DOI: ADD DOINUMBER HERE

Design Simulation and Software Validation for On-board and Real-Time Optimal Guidance

Carlo Alberto Pascucci[⋆] *and Alvaro Fernandez*[⋆] *and Juan Jerez*[⋆]
and Samir Bennani[†]
and Jean-Philippe Preaud[‡] and Jorgen Bru[‡]
[⋆]*embotech AG*
Technoparkstrasse 1, CH-8005 Zurich, Switzerland
[†]*ESA ESTEC*
Keplerlaan 1, NL-2200 AG Noordwijk, The Netherlands
[‡]*ESA HQ Daumesnil*
52 rue Jacques Hillairet, 75012 Paris, France
pascucci@embotech.com · fernandez@embotech.com · jerez@embotech.com
samir.bennani@esa.int
jean-philippe.preaud@esa.int · jorgen.bru@esa.int
[†]Carlo Alberto Pascucci

Abstract

Current and future trends for launch vehicles include increasingly stringent requirements for mission responsiveness and adaptability. In a previous FLPP activity we developed a tool to analyze state-of-the-art approaches for on-board optimized guidance. FORCES Pro was used in the backend to generate fast embeddable optimization solvers. This paper describes the implementation of the simulation tools, the guidance and control algorithms, and the initial steps taken to qualify the flight software for demonstrator vehicles currently under development by FLPP and INCAS. Our software has been used to simulate thousands of end-to-end scenarios involving nominal and off-nominal conditions, different levels of dispersion, mid-flight target changes and partial engine failures. All missions in the test plan completed successfully with adequate landing states and constraint satisfaction throughout the entire flight.

1. Introduction

To increase the science return of space missions, and improve the access to space, launchers and spacecrafts must satisfy increasingly stringent requirements for mission responsiveness. The vehicles must be able to adapt to failures, abort scenarios, modification of the target orbit, or varying orbital pointing constraints among others. In addition, reusable launch vehicles require pin-point landing capabilities and similar adaptability to partial failures and modifications of the landing target. To date, however, each launch must be carefully planned with large advance, accounting for wide tolerances on all engineering aspects; moreover, launch windows are heavily constrained by the actual weather that can be known only a few hours before. Recent advances on algorithms and embedded processing power are changing this landscape thanks to what is known in literature as *computational guidance and control*.[10] In the last few years private space companies like SpaceX[3] and Blue Origin[1] are demonstrating the feasibility and the advantages of reusable launchers compared to expendable vehicles. One of the key technologies that is enabling this paradigm shift is numerical optimization.[6] To advance, also in Europe, the use of on-board and real-time optimization for space application, in a previous FLPP activity we developed a tool to analyze various state-of-the-art approaches for the powered descent phase and on-orbit reorientation. The results demonstrated the feasibility and the potential of both nonlinear programming formulations and convexification approaches for real-time implementation. In both scenarios, FORCES Pro was used in the back-end to generate fast embeddable optimization solvers. In parallel, FLPP and INCAS are developing several re-usable flying testbeds for maturing endo-atmospheric ascent, fly-back strategies, powered descent, and pinpoint landing technologies. This paper describes the implementation of the simulation tools, the guidance and control algorithms, and the initial steps taken to qualify the flight software for these demonstrator vehicles. In this work the optimal trajectory generation algorithms have been extended to handle both the ascent and descent phases of a mission. The entire GNC is coded in C and implemented in real-time Linux running on the

target flight computer (Aries PC104 featuring an Intel Atom E3845 CPU at 1.91 GHz with 4 GB of memory). A synchronous task with the highest priority executes the position and attitude control, state estimation and the mission and vehicle management logic. A separate thread runs an asynchronous and lower priority task that generates a new flight trajectory on-board. This is triggered whenever the vehicle drifts too far from the current trajectory due to uncertainty in actual flight conditions, or whenever there is a change in flight phase (e.g. ascent and descent). On-board trajectory re-generation can also be triggered if there is a change in target command or a detected failure in the vehicle. A high-fidelity simulator used for functional validation has been deployed on a Speedgoat standalone computer to build a real-time vehicle simulator. This is connected to the flight computer to form a facility for validating the real-time software. Our software has been used to simulate thousands of end-to-end scenarios involving nominal and off-nominal conditions, different levels of dispersions, mid-flight target changes and partial engine failures. All missions in the test plan completed successfully with adequate landing states and constraint satisfaction throughout the entire flight.

This paper is organized as follows. In Section 2, we detail the general mission profile that was the focus of the current study. In Section 3, the developed guidance and control architecture is delineated. Section 4 and 5, we specify the vehicle modeling and the mathematical formulation of the guidance optimization problem, respectively. In Section 6, the developed software framework is presented, while, in Section 7, we detail the testing framework and analyze the test results. Finally some conclusions are drawn in Section 8.

2. General Mission Profile

The vehicle considered in this project is called *Demonstrator for Technology Validation* (DTV) and it is currently under development by INCAS in the framework of FLPP3. It is a Vertical Take Off and Landing (VTOL) aircraft that can be powered by one or three jet engines. However, in this paper we will focus on the single engine configuration. A CAD rendering of the two vehicles is shown in Figure 1. This vehicle was chosen because being powered by a jet

Figure 1: The DTV vehicle: 61.6 [Kg] single engine or 126.62 [Kg] triple engine configuration, 1.2 [m] total height. Courtesy of INCAS.

engine makes it easier to operate and allows for a quicker turnaround between flight tests. Nevertheless, its dynamics shares many similarities with the ones of small test rockets. Two notable examples are the “Grasshopper"[14] and the “Xombie".[15] The first one was used in early experiments for the development SpaceX’s rocket landing GNC, while the second one flew for the ADAPT project.[13] The Xombie was built by Masten Space Systems and was instrumental in testing the G-FOLD algorithm and the terrain relative navigation system that will be used on NASA’s Mars 2020 mission.

One of the objectives of this project is to simulate flight missions of increasing complexity to gradually evaluate performance and assess the reliability of the proposed G&C architecture. The typical mission sequence is depicted in Figure 2 and it comprises four main flight phases:

1. vertical takeoff, to clear the launch pad;
2. ascent, to set the vehicle to the desired state (e.g. altitude, velocity, etc.) for the powered descent;
3. descent, to steer the VTOL craft above the target position;
4. vertical landing, to touchdown on the desired spot.

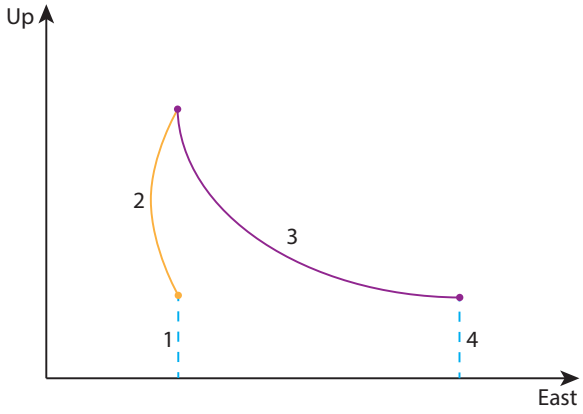


Figure 2: Typical mission profile, in-plane example. We will denote time at takeoff as t_0 and at the end of each phase as t_1, t_2, t_3, and t_4, respectively.

Phases 1 and 4 are managed by a specialized guidance scheme for takeoff and landing which does not involve on-line optimization. The trajectories for the ascent and the descent phases 2 and 3 instead are computed by real-time optimal guidance. All trajectories are then tracked by the same low-level position and attitude controller. Regardless the specific mission scenario, the objective of the optimal guidance is to minimize fuel consumption during flight, while accounting for vehicle's limitations and flight path constraints. However, if there is not enough fuel to reach the desired landing pad, the objective switches to minimizing the landing error. This means that instead of aiming at a specific point in space, the optimal guidance will compute a trajectory to steer the vehicle as close as possible to the desired landing target with the available fuel.

