COMET INTERCEPTOR TRAJECTORY GENERATOR FOR COSMOGRAPHIA VISUALIZATION.

Prepared by	Jaime Fernández Diz

Document Type	MAN - Manual / User Guide / Handbook
Reference	CITGCV
Issue/Revision	1 1.
Date of Issue	11/05/2022
Status	Draft

APPROVAL

Title	Comet interceptor trajectory generator for cosmographia visualization.		
Issue Number	1	Revision Number	
Author	Jaime Fernández Diz	Date	11/05/2022
Approved By		Date of Approval	

CHANGE LOG

Reason for change	Issue Nr	Revision Number	Date
Initial release	1	1	11/05/2022

CHANGE RECORD

Issue Number	1	Revision Number	
Reason for change	Date	Pages	Paragraph(s)

DISTRIBUTION

Name/Organisational Unit

Table of Contents

List of tables

List of figures

1. INTRODUCTION

In the framework of the Comet Interceptor mission, a parametric trajectory generator for Cosmographia is developed. This document aims to explain the main functionalities of the current version of said software, as well as the algorithm and conventions taken for its development, and suggestions on future improvements to implement.

2. SET UP

To visualize pre-generated trajectories on Cosmographia, two elements are needed: Spice Enhanced Cosmographia, and the cosmos repository for the Comet Interceptor mission. Cosmographia can be downloaded from this webpage, and the repository of Comet Interceptor from this one. If the parametric trajectory generator is going to be used, python 3 is also needed, and the mkspk app provided by naif here (choose the proper operative system).

Once all the elements are downloaded, some modifications have to be done to the git repository:

- In comet-interceptor/misc/cosmo/spice_interceptor.json, change the “spiceKernels” variable to “../../kernels/mk/interceptor_study_v02.tm” (remove the _local part of the name).

```
1 {
2    "version": "1.0",
3    "name": "Cosmographia Comet-Interceptor Example",
4    "spiceKernels": [
5       "../../kernels/mk/interceptor_study_v02_local.tm"
6    ]
7 }
```

Figure 1: Remove “_local” from line 5 in this file.

- In comet-interceptor/kernels/mk/interceptor_study_v02.tm and comet-interceptor/kernels/mk/interceptor_study_parametric.tm, change the PATH_VALUES to the absolute path to the kernels folder.

```
1  KPL/MK
2
3  Comet-Interceptor Example Trajectories
4  ======================================
5
6
7     \begindata
8
9       PATH_VALUES       = ( '..' )
10
11      PATH_SYMBOLS      = ( 'KERNELS' )
12
13      KERNELS_TO_LOAD   = (
```

Figure 2: In line 9, substitute the two dots with the full path to the kernels folder.

Now the setup is finished. To see the predefined trajectories of the spacecrafts, in cosmographia the file load_interceptor_001.json, located in comet-interceptor/misc/cosmo should be loaded. For generating a new trajectory, the inputs of the trajectory_generator_v02.py should be updated in the source code according to the user's needs and the code should be run. In order to see the last user defined trajectory, the file load_interceptor_parametric.json can be imported like for predefined trajectories.

3. USER GUIDE

The goal of the program is to generate a trajectory for the Comet Interceptor Spacecraft A and B1 and B2 probes.

The central body must be the comet 8P (it is not possible to simulate a heliocentric trajectory with this code). The reference frame on which the trajectory is defined is the "INTERCEPTOR_TUTTLE_ION", a reference frame defined for Comet Interceptor mission, whose z axis points towards the Sun, the x axis is the component of the velocity of the comet with regard to the Sun orthogonal to the z axis, and the y axis completes the right handed frame.

The program takes 14 inputs for trajectory generation and 9 more for selecting the observation dates of each instrument. The inputs are explained in the table below:

