MULTIPLATFORM SIMULATION USING ROS

Caroline Cristine Duarte da Silva[1], Yasmin Castro[1], Andrew Sarmento[1] & Emilia Villani[1]

[1]Aeronautics Institute of Technology(ITA)

Abstract

Robotics is a constantly evolving field that benefits from the use of tools powerful tools for robot development and simulation. Two of these tools, widely used ROS (Robot Operating System) and CoppeliaSim (formerly known as V-REP). ROS is an open-source framework widely used in the robotics community. On the other hand, CoppeliaSim is a simulation platform for powerful and versatile 3D robots. In this paper, we distributed an F16 simulation using ROS to create tasks, combining flight visualization by Flight Gear and collision analysis of a robotic flight simulator using CoppeliaSim.

Keywords: Flight Simulator, Distributed Simulation, ROS.

1. Introduction

Distributed simulation in aircraft refers to the utilization of interconnected, networked simulations to replicate the behavior, functionalities, and interactions of various aviation systems. This approach allows for comprehensive testing and analysis of aircraft technologies, flight procedures, and scenarios in a collaborative virtual environment.

The implementation of distributed simulation involves linking simulators or computational models of different aircraft components, such as flight controls, interceptors, engines, and environmental systems, into a cohesive network. These simulations communicate in real-time, exchanging data and responses to mimic the complexities of actual flight conditions.

One of the primary advantages of distributed simulation in aircraft is its ability to facilitate cost-effective and comprehensive scenarios. Pilots, maintenance personnel, and other aviation professionals can engage in simulated flight operations, emergency procedures, or system malfunctions without requiring access to a physical aircraft.

Moreover, distributed simulation enhances the evaluation and validation of new technologies or modifications to existing systems. Engineers and researchers can conduct thorough tests on software upgrades, system integration, or aircraft designs within a controlled virtual environment before implementing them in actual aircraft. This helps in identifying potential issues, ensuring safety, and refining the performance of aviation systems before deployment.

However, distributed simulation in aircraft also presents challenges. Achieving synchronization among distributed simulations, ensuring real-time data exchange, and maintaining consistency across interconnected models are crucial technical hurdles. Additionally, cybersecurity concerns, data integrity, and network reliability must be addressed to guarantee the accuracy and security of simulated environments.

2. Literature Review

Flight simulators are sophisticated training tools designed to replicate the experience of flying an aircraft in a safe and controlled environment. These simulations have evolved significantly since their inception, offering an immersive experience that closely real-world flight conditions [1].

Flight simulators serve multiple purposes, primarily for pilot training and aircraft development. They provide a cost-effective and risk-free environment for pilots to learn and practice various maneuvers, emergency procedures, and instrument operations [2].

Moreover, the paper[2] reflects the advancements in the foundational technologies of flight simulation, ranging from mathematical modeling to real-time computation, motion actuation, visual image generation systems, and projection systems.
A collaboration between ITA and EMBRAER has given rise to the SIVOR project (Robotic Flight Simulator), aimed at exploring the feasibility and boundaries of employing anthropomorphic robots as motion systems for flight simulators. This innovative simulator features a high-fidelity cockpit equipped with an integrated vision system. Central to its design is a 6 degree-of-freedom robot mounted on a rail, effectively serving as a seventh axis for the robot's motion [3]. 'Figure 1' illustrates SIVOR.

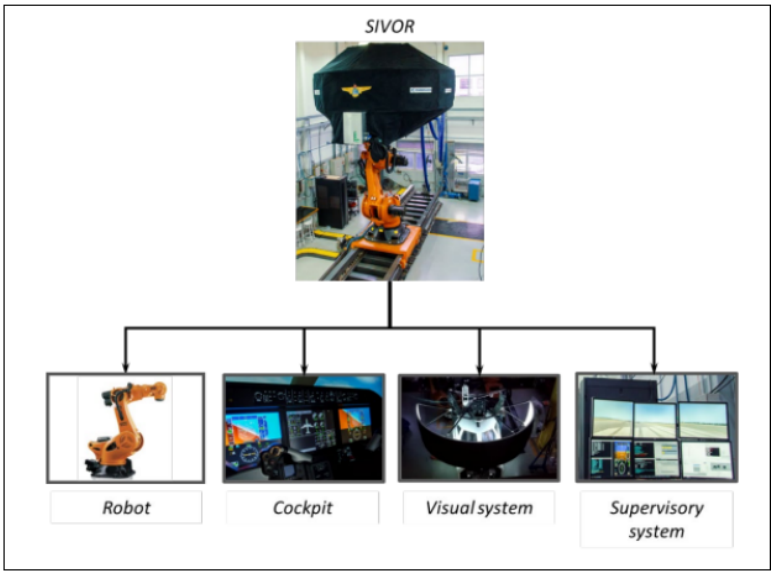

Figure 1 – SIVOR from [3].

