15

Multi-Functional Guidance, Navigation and Control Simulation Environment - Rapid Prototyping of Space Simulations

Erwin Mooij[1] and Marcel Ellenbroek[2]
[1]*Delft University of Technology, Faculty of Aerospace Engineering*
[2]*University of Twente, Department of Applied Mechanics*
[2]*Dutch Space B.V., Leiden*
The Netherlands

1. Introduction

Many space projects involve at one stage or the other extensive mission analysis, either to serve as an indication of system performance or as input to the design of sub-systems, such as the satellite's guidance, navigation and control (GNC) system. From the large difference in nature of these space projects one would expect a huge diversity of simulation models. A few typical examples include GPS satellites orbiting the Earth, the Voyager-1 and -2 flying in a heliocentric orbit through the solar system, Apollo's mission to the Moon, the European robotic spacecraft Giotto flying to Halley's comet and providing pictures of the cometary nucleus, Huygens entering the atmosphere of Saturn's moon Titan, and the Viking 1 and 2 spacecraft landing on Mars.
However, upon closer study it seems that there are many commonalities in both simulation models and simulation approach. Also the experience from several major projects has resulted in a generic approach for development, integration, verification and validation of on-board software for GNC, and Data/Handling systems (Mooij and Wijnands, 2002; Neefs and Haye, 2002; Mooij and Ellenbroek, 2007). This approach contains inter-connected paths for rapid prototyping, control-algorithm design and verification, on-board software development, and integration thereof with dedicated (flight) hardware in the control loop. To allow for a modular design of a particular simulator that is independent of the chosen spacecraft, (space) environment and mission, a (large) number of elementary functions and models is available to the user through a number of model libraries. These models can easily be combined by means of 'drag and drop'. In this way a significant cost reduction in terms of man-hours, as well as a short turnaround time can be achieved. Of course, this can only be guaranteed if each individual model is extensively tested and well documented.
Worldwide, MATLAB®/Simulink® is the most commonly used simulation environment for the design of control systems, not only in the aerospace industry, but also in, for instance, the automotive industry. So, for the sake of the current discussion, the programming environment of our choice is MATLAB®/Simulink®, although it must be stressed that the philosophy behind the generic simulation environment is independent of programming language.

The standard way to obtain source code that can be implemented in an external, real-time simulation environment is to use MATLAB®'s Real-Time Workshop. This toolbox supports the re-use of developed Simulink® models and hence results in a real-time simulator with identical models and a similar architecture. This allows for a simple model exchange between the two environments, as well as a sensible comparison of the results. However, there are several drawbacks to this approach. First, the generated source code will not be independent from the MATLAB® suite, since some (large) binary libraries have to be linked with the compiled source code to get an executable. Second, the generated code can be very complex at times and not be very readable. This makes a direct extension or adaptation of this source code a complex not to say impossible task.
If there would be a way to separate the actual application code from the MATLAB®/ Simulink® dependencies, but still use the architectural information of the Simulink® simulator we would have an efficient and flexible way to go from design simulator to real-time simulator and back. Fortunately, the solution is relatively simple. The application code can communicate with Simulink® by means of dedicated interface code (also called wrapper functions), and the Simulink® file can be parsed to extract architecture information. This information can subsequently be used to automate the setup of a real-time simulator. In conclusion, Simulink® should be used to set up and test the simulator architecture, and the combination of MATLAB® and Simulink® to design, analyse and test GNC systems. Once the testing is finalized, the application code can be transferred to the real-time environment and combined according to the architecture information.
In this chapter, the following aspects will be discussed in more detail. Starting out with a set of top-level requirements, the architecture of the generic GNC simulation environment will be discussed in Section 2, including an overview of all required (and available) library models. Section 3 discusses the verification, evaluation and validation of the simulation environment. In Section 4 a number of examples of increasing complexity will be presented to show the versatility of the presented modelling and simulation approach. Section 5 concludes this chapter with some final remarks.

2. The generic GNC simulation environment

2.1 System description

The GNC simulation environment is a toolbox facilitating the development of a dynamics simulator of a spacecraft and its natural environment. Such a simulator can not only be used for many different projects, but also in several simulation facilities during the full life cycle of the GNC system. This can vary from the design to the assessment of the functionality and performance. For instance, in the design phase of a GNC system, the dynamics simulator is initially applied in a non real-time Design Simulation Facility (DSF). After this phase, the on-board GNC software is designed and built, and for verification a real-time Software Verification Facility (SVF) with additional functionality is needed. For qualification purposes the real-time facility is further extended. In the operational phase of the spacecraft, important parts of the DSF may be reused in the so-called Spacecraft Training Facility (STF) and the Software Maintenance Facility (SMF), as well as the Electric Ground Support Equipment (EGSE) and the Operation Control Center (OCC).
The following main characteristics form the foundation of the generic GNC simulation environment:

- *System Description*
  In general, a satellite consists of a central bus and possibly one or more (flexible) appendages, such as solar arrays and communication antennae. To facilitate a realistic modelling of such a satellite, it can be built up from multiple bodies. To avoid overlapping data segments when, for instance, a complete satellite model is copied (or *instantiated*), in each library model the data segment is protected and can only be accessed by the model itself. The following configurations are possible:
  - Single rigid system with (or without) rigid appendages
  - Single rigid system with flexible appendages. To study the impact of flexible modes of an appendage on the performance of the guidance, navigation and control system, these appendages will be modelled as flexible bodies. They can be coupled with the main satellite body by dragging and dropping, and by properly connecting the input and output ports.
  - Multiple, free-flying rigid systems with or without (flexible) appendages. In close proximity, such a configuration allows for the study of formation flying.
  - Multiple, coupled rigid systems with or without (flexible) appendages. For rendezvous and docking missions it may be required to analyze the behaviour of the combined satellites before, during and after docking. Therefore, apart from the mentioned instantiation mechanism also a (flexible) link between two or more systems can be defined.
- *Model libraries*
  To define one of the above systems in a simulator, the user can combine building blocks from different model libraries, e.g., one with fundamental mathematical functions, such as coordinate transformations, matrix and vector operations, or one with satellite-dynamics models, space-environment aspects, and sensor and actuator models.
- *Verification & Validation*
  Each of the library models has to be verified and validated, such that the user is convinced of the proper functioning of the individual models. He should only focus on building the simulator, and possibly add some missing, project-specific functionality. Individual models should undergo unit testing, and combinations of models (so-called metamodels) should undergo system testing.
- *Documentation*
  The complete development of the model libraries should be extensively documented, thereby following the applicable procedures established by the European Cooperation for Space Standardization (ECSS, 2009). Included documents are the Software Requirements Document (SRD), Architectural Design Document (ADD), Interface Control Document (ICD), Detailed Design Document (DDD), the Software Verification and Validation Plan (SVVP), Test Reports (TR) and the Software User Manual (SUM).
- *Choice of inertial frame*
  To simulate missions that require a change in main attracting body, the user should be free to define an arbitrary inertial reference frame. In this way one can simulate, for instance, Earth-orbiting satellites, interplanetary missions, planetary entry and descent into the Martian atmosphere, and orbits around asteroids or moons of Jupiter.
- *Pre- and post-processing*
  Verification of dynamical models is always an important issue. Basic physical properties derived from conservation laws can aid the user in model and simulator

verification. For this reason, a dedicated library has been set up. This library is also used to create a clear architecture of the dynamics core, without the calculation of state-derived parameters. These will be implemented in separate library modules.

2.2 Development philosophy

Each of the aforementioned simulation facilities has the main parts of a dynamics simulator in common, even though they each have their own dedicated purpose. To avoid inefficiency and for a better control of the software simulating the dynamic system, we have established the need for a so-called Generic GNC Simulation (GGNCS) toolbox. As mentioned, this toolbox contains the fundamental models to build a dynamics simulator. Even though the architecture of the simulator is designed with Simulink®, the actual application software with the algorithms remains independent of the simulation environment.

The variety and simplicity of the available library blocks will lead to a common, modular simulator architecture with well defined input and output interfaces. Since the architecture will reflect the physics of the spacecraft system it has a clear and well-defined structure that facilitates the extension of the simulator architecture with sensor models, actuator models and the control logic. The initial architecture of the spacecraft in its environment is then not affected.

The modular simulator architecture simplifies the development of the simulator, because blocks can simply be replaced with more detailed models. When these blocks are added to the libraries, the functionality of the GGNCS Environment will evolve over time. To support this design philosophy the main characteristics introduced in the previous section need to be translated to some more specific design rules.

General

1. Concurrent Versions System (CVS) shall be used for configuration control of the GGNCS source code.
2. Problems (e.g., bugs) shall be reported using a Software Problem Report (SPR) tool
3. The GGNCS environment shall facilitate the re-use of knowledge and models from previous projects.

