Efficient Solution for Thrust Distribution of Simultaneous Forces and Torques

Mar Cols-Margenet * and Yasunori Tsumura †
and Keita Ogo ‡

Nomenclature

GNC = Guidance, Navigation & Controls
FSW = Flight Software
CG = Center of Gravity

I Introduction

Solving the thrust-distribution problem consists in finding a set of non-negative thruster forces that yields the requested set of three-dimensional control forces and torques, while using the minimum fuel possible. This is a convex-optimization problem with constraints and, as such, solving it rigorously generally requires a linear or quadratic programming solver (depending on whether the L_1 or L_2 norm of the thrust vector is minimized). In spacecraft systems, the solution for the thrust-distribution problem is necessary in the interface between GNC (Guidance, Navigation & Controls) and the thruster actuators. Generally speaking, such solution can be either computed online (i.e., autonomously onboard) or offline (i.e., on the ground beforehand).

A classic offline approach is to perform an a priori optimization by decomposing the space of forces and moments into signed canonical directions and then finding the minimum-thrust solution along these directions. This offline process results in fixed torque-and-force mapping matrices that are then included as tables of constants in the onboard FSW (flight software) implementation. Although such approach provides the mapping capability that is necessary onboard to produce any combination of requested control forces and torques, the solution is far from optimal and more thrust than necessary is created.

Online approaches tend to be more fuel efficient than offline ones but there is a caveat: the algorithm execution must fit within the allocated processing time, which is limited by the capabilities of the onboard flight processor. Therefore, fully solving the constrained convex-optimization problem online might not be feasible with a conventional flight processor. In addition, there is the challenge of implementing such a complex solver and verifying it.

In the light of the aforementioned limitation, this paper presents a novel thrust-distribution method that is considerably more efficient than conventional offline approaches yet suitable for onboard applications –as it does not fully solve the constrained convex-optimization problem and therefore is simpler and faster. The proposed strategy is based on the minimum-norm inverse (to solve for the requested forces and torques) and the null-space of the thruster-configuration matrix (to enforce that all thruster forces are non-negative). Such approach is simple, robust and computationally fast. Hence, presenting the right balance between computational cost and fuel efficiency, which makes it extremely compelling to use onboard spacecraft systems.

Saving as much fuel as possible during operations is a critical goal for all spacecraft missions and, for this reason, Astroscale Japan is looking into developing efficient fuel solutions for the current and upcoming active-debris-removal missions.

*Flight Software Engineer, Astroscale Japan, Tokyo 130-0013
†GNC Engineer, Astroscale Japan, Tokyo 130-0013
‡GNC Group Manager, Astroscale Japan, Tokyo 130-0013

The paper is outlined as follows: First, Section II describes mathematically the problem to be solved (i.e. the thrust-distribution problem) and frames it within the context of the onboard FSW. The upcoming three sections present the math behind the three different algorithms that are being considered to solve the problem:

1. Section III describes the computation of the constant thrust matrices or tables (offline optimization), which can then readily be used onboard.

2. Section IV presents the formulation of the convex optimization problem with constraints, which is to be fully solved online using a linear programming solver.

3. Section V describes the novel online method that is based on the minimum-norm solution with the addition of a null-space offset.

Next, Section VI presents numerical simulations that are used to compare the performance of the algorithms above. These numerical simulations are done in a unit-test fashion by fixing the inputs and observing the outputs. In turn, Section VII provides insight on how different thrust-distribution algorithms affect the overall ΔV usage in mission-like scenarios that involve closed-loop dynamics simulation. Finally, Section VIII provides a summary of results and conclusions.

II Problem Statement

The thrust-distribution problem consists in finding a vector of non-negative thrust levels $\boldsymbol{T}$ that satisfies the requested set of forces and torques $\boldsymbol{Y}_{\text{req}}$:

$$[A]\boldsymbol{T} = \boldsymbol{Y}_{\text{req}} \tag{1a}$$

$$\boldsymbol{T} \geq 0 \tag{1b}$$

where $\boldsymbol{Y}_{\text{req}}$ is defined as

$$\boldsymbol{Y}_{\text{req}} = \begin{bmatrix} F_x & F_y, & F_z, & M_x & M_y, & M_z \end{bmatrix} \tag{2}$$

and $\boldsymbol{T} \in \mathbb{R}^{N\times 1}$ is the set of individual thrust levels we are looking for:

$$\boldsymbol{T} = \begin{bmatrix} T_1, & \ldots, & T_N \end{bmatrix} \tag{3}$$

with N being the number of thrusters available.

The matrix $[A]$ is a function of the CG (center of gravity) as well as the orientation and position of the individual thrusters. Before presenting the expression of $[A]$, it is convenient to describe a few variables first. Let $\hat{\boldsymbol{g}}_i \in \mathbb{R}^{3\times 1}$ be the firing direction of the i^{th} thruster, such that the force vector induced on the spacecraft by this individual thruster is given by

$$\boldsymbol{F}_i = F_i \hat{\boldsymbol{g}}_i \tag{4}$$

This force is illustrated in Fig. 1

Let $\boldsymbol{r}_i \in \mathbb{R}^{3\times 1}$ be the position of the i^{th} thruster. The torque induced on the spacecraft by this thruster is then given by:

$$\tau_i = (\boldsymbol{r}_i - \boldsymbol{r}_{CG}) \times F_i \hat{\boldsymbol{g}}_i = \boldsymbol{d}_i F_i \tag{5}$$

The $[A]$ matrix, which maps the individual thruster forces T_i into forces and torques is given by $\hat{\boldsymbol{g}}_i \in \mathbb{R}^{3\times 1}$ and

$\boldsymbol{d}_i \in \mathbb{R}^{3\times 1}$ as follows:

$$[A] = \begin{bmatrix} \hat{\boldsymbol{g}}_1 & \dots & \hat{\boldsymbol{g}}_N \\ \boldsymbol{d}_1 & \dots & \boldsymbol{d}_N \end{bmatrix} \in \mathbb{R}^{6\times N} \tag{6}$$

Note that the size of the matrix $[A]_{6\times N}$ is not fixed but will change depending on the number of thrusters N that are available. In addition, the values in $[A]$ will also change if the CG location changes.