2.1 Model Background

2.1.1 Aircraft Model

In this paper, we use a linearized model of the F16 aircraft. From [4], the state-space is described by the 'Equation 1 '.

$$\begin{aligned} \dot{X} &= AX + BU \\ Y &= CX + DU. \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{15\times15}$ is the state matrix; $B \in \mathbb{R}^{15\times4}$ is the input matrix; $C \in \mathbb{R}^{28\times15}$ is the output matrix; $D \in \mathbb{R}^{28\times4}$ is the direct transition (or feedthrough) matrix; $X \in \mathbb{R}^{15\times1}$ is the states; $U \in \mathbb{R}^{4\times1}$ is the control inputs and $Y \in \mathbb{R}^{28\times1}$ is the output variable.
The transposed X vector contains the following variables: $x' = [u\ w\ v\ p\ q\ r\ x\ y\ h\ \phi\ \theta\ \psi\ power\ lat\ lon]$, where u is the x-axis component of body velocity; w is the y-axis component of body velocity; v is the z-axis component of body velocity; p is the roll rate; q is the pitch rate; r is the yaw rate; x is the position in x-axis; y is the position in y-axis; h is the altitude; ϕ, θ and ψ are the Euler angles; $power$ is the engine power; lat is the latitude; lon is the longitude.
The transposed U vector contains the following variables: $U' = [\delta_t\ \delta_e\ \delta_a\ \delta_r]$, where δ_t is the throttle; δ_e is the elevator deflection; δ_a is the aileron deflection; δ_r is the rudder deflection.
The transposed y vector contains the following variables:
$y' = [V\ \alpha\ \beta\ CX\ CY\ CZ\ Cl\ Cm\ Cn\ CD\ CS\ CL\ T\ cpower\ power\ n_{C,b}\ n_{P,b}\ \dot{\phi}\ \dot{\theta}\ \dot{\psi}\ \dot{h}\ \dot{x}\ \dot{y}\ \dot{z}]$, where V is the aircraft's body velocity; α is the angle of attack; β is the sideslip angle; CX, CY, CZ, Cl, Cm, Cn, CD, CS and CL are the aerodynamic coefficients; T is the thrust; $cpower$ is the commanded throttle; $power$ is the power; $n_{C,b} \in \mathbb{R}^{3\times1}$ is the load factor at the aircraft center of gravity; $n_{P,b} \in \mathbb{R}^{3\times1}$ is the load factor at the pilot's position; $\dot{\phi}$ is the angular velocity about x-axis; $\dot{\theta}$ is the angular velocity about y-axis; $\dot{\psi}$ is the

angular velocity about z-axis; $\dot{h}$ is the oposite of $\dot{z}$; $\dot{x}$ is the linear velocity about x-axis; $\dot{y}$ is the linear velocity about y-axis; $\dot{z}$ is the linear velocity about z-axis.
All of the conditions used during the equilibrium calcularion are already displayed in the trimmed flight parameters[1], as well as the space-state matrices[2]: A, B, C and D.

2.1.2 Washout Filter Model

The washout filter, often referred to as the motion cueing algorithm, serves to translate the expansive movement range of an aircraft into the confined space of a flight simulator[1]. Typically, it takes inputs such as linear accelerations and angular velocities of the aircraft, producing position and rotation commands for the motion platform.
The traditional washout filter operates through three channels, aiming to prevent movement commands from surpassing the physical constraints of the platform[3]. In 'Figure 2', the translational channel consists of high-pass filters, designed to capture high-frequency maneuvers across the three axes of the aircraft. Likewise, the rotation channel incorporates high-pass filters to depict high-frequency rotations along the three axes of the plane.
Meanwhile, the tilt coordination channel aims to emulate sustained acceleration maneuvers along the lateral and longitudinal axes of the airplane. Attempting to replicate low-frequency accelerations could lead the platform to surpass its operational limits too easily. The entire SIVOR washout model is in [5].