Simulator-architecture modularity

4. The architecture of the simulator shall be designed in the MATLAB®/Simulink® environment.
5. The architecture shall be defined by "drag and drop" of blocks from the GGNCS model libraries.
6. The physical system under consideration shall be clearly recognizable within the architecture.
7. The model blocks shall be combined in distinctive Simulink® libraries.
8. The GGNCS libraries shall contain sufficient model blocks to comply with the requirements defined in the Software System Specification
9. Each model block shall be coded with one particular functional property, with an easy verification and validation process; no complex mix of functionality is allowed in a single block.
10. Each model block shall be verified, validated and documented.
11. Certain combinations of frequently used blocks shall be combined in so-called meta blocks

12. No recompilation shall be needed when a user wants to investigate different spacecraft configurations or different missions. (After compilation of the source code binary libraries are obtained. These libraries are linked with the blocks in the Simulink® simulator. Since these libraries are not dependent on the architecture, there is no need for recompilation. The user *only* defines an architecture with existing libraries.)
13. The application part of the model blocks is coded in ANSII C. To be applicable in the MATLAB®/Simulink® environment, MATLAB®/Simulink® dependent interface code is added in a separate so-called wrapper function.

2.3 Top-level simulator architecture

In Fig. 1 the top-level system architecture of the GNC simulator is schematically depicted. The modelled "equations of motion" include the effect(s) of the changing inertia properties of the spacecraft and the contributions from the relevant loads. They can be split up in the environmental loads and the loads exerted by the spacecraft itself. The inertia of the spacecraft and the environmental loads are intrinsic to the flight dynamics of the spacecraft. In the rest of this chapter this part is referred to as the flight dynamics-model or simulator kernel.
The control loads applied by the spacecraft are introduced as externally applied loads and therefore input to the dynamic system. They stem from the actuators that are part of the Avionics and which are controlled by the GNC. To do this, the GNC requires information on the state of the spacecraft, which is provided by the sensor readings. As indicated, the actuators and sensors define the interface between the spacecraft flight dynamics and the GNC system. The type of the simulation facility that employs the dynamics simulator

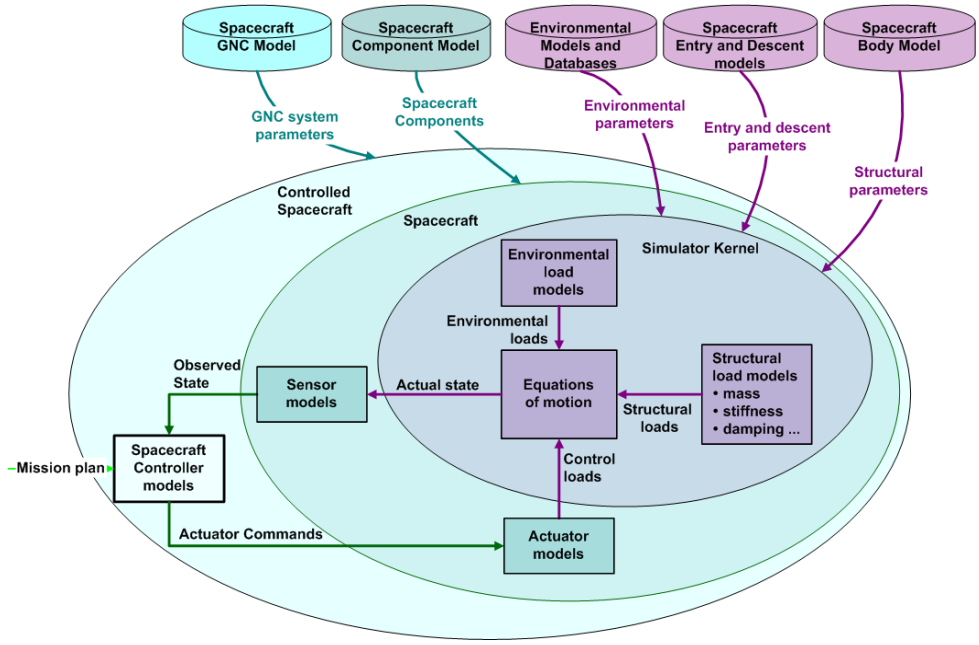

Fig. 1. Top level system architecture of the GNC simulator: the user configures a simulator by selecting spacecraft component models and environmental databases from libraries

defines the required details that are incorporated in the models, varying from simply functional to the actual hardware. The GNC simulator environment facilitates the simulation of the spacecraft dynamics and the evolution of the GNC system from the beginning with non-real time simulations till the actual hard real-time testing phase.
Since the flight-dynamics box has fixed inputs and outputs the simulator environment provides a library with sub-boxes to support the transformation of the actuator loads from one frame into another one, and the actual state into signals relevant for the type of sensor.
It is stressed once more that the top-level GNC system, sensors and actuators are for the user to fill in, as long as the input-output interfaces are met. A GNC system typically consists of a mission planner to provide reference signals, guidance algorithms to counter translational errors and control algorithms to do the same for rotational errors. In addition there is usually a state estimator that combines the sensor data into something sensible that the guidance and control algorithms can actually use. Such an estimator can simply be an equation that calculates a distance norm from three Cartesian position coordinates, or something more advanced like an Extended Kalman Filter that combines GPS and inertial measurements into a best estimate of position and velocity. The actual state is input to the sensor block, so in principle any state(-derived) value can be transformed to a sensor output. Depending on the level of detail, many different errors can be added to the sensor measurements. For the actuators a similar reasoning holds. Inputs to this block are actuator commands, issued by the control system. These can be commanded control-surface deflections for a winged re-entry vehicle, required moments for a reaction-wheel assemblage, or an average thruster moment for a pulsed, reaction-control system. However, the actuator block also enables the user to include any force and/or moment generating device that may not be controlled by the GNC system. Typical examples are a propulsion system that produces a constant thrust until it runs out of fuel, the aerodynamic properties of an entry vehicle, or a parachute system. It should also be clear, though, that the user is responsible to provide consistent data flows between sensors, GNC system and actuators.
Schematically, an abstract version of a dynamics simulator including the GNC units, as part of a number of simulator facilities, is shown in Fig. 2. In an industrial context, different companies may contribute to the definition, design, implementation and testing of the dynamics simulator. Moreover, the GNC units may be applied in different simulation facilities for different purposes. Therefore, to enable a controlled translation of software units (e.g., the actuator and sensor models), the unit models must be structured considering predefined I/O ports. Fig. 2 also identifies these interfaces ports required by the different facilities.
The I/O ports will usually not change going from one facility to the other. However, the interfaces may not always be known right from the beginning. Sometimes one assumes standard interfaces for simplified models (so-called level-1 models that consist mainly of the physical implementation), which may even reach the real-time simulator. As more detailed models will become available, the interfaces may need to be adapted. They may change in the real-time simulator, and for design purposes they will also have to be adapted in the design simulator. However, depending on the simulation facility, sometimes the interfaces change to such an extent, for instance due to the inclusion of hardware or software in-the-loop (HIL/SIL), and will then include a detailed communication interface. In that case there is no need anymore (or simply not possible) to match the interfaces in the design simulator. From that moment on the design simulator and real-time simulator become uncoupled. The standard models provided with the GNC simulation environment will all comply with this interface specification.

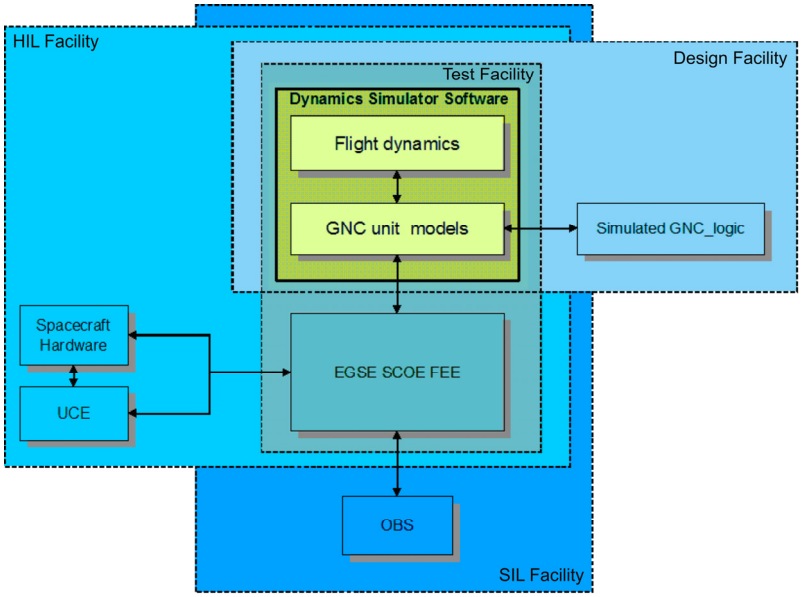

Fig. 2. Schematic view of the dynamic simulator as part of several simulation and test facilities

2.4 Library models

The idea behind the GGNCS Environment is that the software environment should be suitable for the development of a so-called end-to-end simulator. This simulator can be used for the complete lifecycle of a sub-system that is part of or can exert influence on the control of a spacecraft. Typically, this is the GNC system that executes on/with certain hardware, although it could be the hardware itself as well.

To achieve this the simulation environment should be a collection of libraries with predefined (functional) models that have a well-defined and documented interface. It includes everything that is required to simulate the operation of a GNC system (i.e., vehicle, environment, operations, etc.), but not the GNC system itself, although it would be possible to have a library with some pre-defined and tested GNC models to use for a quick closing of the loop. The so-called state vector that contains only that information for an unambiguous definition gives the state of the system. The state of the system is propagated in time by solving the equations of motion. These equations are derived starting with force (or moment) equilibrium using d'Alembert's Principle.

