73rd International Astronautical Congress (IAC), Paris, France, 18-22 September 2022.
Copyright 2022 by the Authors. Published by the IAF, with permission and released to the IAF to publish in all forms

IAC-22,D1,4A,10,x69056

A ROS/Gazebo-based framework for simulation and control of on-orbit robotic systems

J. L. Ramón[a]*, J. Pomares[a], L. Felicetti[b]

[a] University of Alicante, San Vicente del Raspeig, Alicante, 03690, Spain, jl.ramon@ua.es, jpomares@ua.es
[b] *Cranfield University, Cranfield, MK43 0AL, United Kingdom,* Leonard.Felicetti@cranfield.ac.uk
* Corresponding Author

Abstract

The use of simulation tools such as ROS/Gazebo is currently common practice for testing and developing control algorithms for typical ground-based robotic systems but still is not commonly accepted within the space community. Numerous studies in this field use ad-hoc built, but not standardized, not open-source, and, sometimes, not verified tools that complicate, rather than promote, the development and realization of versatile robotic systems and algorithms for space robotics. This paper proposes an open-source solution for space robotics simulations called OnOrbitROS. This paper presents a description of the architecture, the different software modules, and the simulation possibilities of OnOrbitROS. It shows the key features of the developed tool, with a particular focus on the customization of the simulations and eventual possibilities of further expansion of the tool. In order to show these capabilities, a computed torque-based controller for the guidance of a free-floating manipulator is proposed and simulated using the ROS/Gazebo-based framework described in the paper.

Keywords: space robot, humanoid robot, robot control, ROS.

Acronyms/Abbreviations

Robotic Operating System (ROS), On-Orbit Servicing (OOS), Open Dynamics Engine (ODE), Dynamic Animation and Robotics Toolkit (DART).

1. Introduction

Space missions will require extensive use of robots for active debris removal, on-orbit servicing, assembling and manufacturing applications. The development and testing of such systems often require an iterative approach that, together with the technical difficulties of reproducing space conditions in ground-based test facilities and the high costs associated with these tests, discourage the utilization of hardware-based approaches in the earlier stages of the design. Using simulation tools such as ROS/Gazebo is currently common practice for testing and developing control algorithms for typical ground-based robotic systems but still is not widely accepted within the space community. One of the reasons for this is that this kind of development environment does not allow for a complete and realistic simulation of the space conditions, such as micro-gravity and frictionless conditions. However, ROS is being included in several space robotics systems in order to have versatile robotic systems and algorithms developed for space robotics.

The use of ROS in space began with Robonaut 2, developed by NASA and General Motors. It was the first humanoid robot on the ISS in 2011 [1]. NASA also uses ROS in other space robots, such as Astrobee [2][3][4]. None of these robots was tested in an extravehicular activity scenario, but future robotic missions, such as NASA's Valkyrie, will have improved autonomy and be tested outside the ISS [5]. It is worth mention other developments based on Gazebo that allows the simulation of space and planets conditions such as the moon terrain [6]. Additionally, several different tools and libraries are now available as open source to solve different problems in space. For example, in [7] a model-free hierarchical decoupling optimization algorithm to realize 6D-pose multi-target trajectory planning for the free-floating space robot is presented.

Keeping in mind this panorama, the solution proposed in this paper moves towards developing a unified open-source tool for space-robotic simulations [8]. The adopted framework is based on ROS, an open-source meta operative system to develop robot applications, combined with Gazebo, a tool to simulate populations of robots in customized environments. Such tools were modified to include and reproduce the principal environmental conditions that eventual space robots and manipulators could experience in an OOS scenario. This solution allows for the simulation of complex space robotic systems and at the same time takes advantage of the number of packages already developed in ROS for control, vision, teleoperation and modelling tools. In this way, it is possible to simulate robotic operation for OOS, without the need of ad-hoc coded simulation tools but relying on a well-validated tool, following the ROS principle of "Don't reinvent the wheel". The paper presents a description of the architecture of the different software modules. It shows the key features of the developed tool, with a particular focus on the customization of the simulations and eventual possibilities of further expansion of the tool.