Input	Type	Description
relative_velocity	Scalar	Velocity of the spacecraft A relative to 8P during the closest approach in Km/h. It is assumed constant during all the simulation.
date_range	Tuple (2)	Beginning and end dates of the simulation.
B1_separation_date	String	Date when B1 is released from A in ISO format.
B2_separation_date	String	Date when B2 is released from A in ISO format.
Closest_approach_A_date	String	Date of closest approach of A probe to 8P in ISO format.
Closest_approach_B1_date	String	Date of closest approach of B1 probe to 8P in ISO format.
Closest_approach_B2_date	String	Date of closest approach of B2 probe to 8P in ISO format.
Closest_approach_A_v_dir	NumPy array (3)	Direction of the velocity of Spacecraft A during the flyby, expressed in the Interceptor_tuttle_ion reference frame.
Closest_approach_A_dist	Scalar	Distance between A and 8P at the closest approach in Km.
Closest_approach_A_angle	Scalar	Angular position of the closest approach location of A around the comet. See 4.1 for details.
Closest_approach_B1_dist	Scalar	Distance between B1 and 8P at the closest approach in Km.
Closest_approach_B2_dist	Scalar	Distance between B2 and 8P at the closest approach in Km.
Closest_approach_B1_angle	Scalar	Angle in radians between the real and default trajectory of B1. See 4.2 for details.
Closest_approach_B2_angle	Scalar	Angle in radians between the real and default trajectory of B2. See 4.2 for details.
<*INS*>_OBS_range	Pandas series	Beginning and end dates of the observations of each instrument in ISO format.

Table 1: Inputs required for the program.

When run, the program will modify the file obs_interceptor.json according to the inputs given in <*INS*>_OBS_range.

After that, the trajectory of the three spacecrafts is generated and plotted in three 3D diagrams, showing the whole trajectory and the closest approach to the comet.

This done, the program detects and deletes old input and spk files and creates the new ones, and finally, it launches Cosmographia with the kernels loaded.

4. ALGORITHMS AND CONVENTIONS

The goal of the algorithms of the program are to calculate the trajectory that accomplishes the closest approach conditions imposed by the user.

Two algorithms are required, one for spacecraft A, where the only restriction is the closest approach position, and other for B1 and B2, where there is an initial condition derived from their separation date and the previously calculated trajectory of A, so only a closest approach distance can be imposed.

Both algorithms are explained in this section.

4.1. Comet interceptor A trajectory.

The inputs taken for this calculation are the velocity vector, the closest approach distance and the trajectory angle. For the vector to be really the closest approach of the spacecraft to the comet, the velocity of the spacecraft in that point should be perpendicular to the position, or the trajectory should be perpendicular to the closest approach vector. Given the direction of the velocity vector and the closest approach distance, there is not a point, but a whole circumference where the closest approach position can be. The default point for the closest approach is defined according to the following convention: the trajectory should intersect the z axis in its positive side, unless the direction is parallel to the z axis, in which case the closest approach vector is contained in the x axis.

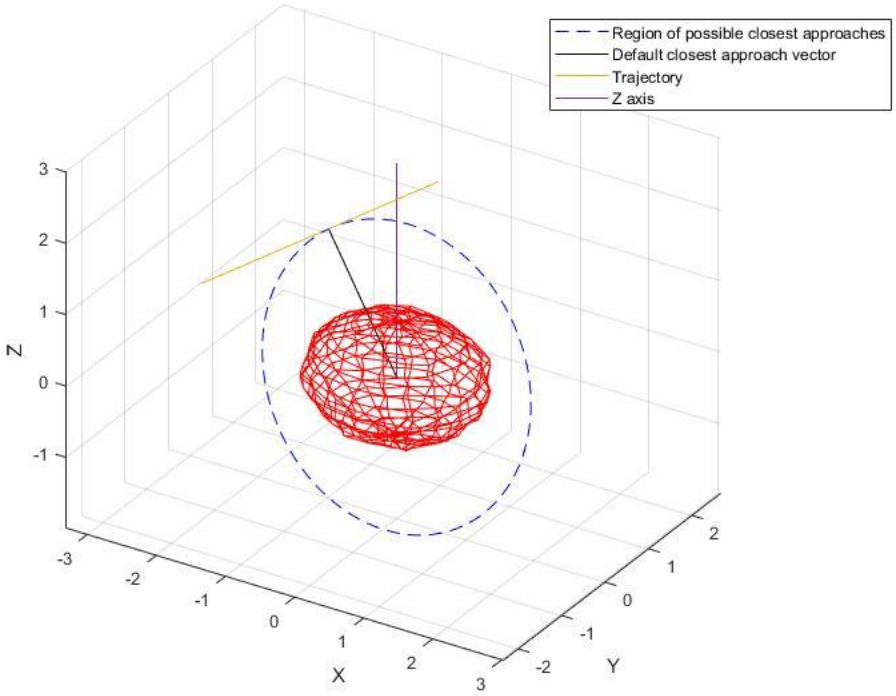

Figure 3: Default velocity for a closest approach not aligned with X.

