Pose estimation of unknown tumbling bodies

Arion Zimmermann

Master's thesis

Spring 2024

Advisors : Professor Soon-Jo Chung
Professor Colin Jones

Contents

1 Abstract

The accurate state estimation of unknown rigid bodies in space is a critical challenge with applications ranging from space debris tracking to asteroid shape estimation. A necessary enabler for this technology is to find and track features on a continuous video stream in real-time. Recent advances in feature extraction make use of Convolutional Neural Networks (CNNs) and Deep Learning to yield higher quality features than classical feature extraction algorithms. Nevertheless, CNNs require large amounts of memory and computational power to be able to perform inference on a real-time video stream, which is not a given onboard spacecraft. Additionally, these CNNs are often not robust to the drastic light occlusions that are common in space.

We introduce COFFEE, a novel fast feature extraction and matching algorithm designed to leverage prior information on the sun phase angle given by sun-tracking sensors commonly available on spacecraft. The algorithm associates salient contours to their projected shadows to detect sparse but highly descriptive features. A fast inference Sparse Neural Network followed by an attention-based Graph Neural Network matcher are then jointly trained to provide a set of correspondences between successive frames.

The resulting pose estimates were found to be more accurate than with classical pose estimation pipelines and an order of magnitude faster than other state-of-the-art deep-learning pipelines on synthetic data and on renderings of the tumbling asteroid Apophis.

2 Introduction

In 2004, astronomers Roy Tucker, David Tholen and Fabrizio Bernardi discovered the Apophis asteroid during its first observable fly-by of Earth [1]. Until recently, it was considered as one of the most threating asteroids, as there was a lot of uncertainty in its trajectory and its proximity to Earth. This hypothesis was fortunately contradicted by recent observations of Apophis' trajectory during its 2021 fly-by to Earth [2].

Nevertheless, the scientific interest in analyzing this asteroid is still of great importance, as it could provide valuable information about the formation of the solar system and give an understanding of the potential resources that are within the asteroid.

In 2029, the Apophis asteroid will undergo yet another fly-by of Earth. Many space agencies have already planned missions to study this asteroid in detail, such as the OSIRIS-APEX [3] from NASA and Ramses [4] from ESA.

Preliminary studies [5] on the trajectory of Apophis suggest that it has a non-uniform inertia matrix and is currently tumbling along its second axis of inertia. As the dynamics of bodies rotating along their second principal axis of inertia is unstable and chaotic by nature, it might be extremely difficult to predict its motion, even in a short time frame. Henceforth, classical motion models would need to account for an unusually large process noise, and it is believed that current state-of-the-art algorithms to estimate the motion of a rigid body in space would not be sufficient to determine the pose of Apophis in real-time.

Some scientific missions require spacecraft to synchronize their orbit to the motion of the asteroid before engaging a final approach to its surface. Typically, ground operators would collect imagery data from the spacecraft and deduce the long-term motion of the asteroid by matching highly descriptive features on its surface, such as the edge of craters and the shape of boulders. Photogrammetry or Stereophotoclinometry (SPC) methods would be used to obtain the shape model of the asteroid and an accurate estimate of its motion. Up to now, missions to asteroids have always included a significant portion of their mission time to analyze the asteroid's trajectory before attempting to extract a sample.

In the case of Apophis, the unpredictable motion of the asteroid and the harsh lighting conditions encountered in space make that method useful to recover the shape of Apophis but impractical to estimate its instantaneous motion, as it is tumbling.

The purpose of this Master's thesis is to design a robust and accurate real-time algorithm capable of estimating the instantaneous pose of a tumbling small body, thereby enabling orbit synchronization, a key capability for potential scientific missions around tumbling bodies in space.

3 Related work

The pose estimation of asteroids usually involves classical feature descriptors and matchers to find keypoint correspondences between two successive frames. The detected features are usually salient points processed by hand-crafted descriptors such as SIFT, ORB or AKAZE. Nesnas et al. [6] provide an excellent framework to process such hand-crafted feature in an autonomous navigation pipeline around asteroids.

Several studies focused on the detection and matching of natural landmarks, such as craters or boulders, which provided features easily trackable in time [7] [8] but make strong assumptions about the distribution of such features.

The relative pose between the spacecraft and the asteroid is then either formulated as a Maximum-Likelihood estimation framework to minimize the reprojection error between multiple frames [9] or as a pose-graph optimization problem [10] [11]. In these classical cases, strong occlusions can mislead the pose estimation algorithm and give completely erroneous readings.

To improve the robustness of these algorithm to different lighting conditions, deep-learning methods have been developed, which leverage Convolutional Neural Networks and Multi-Layer Perceptrons.

Some studies [12] [13] proposed a fully learned pipeline to estimate the pose of the uncooperative spacecraft directly. Nonetheless, these methods do not apply to asteroid as the distinctiveness of features and surface roughness are fundamentally different. Kaluthantrige et al. [14] used a CNN-based approach to regress the relative pose between a camera and an asteroid but trained and tested their model on renderings of a single asteroid.

Many of these studies however acknowledge the effect of moving shadows due to a varying sun-phase angle and asteroid rotation on the accuracy of the pose estimation pipeline [6] [14] [7]. Additionally, deep-learning methods, such as [14] [12] [13], most of the times neglect the computational resources demanded by their models.

Even though deep learning improved the invariance of the pose estimation algorithm to changing lighting conditions, only a few methods have explicitly incorporated the sun phase angle as a prior to the state estimation. [15] used Neural Radiance Field to find a dense solution to represent the shape of an asteroid. Even though not real-time, this method is the first to incorporate information about the camera and the sun phase angle as an input embedding in their model.

We will show that treating the sun phase angle as a prior can not only improve the accuracy of deep-learning pipelines but also significantly accelerate them.

4 Method

The pipeline we propose here to estimate the state of a small body can be decomposed into four independent steps, as depicted in figure 1.

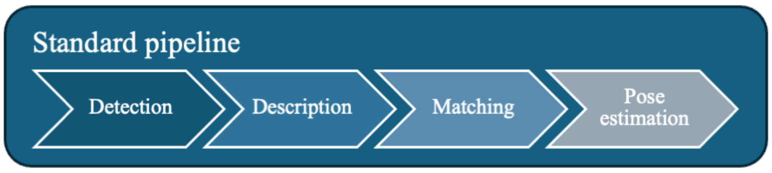

Figure 1: Standard keypoint-based pipeline

Unlike recent methods [12] [13] that involve merging multiple of these steps, we decided to maintain this classical four-step pipeline to better explain how the components of the algorithm are intertwined with each other. Explainability is a key property of algorithms for space applications, as it gives more confidence on the robustness of a given algorithm during its testing and validation phase. Since the onboard computational resources are limited, the algorithms used for each of these steps were carefully selected to minimize the required computational resources.

In our pipeline, we make use of a keypoints-based approach, where the images are first processed to only consider a set of highly descriptive salient features in a step called *feature detection*.

We detect the keypoints classically and leverage novel deep learning methods to process those keypoints through *feature description* and find matches between neighboring frames with *feature matching*. Finally, the attitude of the asteroid can be computed through a *pose estimation* algorithm.

We make use of a keypoint-based pipeline, as opposed to dense matching and optical flow. The latter methods try minimizing the pixel-wise reprojection error for all pixels in an image. Classical methods, such as the Lucas-Kanade [16] optical flow, Farnebäck [17] method or the Horn-Schunck algorithm [18], perform well in scenes with a consistent illumination and constant motion. More recent research on optical flow methods using deep learning, such as RAFT [19], Flownet2 [20], and PWC-Net [21], have shown to be much more robust to changing lighting conditions and non-uniform motions. In all cases, dense matching methods are too computationally expensive to be used onboard spacecraft to enable real-time pose estimation.

Keypoint-based matching, however, is used extensively in real-time applications. Indeed, selecting a subset of pixels from an input image dramatically improves the computational efficiency of the pipeline. The challenge is that the pipeline's algorithms must guarantee that the selected subset of features will represent the same geometrical features of the underlying object (trackability) and will contain enough information to be distinguishable from one another (precision).

4.1 Feature detection

Typical pose estimation pipelines rely on the detection of a sparse set of features from the input images. The advantages of this method are twofold. On the one hand, extracting only a sparse set of features from the input image significantly accelerates the processing of subsequent stages in the pose estimation pipeline. On the other hand, the keypoints are often selected to be highly invariant to changing lighting conditions.

Nevertheless, selecting the most relevant pixels in an image is a challenging step, as typically only a subset of around 1000 pixels from a 1MP image are kept and processed in subsequent steps. This subset must be highly invariant to viewpoint and lighting changes, as the same subset of keypoints should be found in neighboring frames.

Extensive research has already been done on the design of such keypoint detectors. For instance, the Harris corner detector [22] and the Shi-Tomasi [23] detector finds relevant corners by analyzing the eigenvalues of the intensity gradients around each pixel of the image. The Difference of Gaussians method progressively blurs the image and computes the difference between two successive blur levels. Keypoints are then detected as local extrema in scale and space. The FAST [24] detector defines a 16-pixel circle around each pixel and determines if the pixel intensity is changing on the border of this circle.

Deep-learning models were also developed to find the most relevant keypoints. Although most method [25] [26] [27] [28] [29] [30] require simultaneous feature detection and description, Key.Net [31] leverages hand-crafted detector filters and learned filters at different scales to find the most relevant keypoints.

Most of the current algorithms rely on the assumption that edges or corners with high contrast changes are likely to reappear in subsequent frames. This assumption is well verified in indoor and outdoor environments on Earth. Nevertheless, it does not hold for rotating bodies in a space environment. Space bodies typically have a highly uniform color and texture. The only visible property is the amount of reflected light from a given surface, which is the consequence of the self-cast shadows from the geometry of the asteroid onto itself. In other words, the asteroid's material is mostly uniform, but the sunlight reflected from its surface is not. Picture 2 shows a picture from the main camera of the Double Asteroid Redirection Test (DART) mission to the asteroid Dimorphos. The sunlight only makes the surface roughness and boulders visible.

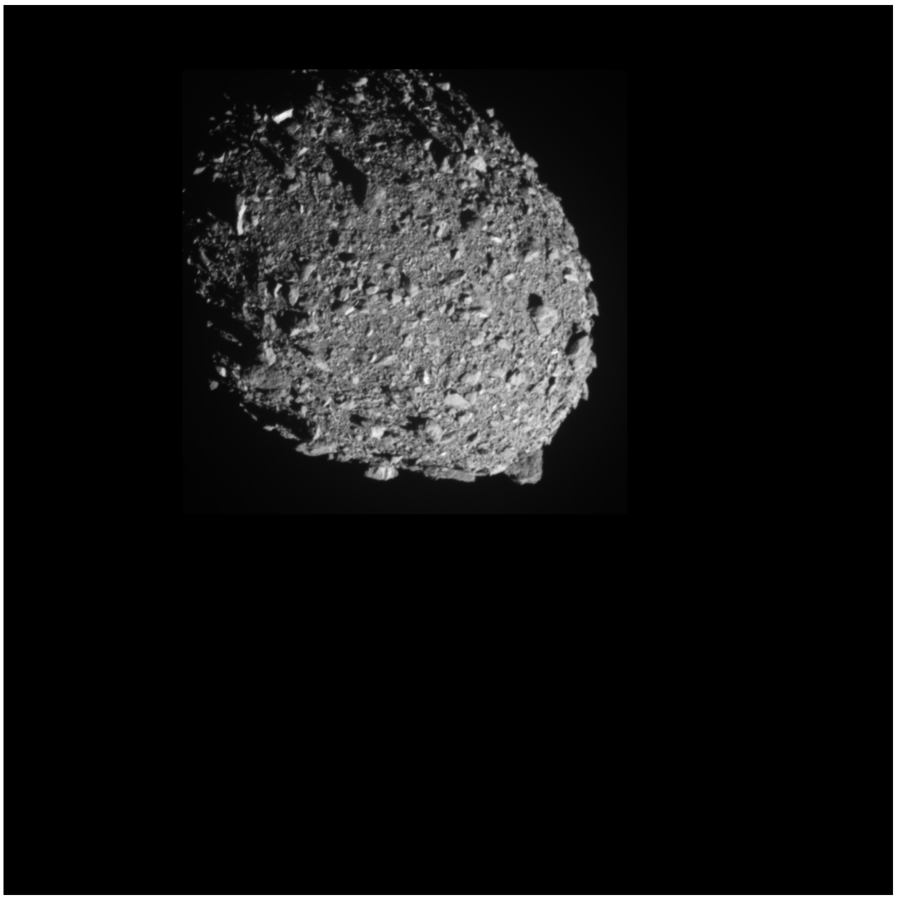

Figure 2: Picture of the Dimorphos asteroid seconds before impact [32]

Current feature detectors fail to recognize which edges are part of the geometry of the asteroid and which edges are self-cast projections of this geometry. This shortcoming may lead to a high bias in the estimation of the asteroid's pose for some distribution of boulders and craters on the asteroid because the shadows do not move in the same way as the asteroid's geometry.

Here, we propose a keypoint detection algorithm that leverages the information from the onboard sun tracker to have a priori information about which parts of the image are self-cast shadows and which parts are not.

To simplify the analysis of this shadow problem, let's define four reference frames.

1. World reference frame: Inertial reference fixed at the center of the asteroid O_a.
2. Asteroid reference frame: Reference frame that is centered and aligned with the body of the asteroid. The rotation matrix $\mathbf{R}_{\mathbf{w}}^{\mathbf{a}}$ transforms vector in the asteroid frame to the world frame.
3. Camera reference frame: Inertial reference frame that is centered and aligned with the camera sensor's image plane. The center of the camera is denoted O_c. Note that the frame is assumed inertial because all non-inertial motion of the camera can be compensated by an equal and opposite motion of the asteroid. Additionally, the camera frame's z-axis can be assumed to be aligned with the world frame's z-axis.
4. Sun reference frame: Inertial reference fixed at the center of the sun O_s.

Additionally, let's define the unit vector $\hat{\mathbf{d}}$ describing the direction of the sun in the world reference frame. The plane formed by the origin of the asteroid and the camera reference frame, along with $\hat{\mathbf{d}}$ is denoted Π.
The sun tracker is used to orient the camera such that the camera reference frame's x-axis is lined up with Π, as shown in figure 3. Without loss of generality, the camera is assumed to point at O_a.

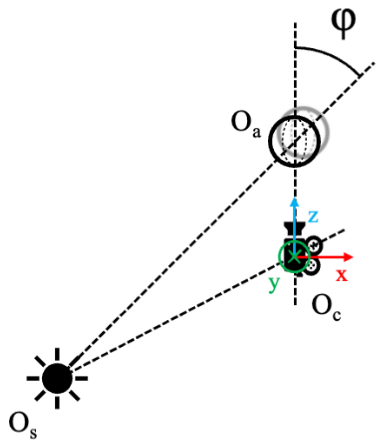

Figure 3: Situation projected in plane Π

Let's denote by ϕ the sun phase angle formed by O_s, O_a and O_c.
To simplify the analysis, define $\mathbf{P}_w = [X_w \quad Y_w \quad Z_w \quad 1]^T$ as the asteroid's points in the world reference frame with homogeneous coordinates. The shadow pattern on the asteroid, as seen from the camera, can be analyzed by decomposing the mesh of the asteroid into a set of 3D points. A ray is traced from each of point $\mathbf{P^i_w}$, going in the direction of the sun vector $\hat{\mathbf{d_w}}$, until it hits again the geometry of the asteroid. All such lines obey to equation (1) for any scalar $\lambda \in \mathbb{R}$.

$$R^i_w(\lambda) = \mathbf{P^i_w} + \lambda \hat{\mathbf{D_w}} \tag{1}$$

where $\hat{\mathbf{D}} = [\hat{\mathbf{d}} \quad 0]^T = [\sin\phi \quad 0 \quad \cos\phi \quad 0]^T$ is the sun direction vector in homogeneous coordinates. The important observation here is that all casted rays are parallel to each other, implying that their projection onto the camera image plane is rendered as lines that intersect themselves into a vanishing point. Indeed, the points $\mathbf{P_w^i}$ can be transformed in the camera's reference frame as $\mathbf{P_c^i}$ through the extrinsic matrix M, such that

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

where $\mathbf{R}$ is a 3×3 rotation matrix and $\mathbf{t}$ is a 3×1 translation vector. The homogeneous coordinates of a 3D point in the world frame can be transformed to the camera frame $\mathbf{P}_{camera} = [X_c \quad Y_c \quad Z_c \quad 1]^\top$ using the extrinsic matrix:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R_w} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

. We assumed that the camera frame was aligned with the plane Π, shared the same z-axis as the world frame and pointed at the center of the asteroid, effectively putting additional constraints on the extrinsic matrix $\mathbf{M}$, such that

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where z_0 is the distance between the camera and the asteroid.