The remaining part of the paper is divided into the following sections. Section 2 describes the main characteristics of ROS and Gazebo and their properties to simulate space robotics. Section 3 shows the main modules of OnOrbitROS. Specifically, it describes the architecture and the main options to simulate robots in different orbital conditions. A computed-based torque controller for the guidance of free-floating manipulators is presented in Section 4. This controller will be simulated using OnOrbitROS and the main results are shown in Section 5. Concluding remarks and future areas of development are presented in Section 5.

2. ROS and Gazebo properties for space simulation

This section presents the main properties of ROS and Gazebo for their use in space robotics. The main limitations of these tools when are applied to space robotics are also described.

2.1 ROS

ROS was initially developed in 2007 by the Stanford Artificial Intelligence Laboratory as part of the project Stanford Artificial Intelligence Robot, a mobile robot with a manipulator. From 2008, Willow Garage continues the development, using ROS for their robot PR2, a robot with two manipulators. Nearly 55 percent of the world's robots will include a ROS package by 2024. Currently, ROS is a standard in robot programming [9]. ROS is an open software project with a wide community of developers around the world, universities, research centres, companies, etc.

In order to make a robotic platform functional it is necessary to develop a large amount of software. The typology of software to be developed spans from hardware-specific, such as the control of the servomotor drivers, through software related to the perception of the workspace by the sensors, to the high-level algorithms of navigation, computer vision, trajectory control, etc. Keeping in mind these needs, the creators of ROS tried to solve two main problems.The first is to create code with the right structure and modularity to avoid having to develop new code when hardware or robot morphology changes. That is, being able to reuse already developed modules in future projects. The second of the problems is related to the possibility of sharing code modules developed by third-parties, with the added advantage of having code tested by large communities of users. Thus, much more user experience information is returned, and a better debugging and evolution of the developed code is allowed.

Other ROS objectives can be synthetized in the following points [10]:

- Non-centralized control is distributed in different modules located on several computers.
- Use of different tools for different aspects during the process of robot programming: design, debugging, monitoring, control, simulation, etc.
- Multilanguage development.
- The choice of the programming language should be based on the language that is more adequate to the requirements of the module that is being developed (e.g. the use of c/c++ or Python).
- Code reusability. This refers to the fact that, habitually, in the software development of a robot, there are certain parts that are common and that could be reused in other projects
- Open-source project. This helps the diffusion of the tool to the research and educational community. So that such communities might have the opportunity of extendiding functionalities of the tools, contributing new modules, and adapting other open-source projects to ROS.

Such key features ideally make ROS more than a suitable option also for space robotics applications. However, several issues limit the utilization of ROS for the control and simulation of robots in orbit. Specifically, in such applications, it is required to simulate orbital conditions, gravity gradient perturbations, free-floating dynamics conditions, etc., that are not part of the standard packages included in ROS, while other typical earth-specific effects, such as wind, gravity, atmospheric friction and drag etc., are not necessary. This paper presents a framework called OnOrbitROS that overcomes such limitations and allows for the simulation and control of space robots in orbital conditions in a ROS-based environment. Fig. 1. presents the main architecture of OnOrbitROS. The different modules will be described in Section 3.

2.2 ROS Control

One of the packages that is used as the basis for the implementation OnOrbitROS and the controllers presented in this paper is ROS Control [11]. ROS Control is a package that complies with the main ROS directives due to its possibilities of reuse and expansion while being independent of the robot properties.

The ROS Control design takes into account that real-time is necessary to control the robot hardware and that it needs to be independent from the typology of the robot used. Therefore, the ROS Control packages can be used for mobile robots, manipulators, or humanoid robots. At the same time, the platform offers numerous tools that make it easy to create customized controllers. OnOrbitROS integrates the ROS Control packages, allowing for the definition of space robot controllers without the modification of ROS Control.