Now that the thrust-distribution problem has been mathematically described, it is interesting to frame it in the context of the onboard FSW. The solution for the thrust-distribution problem outlined in Eq (1a) is necessary in the interface between GNC and the thruster actuators. At every time step, the onboard controller computes the required set of forces and torques $\boldsymbol{Y}_{\text{req}}$ that needs to be applied onto the spacecraft. The requested forces and torques are then mapped to a vector individual thruster levels $\boldsymbol{T}$. The present paper focuses precisely on the algorithm that, at every time step and given the parameter matrix $[A]$, performs the mapping from $\boldsymbol{Y}_{\text{req}}$ to $\boldsymbol{T}$ efficiently and in real time. From now onwards, this algorithm will be referred to as the thrust-distribution algorithm and, ideally, it should present the following features:

- The computed thrust levels $\boldsymbol{T}$ shall be such that Eq (1a) is satisfied with precision. This means that the generation of undesired forces and torques, which must be absorbed elsewhere (e.g., reaction wheels), should be minimized.
- The individual thrust levels T_i shall be non-negative
- The linear (L_1) or quadratic (L_2) norm of the computed thrust level shall be minimized:

$$T_{L_1} = ||\boldsymbol{T}||_{L_1} = T_1 + T_2 + \ldots + T_N \tag{7a}$$

$$T_{L_2} = ||\boldsymbol{T}||_{L_2} = \sqrt{T_1^2 + T_2^2 + \ldots + T_N^2} \tag{7b}$$

- The algorithm shall execute fast enough to fit within the allocated processing time

With this in mind, the upcoming sections will present three different algorithms that can be considered as candidates to solve the thrust-distribution problem:

1. Offline optimization and use of constant thruster matrices
2. Convex optimization solver
3. Minimum-norm solution with null space

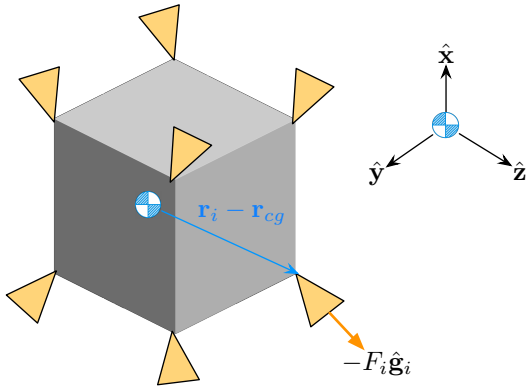

Figure 1. Position and firing direction of the i^{th} thruster

III Offline optimization: Constant thrust matrices or tables

The first approach consist in computing constant thrust tables $[C_{\text{pos}}], [C_{\text{neg}}] \in \mathbb{R}^{N\times 6}$ offline (i.e., on ground). These constant tables or matrices will be used onboard to compute the thrust level $\boldsymbol{T}$ ($= \boldsymbol{T}_{\text{mat}}$) vector at every time step according to the requested force-and-torque vector $\boldsymbol{Y}_{\text{req}}$. The table $[C_{\text{pos}}]$ is used for positive force/torque components while the table $[C_{\text{neg}}]$ is used for negative ones.

These constant tables are first computed on by decomposing the space of forces and torques ($\boldsymbol{Y}$) into signed canonical directions. Since the forces and torques are three-dimensional (x, y, z) and can have either positive or negative sign, there are a total of 12 canonical directions:

$$\hat{\boldsymbol{e}}_{f_x,p} = [1,0,0,0,0,0] \tag{8a}$$
$$\hat{\boldsymbol{e}}_{f_x,n} = [-1,0,0,0,0,0] \tag{8b}$$
$$\dots \tag{8c}$$
$$\hat{\boldsymbol{e}}_{m_z,p} = [0,0,0,0,0,1] \tag{8d}$$
$$\hat{\boldsymbol{e}}_{m_z,n} = [0,0,0,0,0,-1] \tag{8e}$$
$$\tag{8f}$$

Let $\hat{\boldsymbol{e}}$ be one of the basis vectors above. Then, we solve for the minimum $\boldsymbol{T}_e$ solution to the following equation and constraint:

$$[A]\boldsymbol{T}_e = \hat{\boldsymbol{e}} \tag{9a}$$
$$\boldsymbol{T}_e \geq 0 \tag{9b}$$

Note that the problem in Eq (9b) is also a convex-optimization problem and, since this is not solved onboard (and therefore not limited by onboard processing and resources constraints), a regular convex-optimization solver can be used. More details on this kind of problems and solvers are described in Section IV.

The process in Eq (9b) is repeated for each of the basis vectors resulting in a total of twelve $\boldsymbol{T}_e \in \mathbb{R}^{N\times 1}$ vectors, that are then used to build the positive and negative thruster tables $[C_{\text{pos}}]$ and $[C_{\text{neg}}]$ as follows:

$$[C_{\text{pos}}] = \begin{bmatrix} \boldsymbol{T}_{f_x,p} & \boldsymbol{T}_{f_y,p} & \boldsymbol{T}_{f_z,p} & \boldsymbol{T}_{m_x,p} & \boldsymbol{T}_{m_y,p} & \boldsymbol{T}_{m_z,p} \end{bmatrix} \in \mathbb{R}^{N\times 6} \tag{10a}$$
$$[C_{\text{neg}}] = \begin{bmatrix} \boldsymbol{T}_{f_x,n} & \boldsymbol{T}_{f_y,n} & \boldsymbol{T}_{f_z,n} & \boldsymbol{T}_{m_x,n} & \boldsymbol{T}_{m_y,n} & \boldsymbol{T}_{m_z,n} \end{bmatrix} \in \mathbb{R}^{N\times 6} \tag{10b}$$

Onboard FSW, these constant tables are used as follows:

$$\boldsymbol{T}_{\text{mat}} = [C_{\text{pos}}]\boldsymbol{Y}_{\text{req}} \qquad \text{for } \boldsymbol{Y}_{\text{req}}[i] \geq 0 \tag{11a}$$
$$\boldsymbol{T}_{\text{mat}} = [C_{\text{neg}}]\boldsymbol{Y}_{\text{req}} \qquad \text{for } \boldsymbol{Y}_{\text{req}}[i] < 0 \tag{11b}$$

Where the subindex in $\boldsymbol{T}_{\text{mat}}$ indicates that the $\boldsymbol{T}$ vector has been computed using the constant thrust matrices, and $\boldsymbol{Y}_{\text{req}}[i]$ corresponds to the individual components of $\boldsymbol{Y}_{\text{req}} = \begin{bmatrix} F_x & F_y, & F_z, & M_x & M_y, & M_z \end{bmatrix}$.

The beauty of this approach is that the computational load onboard FSW is very low, as the constant thruster matrices can be readily used without further computations. In addition, the solution obtained is, in principle, exact.

