# USER GUIDE

## FOR ASSESSMENT PACKAGE OF MOTION ESTIMATOR FOR UNCOOPERATIVE TARGET (V1.1)

TIAGO PADOVAN | TETSUYA KUSUMOTO | JUNICHIRO KAWAGUCHI

PATCHEDCONICS, LLC | Kanagawa 252-5210, Japan

# Table of Contents

# Agreement

The contents of this Assessment Package are supposed to be use within one year, free of charge. The provision of this package is for assessment and testing. Once tested and verified, the customer shall acquire the Motion Estimator firmware in a chip to be boarded in the spacecraft.

Results obtained using the contents of this Assessment Package are agreed and encouraged to be published, as long as referring to the name of Patchedconics, LLC.

# Introduction

Space debris have captured the attention of people both in and out of Space Industry. They impose a threat but are a natural consequence of the exploration of space. There are databases of tracked debris and collision risks can be assessed prior to launching a new spacecraft to space. Areas such as Low Earth Orbit (LEO) are specially packed with those debris due to heavy military, governmental and commercial usage.

During the planning phase of a mission, methods to safely deorbit spacecrafts can be studied. Not always those methods are implemented, and satellites may become nonoperational, imposing risk for current and future spacecrafts.

Spacecraft can also be designed with in-orbit servicing capability. This allows a “servicer” satellite to approach, dock and perform maintenance, such as refueling, of another spacecraft. Patchedconics has developed a technology that allows a satellite to identify, using the camera sensor available onboard, precious information on the attitude dynamics of an uncooperative target. Identifying parameters such as the direction of the angular momentum vector and inertia tensor is the key for allowing capture of debris or docking for servicing of satellites in orbit. To this end, estimating the inertial tensor of the target becomes essential. Most of the estimators cannot handle this issue, but the Motion Estimator of Patchedconics copes with it.

## The Assessment Package

The purpose of this package is to provide an assessment tool for the Motion Estimator software to interested people.

The Assessment Package is made of a Motion Simulator and a Motion Estimator. The first reads a set of initial conditions of target and observer and generates a set of projections of the points of interest in a frame representing a camera sensor for each time. The second receives a set of information, containing the projections’ locations on the frame, and estimates parameters regarding the attitude dynamics of the target.

Two Motion Simulators are provided in the package, in Python and MATLAB/Octave, and each of them have differences in the operation method, which will be highlighted in this User Guide. The two of them are designed and tested in Windows 10 and macOS 11 environments.

Two Motion Estimators executables for each operational systems are included in the package. For each of the OS, each of the Estimators works for a specific configuration of the target. Arbitrary configuration of the target is built and defined by the users by modifying the Simulator codes, and the estimated results are obtained.

The Motion Estimators are designed to be used with the Simulator included in the package and also with third-party simulators. A set of inputs and arguments must be provided for the Estimator to function properly, and those inputs will be discussed in this guide.

## What is Included?

| **Assessment Package** | |
|---|---|
| **Operational System**<br>(chosen on download) | Windows / macOS |
| **Simulators Languages** | MATLAB / Python |
| **Estimators*** | For Planar Markers / For Tetrahedron Markers |

The Assessment Package file structure is as follows:

First Level: (**MEAP_[OS]**) Where this User Guide is found. There are 4 folders: **data** is used by the package to transfer information between the Simulator and the Estimator, **results** is where the results of the simulation/estimation are stored, **ME** contains the Motion Estimators executables and auxiliary files and **figures** contains images used by the python version of the Simulator.

Second Level: The **ME** directory contains two folders, **1** and **2**. **1** is for the target configuration in which the markers are distributed on 1 surface. **2** is for the target configuration in which the markers are distributed on 2 surfaces.

*For better estimation results, this package provides two kinds of Estimators tuned for the specific markers layouts. One estimator is optimized for the markers on a single plane, while the other is optimized for the markers on two planes (on a tetrahedron).

## Motion Estimator

As mentioned in the Introduction, two Estimators are provided for each OS. They are executable files which are exclusive to each of the supported markers configurations. The executable files can be called using the provided Simulators written in MATLAB and Python, as well as other simulators developed by the users, as long as the input format is respected. The Estimators for both macOS and Windows accept the same file format as input, and outputs in the same format. Each of the estimators shall be used with the specific markers configuration (planar or tetrahedral).

The idea of the Motion Estimator is to provide to the observer with estimated angular momentum, *L*, angular velocity, $\omega$, and inertia tensor, *I*, given the information of the projection of 4 markers, which are fixed on the surface of the target, on a figure. The estimator also outputs the estimated position and orientation of the target.

