SCHOOL OF COMPUTER AND COMMUNICATION SCIENCES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Computer Vision Laboratory
Unseen Spacecraft Pose Estimation

Baseline solution implementation of a deep learning model for unseen spacecraft pose estimation

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, 26.01.24 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

Nowadays, the significant number of objects in Low Earth Orbit (LEO) is becoming a matter of concern, with the potential to create even more space debris. Therefore, it's becoming crucial to develop technologies to tackle this pollution. One can envision a clean-up mission where devices are equipped with quantized deep learning models to estimate the position of these objects for retrieval. Training these models is challenging, as the real image database is quite sparse. To overcome this, high-quality, realistic synthetic images of spacecraft are used, with the ultimate goal of generalizing to unseen objects.

Contents

1. Introduction

This project falls within an Unseen 6 Degrees of Freedom (DoF) competition, organized in collaboration with the European Space Agency (ESA) Advanced Concept Team. Essentially, we are dealing with space objects that are unfamiliar to us, and our objective is to accurately predict their 6DoF poses. The action of the Computer Vision Laboratory (CVLab) team is twofold: firstly, we are tasked with creating a challenging dataset featuring multi-object, unseen, and occluded spacecraft scenarios. This involves ensuring a high degree of rendering realism. Secondly, we are focused on developing a baseline solution, which entails implementing a chosen pose estimation model and conducting thorough training and testing on our dataset. My role this semester was primarily concentrated on the latter aspect, specifically on a track that incorporated 3D target models.

1.1. Problem Formulation

To put it simply, our task involves analyzing images of space objects with the aim of accurately determining the relative 3D translation and 3D rotation of the spacecraft in relation to the camera's position.

For the training and testing of the selected deep learning architecture, we utilize synthetic images provided by the SPACECRAFT dataset team. This dataset encompasses four distinct models: the Hubble Space Telescope, the James Webb Space Telescope, the Cosmos Link, and the Rocket Body. These images may feature an Earth-rendered background or be without it. In addition to the images, our dataset includes 3D models, masks, segmented images, and camera settings. Furthermore, the ground truth poses of the objects are available in a format that combines quaternions with a translation vector.

1.2. Applications

Beyond the scope of the ESA competition, the project has numerous practical applications in the aerospace industry.

Remediation of Space Debris in LEO The developed pose estimation models can significantly enhance the identification and tracking of space debris, enabling precise navigation for cleanup missions.

Space Monitoring The technology can also be utilized for the constant surveillance and cataloging of artificial and natural objects in space, improving awareness of the space environment and collision avoidance systems.

Planetary Defense Technology Finally we could think about the accurate pose estimation capabilities serving as a critical component in planetary defense missions by ensuring precise targeting of potentially hazardous small celestial bodies, for instance the NASA's Double Asteroid Redirection Test (DART) mission in 2021, which was the first-ever asteroid deflection mission through kinetic impact.

1.3. The Work Environment: EPFL Scitas Izar

We needed to establish a suitable environment for executing the code: EPFL Scitas Izar servers. They are are ideally configured for our task: equipped with two NVIDIA V100 PCIe 32 GB GPUs, the most advanced data center GPUs ever built to accelerate Artificial Intelligence (AI), High Performance Computing (HPC), data science and graphics (very expensive as well). Additionally, the server enables users to execute the code from any location, eliminating the need for a sufficiently powerful hardware configuration.

However, this stage proved to be more time-consuming than expected. It involved software engineering and various technical challenges, including setting up the virtual environment, installing the necessary dependencies in a manner that avoids conflicts with the modules already installed on the server, composing the bash execution script, and, fundamentally, learning the correct way to utilize the server. Special recognition goes to Emily Bourne from the EPFL HPC team for her essential help and support.

For those who wish to run the model, instructions for setting up the environment are provided in Appendix B of this report.

2. Gen6D: Formal Description

2.1. Overview of the Network

After conducting a scientific literature search, we decided on an interesting model known as Gen6D: Generalizable Model-Free 6-DoF Object Pose Estimation from RGB Images [1]. This architecture is fed with reference and query images, and operates in a relatively straightforward manner, comprising three stages shown in Figure 2.1. Initially, there's the detector, which, as its name suggests, detects the object in the query image. Next is the viewpoint selection, which selects reference images that have the closest viewing angle to the detected object in the query image by using a similarity score. Finally, there's the pose refiner, which enhances the accuracy of the estimated pose through a 3D volume-based process. We have compiled a table summarizing the advantages and disadvantages for a clearer understanding:

Table 2.1.: Summary of Gen6D

Pros	Cons
Generalizability	Limited by Reference Image Quality
Model-Free	Everyday Life Objects Training Data
Simple Input Requirements	Difficulty with Symmetric Objects
Robustness to Background Clutter	Dependence on Initial Detection and Selection
Effective in Diverse Environments	Potential Challenges with Severe Occlusions
Competitive Performance	Computationally Intensive

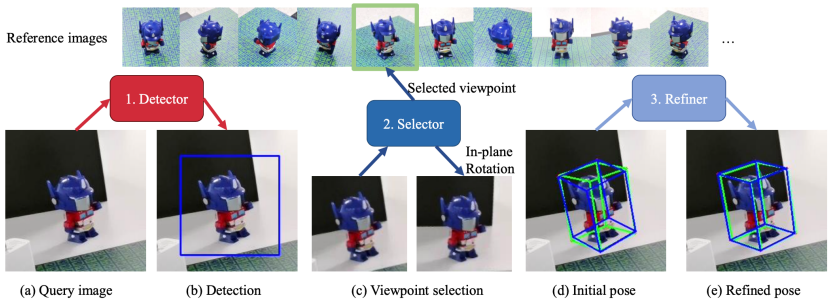


Figure 2.1: Overview of Gen6D, image sourced from Gen6D's research paper.

2.2. Detection

In the detection phase, the initial step involves resizing both reference and query images. The reference images are resized to a dimension of (128×128), with object

centered. The query images are resized at various predefined scales. Subsequently, all the resized images are processed through a Visual Geometry Group Network (VGGNet), which essentially functions as a deep convolutional network for extracting feature maps.

Following this, the feature maps from the reference images are utilized as convolutional kernels to convolve with one of the query images, resulting in the generation of a score map. Leveraging the multi-scale score map, we perform regression to obtain a heat map and a scale map. The final 2D detection is acquired by identifying the maximum value in the heat map, which provides the object's center coordinates, while the scale (s) is obtained from the same location in the scale map, by this way determining the size of the 2D bounding box.

