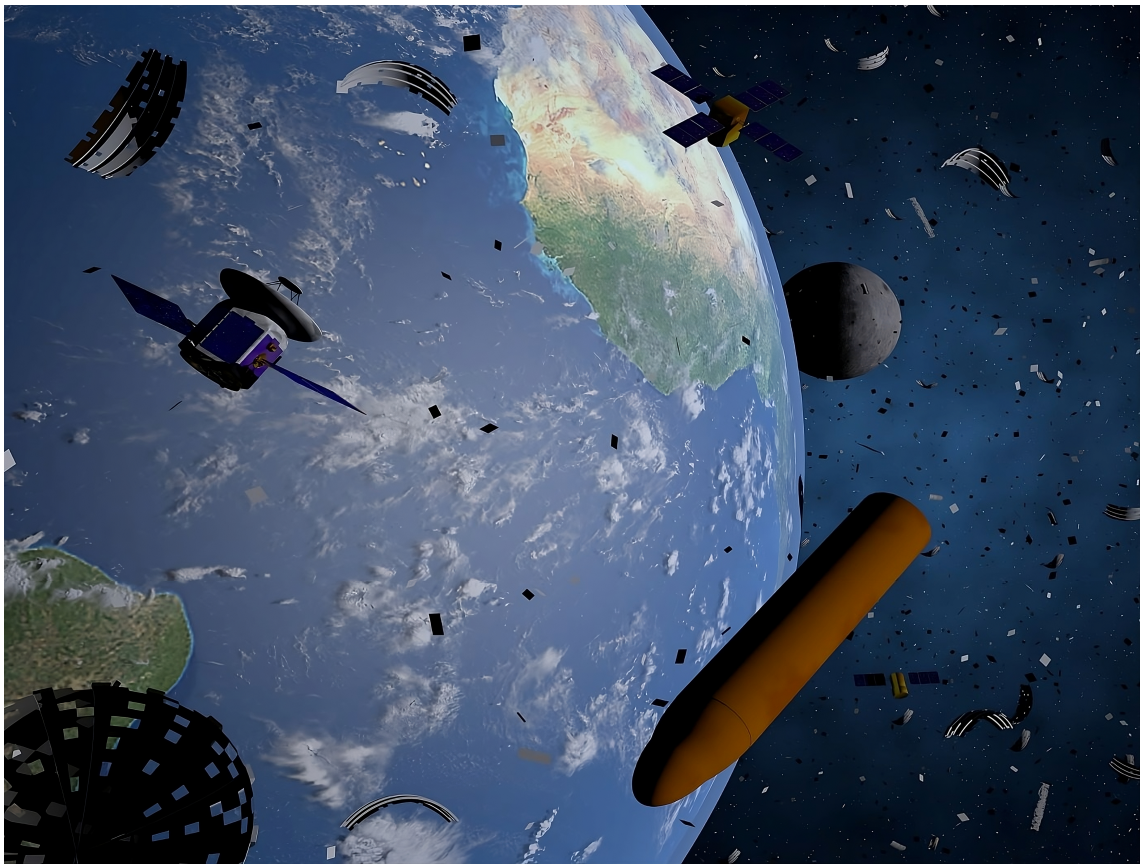


Figure 1.1: Modeled image of debris generated by past human space missions in low-Earth orbit (originally published in [6]).

In the course of space exploration, humanity, albeit not immediately, has encountered the issue of space debris. This term encompasses all artificial objects and their fragments present in space that are no longer operational, devoid of functionality, and incapable of serving any useful purpose [47]. This includes exhausted satellites, upper stages of launch vehicles, fragments resulting from rocket explosions etc. [12]. An example illustration of the space debris around the earth is depicted in figure 1.1. With the proliferation of space wreckage, both in terms of quantity and size, the potential for causing destruction has escalated. The high velocity of small orbital particles exacerbates the severity of collisions, posing numerous threats, with the foremost being the endangerment of space flights' safety [3].

1.1 Non-cooperative tumbling Object

A non-cooperative tumbling object refers to an object, typically in a space environment, that is rotating or tumbling uncontrollably and is not actively cooperating with attempts to interact with, stabilize, or capture it [87]. The term is most commonly used in space missions and satellite operations. When satellites malfunction, or debris is created from collisions or other events in space, these items can start tumbling, making them non-cooperative. Due to the lack of external forces in space (like air resistance), a tumbling object can continue its motion for a long time. The motion can be challenging to predict, especially if the initial conditions of the tumbling are unknown. Such tumbling objects typically lack cooperative behavior, distinguishing it from targets of on-orbit servicing missions. As a result, the capture and removal of space debris present significantly greater challenges. In order to manipulate or interact with the debris, it is necessary to accurately predict its pose [69].

1.2 Background

In recent years, there has been a growing interest in developing advanced technologies for object pose estimation using vision-based systems. Pose estimation, the process of determining the position and orientation of an object in a given environment, plays a crucial role in various fields, including robotics, augmented reality, object tracking, and human-computer interaction. Traditional pose estimation methods heavily rely on cooperative objects that have identifiable features or markers [84]. However, accurately estimating the pose of non-cooperative objects, particularly those that are tumbling or in motion, remains a challenging problem [87].

1.3 Objectives

The primary objective of this research is to develop a vision-based pose estimation system that can accurately determine the position and orientation of a non-cooperative tumbling object. To achieve this, the following specific objectives will be pursued:

1. Investigate existing literature and SOTA techniques in vision-based pose estimation and ML.
2. Collect a diverse dataset comprising images of a non-cooperative tumbling object with ground truth pose information.
3. Design and implement a suitable DL architecture capable of capturing and analyzing visual data for pose estimation.
4. Train and evaluate the proposed ML model using the collected dataset to achieve accurate pose estimation results.
5. Validate the proposed approach through experiments and comparative analysis with existing methods.

Through this research, we aim to contribute to the development of robust and accurate pose estimation methods for a non-cooperative tumbling object, leveraging the power of ML and computer vision (CV) techniques.

2. Pose Estimation and Robotics

Some important concepts related to pose estimation and a few theories regarding CV, Astrobee Simulation Environment and ROS will be discussed in this chapter for better understanding.

2.1 Pose Estimation and Computer Vision

Pose estimation involves determining the position and orientation of an object in 3D space. Specifically, 6D pose refers to an object's posture, defined by a translation vector and a rotation vector. This estimation is crucial in various industrial fields [28]. In CV and robotics, the term is commonly used in two main contexts: human pose estimation and object pose estimation.
In human pose estimation if we have a 2-dimensional portrayal of a human being, 3D pose estimation entails generating a three-dimensional representation that aligns with the spatial positioning of the depicted person. To transition from an image to a 3D pose, an algorithm must exhibit invariance to various factors, such as background scenes, lighting conditions, clothing shape and texture, skin color, and image imperfections, among other considerations. Lately, certain systems have investigated directly inferring 3D poses from images using end-to-end deep architectures [35]. In contrast, earlier methods for pose estimation relied on image processing with manually crafted features and pre-existing pose knowledge [71]. The effectiveness of these methods significantly depends on the availability of annotated training images that accurately capture the details of an object [53].
Object pose estimation determines the position and orientation of objects relative to a camera coordinate system in 3D space from their 2D projections in images or videos. Recently, there has been a growing number of pose estimation applications. Autonomous vehicles utilize 6D pose estimation to identify roads and obstacles, while factory robots employ the same technology for recognizing and grasping objects [28]. In augmented reality, 6D pose estimation measures the pose of real-world objects, enabling the correct placement of virtual objects onto them [77]. This task can again be challenging due to factors such as occlusions, varying lighting conditions, and the presence of similar-looking objects [7]. The pose of an object can be represented using 6-degrees-of-freedom (DOF) - three for translation (position) in the x, y, z axes and three for rotation, often referred to as pitch, yaw, and roll [28]. However, in this work, we opt for quaternions as our rotation representation due to their compactness, numerical stability, and immunity to singularities or gimbal lock. This choice is widely favored in the literature for 3D orientation in both robotics and DL, given the well-studied parametric distributions and various uniform sampling strategies associated with quaternions [57]. A quaternion is a four-dimensional vector that

can be written as $q = (x, y, z, w)$, where w is the scalar part of the quaternion and x, y and z are the vector components. Together, the scalar and vector components encode the angle and axis of rotation.
CV is a rapidly evolving field within artificial intelligence (AI) that focuses on developing algorithms and techniques for computers to interpret visual data akin to human visual perception [48]. By extracting meaningful information from images or videos, machines can recognize objects, comprehend scenes, and make intelligent decisions based on visual input [20].
Image classification and object detection are key tasks in CV, aiming to assign predefined labels to images and detect keypoints on objects of interest, respectively [9]. It finds applications in face detection, medical imaging analysis, manufacturing, transportation and entertainment industries. Through ML algorithms, CV systems can learn patterns and features indicative of specific objects or classes by training on extensive datasets. Object detection and tracking are essential aspects of CV that involve identifying and localizing specific objects within images or video sequences. This has implications in surveillance systems, autonomous vehicles, augmented reality and human pose estimation applications [37]. Sophisticated algorithms are required to accurately identify objects, handle occlusions, and track their movements over time [41].

2.2 Common Computer Vision Pose Estimation Tasks

This section discusses prevalent tasks in CV related to pose estimation, providing an overview of various challenges and applications in this domain.

2.2.1 Human and Face Pose Estimation

Human pose estimation involves determining the positions and articulations of body joints such as arms, legs, and head in human subjects captured in images or videos. This process is integral to a range of applications, including action recognition, gesture analysis, sports analytics, and facilitating interactions between humans and computers [53].
Additionally, the estimation of the 3D position and orientation of a person's face in images or videos is another critical task in this field. This aspect of pose estimation is pivotal for facial analysis, the creation of augmented reality face filters, and accurate gaze estimation [31].

2.2.2 Robotics

Pose estimation plays a crucial role in robotics, providing valuable spatial information about the robot, its surroundings, and objects it interacts with. It enables robots to perceive and understand their environment, which is essential for performing various tasks and interacting effectively with the world [15].
Pose estimation helps robots determine their own position and orientation in relation to a given map or the environment. This allows the robot to navigate and move autonomously, avoiding obstacles and reaching specific destinations [50].

2.3 Robotics and the Astrobee Framework

Astrobee Simulation Environment is a virtual platform designed to simulate and test the Astrobee robot system. Astrobee is a free-flying robotic system developed by National Aeronautics and Space Administration (NASA) for assisting astronauts aboard the ISS [10]. The simulation environment replicates the behavior and capabilities of the Astrobee robots, allowing researchers, engineers, and developers to experiment with and evaluate the robot's performance in a controlled virtual setting [22].
The simulation environment provides a realistic representation of the physical dynamics, sensors, and functionalities of the Astrobee robots. It enables users to simulate various scenarios, such as maneuvering in microgravity, interacting with objects in the environment, and executing specific tasks or missions. Users can modify parameters, test algorithms, and analyze the robot's behavior without the need for physical hardware [56]. The illustration 2.1 depicts two Astrobee robots within the Astrobee ROS simulation environment. The target is tumbling in the JEM on the ISS. The two Astrobees are part of a set of three. The yellow one is called Honey Astrobee, and the blue one is called Bumble. They act as the chaser and the target in our scenario.

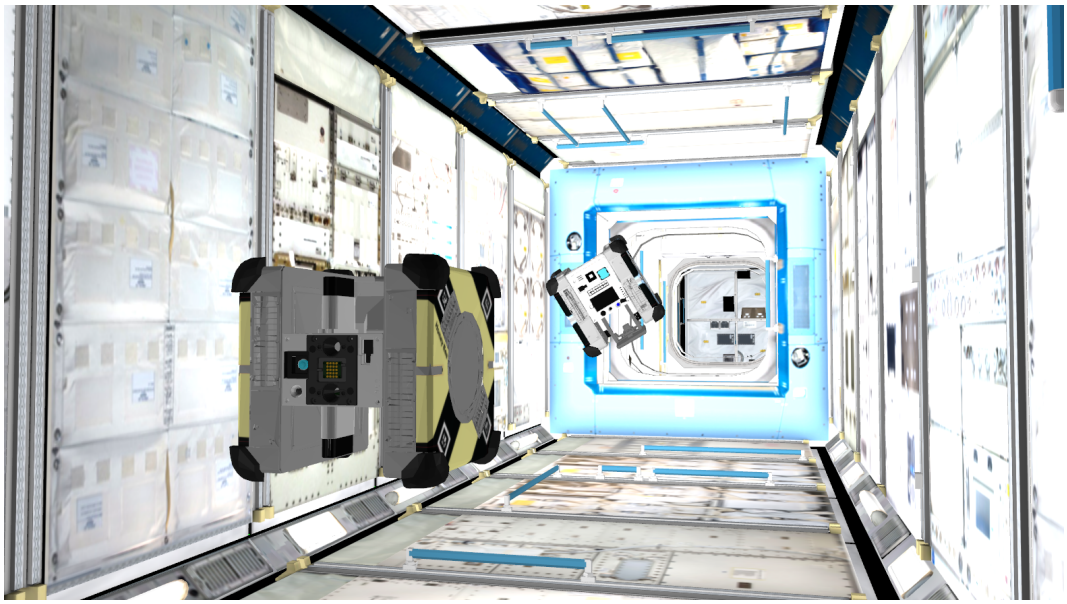


Figure 2.1: Two Astrobee robots in the ISS simulation environment within the JEM.

The Astrobee Simulation Environment is a valuable tool for validating and refining robot control algorithms, planning and coordination strategies, and human-robot interaction interfaces. It allows researchers to iterate and optimize the behavior of the Astrobee system before deploying it on the actual ISS, minimizing risks and enhancing the overall performance of the robots in space.
ROS is an open-source framework widely used in robotics for developing and controlling robot systems. It provides a collection of software libraries, tools, and

communication protocols that enable seamless integration and interoperability between various components of a robotic system. ROS offers a modular and distributed architecture, allowing developers to build complex robotic applications by connecting and coordinating different software modules called nodes [11]. These nodes can perform specific tasks, such as sensor data processing, path planning, motion control, or perception. They communicate with each other using a publish-subscribe messaging system, where nodes can publish data (topics) or subscribe to receive data from other nodes [39]. Figure 2.2 shows the data interchange process.

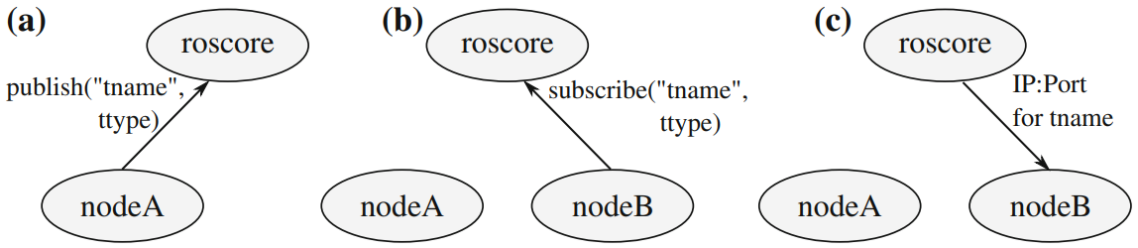


Figure 2.2: Overview of the ROS topic mechanism. When a publisher nodeA (a) and subscriber nodeB (b) are connected to the same topic roscore, the subscriber receives an IP and Port of all publishers (c) [38].

ROS provides a range of libraries and packages for common robotic functionalities, including perception, localization, mapping, and manipulation. It supports different programming languages such as C++, Python, and provides a rich set of tools for visualization, simulation, debugging, and analysis [63]. One of the strengths of ROS is its large and active community, which contributes to an extensive ecosystem of existing packages, libraries, and resources. This allows developers to leverage pre-existing solutions, share code, and collaborate on robotics projects more efficiently. Overall, ROS simplifies the development and integration of robotic systems by providing a flexible and standardized framework. It has become a widely adopted platform for prototyping, research, and commercial robotic applications, enabling faster development cycles, code reusability, and interoperability among different robotic components [5].

3. Machine Learning

Few theories regarding ML, NNs, CNN and Existing DL-Based Approaches for pose estimation will be discussed in this chapter.
The term ML denotes the automated identification and recognition of significant patterns within datasets [68]. It is a field of study within computer science that encompasses the development and implementation of algorithms and statistical models, enabling computational systems to automatically acquire knowledge, improve performance, and make predictions or decisions through iterative learning from data without explicit programming [32]. This field is garnering significant attention nowadays as it facilitates the direct training of machines with reduced reliance on human interaction [37].
Supervised learning is a subfield of ML which involves the task of training a machine to learn a generalized mapping or function f, that can map pre-labelled inputs from X to corresponding output labels from Y [54]. The objective is for the trained model, with the assistance of the given pairs $(x, y) \in (X, Y)$, to accurately predict the output labels for new or unseen input data [73]. In the context of ML, this function is denoted as $f : X \to Y$. For instance, in the field of CV, X can represent a set of satellite images, while Y can represent a set of labels or a probability interval $[0, 1]$ indicating the likelihood of a satellite being present in the image. By leveraging a dataset of labeled images, where humans have identified whether a satellite is present or not, we can establish a function that can determine whether an image contains a satellite.
In order to derive this function, the initial step involves acquiring a training dataset that comprises a set of n examples, denoted as $\{(x_1, y_1), ..., (x_n, y_n)\}$. These examples are assumed to be independent and identically distributed (i.i.d.) according to a distribution D.
The domain gap (DG) reflects the disparity between datasets, including differences in data distribution. If a DG exists, representing the difference between datasets or environments, the evaluation of the model is limited to synthetic data when authentic data is unavailable. This challenge emerges when the model, typically trained on a synthetic dataset, is expected to perform on a different, real-world dataset. Such a scenario can impact the model's real-world performance due to discrepancies between the two datasets [65].
Once the training samples are obtained, the next step usually involves identifying the most consistent mapping function f for the given dataset. This entails exploring and

selecting a suitable loss function that quantifies the disparity between the predicted label $\hat{y}$ and the true label y. An example of a loss function is:

$$L = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{3.1}$$

which is called the root-mean-square error (RMSE) and is used to minimize the error in a regression model by calculating the square root of the average squared differences between the true and predicted values in a dataset [30].
However, when it comes to classification tasks, other metrics such as precision and recall become more relevant. Precision is a measure of a classifier's exactness. A higher precision score indicates that the majority of predictions labeled as positive are indeed positive. Mathematically, it is defined as the number of true positive results divided by the number of all positive results, including those not identified correctly [24].
Recall, on the other hand, is a measure of a classifier's completeness. A higher recall score indicates that the classifier returned most of the relevant results. It is calculated as the number of true positive results divided by the number of all samples that should have been identified as positive [24].
Both precision and recall are therefore critical to understanding the effectiveness of a classifier, especially in scenarios where the balance between identifying true positives and avoiding false positives is essential.
Once we have determined our objective function f, we can utilize the learned function to map elements from X to Y.

3.1 Neural Networks

A NN is composed of interconnected perceptrons, also known as artificial neurons or nodes [91]. A single perceptron is illustrated in Figure 3.1

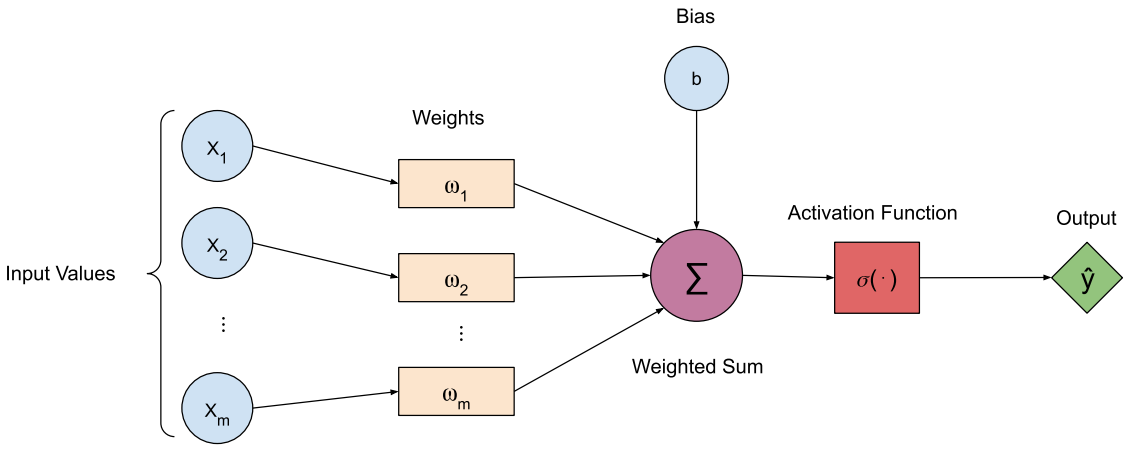


Figure 3.1: Example of a single artificial neuron or perceptron.

At its core, a perceptron is a binary classifier that takes a set of input values and produces a single binary output. It mimics the functioning of a biological neuron by receiving input signals, applying weights to those inputs, and producing an output based on a threshold activation function [91]. The perceptron model consists of the following components:

1. Input Values: The perceptron receives a set of input values, usually represented as a vector, which could represent features or attributes of a given input sample. Each input is associated with a weight that determines its relative importance.
2. Weights: Each input value is multiplied by a corresponding weight, which reflects the strength or significance of that input in the decision-making process. These weights are adjustable parameters that are learned during the training process.
3. Summation Function: The weighted inputs are summed together, often using a linear combination, to compute the weighted sum.
4. Activation Function: The weighted sum is passed through an activation function, which introduces non-linearity and determines the output of the perceptron. One of the most common activation function used in perceptrons is the binary step function or the threshold function,

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}, \tag{3.2}$$

   which outputs a 1 if the weighted sum exceeds a certain threshold, and 0 otherwise.
5. Bias: A bias b is often included in the perceptron to provide an additional adjustable parameter that can shift the activation threshold. It allows the perceptron to make decisions even when all input values are zero [24; 26; 70].

During the training process, the perceptron adjusts its weights based on the observed errors in its predictions [23]. This adjustment aims to minimize the error by updating the weights in the direction that reduces the discrepancy between the predicted output,

$$\hat{y} = \sigma(\sum_{i=1}^{m}(x_i \cdot w_i) + b), \tag{3.3}$$

and the desired output y [86].

While a single perceptron can only classify linearly separable patterns, multiple artificial neurons can be combined to form more complex models [51]. A combination of multiple neurons is called a NN [19]. Figure 3.2 illustrates an example of a NN consisting of 2 hidden layers. A NN with more than 1 hidden layer is also called a deep neural network (DNN) [24; 52].

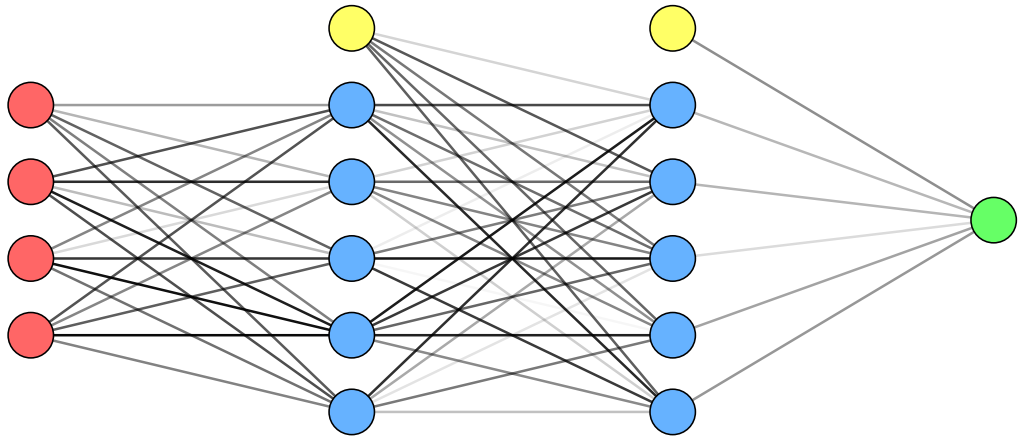


Figure 3.2: A simple NN consisting of an input layer, two hidden layers including bias nodes (highlighted in yellow), and an output layer. The opacity of each edge represents the strength or importance of a connection.

3.2 Loss, Gradient Descent and Backpropagation

In this subsection, we will define several important terms, such as Loss, Gradient Descent, Backpropagation, and other terms often used while building a ML model.

Loss function is employed to assess the performance of the system as a whole. It quantifies the errors, enabling the model to evaluate the effectiveness of the current system. The choice of loss function can influence and guide specific behaviors, either promoting or discouraging them [16].

Gradient Descent is an optimization algorithm commonly used in ML and NN to minimize the loss function and find the optimal values for the model's parameters (weights) [67].
The goal of gradient descent is to iteratively adjust the model's parameters, as depicted in Figure 3.3, in the direction of the steepest descent of the loss function. The idea is to gradually approach the minimum of the loss function, where the model achieves its best performance [67].
The algorithm starts by initializing the parameters with some initial values. It then calculates the gradient of the loss function with respect to each parameter [4]. The gradient represents the direction of the steepest ascent of the loss function. However, since the objective is to minimize the loss, the algorithm takes a step in the opposite direction [26].

