# VERIFICATION & VALIDATION OF OPTIMISATION-BASED CONTROL SYSTEMS: METHODS AND OUTCOMES OF VV4RTOS

**Pedro Lourenço[(1)], Hugo Costa[(1)], Pedro Cachim[(1)], João Branco[(1)], Pierre-Loïc Garoche[(2)], Arash Sadeghzadeh[(2)], Jonathan Frey[(3)], Gianluca Frison[(3)], Moritz Diehl[(3)], Anthea Comellini[(4)], Valentin Preda[(5)]**

[(1)]*GMV, Alameda dos Oceanos 115, 1990-392 Lisboa, Portugal, +351 213829366, {palourenco,hsequeira,pedro.cachim,jbranco}@gmv.com*
[(2)]*Ecole Nationale de l'Aviation Civile, Universite de Toulouse, France, {first.last}@enac.fr*
[(3)]*SYSCOP, IMTEK, Georges-Köhler-Allee 102, 79110 Freiburg, Germany, {first.last}@imtek.uni-freiburg.de*
[(4)]*Thales Alenia Space France, 5 Allée des Gabians, 06150 Cannes, France, anthea.comellini@thalesaleniaspace.com*
[(5)]*ESA-ESTEC, Keplerlaan 1, PO Box 299 2200 AG Noordwijk, The Netherlands, valentin.preda@esa.int*

## ABSTRACT

VV4RTOS is an activity supported by the European Space Agency aimed at the development and validation of a framework for the verification and validation of spacecraft guidance, navigation, and control (GNC) systems based on embedded optimisation, tailored to handle different layers of abstraction, from guidance and control (G&C) requirements down to hardware level. This is grounded on the parallel design and development of real-time optimisation-based G&C software, allowing to concurrently identify, develop, consolidate, and validate a set of engineering practices and analysis & verification tools to ensure safe code execution of the designed G&C software as test cases but aimed at streamlining general industrial V&V processes. This paper presents: 1) a review of the challenges and the state-of-the-art of formal verification methods applicable to optimization-based software; 2) the implementation for an embedded application and the analysis from a V&V standpoint of a conic optimization solver; 3) the technical approach devised towards and enhanced V&V process; and 4) model-in-the-loop test results and conclusions.

## 1 INTRODUCTION

Iterative embedded optimisation algorithms [1], [2] are paramount for the success of new and extremely relevant space applications: from launcher operations [3] to orbital servicing (including active debris removal), assembly, and manufacturing [4], from actuator allocation [5] to attitude guidance & control [6]. Guaranteeing the safe, reliable, repeatable, and accurate execution of such software is extremely important, not only due to their safety-critical applications but also to increase the trust in on-board optimisation-based algorithms and foster their adoption throughout the industry.
Convex optimization offers numerous benefits and is of great practical importance in various fields, especially when deploying optimization algorithms in real-time and safety critical system. The main advantages are the theoretical guarantees of global optimality, ensuring that numerical solvers can not get stuck in local minima and global convergence can be guaranteed independent of the initial guess.

Highly efficient convex optimization solvers exist which allow solving the problems in polynomial runtime and have been successfully deployed for large-scale systems and in real-time scenarios, [7]. Classes of convex problems range from linear programming over quadratic programming to second order cone programming. Even more general nonlinear programming solvers, such as sequential quadratic programming (SQP) often rely on solving convex sub-problems. The essential elements of a G&C, when implemented solving optimisation problems (OP) as shown in Figure 1, typically linearise, relax and convexify underlying Nonlinear Programming (NLP) formulations that often stem from the discretization of a continuous time Optimal Control Problem (OCP). Thus, it is essential to have a reliable and efficient convex optimization solver at hand.

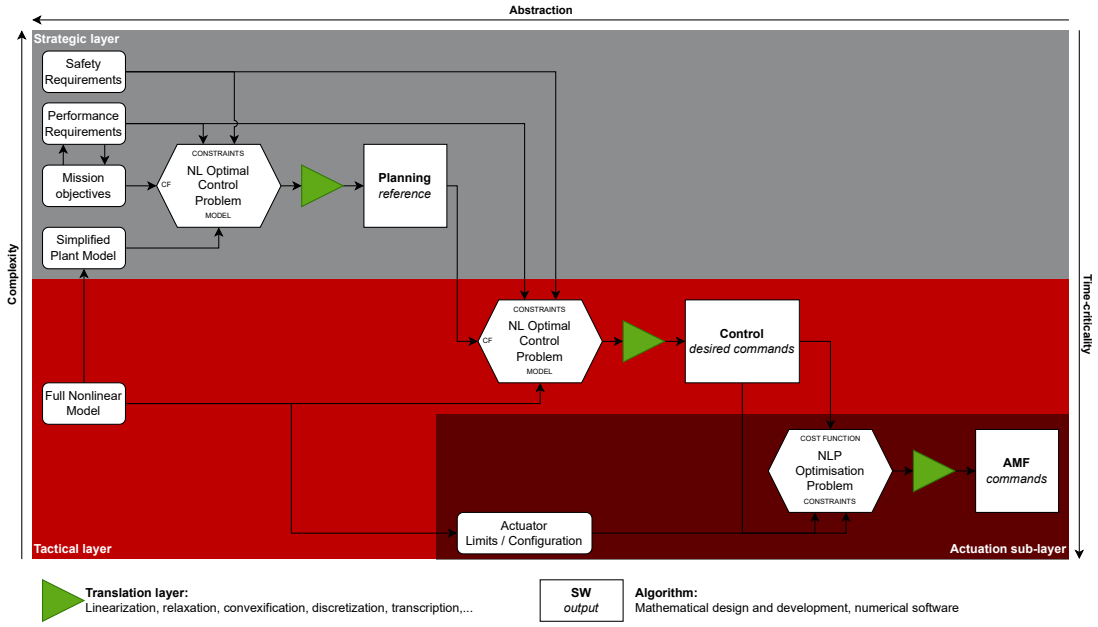



Figure 1: The development and operation of optimisation-based guidance and control systems: trajectory planning (guidance), tracking or regulation (control), and actuation allocation (actuator management).

Classical verification and validation (V&V) of G&C places a severe emphasis on comprehensive testing of the developed software within a model-in-the-loop (MIL) setting, thus enlarging the effort in the final stages of development and precluding the potentially beneficial reiteration of earlier stages. Even though testing is the ultimate tool for qualifying a new design, it is very difficult to generalise any results. Introducing formal verification methods to locally V&V components of the G&C software will not only ensure the safe execution of the related code, but also allow their reuse in other designs. When the V&V-ed component is a core one such as the optimisation solver, which is indeed the workhorse of an optimisation-based G&C system, this significantly enhances and expedites future designs.

It is therefore of utmost importance to develop verification means appropriate to support optimisation-based control systems. One of the issue in control system design processes is the gap between the design of a proper controller and its implementation as a system or a software. We propose here to consider the convex optimisation algorithm itself as a regular system subject to classical system engineering approaches. System Engineering provides a framework based on both the early identification of requirements and their later verification and validation. Instead of considering such numerical solvers as off-the-shelf black-box components, we rather define an adequate set of requirements and their validation means. Formal methods, such as formal specification and deductive methods, can be used to formally specify these requirements and reason on their validity.