### For Using with Third-Party Simulators

The operation of the Motion Estimator in the Assessment Package can be described by a simple diagram:

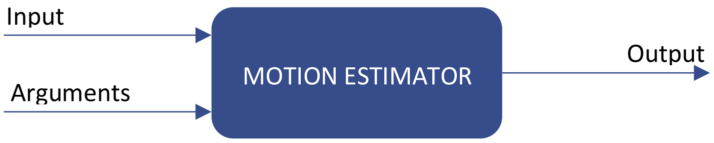


Figure 1: Assessment Package's Motion Estimator diagram

*Input* is a set of data written as a comma separated CSV file with no header. It is composed of a single line containing the following information:

| **Marker 1, px** | | **Marker 2, px** | | **Marker 3, px** | | **Marker 4, px** | | ***I, L* *update, steps*** | **Start, sec** | **Time, sec** | **Interval, sec** | **Memory, sec** | **reset** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_{1,i}$ | $y_{1,i}$ | $x_{2,i}$ | $y_{2,i}$ | $x_{3,i}$ | $y_{3,i}$ | $x_{4,i}$ | $y_{4,i}$ | $IL$ | $ST$ | $t_i$ | $dte$ | $B$ | $reset$ |

- The markers are assumed to be identified, so $x_{1,i}$, for example, is always the position in the **X** axis of the projection of **Marker 1** [in pixels]
- $IL$ is the amount of stored Inertia Tensor (*I*) estimations required for the Angular Momentum (*L*) to be updated (we recommend using $IL = dte$) [in steps]
- $ST$ is the amount of time taken from $t_0$ to the first estimation results (we recommend using $ST = 0$) [in seconds]
- $t_i$ is the time in which the projections were captured [in seconds]
- $dte$ is the expected interval, $t_i - t_{i-1}$ [in seconds]
- $B$ is time span of most recent estimations stored by the estimator (we recommend using $B$ to be near the nutation period or longer, but even shorter values shall work) [in seconds]
- $reset$ is a flag – when $reset = 1$ the information stored by the estimator is erased and when $reset = 0$ the estimator saves the information for current and future estimation.

*Arguments* are 4 strings:

| Argument 1 | Argument 2 | Argument 3 | Argument 4 |
|---|---|---|---|
| *Input file name* | *Directory* | *Output file name* | *Markers shape file* |

In order to generate an output, the 4 markers must be visible. If one or more markers are hidden or out of the frame, the estimator returns same results as $t_{i-1}$ and waits for the 4 markers to be again visible to continue the estimations.

## Assumptions

As mentioned in the previous section:

- The 4 markers are assumed to be distinguished by the image processor.

Additional assumptions are:

1 The 4 markers maintain their relative positions, meaning that the tetrahedron formed by the markers maintain its shape and size as defined in the file whose path is the fourth argument
2 The projection is made in a 2000 x 2000 pixels frame
3 The positions of the markers in the frame have their origin at the center of the frame, so the edges of the frame are [-1000, -1000], [-1000, 1000], [1000, -1000], [1000, 1000]
4 If any of the projections have **X** or **Y** components larger than 1000 or smaller than -1000, the marker is considered not visible.

In case the markers form a different geometry than that specified in the file, the estimator will not work as expected. And in case the distances between the markers are different (different scale), the estimator will provide scaled information on the position of the target.

Two version of each simulator are provided for each operational system. One assumes the 4 markers are placed in 2 different surfaces (3 in a surface and 1 in the other adjacent surface), the other assumes the markers are all coplanar (all markers on same surface). For each of the cases, the user has to respect the markers geometry set in the file provided to the Estimator:

1 Markers disposed on 2 surfaces:
In the case of the markers being distributed in different surfaces of the spacecraft, an example of configuration can be seen on figure 2.
The coordinates are in meters.

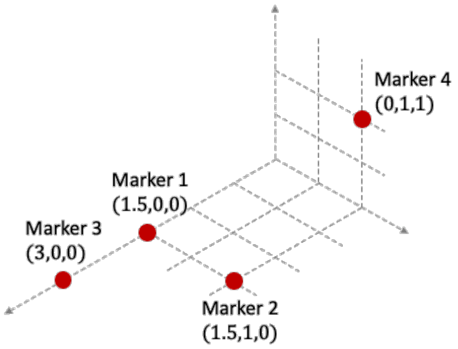


Figure 2: Example of markers configuration on 2 different planes

2 Markers disposed on a single surface:
In case of the markers being disposed in a coplanar configuration, an example of setup can be seen on figure 3.
The coordinates are in meters.

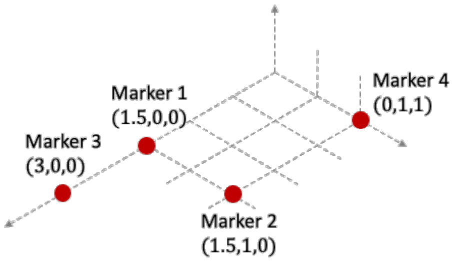


Figure 3: Example of coplanar markers configuration

Each of the configurations shown above work with a specific Estimator. For the markers disposed on 2 surfaces use the set of files in the subfolder `ME/2`. For the markers disposed in a single plane, use the set of files in the subfolder `ME/1`.

