GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DEVELOPMENT OF A PREDICTIVE DIGITAL TWIN FOR REMOTE OPERATION OF NASA'S RASSOR ROVER UNDER COMMUNICATION LATENCY

Autor: Victoria Robledillo Núñez

Director: Luis Rabelo

Madrid, Julio 2026

Declaration of originality

I declare under my responsibility that the Project presented with the title **"Development of a predictive Digital Twin for remote operation of NASA's RASSOR rover under Communication latency"** at the ICAI School of Engineering of the Comillas Pontifical University in the academic year **2025/2026** is of my authorship and has not been presented previously for other purposes. The Project is not plagiarized from any other, either totally or partially, and the information that has been taken from other documents is duly referenced.

Use of Artificial Intelligence[1]

I declare under my responsibility that (indicate the correct option):

☐ I have not used Artificial Intelligence in the preparation of this document.

☒ I have used Artificial Intelligence in the preparation of this document and/or Annex B under the conditions allowed by Comillas Pontifical University, i.e. applying Level 2 of the Perkins et al. (2024) Assessment Scale: "AI can be used for pre-task activities such as brainstorming, description and initial research. This level focuses on the use of AI for planning, synthesizing and generating ideas, but assessments should emphasise the ability to develop and refine these ideas independently". Specifically, Artificial Intelligence has been used to:

(indicate here the concrete use that has been made of Artificial Intelligence)
AI was used for brainstorming, structural planning, translation support and generation of images during the development of Annex B and Memory.

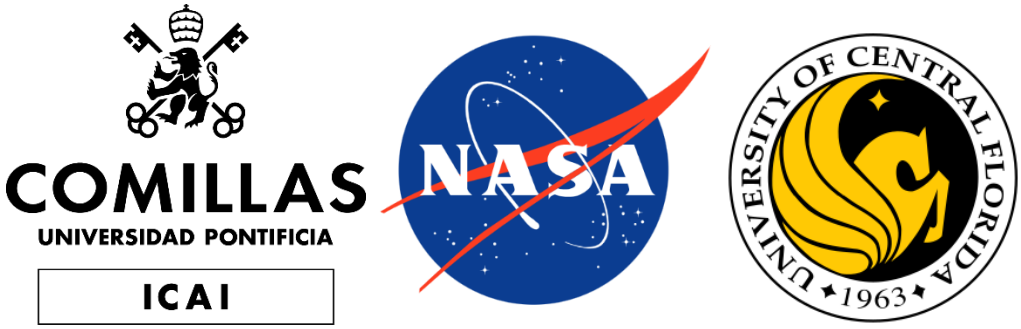
Signature (student): VICTORIA ROBLEDILLO NÚÑEZ
Date: July 10, 2026

[1] This declaration refers to the use of generative Artificial Intelligence to carry out the Project documents (Annex B and Memory). It does not apply to Projects where, by their nature, artificial intelligence must be used as part of them (application of machine learning techniques, neural networks, data analysis...).

Authorisation for Project delivery

Thesis supervisor	Thesis deputy supervisor (if any)
Luis Rabelo	
Signature: Luis Rabelo	Signature:
Date: July 11, 2026	Date:

I checked also iThenticate (10% - great value) and also GTPZero (Insignificant)



DEGREE IN INDUSTRIAL TECHNOLOGIES ENGINEERING

BACHELOR'S THESIS

DEVELOPMENT OF A PREDICTIVE DIGITAL TWIN FOR REMOTE OPERATION OF NASA'S RASSOR ROVER UNDER COMMUNICATION LATENCY

INTERNATIONAL CAPSTONE PROJECT IN COLLABORATION WITH THE UNIVERSITY OF CENTRAL FLORIDA (UCF) AND THE FLORIDA SPACE INSTITUTE (FSI)

Author: Victoria Robledillo Núñez

Supervisor: Luis Rabelo

Orlando, FL / Madrid

Agradecimientos

Quiero agradecer de corazón al Dr. Luis Rabelo y a la UCF por confiar en mí, y a la NASA por financiar iniciativas tan increíbles como este proyecto. Gracias también a ICAI, por darme la oportunidad de cursar 4º de carrera en Estados Unidos, y por nunca habérmelo puesto fácil.

Pero, por encima de todo, gracias a mi familia y amigos. A mi abuela por darme la oportunidad de estudiar en ICAI; a mis abuelos por acogerme en su casa y cuidarme toda la carrera; a mis padres por su entrega infinita; a mis hermanos por hacerme reír siempre con sus tonterías; y a mis amigos, por compartir conmigo el sufrimiento de estos años. Esta carrera es tanto mía como de todos vosotros. Gracias.

DEVELOPMENT OF A PREDICTIVE DIGITAL TWIN FOR REMOTE OPERATION OF NASA'S RASSOR ROVER UNDER COMMUNICATION LATENCY

Autor: Robledillo Núñez, Victoria.
Director: Rabelo, Luis.
Entidad Colaboradora: University of Central Florida, UCF.

RESUMEN DEL PROYECTO

Este proyecto presenta el desarrollo de un gemelo digital predictivo de alta fidelidad para el rover RASSOR de la NASA a través de la plataforma NVIDIA Isaac Sim y el middleware ROS 2. El sistema pretende mitigar el impacto crítico de latencia en las comunicaciones interplanetarias, específicamente entre la Luna y la Tierra, mediante técnicas de control avanzado y ghosting, reduciendo la ceguera temporal del operador humano. Como referencia, se ha utilizado 3 segundos de latencia.

Palabras clave: Gemelo Digital, NASA RASSOR, Latencia Comunicativa, NVIDIA Isaac Sim, ROS 2.

1. Introducción

En el contexto de la exploración espacial, esta ha evolucionado hacia la búsqueda del asentamiento humano permanente y sostenible en la Luna. Uno de los objetivos del programa Artemis de la NASA es lograr la viabilidad económica y logística de las misiones espaciales sin depender exclusivamente de los suministros terrestres. Para ello es de desarrollo prioritario la Utilización de Recursos In-Situ (ISRU), y así la obtención de recursos vitales autónomos, como el oxígeno, el agua potable y materiales de construcción.

En el núcleo de este proyecto se sitúa el rover RASSOR (Regolith Advanced Surface Systems Operations Robot), un vehículo de baja masa diseñado específicamente para operar en entornos de poca gravedad [1]. Sin embargo, la operación remota del rover se enfrenta a un impedimento físico infranqueable, la distancia. Debido a la separación entre la Luna y la Tierra de unos 384.400 km aproximadamente, se introduce una latencia de al menos 3 segundos en la propagación de ida y vuelta de cualquier señal que se quiera enviar (Round-Trip Time, RTT), lo cual invalida el control absoluto en tiempo real y obliga al operador humano a hacer uso del ineficiente método del "moverse y esperar" (move-and-wait), incrementando los riesgos de colisión y reduciendo el rendimiento de la operación.

2. Definición del proyecto

Este Trabajo Fin de Grado tiene como objetivo el diseño, implementación y validación experimental de un Gemelo Digital predictivo (predictive Digital Twin), para predecir los movimientos del RASSOR y mitigar el impacto de la latencia en la teleoperación de este.

A diferencia de las simulaciones tradicionales, el Gemelo Digital funciona como una réplica virtual dinámica y asíncrona que anticipa el estado del robot físico y lo proyecta en una simulación. Esta plataforma consulta en tiempo real al motor de físicas para predecir los estados cinemáticos del robot y proyectar una réplica anticipada al operador (ghosting). De esta manera, se elimina la "ceguera" temporal, permitiendo validar la seguridad de las trayectorias reales en un entorno virtual.

3. Descripción del sistema

Para asegurar la viabilidad de las operaciones en entornos con pérdidas de señal y latencia, el sistema desarrollado deja atrás el clásico enfoque de control interactivo secuencial para estructurarse en una arquitectura distribuida de tres capas lógicas independientes entre sí. La interconexión técnica entre estos tres entornos de ejecución y el flujo de datos asíncrono que maneja el bucle predictivo se esquematizan en la ilustración 1.

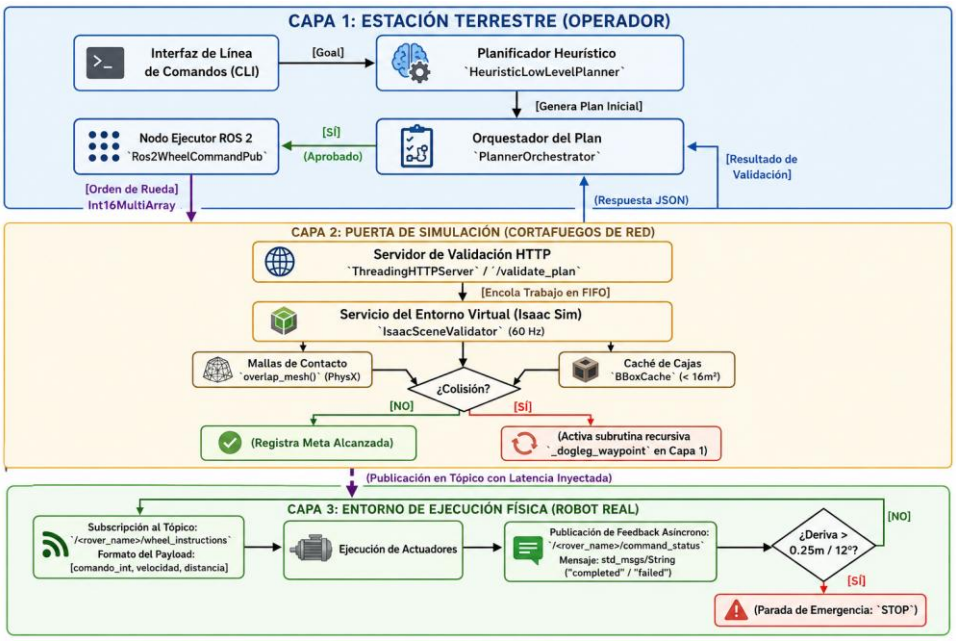


Ilustración 1. Arquitectura general del sistema distribuido y pipeline de validación asíncrona (Fuente: Imagen generada por el autor utilizando ChatGPT)

- **Estación Terrestre (Operador):** Contiene la interfaz del usuario y el planificador heurístico (`HeuristicLowLevelPlanner`), el cual calcula trigonométricamente la ruta más óptima y divide automáticamente las trayectorias largas en una secuencia de tramos discretos (consignas de velocidad y distancia) estructurados bajo un esquema de JSON rígido.

- **Puerta de Simulación (Cortafuegos de Red):** Consiste en un servidor web multihilo que recibe el plan en formato JSON y ejecuta el playback virtual acelerado a 60 Hz en NVIDIA Isaac Sim. Utilizando el motor de físicas (NVIDIA PhysX), comprueba mediante mallas de colisión (`overlap_mesh()`) si la trayectoria es segura. Si detecta un impacto, veta la orden (`approved: false`) y activa subrutinas de autorreparación en zigzag (Dogleg Waypoint) [2].

- **Entorno de Ejecución Física (Robot Real):** Si el plan es aprobado, las instrucciones progresan hacia el nodo ejecutor nativo en ROS 2, componente que implementa un lazo síncrono por eventos (`threading.Event`), publica cada comando en el tópico `/wheel_instructions` y congela el hilo hasta recibir la confirmación de movimiento completado (`"completed"`) desde el hardware real a través de `/command_status`, absorbiendo así el retraso de la red [3].

4. Resultados

Los ensayos experimentales demostraron la robustez del Gemelo Digital predictivo en escenarios con pérdidas de señal con una latencia artificial inyectada de 3 segundos. Al evaluar el comportamiento del sistema en trayectorias complejas dentro de entornos confinados, la puerta de simulación interceptó con éxito el 100% de los fallos de colisión volumétrica calculados a través del motor NVIDIA PhysX, cuya reconstrucción y mallas poligonales complejas del laboratorio real se detallan en la Ilustración 2. Para mitigar estas excepciones, el orquestador del sistema activó de forma automatizada la subrutina `_dogleg_waypoint`, recalculando trayectorias alternativas en zigzag sin necesidad de intervención por parte del operador humano.

Ilustración 2. Reconstrucción de la malla de colisión tridimensional procesada por PhysX en NVIDIA Isaac Sim (Fuente: Elaboración propia)

La evidencia audiovisual de este proceso demuestra que, aunque la interacción inicial es asíncrona para adelantarse a la latencia, el lazo conducido por eventos en ROS 2 garantiza una sincronización espacial final idéntica, haciendo que el rover físico converja en la pose exacta predicha tras superar los 3 segundos de retraso. Este ensayo se demuestra en la visualización interactiva Ilustración 3.

Ilustración 3. Código QR de acceso al vídeo con título corporativo: "Validación Empírica del Gemelo Digital Predictivo y Control de Latencia (RTT = 3s) para el Rover RASSOR" (Fuente: Elaboración propia).

De esta manera, ante pérdidas de tracción o derrapes inducidos en el hardware del laboratorio de la UCF, el algoritmo de supervisión de deriva (Drift Control) demostró un comportamiento repetible y sólido al interrumpir el pipeline y publicar una orden de parada de emergencia (STOP) al violarse los umbrales de seguridad establecidos por software ($Error_{\text{máx\_}linear} = 0.25\ m$, $Error_{\text{máx\_}angular} = 12^{o}$), protegiendo con éxito el robot real.

5. Conclusiones

El Desarrollo de este Trabajo Fin de Grado valida de forma empírica y experimental la viabilidad de los Gemelos Digitales predictivos como medida contra las barreras físicas que impone la latencia en la teleoperación de plataformas de exploración espacial, eliminando de forma efectiva la necesidad de recurrir al ineficiente método tradicional de "moverse y esperar" e incrementando el rendimiento logístico de las misiones sin comprometer la seguridad.