3. Guidance and Control Architecture

Besides vehicle and path constraints, sensors noise and model-to-plant mismatches, the proposed guidance and control system needs to be able to handle unpredictable wind gusts, partial faults in the propulsion system and mid-flight changes in landing target. To this end the developed G&C architecture can be decomposed into three main blocks: Mission and Vehicle management (MVM); Guidance and Control. Their interconnection is shown in Figure 3. In this work we did not address the Navigation component in the GNC scheme. Therefore, inputs to the G&C block are the filtered vehicle's state and *configuration* files to set the initialization parameters and the mission profile. The outputs include the control commands to steer the vehicle towards the target, and the *flight log* files to store the GNC data for post flight analysis.

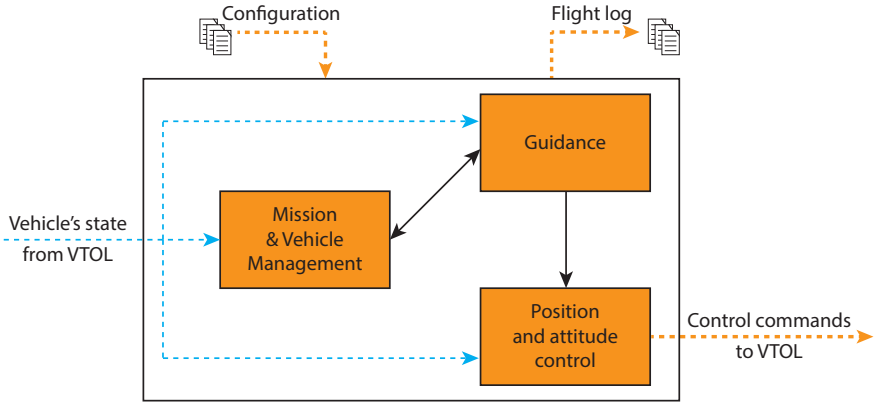


Figure 3: GNC Block Diagram

3.1 Mission and Vehicle Management

One of the design objectives for this project is to develop a single software solution to handle various mission scenarios. In this regard the MVM has a crucial role as it is in charge of ensuring the correct mission execution. The two main functions are a flight mode selector (e.g. idle, takeoff, optimal, landing), modeled as a state machine, and a target manager that handles exceptions (e.g. tracking errors, faults, retargeting) during the optimal guidance phase and changes the guidance targets.

The available flight modes are: engine off (mode -1); engine startup (mode 0); takeoff guidance (mode 1); optimal guidance (mode 2); landing guidance (mode 3); idle (mode 4). Their interconnection and the signals triggering state transitions are depicted in Figure 4. The vehicle starts always from *mode -1* with the engine off. The start signal

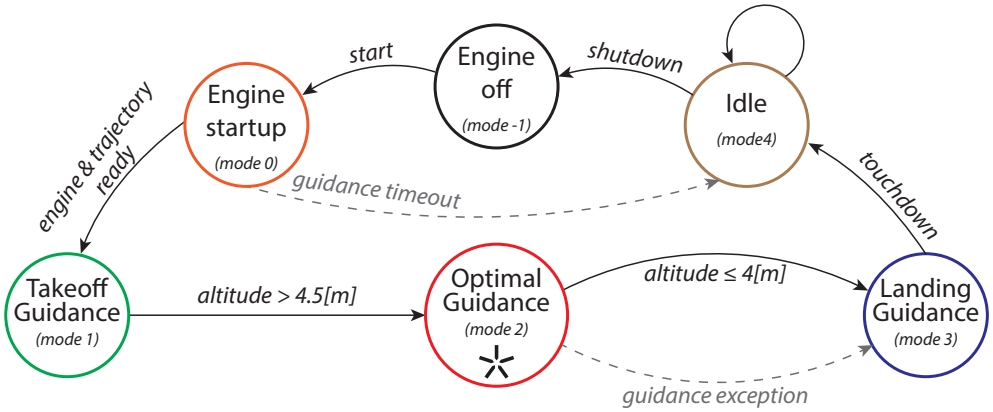


Figure 4: Mode transitions operated by the flight mode selector

kicks off the beginning of a mission with the engine entering its startup phase. When in *mode 0*, while the engine warms up, the optimal solver computes the first trajectory towards the ascent target. When the engine is ready and an optimal trajectory is available, the vehicle is cleared for takeoff, *mode 1*. After a vertical ascent to a predefined altitude (e.g. 4.5$[m]$), the flight mode selector switches to *mode 2* where any new trajectory is generated by solving an optimal guidance problem. At the end of the descent phase, when the vehicle crosses a predefined altitude (e.g. 4$[m]$), the landing guidance (*mode 3*) is engaged. At this point the touchdown signal triggers the switch to *mode 4* where the vehicle is in idle (on the landing pad with the engine throttled to its minimum). Finally, after a predefined amount of time or user input, the shutdown signal switches off the engine. As a safety precaution, the first optimal trajectory is computed during the warm up phase. If after a predefined amount of time a trajectory is not available, the vehicle goes directly into idle rather than starting to takeoff. On top of that, while in *mode 2*, if the optimal guidance returns an exception code at any point, an emergency landing is triggered and the landing guidance steers the vehicle to the closest abort pad available.

The other major MVM component is the target manager. In nominal conditions it selects the next target from the list created during the mission definition and it calls the optimal solver to compute a new trajectory when needed. It is worth noting that to account for the time it takes to compute a new trajectory, the vehicle's state used by the optimal path planner is a prediction of the aircraft's position at the time the next trajectory will be actually flown. In an off-nominal situation, when, due to wind or other disturbances, the vehicle deviates too much from the prescribed path, a new trajectory for the same target will be requested. This improves landing accuracy and reduces fuel consumption as it avoids excessive corrective actions from the controller. In case of a partial fault in the propulsion system a new trajectory will be computed accounting for the current vehicle's specifications. Finally, if at a certain point during the mission there is a change in the desired landing target, the optimal guidance will update the flight path to reach the new goal.

3.2 Guidance

As mentioned earlier we employ three guidance modes. The takeoff guidance to clear the launchpad is very simple: it generates a constant vertical velocity reference (e.g. 3$[m/s]$) with the lateral velocity set to zero and the horizontal position matching the takeoff pad. The optimal guidance uses the inputs from the MVM to call FORCES Pro and generate a trajectory. More details about this process will be given in Section 5 where we describe the optimal problem

formulation. The landing guidance sets a constant reference in position that corresponds to the landing pad, while enforcing a predefined vertical landing speed (e.g. $0.5[m/s]$). In case the aircraft has not enough fuel to reach the desired landing pad, instead of generating a constant reference to match a specific position, the landing guidance will only zero out the remaining lateral velocity.

3.3 Control

The position and attitude feedback controllers are based on an LQR design augmented by tunable feed forward terms to help compensate for atmospheric and ram drag. The error between the actual and the desired position defines the reference for the position controller. The resulting force vector is then used to compute the desired angles. The attitude error is the input that the attitude controller uses to compute the desired torques. By means of a static allocation matrix, the desired forces and torques are then mapped into commands for the propulsion system, a jet engine in this case.

4. Vehicle Modeling

4.1 Notation And Reference Frames

Here we introduce the notation and the reference frames that will be used in the following. Given the short range and flight duration of the mission scenarios envisioned for this activity, it is safe to assume a flat and non-rotating Earth model. Therefore we define the inertially-fixed frame $\mathcal{I}$ with its origin located at the launch site, where the “Up, East, North" convention is used. The body-fixed frame $\mathcal{B}$ instead is placed at the Center of Mass (CoM) with its x axis along the vehicle’s vertical axis, that is parallel to the thrust vector when there is no deflection on the control vanes. When there is no rotation between the two frames, the body frame is oriented with respect to the inertial frame such that the vehicle’s x, y, and z axes are aligned with the Up, East, and North axes, respectively. Inertial and body frames are depicted in Figure 5. The state of the vehicle is defined as $\mathbb{X} = [m,\ \mathbf{r}_{\mathcal{I}},\ \dot{\mathbf{r}}_{\mathcal{I}},\ \mathbf{q}_{\mathcal{B}\to\mathcal{I}},\ \omega_{\mathcal{B}}]^\top \in \mathbb{R}^{14}$, where the dependence on time t is omitted. m represents the mass, $\mathbf{r}_{\mathcal{I}}$, and $\dot{\mathbf{r}}_{\mathcal{I}}$ are the position and velocity vectors expressed in the inertial frame, respectively, $\mathbf{q}_{\mathcal{B}\to\mathcal{I}}$ is a unit quaternion representing the rotation of the body frame relative to the inertial frame (i.e. vehicle’s attitude), and $\boldsymbol{\omega}_{\mathcal{B}}$ is the angular velocity vector in the body frame. We denote by $\mathbf{e}_i$ a unit vector pointing along the the i^{th} axis. Unless differently specified, $\|\cdot\|$ represents the two-norm, and SI units will be used throughout the document.

