SCHOOL OF COMPUTER AND
COMMUNICATION SCIENCES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

# Computer Vision Laboratory
# Unseen Spacecraft Pose Estimation

Baseline solution by implementing a machine learning framework with target models included

Bachelor's Thesis in Computer Science

Author: Jérémy Chaverot
Supervisor: Dr. Mathieu Salzmann
Advisors: Dr. Andrew Price, PhD. Chen Zhao
Semester: Fall 2023

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the advisors.

Lausanne, Switzerland, 05.01.24 Jérémy Chaverot

## Acknowledgments

Before delving into the topic, I'd like to express some acknowledgments. First and foremost I must thank my advisors Andrew Price and Chen Zhao for accepting my request to take part in a semester project under their guidance. I am grateful to have been able to practice my skills with them, and can only hope that the feeling is mutual. Moreover I would also like to thoroughly thank my friends and family for supporting me in my academic journey, despite a rather unstable start in my studies.

## Abstract

# Contents

# 1. Introduction

Test ref to Listing A.1. Test ref to Listing A.3 Test ref to Gen6D [1]

## 1.1. Problem Formulation

### 1.1.1. The settings

### 1.1.2. The goal

## 1.2. The work environment: Scitas Izar

Test to refer to the video from 3B1B: [2].

```bash
#!/bin/bash
#SBATCH --chdir /scratch/izar/jchavero
#SBATCH --partition=gpu
#SBATCH --qos=gpu_free
#SBATCH --gres=gpu:2
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem 16G

echo STARTING AT `date`

echo "Loading modules"
module load gcc openmpi py-torch py-torchvision cuda

echo "Launching the virtual environment"
source ~/opt/izar1/venv-gcc/bin/activate

echo "Navigating to the directory and executing the task"
cd ~/Gen6D
python eval.py --cfg configs/gen6d_pretrain.yaml --object_name spacecraft/hubble

echo FINISHED AT `date`
```

Listing 1.1: Bash script `execute.sh` to run a machine learning model on Scitas Izar EPFL. While the overall structure remains consistent, this script is specific to Gen6D's architecture, further discussed later.

Then to run the script we use the following command:

```bash
$ sbatch execute.sh
```

Listing 1.2: Linux command to run the bash script.

# 2. Gen6D: Formal Description

## 2.1. Overview of the Network

## 2.2. Detection

## 2.3. Viewpoint Selection

## 2.4. Pose Refinement

# 3. Implementation of the model

## 3.1. Data Loader

abstract base classes (ABC) each and every abstract method

## 3.2. From Quaternions to Rotation Matrices

In Gen6D, the model represents the ground truth and estimated poses using the format $\mathbf{P} = (\mathbf{R}, \mathbf{t})$. Here, $\mathbf{R}$ denotes the rotation matrix and $\mathbf{t}$ is the translation vector. The thing is, in the SPACECRAFT dataset, the poses have the format $\mathbf{P} = (\underline{\mathbf{q}}, \mathbf{t})$, where $\underline{\mathbf{q}}$ is a *quaternion*. We use quaternions for three-dimensional rotation calculations because they offer several benefits over rotation matrices. Notably, quaternions are more compact, requiring only four elements to be stored compared to nine for a matrix. Additionally, they are more efficient when composing rotations thanks to their algebraic properties.

Despite the benefits of quaternions mentioned earlier, other datasets frequently represent poses using a combination of rotation matrices and translation vectors. Therefore, to align with Gen6D's pose format, this section will focus on converting quaternions into rotation matrices.

Quaternions were first introduced by the Irish mathematician W. R. Hamilton in 1843 as an extension of the complex numbers [3]. In the first place, we provide the definition of a *quaternion*: it is the sum of a scalar $q_0$ and a vector $\mathbf{q} = (q_1, q_2, q_3)$, that is,

$$\underline{\mathbf{q}} \overset{\text{def.}}{=} q_0 + \mathbf{q} = q_0 + q_1 \boldsymbol{i} + q_2 \boldsymbol{j} + q_3 \boldsymbol{k}.$$

In the above, $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ denote the three unit vectors of the canonical basis for the set of all ordered triples of real numbers $\mathbb{R}^3$. The set of quaternions is denoted by the 4-space $\mathbb{H}$.

The quaternion addition is component-wise. Regarding the product of two quaternions, it is essential to first outline the foundational rule established by Hamilton:

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1.$$

We derive the following multiplication table:

| $\times$ | $1$ | $\boldsymbol{i}$ | $\boldsymbol{j}$ | $\boldsymbol{k}$ |
|---|---|---|---|---|
| $1$ | $1$ | $\boldsymbol{i}$ | $\boldsymbol{j}$ | $\boldsymbol{k}$ |
| $\boldsymbol{i}$ | $\boldsymbol{i}$ | $-1$ | $\boldsymbol{k}$ | $-\boldsymbol{j}$ |
| $\boldsymbol{j}$ | $\boldsymbol{j}$ | $-\boldsymbol{k}$ | $-1$ | $\boldsymbol{i}$ |
| $\boldsymbol{k}$ | $\boldsymbol{k}$ | $\boldsymbol{j}$ | $-\boldsymbol{i}$ | $-1$ |

Let $(\underline{\mathbf{p}}, \underline{\mathbf{q}}) \in \mathbb{H}^2$, we are now able to present the multiplication of $\underline{\mathbf{p}}$ and $\underline{\mathbf{q}}$:

$$\underline{\mathbf{pq}} = \underbrace{p_0 q_0 - \mathbf{p} \cdot \mathbf{q}}_{\text{scalar part}} + \underbrace{p_0 \mathbf{q} + q_0 \mathbf{p} + \mathbf{p} \times \mathbf{q}}_{\text{vector part}}.$$

In the preceding, we recall that the binary operation $\mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R} : (\mathbf{p}, \mathbf{q}) \mapsto \mathbf{p} \cdot \mathbf{q} \overset{\text{def.}}{=} \mathbf{p}^\intercal \mathbf{q}$ is the *dot product*, and the other

$$\mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3 : (\mathbf{p}, \mathbf{q}) \mapsto \mathbf{p} \times \mathbf{q} \overset{\text{def.}}{=} \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ p_1 & p_2 & p_3 \\ q_1 & q_2 & q_3 \end{vmatrix} = [\mathbf{p}]_\times \mathbf{q}$$

is the *cross product*, and $[\mathbf{p}]_\times$ denotes the transformation matrix that when multiplied from the right with a vector $\mathbf{q}$ gives $\mathbf{p} \times \mathbf{q}$.

