



Universidad de
Oviedo



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

ÁREA DE TEORÍA DE LA SEÑAL Y COMUNICACIONES

**DESARROLLO DE UNA HERRAMIENTA DE EVALUACIÓN DE
SUBSISTEMAS DE COMUNICACIONES PARA ANÁLISIS DE
MISIONES DE SATÉLITES DE ÓRBITA BAJA**

**D. LENA DÍAZ, Guillermo Segifredo
TUTOR: D. LEÓN FERNÁNDEZ, Germán**

FECHA: Julio 2023



ACRÓNIMOS

- ADCS: Attitude Determination and Control System
- AOS: Acquisition of Signal
- ASM: Attached Sync Marker
- BER: Bit Error Rate
- CAN: Controller Area Network
- CC: Convolutional Codes
- CCSDS: The Consultative Committee for Space Data Systems
- CRC: Cyclic Redundancy Check
- CSP: CubeSat Space Protocol
- EPS: Electrical Power System
- FEC: Forward Error Correction
- GMSK: Gaussian Minimum Shift Keying
- GRC: GNU Radio Companion
- LEO: Low Earth Orbit
- LNA: Low Noise Amplifier
- MCS: Mission Control Software
- OBC: Onboard Computer
- OOT: Out-Of-Tree
- PER: Packet Error Rate
- PIRE: Potencia Isotrópica Radiada Equivalente
- PDU: Protocol Data Unit
- RF: Radio Frecuencia
- RS: Reed Solomon
- RX: Recepción
- SDR: Software Defined Radio
- SNR: Signal to Noise Ratio
- TLE: Two Line Element
- TNC: Terminal Node Controller
- TTC: Telemetry Tracking & Command
- TX: Transmisión
- UIT: Unión Internacional de Telecomunicaciones
- UHF: Ultra High Frequency
- VHF: Very High Frequency
- ZMQ: Zero Message Queue



ÍNDICE GENERAL

1.- Introducción	11
1.1.- Motivación	11
1.2.- Objetivo	12
1.3.- Planificación	12
1.4.- Estructura de la memoria	13
2.- Misiones LEO	14
2.1.- Low Earth Orbit	15
2.2.- Estación terrena.....	16
2.2.1.- Mission Control Software	18
2.3.- CubeSat.....	19
2.4.- Canal de comunicaciones.....	22
2.4.1.- Balance de enlace.....	24
3.- Herramientas empleadas.....	31
3.1.- GPredict	32
3.2.- Docker.....	33
3.2.1.- Introducción	33
3.2.2.- Contenedores.....	35
3.2.3.- Dockerfile, Build y Run	37
3.3.- GNU Radio	38
3.3.1.- Introducción	38
3.3.2.- Bloques en GNU Radio y GRC como entorno de desarrollo	39
3.3.3.- Comunicación entre bloques	41
3.4.- Gr-leo	43
3.4.1.- Introducción	43
3.4.2.- Perturbaciones del canal consideradas	44
3.4.3.- Arquitectura de bloques	45
3.4.4.- Limitaciones.....	56
3.4.5.- Ejemplo	57
3.5.- LibCSP.....	60



3.5.1.- Introducción	60
3.5.2.- Conceptos básicos (v.1.6)	62
3.5.3.- Terminología de red	64
3.5.4.- Python bindings.....	66
3.6.- ZeroMQ	66
3.6.1.- Publicador-Subscriptor.....	67
3.6.2.- Proxy ZeroMQ	67
4.- Desarrollo y Funcionamiento.....	69
4.1.- MCS/CubeSat	70
4.1.1.- Nodo Transmisor (cliente)	70
4.1.2.- Nodo Receptor (servidor).....	71
4.2.- Bloque CSP ZMQ Hub	72
4.3.- Implementación de módems	74
4.3.1.- GMSK	74
4.3.2.- Satlab SRS-3	77
4.4.- PER	81
4.4.1.- Algoritmo PER.....	82
4.4.2.- Ventana de AOS y paquetes totales	85
4.4.3.- Potencia de señal en GNU Radio	85
5.- Pruebas realizadas	88
5.1.- Balance de enlace.....	88
5.1.1.- Resultados	90
5.1.2.- Análisis de los resultados	100
5.2.- Evaluación PER	101
5.2.1.- Resultados	102
5.2.2.- Análisis de los resultados	105
6.- Conclusiones y líneas de trabajo futuras	107
6.1.- Conclusiones.....	107
6.2.- Líneas de trabajo futuras.....	108
Referencias.....	109
Anexo A. Balance de enlace Python.....	113
Anexo B. Docker	120



Anexo C. Nodo TX/RX Python 126



ÍNDICE DE FIGURAS

Figura 1.1.- Diagrama de Gantt de la planificación llevada a cabo durante este proyecto.	13
Figura 2.1.- Esquema simplificado de los elementos básicos involucrados en una misión de órbita baja terrestre o LEO.	14
Figura 2.2.- Esquema simplificado de los elementos básicos que forman una estación terrena.	17
Figura 2.3.- Esquema de los subsistemas básicos que forman parte de un CubeSat [2].	19
Figura 2.4.- Escenario básico de un canal de comunicaciones para misiones de órbita baja terrestre o LEO.	22
Figura 2.5.- Método geométrico del cálculo del rango oblicuo máximo entre la estación terrena y el CubeSat para una determinada elevación mínima en el horizonte. [7]	25
Figura 3.1.- Esquema de la relación entre las herramientas empleadas en el proyecto.	31
Figura 3.2.- Captura de pantalla de la herramienta GPredict mostrando su interfaz gráfica con el seguimiento e información orbital de dos satélites.	32
Figura 3.3.- Escenario de despliegue de diferentes aplicaciones por medio de la tecnología Docker. [16]	35
Figura 3.4.- Captura de pantalla de la interfaz gráfica de la herramienta GRC.	39
Figura 3.5.- Captura de pantalla de la ventana de ayuda mostrando la leyenda de colores de los diferentes tipos de datos en GRC.	40
Figura 3.6.- Ejemplo de un diagrama de bloques en GRC por medio del cual se emplea el uso de las stream tags.	41
Figura 3.7.- Resultado de la ejecución del diagrama de bloques de la Figura 3.6.	42
Figura 3.8.- Ejemplo de un diagrama de bloques en GRC que emplea el uso del tipo de datos Async Message.	42
Figura 3.9.- Captura de pantalla del resultado en la terminal de GRC tras la ejecución del programa de la Figura 3.8.	43
Figura 3.10.- Representación del bloque Channel Model del módulo OOT gr-leo en GRC.	45
Figura 3.11.- Representación del bloque LEO Model Definition del módulo OOT gr-leo en GRC.	46
Figura 3.12.- Representación del bloque Tracker del módulo OOT gr-leo en GRC.	48
Figura 3.13.- Representación del bloque Satellite del módulo OOT gr-leo en GRC.	50



Figura 3.14.- Representación del bloque Antenna del módulo OOT gr-leo en GRC mostrando 4 diferentes tipos de antena disponibles.	51
Figura 3.15.- Diagrama de bloques desarrollado en GRC para el ejemplo donde se emplean todos los bloques y funcionalidades del módulo OOT gr-leo.	57
Figura 3.16.- Captura de pantalla del resultado de la ejecución del diagrama de bloques de la Figura 3.15.	58
Figura 3.17.- Captura de pantalla del archivo CSV donde se guardan los datos del balance de enlace generados por el módulo OOT gr-leo.	58
Figura 3.18.- Captura de pantalla de la herramienta GPredict con la configuración de seguimiento del CubeSat Lume 1.	59
Figura 3.19.- Captura de pantalla de la ventana donde se muestran los detalles de la información de la ventana de AOS para el CubeSat Lume 1.	60
Figura 3.20.- Formato de las tramas de la red CSP versión 1.6. [34].	62
Figura 3.21.- Esquema conceptual de los diferentes bloques CSP haciendo uso de la interfaz CAN [35].	63
Figura 3.22.- Esquema de topología típica de una red CSP [36].	65
Figura 4.1.- Esquema básico de funcionamiento y conectividad a nivel de red CSP de la herramienta final desarrollada.	69
Figura 4.2.- Captura de pantalla de la terminal donde se ha ejecutado un ejemplo de estos nodos transmisor y receptor.	72
Figura 4.3.- Representación del bloque csp_zmqhub_pdu en GRC.	73
Figura 4.4.- captura de pantalla de la ejecución del ZMQ proxy utilizado.	74
Figura 4.5.- Diagrama de bloques donde se implementa el modem GMSK junto con el módulo OOT gr-leo. Parte de gr-leo.	75
Figura 4.6.- Captura de pantalla del resultado de la ejecución del diagrama de bloques de las Figuras 4.5.	76
Figura 4.7.- Captura de pantalla del resultado en la terminal de GRC de la ejecución del diagrama de bloques de las Figuras 4.5.	77
Figura 4.8.- Diagrama de bloques donde se implementa el modem de Satlab SRS-3 junto con el módulo OOT gr-leo.	78
Figura 4.9.- Formato de la trama empleada por el modem de Satlab SRS-3 [42].	79
Figura 4.10.- Captura de pantalla del resultado de la ejecución del diagrama de bloques de la Figura 4.8.	80
Figura 4.11.- Captura de pantalla del resultado en la terminal de GRC de la ejecución del diagrama de bloques de la Figura 4.8.	81



Figura 4.12.- Diagrama de flujo del algoritmo empleado en la transmisión de la prueba PER.....	82
Figura 4.13.- Diagrama de flujo del algoritmo empleado en la recepción de la prueba PER.	83
Figura 4.14.- Captura de pantalla del resultado final de la ejecución de una prueba PER..	84
Figura 4.15.- Diagrama de bloques utilizado para la comprensión de la potencia de señal en GNU Radio.....	86
Figura 4.16.- Captura del resultado de la ejecución del diagrama de bloques de la Figura 4.18.	87
Figura 5.1.- Diagrama de bloques empleado para obtener el balance de enlace del CubeSat Lume 1 con la estación terrena.	90
Figura 5.2.- Resultado del rango oblicuo del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	92
Figura 5.3.- Resultado de las pérdidas de apuntamiento del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	92
Figura 5.4.- Resultado del desplazamiento Doppler del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	93
Figura 5.5.- Resultado de la elevación del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	93
Figura 5.6.- Resultado de las pérdidas de espacio libre del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	94
Figura 5.7.- Resultado de las pérdidas por gases atmosféricos del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.	94
Figura 5.8.- Resultado de las pérdidas por precipitación atmosférica del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.	95
Figura 5.9.- Resultado de la SNR del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	95



Figura 5.10.- Resultado del rango oblicuo del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	96
Figura 5.11.- Resultado de las pérdidas de apuntamiento del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	96
Figura 5.12.- Resultado del desplazamiento Doppler del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	97
Figura 5.13.- Resultado de la elevación del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	97
Figura 5.14.- Resultado de las pérdidas de espacio libre del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	98
Figura 5.15.- Resultado de las pérdidas por gases atmosféricos del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.	98
Figura 5.16.- Resultado de las pérdidas por precipitación atmosféricos del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.	99
Figura 5.17.- Resultado de la SNR del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	99



ÍNDICE DE TABLAS

Tabla 2.1.- Datos orbitales arbitrarios para el balance de enlace.	24
Tabla 2.2.- Datos utilizados para el enlace ascendente del balance de enlace.	26
Tabla 2.3.- Datos utilizados para el enlace descendente del balance de enlace.	26
Tabla 2.4.- Datos de la estación terrena utilizados para el balance de enlace.	27
Tabla 2.5.- Datos del satélite utilizados para el balance de enlace.....	28
Tabla 2.6.- Resultados del balance de enlace para ambos enlace ascendente y descendente.	28
Tabla 3.1.- Valores de configuración de la antena Yagi utilizados por el módulo OOT gr- leo.	51
Tabla 5.1.- Datos del enlace ascendente utilizados para el cálculo del balance de enlace..	88
Tabla 5.2.- Datos del enlace descendente utilizados para el cálculo del balance de enlace.	88
Tabla 5.3.- Datos de la estación terrena utilizada para el cálculo del balance de enlace. ...	89
Tabla 5.4.- Datos del CubeSat Lume 1 utilizados para el cálculo del balance de enlace....	89
Tabla 5.5.- Resultados del balance de enlace entre una estación terrena y el CubeSat Lume 1.	90
Tabla 5.6.- Resultados de las pruebas PER realizadas del modem de Satlab para diferentes escenarios del enlace ascendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	102
Tabla 5.7.- Resultados de las pruebas PER realizadas del modem de Satlab para diferentes escenarios del enlace descendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	103
Tabla 5.8.- Resultados de las pruebas PER realizadas del modem GMSK para diferentes escenarios del enlace ascendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	104
Tabla 5.9.- Resultados de las pruebas PER realizadas del modem GMSK para diferentes escenarios del enlace descendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	104
Tabla 5.10.- Resultados de las pruebas PER realizadas del modem de Satlab en banda S para diferentes escenarios del enlace ascendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.....	104



Tabla 5.11.- Resultados de las pruebas PER realizadas del modem de Satlab en banda S para diferentes escenarios del enlace descendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30..... 105



1.- INTRODUCCIÓN

1.1.- Motivación

Actualmente estamos en una época de auge en el desarrollo de tecnologías en el sector espacial. Esto se debe, en gran medida, al conocido como movimiento New Space, el cual hace referencia a una nueva era en la exploración espacial que ha surgido en los últimos años. A diferencia de la antigua forma de hacer negocios en el sector espacial, dominada por agencias gubernamentales y grandes corporaciones, el New Space se caracteriza por la participación de empresas privadas y startups en la industria espacial. Estas compañías están impulsando la innovación, la optimización y la reducción de costos en el acceso a este sector, y están llevando a cabo misiones y proyectos que están transformando la forma en que utilizamos y exploramos el cosmos.

Una de las áreas más crecientes de este movimiento es la realización de misiones de satélites en órbita baja terrestre. Este tipo de satélites, ubicados a altitudes relativamente bajas, generalmente por debajo de 2000 kilómetros, ofrecen una serie de ventajas y oportunidades. En primer lugar, las misiones de satélites de órbita baja permiten una menor latencia y mayor ancho de banda en comparación con los satélites en órbitas más altas. Esto los hace ideales para aplicaciones que requieren una comunicación en tiempo real, como las redes de telecomunicaciones y la transmisión de datos de alta velocidad. Además, este tipo de satélites ofrecen una mayor resolución en aplicaciones como la observación de la Tierra ya que, al estar más cerca de nuestro planeta, se puede capturar imágenes más detalladas y recopilar datos más precisos sobre la superficie terrestre, la atmósfera y otros fenómenos. Esto tiene un gran potencial en áreas como la agricultura, la monitorización del medio ambiente, la gestión de desastres y la navegación precisa.

El auge de las misiones de satélites de órbita baja terrestre ha sido impulsado en gran medida, también, por la miniaturización de la tecnología espacial y la disminución de los costos de lanzamiento. De esta forma, se hace posible actualmente la construcción y el lanzamiento de satélites más pequeños y asequibles, lo que ha democratizado el acceso al espacio y ha abierto la puerta a nuevas empresas y proyectos. Esto ha llevado a un aumento significativo en el número de este tipo de satélites, con constelaciones completas de satélites, por ejemplo, que trabajan en conjunto para proporcionar cobertura global y servicios innovadores.

En esta misma línea, la creación de herramientas de asistencia al desarrollo de sistemas, así como de mantenimiento de misión, se ha convertido en una prioridad para maximizar el rendimiento en este tipo de misiones. El uso de herramientas de simulación y entrenamiento se ha vuelto fundamental en el diseño y operación de sistemas empleados en estas misiones de órbita baja terrestre. Estas herramientas permiten a los ingenieros y desarrolladores probar y validar los sistemas antes de su implementación, lo que ayuda a reducir los riesgos y mejorar la eficiencia de las misiones espaciales.



Una de las principales aplicaciones de las herramientas de simulación es el modelado y simulación de la dinámica orbital de los satélites en órbita baja. Estos modelos permiten predecir con precisión el movimiento y la posición de los satélites en relación con la Tierra y otros cuerpos celestes, teniendo en cuenta factores como la gravedad, el arrastre atmosférico y las perturbaciones gravitacionales. Esta información es crucial para la planificación de las maniobras de los satélites, como las correcciones de órbita y las maniobras de evasión de basura espacial, así como para el cálculo de ventanas de comunicación donde existe una línea de visión directa entre Tierra y espacio. Además, las herramientas de simulación también son utilizadas para entrenar a los operadores de sistemas satelitales en escenarios realistas. Los simuladores permiten recrear diferentes escenarios y eventos que pueden ocurrir durante una misión, como fallos en los sistemas, interacciones con otros satélites o la detección de objetos en la órbita. Esto ayuda a los operadores a familiarizarse con el funcionamiento del sistema y a adquirir las habilidades necesarias para tomar decisiones rápidas y efectivas en situaciones de emergencia.

Es entonces, bajo este contexto y de la mano de la empresa Alen Space [1], que surge la idea de desarrollo del presente proyecto titulado como "Desarrollo de una herramienta de evaluación de subsistemas de comunicaciones para análisis de misiones de satélites de órbita baja". De esta forma, se pretende aportar una ayuda a la hora de entrenar subsistemas de comunicaciones y poder planear las comunicaciones Tierra-espacio de forma óptima y eficiente.

1.2.- Objetivo

El objetivo de este proyecto es el de desarrollar una herramienta que permita evaluar y analizar subsistemas de comunicaciones destinados a misiones de satélites pequeños de órbita baja terrestre.

Con este fin se estudiará en primer lugar el entorno que ha de simular la herramienta, desde cualquiera de los dos puntos iniciales/finales, véase CubeSat o estación terrena hasta el canal de comunicaciones en sí. Además, a lo largo del proyecto se analizarán también las diferentes herramientas que serán empleadas para lograr esto. Finalmente, se presentará un diseño final implementado con la finalidad de evaluar algunos módems de comunicaciones por medio de una medida de PER durante el desarrollo de una misión arbitraria.

1.3.- Planificación

A continuación, en la Figura 1.1 se muestra el diagrama de Gantt que expone la planificación que ha sido llevada a cabo para el correcto desarrollo de este proyecto. En este se muestra a la izquierda las principales tareas llevadas a cabo, mientras que a la derecha se muestran los intervalos de tiempo que han sido utilizados para el cumplimiento de estas.

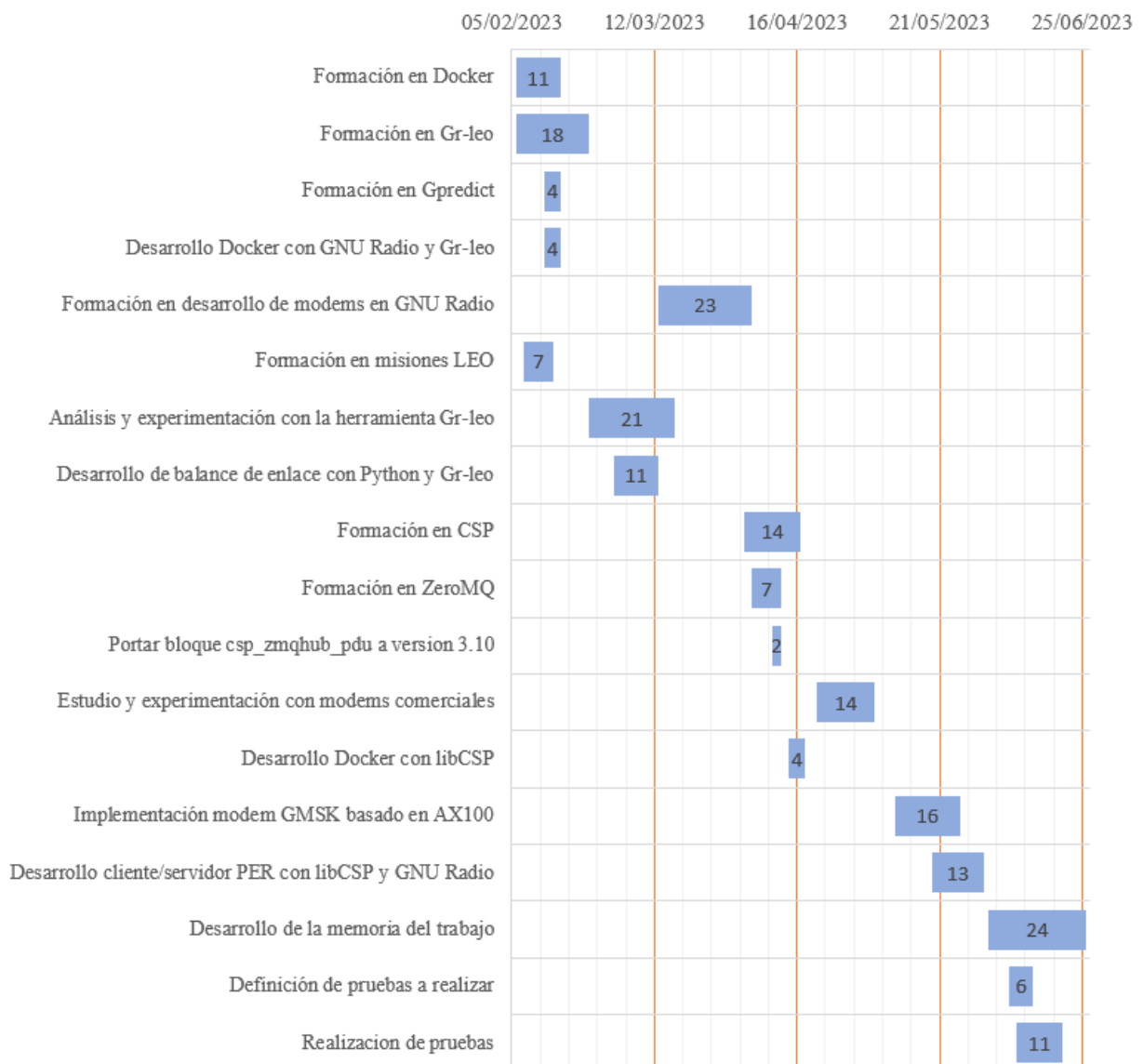


Figura 1.1.- Diagrama de Gantt de la planificación llevada a cabo durante este proyecto.

1.4.- Estructura de la memoria

Esta memoria se organiza de la siguiente forma. En primer lugar, se introduce el contexto teórico necesario acerca de las misiones de satélites de órbita baja terrestre para el correcto entendimiento del entorno que pretende simular este proyecto. Seguidamente se presentan y se analizan las herramientas que han sido utilizadas y las razones por las que han sido escogidas. Después, se explica en detalle el desarrollo llevado a cabo para la implementación de esta herramienta, así como su mismo funcionamiento. Luego, se presentan las pruebas realizadas junto con los resultados obtenidos y, por último, se exponen las conclusiones extraídas de este proyecto, así como el análisis de sus casos de uso y sus trabajos futuros.



2.- MISIONES LEO

En este capítulo se presenta el escenario típico de comunicaciones en misiones de satélites de órbita baja terrestre o LEO (Low Earth Orbit). El objetivo principal es proporcionar una visión general de los elementos clave involucrados en las comunicaciones de dichas misiones y las diferentes consideraciones que afectan al rendimiento del sistema total.

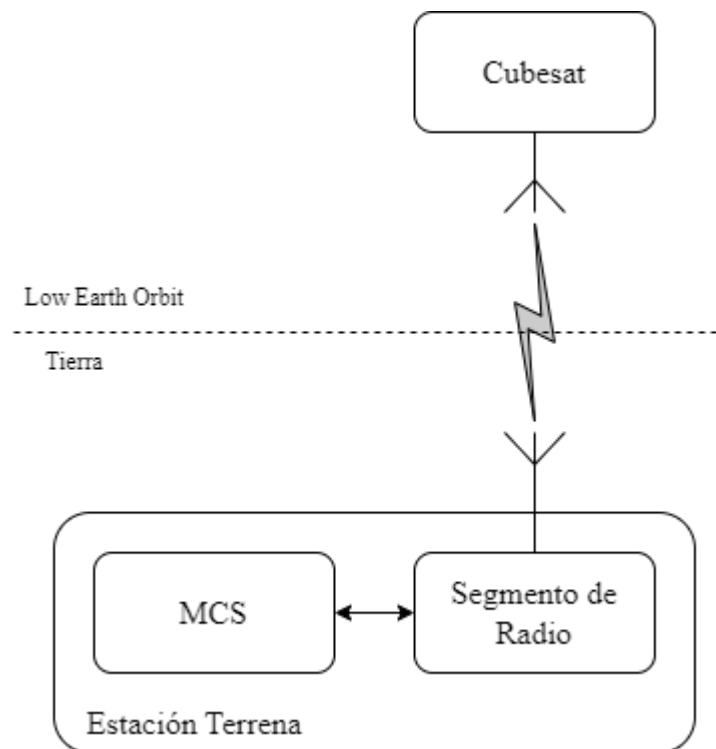


Figura 2.1.- Esquema simplificado de los elementos básicos involucrados en una misión de órbita baja terrestre o LEO.

A continuación, tras proporcionar el contexto del entorno en el que tienen lugar este tipo de misiones, los siguientes apartados describen los elementos clave presentes en el escenario de comunicaciones ilustrado en la Figura 2.1. En este esquema el MCS (Mission Control Software) es el encargado de generar los datos a transmitir por medio del segmento de radio en el enlace ascendente y, posteriormente, ser recibidos y procesados por el CubeSat. Por el otro lado, es el CubeSat el que se encarga de transmitir datos en el enlace descendente. Estos datos son recibidos y correctamente demodulados por el segmento de radio y posteriormente procesados en el MCS. Cada uno de estos elementos mencionados son explicados a continuación en los apartados siguientes. Por último, se ofrece una descripción del canal inalámbrico de comunicaciones que también está presente en este escenario, aunque no se clasifica como un elemento en sí mismo.



2.1.- Low Earth Orbit

Las órbitas bajas terrestres o LEO desempeñan un papel fundamental en las misiones de satélites pequeños, como los CubeSats. Estas órbitas se encuentran a altitudes relativamente cercanas a la Tierra, generalmente entre 200 y 2000 kilómetros, lo que permite una serie de ventajas, discutidas a continuación.

Una de las principales ventajas de las órbitas LEO es que ofrecen una menor latencia en las comunicaciones. Al estar más cerca de la Tierra, los satélites en LEO experimentan un menor retardo en las señales, del orden de entre 20 y 50 milisegundos, frente a una órbita típica geoestacionaria con retardos de entre 240 y 280 milisegundos, lo que resulta en una comunicación más rápida y eficiente con las estaciones terrestres. Esto es especialmente beneficioso para las misiones de CubeSats, ya que suelen tener requisitos de tiempo real y necesitan una comunicación constante con los controladores en tierra. Además, las órbitas LEO permiten una mayor resolución en la adquisición de imágenes y datos. Al estar más cerca de la superficie terrestre, los satélites en LEO pueden capturar imágenes con mayor detalle y precisión, lo cual es especialmente valioso en aplicaciones como la observación de la Tierra, la cartografía y la monitorización ambiental, donde la alta resolución es crucial para obtener información detallada y precisa.