Stochastic gradient descent (SGD) is a variant of gradient descent. SGD updates the parameters using the gradient of the error with respect to a single training example, rather than the sum of the gradient of the errors for all training examples. This makes it faster and more scalable [8].

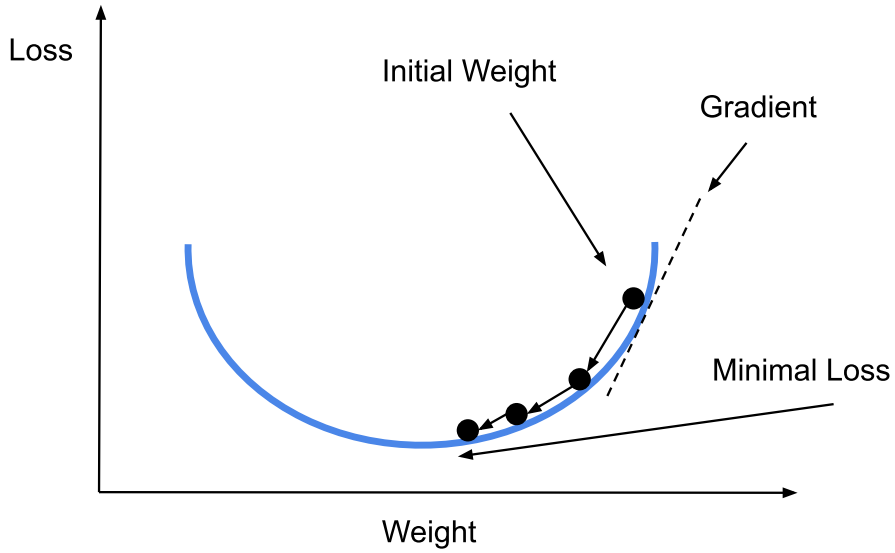


Figure 3.3: Visualisation of Gradient Descent.

Backpropagation is an algorithm used to train NNs by iteratively adjusting the weights of the connections between neurons [81].
During the forward pass of backpropagation, input data is fed through the network, and the output is computed. Then, the difference between the predicted output and the expected output (the error) is calculated using a chosen loss function [17].
Next, the algorithm performs the backward pass. It starts from the output layer and propagates the error backward through the network, layer by layer. For each layer, the error is distributed among the connections based on the weights and the derivative of the activation function.
The gradients of the weights with respect to the error are computed during the backward pass [17]. Consequently, the algorithm adjusts the weights as described in Gradient Descent.

Hyperparameters control various aspects of the learning algorithm and the model's architecture, influencing how the model is trained and how it generalizes to new data [79]. The choice of hyperparameters can significantly impact the model's performance, training time, and ability to avoid issues like overfitting and underfitting. Common examples of hyperparameters in ML include:

1. Epochs: The number of epochs is a hyperparameter that determines how many times the model will iterate through the entire training dataset during the training process. It influences how long the model is trained and how well it can capture the underlying patterns in the data [74].
2. Learning Rate: The learning rate determines the step size at which the model updates its parameters during training. A high learning rate may cause the model to overshoot optimal solutions, while a low learning rate may lead to slow convergence or getting stuck in local minima [26; 85].
3. Batch Size: The batch size determines the number of samples used in each iteration of training. A smaller batch size may lead to more frequent updates. With a smaller batch size, we are processing and updating the model's weights

more frequently during each epoch, while a larger batch size can speed up training but requires more memory. This can be a limitation on hardware with limited memory capacity, potentially causing out-of-memory errors [36].

The choice of hyperparameters is an essential aspect of the ML pipeline, and tuning them effectively can lead to significant improvements in model performance and generalization.

Underfitting and Overfitting are common issues that occur when training ML models, including NN. They both refer to the model's performance and its ability to generalize to unseen data [24].
Underfitting occurs when a model is too simplistic to capture the underlying patterns and complexities present in the training data. In other words, the model is not able to learn the relationships between the input features and the target output accurately. As a result, the model's performance is poor both on the training data and on new, unseen data. Signs of underfitting:

- The model's performance on the training data is low.
- The model's performance on the validation or test data is also low.
- The model struggles to learn from the training data and fails to make meaningful predictions.

Causes of underfitting:

- Using a model with too few parameters or too simple architecture.
- Insufficient training data or a lack of diversity in the data.

To address underfitting, we can try:

- Use a more complex model with more parameters.
- Collect more diverse and representative training data.
- Reduce regularization or constraints during training [24].

Overfitting occurs when a model becomes too complex and starts to memorize noise or random fluctuations in the training data rather than learning general patterns [33]. The model becomes too tailored to the training data, leading to poor performance on new, unseen data. Signs of overfitting:

- The model's performance on the training data is excellent.
- However, the model's performance on the validation or test data is significantly worse.
- The model shows erratic or fluctuating behavior when evaluated on different subsets of the data.

Causes of overfitting: Using a model with too many parameters or an overly complex architecture. Having limited training data, leading the model to over-adapt to the available examples. To address overfitting, we can try:

- Reducing the complexity of the model by using fewer parameters or a simpler architecture.
- Increasing the size of the training dataset to provide more diverse examples.

Finding the right balance between underfitting and overfitting is a crucial aspect of model training. Ideally, the model should be able to generalize well to new data while accurately capturing the underlying patterns in the training data. Regular model evaluation and adjustment of hyperparameters are essential to achieve a well-generalized and high-performing model [24; 33; 64].

3.3 Convolutional Neural Networks

CNNs are a specialized type of NN designed for processing structured grid-like data, such as images and audio spectrograms. They are particularly effective in tasks that involve visual recognition and analysis [26; 40].
CNNs are characterized by their unique architectural elements, which include convolutional layers, pooling layers, and fully connected layers [43].
Convolutional layers perform feature extraction by applying a set of learnable filters, also known as kernels or convolutional filters, to the input data. These filters scan the input spatially and generate feature maps by computing convolutions. The convolutions capture local patterns and spatial dependencies in the data.
Pooling layers reduce the spatial dimensions of the feature maps while retaining their essential features. Max pooling, one of the most common types of pooling layers, selects the maximum value within each pooling region, effectively downsampling the data. Pooling helps make the network translation invariant, enabling it to recognize patterns regardless of their precise location in the input [24; 43].
Fully connected layers, commonly found at the end of the network, integrate the extracted features and make predictions based on the learned representations. These layers connect every neuron from the previous layer to every neuron in the next layer, enabling the network to learn complex relationships and classify the input into various classes [45].
Once these layers are stacked together, a CNN architecture is created. Figure 3.4 illustrates a simplified CNN architecture specifically designed for modified National Institute of Standards and Technology (MNIST) database, which consists of a large collection of handwritten digits.

CNNs also often incorporate non-linear activation functions, such as rectified linear unit (ReLU)

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}, \tag{3.4}$$

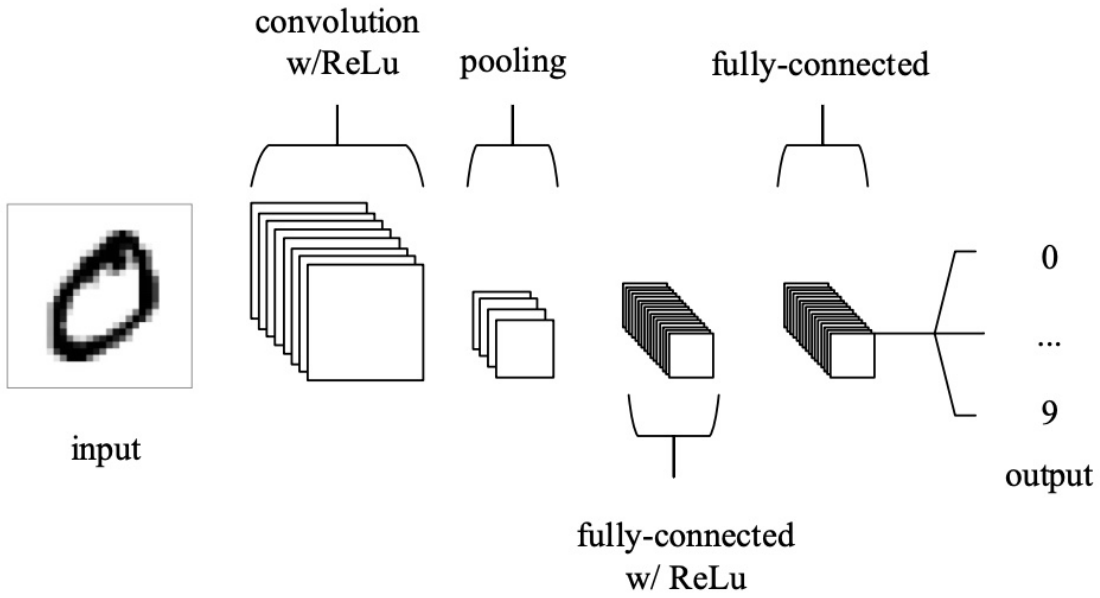


Figure 3.4: An example of a CNN architecture designed for classifying handwritten digits (originally published in [58]).

to introduce non-linearity and increase the network's representational power [58]. Through the combination of convolutional layers, pooling layers, fully connected layers, and non-linear activations, CNNs can automatically learn and extract meaningful features from raw input data. This makes them highly effective in tasks such as image classification or object detection. CNNs have achieved remarkable success in various fields, including CV and natural language processing [2; 24].

3.4 Transfer Learning and Pretrained Models

Transfer learning is a ML technique that involves utilizing knowledge or representations learned from one task or domain to improve learning or performance on a different but related task or domain. Instead of training a model from scratch for the target task, transfer learning leverages the pre-existing knowledge from a source task, often with a large dataset, to enhance the performance of the target task with limited data [55; 80].
Pretrained models are models that have been trained on large-scale datasets for a specific task, such as image classification or object detection. These models have learned general features and patterns that are transferable to other tasks. In transfer learning, pretrained models are used as a starting point, and their learned weights and parameters are reused. Instead of training the model from scratch, only a portion of the model, typically the last few layers, is adapted or fine-tuned to the target task's specific requirements and data [24; 27].
Pretrained models provide several advantages in transfer learning. They offer a head start by providing a strong initial set of weights, which are already optimized for general features in the source task. This initialization allows the model to converge faster and achieve better performance with less training data on the target task. Pretrained models also capture valuable knowledge about complex features and

patterns in the data, which can be beneficial for tasks with limited labeled data [27]. By leveraging pretrained models, transfer learning enables researchers and developers to benefit from the collective knowledge and expertise gained through extensive training on large datasets [55]. It has become a popular technique, especially in CV and natural language processing, where pretrained models, such as those based on CNNs, have achieved SOTA performance on various tasks [90].

Backbone Model refers to the pre-trained portion of a NN, often derived from a larger model trained on a comprehensive dataset like ImageNet or common objects in context (COCO). In transfer learning, this backbone captures general features while the added layers, which are typically retrained, adapt to the specific task at hand [14].

Classifier Head is the portion of a NN added to a pre-trained backbone model. It is typically trained on the target task and is responsible for producing the final task-specific predictions or classifications [25].

3.5 Object Detection Networks

An object detection network is a type of DL model specifically designed to identify and localize objects within an image or video. It is a fundamental component of CV systems and finds applications in various fields, including autonomous driving, intelligent transportation, and intelligent surveillance [88].
Object detection networks employ a combination of CNNs and other architectural components to accomplish their task. These networks are trained on large-scale datasets with annotated images, where objects of interest are labeled with bounding boxes or landmarks [88].
During training, the object detection network learns to optimize both the classification and regression tasks using a combination of supervised loss functions. Common loss functions used in object detection include cross-entropy loss for classification and smooth L1 loss for bounding box regression. Object detection networks have evolved over time, with popular architectures like fast region-based convolutional network (Fast R-CNN), Mask R-CNN, YOLO, and single shot multibox detector (SSD) being widely used. These architectures vary in terms of speed, accuracy, and trade-offs between localization precision and computational efficiency [34; 42; 78].

3.6 Existing Deep Learning-based Pose Estimation Approaches

In this section, we present the related work pertaining to our solution and provide examples of two primary approaches.

Single-stage, end-to-end or direct approach for pose estimation aim to directly predict the pose of an object from input data, such as images or sensor measurements [7; 44]. These methods utilize DL techniques such as CNNs to learn the mapping between the input data and the pose parameters [82]. The network is trained on

a labeled dataset that contains images or sensor measurements along with their corresponding ground truth pose information. The network learns to extract relevant features from the input data and regress the pose parameters directly [44; 59]. By training on large datasets and leveraging CNN architectures, single-stage pose estimation methods can capture complex patterns and relationships in the data, enabling accurate and efficient estimation of object attitude and position [83].
In a study referenced in [72], several methodologies were explored, shedding light on key findings: Firstly, the research highlighted a strong correlation between the size of the training dataset and the achieved accuracy. This underscored the importance of larger synthetic datasets, emphasizing the need for diverse variations to enhance accuracy effectively.
The study included the creation of ten distinct datasets, each introducing different types of added noise derived from synthetic images. These varied datasets proved beneficial, exhibiting promising performance on noisy test images. Such results indicated potential success in accurately determining poses in space imagery, particularly following noise modeling or removal.
Another significant discovery was related to transfer learning. By training only the final layers, the study revealed shared features between spaceborne and terrestrial imagery. This approach showed promising adaptability and potential for effective utilization.
Regarding performance metrics, the NN was trained using an extensive dataset of 75,000 images associated with 3000 pose labels. This training resulted in superior performance when compared to classical feature detection algorithms. As a result, the study concluded that this NN approach offered a confident and reliable solution for pose estimation tasks.
While acknowledging the potential for increased accuracy with larger datasets and more pose labels, the study also noted a potential downside: the utilization of larger networks might require increased memory on the servicer spacecraft.
An alternative method for direct attitude estimation has been presented in [56], exploring the potential of DL techniques in spacecraft pose estimation. This approach utilizes AlexNet, illustrated in 3.5, as its backbone, primarily modifying the final fully connected layer to generate attitude labels.
The resultant attitude is further refined using a direction cosine matrix, a technique that averages the eight most probable labels to enhance accuracy.
Synthetic images of the SpaceX Dragon capsule were synthesized through Blender, producing four distinct image sets used to evaluate the approach. These sets included variations in backgrounds, encompassing black and empty backgrounds, different Sun angles, Earth backgrounds, and simulated sensor noise.
The experiment findings showcased an intriguing trend: both classification accuracy and attitude error demonstrated an asymptotic pattern. Notably, the CNN exhibited robust performance in challenging lighting conditions, particularly in datasets with varied Sun angles. However, its performance notably declined in datasets featuring Earth backgrounds and sensor noise.
Notably, incorporating a confidence rejection threshold in the refinement step could potentially enhance accuracy.
In conclusion, these insights into DL-based spacecraft pose estimation lay a foundational groundwork for potential advancements and future research in this field.

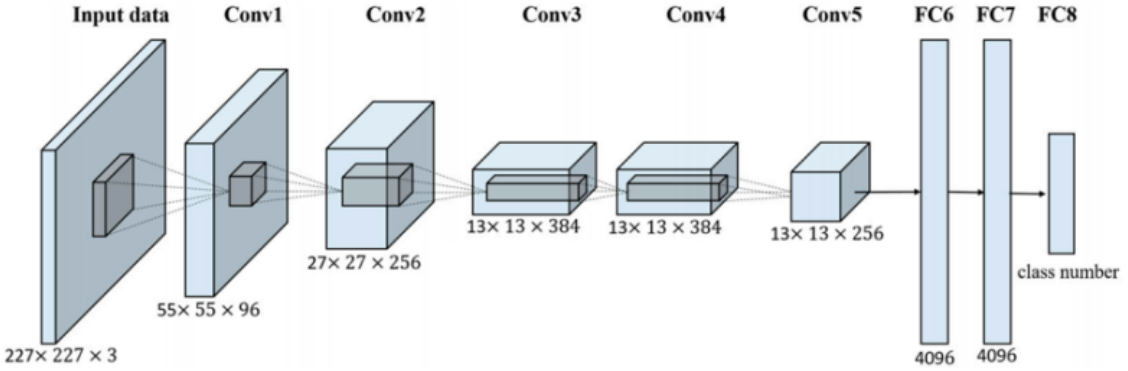


Figure 3.5: Overview of AlexNet architecture (originally published in [56]).

Two-stage or indirect pose estimation methods breaks down the process into distinct phases, simplifying the challenge and leveraging different strengths of the ML models involved [61].

1. Keypoint Detection
   The goal here is to identify specific points (keypoints) on the object in the image. These keypoints should correspond to known points on the 3D model of the object [60].
   The result of this stage is a set of 2D points in the image that correspond to specific, known points on the object [61]. Ideally, these points should be detected reliably across different views, lighting conditions, and occlusions.
2. Pose Estimation using PnP
   Given the 2D keypoints from the image and their corresponding 3D points on the object's model, the goal is to estimate the object's pose (position and orientation) relative to the camera [61]. The PnP problem is about finding the pose of a camera given a set of 3D-to-2D point correspondences [89]. Various algorithms exist to solve the PnP problem, such as direct linear transformation (DLT), efficient PnP (EPnP), and P3P (for cases with exactly three point correspondences). Often, the PnP step is combined with a method like RANdom SAmple consensus (RANSAC) to handle noise, mismatches, and outliers in the detected keypoints [46]. RANSAC ensures that the computed pose is robust to such anomalies by iteratively estimating the pose using random subsets of keypoints and choosing the pose that fits the maximum number of points [21].

An example of such an approach has been proposed in [59]. Their approach involves modifying a CNN-based Spacecraft Pose Network by integrating a novel CNN for target detection and employing RANSAC algorithms to solve the PnP problem.
Their method employs two networks: the object detection network (ODN) responsible for detecting the 2D bounding box of the region of interest (RoI), and the keypoint regression network (KRN) dedicated to regressing the 2D location of keypoints.
The architecture of ODN and KRN aligns with the pipeline of YOLOv2/YOLOv3, leveraging MobileNetv2 and MobileNet, respectively, to achieve their objectives. To reduce network parameters, convolution operations within the network are substituted with depth-wise convolutions followed by point-wise convolutions, improving

computational efficiency.
The primary goal of this approach is to augment the effectiveness and feasibility of DL-based pose estimation within space missions. To bolster their methodology, they generated a new dataset by employing neural style transfer techniques.
Following training on this novel texture-randomized dataset, their proposed network exhibits enhanced performance on spaceborne images and achieves the notable position of 4th place in Kelvins Pose Estimation Challenge. The architecture of the pipeline can be seen in 3.6.

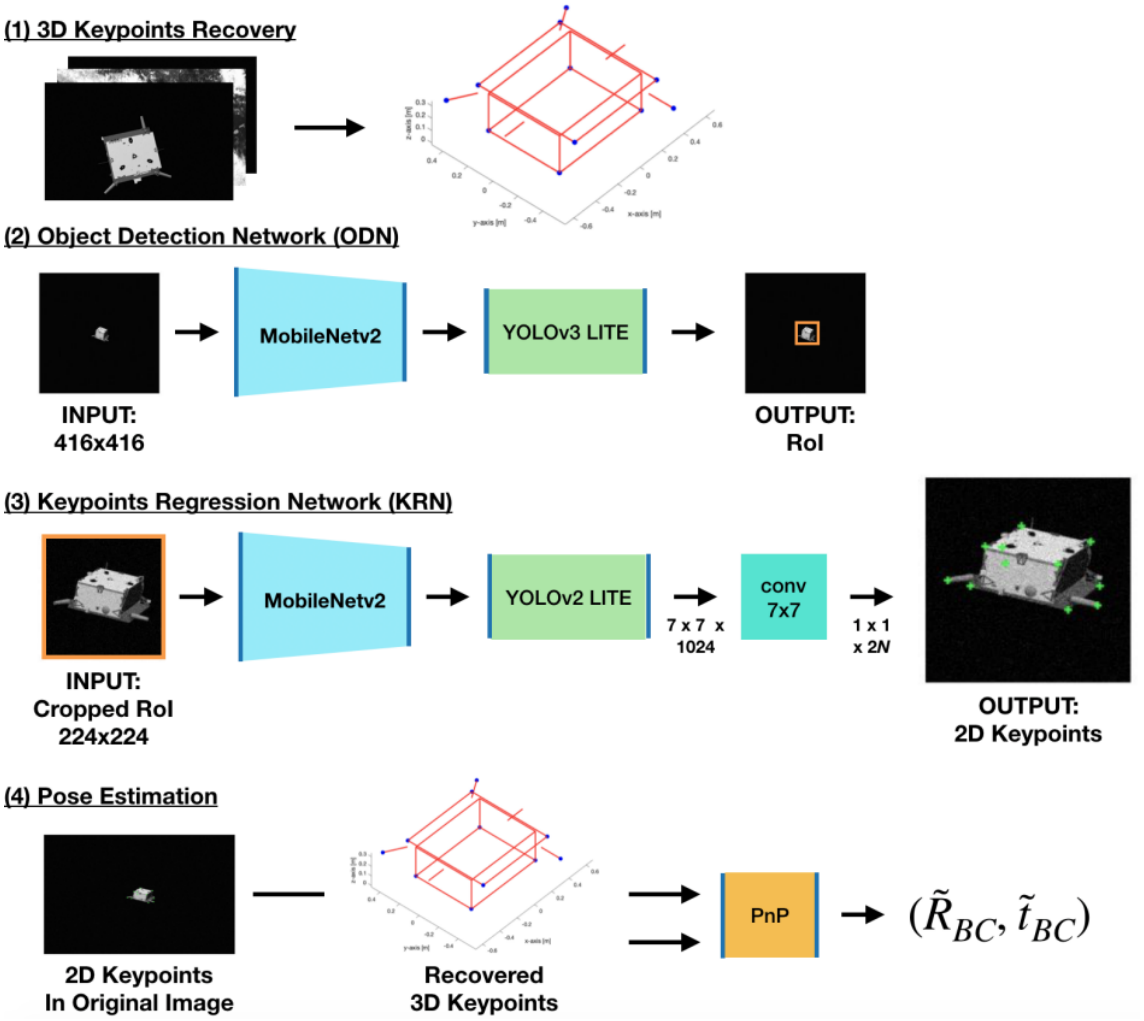


Figure 3.6: Architecture of the CNN-based Spacecraft Pose Network (originally published in [59]).

To highlight another indirect approach, the research detailed in [13] presents an innovative fusion of DL methodologies with geometric optimization techniques, specifically designed for the precise estimation of pose from a single image.
Their methodology entails various steps, starting with the computation of a 3D model of the satellite derived from multiview triangulation within the training dataset. Subsequently, they employ a high-resolution net (HRNet) to regress the location of 2D corner point landmarks projected onto the spacecraft from the input greyscale image. The initial pose estimation is achieved via a PnP solver.
This approach, illustrated in 3.7 comprises two main modules: the first module

integrates an HRNet frontend with Fast R-CNN, aiming to detect the 2D bounding box of the target. The RoI extracted is then resized and employed in the second model—a standalone HRNet trained using a mean squared error loss function between the predicted and ground truth heatmaps of the visible landmarks in each image.

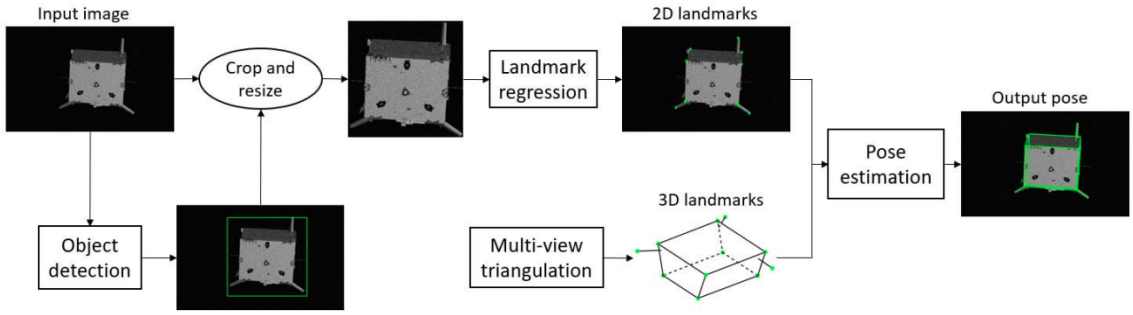


Figure 3.7: Architecture of the pipeline proposen by Bo Chen et al. (originally published in [13]).