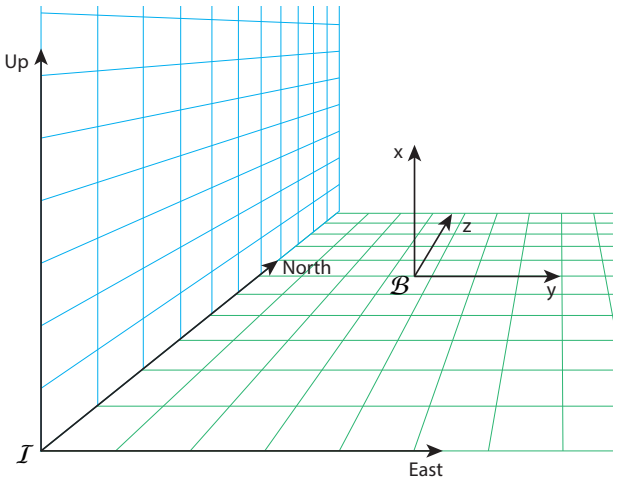


Figure 5: Inertial $\mathcal{I}$ and body $\mathcal{B}$ frames

4.2 Equations of Motion

Thanks to its mechanical characteristics (compact body and landing gear structure) we can model the targeted vehicle as a rigid body. The motion is determined by the forces and torques due to aerodynamics, gravity and the propulsion system. The model has a translational and a rotational component that can be taken into account jointly or separately. For guidance and control design, we refer to 3-DoF (three degrees of freedom) dynamics when considering only the

equations of motion (EOM) concerning the translational part, while by 6-DoF (six degrees of freedom) dynamics we imply that also the rotational part is included.

The mass evolution is considered to be proportional to the thrust magnitude generated by the engine as:

$$\dot{m}(t) = -\alpha \|\mathbf{T}_{\mathcal{B}}(t)\| \tag{1}$$

where $\mathbf{T}_{\mathcal{B}}(t)$ is the resultant thrust vector and α $[s/m]$ is a mass depletion proportionality constant. Following Newton's second law, the vehicle's acceleration expressed in the inertial frame is defined as:

$$\ddot{\mathbf{r}}_{\mathcal{I}}(t) = \frac{1}{m(t)}\mathbf{F}_{\mathcal{I}}(t) + \mathbf{g}_{\mathcal{I}} \tag{2}$$

where $\mathbf{F}_{\mathcal{I}}$ represents the combined action of the forces acting on the vehicle due to the propulsion system and the aerodynamics, and $\mathbf{g}_{\mathcal{I}}$ is the gravitational acceleration. The expected maximum altitude that can be reached by the DTV is 3500 $[m]$, hence it is safe to assume a constant gravitational field $\mathbf{g}_{\mathcal{I}} = [-g_0\ 0\ 0]^\top$, where $g_0 = 9.81$ $[m/s^2]$ is the average gravitational acceleration at sea level on Earth. Equation (2) represents the 3-DoF model. Velocity and position are obtained by single and double integration, respectively and, in this case, the thrust vector can also be used as a proxy for the vehicle's orientation.

Although Euler angles are a very intuitive way to represent rotations and involve only three parameters, this simplification can lead to a phenomenon known as "gimbal lock" resulting in the loss of a degree of freedom. Hence, here we parameterize the rotation of the body frame relative to the inertial frame as a unit quaternion. This representation increments the parameters to four, however it prevents the numerical issues associated with Euler angles. The attitude of the vehicle is then defined as:

$$\mathbf{q}_{\mathcal{B}\to\mathcal{I}} \triangleq \begin{bmatrix}\cos(\xi/2)\\ \sin(\xi/2)\hat{\mathbf{n}}\end{bmatrix} = \begin{bmatrix}q_1\\ q_2\\ q_3\\ q_4\end{bmatrix} \tag{3}$$

where the "scalar first" convention is used. ξ is a placeholder for a rotation angle and $\hat{\mathbf{n}}$ is the unit vector around which the rotation is performed. We can now define the Direction Cosine Matrix (DCM) that encodes the transformation of a vector from the inertial to the body frame as:

$$C_{\mathcal{I}\to\mathcal{B}}(t) = \begin{bmatrix}1-2(q_3^2+q_4^2) & 2(q_2q_3+q_1q_4) & 2(q_2q_4-q_1q_3)\\ 2(q_2q_3-q_1q_4) & 1-2(q_2^2+q_4^2) & 2(q_3q_4+q_1q_2)\\ 2(q_2q_4+q_1q_3) & 2(q_3q_4-q_1q_2) & 1-2(q_2^2+q_3^2)\end{bmatrix} \tag{4}$$

with its inverse transformation expressed as:

$$C_{\mathcal{B}\to\mathcal{I}}(t) \triangleq C^{-1}_{\mathcal{I}\to\mathcal{B}}(t) = C^{\top}_{\mathcal{I}\to\mathcal{B}}(t) \tag{5}$$

The quaternion kinematics and the attitude dynamics can now be defined as:

$$\dot{\mathbf{q}}_{\mathcal{B}\to\mathcal{I}}(t) = \frac{1}{2}\Omega(\boldsymbol{\omega}_{\mathcal{B}}(t))\mathbf{q}_{\mathcal{B}\to\mathcal{I}}(t) \tag{6a}$$

$$\dot{\boldsymbol{\omega}}_{\mathcal{B}}(t) = J^{-1}_{\mathcal{B}}(t)\Big(\mathbf{M}_{\mathcal{B}}(t) - [\boldsymbol{\omega}_{\mathcal{B}}(t)\times]J_{\mathcal{B}}(t)\boldsymbol{\omega}_{\mathcal{B}}(t)\Big) \tag{6b}$$

where $J_{\mathcal{B}}(t)$ is the inertia tensor, and $\mathbf{M}_{\mathcal{B}}(t)$ represents the combined action of all torques acting on the vehicle expressed in the body frame. $\Omega(\boldsymbol{\xi})$ and $[\boldsymbol{\xi}\times]$ are two skew-symmetric matrices defined for a generic vector $\boldsymbol{\xi} \in \mathbb{R}^3$ as:

$$\Omega(\boldsymbol{\xi}) = \begin{bmatrix}0 & -\xi_1 & -\xi_2 & -\xi_3\\ \xi_1 & 0 & \xi_3 & -\xi_2\\ \xi_2 & -\xi_3 & 0 & \xi_1\\ \xi_3 & \xi_2 & -\xi_1 & 0\end{bmatrix}, \quad [\boldsymbol{\xi}\times] = \begin{bmatrix}0 & -\xi_3 & \xi_2\\ \xi_3 & 0 & -\xi_1\\ -\xi_2 & \xi_1 & 0\end{bmatrix}.$$

Equation (2) together with equations (6) completely define the EOM for the 6-DoF rigid body dynamics.