Otra ventaja fundamental que presenta este tipo de orbitas es el hecho de que se encuentran por debajo del cinturón de Van Allen. El cinturón de Van Allen es una región alrededor de la Tierra que contiene partículas cargadas atrapadas por el campo magnético terrestre. Estas partículas pueden ser perjudiciales para los satélites y las misiones espaciales, por lo que evadir o minimizar la exposición a ellas es ventajoso. Algunas ventajas que supone encontrarse por debajo de este cinturón incluyen las siguientes: una reducción en la radiación sufrida a, lo que se resume en una mayor vida útil, una mayor estabilidad orbital debida a la influencia gravitacional del propio cinturón de Van Allen y una mejor cobertura con la Tierra al no sufrir las interferencias propias de este fenómeno.

Por otro lado, las órbitas LEO presentan desafíos adicionales para las misiones de CubeSats. Debido a su baja altitud, los satélites en LEO están sujetos a una mayor fricción atmosférica, lo que provoca una degradación gradual de la órbita debido a la resistencia atmosférica. Esto requiere ajustes periódicos en la órbita o incluso la reentrada controlada del satélite al final de su vida útil. Además, debido a su proximidad a la Tierra, los satélites en LEO tienen una menor duración de visibilidad durante cada órbita, lo que implica una ventana de comunicación más limitada.

En resumen, las órbitas LEO ofrecen ventajas en términos de menor latencia en las comunicaciones y mayor resolución en la adquisición de datos. Sin embargo, también presentan desafíos como la degradación orbital debido a la fricción atmosférica y una menor duración de visibilidad durante cada órbita. Estos factores deben ser considerados y gestionados adecuadamente en el diseño y operación de misiones de CubeSats en órbitas bajas terrestres.



2.2.- Estación terrena

Las estaciones terrenas juegan un papel fundamental en las misiones de satélites de órbita baja (LEO). Estas estaciones son centros de comunicaciones que establecen un enlace bidireccional entre los satélites en órbita y los equipos de control, monitorización y operación en la Tierra.

En cuanto a sus funcionalidades, en primer lugar, las estaciones terrenas permiten el seguimiento constante de los satélites de órbita baja. Típicamente el satélite solo pasa unas pocas veces al día por una misma localidad donde se encuentre la estación, por lo que se hace indispensable rastrear la posición y la trayectoria de los satélites, basándose en modelos de predicción y simulación de órbitas. Esto permite a los equipos de control calcular con precisión su ubicación y programar las maniobras necesarias para optimizar la directividad y ganancia de las antenas utilizadas apuntando correctamente a la órbita deseada. Estas maniobras son ejecutadas por un equipo de motores y rotores colocados en las antenas de transmisión y recepción de la propia estación. Nótese que en el esquema de la Figura 2.1 se ha obviado esta parte de control de apuntamiento ya que, al tratarse de un proyecto de simulación, no se ha considerado relevante para el análisis y evaluación del comportamiento de un subsistema de comunicaciones.

En segundo lugar, las estaciones terrenas son responsables de controlar y monitorear los satélites en tiempo real. A través de los enlaces de comunicaciones establecidos, las estaciones terrenas envían comandos a los satélites para realizar diversas operaciones, como ajustar la orientación, activar o desactivar instrumentos o llevar a cabo maniobras orbitales. Además, las estaciones reciben la telemetría del satélite, que contiene información sobre su estado y rendimiento, lo que permite a los equipos de control evaluar su salud.

Las estaciones terrenas también desempeñan un papel vital en la transferencia de datos entre los satélites LEO y las operaciones terrestres. A medida que los satélites recopilan información propia del cumplimiento de la misión, es decir, la payload o carga útil, como imágenes de la Tierra o datos científicos, necesitan enviarlos a las estaciones terrenas para su procesamiento y distribución. Del mismo modo, las estaciones terrenas transmiten comandos y actualizaciones de software a los satélites. Estos enlaces de comunicación bidireccionales son esenciales para el intercambio de datos críticos durante las misiones de satélites de órbita baja.

Por otro lado, el diseño de la estación terrena para una misión para satélites pequeños de órbita baja requiere del estudio previo de varios factores: las frecuencias de comunicaciones, el tipo de órbita, la tasa de transmisión de los datos, el tipo de modulación de las señales y la potencia de transmisión del satélite. Debido a sus limitaciones de tamaño, peso y potencia, los CubeSats, sobre todo los del tipo 1U, a veces no incluyen un ADCS (Attitude Determination and Control System), por lo que, buscando asegurar la transmisión de la señal en dirección a la estación terrena, típicamente se utilizan dipolos, antenas de tipo parche o de tipo Turnstile, con patrones de radiación que pretenden ser lo



más omnidireccionales posibles. Esto, aunado a la poca energía que pueden captar en la limitada superficie disponible para celdas solares, hace que la potencia de transmisión sea en la mayoría de los casos del orden de 1W. Asimismo, típicamente los CubeSats operan en las frecuencias de radioaficionados en las bandas VHF (Very High Frequency) y UHF (Ultra High Frequency) del espectro electromagnético y emplean tasas de transmisión que no suelen superar los 100 kbps. En estas condiciones es necesario hacer los cálculos de enlace para garantizar la adecuada comunicación con el satélite y caracterizar los componentes de la estación terrena.

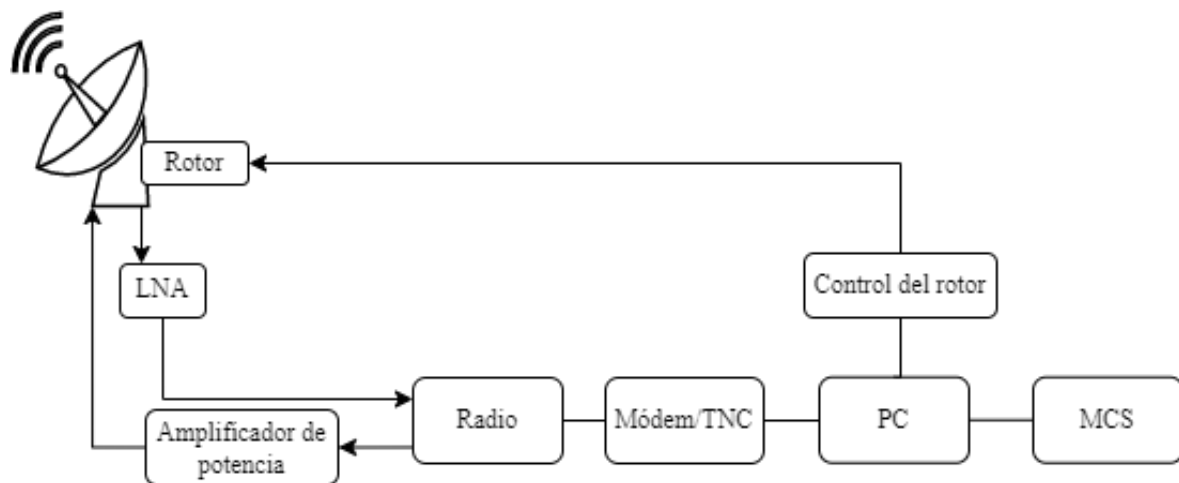


Figura 2.2.- Esquema simplificado de los elementos básicos que forman una estación terrena.

En la Figura 2.2 se muestra un esquema de configuración y diseño básico de una estación terrena. En esta aparecen los componentes básicos, donde se incluyen antenas, amplificadores de bajo ruido o LNA (Low Noise Amplifier), rotores con control de azimut y elevación, sistemas de control de los rotores, módems o TNCs (Terminal Node Controller), sistemas de radio, amplificadores de potencia, ordenadores, software para predecir la posición orbital del satélite y MCS. Estos elementos desempeñan roles fundamentales en el cumplimiento de las funcionalidades previamente mencionadas al principio de este apartado.

Existen muchas alternativas al esquemático básico mostrado en función de la misión que se desea desempeñar. De entre estos componentes, merece la pena destacar el módem o TNC ya que juega un papel muy importante en este proyecto a la hora de simular las comunicaciones entre estación y satélite. Típicamente este elemento ha sido desarrollado como un circuito de hardware dedicado a cumplir los requisitos de la misión únicamente. No obstante, con el auge del desarrollo digital y de software en las últimas décadas, existe la posibilidad de la implementación de módems basados en el paradigma conocido como SDR (Software Defined Radio). El empleo de este tipo de tecnología permite emplear algoritmos de procesamiento digital de comunicaciones para desarrollar módems por



medio de una CPU, como puede ser la de un ordenador, gracias a la conversión analógico a digital y viceversa de las señales recibidas o a transmitir. Cabe destacar que este tipo de tecnología ofrece una gran flexibilidad al poder rediseñar en cualquier momento mediante software el procesado de comunicaciones empleado para transmitir y recibir la información. No obstante, este tema será tratado con más profundidad en capítulos posteriores.

2.2.1.- Mission Control Software

El MCS es una pieza clave en el contexto de misiones de satélites CubeSat en órbita baja como parte de una estación terrena. Una forma interesante de ver este sistema es que el resto de los elementos que componen la estación terrena actúan como una interfaz entre el MCS y el satélite en órbita, permitiendo el control y la supervisión de la misión desde tierra.

Un MCS típicamente incluye varios componentes clave para el funcionamiento de la misión los cuales se listan a continuación.

- OCS (Operations Control System), el cual abarca las operaciones de vuelo, las operaciones en tierra y el control y análisis del rendimiento de la misión. El OCS también se encarga del control sobre la configuración del sistema total, asegurando que todos los aspectos operativos se lleven a cabo de manera eficiente y efectiva.
- FDS (Flight Dynamics System), el cual se encarga de la predicción de los elementos orbitales del satélite, la predicción de las ventanas de adquisición de señal o AOS (Acquisition Of Signal), la planificación de maniobras y la especificación de las diferentes etapas operacionales de la misión. El FDS permite realizar un seguimiento preciso de la posición y la trayectoria del satélite, asegurando un control adecuado durante la misión.
- PCS (Payload Control System). Este sistema se encarga de la planificación, control y monitorización de los elementos involucrados en la carga útil del satélite, así como de la evaluación de su rendimiento. Este componente garantiza el correcto funcionamiento y la obtención de datos de la carga útil, que pueden ser cruciales para el éxito de la misión.
- Sistema de distribución de datos que actúa como un repositorio centralizado de todos los datos recolectados durante la misión. Esto permite un acceso fácil y eficiente a los datos para su análisis y evaluación. Por último, el MCS también cuenta con una base de datos que almacena y define todos los datos relevantes de la misión, proporcionando una referencia centralizada para la configuración y el seguimiento de la misión.

Como se ha explicado, el MCS es un sistema complejo y consta de diferentes sistemas. No obstante, se ha simplificado cuantitativamente para la realización de este



proyecto. Así pues, en capítulos posteriores se presentará como un mero programa que actuará como cliente para simular el envío de paquetes de datos cuando sea el transmisor y otro que actuará como servidor para simular el procesado de los datos recibidos cuando sea el receptor.

2.3.- CubeSat

Como tercer elemento del esquema inicial básico de los componentes de una misión LEO está el CubeSat. Los CubeSats, o nanosatélites, son una categoría de satélites pequeños y estandarizados que han ganado popularidad en la industria espacial. Estos satélites tienen una forma cúbica de 10 centímetros de lado y se componen de diferentes subsistemas que, en combinación entre sí, permiten realizar diversas misiones espaciales. Un esquema de los subsistemas básicos que se pueden encontrar en casi cualquier nanosatélite aparece en la Figura 2.3, no obstante, dependiendo de la misión, algunos de estos subsistemas podrían ser prescindibles o ser necesario alguno no mencionado.

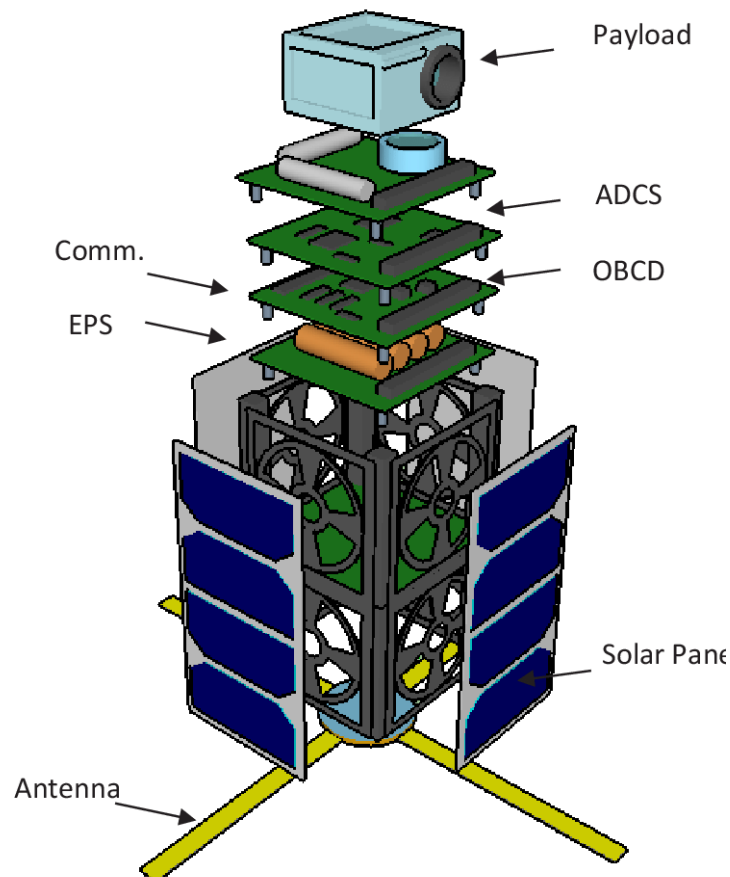


Figura 2.3.- Esquema de los subsistemas básicos que forman parte de un CubeSat [2].



Un subsistema importante propio de estos satélites es el sistema de energía, típicamente denominado como EPS (Electrical Power System), que proporciona la electricidad necesaria para el funcionamiento del satélite. Generalmente, utilizan paneles solares que capturan la energía solar y la almacenan en baterías internas. Esta solución energética es eficiente y sostenible, lo que permite una larga vida útil en el espacio.

Además, muchos CubeSats cuentan con un subsistema de control de actitud o ADCS, que se encarga de mantener la orientación y estabilidad del satélite en el espacio. Esto se logra mediante el uso de sensores y actuadores que ajustan la posición y la velocidad de rotación del satélite. No obstante, nótese que este elemento no es primordial en todo tipo de misiones. Imagínese el lector una misión en la que la orientación del satélite no sea un factor clave y, la antena de comunicaciones que se utilice sea omnidireccional. Entonces, podría utilizarse su espacio sobrante para incorporar algún otro elemento necesario para el correcto cumplimiento de la misión.

También tienen un ordenador de a bordo u OBC (Onboard Computer), que son sistemas informáticos compactos y con bajo consumo de energía diseñados para funcionar en el entorno espacial. Este tipo de dispositivos controla y gestiona todas las operaciones y funciones del satélite, desde la recopilación de datos hasta la transmisión de información. Su capacidad de cómputo y procesamiento puede variar según el tamaño y la complejidad de la misión. Algunos nanosatélites más pequeños pueden tener computadoras con capacidades limitadas, mientras que otros pueden contar con sistemas más avanzados capaces de ejecutar algoritmos sofisticados y realizar tareas de procesamiento de datos más complejas.

Por otro lado, la payload o carga útil de un CubeSat se refiere al conjunto de instrumentos, sensores o experimentos científicos que se llevan a cabo a bordo del satélite con el propósito de cumplir con los objetivos específicos de la misión. La carga útil es la parte central y significativa del CubeSat, ya que determina la naturaleza y el alcance de los datos que se recopilarán y se transmitirán desde el espacio. La payload de un CubeSat puede variar ampliamente según la finalidad de la misión. Puede incluir instrumentos para la observación de la Tierra, como cámaras o espectrómetros, o puede contener experimentos científicos para investigar fenómenos espaciales, como el comportamiento de materiales en el entorno de microgravedad, el estudio de la radiación o la detección de partículas cósmicas.

Al igual que sucedía con las estaciones terrenas, para el desarrollo de la payload, en el caso de tratarse esta de una payload de comunicaciones también se puede emplear el paradigma de la tecnología SDR. Esto juega un papel aún más importante en este segmento espacial al suministrar, de la misma forma, una flexibilidad a la hora de adaptar y optimizar las comunicaciones incluso mientras el satélite se encuentra en órbita, cosa que con un circuito de comunicaciones dedicado sería prácticamente imposible si no hubiese sido diseñado previo al lanzamiento.



Por último, uno de los subsistemas más importantes, así como uno de los elementos protagonista de este proyecto, es el sistema de comunicaciones, que permite la transmisión y recepción de datos con las estaciones terrestres. Estos subsistemas son vitales para establecer enlaces de comunicación con la estación terrestre y permitir la transferencia de información entre el CubeSat y los operadores en tierra.

El subsistema de comunicaciones de un CubeSat, típicamente conocido como TTC (Telemetry Tracking & Command) compuesto de varios componentes clave. En primer lugar, está el transceptor de radio, que es responsable de la transmisión y recepción de señales de radio. Los transceptores de los CubeSats suelen operar en frecuencias específicas, como las bandas UHF o VHF, aunque recientemente se empieza a emplear comunicaciones en banda S para transmisiones de grandes cantidades de datos y grandes tasas de velocidad de datos. No obstante, éstos deben cumplir con las regulaciones y requisitos del espectro radioeléctrico. Cabe destacar que para el caso de los TTCs no es común emplear la tecnología SDR ya que estas requieren de un alto consumo.

Además del transceptor de radio, estos satélites están equipados con antenas, que son cruciales para la propagación de las señales de radio. Estas antenas pueden variar en tamaño y configuración según los requisitos de la misión y las limitaciones de espacio. Algunos pueden disponer de antenas desplegadas para maximizar la eficiencia de las comunicaciones. No obstante, como ya se comentó anteriormente, estas antenas suelen ser de tipo parche, o agrupaciones de dipolos o de tipo Turnstile.

Una de las principales ventajas de los CubeSats es su tamaño compacto y su bajo costo de desarrollo y lanzamiento. Esto ha permitido que un mayor número de organizaciones y empresas puedan acceder al espacio y llevar a cabo misiones espaciales a un costo mucho más asequible en comparación con los satélites convencionales. Además, los CubeSats son altamente modulares y pueden ser desplegados en constelaciones, lo que aumenta la cobertura y la eficiencia en diversas aplicaciones, como la observación de la Tierra, la investigación científica y la comunicación global.

Sin embargo, también presentan algunas limitaciones. Debido a su tamaño reducido, tienen una capacidad de carga útil limitada y no pueden llevar grandes instrumentos científicos o sistemas de comunicación de alta potencia. Además, su vida útil puede verse afectada por factores como la radiación espacial y el desgaste mecánico, lo que implica que su funcionamiento sea limitado en comparación con los satélites de mayor tamaño. No obstante, la vida útil de este tipo de satélites suele venir limitada por el lanzador, que suele ser de unos 25 años, aunque depende del lanzador y previamente exigen pruebas de cómo se asegura cumplir que pasado ese tiempo el satélite reentrará en la atmósfera, ya sea mediante un sistema de reentrada o por cálculos y simulaciones del rozamiento sufrido durante ese tiempo.

En resumen, los CubeSats son una solución innovadora y económica de realizar misiones espaciales. Sus subsistemas principales, como el de comunicaciones, de energía, de control de actitud y el ordenador de a bordo, les permiten llevar a cabo diversas

funciones en el espacio. Aunque tienen algunas limitaciones en términos de capacidad y vida útil, estos dispositivos han revolucionado la forma en que accedemos y utilizamos el espacio, abriendo nuevas posibilidades en la exploración y la monitorización de nuestro planeta y más allá.

2.4.- Canal de comunicaciones

En este último apartado se presenta y se estudia el canal de comunicaciones propio de las misiones de satélites de órbita baja terrestre (LEO). Para realizar este estudio se partirá del escenario presentado en la Figura 2.4. De este escenario se deducen varias afirmaciones que serán tomadas de esta forma para todo el estudio.

En primer lugar, el enlace de subida y de bajada se tomarán como se muestra en la imagen, es decir, el enlace de subida será la transmisión desde la estación base hacia el satélite y, por tanto, el enlace de bajada será la transmisión de datos del satélite hacia la estación base. Por otro lado, se tomará como la distancia del enlace la línea directa que existe entre la estación y el satélite, siendo referida a esta como rango oblicuo.

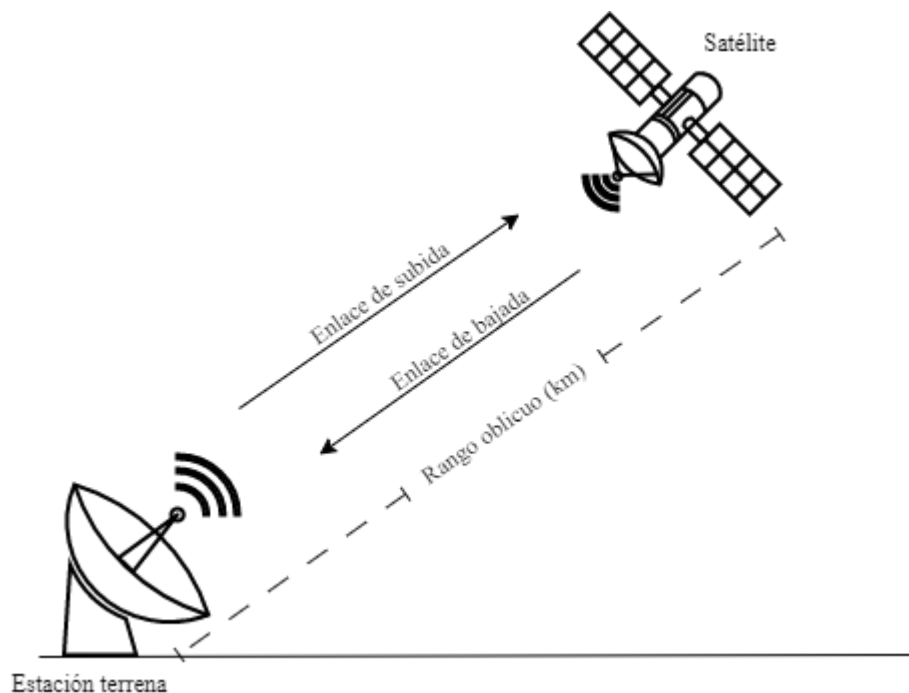


Figura 2.4.- Escenario básico de un canal de comunicaciones para misiones de órbita baja terrestre o LEO.

En cuanto a las perturbaciones que aporta el canal a la señal en su transmisión de un punto a otro, se pueden clasificar de manera simplificada en dos tipos: desplazamiento en frecuencia y pérdidas de propagación.



El primer tipo de perturbación hace referencia al desplazamiento en frecuencia que experimenta la señal en consecuencia al movimiento relativo, ya sea de la estación terrena desde la perspectiva del satélite en el enlace de subida, o del propio satélite desde el punto de vista de la estación terrena durante el enlace de bajada. A este fenómeno se le conoce como efecto Doppler y dicho desplazamiento en frecuencia viene descrito por la Ecuación 2.1, donde Δv es la velocidad del receptor respecto a la de la fuente, c es la velocidad de la luz y f_c es la frecuencia de portadora de la señal transmitida.

$$\Delta f = \frac{\Delta v}{c} \cdot f_c \quad (2.1)$$

Este desplazamiento depende directamente de la velocidad relativa entre ambos, transmisor y receptor. Por tanto, al ser las velocidades alcanzadas por las órbitas de este tipo de satélites del orden de los 8 km/s, se llegan a alcanzar desplazamientos del orden de los 10 KHz en su frecuencia portadora, cantidad que no es despreciable. No obstante, haciendo uso de sistemas de predicción orbitales mencionados en los apartados anteriores, se puede calcular dichos desplazamientos y corregirlos en Tierra mediante la capacidad de cómputo de la estación terrena de forma relativamente sencilla.

El segundo tipo de perturbación hace referencia a las pérdidas de potencia que sufre la señal transmitida al propagarse hasta la antena receptora. Este tipo de pérdidas pueden clasificarse en los siguientes tipos: de propagación por espacio libre, por los gases atmosféricos, por lluvia, por apuntamiento y por polarización.

Las pérdidas por propagación en espacio libre, que es el tipo fundamental de atenuación de señal, asumen que la trayectoria de la onda está completamente despejada y en el vacío. [3] proporciona un método para calcularlas determinado únicamente por la frecuencia de operación f (GHz) y la distancia d (km) entre el satélite y la estación terrestre basada en la Tierra. De esta forma, se pueden calcular dichas pérdidas según la fórmula de la Ecuación (2.2).

$$L_{fs} = 92.45 + 20 \log_{10}(f[\text{GHz}] \cdot d[\text{km}]) \quad (2.2)$$

Por otro lado, la transmisión de señales a través del canal de telecomunicaciones tierra-espacio también experimenta una reducción en la intensidad debido a la absorción de gases atmosféricos. La magnitud de esta absorción depende del arreglo de moléculas presentes a lo largo de la trayectoria que une el satélite y la estación terrena, y su efecto no es notable para frecuencias que caen por debajo de 2 GHz. El método de cálculo de estas pérdidas viene recogido en [4].

En lo que respecta al impacto de la precipitación en la pérdida de señal, representa una preocupación significativa, especialmente para frecuencias que operan por encima de



los 5 GHz. El problema de cálculo de este tipo de atenuación puede ser abordado utilizando el enfoque especificado en [5], que estima la degradación de la amplitud de la señal causada por fuertes lluvias a lo largo del trayecto de la transmisión. Este método depende de varios parámetros, incluyendo la tasa de lluvia puntual respecto a la ubicación actual para el 0.01% de un año promedio (R_{001} , medida en mm/h), la altura de la estación terrena sobre el nivel medio del mar (h_s , medida en km), el ángulo de elevación del satélite (θ , medido en grados), la latitud de la estación terrena (ϕ , medida en grados) y la frecuencia de operación (f , medida en GHz). Por otro lado, si no se dispone de datos locales disponibles, se puede utilizar el mapa digital proporcionado en [6].

Las pérdidas de apuntamiento hacen referencia a la reducción en la intensidad de la señal que puede ocurrir cuando la dirección de la antena del satélite no está alineada con precisión con la antena de la estación terrena. Esto puede llevar a una reducción en la calidad y la intensidad de la señal recibida, lo que afecta en última instancia el rendimiento general del sistema de comunicación. La pérdida de apuntamiento puede ser influenciada por varios factores, como la estabilidad mecánica de la antena, la precisión del sistema de seguimiento o incluso las condiciones atmosféricas.

Por último, cabe mencionar las pérdidas de polarización. Estas pérdidas ocurren debido a la variación en la orientación de la polarización de las ondas electromagnéticas a medida que viajan a través de la atmósfera y se propagan hacia el satélite o en sentido inverso. Los principales factores que contribuyen a las pérdidas de polarización en este tipo de canal son la dispersión de los rayos atmosféricos y las interacciones con partículas en suspensión, como gotas de agua o moléculas de polvo. Estas perturbaciones pueden alterar la polarización original de la señal transmitida, lo que resulta en una degradación de la calidad de la comunicación. Para el valor de estas pérdidas suele considerarse 3 dB.