This can be checked through numerical simulation by performing the following check:

$$\boldsymbol{Y}_{\text{cmd}} \approx \boldsymbol{Y}_{\text{req}} \tag{12}$$

where

$$\boldsymbol{Y}_{\text{cmd}} = [A]\boldsymbol{T}_{\text{mat}} \tag{13}$$

The idea behind this offline-optimization approach is the following: the optimal solution to the full thrust-distribution problem is equivalent to summing the optimal solutions to canonical thrust-distribution problems. However, as it will be shown in the numerical simulations, this assumption does not hold true for the constrained problem defined in Eq (1) and the solution $\boldsymbol{T}$ provided by these constant thruster matrices ends up using more thrust than what is actually necessary.

$$\boldsymbol{T}_{\text{mat}} \gg \boldsymbol{T}_{\text{opt}} \tag{14}$$

The fuel efficiency could be slightly improved by allowing some amount of undesired thrust in certain directions. For instance, it could be acceptable to create more torque than requested and allow some difference in Eq. (12) provided that: 1) the undesired torque can be absorbed by other actuators such as reaction wheels and 2) this tradeoff means saving fuel. Note that such small optimizations and tradeoffs need to be performed at the time of solving Eq (9b) for each canonical direction.

Overall, the offline approach with the use of the constant thrust matrices/tables will always face limitations, as the thrust-distribution problem is being solved a priori for any generic set of requested forces and torques –rather than for the specific request at the current time step.

IV Online approach: Convex optimization solver

By looking at the definition of the thrust-distribution problem outlined in Eq (1), it becomes apparent that this is a convex optimization problem with constraints. In general terms, such problem is formulated as follows:

The goal is to minimize

$$\frac{1}{2}\boldsymbol{x}^T[P]\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} \tag{15}$$

subject to

$$[A]\boldsymbol{x} = \boldsymbol{b} \tag{16a}$$

$$[G]\boldsymbol{x} \leq \boldsymbol{h} \tag{16b}$$

In the thrust-distribution problem, the terms defined above take the following nomenclature and values:

$$\boldsymbol{x} = \boldsymbol{T} \in \mathbb{R}^{N\times 1} \rightarrow \text{ thrust-level vector we are solving for} \tag{17a}$$

$$\boldsymbol{q} = \texttt{ones} \in \mathbb{R}^{N\times 1} \rightarrow \text{ linear term in cost function} \tag{17b}$$

$$[P] = \texttt{zeroes} \in \mathbb{R}^{N\times N} \rightarrow \text{ quadratic term in cost function} \tag{17c}$$

$$\boldsymbol{b} = \boldsymbol{Y}_{\text{req}} \in \mathbb{R}^{N\times 1} \rightarrow \text{ vector of requested forces and torques} \tag{17d}$$

$$\boldsymbol{h} = \texttt{zeroes} \in \mathbb{R}^{N\times 1} \rightarrow \text{ lower bound vector} \tag{17e}$$

$$[A] = [A] \in \mathbb{R}^{N\times N} \rightarrow \text{ thruster-configuration matrix} \tag{17f}$$

$$[G] = -1 * \texttt{ones} \in \mathbb{R}^{N\times N} \rightarrow \text{ lower bound matrix} \tag{17g}$$

This implies solving for Eq (1) while minimizing the norm of the thrust level. By setting $[P]$ as zeroes and $\boldsymbol{q}$ as ones, we are using a linear programming solver instead of a quadratic one. A linear solver is more suitable because the total thrust (fuel) used, is computed by simple addition of the individual thrusts. Therefore, the L_1 norm of the thrust-level vectors is the most relevant. Regarding the solver itself, the reader should refer, for example, to the Python implementation and associated documentation of the `cvxopt`[1] package.

Solving the thrust-distribution problem online with an onboard linear programming solver provides the most fuel-efficient solution (as the true minimum is found upon convergence). However, this involves an iterative process that takes considerable computational time to converge. Therefore, solving this iterative problem onboard and in real-time on a radiation-hardened processor is not a very feasible option.

V Online approach: Minimum-norm solution with null space

The fuel inefficiency of the constant thrust matrices and the computational cost of an online convex optimization solver led the way to explore other solutions with a better tradeoff of features. An improved tradeoff is precisely what the minimum-norm solution with null space offers. The proposed method is a novel approach that, mathematically, can be divided in two steps:

1. Minimum-norm solution: this first step is taken in order to obtain a minimum $\boldsymbol{T}$ without imposing the non-negativity requirement.
2. Null-space offset: the second step consist in computing an offset vector $\boldsymbol{T}_{\text{offset}}$ that, when added to $\boldsymbol{T}$, guarantees that all the individual thrust levels are non-negative and yet does not modify the resulting net forces and torques.

Each of these steps is thoroughly described in the upcoming subsections.

V.A First step: minimum norm

Firstly, the minimum-norm solution to Eq (1a) is found as follows:

$$\boldsymbol{T} = [A]^T([A][A]^T)^{-1}\boldsymbol{Y}_{\text{req}} \tag{18}$$

This solution guarantees that the commanded force and torque $\boldsymbol{Y}_{\text{cmd}}$ computed as

$$\boldsymbol{Y}_{\text{cmd}} = [A]\boldsymbol{T} \tag{19}$$

is an exact solution such that

$$\boldsymbol{Y}_{\text{cmd}} = \boldsymbol{Y}_{\text{req}} \tag{20}$$

However, Eq (18) does not guarantee that all the elements in $\boldsymbol{T}$ are non negative. A simple way to make the negative elements equal or larger than zero would to add a certain offset to them.

$$\boldsymbol{T}_{\text{min-offset}} = \boldsymbol{T} + \boldsymbol{T}_{\text{offset}} \tag{21}$$

This, however, needs to be performed with the following considerations:

1. The L_1 norm of $\boldsymbol{T}_{\text{new}}$ shall be as small as possible
2. The resulting forces and torques shall remain exact:

$$\boldsymbol{Y}_{\text{cmd}} = \boldsymbol{Y}_{\text{min-offset}} \qquad \rightarrow \qquad [A]\boldsymbol{T} = [A]\boldsymbol{T}_{\text{min-offset}} \tag{22}$$

[1] https://cvxopt.org

How to compute $\boldsymbol{T}_{\text{offset}}$ for any generic thruster configuration while satisfying the conditions above is demonstrated in the next step.