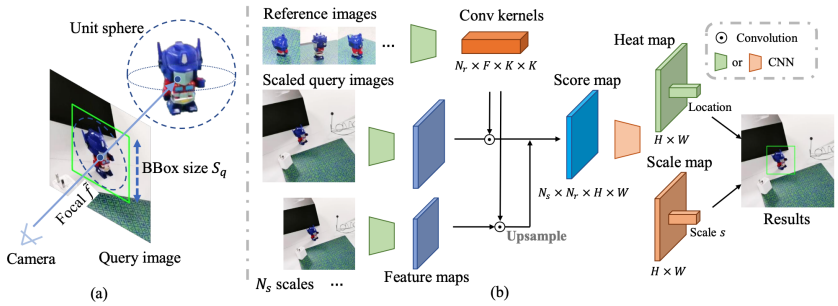


Figure 2.2: Architecture of the detector, image sourced from Gen6D's research paper.

2.3. Viewpoint Selection

During the viewpoint selection phase, we begin by applying predefined rotations to the reference images to accommodate in-plane rotation variations. After that, we once again extract feature maps from both the query image, which has been cropped based on the detection results, and the rotated reference images.

Next we calculate the element-wise product of the query image's features with those of each rotated reference image, yielding a correlation score map. These scores are then inputted into a similarity network, which produces a similarity score and the relative in-plane rotation.

It's worth highlighting that within the similarity network, the authors have implemented global normalization and incorporated a transformer to facilitate information sharing among the reference images.

2.4. Pose Refinement

From the two previous stages, we have an initial coarse object pose. In this refinement process, we one last time extract feature maps from the selected reference images using a 2D Convolutional Neural Network (CNN). Afterwards, these feature maps are unprojected into the 3D volume, and we compute their mean and variance, used as features associated with the volume vertices.

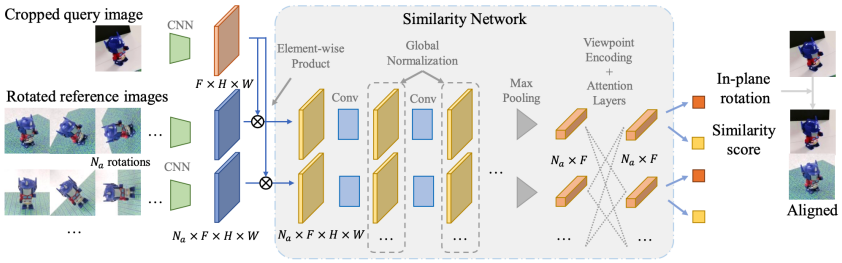
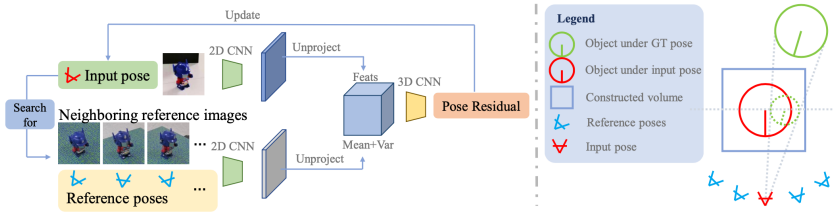


Figure 2.3: Architecture of the viewpoint selector, image sourced from Gen6D's research paper.

Similarly, for the query image, we perform the same process but utilize the input pose. The unprojected query image features are then concatenated with the previously obtained mean and variance features.

At the end, we have a 3D CNN that operates on the concatenated feature set of the volume, so as to predict a pose residual that updates the input pose. We iterate through the loop three times, which has been determined empirically as the optimal number of steps, in order to enhance the accuracy of the estimated pose.



(a) Architecture of the pose refiner (b) Similarity approximation

Figure 2.4: Architecture of the pose refiner, image sourced from Gen6D's research paper.

3. Implementation of the model

3.1. Data Loader

Now that we have a better understanding of how Gen6D operates, we're interested in testing it with the synthetic images from the SpaceCraft dataset. To accomplish this, we need to redefine the Data Loader to supply the model with the necessary data in the required format. In the file `database.py`, located in the `Gen6D/dataset` directory, there is an Abstract Base Class (ABC) named `BaseDatabase()` containing all the essential methods, as shown in the code below:

```
class BaseDatabase(abc.ABC):
  def __init__(self, database_name):
      self.database_name = database_name

  @abc.abstractmethod
  def get_image(self, img_id):
      pass

  @abc.abstractmethod
  def get_K(self, img_id):
      pass

  @abc.abstractmethod
  def get_pose(self, img_id):
      pass

  @abc.abstractmethod
  def get_img_ids(self):
      pass

  def get_mask(self,img_id):
      # dummy mask
      img = self.get_image(img_id)
      h, w = img.shape[:2]
      return np.ones([h,w],np.bool_)
```

Listing 3.1: Python code of ABC `BaseDatabase()`, from file `database.py`.

Our goal, therefore, is to write a new class that inherits from the ABC `BaseDatabase()`, and implement each abstract method in the same manner as it was done for the Linemod and GenMOP datasets, for example. We will also adapt the code in various places, especially in the `eval.py` file, to ensure compatibility with the SpaceCraft dataset. The Python code for the new data loading class `SpaceCraftCVLabDatabase()` can be found in Listing A.1.

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 [3].

Quaternions were first introduced by the Irish mathematician W. R. Hamilton in 1843 as an extension of the complex numbers [2]. 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,

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

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 $\mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R} : (\mathbf{p}, \mathbf{q}) \mapsto \mathbf{p} \cdot \mathbf{q} \stackrel{\text{def.}}{=} \mathbf{p}^\mathsf{T}\mathbf{q}$ is the *dot product*, and the other

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

is the *cross product*, and the operator $[\mathbf{p}]_\times$ yields 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

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

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

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

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

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

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 $\mathbf{u} \stackrel{\text{def.}}{=} \frac{\mathbf{q}}{|\mathbf{q}|}$.

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

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

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

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

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

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:

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

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:

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

$$
= 2 \begin{bmatrix} ({q_0}^2 + {q_1}^2) - \frac{1}{2} & q_1q_2 - q_0q_3 & q_1q_3 + q_0q_2 \\ q_1q_2 + q_0q_3 & ({q_0}^2 + {q_2}^2) - \frac{1}{2} & q_2q_3 - q_0q_1 \\ q_1q_3 - q_0q_2 & q_2q_3 + q_0q_1 & ({q_0}^2 + {q_3}^2) - \frac{1}{2} \end{bmatrix}
$$

This completes our derivation of the rotation matrix $\mathbf{R}$. The Python code can be found in Listing A.4.