Now, the projection of the ray $\mathbf{R_w^i}(\lambda)$ through $\mathbf{M}$ yields

$$\mathbf{M}\mathbf{R_w^i}(\lambda) = \lambda \begin{bmatrix} \sin(\phi) \\ 0 \\ \cos(\phi) \\ 0 \end{bmatrix} + \mathbf{P_w^i}$$

The vanishing point v can now be expressed by taking the limit as $\lambda \to \infty$ and apply the perspective projection by dividing all coordinates by their depth:

$$v = \begin{bmatrix} \tan(\phi) \\ 0 \end{bmatrix}$$

Then, $\delta = f_x \tan(\phi)$ is the offset from the vanishing point to the camera's optical center, where f_x is the horizontal focal length. Figure 4 illustrates this phenomenon by depicting the projection of 3D lines and their vanishing point in this configuration.

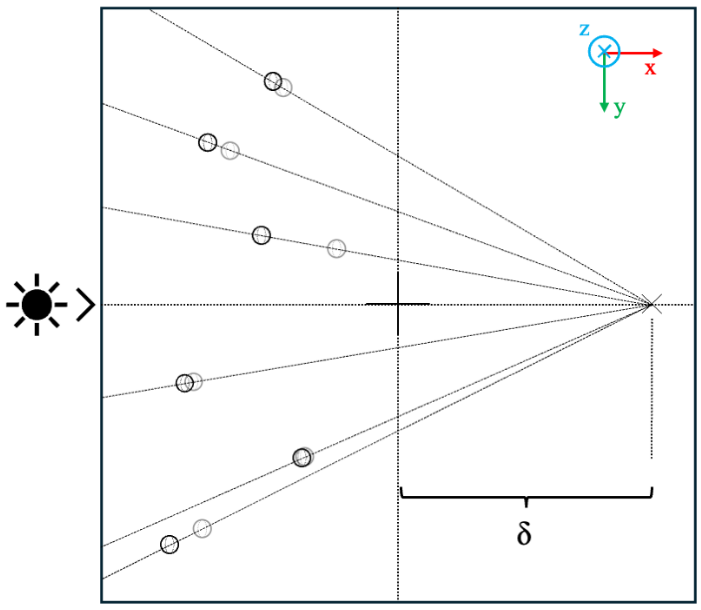

Figure 4: Projection of 3D lines

The main idea here is that all points belonging to the geometry of the asteroid may only cast a shadow in the direction of the vanishing point. Thereby, we can design an algorithm that takes advantage of this observation to distinguish between features and their cast shadow. The advantages of this method are twofold:

1. It allows us to find points that are protruding from the asteroid (e.g. craters and boulders) and that will therefore remain illuminated through time.

2. It prevents us from selecting untrackable keypoints at the edge of the shadow ray.

We designed the Celestial Occlusion Fast FEature Extractor classical detector in 1 that scans each of these rays and encodes the protruding points as a sparse set of coordinates and the size of their cast shadow as a feature value.

Algorithm 1 COFFEE detector

```
1:  **Input:** input dense image, $\phi$ sun phase angle
2:  **Output:** output, sparse image
3:  rectified ← Rectify(input, $\phi$)
4:  thresholded ← Thresholding(rectified)
5:  filtered ← EdgeFilter(thresholded)
6:  compressed, crows, cols ← CSR(filtered)
7:  for each row $i$ do
8:      for each column $j$ do
9:          start_idx ← crows[row_idx]
10:         end_idx ← crows[row_idx + 1]
11:         index ← start_idx + col_idx
12:         if compressed [index] < 0 then
13:             values[index] ← arctan(cols[index + 1] - cols[index])
14:         end if
15:     end for
16: end for
17: return sparse_representation
```

Note that the *Rectify* module uses the sun phase angle to rotate the image and invert the effect of the vanishing point, the *Thresholding* module applies a common dynamic thresholding algorithm to obtain a binary image, the *StepFilter* module scans the image with an edge detector filter with kernel $[1, -1]$ to find the start and end of the shadows, the *CSR* module finally encodes the output of the filter in the Compressed Sparse Row format which can then be used by our algorithm.

This algorithm heavily compresses (ratio ca. 1:500) the content of the image by only including the location of protruding points and the size of their cast shadow. We will show that this encoding yields enough information to matching the set of features across multiple images.

4.2 Feature description

The feature detection algorithm generates a set of keypoints that are anchored to the geometry of the asteroid. One floating-point value representing the size of the cast shadow is assigned to each of these points.

The purpose of the feature description pipeline is to transform the shadow size feature into a high dimensional space which also describes the relation of a given keypoint to its neighbors. Doing so will allow the matching pipeline to use information about the geometry of the mesh and the shape of the craters and boulders to infer which features are matching with one another.

State-of-the-art data-driven feature descriptors use Convolutional Neural Networks to augment the dimensionality of the input image and extract relevant features from it.

Instead of following this standard approach, we make use of Sparse CNNs directly on the set of extracted keypoints to significantly boost the performance of the deep learning model. Indeed, instead of convolving every point of the input image through a set of filters, we only perform the convolution on the sparse set of extracted keypoints, which is a lower-dimensional embedding of the image space.

4.2.1 Convolutional Neural Networks

CNNs have been extensively used in the literature for feature description tasks. These deep learning layers are used in an encoder/decoder architecture, where features are extracted and merged at different scales from the input image, before being restored to their original scale on a higher-dimensional space.

AlexNet was the first recent breakthrough in computer vision. Krizhevsky et al. [33] introduced the ReLU activation and the Dropout regularization, which allowed their encoder/decoder architecture to outperform other state-of-the-art algorithms in image recognition tasks. Its successor VGGNet [34] showed that building a deeper but network with smaller CNN layers could improve the performance of deep learning models.

We implemented and benchmarked several of variants of this architecture in our Sparse CNNs but the model's performance became saturated when using more than 10 layers. Note that VGGNet is also the backbone for the dense SuperPoint [25] feature detector.

Other feature descriptors use a U-Net [35] encoder/decoder architecture 5, which maintains skip connections between encoder/decoder layers of the same resolution. This method is used to maintain a pixel-level accuracy on the location of the features. We also experimented with this architecture but the use of keypoint encoding in the feature matching made the context encoding of U-Net redundant and computationally more expensive.

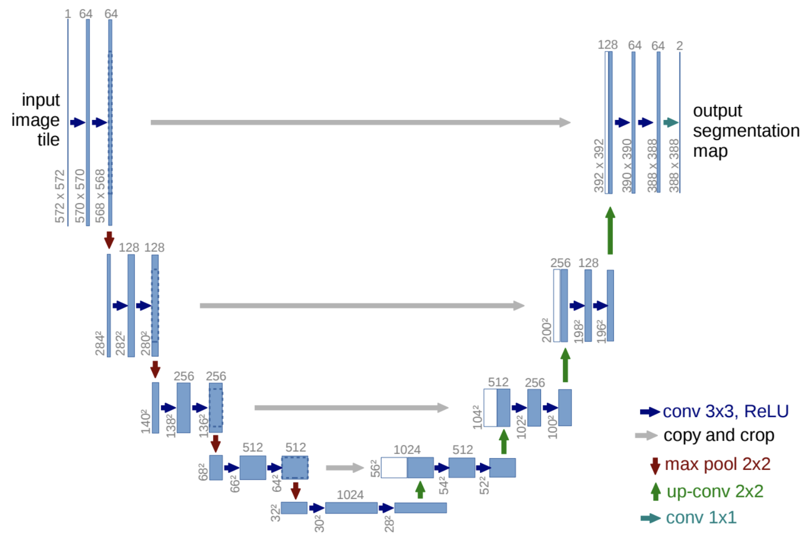

Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Figure 5: U-Net architecture, from [35]

In 2014, GoogLeNet [36] introduced the Inception architecture, which used different kernel sizes at each layer to allow the model to better propagate gradients across layer and build deeper networks. [37] have shown that the Inception architecture, the ResNet architecture or a combination/variant of the two yield similar performance and their superiority over one another is extremely task-dependent. We therefore only explored the ResNet architecture to design the COFFEE learned descriptor.

ResNet [38] solves the gradient propagation issue mentioned for VGGNet by introducing skip connections which add the input to the output around ResNet blocks. In this way, the model does not learn a function of the input but a residual function $F(x) = H(x) - x$, as depicted in figure 6.

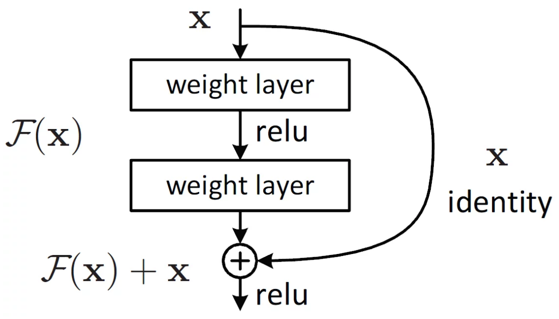
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 6: ResNet architecture, from [38]

ResNet bottleneck blocks 7 are used extensively in our COFFEE learned descriptor because of their computational efficiency and their ability to extract contextual information from the sparse set of keypoints.

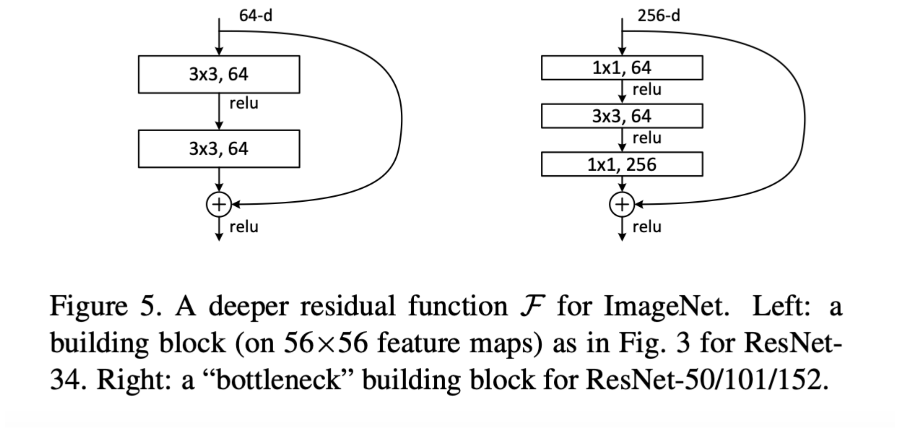

Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

Figure 7: ResNet blocks, from [38]

4.2.2 Sparse CNNs

Our feature description algorithm exploits the unique properties of the keypoints extracted by the COFFEE classical detector. We observe that these are derived from the projection of the geometry of the asteroid onto itself through its self-cast shadow. We encoded the boundaries of this shadow in the detected features. Since the geometry of the asteroid is assumed to be made from a continuous mesh, its projection is a continuous shape and the detected keypoints form a continuous curve.

Instead of describing the input image around these selected keypoints, like most feature description algorithms do, we leverage the remaining continuity structure of keypoint distribution to create the feature vector. This has the advantage of not requiring processing the data from the whole image again. Nonetheless, it assumes that enough information is stored in the distribution of the keypoints to match the features with one another.

The structure of spatially continuous features is typically extracted through nested Convolutional Neural Networks, as detailed in 4.2.1. Notwithstanding, using CNNs directly on our keypoints would be a waste of computational resources since only a very small subset of the image space is filled by keypoints with an average of 2000 keypoints for a 1024x1024 pixels image.

This motivates the implementation of a more efficient method to convolve a sparse set of keypoints through a CNN.

Some research was done on sparse CNNs [39], which perform convolutions on all sparse points and their neighbors. Although computationally efficient, the resulting image has an increased number of features, corresponding to the model's receptive field. The resulting data loses its sparsity structure in a phenomenon called dilation, as more and more points are added to the set of points throughout the layers.

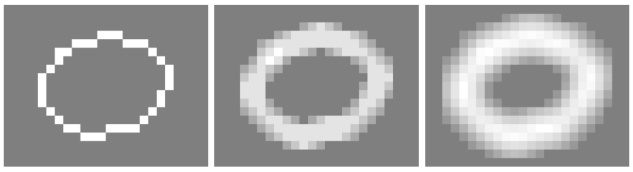

Figure 2: Example of "submanifold" dilation. **Left:** Original curve. **Middle:** Result of applying a regular 3×3 convolution with weights $1/9$. **Right:** Result of applying the same convolution again. The example shows that regular convolutions substantially reduce the sparsity of the features with each convolutional layer.

Figure 8: Effect of dilation, from [40]

Sparse Submanifold Convolutional Neural Networks [40] solve this problem by only providing convolved values on the coordinate locations that were previously in the set. In this way, the size of the sparse keypoint set remains constant throughout the layers of the model. Hash tables and precomputed mappings between the layer's inputs and outputs makes the sparse submanifold convolutions extremely efficient on modern GPUs.

It is notable that the type of convolutions we use is called submanifold because it always remains on the set of detected keypoints. In a sense, the 3D geometrical mesh of the asteroid was projected by the sun as a 2D shadow visible from the camera. We then designed a keypoint extractor that effectively mapped the boundaries of this 2D shadow as a sparse set of 1D curves.

4.3 Feature matching

We match the two sets of described features with an attention-based Graph Neural Network, based on the LightGlue [41] architecture. This type of model define aims at finding a partial assignment matrix $\mathbf{P}$, such that:

$$\mathbf{P}\mathbf{1}_n \leq \mathbf{1}_m$$

$$\mathbf{P}\mathbf{1}_m \leq \mathbf{1}_n$$

Those inequalities encode the fact that all extracted points from image A should have at most one match in image B and vice-versa.

We make use of a Graph Neural Network to express the geometrical links between the sets of image points. We create a undirected complete graph $\mathcal{G}$, where each point is represented as a node. The edges connecting points from image A to image A and from image B to image B are called self-edges $\varepsilon_{\text{self}}$ and the edges connecting points from image A to image B are called cross-edges $\varepsilon_{\text{cross}}$. A state x_i^I is assigned to each node for the image $I \in \{A, B\}$.

After encoding the keypoint location with the descriptor value, a multi-head attention layer is applied separately to $\varepsilon_{\text{self}}$ and $\varepsilon_{\text{cross}}$, so that the state of each node in the graph can be updated accordingly through message passing [42]. Multiple layers of successive self-attention and cross-attention are used to build a faithful latent representation of the underlying scene.