The core of any flight-dynamics simulator is thus formed by the equations of motion. These are typically a form of Newton's second law, which states: the acceleration produced by a force is directly proportional to the force and inversely proportional to the mass, which is being accelerated. This formulation holds in principle for systems of constant mass. By applying the so-called *Solidification Principle* one can use the same formulation for mass-varying systems when two apparent forces are added to the external forces, notably the Coriolis and relative force due to the mass variation (Cornelisse *et al.*, 1979). The Coriolis force can usually be ignored, whereas the relative force (originating from mass expulsion, or in other words, a thrust force) is commonly considered to be an external force.

Inspecting Newton's second law, i.e., $\sum F_{ext} = ma$ with $\sum F_{ext}$ is the sum of all external forces, m is the (current) mass of the system and a is the total acceleration of the system, one can derive the models required to simulate the motion of this system. In the first place, we need models for the external forces. These forces stem on one hand from the environment and on the other hand from hardware elements (i.e., actuators) such as the propulsion system. A satellite, for instance, is subjected to many aspects of the space environment, i.e., gravity of the main attracting body, gravitational perturbations due to third bodies (e.g., the Moon for an Earth-orbiting satellite), solar-radiation pressure, the magnetic field of, for instance, the Earth, and the atmosphere of some planets and moons. This means that we need environmental models that capture the perturbing effects with sufficient detail.

In the second place, we need models that describe the mass properties of the system. When a propulsion system is present that burns fuel, the current system mass needs to be updated accordingly. Because the simulator will not only be simulating translational motion, but also rotational motion, also the inertia properties will be changing in that case. And, to be able to calculate the external moments acting on the system, accurate information about the location of the centre of mass is required when external forces are not acting on this centre of mass.

In the third place, when we have isolated the acceleration a by dividing the total force by the current mass, we need to integrate this acceleration to calculate the change in velocity and position and obtain the state vector as a function of time. Although Simulink® provides a number of integration methods, independent integration methods are required for porting the simulator to a real-time environment.

The selected state variables for modelling the spacecraft systems are Cartesian coordinates for position and velocity, roll, pitch and yaw rate for the angular motion, and so-called quaternions to describe the attitude. Quaternions are derived from a rotational axis and the angle of rotation around this axis, and use four elements to describe the attitude versus the three angles that are actually required. However, while using three (Euler) angles there will be a singularity in the solution for certain attitudes, which make them not robust enough for a generic simulator. Quaternions do not have a singularity albeit at the expense of one extra variable.

To organize the typically large number of models they are grouped together in different libraries, sorted by functionality. Currently, there are 5 main libraries, i.e., the *Flight-Dynamics Library, Environment Library, Sensor & Actuator Library*, *Math library* and *Utility Library*. The underlying theory for the development of these libraries can be found in many textbooks, such as Wie (2008), Schaub and Junkins (2009), Montebruck and Gill (2000), Geradina and Cardona (1989) and Haug (1989). The *Flight-Dynamics Library* consists of the rigid-body models for calculating the accelerations and propagating the state vector, the external load calculation, i.e., due to solar radiation, atmosphere, magnetic field and gravitational field, as well as models to compute the mass properties of a time varying system that consists of multiple bodies. An extensive subset is formed by the flexible-body models, which will be discussed later in this section.

The *Environment Library* (see Fig. 3 for an overview) contains all models related to the space environment. Five categories can be discerned, i.e., gravity models (central field plus optional one or more zonal harmonic terms, and the extensive Earth GRIM-5 spherical harmonics model), magnetic-field models (central field and the spherical harmonics IGRF Earth magnetic field Epoch 1995), atmosphere models (exponential, tabulated MSIS86 models for different solar activity and the United States Standard Atmosphere 1976, plus an

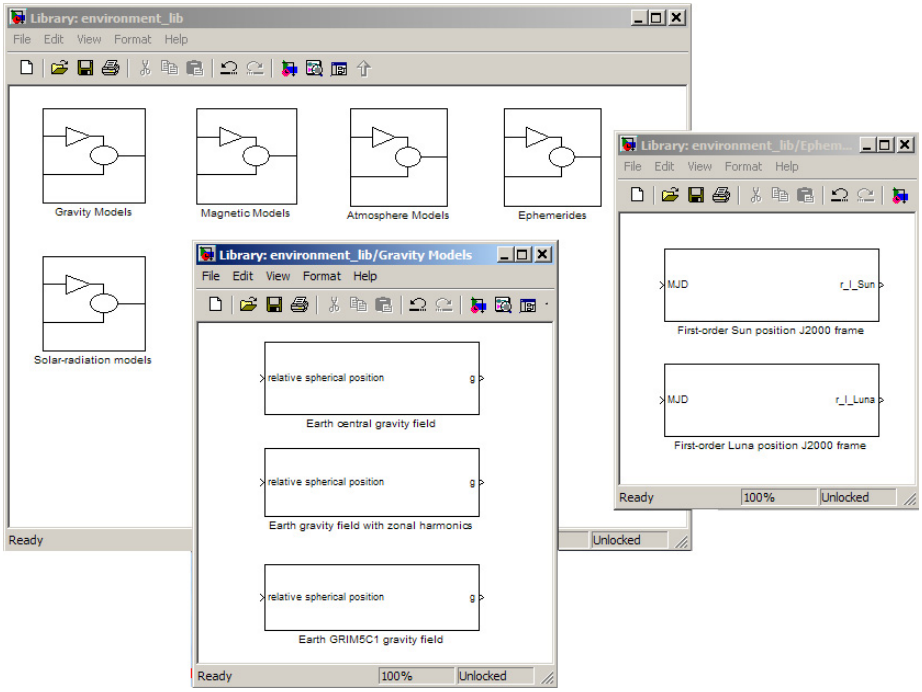

Fig. 3. The *Environment Library*, with the current gravity and ephemerides models detailed.

interface to the ESA Mars Climate Database (Forget et al. 2005)), ephemerides models (low-order orbit models for the Sun and the Earth's Moon), and solar-radiation models (inverse-squared distance solar pressure, eclipse status and illumination-factor calculation). The *Sensor & Actuator Library* contains currently only a limited number of functional models of a three-axis gyroscope (including error modelling), a star tracker, a fine sun sensor, a generic actuator model that adds different error sources to the input, a Reaction-Control System thruster and, finally, a three-wheel reaction-wheel assembly. Such a library will typically grow when dedicated sensors and actuators are developed in projects.

To obtain information about the system state in a format, different from the state variables, the user can define his own conversions, assisted by the availability of a number of standard conversions, stored in a so-called pre and post-processing library. A *Math Library*, as well as a *Utility Library* play an essential role in that vision. Of course there are many standard matrix operations available, such as matrix-vector and matrix-matrix multiplications, vector dot and cross product, transposing and inverting matrices, and quaternion multiplications. However, in this library also many functions are included that, for instance, transform Cartesian position and velocity to spherical components (e.g., latitude, longitude and flight heading) and back, quaternions to Euler angles, and those related to transformation matrices. When dealing with forces and moments from different sources, they are usually not all defined in one and the same reference frame. If the equations of motion require them to be expressed in the inertial frame (forces) or the body frame (moments), pre-defined transformation matrices can be used to transform them from typically any frame. Of course, there are also models available to go from quaternions or Euler angles to a transformation

matrix and back. With this tool set, the user can create his own output in almost any format that he wants. To facilitate analyzing the system behaviour models to calculate the kinetic, potential, elastic and total energy of the system, as well as the power, are included.

The *Multi-Body Library* consists of models that facilitate the design of standard satellite multi-body systems, i.e., a rigid central body with a number of (rigid or flexible) appendices attached to it. Common practice in multi-body dynamics is to model all the bodies independently of each other and to couple the bodies with constraint relations (Haug, 1989, Geradin and Cardona, 2001). However, this method has a number of disadvantages of which the most important one is that the run-time performance of the system decreases significantly. Since the simulator is also to be used in a real-time environment, a different solution has been chosen: the motion of all appendages is described relative to the geometric frame of the central body, the so-called G-frame. This formulation yields a minimum set of degrees of freedom (d.o.f.) to describe the equations of motion (Ellenbroek, 1994). Due to the relative motion of the (flexible) appendices to the satellite, the center of mass of the system moves with the motion of the appendices.. As a consequence the equations of motion cannot be obtained in the system center of mass. Therefore, it is decided to formulate a 'rigid-body' motion of the system by the motion of the G-frame in contrast to the commonly used Newton-Euler formulation. In the present *Multi-Body Library*, the relative motion in the joint between the appendix and the central body is assumed to have one or no d.o.f. relative to the satellite central body. This is sufficient for most satellite systems. It is then possible to model, for example, a solar array with a relative orientation that varies in orbit, or a momentum wheel that spins relative to the satellite.