2.4.1.- Balance de enlace

En esta subsección, se profundiza en el balance de enlace propio de un escenario de comunicaciones de una misión de satélites de órbita baja terrestre en banda UHF. Para esto se proporciona un ejemplo tanto para el enlace ascendente, como para el enlace descendente.

Tabla 2.1.- Datos orbitales arbitrarios para el balance de enlace.

ELEMENTO ORBITAL	VALOR	UNIDAD
APOGEO	550	km
PERIGEO	550	km
ELEVACIÓN	10	°



Los datos orbitales necesarios aparecen en la Tabla 2.1. Mediante estos datos facilitados es posible calcular el rango oblicuo mencionado previamente en este apartado. Cabe remarcar previamente que, para un estudio fiable en un enlace de comunicaciones Tierra-espacio se requiere del peor de los casos mientras haya línea de visión directa, esto es, la mayor distancia a la que se puede encontrar el satélite de la estación terrena, que sucede al aparecer por el horizonte. Así pues, el elemento “elevación” hace referencia a los grados por encima del horizonte a partir de los cuales se considera que hay línea de visión directa con el satélite para establecer el comienzo de la comunicación en esa pasada.

$$d = R_T \left[\sqrt{\left(\frac{H+R_T}{R_T}\right)^2 - \cos^2 \varepsilon_0} - \sin \varepsilon_0 \right] \quad (2.3)$$

De esta forma, se puede obtener el rango oblicuo d según la fórmula de la Ecuación (2.3), donde R_T es el radio de la Tierra, con un valor de 6,378 km, H es la altura del satélite en km, que corresponde al mayor valor entre el perigeo y el apogeo de la órbita y ε_0 es la elevación medida en grados. Este método de cálculo de la máxima distancia puede obtenerse también a partir de la Figura 2.5, donde se puede identificar de manera mejor ilustrada algunos de los parámetros empleados en la fórmula. No obstante, el desarrollo geométrico de la misma puede ser consultada en [7].

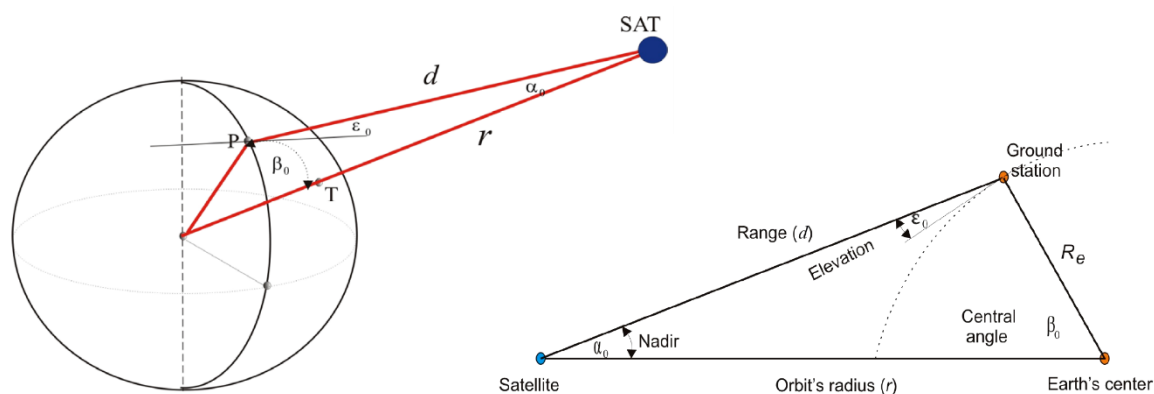


Figura 2.5.- Método geométrico del cálculo del rango oblicuo máximo entre la estación terrena y el CubeSat para una determinada elevación mínima en el horizonte. [7]

Los datos de comunicaciones utilizados para los enlaces ascendente y descendente aparecen respectivamente en la Tablas 2.2 y 2.3. Los valores de las frecuencias seleccionadas para este ejemplo han sido escogidos acorde con [8] correspondiendo ambas frecuencias con las bandas atribuidas a la región europea para el servicio de operaciones espaciales. Por otro lado, el resto de los valores correspondientes a las características del esquema de modulación empleado, codificación de canal, tasa de velocidad de datos y ancho de banda, así como los valores requeridos de BER (Bit Error Rate) y SNR (Signal to



Noise Ratio) han sido seleccionados como datos típicos en escenarios de comunicaciones con satélites pequeños de órbita baja terrestre [9]. También se incluye un margen de seguridad del sistema de 3 dB como recomienda la UIT (Unión Internacional de Telecomunicaciones).

Tabla 2.2.- Datos utilizados para el enlace ascendente del balance de enlace.

ENLACE ASCENDENTE	VALOR	UNIDAD
FRECUENCIA	401.5	MHz
PÉRDIDAS POLARIZACIÓN	0	dB
PÉRDIDAS POR APUNTAMIENTO	0	dB
MODULACIÓN	GMSK (BT = 0.3)	N/A
FEC	Cód. Convolutivos (R=1/2, K=7) + Reed-Solomon (255,223)	N/A
TASA DE DATOS	9600	bps
ANCHO DE BANDA	12480	Hz
BER REQUERIDA	10^{-5}	N/A
SNR REQUERIDA	7.8	dB
MARGEN DEL SISTEMA	3	dB

Tabla 2.3.- Datos utilizados para el enlace descendente del balance de enlace.

ENLACE DESCENDENTE	VALOR	UNIDAD
FRECUENCIA	400.2	MHz
PÉRDIDAS POLARIZACIÓN	0	dB
PÉRDIDAS POR APUNTAMIENTO	0.5	dB
MODULACIÓN	GMSK (BT = 0.3)	N/A
FEC	Cód. Convolutivos (R=1/2, K=7) + Reed-Solomon (255,223)	N/A
TASA DE DATOS	9600	bps
ANCHO DE BANDA	12480	Hz



BER REQUERIDA	10^{-5}	N/A
SNR REQUERIDA	7.8	dB
MARGEN DEL SISTEMA	3	dB

En las tablas 2.4 y 2.5, en cambio, se encuentran los datos propios de la estación terrena y del satélite respectivamente que son necesarios para llevar a cabo el balance de enlace correctamente. El valor de la potencia de transmisión de la estación terrena se ha escogido acorde con los reglamentos de potencia radiada determinados en [10] y también respecto a valores propios de este tipo de enlace [9]. Las pérdidas introducidas por las líneas de transmisión son significativamente superiores en la estación terrena con respecto a las del satélite debido a que suele requerir un despliegue de una longitud de línea del orden de entre 20 y 30 metros en total en toda la instalación [11]. También se incluyen otras pérdidas que pueden existir entre conectores y elementos del sistema.

El valor de la ganancia de la antena de transmisión para el enlace ascendente, y de recepción para el descendente, es decir, la antena de la estación terrena ha sido escogido acorde con los valores de ganancia propios del tipo de antena típicamente utilizado para este cometido. Este tipo de antenas suelen ser del tipo parabólicas, X-Quad o Yagi-Uda y son capaces de alcanzar altos valores de ganancia del orden de entre 10 y 20 dBi [12]. En cambio, respecto a las antenas típicas utilizadas en los satélites pequeños o CubeSats, típicamente tienen una ganancia de 0 dBi y se suelen utilizar antenas de parche o de tipo dipolo cruzado o Turnstile, y no suelen soportar más de 2 W de potencia [13]. Por último, la temperatura de ruido efectiva del receptor se ha seleccionado con un valor típico [9].

Tabla 2.4.- Datos de la estación terrena utilizados para el balance de enlace.

ESTACIÓN TERRENA	VALOR	UNIDAD
POTENCIA DISPONIBLE DE TX	16	dBW
PÉRDIDAS DE LÍNEA DE TRANSMISIÓN	5	dB
OTRAS PÉRDIDAS	1	dB
GANANCIA DE ANTENA	14.95	dBi
TEMPERATURA EFECTIVA DE RUIDO DEL RECEPTOR	1003	K



Tabla 2.5.- Datos del satélite utilizados para el balance de enlace.

SATÉLITE	VALOR	UNIDAD
POTENCIA DISPONIBLE DE TX	0	dBW
PÉRDIDAS DE LÍNEA DE TRANSMISIÓN	0.2	dB
OTRAS PÉRDIDAS	0.3	dB
GANANCIA DE ANTENA	0	dBi
TEMPERATURA EFECTIVA DE RUIDO DEL RECEPTOR	234	K

Finalmente, en la Tabla 2.6 se muestran los resultados del balance de enlace para ambos enlaces, ascendente y descendente. Es importante remarcar que, para todo el cálculo de este balance de enlace se ha supuesto una óptima adaptación de las antenas y líneas de transmisión tanto de la estación terrena como del satélite.

En cuanto a dichos resultados, en primer lugar, las pérdidas, tanto de espacio libre, como por gases, precipitación, apuntamiento y de polarización se calculan en base a los métodos previamente explicados al principio de la sección. No obstante, se añaden algunos tipos de pérdidas extra como son las de polarización e incluso un apartado extra de otras pérdidas. De forma análoga, el parámetro de distancia máximo ha sido calculado mediante el método expuesto según la Ecuación 2.3.

El parámetro PIRE (Potencia Isotrópica Radiada Equivalente) representa la potencia radiada por la antena. Se calcula sumando la potencia disponible del transmisor en dBW más la ganancia de la antena y restándole las pérdidas internas del sistema, esto es, las de la estación terrena y las del satélite. Por otro lado, el nivel de potencia isotrópico recibido simboliza la potencia de la señal recibida en la antena receptora que, tras sumarle la ganancia de esta última, se obtiene el nivel de potencia recibida en dBW.

La figura de mérito G/T es un factor que evalúa la relación entre la ganancia de una antena y la temperatura efectiva de ruido del sistema receptor. Según dicha definición, entonces, un valor más alto indica una mejor relación señal-ruido en el sistema de recepción.

Tabla 2.6.- Resultados del balance de enlace para ambos enlace ascendente y descendente.

BALANCE DE ENLACE	ASCENDENTE	DESCENDENTE	UNIDAD
FRECUENCIA	401.5	400.2	MHz
DISTANCIA MÁXIMA	1815.08	1815.08	km



PÉRDIDAS DE ESPACIO LIBRE	149.6994	149.6712	dB
PÉRDIDAS POR PRECIPITACIÓN	0	0	dB
PÉRDIDAS POR GASES ATMOSFÉRICOS	0	0	dB
PÉRDIDAS POR POLARIZACIÓN	0	0	dB
PÉRDIDAS POR APUNTAMIENTO	0	3	dB
OTRAS PÉRDIDAS	0	0	dB
PÉRDIDAS TOTALES DE LA TRAYECTORIA	149.6994	152.6712	dB
PIRE	24.95	-0.5	dBW
NIVEL RECIBIDO ISOTRÓPICO	-124.7494	-153.1712	dBW
FIGURA DE MÉRITO G/T	-23.6921	-15.063	dB
POTENCIA DE SEÑAL RECIBIDA	-124.7494	-138.2212	dBW
POTENCIA DE RUIDO RECIBIDA	-163.9449	-157.624	dBW
RELACION SEÑAL-RUIDO GAUSSIANO (SN0)	80.1576	60.3649	dBHz
RELACIÓN SEÑAL-RUIDO EN EL RECEPTOR (SNR)	36.1954	16.4027	dB
RELACION SEÑAL-RUIDO POR BIT (EB/N0)	37.3348	17.5422	dB

La potencia isotrópica de la señal recibida se calcula sumando la ganancia de la antena de transmisión en el enlace correspondiente con la ganancia de la antena y restando las pérdidas totales resultantes de la fórmula de Friis. Adicionalmente, si a esta le sumamos la ganancia de la antena receptora del enlace correspondiente obtenemos la potencia de la señal recibida en bornes de la antena receptora.

Por otro lado, la potencia de ruido recibida se calcula multiplicando la temperatura de ruido equivalente por el ancho de banda de la señal a recibir y la constante de Boltzman en sus unidades naturales. No obstante, nótese que en la Tabla 2.6 se encuentra en dBW habiendo hecho la conversión pertinente.

Por último, la relación señal-ruido (SNR) se obtiene como la resta de la potencia de la señal recibida menos la potencia de ruido del receptor. En cambio, la SN0 es la relación señal-ruido gaussiano y se calcula de la misma forma, pero sumando el ancho de banda del



enlace en dBHz. El último parámetro en cambio representa la relación señal-ruido por bit y se calcula respecto a la tasa de datos en vez del ancho de banda del enlace.

Nótese que en el Anexo A se encuentra un script de Python mediante el cual se implementa el balance de enlace estudiado y será utilizado más adelante en el capítulo de pruebas del proyecto.



3.- HERRAMIENTAS EMPLEADAS

En este capítulo se presentan las herramientas que han sido utilizadas para el correcto desarrollo de este proyecto. A continuación, se facilita el conocimiento y contexto mínimo para entender la finalidad y utilidad de cada herramienta seleccionada. Asimismo, para una comprensión inicial de la relación que existe entre cada una de las herramientas, en la Figura 3.1 se presenta un diagrama que relaciona las herramientas entre sí.

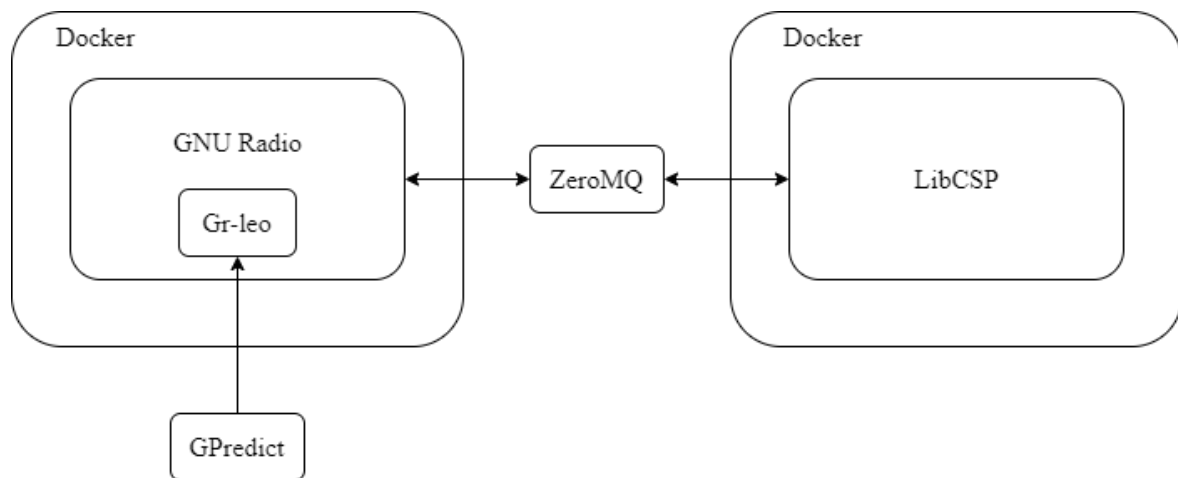


Figura 3.1.- Esquema de la relación entre las herramientas empleadas en el proyecto.

En el diagrama se presenta un total de 6 herramientas: GPredict, Docker, GNU Radio, gr-leo, LibCSP y ZeroMQ. Se muestra principalmente el uso independiente de dos contenedores Docker. El que aparece representado a la izquierda contiene la herramienta GNU Radio, donde, a su vez, se ha instalado el módulo gr-leo, responsable de todo el procesamiento de señal propio de la simulación de canal de comunicaciones para misiones LEO. A su vez, este módulo se ve alimentado por información proveniente de la herramienta de seguimiento de satélites GPredict. Por otro lado, el otro contenedor Docker contiene instalada la librería LibCSP, propia del protocolo de comunicaciones para CubeSats, CSP (CubeSat Space Protocol). Por medio de esta librería se simulan los nodos transmisores y receptores, actuando como CubeSat y como MCS dependiendo del tipo de enlace especificado previo a la simulación (ascendente o descendente). Finalmente, se utiliza ZeroMQ como una pasarela de comunicaciones a nivel de red local para facilitar la transferencia de información entre el simulador de canal y la librería CSP. ZeroMQ permite enviar la información a transmitir desde la librería CSP al simulador de canal, así como recibir la información que ha pasado por el canal y devolverla al nodo receptor implementado también por la librería CSP.



3.1.- GPredict

GPredict [14] es una herramienta de seguimiento de satélites de código abierto que se utiliza para rastrear y predecir la posición de los satélites en tiempo real. Está disponible para varias plataformas, incluyendo Linux, Mac y Windows, y se puede utilizar para una variedad de tareas de seguimiento y comunicaciones de satélites. GPredict utiliza una base de datos de elementos orbitales TLE (Two Line Element) para determinar la posición y trayectoria de los satélites en tiempo real.

La herramienta también incluye una interfaz gráfica de usuario fácil de usar que muestra información detallada sobre la posición actual del satélite, incluyendo su ubicación en un mapa interactivo y su elevación actual sobre el horizonte. Esto se muestra en la Figura 3.2. Además, GPredict se puede integrar con radios y antenas de seguimiento para permitir la recepción y decodificación de señales de satélite en tiempo real.

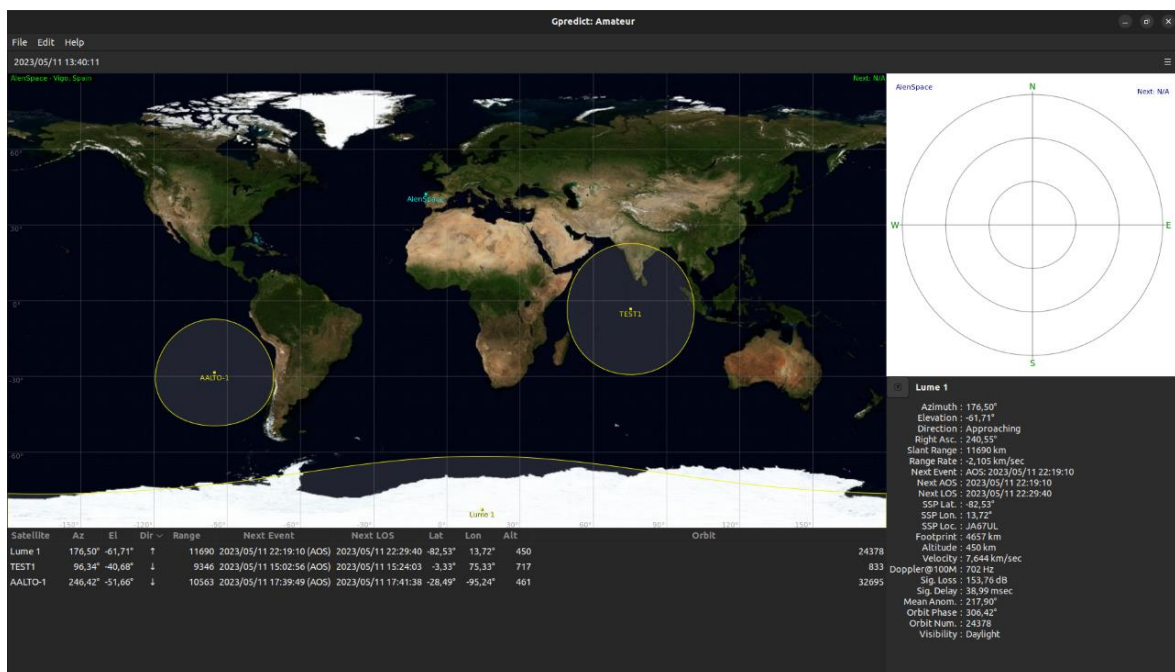


Figura 3.2.- Captura de pantalla de la herramienta GPredict mostrando su interfaz gráfica con el seguimiento e información orbital de dos satélites.

GPredict ha sido utilizada a lo largo de este proyecto para la adquisición de datos orbitales precisos de satélites como sus TLE y ventanas de AOS, gracias a su base de datos actualizada semanalmente. Se puede utilizar también para planificar y ejecutar operaciones de comunicaciones de satélite, así como para la recopilación de datos científicos y telemétricos.



3.2.- Docker

En este apartado se introduce la herramienta Docker [15] junto con los conceptos claves que han sido necesarios para el desarrollo de este proyecto. Por otro lado, también se dan las razones por las cuales se ha escogido esta opción de desarrollo frente a otras como podría haber sido el uso de máquinas virtuales.

3.2.1.- Introducción

Docker es una herramienta de código abierto orientada a la gestión de contenedores, donde un contenedor no es más que una instancia de una imagen, que se ha convertido en una de las herramientas principales utilizadas para desarrollar y distribuir aplicaciones modernas. Es una plataforma de software que permite a los desarrolladores crear, ejecutar y distribuir aplicaciones de forma rápida y sencilla, independientemente del entorno en el que se ejecuten. Docker utiliza contenedores para empaquetar aplicaciones y sus dependencias en un paquete portátil que se puede ejecutar en cualquier entorno.

La tecnología de contenedores de Docker se basa en la idea de aislar aplicaciones y servicios en un contenedor, lo que significa que cada contenedor tiene su propio sistema de archivos, red y recursos de computación. Esto permite que múltiples contenedores se ejecuten en la misma máquina física sin interferir entre sí, lo que hace que la implementación y administración de aplicaciones sea mucho más sencilla y eficiente.

Una de las ventajas clave de Docker es su capacidad para reducir la complejidad de la implementación de aplicaciones en diferentes entornos. Docker proporciona una plataforma de desarrollo unificada y portátil que simplifica la implementación en diferentes sistemas operativos y plataformas en la nube. Además, la capacidad de Docker para ejecutar múltiples contenedores en una sola máquina permite una mejor utilización de los recursos de hardware, lo que se traduce en una mayor eficiencia y escalabilidad en el despliegue de aplicaciones.

3.2.1.1.- Contenedores vs Máquinas Virtuales

Esta subsección se centra en la tecnología de contenedores, la cual, a pesar de solaparse en parte con la funcionalidad de las máquinas virtuales, tiene una tecnología subyacente muy diferente. No obstante, la analogía de los contenedores como máquinas virtuales ligeras es muy popular y puede resultar útil para comprender el concepto de forma inicial, siempre y cuando se tenga en cuenta que en realidad se trata de otra cosa.

El objetivo principal de los contenedores es solucionar el problema de la ejecución de aplicaciones que dependen de bibliotecas o servicios externos. Este problema se refiere a la situación en la que una aplicación que funcionaba perfectamente en el entorno de desarrollo del desarrollador deja de funcionar cuando se implementa en el servidor real, como en la nube, por ejemplo. Esto se debe a que los entornos de desarrollo y de producción no son idénticos, y cualquier pequeña diferencia en las versiones de las



herramientas instaladas o en la localización de archivos puede afectar significativamente al funcionamiento de la aplicación, con resultados imprevisibles.

Para solucionar el problema del despliegue de aplicaciones con múltiples dependencias, una estrategia que se ha utilizado es la de crear una máquina virtual en la que se desarrolla y prueba la aplicación. Una vez que se ha verificado que todo funciona correctamente, se genera una imagen de disco de la máquina virtual y se utiliza esta imagen para crear una máquina virtual idéntica en el entorno de producción. De esta manera, la aplicación se ejecuta en un entorno de producción idéntico al de desarrollo, lo que evita problemas de incompatibilidad entre diferentes entornos. Si bien esta solución puede considerarse un enfoque "a lo bestia", es efectiva para garantizar la coherencia del entorno de ejecución de la aplicación, aunque poco eficiente.

Uno de los principales problemas que implican las máquinas virtuales radica en el elevado consumo de recursos, ya que cada máquina virtual requiere de una cantidad significativa de memoria exclusiva y una copia completa del sistema operativo y su kernel. Esto limita la cantidad de instancias que se pueden lanzar en un mismo ordenador o en la nube, lo que afecta al coste del despliegue, ya que cada máquina virtual se cobra por separado. Además, la replicación de una imagen binaria de disco para crear nuevas máquinas virtuales no es una tarea sencilla, ya que los desarrolladores a menudo instalan paquetes, actualizaciones y herramientas sin documentar todos los pasos. Incluso si se toman notas exhaustivas, replicar el resultado puede ser una tarea tediosa y manual, lo que limita la capacidad de replicar rápidamente un entorno de desarrollo o despliegue de aplicaciones. Por último, las máquinas virtuales no permiten una elasticidad rápida, ya que es necesario crear una nueva máquina virtual y arrancarla para instanciar otra copia de la aplicación. El arranque del sistema operativo típicamente lleva unos minutos, lo que limita la capacidad de respuesta a la demanda del usuario y aumenta el tiempo de inactividad.

Por otro lado, el uso de contenedores es una solución efectiva para superar los problemas previamente planteados como se plantea a continuación. En cuanto a la alta demanda de recursos de las máquinas virtuales los contenedores pueden ser creados con una imagen que solo contenga los archivos necesarios para la aplicación, lo que permite arrancar varios contenedores en una misma máquina, compartiendo la misma imagen sin la necesidad de copiarla. Esto, a su vez, reduce el costo de despliegue en la nube, ya que se cobrará solo por el uso del recurso compartido.

Respecto a la dificultad para replicar la imagen binaria del disco de una máquina virtual, la creación de la imagen de un contenedor puede ser fácilmente replicada a través de archivos con sintaxis especial. Las imágenes para aplicaciones comunes pueden descargarse y utilizarse sin la necesidad de realizar la instalación manual de paquetes y herramientas.

La falta de elasticidad en la creación de copias adicionales de una aplicación no supone un problema ya que la creación de un nuevo contenedor es rápida y sencilla, ya que no requiere la creación de un sistema operativo completo, sino que utiliza el de la máquina



en la que se ejecuta. Por lo tanto, el tiempo de arranque de un contenedor es mucho más rápido que el de una máquina virtual.

3.2.2.- Contenedores

A continuación, se explica el funcionamiento básico de los contenedores a partir del escenario mostrado en la Figura 3.3.

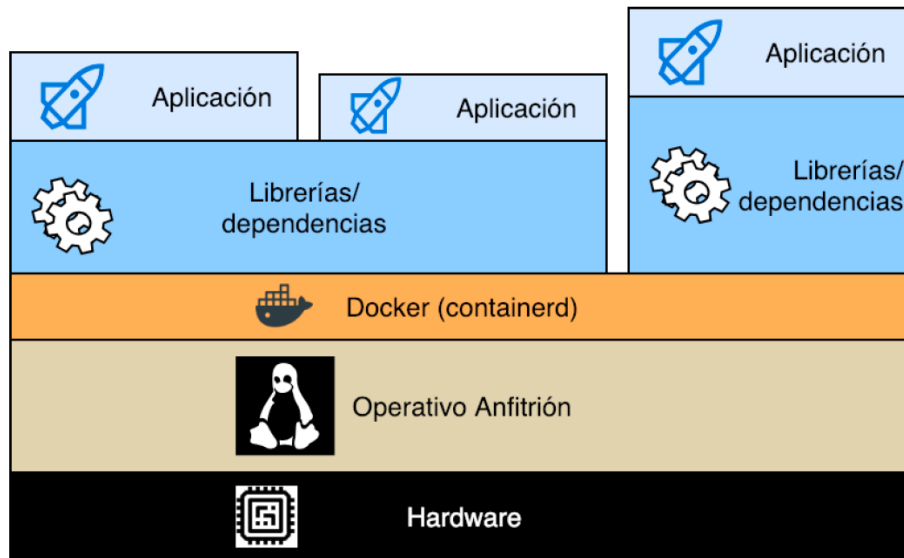


Figura 3.3.- Escenario de despliegue de diferentes aplicaciones por medio de la tecnología Docker. [16]

En este escenario ha sido instalado un sistema operativo Linux en un hardware arbitrario. En este sistema operativo se ha instalado el gestor de contenedores Docker para poder ejecutar tres aplicaciones en contenedores separados. Las dos primeras aplicaciones comparten las mismas bibliotecas y herramientas, lo que permite que estas herramientas estén disponibles una sola vez en el sistema de archivos del anfitrión, ahorrando así espacio. La tercera aplicación no comparte herramientas, por lo que su imagen ocupa su propio espacio en disco separado. A pesar de esto, todos los contenedores se ejecutan en el mismo sistema operativo anfitrión, lo que permite ahorrar recursos y reducir el tiempo de arranque de cada aplicación.