As mentioned above, forces and torques acting on the body frame are the sum of the effects generated by the propulsion system and the aerodynamics, other than from the constant gravitational field. Specifically:

$$\mathbf{F}_{\mathcal{I}}(t) = C_{\mathcal{B}\to\mathcal{I}}\Big(\mathbf{T}_{\mathcal{B}}(t) + \mathbf{A}_{\mathcal{B}}(t)\Big) \tag{7}$$

$$\mathbf{M}_{\mathcal{B}}(t) = \underbrace{[\ell_T(t)\times]\mathbf{T}_{\mathcal{B}}(t)}_{\mathbf{M}_{\mathbf{T}_{\mathcal{B}}}(t)} + \underbrace{[\ell_P(t)\times]\mathbf{A}_{\mathcal{B}}(t)}_{\mathbf{M}_{\mathbf{A}_{\mathcal{B}}}(t)} \tag{8}$$

where $\mathbf{T}_{\mathcal{B}}(t)$ is the resultant thrust vector generated by the engine, and $\mathbf{A}_{\mathcal{B}}(t)$ is the sum of the aerodynamic forces acting on the vehicle. ℓ_T represents the distance between the CoM and the hinge point of the control vanes, ℓ_P is the lever arm between the CoM and the Center of Pressure (CoP), while $\mathbf{M}_{\mathbf{T}\mathcal{B}}(t)$ and $\mathbf{M}_{\mathbf{A}\mathcal{B}}(t)$ represent the torques around the CoM due to the propulsion system and the aerodynamic effects, respectively.

The two aerodynamics effects that will be considered here are the “atmospheric drag" and the “ram drag". While the first one is related to the geometry of the vehicle, the second one is an effect introduced by the interaction of the jet engine’s air intake with the crosswind. These forces are modeled as:

$$\mathbf{D}_{\mathcal{B}}(t) := -\frac{1}{2}\rho C_D S_D \|\mathbf{v}_{\mathcal{B}}(t)\|\mathbf{v}_{\mathcal{B}}(t) \tag{9a}$$

$$\mathbf{R}_{\mathcal{B}}(t) := -\begin{bmatrix} 0 \\ \rho A_e v_e(t)[\mathbf{e}_2\ \mathbf{e}_3] \cdot \mathbf{v}_{\mathcal{B}}(t) \end{bmatrix} \tag{9b}$$

$$\mathbf{A}_{\mathcal{B}}(t) := \mathbf{D}_{\mathcal{B}}(t) + \mathbf{R}_{\mathcal{B}}(t) \tag{9c}$$

where ρ is the air density, C_D is the drag coefficient, S_D is the reference area, A_e is the air intake area, v_e is the exit speed of the airflow from the engine, and $\mathbf{v}_{\mathcal{B}} = C_{\mathcal{I}\to\mathcal{B}}\,\dot{\mathbf{r}}_{\mathcal{I}} - \mathbf{w}_{\mathcal{B}}$ is the vehicle’s velocity relative to the wind speed $\mathbf{w}_{\mathcal{B}}$ in the body frame.

4.2.1 Simplifications and Assumptions

While in simulation the focus is on the realism of the modeled environment and vehicle (i.e. the Earth at low altitude and a jet powered VTOL aircraft), for the guidance and control design the target is to use models that are representative enough to ensure satisfactory flight performance, while keeping the problem formulation computationally tractable. To this end, we can consider a reduced set of equations of motion, and simplify the models for the forces acting on the vehicle.

To what concerns the first point, the 6-DoF model is more representative of the actual motion of the vehicle, but this comes with a significant increase in the complexity of the optimal guidance problem. On the other hand a 3-DoF formulation, being a coarser representation, may generate trajectories that require additional corrective actions from the low-level controller to stay on the computed path, hence increasing the fuel consumption. To what concerns the second point, the simplifications and assumptions that can be made when modeling the forces acting on the vehicle are listed below.

Propulsion system: The small perturbation torque due to the rotation of the jet engine’s internal components is not considered in the guidance design as it can be compensated by the low level controllers. The same applies for noise and offsets in the generated thrust vector, as they are assumed to be relatively small compared to the control commands.

Aerodynamic effects: Due to the relatively low altitude and speed range at which the flights are planned, the guidance algorithm will not consider atmospheric nor ram drag. However, jet engine’s performance losses due to the altitude will be accounted for in the problem formulation. Other aerodynamic forces like lift can also be neglected due to the non-lifting shape of the body and the low speed at which the vehicle will fly.

4.2.2 State and control constraints

The guidance system must ensure that the computed trajectory does not require more fuel than available, hence by denoting as m_{dry} the dry mass, the first constraint can be written as:

$$m(t) \geq m_{dry} \tag{10}$$

Defining the tilt angle as the angle between the x-axis in the body frame and the “Up"-axis in the inertial frame, we can express a tilt angle constraint as:

$$\mathbf{e}_1 \cdot C_{\mathcal{B}\to\mathcal{I}}(t)\mathbf{e}_1 \geq \cos(\theta_{max}) \tag{11}$$

where θ_{max} is the maximum allowed tilt angle. Avoiding large tilting angles increments operational safety (especially close to the ground) and helps to mimic the limitations on attitude that would arise from having a terrain relative navigation system or other sensors that require pointing to the ground at all times during flight. In the 3-DoF case this constraint cannot be directly enforced since the model does not consider attitude. Instead, we limit the resultant thrust vector’s tilt angle. To prevent excessive rotational speed, the angular rates can be constrained as:

$$\|\boldsymbol{\omega}_{\mathcal{B}}(t)\| \leq \omega_{max} \tag{12}$$

where ω_{max} is the maximum allowed angular velocity. This cannot be enforced in the 3-DoF formulation. To avoid too shallow trajectories, that could result in hitting obstacles on the ground, we restrict the path of the vehicle to evolve on

the inside of a cone with the origin placed at the launchpad during ascent and at the landing pad during descent. In the literature, this is known as “glide-slope" constraint and can be expressed as:

$$\mathbf{e}_1 \cdot (\mathbf{r}_\mathcal{I}(t) - \mathbf{r}_\mathcal{I}(t_{0,4})) \geq \tan(\gamma)[\mathbf{e}_2\ \mathbf{e}_3] \cdot \|\mathbf{r}_\mathcal{I}(t) - \mathbf{r}_\mathcal{I}(t_{0,4})\| \tag{13}$$

where γ is the angle with the horizontal made by the cone boundary. Excessive structural stress on the body, can be avoided by limiting also the linear velocity as follows:

$$\|\dot{\mathbf{r}}_\mathcal{I}(t)\| \leq v_{max} \tag{14}$$

where v_{max} is the maximum allowed speed. To what concerns the control variables, there are limitations on the maximum and minimum thrust that the engine can generate and restrictions on the deflection of the control vanes. These are detailed by equations (15) and (16), respectively:

$$0 < T_{min} \leq \|\mathbf{T}_\mathcal{B}(t)\| \leq T_{max} \tag{15}$$

$$\mathbf{e}_1 \cdot \mathbf{T}_\mathcal{B}(t) \geq \cos(\delta_{max})\|\mathbf{T}_\mathcal{B}(t)\| \tag{16}$$

where T_{min} and T_{max} are the lower and upper bounds on each engine's thrust, and δ_{max} is the maximum allowed thrust deflection angle. In the DTV, the engine is fixed and $\delta(t)$ is the result of the combined action of the two control vanes bending the jet stream at the turbine's exit. The dynamics of the jet engine, the thrust losses due to the altitude and the drag induced by the control vanes cannot be neglected. These phenomena can be accounted for by implementing the constraints expressed by (17) and (18), respectively.

$$\dot{T}_{min} \leq \dot{\mathbf{T}}_\mathcal{B}(t) \leq \dot{T}_{max} \tag{17}$$

$$\mathbf{e}_1 \cdot \mathbf{T}_\mathcal{B}(t)(1 + \nu_D) \leq T_{max}(1 - \mathbf{e}_1 \cdot \mathbf{r}_\mathcal{I}(t)\beta) \tag{18}$$

where $\dot{T}_{min}$ and $\dot{T}_{max}$ are the minimum and maximum thrust rates, while β and ν_D are proportionality constants encoding the losses due to the altitude and the control vanes drag.