The flexibility of an appendix is modelled assuming that the linear theory of elasticity is valid. This means that in an appendix reference frame that moves with the appendix it is allowed to use both a linear expression of the strain tensor and a linear relation between the elastic strains and stresses (Hooke's Law). The geometric non-linear motion of the appendix can thus be described with sufficient accuracy. To further improve the run-time performance the elastic deformation is modeled in terms of the sum of normal modes, each of which is multiplied with a time-dependant elastic degree of freedom (d.o.f.). In this way, one can decide to use only those modes that can be excited and that have a frequency in the range of interest of the controller (Ellenbroek, 1994).

Summarizing, the d.o.f. to formulate the motion of the satellite system are the position and orientation of the G-frame of the central body, the corresponding linear and angular velocity, the joint d.o.f. between the central body and the appendices, and finally the elastic d.o.f. and their time derivatives.

To derive the equations of motion of the satellite system, the structural properties of each appendix (mass distribution, stiffness and damping properties) and the loads that are acting on the appendix are first evaluated in the already mentioned appendix reference frame. The environmental parameters should therefore also be available in the same reference frame, which can be achieved by using the available transformations from the *Math* and *Utility Libraries*. Via the interface joint between appendix and central body this information can subsequently be transformed to the G-frame. Finally, all data from the appendices and the central body are assembled in the G-frame and the equations of motion are formulated. Solving the equations of motion provides the time derivative of the state vector, which is then integrated. Extracting the kinematics data of an appendix from the system state vector closes the loop. The *Multi-Body Library* has models for each of the steps that have been described above. In Fig. 11 (see Section 4.3), the top-level models are shown: extracting

kinematics data from the state vector, formulation of the appendix properties in the G-frame, and the formulation and solving of the equations of motion.

3. Verification and validation

The importance of the verification and validation process of the simulation environment should not be underestimated. Only when this environment is well documented, and verified and validated in a transparent manner, a user is willing to use the environment. Therefore, this aspect warrants a great deal of attention from the beginning of the project.
The ECSS standards on software development (ECSS, 2009) are used as guidelines. In line with these standards, among others the following documentation is written:

- The software system specification
- The architectural design document
- The detailed design document
- The verification and validation plan, and test reports
- The system environment release note

The consistency and correctness of these reports are checked, e.g., the traceability and verifiability of the requirements throughout the documents are carefully considered, and the correctness of the mathematical and physical formulations are verified.
To verify and validate software a number of methods is available. Verification of software means: "confirmation by examination and provision of objective evidence that specified requirements have been fulfilled". Validation of software means: "confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled". More information on this topic is given in ECSS (2009). The related methods are:

- *Inspection:*
  Compliancy with requirements is shown with standard quality control methods.
- *Review of Design:*
  Verification is achieved by validation of records, evidence of validated design documents or when approved documents show the requirement is met.
- *Analysis:*
  Compliance to specifications are verified by selected techniques as engineering analyses, statistics, computer and hardware simulations, and analogue modelling.
- *Similarity:*
  A specification is verified by similarity when it is similar in design to another specification that has already been verified
- *Test:*
  Compliancy to requirements is determined by using simulation techniques and the application of established principles and procedures. Testing is the most important method to verify requirements. It is used when verification by analysis is not sufficient. The first step in testing the environment concerns the verification of the basic mathematical functions, e.g., matrix multiplications, frame and co-ordinate transformations, etc. The next step applies to unit testing of more functional models. One can think of, for instance:
  - Time propagation, both in relative and absolute sense, i.e., simulation time (starting from t = 0) and mission time (related to the calendar date),

- Environment, e.g., consisting of the Earth's gravitational and magnetic field, the Earth's atmosphere, the motion of Moon and Sun and the interplanetary environment,
- Equations of motion, focusing on both translational and rotational motion, and the numerical aspects due to the integration of the differential equations,
- Perturbations, of gravitational origin, due to third-bodies (Sun and Moon), the Earth-magnetic field, the Solar radiation and the working of the upper atmosphere.

Finally, the modules integrated in a system simulator are tested on, e.g., a circular orbit around the Earth – to be discussed in Section 4.1. In that and subsequent sections, we will cover a number of representative tests as an example of the verification and validation process.

If during the verification and validation process errors are detected, they are reported and solved. To support this error handling process a dedicated SPR tool is available to facilitate the reporting and handling of software problems and software change requests, i.e. the problem is described, a problem "owner" is indicated, the priority to solve the problem is set as well as the severity, and when and how it is solved.

Finally, we strongly emphasize that the verification and validation activities are performed independently from the design and implementation activities. As an example of the verification and validation process the current section is ended with two representative tests.

Test 1 - Position of the Sun

The first test is meant to show that the low-order Sun orbit is correctly modelled. Astronomically the arrangement of the planes of the orbit of the Earth and its equator are such that the planes intersect at two times, the Equinoxes, when the length of the day and night are equal. Mid-way between these are the Solstices, when the Sun is at its highest and lowest in the sky at mid-day. These times can be determined very accurately and, as they occur near the times when the seasons are changing, have been used to indicate the start of each season. Thus, Spring is deemed to start at the Vernal Equinox (near March 21), Summer at the Summer Solstice (near June 21), Autumn at the Autumnal Equinox (near September 21) and Winter at the Winter Solstice (near December 21). In the southern hemisphere the cycle is displaced by half a year.

For the year 2001, the following simulated data are found for the location of Equinoxes and Solstices: Vernal Equinox: March 20, UTC 13:31, Autumnal Equinox: September 22, UTC 23:04, Summer Solstice: June 21, UTC 07:38, and Winter Solstice: December 21, UTC 19:21. Fig. 4. shows the relative position of the Sun with respect to the Earth-centred inertial frame. The resulting orbit is correct within the models accuracy of about 0.5-1% (Montenbruck and Gill, 2000). Plotting the orbit projections on different planes shows that the Earth's equatorial plane makes an angle ε with the ecliptic plane, notably $\varepsilon = 23.5°$. All these data are close to the expected values, so the model is assumed to be correct.

Test 2 - Magnetic field

The Earth magnetic field has been modelled as a dipole, with a strength of $7.96 \cdot 10^{15}$ Wb m in 1975 (Wertz, 1978). The "south" end of the dipole was in the northern hemisphere at 78.60° N latitude and 289.55° E longitude and drifting westward at about 0.014 °/year. The implemented model has been evaluated for the Earth's surface ($r = R_e = 6371.2$ km, as

specified for the IGRF), the result of which is shown in Fig. 5. Comparing this result with the plot for Epoch 1965 (Wertz, 1978) shows a good comparison, especially when taking the secular drift into account. Within reason, it can be stated that the current geomagnetic model has been properly implemented.

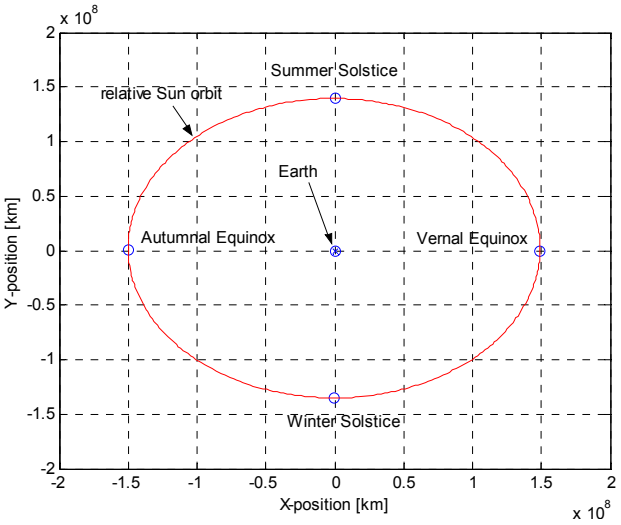

Fig. 4. Relative in-plane orbit of the Sun around the Earth. The Equinoxes appear at y = 0, whereas the Solstices appear at x = 0.

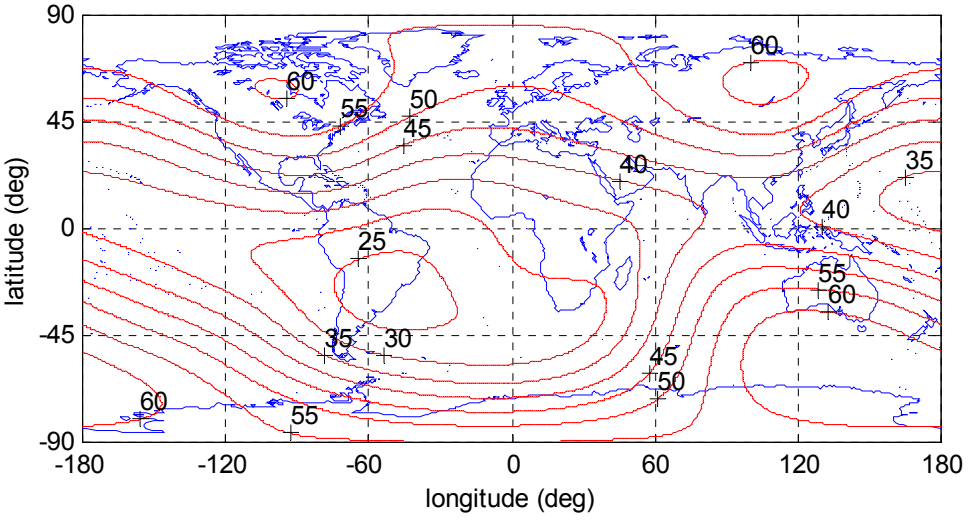

Fig. 5. Total magnetic field intensity at the Earth's surface (in μT Epoch 1995).