Ultimately, this approach amalgamates HRNet, Fast R-CNN, and geometric optimization techniques, presenting a robust framework for precise pose estimation leveraging both DL and geometric methodologies.
The two-stage pose estimation process is attractive and will be used in this work because it modularizes the problem: first, it focus on reliably detecting keypoints and RoI, and then, given these keypoints and RoI, compute the object's pose. This separation allows for the optimization of each stage independently and can lead to more accurate and reliable pose estimations, especially in challenging conditions. Figure 3.8 illustrates the two different DL-based pose estimation approaches.

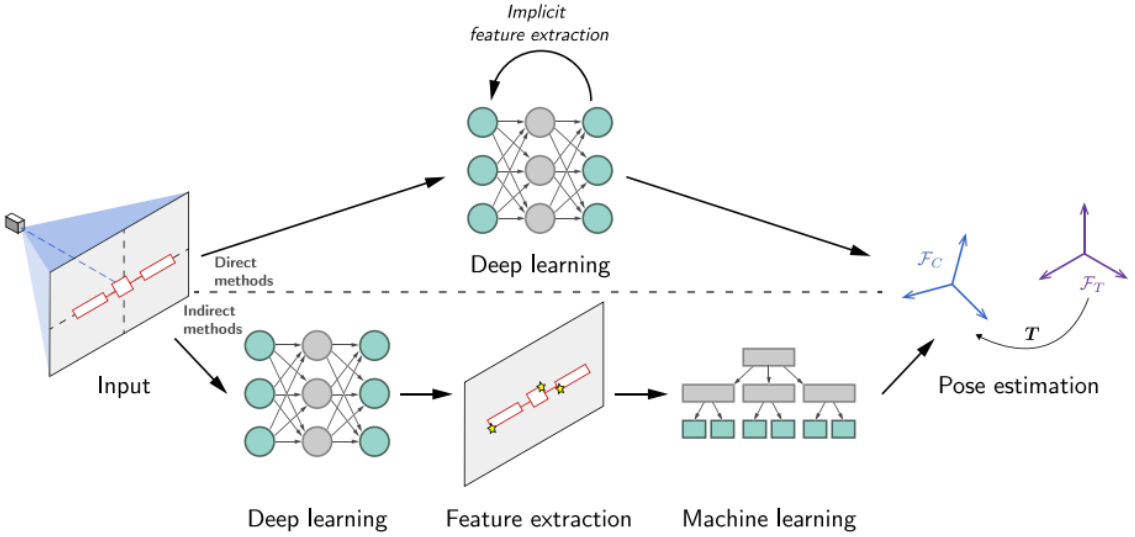


Figure 3.8: Comparing the single-stage and two-stage methods (originally published in [75]).

4. Concept for Pose Estimation

This thesis diverges from direct methodologies, advocating instead for an indirect approach to pose estimation for non-cooperative tumbling objects, specifically emphasizing the Astrobee.

4.1 Overall Architecture

We want to derive the pose of a tumbling object using only image input from monocular camera. The architecture of the system is designed as a multi-stage pipeline, each stage serving a unique but interrelated purpose. This section outlines the three primary steps involved: keypoint detection using YOLOv8, 3D model generation of the target Astrobee and pose estimation with OpenCV's solvePnP function. The schematic architecture of the pipeline is illustrated on figure 4.1.

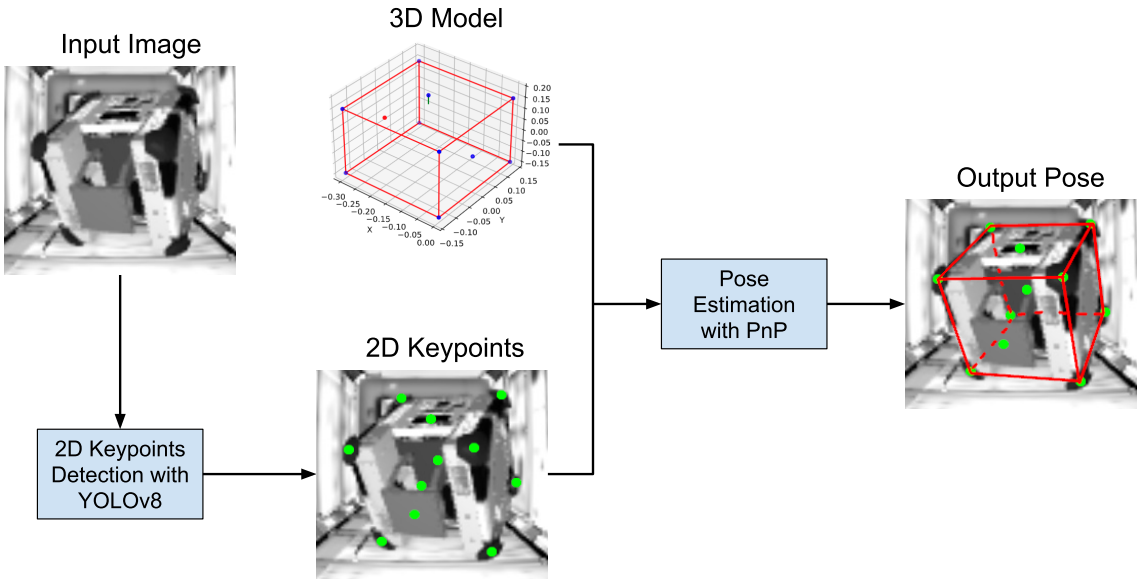


Figure 4.1: Overall pipeline of our Astrobee pose estimator. The input image has been cropped for better visualization.

4.2 Justifying the Selected Pose Estimation Method

In selecting an approach for pose estimation, we have opted for the indirect method after careful consideration of various factors. A primary reason for this choice is

the fundamental nature of the indirect method, which focuses on learning rather than merely memorizing, as highlighted by Song et al. This learning-based approach offers a more robust framework for understanding and predicting poses, in contrast to methods that rely on simple memorization.
Another critical factor influencing our decision is the comparative performance of the indirect method against direct methods. Studies, as referenced in [75], have shown that direct methods often yield subpar performances, especially when compared to geometry-based methods. This difference in efficacy is crucial in contexts where precision and reliability are paramount.
Furthermore, the general consensus in the field, as noted in source [49], is that two-stage approaches, which are characteristic of indirect methods, tend to be more accurate. This increased accuracy is a significant advantage, as pose estimation tasks demand high levels of precision.
Lastly, our choice is also supported by evidence, as cited in source [1], indicating that indirect methods yield better results than direct regression approaches. This superiority in outcomes further validates our decision to employ an indirect approach in our pose estimation tasks.
However, it's important to acknowledge that direct methods have their merits, particularly in terms of computational efficiency and ease of implementation. These methods are generally more straightforward, making them appealing for scenarios where simplicity is prioritized over the highest possible accuracy [29].
In summary, the indirect approach, with its emphasis on learning, superior performance compared to direct methods, general accuracy in two-stage processes, and better results in comparison to direct regression, presents itself as the most suitable methodology for our pose estimation objectives.

4.3 Description of Single Stages

In this section, we provide a detailed description of the individual components and stages that make up the pipeline of our system. Each component plays a crucial role in the overarching architecture, contributing to the pose estimation of the Astrobee.

4.3.1 Automatic Keypoints Detection for Astrobee

This stage involves training a ML model for keypoint detection.
In the process of selecting an optimal framework for the initial stage of our pose estimation task, we conducted a comparative analysis of two leading models: YOLO and Mask R-CNN. The results of this analysis have led us to choose the YOLO model for several compelling reasons.
As illustrated in the table 4.1, YOLO demonstrates a significant advantage in computational efficiency. The mean computational time for YOLO is significantly lower than that for Mask R-CNN, showing YOLO's ability to process images at a faster rate, which is a critical factor in real-time applications where rapid detection is paramount. YOLO's mean processing time across various samples stands at approximately 5.48 milliseconds, compared to Mask R-CNN's 67.63 milliseconds. This stark difference underscores YOLO's superior performance in terms of speed.

The table 4.2 provides a detailed breakdown of the performance of two object detection models across several metrics. True Positives (TP) for 'Head' detection

Sample no.	Mask R-CNN (ms)	Mean	YOLO (ms)	Mean
$n = 1$	85		2	
$n = 2$	57		3	
$n = 3$	71		3.3	
$n = 4$	55		2.9	
$n = 5$	65	67.63215	3	5.48544
$n = 6$	63		2.5	
...	...		...	
...	...		...	
...	...		...	
$n = 100$	73		2	

Table 4.1: Computational time for Mask R-CNN and YOLO [76].

are identical for both models, but YOLO identifies more 'Tail' instances correctly. False Positives (FP) are minimal for both models, with YOLO having an impeccable record in 'Tail' detection. YOLO has a slightly higher total detection count for 'Head' and 'Tail' compared toMask R-CNNs.
Precision over the class is exceptionally high for both models; however, YOLO achieves perfect precision in 'Tail' detection. Recall over the class is consistent for 'Head' detection between the two models, but YOLO shows a notably higher recall for 'Tail' detection. In terms of overall performance, YOLO demonstrates superior Global Precision and Global Recall, indicating a strong balance between accuracy and the ability to identify all relevant instances in the dataset.

Metric	YOLO		Mask R-CNN	
	Head	Tail	Head	Tail
TP	115	76	115	63
FP	2	0	1	5
Total detected	117	76	116	68
Precision over class	98.29	100	99.14	92.65
Recall over class	86.47	73.79	86.47	61.17
Global Precision	98.96		96.73	
Global Recall	80.93		75.43	

Table 4.2: Comparison of YOLO and Mask R-CNN with precision and recall metrics [62].

Choosing the latest iteration, YOLOv8, was informed by these findings, along with the model's good accuracy and its simplicity in terms of implementation and adaptability for custom datasets. YOLOv8's architecture, coupled with the availability of five different pretrained versions, allows for a flexible and streamlined adaptation process, ensuring that the model can be quickly fine-tuned to our specific requirements. These advantages, together with its high speed and the ease of integration into our system, make YOLOv8 the standout choice for keypoint detection in our pose estimation framework.
Each image used for training is paired with a label file containing normalized coordinates of the keypoints and a bounding box that encapsulates the Astrobee.

This bounding box serves as the RoI, directing the model's focus onto the target object.

4.3.2 3D Model Generation of the Astrobee

The 3D model generation stage in our pipeline is a crucial component of our research. During this phase, we utilize TF frames attached to the target Astrobee, as visible in 4.2, each having coordinates relative to the target body center. Initially, we started with a TF frame located at the front upper left corner of the Astrobee, with the initial values set as $x = 0.15$, $y = 0.137$, and $z = -0.148$. To construct a cube structure, these values were methodically altered seven times.
In addition to the primary TF frames, three extra TF frames were integrated, each assigned specific coordinates. The first of these, designated for a display unit, was placed with coordinates $x = 0.1383$, $y = 0$, and $z = -0.005$. The second frame, intended for a flash light, was set at $x = -0.143$, $y = -0.001$, and $z = -0.002$. Finally, a third frame was attached to serve as a docking mechanism, positioned at the origin coordinates $x = 0$, $y = 0$, and $z = 0.185$. The precise positioning of these TF frames is instrumental in the accurate 3D modeling of the Astrobee, ensuring each component is correctly represented relative to the Astrobee's body center.
The reasons for choosing these specific points and their designations will be explained in the next chapter. These frames act as reference points for our 3D model. To establish a consistent reference point, we choose the centroid of the front face of the cube as the origin. This origin serves as the basis for all subsequent operations, providing a uniform coordinate system. By subtracting this origin from every 3D point, we ensure that our calculations and transformations are performed within a standardized spatial framework. This 3D model generation process is fundamental to our goal in this stage, accurately representing the Astrobee in three-dimensional space. The detailed information about the 3D model generation process will be included in subsection 5.2.2.

4.3.3 Pose Estimation using YOLOv8 and PnP

The final stage focuses on estimating the Astrobee's pose using OpenCV's solvePnP function. The function takes several inputs, including the 2D keypoints detected by YOLOv8, the 3D model points, the camera's intrinsic matrix, and distortion coefficients. Additionally, the function accepts a flag parameter to specify the PnP solving method to be used; in our case, we opt for the EPnP algorithm. This flag enables the system to leverage the advantages of EPnP, known for its computational efficiency and robustness. The function provides two vectors alongside a success flag: rvector, which describes the rotation, and tvector, which is the translation of the Astrobee. Following the conversion of rvector into a quaternion, we obtain the desired pose estimation.
The success of the pose estimation is heavily reliant on the preceding steps. Accurate 3D model generation and keypoint detection are essential for obtaining reliable pose estimates. Thus, each stage of the architecture is integral to the functioning of the subsequent stages.

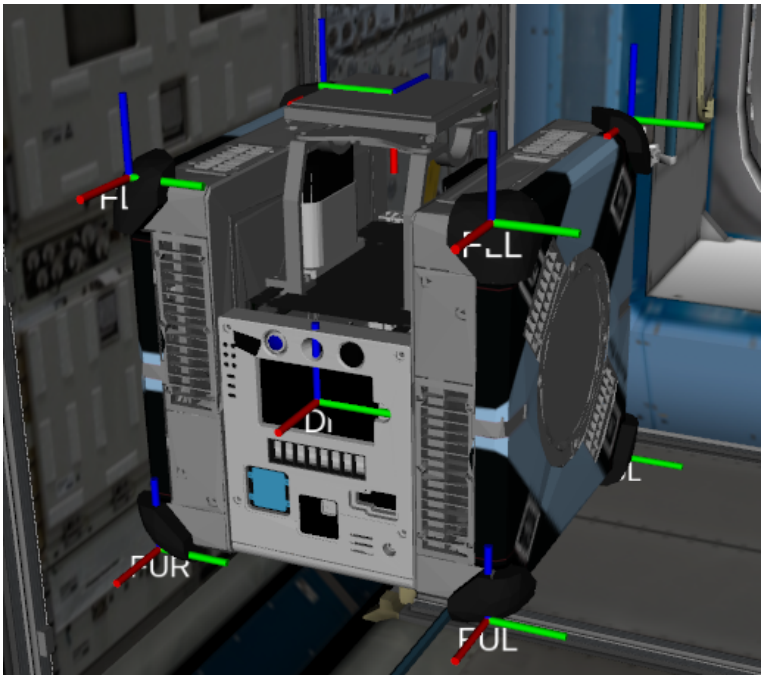


Figure 4.2: 3D visualization of attached TF frames in the ROS simulation.

5. Dataset and Experimental Setup

5.1 Dataset Generation

This section elucidates the methodology employed for the generation of a dataset tailored for YOLOv8-based keypoint detection, 3D model generation and pose estimation.
The initial step involved the selection and designation of key reference points on the target Astrobee. Specifically, 11 keypoints were identified and labeled with corresponding TF in ROS. These keypoints were chosen based on their visibility and representational capacity for the Astrobee. They include eight corners forming a cube, one point on the docking mechanism, another on the display screen, and a final one on the flashlight. The nomenclature for these landmarks is as follows:

- FUL (Front Upper Left)
- FUR (Front Upper Right)
- FLR (Front Lower Right)
- FLL (Front Lower Left)
- BUL (Back Upper Left)
- BUR (Back Upper Right)
- BLR (Back Lower Right)
- BLL (Back Lower Left)
- Di (Display)
- Do (Docking Mechanism)
- FL (Flashlight)

Each TF frame's pose was defined relative to the target Astrobee's primary transform, denoted as *bumble/body*. Subsequently, a conversion process was implemented to map each keypoint TF to the chasing Astrobee's camera frame. This was facilitated through a ROS topic named $/tf\_kp\_pose$, which published the transformations at a rate of 1.0 Hz. This frequency allows for sufficient updates while avoiding an overwhelming volume of transformations, ensuring smoother and more controlled processing.
For synchronization purposes, the ROS message headers were examined to match the timestamps between $Image$ and $TFMessage$ data. After data extraction, the 3D coordinates of the keypoints were projected onto a 2D image frame. This projection

was non-trivial, requiring accounting for the field-of-view (FOV) distortion inherent to the camera model.
The projection function commences by specifying the camera's intrinsic matrix:

$$\text{Intrinsic Matrix} = \begin{bmatrix} 607.64218 & 0.0 & 640 \\ 0.0 & 606.53899 & 480 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

This matrix contains the intrinsic parameters of the camera. The diagonal elements f_x and f_y represent the focal lengths in pixels along the x and y-axes, respectively. The off-diagonal terms are typically zero for simple camera models. The last column contains c_x and c_y, which are the coordinates of the principal point, usually located near the image center and 1.0, which is a scaling factor that makes the matrix a homogeneous coordinate transformation matrix. Homogeneous coordinates are used in computer graphics and CV to handle projective transformations [66]. In this specific case, it is essentially a placeholder that makes the matrix mathematically consistent for transformations.
Subsequently, a distortion coefficient of 0.99872 is defined for the FOV distortion model. The distortion coefficient and intrinsic parameters specific to the Astrobee bumble can be accessed within the Astrobee ROS source files, typically acquired through a camera calibration procedure.
The function takes a 3D point in the camera's coordinate system and normalizes it by dividing its x and y components by its z component. This process removes the effects of the focal length and the principal point offset, converting the point to normalized image coordinates. The radial distance r and the angle θ in the normalized image plane are calculated as $r = \sqrt{x_{\text{norm}}^2 + y_{\text{norm}}^2}$ and $\theta = \arctan(r)$.
After that, the FOV distortion model is applied to θ to obtain θ_d:

$$\theta_d = \frac{1}{k} \arctan\left(2 \tan\left(\frac{k}{2}\right) \tan\left(\frac{\theta}{2}\right)\right)$$

where k is the distortion coefficient [18].
A scaling factor is then calculated as scale $= \frac{\theta_d}{r}$ if $r \neq 0$ or 1.0 otherwise. This scaling factor is applied to obtain the distorted, normalized coordinates $x_{\text{distorted}} = x_{\text{norm}} \times \text{scale}$ and $y_{\text{distorted}} = y_{\text{norm}} \times \text{scale}$. Finally, these coordinates are converted to pixel coordinates using the intrinsic parameters:

$$x = f_x \times x_{\text{distorted}} + c_x$$

$$y = f_y \times y_{\text{distorted}} + c_y$$

In summary, the projection algorithm includes the following steps:

1. Normalize the 3D coordinates by the z-coordinate.
2. Apply the FOV distortion model.
3. Convert the distorted, normalized coordinates to pixel coordinates using the intrinsic parameters.

This algorithm is iteratively applied to each keypoint to generate the x and y pixel coordinates for the 2D projection of the original 3D point.
Bounding boxes were calculated as well, with the purpose of confining the area of interest for the YOLOv8 object detection algorithm. The function for bounding box calculates a square bounding box around a set of 2D keypoints. It starts by extracting the x and y coordinates from the given keypoints. From these coordinates, the function identifies the minimum and maximum values for both x and y to calculate the dimensions of the smallest rectangle that can enclose all the keypoints. However, the function aims to create a square bounding box, so it uses the larger of the rectangle's width and height as the side length of the square.
To give some flexibility and ensure that the keypoints are well-enclosed, a margin is added around this square. The margin is defined by the optional parameter, which specifies how much extra space to add around the square, expressed as a percentage of its original size. The function then calculates the center of this square bounding box using the average of the minimum and maximum x and y coordinates.
Finally, the function returns the center coordinates along with the adjusted side length, effectively describing a square bounding box that encloses the original keypoints with some optional margin for flexibility.
Despite rigorous fine-tuning, which includes modifying the intrinsic matrix, distortion coefficient, and θ_d for optimizing projection accuracy, the methodology exhibits limitations. Particularly, when the target object deviates from the center of the image frame, the pose estimation becomes less robust, emphasizing the importance of centering the object in the camera frame.
Figure 5.1 illustrates an example image with accurately overlaid keypoints and a bounding box from the dataset. In contrast, Figure 5.2 shows an example image with keypoints and a bounding box from the dataset, where the keypoints are not accurately overlaid.

5.1.1 Images and Labels

For our dataset generation, images were captured using the navigation camera of the Astrobee. Once the ROS was initiated, both the chaser and target Astrobees were positioned within the JEM. The target Astrobee was situated at a fixed distance of 1 meter from the chaser. Using the ROS bagfile format to record ROS message data for future use, we initiated recording by executing the command *rosbag record /honey/hw/cam_nav /tf_kp_pose*. This recording captured the publishing of images and poses of the target while it started tumbling at the designated distance.
The tumbling began with a rotation around the x-axis, commencing at 0 degrees and increasing by approximately 10 degrees every 2 seconds, covering a full 360-degree rotation. This process was then repeated for rotations around the y and z axes. Afterward, the distance was increased by 0.5 meters, and the rotation procedure was reiterated. Once the distance reached 2 meters, the x-axis was set to 30 degrees,

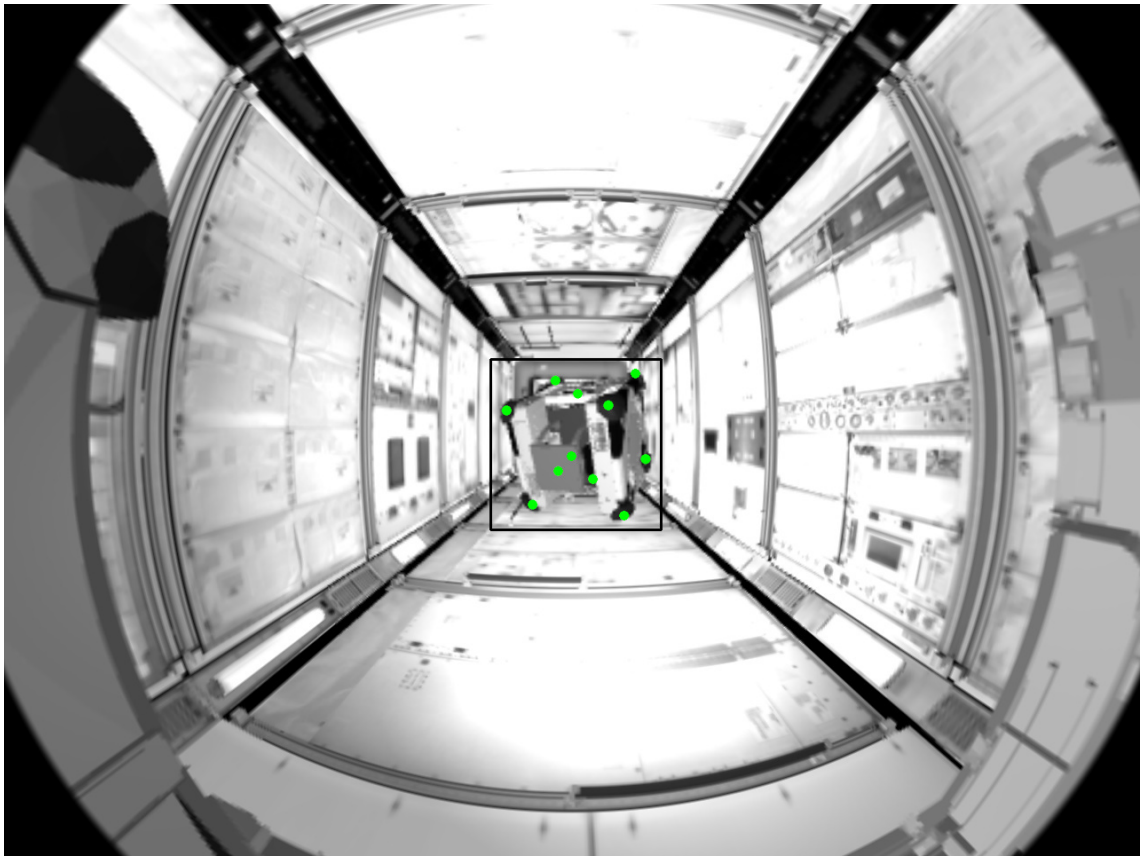


Figure 5.1: An example image from the dataset shows keypoints and a bounding box (black square around the Astrobee). The image reveals 7 corners and the docking mechanism attached to the Astrobee.

and the y-axis rotation was initiated, followed by the z-axis. The same protocol was carried out, shifting the distance to 1.5 meters and then to 1 meter, with angle variations.
At certain stages, random adjustments were made to angle and distance values. These modifications included slight alterations to translation values, causing the target to shift slightly up/down or left/right by approximately 0.3 meters. After about an hour of continuous recording, the process was halted, and the bagfile, containing images along with the target's poses, was saved. The bagfile also contained images that displayed overlaid ground truth keypoints and bounding boxes, which were saved using our topic: $/tf\_kp\_pose$.
Following this, the bagfile was played in ROS, enabling data extraction from the topics $/honey/hw/cam\_nav$ and $/tf\_kp\_pose$. To ensure accuracy in the 3D to 2D projected keypoints, images with imprecise projections were identified by observing the images with overlays. Subsequently, corresponding image and pose files without accurate projections were eliminated, resulting in a final dataset, albeit smaller in size, containing some outliers like images with imprecise projections. Each image in our dataset is approximately 245 kilobytes in file size. A detailed breakdown of the format and specifications of these images is as follows:

1. **Camera**: Astrobee Navigation Camera.

