A SIMULATION TOOL FOR IN-ORBIT ASSEMBLY OF LARGE STRUCTURES

Virtual Conference 19–23 October 2020

V. Bissonnette[1], C. Bazerque[1], S. Trinh[1], C. Porte[1], G.Arcin[1], M. Rognant[2], J.-M. Biannic[2], C. Cumer[2], T. Loquen[2], X. Pucel[2]

[1]Magellium, 31520 Ramonville Saint-Agne, France, E-mail: vincent.bissonnette@magellium.fr
[2]ONERA-The French Aerospace Lab, 31055 Toulouse, France, E-mail: mathieu.rognant@onera.fr

ABSTRACT

The PULSAR (Prototype for an Ultra Large Structure Assembly Robot) project, aims at developing and demonstrating core technologies enabling the in-orbit assembly of the 8m-diameter primary mirror of a space telescope with an autonomous robotic system. This paper presents the demonstrator of In-Space Assembly in Simulation, which is designed as an integrated simulation tool for the prototyping and development of these autonomous assembly technologies.

1 INTRODUCTION

Autonomous assembly of large structures in space is a key enabling technology for future missions which will require structures of increasing size, exceeding the capacity of modern launch vehicles when deployed as a single piece. One such type of missions is the deployment of space-based telescopes. The James Webb Space Telescope (JWST)[1], planned for launch in 2021, is a good example of how constraints of the launch vehicle affect the complexity of the spacecraft design, with the need for a large number of release mechanisms to deploy its 6.5m segmented primary mirror. Future projects, such as the Large UV Optical Infrared Surveyor (LUVOIR) [2], target even larger primary mirrors, further increasing the required launch vehicle volume.

It becomes clear that a paradigm shift is required to enable this continued increase in spacecraft size and complexity. The introduction of in-orbit assembly has the potential to fulfill this need. To this end, the European Commission, through its Space Robotic Technologies Research Cluster (SRC), has funded the PULSAR (Prototype for an Ultra Large Structure Assembly Robot) project [3]. It aims at developing and demonstrating core technologies enabling the in-orbit assembly of the 8m-diameter primary mirror of a space telescope with an autonomous robotic system.

In the scope of PULSAR, a demonstrator of In-Space Assembly in Simulation (dISAS) is developed. The demonstrator addresses the autonomous assembly of a primary mirror composed of 36 individual Segmented Mirror Tiles (SMT), using an on-board robotic manipulator. SMTs are equipped with Standard Interconnects (SI), which provide mechanical load, power, data and thermal transfers between components. SIs can be actively commanded to connect or disconnect, enabling the assembly of modular components. Throughout the assembly process, spacecraft pointing requirements must be satisfied to maintain communication with the ground station.

For this purpose, dISAS is designed as an integrated simulation tool in which a user can model an operational scenario, create an assembly sequence, and assess its feasibility as a system. The manipulator workspace can be analyzed and the perturbations it creates on the spacecraft platform evaluated to help in designing a suitable attitude controller. Moreover, realistic sensors and actuators can be implemented and directly interfaced with existing software to enable simulated hardware-in-the-loop testing.

In this paper we first introduce the use case considered within the framework of PULSAR. Then the architecture and implementation choices of dISAS are presented. In section 4, the main hardware components simulated by dISAS are detailed, and section 5 deals with the prototyping of the system's perception and control features. Section 6 presents the autonomy components of the system. Finally, preliminary results and an outlook on future work is provided.

2 OPERATIONAL SCENARIO

In the baseline dISAS scenario, the spacecraft is in its final orbit at L2, and primary structures (sunshield, solar panels) are already deployed. All 36 SMT are stacked in a tile container, and secured using their Standard Interfaces

The goal of the demonstration is to assemble, using a robotic manipulator mounted on a linear rail, the complete primary mirror structure on its final assembly site. All SMTs are successively grasped by the manipulator and moved to their final location according to the assembly plan. The SI are also autonomously commanded to dynamically create the necessary mechani-

cal and data links. In order to reduce the required manipulator workspace while still allowing construction of very large structures, the concept of *extended mobility* is introduced. To this end, an intermediate Preassembly Site is used to build subassemblies of 5 SMTs, with a sequence of successive rotations and mating of new SMTs. Subassemblies are then moved to their final position in a single block, as shown on Figure 1. During the full scenario, the AOCS subsystem is active, managing the disturbances created by the movements of the manipulator.