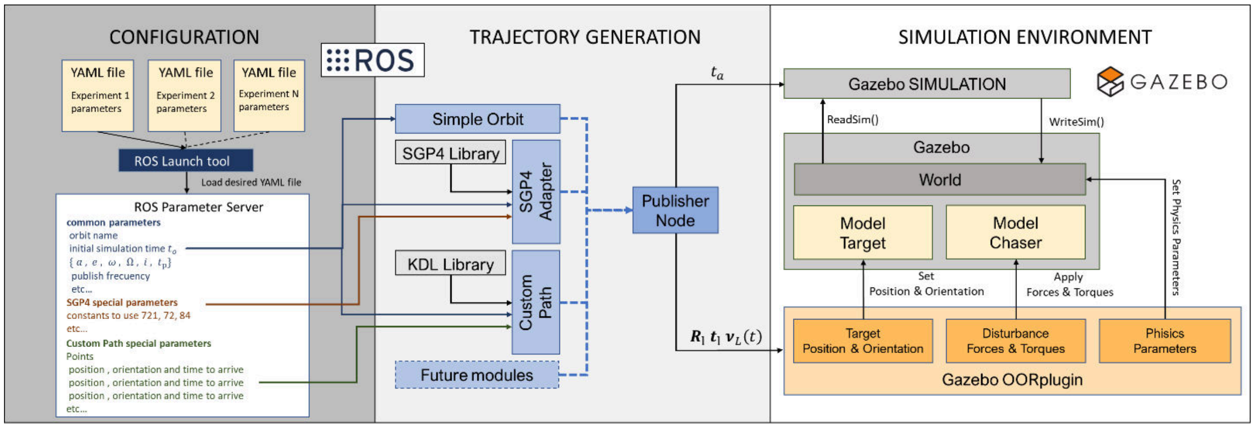

Fig. 1. OnOrbitROS architecture

Additional controllers have been designed to implement the controllers presented in this paper, taking into account the specific dynamics of the space robot manipulators while maintaining the architecture of ROS Control.

2.3 Gazebo

Gazebo is an open-source simulator that allows for realistic simulations of robots, environments, and objects. Gazebo provides the necessary tools not only to visualize a 3D representation of a robot but also to see its relationship with a certain workspace and the manipulation of objects or the interaction with other robots that coexist in the same environment. All the simulated objects in Gazebo have mass, speed, friction, and many other attributes that allow them to react realistically when pushed, pulled, lifted, etc. [12]. Other features that make it especially interesting in space robotics are the possibility of introducing sensors in the environment, modeling their operation and responding to the environment, being able to simulate 3D cameras, force sensors, etc.

Gazebo does not use its own physics engine but rather implements an interface to be able to use external physics engines. At the time of writing this paper, Gazebo can use four different physics engines: ODE, Bullet, Simbody and DART. Each physics engine has different characteristics and configuration parameters. There are numerous studies where comparisons are made among such engines based on a specific test bench [13]. There is no physics engine that is better than the others in all aspects. Thus, depending on the configuration and tests carried out, the engine that obtains the best results may vary.

Gazebo allows for easily interchange of them as well as for setting ad-hoc parameters of each of the physics engines. A very important feature is that Gazebo allows the modification and/or extension of the functions of the physics engines through plugins. OnOrbitRos uses an ad-hoc developed plugin, called OORplugin and also shown in Fig. 1, to modify the parameters of the physics engine used by Gazebo to eliminate effects such as gravity, wind or magnetism, while applying the torques and forces corresponding to OOS applications (such as gravity gradient) and simulating a frictionless environment.

3. OnOrbitROS architecture

This section describes the main properties and architecture of the ROS framework OnOrbitROS for the simulation of space robotics.

Fig. 1 represents the main modules and architecture of OnOrbitROS. As described in the next paragraphs, OnOrbitROS allows the simulation of simple orbits and other more complex orbits by using external libraries or even trajectories that do not follow a specific orbit. OnOrbitROS is implemented as a ROS publisher node that generates the trajectories described by the bodies that compose the simulation.

Section 3.1 shows the main reference frames considered to simulate a robotic system by using OnOrbitROS. Once the references frames have been defined, the configuration required to simulate the different kind of trajectories are indicated in Section 3.2.