4. Applications

In this section a number of application examples of varying complexity is discussed. Starting with a single satellite, subsequent examples will build on this one by adding functionality. It is stressed that the examples focus on the versatility and ease of modelling of the GGNCS Environment, and not on the portability from a functional to a real-time simulator. Due to space limitations the reader is referred to two references for that. Mooij and Wijnands (2003) discuss the development of a complex satellite control system based on Model Reference Adaptive Control. A C-version of the simulator was implemented in the real-time simulation environment EuroSim, including communication-interface facilities in the form of a (hardware) MIL-1553 bus. In addition, a generic set-up was made for individual, real-time testing of the control algorithms. Neefs and Haye (2002) describe a strategy for the design of a set of simulation facilities for the development and flight-qualification of the Attitude Control and Measurement System of the Herschel/Planck satellites. A modular design for the simulation infrastructure complemented by a keen design of the simulation model software resulted in a set of (real-time) simulation facilities with one common design, and a single source for the simulations models.

4.1 Single satellite

In terms of space simulators, the model of a single satellite orbiting the Earth is a relatively simple one. In this example, we will show how to model the satellite dynamics, the space environment in terms of main attracting force and perturbing forces, starting from the elementary building blocks. We will show a particular type of orbits around the Earth, i.e., a highly eccentric orbit of the Molnyia type. This orbit has the characteristic that it stays over Russia for a long time. The Russians use this type of orbit for telecommunication purposes.
The architecture of the simulator for a single satellite is shown in Fig. 6. The satellite consists of a central body and two solar arrays. Each of the three bodies includes its own space environment, as its influence on the body is a function of not only position and velocity, but also the individual orientation. Within the body sub-system the mass properties and the

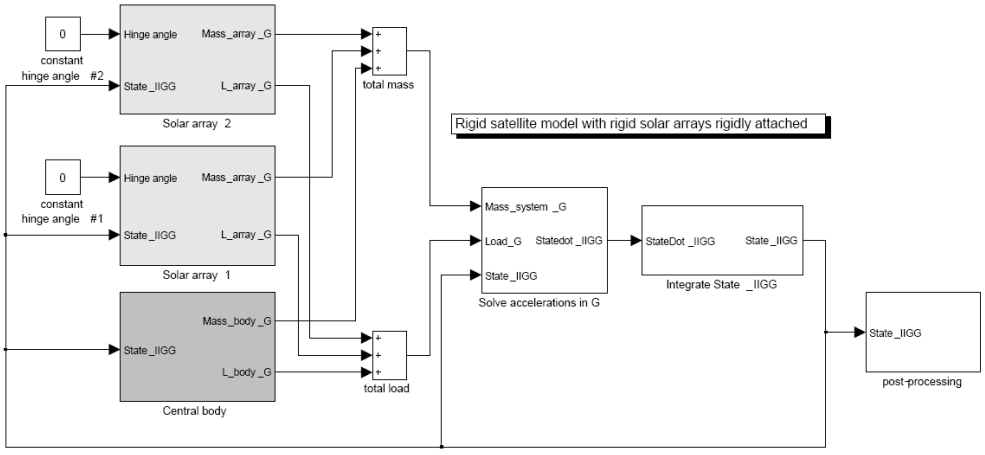

Fig. 6. Top-level architecture of a single-satellite simulator.

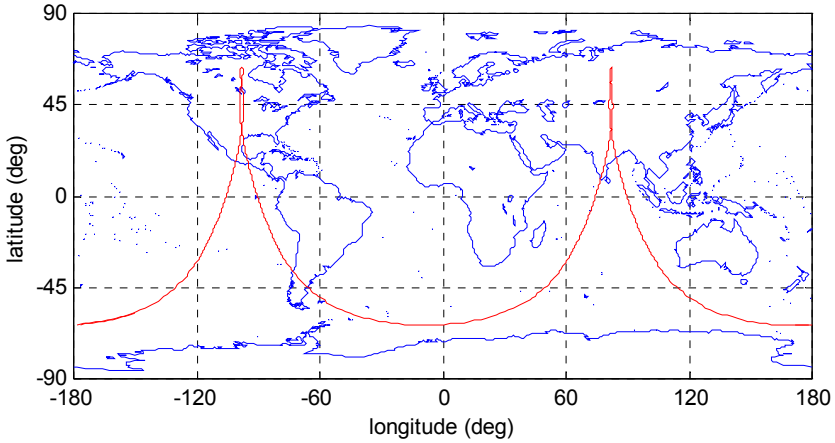

Fig. 7. – 24h groundtrack of a sample Molniya-type orbit (a = 26,555 km, e = 0.7222, i = 63.4°, ω = 270.0°, T = 12h.0).

external forces are calculated, which are combined before the calculation of the total acceleration takes place. This acceleration is resolved in the earlier mentioned G-frame to be in line with the later example of a satellite with flexible appendages (Section 4.3). Once the acceleration has been calculated it is integrated to a new state. Note that the current satellite state, shown as a dataline from the block *Integrate State_IIGG*, is required in several blocks. These blocks are the three bodies (to calculate the external loads) and the block *Solve accelerations in G*, because of the kinematic relation between position and velocity. In the post-processing block, amongst others the inertial, Cartesian position is converted to latitude and longitude, which will allow us to plot a groundtrack.

Simulating the Molnya orbit results in the groundtrack shown in Fig. 7. When compared with the corresponding plot in Montenbruck and Gill (2000), it shows a close resemblance. There is only a shift of the points of intersection with the equator - the date and time of day for the orbit propagation was chosen arbitrarily, and was obviously different from the orbit given by Montenbruck and Gill.

In a second test, the above simulation model is extended with a GNC system, as well as sensors and actuators. The mission objective for the satellite is now to permanently "look down" towards the Earth's surface (as if there was an Earth observation instrument). The new simulator architecture is shown in Fig. 8 . In this figure, the indicated simulator kernel (the grey block) is identical to the complete simulator shown in Fig. 6, i.e., a satellite with two rigid solar panels. We have added three sub-systems to the top level, i.e., the *Sensors*, *Actuators* and the *GNC_Logic*, consisting of a *Mission Manager*, *Navigation Filters*, *Guidance Logic* and *Control Algorithms*. It may be obvious that a detailed discussion of the design of the *GNC_Logic* is beyond the scope of the current example, so we will suffice with a high-level description.

For the current example, both the sensors and actuators are modelled as ideal systems. The satellite state vector is only separated in a measured orbit state and a measured attitude state. The *Mission Manager* provides the control setpoints, i..e, the required attitude to look down towards the Earth. This corresponds with a fixed satellite orientation with respect to the local horizontal plane, i.e., the plane tangential to the Earth's surface. The *Navigation*

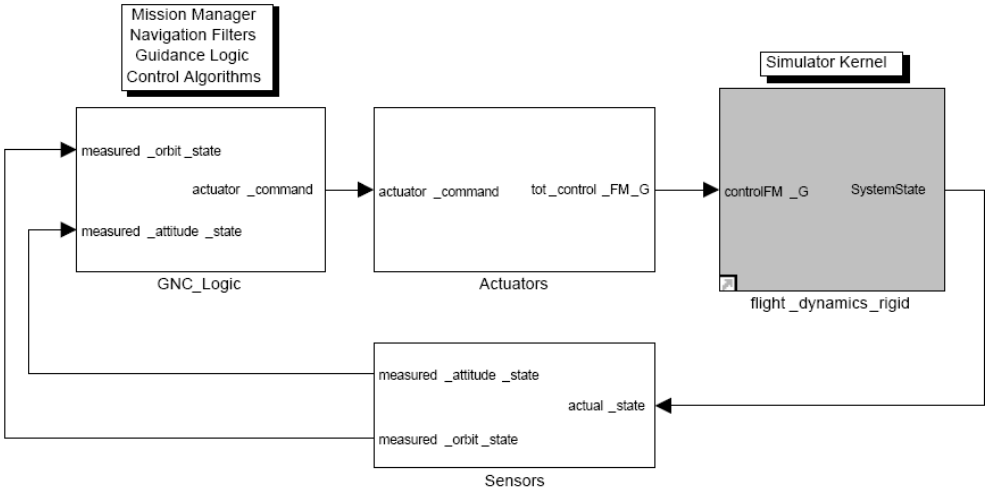

Fig. 8. Top-level architecture of a single-satellite avionics simulator for control-system testing.

Filter converts the satellite attitude (i.e., the quaternions) to the local attitude angles, with standard blocks from the *Math & Utility* library. The difference between the setpoint and the actual angles (the *control error*) enters the *Control Algorithm* (a simple proportional-derivative controller) and calculates the corrective moment to reduce the control error to zero. This corrective moment enters the *Actuators* block as *actuator_command*. The commanded moments pass through the actuator block and enter the *Simulator Kernel*. There, the angular accelerations are calculated and integrated to a new state vector. The results show that indeed the satellite is "looking down" all the time.