In general, the VV4RTOS - Verification & Validation of Safety-Critical Real-time Optimised Guid-

ance Navigation and Control (GNC) SW Systems - study aims to contribute to the widespread usage of convex optimisation-based techniques across the space industry by 1) augmenting the traditional GNC software Design & Development Verification & Validation (DDVV) methodologies to explicitly address iterative embedded optimisation algorithms; and 2) consolidating the necessary tools for the fast prototyping and qualification of G&C software, grounded on strong theoretical foundations for the solution of convex optimisation problems generated by posing, discretization, convexification, and transcription of nonlinear non-convex optimal control problems to online-solvable optimisation problems.

To fulfil this mandate, two avenues of research and development were followed: the design and implementation of a benchmarking framework with optimisation-based G&C implementations and the improvement of the V&V process — two radical advances with respect to traditional GNC DDVV. On the first topic, the new optimisation-based hierarchy was exploited, from high-level requirements and objectives that can be mathematically posed as optimal control problems, themselves organised in different levels of abstraction, complexity, and time-criticality depending on how close to the actuator level they are. The main line of this work is then focused on the core component of optimisation-based G&C — the optimisation solver -– starting with a formal analysis of its mathematical properties that allowed to identify meaningful requirements for V&V, and, concurrently, with a thorough, step-by-step, design and implementation for embedding in a space target board. This application-agnostic analysis and development was associated with the DDVV of specific use-cases of optimisation-based G&C for common space applications of growing complexity, exploring different challenges in the form of convex problem complexity (up to second-order cone programs), problem size (model predictive control and trajectory optimization), and nonlinearity.

The novel V&V approach relies on the combination and exploitation of the two main approaches: classical testing of the global on-board software, and local and compositional, formal math-driven verification. While the former sees systems as black boxes, feeding it with comprehensive inputs and analysing statistically the outputs, the latter delves deep into the sub-components of the software, effectively seeing it as white boxes whenever mathematically possible. The deep analysis of the mathematical properties of the optimisation algorithm allows to derive requirements with increasing complexity (e.g., from “the code implements the proper computations”, to higher level mathematical properties such as optimality, convergence, and feasibility). These are related to quantities of interest that can be both verified resorting to E-ACSL [8] specifications and Frama-C [9] in a C-code implementation of the solver [10], but also observed in online monitors in Simulink or in post-processing during the model/software-in-the-loop testing.

Finally, the activity will apply the devised V&V process to the benchmark designs, from model-in-the-loop testing, followed by auto-coding and software-in-the-loop equivalence testing in parallel with the Frama-C runtime analysis, and will be concluded by processor-in-the-loop testing in a Hyperion on-board computer based around a Xilinx Zynq 7000 SoC.

## 2 V&V CHALLENGES AND STATE-OF-THE-ART

Verification & Validation are, typically, the final stages of the development of a G&C software. They ensure that there is sufficient confidence that the system will perform in operation as required by the mission objectives and as intended in the design. Given that it is generally not possible to accurately reproduce operational conditions on ground, this process must include the mathematical modelling of the operational setting and resort to analysis and simulation based on these models. It is divided in [11]: 1) Verification, which is the proof that the G&C function performs according to the specification under which it was developed; and 2) Validation, which is the proof that the G&C function will behave as expected under real world conditions.

In general, whenever safety-critical functions are discussed, it is particularly important to minimise all concerning risks prior to operation. The typical process is in accordance with the relevant ECSS standard [12], i.e., the reference for the development of AOCS/GNC systems. The AOCS/GNC design is consolidated using a Model-In-The-Loop (MIL) simulator that implements mathematical models of the hardware equipment, simulation of spacecraft dynamics and space environment, and model prototypes of the AOCS/GNC algorithms. After verification of the main performance requirements, the design is frozen and the AOCS/GNC engineers can produce the software requirement specification that is used as reference for production and implementation of the AOCS/GNC software application. The software application is validated with simulations in Software-In-The-Loop (SIL) environment, similar to the MIL simulator but this time implementing the final algorithms (usually C code) and the complete software application (including the mode logic, telecommands, telemetries etc.) instead of the prototypes (MATLAB code; and usually simplified functional implementation). A phase of Processor-In-The-Loop (PIL) testing is done just after to verify the correct behaviour (run time, memory usage, data precision etc.) of the software application on a processor with identical (or similar) features than the one that will be used in flight.
Finally, once the other hardware elements are available (on-board computer, sensors, actuators) the whole system is tested in the avionic test bench (Hardware-In-The-Loop, HIL, verification).
This approach is fundamentally grounded on the testing paradigm, i.e., on statistical analysis of numerical simulations – including specific cases, nominal scenarios, identification of worst cases, Monte Carlo techniques. This classical testing of the global on-board software sees systems as black boxes, feeding them with comprehensive inputs and analysing statistically the outputs. In opposition, local, math-driven, verification delves deep into the sub-components of the software, effectively seeing them as white boxes whenever mathematically possible. This dichotomy is further covered by figure 2. In between the two approaches lies the optimal path to a thorough, dependable, mathematically sound verification and validation process: local, potentially application-agnostic, V&V of the building blocks with respect to mathematical specifications leading up to application-specific testing of global complex systems, this time informed by the results of local validation and testing.

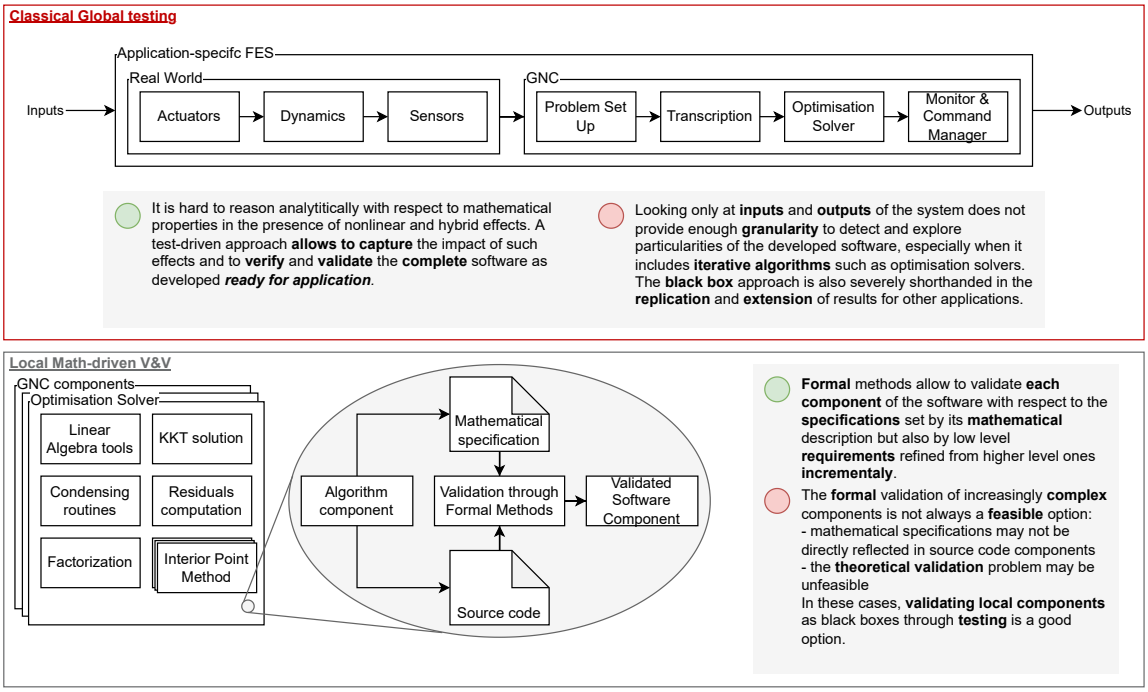



Figure 2: Comparison of the classical global testing approach followed by the GNC industry with formalism-inspired local math-driven approach from the computer science field and software industry.