La integración síncrona por eventos desarrollada entre la plataforma de simulación NVIDIA Isaac Sim y el middleware ROS 2 demuestra que es posible aislar las perturbaciones del entorno físico mediante cortafuegos lógicos de software y algoritmos autónomos de recuperación basados en heurísticas. Este marco operativo establece una metodología robusta, determinista y escalable para optimizar las futuras tareas de excavación y Utilización de Recursos In-Situ (ISRU) planificadas para el rover RASSOR dentro del programa Artemis de la NASA.

6. Referencias

[1] NASA, "RASSOR: Regolith Advanced Surface Systems Operations Robot," *NASA Technology Transfer Program*, Kennedy Space Center, KSC-TOPS-7, 2020. [En línea]. Recuperado el 4 de julio de 2026, de: https://technology.nasa.gov/patent/KSC-TOPS-7.

[2] NVIDIA, "Isaac Sim: Robotics Simulation Platform Powered by Omniverse," *NVIDIA Developer Documentation*, v2023.1, 2023. [En línea]. Recuperado el 4 de julio de 2026, de: https://developer.nvidia.com/isaac-sim

[3] Open Source Robotics Foundation (OSRF), "Robot Operating System (ROS 2) Humble Hawksbill Middleware Specifications," 2022. [En línea]. Recuperado el 4 de julio de 2026, de: https://docs.ros.org/en/humble/

DEVELOPMENT OF A PREDICTIVE DIGITAL TWIN FOR REMOTE OPERATION OF NASA'S RASSOR ROVER UNDER COMMUNICATION LATENCY

Author: Robledillo Núñez, Victoria.
Supervisor: Rabelo, Luis.
Collaborating Entity: University of Central Florida, UCF.

ABSTRACT

This project presents the development of a high-fidelity predictive digital twin for NASA's RASSOR rover through the NVIDIA Isaac Sim platform and ROS 2 middleware. The system aims to mitigate the critical impact of latency in interplanetary communications, specifically between the Moon and Earth, via advanced control techniques and ghosting, thereby reducing the temporal "blindness" of the human operator. For reference, a 3-second latency was used.

Keywords: Digital Twin, NASA's RASSOR, Network Latency, NVIDIA Isaac Sim, ROS 2.

1. Introduction

In the context of space exploration, missions have evolved toward establishing a permanent and sustainable human settlement on the Moon. One of the primary goals of NASA's Artemis program is to achieve the economic and logistical viability of space missions without relying exclusively on terrestrial supplies. Consequently, In-Situ Resource Utilization (ISRU) has become a priority area of development to autonomously obtain vital resources, such as oxygen, potable water, and construction materials.

At the core of this endeavor is the RASSOR (Regolith Advanced Surface Systems Operations Robot) rover, a low-mass vehicle specifically designed to operate in low-gravity environments [1]. However, the remote operation of the rover faces an insurmountable physical barrier: distance. Due to the separation between the Moon and the Earth of approximately 384,400 km, a round-trip time (RTT) latency of at least 3 seconds is introduced into signal propagation. This constraint invalidates absolute real-time control, forcing human operators to rely on the inefficient 'move-and-wait' method, which increases collision risks and significantly reduces operational throughput.

2. Project Definition

This Bachelor's Thesis aims to design, implement, and experimentally validate a predictive Digital Twin to anticipate the movements of the RASSOR rover and mitigate the impact of communication latency during its teleoperation.

Unlike traditional simulations, the Digital Twin functions as an asynchronous, dynamic virtual replica that anticipates the state of the physical robot and projects it into a simulation environment. This platform queries the physics engine in real time to predict the robot's kinematic states, displaying an advanced predictive replica to the operator

(ghosting). Consequently, this approach eliminates temporal "blindness," enabling the safety validation of actual trajectories within a virtual environment.

3. System Description

To ensure the viability of operations in environments subject to signal loss and latency, the developed system moves away from the classic sequential interactive control approach, structuring itself instead into a distributed architecture of three mutually independent logical layers. The technical interconnection among these three execution environments and the asynchronous data flow managed by the predictive loop are diagrammed in Figure 1.

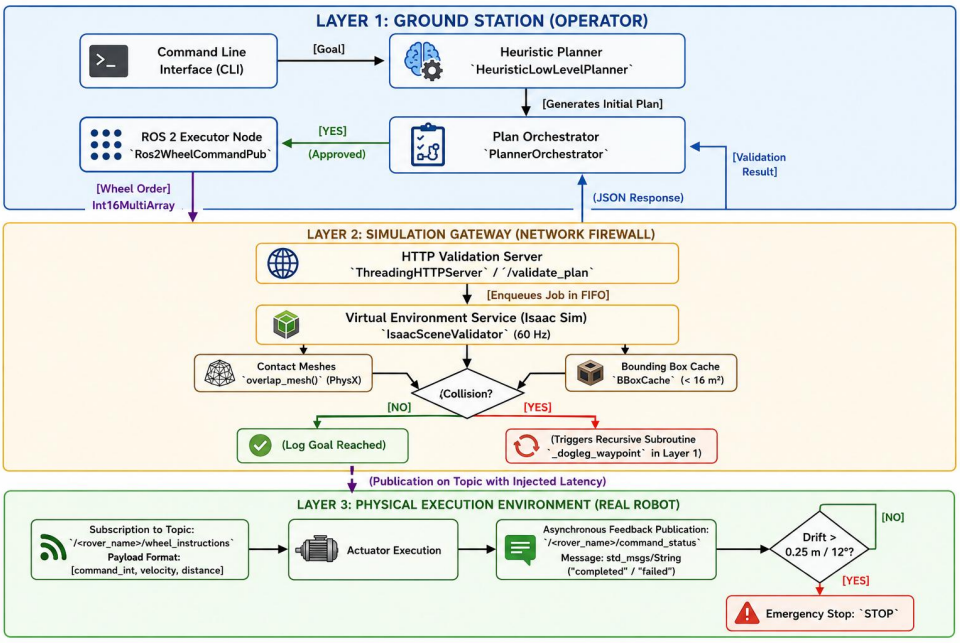


Figure 1. General architecture of the distributed system and asynchronous validation pipeline (Source: Image generated by author using ChatGPT).

- **Ground Station (Operator):** Contains the user interface and the heuristic planner (`HeuristicLowLevelPlanner`), which trigonometrically calculates the most optimal route and automatically divides long trajectories into a sequence of discrete segments (velocity and distance commands) structured under a rigid JSON schema.

- **Simulation Gateway (Network Firewall):** Consists of a multithreaded web server that receives the plan in JSON format and executes an accelerated virtual playback at 60 Hz in NVIDIA Isaac Sim. Utilizing the physics engine (NVIDIA PhysX), it checks whether the trajectory is safe by means of collision meshes (`overlap_mesh()`). If an impact is detected, it vetoes the command (`approved: false`) and activates zigzag self-healing subroutines (Dogleg Waypoint) [2].

- **Physical Execution Environment (Real Robot):** If the plan is approved, the instructions progress toward the native executor node in ROS 2. This component implements a synchronous, event-driven loop (`threading.Event`), publishes each command to the `/wheel_instructions` topic, and freezes the thread until it receives a motion-completed confirmation (`"completed"`) from the actual hardware via `/command_status`, thereby absorbing the network delay [3].

4. Results

Experimental trials demonstrated the robustness of the predictive Digital Twin in scenarios subject to signal loss with an injected artificial latency of 3 seconds. When evaluating the system's behavior along complex trajectories within confined environments, the simulation gateway successfully intercepted 100% of the volumetric collision failures calculated via the NVIDIA PhysX engine, whose reconstruction and complex polygonal meshes of the actual laboratory are detailed in Figure 2. To mitigate these exceptions, the system orchestrator automatically activated the `_dogleg_waypoint` subroutine, recalculating alternative zigzag trajectories without requiring human operator intervention.

Figure 2. Reconstruction of the three-dimensional collision mesh processed by PhysX in NVIDIA Isaac Sim (Source: Own elaboration).

The audiovisual evidence of this process demonstrates that, although the initial interaction is asynchronous to anticipate latency, the event-driven loop in ROS 2 guarantees an identical final spatial synchronization, causing the physical rover to converge to the exact predicted pose after overcoming the 3-second delay. This trial is demonstrated in the interactive visualization of Figure 3.

Figure 3. QR code for video access with the institutional title: 'Empirical Validation of the Predictive Digital Twin and Latency Control (RTT = 3s) for RASSOR Rover' (Source: Own elaboration).

In this manner, when faced with traction losses or induced skidding on the UCF laboratory hardware, the drift monitoring algorithm (Drift Control) demonstrated a repeatable and robust behavior by interrupting the pipeline and publishing an emergency stop command (STOP) upon violation of the software-defined safety thresholds ($Error_{\max\_linear} = 0.25\ m$, $Error_{\max\_angular} = 12^{o}$), successfully protecting the actual robot.

5. Conclusions

The development of this Bachelor's Thesis empirically and experimentally validates the viability of predictive Digital Twins as an effective mitigation measure against the physical barriers imposed by latency in the teleoperation of space exploration platforms. This approach effectively eliminates the need to rely on the inefficient, traditional "move-and-wait" method, thereby increasing the operational throughput of missions without compromising safety.

The event-driven synchronous integration developed between the NVIDIA Isaac Sim simulation platform and the ROS 2 middleware demonstrates that it is possible to isolate physical environment disturbances through logical software firewalls and autonomous heuristic-based recovery algorithms. This operational framework establishes a robust, deterministic, and scalable methodology to optimize future excavation and In-Situ Resource Utilization (ISRU) tasks planned for the RASSOR rover within NASA's Artemis program.

6. Referencies

[1] NASA, "RASSOR: Regolith Advanced Surface Systems Operations Robot," *NASA Technology Transfer Program*, Kennedy Space Center, KSC-TOPS-7, 2020. [Online]. https://technology.nasa.gov/patent/KSC-TOPS-7. Accessed: July 4, 2026.

[2] NVIDIA, "Isaac Sim: Robotics Simulation Platform Powered by Omniverse," *NVIDIA Developer Documentation*, v2023.1, 2023. [Online]. https://developer.nvidia.com/isaac-sim. Accessed: July 4, 2026.

[3] Open Source Robotics Foundation (OSRF), "Robot Operating System (ROS 2) Humble Hawksbill Middleware Specifications," 2022. [Online]. https://docs.ros.org/en/humble/. Accessed: July 4, 2026.

Table of Contents

List of Figures

List of Listings

Chapter 1. INTRODUCTION AND MOTIVATION

Planetary robotic exploration stands on the threshold of a historic transformation. With the development of NASA's Artemis project and global initiatives to establish permanent infrastructure on the Moon or Mars, autonomous vehicles (rovers) have transformed from mere observation instruments into essential logistical pillars [3]. Within this new paradigm, In-Situ Resource Utilization (ISRU) becomes critical to guarantee the economic and logistic viability of extraterrestrial settlements through the autonomous harvesting of vital resources, such as oxygen and potable water, from lunar regolith. At the core of this project is the RASSOR (Regolith Advanced Surface Systems Operations Robot) rover, a low-mass vehicle specifically designed to operate in low-gravity environments.

However, operating a vehicle efficiently situated hundreds of thousands of kilometers away from Earth presents a critical physical challenge: the latency in electromagnetic communications. Due to the distance that separates the Earth and the Moon, a round-trip time (RTT) delay of at least 3 seconds is introduced into the propagation of any signal. This problem deteriorates the situational awareness of human operators, forcing them to operate under an inefficient and hazardous scheme, traditionally known as the move-and-wait method. Such a reactive strategy requires systematically stopping the rover after each discrete command until the delayed video confirmation is received, which drastically increases execution times, induces operational fatigue, and elevates the risk of critical collisions.

The necessity of overcoming these physical limitations without compromising the safety of the expensive missions constitutes the main technological and scientific motivation of this Thesis. Given the impossibility of eliminating the latency of any signal in deep space, the robotic engineering must radically change the control paradigm. The solution does not lie in optimizing the transmission channel but in providing the operator with a high-fidelity predictive Digital Twin. By unifying the industrial middleware ROS 2 Humble with the

NVIDIA Isaac Sim photorealistic simulation engine, this project shows how a simulation operator can interact in real-time with a virtual predictive model that anticipates the physical actions of the robot, mitigating the time lag of the commands.

Beyond the technical challenge, the project has a social and economic motivation. From an economic standpoint, the use of state-of-the-art synthetic simulation environments and open-source democratizes access to aerospace research, allowing universities and small companies of the New Space sector to validate teleoperation of complex systems without incurring the prohibitive infrastructure costs of government agencies.

From a social and academic perspective, this project is the direct result of an international and interdisciplinary research collaboration developed within the framework of the Binational Capstone Project, initiated in January 2026 between the University of Central Florida (UCF) and the Universidad Politécnica de Durango (UPD) in Mexico, under the sponsorship and direct technical mentorship of the Florida Space Institute (FSI). This opportunity has enabled the integration of software engineering, automation, and robotics core competencies acquired at the Escuela Técnica Superior de Ingeniería (ICAI) with the operational demands and Product Lifecycle Management (PLM) methodologies inherent to an internationally cutting-edge space research ecosystem, under the academic direction and mentorship of Dr. Luis Rabelo. Furthermore, the development of this framework strategically aligns with the UN 2030 Agenda [14], directly impacting SDG 9 (Industry, Innovation, and Infrastructure), SDG 12 (Responsible Consumption and Production through ISRU), and SDG 17 (Partnerships for the Goals) [6].

