73rd International Astronautical Congress (IAC), Paris, France, 18-22 September 2022.

IAC-22-D2-IPB-5-x69380

The Vertical Landing Vehicles Library (VLVLib): a Modelica-based approach to high-fidelity simulation and verification of GNC systems for reusable rockets

Stefano Farì [a]*

[a] *German Aerospace Center (DLR), Institute of Space Systems, Department of Navigation and Control Systems, Robert-Hooke-Str. 7, 28359, Bremen, Germany,* stefano.fari@dlr.de

* Corresponding author

Abstract

Vertical Landing (VL) Reusable Launch Vehicles (RLVs) must rely on performing and robust Guidance, Navigation and Control (GNC) algorithms, that are normally tested and verified in closed-loop high-fidelity simulators. This paper introduces the Vertical Landing Vehicles Library (VLVLib), a new Modelica-based tool for the advanced physical modeling and simulation of VL RLV dynamics. Modelica, an acausal open-source object-oriented modeling language, is exploited to produce a multibody vehicle model where multiple effects can be easily added, modified or switched in complexity. At the same time, it offers a modular modeling methodology to quickly adapt to vehicle changes or potential new features during the development process, with benefits in terms of project speed and costs. The way Modelica features are exploited is detailed throughout the paper. At its current development status, the library is able to account for, in more detail, four main effects relevant for the GNC robustness tests: (1) propellant slosh dynamics; (2) Thrust Vector Control (TVC) dynamics; (3) landing legs deployment disturbances; (4) touchdown dynamics. They are modeled and integrated so to allow an easy generation of a vehicle model that holds a prescribed fidelity level for each sub-system, and can be compiled and exported and harmoniously integrate other simulation environments (*e.g.* Simulink). The potential of VLVLib is demonstrated with representative examples as applied to CALLISTO reusable rocket demonstrative mission.
Keywords: Advanced modeling, Modelica, Simulation, Reusable Launch Vehicles, GNC, Verification

1. Introduction

Reusable rockets, technically addressed as Vertical Take-off, Vertical Landing (VTVL) Reusable Launch Vehicles (RLVs), are nowadays imposing themselves as the cheaper and most viable way to access space [1]. The (currently partial) reusability concept has been technologically and commercially proven for the first time by SpaceX [2] with its Falcon 9 (Figure 1). December 21st, 2015 marked the formal beginning of the reusability era for rockets at industrial level. On this track, several other companies followed this example, like Blue Origin or Rocket Lab [3]. Nowadays, the entire SpaceX's Falcon 9 program is founded on an impressive frequency in launches per year. The keys of success are surely multiple, but it is undoubted that one of the most preeminent ones is the Guidance, Navigation and Control system. Reusable rockets performance and reliability requirements fulfillment strongly depend on its "quality". Its design normally relies on model-based methods which exploits appropriate mathematical models to capture the relevant vehicle dynamics. Based on it, the final algorithms design can provide the highest stability margins and match the mission specifications. Such a model is referred to as "synthesis model" and typically neglects some effects that are foreseen to have little impact on its behavior or can possibly be counteracted by the control system.

For Validation and Verification (V&V), the Guidance, Navigation and Control (GNC) system must be tested in presence of nominal and perturbed vehicle and environ-

The author thanks Dr. Stephan Theil and Dr. David Seelbinder *(German Aerospace Center (DLR), Institute of Space Systems, Robert-Hooke-Str. 7, 28359, Bremen, Germany)* for guiding and supporting this work, as well as providing the tools to carry it on. The author thanks Dr. Samir Bennani and Dr. Pedro Simplicio *(European Space Agency (ESA), 2201 AZ Noordwijk, The Netherlands)* for providing guidance related to Section 5. This work is partly related with ESA's Co-Founded Ph.D. program (grant I-2020-06089). Lastly, the author thanks M.Sc. Davide Grande (*University College London, Torrington Place, London, WC1E7JE, UK*) for the useful insights that assisted the construction of this article.

Fig. 1: SpaceX's Falcon 9 reusable rocket at landing [4].

mental conditions. Moreover, it is important to use more complex vehicle models to assess the degradation in performance and robustness in presence of closer-to-real dynamics. They are usually referred to as "simulation models". This is why a simulation model can be seen as a much more complex version of the synthesis model. In the following, the best strategies to obtain high fidelity models (from a GNC engineering V&V perspective) are discussed.

1.1 Adopting physical models for RLV's high-fidelity dynamic simulations

Within the GNC world, the state-of-practice environment for dynamic simulations is Simulink, a Matlab extension with several blockset allowing the user to develop and simulate dynamic models [5]. Simulink is convenient for rapid prototyping, for building up more complex dynamic models, as well as for testing of the whole GNC architecture and implementing requirement checks or performing Monte-Carlo campaigns. Simulink is a 'causal' or 'block-oriented' modeling environment. Causal models define the relation between different dynamical elements solely by the output of one block being fed into the input of another block. From the point of view of the numerical solver, at each instant of calculation, the values of the state variables are considered known, while the unknowns are the derivatives of the state variables.

Despite Simulink is designed for easy integration with control systems (which are inherently causal), its main limitation consists in the blocks having a unidirectional data flow. As such, certain "objects" cannot be dealt with directly. Electrical machines or gearboxes, for example may have a direct or indirect power flow. In such cases, several manual steps are required to transform the equations to a form implementable on Simulink. This separates the implementation from the actual physical phenomenon and/or the system architecture. Above all, it offers very poor reusability of the elementary components, inducing a major rework of the whole set of underlining equations as soon as the system configuration changes. These statements are especially justified for launch vehicles where some specific (sub-)dynamics may (or may not) be added to the overall vehicle model depending on the required fidelity level. Furthermore, certain models requiring algebraic constraints are largely unfeasible or cumbersome to implement in Simulink. Lastly, the fidelity level of each sub-component may not be exchangeable with ease, as well as modeling cross-coupling between different physical domains.

Acausal modeling, on the other hand, is based on equations instead of assignment statements. It is often referred to as physical modeling because is very well suited for representing the physical structure of modeled systems. Therefore, acausal environments can be used to overcome these limitations and give additional modeling flexibility.[1] Modelica modeling language [6,7] is a open-source acausal Object-Oriented Modeling (OOM) language for large system descriptions. It enables a clear representation of physical meaning within an object-oriented structure; furthermore, OOM delivers modularization and component reuse features, thus achieving a greater model adaptiveness against changes. The class concept and inheritance property are the keys for reuse of modeling knowledge in Modelica. The 'inheritance through modification' property via the so-called 'modifiers' renders reusing the models even more straightforward and the implementation concise. A free Modelica Standard Library (MSL) exists [8] to facilitate the user with several basic components belonging to different physical domains. This also enables a relatively accessible creation of multibody models, crucial aspect of this work [9].

This approach has been used already, for example, in ve-

[1]For maximum compatibility with Simulink environment, one can consider Mathworks SimMechanics payware add-on package. Even in this latter way, some issues exist: above all, SimMechanics hides the models implementation to the user. This compromises flexibility, adaptability and, to some extent, comparisons with other models.

hicle re-entry scenarios via a guided parafoil, lunar landing mission or modeling slosh dynamics [10–12]. For similar reasons, several libraries have been created, like the DLR Flight Dynamics Library [13] or the Space Flight Dynamic library [14]. Modelica has been already used to describe launch vehicle dynamics and control [15], stage separation [16], or trajectory optimization [15]. The language, therefore, can well suit the creation of a high-fidelity RLV models.

1.2 VLVLib: a new tool for advanced RLV modeling

This work aims at presenting the **Vertical Landing Vehicles Library (VLVLib)** Modelica library. Its purpose is to enable high-fidelity modeling of Vertical Landing (VL) RLVs, achieved by implementing specific dynamics deeply affecting VL RLVs[2]. Currently they are: (i) propellant slosh; (ii) the Thrust Vector Control (TVC); (iii) the landing legs deployment; (iv) and the ground contact. In this way, they can concur in determining a more accurate overall vehicle dynamics. The key point is to exploit OOM to deliver a high flexibility to the user such that: (i) new models can be created easily by inheriting a basic vehicle model; (ii) this new model can include specific dynamics at user's will; (iii) these dynamics can be easily exchanged according to the required fidelity level[3].

Lateral propellant slosh dynamics is a deeply studied phenomenon and a well known factor to consider in the GNC system design since the early days of large liquid-fuel rockets [17]. If the propellant contained in a tank has a free surface, parasitic interactions with its structure may arise as an effect of the liquid movements. Normally, these adverse dynamics cannot be neglected and should be appropriately tackled to avoid critical performance degradation or instability.

The TVC system is essential to provide enough control authority to any launch vehicle. It is often initially modeled as a first or second order dynamical system; however, it is a highly complex system composed by several components. In this paper, focus is placed on Power-by-Wire-actuated TVC systems, thus employing Electro-Mechanical Actuators (EMAs)[4] [20, 21], and composed by a mechanical transmission, an electric motor and its electrical drive. Each of these component introduces several nonlinear changes in the overall actuator dynamics. Therefore, a physical model of all these components is beneficial for GNC or even for a local model-based fault detection performance verification [22].

In the landing phase, most VL RLVs must eject their landing legs. This process may destabilize the vehicle because of the excessive asymmetrically-induced forces and torques due to an uneven unfolding. It can be caused by specific aerodynamic conditions or the deployment mechanisms itself. As such, the modeling of the landing leg system is also important. Furthermore, it must be understood how vehicle's attitude and speed affect touchdown; this is possible only if contacts are modeled as well, an aspect rarely addressed in literature [23].

The rest paper is structured as follows: Section 2 covers the modeling rationale, conceptual structuring, and related Modelica features enabling the taken choices; Section 3 describes the VLVLib main packages and explains how to use them to generate a new vehicle model; Sections 4 to 7 explain in details the propellant slosh dynamic, the TVC system, legs deployment, and ground contact models, respectively. Section 8 shows representative simulation results of those dynamics. Lastly, in Section 9 conclusions are discussed.

2. Library development approach

Hereafter, the key-concepts which drove the library development are discussed in detail. Dymola IDE[5] has been used [24]. A certain knowledge of the Modelica language is assumed: Fritzon [25] covers extensively Modelica principles, syntax, main libraries and the compilation and simulation best practices. The latest Modelica language specifications can be found in [26].

Nomenclature Being Modelica an object-oriented language, almost everything is a class. However, there exist special types of classes specialized for different contexts and uses. They are named as: *model*, *connector*, *record*, *block*, *function*, *type*, *package*. As the use of these keywords may cause ambiguity, the italic font will be hereafter applied when these words associate with the Modelica language itself. On the other hand, the class names and objects are written using a teletype font (`Class`), while the Modelica keywords are blued too (`if, then, else`).

2.1 Package classes hierarchy

The VLVLib is composed by several nested *packages*. The main and the child *packages* all have the `encapsulated` property, which guarantees that any instantiated class name lookup stops at the boundary of the

[2] The library name does not include "Vertical Take-off" simply to not restrict its scope of application. For example, a planetary lander can be potentially modeled with the VLVLib too [12].

[3] Supposed that multiple models are available for the same component and they are 'plug-compatible' (see Section 2).

[4] EMAs are used in several rockets already, *e.g.* Avio's Vega [18, 19].

[5] Integrated Development Environment.

```
encapsulated package VLVLib
  "Vertical Landing Vehicles Library"
 import Modelica;
 // <...> Other declarations and dyn. equations
end VLVLib;
```

Fig. 2: VLVLib package root definition. The MSL (`Modelica` *package*) is made available by means of the 'import' command.