Summarizing, in the first example we simulated only the orbit (with so-called three degrees of freedom), without any orbit control. In the second example we added attitude control and set up the architecture for a simulator with which one can test GNC systems. Although the sensors and actuators were modelled as ideal systems, given the interfaces we can replace these ideal systems with more realistic models. That would allow us to do a detailed analysis of the GNC-system performance.

4.2 Formation flying

Building on the previous example, we will show the orbit characteristics of four satellites flying in formation. In essence, it means that we instantiate (or copy) the single satellite model (in its space environment) four times, and use the related output of each satellite to calculate the relative motion. In essence, we could either copy the simulator in Fig. 6 (which became the grey block in Fig. 8.), or we can go one step further by creating a meta block of the simulator shown in Fig. 8. This would allow us to study formation flying with fully controllable satellites. For the current example this is arbitrary, since we will only simulate the open-loop orbits of the four formation-flying satellites.

In Fig. 9. the top-level architecture of the simulator is shown. Each of the four blocks represents a satellite with two rigid solar panels, as introduced in Section 4.1. The Master

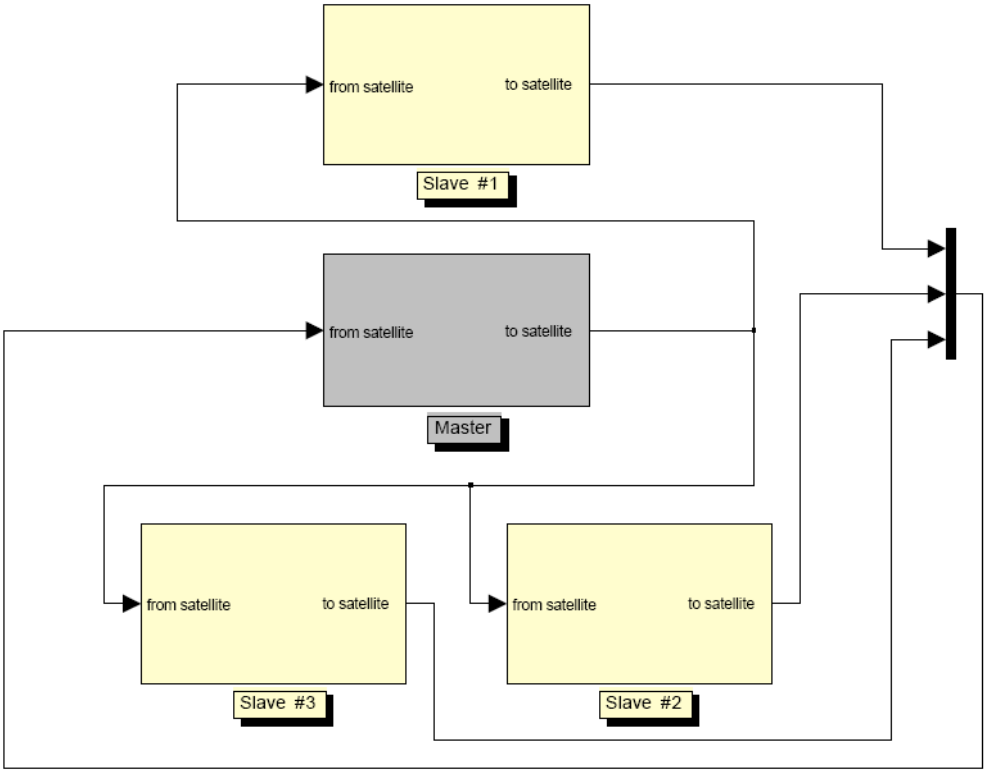

Fig. 9. Formation-flying simulator for four satellites.

satellite takes the state vector of each of the three Slave satellites as input, whereas the Slaves only get information from the Master. This configuration is quite common, where a single Master controls the formation based on input it receives from the Slaves. In this example, however, we do not control the Master nor the slaves, and the input to the Master is only used for post-processing purposes, i.e., to calculate the relative position difference between each of the Slaves and the Master.

The orbit of the Master satellite is circular. The variation of the orbital elements is selected in such a way that the position distance for each of the slaves starts in the range of 1000 m. Running the simulation, yields the results of Fig. 10. The curves show the relative motion in along-track and cross-track direction of the three slave satellites with eccentricity differences of $\Delta e_1 = 0.0001$, $\Delta e_2 = 0.0002$ and $\Delta e_3 = 0.0003$. It is clear that each of the Slave satellites follows a perfect ellipse around the Master satellite. And, at the same time the formation orbits the Earth in a circular orbit (not shown here). This behaviour is in line with results found in Schaub and Junkins (2009).

Concluding, this example shows that it is easy to instantiate a complete satellite and simultaneously simulate multiple satellites. The data segments in each satellite block are properly shielded from each other.

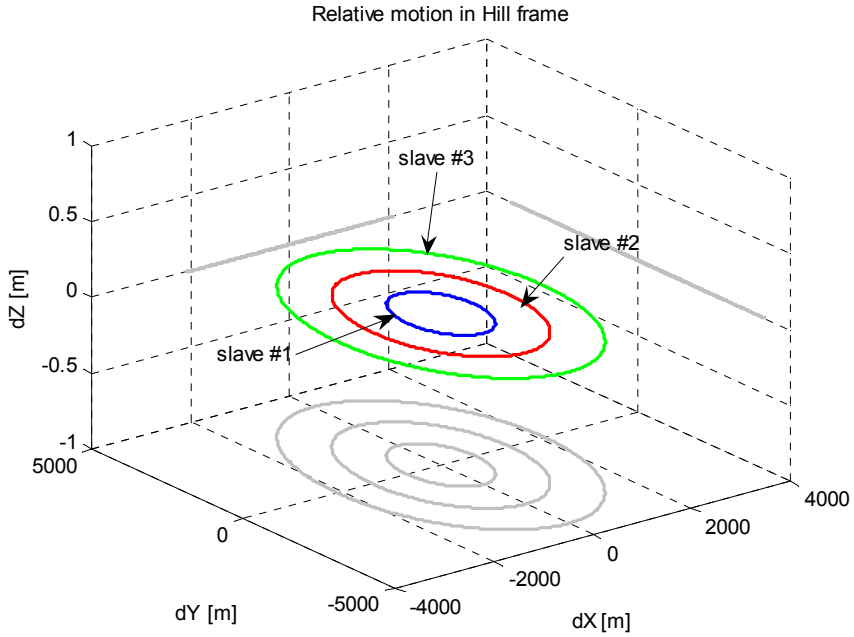

Fig. 10. Simulation of relative motion for four formation-flying satellites in an equatorial orbit with small difference in eccentricity.

4.3 Multi-body Satellite with flexible appendages

To test the implementation of flexible bodies in the GGNCS Environment a satellite with a rigid central body and two flexible solar arrays attached to it is simulated. Each body is defined in its own body frame. The position and orientation of the solar-array reference frames are defined relatively to the reference frame of the central body. The solar arrays are connected to the central body by a revolute joint, which allows for a single degree of freedom rotation. The elastic deformation of the solar array is calculated locally in the solar-array reference frame. The resulting model of the solar array is described in terms of a mass, stiffness and damping matrix. The mass matrix depends on the elastic deformations, whereas the stiffness and damping matrix are constant. They only depend on the spatial deformation shapes. The time dependence of the elastic deformations is introduced by so-called generalized coordinates, which are included in the state-vector. The mass matrix of the undeformed body, and the stiffness and damping matrix are derived with the aid of an accurate finite-element model of the solar arrays.

Fig. 11. shows the MATLAB®/Simulink® architecture of the described satellite system. The satellite central body and both solar arrays are clearly identified. The kinematics of each solar array in its own frame is extracted from the state-vector by the blocks *SA kinem 1* and *SA kinem 2*. The blocks *satellite body*, *solar array 1* and *solar array 2* calculate the mass matrix, the time-varying mass properties and the loads per body. The equations of motions are assembled and solved for the time derivative of the state-vector in block *Solve EquationsOfMotion*. This block also performs the integration, so that in the end the updated state-vector is obtained.