For the self-edges in image $I \in \{A, B\}$, the self-attention message passing is described by the equation (2).

$$\begin{aligned} x_i^I &\leftarrow x_i^I + MLP([x_i^I | m_i^{I \leftarrow I}]) \\ m_i^{I \leftarrow I} &= \sum_{j \in I} \underset{k \in I}{\text{Softmax}}(a_{ik}^{II})_j \mathbf{W} x_j^I \end{aligned} \tag{2}$$

Here, $[\cdot|\cdot]$ is the concatenation operator, W is a projection operator and a_{ik}^{II} is the attention score computed by a rotary encoder on the relative location of the keypoints, such that

$$a_{ij}^{II} = q_i^T R(p_i - p_j) k_j \tag{3}$$

.

For the cross-edges in image I and J, the cross-attention message passing is described by

$$\begin{aligned} x_i^I &\leftarrow x_i^I + MLP([x_i^I | m_i^{I \leftarrow J}]) \\ m_i^{I \leftarrow J} &= \sum_{j \in J} \underset{k \in J}{\text{Softmax}}(a_{ik}^{IJ})_j \mathbf{W} x_j^J \end{aligned} \tag{4}$$

A different attention score was used for the cross-attention layer, where a_{ij}^{IJ} is the bidirectional attention score.

$$a_{ij}^{IJ} = {k_i^I}^T k_j^J \tag{5}$$

LightGlue additionally keeps track of a score across the multiple attention layers to decide if a given node was sufficienly updated or not.

Finally, the feature vectors at the nodes x_i^j are used to construct a similarity matrix M, where the dot product is taken between each pair of features from image A and image B. A row-wise and column-wise softmax operator is finally applied to convert the matrix M to a log-score S. The model can thus be trained by minimizing the cross-entropy between the log-score S and the ground-truth correspondences.

In the COFFEE algorithm, the sparse feature descriptor is jointly trained with the matching model.

4.4 Pose estimation

Given a set of correspondences between keypoints, it is possible to recover the relative pose between the camera reference frame and the asteroid reference frame.

This problem is usually formulated as solving the camera pose with respect to a fixed reference frame, which in our case is the asteroid reference frame. Note that the resulting pose estimate has its translation component always normalized, since it is not possible to know the distance to the asteroid without assuming additional constraints in the geometry of the asteroid or using additional sensors.

This problem formulation requires solving 5 degrees of freedom. In our case, the relative motion of the camera reference frame with respect to the initial frame is assumed to be known between two successive images, implying that the translation component of the pose should be known. This would reduce the problem to solving only 3 degrees of freedom. Nevertheless, extensive literature already exists on solving the 5 DoFs problem, which is why we opted for simplicity in reusing existing research to estimate the asteroid's attitude.

Any pair of corresponding features in two views of the asteroid satisfy the epipolar constraint (6), encoding the fact that each point in 2D from one view must lie on the so-called epipolar line in the other view.

$$\mathbf{x'}^\top \mathbf{F}\mathbf{x} = 0 \tag{6}$$

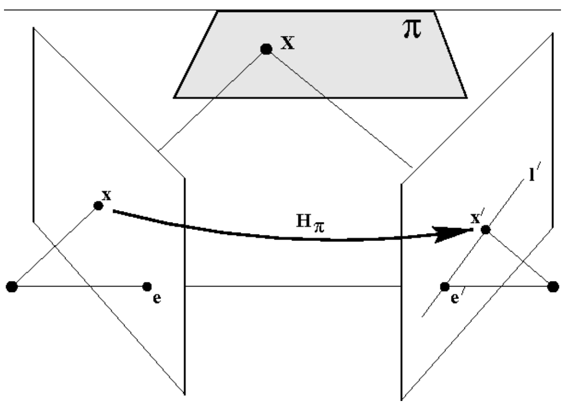

Figure 9: Epipolar constraint from [43]. The point X on plane Π is observed from two perspectives. On the left projection, x is a point. On the right projection, x is a line.

We assume that we know the camera's intrinsic parameters K and can use this information to express the fundamental matrix F in terms of the essential matrix E, such that

$$\mathbf{F} = \mathbf{K}^{-\top}\mathbf{E}\mathbf{K}^{-1} \tag{7}$$

Moreover, the essential matrix must have a rank of two (8) and have the same non-zero singular values (9). Those additional constraints are necessary to reduce the number of degrees of freedoms to 5 (9 DoFs from the 3x3 essential matrix, minus one rank constraint, minus two singular value constraint, minus one scale ambiguity).

$$\det(\mathbf{E}) = 0 \tag{8}$$

$$2\mathbf{E}\mathbf{E}^\top\mathbf{E} - \mathrm{trace}(\mathbf{E}\mathbf{E}^\top)\mathbf{E} = 0 \tag{9}$$

Expanding the essential matrix epipolar constraint in (6), we get

$$x'_i e_{11} x_i + x'_i e_{12} y_i + x'_i e_{13} + y'_i e_{21} x_i + y'_i e_{22} y_i + y'_i e_{23} + e_{31} x_i + e_{32} y_i + e_{33} = 0 \tag{10}$$

The 5-point algorithm [44] is used to find the relative attitude of the asteroid by solving (10) under the constraints (8) and (9) up to a scale ambiguity.

The relative pose $(R,\ t)$ between is then easily recovered through singular value decomposition since

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R} \tag{11}$$

where

$$[\mathbf{t}]_\times = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix} \tag{12}$$

is the skew-symmetric matrix representation of the translation vector.

4.5 COFFEE feature descriptor

The methods and tools mentioned in the previous sections were all used to design our model architecture. We use an encoder-decoder architecture operating in the submanifold space of the points detected by the COFFEE feature detector. Submanifold Sparse CNNs are used to efficiently encode and decode the features.

A first 3x3 CNN layer is used to extract features at the original scale level. Multiple layers consisting of one 3x3 downsampling CNN and one ResNet bottleneck layer are then used in the encoder. Some 3x3 upsampling layers are finally employed to restore the resolution of the features.

The resulting features are then encoded with their keypoint locations and passed to the LightGlue architecture. LightGlue applies successive self-attention and cross-attention layers to the complete graph formed by all keypoints connecting two successive frames, so that a correspondences between features can be established.

Finally, a classical 5-point algorithm is used to estimate the pose of the asteroid. A robust Random Sample Consensus (RANSAC) scheme is used in this stage, so that all potential outliers are filtered out before estimating the final pose.

Figure 10 shows an overview of the complete feature description architecture.

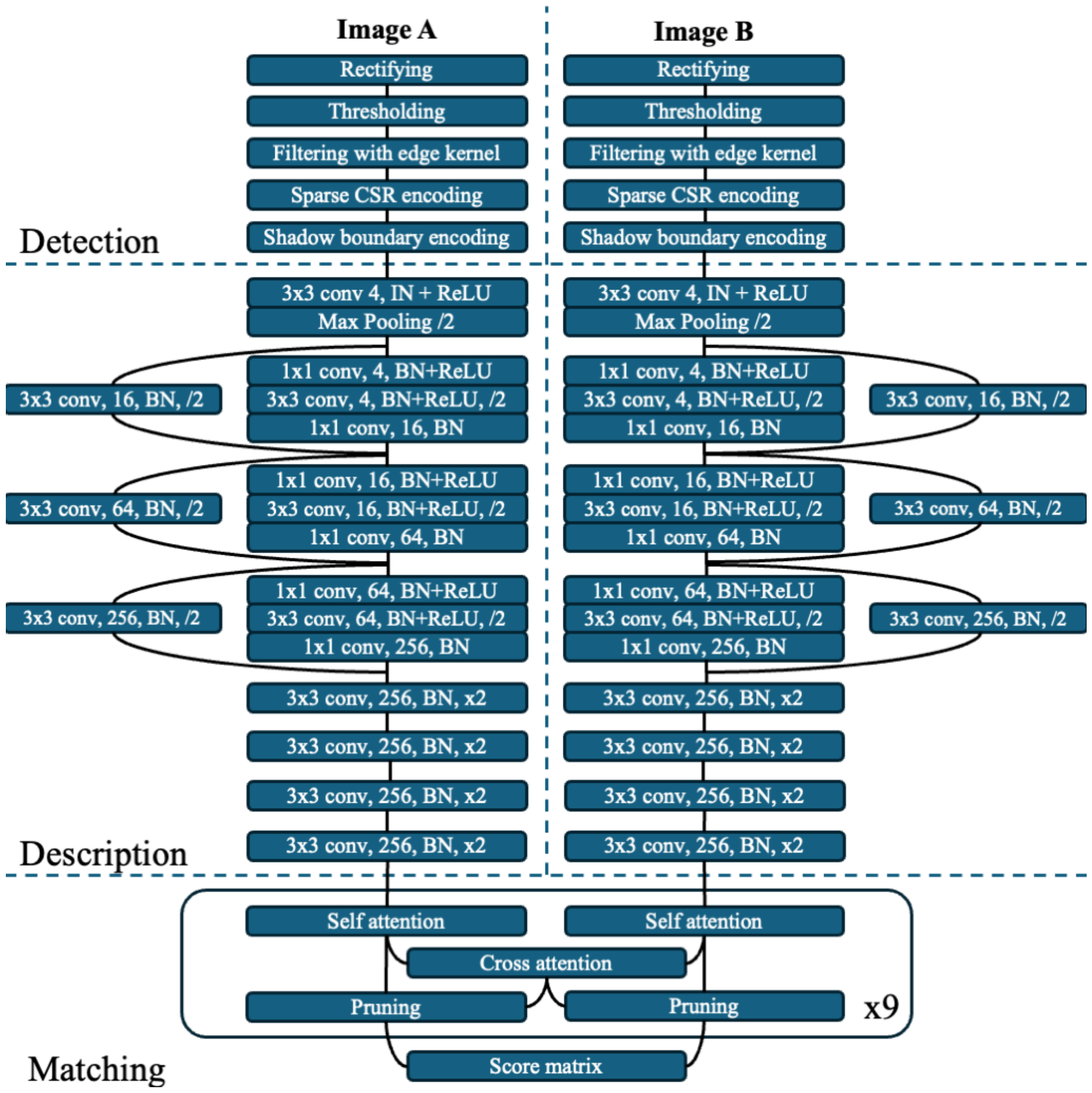

Figure 10: COFFEE architecture (omitting pose estimation)

4.6 Dataset generation

Data-driven algorithms for image processing require large datasets of high-quality images. The challenge of this thesis is that the number of such images for small bodies in space is very little and often affected by the low-resolution and visual artefacts of spaceborne cameras.

Most missions to asteroids unfortunately only consist of flybys, preventing the spacecraft to collect significant datasets as the relative velocity between the spacecraft and the asteroid is in the order of magnitude of several kilometers per second. Additionally, the number of missions that consisted in a spacecraft hovering and statically observing an asteroid is so small that it cannot be used to create a dataset that would generalize to any kind of asteroid.

Table 1 lists the missions that sent a spacecraft to hover an asteroid and that successfully returned detailed imagery of its surface.

Mission	**Year**	**Camera resolution**
OSIRIS-Rex	2016	1024x1024
Hayabusa2	2014	1024x1024
Rosetta	2004	1024x1024
Hayabusa	2003	1024x1000
NEAR Shoemaker	1996	537x244

Table 1: Asteroid hovering missions

The limiting factor of all computer vision algorithms, especially data-driven ones, is the impossibility to prove their generalization to all possible situations because the space of possible images is too large to train, validate and test algorithms upon. For instance, a grayscale 8-bits 1024x1024 pixels camera sensor would have a space of $2^{8388608}$ possible images. Usually, we employ datasets that represent faithfully the class of images that we expect to see in order to train, validate and test computer vision algorithms.

Since only four asteroids were exhaustively imaged by the OSIRIS-REx, Hayabusa2, Rosetta and Hayabusa1 missions, we believe that they might not be representative of all possible asteroids that could be observed in future missions. Developing algorithms using these datasets could create a large bias that would affect their robustness when observing new unknown asteroids.

Therefore, we decided to opt for the generation of a synthetic dataset of asteroid images and validate the generative algorithm by visual inspection and comparison with true images of asteroids taken by spaceborne cameras. Note that no input is given to generative algorithm: the datasets are generated out of random noise.

The dataset is constructed such that each element consists of a pair of rendered images alongside with their ground-truth depth with respect to the camera's image plane. Within each pair, the asteroid is subject to a random rotation and the resulting camera's intrinsic and extrinsic matrices are stored as part of the pair.

The generation of multiple datasets is necessary to obtain evaluation metrics that are representative of the ability of the model to generalize to situations outside the dataset on which the model was trained. Four such datasets are therefore generated:

1. The training dataset is a large dataset on which our algorithm will be trained.
2. The validation dataset is a smaller dataset used to compute basic metrics that are helpful to tune the hyperparameters of the model. It is also used to keep track of any potential bias introduced towards the training dataset, so-called overfitting.
3. The testing dataset is used to evaluate the model and compute metrics that can be used to compare our algorithm to other algorithms.
4. The benchmarking dataset consists of rendered images using the shape model of a real asteroid and a physics-driven rotating motion. This dataset is used to evaluate the end-to-end pose estimation of the asteroid.

The training, validation and testing dataset are generated by creating several shape models, i.e. sets of vertices representing the geometry of the generated asteroids. There are 256 shape models generated for the training dataset and 32 each for the validation and testing datasets. For each of these models, an image renderer rotates the asteroid model randomly and projects its geometry onto the camera frame. Each shape model is used to render 256 images, after which it is discarded. Limiting the number of images generated from a single shape model prevents our algorithm from learning the geometry of the asteroid itself, which would create a bias that would negatively impact our algorithm's ability to generalize. The generation hierarchy is summarized in 11.

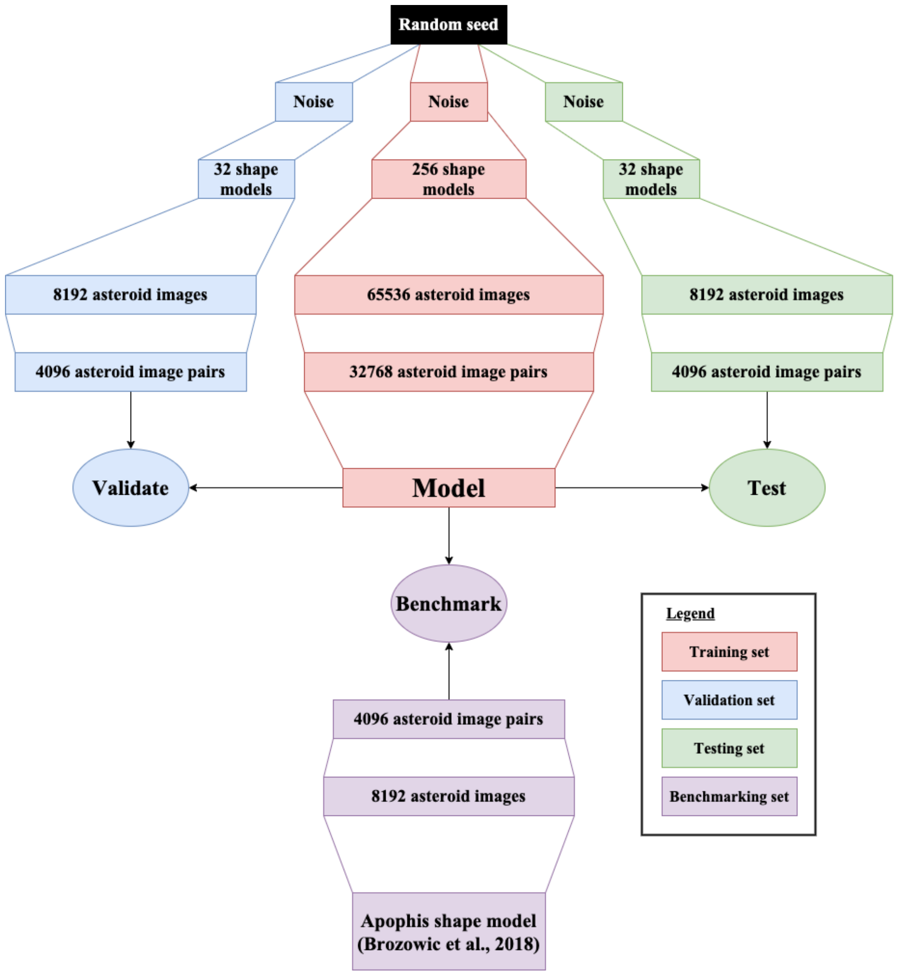

Figure 11: Hierarchy of the dataset generation.