## Markers Configuration Describing File

On Version 1.1, the option to define the configuration of the markers and provide it to the Estimator was added. The user can define three coordinates for each marker in a body fixed frame. Results of the estimator (regarding attitude dynamics) will be with respect to that same coordinate system.

For simplification, we made the examples described on figures 2 and 3 predefined in the MATLAB/Octave (**MS.m**) and in the Python (**MS.py**) simulators.

The file which defines the position of the markers is in csv format and it contains a single row with 12 entries. Each element is a coordinate of the markers in meters:

| Marker 1 | | | Marker 2 | | | Marker 3 | | | Marker 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | $y_1$ | $z_1$ | $x_2$ | $y_2$ | $z_2$ | $x_3$ | $y_3$ | $z_3$ | $x_4$ | $y_4$ | $z_4$ |

## Simplified Estimator-Calling Algorithm

This Version (1.1) of the Assessment Package was specifically made to give users freedom to explore the Motion Estimator and help them designing their own simulator and their own setups, and for that reason a new “simulator” named `MSsimple.m` (as for now MATLAB/Octave only) was also created.

Although not being the “Simulator” section, this file will be described here, as it does not exactly simulate the dynamics of the target.

`MSsimple.m` script was created to simply generate an input file and a marker configuration file, produce a figure to illustrate the example, and call the Motion Estimator inside `ME/1` directory. This code assumes no movement between target and observer. The Motion Estimator is called twice, since it’s the minimum amount of time to generate an output.

In the first time the Motion Estimator is called, it is reset. On the second time, only variables $t_i$ and $reset$ are modified as

$$t_i = t_i + dte, \quad reset = 1$$

Since the position of the markers is not different between the two instants, the Motion Estimator only outputs the position of the markers and the direction cosine matrix. Other parameters are output with a general value.

The user shall use this script as the simplest reference to usage of the Motion Estimator. It works on both Windows and macOS platforms.

## Limitations of the Motion Estimator

The Motion Estimator has a limitation regarding the angular velocity of the target. If the angular velocity is too large, the estimation will not provide reliable results. For such cases, we recommend decreasing the estimator interval, so information is collected with higher frequency.

# Motion Simulator

Although the executable files of the Motion Estimator can be run using your own inputs, Patchedconics provides in the Assessment Package our own simulator in two languages: Python (**MS.py**) and MATLAB (**MS.m**). Right when you are provided with the Package, you can start performing simulations.

The user may develop their own simulator with features like torque being applied to the target, collision leading to change of angular momentum, fragmentation of solar panels leading to change in inertia tensor. And the simulator provided can be used as a reference on how to generate the input file and call the executable of the Estimator.

## Target Properties

The target spacecraft used in the simulator has a simple geometry composed of 3 cuboids: 1 representing the main body, with dimensions **a** x **b** x **c** and 2 representing the Solar Array Panels (SAPs) with dimensions **d** x **e** x **f** (in the code **f** is named as **g**). The SAPs are connected to the main body's top panel through one of their edges as seen on figure 4.

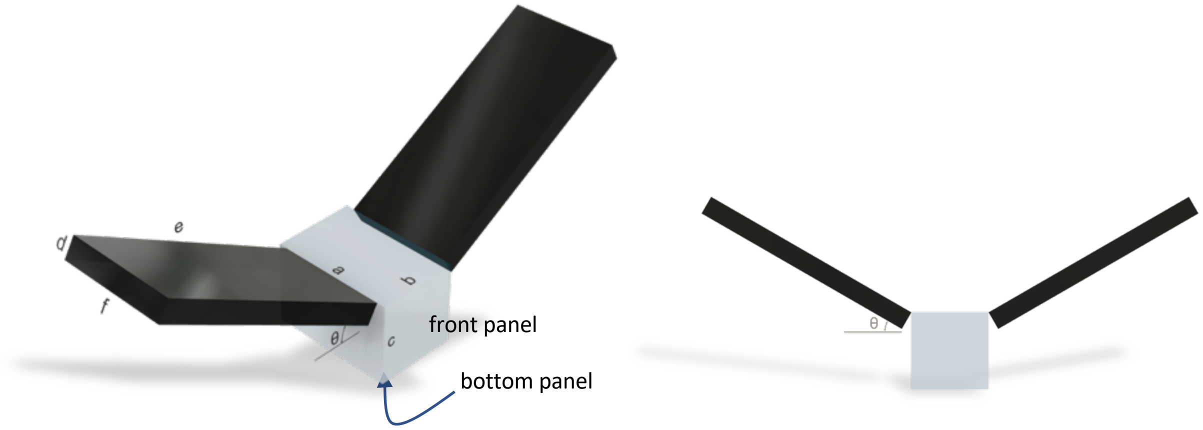


Figure 4: Target geometry used on the provided Motion Simulator

The 4 markers are disposed in the configurations seen on figures 5 or 6. On the first option, three of the markers are placed on the same surface (bottom) and one on an adjacent surface (front). In the second option, the markers are all placed in a single surface (bottom).