3.1 Main reference frames

Fig. 2 represents the main coordinate frames of the simulated OOS scenario. The on-orbit spacecraft is denoted by target spacecraft and the corresponding coordinate frame, $\boldsymbol{F}_\mathrm{t}$, is located at its center of gravity. An additional coordinate frame, $\boldsymbol{F}_\mathrm{c}$, is located at the center of gravity of the chaser spacecraft. In a robotic OOS application, a robot can be located in this last spacecraft.

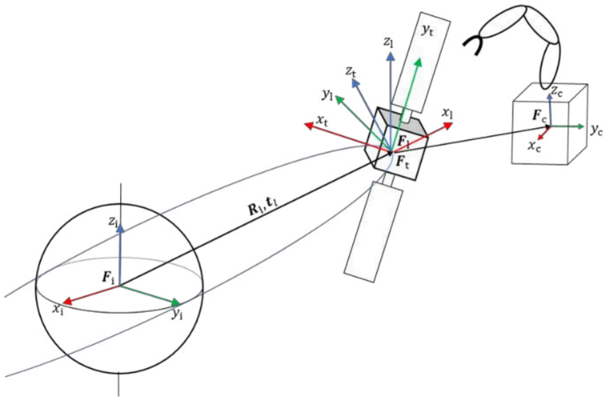

Fig 2. OOS scenario and main reference frames

On the other hand, it is necessary to define an inertial reference frame. This reference frame, commonly known as Earth-Centered Inertial frame (ECI), is referenced by $\boldsymbol{F}_\mathrm{i}$. The reference frame describing the theoretical position of the orbit described by a spacecraft is known as LVLH and is defined by $\boldsymbol{F}_\mathrm{l}$. As shown in Fig. 2, the matrices $\boldsymbol{R}_\mathrm{l}$ and $\boldsymbol{t}_\mathrm{l}$ represents the rotation and translation of the LVLH frame with respect the inertial frame. Both matrices are computed by the module that calculates the theoretical orbit of the target spacecraft (shown in blue in Fig. 1, "Trajectory generation", see also section 3.2).

3.2 Orbit and trajectory modules

OnOrbitROS allows for the simulation of simple orbits, other orbits by using external libraries or even trajectories that do not follow a specific orbit:

Simple Orbit. This module can be used for the generation of trajectories that describe elliptical orbits. It is the default functioning of OnOrbitROS and it does not requires any external library. It allows an easy execution with an optimal implementation in ROS. In this case, orbital parameters such as the semi major axis, a , eccentricity, e, argument of periapsis, ω, right ascension of ascending node, Ω, inclination, i, and instant of time of perigee passage, t_p should be defined and the simulation is carried on through a simple Keplerian propagation.

SGP4 Adapter. This module can be used to simulate other kind of orbits. This module creates an interface to the spacetrack library [14][15] so that it can obtain the parameterization from the ROS parameter server, in a similar way to the module described above, while publishing the position and orientation (and the corresponding velocity) of the LVLH frame as mentioned above.

Custom Path. This module can be used to model trajectories that do not correspond to an elliptical orbit. This last module makes use of the KDL library [16] to generate trajectories based on a series of points by interpolation between them by splines. In this case, a series of points are introduced by the user in the ROS parameter server (a data structure composed by a set of positions, velocities and accelerations, and the time instant in seconds to achieve each pose), and the Custom Path module generates the interpolated trajectory.

All the previous information is provided by the user in the YAML configuration file and stored in the ROS Parameter Server. The ROS Parameter server is common to all nodes and this allows for sharing parameters between different nodes. The user defines the orbit or trajectory that best fits the trajectory to be modelled in the launch file, and the Publisher Node instantiates the corresponding object.

4. Robot Control and OnOrbitROS

In order to show the capabilities of the tool, a simulation of an on-orbit operation is presented in this and the next section. Specifically, this section describes the implementation and simulation of a computed torque-based controller in a free-floating manipulator using OnOrbitROS.