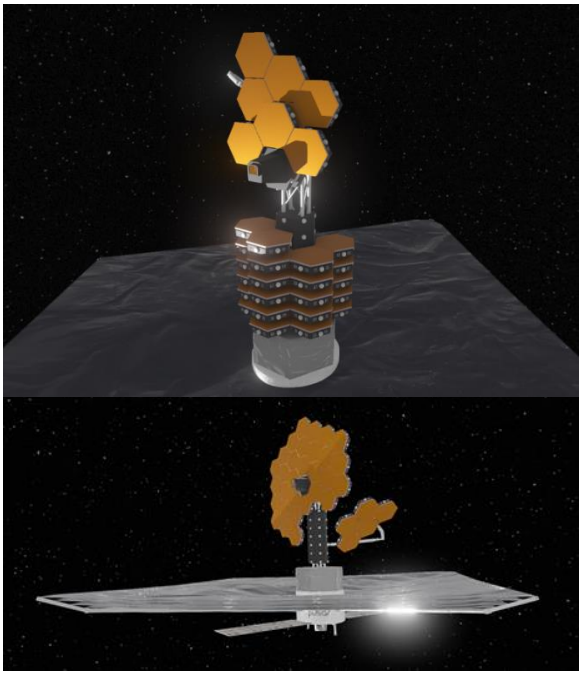

Figure 1: Snapshots of dISAS. Ongoing first preassembly (top). Robotic arm is moving final subassembly to the primary mirror (bottom).

3 SOFTWARE ARCHITECTURE

At software level, dISAS is organized in three main layers, illustrated in Figure 2. The Simulation layer creates the 3D virtual environment in microgravity, including physics and a realistic visual rendering. This layer also creates the geometry and behavior models of the simulated hardware components, such as the spacecraft platform, SMTs, manipulator, AOCS actuators, etc.

The Functional Layer is where all high-level functionalities of the autonomous assembly system reside. In the current dISAS implementation, it contains various components for the robotic arm path planning and control, perception functions to perform visual monitoring and arm servoing based on camera acquisitions, and the platform AOCS controller.

Finally, the Autonomy layer contains the software components responsible for the specification and execution of the assembly sequence. In dISAS, Behavior Trees are used to compose complex sequences from atomic actions based on the capabilities of the robotic system.

The layers of the system are integrated in TASTE/ESROCOS [4] [5], a novel software framework developed by a related EU-funded project, and targeted towards space robotics.

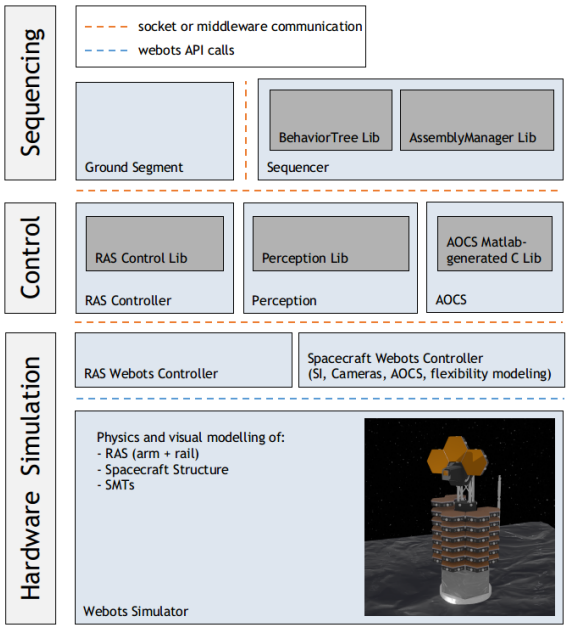

Figure 2: dISAS architecture overview

4 HARDWARE SIMULATION LAYER

4.1 Simulation Engine

The open-source Webots [6] simulation environment is used as the core simulation engine in dISAS.