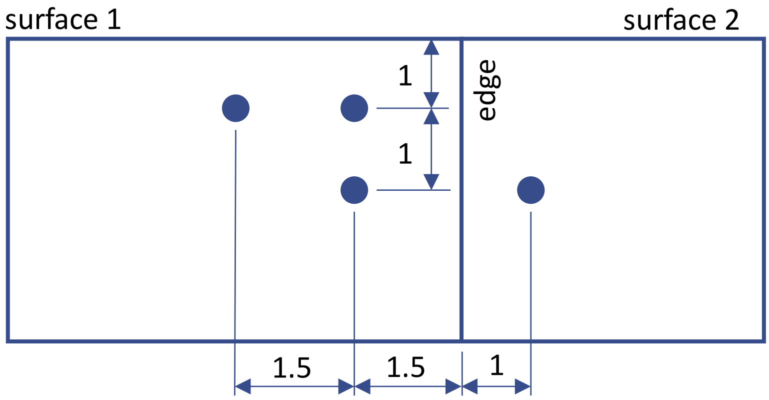


Figure 5: Position of markers on 2 surfaces of target used by the simulator

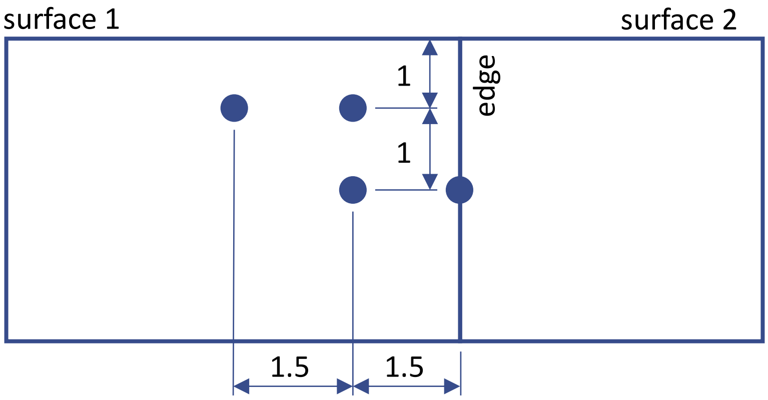


Figure 6: Position of markers in coplanar option used by the simulator

The Markers configurations seen on figures 5 and 6 (example models included in the simulators) are compatible with the marker geometries shown on figures 2 and 3.

## MATLAB/Octave Version (Recommended)

The MATLAB version of the Motion Simulator (**MS.m**) does not have a GUI, so variables shall be modified by the user directly in the script.

We recommend using the MATLAB version of the software because it has shown to run more smoothly in the tests we performed, and the visualization of the results is improved when compared to the Python version. The MATLAB version also includes the option for the user to perform the simulation only, without the estimator, for verification purposes, and generating a simple animation of the motion of the spacecraft (see figure 7). For performance purposes, we recommend not generating the animation of the markers when doing long simulations.

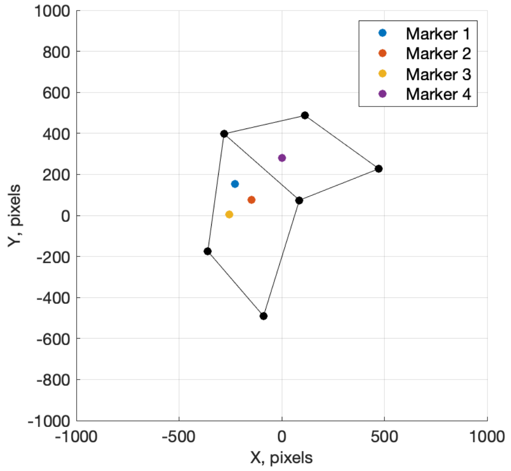


Figure 7: Frame from MATLAB animation of markers' projections

This function (**MS**) contains information to be modified by the user, such as spacecraft dimensions, initial orientation and distance from observer. The variables will be explained in this section. Some variables were already discussed in the Estimator input section, they are **IL**, **ST**, **dte** and **B**.

There one limitation of the Octave version of the Simulator which did not allow the insertion of a header in the results file (**results.csv**). For this reason, we will specify what is each of the 48 columns in the results file:

t, $I_{xx\_true}$, $I_{yy\_true}$, $I_{zz\_true}$, $I_{xy\_true}$, $I_{xz\_true}$, $I_{yz\_true}$, $\omega_{1\_true}$, $\omega_{2\_true}$, $\omega_{3\_true}$, $L_{1\_true}$, $L_{2\_true}$, $L_{3\_true}$, $I_{xx\_est}$, $I_{yy\_est}$, $I_{zz\_est}$, $I_{xy\_est}$, $I_{xz\_est}$, $I_{yz\_est}$, $\omega_{1\_est}$, $\omega_{2\_est}$, $\omega_{3\_est}$, $L_{1\_est}$, $L_{2\_est}$, $L_{3\_est}$, “Markers hidden?”, “Estimator restart?”, Marker $1_x$, Marker $1_y$, Marker $1_z$, Marker $2_x$, Marker $2_y$, Marker $2_z$, Marker $3_x$, Marker $3_y$, Marker $3_z$, Marker $4_x$, Marker $4_y$, Marker $4_z$, $DCM_{11}$, $DCM_{12}$, $DCM_{13}$, $DCM_{21}$, $DCM_{22}$, $DCM_{23}$, $DCM_{31}$, $DCM_{32}$, $DCM_{33}$