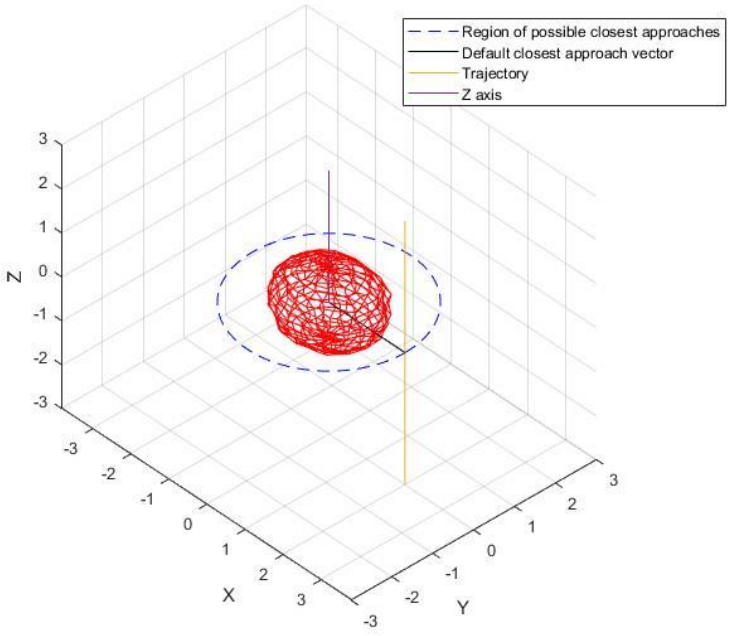

Figure 4. Default velocity for a closest approach aligned with X.

The closest approach vector in the first case is calculated as:

$$\overrightarrow{u_r} = \frac{\overrightarrow{u_v} \times \left(\overrightarrow{u_v} \times \frac{(\overrightarrow{u_v} \times \vec{k})}{|(\overrightarrow{u_v} \times \vec{k})|} \right)}{norm\left(\overrightarrow{u_v} \times \left(\overrightarrow{u_v} \times \frac{(\overrightarrow{u_v} \times \vec{k})}{|(\overrightarrow{u_v} \times \vec{k})|} \right) \right)}$$

Where $\overrightarrow{u_r}$ is the unitary vector in the direction of the closest approach position, $\vec{k}$ the unitary vector in the z direction and $\overrightarrow{u_v}$ the unitary vector in the direction of the velocity relative to the comet.

In the second case, the closest approach vector is just:

$$\overrightarrow{u_r} = \vec{\imath}$$

With $\vec{\imath}$ being the unitary vector in the x direction.

After that, the closest approach vector is rotated the angle defined by the user in "Closest_approach_A_angle", around the direction vector, following the "right hand rule".

On the code, the rotation is performed with the function *"rotate_around_axis",* whose output is the same vector if it is aligned with the rotation axis, or the rotated vector if not, calculated as follows:

$$\vec{r}' = \vec{r}_{par} + \cos\theta \times \vec{r}_{perp} + \sin\theta \times \vec{z}$$

$$\vec{z} = \vec{u}_{ax} \times \vec{v}_{perp}$$

In this expression, $\vec{r}_{par}$ is the component of the vector parallel to the rotation axis, $\vec{r}_{perp}$ the component perpendicular to the rotation axis, theta the angle of rotation and $\vec{u}_{ax}$ the unitary vector in the direction of the rotation axis, which coincides with the unitary vector in the direction of the velocity.

With this rotated velocity $\vec{r}'$, the trajectory is propagated. As the velocity is considered constant for this simulation, this results in a straight line, perpendicular to the closest approach vector. The propagation is done with a gauss method, as follows:

$$\vec{r} = \vec{r}_{ap} + \left(t - t_{ap}\right) * \vec{v}$$

Where $\vec{r}$ is the position of the spacecraft in each instant of time, $\vec{r}_{ap}$ and t_{ap} are the closest approach position vector after the rotation and instant of the closest approach, and $\vec{v}$ is the velocity vector.

In summary:

- 1 – Check if the velocity vector is aligned with the Z direction.
  - If it is, $\overrightarrow{u_r} = \vec{\imath}$
  - If it is not, $\overrightarrow{u_r}$ is perpendicular to the velocity vector and makes the trajectory intersect the Z axis in its positive side.
- 2 – The closest approach vector is rotated the angle specified by the user around the axis defined by the velocity vector.
- 3 – With the boundary condition of the position and the velocity at a given time, the trajectory is propagated forwards and backwards up to the limits of “date_range”.