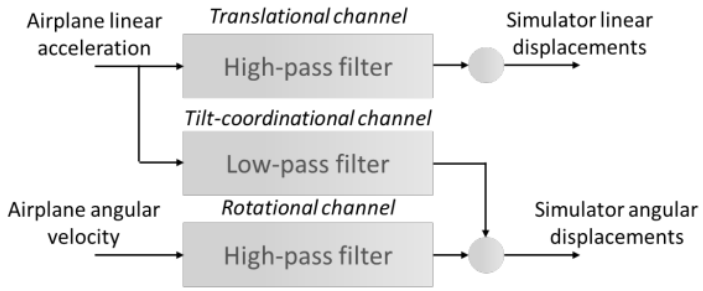

Figure 2 – Classical washout filter from [1].

2.1.3 KR-Titan Robot Model

In [6], the authors propose an alternative method for discerning the dynamics of industrial robotic systems, treating them as closed-loop Position IN/Position OUT systems. They focus on the external dynamics of a conventional 7-degrees-of-freedom (7-DoF) serial manipulator alongside its real-time positional controller.
The identified model illustrates how an external client application interacts with the internal control loop. The experiment detailing the identification process and the resulting dynamics are discussed. Its application is particularly relevant in the context of robotic flight simulators and sensor-integrated industrial robotic systems.

2.1.4 Coppellia Model

The CoppeliaSim model encompasses collision assessments between the cockpit and the robot arm, irrespective of the pilot's commands, ensuring a comprehensive evaluation of simulated actions. Subsequently, with physical SIVOR we can realize research encompassing the development of novel technologies and the analysis of human factors. In 'Figure 4', we have a front and side view of the model in CoppeliaSim model.

[1] https://drive.google.com/file/d/1cN8V4D2-QobNFCeRAuGH5VD23WZEh8Xa/view?usp=sharing
[2] https://drive.google.com/file/d/1cN8V4D2-QobNFCeRAuGH5VD23WZEh8Xa/view?usp=sharing

Figure 3 – KR-Titan from [5].

3. Methodology

Given these challenges, this paper aims to address them by utilizing ROS[7], which stands for Robot Operating System, as the primary connectivity tool to establish a distributed simulation of aircraft, focusing on its flight dynamics.

ROS is an open-source framework designed to simplify the development of robotics software. Despite its name, ROS is not an operating system in the traditional sense; rather, it serves as a middleware that provides a structured environment for various robotic applications and functions.

At its core, ROS offers a collection of tools, libraries, and conventions aimed at assisting developers in building complex and modular robotic systems. It provides a framework that facilitates communication between different components of a robot's software architecture, enabling seamless interaction between sensors, actuators, controllers, and higher-level algorithms.

One of the key strengths of ROS is its emphasis on modularity and reusability. It utilizes a decentralized architecture, allowing developers to create individual software modules known as "nodes." These nodes can perform specific tasks, such as processing sensor data, controlling actuators, or implementing algorithms, and communicate with each other through a publish-subscribe messaging system.

ROS provides a set of communication protocols that enable nodes to share data, such as sensor readings or control commands, using topics, services, and actions. Topics facilitate asynchronous communication by allowing nodes to publish and subscribe to messages, while services enable synchronous communication for requesting specific tasks from other nodes. Actions offer a way to execute long-running tasks with feedback.

Another notable aspect of ROS is its extensive library of packages, tools, and resources contributed by a large community of developers. These packages cover various functionalities, including mapping and localization, path planning, manipulation, perception, and simulation. This rich ecosystem allows developers to leverage existing solutions and integrate them into their robotics projects, accelerating development and fostering collaboration within the robotics community. ROS is platform-independent and supports multiple programming languages, primarily C++ and Python, making it accessible to a broad range of developers.

Furthermore, it provides simulation capabilities through tools like Gazebo, enabling developers to test and validate their algorithms and applications in simulated environments before deploying them into physical robots[8]. Overall, ROS has become an achievement in the field of robotics, empowering researchers, hobbyists, and industry professionals to create sophisticated robotic systems by providing a flexible, modular, and collaborative framework for developing robotics software. Its continued development and widespread adoption contribute significantly to the advancement of robotics technology.