Shape model generation The Blender python library, alongside with the MONET [45] software, are used to generatively create the 320 shape models. Blender is a well-known rendering software, mostly used to create realistic animations. MONET is a software which creates realistic shape models of asteroids. MONET is not directly suitable for our application because it uses normal maps to render the surface roughness of the asteroids. Rendering with normal maps, as opposed to fully raytraced rendering, does not allow for realistic ray-traced shadows, which are crucial for our algorithm.

Base mesh The Blender rock generator is used to create the base mesh of the asteroid. The parameters of the rock generator are heavily randomized in such a way as to allow a wide range of possible shapes. Table 2 represents the multiple parameters passed to Blender to generate the base model.

Parameter	**Value**
Number of rocks	1
Roughness	0
Detail	4
Scale factor	$U^3(1,3)$
Deform	$U(0,10)$

Table 2: Base mesh properties

The roughness of the generated shape is set to zero, as the roughness will be customized later by the MONET shape generator which already adds craters and surface roughness to the asteroid. The detail level of the asteroid is set to 4, as it is a good trade-off between shape complexity and quality. Increasing the detail level further would make the rendering process prohibitively slow.

The scale factor determines the elongation of the asteroid along each of x-y-z dimensions. Note that the size of the resulting asteroid is normalized by its largest dimension. Figure 12 shows the effect of different elongation factors.

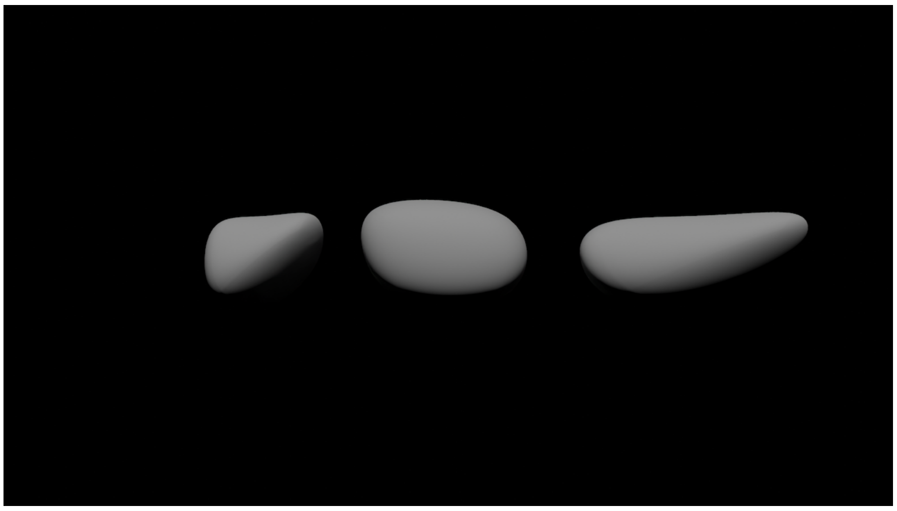

Figure 12: Multiple scale factors for the bash mesh (from left to right: s=1, s=2, s=3).

The deform factor represents how much entropy is added to the shape of the asteroid, making it more spherical, as the deform factor is reduced. Figure 13 shows the effect of different deform factors.

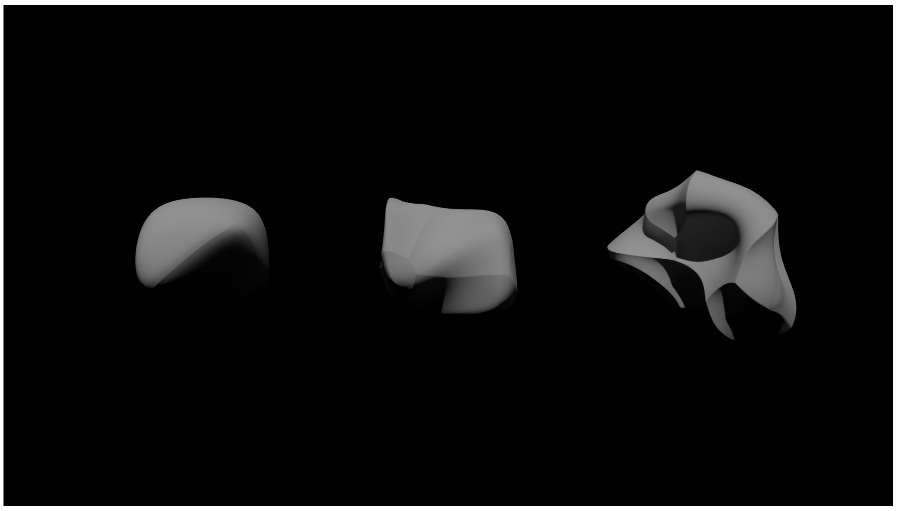

Figure 13: Multiple deform factors for the bash mesh (from left to right: f=0, f=2, f=10).

For the benchmarking dataset, we use the shape model of the Apophis asteroid [46] and enhance it by adding craters, surface roughness and boulders, so that the resulting renderings of Apophis become realistic.

Craters and roughness Realistic shape model requires the addition of features to the base mesh, such as craters and boulders. Craters and surface roughness are created by applying a custom material to the base mesh that will displace the surface. The procedure to generate these features is implemented by the MONET software.

Small and large craters are created by applying Voronoi noise to displace radially the vertices of the base mesh. Voronoi noise is especially suitable for the generation of craters since it can be used to create random circular patterns [45].

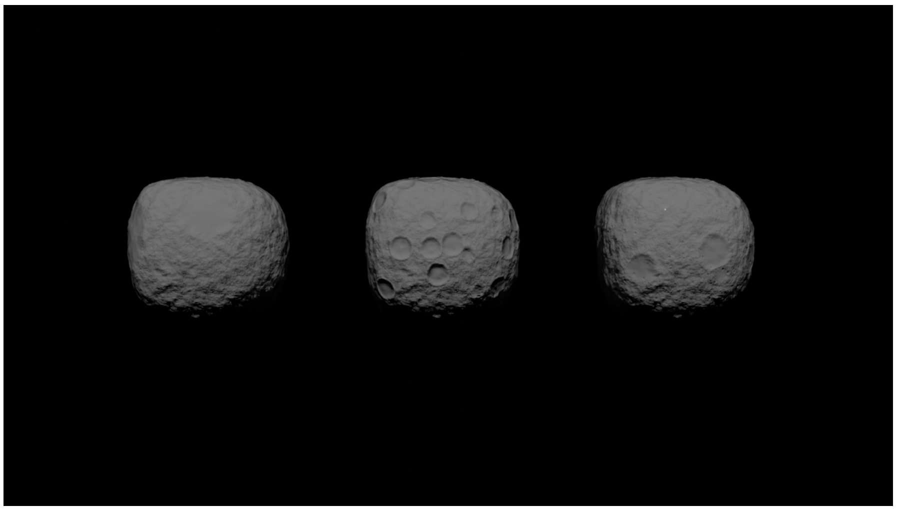

Figure 14: Multiple crater scales (from left to right [small crater scale, large crater scale]: c=[2, 5], r=[5, 10], r=[3, 20]).

The surface roughness is a property that determines the quantity of noise added to the surface of the asteroid. The roughness makes the asteroid more realistic, as it models the very small boulders and particles present on its surface. Fractal Perlin noise texture is generated and scaled according to a roughness value drawn from 3. The effect of this property is shown for multiple roughness values in figure 15.

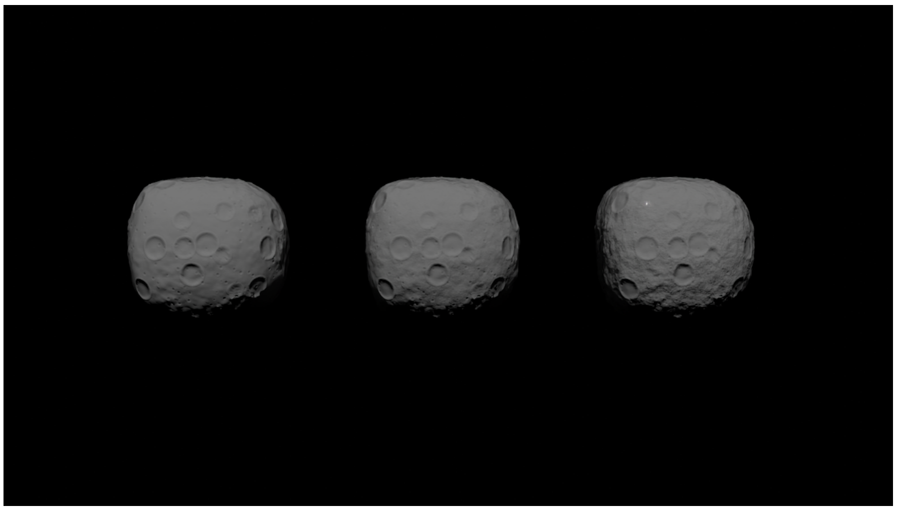

Figure 15: Multiple roughness factors (from left to right: r=2, r=5, r=10).

The craters and surface roughness effectively displace the vertices of the base mesh by a certain amount. We scale this displacement by a depth property drawn from 3 and show its effect for multiple depths in figure 16.

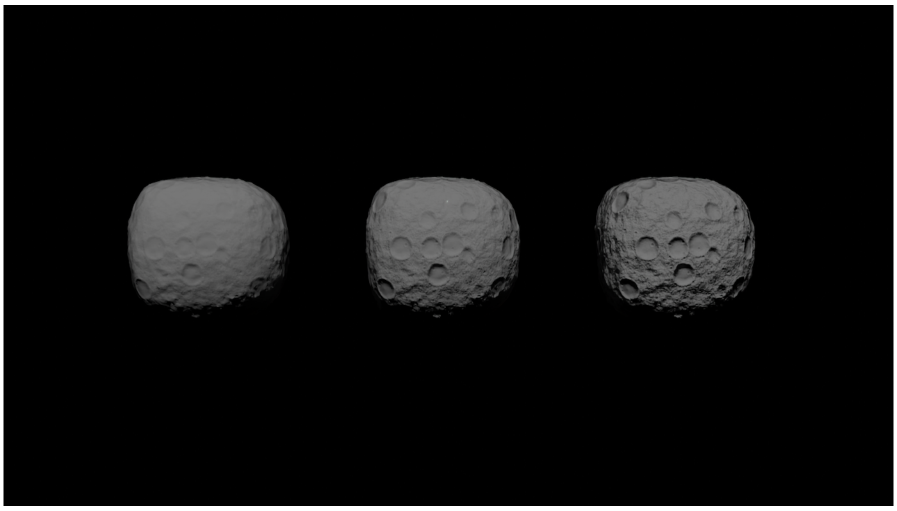
Figure 16: Multiple depth factors (from left to right: d=0.1, d=0.3, d=0.5).

The surface roughness, the crater distribution parameters, as well as the depth at which the displacements are applied are drawn from the distributions stated in table 3.

Parameter	**Value**
Roughness	$U(2, 10)$
Depth	$U(0.1, 0.5)$
Small crater size	$U(2, 5)$
Large crater size	$U(5, 20)$

Table 3: Surface properties

Note that Blender materials do not directly affect the geometry of the mesh and are instead only applied in the rendering stage. To overcome this issue and gain the ability to generate an exportable shape model, it is necessary to render the applied material to a displacement texture and manually displace the mesh's vertices.

Blender incorporates the ability to "bake" a material to a texture by mapping the displacement vector that represents the craters and surface roughness to a separate "emission" channel. The challenge of this procedure lies in mapping a curved mesh to a flat 2D texture to which Blender can render the displacement vectors. The mesh is therefore unwrapped to a flat map by cutting edges that connect faces exceeding a predefined angle limit.

After baking, the displacement texture can be incorporated as a mesh modifier and all vertices belonging to the original base mesh are now representing an asteroid with a realistic crater distribution and surface roughness.

Boulders The MONET software adds boulders using the Blender rock generator. To mimic the nonuniform distribution of the boulders on the surface of asteroids, three different sizes of boulders are generated for each model. The number of such boulders and their size is selected randomly for each of the three classes and is summarized in table 4.

Parameter	**Value**
Large rock count	$U(1, 10)$
Medium rock count	$U(10, 100)$
Small rock count	$U(100, 1000)$
Large rock size	$U(0.01, 0.03)$
Medium rock size	$U(0.003, 0.01)$
Small rock size	$U(0.001, 0.003)$

Table 4: Boulder distribution properties

All parameters pertaining to the generation of the shape models are summarized in table 5.

Parameter	**Value**
Number of rocks	1
Roughness	$U(2, 10)$
Detail	4
Scale factor	$U^3(1, 3)$
Deform	$U(1, 10)$
Depth	$U(0.1, 0.5)$
Large rock count	$U(1, 10)$
Medium rock count	$U(10, 100)$
Small rock count	$U(100, 1000)$
Large rock size	$U(0.01, 0.03)$
Medium rock size	$U(0.003, 0.01)$
Small rock size	$U(0.001, 0.003)$

Table 5: Parameter summary for shape model generation

Figure 17 show a few examples of generated shape models, randomly sampled from the distribution in 5.

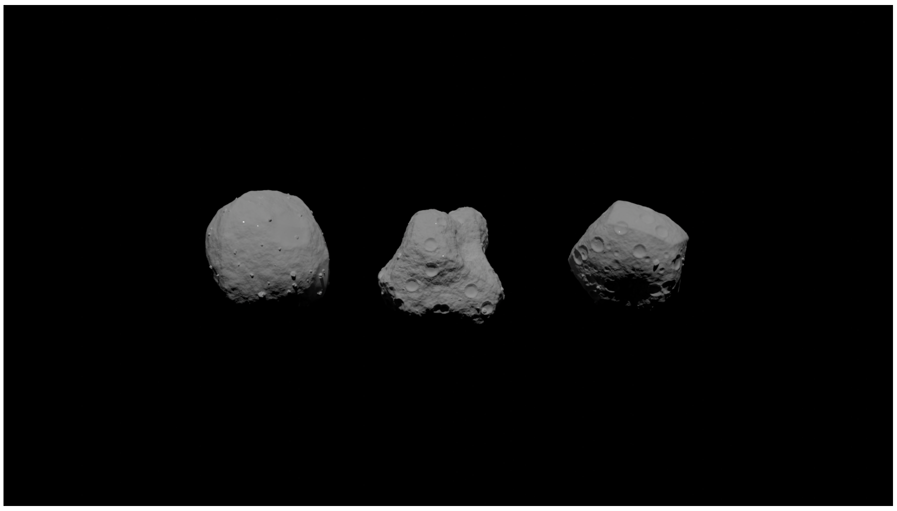

Figure 17: Asteroid shape model samples

4.6.1 Rendering

The shape models can now be used to create realistic images of asteroids under different viewpoints and illuminations.

Since the algorithm we developed is heavily reliant on realistic shadows, we must employ a technology called raytracing, which consists of projecting rays from the light sources to every point on every mesh and record where they intersect with the scene.