3.3. Inverted Masks

During the initial testing of Gen6D on the SpaceCraft dataset, we noticed that the estimated poses were particularly inaccurate. Upon investigation, we discovered that in the reference images folder, the masks were incorrectly inverted: typically, one expects a white object on a black background, but it was the opposite (see Figure 3.1 below).

After discussing this, we realized that this was actually an error in the current version of the dataset. Therefore, we needed to correct the masks we were using through a Python script utilizing the OpenCV library.

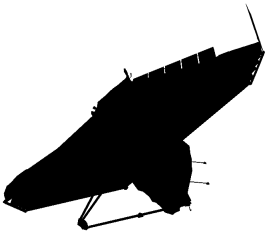


Figure 3.1: James Webb Space Telescope, mask before correction.

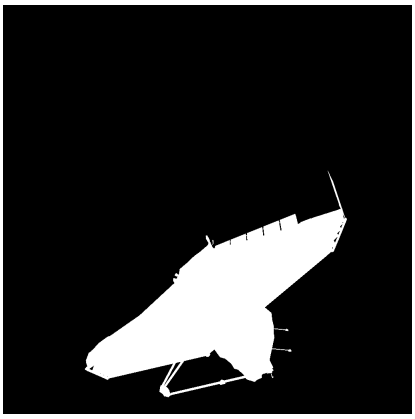


Figure 3.2: James Webb Space Telescope, mask after correction using a script.

The Python script can be found in Listing A.5.

3.4. Resized Query Images

During the initial testing of Gen6D, it became apparent that the deep learning model struggled when the object in the query image was either very large or very small (see Figure 3.3 below). Consultation of the GitHub Issues section of the Gen6D repository revealed that this is a recurring problem: since the reference images are resized to 128×128, the model's detector is trained to recognize an object in query images that ranges from half to double the size of the reference images, i.e., from 64×64 to 256×256.

Therefore, to improve the detector's results, resizing the query images is necessary. This is accomplished using a Python script that can be found in Listing A.6. It's important to note that this adjustment also requires modifying the camera intrinsic matrix $\mathcal{K}$ in the data loader, which converts points from the camera coordinate system to the pixel coordinate system.

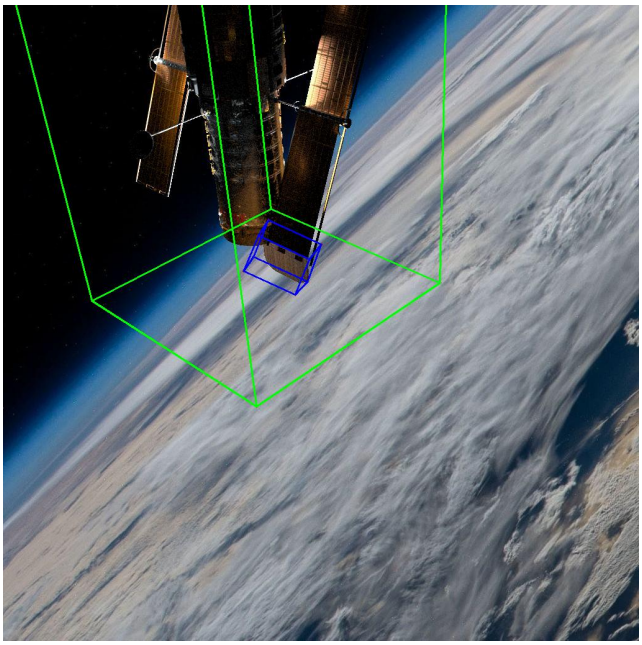


Figure 3.3: Hubble Space Telescope with earth rendered background, 1024x1024 query image number 1, poor pose estimation (blue 3D bounding box) when the object inside the query image is too big.

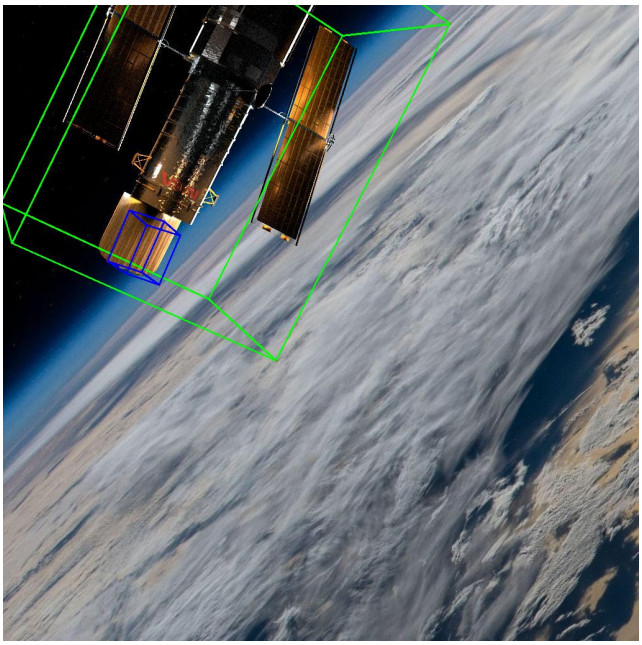


Figure 3.4: Hubble Space Telescope with earth rendered background, 1024x1024 query image number 2, poor pose estimation (blue 3D bounding box) when the object inside the query image is too big.

4. Experimental Results and Analysis

4.1. 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:

$$
\begin{aligned}
{}^{1}e_{\text{ADD}}(\hat{\mathbf{P}}, \bar{\mathbf{P}}, \mathcal{M}) &\overset{\text{def.}}{=} \operatorname*{avg}_{\mathbf{x}\in\mathcal{M}} \left\| \bar{\mathbf{P}}\mathbf{x}^{\star} - \hat{\mathbf{P}}\mathbf{x}^{\star} \right\|_2 && (4.1)\\
&= \operatorname*{avg}_{\mathbf{x}\in\mathcal{M}} \left\| (\bar{\mathbf{R}}\mathbf{x} + \bar{\mathbf{T}}) - (\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{T}}) \right\|_2 && (4.2)
\end{aligned}
$$

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:

$$
\begin{aligned}
e_{\text{ADD-S}}(\hat{\mathbf{P}}, \bar{\mathbf{P}}, \mathcal{M}) &\overset{\text{def.}}{=} \operatorname*{avg}_{\mathbf{x}_1\in\mathcal{M}} \min_{\mathbf{x}_2\in\mathcal{M}} \left\| \bar{\mathbf{P}}\mathbf{x}_1^{\star} - \hat{\mathbf{P}}\mathbf{x}_2^{\star} \right\|_2 && (4.3)\\
&= \operatorname*{avg}_{\mathbf{x}_1\in\mathcal{M}} \min_{\mathbf{x}_2\in\mathcal{M}} \left\| (\bar{\mathbf{R}}\mathbf{x}_1 + \bar{\mathbf{T}}) - (\hat{\mathbf{R}}\mathbf{x}_2 + \hat{\mathbf{T}}) \right\|_2 && (4.4)
\end{aligned}
$$

It's important to point out that $e_{\text{ADD-S}}$ is more lenient compared to e_{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.