The joint coordinates of the manipulator are represented by $\boldsymbol{q} \in \Re^\mathrm{n}$. This manipulator is located in a base spacecraft. A coordinate frame, B, is located at the centre of mass of the spacecraft. Its position and orientation are represented by $[\boldsymbol{r}^T, \boldsymbol{\theta}^T]^T$ with respect the earth centred inertia coordinate frame. $\boldsymbol{\theta}$ contains the yaw, pitch, roll Euler angles representing the orientation of frame B with respect to the inertial coordinate frame

The system dynamics of the manipulator can be represented in the following form:

$$\boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{M}_{bb} & \boldsymbol{M}_{b\mathrm{m}} \\ \boldsymbol{M}_{b\mathrm{m}}^T & \boldsymbol{M}_{\mathrm{mm}} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{r}} \\ \ddot{\boldsymbol{\theta}} \\ \ddot{\boldsymbol{q}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{c}_b \\ \boldsymbol{c}_\mathrm{m} \end{bmatrix} \tag{1}$$

where $\boldsymbol{\tau}$ is the input vector, $\boldsymbol{M}_{bb} \in \Re^{6\times 6}$ is the inertia matrix of the base spacecraft, $\boldsymbol{M}_{bm} \in \Re^{6\times \mathrm{n}}$ is the coupled inertia matrix of the spacecraft and the manipulator, $\boldsymbol{M}_\mathrm{mm} \in \Re^{\mathrm{n}\times\mathrm{n}}$ is the inertia matrix of the manipulator; $\boldsymbol{c}_b$, and $\boldsymbol{c}_\mathrm{m} \in \Re^6$ are a velocity/displacement-dependent, nonlinear terms for the spacecraft, and manipulator, respectively.

The dynamic equation in (1) can also be represented as:

$$\boldsymbol{M}^* \ddot{\boldsymbol{q}} + \boldsymbol{C}^* = \boldsymbol{\tau}^* \tag{2}$$

where $\boldsymbol{M}^* \in \Re^{\mathrm{n}\times\mathrm{n}}$ is the generalised inertia matrix, $\boldsymbol{C}^* \in \Re^\mathrm{n}$ is the generalised Coriolis and centrifugal vector for the manipulator arm, defined explicitly as:

Table 1. Kinematic and dynamic parameters of the humanoid robot

	Mass	Height	Inertia (kg·m2)					
	(kg)	(m)	Ixx	Iyy	Izz	Ixy	Ixz	Iyz
Body Parameters	93	0.843	18.6	15.4	4.1	-0.008	-0.027	0.058

	Mass	Length	Inertia (kg·m2)					
	(kg)	(m)	Ixx	Iyy	Izz	Ixy	Ixz	Iyz
Link 1	2.741	0.28	0.0124	0.0042	0.0136	3.6e-05	7.1e-05	-0.0002
Link 2	2.425	0.144	0.013	0.0138	0.0049	1.2e-05	-0.0032	-0.0001
Link 3	2.209	0	0.007	0.0069	0.0039	-0.0001	0.0007	0.0004
Link 4	0.877	0.274	0.0025	0.0027	0.0012	0.0001	-0.0003	0.0004
Link 5	1.878	0.265	0.0035	0.0044	0.0023	1.3e-05	1.03e-05	-9.7e-05
Link 6	0.409	0	0.0001	0.00014	0.00015	-8.9e-08	-4.4e-08	4.2e-07
Link 7	0.308	0	0.0003	0.0002	0.00017	-1.6e-06	1.7e-06	-1.2e-05

$$\boldsymbol{M}^* = \boldsymbol{M}_{mm} - \boldsymbol{M}_{bm}^T \boldsymbol{M}_{bb}^{-1} \boldsymbol{M}_{bm} \qquad (3)$$

$$\boldsymbol{C}^* = \boldsymbol{c}_m - \boldsymbol{M}_{bm}^T \boldsymbol{M}_{bb}^{-1} \boldsymbol{c}_b \qquad (4)$$