Aunque todos los contenedores tienen el mismo sistema operativo, es posible tener diferentes distribuciones Linux en cada contenedor, lo que permite tener diferentes conjuntos de herramientas y configuraciones en cada uno. Por último, es importante destacar que este modelo se puede utilizar en la nube, ya que se podría incluso sustituir el hardware por una máquina virtual.



3.2.2.1.- Sistema de archivos por capas

El sistema de archivos por capas que utiliza Docker es una característica clave para el eficiente funcionamiento de los contenedores. Este sistema se basa en una serie de capas que se apilan unas sobre otras para construir la imagen de un contenedor. Cada capa representa una modificación o adición a la imagen anterior, lo que permite que las imágenes de los contenedores sean muy eficientes en términos de espacio.

Cuando se crea una imagen de contenedor, Docker comienza por una capa base que contiene el sistema de archivos mínimo requerido para que el contenedor funcione. A continuación, Docker agrega una serie de capas adicionales que incluyen cualquier cambio en el sistema de archivos necesario para que la aplicación del contenedor se ejecute. Por ejemplo, si una aplicación necesita una biblioteca específica, Docker agregará una capa al sistema de archivos que incluye esa biblioteca.

El resultado de este proceso es una imagen que contiene todas las capas apiladas juntas para formar un sistema de archivos completo que incluye todo lo necesario para ejecutar el contenedor. Cada capa es de solo lectura y se puede compartir entre diferentes imágenes de contenedores, lo que ahorra espacio en el disco y reduce el tiempo de construcción de la imagen. Además, como las capas son independientes, se pueden actualizar o reemplazar individualmente sin afectar a las demás capas, lo que permite una mayor flexibilidad y facilidad de mantenimiento.

No obstante, el hecho de que las capas sean únicamente de lectura plantea el problema de dónde guardar los datos generados por procesos ejecutados en el contenedor, e incluso el cómo modificar cualquier fichero, por ejemplo, de configuración, si se quieren modificar los valores por defecto. Para solucionar este problema Docker plantea dos soluciones.

Como primera solución, el sistema de archivos por capas en Docker agrega una capa adicional de escritura encima de todas las capas que componen la imagen. Esta capa es específica del contenedor en ejecución y permite que el contenedor cree y modifique archivos en ella. Incluso si el contenedor intenta modificar un archivo en una de las capas inferiores, en realidad se crea una nueva versión de ese archivo en la capa superior de escritura, lo que oculta al archivo original en la capa inferior. De esta manera, cada contenedor tiene su propia capa de escritura que le permite modificar y personalizar la imagen base sin afectar a otras instancias de contenedor que utilizan la misma imagen. Si todos los procesos dentro del contenedor en ejecución finalizan, se considera que el contenedor ha llegado al final de su ciclo de vida, pero aún no se descarta de inmediato. Esto se debe a que es posible que se hayan realizado modificaciones en la capa de escritura que se perderían si se descarta por completo. En este caso, el usuario tiene la oportunidad de añadir los cambios como una nueva capa inmutable en la jerarquía de capas, para crear con ellos una nueva imagen de contenedor.

Como segunda solución al problema se plantean los conocidos como volúmenes. Los volúmenes son un tipo de almacenamiento que se puede "montar" en un contenedor y que



existe fuera del mismo. Aparece como una carpeta en el sistema de archivos del contenedor y puede ser compartida entre varios contenedores para intercambiar información. Puede ser una carpeta del anfitrión, permitiendo compartir datos con el contenedor y si un proceso escribe en esa carpeta, los cambios serán inmediatamente visibles en el anfitrión. El volumen sobrevive al contenedor y no se pierde si el contenedor finaliza su ejecución. Los datos guardados en el volumen externo se conservan, aunque la capa de escritura del contenedor se pierda. Por esta razón, los volúmenes suelen ser utilizados para almacenamiento en contenedores que ejecutan bases de datos.

3.2.3.- Dockerfile, Build y Run

Docker utiliza imágenes para definir y distribuir aplicaciones. Para construir imágenes personalizadas, se utiliza una herramienta llamada Dockerfile, que es un archivo de texto plano que contiene una serie de instrucciones para construir una imagen a partir de una imagen base.

El Dockerfile es esencialmente un script que describe cómo se debe construir la imagen. Cada línea en este archivo representa una instrucción que se ejecutará en el proceso de construcción. Las instrucciones pueden incluir comandos de Linux, como la instalación de paquetes o la configuración de variables de entorno, o comandos específicos de Docker, como la exposición de puertos o la copia de archivos.

Una vez que se ha creado un Dockerfile, se puede usar el comando "docker build", acompañado de algunos parámetros opcionales [17], para construir una imagen a partir de ese archivo. Dicho comando se encarga de leerlo y ejecutar las instrucciones en él para construir la imagen. El resultado es una nueva imagen de Docker que contiene todas las dependencias y configuraciones necesarias para ejecutar la aplicación o servicio que se describe en el Dockerfile. Este proceso es esencial para asegurar que las imágenes sean replicables y consistentes en diferentes entornos y sistemas operativos.

Después de haber creado una imagen de Docker utilizando un Dockerfile, la siguiente tarea es crear contenedores a partir de esa imagen. El comando "docker run" es utilizado para crear e iniciar un contenedor a partir de una imagen existente. El comando "docker run" acepta varios argumentos [18], como el nombre de la imagen que se utilizará, las variables de entorno necesarias, los puertos que deben ser expuestos, y el comando que se ejecutará en el contenedor, para, por ejemplo, ejecutar directamente un archivo concreto o abrir una terminal de bash.

Nótese que los archivos Dockerfile utilizados en este proyecto, así como los scripts ejecutables conteniendo los comandos "docker build" y "docker run", se pueden encontrar en el Anexo B.



3.3.- GNU Radio

En este apartado se introducen la herramienta GNU Radio junto con sus fundamentos y funcionalidades básicas. Asimismo, se presenta su entorno gráfico de desarrollo, conocido como GNU Radio Companion (GRC) [19]. Cabe destacar que, para el desarrollo de este proyecto, se ha utilizado la versión más reciente de ambas herramientas, la 3.10, y el sistema operativo utilizado ha sido Ubuntu 22.04 LTS (Long Term Support) [20]. No obstante, como se mencionó en el apartado anterior, todo se ha llevado a cabo en un contenedor de Docker.

3.3.1.- Introducción

GNU Radio es una herramienta de desarrollo de software de código abierto que proporciona bloques para el procesamiento de señales digitales, principalmente utilizados en la implementación de sistemas de comunicación. Esta herramienta permite conectar un sistema desarrollado de esta manera a un equipo de hardware de RF para crear un transmisor o receptor real. Asimismo, puede utilizarse en un entorno de software como una herramienta de simulación, como es el caso de este proyecto.

En esencia, GNU Radio es un entorno de programación desarrollado en C++ que incluye una amplia variedad de bloques de procesamiento de señales, cada uno de los cuales realiza una tarea específica. Lo más importante de GNU Radio es que también define cómo se relacionan estos bloques entre sí, y como herramienta de código abierto, permite a cualquier usuario programar sus propios bloques o modificar los existentes. El código fuente se puede encontrar en su repositorio GitHub [21], y su manual y página de referencia se recogen en [22]. Además, GNU Radio cuenta con una interfaz gráfica llamada GRC que permite al usuario manejar e interconectar bloques de manera intuitiva. Aunque GRC tiene algunas limitaciones en cuanto a funcionalidad, combinada con la programación del código, puede acelerar la configuración de muchos bloques y parámetros.

Por otro lado, parte de las funcionalidades de usuario de GNU Radio se basan en el lenguaje de programación Python. Por ejemplo, los esquemas diseñados en GRC generan un ejecutable en Python, que controla la capa superior del programa invocando los bloques que lo componen. Además, existe la opción de programar bloques personalizados en Python, que GNU Radio se encarga de compilar posteriormente en C++.

En resumen, las características principales que han llevado a elegir GNU Radio como una de las herramientas de software para este proyecto son las siguientes: el hecho de ser un proyecto de software libre y de código abierto; la flexibilidad y la versatilidad que ofrecen el uso de lenguajes de programación diferentes, como C++ y Python, para el procesamiento digital; la existencia de una comunidad amplia y activa, resultado de su creciente popularidad en diversos ámbitos, no solo en educación e investigación, sino



también en el mundo empresarial; y su compatibilidad con una amplia gama de dispositivos de hardware de diferentes fabricantes.

3.3.2.- Bloques en GNU Radio y GRC como entorno de desarrollo

En GNU Radio, la palabra "bloque" se refiere a la unidad básica que compone la estructura de la herramienta. Cada bloque es un subsistema autónomo que actúa como fuente, destino o procesador de valores de diferentes tipos. Todos los bloques se derivan de una clase abstracta en C++ que define su estructura y funcionalidades básicas. Por lo tanto, tanto los bloques predeterminados como los específicamente desarrollados para este proyecto de fin de máster son clases derivadas de la clase madre.

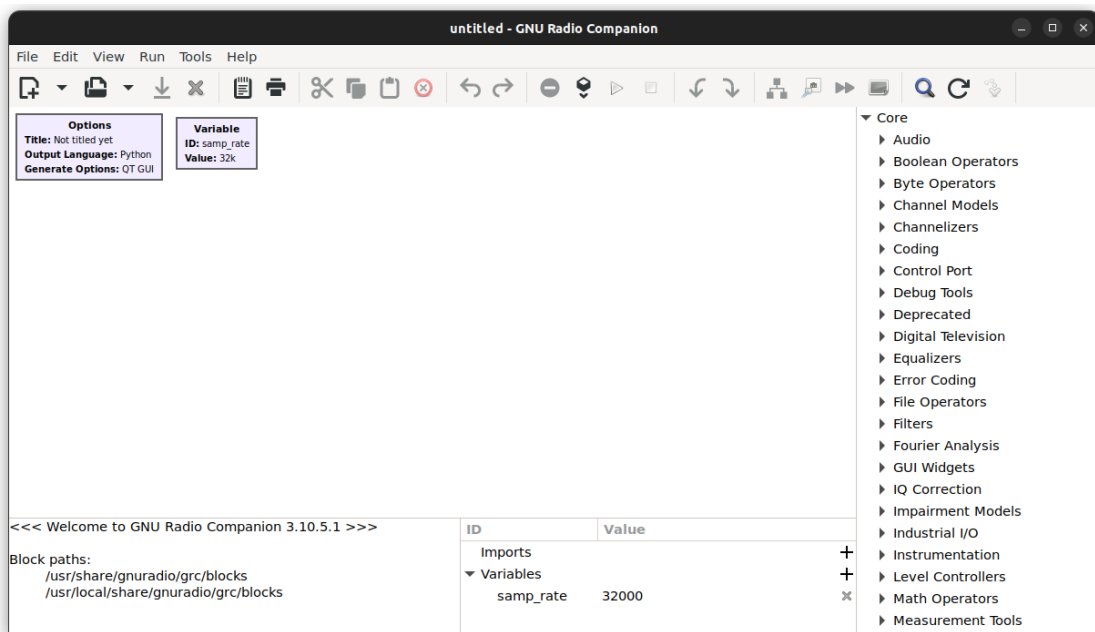


Figura 3.4.- Captura de pantalla de la interfaz gráfica de la herramienta GRC.

El análisis de los bloques en GNU Radio puede implicar diferentes clasificaciones basadas en varios criterios. Por ejemplo, los bloques pueden ser clasificados como fuente o sumidero según si generan o no información a su salida sin recibir ninguna entrada respectivamente y viceversa. También es importante la distinción entre entradas y salidas síncronas y asíncronas. Las entradas y salidas síncronas trabajan con una tasa de muestreo fija y las entradas asíncronas no esperan la recepción de información de manera periódica. Por lo tanto, las salidas asíncronas no generan información periódicamente. También cabe destacar que existe una incompatibilidad entre la conexión de bloques síncronos y asíncronos, así como entre entradas y salidas que manejan diferentes tipos de valores. Por



ejemplo, una salida que genera valores de tipo entero solo puede ser conectada a una entrada que espera muestras de tipo entero.

Los bloques en GNU Radio están organizados en módulos según su función, tales como gr-digital o gr-analog, los cuales se pueden encontrar en [21]. Para programar bloques, funciones o clases personalizadas, el primer paso es crear un OOT (Out-Of-Tree). Una vez programado el módulo, este se puede integrar en cualquier instancia de GNU Radio, siempre que la versión coincida con la del entorno utilizado para su desarrollo.

El proceso de conectar una serie de bloques y configurar los parámetros generales en GNU Radio puede resultar tedioso. Sin embargo, la herramienta gráfica GRC simplifica esta tarea al permitir crear un archivo de Python que define dichas conexiones y variables de manera intuitiva y ágil. Además, desde GRC se pueden configurar los atributos de cada bloque mediante una interfaz definida en un archivo YAML. Durante la ejecución del programa resultante, se instancia un objeto de la clase correspondiente para cada bloque y se gestiona el intercambio de información entre ellos.

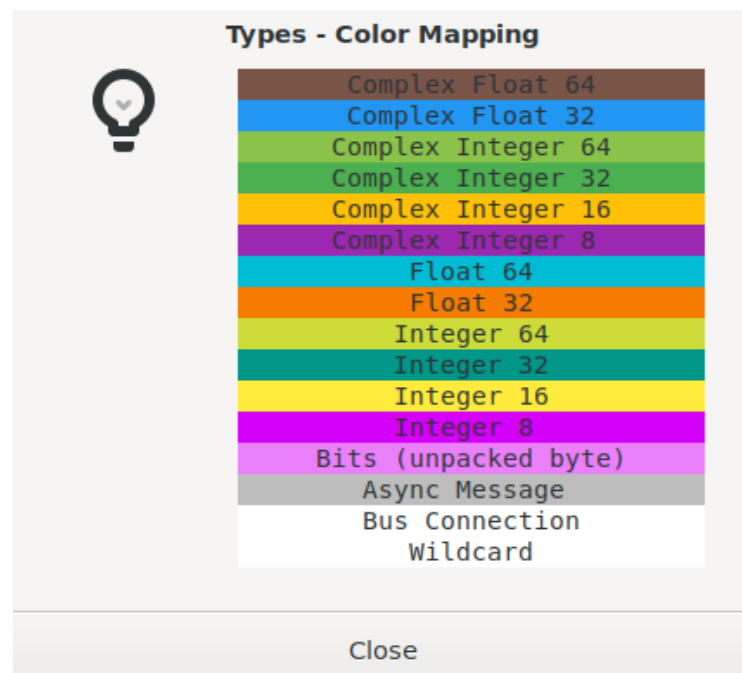


Figura 3.5.- Captura de pantalla de la ventana de ayuda mostrando la leyenda de colores de los diferentes tipos de datos en GRC.

Finalmente, es importante mencionar que en GRC se facilita una leyenda de colores para distinguir los distintos tipos de valores de las entradas y salidas, tal y como se ilustra en la Figura 3.5. Es importante destacar que las entradas y salidas asíncronas se identifican con una etiqueta propia, Async Message, lo que se tratará con más detalle en la siguiente sección.



3.3.3.- Comunicación entre bloques

GNU Radio establece dos formas de comunicación entre bloques para permitir el intercambio de información de control entre ellos y modificar su comportamiento o el de otros bloques: stream tags y message passing.

Las stream tags se propagan a través de conexiones síncronas y funcionan de forma paralela al flujo principal de datos, que implica el intercambio de muestras de un determinado tipo y frecuencia fija entre bloques. Cada stream tag está asociado a una muestra del flujo principal y se genera a través de un bloque encargado de dicha asociación. Los bloques aguas abajo en el esquema pueden interpretar, modificar, suprimir o propagar los datos de la etiqueta de forma transparente. Las stream tags se definen mediante un par clave-valor de tipo polimórfico (PMT) [23].

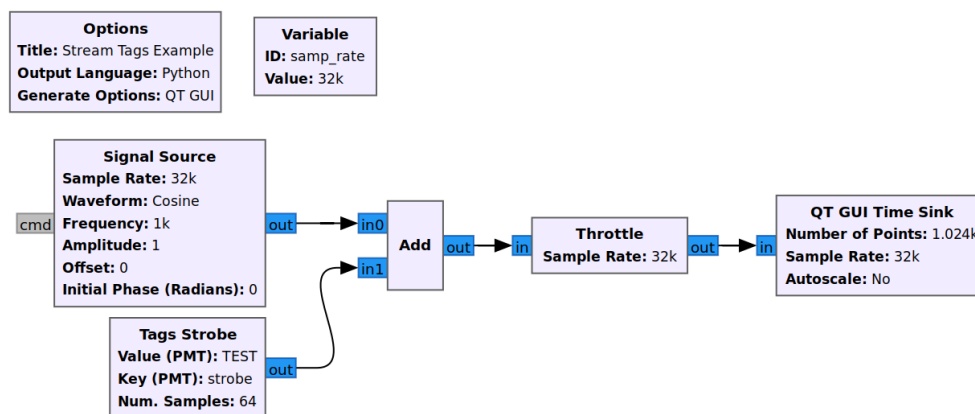


Figura 3.6.- Ejemplo de un diagrama de bloques en GRC por medio del cual se emplea el uso de las stream tags.

En la Figura 3.6 se ilustra cómo funcionan las stream tags por medio de un esquema de ejemplo. En este caso, el bloque Tags Strobe produce etiquetas con clave strobe y valor TEST cada 64 muestras. Dado que la variable samp_rate está definida como 32k muestras por segundo y la frecuencia del coseno generado por el bloque Signal Source es de 1 kHz, se genera una etiqueta cada dos periodos de la señal sinusoidal. El bloque Add es responsable de asociar las etiquetas con el flujo de muestras, mientras que el bloque Throttle controla la congestión en función de la tasa de muestreo. Para visualizar las muestras complejas junto con sus etiquetas correspondientes, se utiliza el bloque QT GUI Time Sink. La Figura 3.7 muestra el resultado de esta visualización.

Por otra parte, el envío de mensajes ofrece una opción asíncrona para la comunicación entre bloques. Este enfoque permite que los bloques aguas abajo se comuniquen con los bloques aguas arriba en el diagrama, lo que facilita la comunicación entre el ejecutable de GNU Radio y una aplicación externa. Tanto las entradas como las salidas que generan y reciben estos mensajes se identifican con el tipo especial "Async



Message", que se muestra en la Figura 3.5. Al igual que las stream tags, el contenido de los mensajes es de tipo PMT, y puede, aunque no necesariamente, estar estructurado como un par clave-valor. En particular, GNU Radio define el PDU (Protocol Data Unit) como un tipo de PMT especial utilizado para enviar mensajes, que está compuesto por un par de metadatos y datos. Los metadatos se representan como un diccionario que consta de un conjunto de claves y valores, mientras que los datos se representan como un vector cuyos valores son de un tipo específico que se determina en cada caso.

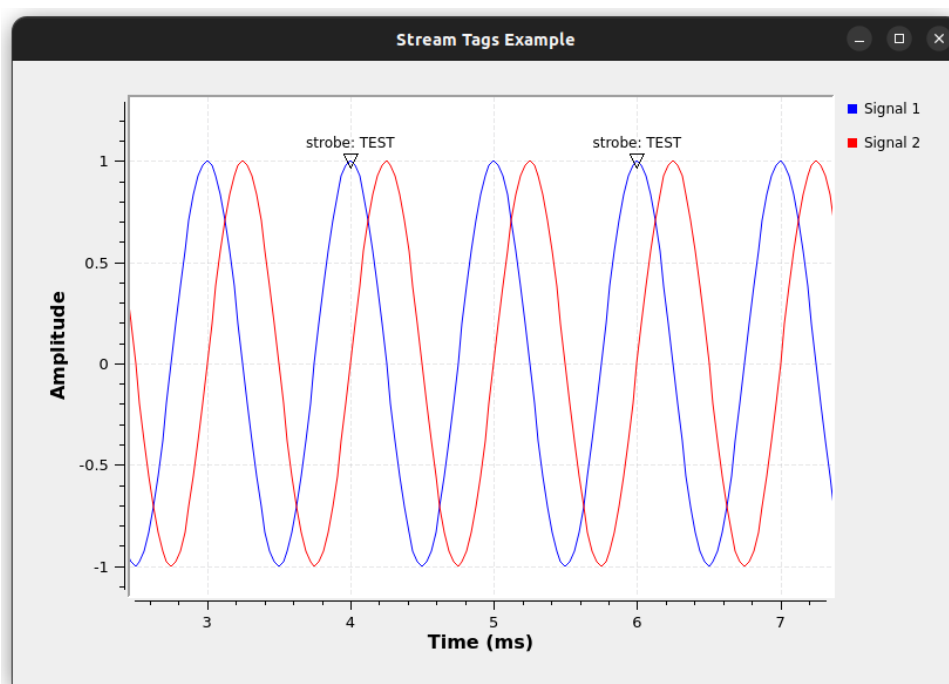


Figura 3.7.- Resultado de la ejecución del diagrama de bloques de la Figura 3.6.

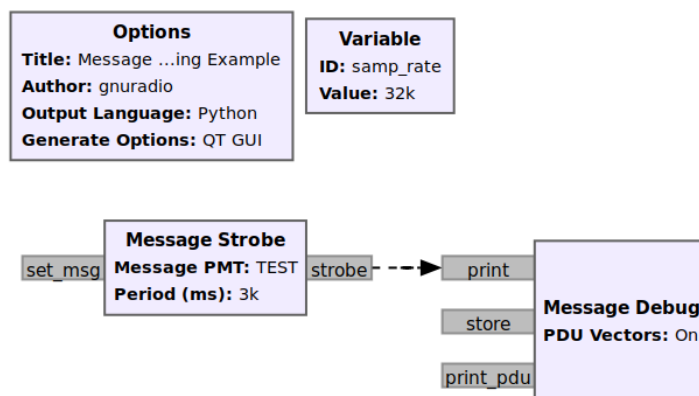


Figura 3.8.- Ejemplo de un diagrama de bloques en GRC que emplea el uso del tipo de datos Async Message.



En la Figura 3.8 se puede apreciar un ejemplo muy básico de funcionamiento del envío de mensajes. El bloque Message Strobe genera un mensaje cada 3 segundos con el string TEST como contenido, que es impreso por la terminal por el bloque Message Debug como se muestra en la Figura 3.9.

```

***** MESSAGE DEBUG PRINT *****
TEST
*****
***** MESSAGE DEBUG PRINT *****
TEST
*****
***** MESSAGE DEBUG PRINT *****
TEST
*****

```

Figura 3.9.- Captura de pantalla del resultado en la terminal de GRC tras la ejecución del programa de la Figura 3.8.

Por último, es importante destacar que la naturaleza asíncrona de este enfoque implica que el momento exacto en que se genera o se interpreta un mensaje no es determinista. Esta es la diferencia más significativa con respecto a las etiquetas de flujo, que se transmiten, reciben e interpretan junto con una muestra del flujo principal. En el caso de los mensajes, el tiempo de respuesta dependerá del estado general del programa, así como de los hilos y procesos que se ejecutan en la CPU del sistema en cualquier momento dado. Aunque el tiempo transcurrido desde que se genera, se recibe e interpreta un mensaje suele ser del orden de milisegundos, esta realidad debe tenerse en cuenta al elegir el modo de comunicación más adecuado entre bloques según la aplicación en cuestión.

3.4.- Gr-leo

En este apartado se introduce la herramienta gr-leo, así como los detalles básicos de cómo se implementa el canal estudiado en el Capítulo 2, los bloques de GNU Radio que tiene, algunos usos que se le puede dar y sus limitaciones.

3.4.1.- Introducción

Gr-leo [24] es una herramienta de código abierto desarrollada en el marco de SDR Makerspace, una iniciativa de la Agencia Espacial Europea (ESA) y la Fundación Libre Space. Esta herramienta está escrita en C++ como un módulo externo, o módulo OOT de GNU Radio que simula el canal de comunicaciones entre satélites en órbita baja y estaciones terrestres.



El propósito de esta herramienta es replicar el funcionamiento de un sistema que involucra la comunicación entre la Tierra y los satélites de órbita baja, y simular los diferentes fenómenos que pueden perturbar y modificar la señal desde el proceso de transmisión, hasta su recepción. De esta manera, esta herramienta sirve de gran ayuda para la evaluación de subsistemas de comunicaciones involucrados en comunicaciones con satélites de órbita baja.

Este módulo se ve implementado de forma que cada uno de los componentes principales del enlace de comunicaciones LEO se ven divididos en un bloque individual, esto es, las antenas, el canal, el modelo de canal, el satélite y la estación terrena, estación de seguimiento o tracker. Adicionalmente, a la hora de definir algunos bloques, se puede optar por definir el modo de transmisión, esto es, el modo de enlace ascendente, o el modo de enlace descendente.

El código de esta herramienta y su manual de instalación se encuentran en [24]. No obstante, es importante destacar, que, para su correcta instalación acorde con la versión de GNU Radio utilizada en este proyecto (versión 3.10), ha tenido que hacerse unas correcciones indicadas hilo de problemas del repositorio, en [25].

3.4.2.- Perturbaciones del canal consideradas

Tal y como se mostró en el Capítulo 2, existen varias limitaciones en el canal de comunicación entre una estación terrestre y un satélite en órbita baja. El movimiento relativo del satélite en órbita, las frecuencias de operación, la configuración de la antena del sistema de telecomunicaciones y los diversos fenómenos atmosféricos, así como la atenuación que sufre la señal por viajar por el espacio son factores que afectan significativamente el rendimiento de la comunicación. A continuación, se lista las perturbaciones consideradas por parte del módulo gr-leo.

- Desplazamiento en frecuencia por efecto Doppler.
- Atenuación por propagación en espacio libre.
- Atenuación por gases atmosféricos.
- Atenuación por precipitación atmosférica.
- Pérdidas por apuntamiento.

Cada una de estas perturbaciones del canal se implementan en el módulo gr-leo de la misma forma que fueron descritos en el Capítulo 2. No obstante, cabe remarcar un par de detalles acerca de su implementación.

Por ejemplo, respecto a la atenuación por gases atmosféricos, el módulo no solo utiliza las recomendaciones de la UIT mencionadas en el Capítulo 2, sino que también aporta un cálculo de estas pérdidas descrito en el libro [26].

Por otro lado, gr-leo implementa el fenómeno de las pérdidas de apuntamiento a través de la disminución de ganancia de la antena, también conocido como roll off gain, el



cual es un parámetro modificable en el bloque de definición de antenas sobre el cual se habla más en detalle en el subapartado siguiente.