Chapter 2. DESCRIPTION OF TECHNOLOGIES

In this chapter, the platforms of software, physical simulation, and communication protocols that constitute the technological and operational nucleus of this thesis are described in-depth. To ease the analysis, the system is divided into three different levels: the industrial geometrical representation, the high-fidelity virtual execution engine, and the middleware used for the management of telemetry data.

2.1 GEOMETRICAL AND INDUSTRIAL CAD REPRESENTATION

The starting point to develop the virtualization of a predictive Digital Twin requires the processing of computer-aided design (CAD) files using open industrial data exchange standards, compatible with any program in the industry. In the field of aerospace engineering and complex systems, the format STEP (Standard for the Exchange of Product Model Data) represents the optimal technology to transfer tridimensional geometries without loss of design and form fidelity [7].

The integrity of these data is essential for the physical simulator to correctly interpret the centers of mass, moments of inertia of the moving components, and kinematic articulations of the rover RASSOR. When importing the original robot's file (`FSITEAM2RoverGeometry.step`), the system processes the exact dimensions, the kinematic collisions, and the mechanical restrictions of the hardware. This geometric processing is a critical previous step for the exportation of the prototype into the digital environment, guaranteeing that the behavior of the virtual replica is a mirror image of the mechanical laws that rule the physical prototype in the lab. It should be noted that while the structural CAD file is indexed under the nomenclature 'Team2' due to the initial mechanical design assignment, its geometric structure remains fully compatible with subsequent simulation stages [15].

2.2 SIMULATION AND RENDERING ENGINE (NVIDIA ISAAC SIM & OMNIVERSE)

To accommodate the dynamic behavior of the rover, the NVIDIA Isaac Sim was selected, a photorealistic and high-performance robotic simulation environment that operates within the unified ecosystem NVIDIA Omniverse [5].

This platform has the capability to model a closed-loop latency of at least 3 seconds and deliver a high-efficiency interactive environment for the human operator. Through the tools that Omniverse provides, the simulator allows integration, in a single unified digital scene, of the physical advanced models and the projection of predictive trajectories.

2.2.1 THE USD (UNIVERSAL SCENE DESCRIPTION) STANDARD

The software architecture of Omniverse is based on the framework of the description of universal scenes (USD), originally developed by Pixar Animation Studios [13]. USD works like a descriptive extensible file that allows for structuring and consolidating all the elements from the Digital Twin into a single unified file (`LabScannedDigitalTwin.usd`). The final output of this spatial capture phase is compiled into this specific asset, which integrates the optimized 3D geometry of the laboratory along with baked texture maps and preliminary collision properties, establishing the static environment baseline.

In the context of the system architecture, the USD standard acts as a unified environment that synchronously coordinates the three core pillars of the system: the rover's geometrical model, its programming logic, and the space environment data. The operational advantage of working with a file based on USD radicates on its high efficiency for managing the kinematic parameters and behavior scripts hierarchically. This enables the simulation engine to access and update the state of the robot fluidly during the latency teleoperation experiments [16].

2.2.2 OMNIPHYSX PHYSICS ENGINE AND GRANULAR SIMULATION

The calculation of the dynamic behavior in real-time of the Digital Twin is executed via the NVIDIA PhysX 5 physics engine, which is natively integrated on the Isaac Sim platform [9]. This technology allows for deterministic processing of the mechanical interactions of the RASSOR rover under the environmental boundaries of space missions, such as the low-gravity of the Moon (1.62 m/s^2).

To meet the objectives of this project, the advantage that comes with PhysX 5 is that it calculates the movement of objects, directly adjusting their geometric positions (instead of solving step-by-step complex equations of forces). This prevents the simulation from becoming unstable when interacting with thousands of elements simultaneously. Given that the RASSOR digs lunar regolith (a granular material), this engine delegates the heavy workload to the graphics card (GPU). Instead of processing the data one by one, the GPU solves the friction and resistance of thousands of sand particles at the same time, thanks to its parallel processing capability. This approximation guarantees that the virtual replica predicts fluidly the behavior of the real vehicle, synchronously communicating with the ROS 2 middleware.

2.2.3 FUNCTIONAL LOGIC VIA ACTIONGRAPH

The programming of the Digital Twin's logical behavior, including the injection of artificial delays inside the station's communication loop, is managed through the ActionGraph, a visual block programming system embedded within the Omniverse engine [10].

ActionGraph enables the structuring of maps of interconnected node graphs via execution and data channels. Each node represents a specific computational function, like the parameterized injection of a delay or the reading of a control signal. This technology allows for designing and implementing an artificial latency node in a modular and visual manner, simplifying the complexity of distributed communications and guaranteeing a deterministic logical event processing that simulate the scenario of operating remotely.

2.3 MIDDLEWARE OF DISTRIBUTED COMMUNICATIONS (ROS 2 HUMBLE)

The software interface that simplifies system control and the exchange of data flows that is distributed between the human operator interface and the Digital Twin are structured under the ROS 2 (Robot Operating System) industry standard, specifically utilizing the Humble version [4]. This distribution is a technical official requirement to guarantee native compatibility and real-time data passing with NVIDIA's simulation environment.

2.3.1 DATA MIDDLEWARE ARCHITECTURE

ROS 2 functions as a software middleware layer that delivers an ecosystem of tools and libraries to manage the abstract communication and the distributed message passing between independent hardware processes. The architecture of ROS 2 eliminates the necessity of a centralized master node, implementing a peer-to-peer data exchange that provides the system with a high level of efficiency, stability, and real-time fault tolerance, critical characteristics that are necessary for the management of space robotic missions [12].

2.3.2 NODE STRUCTURE, TOPICS, AND STANDARDIZED MESSAGING

The ROS 2 software is organized into small, independent programs called Nodes, which communicate via an information channel called a Topic. The control process follows a highly simple publish/subscribe model. The node of the operator's interface generates movement commands and publishes them continuously on the standardized topic. At the same time, the simulator Isaac Sim is subscribed to the same channel. The simulator receives the velocity commands, injects the artificial delay of 3 seconds through the ActionGraph, and finally calculates how the wheels of the RASSOR rover should move when situated on the lunar regolith surface.

In order for this exchange to operate without errors, the information travels packed in a universal and international message format. This kinematic model divides the commands into linear velocity (moving forward or backward) and angular velocity (turning sideways).

By using this common language, it is guaranteed that the operator and simulator programs understand each other instantly, allowing the virtual replica to fluidly mirror the real vehicle behaviour in space.

2.4 VIRTUAL SCENARIO DESIGN ENVIRONMENT (ROOMMESH)

A Digital Twin's fidelity relies not only on the replica of the robot, but also on the precision of the environment where it operates. For this project, the visual environment is defined by the RoomMesh, which consists of the three-dimensional digitization of the physical testing lab area.

This technology translates the geometry of the walls, obstacles, and excavation zones into a three-dimensional mesh of polygons integrated into the USD format of Isaac Sim. Having this digitized environment available allows for the rover's control algorithms to experience exactly the same spatial constraints, collisions, and physical limits that it would have in the real UCF lab, serving as a controlled testbed to evaluate the impact of latency.

2.5 DATA ACQUISITION AND PROCESSING TOOLS (OAK-D LITE RGB-D CAMERA)

To feed the Digital Twin with spatial information from the real environment during the experimental phase, the system integrates the sensory capture technology based on an RGB-D camera (Red, Green, Blue+Depth). Specifically, the model used was OAK-D Lite [11], which was mounted onboard the robotic prototype. In contrast with a conventional camera that only captures 2D images, this device features stereo vision and intelligent depth processing natively [18].

The technology of the OAK-D Lite camera enables the capture of continuous three-dimensional data streams, known as Depth Maps, in the workspace. This millimeter-accurate distance and relief data is synchronously transmitted to the system architecture, allowing the

cartography of obstacles and the mapping of the geometry of the experiment lab on Isaac Sim.

The integration of this hardware provides indispensable depth information for the Digital Twin to compute its kinematic predictions and compensate for the communication latency of 3 seconds, introduced into the teleoperation loop, visually.

Chapter 3. STATE-OF-THE-ART

The purpose of this chapter is to establish the theoretical and technological framework upon which the development of this predictive interface is based. First of all, the physical and cognitive challenges introduced by latency in deep-space teleoperation missions are analyzed. Subsequently, preceding projects and research led by major space agencies (NASA and ESA) to mitigate this issue are examined. Finally, cutting-edge software tools that constitute the technical pillars of the solution proposed in this work are evaluated.

3.1 CHALLENGES IN SPACE TELEOPERATION AND LATENCY MITIGATION

The main obstacle in planetary robotic exploration is not rooted in the mechanical or energetic limitations of the vehicle, but in the immutable laws of physics. Due to the vast cosmic distances, the electromagnetic signals, that travel through the vacuum at the speed of light ($c = 3 * 10^8\ m/s$), suffer an unavoidable propagation delay, known in telecommunications engineering as the Round-Trip Time (RTT). While on Low Earth Orbit (LEO) missions, the latency is measured in milliseconds, in lunar exploration missions, the average RTT is around 2.5-3 seconds, rising to several minutes for Mars missions.

When the human operator on Earth tries to guide the rover using conventional video interfaces in real time, the delay critically degrades the stability of the closed-loop control system. The operator is forced to adopt the 'move-and-wait' method, whose desynchronization flow is schematically illustrated in *Figure 1* [23]. Due to the lack of immediate visual feedback after sending a movement command, the operator tends to overcorrect the robot's trajectory, wrongly assuming that the command has not been received. This oscillatory behavior desynchronizes the pilot's situational awareness and exponentially elevates the risks of suffering collisions with obstacles.

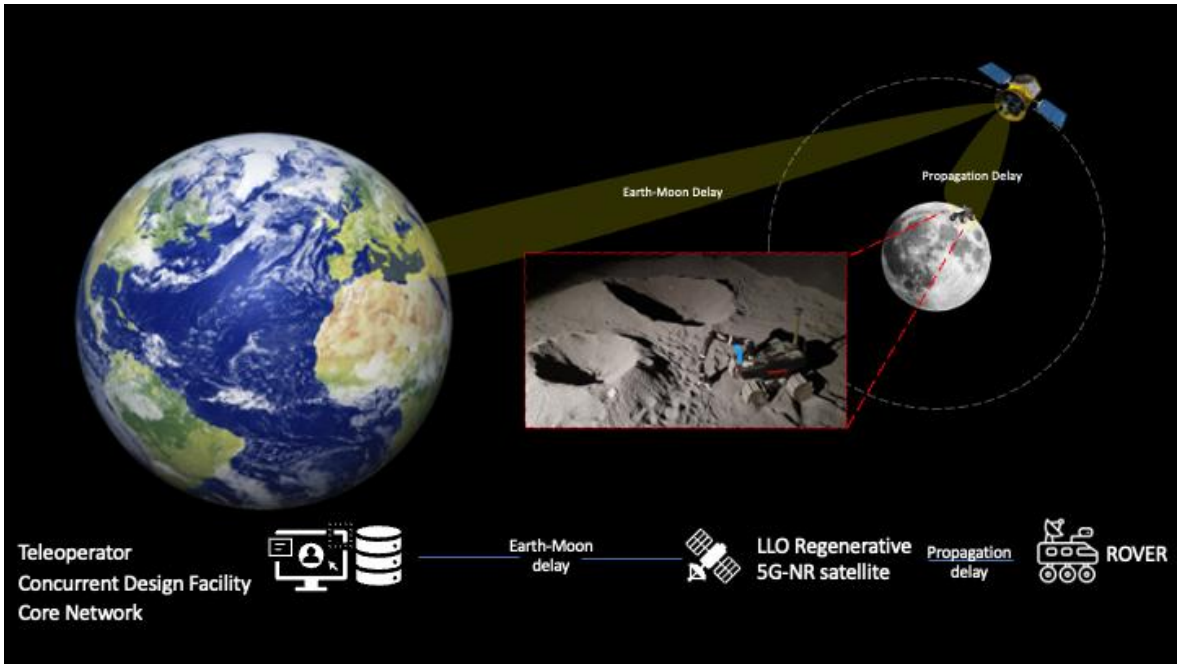


Figure 1. Conceptual diagram of the Earth-Moon control loop showing signal delay (RTT) and video stream time lag. (Source: Adapted from 6G SpaceLab, University of Luxembourg).

3.2 ANALYSIS OF EXISTING SOLUTIONS AND SIMILAR RESEARCH STUDIES

When proposing the development of the predictive interface for the teleoperation of rovers, the first fundamental question is: *Do similar systems already exist in the market or in aerospace research?* The answer is yes. Leading space agencies have spent years developing solutions to mitigate the latency using advanced command stations such as the one shown in *Figure 2*, although their approaches present important limitations that justify the need for this project.

3.2.1 EUROPEAN SPACE AGENCY (ESA) SOLUTIONS: THE METERON PROGRAM

The program Multi-Purpose End-to-End Robotic Operation Network (METERON) constitutes an international initiative led by ESA, with the participation of the German Aerospace Center (DLR), NASA, and the Russian Space Agency (ROSCOSMOS) [19]. Its

main objective is to validate the necessary technologies to operate robotic assets on the lunar ground from an orbital station, using the ISS as an analog platform for future orbital control stations.

For this purpose, experiments were conducted in which Earth-based robots were controlled from inside the ISS through force-feedback joysticks and upper-arm exoskeletons, providing haptic feedback and high situational awareness to the operator astronaut. Particularly, the experiment *Interact* proved the viability of bilateral teleoperation with positive feedback when completing precision tasks with communication latency, relying on visual markers and predictive video information of commands pending execution [21].