## Variables to be modified by the user

**dts**: Simulator step size [seconds]

**dte**: the expected time interval between information sent to the estimator [seconds]

**B**: the time span of stored information by the estimator (recommended: same as nutation period of the target or larger) [seconds]

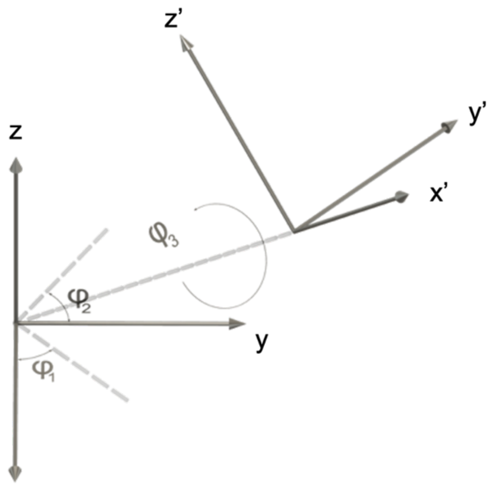


Figure 8: Euler angles

**ST**: the time taken for the estimator to perform first estimations (recommender: 0) [seconds]

**IL**: the number of Moment of Inertia estimations required for next angular momentum estimation Moment of Inertia variable update (recommended: 1) [steps]

**N**: simulation time [seconds]

**f**: camera sensor size (assumed to be square) [pixels]

**M**: Mass of the main body of the target spacecraft [kilograms]

**Msap**: Mass of each Solar Array Panel [kilograms]

**perr**: pixel error (is multiplied by a uniformly distributed random number), which is the error on the pixel reading [pixels]

**point_animation**: the user can choose whether to display an animation of the bottom and front surfaces of the target, as well as the markers (1 is yes, 0 is no)

**estimation**: the user can choose whether to run the estimator (1 is yes, 0 is no)

**a, b, c, d, e, g**: the spacecraft dimensions discussed previously, (g is the length f described on figure 4) [meters]

**theta**: the angle which describes the position of the SAPs, as seen on figure 4, which makes the target symmetric in the **ZX** plane [rad]

**X, Y, Z**: the position of the observer with respect to the center of mass of the target [meters]

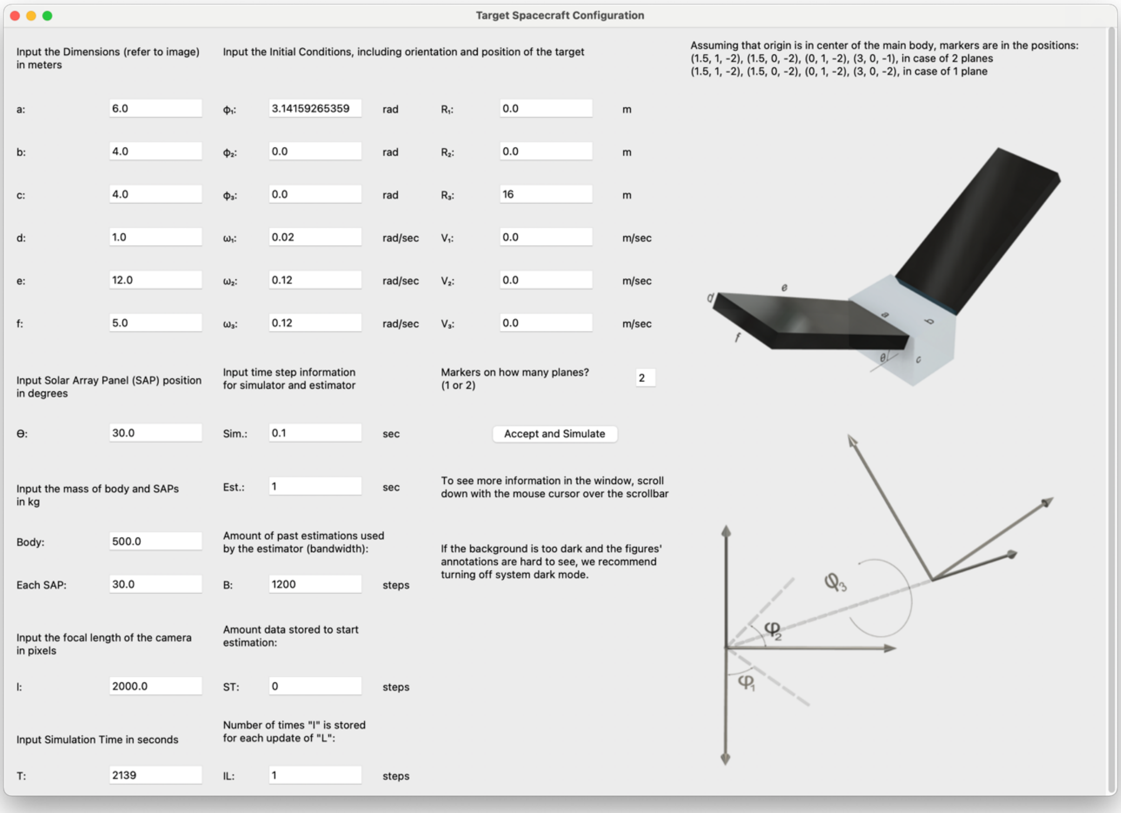
**VX, VY, VZ**: the velocity of the observer [meters/second]