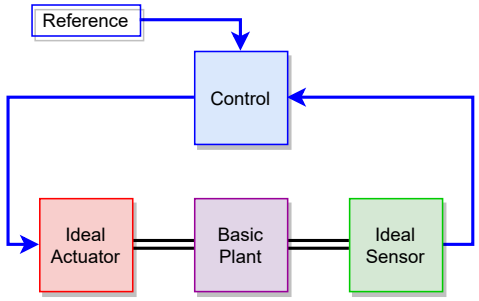

Fig. 3: Architecture-driven example scheme. Blue connectors are casual, whilst black ones have no direction, hence acausal. All components can be swapped with other variants provided the old and new interfaces match.

package itself. As a consequence, each package cannot include classes from other *packages* or from the MSL, unless they are included via the `import` command within the class declaration (see example in Figure 2).

2.2 The architecture-driven approach

The library is designed with modularity and flexibility as key objectives by adopting the so-called "architecture-driven approach". Architectures can be defined as models where a collection of subsystems have been pre-connected. The composition of the system is carried out by selecting specific implementations (*models*) for each subsystem. In this way, the user only needs to choose the specific model to use for each particular subsystem. A classic scenario is depicted in Figure 3 where the fundamental components of a closed-loop control system are shown. With the architecture-driven paradigma, the user can swap the control, actuator, plant or sensor classes according to the required fidelity level or simulation needs. This is possible in Modelica language by declaring a specific component as `replaceable`. Such an attribute makes possible to another *model* instantiating the first one, to eventually access it and swap the replaceable component with another one. This is carried out via the `redeclare` command, as clarified in Figure 4.

However, to make the models swapping well defined, the old and new models must be "plug-compatible". A model X is plug-compatible with a model Y if for every public variable in Y, there is a corresponding public variable in X with the same name. Furthermore, every such variable in X must itself be plug-compatible with its counterpart in Y. This can be enforced by making the replaceable component constrained to have a specific type by using the syntactic construct `constrained by`.

A clean, functional and flexible way to address all the above is to operate as in Figure 5: the model of a specific complex system can be built with a top down architectural approach, defining firstly the structure of it, and then adding specific subsystem implementations. The starting point is to create a *partial model* for each architectural component (*e.g.* `Sensor`) just to define its interfaces. An interface is essentially a formal definition of the constraining type. Each component must inherit its own defining interface class (like `IdealSensor`). Then, another *partial model* (`SystemArchitecture`) defining the system architecture instantiates all these interfaces classes. They are declared as replaceable and suitably connected. Note that this *model* has no implementation details on its own.[6] Finally, in a new model (such as `BaseSystem`) the architecture *model* is inherited and the interface classes redeclared to be new classes containing the wanted components implementation. When a component is redeclared, the new redeclaration supersedes the previous one. The redeclaration is legal only if the new *model* shares the constraining type of the replaceable one (*i.e.* `IdealSensor` can be swapped with `Sensor`, because they share the same interface, defined by `Sensor` itself). In the example, `BaseSystem` redeclares the (empty, partial) `Sensor` *model* with `IdealSensor`; this is legal because `IdealSensor` is plug-compatible with `Sensor`.

2.3 Models export and Simulink integration

Models built by means of the VLVLib are to be included within the Matlab/Simulink environment. In the latter, a whole simulation infrastructure is usually built for the purpose of developing and testing the GNC system [27]. This may include tools like Monte-Carlo frameworks or an automated requirement evaluation logic. In this context, it is possible that functions like frame conversions, environment properties or even certain dynamic models, are

[6] This means that this *model* cannot be simulated in any way, also because it is declared as 'partial'. *Partial models* cannot get directly instantiated and must be extended by other *models*.

```
model HierarchicalSystem
 BasicPlant plant;
 IdealActuator actuator;
 replaceable IdealSensor sensor;
 // <...> Other declarations and dyn. equations
end HierarchicalSystem;

model SystemWithModifiedSensor
 extends HierarchicalSystem(
    redeclare SampleHoldSensor sensor);
end SystemWithModifiedSensor;
```

Fig. 4: Example of component redeclaration. A generic sensor implementation within `HierarchicalSystem` is replaced by another one in the extended model.

already present and do not need to be duplicated in the form of Modelica code. This is accounted for in the implementation of the VLVLib. For example, in the VLVLib, the outputted vehicle states are expressed with respect to a single body-fixed frame (states expansion to other frame can be done within Simulink). Or, certain model parameters can be implemented as external inputs instead.

2.4 Information propagation across components

The final vehicle *model* is composed by dozens of other *models* spread throughout several instantiation layers. This may cause the quantities to be logged to multiply in number, causing cluttering and making any modification to the root vehicle class tedious and error-prone. For this reason, the relevant *connectors* of a specific *model* are grouped into a specific communication interface, called 'expandable connector' (or 'bus'). A Modelica *connector* declared as expandable has no minimum requirement for information to be carried onto. The bus can be expanded to include additional information, even if they are coming from connectors of different types. The signals on an expandable bus are determined by the connections themselves, therefore by connecting something to an expandable bus, a signal is implicitly added to that connector. The Modelica compiler looks at all the connectors in a connection set and expands each one so that the carried information is suitably matched. An expandable bus is defined as in Figure 6.

This feature turns essential when a specific component implementation has a different interface with its constraining type (not plug-compatible). As, instead, the expandable bus is part of the constraining class, it is enough that the information carried by it is always defined and legibly used by other interconnected *models*. This is adopted in the case of some EMA components and will be further detailed in Section 5.4 (note nr. 16).

```
partial model Sensor
 Flange_a shaft; // input connector
 RealOutput w;   // output connector
end Sensor;

model IdealSensor
 extends Sensor;
 // <...> Actual implementation of the sensor
end Sensor;

partial model SystemArchitecture
 replaceable Sensor sensor
    constrained by Sensor;
 // <...> Other declarations and dyn. equations
end SystemArchitecture;

model BaseSystem
 extends SystemArchitecture(
    redeclare IdealSensor sensor);
 // <...> Other redeclarations or modifications
end BaseSystem;
```

Fig. 5: Example of the architecture-driven approach. A partial model `SystemArchitecture` builds up the system architecture, while `BaseSystem` model defines the models populating its replaceable components.

```
expandable connector ExpandableBus
end ExpandableBus;
```

Fig. 6: Example of an expandable bus definition. No declaration must be necessarily made within the *expandable connector*, as the connected types are automatically determined at compilation time.

3. Description of the Vertical Landing Vehicles Library (VLVLib)

The VLVLib is currently based on seven main subpackages (see Figure 7) described below, whereas their logic dependencies are shown in Figure 8.

#1 The `Vehicles` *package* contains the full classes implementing user's vehicle models (*e.g.* Alina[7], Callisto,...). As such, an arbitrary number of them may be present. However, one main class, called

[7]ALINA is a lunar lander developed by Planetary Transportation Systems and meant to enable payload deliveries on the surface of the Moon [28].

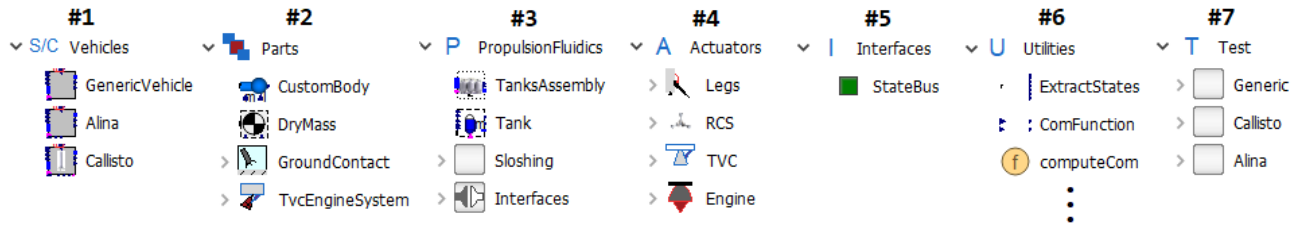

Fig. 7: Listing of the current VLVLib *packages* and their first sub-level content.

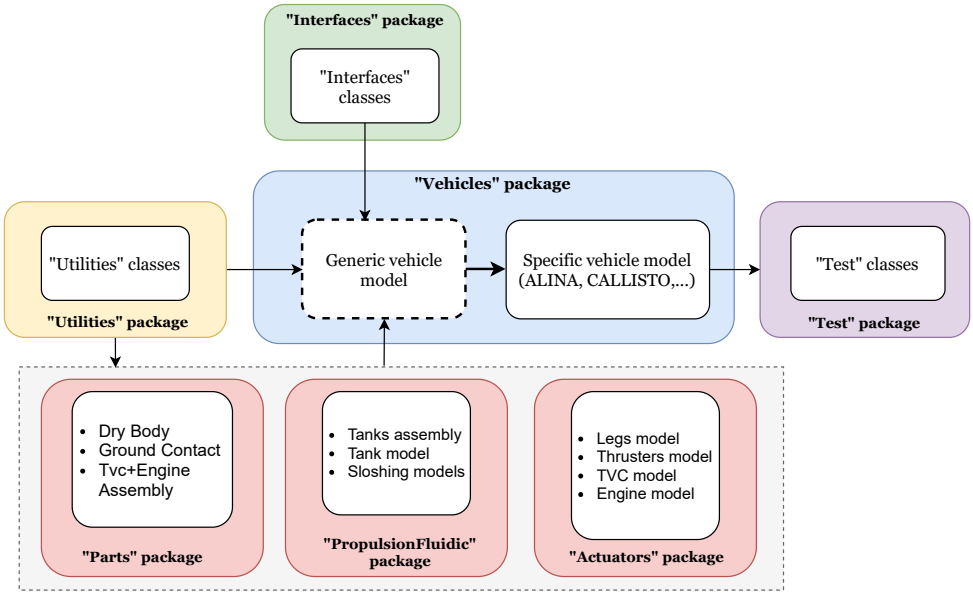

Fig. 8: VLVLib *packages* functional dependencies.

`GenericVehicle`, is the baseline to define these specific vehicles in order to adhere to the explained modeling philosophy. It contains only the main infrastructural components and interfaces, as well as the `DryBody` (representing the propellant-free structural mass and moment of inertia) and `TanksAssembly` classes. To create a new spacecraft, `GenericVehicle` has to be extended and (eventually) augmented with other components of interest implemented within #2-4 *packages*. The procedure is explained in Section 3.1.

#2 Contains spare classes which can concur in building the `GenericVehicle` class or other components.

#3-4 Contain the classes necessary to include the tank mechanical models, as described in Section 4, and the actuator dynamics, like the landing legs and TVC system, respectively detailed in Sections 5 and 6.

#5 Contains the output bus needed by `GenericVehicle`.

#6 Includes spare classes to achieve specific functionalities across the whole library.

#7 Contains *models* for the functional testing of the each vehicle's implementation.

3.1 Using the VLVLib to model a custom vehicle

The `GenericVehicle` class declares all the components needed to model a basic vehicle; specific configurations are derived starting from this class. Its diagram view is depicted in Figure 9. All *connectors* in this and other vehicle models must be 'causal', or busses made of causal *connectors*[8]: this allows the *model* to be afterwards compiled and embedded in Simulink or other softwares. Aside

[8]Causal connectors are here intended as those including (or inheriting) Real, Integer, Boolean or Enumeration *types* only. In Modelica formal terms, the *connectors* do not declare any `flow` variable.