Webots is a generic robot simulator actively developed since 1998 by Cyberbotics Ltd. It is used in industry, education, research and several EU-financed research projects. Since December 2018, it is released under the free and open-source Apache 2 license. It includes a large collection of robots, sensors and actuators frequently used in robotics. It uses a custom physics engine based on Open Dynamics Engine (ODE), and a custom realistic rendering engine based on OpenGL (WREN). These characteristics made Webots the ideal candidate for the level of fidelity sought for dISAS.

The geometry, visual representation, and physical characteristics of the spacecraft components are specified in configuration files using the Virtual Reality Modeling Language (VRML). The modelled behavior

of components such as sensors and actuators is created by the user in separate executables called *controllers*. These controllers can be implemented in a variety of supported languages such as C, C++ or Python, and are also used to expose interfaces to other software elements and libraries, through API calls, or middlewares such as ROS and ESROCOS.

This is how dISAS models the spacecraft structural elements, such as the bus, payload module, sunshield beams, and solar panels. The currently-used spacecraft properties are based on data from the JWST mission, extrapolated to fit the PULSAR mission scenario.

4.2 Robotic Assembly System

The design process of the dISAS Robotic Assembly System (RAS) is challenging for numerous reasons, especially because of the need for high dexterity. RAS dexterity is a key element as the arm has to achieve complex operations such as:

- reaching the tiles in the container located at the very limit of the robot workspace,
- manipulating multiples tiles,
- building the preassemblies.

The current dISAS robotic arm follows a modular design, based on inputs from another PULSAR demonstrator. Its specific L-shaped articulation at the center of the arm allows minimizing the arm when stowed, while maintaining the necessary reachability capability of the arm when fully extended, as shown in Figure 3.

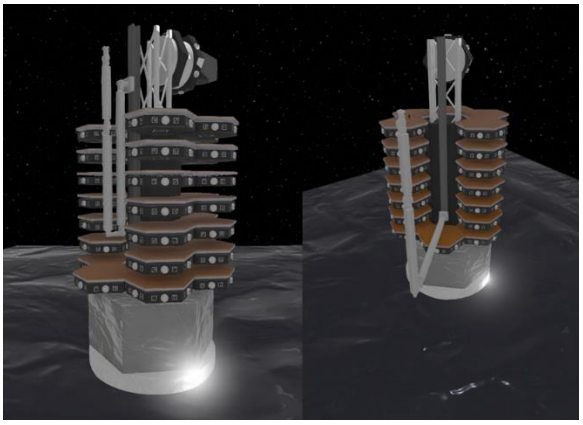

Figure 3: Visualization of RAS and SMT container is dISAS

The kinematic model of the Robotic Assembly System includes a rail and a robotic arm. The importance of the rail is crucial in the reachability ability of the Robotic Assembly System, enabling it to operate the tiles at the bottom layer of the SMT container, and to perform the necessary movements for reaching the large primary mirror in the upper part of the spacecraft.

Having the arm and the rail in the same kinematic model also allows the motion planning block and the control block of the RAS to simultaneously command both entities to perform the desired task.

The rail is modelled by a prismatic joint, while the robotic arm is modelled by 7 revolute joints. dISAS allowed to observe that a supplementary degree of freedom improves the manipulability of the arm, typically to reach the tiles stored at the front of the spacecraft. The redundancy in the kinematic model of the Robotic Assembly System improves the dexterity of the RAS, but at the expense of the complexity and the computation time required to find a motion planning solution.

Low-level simulated hardware control of the manipulator is achieved through a dedicated Webots controller. It exposes a motion command interface from the RAS control block in the Functional Layer using joint trajectory data structures. Robot states are published back in real time as joint states. When a trajectory is received, a trajectory interpolation method is used in order to send an admissible robot command at each simulated time step. The final robot command is executed using Webots API calls.

During the remaining months of the PULSAR project, it is foreseen to perform an optimization of the manipulator to reduce its size, and bring its design more in line with robotic arms designed for space operations, such as the Compliant Assistance and Exploration SpAce Robot (CAESAR) [7].