### 2.1 Review of optimisation-based software

It was clear from the start of the project that the immensity of methods and software available, from active-set methods, interior-point methods, or first-order methods, would require essentially a separate in-depth study of each of these optimisation algorithms. In the scope of this activity, a single method had to be selected. Regarding deterministic runtime, interior point and first order methods are preferred over active-set methods, since limiting the number of iterations typically leads to a solution that is acceptable in the context of embedded optimization, while this is not generally true for active set methods, although some upper iteration bounds can be established with explicit MPC techniques. With respect to V&V, first order methods have some major benefits. Firstly, they only require simple linear algebra operations (like matrix-vector multiplications), while IPMs typically require matrix decomposition or inversions. If the underlying linear algebra implementation needs to be V&Ved, this simplifies the verification process of a first-order method compared to an IPM. Additionally, factorization and decomposition is more prone to numerical issues compared to the basic operations used in a first-order method. On the other hand, these methods are more sensitive to the scaling and conditioning of the optimisation problems, potentially requiring the usage of complex pre-conditioning techniques.

### 2.2 Survey of formal V&V approaches

In "An Axiomatic Basis for Computer Programming" [13], HOARE defines a deductive reasoning to validate code level annotations. This paper introduces the concept of HOARE triple $\{Pre\}code\{Post\}$ as a way to express the semantics of a piece of code by specifying the postconditions ($Post$) that are guaranteed after the execution of the code, assuming that a set of preconditions ($Pre$) was satisfied. HOARE supports a vision in which this axiomatic semantics is used as the "ultimately definitive specification of the meaning of the language [. . . ], leaving certain aspects undefined. [...] Axioms enable the language designer to express its general *intentions* quite simply and directly, without the mass of detail which usually accompanies algorithmic descriptions."
This formalization is an essential step to specify the intended behaviour of a program. It shall be the main reference when describing the specification, rather than, or in addition to, natural language specification. Furthermore, formal specification, in the form of Hoare triple, can support the validation of test-based activities but also enable the use of formal reasoning algorithm to ensure its validity *in all possible uses*.

**Formalizing specification.** Multiple frameworks such as the B method [14], ACSL [15] (ANSI C Specification Language) or SPARK ADA [16] provide means to specify formally the expected behaviour of a model or software function. These languages allow to define mathematical objects such as sets, theorems, predicates, axioms and attach them to code using these Hoare pre- and post-conditions.

**Reasoning formally on code using logics.** After the proposition by Hoare [13] to attach contract to functions, Dijkstra proposed to mechanically transform these predicates along the code. Depending on the direction of propagation, one obtains the weakest precondition computation or the strongest postcondition computation. In the former case, the post-condition is propagated backward from the last instructions of the code towards the beginning of the function body. The resulting predicate are then compared with the other element of the Hoare triple. In the weakest precondition computation setting, one needs to ensure that the specified pre-condition is logically stronger than the weakest one computed that ensures to obtain the post-condition after executing the code. It is an implication: $Pre \implies WeakestPrecondition(code, Post)$.

Once this proof objective is produced, it remains to prove it. The two main options are either the use of automated reasoning – Satisfiability Modulo Theory (SMT) solvers, e.g., Z3 [17], CVC5 [18], Alt-Ergo [19] – or manual proof using proof assistants such as Coq, PVS or Isabelle.

**Reasoning formally on code using sets.** The previous methods are extremelly expressive but limited to the capabilities of the underlying solvers to discharge the proof objectives. When considering numerical software or numerical properties performed with machine types such as floating point numbers, the axiomatisation of the computation produces difficult proof objectives.
Another alternative method to reason about the validity of safety properties, ie. invariants of program states, is static analysis, also known as abstract interpretation. It was introduced by Cousot and Cousot [20] in the 70s. Here, checking the validity of property amounts to check set inclusion. Let $R$(each) be the set of reachable states and $G$(ood) be the set of good, of valid, states. Checking that all computed values – the reachable ones – are valid, amounts to check that $R \subseteq G$.
Abstract interpretation provides tools to compute *over-approximation* of sets. The idea is that if we are able to capture the set of good states in a set $G$ but have issues computing $R$ exactly, we can compute an over-approximation $\tilde{R}$ such that $R \subseteq \tilde{R}$ and check the sufficient condition $\tilde{R} \subseteq G$. Sometimes $G$ is also difficult to represent but one could characterize its complement $B$(ad), or even an over-approximation $\tilde{B}$ of $B$: $B \subseteq \tilde{B}$, and verify that $\tilde{R} \cap B = \emptyset$, or the sufficient condition $\tilde{R} \cap \tilde{B} = \emptyset$, i.e. no reachable state is bad.
These methods are usually more fitted to bound numerical errors due to floating point computations and can be combined with the previous *deductive methods*.

## 3 DESIGN AND DEVELOPMENT OF AN EMBEDDED CONIC OPTIMISATION SOLVER

In the context of optimisation-based control, quadratic programming (QP) or second order cone programming (SOCP) formulations are essential. In order to react to disturbances, model-errors and changing references quickly, the control loop has to be carried out in a short and deterministic runtime. Fast and robust QP, QCQP or SOCP solver implementations are thus an essential backbone for the embedded deployment of optimization based control and are the main scope of optimization software to be V&Ved in this project.

### 3.1 Background

The previously mentioned categories of OPs can fit within the larger category of conic optimisation, which aims at the minimisation of a differentiable convex objective function subject to conic constraints, i.e.,

$$\begin{aligned} \underset{\mathbf{z}}{\text{minimize}} \quad & f(\mathbf{z}) \\ \text{subject to} \quad & \mathbf{z} \in \mathbb{D}, \\ & \mathbf{H}\mathbf{z} - \mathbf{g} \in \mathbb{K} \end{aligned} \tag{1}$$

where $\mathbf{z} \in \mathbb{R}^n$ is the decision variable, $f : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable and convex objective function, $\mathbb{K} \subset \mathbb{R}^m$ is a closed convex cone and $\mathbb{D} \subset \mathbb{R}^n$ is a closed convex set, $\mathbf{H} \in \mathbb{R}^{m \times n}$ and $\mathbf{g} \in \mathbb{R}^m$ are constraint parameters. A convex set $\mathbb{K}$ is a cone [7, Section 2.1.5] if $\forall \mathbf{x} \in \mathbb{K}$ and $\gamma > 0$ then $\gamma\mathbf{x} \in \mathbb{K}$. The *polar cone* of $\mathbb{K}$ is also a closed convex cone given by

$$\mathbb{K}^\circ := \{\, \mathbf{w} \in \mathbb{R}^m \mid \langle \mathbf{w}, \mathbf{y} \rangle \leq 0, \forall \mathbf{y} \in \mathbb{K} \,\}. \tag{2}$$

The burden lies on the GNC engineer to transform a performance or safety requirement into constraints that can be expressed as these convex sets or cones, so that the problem can be specified in

a form that the optimisation solver can handle. Fortunately, the typical constraints that appear on GNC problems (actuation limits – limited norms, pointing restrictions – "ice-cream"cones, collision avoidance – balls) can be expressed as small list of convex sets (balls, boxes, half-spaces) and cones (the origin, the set of reals, orthants, second-order cones) which accept a closed-form projection: $\pi_C[\mathbf{x}] = \arg\min_{\forall \mathbf{y} \in \mathrm{C}} \|\mathbf{x} - \mathbf{y}\|$ is the projection of $\mathbf{x}$ to set $C$.