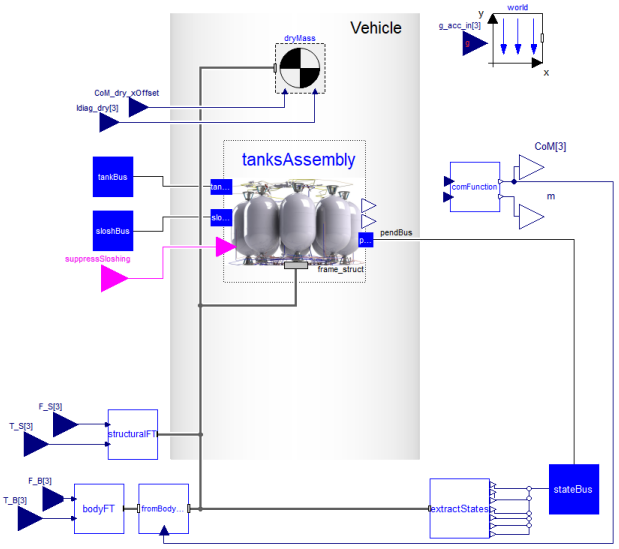

Fig. 9: `GenericVehicle` class diagram view.

```
model Alina "Advanced ALINA spacecraft model"
  extends GenericVehicle(
    r_ComDry={0.2,0,0}, // dry CoM position
                        //confidential
    m_dry=1000, // dry mass, confidential
    tanksAssembly(
     nTanks=11, // number of tanks
     // position of each tank frame wrt frame_S
     r_wrt_S=[0.1,     -0.2,    0.7;
              0.1,      0.2,   -0.95;
              0.1,      0.2,    0.95 //...
              // Other data
              ],// confidential
      // Init. position of each pendula per mode
      anglesYz_t0_mode1 = zeros(nTanks,2),
      anglesYz_t0_mode2 = zeros(nTanks,2),
      anglesYz_t0_mode3 = zeros(nTanks,2),
      // number of sloshing modes per tank
      nSloshModes = {3,3,3,3,3,3,3,3,0,0,0}
    )
  );
end Alina;
```

Fig. 10: `ALINA` spacecraft class extends `GenericVehicle`. Lines marked with "confidential" comment have modified values for confidentiality reasons.

from the necessary interfaces, `GenericVehicle` includes the following classes:

1. the `World` class, which defines the inertial frame (called 'World') for referencing all model states and the gravity field. This is needed to operate with the MSL's Multibody library;
2. the `DryBody`, modeling the dry structural mass and moment of inertia;
3. the `TanksAssembly`, to model the propellant distribution and, if activated, the slosh dynamics;
4. the support classes enabling the input of generic forces and torques acting either on the vehicle-fixed frame or derived ones (*e.g.* placed at the overall vehicle center of mass position).

The components with mechanical interfaces are connected together via the gray wire in Figure 9 which represents the main vehicle-fixed frame describing vehicle dynamics. Note that to enable custom implementations within Matlab/Simulink of gravity gradients, the basic MSL's Multibody library gravity models have been disabled; the gravitational acceleration is, instead, an input of the model. This implementation is described in detail in [12].

Creating a new vehicle model can be simply done by extending `GenericVehicle`, as shown in Figure 10, where no other component nor connection is added, and the vehicle parameters are simply modified to reflect the vehicle properties. Another example is Callisto rocket model in Figures 11 and 12, which includes additional (and appropriately connected) dynamic models for the TVC system and the actuated legs. Note that, even if these latter elements are declared within `Callisto` class, they may be deactivated via the two Boolean flags `enableTvc` and `enableLegs` (defaulted to 'true'). This possibility goes under the name of "conditional component declaration" and adds flexibility to the model[9] without implementing redundant variants.

4. Propellant sloshing model

From the GNC perspective, it has been demonstrated that a pendulum model is enough to approximate the fluid dynamics of interest as soon as the sloshing natural frequencies are not excited [29–31]. The vehicle motion is therefore represented by a multibody model, where a rigid body representing the dry vehicle is flanked by an equivalent mechanical model of the fluid in each tank. Three modes of oscillation are largely sufficient to characterize the main disturbances produced on the vehicle, as higher sloshing modes have a small impact on the whole dynam-

[9] When the condition is false, the component and all related connections are removed. Naturally, the resulting model must be well-posed.

```
model Callisto "Advanced Callisto dynamic model"
  extends GenericVehicle(
    tanksAssembly(
      // <...> Tanks assembly model parameters
    );
  VLVLib.Parts.TvcEngineSystem.Variants.TvcSystem_v1 TvcSystem      if enableTvc;
  VLVLib.Actuators.Legs.LegsAssembly legsAssembly                   if enableLegs;
  Modelica.Blocks.Interfaces.RealInput Thrust[3](unit="N")          if enableTvc;
  Modelica.Blocks.Interfaces.BooleanInput deployCmd                 if enableLegs;
  Modelica.Blocks.Interfaces.RealInput beta1(unit="rad")            if enableTvc;
  Modelica.Blocks.Interfaces.RealInput beta2(unit="rad")            if enableTvc;
  parameter Boolean enableLegs = true;
  parameter Boolean enableTvc  = true;
equation
  // <...> "connect" statements between components
end Callisto;
```

Fig. 11: `Callisto` class Modelica code.

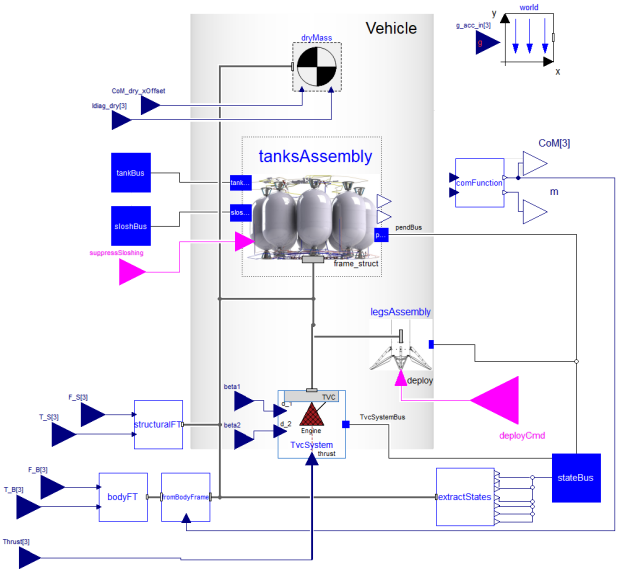

Fig. 12: `Callisto` class diagram view.

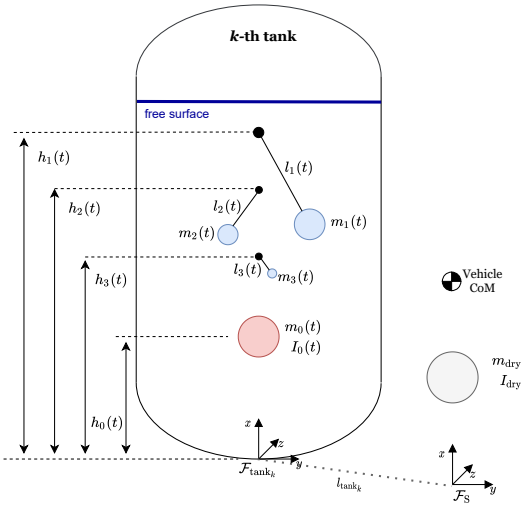

Fig. 13: Representation of the employed pendulum equivalent model.

ics.

What follows is extensively treated in [12]. The development requirements for the sloshing model are:

1. the vehicle model shall address the sloshing phenomenon without altering the single tanks MCI[10] static properties (see Eqs. 1 and 2);
2. the model shall not make any planar motion assumption;
3. the fuel draining effect shall be considered, i.e. sloshing parameters are considered as slowly time-varying;
4. the capability of including up to three sloshing modes per tank shall be given;
5. the capability to stop the sloshing motion during simulation under non-accelerated phases shall be given.

[10]Mass, Center of Mass, Inertia.

4.1 The sloshing model

The equivalent model characteristic parameters correspond to one specific mode of liquid oscillation at a specific tank filling level, and depend on the liquid properties and tank geometry [32]. They can be obtained with CFD meth-

ods [33] or analytically for simple tank shapes [34], and then validated experimentally [35].

Figure 13 illustrates the tank model together with the main reference frames. The model distinguishes between the sloshing and the non-sloshing parts of the liquid: the liquid mass m_{liq} is split between the single pendulum masses m_i, with $i \in [1, 2, 3]$ being the sloshing mode, and the mass m_0 representing the idle liquid. Furthermore, h_0 and h_i are the non-sloshing body and the i-th pendulum hinge height with respect to the bottom of the tank; m_i and l_i are the mass and arm length of the i-th pendulum. Parameters m_0, I_0, h_i, l_i and m_i are provided with respect to discrete tank filling levels and have to be interpolated during the simulation. The frame $\mathcal{F}_{\text{S}}$ is a vehicle-structure-fixed frame conveniently chosen, whereas $\mathcal{F}_{\text{tank}_k}$ is a frame at the bottom point of the k-th tank, with $k = 1, 2, ..., K$. The whole liquid inertia in a tank I_0 is considered and placed together with the non-sloshing mass m_0. The gray body in Figure 13 models the dry vehicle: the mass and moment of inertia have been labeled with m_{dry} and I_{dry}. The depicted position of the overall vehicle's center of mass is influenced by the idle masses distribution, but also by each pendulum motion.

To ensure the model validity, the static properties of the liquid must be preserved at every time instant for every k-th tank. Hence,

$$m_0 + \sum_i m_i = m_{\text{liq}}, \tag{1}$$

$$m_0 h_0 + \sum_i m_i h_i = 0. \tag{2}$$

Despite Figure 13 planar depiction, each modeled pendulum is enabled to move in three dimensions.

Damping is included as well as function of the vehicle longitudinal acceleration and the damping coefficient. The latter is derived by the tank shape and liquid properties and is not further described here [32].

4.2 Library implementation

The main implemented library components are:

1. a class for the tanks assembly (`TanksAssembly`);
2. a `Tank` class where the discussed three sloshing modes are included;
3. a `Pendulum` class describing one sloshing mode.

The `Pendulum` class is responsible for modeling the pendulum for slosh dynamics representation. It is allowed

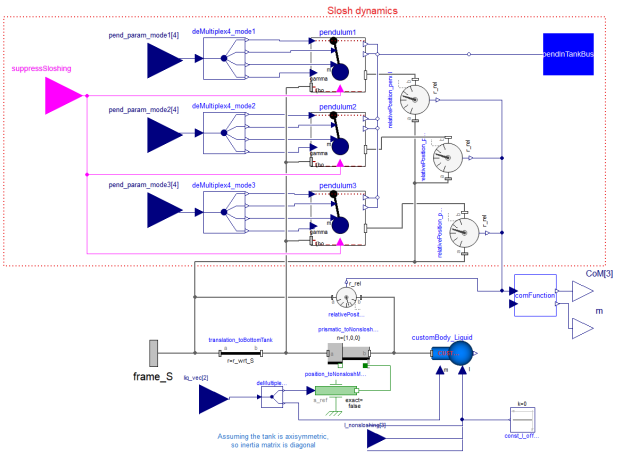

Fig. 14: `Tank` class diagram view.

to move in the $x - y$ and $x - z$ planes by means of a universal joint which allows two rotational degrees of freedom at the hinge point. As from requirements, the model can adapt to the slowly-varying sloshing parameters changes by means of prismatic joints: they allow for the translational motion of the model elements to reflect the changes in hinge height and pendulum arm. A rotational damper is added at the pendulum hinge point in order to simulate energy dissipation. This component can also act in synergy with a rotational spring, to stop the sloshing motion when a flag `suppressSloshing` is set to 'true' and restore the pendulum neutral position at $\theta = 0$. When this happens, the damping coefficient and the spring stiffness pass, respectively, from C_θ and zero to suitable high values.