4.3 Segmented Mirror Tiles and Standard Interconnects

SMTs are made of a hexagonal base structure with 13 fiducial markers (AprilTag [8]) and 6 Standard Interfaces. Each tile has a unique set of tags for visual detection (see Section 5.2). Its standard interfaces can be individually locked/unlocked through requests to the spacecraft controller by the Function Layer.

The SI lock mechanism is modelled with a Webots connector node, designed to simulate various types of docking hardware (e.g. mechanical links held in place by a latch, magnetic links, pneumatic systems, etc.). The connection supports several kinds of constraints such max tensile strength, max shear strength, distance and rotation tolerance.

The visual and bounding object of the Standard Interface is based on the HOTDOCK connector, an interface developed by a partner within the frame of PULSAR.

Each tile exposes a control interface through an emitter/receiver pattern, modelling a dynamic communication bus linking all SMTs and the spacecraft.

4.5 Flexible Appendages Modelling

Large structures, such as the solar panels, the sunshield and the primary mirror, are more subject to mechanical vibrations due to the low structural damping in the materials and the absence of other forms of damping in the vacuum of space.

These vibrations affect pointing performance and must be taken into account when designing an attitude controller [9]. Thus, the validation of the satellite stability by dISAS requires to simulate the flexible dynamics of these appendages in the bandwidth we wish to control.

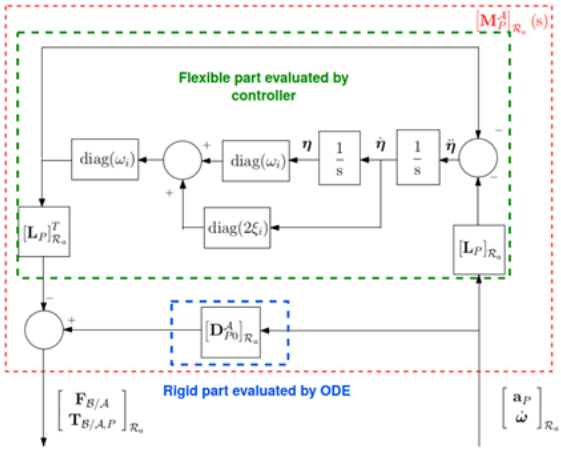

Figure 4: Block diagram of the flexible perturbation computation

Webots relies on an extended version of the ODE physics engine which was improved by the Webots maintainers. In particular, Webots includes functions allowing the application of external torques and forces in the numerical integration scheme to a specific body of the simulation. This functionality is used to apply the reactions wrench of the flexible appendages to the spacecraft. As the rigid parts of these wrenches are already evaluated by the general integration scheme, only the flexible parts of the considered appendices reaction wrenches are applied.

To evaluate these flexible wrenches, a modal description of flexibility are used [10]. For this purpose, specific controllers are associated to each flexible appendage. These controllers use the accelerometer functions already provided by Webots to measure the current acceleration tensor of the fixation point of the flexible appendage. These controllers instantiate FlexAppendage objects which integrate modal states and evaluate reaction wrench at each step (Figure 4).

A simple-use case of a spacecraft with two flexible solar panels is illustrated in appendix (Figure 8).

5 FUNCTIONAL LAYER

The Functional Layer (FL) includes all the algorithms for control, perception and planning as core components that can be easily reused and customized. Each component is implemented as a standalone C++ software library, which is then integrated in the FL as TASTE functions. The TASTE framework takes care of generating middleware data exchange code from a user-created interface mode, and data exchanges use standardized ASN.1 data types. The framework also supports generation of bridges to other middleware, such as ROS, for easy prototyping with existing software.

5.1 Spacecraft Attitude Controller

During the assembly process, the main issue consists in preserving a reasonable pointing accuracy of the satellite to maintain the link with the ground station, despite the slow evolution of the inertia and frequency modes of the system and the torque disturbances that are generated by the motion of the robotic arm.

To address this point, a modular attitude controller structure described in Figure 5 is used.