3.4.3.- Arquitectura de bloques

Gr-leo es un módulo OOT escrito en C++ que utiliza el enfoque de programación en bloques de GNU Radio. Su implementación se divide en bloques de procesamiento que simulan cada componente de un sistema de comunicación de canal espacial. Estos bloques pueden combinarse con los bloques de procesamiento integrados de GNU Radio, que también están disponibles a través de GNU Radio Companion. A continuación, se proporciona una descripción detallada de cada uno de los bloques implementados.

3.4.3.1.- Channel Model

El bloque de Channel Model en GNU Radio funciona como un puente entre la señal de entrada y una definición de modelo de canal especificada. El bloque aplica los efectos del canal en la señal de entrada y dirige la salida hacia la etapa posterior del flujo de procesamiento de señal. La opción de modificar el suelo de ruido mediante la adición de ruido blanco gaussiano es una característica valiosa ya que permite simular condiciones reales del canal y evaluar el rendimiento del sistema de comunicación bajo diferentes escenarios. Adicionalmente, hay un puerto del tipo Async Message opcional que tiene la capacidad de emitir PDUs con la información generada del balance de enlace.

El bloque de modelo de canal juega un papel crucial en representar con precisión el impacto del canal de comunicación en la señal de entrada y proporciona información valiosa sobre el comportamiento de su sistema bajo diferentes condiciones del canal.

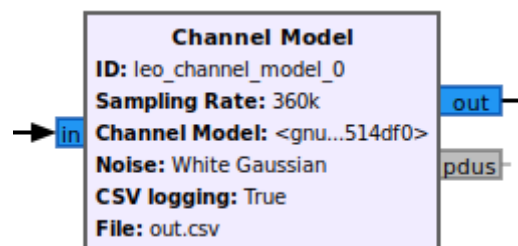


Figura 3.10.- Representación del bloque Channel Model del módulo OOT gr-leo en GRC.

Parámetros de entrada esperados:

- **ID.** Nombre de identificación para el bloque del modelo de canal.
- **Sampling Rate.** Por lo general, la misma tasa de muestreo que en todo el diagrama de flujo. Representa la velocidad a la que se consumen las muestras por el bloque.
- **Channel Model.** Una instancia del bloque de definición del modelo LEO (descrito más adelante).



- **Noise.** Opción para elegir entre sin ruido o ruido blanco gaussiano.
- **CSV logging.** Opción para elegir si se desea obtener datos de registro en formato CSV en un archivo o no.
- **File.** La ruta al archivo en el que se desea escribir los datos generados en formato CSV.

3.4.3.2.- LEO Model Definition

El núcleo de gr-leo radica en las definiciones de modelos de canal, que integran los diversos componentes del sistema de comunicación entre tierra y espacio en un solo bloque de variable de GNU Radio. El bloque de modelo de canal requiere una lista de parámetros de entrada, que incluye el *tracker* de la estación terrestre de la Tierra, el modo de transmisión (enlace descendente o enlace ascendente), y otras opciones que describen las alteraciones a tener en cuenta durante la simulación del canal.

En cuanto a la implementación, cada bloque de definición de modelo de canal está representado por una clase C++ derivada de la clase principal *generic_model* y debe implementar la función esencial *generic_work*. Esta función es llamada por el bloque de modelo de canal, procesa la señal de entrada y escribe la señal resultante en el búfer de salida. La función *generic_work* es el componente central de la definición del modelo de canal y realiza las manipulaciones necesarias para simular los efectos del canal de comunicación en la señal de entrada.

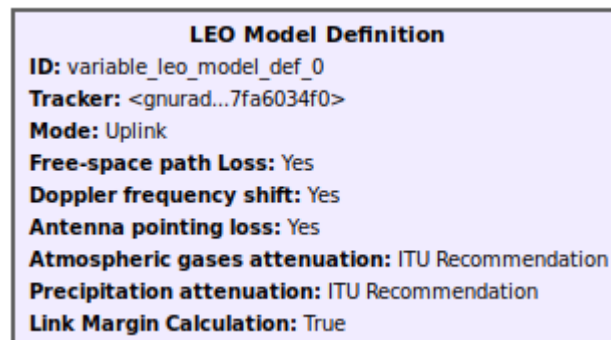


Figura 3.11.- Representación del bloque LEO Model Definition del módulo OOT gr-leo en GRC.

Parámetros de entrada esperados:

- **ID.** Nombre de identificación para el bloque de Definición del Modelo LEO.
- **Tracker.** Una instancia del bloque Tracker (descrito más adelante).
- **Mode.** Opción para elegir entre modo Uplink o Downlink. Esto especifica internamente qué frecuencia utilizar para la transmisión y recepción de cada elemento involucrado.



- **Free-space path Loss.** Opción para elegir si se tienen en cuenta o no las pérdidas de propagación en espacio libre.
- **Doppler frequency shift.** Opción para elegir si se tiene en cuenta o no la desviación en frecuencia Doppler.
- **Antenna pointing loss.** Opción para elegir si se tienen en cuenta o no las pérdidas de apuntamiento de la antena.
- **Atmospheric gases attenuation.** Opción para elegir entre la recomendación de la ITU, el análisis de regresión o la no implementación de la atenuación por gases atmosféricos.
- **Precipitation attenuation.** Opción para elegir entre la recomendación de la ITU, el análisis de regresión o la no implementación de la atenuación por precipitación.
- **Link Margin Calculation.** Opción para elegir si se desea calcular o no el margen de enlace.

3.4.3.3.- Tracker

En gr-leo, la estación terrestre ubicada en la Tierra está representada por una instancia de la clase Tracker. Esta clase contiene información crítica necesaria para describir una estación terrestre que está rastreando un satélite en órbita, como el satélite de interés, las coordenadas geográficas de la estación terrestre (Latitud, Longitud y Altitud), las frecuencias de transmisión y recepción, los objetos de la clase Antenna, el período de observación en formato UTC ISO-8601 y la resolución de tiempo de observación en microsegundos. La clase Tracker también es responsable de utilizar la biblioteca SGP4 [27] de C++ para analizar los datos de TLE y realizar cálculos para determinar información específica de la órbita, como la distancia desde la estación terrestre y su velocidad relativa. Por lo tanto, la clase Tracker es fundamental para representar con precisión la estación terrestre situada en la Tierra y sus interacciones con el satélite en órbita en la simulación gr-leo.

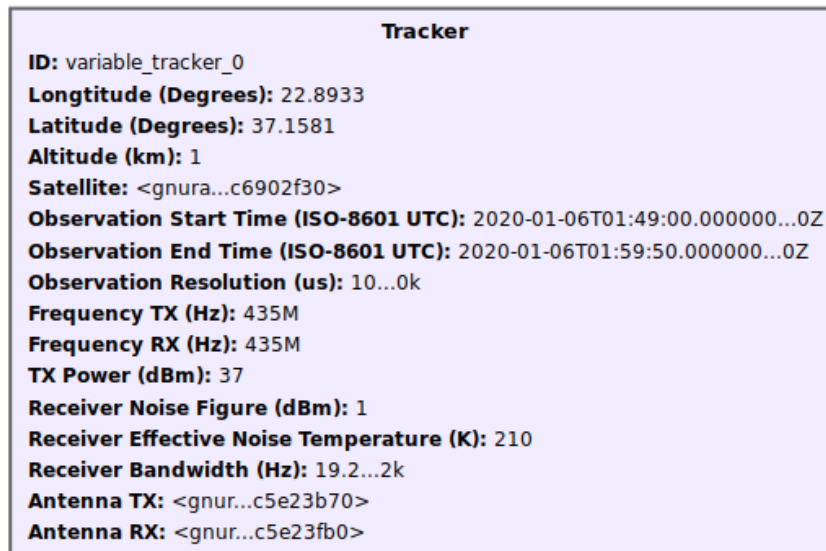


Figura 3.12.- Representación del bloque Tracker del módulo OOT gr-leo en GRC.

Parámetros de entrada esperados:

- **ID.** Nombre identificativo para el bloque Tracker.
- **Longitude (Degrees).** Coordenada geográfica de longitud de la estación terrena en grados.
- **Latitude (Degrees).** Coordenada geográfica de latitud de la estación terrena en grados.
- **Altitude (km).** Altitud geográfica de la estación terrena en kilómetros.
- **Satellite.** Una instancia del bloque Satellite (descrito más adelante).
- **Observation Start Time (ISO-8601 UTC).** Tiempo de inicio para la observación desde la estación terrena. Se espera el formato de tiempo ISO-8601 UTC.
- **Observation End Time (ISO-8601 UTC).** Tiempo de finalización para la observación desde la estación terrena. Se espera el formato de tiempo ISO-8601 UTC.
- **Observation Resolution (us).** Resolución temporal o paso de tiempo considerado para cada medida tomada durante la observación, medido en μ s.
- **Frequency TX (Hz).** Valor de frecuencia utilizado para la transmisión desde la estación terrena, medido en Hz.
- **Frequency RX (Hz).** Valor de frecuencia utilizado para la recepción en la estación terrena, medido en Hz.
- **TX Power (dBm).** Potencia de salida de transmisión de la estación terrena medida en dBm.
- **Receiver Noise Figure (dBm).** Figura de ruido del receptor de la estación terrena medida en dB (*dBm es un error tipográfico).



- **Receiver Effective Noise Temperature (K).** Temperatura efectiva de ruido del receptor de la estación terrena medida en Kelvin.
- **Receiver Bandwidth (Hz).** Ancho de banda del receptor esperado de la estación terrena medido en Hz.
- **Antenna TX.** Una instancia del bloque Antenna (descrito más adelante) como la antena utilizada para la transmisión en la estación terrena.
- **Antenna RX.** Una instancia del bloque Antenna (descrito más adelante) como la antena utilizada para la recepción en la estación terrena.

La resolución temporal de la simulación de canal, implementada por medio de el parámetro **Observation Resolution (us)** juega un papel muy importante en esta herramienta. La forma en que esta se implementa es mediante el cálculo de una ventana de muestras que han de ser procesadas para avanzar la simulación orbital en el intervalo de tiempo especificado en microsegundos. Esta ventana de muestras se calcula a partir de la Ecuación 3.1.

$$\text{Ventana temporal de muestras} = \frac{\text{Tasa de muestreo} * \text{Resolución temporal (us)}}{10^6} \quad (3.1)$$

A partir de dicha ecuación se puede asumir que ha de encontrarse un equilibrio entre la tasa de muestreo y la resolución temporal. Por una parte, se debe evitar tener una ventana de muestras demasiado grande, la cual el programa no podrá procesar adecuadamente y evocará en un fallo o terminación del programa. Por el otro lado, se debe evitar también unos valores demasiado pequeños o la simulación tomará demasiado tiempo en procesar las muestras transmitidas por el canal. Asimismo, si se desea obtener muchos puntos de simulación orbital ha de reducirse el tamaño de resolución temporal, incrementando a su vez la tasa de muestreo y viceversa si se desea una menor cantidad de puntos.

3.4.3.4.- Satellite

En gr-leo, los satélites se representan mediante instancias de la clase Satellite. Esta clase almacena toda la información necesaria para describir un satélite en órbita baja, como su descripción de órbita proporcionada por los datos de TLE, las frecuencias de transmisión y recepción utilizadas por el satélite y los objetos de antena TX/RX asociados con él. La clase Satellite ofrece un enfoque eficiente y estructurado para administrar información sobre un satélite dado en la simulación de gr-leo.



```

Satellite
ID: variable_satellite_0
TLE Title: QUBIK
TLE 1: 1 84001U...0-4 0 08
TLE 2: 2 84001 ...16786 02
Frequency TX (Hz): 435M
TX Power (dBm): 27
Receiver Noise Figure (dB): 12
Receiver Effective Noise Temperature (K): 190
Receiver Bandwidth (Hz): 1.2k
Antenna TX: <gnur...c5e0d570>
Frequency RX (Hz): 435M
Antenna RX: <gnur...c5e218f0>

```

Figura 3.13.- Representación del bloque Satellite del módulo OOT gr-leo en GRC.

- **ID.** Nombre de identificación para el bloque de satélite.
- **TLE Title.** Línea de datos TLE de título. Corresponde a la primera línea de datos TLE.
- **TLE 1.** Línea de datos TLE 1. Corresponde a la segunda línea de datos TLE.
- **TLE 2.** Línea de datos TLE 2. Corresponde a la tercera línea de datos TLE.
- **Frequency TX (Hz).** Valor de frecuencia utilizado para la transmisión del satélite medido en Hz.
- **TX Power (dBm).**
- **Receiver Noise Figure (dB).**
- **Receiver Effective Noise Temperature (K).**
- **Receiver Bandwidth (Hz).** Ancho de banda esperado del receptor del satélite medido en Hz.
- **Antenna TX.** Una instancia del bloque de Antena (descrito más adelante) como la antena utilizada para la transmisión en el satélite.
- **Frequency RX (Hz).** Valor de frecuencia utilizado para la recepción del satélite medido en Hz.
- **Antenna RX.** Una instancia del bloque de Antena (descrito más adelante) como la antena utilizada para la recepción en el satélite.

3.4.3.5.- Antenna

En gr-leo, se admiten varios tipos de antenas, incluyendo yagi, helicoidal, helicoidal cuádruple, monopolo, dipolo, reflector parabólico y antenas personalizadas. Cada tipo está representado por una clase en C++ que hereda de la clase padre `generic_antenna` e implementa las funciones virtuales puras para la ganancia, el ancho de haz y la reducción de ganancia (*roll off gain*). Los parámetros y funciones específicos de cada clase varían según el tipo de antena.



La hoja de cálculo [28] sirve como referencia para calcular la ganancia, el ancho de haz y la reducción de ganancia para cada uno de los tipos de antena admitidos en el bloque Antena de gr-leo. En el GNU Radio Companion, cada antena se representa como un bloque variable que se instancia según el valor seleccionado en el menú desplegable "Type".

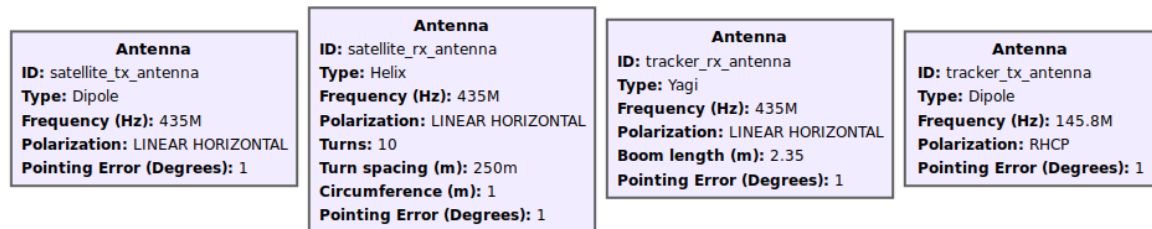


Figura 3.14.- Representación del bloque Antena del módulo OOT gr-leo en GRC mostrando 4 diferentes tipos de antena disponibles.

Los parámetros de entrada para el bloque Antena varían según el tipo de antena utilizada. Este tipo de antena se selecciona a través del parámetro "Tipo", que es un menú desplegable que muestra todos los diferentes tipos disponibles: yagi, helicoidal, helicoidal cuádruple, monopolo, dipolo, reflector parabólico y antenas personalizadas.

La ganancia de la antena Yagi se configura internamente en función del valor introducido en los parámetros de entrada Boom length (m) respecto a la Tabla 3.1 [29].

Tabla 3.1.- Valores de configuración de la antena Yagi utilizados por el módulo OOT gr-leo.

Boom length (m)	Número de elementos	Ganancia Máx. (dBi)
0.35	3	9.65
0.55	4	10.86
0.80	5	11.85
1.15	6	12.45
1.45	7	13.45
1.80	8	14.05
2.10	9	14.40
2.45	10	15.25
2.80	11	15.95
3.15	12	16.30
3.55	13	16.95
4.00	14	17.45
4.40	15	18.15
4.75	16	18.65
5.20	17	19.35
5.55	18	19.85
6.00	19	20.25



6.50	20	20.75
7.00	21	21.35
7.50	22	21.65

Por otro lado, el ancho de haz se define por medio de la Ecuación (3.2), donde el valor *Gain (dBi)* es el correspondiente en la Tabla 3.1.

$$BW = \sqrt{\frac{40000}{10^{Gain(dBi)/10}}} \quad (3.2)$$

La ganancia de Roll Off que define las pérdidas por error de apuntamiento para valores de error de ángulo de apuntamiento α mayores que cero, se ve recogida en la Ecuación (3.3).

$$Roll\ Off\ Gain\ (dB) = -10 \cdot \log\left(3382,81 \frac{\sin(2 \cdot \alpha \cdot 79,76/BW)^2}{(2 \cdot \alpha \cdot 79,76/BW)^2}\right) \quad (3.3)$$

Por último, en cuanto a los parámetros de entrada para la antena tipo Yagi son:

- **ID.** Nombre de identificación para el bloque de antena.
- **Type.** Menú desplegable para elegir el tipo de antena.
- **Frequency (Hz).** Valor de frecuencia de operación de la antena medido en Hz.
- **Polarization.** Menú desplegable para elegir el tipo de polarización. Las opciones son RHCP (polarización circular a la derecha), LHCP (polarización circular a la izquierda), vertical lineal y horizontal lineal.
- **Boom length (m).** Longitud del boom de la antena yagi medida en metros.
- **Pointing error (degrees).** Grados de error de apuntamiento entre las antenas de transmisión y recepción (TX y RX).

La ganancia de la antena Helicoidal es obtenida por gr-leo acorde a la Ecuación (3.4), donde C_λ es la longitud de la circunferencia de la hélice en metros, n es el número de giros y S_λ es el espaciado que hay entre los giros medido en metros.

$$Gain(dBi) = 10 \log(15 \cdot C_\lambda^2 \cdot n \cdot S_\lambda) \quad (3.4)$$



El ancho de haz, medido en grados, se calcula a partir de la Ecuación (3.5) y, la ganancia de Roll Off se define en la Ecuación (3.6).

$$BW = \frac{115}{c_{\lambda} \sqrt{n} S_{\lambda}} \quad (3.5)$$

$$\text{Roll Off Gain (dB)} = -10 \cdot \log\left(3382,81 \frac{\sin(2 \cdot \alpha \cdot 79,76/BW)^2}{(2 \cdot \alpha \cdot 79,76/BW)^2}\right) \quad (3.6)$$

Por último, en cuanto a los parámetros de entrada para la antena tipo Helicoidal son:

- **ID.** Nombre de identificación para el bloque de Antena.
- **Type.** Menú desplegable para elegir el tipo de antena.
- **Frequency (Hz).** Valor de la frecuencia de operación para la antena medido en Hz.
- **Polarization.** Menú desplegable para elegir el tipo de polarización. Las opciones son RHCP (polarización circular de mano derecha), LHCP (polarización circular de mano izquierda), Vertical Lineal y Horizontal Lineal.
- **Turns.** Número de vueltas en la antena helicoidal.
- **Turn spacing (m).** Espaciado entre cada vuelta en la antena de hélice medido en metros.
- **Circunference (m).**
- **Pointing error (degrees).** Grados de error de apuntamiento entre las antenas de TX y RX.

La ganancia y el ancho de haz de la antena Helicoidal Cuádruple están fijos con un valor de 4 dBi y 150 grados respectivamente. Por otro lado, la ganancia de roll off se describe por medio de la Ecuación (3.7).

$$\text{Roll Off Gain (dB)} = -1,5 \cdot (-4 + 10 \cdot \log(1,256 \cdot (1 + \cos(\alpha)))) \quad (3.7)$$

Por último, en cuanto a los parámetros de entrada para la antena tipo Helicoidal Cuádruple son:

- **ID.** Nombre de identificación para el bloque de antena.
- **Type.** Menú desplegable para elegir el tipo de antena.
- **Frequency (Hz).** Valor de frecuencia de operación para la antena medido en Hz.



- **Polarization.** Menú desplegable para elegir el tipo de polarización. Las opciones son RHCP (polarización circular de mano derecha), LHCP (polarización circular de mano izquierda), vertical lineal y horizontal lineal.
- **Loop (m).** Longitud de la antena de bucle medido en metros.
- **Pointing error (degrees).** Grados de error de apuntamiento entre las antenas de transmisión y recepción.

La ganancia de la antena Monopolo y su ancho de haz son valores fijos de 2.15 dBi y 156.2 grados respectivamente. No obstante, nótese que este valor dado a la ganancia del monopolo es inexacto a la realidad, teniendo típicamente un valor de 5.2 dB. Por otro lado, la ganancia de roll off se establece como -10 veces el logaritmo (base 10) del coseno de 90 menos el ángulo de error de puntería cuando el ángulo de error de apuntamiento es inferior a 100 grados. Para ángulos de error de apuntamiento mayores o iguales a 100 grados, no habría señal, o bien, una atenuación infinita.

En cuanto a los parámetros de entrada para la antena tipo Monopolo son:

- **ID.** Nombre de identificación para el bloque de Antena.
- **Type.** Menú desplegable para elegir el tipo de antena.
- **Frequency (Hz).** Valor de la frecuencia de operación para la antena medido en Hz.
- **Polarization.** Menú desplegable para elegir el tipo de polarización. Las opciones son RHCP (polarización circular de mano derecha), LHCP (polarización circular de mano izquierda), Vertical Lineal y Horizontal Lineal.
- **Pointing error (degrees).** Grados de error de apuntamiento entre las antenas de transmisión y recepción.

La ganancia de la antena Dipolo y su ancho de haz son valores fijos de 2.15 dBi y 156.2 grados respectivamente. Por otro lado, al igual que con la antena Monopolo, la ganancia de roll off se establece como -10 veces el logaritmo (base 10) del coseno del ángulo de error de apuntamiento cuando el ángulo de error de apuntamiento es inferior a 90 grados.

En cuanto a los parámetros de entrada para la antena tipo Dipolo son:

- **ID.** Nombre de identificación para el bloque de Antena.
- **Type.** Menú desplegable para elegir el tipo de antena.
- **Frequency (Hz).** Valor de la frecuencia de operación para la antena medido en Hz.
- **Polarization.** Menú desplegable para elegir el tipo de polarización. Las opciones son RHCP (polarización circular de mano derecha), LHCP



(polarización circular de mano izquierda), Vertical Lineal y Horizontal Lineal.

- **Pointing error (degrees).** Grados de error de apuntamiento entre las antenas de transmisión y recepción.

La ganancia de la antena de Reflector Parabólico se define en base a la fórmula de la Ecuación (3.8), donde D es el diámetro del plato reflector en metros, f es la frecuencia de operación en KHz y k es la eficiencia de apertura en porcentaje.

$$Gain (dBi) = 20,4 + 20 \cdot \log(D \cdot f/10^3) + 10 \cdot \log(k/100) \quad (3.8)$$

Por otro lado, el ancho de haz viene definido por la fórmula de la Ecuación (3.9).

$$BW (degrees) = \frac{21}{D \cdot (f/10^3)} \quad (3.9)$$

En cuanto a la ganancia de roll off, nuevamente dependiente del ángulo de error de apuntamiento entre antenas, se define mediante la fórmula de la Ecuación (3.10).

$$Roll\ off\ Gain\ (dB) = -10 \cdot \log\left(3382,81 \frac{\sin(2 \cdot \alpha \cdot 79,76/BW)^2}{(2 \cdot \alpha \cdot 79,76/BW)^2}\right) \quad (3.10)$$

Por último, respecto a la antena de Reflector Parabólico, en cuanto a los parámetros de entrada son:

- **ID.** Nombre de identificación para el bloque de Antena.
- **Type.** Menú desplegable para elegir el tipo de antena.
- **Frequency (Hz).** Valor de la frecuencia de operación para la antena medido en Hz.
- **Polarization.** Menú desplegable para elegir el tipo de polarización. Las opciones son RHCP (polarización circular de mano derecha), LHCP (polarización circular de mano izquierda), Vertical Lineal y Horizontal Lineal.
- **Diameter (m).** Diámetro del reflector parabólico medido en metros.
- **Aperture efficiency (%).** Eficiencia de apertura para el reflector parabólico medida en porcentaje.
- **Pointing error (degrees).** Grados de error de apuntamiento entre las antenas de transmisión y recepción.



Para terminar, respecto la antena Personalizable, gr-leo facilita la opción de insertar los valores deseados para la ganancia, el ancho de haz e incluso la ganancia de roll off. Respecto a este último parámetro cabe destacar que GNU Radio permite la posibilidad de introducir una fórmula escrita en Python en función de una variable que actúe como error de ángulo de apuntamiento para mayor precisión en la simulación.

Los parámetros de entrada para la antena Personalizable son:

- **ID.** Nombre de identificación para el bloque de Antena.
- **Type.** Menú desplegable para elegir el tipo de antena.
- **Frequency (Hz).** Valor de la frecuencia de operación para la antena medido en Hz.
- **Polarization.** Menú desplegable para elegir el tipo de polarización. Las opciones son RHCP (polarización circular de mano derecha), LHCP (polarización circular de mano izquierda), Vertical Lineal y Horizontal Lineal.
- **Gain (dBiC).** Ganancia medida en dBiC.
- **Roll-off Gain (dB).** Ganancia de roll off medida en dB.
- **Beamwidth (degrees).** Ancho de haz de la antena medido en grados.
- **Pointing error (degrees).** Error del ángulo de apuntamiento entre antenas de transmisión y recepción.

3.4.4.- Limitaciones

A continuación, se listan algunas de las limitaciones que se han encontrado a la herramienta gr-leo.

- **Perdidas por polarización cruzada.** Aunque el programa tiene en cuenta el tipo de polarización utilizado para definir las antenas como entrada, este parámetro no afecta a ningún cálculo ni decisión tomada durante la simulación. Sería recomendable que este parámetro se incluyera como factor en el cálculo de atenuación cuando haya una falta de coincidencia en la polarización.
- **Margen de enlace basado en la modulación requerida, codificación de canal y probabilidad de error.** Actualmente, es posible calcular la relación señal-ruido (SNR) de la señal recibida simulada. Sin embargo, sería ventajoso también incluir el cálculo del margen de enlace resultante que considere la SNR necesaria para una modulación deseada específica, junto con una codificación de canal específica, para lograr una probabilidad de error dada o al menos la opción de ingresar la SNR requerida para poder detectar los momentos en que el enlace sería insuficiente.



- **Actualización de parámetros en tiempo de ejecución.** Además de la interacción normal de los bloques de GNU Radio con actualizaciones de variables en tiempo de ejecución, puede ser útil poder cambiar algunos parámetros durante la ejecución tanto para los bloques del satélite como para los bloques de la estación terrena, como su potencia de transmisión, la figura de ruido del receptor o la temperatura efectiva de ruido.
- **Pérdidas extra.** Sería también beneficiosa la posibilidad de añadir pérdidas extras no genéricas al canal para experimentar con situaciones excepcionales y poder preparar los sistemas frente a todas las situaciones posibles.

3.4.5.- Ejemplo

El esquema mostrado en la Figura 3.15, desarrollado en GRC, es un diseño completo que hace uso de todos los bloques y perturbaciones de canal disponibles. Incluye un bloque Random Source que produce símbolos GMSK (Gaussian Minimum Shift Keying) junto con el bloque GMSK Mod, los cuales son amplificados y transmitidos a través del canal con todos los problemas aplicados. La señal recibida es procesada y se muestran gráficos de tiempo y frecuencia (Figura 3.16). La pantalla de tiempo está configurada para actualizarse con cualquier stream tag que se adjunte a la señal a través del bloque del modelo de canal. Además, el diagrama de bloques almacena los datos generados y etiquetados como un archivo CSV en la ruta especificada, lo que permite un análisis y procesamiento adicionales (Figura 3.17) del balance de enlace generado en función del tiempo.