The `Tank` class instantiate the three pendula and models the static liquid (Figure 14). It class uses `CustomBody` component to represent the nonsloshing body with a variable mass m_0 and a variable inertia I_0.

Three pendulum model instances have been used for embedding the three sloshing modes: they are conditional components, such that the number of pendula depends onto the (`nSloshMode`) parameter (Figure 15). Note that is possible to exclude sloshing effect (removing all pendula) by setting `nSloshModes`=0. The parameters m_i, l_i, h_i and C_θ are provided as external inputs to each pendulum model.

The `TanksAssembly` class is meant to instantiate as many `Tank` components to model the complete vehicle tanks configuration. To do so, the `Tank` class has been declared as a `nTanks`-long vector, with `nTanks` = K being the prescribed number of tanks. The tank- and sloshing-related parameters are accordingly inputted to every ele-

```
model Tank "Tank model"
 Sloshing.Pendulum pendulum1 if nSloshModes >= 1
 Sloshing.Pendulum pendulum2 if nSloshModes >= 2
 Sloshing.Pendulum pendulum3 if nSloshModes == 3
 // <...>
end Tank;
```

Fig. 15: Conditional declaration of three `Pendulum` components within `Tank` class. The single pendulum models are instantiated (and its connections activated) for specific values of parameter `nSloshModes`.

```
model TanksAssembly "Tanks assembly"
// <...> All declarations
equation
 for i in 1:nTanks loop
 connect(frame_struct, tank[i].frame_S);
 connect(tankBus.liq_vec[:, i],tank[i].liq_vec);
 end for;
// <...> Other connect statements
end TanksAssembly;
```

Fig. 16: Recursive connections within `TanksAssembly`.

ment of the classes vector. With this implementation, there is no need to manually reinstantiate as many tank components as needed by the specific vehicle configuration. The code structure for recursive connections in Figure 16.

5. Thrust Vector Control modeling

The TVC system needs two direct-drive[11] Electro-Mechanical Actuator (EMAs) to deflect the engine's thrust direction. The detailed modeling of the engine thrust dynamics is currently not implemented, it is assumed that the produced thrust magnitude is a model input.

The EMA operates thanks to the synergism of three main components: a Mechanical Power Transmission (MPT), meant to transform in the most efficient way a rotatory motion into a translational one; an Electric Motor (EM), here considered as a PMSM[12], to produce the mentioned rotational motion; the Power Drive Electronic (PDE) part meant to suitably power the EM using the available the on-board electric current source; a control logic meant to achieve the reference EMA elongation, thus engine deflection. The overall architecture is shown on a plane in Figure 17, but the off-plane direction is also considered in the implementation. The proposed modeling strategy is based, with minor differences, on the incremental prototyping approach presented in [36, 37], where several models with different fidelity levels are described for each component. They are briefly recalled below.

[11] There is no reduction gearbox to make the EMA more compact.
[12] Permanent Magnets Synchronous Machine

5.1 The control system

The EMA synthesis model transfer function $G(s)$ can be expressed as:

$$G(s) = \frac{4\pi^2/p^2}{(M_m + M_s)s}, \tag{3}$$

where M_s is mass reflecting engine's inertia, M_m is the mass reflected at load level by the motor rotor inertia, and p is the lead of the roller-screw.

The control system objective is to regulate the EMA screw position, given the reference input X_c. The adopted scheme is represented in Figure 18, where an architecture with an inner (motor angular velocity ω_m control) and outer (position X_s control) loops is proposed. The inner and outer controllers are simply proportional ones, given each loop has already an integral action. The motor currents dynamic, being faster than the inner loop, is shown with a unitary gain. The control gains are tuned by simply imposing a damping ratio and natural frequency to the second order system resulting by the composition of the two loops.

Generating the necessary torques via the electric motor requires a third internal control loop managing the motor three-phase currents. The overall control scheme is shown in Figure 19 and the details of such a standard implementation can be found in [38]. The current dynamics can be simplified as the one of a simple brushed DC motor as

$$G_m(s) = \frac{1}{R_m + L_m s}, \tag{4}$$

where R_m and L_m are the motor windings resistance and inductance, respectively. The need of a PI controller is common and arises from the fact that there exist back electromotive forces and cross coupling terms (see Section 5.3), here intended as disturbances for control design.

Lastly, the EMA-engine structure is not infinitely stiff. This introduces several elastic effects, for example at the structural anchorage point, the nut-screw transmission or the screw-engine connection mechanisms. This can be represented as an equivalent spring-damper system including all these compliances. As such, it is possible to include a load sensor at the screw-engine connection point in order to include, within the control system, a force feedback

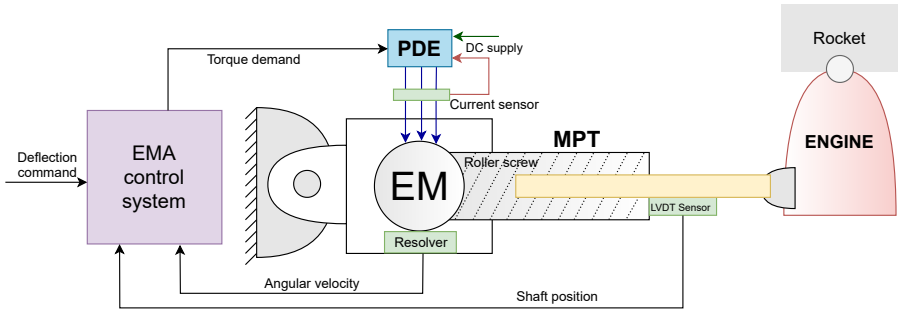

Fig. 17: Simplified planar TVC architecture using an Electro-Mechanical actuator.

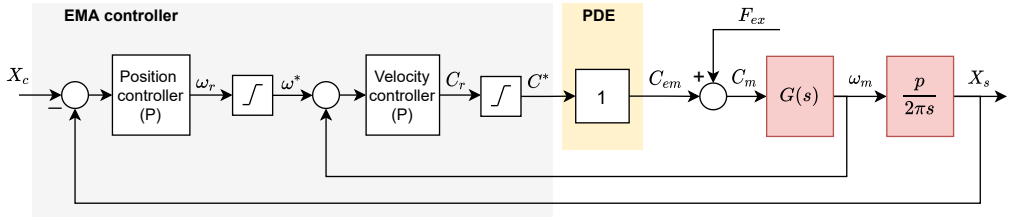

Fig. 18: Schematic simplified architecture of an EMA control system. F_{ex} is an external disturbance load.

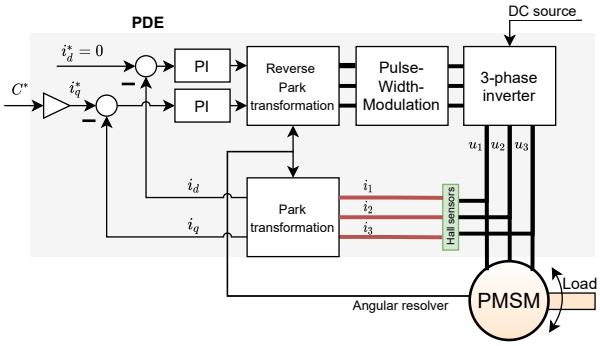

Fig. 19: Schematic of the Field-oriented control for a 3-phase motor.

compensation law which would dampen the parasitic oscillations [39]. This is normally carried out by applying a notch filter to the sensed force which would pass only the relevant frequencies to be compensated for, and, after a suitable conditioning, injecting an additional torque to C^* control command.

5.2 Mechanical Power Transmission modeling

The MPT is in charge of transforming the EM rotary motion into a linear one. The considered transmission is a roller-screw, proven to accept higher loads than the ball-screw one. In the following, the implemented models are discussed.

5.2.1 Functional MPT model

The functional model simply considers a perfect conversion of the motor torque and angular speed into a force and linear velocity. This can be expressed simply by:

$$F_s = \frac{2\pi}{p} C_m, \qquad v_s = \frac{p}{2\pi} \omega_m. \tag{5}$$

where F_s and C_m are the transmitted force and the motor torque, respectively, and v_s the stroke linear velocity.

5.2.2 Basic MPT model

The basic MPT model considers the nut-screw inertia, friction and structural stiffness effects. Friction force is here considered as viscous and linearly dependent on the relative screw velocity. The elastic and damping forces relative to the nut-screw transmission are linearly dependent on relative position and velocity respectively.

5.2.3 Advanced MPT model

In the advanced MPT model, the screw friction is modeled in more detail and the backlash (with preload) effect is included. The former can be expressed as:

$$F_f = F_{cl} + F_p v_r + F_{st} \exp\left(-d_{st} v_r / v_{\text{ref}}\right) \tag{6}$$

where F_{cl} is the Coulomb friction constant force, F_p is a friction force proportional to the screw relative velocity v_r, F_{st} and v_{ref} are the Stribeck force and reference speed respectively, and d_{st} modulates its exponential decay.

Backlash is a "lost motion" happening within a mechanical transmission component, whereas preload is an elastic force occurring "while the motion is lost" [8,40]. Formally, the two combined effects are captured by a length-less component with terminals absolute positions expressed as x_1 and x_2, and whose produced elastic force is

$$F_e = \begin{cases} k_e(x_r - x_0), & x_r > x_0 \\ 2k_e x_r, & |x_r| \leq |x_0| \\ k_e(x_r + x_0), & x_r < x_0 \end{cases} \tag{7}$$

where $x_r = x_2 - x_1$ is the relative displacement, x_0 is the backlash displacement, and k_e a stiffness parameter. At the same time, its damping force is similarly expressed as

$$F_e = \begin{cases} \min(F_e, d_e v_e), & x_r > x_0 \\ 0, & |x_r| \leq x_0 \\ \min(-F_e, d_e v_e), & x_r < -x_0 \end{cases} \tag{8}$$

where $v_e = \frac{d}{dt}(x_r)$ and d_e is a damping coefficient.

5.3 Electric Motor modeling

The EM physical principles are the combination of effects acting in different physical domains: electrical, magnetic and mechanical. As such, the EM motor model choice affects the PDE interconnected model as well. Three EM models are proposed.

5.3.1 Level 1 EM model

This is the simplest model, where the demanded torque is purely applied to the rotor moment of inertia.

5.3.2 Level 2 EM model

In this case, the PMSM is described considering the stator phases, but using space phasor transformation and a rotor-fixed Air-Gap model at implementation level. The three-phase equations of a permanent-magnet isotropic synchronous machine can be written as:

$$v_{s,1} = R_s i_{s,1} + \dot{\Psi}_{s,1} \tag{9}$$
$$v_{s,2} = R_s i_{s,2} + \dot{\Psi}_{s,2} \tag{10}$$
$$v_{s,3} = R_s i_{s,3} + \dot{\Psi}_{s,3} \tag{11}$$
$$\Psi_{s,1} = L_s i_{s,1} + \Psi_{pm} \tag{12}$$
$$\Psi_{s,2} = L_s i_{s,2} + \Psi_{pm}(\theta_m - \tfrac{2}{3}\pi) \tag{13}$$
$$\Psi_{s,3} = L_s i_{s,3} + \Psi_{pm}(\theta_m - \tfrac{4}{3}\pi) \tag{14}$$

where $v_{s,1-3}$ and $i_{s,1-3}$ are voltages and currents per phase, R_s and L_s are the stator resistance and inductance respectively, $\Psi_{s,1-3}$ is the magnetic flux per phase, Ψ_{pm} is the flux linked with the stator due to the permanent magnets depending on the rotor position θ_m. By applying now the Park transformation, we can express the above equations in a stationary reference frame[13]; this transforms the three-phase machine into a two-phase machine, equipped with two windings fixed with the stator and orthogonal each other, expressed as d ('direct') and q ('quadrature'):

$$v_{s,d} = R_s i_{s,d} + \dot{\Psi}_{s,d} - \dot{\theta}_m L_s i_{s,q} \tag{15}$$
$$v_{s,q} = R_s i_{s,q} + \dot{\Psi}_{s,q} + \dot{\theta}_m L_s i_{s,d} + \dot{\theta}_m \Psi_{pm} \tag{16}$$
$$\Psi_{s,d} = L_s i_{s,d} + \Psi_{pm} \tag{17}$$
$$\Psi_{s,q} = L_s i_{s,q} \tag{18}$$

The expression of the torque [38] for such a machine is given by

$$C_{em} = n_p \Psi_{pm} i_{s,q}, \tag{19}$$

where n_p is the number of pole pairs, and justifies how, in the control strategy only the quadrature current is used to produce a torque[14].