The linear and angular momenta of the system $[\boldsymbol{\ell}^{\mathrm{T}}, \boldsymbol{\Psi}^{\mathrm{T}}]^{\mathrm{T}} \in \Re^6$ can be defined as:

$$\begin{bmatrix} \boldsymbol{\ell} \\ \boldsymbol{\psi} \end{bmatrix} = \boldsymbol{M}_{bb} \boldsymbol{v}_b + \boldsymbol{M}_{bm} \dot{\boldsymbol{q}} \qquad (5)$$

where $\dot{\boldsymbol{q}} \in \Re^{\mathrm{n}}$ represents the joint speeds of the arm and $\boldsymbol{v}_b = [\dot{\boldsymbol{r}}^T, \boldsymbol{\omega}^T]^T$. This last vector represents both the linear and angular acceleration of the base spacecraft.

The control action of the computed torque-based controller can be obtained taking into account the dynamics equation indicated in (2) and can be defined by the following equation:

$$\boldsymbol{\tau} = \boldsymbol{M}^* \boldsymbol{J}_g^+ (\ddot{\boldsymbol{p}}_r - \dot{\boldsymbol{J}}_g \dot{\boldsymbol{q}} - \dot{\boldsymbol{v}}_{gm}) + \boldsymbol{C}^* \qquad (6)$$

where $\boldsymbol{J}_g$ is the so-called Generalized Jacobian Matrix, that can be obtained by using the following equation:

$$\boldsymbol{J}_g = \boldsymbol{J}_m - \boldsymbol{J}_b \boldsymbol{M}_{bb}^{-1} \boldsymbol{M}_{bm} \qquad (7)$$

where $\boldsymbol{J}_m \in \Re^{6\times n}$ is the robot manipulator Jacobian, and $\boldsymbol{J}_b \in \Re^{6\times 6}$ is the Jacobian of the base spacecraft. Additionally, the reference acceleration indicated in Equation (6) can be obtained by:

$$\ddot{\boldsymbol{p}}_r = \ddot{\boldsymbol{p}}_d + \boldsymbol{K}_d(\dot{\boldsymbol{p}}_d - \dot{\boldsymbol{p}}) + \boldsymbol{K}_p(\boldsymbol{p}_d - \boldsymbol{p}) \qquad (8)$$

where $\boldsymbol{p}_d$, $\dot{\boldsymbol{p}}_d$, and $\ddot{\boldsymbol{p}}_d$ are the desired position, velocity and acceleration to be tracked and $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are proportional and derivative matrices to be defined. Finally, the term $\dot{\boldsymbol{v}}_{gm}$ in Equation (6) can be computed by using the method described in [17].

4. Results

This section describes the simulation results obtained in the use of OnOrbitROS for the simulation of a humanoid robot (see Fig. 3) during the tracking of trajectories by using the controller presented in Section 3. Table 1 represents the main properties of the humanoid robot considered in this section. This table represents a torso with two manipulators with seven degrees of freedom whose main kinematic and dynamic parameters are shown in this table. Only the simulation of an arm is considered in this Section, so $\boldsymbol{M}_{bb}$ is the inertia of the torso and $\boldsymbol{M}_{\mathrm{mm}}$ the inertia of the simulated arm.

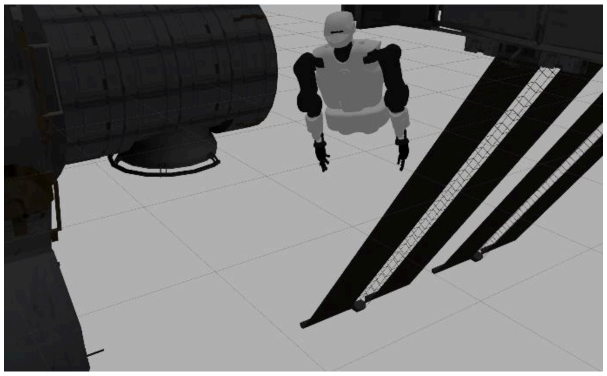

Fig 3. OnOrbitROS simulation of the humanoid robot

To evaluate the tracking controller, the desired trajectory represented in Fig. 4 is considered (the initial, intermediate and final positions of the trajectory are represented in this figure). Only one arm of the humanoid robot is carrying out the tracking. Specifically, the main displacement is performed along the x direction of the robot body coordinate frame (0.5m) while maintaining the position along y direction and a parabolic trajectory is performed along the z direction to allow the arm motion.