4.2. Vizualisation and Quantitative Evaluation

With all these tools and the modifications made, we are now in a position to test Gen6D as-is, that is without retraining the model. For each spacecraft, we select approximately 200 reference images from a random image folder, and about ten hand-picked query images to ensure the best possible conditions. Indeed, we aim for

[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.

the object to be in the foreground, fully within the frame, and under good lighting conditions. Since we are on the ESA competition track with 3D target models included, we skip the Structure From Motion (SFM) step with Colmap. Below are the best pose estimations made by Gen6D for each of the four objects.

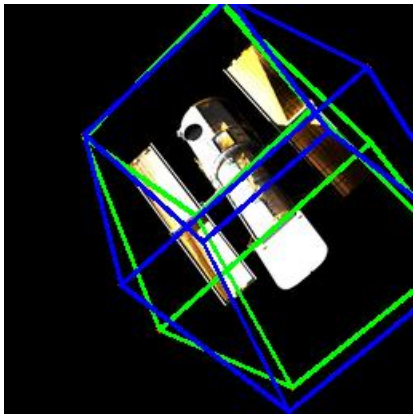


Figure 4.1: Hubble Space Telescope, no background, 256x256 query image, $e_{\text{ADD}} = 2.925$, $e_{\text{ADD-S}} = 1.183$

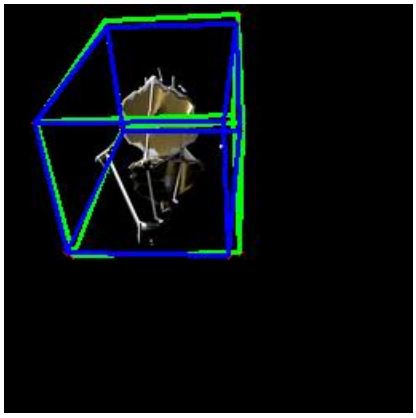


Figure 4.2: James Webb Space Telescope, no background, 256x256 query image, $e_{\text{ADD}} = 1.415$, $e_{\text{ADD-S}} = 0.808$

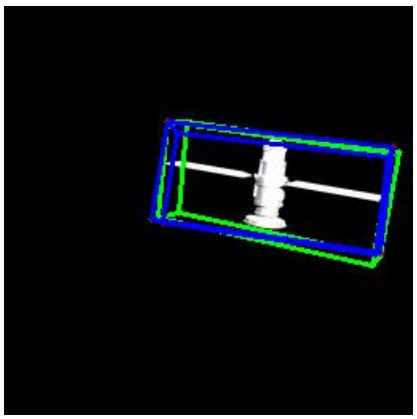


Figure 4.3: Cosmos Link, no background, 256x256 query image, $e_{\text{ADD}} = 1.718$, $e_{\text{ADD-S}} = 0.383$

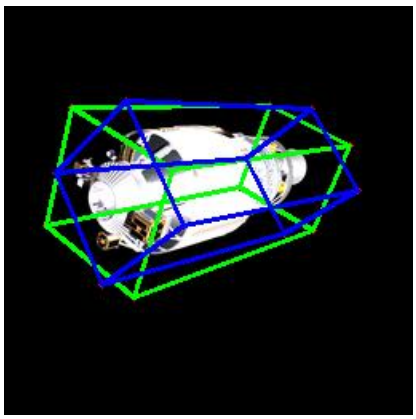


Figure 4.4: Rocket Body, no background, 256x256 query image, $e_{\text{ADD}} = 1.713$, $e_{\text{ADD-S}} = 0.252$

From our observations, it's notable that the results are particularly less accurate with the Hubble Space Telescope. This seems to stem from errors in the ground truth poses, as pointed out by Haochen, another student working on the project, particularly an axis inversion. Also, we often notice that the estimations on the Rocket Body are rotated around the object's axis of symmetry compared to the ground truth pose. It is thus relevant to consider the ADD-S metric in this case.

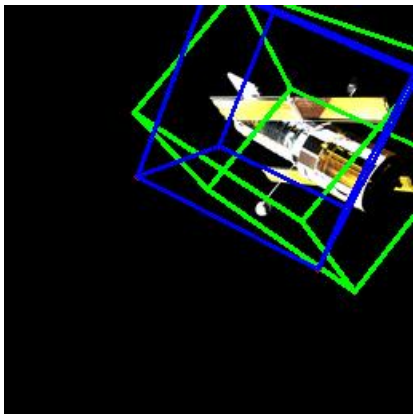


Figure 4.5: Hubble Space Telescope, no background, 256x256 query image, $e_{\text{ADD}} = 6.514$, $e_{\text{ADD-S}} = 1.571$

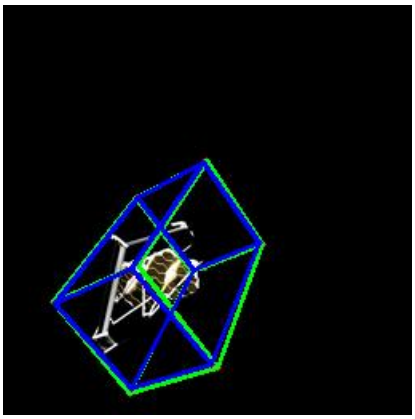


Figure 4.6: James Webb Space Telescope, no background, 256x256 query image, $e_{\text{ADD}} = 2.224$, $e_{\text{ADD-S}} = 1.261$

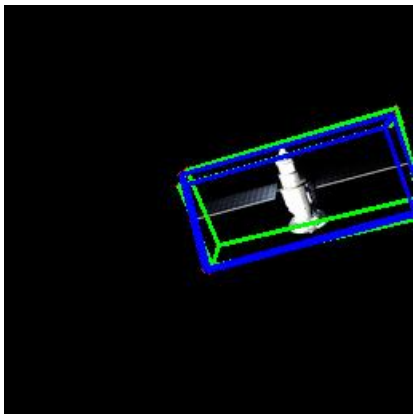


Figure 4.7: Cosmos Link, no background, 256x256 query image, $e_{\text{ADD}} = 1.925$, $e_{\text{ADD-S}} = 0.377$