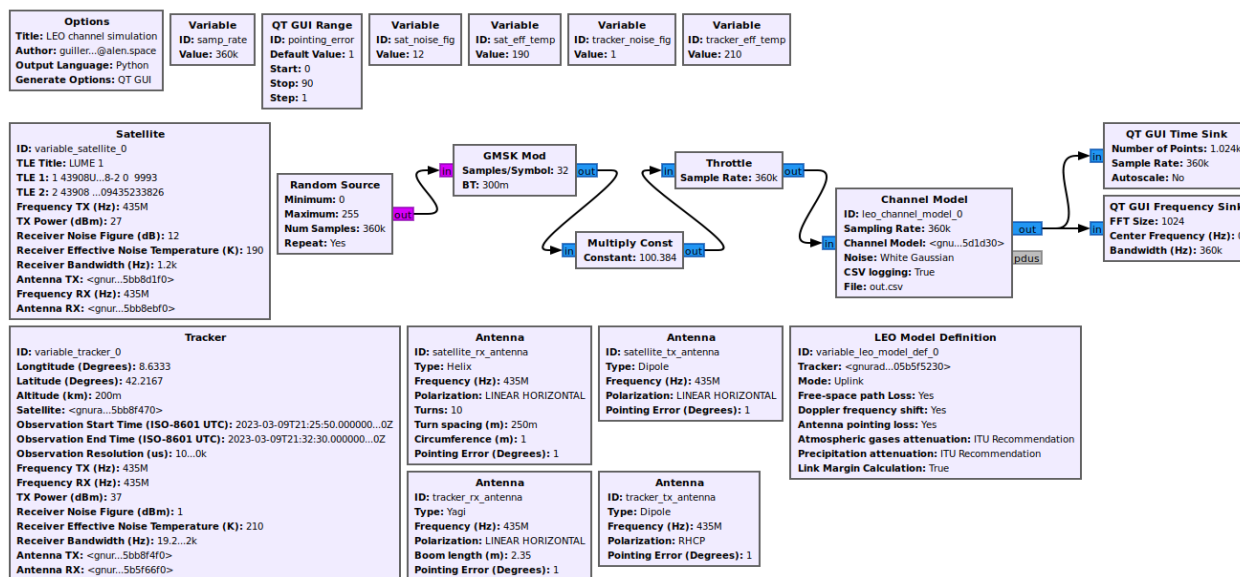


Figura 3.15.- Diagrama de bloques desarrollado en GRC para el ejemplo donde se emplean todos los bloques y funcionalidades del módulo OOT gr-leo.

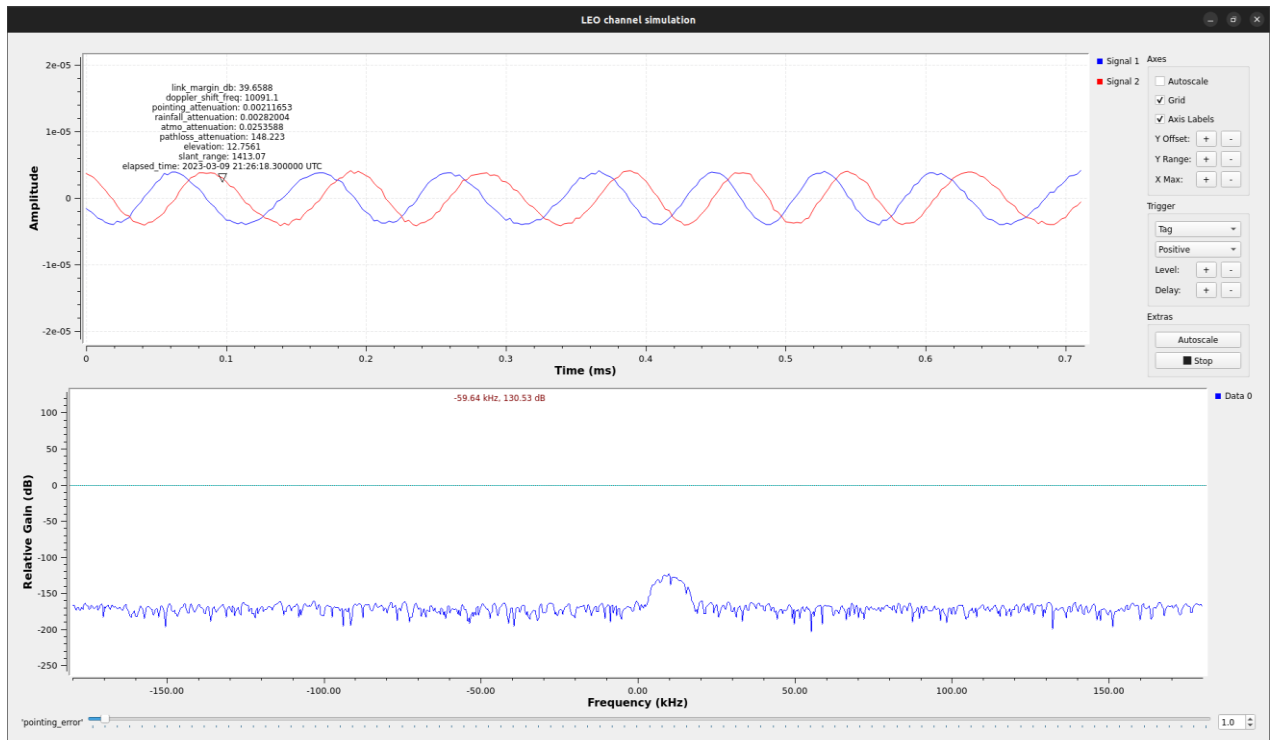


Figura 3.16.- Captura de pantalla del resultado de la ejecución del diagrama de bloques de la Figura 3.15.

```

out.csv
1  Elapsed Time (us),Slant Range (km),Elevation (degrees),Path Loss (dB),Atmospheric Loss (dB),Rainfall Loss (dB),Pointing Loss (dB),Doppler Shift (Hz),Link Margin (dB)
2  2023-03-09 21:25:50.200000 UTC,1609.91,9.5117,149.356,0.0254236,0.00347571,0.00211653,10226.5,38.5253
3  2023-03-09 21:25:50.300000 UTC,1609.21,9.52224,149.352,0.0254233,0.00347322,0.00211653,10226.1,38.5291
4  2023-03-09 21:25:50.400000 UTC,1608.5,9.53279,149.348,0.025423,0.00347073,0.00211653,10225.7,38.533
5  2023-03-09 21:25:50.500000 UTC,1607.8,9.54334,149.344,0.0254227,0.00346823,0.00211653,10225.4,38.5368
6  2023-03-09 21:25:50.600000 UTC,1607.09,9.5539,149.341,0.0254224,0.00346574,0.00211653,10225.38,5406
7  2023-03-09 21:25:50.700000 UTC,1606.39,9.56446,149.337,0.0254221,0.00346325,0.00211653,10224.6,38.5444
8  2023-03-09 21:25:50.800000 UTC,1605.68,9.57503,149.333,0.0254218,0.00346076,0.00211653,10224.2,38.5482
9  2023-03-09 21:25:50.900000 UTC,1604.98,9.58561,149.329,0.0254215,0.00345827,0.00211653,10223.8,38.552
10 2023-03-09 21:25:51.000000 UTC,1604.27,9.59619,149.325,0.0254212,0.00345579,0.00211653,10223.5,38.5558
11 2023-03-09 21:25:51.100000 UTC,1603.57,9.60678,149.322,0.0254209,0.0034533,0.00211653,10223.1,38.5596
12 2023-03-09 21:25:51.200000 UTC,1602.86,9.61738,149.318,0.0254206,0.00345081,0.00211653,10222.7,38.5635
13 2023-03-09 21:25:51.300000 UTC,1602.16,9.62798,149.314,0.0254203,0.00344833,0.00211653,10222.3,38.5673
14 2023-03-09 21:25:51.400000 UTC,1601.46,9.63859,149.31,0.02542,0.00344585,0.00211653,10221.9,38.5711
15 2023-03-09 21:25:51.500000 UTC,1600.75,9.64921,149.306,0.0254197,0.00344337,0.00211653,10221.5,38.5749
16 2023-03-09 21:25:51.600000 UTC,1600.05,9.65983,149.302,0.0254194,0.00344088,0.00211653,10221.2,38.5788
17 2023-03-09 21:25:51.700000 UTC,1599.34,9.67046,149.299,0.0254191,0.0034384,0.00211653,10220.8,38.5826
18 2023-03-09 21:25:51.800000 UTC,1598.64,9.68109,149.295,0.0254188,0.00343593,0.00211653,10220.4,38.5864
19 2023-03-09 21:25:51.900000 UTC,1597.93,9.69174,149.291,0.0254185,0.00343345,0.00211653,10220.38,5902
20 2023-03-09 21:25:52.000000 UTC,1597.23,9.70238,149.287,0.0254182,0.00343097,0.00211653,10219.6,38.5941
21 2023-03-09 21:25:52.100000 UTC,1596.52,9.71304,149.283,0.0254179,0.0034285,0.00211653,10219.2,38.5979
22 2023-03-09 21:25:52.200000 UTC,1595.82,9.7237,149.279,0.0254176,0.00342602,0.00211653,10218.8,38.6017
23 2023-03-09 21:25:52.300000 UTC,1595.12,9.73437,149.276,0.0254173,0.00342355,0.00211653,10218.4,38.6056
24 2023-03-09 21:25:52.400000 UTC,1594.41,9.74504,149.272,0.025417,0.00342108,0.00211653,10218.38,6094
25 2023-03-09 21:25:52.500000 UTC,1593.71,9.75572,149.268,0.0254167,0.00341861,0.00211653,10217.7,38.6133
26 2023-03-09 21:25:52.600000 UTC,1593.0,9.76641,149.264,0.0254165,0.00341614,0.00211653,10217.3,38.6171
27 2023-03-09 21:25:52.700000 UTC,1592.3,9.7771,149.26,0.0254162,0.00341367,0.00211653,10216.9,38.6209
28 2023-03-09 21:25:52.800000 UTC,1591.6,9.7878,149.256,0.0254159,0.0034112,0.00211653,10216.5,38.6248
29 2023-03-09 21:25:52.900000 UTC,1590.89,9.79851,149.253,0.0254156,0.00340874,0.00211653,10216.1,38.6286
30 2023-03-09 21:25:53.000000 UTC,1590.19,9.80922,149.249,0.0254153,0.00340627,0.00211653,10215.7,38.6325
31 2023-03-09 21:25:53.100000 UTC,1589.48,9.81994,149.245,0.025415,0.00340381,0.00211653,10215.3,38.6363
32 2023-03-09 21:25:53.200000 UTC,1588.78,9.83067,149.241,0.0254147,0.00340134,0.00211653,10214.9,38.6402
33 2023-03-09 21:25:53.300000 UTC,1588.07,9.8414,149.237,0.0254144,0.00339888,0.00211653,10214.5,38.644
34 2023-03-09 21:25:53.400000 UTC,1587.37,9.85214,149.233,0.0254142,0.00339642,0.00211653,10214.1,38.6479
35 2023-03-09 21:25:53.500000 UTC,1586.67,9.86289,149.23,0.0254139,0.00339396,0.00211653,10213.7,38.6517
36 2023-03-09 21:25:53.600000 UTC,1585.96,9.87364,149.226,0.0254136,0.0033915,0.00211653,10213.3,38.6556
37 2023-03-09 21:25:53.700000 UTC,1585.26,9.8844,149.222,0.0254133,0.00338904,0.00211653,10212.9,38.6595
    
```

Figura 3.17.- Captura de pantalla del archivo CSV donde se guardan los datos del balance de enlace generados por el módulo OOT gr-leo.

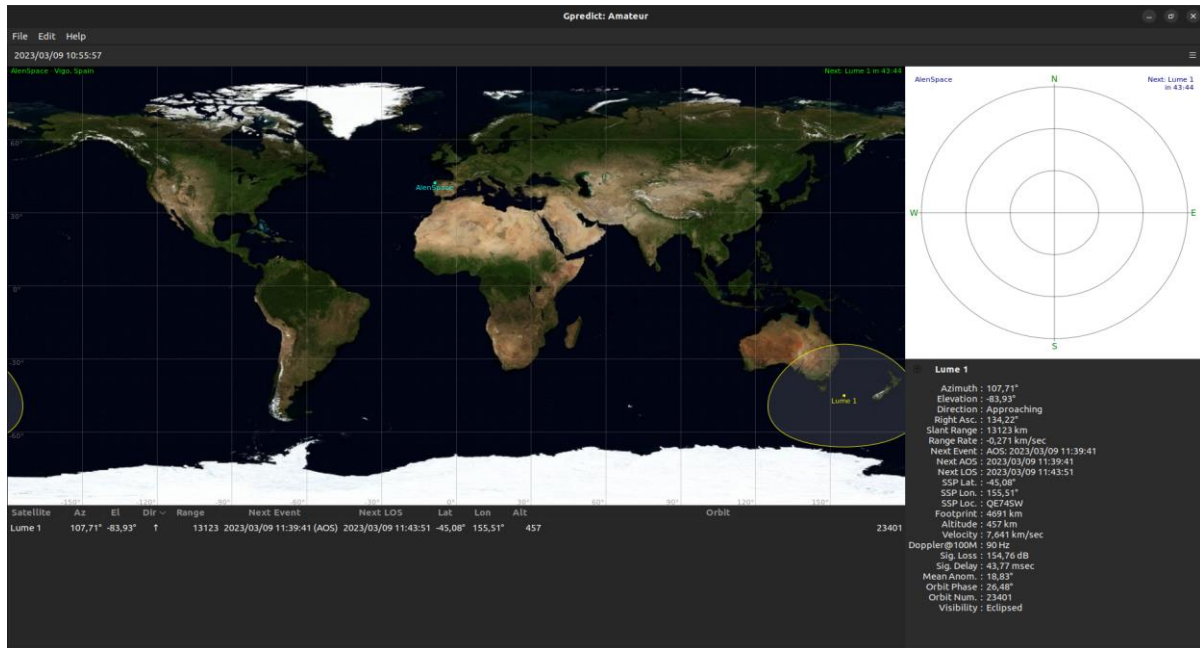


Figura 3.18.- Captura de pantalla de la herramienta GPredict con la configuración de seguimiento del CubeSat Lume 1.

Para este ejemplo se ha utilizado los datos TLE del satélite Lume 1 [30]. Estos datos fueron adquiridos de [31] e insertados como los parámetros de entrada correspondientes en el bloque Satellite gr-leo. Los datos TLE utilizados son los siguientes:

LUME 1

```
1 43908U 18111AJ 23067.24534328 .00042031 00000+0 10718-2 0 9993
2 43908 97.1571 312.3834 0013645 199.2147 160.8586 15.39309435233826
```

Para una mayor precisión en el ejemplo, se utilizó GPredict, herramienta explicada y presentada anteriormente, para adquirir información sobre el estado de Lume 1 y los posibles tiempos de AOS. Una captura de pantalla de este programa configurado para Lume 1 se muestra en la Figura 3.19.

Además, se muestra una captura de pantalla de los detalles del próximo paso, en el momento tomado, para Lume 1 en la siguiente figura. Estos datos se han extraído adecuadamente para introducir los parámetros de entrada correctos para los tiempos de inicio y finalización de la observación.



Pass details for Lume 1 (orbit 23408)

Data Polar Az/El

Time	Az	El	Range	Footp
2023/03/09 21:24:43	41,08°	0,00°	2449	4678
2023/03/09 21:25:07	44,26°	1,07°	2331	4675
2023/03/09 21:25:31	47,76°	2,13°	2221	4672
2023/03/09 21:25:54	51,62°	3,17°	2118	4669
2023/03/09 21:26:18	55,84°	4,16°	2024	4667
2023/03/09 21:26:42	60,46°	5,09°	1940	4664
2023/03/09 21:27:05	65,48°	5,93°	1869	4661
2023/03/09 21:27:29	70,86°	6,64°	1811	4659
2023/03/09 21:27:53	76,56°	7,18°	1767	4656
2023/03/09 21:28:16	82,50°	7,53°	1740	4653
2023/03/09 21:28:40	88,58°	7,65°	1729	4651
2023/03/09 21:29:04	94,68°	7,55°	1735	4648
2023/03/09 21:29:27	100,67°	7,22°	1758	4646
2023/03/09 21:29:51	106,45°	6,70°	1797	4643
2023/03/09 21:30:15	111,92°	5,99°	1852	4641
2023/03/09 21:30:38	117,03°	5,16°	1920	4639
2023/03/09 21:31:02	121,75°	4,22°	2001	4637
2023/03/09 21:31:26	126,07°	3,21°	2093	4634
2023/03/09 21:31:49	130,01°	2,16°	2194	4632
2023/03/09 21:32:13	133,58°	1,08°	2303	4630
2023/03/09 21:32:37	136,83°	-0,00°	2420	4629

Print Save Close

Figura 3.19.- Captura de pantalla de la ventana donde se muestran los detalles de la información de la ventana de AOS para el CubeSat Lume 1.

3.5.- LibCSP

En este apartado se introduce la librería por medio de la cual se implementa y se construye el protocolo CSP. También se presenta la propia estructura y funcionamiento de este protocolo.

3.5.1.- Introducción

El CubeSat Space Protocol (CSP) es una pequeña pila de protocolos escrito en C. CSP fue diseñado y desarrollado por la empresa GomSpace [32] para facilitar la comunicación entre sistemas integrados y distribuidos en redes más pequeñas, como los CubeSats. Su diseño sigue el modelo TCP/IP e incluye un protocolo de transporte, un protocolo de enrutamiento y varias interfaces de capa MAC. La base de LibCSP [33] incluye un enrutador, una API de socket orientada a la conexión y grupos de conexiones/mensajes.

El protocolo se basa en una cabecera de 32 bits que contiene información de la capa de transporte y de red. Su implementación está diseñada para, aunque no limitada a, sistemas integrados como el microprocesador AVR de 8 bits y el ARM y AVR de 32 bits



de Atmel. La implementación está escrita en GNU C y actualmente se ha adaptado para ejecutarse en FreeRTOS, Linux (POSIX), MacOS y Windows. No obstante, las principales plataformas que se utilizan son FreeRTOS y Linux.

Con la librería LibCSP, la idea es brindar a los desarrolladores de subsistemas de CubeSats las mismas características de una pila TCP/IP, pero sin agregar la enorme sobrecarga de la cabecera IP. De esta forma, su simple implementación permite que un sistema de 8 bits esté completamente conectado a la red. Esto permite que todos los subsistemas proporcionen sus servicios en el mismo nivel de red, sin requerir ningún nodo maestro.

Utilizar una arquitectura orientada a servicios brinda varias ventajas en comparación con la topología tradicional maestro/esclavo utilizada en muchas misiones de CubeSats entre las cuales destacan las siguientes:

- Proporciona un protocolo de red estandarizado que permite a todos los subsistemas de un CubeSat comunicarse entre sí de manera efectiva y eficiente adaptándose así a las limitaciones de memoria y procesamiento propias de los nanosatélites.
- Los servicios se desacoplan de manera efectiva para minimizar las dependencias entre subsistemas. Esto permite a los subsistemas trabajar de manera autónoma y permite una mayor flexibilidad y capacidad de mantenimiento.
- Además de las descripciones en el contrato de servicio, los servicios de CSP ocultan su lógica del mundo exterior. Esto significa que los detalles de implementación de los servicios están ocultos de los subsistemas externos y se pueden cambiar sin afectar la funcionalidad externa.
- Promueve la reutilización de servicios dividiendo la lógica en servicios independientes y autónomos. Esto permite que los servicios se utilicen en múltiples subsistemas y reduce el tiempo y los costos de desarrollo.
- Proporciona autonomía a los servicios al permitir que controlen la lógica que encapsulan. Esto significa que los servicios pueden funcionar de manera independiente y tomar decisiones basadas en su lógica interna sin necesidad de la intervención de otros subsistemas.
- Permite la fácil adición de servicios redundantes al bus para garantizar la disponibilidad y confiabilidad del sistema. Los servicios redundantes se pueden activar automáticamente en caso de fallos del servicio principal, lo que permite una recuperación más rápida y eficiente del sistema.
- Al distribuir la complejidad del sistema en múltiples servicios en la red en lugar de depender de un único nodo maestro se reduce el riesgo de un único punto de fallo. De esta forma se aumenta la disponibilidad y confiabilidad del sistema y mejora la capacidad de recuperación en caso de fallos.



3.5.2.- Conceptos básicos (v.1.6)

En primer lugar, es importante remarcar que la versión utilizada en este proyecto ha sido la 1.6. La razón por la que se ha escogido esta versión ha sido que es la misma que emplea la empresa Alén Space para desarrollar sus misiones.

Bit offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Priority		Source				Destination				Destination Port				Source Port				Reserved				H M A C	X T E A	R D P	C R C						
32	Data (0 – 65,535 bytes)																															

Figura 3.20.- Formato de las tramas de la red CSP versión 1.6. [34]

El protocolo CSP, en su versión 1, define la estructura de las tramas utilizadas para la comunicación en sistemas de CubeSats según muestra la Figura 3.20. Una trama CSP consta de una cabecera de 32 bits y una carga útil de datos de hasta 65.535 bytes. La cabecera contiene información sobre el origen y destino del paquete, así como algunas opciones de gestión de flujo y control de errores como autenticación (HMAC), encriptación (XTEA), conexiones (UDP/RDP) y sumas de comprobación de errores (CRC).

En cuanto a la estructura de la carga útil de datos, CSP versión 1 permite el uso de diferentes tipos de paquetes, como paquetes de datos, paquetes de comandos y paquetes de respuesta. La carga útil de datos puede contener información específica para cada tipo de paquete, como datos a transmitir, comandos a ejecutar o respuestas a solicitudes anteriores.

Por otro lado, en la Figura 3.21 se muestra una visión general conceptual de los diferentes bloques CSP utilizando en este caso, la interfaz CAN. Entre estas funcionalidades básicas destacan: buffer, conexión, enviar, recibir y enrutamiento.

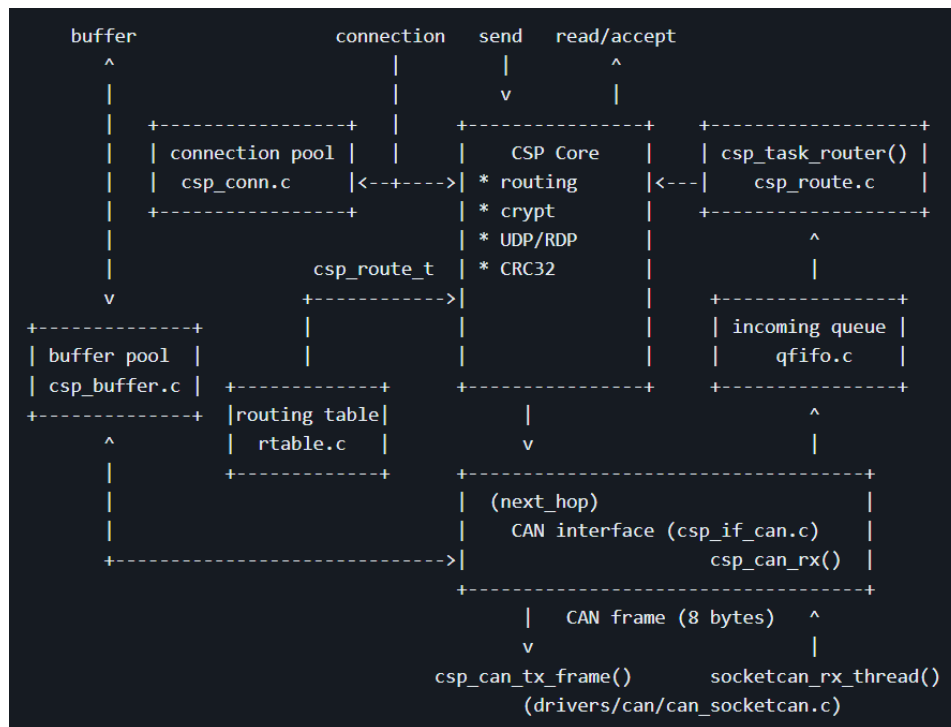


Figura 3.21.- Esquema conceptual de los diferentes bloques CSP haciendo uso de la interfaz CAN [35].

En la inicialización de CSP se asignan los búferes, los cuales tienen un tamaño fijo y están almacenados en una cola que permite acceder a ellos de manera rápida y segura desde diferentes contextos de tarea e interrupción. Además, se utiliza un sistema de "Zero-Copy" para evitar copiar los datos entre diferentes búferes y, de esta manera, mejorar la velocidad y eficiencia de la transferencia de datos. Adicionalmente, se emplea un concepto de "padding" para añadir bytes adicionales en el búfer y permitir que se puedan agregar encabezados sin tener que copiar la carga útil. Esto se utiliza en la interfaz I2C para transmitir los datos al controlador.

Respecto a la conexión, CSP permite intercambiar información utilizando tanto modos de comunicación no orientados como orientados a conexión. Durante la fase de inicialización se asigna un número predefinido de conexiones que varía según la aplicación. Para llevar a cabo esta asignación, se utilizan diferentes funciones, implementadas en la librería, como `csp_connect()` para establecer una conexión como cliente, `csp_socket()` para crear un socket de servidor y `csp_accept()` para aceptar una conexión entrante. Además, durante la inicialización se asigna una cola de recepción para cada conexión, con una longitud predeterminada que se especifica en la configuración.

El flujo de datos desde la aplicación al controlador se realiza en varios pasos. En primer lugar, se establece una conexión mediante funciones como `csp_connect()` o `csp_accept()` si se utiliza una comunicación orientada a la conexión. A continuación, se obtiene un paquete del grupo de búferes mediante `csp_buffer_get()`. Después, se agrega la



carga útil de datos al paquete y se envía utilizando `csp_send()` o `csp_sendto()`. CSP busca la ruta de destino utilizando la tabla de enrutamiento y llama a la función `nextthop()` de la interfaz resuelta. Finalmente, la interfaz, en este caso la interfaz CAN, divide el paquete en tramas CAN de 8 bytes y las envía al controlador. Este flujo de datos permite la transferencia de información desde la aplicación al controlador de manera efectiva y controlada.

Por otro lado, el flujo de datos desde el controlador hacia la aplicación se puede describir en los siguientes pasos: primero, la capa del controlador envía las tramas de datos sin procesar a la interfaz, que en este caso son tramas CAN. Luego, la interfaz adquiere un búfer libre para ensamblar las tramas CAN en un paquete completo. Una vez que la interfaz ha ensamblado correctamente un paquete, este se encola para el enrutamiento, permitiendo el desacoplamiento de la interfaz y los controladores. A continuación, el enrutador recoge el paquete de la cola de entrada y lo enruta hacia su destino, ya sea local o hacia otra interfaz. La aplicación espera nuevos paquetes en su cola de recepción y puede procesarlos utilizando funciones como `csp_read()` o `csp_accept` (en caso de ser un socket de servidor). Finalmente, la aplicación puede enviar el paquete utilizando `csp_send()` o liberar el paquete utilizando `csp_buffer_free()`. Este flujo de datos garantiza la transferencia de información desde el controlador hacia la aplicación de manera eficiente y controlada.