### 3.2 Proportional-Integral Projected Gradient solver for conic optimisation

PIPG is a first-order algorithm for convex optimisation [10] that has important theoretical convergence guarantees: it converges to the optimal solution with $j^2$ and to a feasible solution with $j^3$ for strongly convex problems (convergence is linear for weakly convex ones). Given its very interesting properties, in VV4RTOS this solver was implemented in embedded MATLAB and Simulink and its convergence results were used as a basis for the V&V analysis.

**Algorithm 1** PIPG

**Input:** $M$, $\{\alpha^j, \beta^j\}_{j=1}^M$, $\mathbf{z}^1 \in \mathbb{D}$, $\mathbf{v}^1 \in \mathbb{K}^\circ$
**Output:** $\mathbf{z}^M$

**for** $j = 1, 2, \ldots, M$ **do**
  $\mathbf{w}^{j+1} = \pi_{\mathbb{K}^\circ}[\mathbf{v}^j + \beta^j(\mathbf{H}\mathbf{z}^j - \mathbf{g})]$
  $\mathbf{z}^{j+1} = \pi_{\mathbb{D}}\left[\mathbf{z}^j - \alpha^j(\nabla f(\mathbf{z}^j) + \mathbf{H}^\top \mathbf{w}^{j+1})\right]$
  $\mathbf{v}^{j+1} = \mathbf{w}^{j+1} + \beta^j \mathbf{H}(\mathbf{z}^{j+1} - \mathbf{z}^j)$
  **if** Stopping criterion **then**
    break loop
  **end if**
**end for**

In algorithm 1, which solves (1), each $j$-th iteration outputs an estimate of the optimal dual solution, $\mathbf{w}^j \in \mathbb{R}^{n_d}$, of the optimal primal solution, $\mathbf{z}^j \in \mathbb{R}^n$, and an intermediate dual term $\mathbf{v}^j \in \mathbb{R}^{n_d}$. The input $M \in \mathbb{N}$ is the maximum number of iterations and $\{\alpha^j\}_{j=1}^M$ and $\{\beta^j\}_{j=1}^M$ are sequences of positive scalar step sizes – whose structure of computations is imposed by the theoretical convergence guarantees. This being a first-order method, it relies on the gradient of the cost function for the steps towards the solution, expressed as $\nabla f$. In addition to the maximum number of iterations, other stopping criterion can be used, such as the duality gap, distance to the cone $\mathbb{K}$, or the speed of convergence. The VV4RTOS embedded implementation of algorithm 1 enforces

$$f(\mathbf{z}) = \mathbf{z}^T \mathbf{P} \mathbf{z} + \mathbf{q}^T \mathbf{z}, \tag{3}$$

with positive definite $\mathbf{P}$, and provides simple and efficient projections to both closed convex sets (balls, boxes, halfspaces) and closed convex cones (origin, orthants, the set of reals, second order cones). These can be employed by the user to build their own constrained sets when posing their optimisation problems.

### 3.3 Solver analysis

The excellent convergence results of PIPG, provided by [10, Theorem 2], rely on a number of assumptions related to the convexity and smoothness of the cost function (evaluated through the eigenvalues of $\mathbf{P}$ if $f$ is defined as in (3)), the feasibility of the OP (related to the primal-dual gap), and formulas for the primal ($\alpha^j$) and dual ($\beta^j$) steps

$$\alpha^j = 2\big((j+1)\mu + 2\lambda\big)^{-1}, \qquad \beta^j = (j+1)\mu\,(2\sigma)^{-1}, \qquad \alpha^j\left(\lambda + \sigma\beta^j\right) = 1, \tag{4}$$

where $\mu$ and $\lambda$ are the smallest and largest eigenvalues of $\mathbf{P}$, and $\sigma > \|\mathbf{H}\|^2$. The main result, from which the mentioned convergence rates are extracted, is as follows: given that these assumptions hold, a sequence of primal and dual iterates of PIPG respects

$$L(\bar{\mathbf{z}}^k, \mathbf{w}) - L(\mathbf{z}, \bar{\mathbf{w}}^k) \leq \frac{4\lambda V^1(\mathbf{z}, \mathbf{w})}{\mu k(k+5)} \tag{5}$$

$$\bar{d}_{\mathbb{K}}\big(\mathbf{H}\tilde{\mathbf{z}}^k - \mathbf{g}\big) \leq \frac{12\lambda\sigma V^1(\mathbf{z}^\star, \mathbf{w}^\star)}{\mu^2 k(k^2+6k+11)} \tag{6}$$

where $\bar{\mathbf{z}}^k, \tilde{\mathbf{z}}^k \in \mathbb{D}$ and $\bar{\mathbf{w}}^k \in \mathbb{K}^\circ$ are linear combinations of the elements of the sequence, $L(\cdot,\cdot)$ is the Lagrangian associated with (1), $\bar{d}_{\mathbb{K}}(\cdot)$ is the square of the distance to the cone $\mathbb{K}$, and $V^1$ is a function of the distance to the initial guesses of both primal and dual.
Aside from the above assumptions, the proofs of these results are based on a number of inequalities and conditions that should inform as well the verification process. Examples of these inequalities that should hold for the convergence results to be valid are

$$\bar{d}_{\mathbb{K}}\big(\mathbf{H}\mathbf{z}^j - \mathbf{g}\big) \leq \frac{1}{2}\left\| \frac{1}{\beta^j}\left(\mathbf{v}^j - \mathbf{w}^{j+1}\right)\right\|^2, \tag{7}$$

and

$$L(\mathbf{z}^{j+1}, \mathbf{w}) - L(\mathbf{z}, \mathbf{w}^{j+1}) + \beta d_{\mathbb{K}}\big(\mathbf{H}\mathbf{z}^j - \mathbf{g}\big) \leq \left(\frac{1}{2\alpha^j} - \frac{\mu}{2}\right)\left\|\mathbf{z}^j - \mathbf{z}\right\|^2 + \frac{1}{2\beta^j}\left\|\mathbf{v}^j - \mathbf{w}\right\|^2 - V^{j+1}(\mathbf{z}, \mathbf{w}). \tag{8}$$

As will be seen in the sequel, the requirement for convergence to optimality and feasibility, which is the core requirement underlying the more high-level performance requirements of any G&C software that uses optimisation algorithms, is translated into implementation, problem-specific, parametric, and execution requirements that feed directly from these elements of the formal proofs. This is the natural path to be followed in a formal-infused V&V: to distil high-level requirements to more and more specific requirements associated with the mathematical properties of the algorithms.

## 4 ENHANCED V&V OF OPTIMISATION-BASED G&C ALGORITHMS

Once the choice of a specific algorithm has been made, we now need to build the sets of requirements that will drive its implementation and, later, its verification and validation.
In order to illustrate the different processes to perform this V&V activity, two tracks are followed in the VV4RTOS project: 1) one based on formal methods, where we rely on Frama-C, an open-source toolbox providing different tools to perform formal verification of C code, and more specifically on the E-ACSL plug-in, supporting dynamic checks; 2) one based on a classical development process with MIL/SIL/PIL in which we aim at infusing methods defined in the first track.