One rendering engine natively supporting raytracing is Nvidia Isaac Sim which uses the raytracing hardware capabilities of Nvidia GPUs. This software is extensively used in Reinforcement Learning tasks and closed loop simulation in robotics. We decided to use Isaac Sim because of its tight integration with the Robotic Operating System and possible extensions of this project to add other sensors, such as Lidars or Radars, and develop a vision-in-the-loop control system. Additionally, Isaac Sim is extremely efficient at rendering images of the asteroid, as it only takes 40ms per frame, as opposed to other rendering engines, such as Blender, which takes up to 20s per frame for similar results.

One frame of the Apophis asteroid, rendered with Isaac Sim, is shown in figure 18.

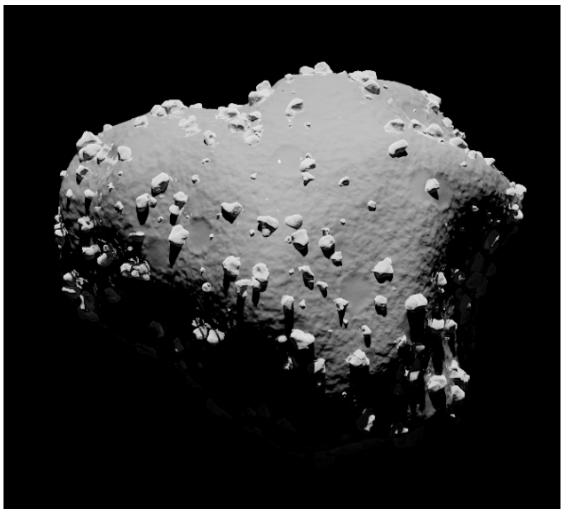

Figure 18: Apophis shape model

4.6.2 Dynamics

During the rendering process, the asteroid is rotated into numerous orientations to allow our algorithm to generalize to different viewpoints. For the training, validation and testing datasets, the asteroid is rendered such that the first frame is captured by setting the asteroid in a random orientation. The second frame of the pair is captured by randomly rotating the asteroid from its initial orientation by a random angle. This random angle is sampled from the following distribution.

Parameter	**Value**
Roll	$U(0, 0.3)$
Pitch	$U(0, 0.3)$
Yaw	$U(0, 0.3)$

Table 6: Orientation Euler-angle difference between successive frames

For the benchmarking dataset, the Apophis asteroid is initially set into a random orientation with an initial angular momentum.

4.6.3 Camera model

The camera model used to render the asteroid is a pinhole camera. The camera parameters are listed in table 7.

Parameter	**Value**
Pixel size	3um
Focal length	3.072mm
F-number	f/16
Focus distance	2.5m

Table 7: Camera parameters

In practice, carful identification of the camera's intrinsic parameters must be performed to avoid any distortion of the resulting images.
The output of the cameras are one grayscale 8-bits image and a 32-bits depth map. Both render products have a resolution of 1024x1024 pixels.

5 Quantitative results

Our algorithm is compared against other state-of-the-algorithms in terms of performance metrics, computational efficiency and accuracy of the complete attitude estimation pipeline. Each of the algorithms used to compare our algorithm against is tuned to perform optimally on our validation dataset, so that the bias in comparing the different algorithms is minimized. It is expected that the learning-based algorithms perform better in terms of performance metrics but are slower than their classical counterparts. With our algorithm, we aim to generally outperform the classical algorithms in terms of accuracy, while still having a higher computational efficiency that data-driven algorithms.

The benchmarking algorithms are the following:

1. SIFT [47]
2. ORB [48]
3. AKAZE [49]
4. Superpoint [25]
5. ContextDesc [30]
6. Disk [27]
7. LFNet [26]
8. R2D2 [29]

5.1 Performance metrics

Since we store the relative camera pose for each image pair and the depth map for each image, it is possible to find the ground-truth correspondences by reprojecting all points from the first image to the second image. If the location of two keypoints after reprojection is less than a given threshold, they are said to correspond to one another. Section

7.7.1 details how such a ground-truth correspondence matrix G can be derived. Each element G_{ij} equals one if the feature i in image A truly matches the feature j in image B. Otherwise, it is zero.

After feature matching, a matrix corresponding to the match probabilities is established. A criterion, such as a threshold, a row-wise maximum, a K-best selection or a Lowe-ratio [47] can be used to obtain a matrix of predicted matches M. Each element M_{ij} equals one if the feature descriptor i in image A is predicted matches the feature descriptor j in image B. Otherwise, it is zero.

Having the ground-truth correspondence matrix and a matrix of predicted matches, we can compute the precision, recall and F1 score for the COFFEE description pipeline, as well as for other benchmarking feature description algorithms. Tuning a feature description algorithm or the matching criterion typically increases the precision but decreases the recall or vice-versa. The F1-score overcomes this bias by introducing a metric that computes the harmonic mean between precision and recall. For completeness, the precision P, recall R and F1-score F_1 are defined as follows:

$$P = \frac{\sum_{i,j} G_{ij} M_{ij}}{\sum_{i,j} M_{ij}} \tag{13}$$

$$R = \frac{\sum_{i,j} G_{ij} M_{ij}}{\sum_{i,j} G_{ij}} \tag{14}$$

$$F_1 = 2 \cdot \frac{P \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}} \tag{15}$$

The K-best features are selected for multiple $K \in 100, 200, 500$ and the precision of all algorithms are stated in table 8. The precision of the feature matching pipeline for a fixed number of features is interesting because only a few correspondences are needed to estimate the pose of the tumbling asteroid.

Algorithm	**Precision (K=100)**	**Precision (K=200)**	**Precision (K=500)**
COFFEE (ours)	**82.5%**	**77.5%**	**68.1%**
Superpoint	69.4%	52.9%	30.4%
ContextDesc	47.1%	45.1%	37.3%
Disk	67.5%	57.4%	41.3%
LFNet	16.2%	14.1%	10.7%
R2D2	16.9%	15.0%	10.9%
SIFT	17.4%	14.8%	10.8%
ORB	5.2%	4.3%	3.1%
AKAZE	9.4%	8.0%	5.1%

Table 8: Precision for a given number of features (best in **bold**)

Some algorithms do not yield a normalized score for each row, implying that multiple matches could be found for the same feature, which is not possible. To prevent skewing the results, we can apply the Non-Maximal Suppression algorithm [47], which discards matches that do not have a significantly higher score than the second-best matches. The ratio separating the best-matches from the second-best matches is called the Lowe-ratio. For multiple Lowe-ratio, we state the precision, recall and F1-score in tables 9, 10 and 11.

Algorithm	**Precision**	**Recall**	**F1-score**
COFFEE (ours)	**54.2%**	48.6%	**51.2%**
Superpoint	32.3%	46.4%	38.1%
ContextDesc	26.3%	35.3%	30.1%
Disk	43.4%	**50.0%**	46.5%
LFNet	29.6%	46.4%	36.2%
R2D2	14.2%	23.0%	17.5%
SIFT	9.5%	13.8%	11.2%
ORB	4%	3.2%	3.5%
AKAZE	5.7%	6.6%	6.1%

Table 9: Precision, recall and F1-score for a Lowe ratio of 1.0 (best in **bold**)

Algorithm	**Precision**	**Recall**	**F1-score**
COFFEE (ours)	54.7%	**48.1%**	**51.1%**
Superpoint	59.8%	30.7%	40.5%
ContextDesc	71.8%	21.2%	32.5%
Disk	**90%**	27.3%	41.7%
LFNet	68.1%	28.1%	39.6%
R2D2	75.6%	1.8%	3.5%
SIFT	22.7%	9.3%	13.2%
ORB	13.3%	1.0%	1.8%
AKAZE	17.0%	2.6%	4.4%

Table 10: Precision, recall and F1-score for a Lowe ratio of 0.9 (best in **bold**)

Algorithm	**Precision**	**Recall**	**F1-score**
COFFEE (ours)	55.3%	**47.5%**	**51.0%**
Superpoint	81.3%	13.0%	22.3%
ContextDesc	91.6%	6.8%	12.5%
Disk	**96.4%**	9.6%	17.4%
LFNet	85.4%	13.8%	23.5%
R2D2	93.3%	0.1%	0.2%
SIFT	39.2%	5.6%	9.7%
ORB	19.7%	0.2%	0.4%
AKAZE	32.3%	0.8%	1.6%

Table 11: Precision, recall and F1-score for a Lowe ratio of 0.8 (best in **bold**)

The COFFEE algorithm outperforms most of the other algorithms in terms of precision, recall and F1-score. However, state-of-the-art deep learning algorithms have a better precision when using the Lowe ratio 0.9 and 0.8 and the Disk algorithm, in particular matches the precision and recall of COFFEE for a Lowe ratio of 0.9 and 1.0.

Note that COFFEE was trained exclusively on synthetic space imagery, which is not the case of the other algorithms, which may suffer from a domain gap and weaken their precision/recall metrics.

5.2 Parameter-invariant metrics

Most metrics derived in the previous section derive from a parameter set arbitrarily, such as the number K for which the K-best matches are selected, or the matching threshold. This particularly affects the precision and recall metrics by increasing the precision and decreasing the recall and vice-versa. The F1-score, employed in the previous section, is the harmonic mean between precision and recall and is designed to avoid the precision-recall trade-off analysis. Nevertheless, it is not necessarily an absolute metric on the quality of a feature matching pipeline.

There are two methods to assess the performance of the algorithms without being affected by this induced bias. One of these is plotting two dependent metrics, such as the precision and the recall, as sweeping through the matching threshold parameter. The other method consists in computing the Area Under the Curve (AUC) produced by this sweep, as it is invariant to the matching threshold.

The Receiver Operating Characteristic is plotted for multiple algorithms in figure 19. The ROC metric is usually used to illustrate the performance of a binary classifier and plots the recall (true positive rate) against the false positive rate.

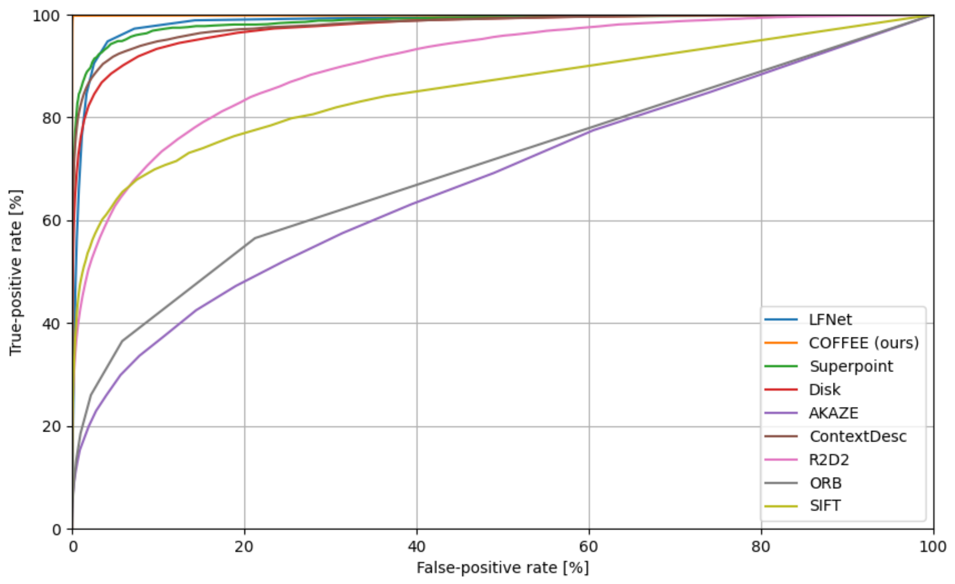

Figure 19: Receiver Operating Characteristics for all benchmarked algorithms

The ROC curve shows that most deep-learning descriptors perform better than classical descriptors and that the COFFEE descriptor shows better characteristics than the other benchmarking algorithms. The Precision-Recall curves for multiple algorithms are plotted in figure 20.

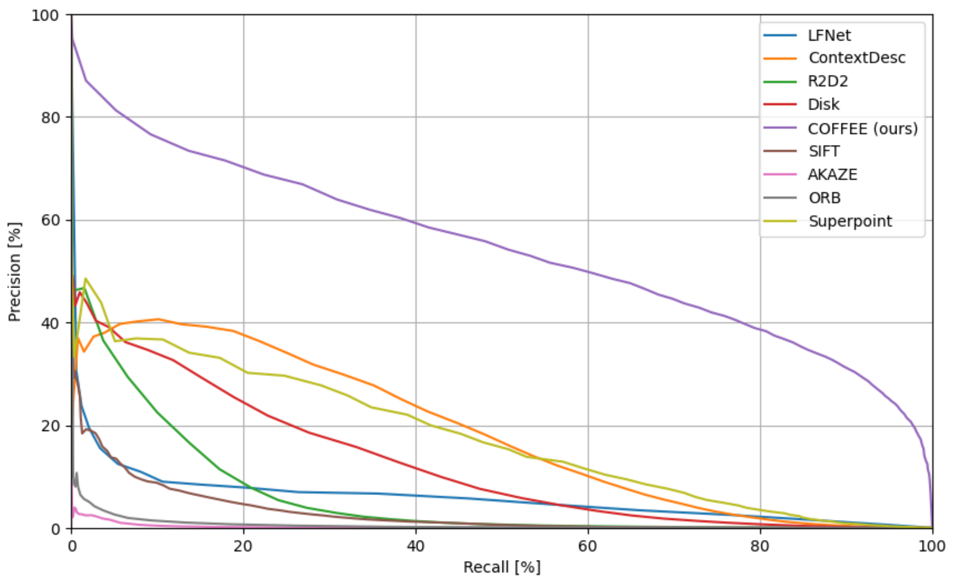

Figure 20: Precision-Recall curve for all benchmarked algorithms

Similarly, the PR curve shows that most deep-learning descriptors perform better than classical descriptors and that the COFFEE descriptor shows better characteristics than the other benchmarking algorithms.

The Receiver Operating Characteristics AUC and the Precision-Recall AUC metric is computed for each algorithm in table 12, alongside with their optimal F1-score.

Algorithm	**Best F1-score**	**ROC AUC**	**PR AUC**
COFFEE (ours)	**54.7%**	**54.2%**	**99.9%**
Superpoint	28.4%	17.6%	98.6%
ContextDesc	30.3%	18.4%	98.0%
Disk	20.8%	12.2%	97.5%
LFNet	9.6%	5.9%	98.5%
R2D2	14.6%	5.9%	90.4%
SIFT	9.1%	2.8%	85.2%
ORB	3.5%	0.7%	70.3%
AKAZE	2.5%	0.2%	67.6%

Table 12: ROC AUC, PR AUC and optimal F1-score for multiple algorithms (best in **bold**)

5.3 Pixel error

The quality of the pose estimation algorithm is directly related to the ability of the matching algorithm to accurately locate corresponding keypoints. The pixel error, defined as the median L_2 distance between matching keypoints, must be minimized.

We compare the baseline SOTA algorithms with our algorithm by plotting the pixel error against the number of matches in 21. Matches are ranked and selected based on their matching score.