Por último, CSP enruta paquetes a través de la tabla de enrutamiento. La tabla de enrutamiento contiene registros que especifican la interfaz a través de la cual enviar el paquete y, opcionalmente, una dirección predeterminada para el caso en que el remitente no pueda llegar directamente al destinatario en una de sus redes conectadas.

Hay dos implementaciones de tabla de enrutamiento: estática y CIDR (Clase Inter-Dominio sin Clase). La tabla de enrutamiento estática es más rápida, pero requiere más configuración, mientras que la tabla CIDR es más fácil de configurar, pero más lenta. Los registros de enrutamiento se pueden configurar utilizando cadenas de texto que incluyen la dirección de destino, la máscara (opcional), el nombre de la interfaz y la dirección predeterminada (opcional).

3.5.3.- Terminología de red

La terminología que CSP utiliza está orientada a redes de forma similar al modelo TCP/IP. Esta red puede ser configurada para varias topologías diferentes, siendo la más común la creación de dos segmentos, uno para el satélite y otro para la estación terrestre. A continuación, en la Figura 3.22, se presenta un esquema que muestra esta topología.

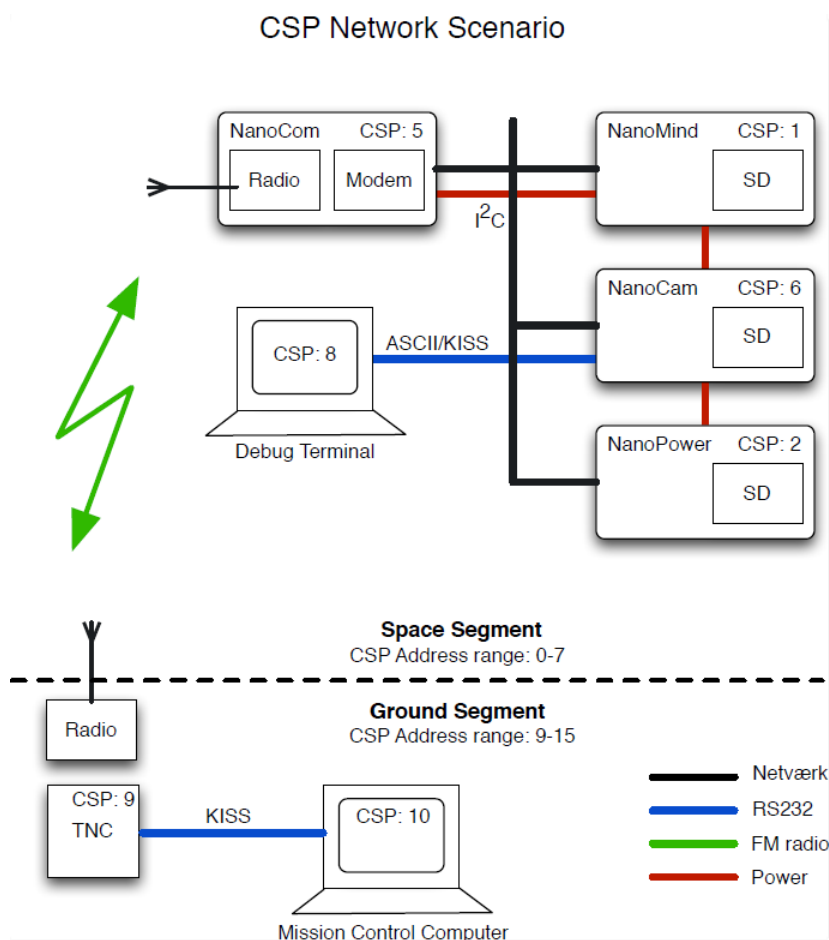


Figura 3.22.- Esquema de topología típica de una red CSP [36].

Analizando la Figura 3.22, se divide el rango de direcciones del 0 al 15 en dos segmentos iguales con el fin de facilitar su enrutamiento en la red. El primer segmento, llamado "Ground Segment", incluye todas las direcciones que empiezan con el número binario 1, mientras que el segundo segmento, llamado "Space Segment", incluye todas las direcciones que empiezan con 0 binario. Cabe destacar que, a pesar de que desde la introducción de CSP 1.0 se han añadido 16 direcciones adicionales al espacio de direcciones, ampliando el rango a 32 direcciones del 0 al 31, la mayoría de los productos de GomSpace todavía utilizan el antiguo rango de direcciones del 0 al 15 por motivos de compatibilidad con versiones anteriores.

En relación con los puertos utilizados en CSP, el rango de puertos se divide en tres segmentos reconfigurables. Los puertos del 0 al 7 se reservan para servicios generales, como ping y estado del búfer, y son implementados por el controlador de servicios CSP. Los puertos del 8 al 47 se destinan a servicios específicos del subsistema, lo que permite la diferenciación y gestión de diferentes funcionalidades dentro del sistema. Por último, los puertos del 48 al 63 se utilizan como puertos efímeros para conexiones salientes. Además, los bits del 28 al 31 se emplean para marcar paquetes con cifrado HMAC, XTEA, cabecera RDP y suma de comprobación CRC32, proporcionando mecanismos de seguridad y



detección de errores en la comunicación. Estos rangos y marcas de puerto permiten un enrutamiento y manejo adecuados de los paquetes en el protocolo CSP versión 1.

3.5.4.- Python bindings

Pese a que la librería LibCSP esta desarrollada en C y C++, se ha hecho uso de esta tecnología por medio del lenguaje de programación Python. Esto se debe a la simplicidad que aporta el lenguaje Python y que la propia librería incluye unos bindings o enlaces de C/C++ a Python.

Pese a que el número de funciones que ofrece en la versión 1.6, utilizada para este proyecto, es reducida frente a la librería total, es suficiente para el desarrollo de este proyecto. No obstante, cabe destacar que es posible añadir nuevos enlaces a funciones sin mayor problema si fuese necesario.

Por otro lado, cabe mencionar que, para la correcta y cómoda ejecución de los programas desarrollados para ejecutar esta librería, se ha tenido que desarrollar un pequeño script de Shell para especificar las rutas de las variables de entorno Linux LD_LIBRARY_PATH y PYTHONPATH para que sepan dónde buscar los archivos compilados de la librería LibCSP. El contenido de dicho script es el que aparece a continuación, aunque puede variar un poco en de si se desea introducir más argumentos de entrada junto con el script de Python a ejecutar.

```
#!/usr/bin/env bash
LD_LIBRARY_PATH=/libcsp/build PYTHONPATH=/libcsp/build python3 $1
```

3.6.- ZeroMQ

La arquitectura de la red de la estación terrena simulada se emplea mediante una red CSP sobre ZeroMQ.

ZeroMQ (Zero Message Queue) es una librería de mensajería de alto rendimiento y bajo nivel que proporciona un mecanismo de comunicación flexible para construir aplicaciones distribuidas. Se enfoca en la transmisión eficiente de mensajes entre procesos, dispositivos o nodos de una red. ZeroMQ se basa en estrategias de comunicación como por ejemplo publicador-suscriptor, solicitud-respuesta y push-pull, lo que permite establecer canales de comunicación asincrónica y sincrónica de manera sencilla.

Una de las características distintivas de ZeroMQ es su capacidad para adaptarse a diversas arquitecturas y escenarios de comunicación. Proporciona abstracciones de sockets, en las que los usuarios pueden enviar y recibir mensajes de manera transparente, sin preocuparse por la complejidad de la comunicación en red subyacente. Además, ZeroMQ



ofrece un enfoque liviano y eficiente, lo que lo hace ideal para aplicaciones que requieren alta escalabilidad y baja latencia.

El protocolo CSP se utiliza en la interconexión de nodos dentro de un segmento de red, y ZMQ actúa como una interfaz para proporcionar dicha comunicación. Esta configuración es comúnmente empleada para conectar el MCS con los diversos subsistemas de una estación terrena por medio de la estrategia publicador-suscriptor. El enrutamiento de los paquetes, por otro lado, se lleva a cabo mediante un proxy que está integrado en la biblioteca LibCSP, simplificando así el proceso de intercambio de datos entre los componentes del sistema.

3.6.1.- Publicador-Subscriber

La filosofía de publicador-suscriptor en ZeroMQ es un patrón de comunicación en el que los componentes se conectan de manera asíncrona y desacoplada. En este modelo, los componentes se dividen en dos roles principales: el publicador y el suscriptor.

El publicador es responsable de enviar mensajes a los suscriptores pudiendo enviar mensajes a múltiples suscriptores simultáneamente. Los mensajes enviados por el publicador se denominan "mensajes publicados". Cada mensaje publicado puede estar asociado con un tema o tópico específico. Por otro lado, los suscriptores, por otro lado, pueden estar interesados en recibir mensajes de uno o más publicadores en función de sus necesidades y pueden elegir suscribirse a todos los mensajes publicados o solo a aquellos mensajes que estén etiquetados con temas específicos de interés.

La comunicación entre los publicadores y los suscriptores en ZeroMQ se establece a través de un elemento intermediario, el socket. Los sockets en ZeroMQ implementan diferentes patrones de comunicación, y en el caso de la filosofía publicador/suscriptor, se utilizan dos tipos de sockets: "PUB" (publicador) y "SUB" (suscriptor). El socket de tipo publicador envía publicaciones a todos los suscriptores que están conectados a él. No hay una relación directa entre un socket publicador y suscriptores específicos. En cambio, los suscriptores se conectan al socket de publicador y deciden qué mensajes reciben en función de suscripciones y filtros de temas. A su vez, el socket de tipo suscriptor se conecta a uno o más sockets publicadores. Además, un suscriptor puede establecer filtros de temas para recibir solo los mensajes que coincidan con esos temas específicos.

Este tipo de patrón de comunicación es la que se sigue la librería LibCSP a la hora de utilizar el protocolo ZeroMQ como interfaz de conexión entre nodos por medio de un proxy que actúa como enrutador o bróker.

3.6.2.- Proxy ZeroMQ

ZeroMQ proporciona una funcionalidad llamada "proxy" que actúa como intermediario entre los sockets de ZeroMQ, permitiendo la comunicación entre diferentes partes de un sistema distribuido. El proxy ZeroMQ puede ser utilizado para enrutar mensajes entre sockets en diferentes estrategias de comunicación, como publicador-



suscriptor, solicitud-respuesta o push-pull. Se trata de una solución eficiente y escalable, ya que puede manejar una gran cantidad de conexiones de sockets y distribuir los mensajes de manera eficiente a través de ellos. Proporciona un mecanismo simple para implementar topologías de red complejas, como patrones de estrella, bus, fan-in o fan-out.

La configuración básica del proxy ZeroMQ implica definir al menos dos sockets: uno para el frontend (entrada) y otro para el backend (salida). Los mensajes enviados al frontend se enrutan al backend y viceversa. También es posible tener múltiples sockets conectados al frontend y backend para admitir patrones más complejos. Además, ZeroMQ proporciona opciones adicionales para el control y el monitoreo del proxy pudiendo, por ejemplo, configurar reglas de filtrado para determinar qué mensajes se enrutan entre los sockets o utilizar mecanismos de seguridad para proteger la comunicación.

La implementación de un proxy ZeroMQ puede variar según las necesidades y el contexto de la aplicación sobre la cual se vaya a implementar. En el caso de este proyecto, el proxy ZeroMQ utilizado es el proporcionado por LibCSP. Dada la posibilidad de interconectividad por medio de ZeroMQ, la propia librería del protocolo CSP incluye un proxy compatible con el patrón publicador-suscriptor listo para utilizarse programado en C.



4.- DESARROLLO Y FUNCIONAMIENTO

En este capítulo se detalla el funcionamiento y desarrollo de las partes principales que construyen la herramienta protagonista de este proyecto.

A continuación, en la Figura 4.1, se muestra un esquema que ejemplifica el funcionamiento genérico de esta herramienta. Un nodo transmisor con una dirección de red CSP de valor 30 transmitiría un paquete de datos cuya dirección de destino sería la que representa al nodo receptor, es decir, la dirección 1. Estos nodos, transmisor y receptor, podrían representar la estación terrena, o MCS, y el CubeSat en una simulación de enlace ascendente y viceversa para un enlace descendente. Seguidamente, el paquete de datos pasaría por el proxy de ZeroMQ, el cual, previo a su redireccionamiento hacia el nodo cuya dirección es la 1, pasaría el paquete de datos al programa GNU Radio con la dirección 6. El programa GNU Radio, con ayuda de gr-leo y los módems implementados simularía la transmisión del paquete de datos por el canal propio de misiones LEO y, tras su recepción y demodulación, lo pasaría de nuevo al proxy de ZeroMQ. Finalmente, el Proxy pasaría el paquete de datos obtenido desde GNU Radio al destinatario final con la dirección 1, donde el nodo receptor gestionaría dichos datos en función de lo requerido.

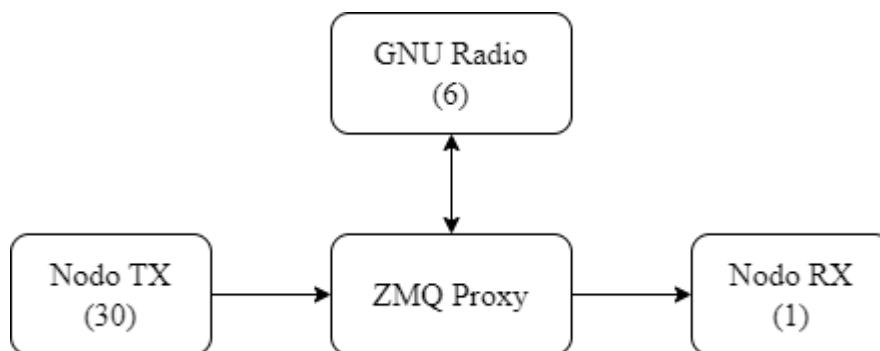


Figura 4.1.- Esquema básico de funcionamiento y conectividad a nivel de red CSP de la herramienta final desarrollada.

Según este esquema general se distinguen las diferentes secciones en las que se organiza este capítulo. En primer lugar, se cuenta con la descripción detallada de los nodos de transmisión y recepción, que representan al MCS o al CubeSat según el tipo de enlace que se desea simular. Le precede una pequeña sección en la que se expone el módulo que hace de interfaz entre GNU Radio y el proxy de ZeroMQ. Seguidamente, se introduce el desarrollo que se ha llevado a cabo de algunos módems sobre GNU Radio a la par que se muestra la posibilidad de integración de módems comerciales ya existentes. Por último, se muestra como se ha desarrollado un programa para evaluar la PER de los diferentes módems desarrollados para así ver un ejemplo de su funcionamiento al completo.



4.1.- MCS/CubeSat

Como se ha comentado con anterioridad, uno de los objetivos de este proyecto es simular la actuación transmisora y receptora de ambos, MCS y CubeSat respectivamente, para el enlace ascendente y viceversa para el enlace descendente. La implementación de dichos elementos se ha realizado por medio de scripts de Python utilizando la librería LibCSP explicada en el capítulo anterior. Concretamente se ha utilizado una arquitectura cliente-servidor orientado a conexión para este desarrollo de nodos finales de forma que el cliente actúa como transmisor que enviaría las peticiones y, por su parte, el servidor actúa como receptor, pudiendo tomar dichas peticiones y responderlas o simplemente procesar su información.

A continuación, se explican las implementaciones básicas de esta librería para la transmisión y recepción de paquetes. Nótese que los scripts de Python desarrollados se pueden encontrar en el Anexo C.

4.1.1.- Nodo Transmisor (cliente)

La descripción y modo de implementación de un nodo transmisor, o cliente, mediante la librería LibCSP se muestra a continuación.

1. Inicializar CSP y configurar y asignar las estructuras básicas por medio de un objeto de configuración que contiene datos como la propia dirección del nodo, nombre del host, el número de buffers que utilizará e incluso el tamaño de estos últimos.
2. Configurar la interfaz de ZMQ. Requiere de la dirección del mismo nodo CSP y la dirección IP en la que se encuentra el proxy de ZMQ que actúa como enrutador o bróker.
3. Cargar la tabla de rutas. Esta será cargada encima de las rutas preexistentes, posiblemente sobrescribiéndolas. Estas rutas tienen el formato <dirección>[/máscara] <interfaz> [vía] [, siguiente entrada de la tabla].

Este paso es crítico para el correcto funcionamiento de la herramienta desarrollada ya que es necesario forzar la tabla a que, aunque el camino más corto para llegar, siguiendo el ejemplo de la Figura 4.1, a la dirección 1 desde la 30, obligatoriamente pase previamente por la 6. Esto, siguiendo el formato se conseguiría de la siguiente forma: “1/5 ZMQHUB 6”.

4. Iniciar la tarea del enrutador. El enrutador se encargará de dirigir los paquetes provenientes de las colas de este y comprobar sus tiempos de espera.



5. Crear el contenido del paquete y prepararlo en el formato esperado, esto es, en bytes. Nótese que este paso puede depender de la finalidad del nodo, de forma que podría ser necesario realizarse más tarde o bajo alguna condición. No obstante, se ha seleccionado de esta forma con el objetivo de simplificar la explicación.
6. Iniciar un bucle infinito que se repita con la periodicidad que se desee.
 - a. Establecer una conexión con el servidor o receptor, pasando su dirección CSP, el puerto por el que realizar la conexión y un tiempo de espera para esperar a confirmar la conexión.
 - b. Reservar espacio para el paquete en un buffer en función del tamaño del contenido previamente definido. Habrá que tener en consideración que el tamaño máximo del buffer es de 300 bytes.
 - c. Incluir el contenido del paquete a enviar en el buffer reservado en el paso anterior.
 - d. Enviar el contenido del buffer.
 - e. Liberar el buffer.

4.1.2.- Nodo Receptor (servidor)

La descripción y modo de implementación de un nodo receptor, o servidor, mediante la librería LibCSP se muestra a continuación.

1. Inicializar CSP y configurar y asignar las estructuras básicas por medio de un objeto de configuración que contiene datos como la propia dirección del nodo, nombre del host, el número de buffers que utilizará e incluso el tamaño de estos últimos.
2. Configurar la interfaz de ZMQ. Requiere de la dirección del mismo nodo CSP y la dirección IP en la que se encuentra el proxy de ZMQ que actúa como enrutador o bróker.
3. Cargar la tabla de rutas. Esta será cargada encima de las rutas preexistentes, posiblemente sobrescribiéndolas. Estas rutas tienen el formato <dirección>[/máscara] <interfaz> [vía] [, siguiente entrada de la tabla].
4. Iniciar la tarea del enrutador. El enrutador se encargará de dirigir los paquetes provenientes de las colas de este y comprobar sus tiempos de espera.
5. Crear un socket, ponerlo en modo escucha en todos los puertos o en el puerto que se había establecido en el cliente y establecer el número de conexiones entrantes máximo.



6. Iniciar un bucle infinito que se repita con la periodicidad que se desee.
 - a. Esperar por una conexión entrante y aceptarla cuando eso suceda.
 - b. Leer los paquetes de la conexión en un buffer y guardar su contenido en una variable, así como su longitud.
 - c. Procesar la información.
 - d. Liberar el buffer.

Nuevamente esto no es más que una explicación general que debe ser adaptada a las necesidades de la implementación deseada. Un ejemplo de esto se verá en el apartado 4.4 con el desarrollo de la evaluación PER (Packet Error Rate). Por otro lado, en la Figura 4.2 se muestra una captura de pantalla de la ejecución de un ejemplo de estos nodos transmisor y receptor.

```

libcsp@alen-HP-Laptop-15s-fq2xxx:/libcsp/tests$ ./run_file.sh tx_node.py
Routes:
1/5 ZMQHUB 6
30/5 LOOP
0/0 ZMQHUB
*****
Tx packet, len=6, data=aaaaaaaaaaaa
Transmitted packet nº 0

Tx packet, len=6, data=aaaaaaaaaaaa
Transmitted packet nº 1

Tx packet, len=6, data=aaaaaaaaaaaa
Transmitted packet nº 2

Tx packet, len=6, data=aaaaaaaaaaaa
Transmitted packet nº 3

Tx packet, len=6, data=aaaaaaaaaaaa
Transmitted packet nº 4

Tx packet, len=6, data=aaaaaaaaaaaa
Transmitted packet nº 5

Tx packet, len=6, data=aaaaaaaaaaaa
Transmitted packet nº 6

libcsp@alen-HP-Laptop-15s-fq2xxx:/libcsp/tests$ ./run_file.sh rx_node.py
Hostname: test_service
Model: bindings
Revision: 1.2.3
Routes:
1/5 LOOP
0/0 ZMQHUB
*****
*** Server task started ***

Connection: source=30:44, dest=1:20
rx_node.py:37: DeprecationWarning: PY_SSIZE_T_CLEAN will be required for '#' formats
data = libcsp.packet_get_data(packet)
Received packet nº 0
Rx packet, len=6, data=aaaaaaaaaaaa

187649984473770

Connection: source=30:45, dest=1:20
Received packet nº 1
Rx packet, len=6, data=aaaaaaaaaaaa

187649984473770

Connection: source=30:46, dest=1:20
Received packet nº 2
Rx packet, len=6, data=aaaaaaaaaaaa

```

Figura 4.2.- Captura de pantalla de la terminal donde se ha ejecutado un ejemplo de estos nodos transmisor y receptor.

4.2.- Bloque CSP ZMQ Hub

La herramienta GNU Radio trae consigo diferentes bloques y módulos que permiten la conectividad con la tecnología ZeroMQ. No obstante, cuando se trata de conectividad de CSP por medio de ZeroMQ, hay algún detalle que cambia y hace imposible el uso directo de dichos módulos y bloques propios de GNU Radio.

El problema principal radica en que ZeroMQ inserta, en el primer byte del paquete CSP a enviar, la dirección de enrutamiento. Esta dirección de enrutamiento, para este caso en el que se utiliza ZeroMQ sobre una red CSP, se hace coincidir con la propia dirección de la red CSP. No obstante, el paquete CSP realmente no lleva ese byte al principio del paquete, por lo que es necesario un procesamiento de ese byte previo a una correcta simulación de la transmisión del paquete CSP por el canal.



Para solventar este problema se ha utilizado un bloque OOT desarrollado por la empresa Alén Space. Este bloque aparece en la Figura 4.3 con el nombre “csp_zmqhub_pdu”. Cabe mencionar que dicho bloque fue desarrollado para la versión 3.8 de GNU Radio y, por tanto, ha sido necesario portarlo a la versión 3.10 siguiendo la guía de [37]. No obstante, el trabajo de portar bloques entre versiones suele conllevar únicamente a recrear el bloque desde cero por medio de la herramienta gr-modtool [38] y traspasar la lógica del código de la versión antigua al nuevo bloque, y esto es precisamente lo que se ha hecho para solventar el problema de versiones.

El bloque presenta 4 parámetros de entrada:

- **Publisher Endpoint.** Parámetro que espera la dirección del proxy de ZeroMQ, así como el puerto por el que publicar los paquetes correspondientes.
- **Subscriber Endpoint.** Parámetro que espera la dirección del proxy de ZeroMQ, así como el puerto por el que subscribirse a los paquetes correspondientes provenientes del propio proxy.
- **Subscriber Address.** Dirección por la cual se suscribe al proxy de ZeroMQ. Esta dirección debe coincidir con la de la red CSP que se le ha sido asignada.
- **Packet deduplication.** Modo de comprobación de duplicación de paquetes. Acepta los valores 0, para desactivarlo, o 1 para activarlo y eliminar las copias duplicadas de los paquetes.

Por otro lado, cuenta con dos puertos de tipo Async message, uno de entrada llamado “in” y otro de salida llamado “out”.

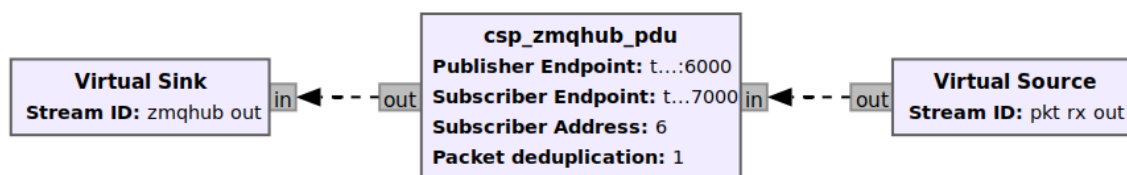


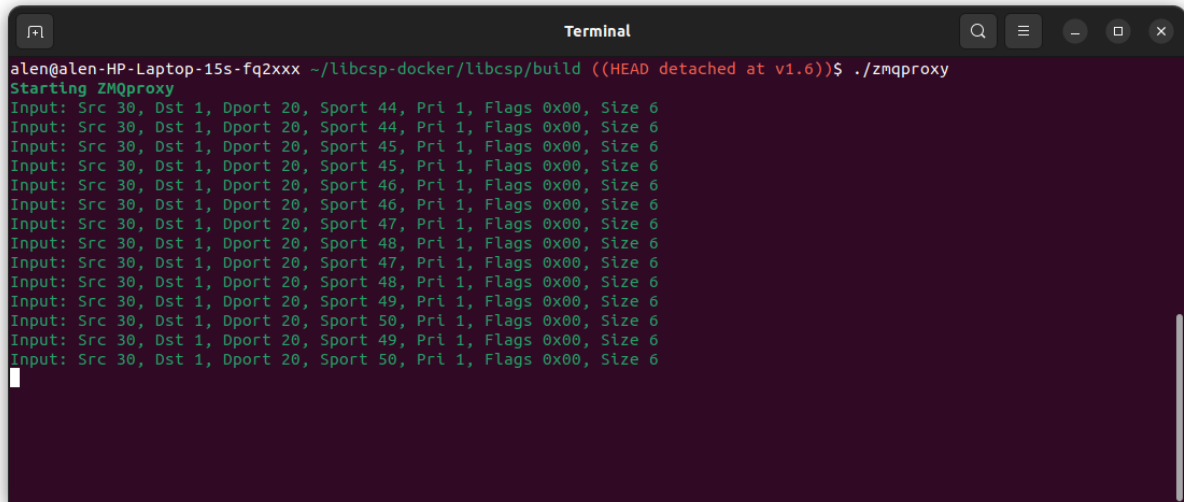
Figura 4.3.- Representación del bloque csp_zmqhub_pdu en GRC.

En cuanto al funcionamiento de este bloque, en primer lugar, cabe mencionar que su funcionamiento sigue la filosofía publicador-subscriptor, la cual ha sido explicada previamente en el Capítulo 3. Por otro lado, se solventa directamente el problema previamente mencionado. La manera en que se solventa esto es que, cuando recibe un paquete por vía de suscripción, lo primero que hace es copiar el propio paquete en otro buffer sin el primer byte y acto seguido volcar este último en el puerto de salida de tipo Async Message “out” para su procesamiento en GNU Radio como si del paquete CSP se tratase



únicamente. Por otro lado, cuando recibe un paquete por el puerto entrante de tipo Async message, previo a su publicación al proxy de ZeroMQ, le inserta un byte correspondiente a la dirección CSP de destino para que el proxy pueda redireccionarlo correctamente de nuevo.