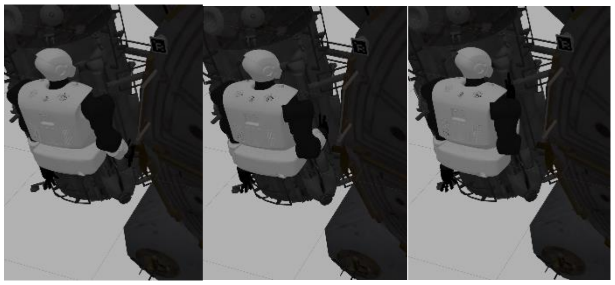

Fig. 4. 3D simulation of the desired trajectory

The following figures show the main results obtained during the tracking of the desired trajectory. Fig. 5 represents the desired trajectory (black) and the obtained trajectory (red) during the tracking. Both trajectories are almost coincident. To see more clearly the error during the tracking, Fig. 6 represents the tracking error (position and attitude respectively). Both these last two figures show that the tracking error remains limited during the tracking, and the desired location is correctly achieved by tracking the desired trajectory.

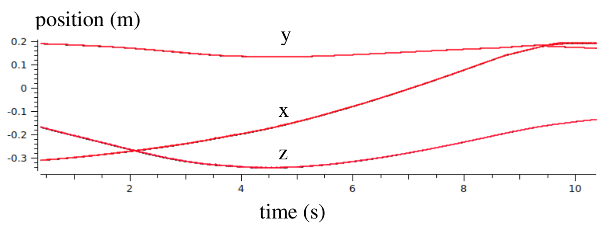

Fig. 5. Desired (black) and obtained 3D trajectory (red) during the tracking

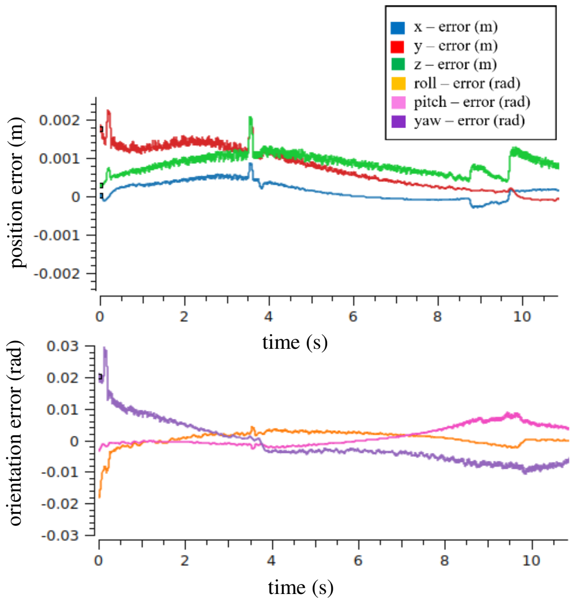

Fig. 6. Control error during the tracking. a) Position error. b) Orientation error.

Fig 7 represents the control action generated by the proposed controller in Equation (6). As it shown, the torques remain contained in a range of $\pm 1\ Nm$ during the tracking.

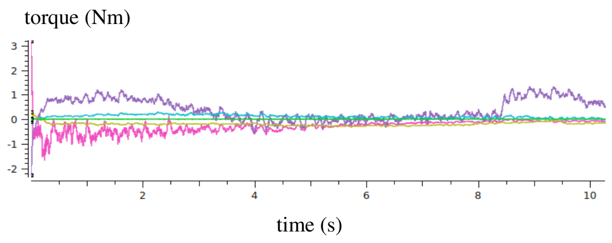

Fig. 7. Control action (torque) during the tracking

6. Conclusions

The use of ROS allows not only the rapid definition of the robot's properties, but also the simulation of complex workspaces with different kind of sensors. This paper presents a ROS framework for the simulation of robots on orbit. More specifically, this paper presents the architecture, main objectives, implementation, and available options of the OnOrbitROS framework. OnOrbitROS allows to extend the previous robotics developments to the specific properties of space robotics. Additionally, a torque-based controller is presented for the guidance of free-floating manipulators. This controller is applied to the guidance of one of the arms of a humanoid robot simulated using OnOrbitROS, showing that the tool can be easily be used for simulating and testing ad-hoc built control strategies for on-orbit manipulation