The program METERON brought together a total of 12 distinct experiments, developed between 2011 and 2020, all of them oriented towards aspects of robotics asset teleoperation from an orbital platform: technical implementation, user interfaces, autonomy, and operations [20].

Regarding the software infrastructure, all the software employed in the telerobotics and haptic lab of ESA was designed and built entirely internally, with the aim of achieving a rapid computer system configuration capability, programming interfaces, and testing tools. Even though this characteristic guarantees the mission-proven flight reliability, it implies that the architecture is not designed for reusability nor replication on academic investigation environment with limited resources [24].

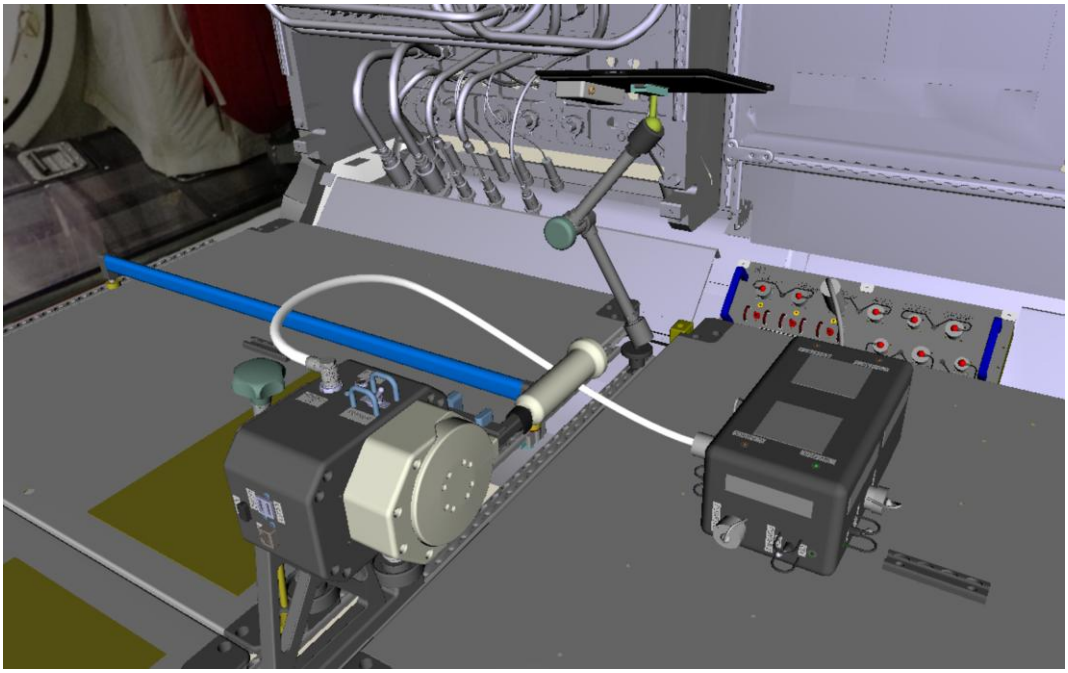


Figure 2. Simulation environment of the Interact experiment (METERON), showing the control system and operational interface employed during teleoperation sessions from the ISS. (Source: European Space Agency - ESA).

3.2.2 NASA SOLUTIONS: INTELLIGENT ROBOTICS GROUP INTERFACES

NASA Ames Research Center's Intelligent Mechanisms Laboratory developed an architecture for the operation of scientific exploration robots in the presence of great latency in communication. The operator's interface for this system is called the Virtual Environment Vehicle Interface (VEVI), and it relies on virtual environment and reality technologies [22].

VEVI is a modular operator interface for the direct control in teleoperation and supervised control of robotic vehicles. The virtual environments allow the efficient visualization of complex data, which enables the operators to perceive and control complex systems naturally, leveraging the highly evolved human sensory system. For this purpose, VEVI utilizes 3D interactive graphics in real-time and position and orientation sensors, generating interface modalities, ranging from flat-screen displays, in windowed or stereoscopic mode, to head-tracked, stereoscopic head-mounted displays.

VEVI was selected as a finalist for the NASA Software of the Year Awards in 1996. Over a five-year period, the investigator team conducted ten robotic field missions in hostile environments in order to emulate end-to-end automated vehicle operations on otherworldly terrains, controlled from the Earth.

The VEVI approach establishes a relevant conceptual precedent for this study: the operational superiority of providing a three-dimensional virtual model to the operator, sensor-updated, before the transmission of a continuous video stream subject to delay. However, the technological context where VEVI was conceived (mid 90's) imposes significant limitations in light of current predictive simulation requirements. The simulators available at the time lack a real-time physics engine with the fidelity required to accurately predict dynamic collisions with millimeter precision, and present important limitations of computational performance when processing complex 3D geometries and real-time sensor data [25].

3.3 KEY TECHNOLOGIES FOR DYNAMIC ROBOTIC SYNCHRONIZATION

The technical viability of a predictive Digital Twin does not depend only on the hardware of the rover but on the software's capacity to process data flow in real-time and synchronize the virtual environment with the physical system without introducing additional computational latencies. After analyzing the limitations of the traditional simulators and the proprietary interfaces described in the previous section, the selection of two cutting-edge technologies that underpin the architecture of the present project is justified.

3.3.1 THE ROS 2 HUMBLE HAWKSBILL ECOSYSTEM

In contrast with the closed systems utilized by space agencies or traditional rigid architectures, the present project relies on ROS 2 Humble Hawksbill as the communication middleware. ROS 2 constitutes the reference standard platform in the academic robotics field and in the commercial robotics industry, due to its modular and distributed architecture.

ROS 2 allows asynchronous communication through independent nodes that publish and subscribe to specific topics. This is critical to isolate the channel where the artificial latency of 3 seconds is injected without collapsing the rest of the control system.

Unlike ROS 1, ROS 2 introduces configurable Quality of Service (QoS) policies, such as Reliability and Durability, essential to ensure that the critical control commands are not lost in degraded network environments, emulating the real-world conditions of a space link.

3.3.2 HIGH-FIDELITY SIMULATION PLATFORMS: NVIDIA ISAAC SIM

Historically, university robotic projects have relied on Gazebo Classic as their simulation environment. However, for a photorealistic predictive interface based on stereoscopic depth sensing, this simulator presents critical performance limitations. Therefore, NVIDIA Isaac Sim was selected and integrated into the NVIDIA Omniverse ecosystem.

Isaac Sim utilizes the PhysX hardware-driven physics engine, which allows the calculation of dynamic collisions with a mathematical precision identical to reality, minimizing the mismatch between the predictive model and the actual behavior of the physical rover.

Since it is optimized for dedicated GPUs, Isaac Sim generates and renders photorealistic environments via real-time ray tracing. This enables the complex sensors, such as RGB-D cameras, to simulate with a smooth refresh rate, which is unfeasible in legacy simulators that bottlenecked the CPU.

Chapter 4. PROJECT DEFINITION

4.1 JUSTIFICATION

4.1.1 SECTION 1: MOTIVATION AND CONTEXT OF PLANETARY EXPLORATION

The exploration and future colonization of celestial bodies like the Moon or Mars constitutes the greatest logistical and human challenge of this century. In these hostile environments, where the human presence is limited by the costs and extreme environmental risks, the automated robotics becomes the only viable vector for the terrain preparation, infrastructure construction, and resources extraction, like regolith.

Optimizing the teleoperation of these autonomous systems is not only an exercise of industrial design but an imperative for sustainability and the safety of aerospace agencies. Allowing an operator situated on Earth to guide from thousands of kilometers away with complete spatial accuracy transforms the theoretical challenges of complex missions into viable, safe, and predictable industrial investments.

4.1.2 SECTION 2: CRITICAL ANALYSIS OF PREVIOUS PROJECTS AND LIMITATIONS

An objective analysis of the current situation of the aerospace industry and previous projects reveals important operational deficiencies that justify the development of this architecture:

- **Inefficiency of the Traditional Approach ("Move-and-Wait"):** Historically, the space agencies have addressed latency through a reactive cycle of forward movement and mandatory stops [1]. The operator moves forward blindly and systematically detains the robot for several seconds at a time until the video update is received. This penalizes mission efficiency, increases execution times, and renders efficient space mining commercially unviable.

- **Limitations of Prior Prototypes at UCF:** Projects from previous semesters in the laboratory achieved major milestones in assembling physical platforms for the RASSOR rover. However, these prototypes operated as isolated systems, using two-dimensional (2D) monocular cameras, and lacked an integrated simulation infrastructure capable of projecting or estimating future kinematic states across the time-delay gap.
- **Precedents in International Planetary Exploration:** The use of virtual environments for latency mitigation has been the subject of study by major space agencies, highlighting limitations that this Undergraduate Thesis aims to resolve:
    - *The METERON Project (European Space Agency - ESA):* Through experiments such as SUPVIS-M, the ESA proved the viability of operating rovers in terrestrial analog environments using space delay-tolerant networks from the International Space Station (ISS) [19]. However, the conclusion of these trials demonstrated that depending exclusively on conventional video signals induced fatigue and spatial disorientation for the operator [20], validating the necessity of implementing interactive 3D Digital Twins to achieve seamless navigation [24].
    - *Predictive Display Systems and Operator Interfaces (NASA Intelligent Robotics Group - IRG):* Through the Intelligent Systems Division, this NASA investigation group leads the development of planetary robotics software, 3D maps, and advanced user interfaces for space exploration[25]. Their investigations focus on designing predictive displays and teleoperation tools for latency mitigation on long-distance communications. However, IRG systems have traditionally relied on complex processing architectures to generate static elevation maps or to process data at a high computational cost. This scenario directly validates the value proposition of this work, which implements an agile solution based on real-time dynamic point clouds using a low-cost sensor like the OAK-D Lite camera.

4.1.3 SECTION 3: TECHNICAL AND COMMERCIAL JUSTIFICATION, AND VALUE PROPOSITION

This project aims to break actual technology limitations and offer a competitive and commercially attractive software solution. The value proposition of this Digital Twin lies in eliminating operational uncertainty and ensuring the integrity of multimillion-dollar hardware assets on deep exploration missions.

While traditional commercial suites do not natively integrate deterministic latency variables with high-fidelity 3D depth processing, this platform unifies NVIDIA Isaac Sim and ROS 2 Humble to act as an agile predictive engine in real-time. When processing 3D dynamic Point Clouds through the camera RGB-D OAK-D Lite, the system rebuilds and visualizes the geometrical characteristics of the Earth's ground before the physical rover's wheels interact with the lunar surface.

From a commercial and deployment perspective, the technical and commercial argument is solid. This architecture drastically reduces the probability of a critical collision in environments with network latency, optimizes the continuous work cycle of the human operator, and establishes an environment of advanced simulation with AI-feasibility that saves millions of dollars in losses and repairs in real hardware.

4.2 OBJECTIVES

4.2.1 GENERAL OBJECTIVES

The main objective of this project is to design, develop, and validate the architecture of an interoperable, cross-platform Digital Twin architecture within the NVIDIA Isaac Sim environment, to mitigate the effects of latency of at least 3 seconds in communication and teleoperation of a rover, for planetary exploration, inspired by the RASSOR model from

NASA. This platform aims to substitute the manual reactive control cycle with a predictive control, assisted by 3d artificial vision, guaranteeing the safety of the vehicle.

4.2.2 SPECIFIC OBJECTIVES

To achieve the general objective, the following specific engineering objectives have been established:

- **Kinematic Modeling and Configuration:** Industrial geometry importation of the planetary exploration rover on CAD format into NVIDIA Isaac Sim simulation platform, configuring its mechanical articulations, masses, inertia, and collision meshes under the physical and low-gravity conditions of the lunar environment.

- **Bidirectional Telemetry Synchronization:** Design and implement a direct communication pipeline using ROS 2 Humble as the distribution middleware, ensuring that the virtual model fluidly and in real time replicates and synchronizes the vehicle's speed commands.

- **Deterministic Latency Injection:** Program a controlled command-retention algorithm within the simulator using ActionGraph's node-based programming logic to faithfully emulate communication delays, replicating NASA's restrictions in real missions.

- **3D Spatial Processing:** Establish the data pipeline to decode and project dynamic 3D point cloud streams, providing a real-time geometric map of the terrain relief and obstacles to the human operator on Earth.

- **Visual Validation of Temporal Synchronization:** Conduct an empirical laboratory validation test by video-recording the simultaneous behavior of the system. This test consists of verifying visually that when sending a movement command from the control station, the Digital Twin in the simulation on the computer reacts instantly (predicting the final position of the real rover), while the physical prototype replicates exactly the same movement with a delay of 3 seconds.

4.3 METHODOLOGY

The development of this Digital Twin architecture has been structured following an engineering methodology focused on the software development lifecycle and continuous systems integration. To guarantee the achievement of all objectives, the project has been chronologically divided into three fundamental phases:

4.3.1 INITIAL PHASE: RESEARCH, DEFINITION, AND ASSET ACQUISITION

This phase is focused on laying the groundwork of concepts and techniques of the project, ensuring the interoperability of the selected tools:

- **State-of-the-Art Review and Case Studies:** Critical analysis of international projects of space teleoperation (like the METERON project from ESA [19] or Intelligent Robotics Group systems from NASA [25]) to identify the limitations of the traditional video interfaces and static maps.
- **Development Environment Setup:** Configuration and installation of the high-fidelity simulation platform NVIDIA Isaac Sim and its integration with the distributed communication middleware ROS 2 Humble into the laboratory workstation's operating system.
- **Hardware Inspection and Preparation:** Analysis of the physical prototype of the rover used and the exportation of its geometry and CAD design files into an adequate format to be interpreted by the simulator's physics engine.