### 4.1 Overview of formal approach

#### 4.1.1 Identifying requirements from algorithm descriptions and proofs

In this study we focus on conic optimisation and a first simple set of requirements could only focus on the expected outcome: a solution to the solved optimisation problem which is a feasible solution of the set of constraints, and optimal with respect to other possible solutions. Typically this is what

is evaluated when considering the component as a *black-box*. A set of experiments will rely on the solver and the obtained solution compared with expected values.
However, we want to trust more the component itself. As a result we need to gather more fine-grained elements to build a complete set of requirements. We rely on the algorithm description and its proof elements, as summarized in the previous section, to identify properties that have to be fulfilled or satisfied by the implementation.
We structured these requirements in four sets:

- *Implementation related requirements* (RIM), mainly focused on the validity of the operations used: linear algebra computations, projections to sets, computation of gradients, etc.
- *Requirements specific to the optimization problem* (RPR); concerns more mathematical properties of the problem description: is the problem convex?, is the cost function differentiable?, etc.
- *Requirements of the parameters* (RPA): local elements definitions, like local variables of the program and their expected properties, e.g., do they belong to specific sets?
- *Requirements of the execution* (REX) of the algorithm. Semantics properties justifying the convergence of the algorithm, or the computation of a feasible and optimal value.

The table 1 gives one example for each of such categories of requirements.

Table 1: Example of requirements extracted from the algorithm description, focused on set $\mathbb{K}^\circ$.

| **Req. ID.** | **Requirement description** | **Rationale** |
|---|---|---|
| VV-RIM-042 | Projecting vector $w \in \mathbb{R}^n$ onto polar cone $\mathbb{K}^\circ$ | Algorithm 1 |
| VV-RPR-041 | $\mathbb{K}^\circ$ is a polar cone of $\mathbb{K}$ | Algorithm 1; see eq. (2) |
| VV-RPA-006 | $w^{j+1}$ belongs to $\mathbb{K}^\circ$ | Algorithm 1; to make sure that the projection step onto set $\mathbb{K}^\circ$ is correctly performed |
| VV-REX-011 | (8) holds for all $z \in \mathbb{D}$ and $w \in \mathbb{K}^\circ$ | [10, Appendix D]; the main condition based on which the convergence to zero of primal-dual gap and violation of constraint are derived. Result of [10, Lemma 1]. |

#### 4.1.2 Associating means of compliance to requirements

We now aim at expressing them as program logical statements, that is, predicates over the program variables. We can partition our set of requirements into different categories depending on their nature or our ability to analyse them. Table 2 presents such a list of categories.

Table 2: Categorisation of requirements with respect to the ability to evaluate them on code.

| | |
|---|---|
| **CAT1** | simple predicates over program variables |
| **CAT2** | predicate relying on mathematical objects or computation that are not directly available as functions in the code |

| | |
|---|---|
| **CAT3** | similar concerns but expressed over quantities that do not appear in the code |
| **CAT4** | requirements that involve universal quantification over elements of a set |
| **CAT5** | requirements that involve existential quantification over elements of a set or over quantities that are not available or computable in the source code |
| **CAT6** | requirements that describe more general properties of mathematical object and that are not directly expressible over program variables |

Once these requirements are identified and classified we have to identify for each of them an appropriate *mean of compliance*, i.e., a verification method aimed at validating them. Here a wide set of methods is available, from classical and less formal ones, such as code reviews, to complete proof using the deductive method mentioned above.
In order to provide a realistic approach to the problem considered, we proposed a three level method:

1. to rely on reviews to check most purely mathematical properties of the problem itself;
2. to formalize specification and evaluate it while testing using *executable formal specification*;
3. to specify and prove basic properties on linear algebra components.

In the first (the weakest) level the properties are both mainly guaranteed by construction and not easy to validate in the code. For example requirements stating that the cost function is differentiable, that the cone $\mathbb{K}^\circ$ is the polar of the cone $\mathbb{K}$ of constraints, etc.
The third level is the most challenging one since it requires a considerable amount of effort to achieve the proofs. For the moment, we focused on basic linear algebra operations such as matrix vector additions and multiplications. Ideally the second level shall be addressed with this exhaustive proof approach, but this is not realistic within the scope of the project. This amounts to introduce numerous predicate, lemmas, theorems and prove them to validate most properties. A similar approach has already been illustrated for the ellipsoid method [21] and a primal interior point method for LP [22].
The main activity is therefore on the second level: specifying formally the requirement in such a way that these requirements can act as test oracle when executing the code, or an instrumented code.

### 4.1.3 Test-based approaches and quantifiers

An interesting intermediate approach between formal proof and test is supported by the tool Frama-C[1] and its E-ACSL plug-in. While ACSL is a very generic language to express specification and C code providing axiomatic semantic definitions, lemmas, properties, predicates and theorems, a lot of its logical elements cannot be executed. A typical example is the use of a quantifier over a large set, i.e., not a small enumerable one.
On the logical side it is perfectly expressible and can be used to prove properties over the program. However, it is difficult to execute as a regular program statement. E-ACSL considers both a restriction of the ACSL language, but also a plug-in to instrument annotations into runtime checks.
In the project, we propose to express the requirements in E-ACSL. Some extra quantities have to be defined, like additional local variables encoding convergence results. These quantities appear in the

[1] Available from `https://frama-c.com/`

proof but typically not in the algorithm itself. Relying on E-ACSL amounts to declare *ghost code*: code that will be executed for verification purpose but is not supposed to be part of the final product. Regarding quantifiers, we are facing two issues. Existential quantifiers are used to describe a specific value which is usually not known *a-priori*; for an example the goal of the computation, i.e., the primal-dual solution. In that case, we propose, for verification purposes only, and for the ones based on test, to run a first computation to get these values and instantiate these quantifiers with them. Universal quantifiers have a different role: they aim to express that a property is valid for a full set. In that case, we can either sample the set using Monte-Carlo-like techniques, or gather elements of these sets during the execution and evaluate these properties on the accumulated values.

### 4.2 Overview of MIL verification

The approach described in the previous sections is focused on the verification of the source code, even though it also resorts to test for runtime verification. The underlying concept is clear: formal requirements from the mathematical proofs should be extracted, and then these can be associated to quantifiers resulting effectively in test oracles. This idea can naturally be ported to the model-in-the-loop level, infusing the traditional MIL test campaigns with more in-depth verification procedures. When V&V-ing a "regular" control system, the main concern is the closed-loop behaviour: high-level performance metrics are evaluated, more esoteric concepts such as stability are addressed, etc. Depending on the nature of the control algorithms, some of these aspects are preliminarily addressed at design level both through design choices (e.g., weights, gains, other tuning parameters) and analysis tools (e.g., by assessing gain and phase margins for linear controllers, using $\mu$-analysis for robust controllers, etc.). This implicit process can be applied to more advanced control systems relying on the solution of optimisation problems, as seen in Table 3.