5.3.3 Level 3 EM model

The most advanced Level 3 EM model replicates the Level 2, but adds several dissipative effects: 1. friction losses; 2. core losses (only eddy current losses, no hysteresis losses); 3. permanent magnet losses. Their equations are those implemented already in the machine losses MSL's *package*[15]. As for the friction, the friction torque τ is:

$$\tau/\tau_{\text{ref}} = (+\omega/\omega_{\text{ref}})^a \qquad \text{for } \omega > +\omega_{\text{lin}} \tag{20}$$
$$-\tau/\tau_{\text{ref}} = (-\omega/\omega_{\text{ref}})^a \qquad \text{for } \omega < -\omega_{\text{lin}} \tag{21}$$

[13] One axis has the same direction of the first winding magnetic axis.

[14] The direct current can be exploited for field weakening operations so to control the machine's operating regions. However, this goes beyond the scopes of this paper.

[15] More specifically, `Electrical.Machines.Losses`.

where the angular speed is ω, τ_{ref} is the reference torque at the reference angular speed ω_{ref}, a is a coefficient depending on the machine ventilation, and ω_{lin} defines the angular speed range in which the function is considered to be linear. Let us define the core power losses P_{core} as:

$$P_{\text{core}}/P_{\text{ref}} = (V/V_{\text{ref}})^2 \tag{22}$$

where V is the actual phase voltage and the subscript $-_{\text{ref}}$ has the same meaning as before. Lastly, permanent magnet losses are modeled dependent on current and speed. The permanent magnet losses are just considered through an equivalent braking torque at the shaft. The permanent magnet loss torque is

$$\tau_{\text{mag}} = P'_{\text{ref}}/\omega'_{\text{ref}}(c + (1-c)(i/i_{\text{ref}})^{a'})(\omega/\omega_{\text{ref}})^b \tag{23}$$

where i is the current of the machine, the parameter c designates the part of the permanent magnet losses independent of current. The dependency of the permanent magnet loss torque on the stator current is modeled by the exponent a'. The dependency of the permanent magnet loss torque on the angular velocity is modeled by the exponent b.

5.4 Power Drive Electronics modeling

The PDE are meant to feed the EM model appropriately. There are four PDE models: PDE Level 1 and 2 would be compatible with Level 1 EM model, whereas PDE models 3 and 4 would fit only EM Level 2 and 3 models[16].

5.4.1 Level 1 PDE model

The input torque is transferred directly to the output. There is no dynamics introduced.

5.4.2 Level 2 PDE model

The torque has a second order dynamics expressed as

$$C_{em}(s) = \frac{\omega_i^2}{s + 2\xi_i\omega_i s + \omega_i^2} C^*(s) \tag{24}$$

where ω_i is the current loop natural frequency and ξ_i its damping factor.

5.4.3 Level 3 PDE model

In Level 3 PDE model the input torque is transformed into the quadrature current $i^*_{s,q} = C^*/K_t$, where K_t is the motor torque constant, that constitutes the reference for the current closed loop, as in Figure 19. The direct and quadrature controllers output the respective reference voltages, $v_{s,d}$ and $v_{s,q}$. These are finally back-transformed into 3-phase via a reverse Park transform and fed into a perfect voltage source representing the inverter dynamics for operating the EM. The currents $i_{s,d}$ and $i_{s,q}$ are obtained via Hall sensors on the inverter output after a Park transformation.

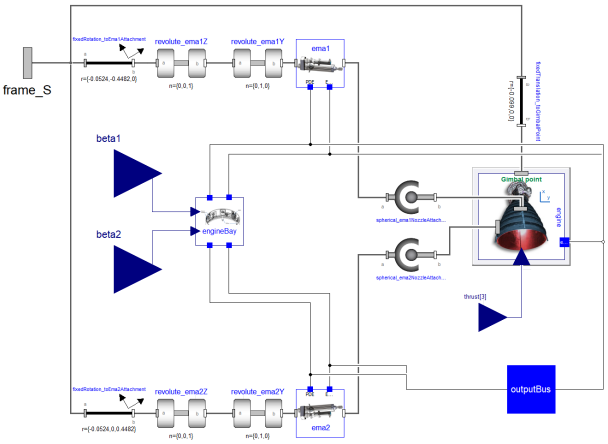

Fig. 20: `TvcSystem_v1` class diagram view.

5.4.4 Level 4 PDE model

Level 4 PDE model adds to the Level 3 implementation the Pulse-Width-Modulation (PWM) and inverter dynamics. Duty cycles for voltages $v_{s,1-3}$ are obtained, based on the DC voltage source available[17]. Then, a PWM signal is produced and fed into a 3-phases inverter model. The latter is composed by two transistors and two anti-parallel free wheeling diodes per phase. Diodes and transistors switching dynamic is modeled as well as in [38].

5.5 Library implementation

The full implementation of the TVC system relies on models spread in different sub-packages. The EM, PDE and MPT model variants are contained within `VLVLib.Actuators.TVC` *package*. Within the latter there are four main sub-*packages*: `ElectroMechanicalActuator`, `Assemblies`, `Interfaces` and `Tests`. The first implements the discussed EMA components together with the control system, while the seconds creates suitable interconenctions

[16] Note that, at implementation level, this means that the interfaces (*connectors*) are not always the same. In particular: "PDE Level 1-2 ↔ EM Level 1" would use a simple Real signal propagating the motor torque, while "PDE Level 3-4 ↔ EM Level 2-3" would use a multi-phase electrical interface.

[17] In launch vehicles this is usually a LiPo or thermal battery.

```
model TvcSystem_v1
 extends VLVLib.Parts.TvcEngineSystem.Architecture.TvcArchitecture(
  ema1(emaBody(
   redeclare TVC.Components.ElectroMechanicalActuator.ElectricalMotor.Motor_level1 motor,
   redeclare TVC.Components.ElectroMechanicalActuator.MechanicalTransmission.MPT_1dof_functional mpt)
   ),
  ema2(emaBody(
   redeclare TVC.Components.ElectroMechanicalActuator.ElectricalMotor.Motor_level1 motor,
   redeclare TVC.Components.ElectroMechanicalActuator.MechanicalTransmission.MPT_1dof_functional mpt)
  ),
   redeclare VLVLib.Actuators.Engine.Engine_v1 engine,
   spherical_ema1NozzleAttachment(w_rel_a_fixed=false, z_rel_a_fixed=false),
   spherical_ema2NozzleAttachment(w_rel_a_fixed=false),
   engineBay(
    redeclare TVC.Components.ElectroMechanicalActuator.PowerDriveElectronics.Pde_level2 pde1,
    redeclare TVC.Components.ElectroMechanicalActuator.PowerDriveElectronics.Pde_level2 pde2 )
 );
end TvcSystem_v1;
```

Fig. 21: The `TvcSystem_v1` class code.

between those. The third sub-package implements the necessary expandable busses to connect the components and store each component's variable to output. Lastly, the `Test` *package* implements unit tests.

The models are implemented by mainly exploiting the 1D `Mechanics`[18] and `Electrical` *packages* of the MSL. All model variants extends their respective interface *partial models* so to conform to the architecture-driven approach. On the other hand, the single component variants are stored in separated *models*. The same holds true for the simple engine model, that just includes its inertial dynamics. It is worth noticing that it is connected to the rocket structure by means of a spherical joint, on which a suitable structural stiffness and damping act. The EM Level 2 and 3 models require a big amount of parameters for the PMSM model to be correctly defined. For this reason, they are stored into a *record* and instantiated in both models. Naturally, Level 3 modifies the otherwise-zeroed friction parameters accordingly.

The `Assemblies` package contains component aggregates: `EngineBody`, `EngineBay` and `EngineMultibody`. The first models the actuator core, made out of the composition of the EM and MPT; `EngineBay` implements the two PDE units, while `EngineMultibody` instantiates `EmaBody` class and connects it with a prismatic joint (which belongs to the 3D Mechanics MSL's package[19]). Models within `Assemblies` do not perform any component redeclaration. This happens only in `VLVLib.Parts.TvcEngineSystem` *package* which contains the actual final implementations of the complete 3D TVC+Engine system. For example `TvcSystem_v1` *model* implements Level 1 EM, the functional MPT and Level 2 PDE models (for both EMAs within the complete TVC system). Figure 20 further explains the architecture, while the code is shown in Figure 21.

[18] Precisely `Mechanics.Translational` and `Mechanics.Rotational` *packages*.

[19] That is `Mechanics.Multibody` *package*.

6. Landing legs deployment model

Landing legs are fundamental for achieving reusability because they enable a soft touchdown. A few moments before landing, while the vehicle is facing the last descent part, the legs must be fully deployed: this is possible thanks to a mechanism able to release them from their folded position and push them outwards, such that gravity can start acting to further extract them and bring them towards their pre-defined final latching position. Gravity itself is not sufficient to win the aerodynamic forces, which strongly depend on the vehicle angle of attack and velocity. For this reason, an additional force is necessary to make sure that the deployment is complete and the legs can actually reach the fully unfolded configuration.

6.1 Leg deployment model

The discussed legs mechanical structure is based on CALLISTO reusable launch vehicle demonstrator. The configuration and the analysis of its performance is fully explained in [41]. In the following, the landing legs system is modeled only mechanically: in fact, the fluidic mech-

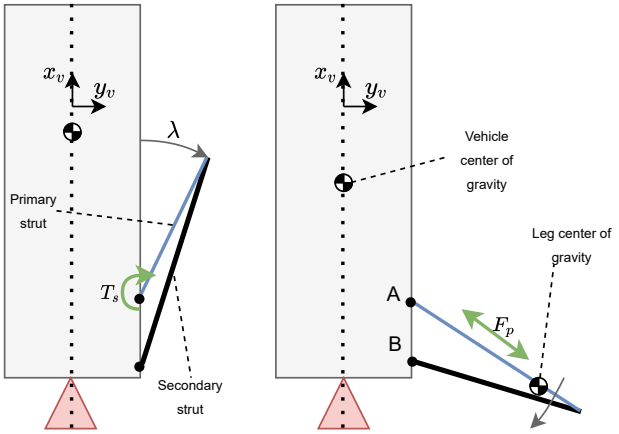

Fig. 22: Simplified view of the legs model.