V.B Second step: null-space offset

The goal of the second step consists oinn finding the $\boldsymbol{T}_{\text{offset}}$ term in Eq (21) such that all elements in $\boldsymbol{T}_{\text{min-offset}}$ are non negative. The approach to take for this computation varies depending on the particular thruster configuration of the spacecraft. We will consider two possible types of thruster configurations: symmetric and non-symmetric ones. Symmetric thruster configurations are such that when all thrusters fire the same amount of thrust, zero-net torque and force is produced on the spacecraft. The contrary would then be considered a non-symmetric thruster configuration. Each of these configuration types is illustrated in Fig. 2 and Fig. 3 respectively. The figures can be interpreted as follows: the blue dots correspond to the thruster locations on the surface of an imaginary geometric cube (the spacecraft), while the blue arrows correspond to the directions of the individual forces that each thruster creates on the spacecraft (i.e., $-\hat{\boldsymbol{g}}_i$ directions). The orange dot in the center represents the location of the spacecraft center of mass (CoM or CG) of the spacecraft.

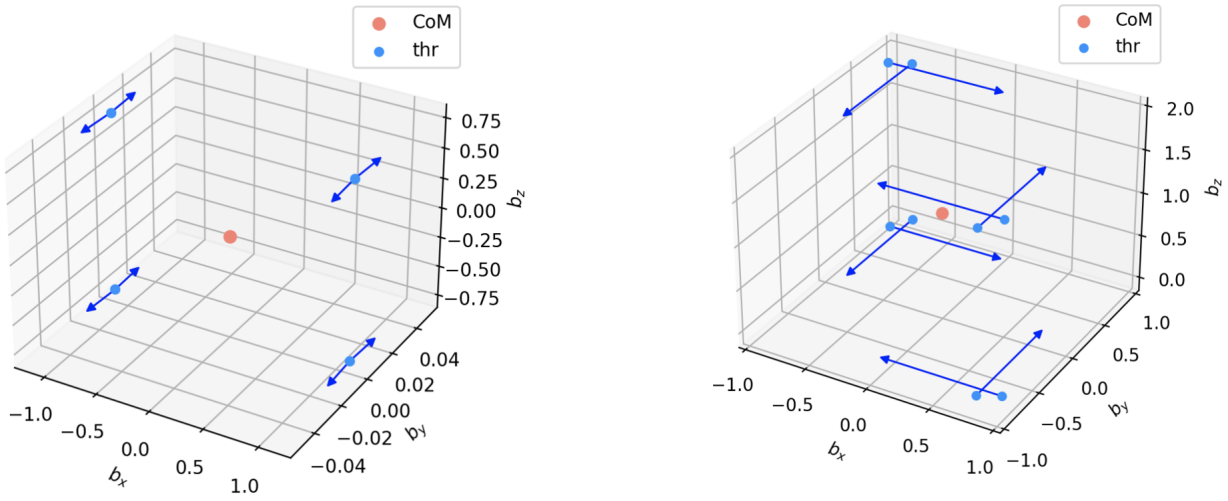

Figure 2. Example of symmetric thruster configurations

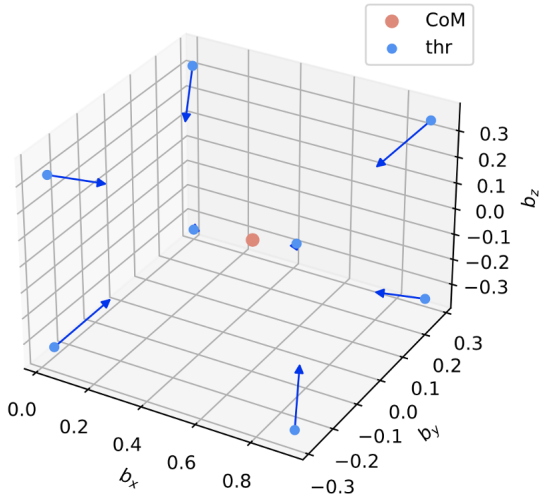

Figure 3. Example of symmetric thruster configurations

The distinction between symmetric and non-symmetric thruster configurations is very important at the time of computing $\boldsymbol{T}_{\text{offset}}$. Basically, in a symmetric configuration you can make all elements of $\boldsymbol{T}$ positive by simply adding the most negative value to all elements:

$$T_{\text{min}} = \texttt{min}(\boldsymbol{T}) \tag{23a}$$

$$\boldsymbol{T}_{\text{offset}} = T_{\text{min}} * \texttt{ones} \tag{23b}$$

$$\boldsymbol{T}_{\text{min-offset}} = \boldsymbol{T} + \boldsymbol{T}_{\text{offset}} \tag{23c}$$

where `ones` is a vector of the same size as $\boldsymbol{T}$ with the value of 1 in all its elements. For symmetric configurations Eq (23) guarantees that all the elements in $\boldsymbol{T}_{\text{new}}$ are non-negative while satisfying Eq (22) and therefore providing an exact solution for the requested forces and torques $\boldsymbol{Y}_{\text{req}}$.

This method is fairly simple and intuitive but it only works for symmetric configurations, so the interesting question is how to generalize this strategy for generic non-symmetric configurations. At this point is where the concept of the null-space matrix comes into play. In particular, we are looking for the null-space projection of the thruster-configuration matrix $[A]$, which is defined as follows:

$$[N] = I - [A]^T([A][A]^T)^{-1}[A] \tag{24}$$

This $[N]$ matrix is called the null-space projection of $[A]$ because, by construction, it presents the following property:

$$[A][N] = [0] \tag{25}$$

which can be easily proven by substitution:

$$[A][N] = [A]\left(I - [A]^T([A][A]^T)^{-1}[A]\right) = [A] - ([A][A]^T)([A][A]^T)^{-1}[A] = [A] - [I][A] = [0] \tag{26}$$

As seen in Eq (24), in order to compute $[N]$ it is necessary that the matrix $M_{\text{inv}} = ([A][A]^T) \in \mathbb{R}_{N\times N}$ is full rank and therefore invertible. As a matter of fact, this matrix will always be full rank for non-symmetric thruster configurations but it will not be full rank for symmetric ones. Since the computation of the null-space matrix is only necessary for non-symmetric configurations, the existence of the inverse is not a problem. In addition, note that this inverse matrix has size $N \times N$ where N is the number of thrusters. This means that inverse matrix will exist for any number of thrusters as long as they are in a non-symmetric configuration.