Figure 5.2: Example image showing keypoints and a bounding box with less accurate overlay.

2. **Resolution and Image Size**:

   - **Width**: 1280 pixels
   - **Height**: 960 pixels

   This resolution ensures a balance between image detail and computational efficiency, providing sufficient granularity for accurate pose estimation without overly taxing the processing resources.

3. **Color Format**: The images are in grayscale. By utilizing grayscale, we prioritize the structural and intensity information of the objects, reducing data dimensions and computational requirements, while retaining key features essential for pose estimation.

4. **Capture Rate**: The camera captured images at a rate of 1.0 Hz. This frequency ensures dynamic object movements can be effectively tracked over time, while avoiding data redundancy from capturing too frequently.

5. **File Format**: The captured images were saved in joint photographic experts group (JPEG) format. JPEG was chosen because it offers a good trade-off between image quality and file size, enabling the storage of large datasets without compromising much on visual detail.

By standardizing our dataset with these image specifications, we aimed to maintain consistency and reliability in our subsequent analyses and experiments, ensuring that variations in results stem primarily from algorithmic differences rather than data discrepancies. Figure 5.3 illustrates an example image from the dataset.

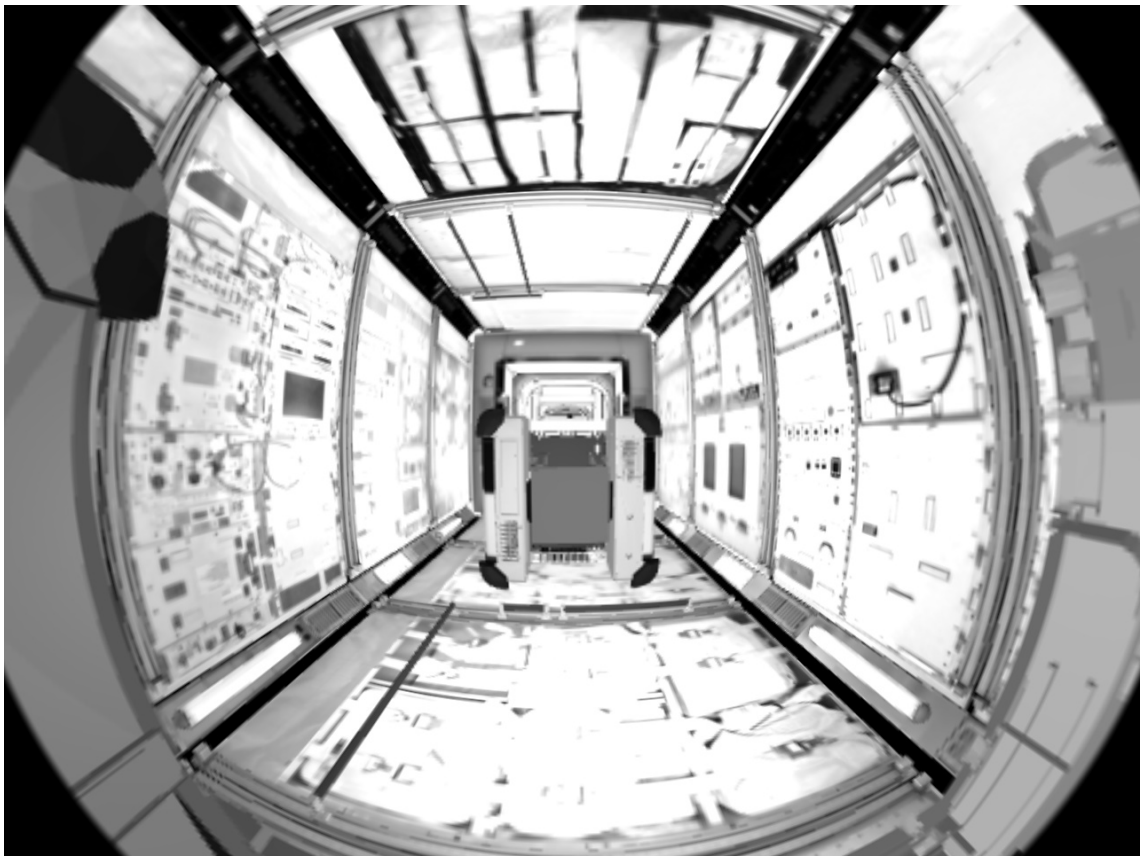


Figure 5.3: Example image from the dataset used for training.

The subsequent phase involved preparing labels for the images in the YOLOv8 format. Initially, labels containing keypoint and bounding box coordinates were stored in a .json file. To adapt these coordinates to the YOLOv8 format, adjustments were made. Each keypoint's x-coordinate was divided by the image's width, and the y-coordinate was divided by the image's height. This normalization process also extended to the bounding box or RoI coordinates.
The YOLOv8 label format followed a specific structure: it included class identification, the bounding box's normalized center coordinates, width, and height. Additionally, the normalized coordinates of the keypoints were listed, each followed by a '2' indicating visibility (for simplicity, '2' was uniformly assigned to denote visibility for all keypoints).

5.1.2 Limitations and Dataset Summary

Despite exhaustive attempts to fine-tune the FOV distortion formula, the projection process exhibited limited accuracy when the target object was not centrally located

in the image frame. This constraint necessitated a dataset predominantly comprising images where the target Astrobee is centered. The dataset consists of 3,814 images, captured at distances of 1 to 2 meters and at various angular orientations. For training and validation purposes, the dataset was split into subsets consisting of 80% and 20% of the total data, resulting in 3,051 training images and 763 validation images for the keypoints detection model. This partitioning strategy, allocating 80% for training and reserving 20% for validation or testing, is a commonly practiced approach in model development [24].

5.2 Experimental Setup

This section elucidates the systematic approach undertaken to develop and evaluate the keypoints detection model and subsequent 3D model construction. It provides a comprehensive overview of the technologies and parameters employed throughout the experimentation phase, ensuring a thorough understanding of the procedural intricacies involved. It serves as a foundation for replicating the experimental conditions and understanding the underlying mechanics of the developed models.

5.2.1 Keypoints Detection Model and Object Detection

YOLO has been instrumental in the field of object detection, and the YOLOv8 version is its latest iteration, endowed with enhanced capabilities and improvements. In the outlined steps, a scientific workflow is described, primarily utilizing Google Colab Pro version and the Ultralytics YOLOv8, the cutting-edge, SOTA model for training a DL model on our dataset. Below is a descriptive step-by-step breakdown:

1. Setup and Data Preparation
    - Google Colab: A platform provided by Google that offers a free-to-use GPU, which is highly beneficial for training DL models.
    - Mounting Google Drive: The user's Google Drive was mounted to Google Colab. This is essential for utilizing files (such as training data, models, or configuration files) stored on the Google Drive in Colab notebooks.
    - Data Upload: Training data/files were uploaded into Google Drive. These files are crucial for training the ML model and include images and labels.
2. Installing Required Libraries
    - Ultralytics YOLO: A popular open-source software library used for object detection tasks.
    - Version 8.0.196 Installation: The specific version 8.0.196 of Ultralytics was installed using pip (a package installer for Python).
3. Model Loading and Configuration
    - Model Importing: Utilized Ultralytics to import the YOLOv8 model.
    - Pretrained Model Loading: A pretrained YOLOv8 model was loaded from the user's Google Drive.

- Configuration Path: The path for the configuration file (config.yaml)—essential for defining parameters, path setups, and settings during training—is specified.

A pre-trained model, YOLOv8x-pose-p6, was employed as the foundation for transfer learning, leveraging its learned patterns and knowledge for our specific task. This model is known for its extensive architecture, comprising 510 layers. It has a total of 99,176,808 parameters, indicative of its complexity and capacity for learning intricate features and representations.
Data augmentation is pivotal in the enhancement of the model's generalization ability, ensuring it can perform accurately on unseen data. The following data augmentation techniques were employed to vary and enrich the training dataset:

1. **Blur**: Applied with a probability of 0.01, with blur limit ranging between 3 and 7.
2. **MedianBlur**: Incorporated with a 0.01 probability, and a blur limit between 3 and 7.
3. **Contrast limited adaptive histogram equalization (CLAHE)** : This was used with a 0.01 probability, a clip limit between 1 and 4.0, and a tile grid size of 8x8.

The training process was configured with specific parameters to optimize the learning process. The model was trained for 100 epochs, with a batch size of 8 and an image size of 1280. The rect=True parameter signifies that rectangular training was applied. SGD was chosen as the optimizer, with a momentum of 0.937 and a weight decay of 0.0005. These parameters were meticulously chosen to regulate the update of the model weights during training and to prevent overfitting, ensuring the model can generalize well to unseen data.
The training was conducted on Google Colab Pro version, utilizing the powerful A100 GPU. Despite the extensive architecture and complexity of the model, the training process concluded in approximately 4 hours, demonstrating the efficiency and capability of the utilized hardware.
Figure 5.4 showcases multiple graphs detailing the training and validation losses and 5.5 demonstrates performance metrics during training of a YOLOv8x-pose-p6 model.

1. Training Losses:
    - Box Loss (train/box_loss): Started around 1.2 and declined sharply to settle just below 0.4.
    - Pose Loss (train/pose_loss): Began around 8 and saw a consistent decrease, with a smoothed curve indicating a settling point slightly above 0.
    - Object Loss (train/kobj_loss): Commenced at approximately 0.5 and steadily reduced, approaching close to 0.
    - Class Loss (train/cls_loss): Began slightly under 1 and showed a clear decline, leveling off near 0.3.

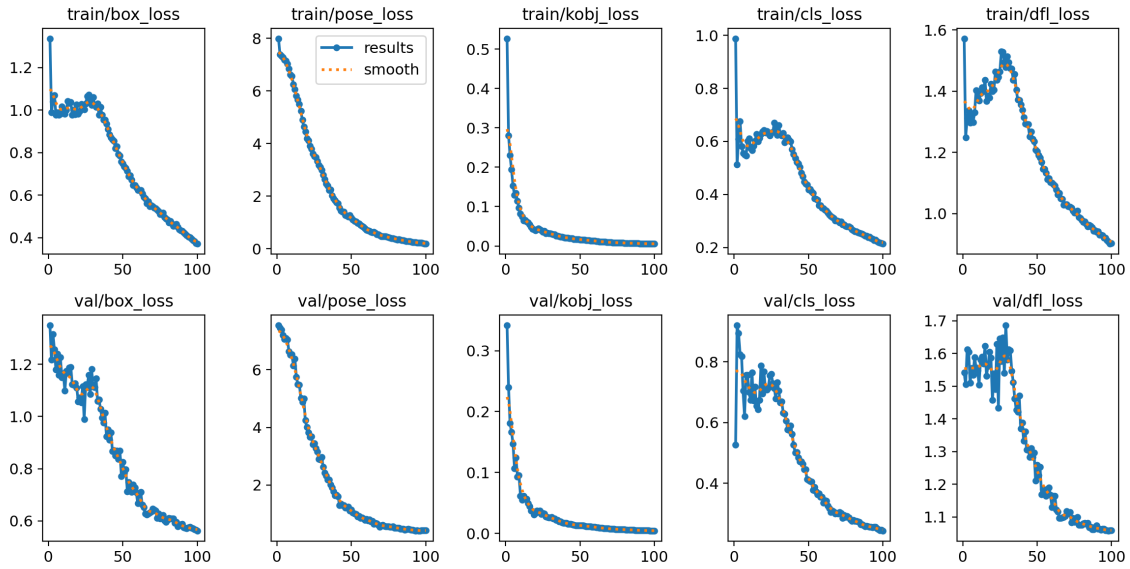


Figure 5.4: Metrics visualizing the training and validation progression of a YOLOv8x-pose-p6 model for keypoint and bounding box detection.

- DF Loss (train/df_loss): Demonstrated high fluctuation in the beginning but gradually stabilized near the 0.4 mark.

2. Validation Losses:

- Box Loss (val/box_loss): Displayed a sharp decline from around 1.2, showing minor fluctuations as it approached the 0.5 range.
- Pose Loss (val/pose_loss): Started near 6 and descended consistently, leveling off just above 0.
- Object Loss (val/kobj_loss): Presented a similar pattern to its training counterpart, stabilizing close to 0.
- Class Loss (val/cls_loss): Saw a considerable amount of variance but seemed to settle around 0.6.
- DF Loss (val/df_loss): Expressed high volatility throughout, with values scattered between 1.1 and 1.7.

3. Performance Metrics:

- Precision (B) (metrics/precision(B)): Indicated a fluctuating pattern before stabilizing around the 0.975 mark.
- Recall (B) (metrics/recall(B)): Started high and maintained its performance, settling close to 1.
- Precision (P) (metrics/precision(P)): Demonstrated consistent performance, staying close to 1 throughout.
- Recall (P) (metrics/recall(P)): Mirrored the precision (P) graph, maintaining a stable high performance near 1.

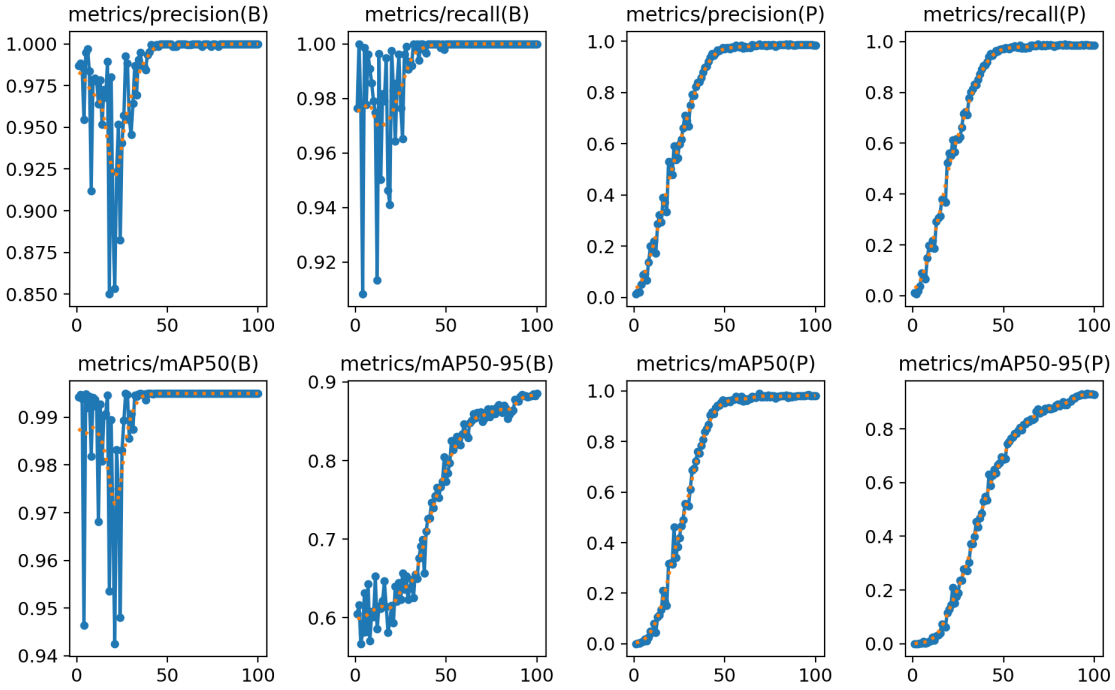


Figure 5.5: Multiple graphs detailing the performance metrics and losses during the training and validation of a YOLOv8x-pose-p6 model. B stands for Box, P is for Pose.

- mAP50 (B) (metrics/mAP50(B)): Saw fluctuations initially but began stabilizing close to 0.99.
- mAP50-95 (B) (metrics/mAP50-95(B)): Started off high and maintained a steady increase, getting closer to 0.9.
- mAP50 (P) (metrics/mAP50(P)): Kept a steady performance throughout, leveling off near 1.
- mAP50-95 (P) (metrics/mAP50-95(P)): Displayed a smooth and consistent rise, approaching the 0.9 range.

Metric	Value
Average Inference Time per Image (seconds)	0.26363
RMSE (pixels)	38.26115
Number of Images with Detected Objects (out of 163)	127

Table 5.1: Model performance metrics for YOLOv8x-pose-p6 model on a testset.

In a manner similar to section 5.1.1, we created a test set where the target Astrobee tumbled randomly within a range of 2 meters from the chaser, maintaining a minimum distance of 1 meter. This set comprised 163 images used to evaluate the model's performance during inference. Table 5.1 shows the achieved test metric results for the YOLOv8x-pose-p6 model.
After achieving an average inference time per image of 0.26363 seconds and an RMSE value of 38.26115 pixels using the pretrained YOLOv8x-pose-p6 model, in order

to reduce the inference time and increase the keypoint detection model accuracy, we developed an additional dataset containing 480 images and opted for another available YOLOv8 model with less parameters. The new dataset in addition captures the target Astrobee in a static position while the chaser orbits around it, aiming to keep the target centered in the frame. The chaser was manually commanded to fly around the stationary target. Waypoints were chosen randomly with a spacing of approximately 10 cm, all while ensuring the chaser remained within a 2-meter range of the target during the fly-around. The chaser's orientation was adjusted at each waypoint to keep the target centered in the camera's field of view.
The next pretrained YOLOv8l-pose model, with 44.4 million parameters, was trained using our new extended dataset and produced the test values represented in Table 5.2. From the previous model, it's evident that the model begins to converge around epoch 80. This time, we trained the model for 80 epochs, keeping all other parameters constant.
Figure 5.6 showcases graphs detailing the training and validation losses and 5.7 demonstrates performance metrics during training of a YOLOv8l-pose model.

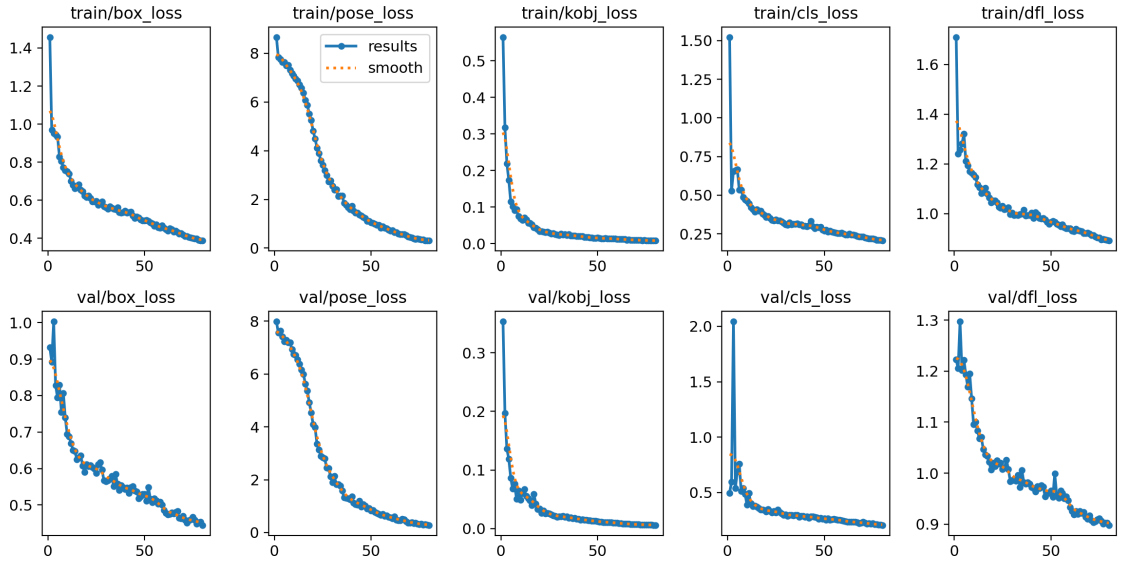


Figure 5.6: Metrics visualizing the training and validation progression of a YOLOv8l-pose model for keypoint and bounding box detection.

1. Training Losses:
    - Box Loss (train/box_loss): Rapid initial decrease followed by steady, slower decline.
    - Pose Loss (train/pose_loss): Sharp initial decrease, continues to decrease at a reduced rate.
    - Object Loss (train/kobj_loss): Steep decline at the start, then levels off indicating quick initial learning.

- Class Loss (train/cls_loss): Quick drop initially, then a gradual decline showing ongoing improvement.
- DF Loss (train/df_loss): Sharp decrease initially, leveling out as training progresses.

2. Validation Losses:

- Box Loss (val/box_loss): Similar to training, with a rapid decrease then a steady decline.
- Pose Loss (val/pose_loss): Decreases sharply at first, then at a slower rate, mirroring training loss.
- Object Loss (val/kobj_loss): Sharp initial decline, flattening out indicating stabilization.
- Class Loss (val/cls_loss): Fast initial decrease, then slows down indicating consistent learning.
- DF Loss (val/df_loss): Quick initial improvement, then gradual decreases indicating good generalization.

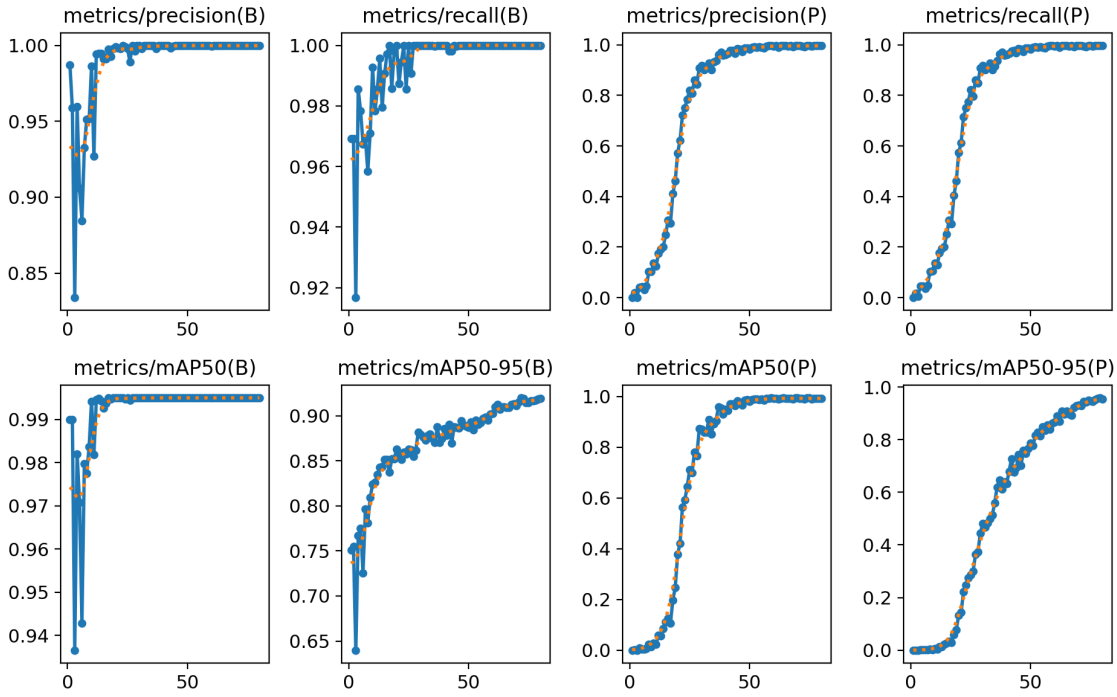


Figure 5.7: Multiple graphs detailing the performance metrics and losses during the training and validation of a YOLOv8l-pose model.

3. Performance Metrics:

- Precision (B) (metrics/precision(B)): Rapidly reaches high values and stabilizes close to 1.

- Recall (B) (metrics/recall(B)): Increases quickly to near-perfect levels and plateaus close to 1.
- Precision (P) (metrics/precision(P)): Sharply increases to high values and levels off near 1.
- Recall (P) (metrics/recall(P)): Shows a steady climb to a plateau near the maximum value.
- mAP50 (B) (metrics/mAP50(B)): Increases sharply and then more gradually to a value close to 1.
- mAP50-95 (B) (metrics/mAP50-95(B)): Gradually increases, with a more moderate ascent, reaching a high value.
- mAP50 (P) (metrics/mAP50(P)): Rises quickly to a high plateau, indicating good performance.
- mAP50-95 (P) (metrics/mAP50-95(P)): Shows a steady increase and eventually plateaus at a high value.

Metric	Value
Average Inference Time per Image (seconds)	0.18532
RMSE	23.55553
Number of Images with Detected Objects (out of 163)	161

Table 5.2: Model performance metrics for YOLOv8l-pose model on a testset.

The performance of the YOLOv8l-pose model showcases improvements over the YOLOv8x-pose-p6 model. Notably, the YOLOv8l-pose model exhibits a significantly reduced average inference time per image, clocking in at 0.18532 seconds compared to 0.26363 seconds for the YOLOv8x-pose-p6 model. Additionally, the YOLOv8l-pose model demonstrates a notably lower RMSE of 23.55553 pixels compared to 38.26115 pixels of the YOLOv8x-pose-p6 model, indicating improved accuracy in keypoint detection. Moreover, the YOLOv8l-pose model detects objects in a higher number of images, detecting objects in 161 out of 163 images compared to 127 for the YOLOv8x-pose-p6 model. These comparative metrics underscore the advancements and enhancements achieved with the YOLOv8l-pose model in terms of faster inference, higher accuracy, and improved object detection capabilities. So we chose the YOLOv8l-pose for our keypoint detection part.
Table 5.3 defines the train losses and table 5.4 defines the train metrics utilized in evaluating the performance of both models.