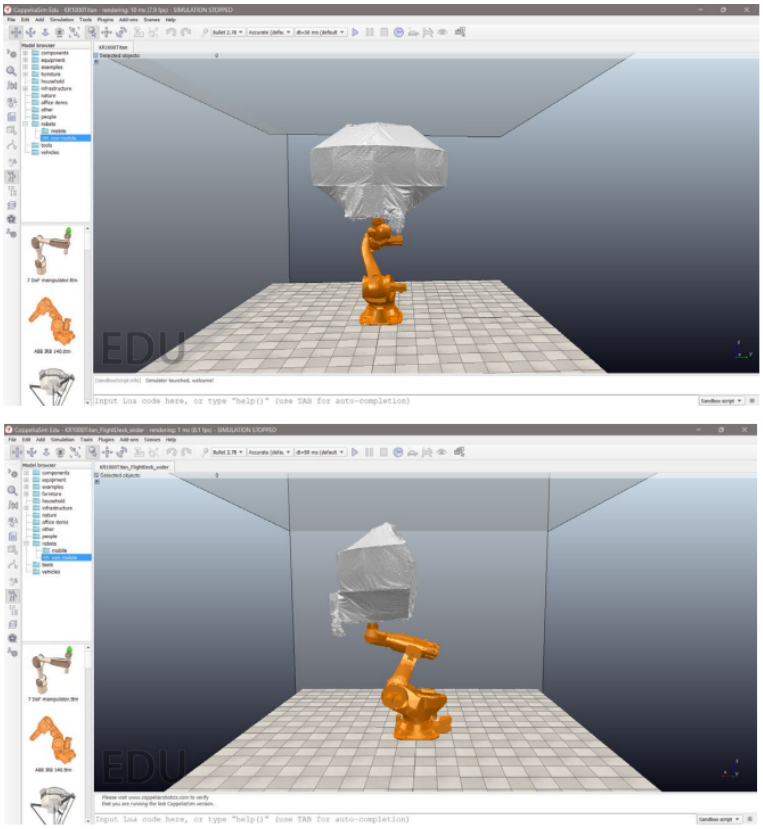

Figure 4 – CoppeliaSim model from [2].

In this paper, we propose that the researchers working on autonomous flight or advanced flight control algorithms can leverage ROS to develop and test these systems in simulated environments. By interfacing with ROS-nodes representing flight controllers, autopilot systems, or autonomous decision-making algorithms, developers can validate these systems before deploying them to actual aircraft.

4. Proposed Architecture

A significant aspect highlighted in our paper is the development of an architecture utilizing ROS. This architecture facilitates the testing of various maneuvers on an aircraft, enabling the validation of simulation movements within a robotic flight simulator built in CoppeliaSim software. Emphasizing the importance of this validation process, it serves as the initial stage before implementing any maneuvers within the SIVOR physical simulator at Aeronautics Institute of Technology(ITA).

In our paper, we replicated the dynamics of an aircraft, the F16 model, distributing both input and output of the model through ROS. Initial tests demonstrated the feasibility of integrating diverse operating systems(OS) and software, creating an interconnected system capable of executing the simulation.

To enhance comprehension of the proposed architecture, a visual representation was crafted in 'Figure 5'.

The diagram illustrates the simulation architecture incorporating two distinct operating systems. We utilized ROS1, establishing topics to manage the inputs and outputs associated with the F16 model. We opted for the Linux operating system as our master platform due to its comprehensive access to all ROS tools, in the diagram this is represented by the black arrow.

On Windows OS, we utilize Simulink, accessing the ROS plugin solely through MATLAB. It is worth noting that the computers are using the same local network. We use publisher and subscriber to transfer data between the created topics, in the diagram this is represented by the red arrow. Our proposal involves a four computer.

The first computer(PC1) manages the input, emphasizing the initial computer significance, it serves as the focal point for the human-machine interface (HMI). However, first computer(PC1) contains the dynamics model and the LQR controller.

The second computer(PC2) contains the washout filter and identified model of the KR-Titan robot.