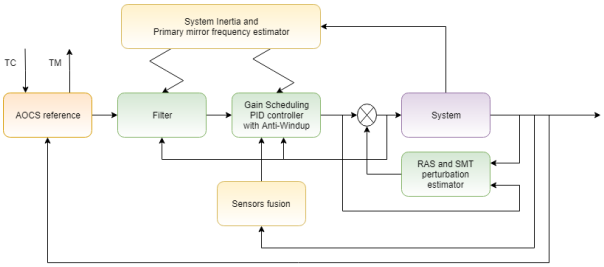

Figure 5: Attitude controller structure

To size this controller, a set of Linear Time Invariant models which cover all the deployment steps are generated by dISAS. Then, a multi-model approach is used to analyze the inertia variations and choose the best controller setting via the Robust Control System toolbox of Matlab. In a second time, the robustness of the controller to the flexibility and RAS perturbations are assessed via the SMAC toolbox [11], and the controller is discretized. Finally a C++ library of this controller is generated by using autocoding tools [12].

5.2 Perception Functions

Perception functions, process images captured on the demonstrator cameras in order to increase the autonomy level and safety of the assembly process.

Two cameras are mounted on the simulated spacecraft: one on the end-effector (EE) and one on the spacecraft structure. Using the captured images, three primary perception functions are deployed: Tile Localization, Assembly Monitoring, and Visual Servoing.

Using the AprilTag fiducial markers mounted on the sides and back of the SMT, Tile Localization will detect the tiles, based on their unique IDs, in images captured by the External or EE-mounted cameras. Tag corners coordinates are then extracted, and the pose with respect to the camera frame is computed using Perspective-n-Point methods. The pose of the tile is finally computed using the 2D/3D correspondences, between 2D coordinates extracted from AprilTag detection and 3D coordinates of the tile model.

This localization is used to enable a safe, step-by-step approach to grasp the tiles with the manipulator end-effector, instead of relying only on open-loop localization based on manipulator forward kinematics.

Further emphasizing safety, the Assembly Monitoring function enables automatic detection of invalid conditions during the assembly. By comparing the currently localized tiles in the observed scene with an internally held ground truth model of each assembly step, it raises an error whenever an out-of-bounds tile of sub-assembly pose is detected. The error can then be handled by the autonomy component to perform a user-defined recovery or wait for operator intervention

Finally, Visual Servoing [13] is used to control the motion of the manipulator using vision feedback. It is suitable for precise robot movement, in our case to perform SMT picking and to precisely position SMTs for the primary mirror assembly. To this end, the goal of the control law is to regulate at zero the considered task function. For position-based visual servoing, this is commonly the current end-effector pose with respect to a desired end-effector pose. For image-based visual servoing, this can be the current positions of image features with respect to their desired image coordinates.

Two configurations are considered: eye-in-hand, where the camera is embedded at the robot end-effector, and eye-to-hand, where the camera is static with respect to the robot base and looking at the robot end-effector. PULSAR exploits both configurations in its scenarios. The first one is the procedure to perform mating between the end-effector SI and any SMT SI, for tile grasping. In this configuration, an eye-in-hand configuration is suitable, since a camera at the end-effector can localize the tags near the SMT SI, as shown in Figure 6. In contrary, when the tile is attached to the robot end-effector, the EE camera viewpoint is almost completely occluded. Switching to an eye-to-hand configuration is then preferable. The external camera is positioned in order to simultaneously localize the SMT attached to the robot end-effector and the SMT targeted for mating, to achieve a desired relative pose between them.

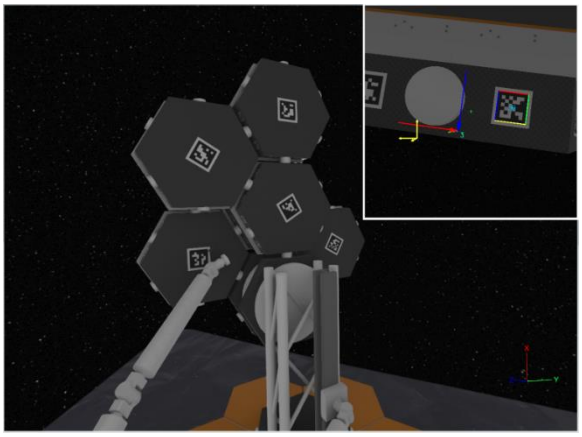

Figure 6: Eye-in-hand visual servoing in dISAS. EE camera point of view with AprilTag detection and desired visual features pose in top-right.