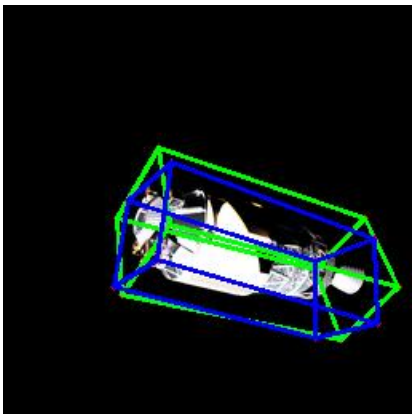


Figure 4.8: Rocket Body, no background, 256x256 query image, $e_{\text{ADD}} = 1.982$, $e_{\text{ADD-S}} = 0.501$

While the above results are great, the issue is that in many instances, the estimation is rather inaccurate. For completeness, we are now studying cases where Gen6D doesn't perform well, and we are trying to identify the reasons behind these shortcomings. The following text is provided by the authors to offer a clearer understanding of the intermediary results presented below: "[The column of vertical images] shows the viewpoint selection results. The first row shows the input image to the selector. The second row shows the input images rotated by the estimated in-plane rotation (left column) or the ground-truth in-plane rotation(right column) Subsequent 5 rows show the predicted (left) or ground-truth (right) 5 reference images with nearest viewpoints to the input image."

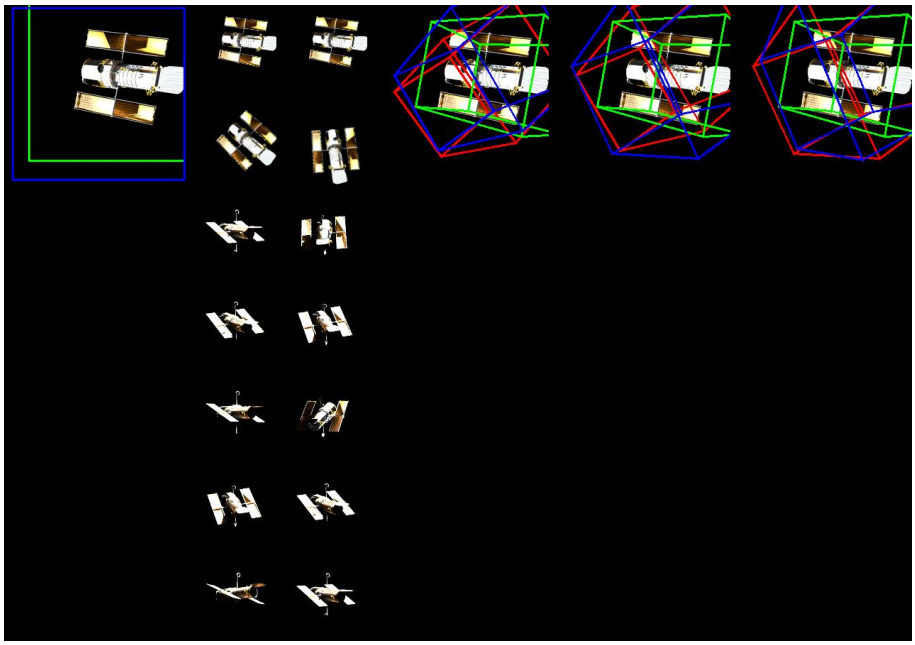


Figure 4.9: Hubble Space Telescope, no background, intermediary result of a poor estimation, $e_{\text{ADD}} = 9.577$, $e_{\text{ADD-S}} = 5.196$

In Figure 4.9, we observe good detection of the object, however, the selection of the nearest viewpoint is lacking, ultimately leading to an inaccurate pose estimation. This supports the earlier explanations regarding errors in the ground truth poses for the Hubble Space Telescope.

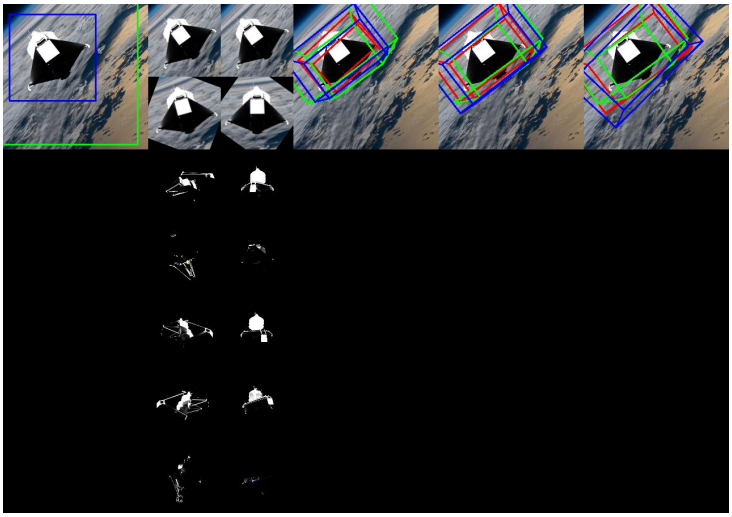


Figure 4.10: James Webb Space Telescope, with earth rendered background, intermediary result of a poor estimation, $e_{\text{ADD}} = 10.934$, $e_{\text{ADD-S}} = 4.317$

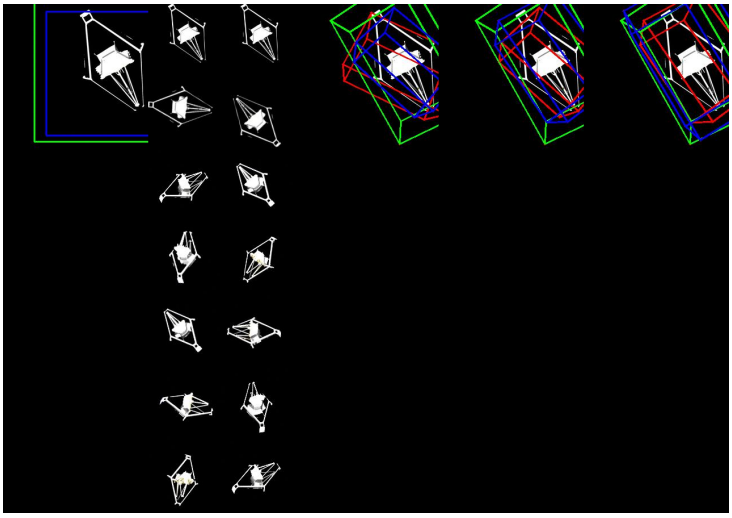


Figure 4.11: James Webb Space Telescope, no background, intermediary result, $e_{\text{ADD}} = 1.060$, $e_{\text{ADD-S}} = 0.556$

In Figure 4.10, we tested the model using reference images without a background and query images featuring the James Webb Telescope in front of the blue planet. While the object detection appears reasonably accurate, there is a significant difference in lighting conditions between the reference and query images. On the positive side, the refiner seems to work effectively as we observe an improvement in pose accuracy over the course of the three steps. Figure 4.11 is another example of the refiner doing great.