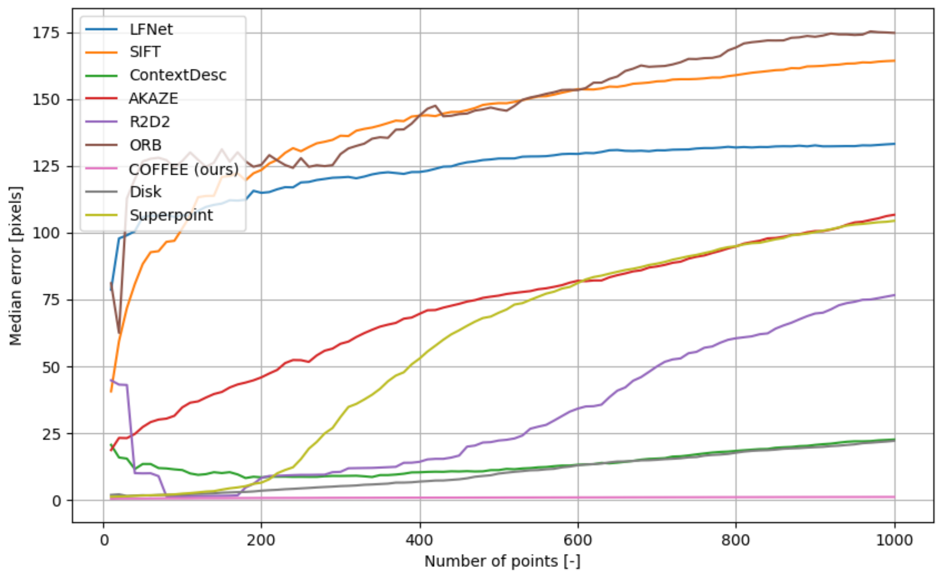

Figure 21: Median pixel error for multiple algorithms

In this configuration, the COFFEE algorithm is one or two orders of magnitude more accurate than any other feature descriptor. We however note that the marked presence of outliers in the benchmarking algorithms can be mitigated by introducing the Lowe-ratio criterion [47], so that the pixel error becomes as in figure 22.

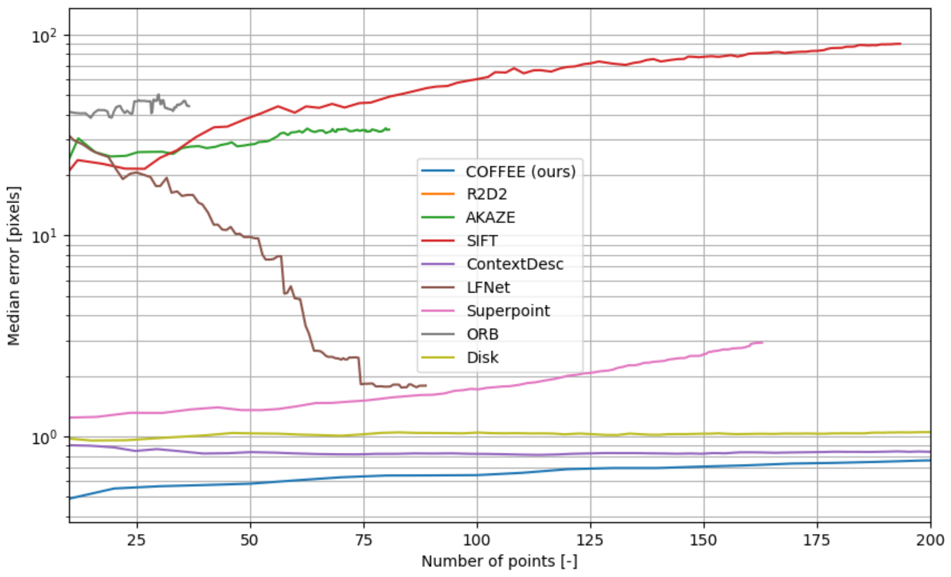

Figure 22: Median pixel error for multiple algorithms with a Lowe ratio of 0.9

The median pixel error then becomes comparable to the Disk [27] and ContextDesc [30] deep learning feature descriptors at a subpixel accuracy. At a low number of points, the Superpoint [25] also achieves subpixel accuracy. We conclude that these three algorithms will likely be the most suitable for the pose estimation algorithm.

5.4 Pose estimation error

We estimate the pose of the asteroid by applying the 5-point algorithm, recovering the essential matrix and inferring the relative rotation matrix and pose, as explained in 4.4. Even though any set of 5-points in the match matrix could be used to recover the pose, a robust scheme should be employed to avoid any outlier during the estimation of the pose. We make use of a Random Sample Consensus (RANSAC) [50] scheme with a reprojection threshold of one pixel and a success probability of 99.9The resulting estimation errors are stated in following tables for the benchmarking algorithms and COFFEE. We select the K-best matches $K \in 100, 200, 500, 2000$ before applying the pose estimation algorithm.

Algorithm	**Error [rad]**	**Standard deviation [rad]**
COFFEE (ours)	0.029	0.018
Superpoint	0.041	0.020
ContextDesc	0.13	0.12
Disk	0.20	0.29
LFNet	0.62	0.85
R2D2	0.12	0.089
SIFT	0.29	0.37
ORB	0.81	0.96
AKAZE	0.41	0.48

Table 13: Pose estimation bias and std dev. for the best K=100 features (best in **bold**)

Algorithm	**Error [rad]**	**Standard deviation [rad]**
COFFEE (ours)	**0.024**	**0.022**
Superpoint	0.066	0.057
ContextDesc	0.083	0.093
Disk	0.17	0.091
LFNet	0.34	0.65
R2D2	0.096	0.087
SIFT	0.28	0.66
ORB	1.05	1.07
AKAZE	0.31	0.29

Table 14: Pose estimation bias and std dev. for the best K=200 features (best in **bold**)

Algorithm	**Error [rad]**	**Standard deviation [rad]**
COFFEE (ours)	**0.028**	**0.021**
Superpoint	0.061	0.042
ContextDesc	0.061	0.049
Disk	0.15	0.14
LFNet	0.11	0.12
R2D2	0.064	0.061
SIFT	0.095	0.077
ORB	0.61	0.69
AKAZE	0.50	0.56

Table 15: Pose estimation bias and std dev. for the best K=500 features (best in **bold**)

Algorithm	**Error [rad]**	**Standard deviation [rad]**
COFFEE (ours)	**0.034**	0.027
Superpoint	0.038	**0.026**
ContextDesc	0.050	0.037
Disk	0.088	0.080
LFNet	0.068	0.066
R2D2	0.061	0.046
SIFT	0.12	0.099
ORB	0.41	0.54
AKAZE	0.20	0.12

Table 16: Pose estimation bias and std dev. for the best K=2000 features (best in **bold**)

The main observation from these results is that only deep-learning feature descriptors can estimate the pose of the asteroid, at a low number of features. When increasing this number to $K = 500$, the SIFT feature descriptor starts showing good results with an error of 0.095 [rad] and a standard deviation of 0.077 [rad].

In all cases, the classical feature descriptors perform significantly worse that COFFEE, which has an error always less than 0.04 [rad] and a standard deviation always less than 0.03 [rad].

5.5 Computational efficiency metrics

A crucial aspect of this project is to design a fast attitude estimator with low memory requirements, so that it can be used on spacecraft. Although parallel computing hardware, in particular GPUs, have seen a swift increase in performance and memory size in the recent years, space hardware remains years behind the current state-of-the-art. This is mainly due to the conservative requirements for space hardware, the slow process of verifying and validating these requirements, as well as the harsh conditions of the space environment.

To do so, our algorithm is compared against other state-of-the-art detectors, by running the algorithms on similar GPUs and similar operating conditions. The execution time is recorded for each algorithm and listed in table 17. The algorithms are tested a desktop computer with a Nvidia Titan RTX GPU and a 20-cores Intel Core i9-7900X.

Algorithm	**Execution time [ms]**
COFFEE (ours)	14.2
Superpoint	52.3
ContextDesc	397
Disk	97.1
LFNet	44
R2D2	490
SIFT	12.2
ORB	**4.4**
AKAZE	5.6

Table 17: Execution time for each algorithm (best in **bold**)

As expected, classical feature descriptors run generally faster than their deep learned counterparts due to their complexity. The execution time of COFFEE is in the same order of magnitude than some classical feature descriptors, such as SIFT (14.2ms vs 12.2ms), which makes it suitable for real-time applications. We also observe that COFFEE is at least 3x faster than any other deep learned feature descriptor.

The trade-off between overall pose estimation accuracy and runtime in figure 23 shows that the COFFEE algorithm is much faster than the other deep learning algorithms but much more accurate than other classical algorithms. We have pushed the pareto optimum in the runtime-accuracy tradeoff and designed an algorithm that leverages the advantages of both classical and learned algorithms to obtain a light pipeline usable in space applications.

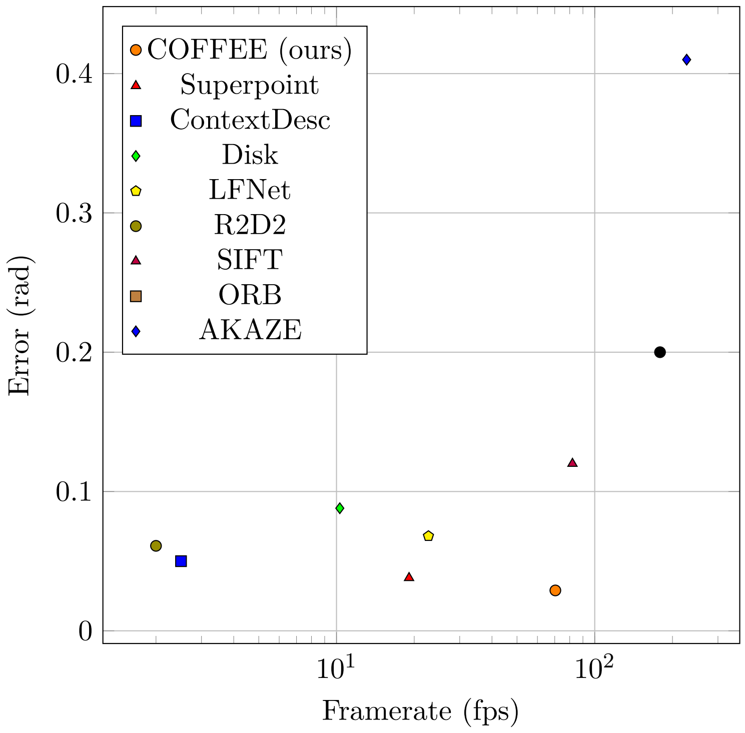

Figure 23: Trade-off between runtime and accuracy

Even with all these results highlighting the advantages of COFFEE over other algorithms, we must bear in mind the following points:

1. COFFEE was specifically trained on a synthetic dataset from images of asteroids, which might be a significant advantage compared to other algorithms that are more general. Ideally, one would train the Superpoint, Disk, LFNet, ContextDesc and Disk algorithms on the asteroid image dataset. One would probably then expect to see the deep learning algorithms outperform COFFEE. The point however is not that COFFEE performs better than other deep learning algorithms, it is that COFFEE is faster than any deep learning algorithm, while outperforming classical algorithms.

2. The execution time listed in this section only pertains to the feature description and not to the feature matching. This decision was made to avoid penalizing feature descriptors that yield a high number of matches, as the matching time is

generally quadratic in the number of features. One could argue that this advantages COFFEE, as the matching pipeline is more complex due to the attention mechanism. In practice however, we observed than the feature matching only added an average of 10ms to the overall execution time compared to standard matching methods.

6 Qualitative results

In this section, we show the intermediate results of the COFFEE pipeline as rendered images, to qualitatively assess the performance of the algorithm.

6.1 Feature detection

Figure 24 shows renderings of the Apophis asteroids. The COFFEE detector extracts information about the shadow on the projection of the rays emanating from the sun. Here, the sun phase angle is 90°, implying that the vanishing point is infinitely east, and that all shadows are cast from the left to the right in the camera's reference frame. The orange points on the rendering show the location of the detected keypoints.

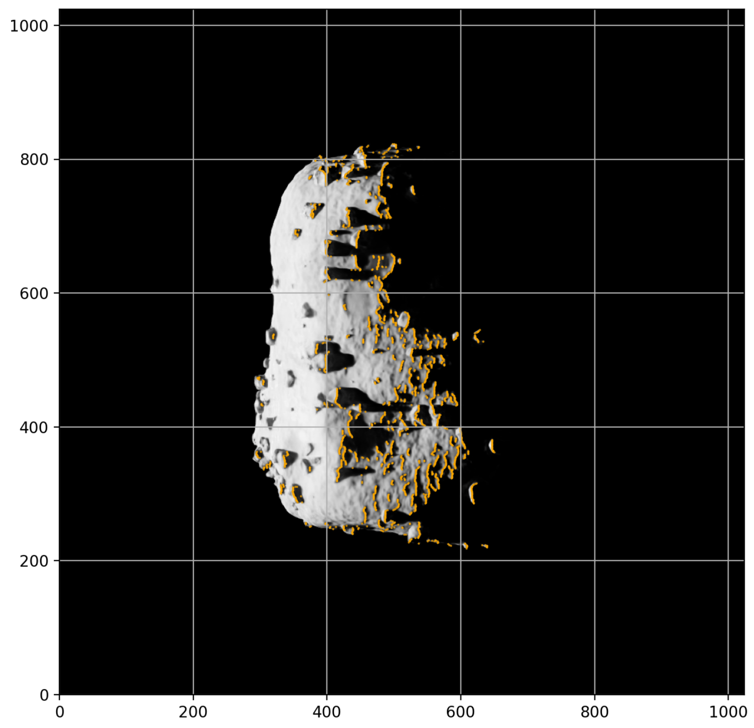

Figure 24: Keypoints detected by the COFFEE algorithm

6.2 Feature description

Figure 25 assigns a highly descriptive feature descriptor to each of the detected keypoints. The 256-dimensional feature space is randomly projected to the RGB color space to ease visualization.

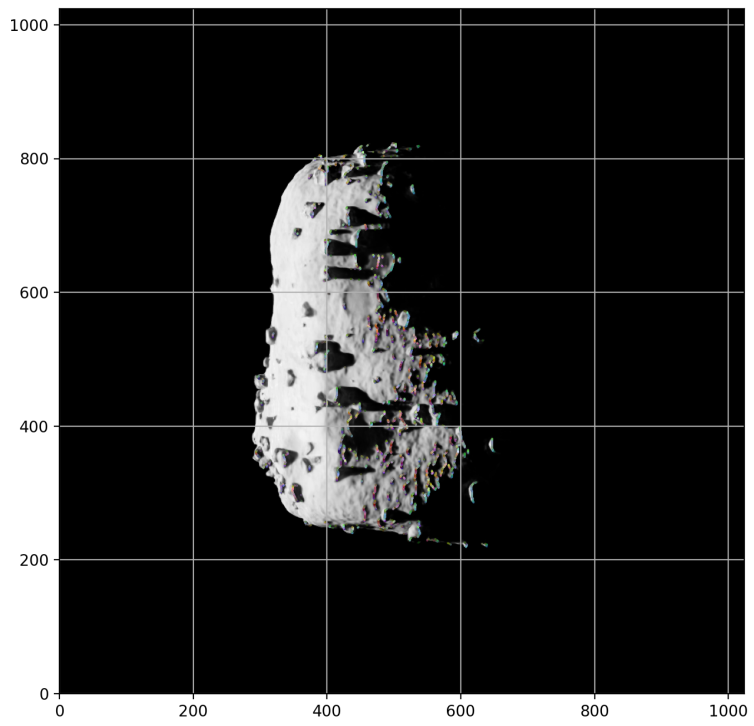

Figure 25: Features described by the COFFEE algorithm

6.3 Feature matching

Figure 26 finds correspondences between two sets of description vectors in successive frames. As with the feature description, the 256-dimensional feature space is randomly projected to the RGB color space to ease visualization.