Currently, we are working on adding new features to OnOrbitROS. These include the possibility to set non-elliptical reference trajectories, the addition of other different controllers and the inclusion of other perturbations (e.g. drag forces and torques) into the physical engine, in addition to the already existing gravity gradient.

References

[1] M. A. Diftler et al., Robonaut 2 - The first humanoid robot in space, 2011 IEEE International Conference on Robotics and Automation, (2011) 2178-2183.

[2] S. Nolet. "The SPHERES Navigation System: from Early Development to On-Orbit Testing," AIAA 2007-6354. AIAA Guidance, Navigation and Control Conference and Exhibit. August 2007. https://doi.org/10.2514/6.2007-6354

[3] M.G. Bualat, T. Smith, E.E. Smith, T. Fong, D.W. Wheeler. "Astrobee: A New Tool for ISS Operations," AIAA 2018-2517. 2018 SpaceOps Conference. May 2018, https://doi.org/10.2514/6.2018-2517

[4] J.L. Ramirez-Riberos, M. Pavone, E. Frazzoli, D.W. Miller, "Distributed Control of Spacecraft Formations via Cyclic Pursuit: Theory and Experiments", Journal of Guidance, Control, and Dynamics 2010 33:5, 1655-1669, https://doi.org/10.2514/1.46511

[5] N. Radford et al., "Valkyrie: NASA's First bipedal humanoid robot", Journal of Field Robotics Vol. 32, No. 3, 2015, pp. 397-419. https://doi.org/10.1002/rob.21560.

[6] M. Allan, I. Chen, Gazebo Renders the Moon, RosCon 2018, Workshop on Lunar Mapping for Precision Landing (2018).

[7] S. Wang, Y. Cao, X. Zheng, T. Zhang, Collision-Free Trajectory Planning for a 6-DoF Free-Floating Space Robot via Hierarchical Decoupling Optimization, IEEE Robotics and Automation Letters, 7(2) (2022) 4953-4960.

[8] J. L. Ramon, J. Pomares, L. Felicetti. OnOrbitROS Github 5 July 2022, https://github.com/OnOrbitROS/, (accessed 20.07.22).

[9] P. Estefo, J. Simmonds, R. Robbes, J. Fabry. The Robot Operating System: Package reuse and community dynamics. The Journal of Systems and Software, 151 (2019) 226-242.

[10] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng. ROS: an open-source Robot Operating System. ICRA Workshop onOpen Source Software (2009).

[11] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, E. Fernandez. ros_control: A generic and simple control framework for ROS. Journal of open source software, 2(20), (2017), 456.

[12] N. Koenig, A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2005).

[13] S. J. Chung, N. Pollard. Predictable behavior during contact simulation: a comparison of selected physics engines: Predictable behavior during contact simulation. Computer Animation and Virtual Worlds, 27 (3-4), (2016) 262-270.

[14] D. Vallado, P. Crawford. SGP4 Orbit Determination, AIAA 2008-6770. AIAA/AAS Astrodynamics Specialist Conference and Exhibit. August (2008).

[15] D. Vallado, P. Crawford, R. Hujsak, T.S. Kelso. Revisiting Spacetrack Report #3," AIAA 2006-6753. AIAA/AAS Astrodynamics Specialist Conference and Exhibit. (2006).

[16] R. Smits, KDL: Kinematics and Dinamics Library, https://www.orocos.org/kdl, (accessed 20.07.22).

[17] N. Kostas, E. G. Papadopoulos, On the Dynamics and Control of Free-floating Space Manipulator Systems in the Presence of Angular Momentum Frontiers in Robotics and AI, 4, (2017).