**phi1, phi2, phi3**: the initial Euler angles of the target (yaw, pitch, roll sequence) (see figure 8) [rad]

**om1, om2, om3**: the components of the angular velocity [rad/second]

## Python Version

The Python version is also offered as a source code and it includes a Graphical User Interface (GUI), where the user can input the initial conditions for the simulator and select some other options. Figure 9 shows the GUI and the fields to be modified.
One of the reasons why we recommend the use of the MATLAB code over the Python version is the runtime of the code, which is shorter on the MATLAB version.



**Target Spacecraft Configuration**

Input the Dimensions (refer to image) in meters

a: 6.0
b: 4.0
c: 4.0
d: 1.0
e: 12.0
f: 5.0

Input the Initial Conditions, including orientation and position of the target

$\phi_1$: 3.14159265359 rad
$\phi_2$: 0.0 rad
$\phi_3$: 0.0 rad
$\omega_1$: 0.02 rad/sec
$\omega_2$: 0.12 rad/sec
$\omega_3$: 0.12 rad/sec

$R_1$: 0.0 m
$R_2$: 0.0 m
$R_3$: 16 m
$V_1$: 0.0 m/sec
$V_2$: 0.0 m/sec
$V_3$: 0.0 m/sec

Assuming that origin is in center of the main body, markers are in the positions:
(1.5, 1, -2), (1.5, 0, -2), (0, 1, -2), (3, 0, -1), in case of 2 planes
(1.5, 1, -2), (1.5, 0, -2), (0, 1, -2), (3, 0, -2), in case of 1 plane

Input Solar Array Panel (SAP) position in degrees

ϴ: 30.0

Input the mass of body and SAPs in kg

Body: 500.0
Each SAP: 30.0

Input the focal length of the camera in pixels

I: 2000.0

Input Simulation Time in seconds

T: 2139

Input time step information for simulator and estimator

Sim.: 0.1 sec
Est.: 1 sec

Amount of past estimations used by the estimator (bandwidth):

B: 1200 steps

Amount data stored to start estimation:

ST: 0 steps

Number of times "I" is stored for each update of "L":

IL: 1 steps

Markers on how many planes? (1 or 2) 2

To see more information in the window, scroll down with the mouse cursor over the scrollbar

If the background is too dark and the figures' annotations are hard to see, we recommend turning off system dark mode.

Figure 9: GUI of the Motion Simulator written in Python

## Simulator Functionality

The simulator gathers all the parameters set by the user, adjust the position of the SAPs, calculates the Moment of Inertia (*I*), the center of gravity (*Cg1*), and generates an array containing all the points of interest on the target (vertices and markers), named *cube* in the scripts. We will refer to the variable *cube* as ***C*** in this guide. We chose to base the simulator on quaternions, so the three-dimensional position of the points of interest are represented as [x, y, z, 1]. The **MS** function sends the important parameters to the **pnp** function which starts the simulation.

The initial quaternion used is obtained by:

$$\{\widehat{\boldsymbol{q}}\} = \begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix} = \begin{Bmatrix} \sin\left(\frac{\varphi_3}{2}\right)\cos\left(\frac{\varphi_2}{2}\right)\cos\left(\frac{\varphi_1}{2}\right) - \cos\left(\frac{\varphi_3}{2}\right)\sin\left(\frac{\varphi_2}{2}\right)\sin\left(\frac{\varphi_1}{2}\right) \\ \cos\left(\frac{\varphi_3}{2}\right)\sin\left(\frac{\varphi_2}{2}\right)\cos\left(\frac{\varphi_1}{2}\right) + \sin\left(\frac{\varphi_3}{2}\right)\cos\left(\frac{\varphi_2}{2}\right)\sin\left(\frac{\varphi_1}{2}\right) \\ \cos\left(\frac{\varphi_3}{2}\right)\cos\left(\frac{\varphi_2}{2}\right)\sin\left(\frac{\varphi_1}{2}\right) - \sin\left(\frac{\varphi_3}{2}\right)\sin\left(\frac{\varphi_2}{2}\right)\cos\left(\frac{\varphi_1}{2}\right) \\ \cos\left(\frac{\varphi_3}{2}\right)\cos\left(\frac{\varphi_2}{2}\right)\cos\left(\frac{\varphi_1}{2}\right) + \sin\left(\frac{\varphi_3}{2}\right)\sin\left(\frac{\varphi_2}{2}\right)\sin\left(\frac{\varphi_1}{2}\right) \end{Bmatrix}$$