Several additional definitions are essential before we can begin the conversion of quaternions to rotation matrices.
Let $\underline{\mathbf{q}} = q_0 + \mathbf{q}$ be a quaternion. The *complex conjugate* of $\underline{\mathbf{q}}$, denoted by $\underline{\mathbf{q}}^\star$, is given by the map

$$\mathbb{H} \to \mathbb{H} : \underline{\mathbf{q}} \mapsto \underline{\mathbf{q}}^\star \overset{\text{def.}}{=} q_0 - \mathbf{q}.$$

The *norm* of a quaternion $\underline{\mathbf{q}}$, denoted $|\underline{\mathbf{q}}|$, is the distance obtained from the map

$$\mathbb{H} \to \mathbb{R}^+ : \underline{\mathbf{q}} \mapsto |\underline{\mathbf{q}}| \overset{\text{def.}}{=} \sqrt{\underline{\mathbf{q}}^\star \underline{\mathbf{q}}}.$$

Note that a quaternion whose norm is 1 is referred to as a *unit quaternion*. The *reciprocal* of a quaternion is defined as the map

$$\mathbb{H}^* \to \mathbb{H}^* : \underline{\mathbf{q}} \mapsto \underline{\mathbf{q}}^{-1} \overset{\text{def.}}{=} \frac{\underline{\mathbf{q}}^\star}{|\underline{\mathbf{q}}|^2}.$$

We observe that if $\underline{\mathbf{q}}$ is a unit quaternion, we simply have $\underline{\mathbf{q}}^{-1} = \underline{\mathbf{q}}^\star$. Furthermore, the subsequent proposition is accepted: if $\underline{\mathbf{q}}$ is a unit quaternion, there exists a unique $\theta \in [0, 2\pi]$ such that

$$\underline{\mathbf{q}} = q_0 + \mathbf{q} = \cos\frac{\theta}{2} + \mathbf{u}\sin\frac{\theta}{2},$$

where the unit vector $\mathbf{u}$ is defined as $\mathbf{u} \overset{\text{def.}}{=} \frac{\mathbf{q}}{|\mathbf{q}|}$.

**Quaternion Rotation Operator** Let $\underline{\mathbf{q}} \in \mathbb{H}$ be a unit quaternion, and let $\mathbf{v} \in \mathbb{R}^3$ be a vector. The action of the $L_{\underline{\mathbf{q}}}$ function

$$\mathbb{R}^3 \to \mathbb{R}^3 : \mathbf{v} \mapsto L_{\underline{\mathbf{q}}}(\mathbf{v}) \overset{\text{def.}}{=} \underline{\mathbf{q}}\mathbf{v}\underline{\mathbf{q}}^\star$$

on $\mathbf{v}$ is equivalent to a rotation of the vector through an angle $\theta$ about the axis of rotation $\mathbf{u}$.

At last, we can proceed with our conversion task: our aim is to find a $3 \times 3$ rotation matrix $\mathbf{R}$, such that

$$\begin{cases} L_{\mathbf{R}}(\mathbf{v}) \overset{\text{def.}}{=} \mathbf{R}\mathbf{v} \\ L_{\mathbf{R}}(\mathbf{v}) = L_{\underline{\mathbf{q}}}(\mathbf{v}), \end{cases}$$

which means we wish to obtain an expression for $\mathbf{R}$ by manipulating $L_{\underline{\mathbf{q}}}$, utilizing principles of linear algebra and vector calculus. We consider the vector $\mathbf{v}$ as a quaternion with a zero scalar component:

$$\begin{aligned}
L_{\underline{\mathbf{q}}}(\mathbf{v}) &= \underline{\mathbf{q}}\mathbf{v}\underline{\mathbf{q}}^{\star} \\
&= (q_0 + \mathbf{q})(0 + \mathbf{v})(q_0 - \mathbf{q}) \\
&= (\underbrace{-\mathbf{q} \cdot \mathbf{v}}_{\text{scalar part}} + \underbrace{q_0\mathbf{v} + \mathbf{q} \times \mathbf{v}}_{\text{vector part}})(q_0 - \mathbf{q}) \\
&= q_0(-\mathbf{q} \cdot \mathbf{v}) - (q_0\mathbf{v} + \mathbf{q} \times \mathbf{v}) \cdot (-\mathbf{q}) \\
&\quad + (-\mathbf{q} \cdot \mathbf{v})(-\mathbf{q}) + q_0(q_0\mathbf{v} + \mathbf{q} \times \mathbf{v}) \\
&\quad + (q_0\mathbf{v} + \mathbf{q} \times \mathbf{v}) \times (-\mathbf{q}) \\
&= \underbrace{\cancel{-q_0(\mathbf{q} \cdot \mathbf{v}) + q_0(\mathbf{q} \cdot \mathbf{v})}}_{\text{scalar part}} \\
&\quad + \underbrace{\mathbf{q}\,(\mathbf{q} \cdot \mathbf{v}) + {q_0}^2\,\mathbf{v} + q_0(\mathbf{q} \times \mathbf{v}) + \mathbf{q} \times (q_0\mathbf{v} + \mathbf{q} \times \mathbf{v})}_{\text{vector part}} \\
&= \mathbf{q}\,(\mathbf{q}^{\intercal}\mathbf{v}) + {q_0}^2\,\mathbf{v} + q_0(\mathbf{q} \times \mathbf{v}) + \mathbf{q} \times (q_0\mathbf{v} + \mathbf{q} \times \mathbf{v}) \\
&= (\mathbf{q} \otimes \mathbf{q} + {q_0}^2\,\mathbf{I}_{3\times3} + 2\,q_0\,[\mathbf{q}]_\times + [\mathbf{q}]_\times^2)\;\mathbf{v}
\end{aligned}$$