The results indicate a well-trained model that is capable of effective keypoint and bounding box detections. With the specified data augmentations and training parameters, the model embodies a meticulous approach towards achieving high accuracy and reliability in keypoint detection tasks. Transfer learning from a pre-trained YOLOv8l-pose model provided a robust starting point, allowing for the utilization of pre-learned features and representations. The selected data augmentations and training parameters were chosen to enhance the model's ability to generalize, optimizing its performance on unseen data.

Loss	Description
Box Loss	Localization accuracy loss for predicted bounding boxes.
Pose Loss	Loss when estimating object's pose, including orientation.
Object Loss	Misclassification loss for the object as a whole.
Class Loss	Error-based loss for specific object category classification.
DF Loss	Difficulty-based loss accounting for object detection complexity.

Table 5.3: Description of losses in YOLO model.

Metric	Description
Precision	Correctly predicted positive instances over total predicted positives.
Recall	Correctly predicted positive instances over actual positive instances.
mAP50	Mean Average Precision with intersection over union (IoU) threshold at 0.5.
mAP50-95	Mean Average Precision across IoU thresholds from 0.5 to 0.95.

Table 5.4: Description of metrics in YOLO model.

5.2.2 3D Model Reconstruction

Each of the 11 previously attached TF frames acted as reference points, collectively contributing to the construction of the model.
To establish a consistent reference point for the model, the centroid of the front face of the cube was chosen as the origin. This selection was underpinned by considerations of symmetry and precision, as the front face of the cube offered a well-defined and easily recognizable feature. By designating this centroid as the origin, a uniform coordinate system was established for all subsequent operations for pose estimation.
In preparation for 3D modeling, a crucial step involved subtracting this newly established origin from every 3D point point. This transformation allowed us to standardize the coordinate system, with the origin now representing a point of reference common to all data points. This subtraction process ensured that all subsequent calculations and transformations were executed within a consistent and coherent spatial framework. Visualization of the 3D model can be seen in figure 5.8.
In summary, the generation of the 3D model of the Astrobee commenced with the strategic attachment of TF frames to the target Astrobee. These frames, characterized by their relative coordinates, served as the foundational points for the model. Furthermore, by designating the centroid of the front face of the cube as the origin and subsequently subtracting this origin from each 3D point, we established a standardized and uniform coordinate system, facilitating the creation of an accurate and comprehensive 3D model of the target Astrobee.

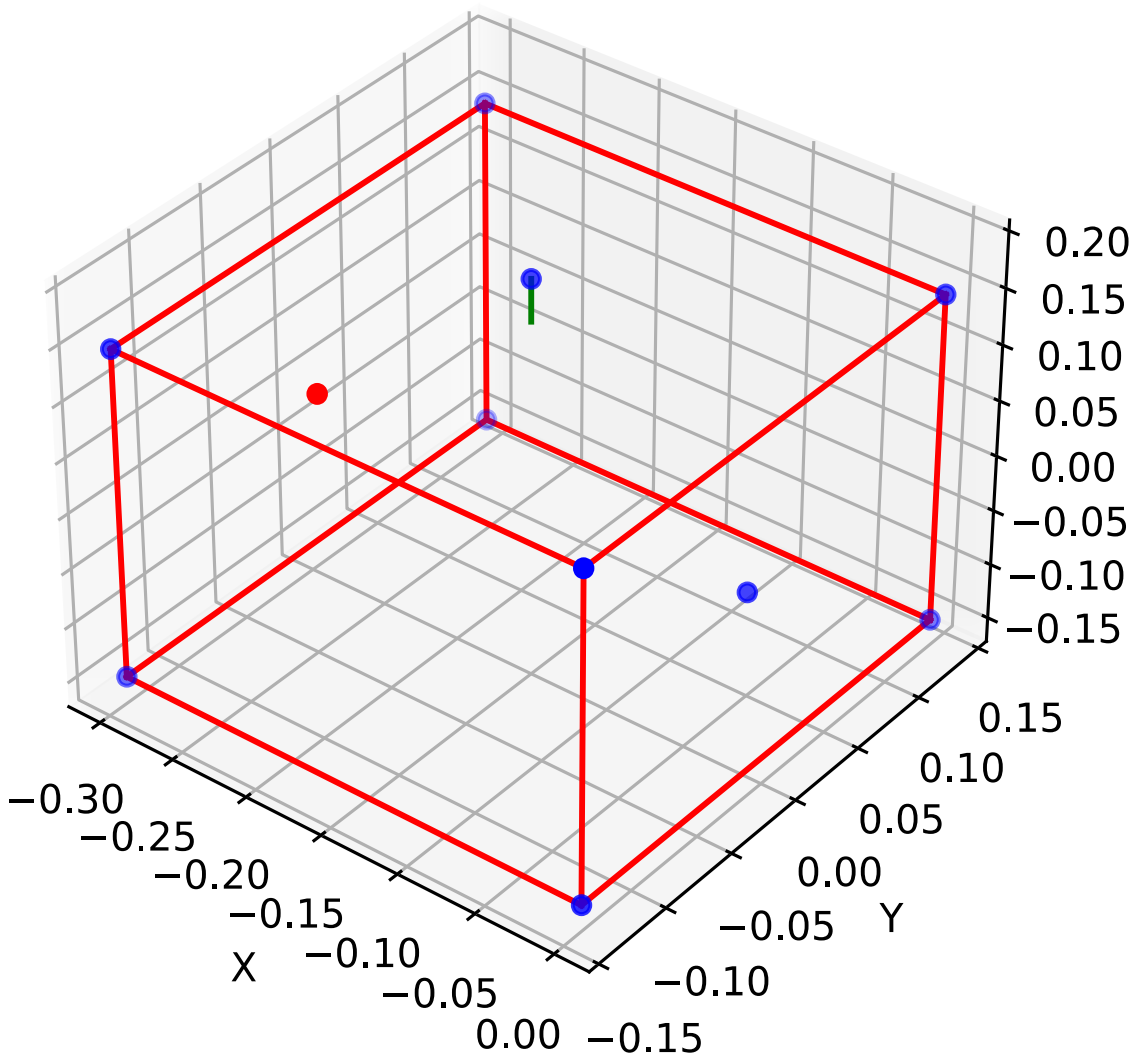


Figure 5.8: 3D model of the target Astrobee. The red dot represents the flashlight on the back side of the Astrobee, the green line with a blue tip symbolizes the docking mechanism, and the blue dot within the front square panel signifies the display.

6. Evaluation

6.1 Pose Estimation

To assess the performance of our pose estimation model, we conducted an evaluation on a testset comprising 307 images, each featuring the Astrobee robot at varying angles and distances. Out of the 307 images processed, the YOLO model successfully detected the Astrobee robot in 289 of them. This evaluation aimed to measure the accuracy and reliability of our pose estimation model within the ROS simulation, which attempts to simulate real-world conditions to some extent. The inference process for the 307 images was completed in 51 seconds on a NVIDIA GeForce GTX 1060 6GB.
To gauge the precision of our pose estimation model, we calculated the Euclidean distances between the predicted quaternion and the ground truth quaternion. The mean Euclidean distance in this context was found to be 0.13657, serving as a measure of the model's performance in estimating the object's orientation in three-dimensional space.
Similarly, we assessed the accuracy of the model's translation predictions by computing the mean Euclidean distance between the ground truth and predicted translation vectors, which yielded a value of 0.99769 meters. This metric provides a measure of the model's ability to estimate the spatial position of the Astrobee.
Table 6.1 presents a summary of the performance evaluation conducted on our pose estimation model.

Metric	Values
Successful detections (out of 307)	289
Inference time (seconds)	51
Mean Euclidean distance (orientation)	0.13657
Mean Euclidean distance (translation in meters)	0.99769
Quaternion error standard deviation	0.08053
Translation error standard deviation (meters)	0.15902
Mean angle error in radians	0.19483
Angle error standard deviation in radians	0.11694
Mean angle error in degrees	11.16346
Angle error standard deviation in degrees	6.70033

Table 6.1: The table captures key metrics, including successful detection rates, inference speed, and the model's ability to estimate both orientation and translation.

For a visual representation of the evaluation results, please refer to the following figures:

Plot 6.1: This plot provides a graphical representation of the errors (Euclidean distances) for each image in the testset. It illustrates the quaternion errors, mean error and standard deviation offering a fine-grained view of the model's performance in estimating orientation for individual images. The presence of the high peak on the plot can be attributed to inaccuracies in the keypoints detection model.
Plots 6.2, 6.3, 6.4 and 6.5: In these four plots, we visualize the components of ground truth quaternions (x, y, z, w) alongside the predicted quaternion components (x, y, z, w).

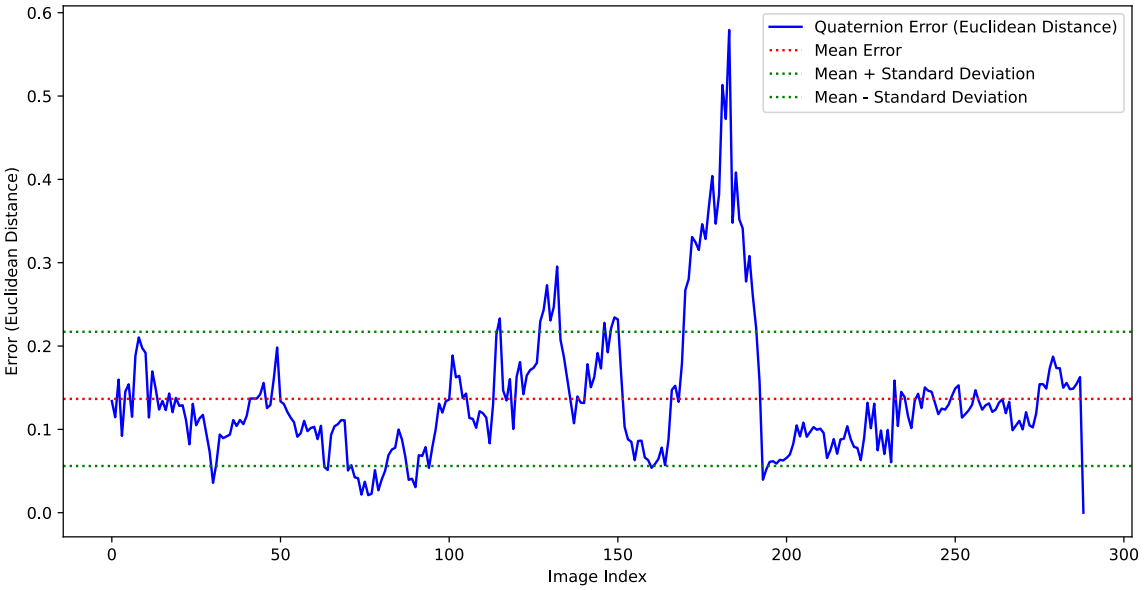


Figure 6.1: Quaternion error representation.

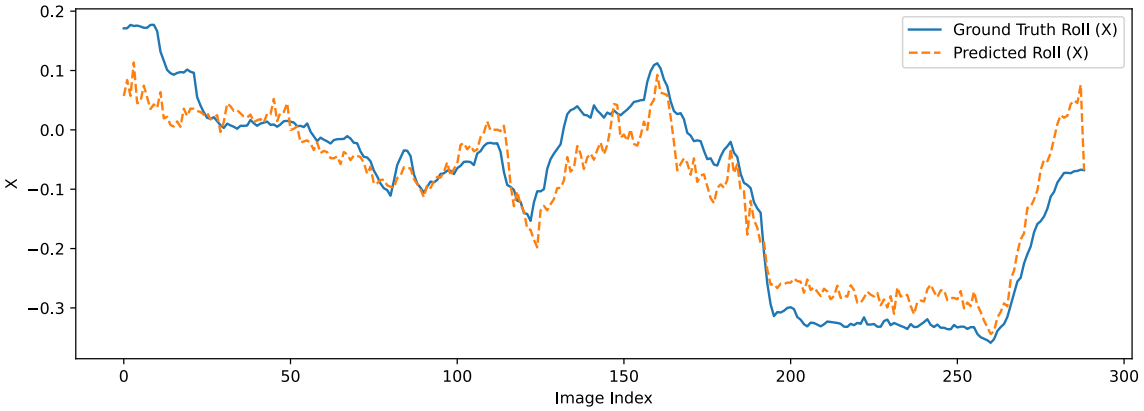


Figure 6.2: Ground truth x component of the quaternion vs predicted x value.

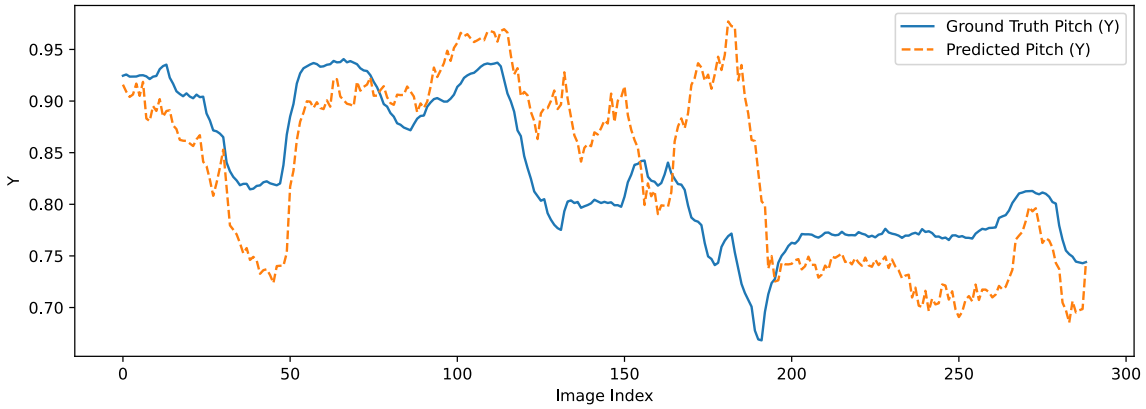


Figure 6.3: Ground truth y component of the quaternion vs predicted y value.

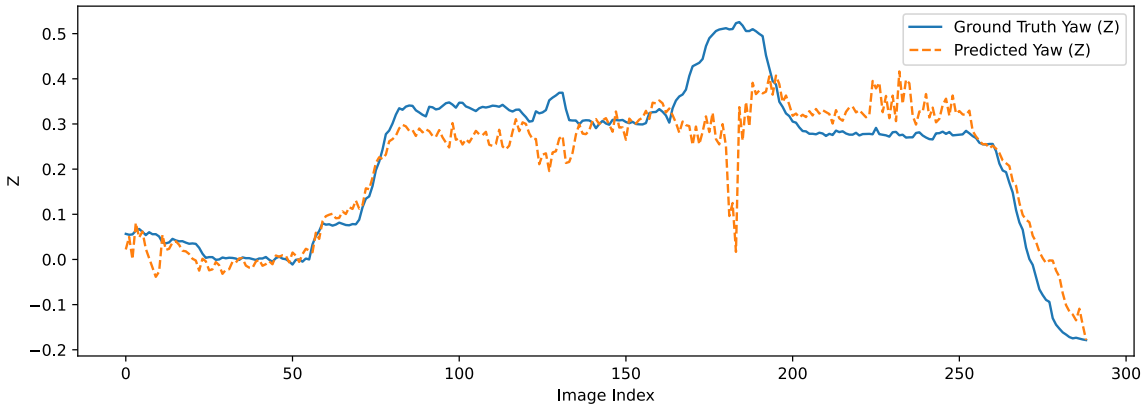


Figure 6.4: Ground truth z component of the quaternion vs predicted z value.

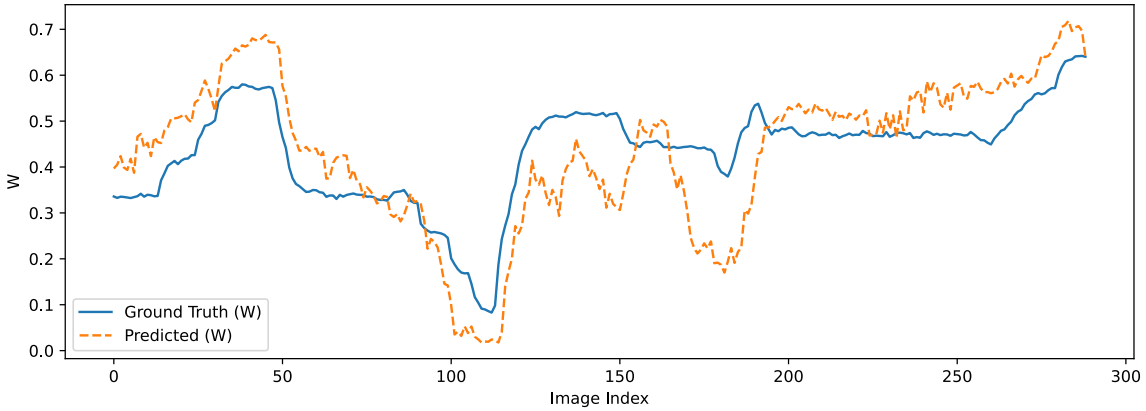


Figure 6.5: Ground truth w component of the quaternion vs predicted w value.

Plots 6.6 through 6.9 present the quantitative errors for each quaternion component. They provide a detailed examination of how well the model estimates each component of the quaternion.

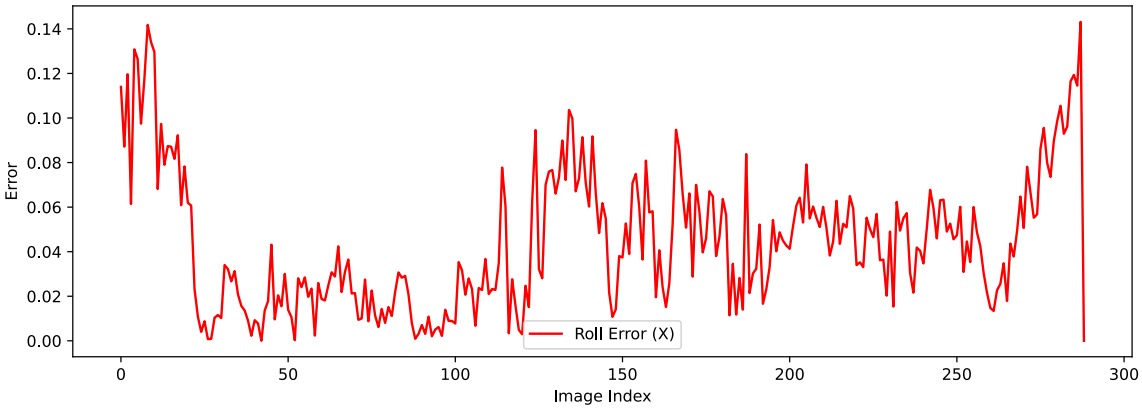


Figure 6.6: Quantitative error analysis of quaternion x component.

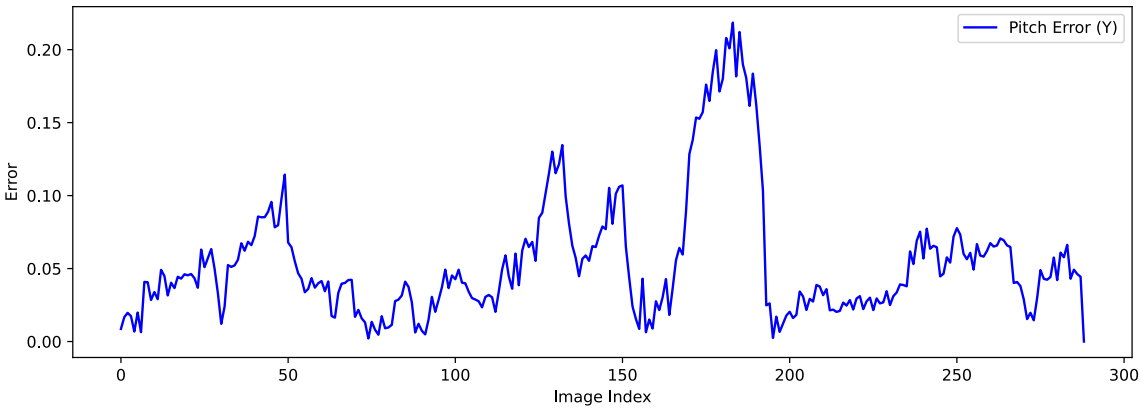


Figure 6.7: Quantitative error analysis of quaternion y component.

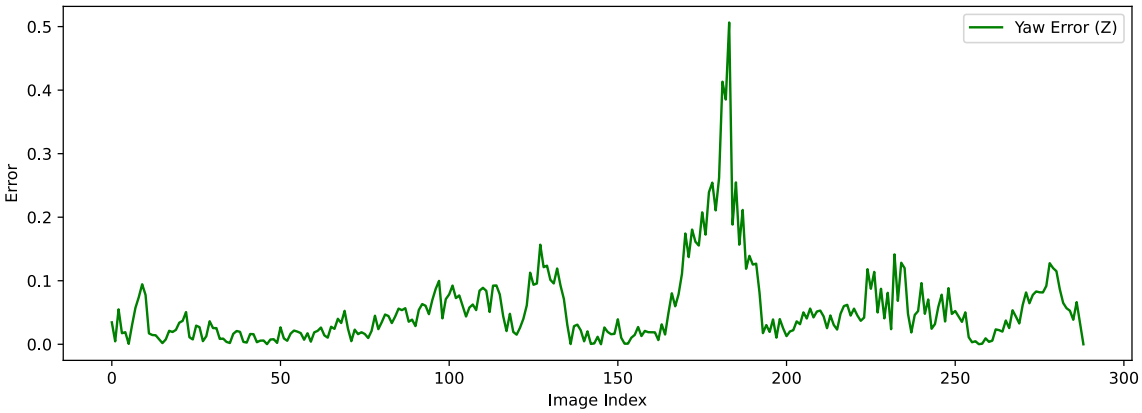


Figure 6.8: Quantitative error analysis of quaternion z component.

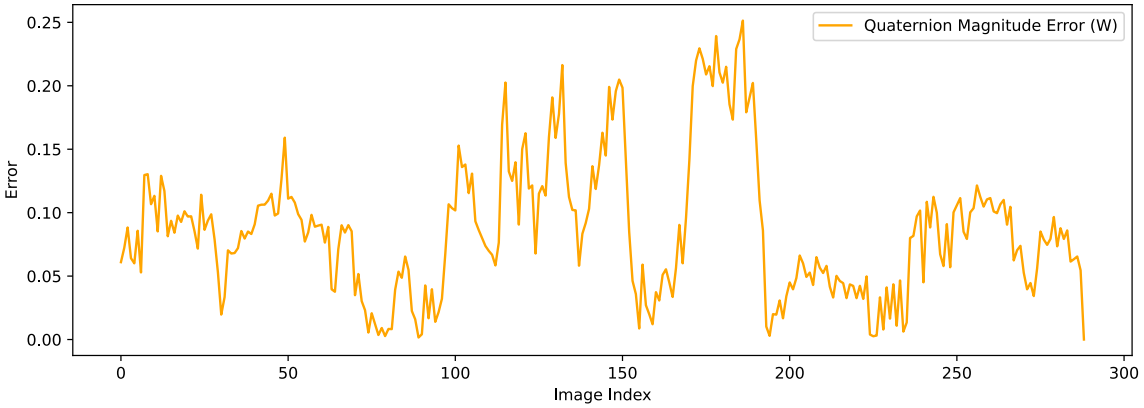


Figure 6.9: Quantitative error analysis of quaternion w component.

In our comprehensive evaluation of the pose estimation model, we have extended our analysis beyond the traditional quaternion errors. To provide a more intuitive understanding of the rotational discrepancies captured by our model, we present two additional plots that illustrate the angle errors. These plots are crucial for a holistic view of the model's performance.
Plot 6.10 offers a detailed visualization of the angle errors in radians. The plot showcases the angle error for each image in the testset, alongside the mean error and the standard deviation.