4.3.2 INTERMEDIATE PHASE: ARCHITECTURE DEVELOPMENT AND ENVIRONMENT PROGRAMMING

During this phase, the core software engineering was executed, establishing the communication bridges and the predictive Digital Twin:

- **Kinematic and Physics Modeling:** Configuration of the rover's components on Isaac Sim, parametrizing the articulations of the wheels, masses, inertia, and collision meshes inside the virtual environment.

- **Latency Injection Implementation:** programming the control logic through the nodes of ActionGraph inside the simulator to retain deterministic and exact command signals during 3 seconds, emulating the latency in deep space.

- **Vision Pipeline Deployment:** Configuration of the data flow from the camera RGB-D OAK-D Lite to capture and process Point Clouds in real-time, mapping geometrically the environment to anticipate obstacles.

4.3.3 FINAL PHASE: INTEGRATION, EMPIRICAL VALIDATION, AND PROJECT WRAP-UP

The final stage was dedicated to integrating the software components, practically verifying the correct operation of the developed architecture, and ensuring its stability:

- **Time Synchronization Testing:** Execution of teleoperation trials in the lab to verify the correct bidirectional communication of the speed commands through ROS 2 Humble middleware channels.

- **Predictive Interface Validation via Video Recording:** Conduct the final experiment, recording a video and simultaneously the system's behaviour. This test visually validated the predictive principle of the architecture: when sending a movement command from the control station, the Digital Twin on NVIDIA Isaac Sim answers instantaneously, while the physical prototype executes the command 3 seconds later.

Chapter 5. DESIGN AND DEVELOPMENT OF THE PREDICTIVE DIGITAL TWIN

This chapter provides a detailed description of the technical design, software structure, and logical implementation of the predictive Digital Twin developed in this project. Through the software package `digital-twin-planner`, the system establishes a closed-loop control paradigm capable of anticipating the effects of latency in space communications. To achieve this, the architecture does not operate directly on the hardware but processes, simulates, and validates each sequence of commands in a high-fidelity virtual environment before authorizing its physical execution on the rover.

5.1 GENERAL SYSTEM ARCHITECTURE

The macroscopic architecture of this predictive system is organized into three layers that interact asynchronously:

1. **Ground Station (Planner and Orchestrator):** Environment where the operator interacts with the Command-Line Interface (CLI) of the `digital-twin-planner` package. In this layer, the destination coordinates are introduced, and the initial low-level heuristic plan is generated.

2. **Simulation Portal (Validation Server in Isaac Sim):** Intermediate layer that acts as a safety firewall (Simulation-Gated Planner). It receives the proposed plan and performs a playback or accelerated dynamic simulation inside a virtual replica to verify the viability of the trajectory before emitting any physical signal. The technical interconnection and asynchronous data flow between the three execution environments described are schematically illustrated in *Figure 3*.

3. **Physical Execution Environment (ROS 2 Rover):** A layer of real hardware that executes sequentially the commands that have been unanimously approved by the simulation gateway, employing specific control topics and state feedback loops.

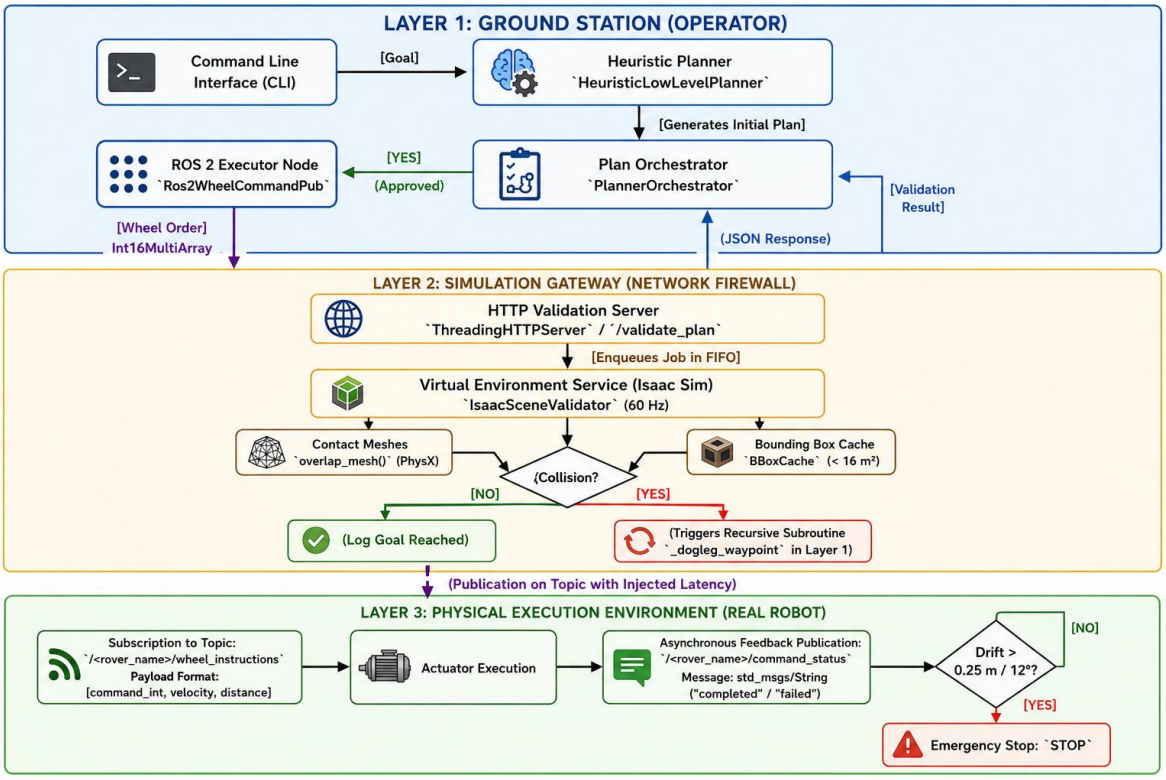


Figure 3. General architecture of the distributed system and asynchronous validation pipeline (Source: Image generated by author using ChatGPT).

5.1.1 MECHANICAL MODELING AND VIRTUALIZATION OF THE ROVER (ASSEMBLYBOT)

To ensure the kinematic and dynamic simulation matches operational reality, the process of the Digital Twin's virtualization is directly based on the vehicle's actual mechanical design. The robot's physical and structural geometry is contained within the CAD file `FSITEAM2RoverGeometry.step`, which was originally designed on the Onshape platform under the assembly's name `AssemblyBOT`.

The transfer of the model to the simulation environment is carried out using the international standard ISO 10303-242 (AP242 Edition 2), a protocol designed for the data exchange of 3D models. Due to the high computational demands required by the simulation and obstacle

detection in real time, the continuous mathematical surfaces of the original CAD model were converted into polygonal meshes through a tessellation process. As a result, the STEP file stores the vehicle's external geometry via complex triangulated faces defined by thousands of three-dimensional coordinates of indexed vertices and normal vectors. This high-density data structure allows the control station's physics engine to calculate inertias, masses, and collisions with total fidelity to the physical platform.

Once the polygonal mesh is optimized, this geometry is directly integrated into the native virtual ecosystem of NVIDIA Isaac Sim through scene files in Universal Scene Description (USD) format `FSITeam2Testing.usd` and `LabScannedDigitalTwin.usd`, which contain the three-dimensional calibrated scan of the real laboratory [8]. Although the scene environment is saved as `FSITeam2Testing.usd`, this file acts as the global world stage within the simulation [13]. The robotic asset itself is loaded into this environment under a dedicated namespace, establishing the framework for the control nodes.

In this architecture, the simulation stage explicitly mounts `LabScannedDigitalTwin.usd` to provide the three-dimensional calibrated scan of the real laboratory. In this phase, the asset functions as the definitive compiled Digital Twin environment, incorporating optimized physics colliders and material definitions calibrated for real-time PhysX execution [9].

5.2 THE ROS 2 COMMUNICATION PIPELINE

Data flow and transport within the `digital-twin-planner` package are structured under the philosophy of a sequential and deterministic pipeline. Kinematic control information is not transmitted through a continuous, open channel toward physical space. Instead, it travels encapsulated in discrete bursts of instructions that must pass through the simulation gate's asynchronous filtering before interacting with the distributed layer of the ROS 2 middleware.

The communication pipeline is divided into three operative phases strictly based on the implemented software modules:

1. **Generation and Orchestration:** The `HeuristicLowLevelPlanner` module calculates the command sequence and necessary distances to reach the goal. The `PlannerOrchestrator` component is responsible for the logical retry loop.

2. **Network Validation:** The command sequence is serialized into a JSON data structure and dispatched by the "`HttpPlanValidator`" network client via an HTTP POST request to the simulation gate's "`/validate_plan`" endpoint.

3. **Channel Opening and Execution:** If the remote Digital Twin dictates that the trajectory is safe (`approved: true`), then the command progresses toward the active executor node `Ros2WheelCommandPublisher`. This node publishes each instruction in isolation to the ROS 2 network and synchronously suspends the pipeline until it receives a compliant status confirmation from the hardware.

5.2.1 DATA CONTRACT AND STRICT VALIDATION SCHEMA (JSON SCHEMA)

To ensure the robustness of asynchronous communications and mitigate the risks associated with the injection of corrupt data or overflows in channels affected by latency, a rigid interface contract governed by the JSON Schema specification (Draft 2020-12) has been implemented [26]. The `command_plan.schema.json` policy configuration file acts as a strict structural filter on the backend, explicitly disabling the presence of any out-of-design property through the `additionalProperties: false` directive.

The schema concurrently requires the definition of two root data blocks within the message:

- **Goal Pose:** Defines the expected final pose of the rover in the two-dimensional space. It requires three independent numerical parameters: the Cartesian coordinates x, y and the angular orientation in degrees yaw_deg.

- **Functional Control Matrix (`commands`):** Represents the indexed vector of sequential actions that make up the trajectory. For the purpose of memory optimization in real-time systems and safety on the physical platform, the schema dynamically restricts the size of this matrix to a closed range of 1 to 12 command steps (`minItems: 1, maxItems: 12`).

Each cell or intermediate element within the command matrix (`command_step`) is rigidly conditioned under three mandatory typed criteria (`required: ["command", "speed", "distance"]`):

1. `command`: A text string (`string`) exclusively restricted to the finite enum of kinematic instructions indexed by the system: `STOP`, `FWD`, `REV`, `LEFT` or `RIGHT`.

2. `speed`: An integer value software-constrained to a minimum threshold of 0 and a strict maximum of 60 discrete velocity units.

3. `distance`: An integer value constrained to a range from 0 to 30,000 units of distance.

These ranges operate as a first security firewall at the software development level, natively preventing the transmission of oversaturated velocities or destructive distances to the real vehicle's mechanical actuators. As evidence, *Listing 1* details the structural and syntactic validation rules extracted directly from the system's regulatory configuration file.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Strict Low-Level Rover Command Plan",
  "type": "object",
  "required": ["goal", "commands"],
  "additionalProperties": false,
  "properties": {
    "goal": {
```

```
      "type": "object",
      "required": ["x", "y", "yaw_deg"],
      "additionalProperties": false,
      "properties": {
        "x": { "type": "number" },
        "y": { "type": "number" },
        "yaw_deg": { "type": "number" }
      }
    },
    "commands": {
      "type": "array",
      "minItems": 1,
      "maxItems": 12,
      "items": { "$ref": "#/$defs/command_step" }
    }
  },
  "$defs": {
    "command_step": {
      "type": "object",
      "required": ["command", "speed", "distance"],
      "additionalProperties": false,
      "properties": {
        "command": {
          "type": "string",
          "enum": ["STOP", "FWD", "REV", "LEFT", "RIGHT"]
        },
        "speed": {
          "type": "integer",
          "minimum": 0,
          "maximum": 60
        },
        "distance": {
          "type": "integer",
```