In the above, $\otimes$ stands for the *outer product* and $\mathbf{I}_{3\times3}$ is the *identity matrix*. Since $L_{\mathbf{R}}(\mathbf{v}) = L_{\underline{\mathbf{q}}}(\mathbf{v})$, we can identify $\mathbf{R}$ as $(\mathbf{q} \otimes \mathbf{q} + {q_0}^2\,\mathbf{I}_{3\times3} + 2\,q_0\,[\mathbf{q}]_\times + [\mathbf{q}]_\times^2)$. We develop and simplify $\mathbf{R}$ to find its final expression:

$$\begin{aligned}
\mathbf{R} &= \mathbf{q} \otimes \mathbf{q} + {q_0}^2\,\mathbf{I}_{3\times3} + 2\,q_0\,[\mathbf{q}]_\times + [\mathbf{q}]_\times^2 \\
&= \begin{bmatrix} {q_1}^2 & q_1q_2 & q_1q_3 \\ q_2q_1 & {q_2}^2 & q_2q_3 \\ q_3q_1 & q_3q_2 & {q_3}^2 \end{bmatrix} + {q_0}^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + 2\,q_0 \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \\
&\quad + \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \\
&= 2 \begin{bmatrix} ({q_0}^2 + {q_1}^2) - \frac{1}{2} & q_1q_2 - q_0q_3 & q_1q_3 + q_0q_2 \\ q_1q_2 + q_0q_3 & ({q_0}^2 + {q_2}^2) - \frac{1}{2} & q_2q_3 - q_0q_1 \\ q_1q_3 - q_0q_2 & q_2q_3 + q_0q_1 & ({q_0}^2 + {q_3}^2) - \frac{1}{2} \end{bmatrix}
\end{aligned}$$

This completes our derivation of the rotation matrix $\mathbf{R}$. The Python code can be found in Listing A.3.

# 4. Experimental Results and Analysis

## 4.1. Reference and Query Images

## 4.2. Evaluation Metrics

To appreciate the quality of the estimations, the most widely used pose error functions are the Average Distance of Model Points (ADD) and the Average Closest Point Distance (ADD-S) metrics, both introduced by Hinterstoisser et al. [4]. For an object model $\mathcal{M}$, we compute the average distance to the corresponding model point. Therefore the error of an estimated pose $\hat{\mathbf{P}} = (\hat{\mathbf{R}}, \hat{\mathbf{T}})$ w.r.t. the ground truth pose $\bar{\mathbf{P}} = (\bar{\mathbf{R}}, \bar{\mathbf{T}})$ is calculated as follows:

$$ {}^{1}e_{\text{ADD}}(\hat{\mathbf{P}}, \bar{\mathbf{P}}, \mathcal{M}) \overset{\text{def.}}{=} \underset{\mathbf{x}\in\mathcal{M}}{\text{avg}} \left\| \bar{\mathbf{P}}\mathbf{x}^{\star} - \hat{\mathbf{P}}\mathbf{x}^{\star} \right\|_2 \tag{4.1} $$

$$ = \underset{\mathbf{x}\in\mathcal{M}}{\text{avg}} \left\| (\bar{\mathbf{R}}\mathbf{x} + \bar{\mathbf{T}}) - (\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{T}}) \right\|_2 \tag{4.2} $$

When the model $\mathcal{M}$ has symmetries that leads to indistinguishable views, the error is computed as the average distance to the closest model point:

$$ e_{\text{ADD-S}}(\hat{\mathbf{P}}, \bar{\mathbf{P}}, \mathcal{M}) \overset{\text{def.}}{=} \underset{\mathbf{x}_1\in\mathcal{M}}{\text{avg}} \ \underset{\mathbf{x}_2\in\mathcal{M}}{\min} \left\| \bar{\mathbf{P}}\mathbf{x}_1^{\star} - \hat{\mathbf{P}}\mathbf{x}_2^{\star} \right\|_2 \tag{4.3} $$

$$ = \underset{\mathbf{x}_1\in\mathcal{M}}{\text{avg}} \ \underset{\mathbf{x}_2\in\mathcal{M}}{\min} \left\| (\bar{\mathbf{R}}\mathbf{x}_1 + \bar{\mathbf{T}}) - (\hat{\mathbf{R}}\mathbf{x}_2 + \hat{\mathbf{T}}) \right\|_2 \tag{4.4} $$

It's important to point out that $e_{\text{ADD-S}}$ is more lenient compared to $e_{\text{ADD}}$, and should only be applied in cases where there is a definite presence of symmetry in the object and the estimated pose is already notably precise. Otherwise, using $e_{\text{ADD-S}}$ becomes irrelevant since the estimation would be unfairly advantaged. In the illustrations below, we consistently provide both metrics, however it is up to the reader to assess the relevance of $e_{\text{ADD-S}}$ based on the observed spacecraft.

## 4.3. Vizualisation and Quantitative Evaluation

[1] In this context, the vector $\mathbf{x}^{\star}$ represents a vector that has been extended by appending a 1, specifically for the purpose of matrix multiplication.