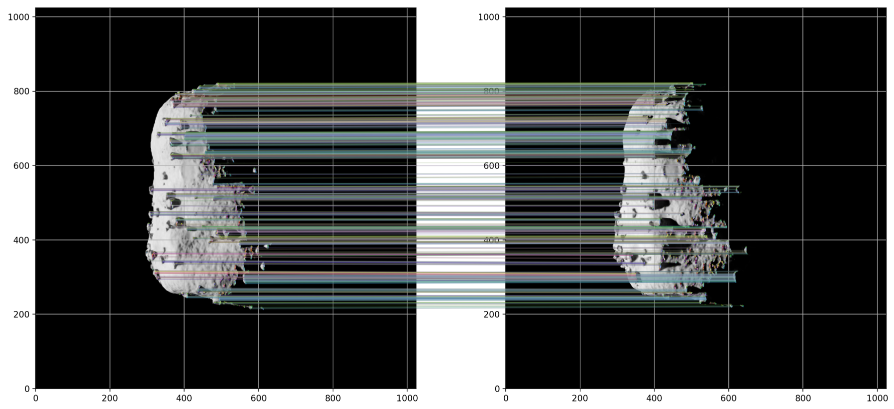

Figure 26: Features matched by the COFFEE algorithm

7 Implementation

A versatile software implementation to train and evaluate our algorithm is provided as an open-source project, under MIT license [51]. Python 3 was used to develop the pipeline, alongside with several custom CUDA modules to speed up the pipeline. The following open-source python libraries were used:

1. Numpy 1.26.4 [52]
2. Numpy quaternion 2023.0.3
3. OpenCV 4.9.0
4. PyTorch 2.2.2 [53]
5. ROS2 Humble [54]
6. Pandas 2.2.1 [55] [56]
7. Matplotlib 3.5.1 [57]
8. MinkowskiEngine 0.5.4 (Forked to add support for CUDA 12) [58] [59] [60]
9. Nvidia Omniverse API

7.1 Architecture

Every step in our pipeline is designed as a standalone python module. All python modules can communicate with each other through so-called frontends. Frontends are abstract interfaces that allow modules to receive input messages and send output messages. The frontend's high-level of abstraction allow a transparent communication, even if the algorithm's modules run through a ROS interface or even if they are not running on the same machine.

Apart from the simulation, the whole training and evaluation pipeline is embedded into a ROS2 environment, which creates a python sandbox and provides a build system for all modules. Launch files and configuration files facilitate the integration between the modules. The simulation module requires its own python sandbox, as shipped with Nvidia Isaac Sim, which prevents us from using the same build system for all modules.

The evaluation pipeline is divided into four stages: feature description, motion synthesizing, feature matching and benchmarking. Our algorithm, as well as the other baseline algorithms, all comply with this pipeline.

For a given description algorithm, the feature description stage selects a set of keypoints and describes them. The keypoint detection and description are aggregated into this stage.

The motion synthesizing stage is common to all algorithms and pairs two feature sets into a single message for the next stage.

For a given matching algorithm, the feature matching stage finds correspondences between the two feature sets created in the motion synthesis stage.

The benchmarking stage finally evaluates the found matches by comparing them to ground-truth correspondences. This last stage generates plots, tables and outputs relevant metrics.

Our learning pipeline is slightly different from the evaluation pipeline, in the sense that it aggregates all the learnable parts of the algorithm into the feature matching stage, so that we can benefit of a significant speed-up in the learning process. Note that this approach does not affect the performance of our algorithm, since the feature descriptor processes images one by one. This description stage can therefore be seamlessly interchanged with the motion synthesis.

The whole pipeline for training and evaluation are depicted in figure 27.

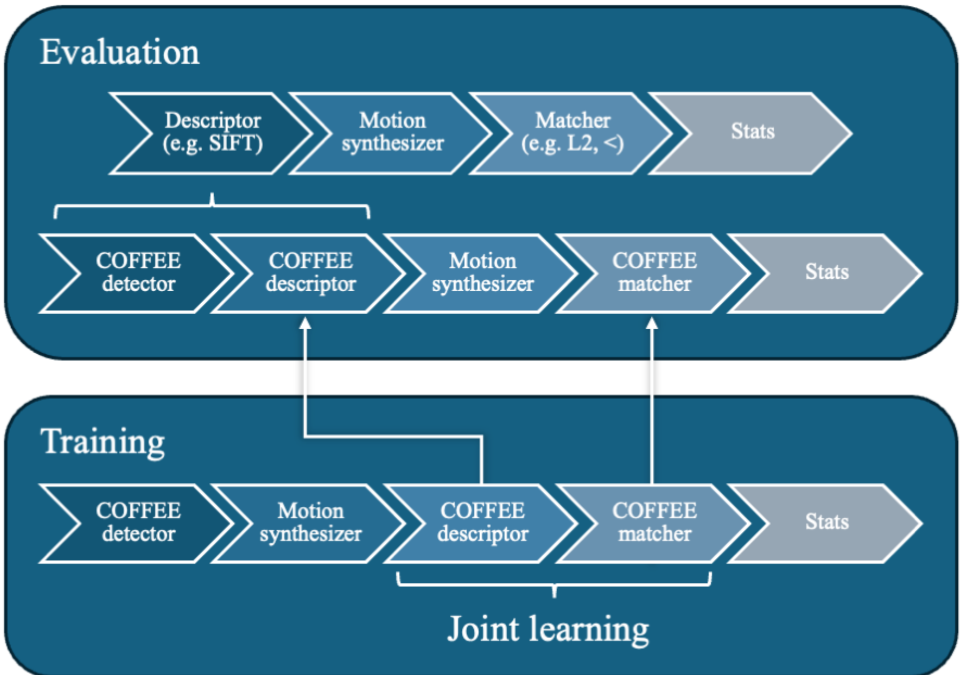

Figure 27: Software architecture

7.2 Messages

A key enabler for the inter-module communication system is the concept so-called messages. These are simple structures that are library-independent and portable across multiple versions of pythons. These structures are said to be "pickleable", as they can be transformed into binary structures by the pickle module in the python standard library. The advantage of using such a structure is that modules can communicate by transmitting this binary structure through the filesystem, through the local network or through the ROS2 DDP network.

A dedicated message was designed for each module interface and is summarized in figure 28. The *astronet_msgs* package contains the source code for all messages.

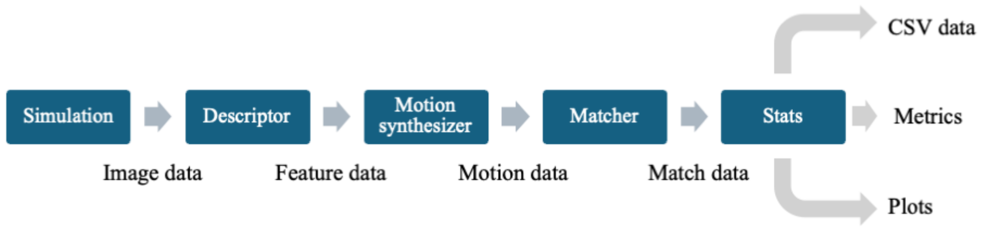

Figure 28: Types of messages that can be transmitted/received through the frontends

The *ImageData* message collects all sensor information and ground-truth data generated from the simulation. A versatile structure was designed, such that additional sensors can be added to the message without affecting the backward compatibility of the message.

The *FeatureData* message has a similar structure to the *ImageData* message but contains a sparse set of features and keypoints instead of the dense image and depth maps. Each keypoint contains the depth value extracted from the depth map in its third dimension, so that the keypoint is effectively expressed as a 3D point in the camera reference frame.

The *MotionData* message removes all information about the sensors present on the spacecraft and instead contains only the sparse feature data for a pair of frames. It specifically contains the sparse set of keypoints, depths and features, as well as the projection matrix to pass from one frame to another.

7.3 Frontends

Three different frontends were developed to allow the numerous modules to send messages to each other: Drive, TCP and ROS. The *astronet_frontends* package contains the source code for all frontends.

The Drive frontend stores the messages into a persistent filesystem, such as the computer's hard drive. The advantage of this frontend is that all elements from the datasets are simultaneously stored and can be retrieved very efficiently. For instance, this is advantageous if we do not want to create a simulation dataset on the fly, which spares a significant amount of computing power and speeds-up the whole pipeline.

The TCP frontend allows to have a real-time pipeline between multiple computers. This could be used if we have multiple computers contributing to the pipeline, such as one computer simulating the asteroid images and another one running the attitude estimation algorithm.

The ROS frontend allows a low-latency communication between elements of the pipeline. This will be useful in the future for the development of a vision-in-the-loop control system, which would require real time access to simulation data and closed-loop feedback to the simulator.

When training our algorithm, the drive frontend is used to store intermediate datasets without the need of regenerating them every time. Nevertheless, accessing sequentially elements from the dataset can significantly slow down the learning process. To overcome this limitation, we developed a wrapper that can encapsulate any frontend and provide parallel access to the wrapped resources. The use of multiple processes and synchronization mechanisms through the python multiprocessing library ensures a minimal access latency, which is a critical step to optimize the performance of our training pipeline.

7.4 Simulation

Nvidia Isaac Sim is used to generate realistic renderings of the asteroids. The USD model exported from Blender is incorporated into an Nvidia Omniverse scene and rendered with raytracing.

Although the *Simulation* module is uses a different python environment that the rest of the pipeline, it still makes use of the frontend and messaging architecture described in sections and .
The module's parameters can be configured according to table 18.

Parameter name	**Description**
type	Type of frontend used
path	Path identifier to which the frontend is connected
dataset_size	Number of images that must be rendered
mode	Target dataset. Can be one of train, validate, test.

Table 18: Simulation module parameters

The module outputs rendered data as a list of *ImageData* messages in the specified frontend. Frames are generated sequentially by first loading an asteroid USD model, adding it to the scene, changing its orientation and rendering the frame. The simulator is designed is such a way that the environment or spacecraft configuration can be easily changed, thanks to a modular software architecture, depicted in 29.

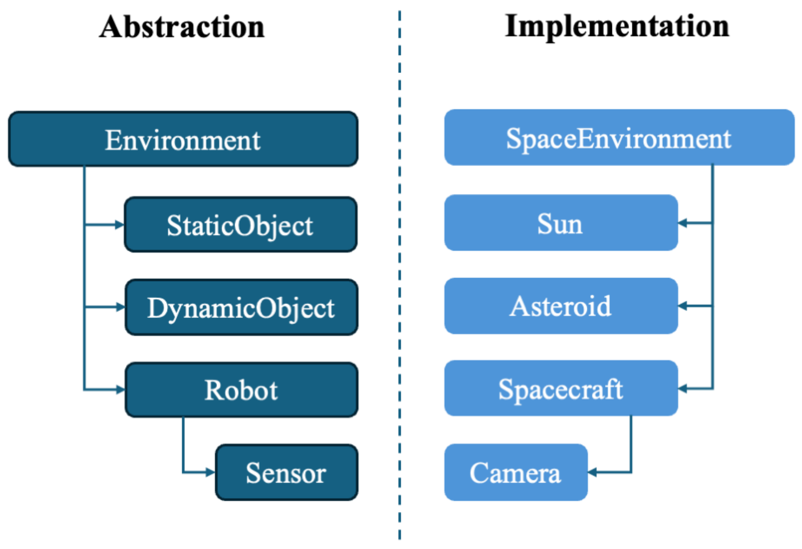

Figure 29: Simulation architecture

The *Environment*, *StaticObject*, *DynamicObject* and *Robot* classes define the abstractions necessary to render a scene. The Environment contains all renderable objects in the scene. The *StaticObject* class defines elements of the scene which are not expected to change over time, such as the Sun in the *SpaceEnvironment*. The *DynamicObject* class defines moving objects in the scene, such as the Asteroid in the *SpaceEnvironment*. Finally, the *Robot* class is an abstraction that encompasses a set of sensors centered at a given location, such as the *Spacecraft*, which contains a single *StandardCamera* sensor. Finally, the *SpaceEnvironement* and *Spacecraft* classes are integrated with Isaac Sim's pipeline and frames are rendered.

7.5 Feature description

The *feature_descriptor* module transforms the *ImageData* from the simulator by extracting a set of relevant keypoints and assigning a description vector to each of them. This module provides a universal interface that can be used in junction with any state-of-the-art detection and description algorithm. Table 19 lists all configuration parameters for the feature_description module.

Parameter name	**Description**
input.type	Type of frontend used for the input
input.path	Path identifier to which the input frontend is connected
output.type	Type of frontend used for the output
output.path	Path identifier to which the output frontend is connected
backend	Feature description algorithm to use
dataset_size	Number of images that must be described
mode	Target dataset. Can be one of train, validate, test.

Table 19: Feature description module parameters

The feature description module takes a list of *ImageData* as an input through the input frontend and outputs a list of *FeatureData* through the output frontend. The *feature_descriptor* module provides a Backend class, to which all feature descriptors must comply. This class contains a single abstract method *detect_feature* that must be implemented by all feature descriptors. This method takes an image as an input and outputs the set of detected keypoints and their descriptors.

7.6 Motion synthesis

The purpose of the *motion_synthesis* module is to create pairs of two successive *FeatureData*. This will allow the feature matcher to compare the features of the two successive frame and find a set of correspondences. Table 20 lists all parameters that can be set in the motion synthesis module.

Parameter name	**Description**
input.type	Type of frontend used for the input
input.path	Path identifier to which the input frontend is connected
output.type	Type of frontend used for the output
output.path	Path identifier to which the output frontend is connected
dataset_size	Number of images that must be described
mode	Target dataset. Can be one of train, validate, test.

Table 20: Motion synthesizer module parameters

The feature description module takes a list of *FeatureData* as an input through the input frontend and outputs a list of *MotionData* through the output frontend.

7.7 COFFEE trainer

Our description and matching pipeline are jointly trained using PyTorch [53], a deep-learning auto-differentiation library. The *coffee_nn* package contains all code related to training our algorithm. The pipeline parameters required by this module are stated in table 21.

Parameter name	**Description**
input.type	Type of frontend used for the input
input.path	Path identifier to which the input frontend is connected
train_size	Size of the training dataset
validate_size	Size of the validation dataset
descriptor_config.model_path	Path to store the best weights of the descriptor model
matcher_config.model_path	Path to store the best weights of the matcher model

Table 21: COFFEE trainer module parameters

The input dataset is first normalized, batched and uploaded to the GPU. For each batch, an optimizer attempts at find the optimal weights that minimize a given non-convex loss function through gradient descent. The gradients are first initialized to zero.

Then, a forward pass computes the predicted matches by our description and matching model. Pytorch takes care of computing the gradients of every operation by the chain rule. After the forward pass, the loss function is computed and used to compute the gradients of the weights in a backward pass. An Adam optimizer is used to update the weights of the model given the gradients previously computed. The Adam algorithm keeps track of an optimal learning rate for each model weight [61]. The maximum allowed learning rate is set to 10^{-5}, as this hyperparameter helped the loss function to converge reliably. The learning rate is scheduled to decay exponentially at a rate of 0.95 per epoch. All hyperparameters are summarized in table 22.