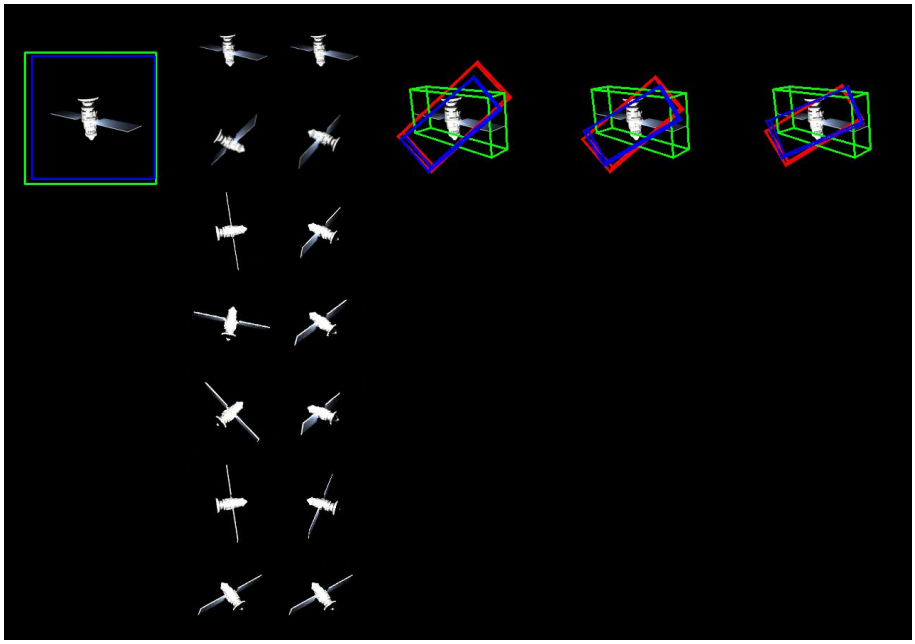


Figure 4.12: Cosmos Link, no background, intermediary result of a poor estimation, $e_{\text{ADD}} = 11.094$, $e_{\text{ADD-S}} = 6.127$

In Figure 4.12, we have a perfect detection, reasonably favorable viewpoints and yet a poor estimated pose at the end. However, it appears that increasing the number of refinement steps could lead to an improvement.

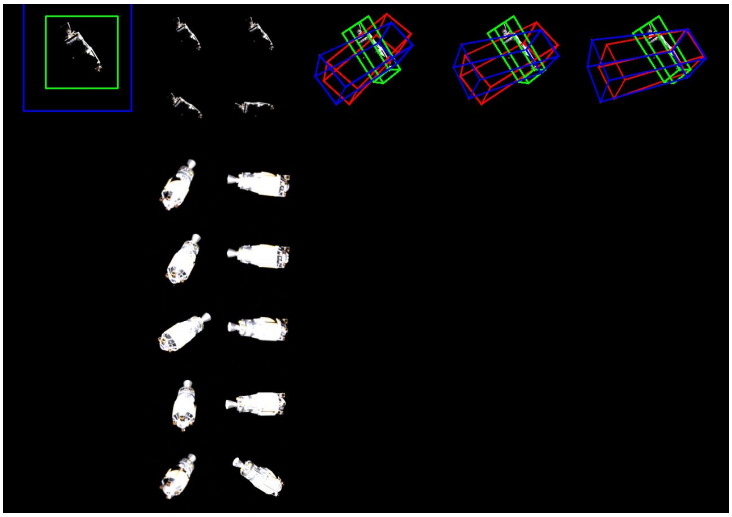


Figure 4.13: Rocket Body, no background, intermediary result of a poor estimation, $e_{\text{ADD}} = 29.335$, $e_{\text{ADD-S}} = 17.743$

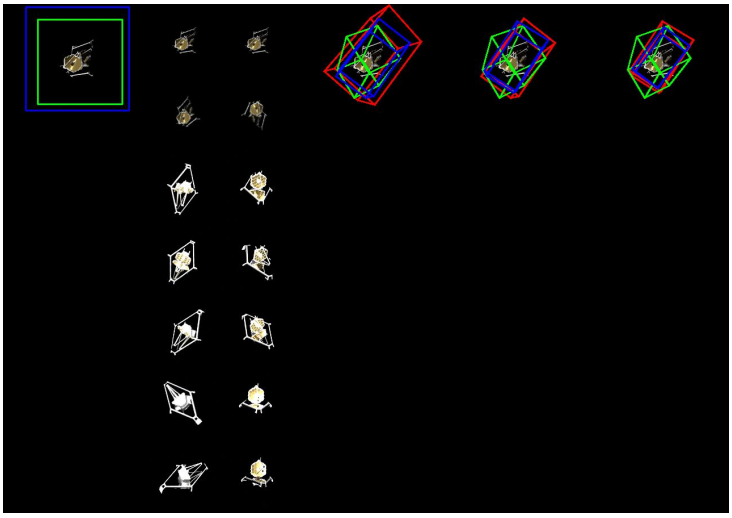


Figure 4.14: James Webb Space Telescope, no background, intermediary result of a poor estimation, $e_{\text{ADD}} = 21.983$, $e_{\text{ADD-S}} = 12.358$

In Figure 4.13 and Figure 4.14, we are encountering large differences in terms of lighting conditions. Specifically, for the James Webb Space Telescope, the query image reveals the interior of the antenna, which is not present in any of the selected viewpoints. Therefore, we conclude that Gen6D struggles with accurately estimating poses when there are substantial variations in lighting conditions.

5. Ways of improvements

As we observed in the previous chapter, Gen6D does not perform very well when generalizing to space objects. In this section, we will attempt to provide some tracks for improving the results of pose estimations.

5.1. More diverse Reference Images

Given the time constraints, we evaluated Gen6D under basic conditions, by using reference images from a folder of random synthetic images. To enhance the model's performance, one approach would be to diversify as much as possible the external conditions affecting the object's image through the camera, particularly lighting. As we observed, lighting plays a significant role during viewpoint selection. Having a more varied set of reference images would enable the model to better adapt to new conditions. Additionally, it's essential to ensure that the reference images are distributed as homogeneously as possible around the object.