Table 3: Example of flow-down of requirements, from higher-level to design choice, from design choice to lower-level.

| **Level** | **Requirements** | **Mapping to design** |
|---|---|---|
| High-level requirements (closed-loop) | Functional requirements (e.g, control system shall minimize propellant consumption) | Cost function selection. |
| | Performance requirements (e.g, final dispersion shall be smaller than X units) | Constraint definition. |
| | Mission requirements (e.g, control function shall use less than Y units of propellant) | Constraint definition. |
| | Safety requirements (e.g, collisions shall be avoided) | Constraint definition. |
| | Hosting requirements (e.g, control function shall compute in less than Z% of the sampling time) | Algorithm choice. |
| G&C-level requirements (closed-loop and function output) | Constraint satisfaction (e.g, do the predicted and effective trajectories respect the (possibly nonconvex) safety constraints?) | Algorithm choice. |
| | Solution optimality (e.g, is this trajectory fuel-optimal?) | Algorithm choice. |

| Level | Requirements | Mapping to design |
|---|---|---|
| | Convergence of the solver (e.g., is a solution obtained within the allotted time?) | Algorithm choice. |
| Solver requirements (function output and inner computations) | Algorithm description- and proof-based requirements (e.g., see table 1) | Parameters meeting algorithm assumptions. |

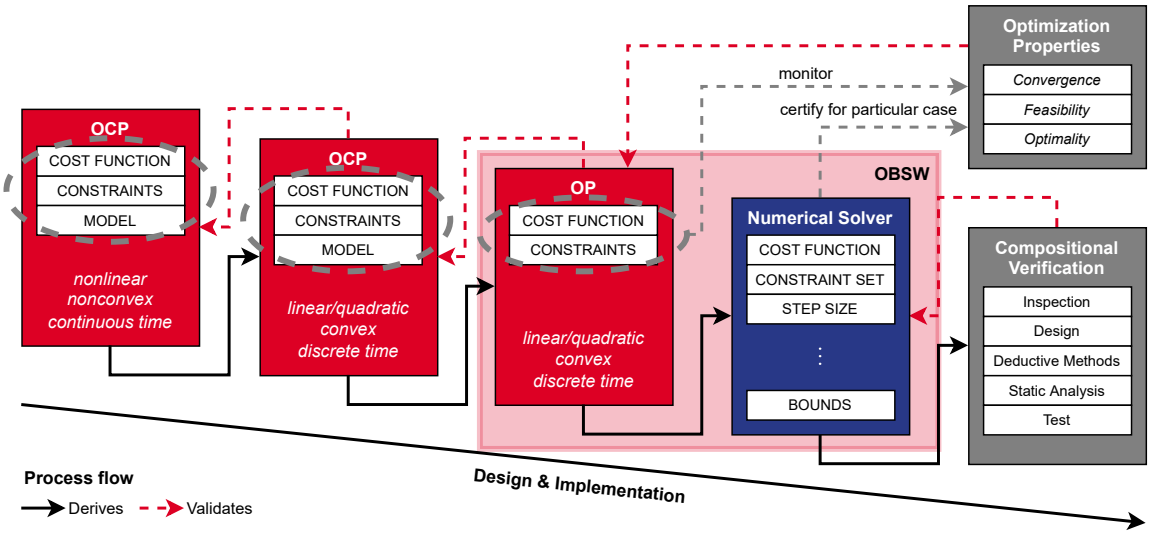



Figure 3: Verification & validation layers for optimization-based G&C software.

High-level requirements, effectively verified by GNC closed-loop simulation, can be translated in the design process to elements of optimal control problems (the objectives written as cost functions, performance and safety requirements as constraints) and then distilled into requirements at the G&C level – verified at the closed-loop but also at function output. Since a relevant number of the problems is nonlinear and nonconvex, sequential convex programming is used, resulting in optimisation problems solved by convex solvers. Therefore, these requirements can again be translated into a deeper level – verified at the output and inner variable level – effectively ensuring the correct execution of the solver software/models. This process is also shown in figure 3.

## 5 PRELIMINARY RESULTS

### 5.1 Benchmark problems

To apply the V&V approach described above, a set of benchmark problems were developed and implemented based on the PIPG solver described in section 3: 1) actuator allocation (thruster opening times); 2) model predictive control for far-range rendezvous; and 3) sequential convex programming for rendezvous.

The second benchmark problem implements a guidance algorithm for a chaser rendezvousing with an uncooperative target in geostationary orbit (GEO). The objective is for the chaser to perform a

sequence of impulsive transfers to a holding-point behind the target, using a reaction control system (RCS). The time-instants at which the impulsive transfers are allowed are predetermined and fixed. The model used in the optimiser for the relative dynamics between the chaser and the target are the Clohessy-Wiltshire equations [23]. The final position and velocity of the chaser relative to the target are constrained to be inside boxes. There is a maximum $\mathbf{\Delta V}$-budget, given as the maximum sum of the 1-norms of each $\mathbf{\Delta V}$. For each impulse, there is a maximum $\mathbf{\Delta V}$ allowed to be applied for each direction in the LVLH frame. The cost penalises the velocity impulses by minimizing the sum of the 1-norm of the $\mathbf{\Delta V}$s, while also penalising quadratically the final distance to the holding-point. The algorithm is implemented as a shrinking-horizon MPC, solving the optimization problem

$$\underset{\substack{\mathbf{x}(t_{i+1}),\,\mathbf{\Delta V}(t_k)\\ i=k,\ldots,f-1}}{\text{minimize}} \quad \frac{1}{2}\left(\mathbf{r}(t_f)-\mathbf{r}_{\mathrm{HP}}\right)^T \mathbf{Q}_{\mathbf{r}}\left(\mathbf{r}(t_f)-\mathbf{r}_{HP}\right)+R\sum_{i=k}^{f-1}\mathbf{1}^T\left|\mathbf{\Delta V}(t_i)\right| \tag{9a}$$

$$\text{subject to} \quad \mathbf{x}(t_{i+1})=\mathbf{\Phi}\left(t_i,t_{i+1}\right)\mathbf{x}(t_i)+\mathbf{G}\left(t_i,t_{i+1}\right)\mathbf{\Delta V}(t_i), \tag{9b}$$

$$\mathbf{x}(t_k) \text{ given}, \tag{9c}$$

$$\mathbf{r}(t_f),\mathbf{v}(t_f)\in\mathcal{R}_f\times\mathcal{V}_f, \tag{9d}$$

$$\sum_{i=k}^{f-1}\mathbf{1}^T\left|\mathbf{\Delta V}(t_i)\right|\leq\mathbf{\Delta V}_{\mathrm{budget}}, \tag{9e}$$

$$\left|\mathbf{\Delta V}(t_i)\right|_\infty\leq\mathbf{\Delta V}_{\max}. \tag{9f}$$

The problem is transformed to a strongly convex by condensing the dynamics to a single equality constraint involving only the initial and final states and the $\mathbf{\Delta V}$s; decomposing the $\mathbf{\Delta V}$ impulses into their positive and negative parts; adding the total spent budget as an optimization variable; and adding regularisation costs to the final relative velocity, spent budget, and positive and negative parts of the $\mathbf{\Delta V}$s.

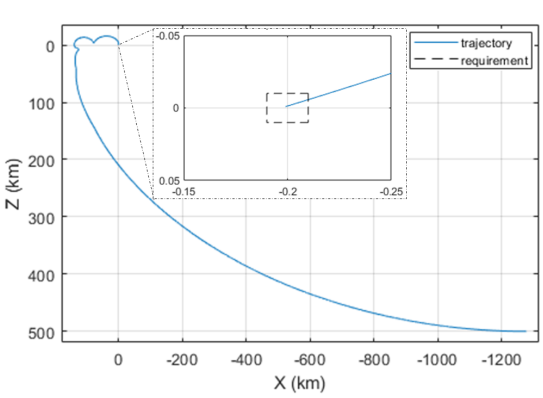
To improve the convergence of the optimisation algorithm, preconditioning of its matrices (cost Hessian and cone Jacobian) is implemented offline.