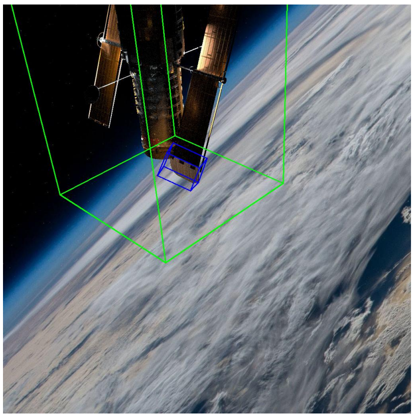

Figure 4.1: Hubble Space Telescope with earth rendered background, 1024x1024 first query image

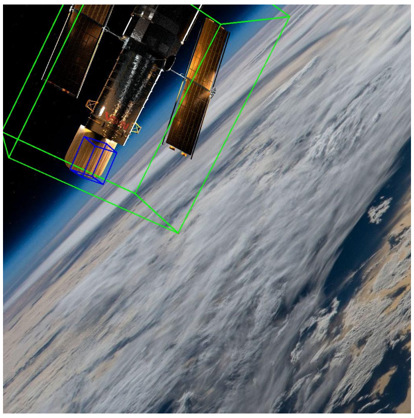

Figure 4.2: Hubble Space Telescope with earth rendered background, 1024x1024 second query image

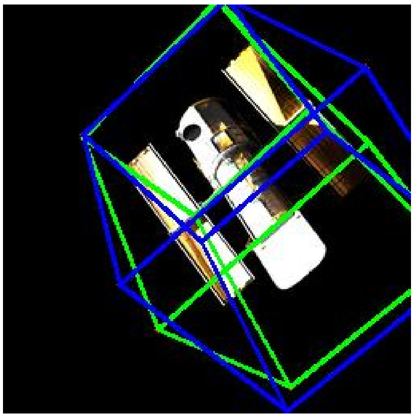

Figure 4.3: Hubble Space Telescope, no background, 256x256 query image, $e_{\text{ADD}} = 2.925$, $e_{\text{ADD-S}} = 1.183$

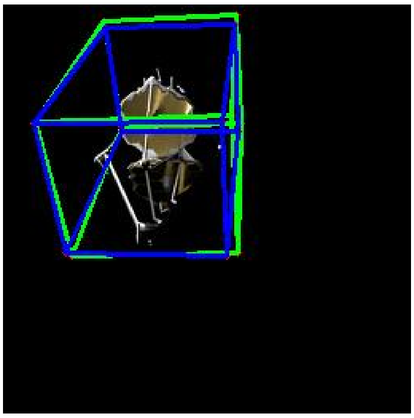

Figure 4.4: James Webb Space Telescope, no background, 256x256 query image, $e_{\text{ADD}} = 1.415$, $e_{\text{ADD-S}} = 0.808$

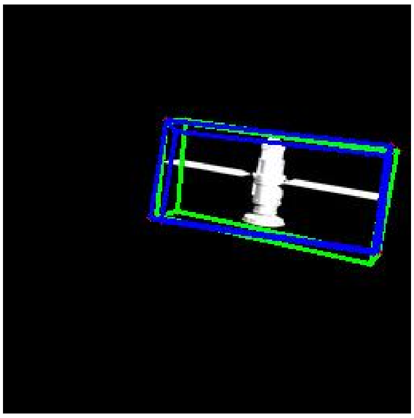

Figure 4.5: Cosmos Link, no background, 256x256 query image, $e_{\text{ADD}} = 1.718$, $e_{\text{ADD-S}} = 0.383$

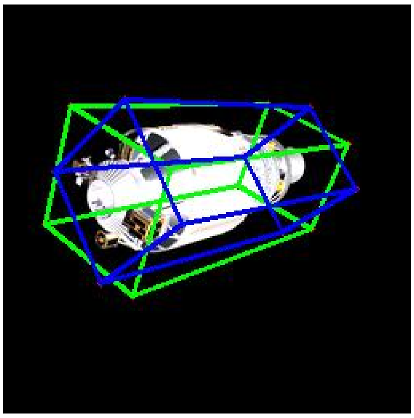

Figure 4.6: Rocket Body, no background, 256x256 query image, $e_{\text{ADD}} = 1.713$, $e_{\text{ADD-S}} = 0.252$

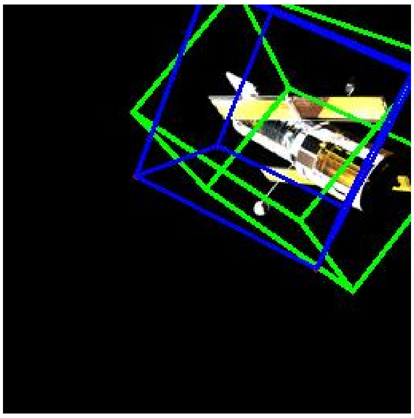

Figure 4.7: Hubble Space Telescope, no background, 256x256 query image, $e_{\text{ADD}} = 6.514$, $e_{\text{ADD-S}} = 1.571$

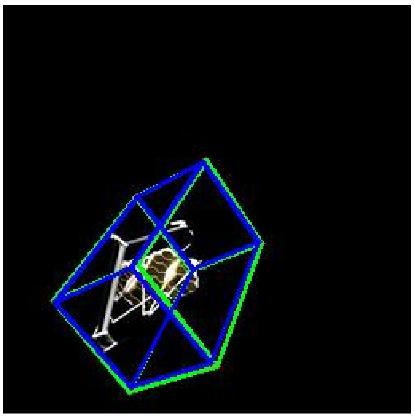

Figure 4.8: James Webb Space Telescope, no background, 256x256 query image, $e_{\text{ADD}} = 2.224$, $e_{\text{ADD-S}} = 1.261$

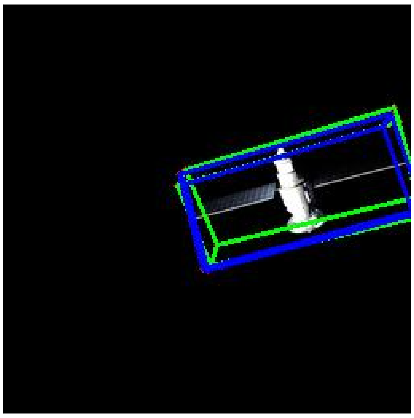

Figure 4.9: Cosmos Link, no background, 256x256 query image, $e_{\text{ADD}} = 1.925$, $e_{\text{ADD-S}} = 0.377$