5. Optimal Guidance Problem Formulation

The optimal guidance described here builds on the one developed during our previous FLPP activity for the Constrained Powered Descent (CPD) and Constrained Attitude Guidance (CAG) scenarios. Detailed information about the CPD and CAG scenarios can be found in.[8] The flight of a rocket-like vehicle shows a coupling between the translational and rotational motion as the aircraft needs to tilt itself to move between two points in space. The rotational dynamics, however, evolves faster than the translational one. Hence, using a 3-DoF guidance is a viable option if there is a controller that acts hierarchically on a lower level, but at a faster pace, to handle the other degrees of freedom not accounted for at the guidance level. The main advantages of implementing a 3-DoF guidance are that the problem is easier to solve and it is possible to use a convex formulation with theoretical guarantees on solver's convergence as detailed in.[5] However, it is worth noting that when using the convex formulation, the problem's final time is fixed. Therefore a Time of Flight (ToF) search is necessary to find the the optimal time (that is also the one for which the lossless convexification holds) ensuring minimum fuel consumption. This search is time consuming and the analysis we performed showed that our nonlinear solver, not requiring a separate ToF search, was faster and equally reliable in computing the optimal trajectory. A way to avoid the ToF search in a convex formulation setting is to use a lookup table with precomputed ToFs and interpolate on that depending on the actual initial conditions[12] . This method significantly reduces the computational time for the 3-DoF convex guidance, however, in off-nominal conditions the precomputed ToF table would not be accurate anymore.

Other than the free final time, including aerodynamic effects and the rotational dynamics break the lossless convexification validity assumptions. As shown by recent research in [17] and [16] , the successive convexification technique can be used to address these issues. However, even though the single problems are approximated using a convex formulation, a stopping criteria for the iterative process, that depends on the specific problem, must be defined by the algorithm designer. Moreover, when the original problem is strongly nonlinear, the convex approximation pays a non negligible optimality price when compared to solving the nonlinear problem directly.

After evaluating the approaches described above, we observed that a non-linear 3-DoF guidance setting would provide the best combination of flexibility, quality of the solution and reliability for the missions being considered for DTV. Therefore, it has been selected as the default algorithm for our trajectory generation engine. For both ascent and descent phases, the 3-DoF nonlinear and free-final-time optimal guidance problem can be expressed as detailed in *Problem 1*

Problem 1 *: 3-DoF, Non-Convex, Continuous-Time, Free-Final-Time, Minimum-Fuel Problem*

Cost Function

$$\underset{t_f,\mathbf{T}_\mathcal{B}(\cdot)}{\text{minimize}} \quad \int_{t_0}^{t_f} \|\mathbf{T}_\mathcal{B}(t)\| + \lambda_1 \|\dot{\mathbf{T}}_\mathcal{B}(t)\|^2 + \lambda_2 \eta(t)^2 \, dt \tag{19}$$

subject to:

Boundary Conditions

$$m(t_0) = m_0 \tag{20}$$

$$\mathbf{r}_\mathcal{I}(t_0) = \mathbf{r}_{\mathcal{I},0} \qquad \|\mathbf{r}_\mathcal{I}(t_f) - \mathbf{r}_{\mathcal{I},f}\| \leq r_{tol} \tag{21}$$

$$\dot{\mathbf{r}}_\mathcal{I}(t_0) = \dot{\mathbf{r}}_{\mathcal{I},0} \qquad \|\dot{\mathbf{r}}_\mathcal{I}(t_f) - \dot{\mathbf{r}}_{\mathcal{I},f}\| \leq \dot{r}_{tol} \tag{22}$$

Point-mass Dynamics

$$\dot{m}(t) = -\alpha \|\mathbf{T}_\mathcal{B}(t)\| \tag{23}$$

$$\ddot{\mathbf{r}}_\mathcal{I}(t) = \frac{1}{m(t)} \mathbf{F}_\mathcal{I}(t) + \mathbf{g}_\mathcal{I} \tag{24}$$

State Constraints

$$\mathbf{e}_1 \cdot (\mathbf{r}_\mathcal{I}(t) - \mathbf{r}_\mathcal{I}(t_{0,f})) \geq \tan(\gamma) \, \| \, [\mathbf{e}_2 \; \mathbf{e}_3] \cdot (\mathbf{r}_\mathcal{I}(t) - \mathbf{r}_\mathcal{I}(t_{0,f})) \, \| \tag{25}$$

$$\|\dot{\mathbf{r}}_\mathcal{I}(t)\| \leq v_{max} + \eta(t) \tag{26}$$

$$\eta(t) \geq 0 \tag{27}$$

Control Constraints

$$0 \leq T_{min} \leq \|\mathbf{T}_\mathcal{B}(t)\| \leq T_{max} \tag{28}$$

$$\dot{T}_{min} \leq \dot{\mathbf{T}}_\mathcal{B}(t) \leq \dot{T}_{max} \tag{29}$$

$$\mathbf{e}_1 \cdot \mathbf{T}_\mathcal{B}(t) \geq \cos(\theta_{max}) \|\mathbf{T}_\mathcal{B}(t)\| \tag{30}$$

$$\mathbf{e}_1 \cdot \mathbf{T}_\mathcal{B}(t)(1 + \nu_D) \leq T_{max}(1 - \mathbf{e}_1 \cdot \mathbf{r}_\mathcal{I}(t)\beta) \tag{31}$$

where the subscripts $_0$ and $_f$ indicate initial and final values, respectively. From our previous work it emerged that small regularization terms have a significant impact on the convergence speed (up to three times faster), hence it is used also here. λ_1 and λ_2 are scalar weighting factors for the regularization term $\|\dot{\mathbf{T}}_\mathcal{B}(t)\|^2$ and the slack variable for the velocity constraint $\eta(t)$, respectively. The slack variable $\eta(t)$ is used to make the velocity constraint *soft*, allowing the solver to temporary violate that bound at an increase of the cost objective. Moreover, it prevents infeasibility when the initial speed is greater than or close to the maximum allowed one. As mentioned earlier, if during the descent phase there is not enough fuel to reach the desired target, the optimal problem is reformulated as *Minimum-Error* landing. The cost function (19) is substituted by

$$\underset{t_f,\mathbf{T}_\mathcal{B}(\cdot)}{\text{minimize}} \quad \int_{t_0}^{t_f} \lambda_1 \|\dot{\mathbf{T}}_\mathcal{B}(t)\|^2 + \lambda_2 \eta(t)^2 \, dt + \| \, [\mathbf{e}_2 \; \mathbf{e}_3] \cdot (\mathbf{r}_\mathcal{I}(t_f) - \mathbf{r}_{\mathcal{I},f}) \, \| \tag{32}$$

the final position constraint (21) is modified to account only for the altitude above the landing pad, the fuel limit (10) is explicitly included in the problem statement, and the glideslope constraint is removed.

The continuous problem is then discretized using a fourth order explicit Runge-Kutta integrator. The trajectory generated by the optimal guidance is a timestamped list of setpoints for the low-level controller. To reduce the computational burden on the CPU, the length of the list amounts to a few tens of elements. To provide a smoother reference to the low-level controller, the computed trajectory is resampled using linear interpolation. The steps of the interpolated trajectory will then match the sampling time of the low level controller.

6. Software Framework

One of the key outcomes of the current and past FLPP activities is to develop a unifying framework to design, evaluate and analyze performance of novel G&C algorithms. The software framework is based on a modular user interface,

trajectory generation and validation engines that can be extended depending on the mission scenario. The diagram in Figure 6 depicts the software architecture and its main components.

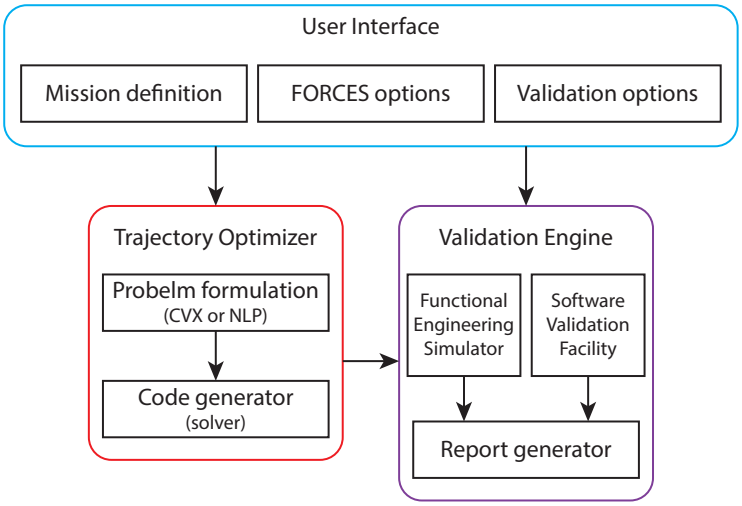


Figure 6: Architecture of the software framework

6.1 User Interface

This is a lightweight MATLAB-based text interface that allows the user to define the mission profile and modify several settings in the guidance algorithm. The interface allows also to configure the validation method by selecting the simulation environment and the report generation options.