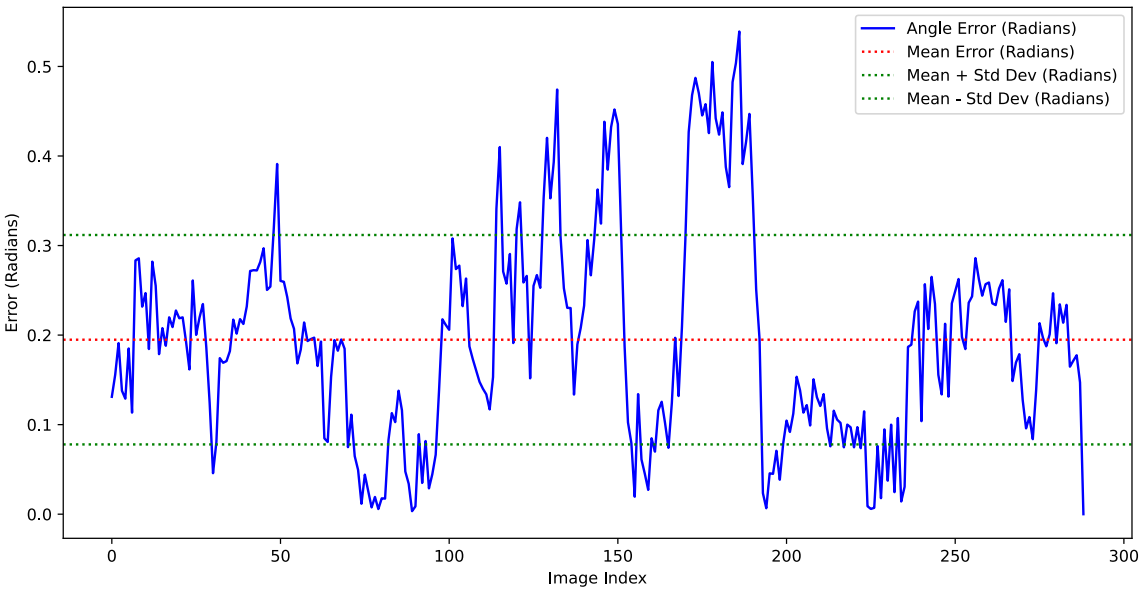


Figure 6.10: Overall angle error in radians.

Plot 6.11, complementing the radian plot, represents the angle errors in degrees. Converting the error measurement to degrees provides an alternative perspective that is often more relatable and easier to interpret, especially for those more accustomed to degree measurements in rotational contexts. Similar to the radian plot, this graph includes the mean error and standard deviation, offering a clear depiction of the model's consistency and reliability in angle estimation.

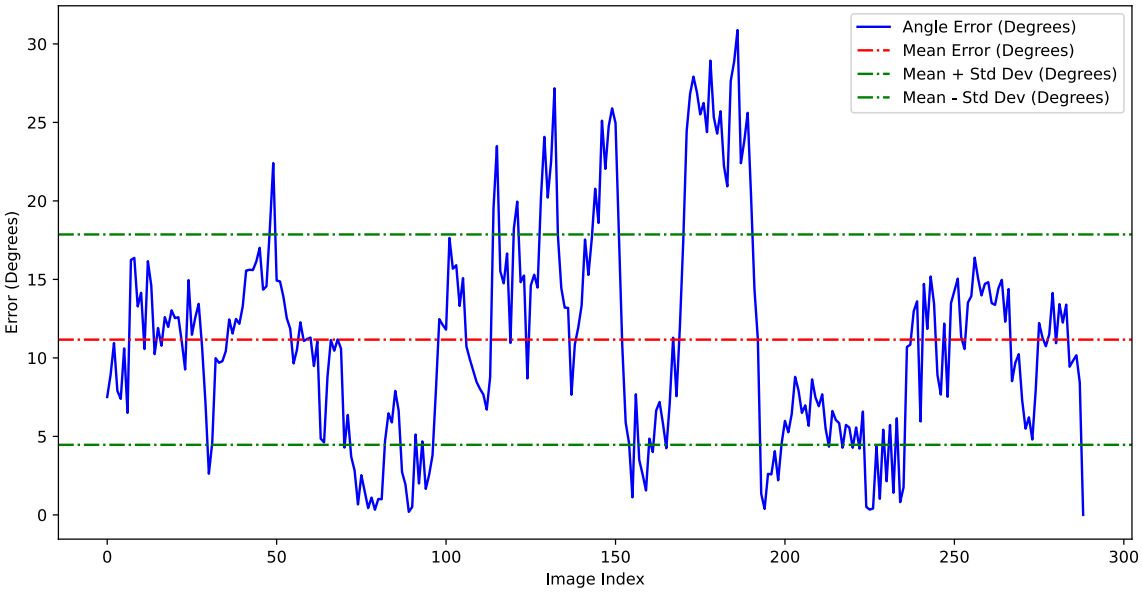


Figure 6.11: Overall angle error in degrees.

Plot 6.12: Similarly, this plot offers a visual representation of the translation for each image. Plots 6.13, 6.14 and 6.15: In these three plots, we visualize the components of ground truth translations (x, y, z) alongside the predicted translation components (x, y, z).

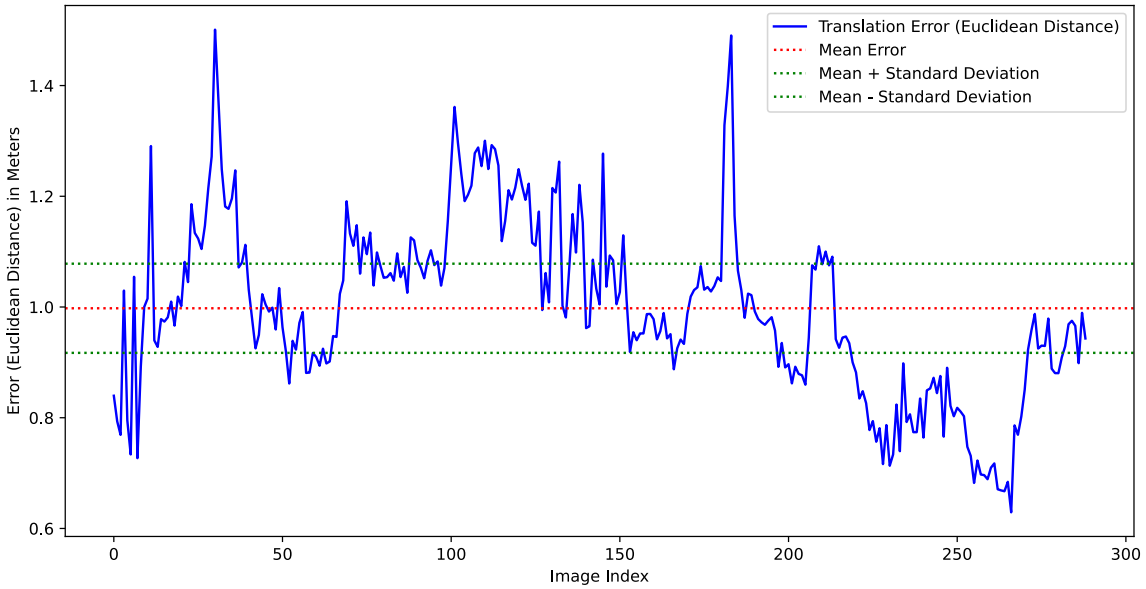


Figure 6.12: Translatioin error representation.

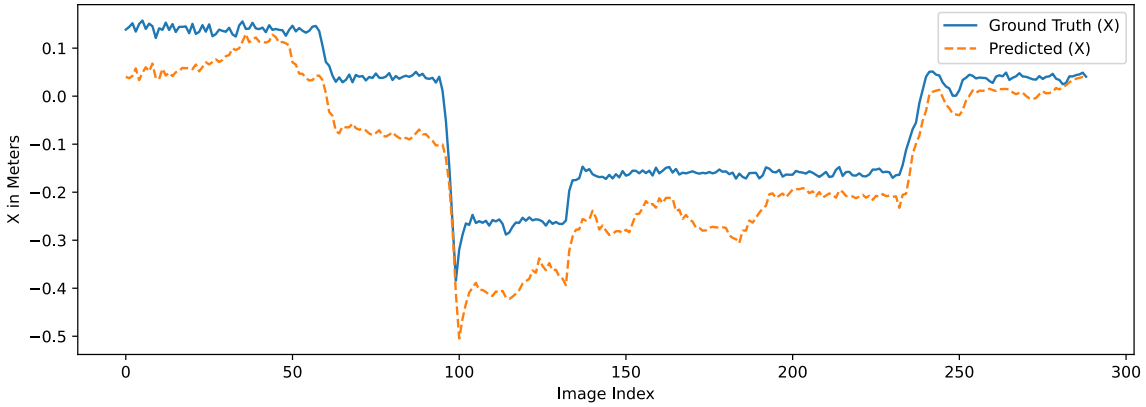


Figure 6.13: Ground truth x component of the translation vs predicted x value.

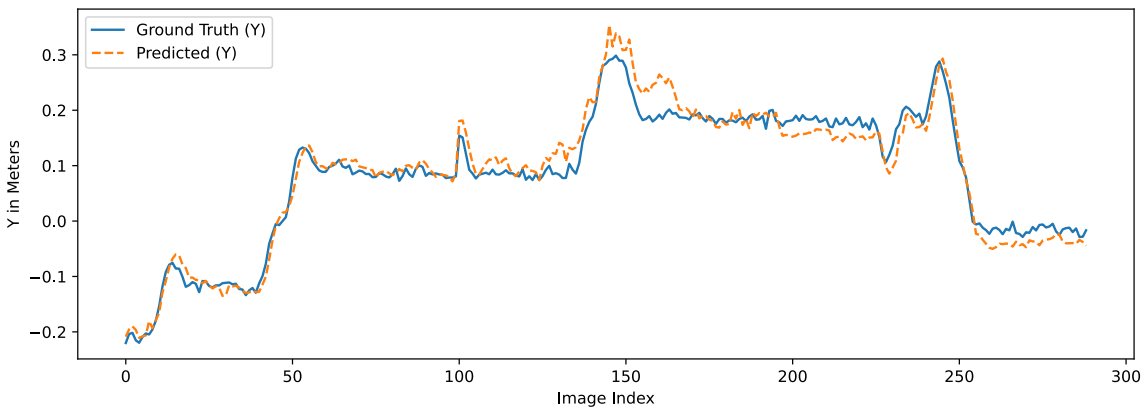


Figure 6.14: Ground truth y component of the translation vs predicted y value.

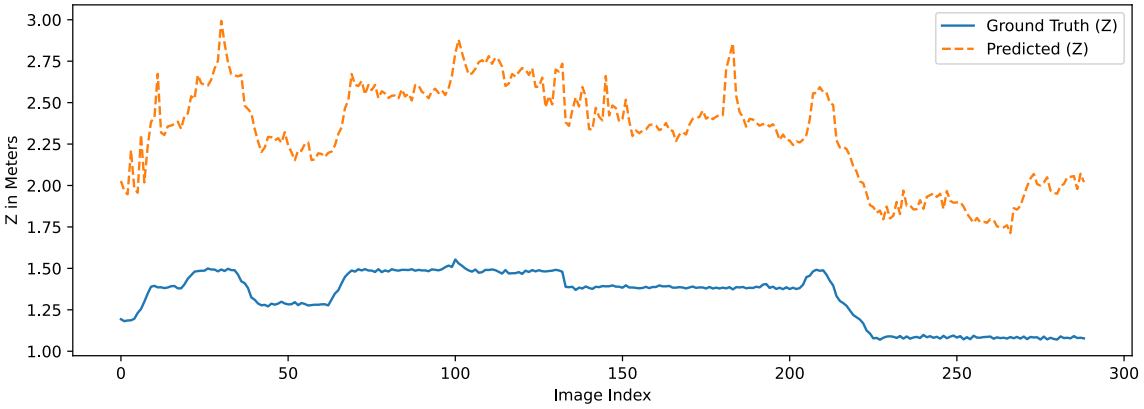


Figure 6.15: Ground truth z component of the translation vs predicted z value.

Plots 6.16 through 6.18 present the quantitative errors for each translation component. These plots provide a detailed examination of how well the model estimates each component of the translation.

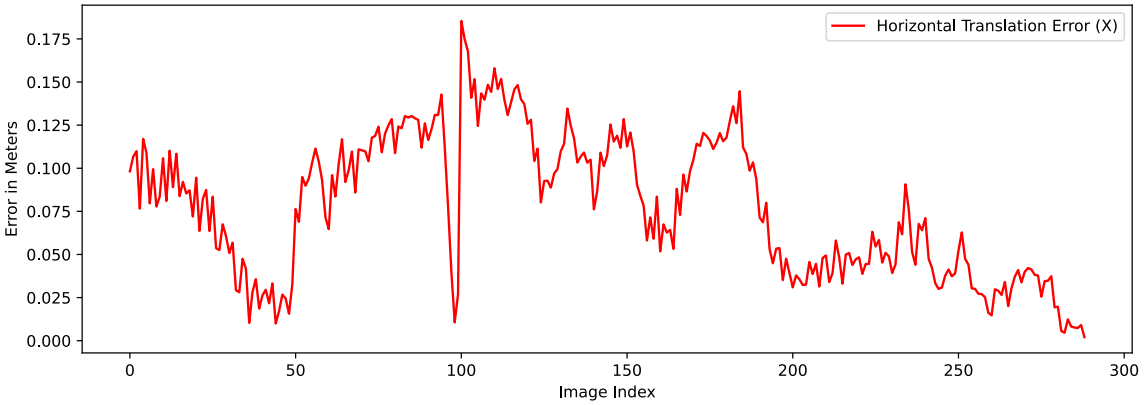


Figure 6.16: Quantitative error analysis of translation x component.

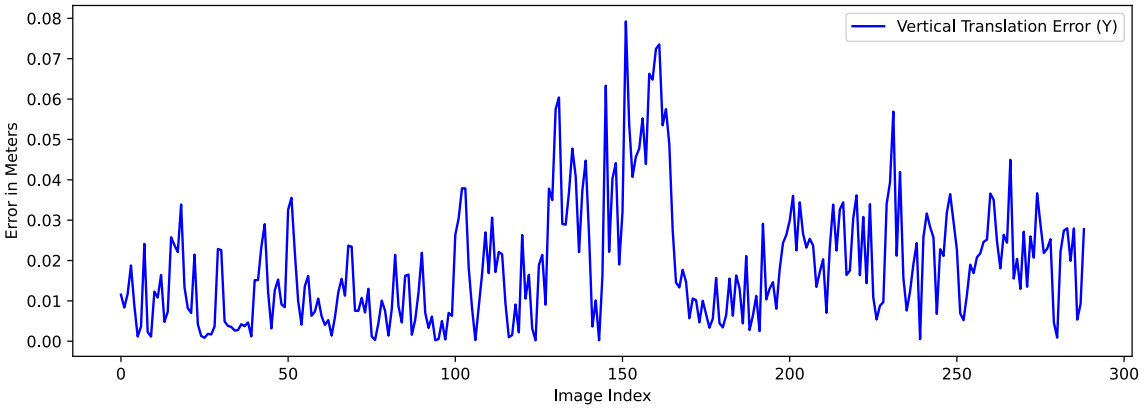


Figure 6.17: Quantitative error analysis of translation y component.

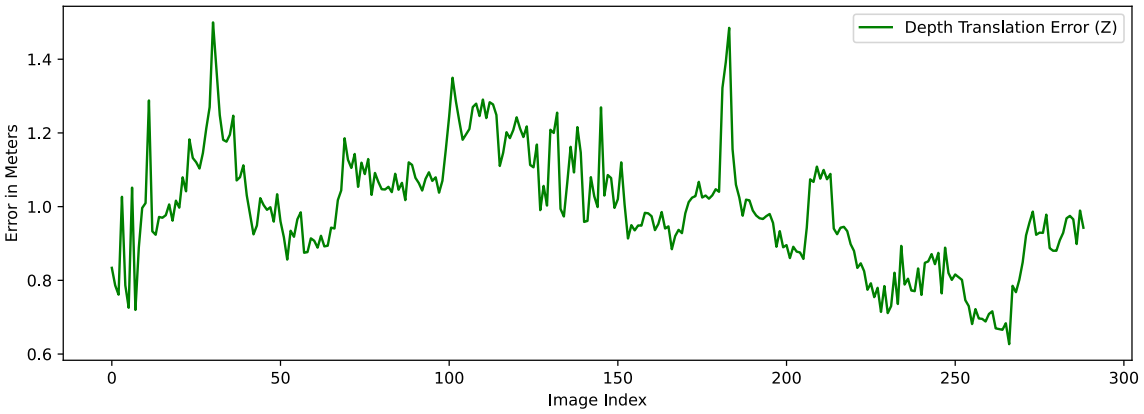


Figure 6.18: Quantitative error analysis of translation z component.

These plots are instrumental in our analysis, shedding light on the model's capabilities and limitations in capturing rotational movements. By examining the errors in both radians and degrees, we gain a comprehensive understanding of the model's accuracy in different units of angular measurement, ensuring a robust evaluation of its performance in diverse scenarios.
After a thorough analysis of the quaternion plots, it becomes evident that the predicted quaternion components exhibit variations in relation to the ground truth values throughout the predictions. This observation suggests that these variations cannot be directly corrected. The accuracy of orientation estimation is closely tied to the precise localization of the detected keypoints within the images.
Moving on to the analysis of the predicted and ground truth translation components, it is apparent that nearly all of the predicted x values are consistently lower than the corresponding ground truth x values. This discrepancy implies that introducing a systematic offset, such as 0.1 meters, to the x component during inference could potentially enhance the mean accuracy of these values. Additionally, a noticeable

trend is observed in which all predicted z values surpass the ground truth z component values. This finding offers the opportunity for direct adjustments to improve the accuracy of these values.
In a previous section 5.1, we addressed the need to fine-tune the principal point values within the intrinsic matrix to enhance the accuracy of 3D-to-2D projections. These projections involve the transformation of 3D points in the camera frame to their 2D counterparts on the image plane. The existing values provided by the Astrobee ROS simulation for the camera are evidently not precise enough, highlighting the necessity for camera calibration. Given this insight, it is reasonable to anticipate that adjusting the focal length values, which govern the distance between the camera lens and the image plane, will have a direct impact on the accuracy of the z values in the translation data.
After meticulous adjustments to the values of the intrinsic matrix, now represented as:

$$\text{Intrinsic Matrix} = \begin{bmatrix} 375.64218 & 0.0 & 640 \\ 0.0 & 375.53899 & 480 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

and the inclusion of a 0.1 meters offset added to the x component and subtracted from the z component of the translation vector, we observed the outcomes summarized in table 6.2.

Metric	Values
Successful detections (out of 307)	289
Inference time (seconds)	51
Mean Euclidean distance (orientation)	0.12985
Mean Euclidean distance (translation in meters)	0.07708
Quaternion error standard deviation	0.08299
Translation error standard deviation (meters)	0.04525
Mean angle error in radians	0.17660
Angle error standard deviation in radians	0.10430
Mean angle error in degrees	10.11869
Angle error standard deviation in degrees	5.97620

Table 6.2: The table captures key metrics, including successful detection rates, inference speed, orientation and translation estimation with modified camera intrinsic parameters, and an included offset for x and z components of the translation.

Plot 6.19: Again, this plot provides a graphical representation of the quaternion errors (Euclidean distances) for each image in the testset, mean error and standard deviation.
Plots 6.20, 6.21, 6.22 and 6.23 visualize the components of ground truth quaternions (x, y, z, w) alongside the predicted quaternion components (x, y, z, w).

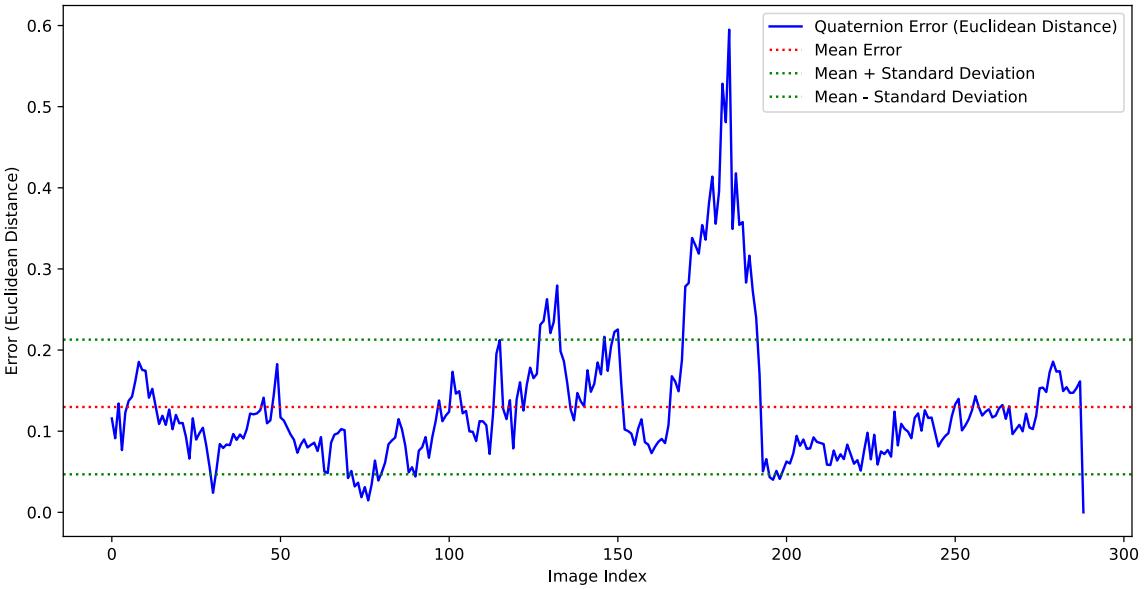


Figure 6.19: Quaternion error representation with adjusted parameters.

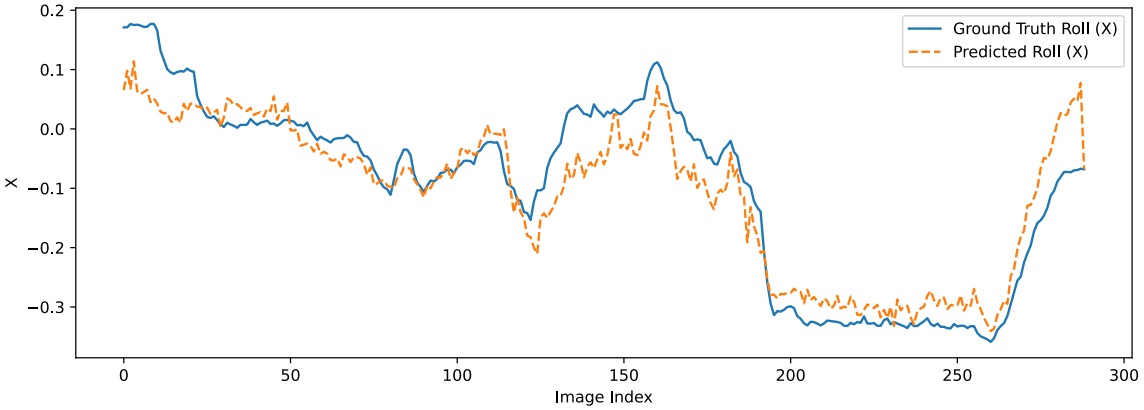


Figure 6.20: Ground truth x component of the quaternion vs predicted x value with adjusted parameters.

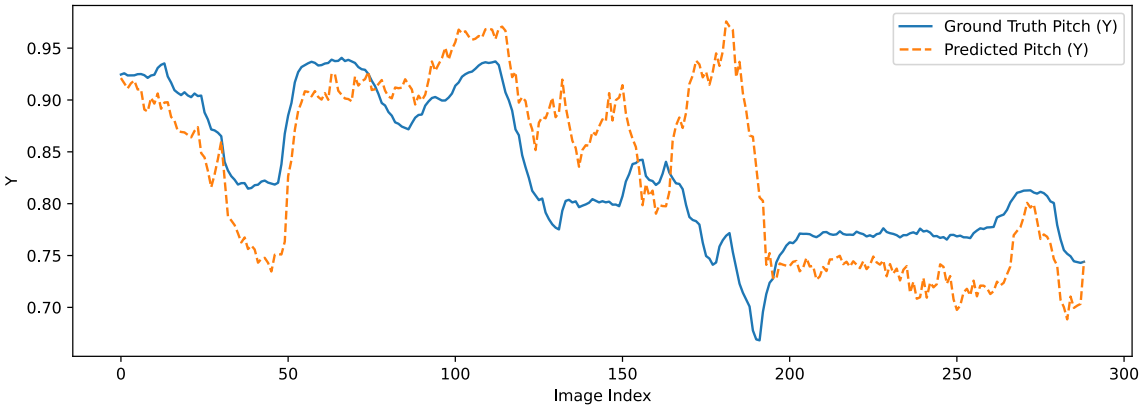


Figure 6.21: Ground truth y component of the quaternion vs predicted y value with adjusted parameters.