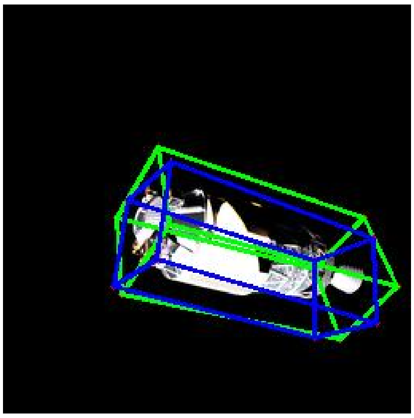

Figure 4.10: Rocket Body, no background, 256x256 query image, $e_{\text{ADD}} = 1.982$, $e_{\text{ADD-S}} = 0.501$

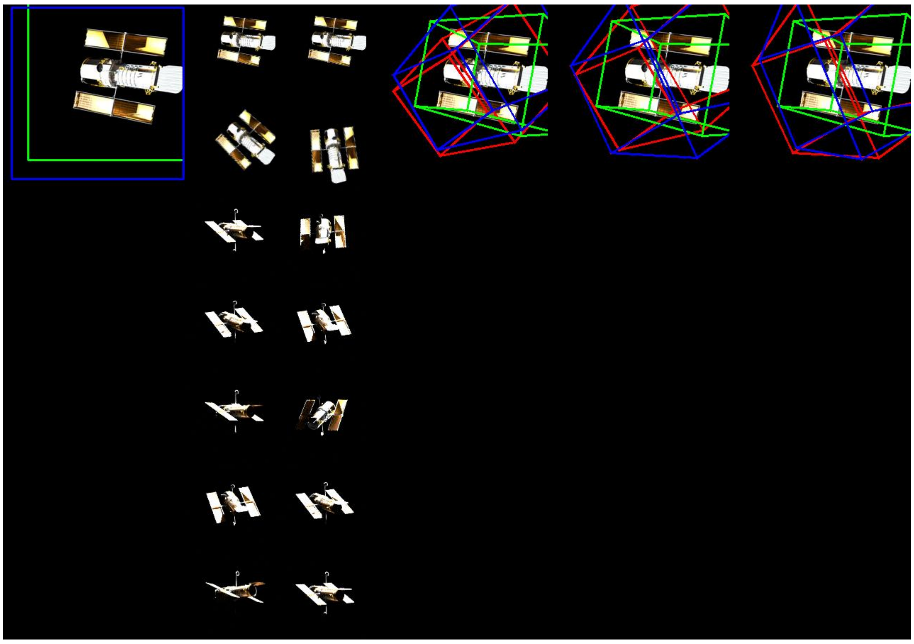

Figure 4.11: Hubble Space Telescope, no background, intermediary result, $e_{\text{ADD}} = 9.577$, $e_{\text{ADD-S}} = 5.196$

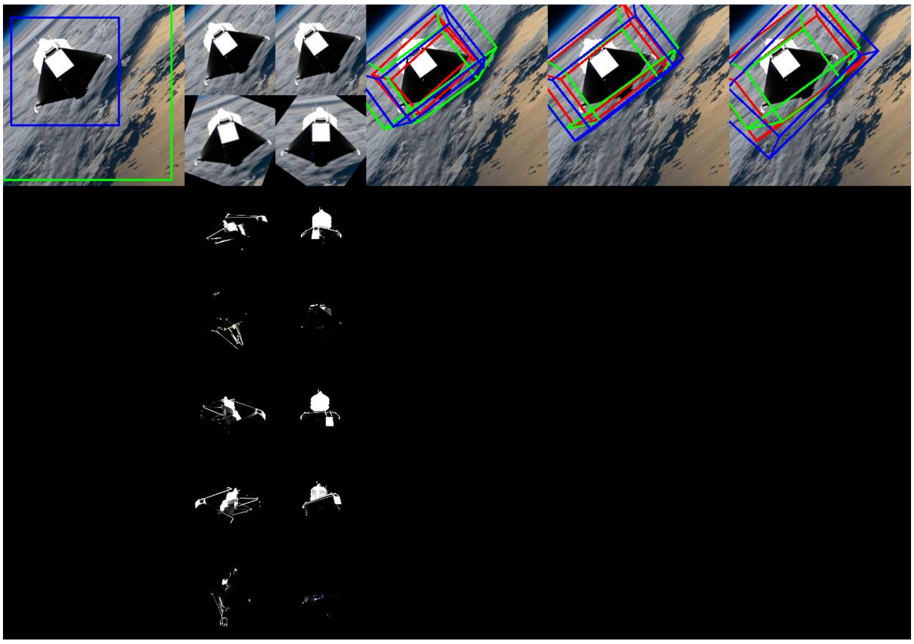

Figure 4.12: James Webb Space Telescope, with earth rendered background, intermediary result, $e_{\text{ADD}} = 10.934$, $e_{\text{ADD-S}} = 4.317$