The algorithm was implemented in a Simulink block capable of being auto-coded. The block receives the current relative position and velocity of the chaser relative to the target, the available $\mathbf{\Delta V}$ budget, and the current time-instant. It outputs the $\mathbf{\Delta V}$ which the optimizer calculates for the current time-step. The block was integrated into a Functional Engineering Simulator that simulates the movement of the target using a Keplerian orbit and the dynamics of the chaser using the universal law of gravitation, with thruster models to perform the velocity impulses, and position and velocity sensor models.

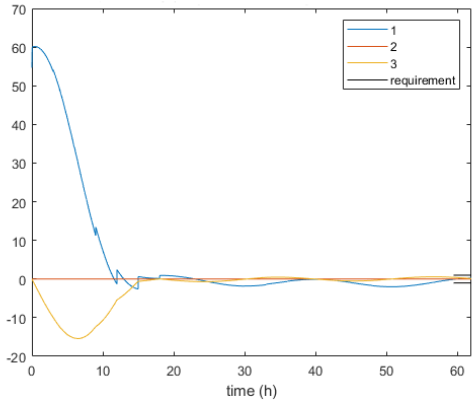
## 5.2 Verification & Validation at MIL level

### 5.2.1 Performance Requirements

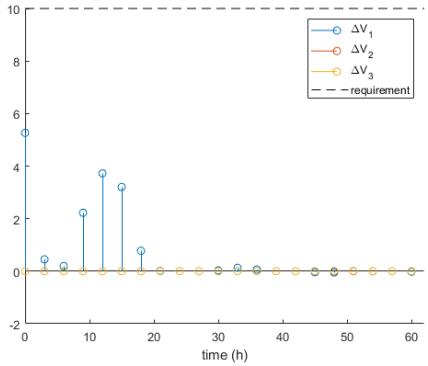
The results of a nominal simulation of the rendezvous benchmark problem are presented in figure 4. The target is in a circular orbit with an altitude of 35 793 km. The chaser starts with a position of $(-1278.1, 0, 500)$ km relative to the target in its LVLH frame, and an initial relative velocity of $(54.69, 0, 0)$ m/s. After 62 hours, the chaser is required to reach the holding-point at the relative position of $(-200, 0, 0)$ m $\pm$ 10 m with each velocity component less than 1 m/s. The impulsive transfers are allowed every 3 hours since the start, with each $\mathbf{\Delta V}$ component not exceeding 10 m/s in each transfer, and not exceeding the $\mathbf{\Delta V}$-budget of 20 m/s. For the cost in the guidance algorithm, $\mathbf{Q}_{\mathbf{r}}$ is chosen as a diagonal matrix with diagonal elements 1 m$^{-2}$, $R$ is $1\times10^{-3}$ s/m, and the regularisation costs as $1\times10^{-2}$ in their respective SI units. It can be seen that in this nominal simulation the requirements on the final relative position (figure 4a) and velocity (figure 4b), $\mathbf{\Delta V}$ impulses (figure 4c),



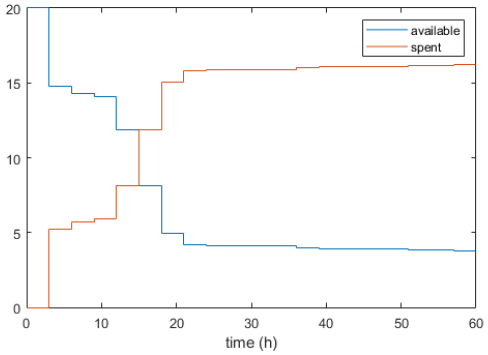

(a) Trajectory in the LVLH frame




(b) Relative velocity (m/s) in the LVLH frame




(c) $\Delta\mathbf{V}$ (m/s) in the LVLH frame




(d) Available and spent $\Delta\mathbf{V}$-budget (m/s)

Figure 4: Rendezvous nominal simulation

and $\Delta\mathbf{V}$-budget (figure 4d) are satisfied. The PIPG optimisation algorithm takes roughly 500 iterations per instant, except at 54 h, when it takes 2 000. The linearised model used in the optimisation algorithm is found to have a significant error relative to the simulated dynamics, but the guidance function is able to successfully re-optimise the $\Delta\mathbf{V}$s during the rendezvous, leading to differences in $\Delta\mathbf{V}$s that are applied relative to the nominal solution; most noticeably the appearance of the $\Delta\mathbf{V}$s smaller than 1 m/s in figure 4c. Nonetheless, the spent $\Delta\mathbf{V}$-budget of 16.2 m/s turns out to be less than nominal one calculated by the algorithm in the initial instant of 18.15 m/s.

### 5.2.2 PIPG Requirements

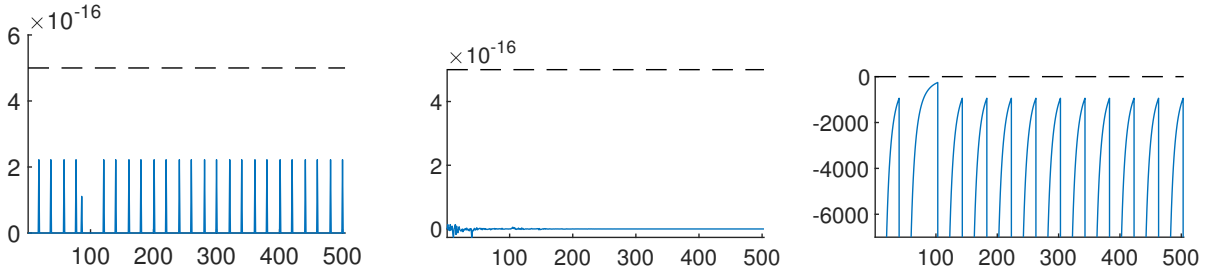
A small subset of the requirements identified for the verification and validation of PIPG are shown in table 4. The compliance with respect to these requirements was verified through the use of test oracles that evaluate a given quantity in each iteration of the algorithm and check if the condition is satisfied. To account for numerical errors a small threshold is considered in the verification of the requirements that include equalities (i.e., those expressed by $f(\cdot) \geq 0$ or $g(\cdot) = 0$). Figure 5 shows the value of the quantities verified in each requirement.

## 5.3 Verification with Frama-C

The verification with Frama-C and E-ACSL follows this process. First, we define, in comment, E-ACSL constructs: here a ghost variable declaring some threshold, set to zero, and an assert.