5.2. Specialized Spacecraft Training Set

To assess the ability of Gen6D to generalize, we opted not to retrain the model but instead utilized the pretrained model made available by the authors. This approach may introduce a challenge, as highlighted in Table 2.1, where one of Gen6D's limitations is its training on everyday objects under favorable lighting conditions.

A viable solution would be to retrain Gen6D, focusing on specific components rather than the entire model. Specifically, retraining only the detector and the viewpoint selector, while keeping the refiner as is, could be effective. Indeed, retraining the detector would allow for the addition of new detecting scales for the object in query images, improving the accuracy of the estimated 2D bounding box. Additionally, retraining the viewpoint selector could prove beneficial for making pose estimations under poor lighting conditions. As for the refiner, as previously observed, it seems to perform pretty well and may not require retraining.

5.3. Improve Object Detection and Viewpoint Selection Algorithms

A limitation of the current architecture of Gen6D is indeed its detector: it requires predefined object scales in the query images for detection. This is a significant constraint, especially in competitive scenarios where the object could be very close or very far away. Additionally, the current necessity to resize query images leads to information loss and adds a manual step to the process.

To address this, the implementation of a Feature Pyramid Network (FPN) could be a substantial improvement [5]. FPNs are known for their efficiency in detecting

objects at various scales through the utilization of a pyramidal hierarchy of deep convolutional networks. Incorporating a FPN into Gen6D's architecture would enhance its ability to detect objects regardless of their distance or size in the query images, thus potentially improving overall performance.

During the detection and viewpoint selection phase, we could also think about relying more on the 3D model (for now it is only used to get the estimated 3D bounding box size) and the segmented images (currently unused), to get better results.

6. Conclusion

In summary, after overcoming several software engineering challenges, we successfully reimplemented Gen6D for the SpaceCraft dataset. While the current results are somewhat disappointing, we have identified several promising tracks for future exploration to enhance the pose estimation accuracy.

Knowing that the dataset team has developed new renderings this semester with dozens of new models, there is ample opportunity to further test Gen6D. More importantly, this development opens up the potential for retraining the model.

From a personal standpoint, this project was my first experience working as part of a significant and motivated team, and I thoroughly enjoyed the experience. It taught me to better organize my work, to operate on a remote server like Scitas Izar, to understand the functioning of a deep learning architecture in a practical, applied setting, to write Python scripts, and finally to improve my English skills through Zoom meetings and the writing of this very report.

Abbreviations

ABC Abstract Base Class

ADD Average Distance of Model Points

ADD-S Average Closest Point Distance

AI Artificial Intelligence

CNN Convolutional Neural Network

CVLab Computer Vision Laboratory

DART Double Asteroid Redirection Test

DoF Degrees of Freedom

ESA European Space Agency

FPN Feature Pyramid Network

HPC High Performance Computing

LEO Low Earth Orbit

SFM Structure From Motion

VGGNet Visual Geometry Group Network

A. Python Scripts

```
SPACECRAFT_ROOT='data/SpaceCraft'
class SpaceCraftCVLabDatabase(BaseDatabase):
    def __init__(self, database_name):
        super().__init__(database_name)
        _, self.model_name = database_name.split('/')
        self.img_ids = [str(k) for k in range(len(os.listdir(f'{SPACECRAFT_ROOT}/{self.model_name}/images')))]
        self.model = self.get_ply_model().astype(np.float32)
        self.object_center = np.zeros(3, dtype=np.float32)
        self.object_vert = np.asarray([0,0,1], np.float32)
        self.img_id2depth_range = {}
        self.img_id2pose = {}
        _, self.database_type = self.model_name.split('-')
        # Track 1: SpaceCraft dataset camera intrinsic matrix
        self.K=np.array([[1406.708374, 0.000000, 512.000000], [0.000000, 1406.708374, 512.000000], [0.000000, 0.000000, 1.000000]], dtype=np.float32)
        # Track 2: SwissCube dataset camera intrinsic matrix
        # self.K=np.array([[607.5730322397038, 0.000000, 512.000000], [0.000000, 607.5730322397038, 512.000000], [0.000000, 0.000000, 1.000000]], dtype=np.float32)
        if self.database_type == 'test':
            resize_factor = np.loadtxt(f"{SPACECRAFT_ROOT}/{self.model_name}/resize_factor.txt")
            self.K *= resize_factor
            self.K[2, 2] = 1
        # Note : This following portion of code is to uncomment when the ground truth poses are store in a JSON file,
        #        like for the SwissCube dataset.
        # with open(f'{SPACECRAFT_ROOT}/{self.model_name}/scene_gt.json', 'r') as file:
        #     self.pose = json.load(file)

    def get_ply_model(self):
        fn = Path(f'{SPACECRAFT_ROOT}/{self.model_name}/{self.model_name}.pkl')
        if fn.exists(): return read_pickle(str(fn))
        ply = plyfile.PlyData.read(f'{SPACECRAFT_ROOT}/{self.model_name}/{self.model_name}.ply')
        data = ply.elements[0].data
        x = data['x']
        y = data['y']
        z = data['z']
        model = np.stack([x, y, z], axis=-1)
        if model.shape[0]>4096:
            idxs = np.arange(model.shape[0])
            np.random.shuffle(idxs)
            model = model[idxs[:4096]]
        save_pickle(model, str(fn))
        return model

    def get_image(self, img_id):
        # Track 1: SpaceCraft dataset image format
```

```
44          return imread(f'{SPACECRAFT_ROOT}/{self.model_name}/images/{int(img_id)
    :04}.png')
45          # Track 2: SwissCube dataset image format
46          # return imread(f'{SPACECRAFT_ROOT}/{self.model_name}/images/{int(img_id
    ):06}.jpg')
47
48      def get_K(self, img_id):
49          return np.copy(self.K)
50
51      def get_pose(self, img_id):
52          # Track 1: Gen6D's pose format
53          if img_id in self.img_id2pose:
54              return self.img_id2pose[img_id]
55          else:
56              pose = np.load(f'{SPACECRAFT_ROOT}/{self.model_name}/pose/pose{int(
    img_id)}.npy')
57              self.img_id2pose[img_id] = pose
58              return pose
59          # Note : This following portion of code is to uncomment when the ground
    truth poses are store in a JSON file,
60          #        like for the SwissCube dataset. Do not forget to comment track
    1 above.
61          # Track 2: SwissCube dataset json pose format
62          # raw_pose = (self.pose[f'{int(img_id) + 1}'])[0]
63          # rot_mat = np.asarray(raw_pose['cam_R_m2c'])
64          # rot_mat = np.reshape(rot_mat, (3, 3))
65          # transl_vec = np.asarray(raw_pose['cam_t_m2c'])
66          # transl_vec = np.reshape(transl_vec, (3, 1))
67          # pose = np.concatenate((rot_mat, transl_vec), axis=1)
68          # return pose
69
70      def get_img_ids(self):
71          return self.img_ids.copy()
72
73      def get_mask(self, img_id):
74          # Track 1: SpaceCraft dataset image format
75          return np.sum(imread(f'{SPACECRAFT_ROOT}/{self.model_name}/mask/{int(
    img_id):04}.png'),-1)>0
76          # Track 2: SwissCube dataset image format
77          #return np.sum(imread(f'{SPACECRAFT_ROOT}/{self.model_name}/mask/{int(
    img_id):06}.png'),-1)>0
```