Because $[A]$ is nominally a constant matrix (assuming all thruster are available and the CoM is not changing), so is $[N]$. Therefore both matrices can be computed offline and uploaded to the spacecraft. Having said that, these matrices are tied to the specific subset of thrusters at use, so if different subsets are to be used in different parts of the mission, the $[A]$ and $[N]$ matrices will have to be updated accordingly.

After having discussed the computation and existence of the null-space matrix $[N]$, let us explain how we can use it in the thrust-distribution problem. This matrix can be effectively used to add an additional offset to the vector T (in order to make the negative components non-negative) without affecting the net forces and torques $\boldsymbol{Y}_{\text{cmd}}$, as follows:

$$\boldsymbol{Y}_{\text{cmd, new}} = [A]\boldsymbol{T}_{\text{min-offset}} = [A](\boldsymbol{T} + [N]\boldsymbol{T}_{\text{offset}}) = A\boldsymbol{T} + [\boldsymbol{0}] \cdot \boldsymbol{T}_{\text{offset}} = [A]\boldsymbol{T} = \boldsymbol{Y}_{\text{cmd}} \tag{27}$$

The question that remains open is then how to compute the minimum $\boldsymbol{T}_{\text{offset}}$ such that $\boldsymbol{T}_{\text{new}} \geq \boldsymbol{0}$. This problem can be approached in several different ways –more rigorously or more practically– and our proposal is a practical one

that can be used efficiently onboard. Similarly to how it is done in Eq (23) for symmetric thruster configurations, the procedure is as follows:

$$T_{\text{min}} = \texttt{min}(\boldsymbol{T}) \tag{28a}$$

$$\boldsymbol{T}_{\text{offset}} = K * T_{\text{min}} * \texttt{ones} \tag{28b}$$

$$\boldsymbol{T}_{\text{new}} = \boldsymbol{T} + [N]\boldsymbol{T}_{\text{offset}} \tag{28c}$$

where K is a tuning gain meant to ensure that all components in $\boldsymbol{T}_{\text{new}}$ are non-negative. This gain can be determined a priori through Monte-Carlo simulation or, through a small iterative process, onboard.

The iterative process onboard would be as follows:

$$K = 1.0 \tag{29a}$$

$$\Delta K = 0.02 \tag{29b}$$

$$K_{\text{max}} = 1.1 \tag{29c}$$

$$\texttt{ng} = \texttt{True} \tag{29d}$$

$$\texttt{while } (K \leq K_{\text{max}} \texttt{ \& ng}): \tag{29e}$$

$$\quad \boldsymbol{T}_{\text{min-offset}} = \boldsymbol{T} + [N](K * T_{\text{min}} * \texttt{ones}) \tag{29f}$$

$$\quad \texttt{if min}(\boldsymbol{T}_{\text{min-offset}}) < 0: \tag{29g}$$

$$\quad\quad K = K + \Delta K \tag{29h}$$

$$\quad \texttt{else}: \tag{29i}$$

$$\quad\quad \texttt{ng = False} \tag{29j}$$

The results of simulations run at Astroscale have shown that K tends to converge within the range $[1.01, 1.10]$. And when it does not, it usually means that the given thruster configuration is not enough to provide the requested forces and torques. In such cases, regardless of how much K is increased, a feasible solution will not be found. This could happen, for instance, when the number of thrusters available is drastically reduced due to an anomaly. Even if the $[N]$ matrix exists, the remaining thrusters might not be enough to compensate for the requested forces and torques. In such case, K will not converge –in the same way that a convex optimization solver would not converge.

Since there is no mathematical proof for the choice of K, it is treated as a control gain whose value (or range) should be determined through numerical simulation. Onboard FSW, it is in general not desired to include loops like the one in Eq (29e). Therefore, a proposal would be to execute the loop only in the numerical simulations and then fix it accordingly onboard.

VI Simulations

The numerical simulations are performed using a set of $N = 12$ thrusters (8 corner and 4 axial) in the configuration illustrated in Fig. 4. The three candidate onboard algorithms described in Section III, Section IV and Section V respectively are compared and contrasted with each other. Two different numerical simulations have been conducted in a unit-test fashion:

1. Comparison of constant thrust matrices vs. convex optimization solver
2. Comparison of minimum norm and null space vs. convex optimization solver

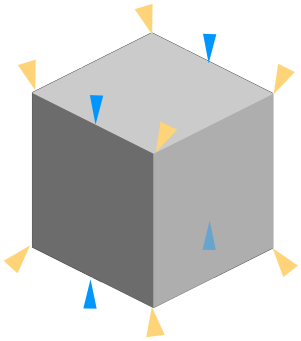

Figure 4. Configuration with 12 thrusters: 8 corner and 4 axial

VI.A Open-Loop Test: Constant Thrust Matrices vs. Convex Optimization Solver

The following simulation provides an idea of how much fuel is used when solving the thrust-distribution problem with the constant thrust matrices vs. a real-time convex optimization solver. This simulation is performed in a unit-test-like fashion as follows:

1. First, a mesh grid of forces and torques $\boldsymbol{Y}_{\text{req}}$ is created randomly with the following sampling limits:

$$F_{\text{lim}} = +/-0.067N \tag{30a}$$

$$M_{\text{lim}} = +/-0.005Nm \tag{30b}$$

   In total, around 60,000 different vectors $\boldsymbol{Y}_{\text{req}} = [F_x, F_y, F_z, M_x, M_y, M_x]$ are pre-computed with components randomly sampled within the limits above. The generated $\boldsymbol{Y}_{\text{req}}$ vectors will be given as input to each of the thrust-distribution algorithms being compared.

2. Next, the constant thrust matrices $[C_{\text{pos}}], [C_{\text{neg}}] \in \mathbb{R}^{N\times 6}$ defined in Eq (10) are pre-computed. This is done by solving Eq (9b) for each of the twelve canonical directions $\hat{\boldsymbol{e}}_i$ vectors defined in Eq (8). These equations can be solved a priori by a convex-optimization solver like the python package `cxvopt`.

3. Finally, the two onboard candidate algorithms (constant thrust matrices and online convex optimization solver) are tested:

   - **Constant thrust matrices:** with the pre-computed thruster matrices $[C_{\text{pos}}], [C_{\text{neg}}]$, the thrust-level vector $\boldsymbol{T}$ is obtained as in Eq (11) for each $\boldsymbol{Y}_{\text{req}}$ vector. The total thrust $||\boldsymbol{T}_{\text{mat}}||_{L_1} = T_{\text{mat}}$ is then obtained by taking the L_1 norm of thrust-level vector. Since approximately 60,000 different $\boldsymbol{Y}_{\text{req}}$ inputs are generated, the same number of output values of T_{mat} is obtained. These values are later used for performance comparison.
   - **Online convex optimization solver:** in this case, the Python package `cvxopt` is used to solve the thrust-distribution problem for each of the requested $\boldsymbol{Y}_{\text{req}}$ vectors. As before, the total thrust $||\boldsymbol{T}_{\text{opt}}||_{L_1} = T_{\text{opt}}$ is then obtained by taking the L_1 norm of thrust-level vector.