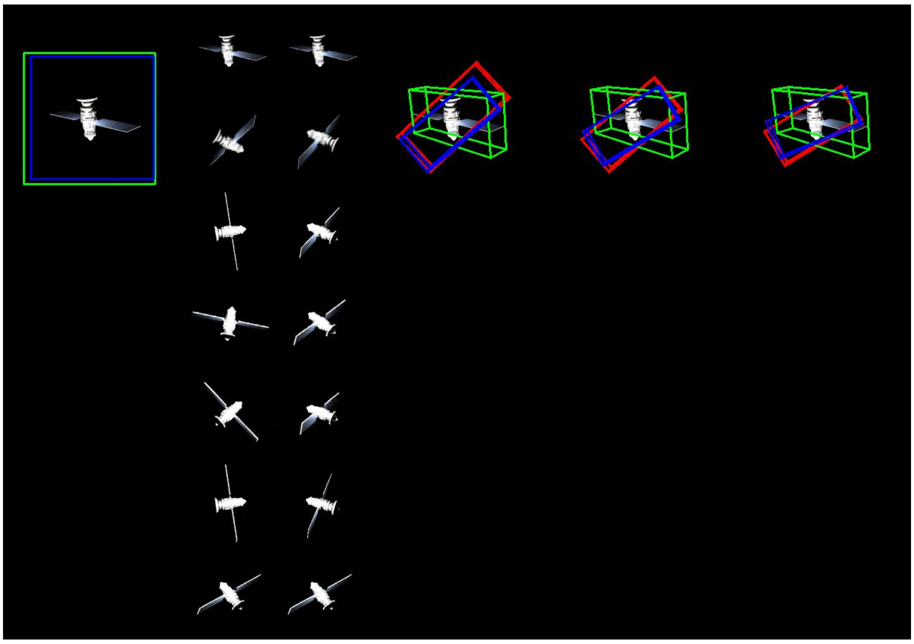

Figure 4.13: Cosmos Link, no background, intermediary result, $e_{\text{ADD}} = 11.094$, $e_{\text{ADD-S}} = 6.127$

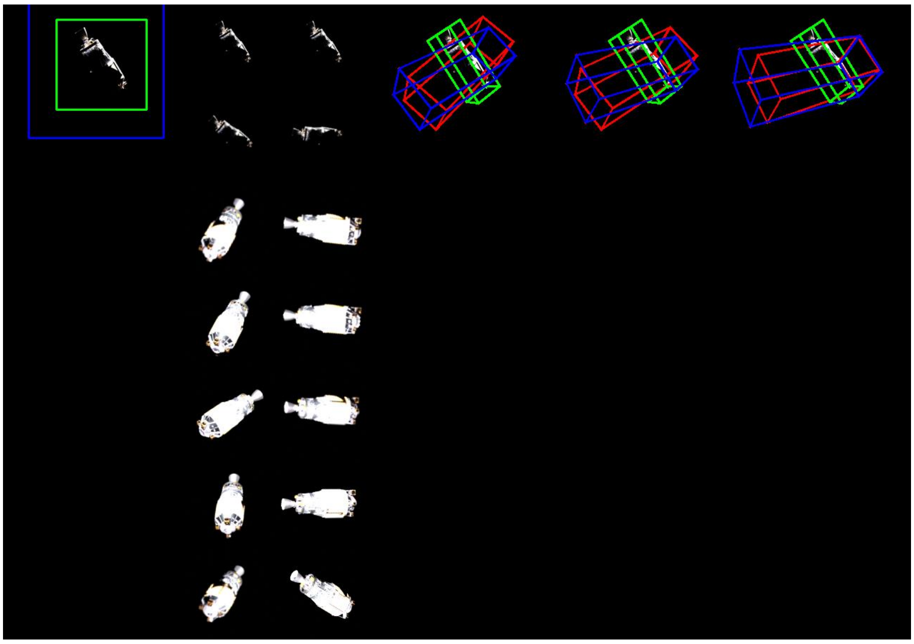

Figure 4.14: Rocket Body, no background, intermediary result, $e_{\text{ADD}} = 29.335$, $e_{\text{ADD-S}} = 17.743$

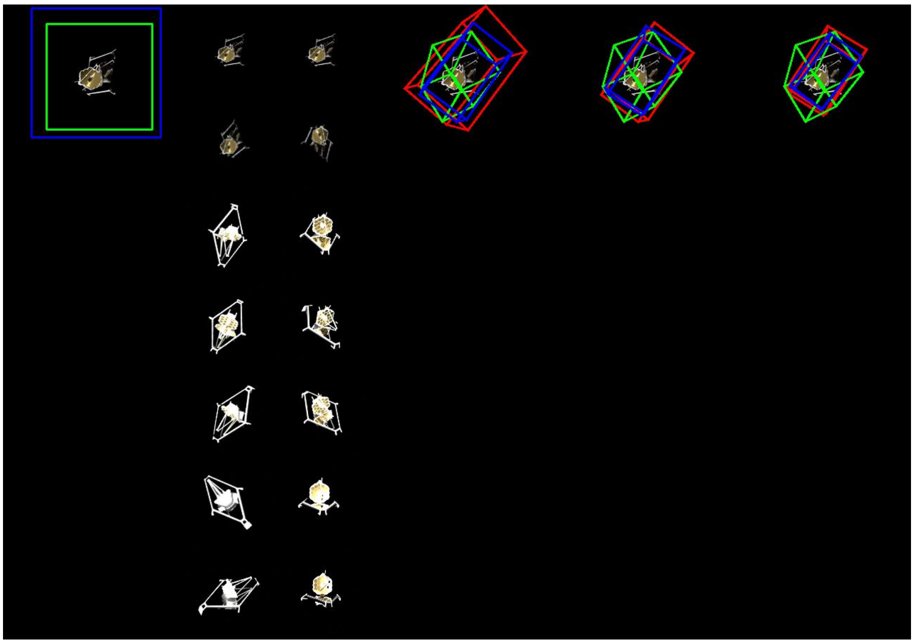

Figure 4.15: James Webb Space Telescope, with no background, intermediary result, $e_{\text{ADD}} = 21.983$, $e_{\text{ADD-S}} = 12.358$