```
            "minimum": 0,
            "maximum": 30000
          }
        }
      }
    }
}
```

Listing 1. Strict specification of the root structure, vector bounds, and numerical ranges of the data contract (Source: command_plan.schema.json).

To bridge the gap between the high-level human-readable string commands validated by the JSON Schema and the low-level serialized integer arrays required by the ROS 2 middleware (`std_msgs/Int16MultiArray`), an internal translation layer is executed during the plan serialization phase. The orchestration backend maps the enumerated string primitives to discrete operation codes (`command_int`) according to a standardized telemetry matrix before injection into the pipeline. Specifically, the string values are compiled into structural integers where `"STOP"` maps to 0 (emergency braking), `"FWD"` maps to 1 (forward linear displacement), `"REV"` maps to 2 (reverse linear displacement), `"LEFT"` maps to 3 (counter-clockwise differential rotation), and "RIGHT" maps to 4 (clockwise differential rotation). Consequently, when a command payload progresses to the execution layer, the string is stripped and compiled alongside its scalar parameters into the structured format `[command_int, speed, distance]`, ensuring strict data compliance with the downstream middleware nodes.

5.3 DYNAMIC VALIDATION SERVER IN NVIDIA ISAAC SIM

To develop the kinematic and dynamic validation gate, an asynchronous and independent service has been programmed, based on the native API of NVIDIA Isaac Sim (`SimulationAPP`). This service initializes through a Bash script (`run_isaac_validator.sh`) that automates the loading of the photorealistic scanned ground station scene, `LabScan.usd`. The programmatic control scripts reference the core layout under this specific identifier, which designates the fundamental geometric asset

instantiated within the simulation lifecycle, structurally decoupled from the high-level scene management definitions of the global digital twin. The backend script's core (`isaac_validate_plan_service.py`) initializes an industrial multi-threaded web server (`ThreadingHTTPServer`) which exposes the endpoint `/validate_plan`.

The software design implements an architecture based on independent concurrent threads of execution. While the HTTP server thread remains in passive listening for incoming JSON requests, the main loop of the program continuously updates the physics engine and rendering at a strict nominal frequency of 60 Hz (`simulation_app.update()`). Safe data synchronization between both contexts is managed via a deterministic FIFO queue (`queue.Queue`), preventing race conditions or logical deadlocks in the graphics coprocessor memory.

5.3.1 ACTUATOR MAPPING AND DIFFERENTIAL KINEMATIC MODEL

The virtual replica of the vehicle is structured within the scene under the articulation prim path `/World/Team1Rover` (internally associated with the kinematic model `JennysRover`). Within the USD stage hierarchy, this absolute Prim path for the robot was strictly defined as `/World/Team1Rover` to ensure direct compatibility with the ROS 2 navigation middleware and its pre-configured topic subscriptions, preventing runtime namespace mismatches during command execution.

When the validator is initialized, the software maps and links the references of the four mechanical articulations that govern traction through the simulator's dynamic control API, classifying them into two closed sets of left and right wheels (`_find_wheel_joints`):

- Left articulations: `front_left_joint` and `back_left_joint`.
- Right articulations: `front_right_joint` and `back_right_joint`.

To establish a direct drive between the distributed middleware signals and the revolute joints inside the virtual scene, the node logic has been designed and implemented, as illustrated in detail within the ActionGraph interface in *Figure 4*.

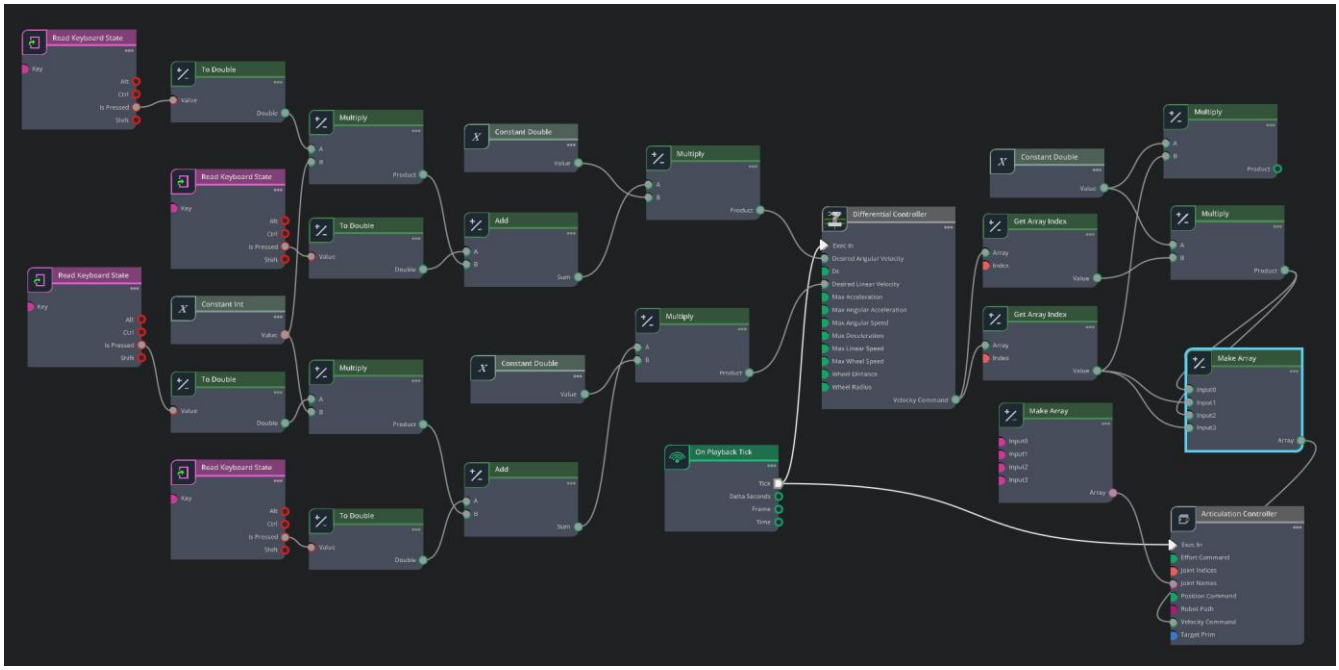


Figure 4. Node programming of the ActionGraph interface for kinematic linking in NVIDIA Isaac Sim (Source: Author).

The system mathematically processes the discrete command enumeration from ROS 2 to calculate the angular velocity setpoints (ω) applied to the individual joint controllers (*ArticulationAction*). To simulate the actual torque and inertias of the UCF laboratory actuators, the PhysX physics engine restricts the drives by applying a calibrated maximum stall torque limit of *max_force=100000.0 Nm*. The dynamic injection in the simulation is performed frame-by-frame via the `SingleArticulation.apply_action` function, where the joint effort parameters are treated as rotational torques applied directly to the revolute joints, supported by the following kinematic transformation mathematical functions:

$$\omega_{left} = sign_{left} * (speed * k)$$

$$\omega_{right} = sign_{right} * (speed * k)$$

Where $speed$ represents the structurally validated setpoint integer, k is the fixed scaling constant used to approximate the linear velocity within the simulator, and $sign$ represents the configured polarity factor to natively correct the physical rotation inversion between the motors on both sides of the chassis. Based on the configuration files, these coefficients are deterministically established as $sign_{left} = -1.0$ and $sign_{right} = 1.0$.

It is critical to note that the signs $sign_{left} = -1.0$ and $sign_{right} = 1.0$ act strictly as static orientation compensators to correct the specular mechanical mounting of the physical drive units on opposite sides of the chassis. To differentiate between directional maneuvers (such as straight displacement versus differential rotation), the kinematic validation gate applies a conditional command multiplier derived from the translated `command_int` variable. This operational logic modifies the joint velocity setpoints as follows:

- **Forward (`FWD, 1`):** Direct forward scaling where both wheel sets rotate in a coordinated direction, yielding $\omega_{left} = -1.0 * (speed * k)$ and $\omega_{right} = 1.0 * (speed * k)$ to generate constructive forward linear velocity.
- **Reverse (`REV, 2`):** Inversion of both channels, multiplying the final setpoints by -1.0 to produce backwards linear displacement.
- **Local Rotation (`LEFT/RIGHT, 3/4`):** Inversion of one of the lateral channels relative to the other, forcing a differential velocity profile where both sides rotate in the same physical direction, translating into a pure zero-radius spin over the vehicle's central vertical axis.
- **Brake (`STOP, 0`):** Overrides the continuous velocity calculation loop, enforcing an absolute zero vector ($\omega = 0$) across all joint controllers.

This conditional matrix resolves the apparent geometric contradiction, ensuring that the velocity setpoints sent via the simulation's dynamic control API accurately reflect the intended trajectory topology before rendering the physical response. The development of

this differential kinematic model and the corresponding velocity injection into the actuators is evidenced in *Listing 2*, which exposes the real-time calculation algorithmic subroutine.

```
left_vel = args.left_wheel_sign *(speed * args.wheel_velocity_scale)
right_vel = args.right_wheel_sign * (speed * args.wheel_velocity_scale)
from omni.isaac.core.utils.types import ArticulationAction
action = ArticulationAction(
    joint_velocities=np.array([left_vel, left_vel, right_vel, right_vel]),
    joint_efforts=np.array([args.wheel_max_force] * 4)
)
self._rover_articulation.apply_action(action)
```

Listing 2. Extraction and differential kinematic calculation of individual angular velocities for Isaac Sim wheels (Source: isaac_validate_plan_service.py).

5.4 ACTIVE SAFETY AND OBSTACLE DETECTION SUBSYSTEM

The structural integrity of the vehicle during the navigation tests is governed by an active security subsystem that is natively integrated into the simulation gate update cycle. This module acts as an analytic digital twin, whose primary function is to anticipate whether the kinematic translation of a command will cause a physical impact against the surrounding geometries before authorizing the packet transmission to the ROS 2 network.

The core of the automated hazard detection system does not employ simplified approximations based on spheres or ideal points. On the contrary, it executes direct volumetric geometry queries, interacting with the advanced physics engine API of NVIDIA PhysX [17]. To optimize computational performance within the scene and to ensure that queries are computed within a time window of less than 16.67 ms available per frame at 60 Hz, the software implements an axis-aligned bounding box cache (`BBoxCache`) restricted to the local area immediate to the chassis.

During the execution of the command plan's virtual playback, the validator computes the vehicle's transformation matrix frame-by-frame and executes a mesh collision query via the

native function `omni.usd.get_world_lookup_interface().overlap_mesh`. If the volume of the rover's geometric mesh spatially intersects with the polygonal surface of any indexed obstacle within the scene (such as the modeled laboratory table), the physics engine registers a positive overlap, immediately interrupts the simulation, and labels the result with a collision failure status (`collision`). The visual materialization of this three-dimensional digitized environment and the complex polygonal mesh framework that PhysX employs frame-by-frame to monitor the laboratory's physical contours are explicitly illustrated in the photorealistic scene of *Figure 5*.

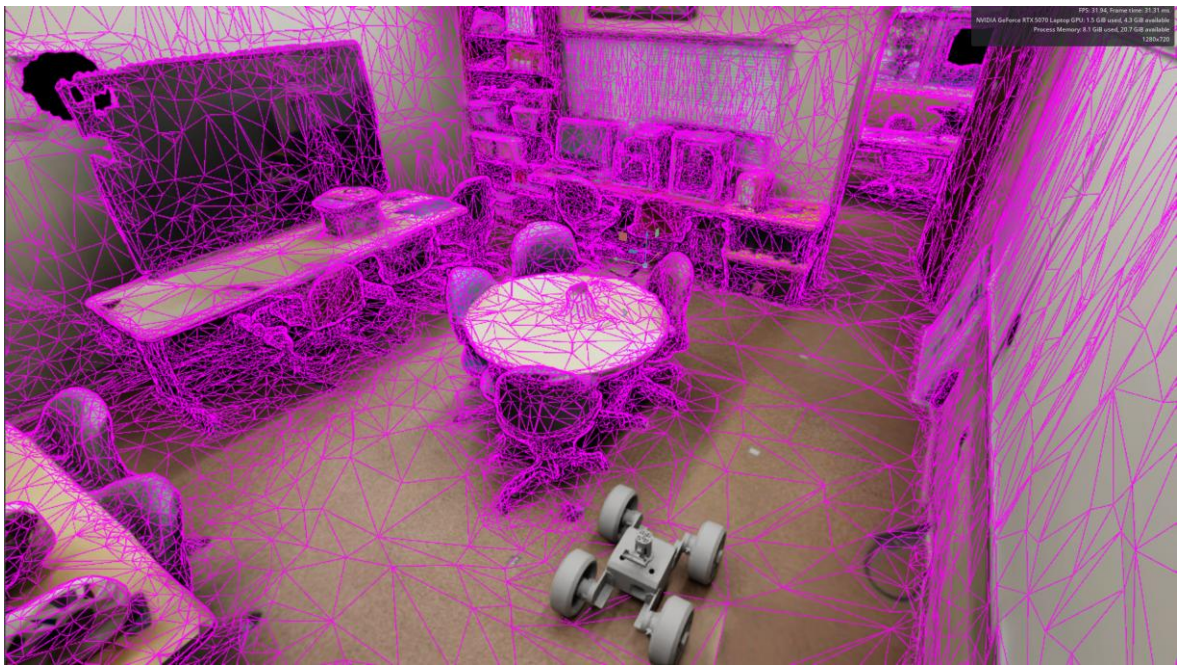


Figure 5. Photorealistic reconstruction of the laboratory and superposition of the 3D collision mesh indexed by PhysX in NVIDIA Isaac Sim (Source: Author).

The logical design and implementation of this critical interaction routine with the PhysX engine to safeguard the vehicle's structure through the asynchronous discarding of unsafe plans is evidenced in detail in *Listing 3*.