Parameter name	**Value**
Batch size	8
Learning rate	10^{-5}
Learning rate decay	0.95
Epochs	50

Table 22: Training hyperparameters

7.7.1 Loss function

The description and matcher models are aggregated into a common training pipeline. A single loss function is therefore computed from the output of the matcher model and gradients are propagated from the loss through the matcher model to the descriptor model. For a given image pairs, the data points A and B must be matched. A contains n features, whereas B contains m features. The keypoints, input features, ground-truth depths are denoted as follows respectively.

$$x_A \in [0, 1024]^{2\times n}, \quad x_B \in [0, 1024]^{2\times m}$$

$$f_A \in^{1\times n}, \quad f_B \in^{1\times m}$$

$$d_A \in^{1\times n}, \quad d_A \in^{1\times m}$$

Additionally, each image pair stores two matrices to recover the relative projection between the two viewpoints. The intrinsic and extrinsic matrices are denoted as follows respectively.

$$K \in^{4\times 4}, \quad M \in^{4\times 4}$$

The descriptor model $g : (x, f) \to (x, f')$ transforms the input features into described features f'_A and f'_B, where $f'_A \in^{256\times n}$ and $f'_B \in^{256\times m}$. The described features are then forwarded to the matcher model $h : (x_A, f'_A, x_B, f'_B) \to S_{AB}$, where S_{AB} is a score matrix for which each entry s_{ij}, $0 \leq i \leq n, 0 \leq i \leq m$ represents the probability that the feature i of input A corresponds to the feature j of input B. The last row and columns of the score matrix S are called the dustbin entries, since they represent the probability that a given feature does not have any match on the other input. This score matrix contains the prediction of our models and must be compared against the ground-truth score matrix. The latter is computed by projecting all points of input A to the viewpoint of input B. To this end, the input A's screen coordinates are extended with their depth values and transformed from the A frame to the B frame through the camera's intrinsic and extrinsic matrices. The keypoint location x_A can be expressed in the B reference frame as follows:

$$x_A^B = PK_B M_B M_A^{-1} K_A^{-1} (x_A | d_A)$$

where P is the projection operator

$$P = ((1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0))$$

$x_A|d_A$ denotes the concatenation of the 2D keypoint location and their depth value. We can then build the distance matrix D such that each element d_{ij} is defined as:

$$d_{ij} = ||x_{B,i}^A - x_{B,j}^B||$$

where $||.||$ denotes the L_2 Euclidean distance. To determine if the keypoints from input A and B correspond to each other in the same reference frame, we compare their distance to the unit length. If two keypoints are closer than one pixel, they are said to correspond with one another. Additionally, the LightGlue architecture enforces the constraint that there is at most one correspondence for each keypoint. Therefore, for each row in the ground-truth matrix, only the minimum distance match is kept. The matrix thus obtained is concatenated with a dustbin row and column, so that each row or column that does not contain any match has a unit value in its dustbin. This full criterion is applied to the distance matrix D and is denoted as the match matrix $C = c(D)$. The loss function used to train our algorithm is the cross-entropy loss, which can be computed with the score S and the ground-truth matches C such that:

$$L = -\sum_i \sum_j s_{ij} c_{ij}$$

Regularization of the loss function is achieved by introducing a uniform noise to the ground-truth match matrix C.

7.7.2 Online domain randomization

The synthetic dataset already performs offline domain randomization by sampling from a distribution to select the asteroid's base mesh, its surface roughness, orientation and motion, as well as the number and size of craters and boulders. Additional domain randomization can be performed by transforming the detected keypoints and features before learning from them. Before passing the data through the descriptor model, gaussian noise is added to the normalized features. Moreover, all keypoints are transformed by a random translation, scaling and rotation before passing through the matcher model. Note that the keypoint transformation is applied only after passing through the descriptor model, since the convolutional neural network are by design not scale-invariant. The distributions used for the online domain randomization are stated in table 23.

Parameter name	**Value**
Feature value	$N(0, 0.1)$
Scale	$N^2(1, 0.5)$
Translation	$N^2(0, 512)$
Rotation	$U(0, 2\pi)$

Table 23: Distributions for online domain randomization

7.8 Benchmarking

The benchmarking module evaluates an algorithm against a given dataset by computing essential statistics about the ability of an algorithm to predict correct matches. The feature matching algorithm and matching criterion is also passed to this module, as it must sweep through multiple matcher configurations to produce the Precision-Recall or the Receiver Operating Characteristics for instance. All configurable parameters are stated in table 24.

Parameter name	**Description**
input.type	Type of frontend used for the input
input.path	Path identifier to which the input frontend is connected
output.path	Path to which to write the resulting plots and tables
size	Size of the benchmarking dataset
config.features.matcher	Type of feature matching algorithm to use
config.features.criterion	Type of criterion to produce the match matrix
config.features.criterion_args	Initial value for the sweepable criterion

Table 24: Benchmarker module parameters

This module outputs a set of plots, tables and figures that pertain to the quantitative and qualitative results shown in 5 and 6 respectively.

8 Conclusion

By carefully analyzing the effect of self-cast shadows on asteroids, we found an efficient algorithm to extract keypoints from images taken onboard a spacecraft. The 3D geometry of the asteroid, embedded in a 2D shadow and extracted as a 1D boundary proved to be sufficiently informative to recover the pose of the asteroid. To this end, we designed a deep-learning model to efficiently capture all information about this 1D boundary. Employing a Residual Network architecture, implemented as a Submanifold Sparse Neural Network, we could describe each extracted keypoint by a unique FP-256 vector. An attention-based mechanism, alongside with a Graph Neural Network, was then used to find feature correspondences between the two images. A robust RANSAC scheme was then applied with the 5-points algorithm to finally infer the relative pose of the asteroid between subsequent frames.

In addition to qualitative results, we provided quantitative metrics proving that our algorithm performed better than classical algorithms in terms of precision, recall, F_1 score, ROC AUC, PR AUC and other metrics. The benchmarking dataset was generated with the observed shape models of the Apophis asteroid and enhanced by adding craters, boulders and surface roughness to it. Moreover, COFFEE runs at least 3x faster than the other benchmarked deep-learning algorithms.

We nevertheless acknowledge that the large domain gap faced by the deep learning benchmarking algorithms could have slightly skewed the results in our favor, but this does not affect any conclusion regarding the higher computational efficiency of COFFEE. We believe that the absolute accuracy in pose estimation and the very fast runtime of our method could make it a good candidate for future asteroid missions during their orbit synchronization and final approach phase.

This research could be extended by exploring the benefits of online domain adaptation for the pose estimation pipeline and integrating the latter within a complete Simultaneous

Localization and Mapping (SLAM) system to increase the robustness and accuracy of the method. Moreover, COFFEE could be integrated in a complete vision-in-the-loop control system, so that the robustness and efficiency of our method would benefit the whole navigation stack of the spacecraft.

References

[1] S. R. Chesley, "Potential impact detection for near-earth asteroids: The case of 99942 apophis (2004 mn4)," *Proceedings of the International Astronomical Union*, vol. 1, no. S229, 215–228, 2005. DOI: 10.1017/S1743921305006769.

[2] L. P. Institute, J. L. Dotson, M. Brozovic, *et al.*, "Apophis specific action team report," Tech. Rep., 2022. [Online]. Available: https://pubs.usgs.gov/publication/70238398.

[3] D. N. DellaGiustina, M. C. Nolan, A. T. Polit, *et al.*, "Osiris-apex: An osiris-rex extended mission to asteroid apophis," *The Planetary Science Journal*, vol. 4, no. 10, p. 198, 2023. DOI: 10.3847/PSJ/acf75e. [Online]. Available: https://dx.doi.org/10.3847/PSJ/acf75e.

[4] M. Kueppers, P. Martino, I. Carnelli, and P. Michel, "Ramses – ESA's Study for a Small Mission to Apophis," in *AAS/Division for Planetary Sciences Meeting Abstracts*, ser. AAS/Division for Planetary Sciences Meeting Abstracts, vol. 55, Oct. 2023, 201.08, p. 201.08.

[5] P. Pravec, P. Scheirich, J. Ďurech, *et al.*, "The tumbling spin state of (99942) apophis," *Icarus*, vol. 233, pp. 48–60, 2014, ISSN: 0019-1035. DOI: https://doi.org/10.1016/j.icarus.2014.01.026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019103514000578.

[6] I. A. D. Nesnas, B. J. Hockman, S. Bandopadhyay, *et al.*, "Autonomous exploration of small bodies toward greater autonomy for deep space missions," *Frontiers in Robotics and AI*, vol. 8, 2021, ISSN: 2296-9144. DOI: 10.3389/frobt.2021.650885. [Online]. Available: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2021.650885.

[7] Z. Chen and J. Jiang, "Crater detection and recognition method for pose estimation," *Remote Sensing*, vol. 13, no. 17, 2021, ISSN: 2072-4292. DOI: 10.3390/rs13173467. [Online]. Available: https://www.mdpi.com/2072-4292/13/17/3467.

[8] B Leroy, G Medioni, E Johnson, and L Matthies, "Crater detection for autonomous landing on asteroids," *Image and Vision Computing*, vol. 19, no. 11, pp. 787–792, 2001, ISSN: 0262-8856. DOI: https://doi.org/10.1016/S0262-8856(00)00111-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885600001116.

[9] N. Takeishi, T. Yairi, Y. Tsuda, F. Terui, N. Ogawa, and Y. Mimasu, "Simultaneous estimation of shape and motion of an asteroid for automatic navigation," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2861–2866. DOI: 10.1109/ICRA.2015.7139589.

[10] M. Dor, K. A. Skinner, P. Tsiotras, and T. Driver, "Visual slam for asteroid relative navigation," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 2066–2075. DOI: 10.1109/CVPRW53098.2021.00235.

[11] D. Nakath, J. Clemens, and C. Rachuy, "Active asteroid-slam," *Journal of Intelligent & Robotic Systems*, vol. 99, no. 2, pp. 303–333, 2020. DOI: 10.1007/s10846-019-01103-0. [Online]. Available: https://doi.org/10.1007/s10846-019-01103-0.

[12] S. Sharma, C. Beierle, and S. D'Amico, "Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks," in *2018 IEEE Aerospace Conference*, 2018, pp. 1–12. DOI: 10.1109/AERO.2018.8396425.

[13] T. H. Park and S. D'Amico, "Robust multi-task learning and online refinement for spacecraft pose estimation across domain gap," *Advances in Space Research*, vol. 73, no. 11, pp. 5726–5740, 2024, Recent Advances in Satellite Constellations and Formation Flying, ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2023.03.036. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0273117723002284.

[14] A. Kaluthantrige, J. Feng, and J. Gil-Fernández, "Cnn-based image processing algorithm for autonomous optical navigation of hera mission to the binary asteroid

didymos," *Acta Astronautica*, vol. 211, pp. 60–75, 2023, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2023.05.029. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576523002709.

[15] Chen, Shihan, Wu, Bo, Li, Hongliang, Li, Zhaojin, and Liu, Yi, "Asteroid-nerf: A deep-learning method for 3d surface reconstruction of asteroids," *A&A*, vol. 687, A278, 2024. DOI: 10.1051/0004-6361/202450053. [Online]. Available: https://doi.org/10.1051/0004-6361/202450053.

[16] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: 7th international joint conference on Artificial intelligence*, vol. 2, 1981, pp. 674–679.

[17] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis*, J. Bigun and T. Gustavsson, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370, ISBN: 978-3-540-45103-7.

[18] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185–203, 1981, ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(81)90024-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370281900242.

[19] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, Springer, 2020, pp. 402–419.

[20] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462–2470.

[21] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8934–8943.

[22] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, 1988, pp. 147–151.

[23] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[24] D. Viswanathan, "Features from accelerated segment test (fast)," 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:17031649.

[25] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, pp. 337–33 712. DOI: 10.1109/CVPRW.2018.00060.

[26] Y. Ono, E. Trulls, P. Fua, and K. M. Yi, "Lf-net: Learning local features from images," *Advances in neural information processing systems*, vol. 31, 2018.

[27] M. Tyszkiewicz, P. Fua, and E. Trulls, "Disk: Learning local features with policy gradient," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 254–14 265, 2020.

[28] M. Dusmanu, I. Rocco, T. Pajdla, *et al.*, "D2-net: A trainable cnn for joint description and detection of local features," in *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 2019, pp. 8092–8101.

[29] J. Revaud, C. De Souza, M. Humenberger, and P. Weinzaepfel, "R2d2: Reliable and repeatable detector and descriptor," *Advances in neural information processing systems*, vol. 32, 2019.

[30] Z. Luo, T. Shen, L. Zhou, *et al.*, "Contextdesc: Local descriptor augmentation with cross-modality context," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2522–2531. DOI: 10.1109/CVPR.2019.00263.

[31] A. Barroso-Laguna, E. Riba, D. Ponsa, and K. Mikolajczyk, "Key. net: Keypoint detection by handcrafted and learned cnn filters," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 5836–5844.

[32] C. Ernst, T. Daly, O. Barnouin, and R. Espiritu, *Dart raw images for the didymos reconnaissance and asteroid camera for opnav (draco) instrument v3.0*, urn:nasa:pds:dart:data_dracoraw::3.0, Status: IN LIEN RESOLUTION - CERTIFIED, 2023. DOI: 10.26007/6s1n-pk84.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[35] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, Springer, 2015, pp. 234–241.

[36] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015. [Online]. Available: http://arxiv.org/abs/1409.4842.

[37] D. McNeely-White, J. R. Beveridge, and B. A. Draper, "Inception and resnet features are (almost) equivalent," *Cognitive Systems Research*, vol. 59, pp. 312–318, 2020. DOI: https://doi.org/10.1016/j.cogsys.2019.10.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389041719305066.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[39] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse convolutional neural networks," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 806–814. DOI: 10.1109/CVPR.2015.7298681.

[40] B. Graham, M. Engelcke, and L. v. d. Maaten, "3d semantic segmentation with submanifold sparse convolutional networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9224–9232. DOI: 10.1109/CVPR.2018.00961.

[41] P. Lindenberger, P.-E. Sarlin, and M. Pollefeys, "Lightglue: Local feature matching at light speed," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 17 581–17 592. DOI: 10.1109/ICCV51070.2023.01616.

[42] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[43] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Second. Cambridge University Press, ISBN: 0521540518, 2004.

[44] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004. DOI: 10.1109/TPAMI.2004.17.

[45] C. Buonagura, M. Pugliatti, and F. Topputo, "Monet: The minor body generator tool at dart lab," *Sensors*, vol. 24, no. 11, 2024, ISSN: 1424-8220. DOI: 10.3390/s24113658. [Online]. Available: https://www.mdpi.com/1424-8220/24/11/3658.

[46] M. Brozović, L. A. Benner, J. G. McMichael, *et al.*, "Goldstone and arecibo radar observations of (99942) apophis in 2012–2013," *Icarus*, vol. 300, pp. 115–128, 2018, ISSN: 0019-1035. DOI: https://doi.org/10.1016/j.icarus.2017.08.032. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019103517302865.

[47] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[48] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[49] P. F. Alcantarilla and T Solutions, “Fast explicit diffusion for accelerated features in nonlinear scale spaces,” *IEEE Trans. Patt. Anal. Mach. Intell*, vol. 34, no. 7, pp. 1281–1298, 2011.

[50] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, 381–395, 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692. [Online]. Available: https://doi.org/10.1145/358669.358692.

[51] A. Zimmermann, *Pose estimation of unknown tumbling bodies: Source code*, commit a4e23c1, 2024. [Online]. Available: https://github.com/Ar-Ion/AsteroidEstimation.

[52] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[53] A. Paszke, S. Gross, F. Massa, *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *Advances in neural information processing systems*, vol. 32, 2019.

[54] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[55] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134. [Online]. Available: https://doi.org/10.5281/zenodo.3509134.

[56] Wes McKinney, “Data Structures for Statistical Computing in Python,” in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 –61. DOI: 10.25080/Majora-92bf1922-00a.

[57] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[58] C. Choy, J. Gwak, and S. Savarese, “4d spatio-temporal convnets: Minkowski convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3075–3084.

[59] C. Choy, J. Lee, R. Ranftl, J. Park, and V. Koltun, “High-dimensional convolutional networks for geometric pattern recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[60] J. Gwak, C. B. Choy, and S. Savarese, “Generative sparse detection networks for 3d single-shot object detection,” in *European conference on computer vision*, 2020.

[61] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.