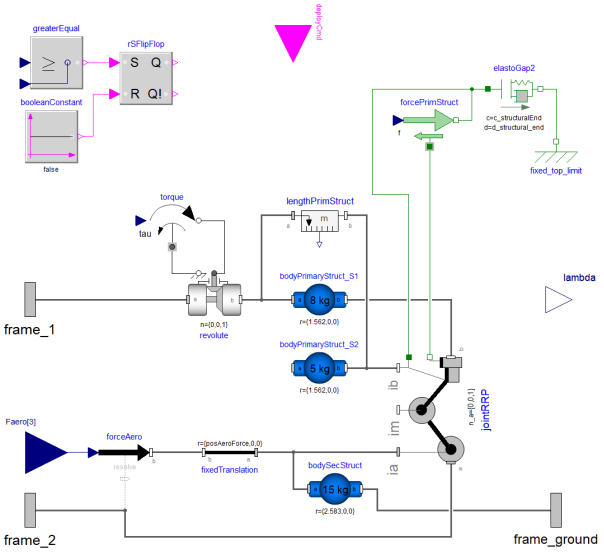

Fig. 23: `LegSimple` class diagram view.

anisms[20] meant to push out the legs is not considered. Figure 22 highlights the main model components. The primary strut can be seen as a telescopic rod enabling the legs unfolding, which operates thanks to a pneumatic force F_p. The secondary strut, instead, is in charge of holding the vehicle in position at touchdown and stand the heat stress during the deployment. At the deployment command trigger, a push-off mechanisms provides a minimal hinge torque to allow the leg to start the deployment; if this was absent, the gravitational force and pneumatic one would not cause the deployment angle λ to ever increase.

6.2 Library implementation

The difficulty in implementing a simulation model of a leg unfolding lies in the fact that the leg mechanical structure is a closed kinematic chain. Normally, with an open chain configuration the compiler is able to solve for the states of the next chain elements based on the current one. In closed chains, however, for each body there exists more than one path connecting to a uniquely defined set of states. Closed chains can be structurally non-singular or singular. In non-singular closed chains, the model is pre-processed in such a way that the tree roots are identified and the algebraic constraints determined.

The main problem of closed chains is that they can easily generate statically indeterminate systems (more equations than unknowns), like for example, in planar closed chains. The software symbolic manipulation applicable to a generic DAE system can not distinguish between consistent statically indeterminate systems (for which would be enough to ignore some equations) and inconsistent systems (thus defined by contradictory equations). The structural singularity of the DAE system is detected and the compilation fails. It is therefore necessary to eliminate the redundant equations before applying the symbolic manipulation.

The nonlinear algebraic constraint equations arising from most mechanical loops can be solved analytically However, it is difficult to perform this automatically. For this reason, the MSL has implemented a set of components where a predefined set of joints are already merged and the resulting equations of motion pre-solved and implemented. In this way, when a joint assembly takes part into a closed chain, the solver is able to solve for it. The component `JointRRP` implemented in the class `LegSimple` (Figure 23) has exactly this role. The same Figure also shows how the primary and secondary struts are implemented. The release mechanism acts as a torque at point A generated by a torsional spring which is active only when the deployment command has been triggered and λ angle is small (*e.g.* less that 1deg). The deployment command is a Boolean input which gets latched internally such that conditional expressions can be used to manage the force and torque acting on the leg. Note that a connector `frame_ground` is present. This allows the leg *model* to simulate the ground contact at the tip of the primary strut. This will be discussed in the next Section. The *model* `LegsAssembly` instantiates the four legs and can be used at vehicle class root level, as happening in `Callisto` *model*.

[20] In CALLISTO, a Helium-based pneumatic system is used.

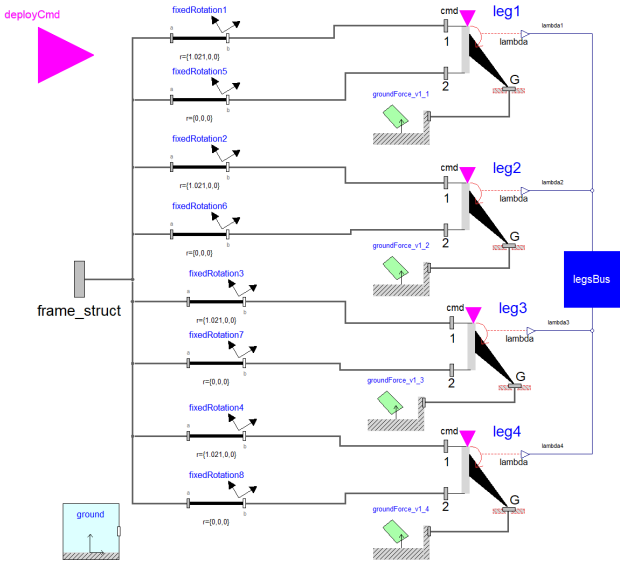

Fig. 24: `LegsAssembly` class diagram view.

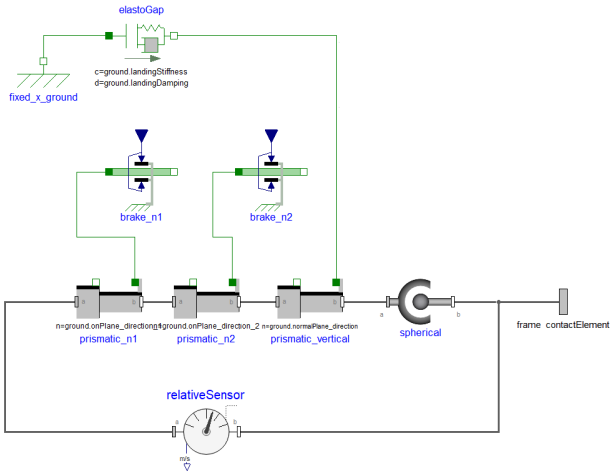

Fig. 25: `GroundForce_v1` class diagram view.

```
model GroundForce_v1
// <...> All declarations
  outer Ground ground;
equation
  if elastoGap.contact then
    brake_n1.f_normalized = 1;
    brake_n2.f_normalized = 1;
    else
    brake_n1.f_normalized = 0;
    brake_n2.f_normalized = 0;
  end if;
  // <...> Other connection statements
end TanksAssembly;
```

Fig. 26: Parts of the `GroundForce_v1` class code.

7. Ground contact modeling

The problem of modeling body impacts has been already addressed in [42,43]. Having an accurate representation of the behavior of the vehicle when it touches the ground may provide useful insights into the vehicle stability and the terminal landing conditions to be fulfilled to avoid vehicle tipping. This has been already cause of numerous failures in the early life stage of Falcon 9 rocket [44].

The approach used here is the penalty-based approach described in [42], where a high elastic and damping force is applied to the impacting body on the principles of a spring-damper mechanism. The price to pay with such a choice is that the simulation model becomes stiff, often implying a high computational cost.

7.1 Library implementation

The *models* to simulate ground impact are contained in the `Parts` *package*. Two components need to be instantiated: the `Ground` and the `GroundForce_v1` *models*. The `Ground` model contain all the properties for defining where the "ground" is located. This is fundamentally a fixed translation with respect to the inertial frame. At the same time, the reference frame defining the ground can be fixedly or continuously rotated with respect to the inertial. The same class also contains the definition of the "vertical" axis ("off-ground") and the "horizontal" axes ("on-ground") of the local ground frame. The `Ground` class defines pervasive properties of the overall model. For this reason, it has to be instantiated with the `inner` attribute, which makes available the instantiated component (*i.e.* `ground`) to all models down the hierarchy, provided it is recalled using the `outer` command, like in Figure 26.

The class `GroundForce_v1` (Figures 25 and 26) uses the properties defined in `Ground`. `GroundForce_v1` operates as follows: it connects[21] the inertial frame to `frame_contactElement` (*i.e.* the physical mechanical *connector* that must "expose" to the ground contact), but it enables all its degrees of freedom (three rotational and three translational). However, while the three prismatic joints allow free motion, they are set to exert specific forces as contact is occurring. On the "vertical" axis, an `ElastoGap` model is instantiated: this enables a free motion of the connected component until the ground is reached. At that

[21] Connecting two mechanical *connectors* corresponds to "welding" them.

point, a spring-damper mechanism starts acting avoiding the penetration into the ground of the impacting body. The contact presence is governed by an `ElastoGap` Boolean internal variable. When this is 'true', a breaking force acts on the remaining two translational degrees of freedom to avoid that the body "slips" onto the (fictitious) ground surface.

8. Simulation Results

In this Section, simulation results are shown for the four main considered dynamics with representative examples carried out onto CALLISTO[22], a VTVL reusable launch vehicle demonstrator being developed by DLR, CNES and JAXA in a trilateral project [45]. Its main objective is to demonstrate the capability to recover a vehicle with toss-back trajectory. This goal implies many sub-objectives for the propulsion system, such as in-flight engine reignition, propellant settling and sloshing management. An on-board computer has to run suitable robust algorithms for guidance, control and navigation during all phases for enabling vehicle stability and a safe landing.

8.1 Sloshing dynamics

To simulate propellant sloshing, the full end-to-end CALLISTO mission is considered. It has two tanks, one for the oxidizer (liquid oxygen - LOx) and one for the fuel (liquid hydrogen - LH_2), full at the take off. For both, only one sloshing mode is accounted for. Pendulum angles for both tanks obtained from a closed-loop simulation where the entire GNC system is implemented. This includes the control laws managing position and attitude, complex navigation algorithms for state estimation and a a guidance logic to provide the best control references over time.

In Figure 27, LH_2 tank pendulum angles with respect to vehicle's x- and y-axes are shown normalized. Even at this GNC system design early stage, the validity boundaries for the sloshing model are fulfilled, despite not directly visible in the plot for data confidentiality reasons.

At boostback, the vehicle experiences a null non-gravitational acceleration on its longitudinal axis; in this phase the liquid undergoes a chaotic motion and spreads in the tank. From a simulation viewpoint, this implies that the equivalent sloshing model is not valid anymore, and the pendulum motion must be inhibited via the boolean variable `suppressSloshing`. In fact, Figure 27 evidences how, in the highlighted area where `suppressSloshing=true`, the pendulum restores its neutral position and stops oscillating.

[22]Cooperative Action Leading to Launcher Innovation in Stage Toss back Operations

8.2 TVC system

The different versions for each EMA sub-component are shown in terms of step response to a 10mm screw position command. In the same plots, the difference of each response with respect to the one obtained using their respective simplest model is shown. The control scheme for all the proposed simulations is the same and is the one described in Section 5.1 and [37]. All parameters are taken from [37]. In this case, simulations are performed without any rocket model in the loop. This means that the vehicle-induced loads (hence the loads produced onto the TVC system as effect of the vehicle dynamics) are not considered. Note that the structural compliance is always modeled, hence there is an elastic (and damping) force always acting between the screw and the engine model. In fact, for rocket engines this cannot be neglected and usually has an high impact on the TVC control system design.

Figure 28 shows the differences between: 1. the screw position with infinite nut-screw stiffness (no nut-screw compliance); 2. the screw position with a finite nut-screw stiffness; 3. the load position with a finite nut-screw stiffness; 4. the load position with a finite nut-screw stiffness and force feedback compensation active. Only in this Figure, all simulations are performed without any structural damping, to demonstrate the capability of the force feedback compensation. In the following, this simplification is abandoned and the damping embedded in the architecture.

Figure 29 highlights the differences between the Functional, Basic and Advanced MPT models. As the friction is captured with more accurate models, the responses show less overshoot. Furthermore, the backlash effect (simulated with $x_0 = 0.1$mm) generates an initial delay in the Advanced model response, as well as high frequency dynamics visible only in the purple line due to some effect roughly definable as "mechanical chattering".

Figure 30 shows the responses of the EM models: here, the differences are much more contained. A slower dynamics in Level 2 and 3 models is justifiable with the effect of friction and other power losses.