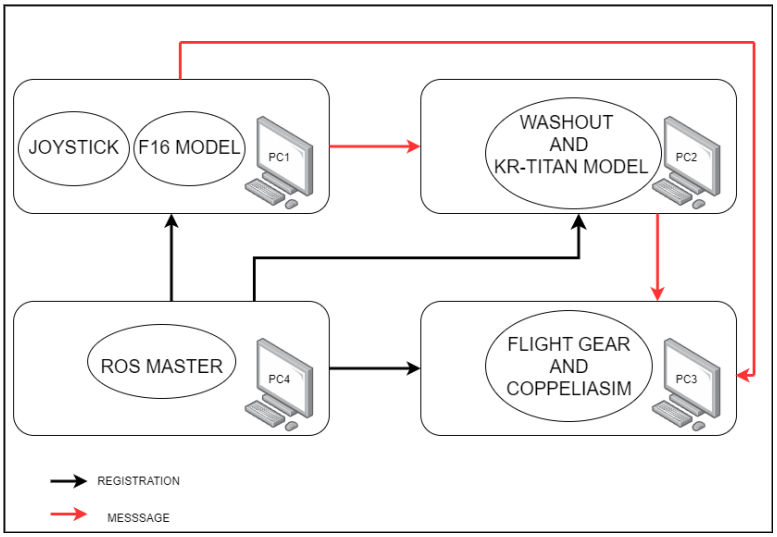

Figure 5 – Simulation architecture.

The third computer(PC3) contains the SIVOR model in CoppeliaSim and FlightGear visualization. Lastly, the fourth computer(PC4) contains ROS master.
The rqt serves as a software framework within ROS, embodying a range of GUI tools through plugins. It allows users to execute all available GUI tools as dockable windows within its interface.
In 'Figure 6', the ROS rqt graph for the simulation depicts nodes within the system and the topics facilitating their communication. Nodes are represented by ovals, while rectangles represent ROS topics. Arrows indicate the direction of data flow, indicating whether a node is publishing to or subscribing from a topic.