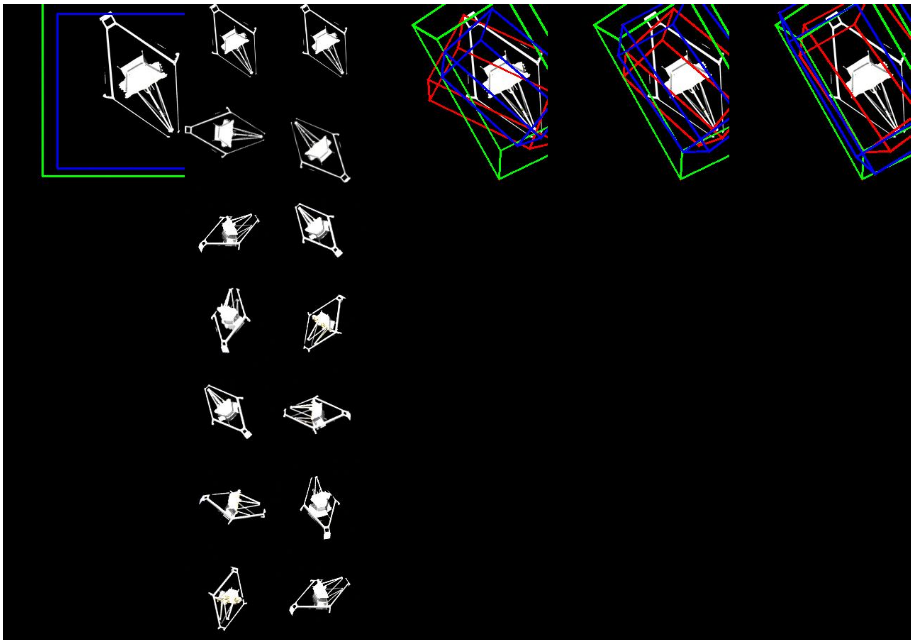

Figure 4.16: James Webb Space Telescope, with no background, intermediary result, $e_{\text{ADD}} = 1.060$, $e_{\text{ADD-S}} = 0.556$

# 5. Ways of improvements

## 5.1. Specialized spacecraft training set

## 5.2. Improved object detection algorithms

Rely more on the 3D model (for now only the size) and the segmented images, would optimize for symmetric and irregular shaped spacecrafts

## 5.3. Robustness to occlusion

# 6. Conclusion

Limitations Acknowledgments My personal contribution

# Abbreviations

**ADD** Average Distance of Model Points

**ADD-S** Average Closest Point Distance

# A. Python Scripts

```python
"""
Author:      Jeremy Chaverot
Date:        November 29, 2023
Description: Create the files val.txt, train.txt and test.txt according to a test percentage
"""

import os
import sys
import random


if __name__ == "__main__":

  # Check if the correct number of arguments is provided
    if len(sys.argv) != 3:
        print("Usage: python format.py <object_name> <test_percentage>")
        sys.exit(1)

    object = sys.argv[1]
    test_percentage = float(sys.argv[2])

    if (test_percentage < 0 or 1 < test_percentage):
        print("Wrong value for the variable <test_percentage>. Should be between 0 and 1 included.")
        sys.exit(1)

    # Get a list of all files in the folder
    all_files = os.listdir(f'data/SpaceCraft/{object}/images')

    # Filter the list to include only image files and exclude MacOS temporary files
    image_files = [file for file in all_files if file.lower().endswith(('.jpg')) and not file.startswith('._')]

    # Get the number of images in the folder
    num_images = len(image_files)

    # Iterate through each image and apply the transformation
    with open(f'data/SpaceCraft/{object}/train.txt', 'w') as train, open(f'data/SpaceCraft/{object}/test.txt', 'w') as test:
        for image_file in image_files:
            rand = random.random()
            image_path = 'SpaceCraft/hubble/images/' + image_file
            if (rand < test_percentage):
                test.write(image_path + '\n')
            else: train.write(image_path + '\n')

    print(f"Done splitting {num_images} images in train.txt and test.txt")
```

Listing A.1: Python script `format.py` to randomly generate the training set and the test set based on a specified probability. Should be run from Gen6D's root folder.

```python
"""
Author:      Jeremy Chaverot
Date:        November 20, 2023
Description: Transform every images of a folder into jpg format.
"""

import os
```

```python
import sys
from PIL import Image


def transform_image(image_path):
    img = Image.open(image_path)
    new_image_path = image_path.split('.')[0] + '.jpg'
    img.save(new_image_path)


if __name__ == "__main__":

  # Check if the correct number of arguments is provided
    if len(sys.argv) != 2:
        print("Usage: python to_jpg.py </path/to/your/images>")
        sys.exit(1)

    folder_path = sys.argv[1]

    # Get a list of all files in the folder
    all_files = os.listdir(folder_path)

    # Filter the list to include only image files and exclude MacOS temporary files
    image_files = [file for file in all_files if file.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp')) and not file.startswith('._')]

    # Get the number of images in the folder
    num_images = len(image_files)

    # Iterate through each image and apply the transformation
    for image_file in image_files:
        image_path = os.path.join(folder_path, image_file)
        transform_image(image_path)
        os.remove(image_path)

    print(f"Number of images transformed into .jpg: {num_images}")
```