where $\boldsymbol{\varphi}$ is the vector containing the Euler angles described in the variables section. The conversion from the quaternion to the direction cosine matrix is given by:

$$[\boldsymbol{Q}]_{Xx} = \begin{bmatrix} {q_1}^2 - {q_2}^2 - {q_3}^2 + {q_4}^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -{q_1}^2 + {q_2}^2 - {q_3}^2 + {q_4}^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -{q_1}^2 - {q_2}^2 + {q_3}^2 + {q_4}^2 \end{bmatrix}$$

The derivation of the angular velocity $\omega$ can be obtained through Euler's Equation in case no torque is being applied:

$$\dot{\boldsymbol{\omega}} = -\boldsymbol{I}^{-\boldsymbol{1}}\boldsymbol{\omega} \times (\boldsymbol{I\omega})$$

and the time derivation of the quaternion $\widehat{\boldsymbol{q}}$ is given by:

$$\frac{d}{dx}\{\widehat{\boldsymbol{q}}\} = \frac{1}{2}[\boldsymbol{\Omega}]\{\widehat{\boldsymbol{q}}\}$$

$$[\boldsymbol{\Omega}] = \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix}$$

where $\boldsymbol{\omega}$ is the angular velocity vector, whose initial conditions are described in the variables section.

We use a simple 4th order Runge-Kutta integration to calculate $\boldsymbol{\omega}_{t_0+\Delta t}$ and $\widehat{\boldsymbol{q}}_{t_0+\Delta t}$ from $\boldsymbol{\omega}_{t_0}$ and $\widehat{\boldsymbol{q}}_{t_0}$, knowing the moment of inertia $\boldsymbol{I}$. For the angular velocity, $\boldsymbol{\omega}$:

$$
\begin{aligned}
\boldsymbol{k}_1 &= -\boldsymbol{I}^{-1}\boldsymbol{\omega}_{t_0} \times \left(\boldsymbol{I}\boldsymbol{\omega}_{t_0}\right) \\
\boldsymbol{k}_2 &= -\boldsymbol{I}^{-1}\left(\boldsymbol{\omega}_{t_0} + \boldsymbol{k}_1\frac{\Delta t}{2}\right) \times \left(\boldsymbol{I}\boldsymbol{\omega}_{t_0} + \boldsymbol{k}_1\frac{\Delta t}{2}\right) \\
\boldsymbol{k}_3 &= -\boldsymbol{I}^{-1}\left(\boldsymbol{\omega}_{t_0} + \boldsymbol{k}_2\frac{\Delta t}{2}\right) \times \left(\boldsymbol{I}\boldsymbol{\omega}_{t_0} + \boldsymbol{k}_2\frac{\Delta t}{2}\right) \\
\boldsymbol{k}_4 &= -\boldsymbol{I}^{-1}\left(\boldsymbol{\omega}_{t_0} + \boldsymbol{k}_3\Delta t\right) \times \left(\boldsymbol{I}\boldsymbol{\omega}_{t_0} + \boldsymbol{k}_3\Delta t\right)
\end{aligned}
$$

$$
\boldsymbol{\omega}_{t_0+\Delta t} = \boldsymbol{\omega}_{t_0} + \left(\frac{\boldsymbol{k}_1 + 2\,\boldsymbol{k}_2 + 2\,\boldsymbol{k}_3 + \boldsymbol{k}_4}{6}\right)\Delta t
$$

And for the quaternion $\widehat{\boldsymbol{q}}$:

$$
\begin{aligned}
\boldsymbol{k}_1 &= \frac{1}{2}\,\boldsymbol{\Omega}_{t_0}\,\widehat{\boldsymbol{q}}_{t_0} \\
\boldsymbol{k}_2 &= \frac{1}{2}\,\boldsymbol{\Omega}_{t_0}\,\widehat{\boldsymbol{q}}_{t_0} + \frac{\boldsymbol{k}_1}{2}\Delta t \\
\boldsymbol{k}_3 &= \frac{1}{2}\,\boldsymbol{\Omega}_{t_0}\,\widehat{\boldsymbol{q}}_{t_0} + \frac{\boldsymbol{k}_2}{2}\Delta t \\
\boldsymbol{k}_4 &= \frac{1}{2}\,\boldsymbol{\Omega}_{t_0}\,\widehat{\boldsymbol{q}}_{t_0} + \boldsymbol{k}_3\Delta t
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{q}_{t_0+\Delta t} &= \widehat{\boldsymbol{q}}_{t_0} + \left(\frac{\boldsymbol{k}_1 + 2\,\boldsymbol{k}_2 + 2\,\boldsymbol{k}_3 + \boldsymbol{k}_4}{6}\right)\Delta t \\
\widehat{\boldsymbol{q}}_{t_0+\Delta t} &= \frac{\boldsymbol{q}_{t_0+\Delta t}}{\left\|\boldsymbol{q}_{t_0+\Delta t}\right\|}
\end{aligned}
$$