Figure 31 shows the responses of the PDE models. Clearly, a difference between Level 2 and Level 1 exists due to the fact that Level 1 does not implement any dynamics. Level 3 and 4 are almost identical, as expected, due to the high frequency dynamics of the modulator and converter which can be fairly neglected in almost all GNC scenarios. However, Level 3 introduces an additional delay with respect to Level 2 due to the inverter diodes and transistors switching dynamics. Figure 32 also highlights

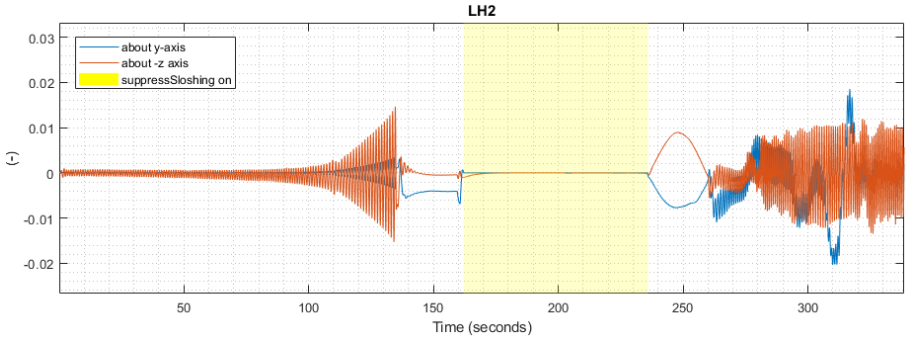

Fig. 27: CALLISTO normalized LH_2 pendulum angles.

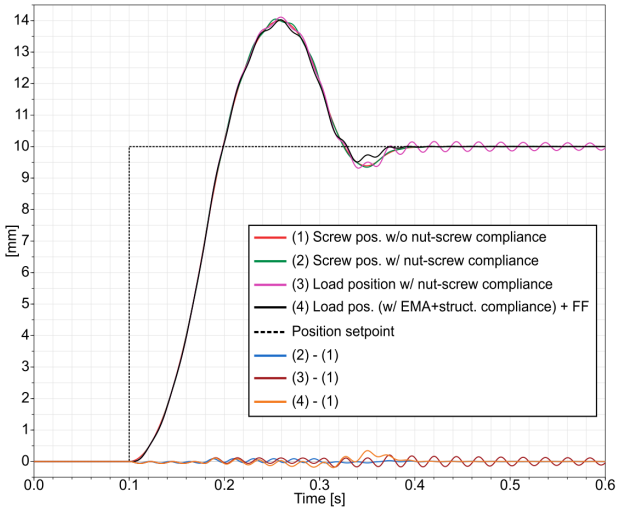

Fig. 28: Stroke position and load position comparison to highlight the control system performance.

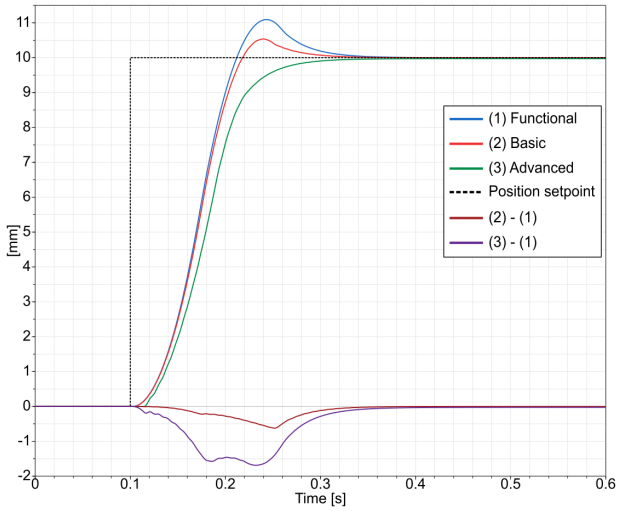

Fig. 29: Stroke position step response with the three different MPT models.

the measured quadrature current $i_{s,q}$ as compared to the controller current setpoint in Level 4 model.

8.3 Legs deployment

To show the legs deployment effects on the vehicle dynamics, CALLISTO vehicle is taken again as example. No real trajectory is simulated; here, the vehicle is freely falling from an altitude of 13m, with null Euler (3-2-1) angles, angular rates and velocity with respect to the inertial frame. It is supposed that the legs deployment command is generated at 0.1s and that each leg receives it with a sequential delay of 0.1s. More realistic conditions will be accounted for in subsequent works. Figure 34 shows that the angular rate of the vehicle is severely affected by the legs unfolding. Note that vehicle attitude is uncontrolled. Furthermore, there is no aerodynamic force acting on the legs which opposes the pneumatic force F_p. Lastly, the T_s and F_p values are not resembling the actual CALLISTO ones. Despite the simplifications, the effectiveness of the model is demonstrated and and could be made even more realistic in the future.

8.4 Ground contact

The simulated scenario is similar to the one before, but with the presence of a ground model. Due to the non-null attitude (caused by the legs deployment), the vehicle touchdown does not happen with the four legs all at the same time; the final dynamics is captured by the angular rates shown in Figures 35 and 36. Immediately after the first legs touch the ground, the vehicle starts wobbling

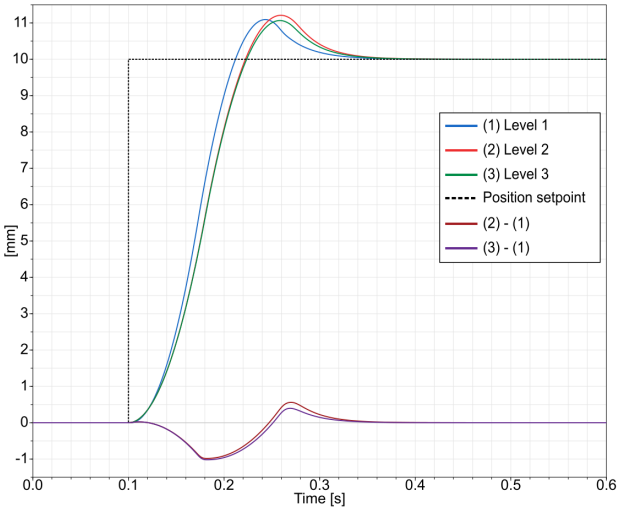

Fig. 30: Stroke position step response with the three different EM models.

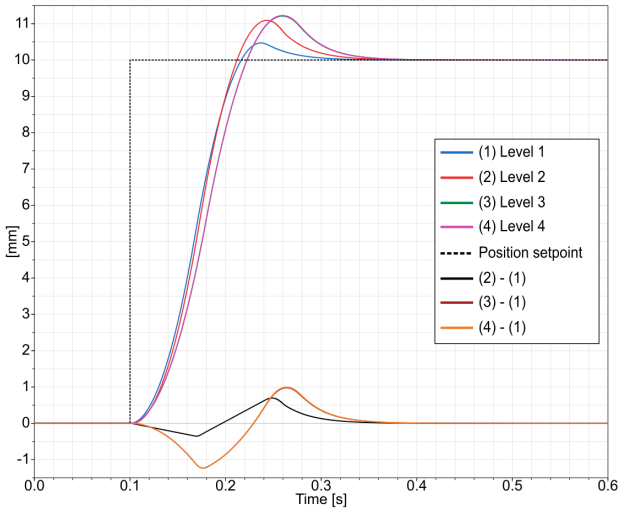

Fig. 31: Stroke position step response with the four different PDE models.

intermittently onto two legs periodically until this motion stops, after about 5s. Thus, the vehicle does not undergo any tipping. However, it is clear how the final stability depends on the touchdown attitude, but also on the ground-leg stiffness and damping factors.

9. Conclusions

This paper shows the potential of the Vertical Landing Vehicle Library (VLVLib) written using Modelica, an acausal object-oriented modeling language. The library implements several key dynamics of Vertical Landing (VL) Reusable Launch Vehicles (RLVs) sub-components, in order to produce advanced vehicle models and successively achieve a thorough GNC system testing, verification and validation. The modeling and implementation rationale have been explained and proved successful for maintaining a big flexibility when creating models with different fidelity levels, as well as managing a large amount of sub-models and interfaces. In this sense, Modelica language features, like inheritance, redeclarations, or fine class name scoping control, have been exploited at their full extent.

Four main dynamics affecting vehicle overall behavior have been inivestigated and the modeling strategy explained for each of them. They are propellant slosh dynamics, the TVC system, and the landing legs unfolding and the ground contact dynamics. Simulation results have been proposed to demonstrate the library implementation and that the resulting reusable launch vehicle overall model can be built so to include any of those dynamics of interest.

In conclusion, the presented library can perspectively integrate the GNC simulation, validation and verification work logic within any stage of a reusable launch vehicle development process.

References

[1] Harry Jones. The recent large reduction in space launch cost. In *48th International Conference on Environmental Systems*, 2018.

[2] Lars Blackmore. Autonomous Precision Landing of Space Rockets. *National Academy of Engineering: "The Bridge on Frontiers of Engineering*, 4(46):15–20, 2016.

[3] Lars Hoffman, Murielle Baker, Shane Glynn, Matt Darley, and Peter Beck. Reusable Electron: Analysis of Progress Toward the World's First Reusable Commercial Small Rocket. In *Small Satellite Conference*, page 7, Logan (Utah), USA, 2022.

[4] SpaceX Photos. CRS-8 first stage landing - Wikipedia, the free encyclopedia. Available at `https://commons.wikimedia.org/wiki/File:CRS-8_first_stage_landing_(26366878046).jpg`, 2016. [Online; accessed 31-August-2022].

[5] Dingyü Xue and YangQuan Chen. *System Simulation Techniques with MATLAB and Simulink*. John Wiley & Sons, 2014.

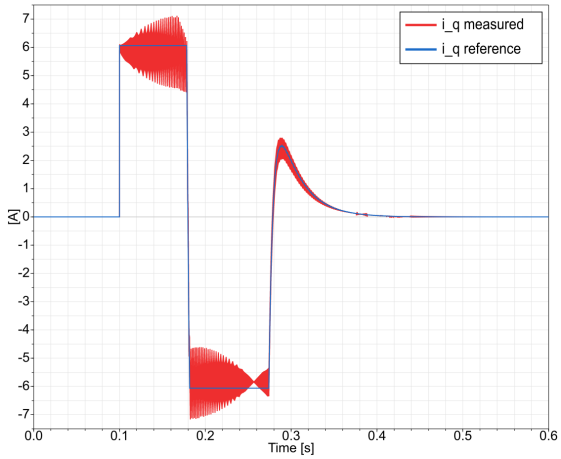

Fig. 32: Current dynamics as compared to its reference (Level 4 PDE model).

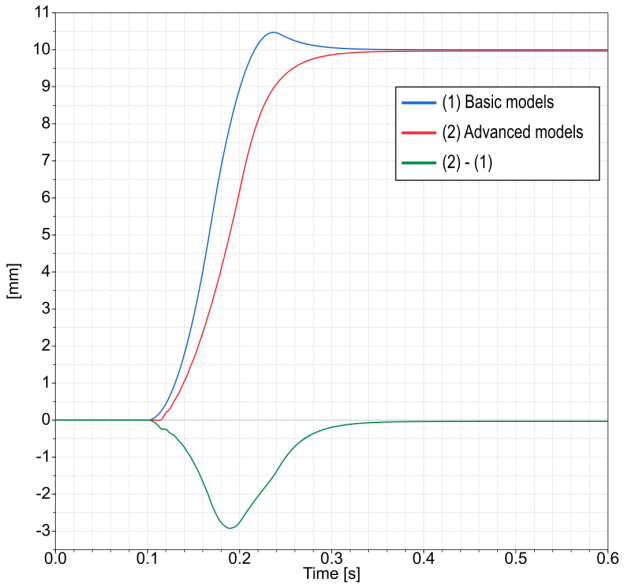

Fig. 33: Stroke position step response: most basic vs most advanced component models.

[6] Sven Erik Mattsson, Hilding Elmqvist, and Martin Otter. Physical system modeling with Modelica. *Control Engineering Practice*, 6(4):501–510, April 1998.