Listing A.1: Python code of the data loading class `SpaceCraftCVLabDatabase()` from file `database.py`.

```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)

    # List of files in the given folder
    all_files = os.listdir(f'data/SpaceCraft/{object}/images')

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

    num_images = len(image_files)

    # Transform each image
    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.2: Python script `format.py` to randomly generate the training set and the test set based on a specified probability. This script is used when creating both the reference set and the query set from a common set of images. 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
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]

    # List of files in the given folder
    all_files = os.listdir(folder_path)

    # Filter the list to select only images 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('._')]

    num_images = len(image_files)

    # Transform each image
    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.3: 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]
    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__":
```

```
56
57      # Check if the correct number of arguments is provided
58      if len(sys.argv) != 3:
59          print("Usage: python quaternion_to_matrix.py </path/to/your/text/file>
        </path/to/the/pose/folder>")
60          sys.exit(1)
61
62      file_path = sys.argv[1]
63      pose_folder_path = sys.argv[2]
64      file_content = None
65
66      try:
67          with open(file_path, 'r') as file:
68              file_content = file.read()
69      except FileNotFoundError:
70          print(f"The file {file_path} was not found.")
71          sys.exit(1)
72      except Exception as e:
73          print(f"An error occurred: {e}")
74          sys.exit(1)
75
76      poses = file_content.split('\n')[:-1]
77
78      # Iterate through each pose, apply the transformation and save it
79      for pose in poses:
80          image_id, obj_id, q0, q1, q2, q3, x, y, z = pose.split(',')
81          Q = np.array([q0, q1, q2, q3], dtype=np.float32)
82          translation = np.array([x, y, z], dtype=np.float32)
83          matrix = quaternion_to_matrix(Q, translation)
84          np.save(pose_folder_path + '/pose' + str(int(image_id)), matrix)
85
86      print(f"Number of transformation processed: {len(poses)}.")
```

Listing A.4: 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.
"""

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:

                mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
                if mask is None:
                    print(f"Failed to read image: {mask_path}")
                    continue
                inverted_mask = cv2.bitwise_not(mask)
                temp_path = os.path.join(folder_path, "temp_" + filename)
                cv2.imwrite(temp_path, inverted_mask)

                # Delete the original wrong mask
                os.remove(mask_path)

                # Rename the inverted mask
                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.5: 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:
                new_size = tuple([int(dim * resize_factor) for dim in img.size])
                resized_img = img.resize(new_size, Image.ANTIALIAS)
                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
            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.6: Python script `resize.py` designed to alter an image's size with respect to a specified resize factor.

B. Scitas Izar Setup Tutorial

This tutorial aims to set up the Scitas Izar environment step by step in order to run Gen6D.

Step 1: Request permission via email to use the Scitas servers. Unfortunately, I can't provide more details, as I had the privilege of attending an EPFL course that automatically granted me access.

Step 2: Download the following GitHub repository:
`https://github.com/JCHAVEROT/Gen6D/`

Step 3: Follow the instructions provided by the authors of Gen6D to set up the model (specifically, download the Gen6D pretrained model and place it in the correct location).

Step 3: Ensure you're on the EPFL internet network in person or connect to the EPFL VPN.

Step 4: Transfer the Gen6D files to your Scitas Izar session using on your computer the command:

```
rsync -azP <your/path/to/Gen6D> <username>@izar.epfl.ch:~/
```

Step 5: Create a virtual environment on your Scitas Izar Session using the instructions at: `https://scitas-doc.epfl.ch/user-guide/software/python/python-venv/` and activate it.

Step 6: Install all the necessary libraries for Gen6D to function, listed in the `requirements.txt` file, except for openmpi, py-torch, py-torchvision, and cuda.

Step 7: Compose your execute.sh file, in which you put the commands you want to run (use the provided template, do not modify the modules load). Notice that you need to replace the `<username>` section with yours.

```
#!/bin/bash
#SBATCH --chdir /scratch/izar/<username>
#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
      --symmetric
python eval.py --cfg configs/gen6d_pretrain.yaml --object_name spacecraft/jwst
     --symmetric
python eval.py --cfg configs/gen6d_pretrain.yaml --object_name spacecraft/cosmos
      --symmetric
python eval.py --cfg configs/gen6d_pretrain.yaml --object_name spacecraft/rocket
      --symmetric

echo FINISHED AT ‘date‘
```

Listing B.2: 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.

Step 8: Use the following command to submit the job:

```
sbatch execute.sh
```

Step 9: Find the pose estimation results in folder `/Gen6D/data`. You can get them back on your computer using the commands:

```
rsync -azP <username>@izar.epfl.ch:~/Gen6D/data/performance.log <folder/on/your/
     computer>
rsync -azP <username>@izar.epfl.ch:~/Gen6D/data/eval <folder/on/your/computer>
rsync -azP <username>@izar.epfl.ch:~/Gen6D/data/vis_inter <folder/on/your/
     computer>
rsync -azP <username>@izar.epfl.ch:~/Gen6D/data/vis_final <folder/on/your/
     computer>
```

Pro Tip Once the job submitted, you'll have access to the console log in the folder `scratch/izar/<username>`. Once on Scitas Izar, to go there, do the following command:

```
cd && cd /scratch/izar/<username>
```

You can find a lot of useful command to control your job at this link: `https://scitas-doc.epfl.ch/user-guide/using-clusters/running-jobs/`

I will keep the `README.md` file in the repository `https://github.com/JCHAVEROT/Gen6D/` updated to provide more information if necessary. Additionally, you'll find a compressed ‘zip‘ file that includes well-organized SpaceCraft images, allowing you to promptly test your setup and providing a template folder for your convenience.

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

[3] Y.-B. Jia. "Quaternions." In: (2022).

[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.

[5] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144.