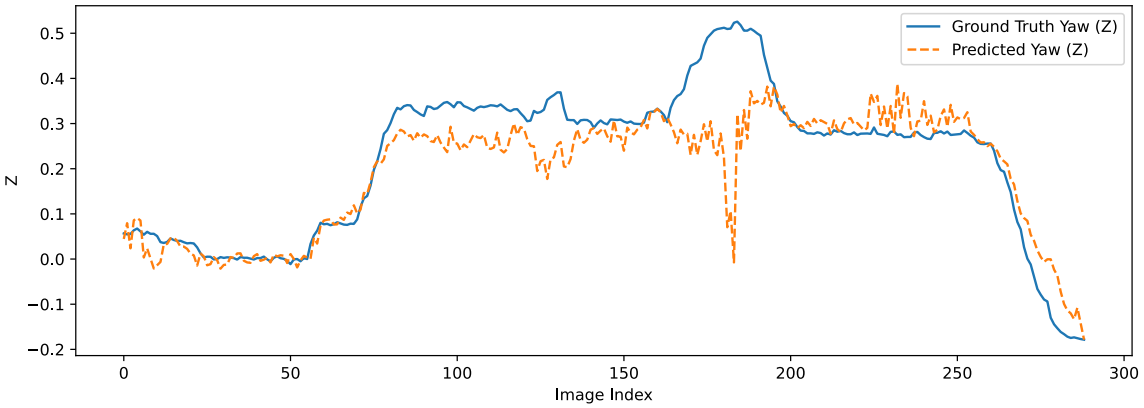


Figure 6.22: Ground truth z component of the quaternion vs predicted z value with adjusted parameters.

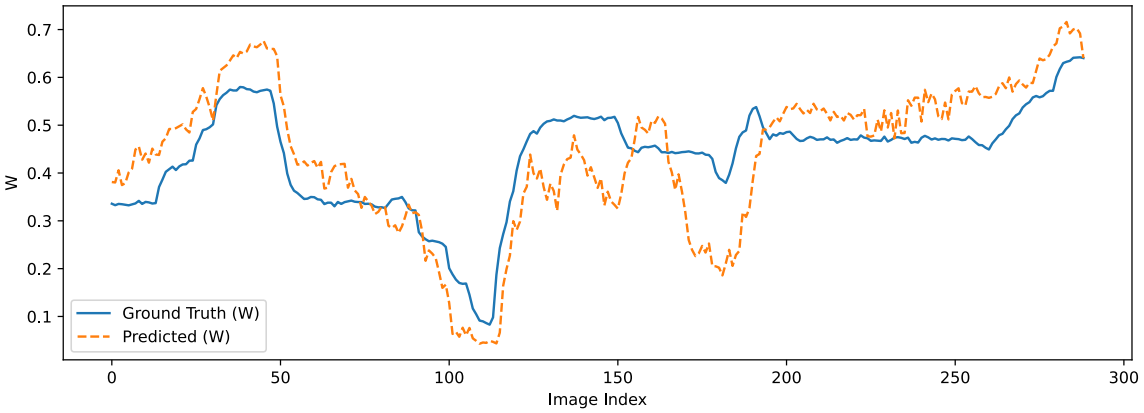


Figure 6.23: Ground truth w component of the quaternion vs predicted w value with adjusted parameters.

Plots 6.24 through 6.27 present the quantitative errors for each quaternion component. These plots provide a detailed examination of how well the model estimates each component of the quaternion.

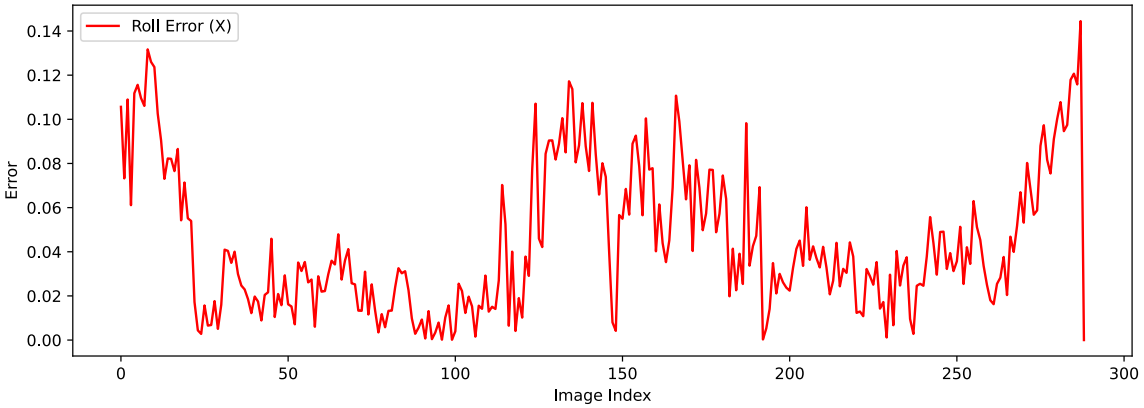


Figure 6.24: Quantitative error analysis of quaternion x component with adjusted parameters.

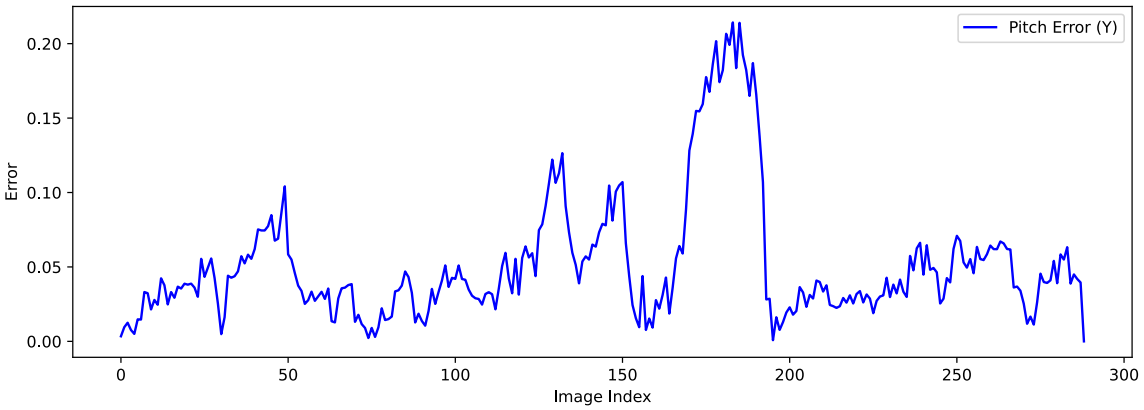


Figure 6.25: Quantitative error analysis of quaternion y component with adjusted parameters.

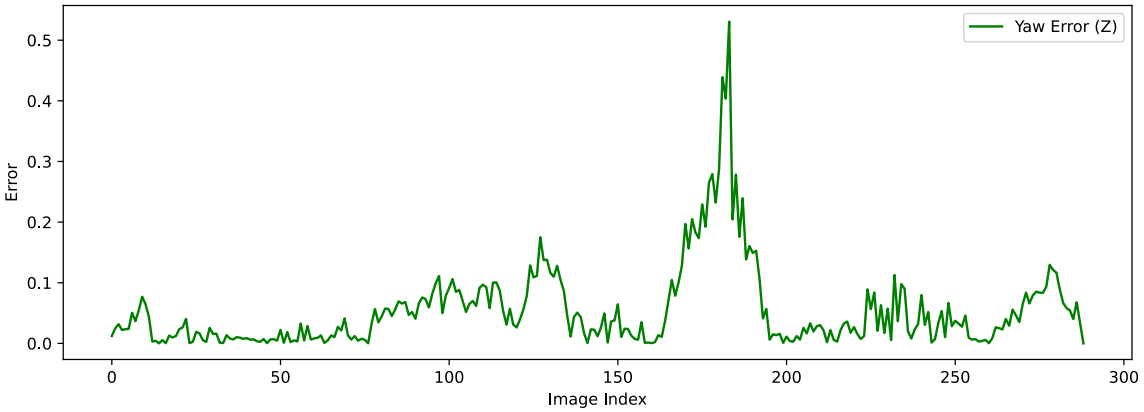


Figure 6.26: Quantitative error analysis of quaternion z component with adjusted parameters.

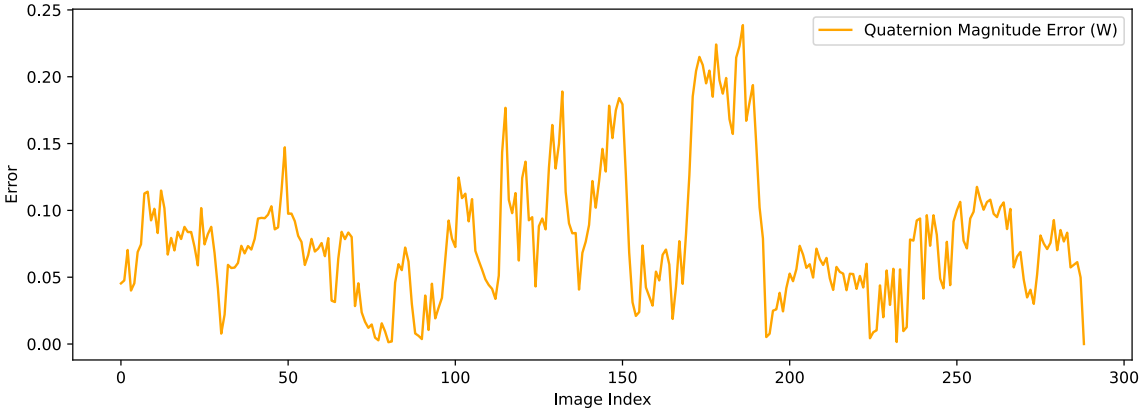


Figure 6.27: Quantitative error analysis of quaternion w component with adjusted parameters.

The two plots, 6.28 and 6.29 represent the angle in radians and degree respectively.

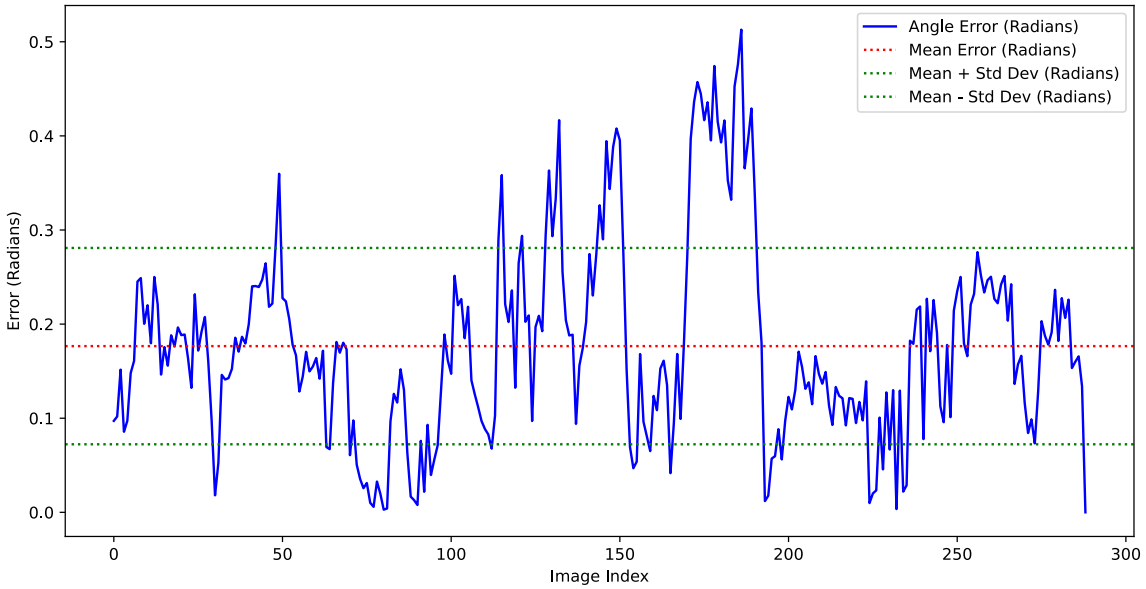


Figure 6.28: Overall angle error in radians with adjusted parameters.

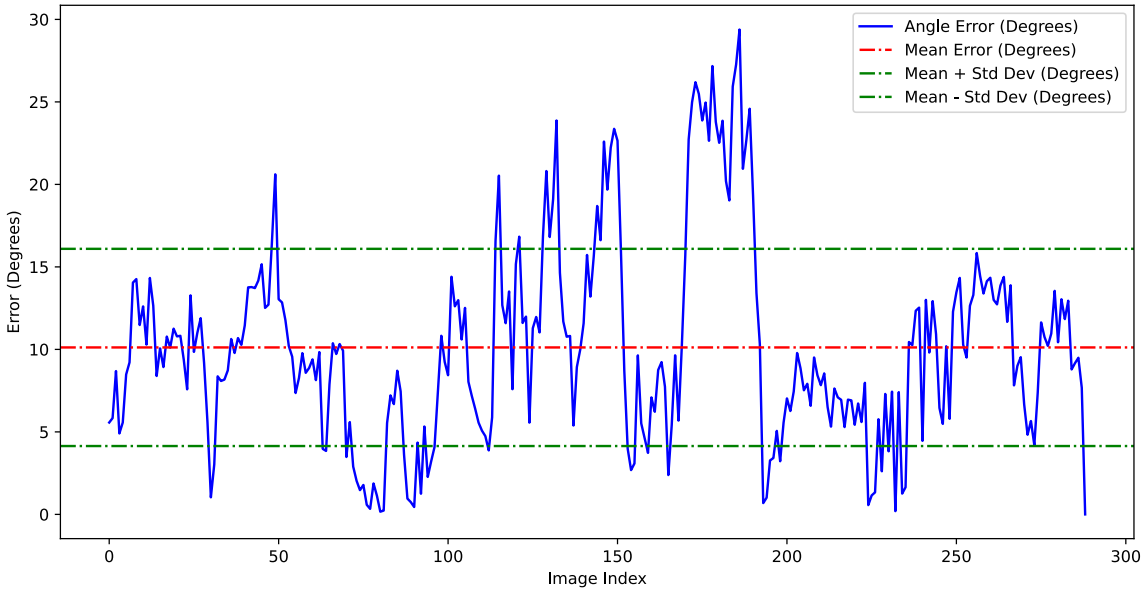


Figure 6.29: Overall angle error in degrees with adjusted parameters.

Plot 6.30 offers a visual representation of the translation error (Euclidean distances). Plots 6.31, 6.32 and 6.33: In these three plots, we visualize the components of ground truth translations (x, y, z) alongside the predicted translation components (x, y, z).

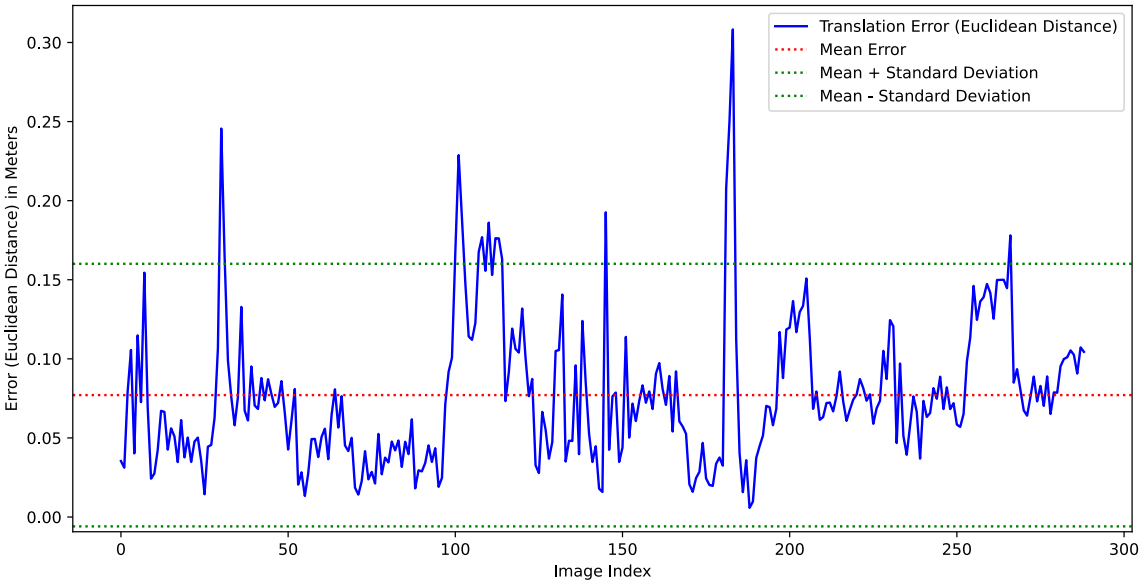


Figure 6.30: Translatioin error representation with adjusted parameters.

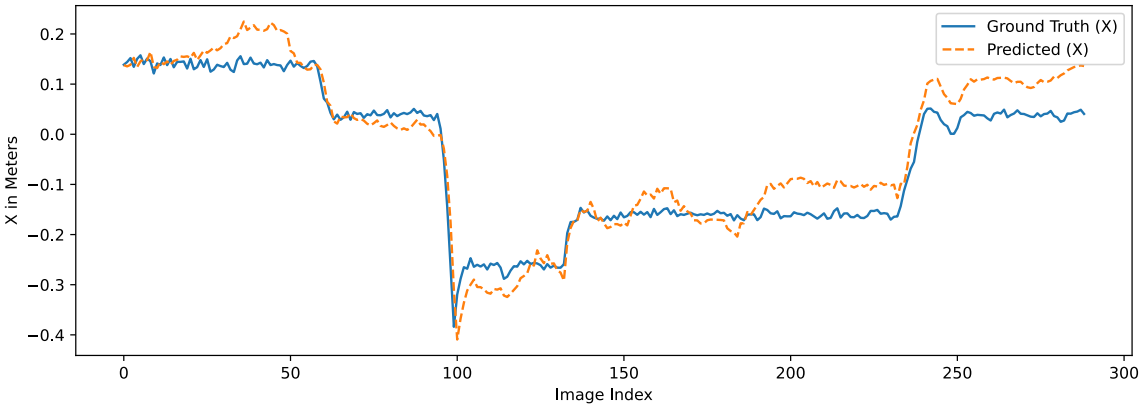


Figure 6.31: Ground truth x component of the translation vs predicted x value with adjusted parameters.

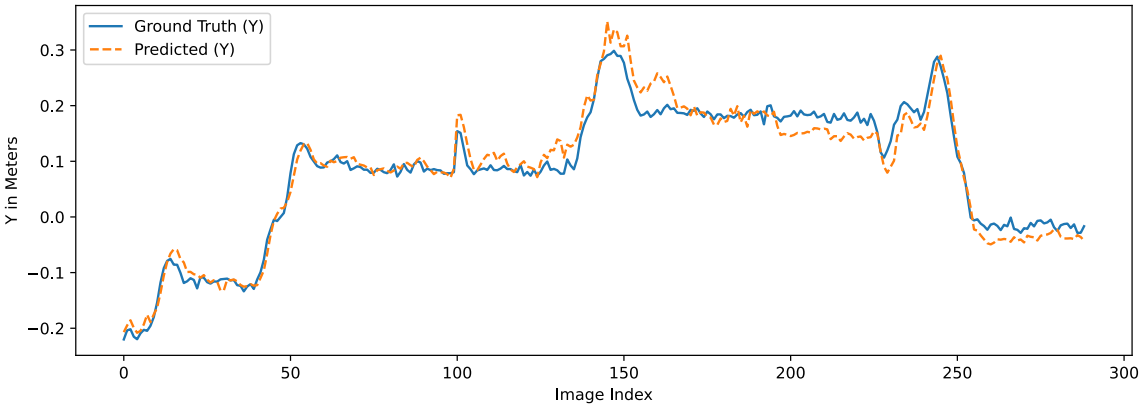


Figure 6.32: Ground truth y component of the translation vs predicted y value with adjusted parameters.

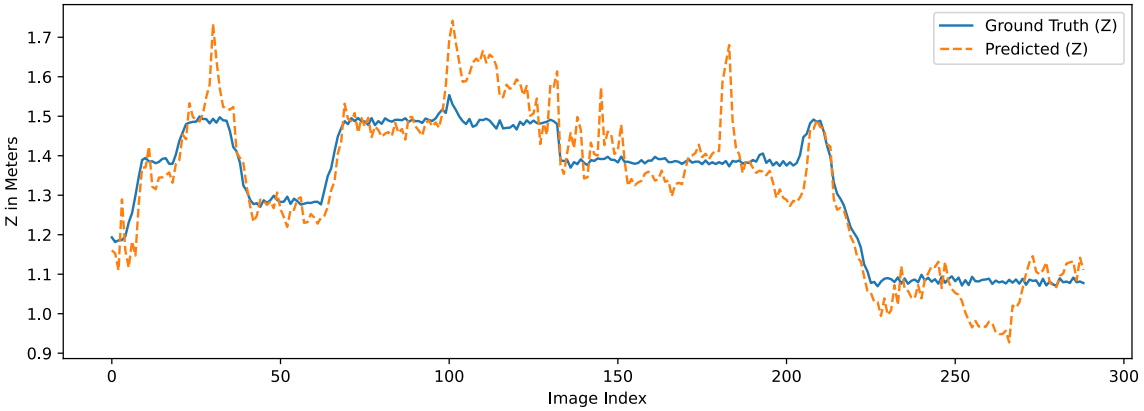


Figure 6.33: Ground truth z component of the translation vs predicted z value with adjusted parameters.

Plots 6.34 through 6.36 present the quantitative errors for each translation component. These plots provide a detailed examination of how well the model estimates each component of the translation.

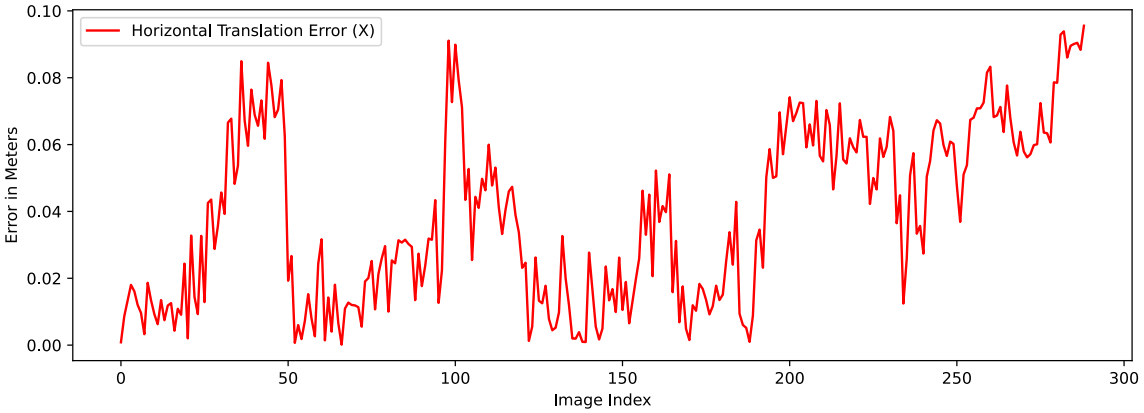


Figure 6.34: Quantitative error analysis of translation x component with adjusted parameters.

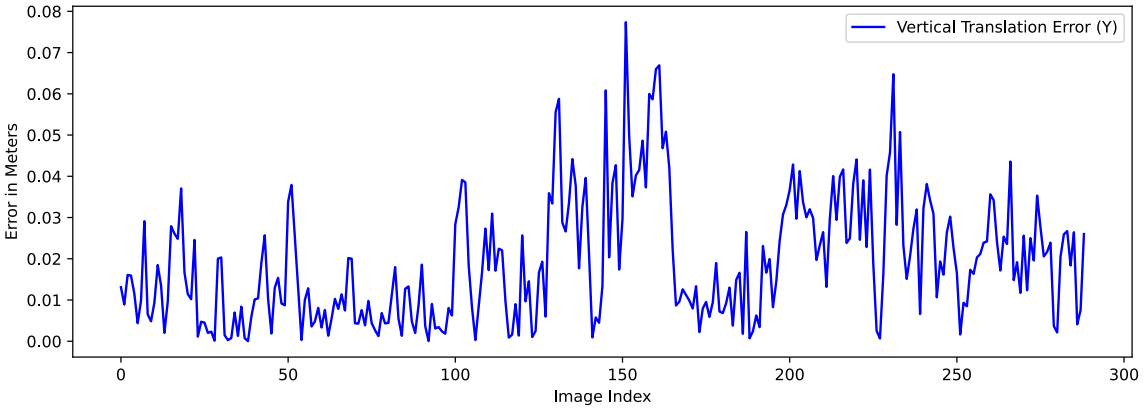


Figure 6.35: Quantitative error analysis of translation y component with adjusted parameters.