Listing A.2: Python script to_jpg.py to transform every images of a specified folder into jpg format.

```python
"""
Author:      Jeremy Chaverot
Date:        November 20, 2023
Description: Transform a txt file with quaternions and the translation vector into multiple npy files containing the rotation matrix concatenated with the translation vector.
"""

import numpy as np
import sys
import os


def quaternion_to_matrix(Q, translation):
    """
        Covert a quaternion and translation into a full three-dimensional augmented rotation matrix.

        Input
        :param Q: A 4 element array representing the quaternion (q0, q1, q2, q3).
        :param translation: A 3 element array representing the translation (x, y, z).

        Output
        :return: A 3x4 element matrix representing the 3D rotation matrix concatenated with the translation vector.
    """

    # Extract the values from arguments
    q0 = Q[0]
    q1 = Q[1]
    q2 = Q[2]
```

```python
    q3 = Q[3]

    x = translation[0]
    y = translation[1]
    z = translation[2]

    # Compute the rotation matrix
    r00 = 2 * (q0 * q0 + q1 * q1) - 1
    r01 = 2 * (q1 * q2 - q0 * q3)
    r02 = 2 * (q1 * q3 + q0 * q2)

    r10 = 2 * (q1 * q2 + q0 * q3)
    r11 = 2 * (q0 * q0 + q2 * q2) - 1
    r12 = 2 * (q2 * q3 - q0 * q1)

    r20 = 2 * (q1 * q3 - q0 * q2)
    r21 = 2 * (q2 * q3 + q0 * q1)
    r22 = 2 * (q0 * q0 + q3 * q3) - 1

    # 3x3 rotation matrix concatenated with the 3x1 translation vector
    matrix = np.array([[r00, r01, r02, x],
                       [r10, r11, r12, y],
                       [r20, r21, r22, z]])

    return matrix


if __name__ == "__main__":

    # Check if the correct number of arguments is provided
    if len(sys.argv) != 3:
        print("Usage: python quaternion_to_matrix.py </path/to/your/text/file> </path/to/the/pose/folder>")
        sys.exit(1)

    file_path = sys.argv[1]
    pose_folder_path = sys.argv[2]
    file_content = None

    try:
        with open(file_path, 'r') as file:
            file_content = file.read()
    except FileNotFoundError:
        print(f"The file {file_path} was not found.")
        sys.exit(1)
    except Exception as e:
        print(f"An error occurred: {e}")
        sys.exit(1)

    poses = file_content.split('\n')[:-1]

    # Iterate through each pose, apply the transformation and save it
    for pose in poses:
        image_id, obj_id, q0, q1, q2, q3, x, y, z = pose.split(',')
        Q = np.array([q0, q1, q2, q3], dtype=np.float32)
        translation = np.array([x, y, z], dtype=np.float32)
        matrix = quaternion_to_matrix(Q, translation)
        np.save(pose_folder_path + '/pose' + str(int(image_id)), matrix)

    print(f"Number of transformation processed: {len(poses)}.")
```

Listing A.3: Python script `quaternion_to_matrix.py` to transform a txt file with quaternions and the translation vector into multiple npy files containing the rotation matrix augmented with the translation vector.

```python
"""
Author:      Jeremy Chaverot
Date:        December 10, 2023
Description: Invert the masks from a given folder.
"""
```

```python

import cv2
import os
import sys


def inverse_masks_in_folder(folder_path):
  # Iterate through the list of files at the specified path
    for filename in os.listdir(folder_path):
      # Filter to include only png image files and exclude MacOS temporary files
        if filename.endswith(".png") and not filename.startswith('._'):
            mask_path = os.path.join(folder_path, filename)
            try:
                # Read the mask image
                mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
                if mask is None:
                    print(f"Failed to read image: {mask_path}")
                    continue

                # Invert the mask
                inverted_mask = cv2.bitwise_not(mask)

                # Save the inverted mask with a temporary name
                temp_path = os.path.join(folder_path, "temp_" + filename)
                cv2.imwrite(temp_path, inverted_mask)

                # Delete the original mask
                os.remove(mask_path)

                # Rename the inverted mask to the original filename
                os.rename(temp_path, mask_path)
                print(f"Inverted and replaced mask for: {mask_path}")
            except Exception as e:
                print(f"Error processing {mask_path}: {e}")


if __name__ == "__main__":

  # Check if the correct number of arguments is provided
    if len(sys.argv) != 2:
        print("Usage: python invert_mask.py <folder_path>")
        sys.exit(1)

    folder_path = sys.argv[1]
    inverse_masks_in_folder(folder_path)
```

Listing A.4: Python script `invert_mask.py` to invert the masks from a specified folder. We aim to have a black object set against a white background.

```python
"""
Author:      Jeremy Chaverot
Date:        January 01, 2024
Description: Resize the images from a given folder.
"""

import os
import sys
from PIL import Image


def resize_images(folder_path, resize_factor):
  # Iterate through the list of files at the specified path
    for filename in os.listdir(folder_path):
      # Filter to include only png image files and exclude MacOS temporary files
        if filename.endswith(".png") and not filename.startswith('._'):
            img_path = os.path.join(folder_path, filename)
            with Image.open(img_path) as img:
                # Calculate new size
                new_size = tuple([int(dim / resize_factor) for dim in img.size])
                # Resize the image
```

```
                resized_img = img.resize(new_size, Image.ANTIALIAS)
                # Save the resized image with a different name temporarily
                temp_path = os.path.join(folder_path, "temp_" + filename)
                resized_img.save(temp_path)

            # Delete the original image
            os.remove(img_path)

            # Rename the resized image to the original filename
            os.rename(temp_path, img_path)


if __name__ == "__main__":

  # Check if the correct number of arguments is provided
    if len(sys.argv) != 3:
        print("Usage: resize.py <folder_path> <resize_factor>")
        sys.exit(1)

    folder_path = sys.argv[1]
    factor = int(sys.argv[2])

    resize_images(folder_path, factor)
```

Listing A.5: Python script `resize.py` designed to alter an image's size with respect to a specified resize factor.

# B. Scitas Izar Setup Tutorial

# Bibliography


[1] Y. Liu, Y. Wen, S. Peng, C. Lin, X. Long, T. Komura, and W. Wang. *Gen6D: Generalizable Model-Free 6-DoF Object Pose Estimation from RGB Images*. 2023. arXiv: 2204.10776.

[2] G. S. 3Blue1Brown. *Quaternions and 3d rotation, explained interactively*. Youtube. 2018. URL: https://www.youtube.com/watch?v=zjMuIxRvygQ.

[3] W. R. Hamilton and W. E. Hamilton. *Elements of Quaternions*. London: Longmans, Green, & Company, 1866.

[4] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. "Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes." In: *Computer Vision – ACCV 2012*. Springer Berlin Heidelberg, 2013, pp. 548–562.