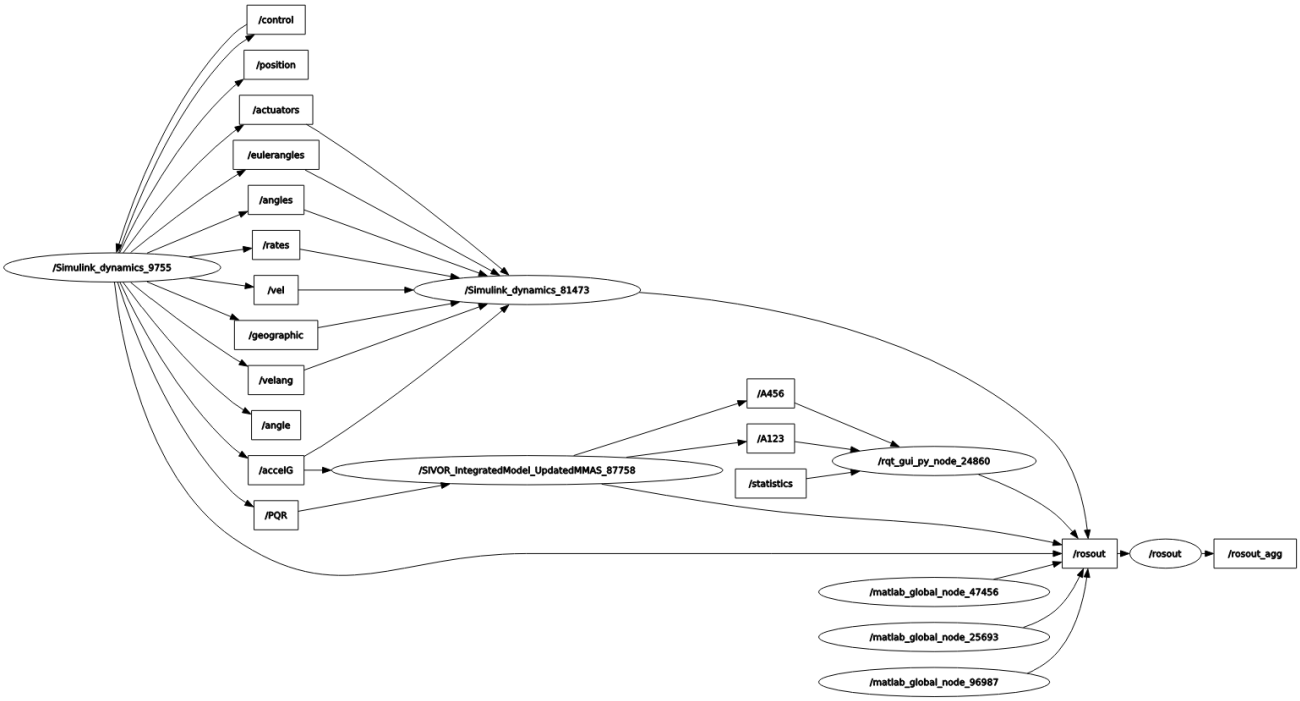

Figure 6 – ROS rqt graph.

According to 'Figure 6', the system has 17 topics; 14 were created by us; the other 3 are standard ROS masters; they are $/statistics$, $/rosout$ and $/rosout\_agg$. As for nodes, we have 8, 6 were created by us; 1 of which is created by default by ROS ($/rosout$) and 1 is the rqt tool.

5. Results

For the results section, we simulate two cases: a flight without a collision in the CoppeliaSim model and a flight with a collision in the CoppeliaSim model. It is important to remember that in a distributed simulation, we have different frequencies in different systems. In our case, we have the F16 frequency and the washout frequency with the KR-Titan model. We choose $\frac{1}{30}\ s$ for F16 dynamics and $0.012s$ for washout filter.

There is a way to get the operating freqency (Hz) of nodes, we use the rqt plot and tick the topics to see the message rate in real-time. In 'Figure 7', we can see the diferents rates. Topics $/A123$ and $/A456$ correspond to the angles leaving the node, which correspond to the washout filter and KR-Titan model, their frequencies are the same as the robot model. These angles will be used to move the robotic joints of the CoppeliaSim model.

Still in 'Figure 7', only topic with the same frequency of the F16 dynamics is the $/control$. The $/control$ topic corresponds to the input of the control surfaces by the joystick. The remaining topics were set at a rate of 10 Hz, which can be seen in 'Figure 7'.

Topic	Type	Bandwidth	Hz	Value
/control	geometry_msgs/Quaternion	952.94B/s	30.36	
/PQR	geometry_msgs/Point	240.64B/s	10.00	
/vel	geometry_msgs/Point	240.62B/s	10.00	
/rates	geometry_msgs/Quaternion	320.79B/s	10.00	
/velang	geometry_msgs/Point	240.58B/s	10.00	
/accelG	geometry_msgs/Point	240.56B/s	10.00	
/angles	geometry_msgs/Point	240.54B/s	10.00	
/eulerangles	geometry_msgs/Point	240.53B/s	10.00	
/geographic	geometry_msgs/Point	240.51B/s	10.00	
/angle	geometry_msgs/Point	240.49B/s	10.00	
/position	geometry_msgs/Point	240.47B/s	10.00	
/actuators	geometry_msgs/Quaternion	320.59B/s	10.01	
/A456	geometry_msgs/Point	2.00KB/s	83.51	
/A123	geometry_msgs/Point	2.00KB/s	83.51	

Figure 7 – Topics rates.

5.1 Case 1- Without Collision

In the first case, we perform a pitching movement. According to 'Figure 8', the black line represents the G-acceleration in the x-direction, the blue line represents the G-acceleration in the y-direction, and the green line represents the G-acceleration in the z-direction. Here the term G-acceleration is used for load factors.