The simulation results are illustrated in Fig 5. This histogram shows which percentage of the optimal solution the constant thrust-matrices method uses, and how many times this happens among the 60,000 simulation cases. On average, the constant thrust-matrices approach is 92.27% more (L_1-norm) fuel intensive than the online convex optimization solver. In some rare cases, the thrust-matrices approach can use up to 300% of the optimal thrust solution. As expected, there was not any case in which the thrust matrices provided a more fuel-efficient result than the optimization solver (which would translate into a negative percentage in Fig 5).

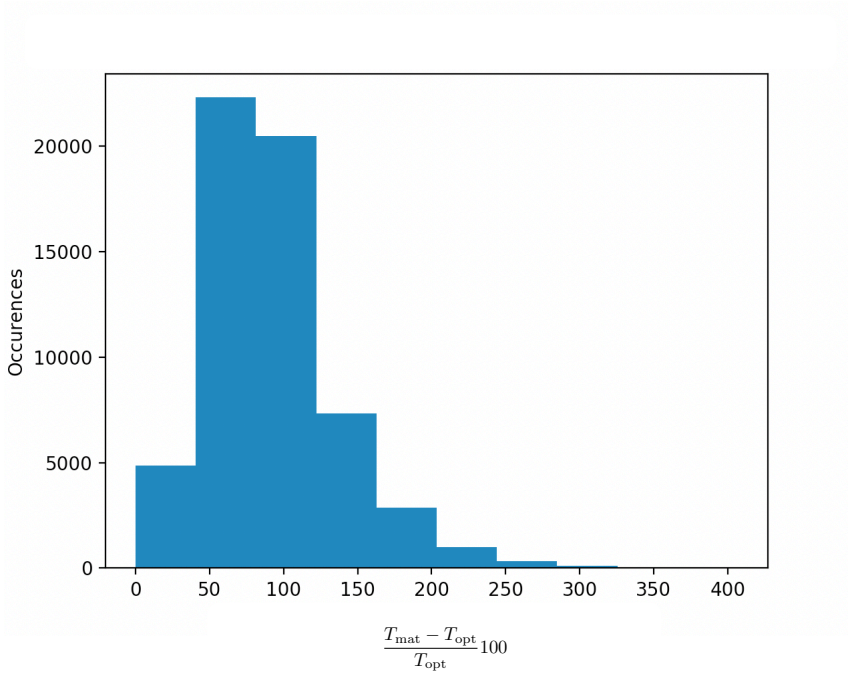

Figure 5. Histogram of relative difference in L_1 norms between T_{mat} and T_{opt}. On average: $T_{\text{mat}} = 1,92 \cdot T_{\text{opt}}$

In this simulation, both algorithms use the same `cvxopt` solver to resolve the thrust-distribution problem and they provide exact solutions for $\boldsymbol{Y}_{\text{req}}$. The difference is that one performs the optimization a priori (i.e., offline) and looks for a generic solution (i.e., the constant thrust matrices) that can be applied to any request of forces on torques; while the other finds the truly optimal solution for each and every $\boldsymbol{Y}_{\text{req}}$, which of course requires considerable processing time. Further insight on the processing time for the online convex optimization solver is provided in the next simulation.

As future work it would be interesting to investigate whether there is any trend between the magnitude of the commands and the efficiency percentage of the thruster matrices.

VI.B Open-Loop Test: Minimum Norm and Null Space vs. Convex Optimization Solver

The previous simulation has compared the constant thrust matrices (offline process) with the truly optimal solution (online process). The next simulation compares the novel minimum-norm and null-space strategy (online process) with the truly optimal solution (also online process). For this purpose, a similar unit-test like simulation has been done by creating a set of 60,000 different vectors $\boldsymbol{Y}_{\text{req}}$ with components within the sampling limits outlined in Eq (30). Each algorithm has been provided these inputs and, for each case, the thrust-level vector has been computed. After taking the L_1 norm, the total thrust used is obtained.

Before comparing both algorithms in terms of thrust/fuel efficiency, it is important to provide further insight on the minimum-norm and null-space results, since this is a novel strategy whose validity is still to be proved. The first item to look at is the accuracy of the commanded forces and torques $\boldsymbol{Y}_{\text{min-offset}} = [A]\boldsymbol{T}_{\text{min-offset}}$, which ideally should agree with the requested $\boldsymbol{Y}_{\text{req}}$. As shown in Fig 6, the accuracy requirement is satisfied within numerical precision.

The next variable that is interesting to look at is the convergence of the gain K for each case. For every $\boldsymbol{Y}_{\text{req}}$ requested, the iterative process described in Eq (29) is performed. As shown in Fig 7, the gain converges to $K = 1.06$ for most values. It is important to note that the value of K will differ depending on the thruster configuration itself (as captured by the matrix $[A]$) and also on the limits of the expected forces and torques (as defined in Eq (30)). Having said that, in the authors' experience, values of K that fall outside the bounds $K \in [1.01, 1.10]$ will provide a

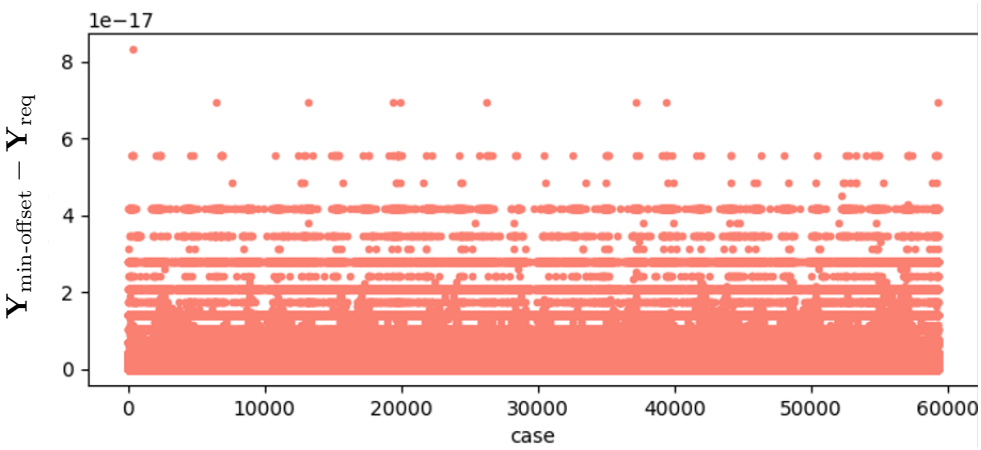

Figure 6. Accuracy of $\boldsymbol{Y}_{\text{min-offset}} = [A]\boldsymbol{T}_{\text{min-offset}}$ compared to the requested forces and torques $\boldsymbol{Y}_{\text{req}}$