Using the resulting quaternion, we can calculate the direction cosine matrix and from it, the position of the points of interest $\boldsymbol{C}_{t_0+\Delta t}$ following the relation:

$$
\boldsymbol{C}_{t_0+\Delta t} = \boldsymbol{R}\,\boldsymbol{C}_{t_0}
$$

$$\boldsymbol{R} = \begin{bmatrix} [Q]_{Xx_{11}} & [Q]_{Xx_{12}} & [Q]_{Xx_{13}} & P_{t_0+\Delta t_1} \\ [Q]_{Xx_{21}} & [Q]_{Xx_{22}} & [Q]_{Xx_{23}} & P_{t_0+\Delta t_2} \\ [Q]_{Xx_{31}} & [Q]_{Xx_{32}} & [Q]_{Xx_{33}} & P_{t_0+\Delta t_3} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{P}_{t_0+\Delta t_1} = \boldsymbol{P}_{t_0} + \boldsymbol{V}\Delta t$$

where $\boldsymbol{P}$ is the translation between observer and target.

The camera should have a 90º field of view (on the middle of the edge of the frame, which is **f** x **f** pixels, meaning the field of view is larger than 90º).

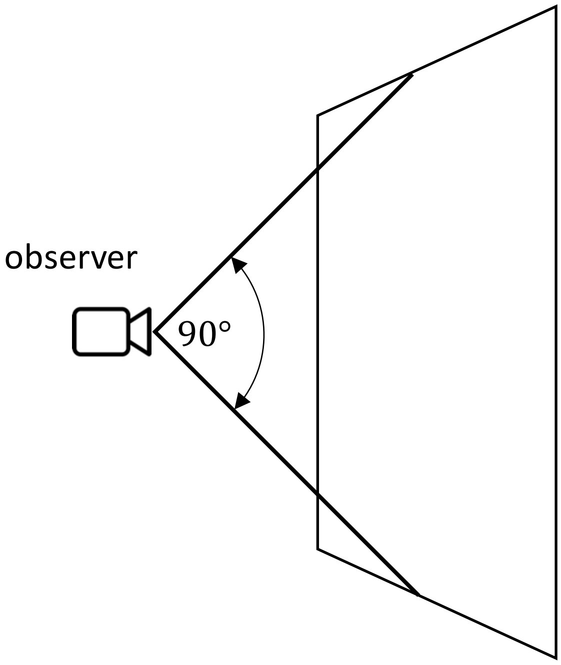


Figure 10: Field of View

The transformations of the markers' positions found in $\boldsymbol{P}$ to projections on the image $\boldsymbol{Z}$ were possible using the following equations:

$$\boldsymbol{Y} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{P}$$

$$\boldsymbol{Z} = \begin{bmatrix} {}^{Y_{1,1}}\!/\!_{Y_{3,1}} & {}^{Y_{1,2}}\!/\!_{Y_{3,2}} & \cdots & {}^{Y_{1,n}}\!/\!_{Y_{3,n}} \\ {}^{Y_{2,1}}\!/\!_{Y_{3,1}} & {}^{Y_{2,2}}\!/\!_{Y_{3,2}} & \cdots & {}^{Y_{2,n}}\!/\!_{Y_{2,n}} \end{bmatrix}$$

There will be moments when the markers will be hidden by the structure of the spacecraft, we assumed that the main body is a cuboid, and the solar panels don't interfere in the visibility of the markers. In fact, with the configuration proposed in the simulator, the solar panels are never interfering in the visibility of the markers.

To help us knowing when the markers are visible or not, two auxiliary vectors are in the $\boldsymbol{C}$ array. These auxiliary vectors are normal to the surfaces containing markers. If the angle between the line from the observer to a marker and the auxiliary vector correspondent to the surface containing that marker is 90º or smaller, the markers on that surface are invisible.

In the Simulators provided, we change the value of Marker 1 projection **X** component to 2000 if any of the markers is invisible. If any of the markers' projections has a component larger than 1000, the Estimator knows one of the markers is not visible and starts a hold state, resuming the estimations when all the markers are again visible.

# References

Curtis, H.D. (2014). *Orbital Mechanics for Engineering Students*. [online] Elsevier. Available at: https://doi.org/10.1016/C2011-0-69685-1 [Accessed 16 Oct. 2019].