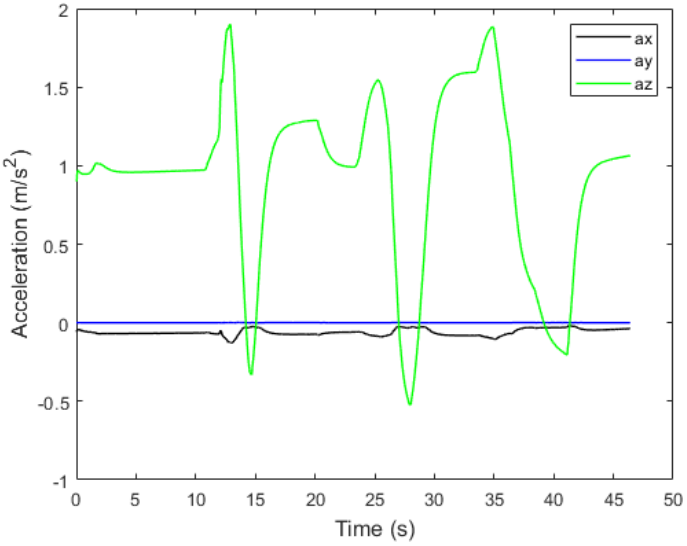

Figure 8 – F16 G-acceleration, without collision.

Still in 'Figure 8' where 'z' indicates the same direction of gravity. We can observe, through the oscillation of the graph, the movement of pitch up and pitch down. With a maximum value of almost $2G$.

In ‘Figure 9’, the black line represents the angular velocity associated with roll, the blue line represents the angular velocity associated with pitch, and the green line represents the angular velocity associated with yaw.
As expected, we see oscillations mainly in pitch, since we only performed a pitching maneuver. For a better visualization, we made a video[3] of PC3, in which we compared the pilot’s view through Flight Gear and the robot simulation with CoppeliaSim.

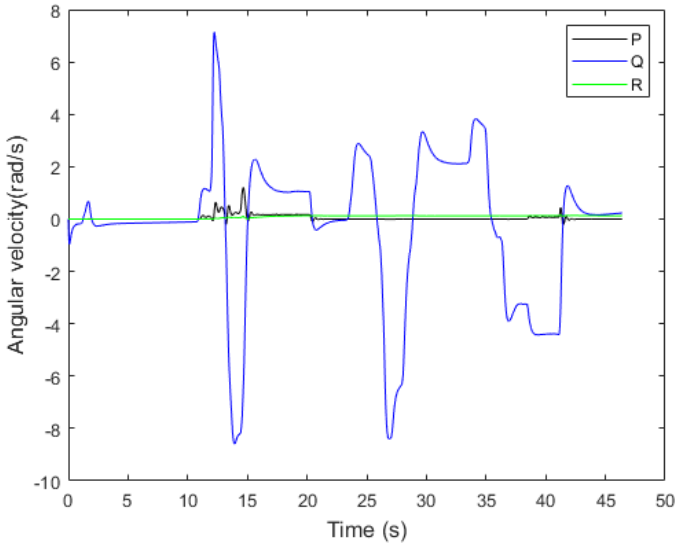

Figure 9 – F16 angular velocity, without collision.

5.2 Case 2- With Collision

In the second case, we perform again a pitching movement. According to ‘Figure 10’, the black line represents the G-acceleration in the x-direction, the blue line represents the G-acceleration in the y-direction, and the green line represents the G-acceleration in the z-direction. As previously discussed, the term G-acceleration is used for load factors

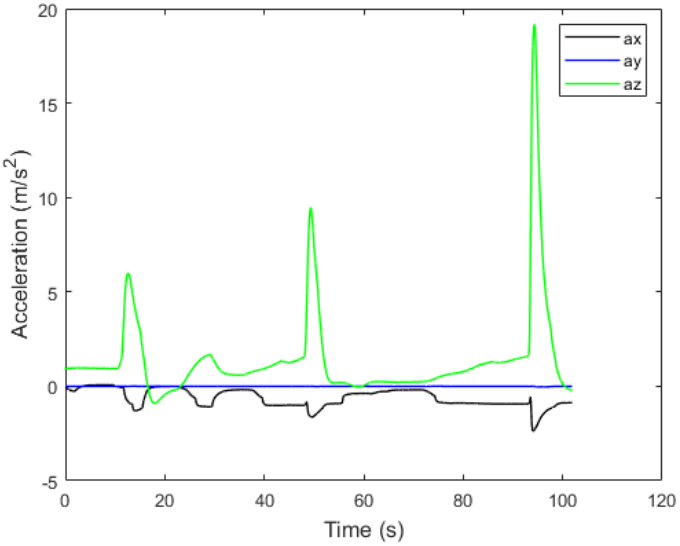

Figure 10 – F16 G-acceleration, with collision.