4.2. Comet interceptor B1 and B2 trajectory.

In the case of B1 and B2 there is not a closest approach vector, but a distance and an initial condition, so the problem is finding a vector with module equal to the closest approach distance of the probes, whose perpendicular plane contains the point where the spacecrafts B1 and B2 are separated from A, and then proceed like in the previous case. In the code, this is done in the function *“find_r_approach_point”.*

Again, there is not one but a whole family of vectors which have this property. The additional constraint was arbitrarily defined as that the trajectory of the probes shall cross the Z axis, or the X axis if the separation point is contained in the Z axis.

For the calculation of the closest approach vector, first an axis orthogonal to the separation vector is defined:

$$\vec{u}_o = \vec{u}_s \times (\vec{u}_s \times \vec{z})$$

If the separation point is contained in the z axis, instead:

$$\vec{u}_o = \vec{u}_s \times (\vec{u}_s \times \vec{x})$$

The separation vector $\vec{u}_s$ is the position of the spacecraft A relative to the comet at the date of separation of each of the probes, so it is obtained in the previous step.

With this definition of $\vec{u}_o$, it is contained in the same plane as the one defined by the separation vector and the x or z axis, thus accomplishing the restriction that the trajectory should cross one of those axis.

$\vec{u}_o$ and $\vec{u}_{r_{sep}}$define an orthogonal base for the closest approach distance vector of the probes.

Furthermore, the closest approach vector, the trajectory, and the position of the separation of the probes define a rectangle triangle, with angle θ between the closest approach and separation vectors, and φ the angle between the closest approach and $\vec{u}_o$. The geometry is depicted in figure 3:

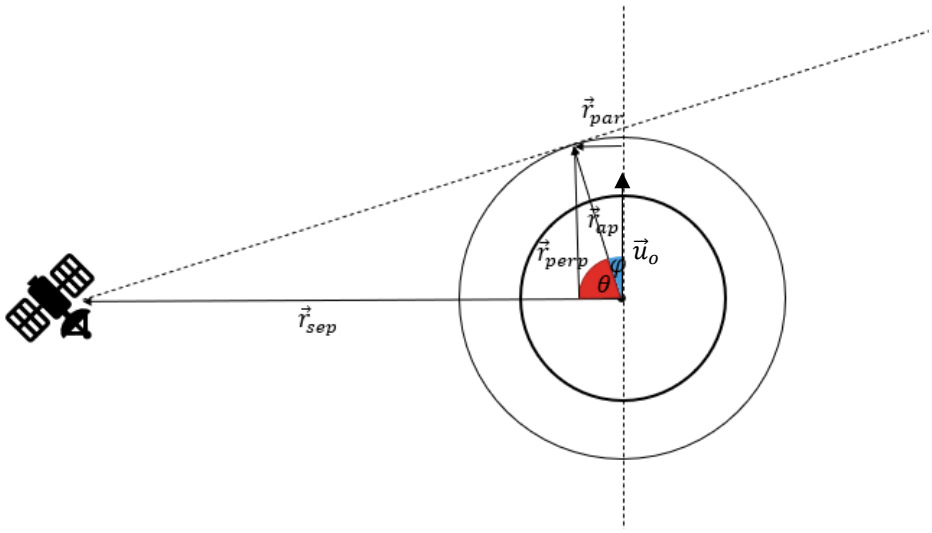

Figure 5: Geometry for B1 and B2 closest approach

The closest approach vector will be the value of the closest approach distance d, specified by the user, projected along $\vec{u}_o$ and $\vec{u}_{r_{sep}}$ according to φ. Once the vector is obtained, it can be rotated around the axis defined by $\vec{u}_{r_{sep}}$ like in the previous case.

With the closest approach vector obtained, the velocity is calculated as the difference between the separation and closest approach positions divided the time span between both, as:

$$\vec{v} = \frac{\vec{r}_{ap} - \vec{r}_{sep}}{t_{ap} - t_{sep}}$$

With the velocity and the closest approach vector obtained, the trajectory is propagated with the same method used for spacecraft A.

5. SUGGESTED IMPROVEMENTS

- Add the SpicePy library to manage different reference frames in the program without need to go to the setup files.
- Create a GUI or other tool to allow typing the inputs without need to modify the source code.
- Show the approach from the point of view of an instrument.
- Improving the comet tail visualization (check https://github.com/isenberg/cosmographia_catalogs).
- Implementing the option of changing the target comet in the code, and create the appropriate Cosmographia and spice files.
- Change the parametric trajectory generator so it creates new spks with version tracking instead of deleting the previous trajectory.
- Add a warning if the velocity of the probes after separation and spacecraft A is bigger than a certain threshold.