solution that is too expensive in terms of fuel and should be discarded. As mentioned earlier, a value of K outside of the bounds above generally indicates that the requested forces and torques cannot be satisfied with the given thruster configuration.

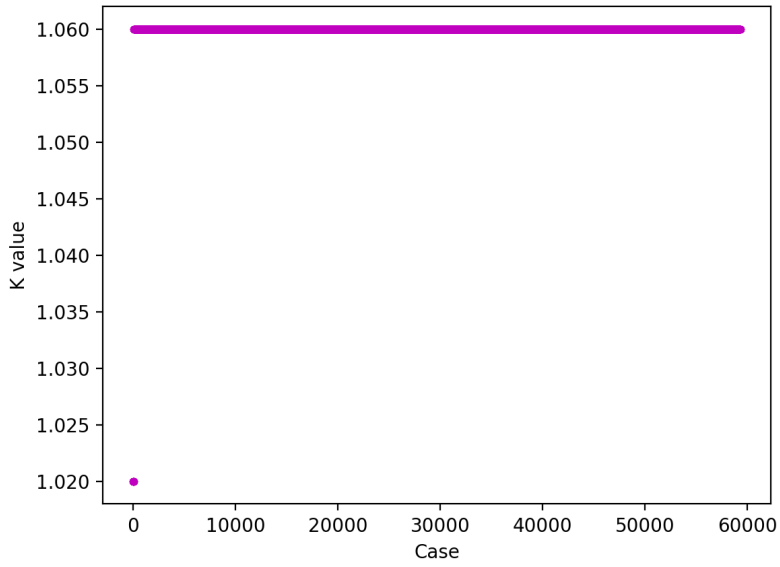

Figure 7. Convergence of K gain selection

After seeing that the minimum-norm and null-space algorithm works as expected, let us compare the fuel usage between this algorithm and the optimal solution. The total thrust computed by the minimum-norm and null-space algorithm is shown in Fig 8 while the total thrust computed by the convex-optimization solver is provided in Fig 9. As expected, the thrust (fuel) usage is higher with the minimum-norm and null-space algorithm. Having said, it is interesting to look at the histogram in Fig 10, which shows which percentage of the optimal solution the minimum-norm and null-space method uses, and how many times this happens among the 60,000 simulation cases. On average, the minimum-norm and null-space approach is 34% more (L_1-norm) fuel intensive than the online convex optimization solver. However, this value is already great improvement from the 92.27% of the constant thrust matrices observed in

Fig 5.

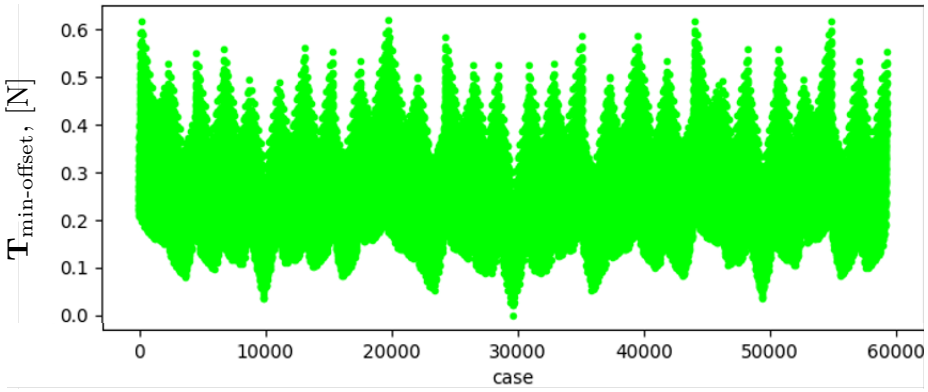

Figure 8. L_1 **norm of thrust vectors** $T_{\text{min-offset}}$**, expressed in Newtons**