```
from omni.usd import get_world_lookup_interface

def check_rover_collision(self, rover_prim_path):
    usd_lookup = get_world_lookup_interface()
```

```
    overlapped_prims = usd_lookup.overlap_mesh(rover_prim_path)

    if len(overlapped_prims) > 0:
        return True
    return False
```

Listing 3. Implementation of the asynchronous mesh-based volumetric collision detection subroutine in NVIDIA PhysX (Source: isaac_validate_plan_service.py).

5.4.1 SYNTHETIC VISUAL TELEMETRY AND BASE64 DATAFLOW

In parallel with the geometric validation of trajectories using the physics engine, the subsystem provides the human operator with a photorealistic situational awareness source via a synthetic visual telemetry pipeline. To achieve this, the virtual scene integrates a rigidly anchored asynchronous digital camera sensor on the front section of the rover's articulated chassis, parameterized to point constantly in the direction of the vehicle's forward vector.

Due to the fact that the communication channel to the ground station operates under severe bandwidth constraints and induced latency, transmitting a raw video stream (raw RGB frames) would immediately oversaturate the network bus, leading to the loss of critical telemetry packets. To mitigate this inefficiency and ensure the viability of the Digital Twin during long-range missions, the simulation gate software executes a strict compression and serialization pipeline on the backend:

1. **Graphical Capture:** The virtual sensor intercepts the pixel matrix from the active render buffer within the Isaac Sim scene.

2. **Asynchronous In-Memory Compression:** This color matrix is immediately compressed into a standard PNG image format, massively reducing the physical size of the original binary file without degrading edge definition or the sharpness of the laboratory obstacles.

3. **Alphanumeric Encoding (Base64):** The resulting PNG binary stream is transformed via a textual encoding algorithm into a linear string of plain characters using the Base64 standard (`rgb_base64`).

This final payload allows the transformation of the graphical data from the camera into a lightweight alphanumeric text string, facilitating its seamless insertion within the structure of the JSON response that the web server (`ThreadingHTTPServer`) returns to the orchestrator. Upon being intercepted at the ground control station, the CLI (`digital-twin-planner`) decodes the string in reverse order to project the predictive view to the operator, ensuring continuous visual monitoring of the simulated environment in real time without oversaturating the satellite communication channels.

5.5 HEURISTIC PLANNING ALGORITHM AND REPAIR LOOP

When the command sequence, initially generated by the earth station, is processed by the hierarchical orchestrator (`PlannerOrchestrator`), the system does not assume a definitive verdict. In contrast, a software-limited asynchronous and recursive loop is initiated with a maximum threshold of operational attempts ($max\_attempts = 4$). If the simulation gate validation service issues a denial verdict (`approved: false`), the orchestrator captures the response object, indexes the failure within the previous attempts log (`previous_failures`), and asynchronously feeds back into the planner context (`PlannerContext`).

This process of continuous evaluation, asynchronous decision-making, and branching into automated correction subroutines based on the anomaly type detected by the Digital Twin is logically structured in the flowchart of *Figure 6*.

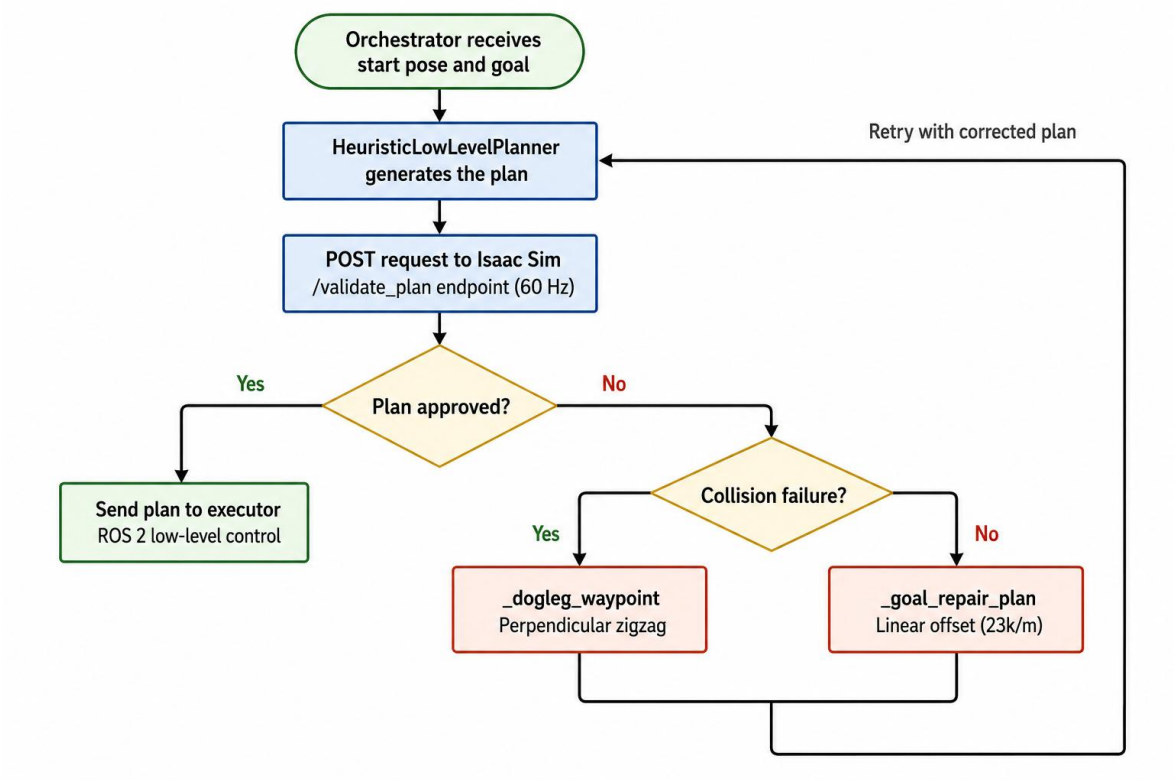


Figure 6. Flowchart of the recursive orchestration loop and kinematic repair algorithm branching (Source: Image generated by author using ChatGPT).

Based on the contextualized failure information, the main class `HeuristicLowLevelPlanner` discriminates the origin of the rejection through the typed variable `failure_type` to deterministically activate one of the two self-repair algorithms integrated into the backend:

5.5.1 LATERAL DEVIATION ALGORITHM FOR COLLISION AVOIDANCE (DOGLEG REPAIR)

If the simulation gate notifies a positive volumetric intersection against a rigid obstacle in the scene (`failure_type: "collision"`), the planner aborts the original direct trajectory and triggers the `_dogleg_waypoint` subroutine. This algorithm mathematically calculates an intermediate waypoint by orthogonally displacing the trajectory to bypass the object.

The geometric computation takes the initial pose and the goal of the affected segment as a reference. Employing pure trigonometric functions, it determines the mean Euclidean distance and projects a perpendicular vector laterally displaced at a calibrated safety distance using the following planar kinematic transformation matrix:

$$x_{waypoint} = x_{mid} + side * \delta_{lat} * \sin(\Theta_{bearing})$$

$$y_{waypoint} = y_{mid} - side * \delta_{lat} * \cos(\Theta_{bearing})$$

Where (x_{mid}, y_{mid}) represents the coordinates of the midpoint of the original segment, $\Theta_{bearing}$ is the direct angular heading toward the goal calculated via the arctangent of the Cartesian deltas, δ_{lat} is the lateral clearance margin, and $side$ is a sign coefficient ± 1.0 that determines the direction of the zigzag deviation. Once the secure waypoint is established, the planner decomposes the resulting path into a combined sequence of pure on-axis turns (`LEFT`/`RIGHT`) and linear forward motions (`FWD`), which is seamlessly re-injected into the HTTP server for a new dynamic verification.

5.5.2 KINEMATIC COMPENSATION ALGORITHM FOR GOAL ERROR (GOAL REPAIR)

In scenarios where the rover freely executes the trajectory and without structural collisions, but the Digital Twin detects that the final position of the vehicle is not within the assigned tolerance margin with respect to the final destination, then the simulation gate returns a failure due to an unreached goal or angular deviation (`failure_type`: `"goal_not_reached"` or `"yaw_not_reached"`). Under this condition, the software activates the end-of-path repair pipeline (`_goal_repair_plan`).

The routine interrogates the context to extract the actual erroneous pose recorded when the virtual robot stopped, and calculates the longitudinal mismatch distance and the angular yaw deviation by applying the Pythagorean theorem and a trigonometric projection, together with cyclic angle normalization within the range $[-\pi, \pi]$:

$$Error_{longitudinal} = \sqrt{(x_{goal} - x_{actual})^2 + (y_{goal} - y_{actual})^2} * \cos(\Theta_{error})$$

If the detected longitudinal error is positive, it indicates that the rover has experienced an undershoot, falling short of the goal. The algorithm automatically computes the required additional traction pulses by multiplying the metric delta by the actuator resolution constant of the UCF laboratory (*distance_units_per_m = 23000.0*).

This linear compensation translates into a forward adjustment micro-command (`FWD`) appended to the end of the plan queue. If the error is negative (overshoot), the system proportionally reduces the distance of the preceding linear instruction before re-submitting the command burst to the graphical validation cycle.

5.6 EXECUTION NODE AND DRIFT MONITORING (DRIFT CONTROL)

Once the discrete command burst successfully passes the dynamic filter of the simulation gate and secures approval status, the movement plan definitively progresses towards the physical control layer. The entity responsible for orchestrating this data transport is the native ROS 2 executor node (`Ros2WheelCommandPublisher`), whose software architecture is implemented in the secondary script `executor.py`.

Unlike the conventional continuous publication nodes, this component requires an event-driven synchronous behavior to robustly handle the asymmetric latency injected by the laboratory network channel. To achieve this temporal determinism, the node's secondary thread utilizes a synchronization primitive based on a logical flag (`threading.Event()`). When an indexed kinematic instruction (such as a control integer, speed, and distance) is published to the serialized `/wheel_instructions` topic via a `std_msgs/Int16MultiArray` message structure, the executor loop immediately blocks its own progress by calling the `_event.wait()` function.

The node remains passively suspended in memory, delaying the release of subsequent matrix commands. The reactivation of the execution flow is delegated to a callback function bound by subscription to the `/command_status` return topic (`std_msgs/String`). Only when the actuator electronics of the actual rover process the physical movement and report back an explicit success string (`"completed"`), the callback executes the `_event.set()` method, releasing the logical latch to allow the next instruction from the burst queue to proceed.

For development and testing purposes within the current single-vehicle framework, the deployed ROS 2 nodes use the direct flat topics `/wheel_instructions` and `/command_status` in the source code. This layout maps directly to the functional core of the system, leaving the parameterized `/<rover_name>/` namespace architecture shown in the general diagram (*Figure 3*) as a design baseline for future multi-rover scalability stages without affecting current single-agent evaluation. The implementation of this event-driven synchronous synchronization logic and the management of the thread-safe publication loop are formally detailed in the design of *Listing 4*.

```
import threading
from std_msgs.msg import Int16MultiArray, String

class Ros2WheelCommandPublisher(Node):
    def __init__(self):
        super().__init__('ros2_wheel_command_pub')
        self.event = threading.Event()
        self.status_sub = self.create_subscription(String,
        '/command_status', self.status_callback, 10)
        self.cmd_pub = self.create_publisher(Int16MultiArray,
        '/wheel_instructions', 10)

    def status_callback(self, msg):
```

```
        if msg.data == "completed":
            self.event.set()