[7] Modelica Association. Modelica 3.4- A Unified Object-Oriented Language for Systems Modeling. Language Specification, 2017.

[8] Modelica Association. Modelica Standard Library. `https://github.com/modelica/ModelicaStandardLibrary`, January 2021.

[9] Martin Otter, Hilding Elmqvist, and Sven Erik Mattsson. The New Modelica MultiBody Library. In *3rd International Modelica Conference*, page 21, 2003.

[10] Stefano Farì and Davide Grande. Vector Field-based Guidance Development for Launch Vehicle Re-entry via Actuated Parafoil. In *Proceedings of the International Astronautical Congress*, volume D2, Dubai (UAE), October 2021. International Astronautical Federation.

[11] Jose Luis Redondo Gutierrez, Stefano Farì, and Matthias Winter. Control System Design for the ALINA Lunar Lander. In *Proceedings of the International Astronautical Congress*, volume A3, Dubai (UAE), October 2021. International Astronautical Federation.

[12] Stefano Farì, David Seelbinder, and Stephan Theil. Advanced GNC-oriented Modeling and Simulation of Vertical Landing Vehicles with Fuel Slosh Dynamics. *Paper submitted for publication*, 2022.

[13] Gertjan Looye. The New DLR Flight Dynamics Library. In *6th Modelica Conference*, page 10, 2008.

[14] T Pulecchi, F Casella, and M Lovera. A Modelica Library for Space Flight Dynamics. In *5th International Modelica Conference*, page 10, 2006.

[15] Lale Evrim Briese, Klaus Schnepper, and Paul Acquatella B. Advanced modeling and trajectory optimization framework for reusable launch vehicles. In *2018 IEEE Aerospace Conference*, pages 1–18, Big Sky, MT, March 2018. IEEE.

[16] Paul Acquatella and Matthias J. Reiner. Modelica stage separation dynamics modeling for End-to-End launch vehicle trajectory simulations. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings of the 10th International Modelica Conference - Lund, Sweden - Mar 10-12, 2014*, volume 96, pages 589–598, Lund, Sweden, March 2014. LiU Electronic Press.

[17] James A. Frosch and Donald P. Vallely. Saturn AS-501/S-IC flight control system design. *Journal of Spacecraft and Rockets*, 4(8):1003–1009, 1967.

[18] Cesare Carnevale and Pier Resta. Vega Electromechanical Thrust Vector Control Development. In

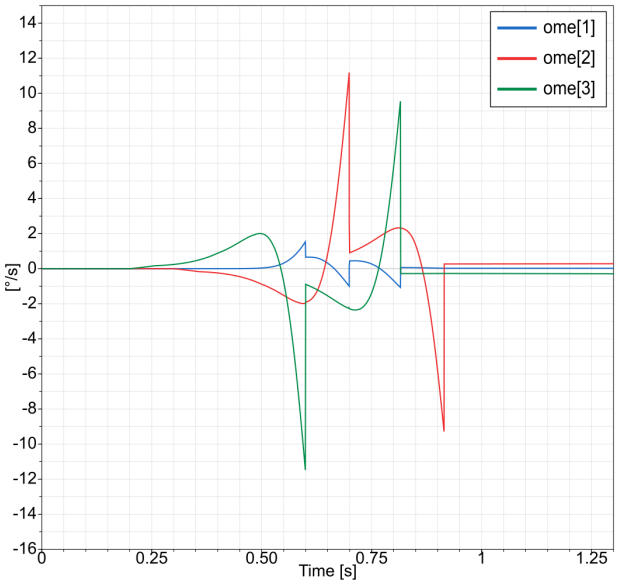

Fig. 34: CALLISTO angular rates as induced by the legs deployment.

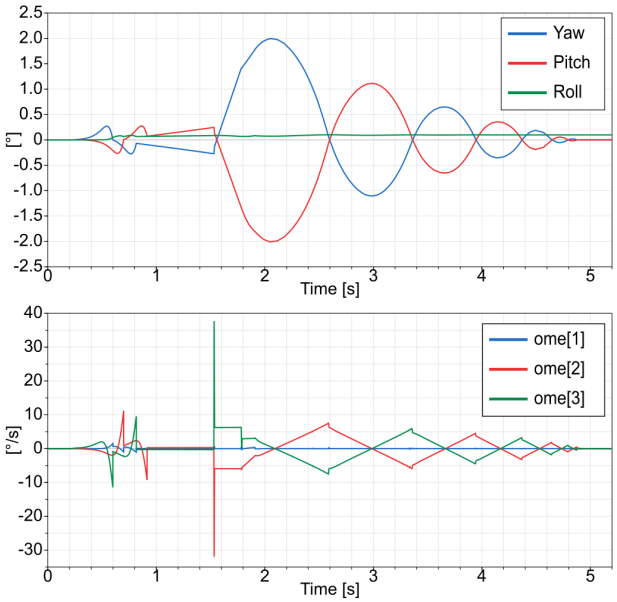

Fig. 35: Euler angles (3-2-1 rotation) and angular rates of CALLISTO at touchdown.

43rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, Cincinnati, OH, July 2007. American Institute of Aeronautics and Astronautics.

[19] Tillo Vanthuyne. An electrical thrust vector control system for the VEGA launcher. In *13th European Space Mechanisms and Tribology Symposium*, page 7, 2009.

[20] Jean-Charles Maré and Jian Fu. Review on signal-by-wire and power-by-wire actuation for more electric aircraft. *Chinese Journal of Aeronautics*, 30(3):857–870, June 2017.

[21] J. R. Cowan and Rae Ann Weir. Design and test of electromechanical actuators for thrust vector control. Technical report, NASA report, May 1993.

[22] Kaname Kawatsu, Seiji Tsutsumi, Miki Hirabayashi, and Daiwa Sato. Model-based fault diagnostics in an electromechanical actuator of reusable liquid rocket engine. In *AIAA Scitech 2020 Forum*, page 1624, 2020.

[23] Hiroshi Yamakawa, Ken Higuchi, Tatsuya Maruyama, and Nobuyuki Kubota. Flight evaluation of gn&c system and vertical landing dynamics of reusable rocket vehicle. In *10th AIAA/NAL-NASDA-ISAS International Space Planes and Hypersonic Systems and Technologies Conference*, page 1907, 2001.

[24] Dag Brück, Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Dymola for Multi-Engineering Modeling and Simulation. In *2nd International Modelica Conference*, pages 55–1, 55–8, 2002.

[25] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. John Wiley & Sons, 2014.

[26] Modelica Association. Documents. `https://modelica.org/documents.html`, 2021.

[27] Carla Martin, Alfonso Urquia, Jose Sanchez, and Sebastian Dormido. Interactive Simulation of Object-Oriented Hybrid Models, by Combined Use of Ejs, Matlab/Simulink and Modelica/Dymola. In *18th European Simulation Multiconference*, 2004.

[28] Torsten Kriening. Mission to the Moon and the return to Apollo 17. *European Planetary Science Congress*, 12:EPSC2018–819, September 2018.

[29] H. Norman Abramson. Dynamic behavior of liquids in moving containers with applications to space vehicle technology. Special Publication (SP) 19670006555, NASA, Washington DC, 1966.

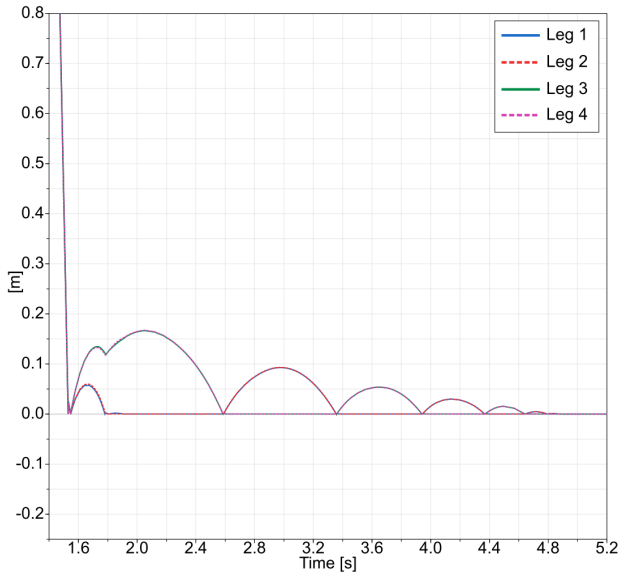

Fig. 36: Legs distance from ground during touchdown.

[30] Thomas A. Lance. Analysis of Propellant Slosh Dynamics and Generation of an Equivalent Mechanical Model f o r Use in Preliminary Voyager Autopilot Design Studies. Technical Memorandum 19680017770, NASA, 1966.

[31] Olivier Bayle, Valérie L'Hullier, Martine Ganet, Patrick Delpy, Jean-Louis Francart, and Daniel Paris. Influence of the ATV Propellant Sloshing on the GNC Performance. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Monterey, California, August 2002. American Institute of Aeronautics and Astronautics.

[32] Franklin T Dodge and San Antonio. *The New "Dynamic Behavior of Liquids in Moving Containers"*. Southwest Research Inst., 2000.

[33] H. Q. Yang and John Peugeot. Propellant Sloshing Parameter Extraction from CFD Analysis. In *46th AIAA Joint Propulsion Conference*, page 27, 2010.

[34] Raouf A Ibrahim. *Liquid Sloshing Dynamics: Theory and Applications*. Cambridge University Press, 2005.

[35] José G Pérez, Russel A Parks, and Daniel R Lazor. Validation of Slosh Model Parameters and Anti-Slosh Baffle Designs of Propellant Tanks by Using Lateral Slosh Testing. In *27th Aerospace Testing Seminar*, page 39, 2012.

[36] Jian Fu. *Incremental Virtual Prototyping of Electromechanical Actuators for Position Synchronization*. Doctoral thesis, Université de Toulouse, Toulouse, France, 2016.

[37] Jian Fu, Jean-Charles Maré, and Yongling Fu. Modelling and simulation of flight control electromechanical actuators with special focus on model architecting, multidisciplinary effects and power flows. *Chinese Journal of Aeronautics*, 30(1):47–65, February 2017.

[38] R. Krishnan. *Permanent Magnet Synchronous and Brushless DC Motor Drives*. CRC Press, first edition, December 2017.

[39] Cédric Renault. Usefulness of a force feedback on electromechanical actuator. In *Proceedings of the 6th International ESA Conference on Guidance, Navigation and Control Systems*, page 10, Loutraki, Greece, 2005.

[40] Mattias Nordin and Per-Olof Gutman. Controlling mechanical systems with backlash—a survey. *Automatica*, 38(10):1633–1649, October 2002.

[41] Anton Schneider, Jean Desmariaux, Josef Klevanski, Silvio Schröder, and Lars Witte. Deployment dynamics analysis of CALLISTO's approach and landing system. *CEAS Space Journal*, December 2021.

[42] Andreas Hofmann, Lars Mikelsons, Ines Gubsch, and Christian Schubert. Simulating Collisions within the Modelica MultiBody library. In *The 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, pages 949–957, March 2014.

[43] Felix Oestersötebier, Peng Wang, and Ansgar Trächtler. A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces. In *The 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, pages 929–937, March 2014.

[44] Etienne Dumont and Martin Sippel. Analysis of the Attempt of Launcher Stage Return by SpaceX' Falcon 9. In *SpaceX Workshop*, 2015.

[45] Etienne Dumont, Tobias Ecker, Christophe Chavagnac, Lars Witte, Jens Windelberg, Josef Klevanski, and Sofia Giagkozoglou. CALLISTO - Reusable VTVL launcher first stage demonstrator. In *Space Propulsion Conference 2018*, Seville, Spain, May 2018.