6.2 Trajectory Generation Engine

This block is the core of our software. It is based on FORCES Pro[7] to create the customized code that generates the guidance trajectories by solving an optimization problem. Depending on the formulation, FORCES Pro can solve convex or nonlinear problems generating a highly customized solver tailored for deployment on resource-constrained embedded systems. The generated code uses only static memory allocation, it is MISRA-C compliant, and it does not depend on external libraries. As illustrated in Figure 7, the solver design follows a client/server architecture in which the user defines the optimization problem on its machine, and the server returns the solver that can be executed on a development PC for testing or deployed on the embedded hardware. Users can interface with FORCES Pro via MATLAB or Python. Alternatively, optimization problems can also be defined using the YALMIP modeling language.[9] In this work we used the MATLAB interface to obtain the solver's C code and wrap it into a custom Simuink S-Function for easy integration in our functional engineering simulator and software validation facility.

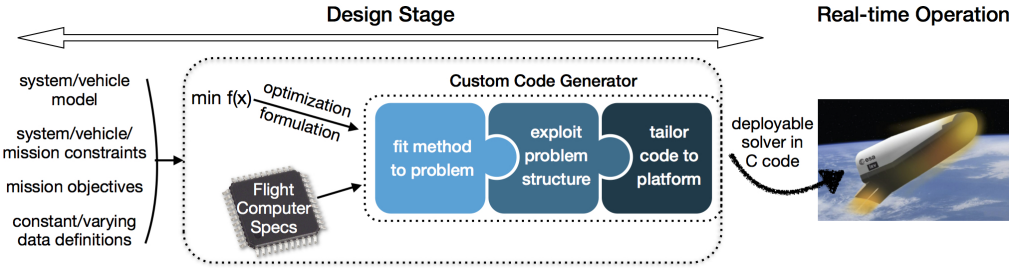


Figure 7: FORCES Pro design flow. A customized solver implemented in library-free C code is generated for every problem description.

6.3 Validation Engine

This component of the software framework provides both functional and software validation capabilities. Mission scenarios can be simulated in Simulink using a Functional Engineering Simulator (FES) with a model of the flight software in-the-loop, or on a Software Validation Facility (SVF) with a real-time simulator and a flight processor-in-the-loop setup. The simulation output is then processed to display plots and metrics relevant to the mission scenario and the software implementation.

6.3.1 Functional Engineering Simulator

The FES is built around Simulink in a modular and expandable fashion in order to support different vehicles and environment models, as well as multiple guidance and control architectures. On the base layer there are initialization classes defining the static properties that do not change between mission scenarios. As an example, the dry mass and the maximum wet mass are always constant for a given vehicle. From the user interface a different initialization class can be loaded to simulate a different vehicle or planet without having to make changes to the blocks. The layer above the initialization classes is populated by the custom subsystems that make embotech's space simulation library (ESSL). The library contains physical subsystems, software subsystems, axes transformations and utility blocks. The subsystems are arranged to simulate the G&C algorithms, the vehicle and the environment. Finally, the Simulink diagram at the top level links these models to build the closed loop simulation environment. At the end of a simulation the logged vehicle and the GNC status is saved for post flight analysis and displaying plots.

6.3.2 Software Validation Facility

The goal of the SVF is testing the flight software on a setup that is closer to the actual flight configuration. To this end, we utilized two different computers: one simulates, in real-time, the vehicle dynamics, the environment and the sensors while the flight software is executed on the other one. The two computers are connected in closed-loop in order to exchange the vehicle's state and the control commands. Additionally, a development PC is used for code generation and deployment, simulation control, logs recovery and data analysis.

The code deployed on both simulation computers is auto-generated from the same Simulink models that integrate the FES. As a result, any change to the FES is mirrored on the deployed code. The code generation and deployment of vehicle simulator and the flight software is completely automated. The switching between simulation platforms (FES and SVF) can be done through the user interface with one line of code.

Although the developed SVF is flexible to changes in the simulation computers and communication interface, the specific setup implemented in this project is depicted in Figure 8. The vehicle, environment and sensors are simulated on a dedicated Speedgoat[4] computer. The platform running the flight software is the On-Board Computer (OBC) used for the DTV. This OBC features an Atom E3845 CPU with 1.9$[GHz]$ maximum clock speed and has 4$[GB]$ RAM. Debian Linux with a PREEMPT_RT real-time kernel patch[11] is installed on the OBC. A serial RS232 interface implements the communication channel between the Speedgoat and the OBC. Both computers are connected via Ethernet to a development PC for data logging and post flight analysis.

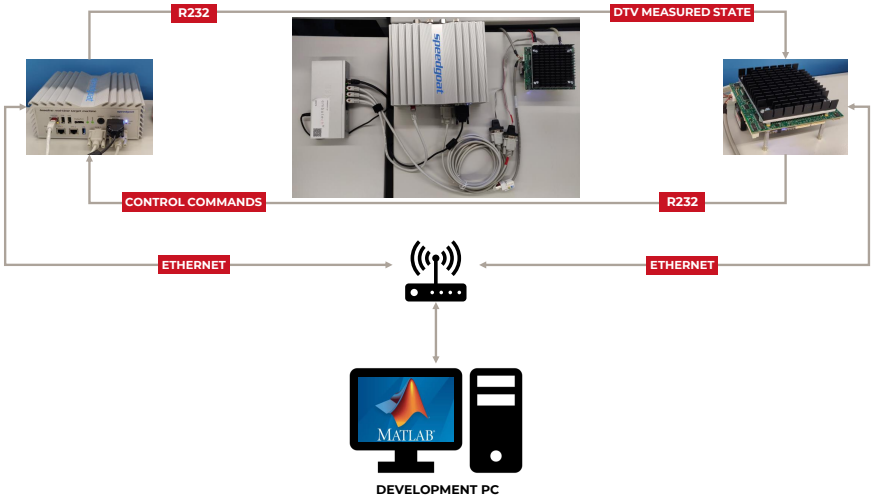


Figure 8: Setup for the Software Validation Facility

The flight software runs three different tasks in parallel. The one with the highest priority is the periodic G&C, containing he MVM, as well as the position and attitude controllers and lower level functions. The optimal solver is executed asynchronously on-demand as a lower priority task to avoid blocking other critical functions. Finally, a data logging task is run with the lowest priority in order to prevent any interference with the flight control threads. The static footprint of the generated flight software is 3.26$[MB]$, making it amenable to on-board implementation on a wide variety of embedded platforms. Simulating different scenarios does not requires code recompilation since both platforms load the mission parameters from a text file.

7. Testing Framework

In this project we adopted a Continuous Integration (CI) framework to constantly monitor the functionality of our evolving designs. Unit tests for components and the mission scenarios used for functional and software testing are implemented using MATLAB's Unit Test Framework. A Jenkins automation server runs all the tests every night. This allowed us to assess every day's changes to the code over a wide range of mission scenarios.