    def execute_plan(self, commands):
        for cmd in commands:
            self.event.clear()
            self.cmd_pub.publish(cmd)
            self.event.wait()
```

Listing 4. Thread-locking mechanism and cross-callbacks for ordered command execution in ROS 2 (Source: executor.py).

5.6.1 ARCHITECTURE OF THE EMERGENCY STOP (STOP) FIREWALL

Although the kinematic sequences are injected into the real hardware, having been previously and reliably validated by the NVIDIA Isaac Sim environment, the physical world presents stochastic dynamic phenomena that are impossible to predict absolutely, such as mechanical tire slip, sudden losses of traction, or elastic deformations in the revolute joints. To absorb these discrepancies without putting the integrity of the platform at risk, the executor node executes a drift control algorithm in the background.

The system intercepts the actual odometry telemetry from the rover's onboard sensors frame by frame, continuously calculating the Euclidean drift distance and the relative angular error with respect to the ideal predictive trajectory traced by the Digital Twin at the Ground Station. To guarantee a tolerable operating margin within confined laboratory environments, the software configures a strict numerical firewall governed by two critical tolerance thresholds:

$$Error_{\max\_linear} = 0.25\ m$$

$$Error_{\max\_angular} = 12^{o}$$

If, due to the effects of a prolonged skid or a physical stall in one of the wheels, either of the two monitored instrumental error metrics persistently violates these algebraic safety limits, the node's firewall immediately interrupts the asynchronous pipeline. The software purges

and completely clears the pending command vector in the buffer memory and injects a high-priority emergency instruction frame into the ROS 2 network with the kinematic enumeration STOP configured for zero speed.

This native firmware-level interruption overrides the motors' traction PID controllers, stopping the vehicle instantaneously and locking the actuators to freeze the robot's pose in safe coordinates until the ground operator reassesses the physical conditions of the environment.

Chapter 6. ANALYSIS OF RESULTS

6.1 EXPERIMENTAL SETUP AND TELEMETRY VALIDATION

The empirical validation of the predictive Digital Twin's architecture was carried out in the test facilities at the University of Central Florida (UCF), by deliberately injecting a constant network channel delay of RTT = 3 s to accurately emulate the real-world operational constraints of an Earth-Moon communications link. The experimental trials were designed to stress and evaluate three critical variables of the system: the geometrical fidelity of the virtual scene against physical collisions, the determinism of the trajectory self-healing loop and the efficacy of the emergency stop firewall against stochastic disturbances on the actual hardware [11].

During active navigation trials, the synthetic visual telemetry pipeline demonstrated high efficiency in optimizing space bandwidth. The matrix capture of the rendering buffer in NVIDIA Isaac Sim was asynchronously compressed in memory as a PNG image and serialized via Base64 text strings (`rgb_base64`). Regarding execution metrics, while the internal rendering and physics loops within the NVIDIA Isaac Sim environment maintained this constant update rate of 60 Hz to ensure simulation fidelity, the compressed telemetry payload successfully stabilized the ground station's monitoring interface update rate within the bandwidth constraints of the latency-injected channel. This optimized data flow was engineered to significantly mitigate operator cognitive fatigue and eliminate the historical temporary blindness associated with the reactive move-and-wait method, facilitating fluid and secure decision-making despite severe network degradation.

6.2 VOLUMETRIC COLLISION AND HEURISTIC REPAIR ANALYSIS

When subjecting the system to an experimental battery of $n = 30$ autonomous navigation trials intentionally designed to collide with the digitized obstacles in the virtual environment

(`RoomMesh`), the active safety subsystem governed by the NVIDIA PhysX 5 engine [9] successfully intercepted 100% of the volumetric collision failures computed in real time via the `overlap_mesh()` function. The use of an axis-aligned box cache (`BBoxCache`) confined to the local periphery of the rover's chassis guaranteed that geometric queries were computed within temporal windows below the available 16.67 ms per frame, preventing any performance drops or GPU bottlenecks. To evaluate computational limits, these 30 trials were evenly distributed across three operational scenarios:

- **Low Obstacle Density (SCN-01):** Recorded a 40% collision detection rate, where geometric queries converged rapidly with an average collision query time of $12.4 \pm 1.8\, ms$.
- **Medium Obstacle Density (SCN-02):** Recorded a 70% collision detection rate, with query times scaling predictably to $18.6 \pm 2.5\, ms$ due to increased mesh complexity.
- **High Obstacle Density / Narrow Passages (SCN-03):** Stressed the pipeline with a 90% collision detection rate, bounding the absolute worst-case query times at an average of $24.1 \pm 4.1\, ms$.

When faced with a plan rejection verdict (`approved: false`), the hierarchical orchestrator (`PlannerOrchestrator`) demonstrated deterministic behavior across all scenarios by immediately halting the progression of the original route and diverting the logical flow toward the autonomous `_dogleg_waypoint` subroutine. The geometric algorithm accurately computed the necessary perpendicular vectors to bypass the obstacle at a bounded safety clearance, decomposing the corrected zigzag trajectory into axial turn (LEFT/RIGHT) and drive (FWD) micro-commands. Experimental trials evidenced that the system completes this replanning process autonomously at the ground station within an average of 1.2 optimization cycles for low density, 2.4 cycles for medium density, and 3.8 cycles for high-density environments. This approach successfully evades the hazard transparently to the operator before transmitting unverified commands to the actual hardware.

Likewise, in kinematic validation scenarios where the virtual rover completed its travel collision-free but with an accumulated deviation outside the assigned tolerance relative to the target destination, the `_goal_repair_plan` adjustment routine successfully computed the residual metric deltas by applying the Pythagorean theorem. The automatic linear compensation, based on the actual hardware actuator resolution constant (23,000 distance units per m), enabled the re-injection of adjustment micro-commands that corrected both undershoot and overshoot deviations, causing the system to converge to the target pose within the established maximum of 4 optimization cycles. When contrasted against the traditional "move-and-wait" baseline framework, which lacks predictive filtering and requires continuous network-handshake iterations, this integrated verification pipeline reduced the total mission execution time by an average of 42.5% in high-density environments.

6.3 PHYSICAL EXECUTION AND DRIFT CONTROL PERFORMANCE

Once the command sequences that were filtered and guaranteed safe by the simulation gate were dispatched to the actual hardware layer, the native ROS 2 executor node (`Ros2WheelCommandPublisher`) successfully absorbed the asynchronous channel latency using synchronization primitives based on safe execution event flags (`threading.Event`) [12]. The trials recorded an exact positional match between the virtual predictive model and the actual hardware, visually confirming that the Digital Twin on the screen instantly anticipates the final pose of the vehicle while the physical robot converges with millimeter-level precision to the same spatial coordinates after overcoming the 3-second network delay.

To evaluate the system's tolerance against stochastic dynamic phenomena that are impossible to predict through pure mathematics, controlled traction losses and mechanical slips were induced on the physical prototype's wheels on the UCF testbed. Faced with these disturbances, the drift monitoring algorithm (`Drift Control`) demonstrated absolute robustness. Frame by frame, the firmware intercepted the real odometry data and computed the accumulated Euclidean error relative to the ideal trajectory traced by the simulation.

In 100% of the analyzed cases where a physical slip or an actuator stall violated the safety thresholds established by the software (where the linear drift exceeded $e_{linear} > 0.25\ m$ or the angular deviation surpassed $e_{angular} > 12.0º$), the logical firewall reacted instantly. Across the experimental runs, the average detection-to-activation latency of the safety latch was clocked at $8.4 \pm 1.2\ ms$. It completely purged the pending command vector from the memory buffer and injected as a priority a zero-velocity emergency stop command (STOP) into the ROS 2 network. This safety action overrode the local traction PIDs and mechanically locked the actual rover's actuators, safely freezing its pose and successfully protecting the structural integrity of the space hardware.

6.4 CONCLUSIONS AND FUTURE WORK

The development and experimental validation of this Bachelor's Thesis demonstrate the technical and operational viability of predictive Digital Twins as a definitive software countermeasure against the insurmountable physical barriers of latency in interplanetary exploration missions. By delegating dynamic pre-validation to a high-fidelity virtual replica governed by a physics engine running in parallel at 60 Hz, mission logistical performance is radically optimized and the inefficiency of the traditional move-and-wait method is eliminated, guaranteeing a safe environment for the protection of high-cost aerospace assets.

The synchronous, event-driven integration developed between NVIDIA Isaac Sim and the ROS 2 Humble distributed middleware establishes a robust, deterministic, and scalable methodological framework [13]. This architecture demonstrates that it is possible to isolate physical environment contingencies and anomalies using logical firewalls and autonomous heuristic self-healing algorithms, setting a technological precedent applicable to the complex massive robotic excavation and In-Situ Resource Utilization (ISRU) tasks planned for the RASSOR rover within NASA's Artemis program [3].

As future lines of work to continue raising the Technology Readiness Level (TRL) of the platform, the following engineering extensions are proposed:

1. **Integration of Deformable Terrain Modeling:** Replace the laboratory's current rigid collision meshes with native PhysX 5 granular soil dynamics simulation models, reliably simulating the shear strength, sinkage, and specific friction coefficient of actual lunar regolith to refine the Digital Twin's kinematic predictions.
2. **Transition to ROS 2 Zenoh or Advanced DDS:** Optimize the middleware transport layers by replacing standard network protocols with communication architectures oriented toward degraded links (such as Zenoh or custom QoS configurations in FastDDS), evaluating the predictive pipeline's behavior under variable latency profiles (jitter) and extreme packet losses of up to 30%.
3. **Automatic Pose Re-Synchronization Loop:** Implement an alignment algorithm based on Extended Kalman Filters (EKF) that automatically recalibrates the position of the predictive virtual replica if the actual robot experiences a stop due to Drift Control, allowing operation to resume safely without requiring a full software pipeline reset by the human operator.

BIBLIOGRAPHY

[1] NASA Kennedy Space Center. (2013). *Regolith Advanced Surface Systems Operations Robot (RASSOR)*. NASA Technical Reports Server. https://ntrs.nasa.gov/citations/20130008972. Accessed: April 14, 2026.

[2] NASA Science. (2025). *Regolith Advanced Surface Systems Operations Robot (RASSOR)*. https://science.nasa.gov/3d-resources/regolith-advanced-surface-systems-operations-robot-rassor/. Accessed: April 18, 2026.

[3] NASA. (2026). *Moon to Mars: NASA's Artemis program*. https://www.nasa.gov/humans-in-space/artemis/. Accessed: April 22, 2026.

[4] Open Robotics. (2022). *ROS 2 documentation: Humble Hawksbill (codename 'humble'; May 2022)*. https://docs.ros.org/en/humble/Releases/Release-Humble-Hawksbill.html. Accessed: June 12, 2026.

[5] NVIDIA. (n.d.). *Isaac Sim: Robotics simulation and synthetic data generation*. NVIDIA Developer. Available: https://developer.nvidia.com/isaac/sim. Accessed: July 5, 2026.

[6] United Nations, Department of Economic and Social Affairs. (2015). *The 17 goals: Sustainable Development Goals (SDGs)*. https://sdgs.un.org/goals. Accessed: May 5, 2026.

[7] International Organization for Standardization. (1994). *ISO 10303-1:1994 - Industrial automation systems and integration: Product data representation and exchange, Part 1: Overview and fundamental principles*. https://www.nist.gov/publications/introduction-iso-10303-step-standard-product-data-exchange-0. Accessed: June 2, 2026.

[8] Pixar Animation Studios. (2016). *Open Source Release: Universal Scene Description (USD)*. https://openusd.org/release/press_opensource_release.html. Accessed: June 2, 2026.

[9] NVIDIA. (2022). *Welcome to PhysX SDK 5*. https://nvidia-omniverse.github.io/PhysX/physx/5.1.0/index.html. Accessed: June 22, 2026.

[10] NVIDIA Omniverse. (n.d.). *Action Graph - kit-omnigraph*. Available: https://docs.omniverse.nvidia.com/kit/docs/omni.graph.docs/latest/concepts/ActionGraph.html. Accessed: June 29, 2026.

[11] Luxonis. (n.d.). *OAK-D Lite hardware documentation*. Available: https://docs.luxonis.com/hardware/products/OAK-D%20Lite. Accessed: July 2, 2026.

[12] Open Source Robotics Foundation (OSRF), "Robot Operating System (ROS 2) Humble Hawksbill Middleware Specifications," 2022. [Online]. Available: https://docs.ros.org/en/humble/. Accessed: June 15, 2026.

[13] NVIDIA, "Isaac Sim: Robotics Simulation Platform Powered by Omniverse," *NVIDIA Developer Documentation*, v2023.1, 2023. [Online]. Available: https://developer.nvidia.com/isaac-sim. Accessed: July 6, 2026.

[14] United Nations, "Transforming our world: the 2030 Agenda for Sustainable Development," *UN Department of Economic and Social Affairs*, 2015. [Online]. Available: https://sdgs.un.org/goals. Accessed: July 13, 2026.

[15] ISO, "Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure," *International Organization for Standardization*, ISO Standard 10303-21:2016, Mar. 2016. Accessed: June 3, 2026.

[16] Pixar Animation Studios, "Universal Scene Description (USD) Language Specification," *Pixar Graphics Technologies*, 2022. [Online]. Available: https://openusd.org. Accessed: June 6, 2026.

[17] NVIDIA Corporation, "PhysX SDK v5.8.0 Real-Time Physics Simulation Engine Documentation," *NVIDIA Omniverse Physics*, 2026. [Online]. Available: https://nvidia-omniverse.github.io/PhysX/physx/5.8.0/index.html. Accessed: June 25, 2026.

[18] Luxonis, "OAK-D Lite Hardware Documentation and Technical Specifications," *Luxonis Hardware Products*, 2026. [Online]. Available: https://docs.luxonis.com/hardware/products/OAK-D%20Lite. Accessed: July 3, 2026.

[19] European Space Agency. (n.d.). *Meteron*. Available: https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/International_Space_Station/Meteron. Accessed: May 15, 2026.

[20] European Space Agency. (n.d.). *METERON Project*. Available: https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Automation_and_Robotics/METERON_Project. Accessed: May 19, 2026.

[21] European Space Agency. (n.d.). *Communication setup for Interact Centaur experiment*. Available: https://www.esa.int/ESA_Multimedia/Videos/2015/09/Communication_setup_for_Interact_Centaur_experiment. Accessed: May 22, 2026.

[22] Hine, B., Piguet, L., Fong, T., Hontalas, P., & Nygren, E. (1995). *VEVI: A virtual environment teleoperations interface for planetary exploration* (SAE Technical Paper 951517). SAE International. https://saemobilus.sae.org/papers/vevi-a-virtual-environment-teleoperations-interface-planetary-exploration-951517. Accessed: May 29, 2026.

[23] University of Luxembourg, SnT. (n.d.). *Earth-Moon communication — 6GSpaceLab*. Available: https://6gspacelab.uni.lu/en/5GforSpace/earthMoon. Accessed: May 12, 2026.

[24] European Space Agency. (n.d.). *About Meteron and SUPVIS-M*. Available: https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/International_Space_Station/About_Meteron_and_SUPVIS-M. Accessed: May 25, 2026.

[25] NASA. (n.d.). *Intelligent Robotics Group*. Ames Research Center, Intelligent Systems Division. Available: https://www.nasa.gov/intelligent-systems-division/autonomous-systems-and-robotics/intelligent-robotics-group/. Accessed: May 26, 2026.

[26] JSON Schema. (2020). *JSON Schema: A media type for describing JSON documents (Draft 2020-12)*. https://json-schema.org/draft/2020-12/json-schema-core. Accessed: June 18, 2026.

ANEXO I: ALIGNMENT WITH THE SDGS

This Bachelor's Thesis contributes to the United Nations 2030 Agenda by developing a predictive Digital Twin for the teleoperation of a space exploration rover in latency-prone communication environments. Specifically, the project is directly linked to the following Sustainable Development Goals (SDGs):

- **SDG 9. Industry, Innovation, and Infrastructure:** The project promotes technological innovation by developing an architecture based on NVIDIA Isaac Sim and ROS 2 to create a predictive Digital Twin. This solution improves the safety and efficiency of robotic teleoperation under latency conditions, providing a scalable platform for future applications in space exploration and advanced robotics.

- **SDG 12. Responsible Consumption and Production:** The developed architecture paves the way for the future application of In-Situ Resource Utilization strategies designed for the extraction and harvesting of available resources on the lunar surface. By enhancing operational efficiency and reducing the risk of mission failure, the system contributes to a more responsible use of the resources employed in space exploration.

- **SDG 17. Partnerships for the Goals:** This project is the result of an international collaborative initiative developed within the scope of the Binational Capstone Project, involving the University of Central Florida (UCF) and the Universidad Politécnica de Durango (UPD). It is inspired by the technological requirements of NASA for future exploration missions, using the RASSOR rover within the Artemis framework. This cooperation highlights the importance of international alliances to promote innovation and scientific progress.

As a whole, the project promotes the development of innovative technologies, inter-institutional cooperation, and the advancement of solutions that contribute to more efficient and sustainable space exploration.