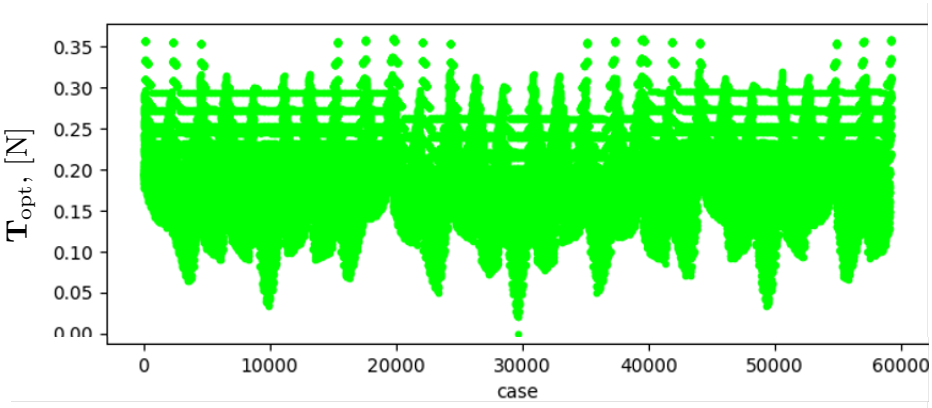

Figure 9. L_1 **norm of thrust vectors** T_{opt}**, expressed in Newtons**

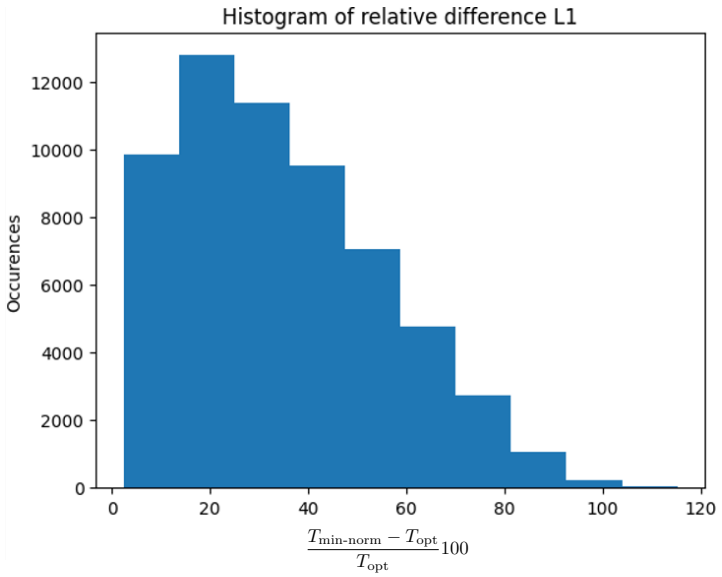

Figure 10. Histogram of relative difference in L_1 norms between $T_{\text{min-offset}}$ and T_{opt}. On average: $T_{\text{min-offset}} = 1,34 \cdot T_{\text{opt}}$

Finally, Table 1 provides a summary of the algorithms performance in terms of: 1) Total thrust (average of the 60,000 cases) and 2) ¡on time (total time necessary for running the 60,000 cases). Note that the execution time has been benchmarked using Python and not an embedded programming language. Having said that, these results already provide a good idea of how much more processing is needed when solving the full convex optimization problem instead of using an approximate method. The convex-optimization solver appears to be more than 10 times slower than the minimum-norm and null-space algorithm.

Table 1. Comparison of minimum-norm and null-space algorithm vs. convex optimization solver

Method	Average T_{L_1} (N)	Algorithm execution time (s)
Min-norm and null-space	0.2536	2.44
Convex optimization	0.1883	60.82

Although the convex-optimization solver is slower, the total execution time (roughly 0.001 sec/case) does not seem to be a complete bottleneck, so the feasibility of implementing this solver onboard would have to be judged by each mission. Additionally, please note that the convergence properties of the optimization solver (like stopping criterion and iterations) have not been investigated and this is left for future work.

VII Lessons Learnt from Closed-Loop Simulations: thrust matrices vs. min norm and null space

Astroscale is an orbital debris removal company and, as such, we have tested the thrust-distribution algorithms in several closed-loop simulations that encompass rendezvous and proximity operations with a target debris spacecraft. An example of such scenario is illustrated in Fig 11.

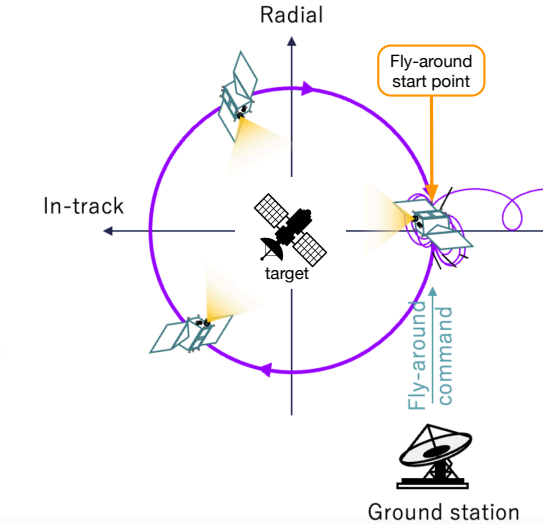

Figure 11. Illustration of a fly-around mission scenario

Although these simulations are proprietary and therefore not included in this paper, the authors would like to share some of the findings from the results. The two thrust-distribution algorithms compared are the constant thrust matrices and the minimum norm with null space. For each case, the total ΔV used throughout the entire scenario was evaluated. All the scenarios used in this comparison encompass a forced spacecraft motion (i.e., un-natural orbits) like the spacecraft fly-around illustrated in Fig 11. Depending on the scenario, total ΔV reductions in the range between 6% and 25% were observed. This is less than expected according to the open loop simulations, where:

$$T_{\text{mat}} = 1.99 \cdot T_{\text{opt}} \tag{31a}$$

$$T_{\text{min-offset}} = 1.34 \cdot T_{\text{opt}} \tag{31b}$$

Hence, expecting an improvement of approximately:

$$\frac{T_{\text{min-offset}}}{T_{\text{mat}}} \approx 67\% \tag{32}$$

The authors have found that one of the main reasons for this reduction in the percentage of improvement is the following: Not all the ΔV inefficiency comes from the thrust-distribution algorithm. In general, the requested forces and torques $\boldsymbol{Y}_{\text{req}}$ are computed by an onboard controller, and the way that the controller operates and cuts off the thrust in long maneuvers is critical. In that sense, a certain percentage of improvement in the thrust distribution has less impact in the overall ΔV usage, if the thrust distribution itself only accounts for a small percentage of the overall inefficiency.

In addition to the above, it would be interesting to investigate –in future work– whether there is any relationship between the percentage of improvement and the size of the requested forces and torques. Recall that in the open-loop simulations, the $\boldsymbol{Y}_{\text{req}}$ inputs were created with random values within some reference limits –which can be thought of

as a mesh-grid of forces and torques). However, the algorithm behavior (and in particular its optimality / efficiency) might differ in different areas of that mesh-grid.

That being said, the novel minimum-norm and null-space algorithm still proved to be an improved online strategy with respect to the conventional offline optimization.

VIII Results and Conclusions

On paper (i.e., based on the comparative open-loop simulations) the following conclusions can be made:

- Regarding the overall thrust/fuel, only the `cvxopt` online approach provides the truly optimal solution. Constant thrust matrices show to be 99% more fuel costly (in L_1) than `cvxopt`. In turn, minim-norm and null-space approach is about 34% more costly.
- Regarding the algorithms themselves, the `cvxopt` method is a recursive process that takes more time than the other methods (in Python, 60,000 cases were executed in 60.8s). In addition, the C implementation of this method –which would be required for onboard usage– is not trivial. In turn, the minimum-norm online approach is simple, robust and computationally fast (in Python, 60,000 cases were executed in 2.4s). Hence, being more suitable for onboard applications.

In practice (i.e., based on the closed-loop mission simulations performed internally in Astroscale), the exact impact that a more efficient thrust-distribution algorithm will have on the overall ΔV budget, depends on many other factors such as operational constraints and the GNC controller being used.

Overall, though, the novel minimum-norm and null-space algorithm seems to provide the best tradeoff of features: improved fuel usage and fast processing time, while satisfying force and torque accuracy requirements.