Still in ‘Figure 10’ where ‘z’ indicates the same direction of gravity. As we wanted to demonstrate

[3]https://drive.google.com/file/d/1VMLU2vK4SWPgXp1V_QualQRtfWrsNZPm/view?usp=sharing

a robot collision, we increased the load factor, through the movement of pitch up and pitch down, represented by the green line.
Again, in 'Figure 11', the black line represents the angular velocity associated with roll, the blue line represents the angular velocity associated with pitch, and the green line represents the angular velocity associated with yaw. As expected, we see oscillations mainly in pitch, since we only performed a pitching maneuver
However, we performed a collision and exited the flight envelope. In video[4], we can notice that when a collision occurs, the cockpit turns red in CoppeliaSim.

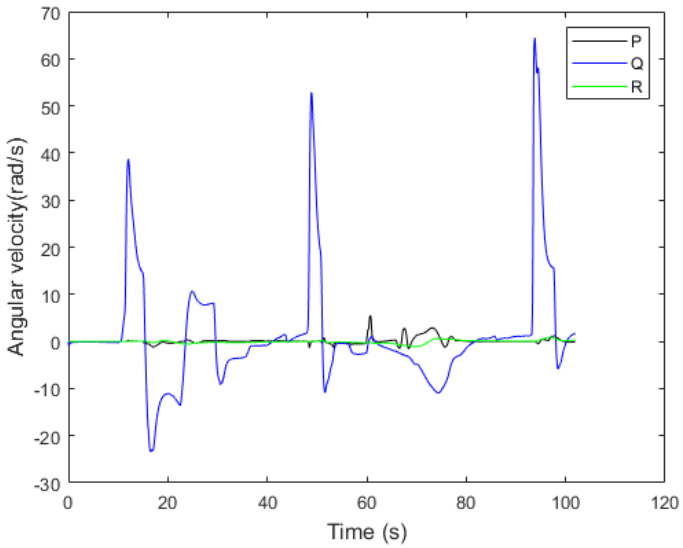

Figure 11 – F16 angular velocity, with collision.

6. Conclusion

We concluded that the use of ROS in the robotic flight simulator proved to be satisfactory, since we were able to reach the different frequencies of both the aircraft model and the robot model. Its flexibility, modularity, and integration capabilities make ROS a powerful tool for advancing the state-of-the-art in simulation, can aid research and development in the aviation industry.

7. Contact Author Email Address

The contact author email address: caroline.duarte@ccm-ita.org.br

8. Copyright Statement

9. Acknowledgements

The authors would like to thank the financial support of the following entities and projects: CAPES-PRINT Project no. 88887.310617/2018-00, FINEP Project 01.20.0195.01, FINEP Project 01.22.0478.00 and FINEP Project 01.22.0313.00.

[4]https://drive.google.com/file/d/1i8-bD1bpIZh0gCGM-jLkHYNEDKX1tFQ4/view?usp=sharing

References

[1] Vargas M. Controle de uma plataforma de movimento de um simulador de voo. *PhD Thesis*, 2009.

[2] Allerton D. The impact of flight simulation in aerospace. *Aeronautical Journal*, Vol. 114, pp 747-756, 2010.

[3] Matheus A. An optimization method for a robotic flight simulator washout filter meeting operational safety criteria. *PhD Thesis*, 2022.

[4] Stevens B, Lewis F and Johnson E. *Aircraft control and simulation*. 3rd edition, Wiley, 2016.

[5] Oliveira W, Matheus A, Rodamilans G, Nicola R, Arjoni D, Trabasso L G, Villani E and Silva E. Evaluation of the pilot perception in a robotic flight simulator with and without a linear unit. *AIAA Scitech 2019 Forum*, San Diego, California , Vol. 1, pp 7-11, 2019.

[6] Oliveira W, Matheus A, Marques W, Trabasso L G, Villani E and Rodamilans G. External dynamic behavior of an industrial robotic system. *25 th ABCM International Congress of Mechanical Engineering*, Uberlândia, Brazil, 2019.

[7] Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, Berger E, Wheeler R and Ng A. ROS: an open-source Robot Operating System. *ICRA WORKSHOP ON OPEN SOURCE SOFTWARE*, Kobe, 2009.

[8] Silva C C D. Strategies for energy efficiency in humanoid robot walking. *PhD Thesis*, 2023.