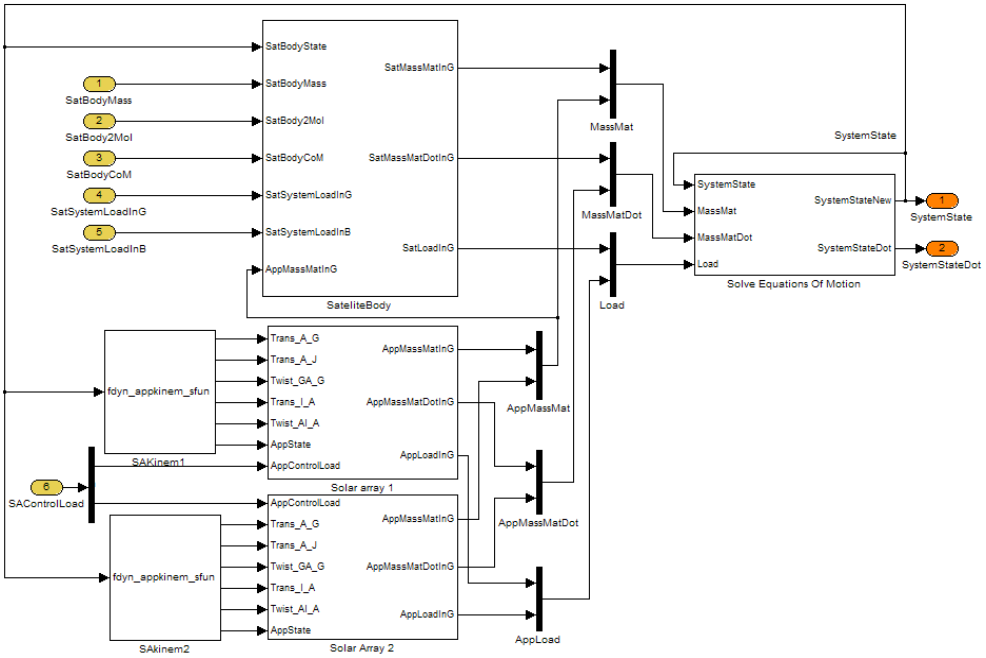

Fig. 11. The architecture of the satellite system with a central body two flexible appendices

To verify the correct implementation of the flexible appendages in the MATLAB®/Simulink® environment, the simulation results obtained with the model of the rigid central body with two flexible solar arrays were compared with the results from the multi-body package DCAP (Franco et al., 1996). It is then observed that the system mass matrix and the eigenfrequencies of the total system obtained in both simulation environments are exactly the same. Also the time histories of the displacements, orientations and velocities showed the same behaviour. Finally, the energy conservation and power balance laws were verified.

Some of these results are presented with the following simple example. Consider the case where a step moment of 1 Nm about the Z-axis of the G-frame is applied to the central body after 1 second. The solar array joints are fixed and no further environmental load is introduced. The flexibility in each solar array is modelled with 9 normal modes. The corresponding frequencies varied between 0.25 Hz and 7.37 Hz. The data are obtained from the linear module of the finite-element software MSC/NASTRAN. The power balance is shown in Fig. 12. It clearly shows the presence of the flexible modes. Since the elastic and damping loads are added to the load vector, the power from the external loads includes the elastic power and the damping power. To show that the elastic modes are indeed active, Fig. 13 shows the power due the elastic load, the damping loads and also the elastic energy. The power of the loads can also be derived to verify the power flow in the system. In fact, the ΔPower defines how accurate the equations of motions are solved. Although the presented figures are only illustrative, they show the use of the post-processing modules to verify the GGNCS Environment.

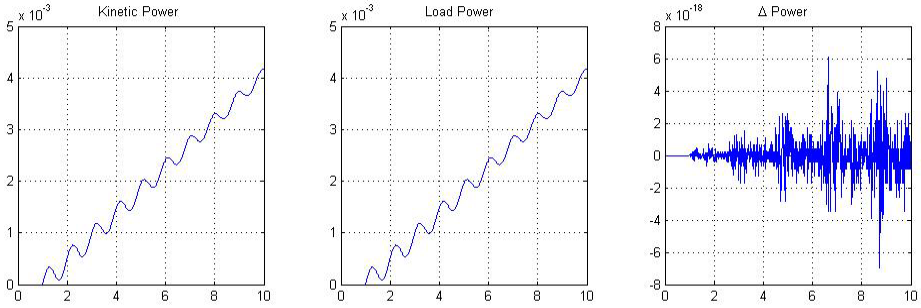

Fig. 12. Time history of the power conservation laws

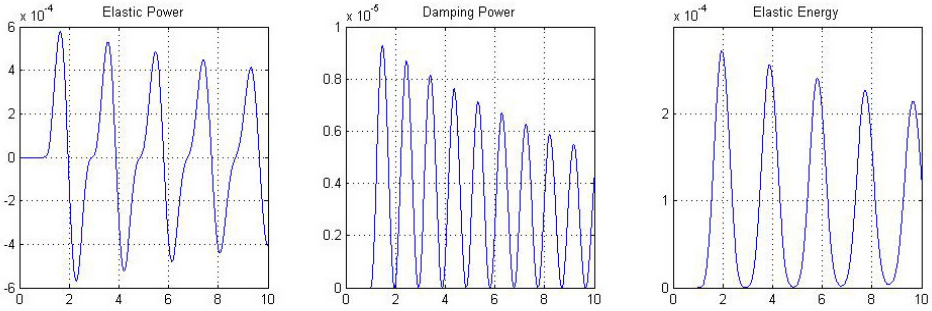

Fig. 13. Time history of the elastic energy and power

4.4 Re-entry vehicle

To illustrate the use of the simulation environment for atmospheric flight, we show two simple examples of a vehicle entering a planetary atmosphere. The first example deals with an uncontrolled entry of an entry capsule in the Earth's atmosphere. The model that is used is that of an Apollo-like entry capsule (mass m = 4976 kg), which is on a return leg from the Moon. It enters the atmosphere at 220 km altitude with a relative velocity of V = 11 km/s. The corresponding flight-path angle γ = -9.536°, which means that the velocity vector is below the local horizon.

Of course, the motion of the entry capsule is still governed by Newton's second law, but compared to the satellite examples shown earlier, the space environment is different. For Earth-orbiting satellites the dominating force is the gravitational force of the main attracting body (the Earth). The influence of the atmosphere gives rise to perturbing accelerations at most. In case of a vehicle entering a planetary atmosphere, the gravitational acceleration is mostly of secondary importance when compared to the very large aerodynamic forces (and moments).

So, compared with the previous examples the simulator will include some additional models. The simulator kernel (Fig. 8.) can in this case be somewhat simplified: of course, the

solar arrays can be removed. The satellite central body can be treated as the entry capsule. However, the flight of the entry capsule will be inside the atmosphere and the main force will be of aerodynamic origin. Therefore, the aerodynamic characteristics of the vehicle need to be modelled much more accurately than that of a satellite that is only perturbed by atmospheric drag. Since we do not want to change the (generic) flight dynamics kernel, it is most obvious to include the aerodynamic force and moment model in the *Actuators* block. However, since the aerodynamic properties are dependent on the actual state-derived parameters (angle of attack, angle of sideslip, Mach number, dynamic pressure) we need to feedback these data from the kernel.
In Fig. 14. the adapted *Actuators* block is shown. Apart from the reaction-control thrusters – the actual actuators – two blocks have been added. One block calculates and outputs the aerodynamic forces and moments in the body frame. The actual implementation is, of course, depending on the available aerodynamic data. In the case of Apollo, the data came from wind-tunnel measurements and consisted of several (tabulated) force and moment coefficients as a function of Mach number, angle of attack and angle of sideslip. Linear interpolation was used to obtain the actual values as a function of the flight condition. A second block was added for future use, i.e., the block *User_FM_body*, which can in principle be used for anything the user wants. For the current example, it outputs zero values. All forces and moments are added together before they are outputted.
The simulation of a free-fall entry (translations and rotations), i..e, without guidance and control, gives the trajectory and attitude motion shown in Fig. 15. This type of trajectory is typical for entry capsules, see also Vinh (1981) and Mooij (1998). Comparing results shows a correct implementation of this type of problem involving large aerodynamic forces and moments. On a sidenote: in Mooij and Ellenbroek (2007) the implementation of a controlled, winged re-entry vehicle is discussed, with an extensive aerodynamic database including several control surfaces. That particular model has been used for many guidance and control studies, which can be found in the quoted references.

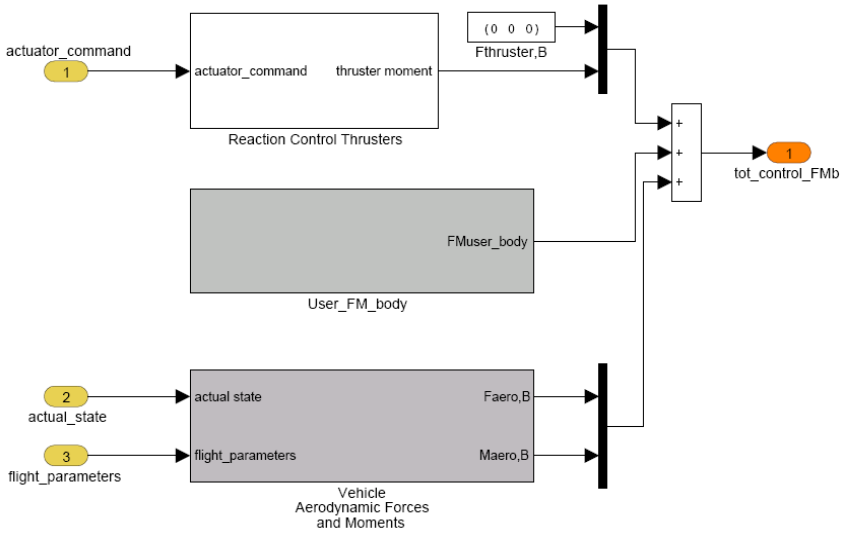

Fig. 14. Inclusion of aerodynamic forces and moments computation in the *Actuators* block.