5.3 Robotic Assembly System Control

This component is responsible for commanding the Robotic Assembly System. It processes user command requests (Cartesian pose, joint position, etc.) in order to plan a collision-free trajectory in the task space. Then, the necessary joint trajectory commands are computed and sent to the Webots controller of the RAS.

The current implementation is based upon the open-source ROS and MoveIt ecosystem [14]. The MoveIt framework is exploited for motion planning and collision checking capabilities. MoveIt is currently used for testing and development until a custom motion planning library developed in the frame of PULSAR is available. Standardized interfaces will allow users to change the underlying motion planning library with their own, with minimal changes.

As inputs, the RAS kinematic model is described using the widely-used URDF specification. Joint types, joint limits, dynamic properties are detailed in this file. Semantic description of the robot (e.g. which link is the

robot end-effector) uses the SRDF specification. 3D mesh data is also provided for collision checking during the path planning process, and is extracted from the Webots scene model beforehand.

Communication between the RAS control block and the simulated hardware layer is achieved with ROS actions and topics, using a TASTE/ROS bridge. The RAS control component exposes the following interfaces:

- end-effector pose goal request, that is the desired pose of the RAS end-effector with respect to the robot base frame,
- service to update the state of the RAS, for instance to attach/detach a tile to/from the robot end-effector. This is necessary in order to correctly plan a trajectory when one or multiple SMT are attached to the RAS end-effector,
- service to update the planning scene, for instance to initialize the position of the various collision objects in the scene at start-up,
- online robot state input from hardware simulation layer to perform closed-loop control.

6 AUTONOMY LAYER

6.1 Assembly Sequencing

The type of autonomy required for the PULSAR scenarios can be considered as sequential rather than reactive. The system follows a completely predefined assembly plan. Therefore, a sequencing functionality is used to schedule the assembly steps, keep track of the various states of the system, ensure safety of operations by validating input/output conditions, and implement failure recovery.

We use behavior tree formalism to describe the different steps of the mission scenario. Behavior trees are very similar to hierarchical state machines with the key difference that the main building block is a task (e.g. “do a pre-assembly”) rather than a state. Formally speaking, a behavior tree is a single root directed tree where internal nodes are called “control flow nodes” and leaf nodes are called “task execution nodes”.

In dISAS, behavior trees are implemented using the open-source BehaviorTree.CPP library [15]. The sequencer library embeds a list of atomic actions which are bound with functions exposed by the functional layer components, and can be reused for the various steps of the assembly. This allows the creation of high-abstraction autonomous tasks. Trees are described as an XML file which can include other trees, enabling complex task composition. Figure 7 illustrates an example of such dISAS behavior tree structure.

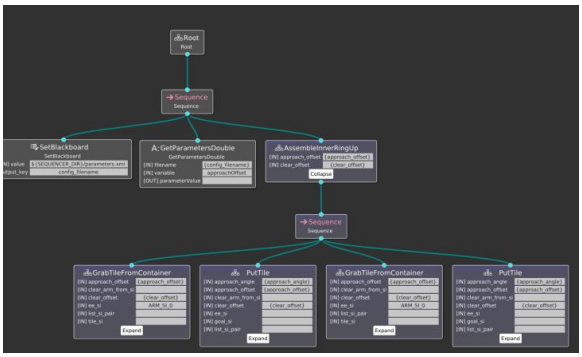

Figure 7: Example behavior tree structure of assembly sequence

4 CONCLUSION

This paper provided an overview of the current implementation of the demonstrator of In-Space Assembly in Simulation of the PULSAR project.

dISAS has already proven useful in providing a simulation environment to support prototyping and developments in PULSAR. Next steps include finalizing the integration and executing a complete, fully-integrated demonstration mission.