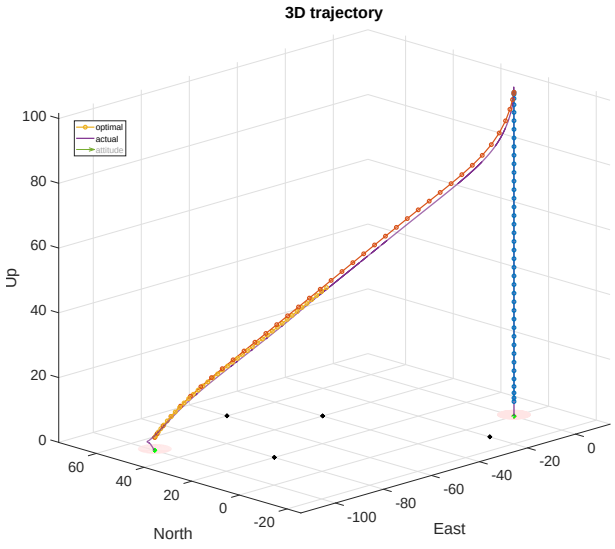
The goal of the functional test plan is to create a comprehensive sequence of tests. To this end, we define three main categories: *1D*, *3D*, and *Retargeting (RTG)*. 1D missions are the simplest kind. Here the vehicle is commanded to fly vertically up to a specified altitude with a certain velocity and then come straight down. No lateral movement is involved in these simulations. In a 3D mission the vehicle climbs vertically to a desired altitude with a certain velocity and then it is commanded to land at a specific position. A retargeting mission evolves similarly to a 3D one, however here there is a change of target while the vehicle is in the descent phase. The aircraft can be redirected to an alternative target or back to the launch site. All three scenario classes are tested in *nominal*, *robust*, or *fault* conditions. A nominal scenario assumes that dispersion and noise levels are within the expected range, while robust scenarios include higher than expected noise, dispersion and wind. A fault scenario is characterized by a mid-flight change in the performance of the propulsion system. That is a combination of an unexpected loss of power from the jet engine and reduced control vanes tilt range.

As an example, Figure 9 shows how the trajectory generation changes with different mission scenario options. The solid line represents the actual trajectory flown by the aircraft. The dotted lines are the optimal trajectories as computed in real-time by the solver, where different colors are used for different trajectories. The red, green and black dots on the ground are the takeoff, landing and emergency pads respectively. The glideslope constraint is depicted as transparent pink cones around the takeoff and landing pads. Figure 9a depicts the trajectory evolution for a nominal case in which the vehicle climbs vertically to 100$[m]$ with a landing target of $[Up,\ East,\ North]^\top = [0,\ -100,\ 50]^\top$. Figure 9b instead represents the trajectory evolution when a new landing target ($[Up,\ East,\ North]^\top = [0,\ -40,\ -40]^\top$) is commanded 25$[s]$ into the flight. New trajectories are computed and the vehicle deviates from its original path to match the retargeting goal. Figure 9c depicts the same scenario, but after the retargeting there is an engine fault. Specifically, at 33$[s]$ into the flight the engine's thrust range goes from $[T_{min},\ T_{max}]^\top = [130,\ 900]^\top[N]$ to $[T_{min},\ T_{max}]^\top = [300,\ 700]^\top[N]$, while the control vanes' deflection range goes from $[\hat{\delta}_{min},\ \hat{\delta}_{max}]^\top = [-30,\ 30]^\top[deg]$ to $[\hat{\delta}_{min},\ \hat{\delta}_{max}]^\top = [-10,\ 10]^\top[deg]$. Compared to Figure 9b an additional trajectory (green dotted line) is computed towards the end to account for the change in engine performance. A video of this simulation is available on our website.[2] Finally Figure 9d illustrates the guidance algorithm behavior when the wind blows during the ascent phase causing a tracking error. The wind gust starts at 2$[s]$ into the flight and lasts for two seconds. The speed and direction is 14$[m/s]$ towards the West. The wind causes the vehicle to drift from its straight ascent, hence the tracking error triggers a new trajectory computation. Because of the strength of the wind gust in this case a second corrective maneuver is needed to converge to the ascent target.

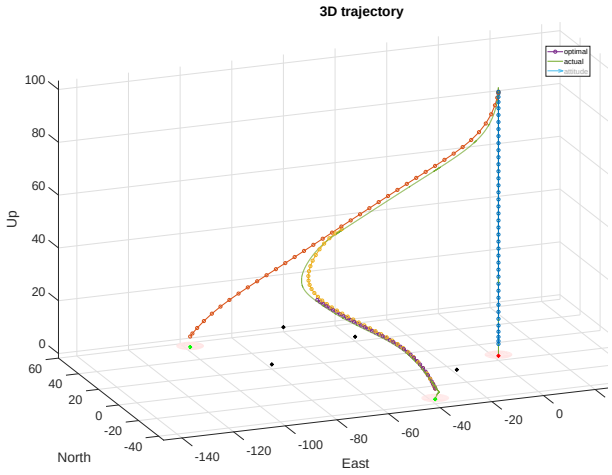
The optimal solver execution times on the FES and SVF for the mission scenario depicted in Figure 9d are reported Table 1. It can be noted that, on average, the embedded CPU is about ten times slower than the Intel Core i7 8550U at 1.8$[GHz]$ on which the FES was running during this test. Since they are both x86 CPUs and their clocks are comparable, one of the possible cause may be the lack of AVX instructions on the Atom E3845. Moreover in some sporadic cases the execution time on the OBC is in the order of two seconds. At the time of writing these effects, and techniques to potentially speedup the solver execution, are under investigation. However, of our guidance and control architecture is robust to longer execution times.

Table 1: Optimal solver execution times for the retargeting scenario with wind and engine fault

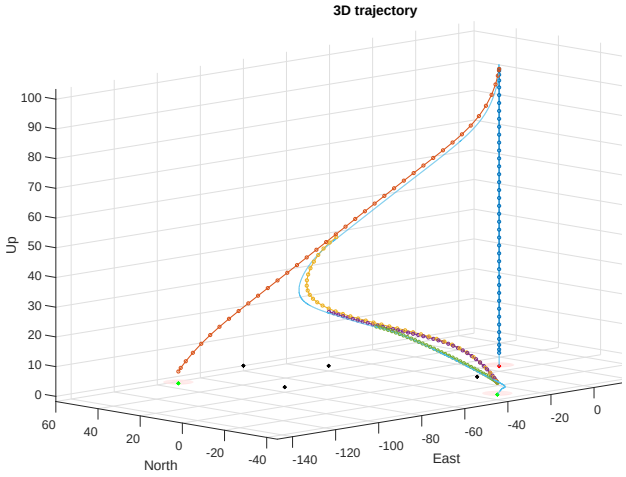
	Median [s]	Mean [s]	Maximum [s]	Minimum [s]
FES	0.07	0.07	0.15	0.04
SVF	0.59	0.77	2.53	0.34



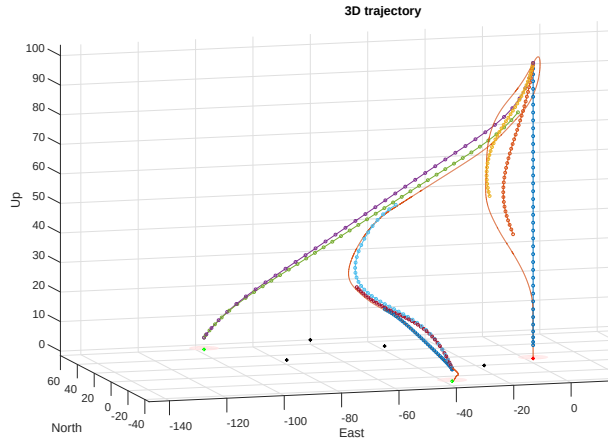
(a) Nominal 3D trajectory scenario



(b) Nominal retargeting scenario



(c) Retargeting scenario with engine fault



(d) Retargeting scenario with wind and engine fault

Figure 9: Mission scenario evolution

On Jenkins we perform the functional validation over 2576 mission scenarios. Specifically, 882 test cases for 1D missions, 1036 3D scenarios, and 658 retargeting cases. All of them divided in their nominal, robust and fault variants. Jenkins is used also to automate real-time tests on the SVF and equivalence tests between FES and SVF. The automation server provides a clean interface to check on the execution status and the final reports produced at the end of a batch of tests. The pass fail criteria for every test is based on the position error, the vertical velocity, the tilt angle and the angular rate norm at landing. If these values lie within a user defined range the test passes, otherwise it fails. In a nominal scenario for instance we want the landing error to be within $1[m]$, while the targeted touchdown speed is $0.5[m/s]$ with $0.1[m/s]$ tolerance. An example for a subset of 3D mission scenarios is shown in Figure 10.