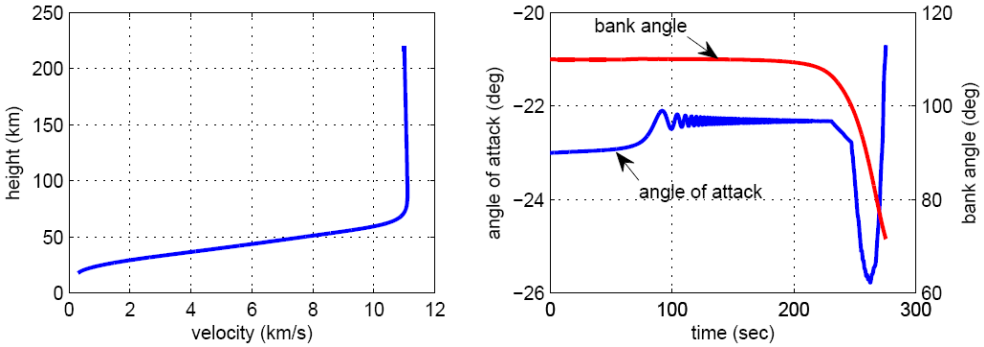

Fig. 15. Alitude-velocity profile (left) and attitude angles versus time (right).

The second example deals with the entry and parachute descent of a planetary lander in the atmosphere of Mars. In principle this example is similar to the previous one, in terms of aerodynamic implementation. However, also two parachute models are implemented in the *Actuators* block. The first parachute is a so-called drogue parachute that is typically deployed at supersonic speeds (Mach number of 2.1) and serves to stabilize the system and remove part of the velocity. The second parachute is the main, and should bring the final velocity down to a certain required value. Both parachutes are modelled as a drag area that can gradually inflate once triggered.

New in this example is the choice of inertial reference frame (and thus main attracting body): the frame has its origin in the centre of Mars. This means that the planetary characteristics have been adapted as well, notably the equatorial radius, the rotational rate, the gravity model and the atmosphere. This atmosphere is the state-of-the-art ESA Mars Climate Database (Forget et al., 2005), for which a dedicated interface has been written to communicate with the database's Fortran interface.

The deployment of the two parachutes is triggered by a timer, starting at Mach = 2.1. Two scenarios are considered, both aiming at a certain final velocity, i.e., V_f = 50 m/s and V_f = 80 m/s. For both scenarios, the drogue is inflated after 0.5 s. For the first scenario it is released after 17 s (14 s for the second scenario). The main parachute is inflated at 18 s (15 s). It is noted that the main chute for the first scenario is considerably larger than for the second one to guarantee the lower final velocity.

For each scenario two simulations are run, i.e., a nominal one and one for which drogue inflation is delayed by 5 s. The results are shown in Fig. 16. Due to the inflation delay the Mars lander impacts with V_f = 65 m/s and V_f = 100 m/s, respectively. This kind of simulations will help the system designers to study sensitivities in the descent and landing system.

This example has clearly shown the versatility of the simulation environment to accommodate complex force models, such as a parachute system. Of course, the complexity of this force model can easily be increased as long as the interface to the flight-dynamics kernel remains the same. To conclude, an extension of this example could be to use the *Multi-Body Library* to model the parachute as a separate body, such that also the relative rotation of parachute and payload can be analysed.

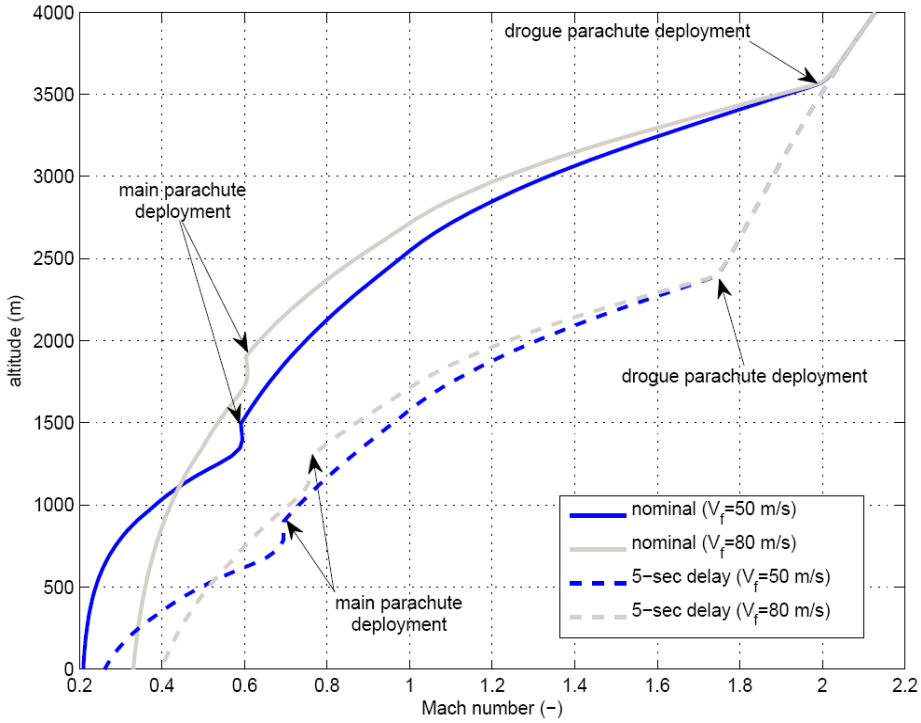

Fig. 16. Alitude-Mach number profile for different parachute-deployment timings.

5. Conclusions

In this paper, the development of a Generic GNC Simulation Environment, starting from a set of User Requirements, has been described. The GGNCS Environment consists of a set of MATLAB®/Simulink® libraries that are available to build a simulator of a spacecraft in its environment. Each library comprises of a number of relatively simple blocks. The blocks simulate/calculate/evaluate only one functional property, and are separated into an application part and an interface part. The interface part takes care of all data communication with the simulation platform, which is currently MATLAB®/Simulink®. The user should use MATLAB®/Simulink® only to design the architecture of the spacecraft simulator, which leads to an architecture that is very modular and reflects the physics. The same architecture will serve as baseline for the development of other simulation facilities that support the complete lifecycle of, for instance, the on-board software. This architecture could be transferred by simply parsing the Simulink® file. Extensive evaluation of the simulation models has indicated that the models are representative for mission and control-algorithm analysis for a multitude of missions and spacecraft configurations.

6. References

Cornelisse, J.W., Schöyer, H.F.R. and Wakker, K.F., Rocket propulsion and spaceflight dynamics, Pitman, London, 1979.

Ellenbroek, M.H.M., "On the fast simulation of the multibody dynamics of flexible space structures". Ph.D. dissertation Enschede, Technische Universiteit Twente, 1994

European Cooperation for Space Standardization, "Software", ECSS-E-ST-40 C, Third issue, 06 March 2009

Forget, F., Dassa, K., Wanherdrick, Y., Lewis, S.R., Collins, M. and Bingham, S.J., "Mars Climate Database v4.0 User Manual", ESTEC contract 11369/95/NL, January 2005.

Franco, R., Dumontel M.L., Portigliotti, S., and Venugopal, R., "The Dynamics and Control Analysis Package (DCAP) - A versatile tool for satellite control", ESA Bulletin 87, 1996.

Geradin, M. and Cardona, A., "Flexible multibody dynamics, A finite element approach", Chichester, John Wiley & Sons Ltd, 2001.

Haug, E.J., *Computer aided kinematics and dynamics of mechanical systems, Volume 1: Basic methods*, Allyn and Bacon, Needham Heights, Massachusetts, 1989.

Montenbruck, O. and and Gill, E., *Satellite orbits. Models, methods, and applications*, Springer Verlag, 2000.

Mooij, E., "Aerospace-Plane Flight Dynamics. Analysis of Guidance and Control Concepts", Ph.D. dissertation, Delft University of Technology, 1998. Available from http://repository.tudelft.nl/

Mooij, E. and Wijnands, Q.G.J., "Generic Attitude and Orbit Control Simulator development supporting the AOCS software life cycle", From: *7th International Workshop on Simulation for European Space Programmes*, November 12-14, 2002, Noordwijk, The Netherlands.

Mooij, E. and Wijnands, Q.G.J., " Real-Time Implementation of a Model Reference Adaptive Control System", AIAA-03-5754, AIAA Modeling and Simulation and Technologies Conference, Austin, TX , August 11-14, 2003.

Mooij, E. and Ellenbroek, M.H.M., "Multi-Functional Guidance, Navigation, and Control Simulation Environment", AIAA-07-6887, AIAA Guidance, Navigation, and Control Conference, Hilton Head, SC, August 20-23, 2007.

Neefs, M.J. and Haye, M.J., "The Herschel-Planck ACMS simulation approach", From: 7th International Workshop on Simulation for European Space Programmes, November 12-14, 2002, Noordwijk, The Netherlands.

Schaub, H.P. and Junkins, J., Analytical Mechanics of Space Systems, Second edition, AIAA Education Series, 2009.

Vinh, N.X., *Optimal trajectories in atmospheric flight*, Elsevier, 1981.

Wertz, J., *Spacecraft Attitude Determination and Control,* (Astrophysics and Space Science Library : Vol 73), 1978.

Wie, B., *Space vehicle dynamics and control*, Second Edition, AIAA Education Series, 2008.