Table 4: Sample PIPG requirement verification

| Req. Id. | Description | Condition | Threshold | Success ratio |
|---|---|---|---|---|
| VV-REX-002 | Assumption $\alpha^j(\lambda + \sigma\beta^j) = 1$ of [10, Lemma 1] holds (see (4)) | equality | $1 \times 10^{-16}$ | 100% |
| VV-REX-006 | Consequence of [10, Theorem 2] on the distance to the cone holds (see (6)). | inequality | $1 \times 10^{-16}$ | 100% |
| VV-REX-010 | Condition B.14 of the proof of [10, Lemma 1] holds (see (7)). | inequality | $1 \times 10^{-16}$ | 100% |




(a) VV-REX-002 (b) VV-REX-006 (c) VV-REX-010

Figure 5: Quantities corresponding to the PIPG requirements, with maximum threshold for acceptance (dashed)

```
1  //@ ghost double eps  = 0;
2  //@ assert z1[0]*z1[0] + z1[1]*z1[1] <= a*a * (1 + eps);
```

Contracts can also be attached to functions.
We can then run the E-ACSL script:

```
% e-acsl-gcc.sh -l "-lm" -c -omonitored.c pipg.c projections.c utils.c linalg.c
```

It will instrument the source code and produce both the original binary and the instrumented one.
A first positive outcome of such an instrumentation is to detect bugs in the specification or in the source. As an example during our experiments, we miss-characterized the definition of set $\mathbb{K}^\circ$: instead of $|x| \leq 0.2y$ we used $|y| \leq 0.2x$. This lead to the definition of an erroneous check `inKnot`$(\cdot)$. Note that the check of inclusion in $\mathbb{K}^\circ$ is usually not part of the algorithm. The set $\mathbb{K}^\circ$ is also typically not explicitly characterized. This is part of the specification activity to do so.
When running the tests, the projections - which were properly coded - were not validated. The error found lead to an analysis of both the code and the specification and allowed us to identify the error in the specification. A black-box test may have missed this case.
Note that we have to be careful with equality constraints such as the one expressed in the right-hand side of (4), and demonstrated as follows.

```
% ./a.out.e-acsl
pipg.c: In function 'main'
pipg.c:166: Error: Assertion failed:
The failing predicate is:
alpha * (cost_lbd + sig * beta) - 1 == 0.
With values at failure point:
- beta: 1.500000e+00
```

```
- sig: 1.000000e+00
- cost_lbd: 1.000000e+00
- alpha: 4.000000e-01
```

While the property shall apply in a Real semantics, the computation with machine-type floating point number lead to a violation. Similar issues arrive when a point is projected at the frontier of a set before checking it belong to the set.
Such requirements have to be relaxed to allow minimal errors, if allowed by the algorithm or the designers.

## 6 CONCLUSIONS AND FUTURE WORK

The VV4RTOS project is studying the verification and validation of optimisation-based guidance and control software, allowing to increase the trust in such methods and advancing the state-of-the-art of these verification procedures. In this paper, sound guidelines are provided, informed by formal verification methods and resulting in a mixed testing-proof approach. The use of optimisation-based algorithms allows to create a clear mapping from high-level requirements to optimisation elements (cost function, constraints) which then transfer the burden of verification to the verification of the underlying optimisation solver. This last step is achieved starting with a formal review of the algorithm description and mathematical specifications, which feed the creation of a new set of requirements that can be enforced through review, executable formal specification, and formal proof depending on their complexity. This V&V approach is employed using Frama-C on a toy optimisation problem, and at MIL level in MATLAB/Simulink on a set of benchmark problems (actuator allocation, rendezvous in GEO). Until the conclusion of the project, the MIL-SIL-PIL test campaign pipeline will be fully executed further demonstrating the combined enhancements of the V&V process and optimisation-based software.

## ACKNOWLEDGEMENTS

The work presented in this article was carried out under, and funded by, the European Space Agency's Technology Development Element (TDE) programme (ESA contract No. 4000136721/21/NL/CRS "Verification and validation of real-time optimised safety-critical GNC SW systems"). The views expressed herein can in no way be taken to reflect the official opinion of the European Space Agency.

## REFERENCES

[1] J. Nocedal and S. Wright, *Numerical Optimization* (Springer Series in Operations Research and Financial Engineering), Second. Springer, 2006.

[2] X. Liu, P. Lu, and B. Pan, "Survey of convex optimization for aerospace applications," *Astrodynamics*, vol. 1, no. 1, pp. 23–40, Sep. 2017.

[3] N. Paulino, C. Roche, L. Ferreira, *et al.*, "Fault Tolerant Control for a Cluster of Rocket Engines – Methods and outcomes for guidance and control recovery strategies in launchers," in *Proceedings of the 12th International ESA Conference on Guidance, Navigation & Control Systems*, Sopot, Poland: ESA, Jun. 2023.

[4] P. Colmenarejo, J. Branco, N. Santos, *et al.*, "Methods and outcomes of the COMRADE project - Design of robust Combined control for robotic spacecraft and manipulator in servicing missions: Comparison between between Hinf and nonlinear Lyapunov-based approaches," in *69th International Astronautical Congress (IAC)*, vol. IAC-18-F1.2.3, Bremen, Germany, Oct. 2018.

[5] T. A. Johansen and T. I. Fossen, "Control allocation—A survey," *Automatica*, vol. 49, no. 5, pp. 1087–1103, May 2013.

[6] V. Preda, A. Hyslop, and S. Bennani, "Optimal science-time reorientation policy for the Comet Interceptor flyby via sequential convex programming," *CEAS Space Journal*, Jun. 2021.

[7] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[8] J. Signoles, N. Kosmatov, and K. Vorobyov, "E-ACSL, a Runtime Verification Tool for Safety and Security of C Programs (tool paper)," in *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*, Kalpa Publications in Computing, Sep. 2017, pp. 164–173.

[9] P. Baudin, F. Bobot, D. Bühler, *et al.*, "The dogged pursuit of bug-free C programs: The Frama-C software analysis platform," *Communications of the ACM*, vol. 64, no. 8, pp. 56–68, Aug. 2021.

[10] Y. Yu, P. Elango, U. Topcu, and B. Açıkmeşe, "Proportional-Integral Projected Gradient Method for Conic Optimization," *Automatica*, vol. 142, p. 110 359, Aug. 2022.

[11] W. Fehse, *Automated Rendezvous and Docking of Spacecraft*. Cambridge University Press, Nov. 2003.

[12] ECSS, "Satellite attitude and orbit control system (AOCS) requirements," European Cooperation for Space Standardization, Standard ECSS-E-ST-60-30C, Aug. 2013.

[13] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, pp. 576–580, Oct. 1969.

[14] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[15] J. Gerlach, "ACSL by example. v22.0.0," Fraunhofer, Tech. Rep., 2021, p. 291.

[16] L. Creuse, J. Huguet, C. Garion, and J. Hugues, "SPARK by example: An introduction to formal verification through the standard c++ library," *Ada Lett.*, vol. 38, no. 2, pp. 89–96, Dec. 2019.

[17] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337–340.

[18] H. Barbosa, C. W. Barrett, M. Brain, *et al.*, "cvc5: A versatile and industrial-strength SMT solver," in *TACAS'22*, ser. LNCS, vol. 13243, 2022, pp. 415–442.

[19] S. Conchon, A. Coquereau, M. Iguernlala, and A. Mebsout, "Alt-Ergo 2.2," in *SMT Workshop: International Workshop on Satisfiability Modulo Theories*, Oxford, United Kingdom, Jul. 2018.

[20] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *POPL '77*, ACM, 1977, pp. 238–252.

[21] R. Cohen, E. Feron, and P. Garoche, "Verification and validation of convex optimization algorithms for model predictive control," *CoRR*, vol. abs/2005.12588, 2020.

[22] G. Davy, E. Feron, P. Garoche, and D. Henrion, "Experiments in verification of linear model predictive control: Automatic generation and formal verification of an interior point method algorithm," in *LPAR-22.*, G. Barthe, G. Sutcliffe, and M. Veanes, Eds., vol. 57, 2018, pp. 290–306.

[23] W. H. Clohessy and R. S. Wiltshire, "Terminal guidance system for satellite rendezvous," *Journal of the Aerospace Sciences*, vol. 27, no. 9, pp. 653–658, Sep. 1960.