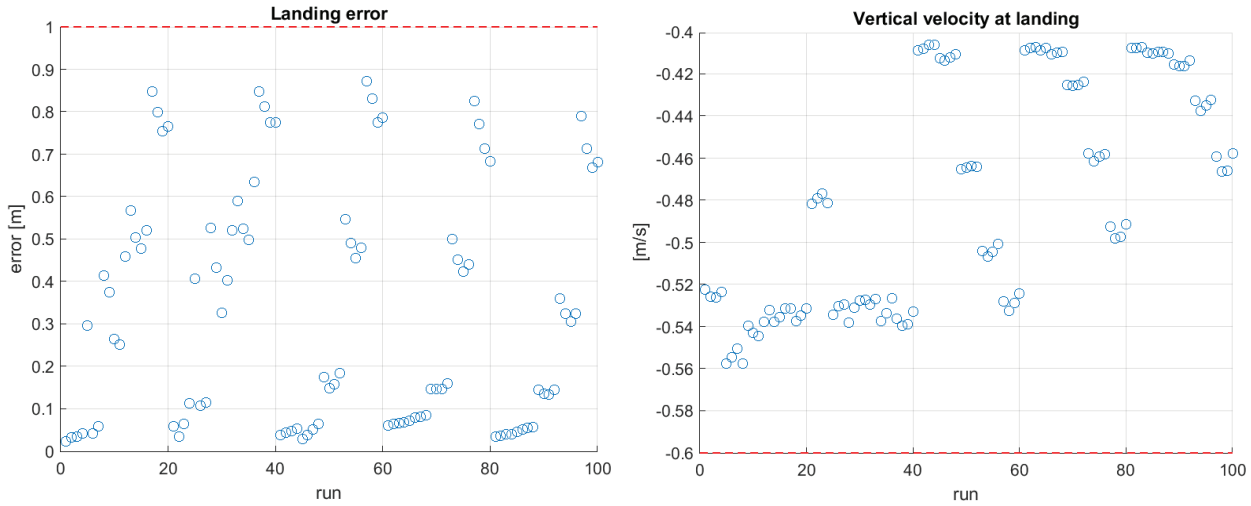


Figure 10: Some of the plots included in the tests summary report

7.1 Dispersion Analysis

Other than simulating a wide variety of mission scenarios, we can leverage on the same testing framework to perform sensitivity analysis. Here the focus is on how noise and plant mismatches affect our guidance and control architecture. With the G&C settings fixed we change atmospheric and ram drag coefficients, CoP location, propulsion system dynamics and sensor noise. Additional metrics considered in this tests are the impact of these dispersion in fuel consumption, optimal solver iterations and failures. As shown in Figure 11 for instance, increasing sensors noise has a negative impact on the landing accuracy and fuel consumption. However, none of the dispersion considered in this analysis had a negative effect the optimal solver reliability.

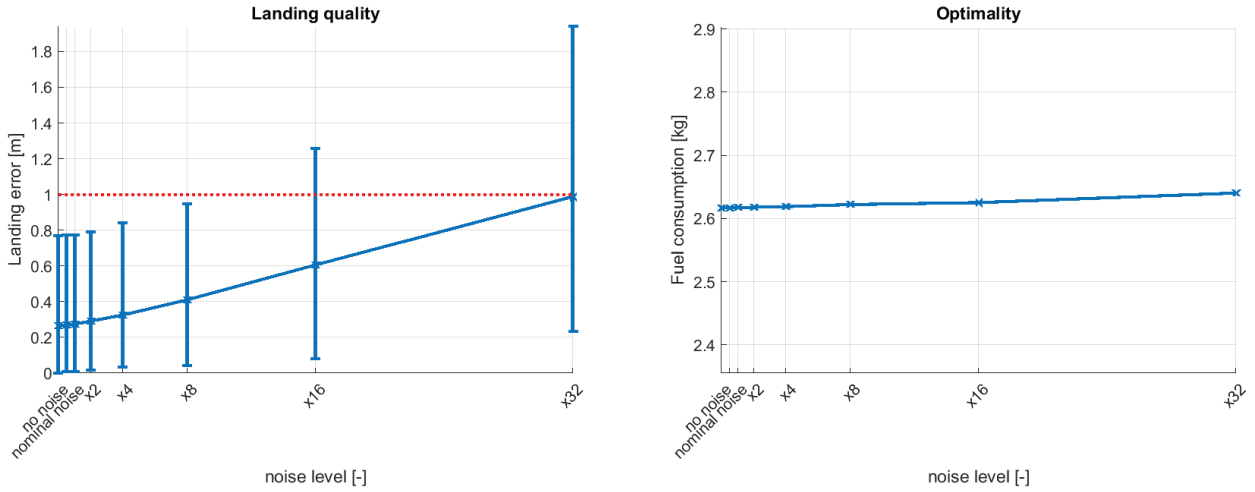


Figure 11: Some of the plots included in the tests noise dispersion report

8. Conclusions

In this paper we presented the algorithms and the software framework developed under the ESA-FLPP program to enable the use of novel G&C architecture for space applications. The results are encouraging and the software is now ready for the flight test campaign that will start in Q3 2019.

9. Acknowledgments

This work has been funded by ESA-FLPP program under the contract number 4000124652. A special thanks also to Hans Strauch (Airbus DS) and INCAS for the technical support provided during this project.

References

[1] Blue origin. *https://www.blueorigin.com/.*

[2] End-to-End retargeting mission with engine fault. *embotech AG*, https://www.embotech.com/aerospace/.

[3] Spacex. *https://www.spacex.com/.*

[4] Speedgoat real-time simulation and testing. *Speedgoat*, https://www.speedgoat.com/.

[5] Behçet Açıkmeşe, John M Carson, and Lars Blackmore. Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem. *IEEE Transactions on Control Systems Technology*, 21(6):2104–2113, 2013.

[6] Lars Blackmore. Autonomous precision landing of space rockets. In *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2016 Symposium*. National Academies Press, 2017.

[7] Alex Domahidi and Juan Jerez. Forces Professional. *embotech AG*, https://www.embotech.com/forces-pro/, 2014-2019.

[8] Juan Jerez, Sandro Merkli, Samir Bennani, and Hans Strauch. Forces-RTTO: A tool for on-board real-time autonomous trajectory planning. In *2017 10th International ESA Conference on Guidance, Navigation & Control Systems 29 May - 2 June, Salzburg, Austria*, 2017.

[9] J. Löfberg. YALMIP: A toolbox for model and optimization in MATLAB. In *IEEE Internation Symposium on Computer Aided Control Systems Design*, pages 284–289, Taipei, Taiwan, Sep 2004.

[10] Ping Lu. Introducing computational guidance and control. *Journal of Guidance, Control, and Dynamics*, 40(2):193–193, 2017.

[11] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. The real-time linux kernel: A survey on preempt_rt. *ACM Computing Surveys (CSUR)*, 52(1):18, 2019.

[12] Daniel P. Scharf, Scott R. Ploen, and Behçet Açıkmeşe. Interpolation-enhanced powered descent guidance for onboard nominal, off-nominal, and multi-x scenarios. In *AIAA Guidance, Navigation, and Control Conference*, page 0850, 2015.

[13] Daniel P. Scharf, Martin W. Regehr, Geoffery M. Vaughan, Joel Benito, Homayoon Ansari, MiMi Aung, Andrew Johnson, Jordi Casoliva, Swati Mohan, Daniel Dueri, et al. Adapt demonstrations of onboard large-divert guidance with a vtvl rocket. In *Aerospace Conference, 2014 IEEE*, pages 1–18. IEEE, 2014.

[14] Spacex. Grasshopper. *http://en.wikipedia.org/wiki/Grasshopper_(rocket).*

[15] Masten Space Systems. Xombie (XA-0.1-B-1). *https://www.masten.aero/xoie-1.*

[16] Michael Szmuk and Behçet Açıkmeşe. Successive convexification for 6-dof mars rocket powered landing with free-final-time. In *2018 AIAA Guidance, Navigation, and Control Conference*, page 0617, 2018.

[17] Michael Szmuk, Behçet Açıkmeşe, and Andrew W. Berning. Successive convexification for fuel-optimal powered landing with aerodynamic drag and non-convex constraints. In *AIAA Guidance, Navigation, and Control Conference*, page 0378, 2016.