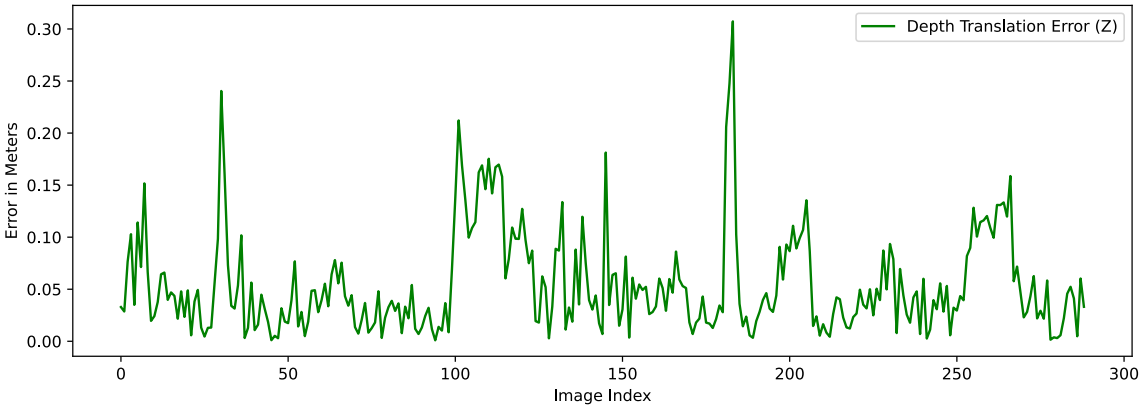


Figure 6.36: Quantitative error analysis of translation z component with adjusted parameters.

The comparison of tables 6.1 and 6.2 reveals the following differences in the recorded values:

1. **Mean Euclidean distance (translation in meters)** has decreased from 0.99769 to 0.07708, which is a substantial improvement of 0.92061 meters. This indicates a significant enhancement in the model's ability to estimate the translation aspect of pose with much higher precision.
2. **Mean Euclidean distance (orientation)** shows a minor improvement, decreasing from 0.13657 to 0.12985. This is a reduction of 0.00672, which reflects a slight enhancement in the model's orientation accuracy.
3. **Quaternion error standard deviation** has a marginal decrease from 0.08053 to 0.08299.
4. **Translation error standard deviation (meters)** has seen a notable reduction from 0.15902 to 0.04525, indicating that the variation in translation estimates has become much tighter and, thus, more consistent.
5. **Mean angle error in radians** has been reduced from 0.19483 to 0.17660, suggesting that the mean angular error is smaller in the updated model.
6. **Angle error standard deviation in radians** has improved, decreasing from 0.11694 to 0.10430, which suggests that the spread of angle errors around the mean has decreased.
7. **Mean angle error in degrees** has been reduced from 11.16346 to 10.11869, which is an improvement indicating that the average angular error in degrees is lower.
8. **Angle error standard deviation in degrees** has also decreased from 6.70033 to 5.97620, indicating a more precise angular error estimation.

Overall, these changes show that the model's performance has improved considerably in terms of translation accuracy, with a remarkable decrease in the mean Euclidean distance for translation. There is also a general improvement in the orientation accuracy, though to a lesser extent, as indicated by the smaller but still significant decrease in the mean Euclidean distance for orientation. The reductions in both the mean angle error and its standard deviation in radians and degrees further underscore the enhanced accuracy of the model in estimating pose after the calibration adjustments.
Our results can be contextualized against the findings of Sharma et al. in [71]. In their study, they reported a mean translation error of 0.294 meters for a synthetic test set. This metric is particularly noteworthy when juxtaposed against our own mean translation error of 0.07708 meters.
Furthermore, Sharma et al. also reported a mean orientation error of 8.4254 degrees for the same synthetic test set. In contrast, our study achieved a mean orientation error of 10.11869 degrees.
Overall, comparing these key metrics with those reported in [71] provides insights into the effectiveness and accuracy of our pose estimation model. It helps in understanding how our method stands in relation to established research.

6.2 Limits

Our study is subject to certain limitations. Firstly, our dataset is constrained by the inaccuracies in the camera parameters. During object detection and keypoints detection training, the Astrobee was predominantly centered in the images. This central positioning had repercussions on the performance of the object detection model since it was primarily trained on images with a centered Astrobee.
Secondly, the resolution of the images presents another limitation. Higher-resolution images would likely yield improved performance in the object detection model. This enhancement stems from the convolutional layers in the NN, which would operate on images with greater clarity and detail. In essence, the limitations are rooted in the dataset constraints and image resolution.
Furthermore, it's essential to acknowledge that we lacked precise camera parameters. These parameters play a critical role in the 3D-2D projection and pose estimation processes, particularly when utilizing functions such as solvePnP. The absence of accurate camera parameters constituted an additional limitation in our study, as it affected the reliability and precision of our pose estimation model.
We have currently had to limit ourselves exclusively to images from the simulator, and the quality of these simulator images is suboptimal, not only in terms of resolution but also in aspects of lighting condition and the interior representation of the ISS.

7. Conclusion and Outlook

In this section, we provide a comprehensive recap of our work's primary objectives, the methodology adopted, the key findings, challenges faced, and the broader impact of our work within the field of close proximity operations and robotics. We also discuss the overall contribution of our research and outline potential avenues for future work. This structured analysis serves as the concluding chapter of our exploration into the vision-based pose estimation of a non-cooperative tumbling object, the Astrobee, within the context of the ROS and showcases the exciting possibilities our work unlocks for space robotics and beyond.

7.1 Summary

In our work, our primary objective was the precise vision-based pose estimation of a non-cooperative tumbling object, the Astrobee, using ML. This objective was accompanied by a range of tasks, including generating a dataset tailored for object detection, with a particular emphasis on the cutting-edge YOLOv8 model, a SOTA model as of 2023. The dataset was meticulously crafted within the realm of the ROS simulation. It featured images of a target Astrobee, diligently captured by the navigation camera of its counterpart, the chaser Astrobee. For each image, we automatically generated a corresponding label file that encapsulated the precise coordinates, in pixels, of predefined keypoints on the target Astrobee. Subsequently, we harmonized these labels with the YOLOv8 format and undertook the crucial phase of model training. Following the successful completion of this step, we proceeded to create a detailed 3D model of the Astrobee robot. This 3D model was subsequently integrated into the broader framework, alongside the detected keypoints within an image.

We chose a hybrid approach that was organized into two distinct stages. In the initial stage, we harnessed a DL-based ML model for the automated detection of keypoints within images. In the second stage, we employed a geometrically based method, utilizing the comprehensive 3D model we constructed, the detected keypoints within the image, and the intrinsic camera parameters in the solvePnP function for the precise pose estimation of the target Astrobee in the image.

We have laid a first important foundational cornerstone for the pose estimation in ROS Astrobee. The developed framework is operational. We can now build upon this and address the identified issues in subsequent research efforts.

Our work underscored the paramount importance of an accurate and meticulously crafted dataset for robust object detection. The quality of keypoint estimation in images is intrinsically linked to the precision of the labels used during the object detection model's training. Conversely, imprecise labels could be regarded as noise

in the data. Notably, these variations could have potential utility, especially in the context of model training.
There were notable limitations to our study. We exclusively used synthetically generated data, as spaceborne images were unavailable. Preparing the synthetic dataset for object detection was not without its challenges. Manual removal of outliers was necessitated due to inaccuracies in automatically generated labels during both the training and testing phases. Furthermore, we encountered limitations concerning the accuracy of the camera parameters provided by ROS, which required manual calibration of the intrinsic parameters.

7.2 Future Work

The implications of this work extend beyond its immediate scope. Given the availability of the requisite parameters and a more extensive dataset, we could further refine our results, enhancing their precision. A potential enhancement for our object detection model involves the incorporation of an object tracking mechanism. In addition, to conduct an accurate pose estimation in space and effectively reduce the DG, it's imperative to integrate authentic space images of the specific object for which the pose estimation is required. Importantly, our approach for ROS-based pose estimation could be readily adapted for estimating the pose of other non-cooperative tumbling objects. Whether for space debris, objects, or robots with known forms or structures, our pipeline could be repurposed for estimating pose, rendezvous, and docking, enabling controlled manipulation and transportation. This versatility makes our work valuable not only for space robotics but also for terrestrial applications where pose estimation and object detection are vital.
In summary, our work underscores the feasibility of accurately estimating the 6-DOF pose of a non-cooperative tumbling object within ROS. The potential applications of this methodology are diverse, promising a range of innovative possibilities for space-based missions, such as ADR. This work not only demonstrates the feasibility of accurate pose estimation but also lays the foundation for advancing the capabilities of robotic systems in complex and dynamic environments, which is crucial for the future of space robotics.

Bibliography

[1] *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, Piscataway, NJ, 2020. ISBN 9781728173955. (cited on Page 22)

[2] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018. (cited on Page 15)

[3] V. V. Adushkin, O. Yu. Aksenov, S. S. Veniaminov, S. I. Kozlov, and V. V. Tyurenkova. The small orbital debris population and its impact on space activities and ecological safety. *Acta Astronautica*, 176:591–597, 2020. (cited on Page 2)

[4] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016. (cited on Page 11)

[5] André Araújo, David Portugal, Micael S Couceiro, and Rui P Rocha. Integrating arduino-based educational mobile robots in ros. *Journal of Intelligent & Robotic Systems*, 77:281–298, 2015. (cited on Page 7)

[6] Innovators at NASA Johnson Space Center. Spacecraft to remove orbital debris. (cited on Page xiii und 2)

[7] Ghazali Bin Sulong and M. Randles. Computer vision using pose estimation. *Wasit Journal of Computer and Mathematics Science*, 2(1):85–92, 2023. (cited on Page 4 und 16)

[8] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 421–436. Springer, 2012. (cited on Page 11)

[9] Jason Brownlee. *Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition in Python*. Machine Learning Mastery, 2019. (cited on Page 5)

[10] Maria Bualat, Jonathan Barlow, Terrence Fong, Chris Provencher, and Trey Smith. Astrobee: Developing a free-flying robot for the international space station. In *AIAA SPACE 2015 conference and exposition*, page 4643, 2015. (cited on Page 6)

[11] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *Proceedings of the international conference on automated planning and scheduling*, volume 25, pages 333–341, 2015. (cited on Page 7)

[12] Marco M. Castronuovo. Active space debris removal—a preliminary mission analysis and design. *Acta Astronautica*, 69(9-10):848–859, 2011. (cited on Page 2)

[13] Bo Chen, Jiewei Cao, Alvaro Parra, and Tat-Jun Chin. Satellite pose estimation with deep landmark regression and nonlinear pose refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. (cited on Page xiii, 19 und 20)

[14] Tomáš Chobola, Daniel Vašata, and Pavel Kordík. Transfer learning based few-shot classification using optimal transport mapping from preprocessed latent space of backbone neural network. In Isabelle Guyon, Jan N. van Rijn, Sébastien Treguer, and Joaquin Vanschoren, editors, *AAAI Workshop on Meta-Learning and MetaDL Challenge*, volume 140 of *Proceedings of Machine Learning Research*, pages 29–37. PMLR, 09 Feb 2021. URL https://proceedings.mlr.press/v140/chobola21a.html. (cited on Page 16)

[15] Changhyun Choi, Yuichi Taguchi, Oncel Tuzel, Ming-Yu Liu, and Srikumar Ramalingam. Voting-based pose estimation for robotic assembly using a 3d sensor. In *2012 IEEE International Conference on Robotics and Automation*, pages 1724–1731, 2012. doi: 10.1109/ICRA.2012.6225371. (cited on Page 5)

[16] Peter Christoffersen and Kris Jacobs. The importance of the loss function in option valuation. *Journal of Financial Economics*, 72(2):291–318, 2004. (cited on Page 11)

[17] Mirza Cilimkovic. Neural networks and back propagation algorithm. *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin*, 15(1), 2015. (cited on Page 12)

[18] Frédéric Devernay and Olivier Faugeras. Straight lines have to be straight automatic calibration and removal of distortion from scenes of structured environments. *Mach Vis Appl*, 13, 08 2001. doi: 10.1007/PL00013269. (cited on Page 27)

[19] AD Dongare, RR Kharde, Amit D Kachare, et al. Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1):189–194, 2012. (cited on Page 10)

[20] Weili Fang, Lieyun Ding, Peter ED Love, Hanbin Luo, Heng Li, Feniosky Pena-Mora, Botao Zhong, and Cheng Zhou. Computer vision applications in construction safety assurance. *Automation in Construction*, 110:103013, 2020. (cited on Page 5)

[21] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. (cited on Page 18)

[22] Lorenzo Fluckiger and Brian Coltin. Astrobee robot software: Enabling mobile autonomy on the iss. Technical report, 2019. (cited on Page 6)

[23] Jean Gaudart, Bernard Giusiano, and Laetitia Huiart. Comparison of the performance of multi-layer perceptron and linear regression for epidemiological data. *Computational statistics & data analysis*, 44(4):547–570, 2004. (cited on Page 10)

[24] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022. (cited on Page 9, 10, 13, 14, 15 und 32)

[25] Burooj Ghani, Tom Denton, Stefan Kahl, and Holger Klinck. Feature embeddings from large-scale acoustic bird classifiers enable few-shot transfer learning. *arXiv preprint arXiv:2307.06292*, 2023. (cited on Page 16)

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. adaptive computation and machine learning. *Massachusetts, USA*, 2017. (cited on Page 10, 11, 12 und 14)

[27] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021. (cited on Page 15 und 16)

[28] Zaixing He, Wuxi Feng, Xinyue Zhao, and Yongfeng Lv. 6d pose estimation of objects: Recent technologies and challenges. *Applied Sciences*, 11(1):228, 2021. (cited on Page 4)

[29] Daichi Hirano, Hiroki Kato, and Tatsuhiko Saito. Deep learning based pose estimation in space. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Madrid, Spain*, pages 4–6, 2018. (cited on Page 22)

[30] Timothy O. Hodson. Root-mean-square error (rmse) or mean absolute error (mae): when to use them or not. *Geoscientific Model Development*, 15(14): 5481–5487, 2022. (cited on Page 9)

[31] Chaoqun Hong, Jun Yu, Jian Zhang, Xiongnan Jin, and Kyong-Ho Lee. Multimodal face-pose estimation with multitask manifold deep learning. *IEEE Transactions on Industrial Informatics*, 15(7):3952–3961, 2019. (cited on Page 5)

[32] Eric Horvitz and Deirdre Mulligan. Policy forum. data, privacy, and the greater good. *Science (New York, N.Y.)*, 349(6245):253–255, 2015. (cited on Page 8)

[33] H Jabbar and Rafiqul Zaman Khan. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices*, 70(10.3850):978–981, 2015. (cited on Page 13 und 14)

[34] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 2016. (cited on Page 16)

[35] Julieta Martinez, Rayat Hossain, Javier Romero, and James J. Little. A simple yet effective baseline for 3d human pose estimation. (cited on Page 4)

[36] Ibrahem Kandel and Mauro Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT express*, 6(4):312–315, 2020. (cited on Page 13)

[37] Abdullah Khan, Asif Laghari, and Shafique Awan. Machine learning in computer vision: A review. *ICST Transactions on Scalable Information Systems*, page 169418, 2021. (cited on Page 5 und 8)

[38] Anis Koubaa, editor. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, volume 625 of *Studies in Computational Intelligence*. Springer International Publishing and Imprint: Springer, Cham, 1st ed. 2016 edition, 2016. ISBN 9783319260549. (cited on Page xiii und 7)

[39] Anis Koubâa et al. *Robot Operating System (ROS).*, volume 1. Springer, 2017. (cited on Page 7)

[40] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015. (cited on Page 14)

[41] B. Y. Lee, L. H. Liew, W. S. Cheah, and Y. C. Wang. Occlusion handling in videos object tracking: A survey. *IOP Conference Series: Earth and Environmental Science*, 18:012020, 2014. (cited on Page 5)

[42] Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. *Computer Vision – ECCV 2016*, volume 9905. Springer International Publishing, Cham, 2016. ISBN 978-3-319-46447-3. (cited on Page 16)

[43] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2022. (cited on Page 14)

[44] Yunzhi Lin, Jonathan Tremblay, Stephen Tyree, Patricio A Vela, and Stan Birchfield. Single-stage keypoint-based category-level object pose estimation from an rgb image. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1547–1553. IEEE, 2022. (cited on Page 16 und 17)

[45] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015. (cited on Page 14)

[46] Xiao Xin Lu. A review of solutions for perspective-n-point problem in camera pose estimation. In *Journal of Physics: Conference Series*, volume 1087, page 052009. IOP Publishing, 2018. (cited on Page 18)

[47] C. Priyant Mark and Surekha Kamath. Review of active space debris removal methods. *Space Policy*, 47:194–206, 2019. ISSN 0265-9646. doi: https://doi.org/10.1016/j.spacepol.2018.12.005. URL https://www.sciencedirect.com/science/article/pii/S0265964618300110. (cited on Page 2)

[48] Pablo Martinez, Mohamed Al-Hussein, and Rafiq Ahmad. A scientometric analysis and critical review of computer vision applications for construction. *Automation in Construction*, 107:102947, 2019. (cited on Page 5)

[49] Giorgia Marullo, Leonardo Tanzi, Pietro Piazzolla, and Enrico Vezzetti. 6d object position estimation from 2d images: a literature review. *Multimedia Tools and Applications*, 82(16):24605–24643, 2023. (cited on Page 22)

[50] Wim Meeussen, Melonee Wise, Stuart Glaser, Sachin Chitta, Conor McGann, Patrick Mihelich, Eitan Marder-Eppstein, Marius Muja, Victor Eruhimov, Tully Foote, John Hsu, Radu Bogdan Rusu, Bhaskara Marthi, Gary Bradski, Kurt Konolige, Brian Gerkey, and Eric Berger. Autonomous door opening and plugging in with a personal robot. In *2010 IEEE International Conference on Robotics and Automation*, pages 729–736. IEEE, 2010. ISBN 978-1-4244-5038-1. (cited on Page 5)

[51] Kishan Mehrotra, Chilukuri K Mohan, and Sanjay Ranka. *Elements of artificial neural networks*. MIT press, 1997. (cited on Page 10)

[52] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital signal processing*, 73:1–15, 2018. (cited on Page 10)

[53] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. (cited on Page 4 und 5)

[54] Vladimir Nasteski. An overview of the supervised machine learning methods. *HORIZONS.B*, 4:51–62, 2017. (cited on Page 8)

[55] Shuteng Niu, Yongxin Liu, Jian Wang, and Houbing Song. A decade survey of transfer learning (2010–2020). *IEEE Transactions on Artificial Intelligence*, 1 (2):151–166, 2020. (cited on Page 15 und 16)

[56] Charles Oestreich, Antonio Terán Espinoza, Jessica Todd, Keenan Albee, and Richard Linares. On-orbit inspection of an unknown, tumbling target using nasa's astrobee robotic free-flyers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2039–2047, 2021. (cited on Page xiii, 6, 17 und 18)

[57] Brian Okorn, Mengyun Xu, Martial Hebert, and David Held. Learning orientation distributions for object pose estimation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10580–10587, 2020. doi: 10.1109/IROS45743.2020.9340860. (cited on Page 4)

[58] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks, 11/26/2015. (cited on Page xiii und 15)

[59] Tae Ha Park, Sumant Sharma, and Simone D'Amico. Towards robust learning-based pose estimation of noncooperative spacecraft, 01.09.2019. (cited on Page xiii, 17, 18 und 19)

[60] Leo Pauly, Wassim Rharbaoui, Carl Shneider, Arunkumar Rathinam, Vincent Gaudilliere, and Djamila Aouada. A survey on deep learning-based monocular spacecraft pose estimation: Current state, limitations and prospects, 12.05.2023. (cited on Page 18)

[61] Juan Ignacio Bravo Pérez-Villar, Álvaro García-Martín, and Jesús Bescós. Spacecraft Pose Estimation Based on Unsupervised Domain Adaptation and on a 3D-Guided Loss Combination, 12/27/2022. (cited on Page 18)

[62] Eko Prasetyo, Nanik Suciati, and Chastine Fatichah. A comparison of yolo and mask r-cnn for segmenting head and tail of fish. In *2020 4th International Conference on Informatics and Computational Sciences (ICICoS)*, pages 1–6. IEEE, 2020. ISBN 978-1-7281-9526-1. (cited on Page xvi und 23)

[63] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009. (cited on Page 7)

[64] Sebastian Raschka, Yuxi Liu, Vahid Mirjalili, and Dmytro Dzhulgakov. *Machine learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python*. Expert insight. Packt, Birmingham and Mumbai, 2022. ISBN 9781801819312. (cited on Page 14)

[65] Jasmine Richter, Florian Faion, Di Feng, Paul Benedikt Becker, Piotr Sielecki, and Claudius Glaeser. Understanding the domain gap in lidar object detection networks. *arXiv preprint arXiv:2204.10024*, 2022. (cited on Page 8)

[66] R. Riesenfeld. Homogeneous coordinates and projective planes in computer graphics. *IEEE Computer Graphics and Applications*, 1(01):50–55, jan 1981. ISSN 1558-1756. doi: 10.1109/MCG.1981.1673814. (cited on Page 27)

[67] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. (cited on Page 11)

[68] Shalev-Shwartz Shai and Ben-David Shai. Understanding machine learning: From theory to algorithms, 2014. (cited on Page 8)

[69] Minghe Shan, Jian Guo, and Eberhard Gill. Review and comparison of active space debris capturing and removal methods. *Progress in Aerospace Sciences*, 80: 18–32, 2016. ISSN 0376-0421. doi: https://doi.org/10.1016/j.paerosci.2015.11.001. URL https://www.sciencedirect.com/science/article/pii/S0376042115300221. (cited on Page 3)

[70] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017. (cited on Page 10)

[71] Sumant Sharma and Simone D'Amico. Pose estimation for non-cooperative rendezvous using neural networks, 24.06.2019. (cited on Page 4 und 59)

[72] Sumant Sharma, Connor Beierle, and Simone D'Amico. Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks. In *2018 IEEE Aerospace Conference*, pages 1–12, 2018. doi: 10.1109/AERO.2018.8396425. (cited on Page 17)

[73] Siddalingeshwar Patil and Umakant Kulkarni. *Proceedings of the International Conference on Trends in Electronics and Informatics (ICOEI 2019): 23-25 April 2019*. IEEE, Piscataway, NJ, 2019. ISBN 9781538694398. (cited on Page 8)

[74] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018. (cited on Page 12)

[75] Jianing Song, Duarte Rondao, and Nabil Aouf. Deep learning-based spacecraft relative navigation methods: A survey. *Acta Astronautica*, 191:22–40, 2022. (cited on Page xiii, 20 und 22)

[76] Shahriar Shakir Sumit, Junzo Watada, Anurava Roy, and D. R.A. Rambli. In object detection deep learning methods, yolo shows supremum to mask r-cnn. *Journal of Physics: Conference Series*, 1529(4):042086, 2020. (cited on Page xvi und 23)

[77] David Joseph Tan, Nassir Navab, and Federico Tombari. 6d object pose estimation with depth images: A seamless approach for robotic interaction and augmented reality, 2017. (cited on Page 4)

[78] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26, 2020. (cited on Page 16)

[79] Hilde JP Weerts, Andreas C Mueller, and Joaquin Vanschoren. Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588*, 2020. (cited on Page 12)

[80] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1), 2016. (cited on Page 15)

[81] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. (cited on Page 12)

[82] Heng Yang and Marco Pavone. Object pose estimation with statistical guarantees: Conformal keypoint detection and geometric uncertainty propagation, 22.03.2023. (cited on Page 16)

[83] Yinlin Hu, Sebastien Speierer, Wenzel Jakob, Pascal Fua, and Mathieu Salzmann. Wide-depth-range 6d object pose estimation in space. (cited on Page 17)

[84] Jincheng Yu, Kaijian Weng, Guoyuan Liang, and Guanghan Xie. A vision-based robotic grasping system using deep learning for 3d object recognition and pose estimation. In *2013 IEEE international conference on robotics and biomimetics (ROBIO)*, pages 1175–1180. IEEE, 2013. (cited on Page 3)

[85] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. (cited on Page 12)

[86] Xiaoqin Zeng and Daniel S Yeung. Sensitivity analysis of multilayer perceptron to input and weight perturbations. *IEEE Transactions on neural networks*, 12 (6):1358–1366, 2001. (cited on Page 10)

[87] Gaopeng Zhao, Sixiong Xu, and Yuming Bo. Lidar-based non-cooperative tumbling spacecraft pose tracking by fusing depth maps and point clouds. *Sensors*, 18(10):3432, 2018. (cited on Page 3)

[88] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019. (cited on Page 16)

[89] Yinqiang Zheng, Yubin Kuang, Shigeki Sugimoto, Kalle Astrom, and Masatoshi Okutomi. Revisiting the pnp problem: A fast, general and optimal solution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2344–2351, 2013. (cited on Page 18)

[90] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020. (cited on Page 16)

[91] Jinming Zou, Yi Han, and Sung-Sau So. *Overview of Artificial Neural Networks*, pages 14–22. Humana Press, Totowa, NJ, 2009. ISBN 978-1-60327-101-1. doi: 10.1007/978-1-60327-101-1_2. URL https://doi.org/10.1007/978-1-60327-101-1_2. (cited on Page 9 und 10)