Appendix

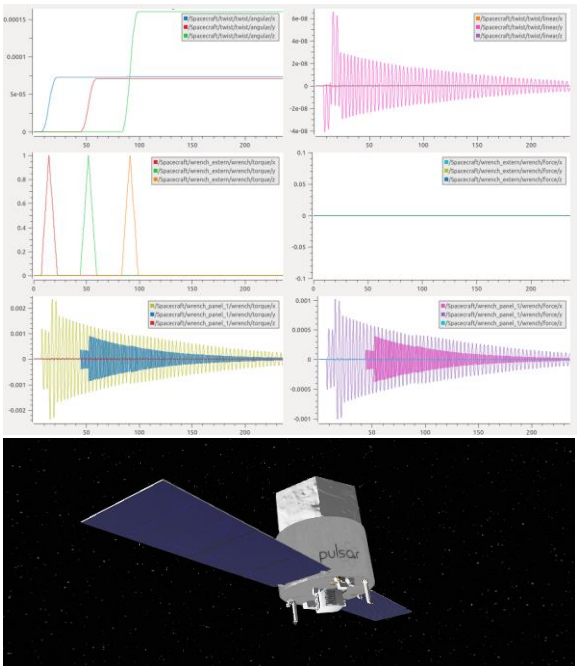

Figure 8: Simple-use case of a spacecraft with two flexible solar panels

We present, on the top left, the angular velocity and, on the top right, the linear velocity of the main body when external torques are applied around the different axes. These external torques are represented in the

middle left plot. On the bottom plots, we can also observe the reaction torques and forces of the solar panel.

Acknowledgements

The PULSAR project is funded under the European Commission's Horizon 2020 Space Strategic Research Cluster Operational Grants, grant number 821858.

The authors would like to thank M. Aurélien Cuffolo, from Thales Alenia Space in France, partner in the PULSAR consortium, for providing inputs on mission analysis, spacecraft sizing and AOCS models.

References

[1] Gardner, J. P. et al (2006) The James Webb Space Telescope. *Space Science Review* 123.4:485-606.

[2] LUVOIR Team. (2019) The LUVOIR Mission Concept Study Final Report. *arXiv*:1912.06219

[3] Prototype for an Ultra-Large Structure Assembly Robot. http://www.h2020-pulsar.eu.

[4] M.Perrotin et al.(2012).TASTE : An Open-Source Tool-Chain for Embedded System and Software Development. *Embedded Real Time Software and Systems (ERTS2012).*

[5] M. Muñoz Arancón, et al.(2017). ESROCOS: a robotic operating system for space and terrestrial applications. *ASTRA 2017* 1-8.

[6] Webots. http://www.cyberbotics.com. Commercial Mobile Robot Simulation Software.

[7] A. Beyer et al. (2018) CAESAR : Space Robotics Technology for Assembly, Maintenance and Repair. *International Astronautical Congress 2018*.

[8] M. Krogius, A. Haggenmiller, and E. Olson. (2019). "Flexible layouts for fiducial tags". *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[9] Blackmore, L., et al. "S. Kang. Instrument pointing capabilities: past, present, and future." *AAS Guidance and Control Conference*. Jet Propulsion Laboratory, National Aeronautics and Space Administration. 2011.

[10] Tantawi, Khalid HM, Daniel Alazard, and Christelle Cumer. "Linear dynamic modeling of spacecraft with various flexible appendages." *IFAC Proceedings Volumes 41.2 (2008):* 11148-11153.

[11] Roos, Clément. "Systems modeling, analysis and control (SMAC) toolbox: An insight into the robustness analysis library." *2013 IEEE Conference on Computer Aided Control System Design (CACSD). IEEE*, 2013.

[12] Bourbouh, Hamza, et al. "CoCoSim, a code generation framework for control/command applications An overview of CoCoSim for multi-periodic discrete Simulink models." *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*. 2020.

[13] F. Chaumette and S. Hutchinson, (2006). "Visual servo control, Part I: Basic approaches," *IEEE Robotics and Automation Magazine.* Vol. 13, no. 4 pp. 82–90.

[14] Ioan A. Sucan and Sachin Chitta, "MoveIt", Available at http://moveit.ros.org/ .

[15] D. Faconti. BehaviorTree.CPP. Available at: https://github.com/BehaviorTree/BehaviorTree.CPP/