Por último, en la Figura 4.4 se muestra una captura de pantalla de la ejecución del ZMQ proxy utilizado.



```
alen@alen-HP-Laptop-15s-fq2xxx ~/libcsp-docker/libcsp/build ((HEAD detached at v1.6))$ ./zmqproxy
Starting ZMQproxy
Input: Src 30, Dst 1, Dport 20, Sport 44, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 44, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 45, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 45, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 46, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 46, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 47, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 48, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 47, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 48, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 49, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 49, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 50, Pri 1, Flags 0x00, Size 6
Input: Src 30, Dst 1, Dport 20, Sport 50, Pri 1, Flags 0x00, Size 6
```

Figura 4.4.- captura de pantalla de la ejecución del ZMQ proxy utilizado.

4.3.- Implementación de módems

A continuación, se detalla la implementación e integración que se ha llevado a cabo de 2 módems implementados sobre GNU Radio con la finalidad de usarlos como ejemplo de posible subsistema de ejemplo bajo evaluación.

4.3.1.- GMSK

El primer módem implementado es uno basado en una modulación GMSK para comunicación de paquetes compuestos por cabecera y carga útil y diseñado por la empresa Alén Space. Su diagrama de bloques se muestra en las Figuras 4.5, 4.6 y 4.8. En la Figura 4.5 se encuentra la parte de inicialización de variables generales y la simulación de canal junto con la interfaz de ZeroMQ explicada en el apartado anterior. En la Figura 4.6, por otro lado, se muestra la implementación final del modulador, mientras que en la Figura 4.8 se muestra el demodulador que lo acompaña en la recepción. Estas dos últimas serán explicadas más en detalle en los siguientes subapartados.

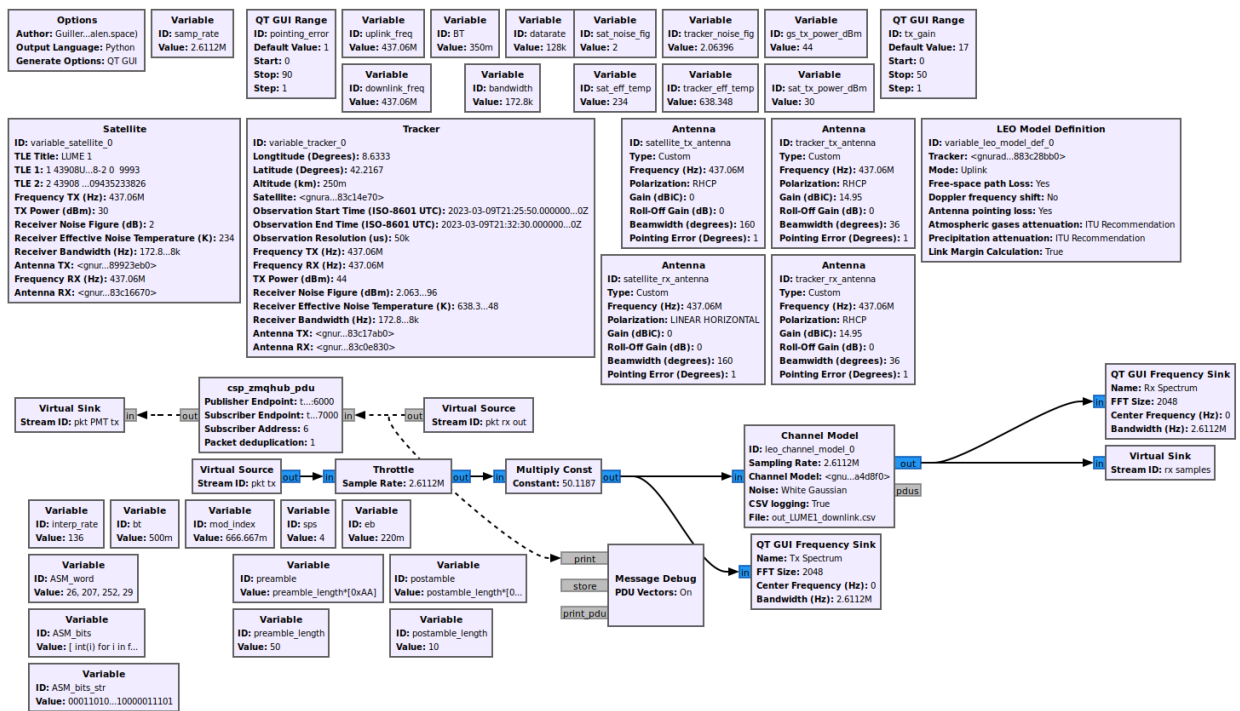


Figura 4.5.- Diagrama de bloques donde se implementa el modem GMSK junto con el módulo OOT gr-leo. Parte de gr-leo.

Como se adelantó, en el esquema de bloques de la Figura 4.5 se inicializan todas las variables y bloques propios del módulo gr-leo. Concretamente, para este caso se han introducido los datos relativos a la misión del CubeSat Lume1 [30]. Por otro lado, respecto a los datos introducidos para la simulación de la estación terrena, o Tracker, corresponden a los datos del GS-Kit [39] desarrollado por la empresa Alén Space.

Tras la correcta inicialización de los bloques de gr-leo, en primer lugar, se implementa el bloque `csp_zmqhub_pdu` encargado de recibir paquetes del proxy de ZeroMQ e inyectarlos en el diagrama de flujo, y viceversa, enviando al proxy los paquetes que han sido demodulados correctamente por el esquema de bloques. Se conecta a su entrada “in”, de tipo Async message, un bloque Virtual Source encargado de suministrar los paquetes que ya han pasado por el canal y han sido correctamente demodulados. Asimismo, se conecta a su salida de tipo Async message un bloque Virtual Sink encargado de pasar los paquetes enviados desde el proxy de ZeroMQ hacia la parte moduladora del esquema.

Por otro lado, la cadena de bloques responsable de la simulación del canal sucede empezando por el bloque Virtual Source, que aporta las muestras de tipo float complejo de 32 bits. Estas muestras pasan en primer lugar por el bloque Throttle, que sirve para el control de la congestión en base a la tasa de muestreo, definida en la variable `samp_rate`. Seguidamente las muestras pasan por un bloque Multiply Const que multiplica el valor de las muestras por el valor definido en la variable de tipo QT GUI Range, `tx_gain`, tras una conversión de decibelios a unidades naturales de la misma en el propio bloque. Gracias a



este último bloque se consigue simular la amplificación de la señal para conseguir la potencia transmitida. A continuación, las muestras amplificadas se introducen en el bloque de gr-leo Channel Model, donde sucede al fin la simulación de canal. Finalmente, las muestras que salen del canal van a parar de nuevo a un bloque Virtual Sink, que se conecta con la etapa demoduladora.

Cabe destacar que, por motivos de privacidad de la empresa Alén Space no se muestran las etapas transmisora y receptora del módem.

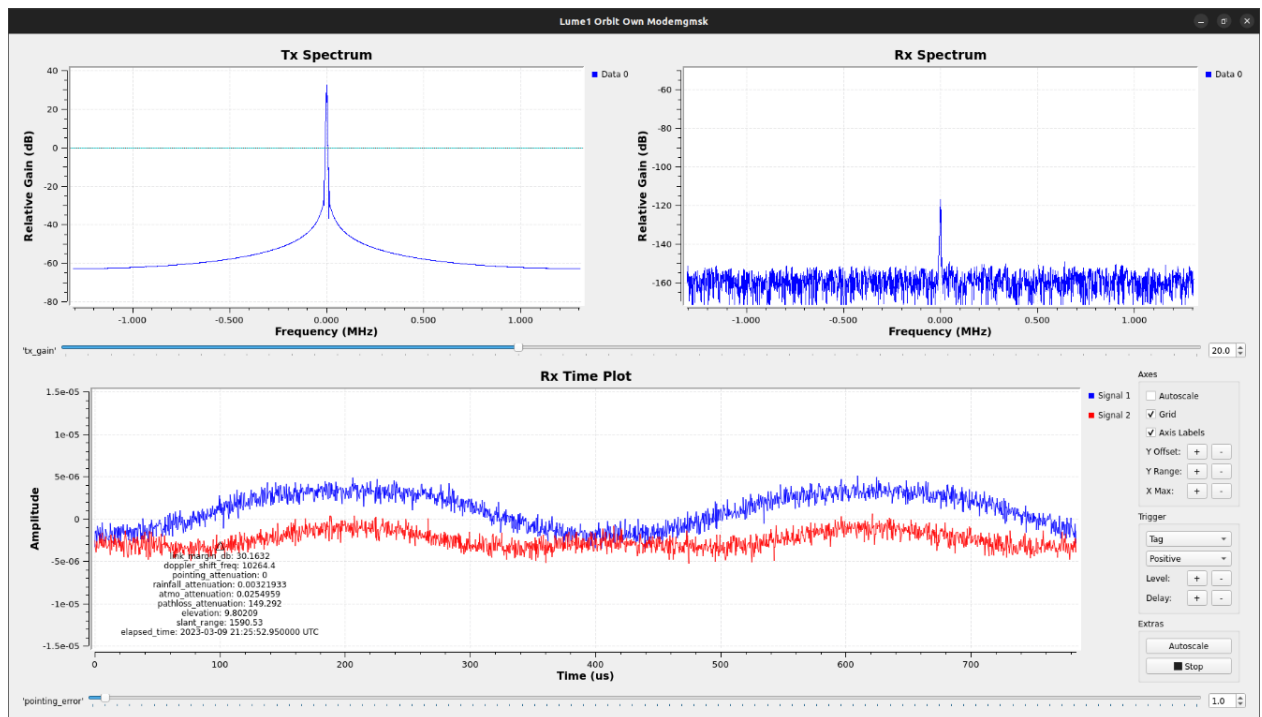


Figura 4.6.- Captura de pantalla del resultado de la ejecución del diagrama de bloques de las Figuras 4.5.

Por último, en las Figuras 4.6 y 4.7 se muestran unas capturas del resultado de la ejecución del diagrama de bloques de las Figuras 4.5. Para esta ejecución se ha utilizado la misma filosofía de transmisión y recepción por medio de los nodos CSP explicados en el apartado 4.1. Concretamente, el contenido de los paquetes enviados es 6 bytes conteniendo el valor hexadecimal 0xAA repetido 6 veces, que en binario representa una secuencia de 1 y 0 alternados, comenzando por 1.

La primera figura muestra la ventana de ejecución gráfica del programa donde aparecen los espectros de frecuencia, tanto de transmisión, previo al canal, como de recepción, posterior al canal. Además, se muestra una gráfica de la señal recibida en el tiempo después del canal, donde aparecen también las etiquetas que incluye la herramienta gr-leo acerca del balance de enlace calculado para cada instante de tiempo respecto a los datos declarados en los objetos del propio gr-leo en el diagrama de bloques. En esta ventana aparece también la variable de rango llamada tx_gain, la cual se puede utilizar



para variar la potencia de transmisión de la simulación de canal durante la ejecución del programa. Por último, abajo del todo aparece, de la misma manera, una variable de rango con el nombre `pointing_error`, responsable de las pérdidas por apuntamiento.

Por otro lado, la segunda muestra la consola del propio GRC donde aparecen, tanto mensajes de depuración del bloque `csp_zmqhub_pdu`, como los datos en hexadecimal de los paquetes recibidos y correctamente demodulados y decodificados, como los mismos datos que se añaden en forma de etiquetas por medio de la herramienta `gr-leo`.

```
Generating: '/home/gnuradio/persistent-folder/libcsp-LEOchannel/LUME1_orbit_own_modemGMSK.py'
Executing: '/usr/bin/python3 -u /home/gnuradio/persistent-folder/libcsp-LEOchannel/LUME1_orbit_own_modemGMSK.py'

Warning: failed to XinitThreads()
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-gnuradio'
correlate_access_code_tag_bb :debug: Access code: 1acffc1d
correlate_access_code_tag_bb :debug: Mask: ffffffff
csp_zmqhub_pdu :info: zmq: publisher connected
csp_zmqhub_pdu :info: zmq: subscriber connected
csp_zmqhub_pdu :debug: ZMQ packet received
correlate_access_code_tag_bb :debug: writing tag at sample 470
***** MESSAGE DEBUG PRINT *****
((pld_len . 80))
*****
csp_zmqhub_pdu :debug: CSP packet received
csp_zmqhub_pdu :debug: Checking duplicated packets
csp_zmqhub_pdu :debug: Packet sent to zmqproxy
**** VERBOSE PDU DEBUG PRINT ****
((link_margin_db . 30.0523) (doppler_shift_freq . 10275.6) (pointing_attenuation . 0) (rainfall_attenuation . 0.00328526) (atmo_attenuation . 0.0255046) (pathloss_attenuation . 149.403) (elevation . 9.49415) (slant_range . 1610.96) (elapsed_time . 2023-03-09 21:25:50.050000 UTC))
pdu length =      10 bytes
pdu vector contents =
0000: 7c 15 3e 00 aa aa aa aa aa
*****
csp_zmqhub_pdu :debug: ZMQ packet received
correlate_access_code_tag_bb :debug: writing tag at sample 1086
***** MESSAGE DEBUG PRINT *****
((pld_len . 80))
*****
**** VERBOSE PDU DEBUG PRINT ****
()
pdu length =      10 bytes
pdu vector contents =
0000: 7c 15 3f 00 aa aa aa aa aa
*****
csp_zmqhub_pdu :debug: CSP packet received
csp_zmqhub_pdu :debug: Checking duplicated packets
csp_zmqhub_pdu :debug: Packet sent to zmqproxy
csp_zmqhub_pdu :debug: ZMQ packet received
correlate_access_code_tag_bb :debug: writing tag at sample 1702
***** MESSAGE DEBUG PRINT *****
((pld_len . 80))
*****
**** VERBOSE PDU DEBUG PRINT ****
()
pdu length =      10 bytes
pdu vector contents =
0000: 7c 15 19 00 aa aa aa aa aa
*****
csp_zmqhub_pdu :debug: CSP packet received
```

Figura 4.7.- Captura de pantalla del resultado en la terminal de GRC de la ejecución del diagrama de bloques de las Figuras 4.5.

4.3.2.- Satlab SRS-3

La implementación de este diagrama de bloques de la Figura 4.8 ha sido realizada utilizando la versión v20230207 del módem para GNU Radio del SRS-3 de Satlab [41].



La integración que se ha realizado junto con la herramienta gr-leo es igual a la cadena de simulación de canal explicada en el subapartado del módem GMSK, solo que se le ha añadido los bloques propios de Satlab. En el caso del transmisor se trata del bloque SRS-3 Transmitter, y en cambio, en la recepción se trata del bloque SRS-3 Receiver. Nótese que se utiliza la entrada csp_in y salida csp_out para el transmisor y receptor respectivamente dado que se está utilizando una red CSP.

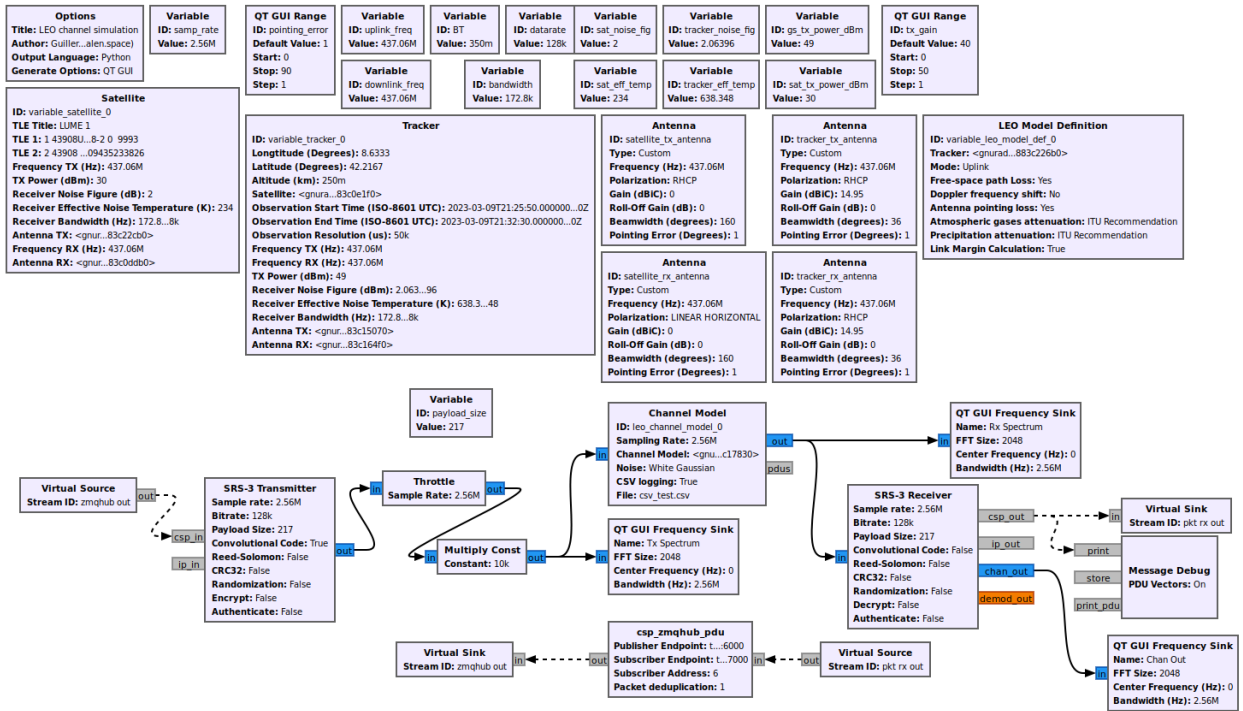


Figura 4.8.- Diagrama de bloques donde se implementa el modem de Satlab SRS-3 junto con el módulo OOT gr-leo.

Estos bloques permiten las siguientes opciones de configuración:

- **Sample rate.** Tasa de muestreo utilizada en el diagrama de bloques.
- **Bitrate.** Bitrate deseado para cada uno de los módems.
- **Payload Size.** Tamaño que se desea para la carga útil. Rellena con ceros si no hay suficiente información.
- **Convolutional Code.** Opción de aplicar FEC por medio de códigos convolucionales.
- **Reed-Solomon.** Opción de aplicar FEC por medio de Reed-Solomon.
- **CRC32.** Opción de aplicar códigos de redundancia cíclica de 32 bits.
- **Randomization.** Opción de utilizar un randomizador para la carga útil.
- **Encrypt:** Opción de encriptar la carga útil en la transmisión y desencriptarla en la recepción.
- **Authenticate.** Opción de utilizar autenticación.



No obstante, estos bloques permiten algunas configuraciones más, aunque para la realización de este proyecto no son de interés y han sido configurados con sus valores por defecto.

Por otro lado, en cuanto al tipo de trama aplicando cada una de las opciones anteriores aparece en la Figura 4.9.

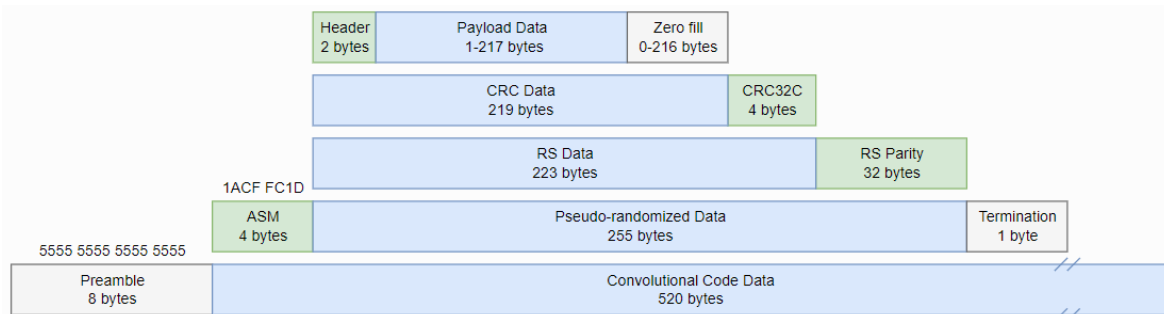


Figura 4.9.- Formato de la trama empleada por el modem de Satlab SRS-3 [42].

Por último, en las Figuras 4.10 y 4.11 se muestran unas capturas del resultado de ejecución del diagrama de bloques de la Figura 4.8. Para esta ejecución se ha utilizado la misma filosofía de transmisión y recepción por medio de los nodos CSP explicados en el apartado 4.1. Concretamente, el contenido de los paquetes enviados es 6 bytes conteniendo el valor hexadecimal 0xAA repetido 6 veces, que en binario representa una secuencia de 1 y 0 alternados, comenzando por 1.

La primera figura muestra, al igual que con el módem GMSK, la ventana de ejecución gráfica del programa. En esta aparecen los espectros de frecuencia, tanto de transmisión, previo al canal, como de recepción, posterior al canal. Además, en la parte inferior de la figura, aparece una gráfica de la señal recibida en el tiempo después del canal donde aparecen también las etiquetas que incluye la herramienta gr-leo acerca del balance de enlace calculado para cada instante de tiempo respecto a los datos declarados en los objetos de gr-leo en el diagrama de bloques. En esta ventana se ve la variable de rango llamada tx_gain, la cual se puede utilizar para variar la potencia de transmisión de la simulación de canal durante la ejecución del programa. Por último, abajo del todo aparece, de la misma manera, una variable de rango con el nombre pointing_error, responsable de las pérdidas por apuntamiento.

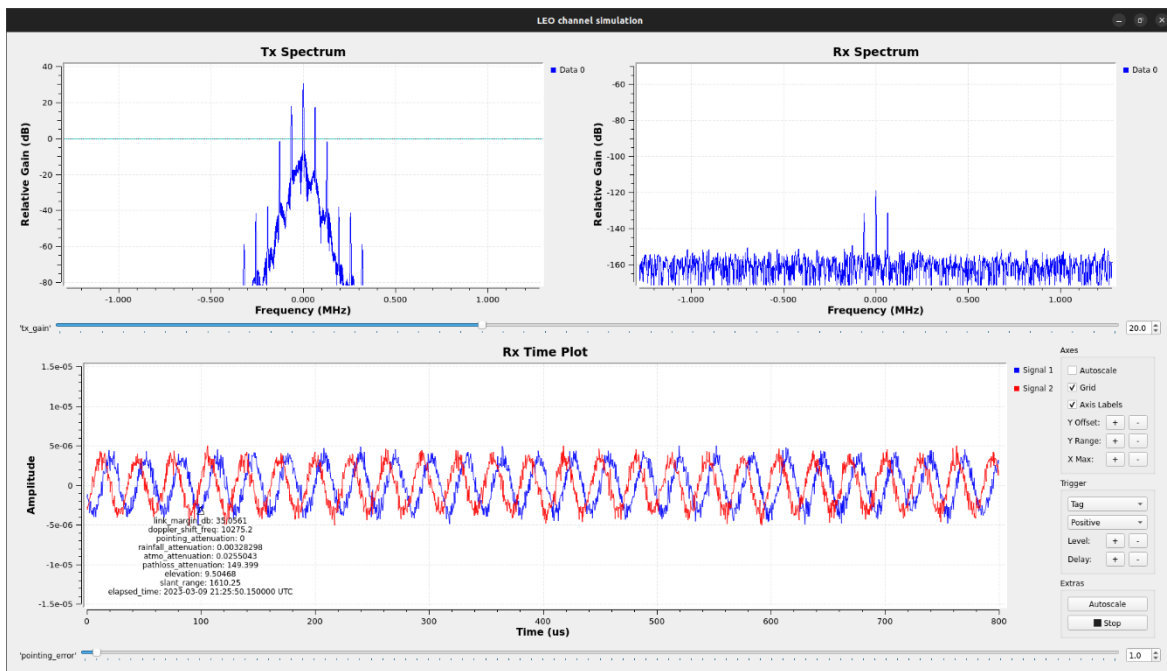


Figura 4.10.- Captura de pantalla del resultado de la ejecución del diagrama de bloques de la Figura 4.8.

Por otro lado, la segunda figura es una captura de la consola del propio GRC donde aparecen, tanto mensajes de depuración del bloque `csp_zmqhub_pdu`, como los datos en hexadecimal de los paquetes recibidos y correctamente demodulados y decodificados.



```

Generating: '/home/gnuradio/persistent-folder/libcsp-LEOchannel/leo_channel_LinkBudget_satlab.py'

Executing: /usr/bin/python3 -u /home/gnuradio/persistent-folder/libcsp-LEOchannel/leo_channel_LinkBudget_satlab.py

Warning: failed to XInitThreads()
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-gnuradio'
csp_zmqhub_pdu :info: zmq: publisher connected
csp_zmqhub_pdu :info: zmq: subscriber connected
csp_zmqhub_pdu :debug: ZMQ packet received
**** VERBOSE PDU DEBUG PRINT ****
((rs_errs . 0) (cc_errs . 0))
pdu length =      10 bytes
pdu vector contents =
csp_zmqhub_pdu :debug: CSP packet received
csp_zmqhub_pdu :debug: Checking duplicated packets
0000: 7c 15 1b 00 aa aa aa aa aa aa
*****
csp_zmqhub_pdu :debug: Packet sent to zmqproxy
csp_zmqhub_pdu :debug: ZMQ packet received
**** VERBOSE PDU DEBUG PRINT ****
((rs_errs . 0) (cc_errs . 0))
pdu length =      10 bytes
pdu vector contents =
0000: 7c 15 1c 00 aa aa aa csp_zmqhub_pdu :debug: CSP packet received
csp_zmqhub_pdu :debug: Checking duplicated packets
aa aa aa
*****
csp_zmqhub_pdu :debug: Packet sent to zmqproxy
csp_zmqhub_pdu :debug: ZMQ packet received
**** VERBOSE PDU DEBUG PRINT ****
((rs_errs . 0) (cc_errs . 0))
csp_zmqhub_pdu :debug: CSP packet received
pdu length =      10 bytes
pdu vector contents =
csp_zmqhub_pdu :debug: Checking duplicated packets
0000: 7c 15 1d 00 aa aa aa aa aa aa
*****
csp_zmqhub_pdu :debug: Packet sent to zmqproxy
csp_zmqhub_pdu :debug: ZMQ packet received
csp_zmqhub_pdu :debug: ZMQ packet received
csp_zmqhub_pdu :debug: CSP packet received
csp_zmqhub_pdu :debug: Checking duplicated packets
**** VERBOSE PDU DEBUG PRINT ****
((rs_errs . 0) (cc_errs . 0))
pdu length =      10 bytes
pdu vector contents =
0000: 7c 15 1e 00 aa aa aa aa aa aa
csp_zmqhub_pdu :debug: Packet sent to zmqproxy
*****

```

Figura 4.11.- Captura de pantalla del resultado en la terminal de GRC de la ejecución del diagrama de bloques de la Figura 4.8.

4.4.- PER

Una forma de evaluar la eficacia de un sistema de comunicaciones es conocer la probabilidad de error de paquetes en un enlace. Normalmente esta evaluación se realiza mandando una gran cantidad de paquetes para unas características de enlace determinadas y se comprueba cuantos paquetes han sido recibidos de forma errónea o no han llegado a su destino siquiera y se compara con el total de paquetes enviados. No obstante, para este caso se varía ligeramente esta prueba ya que se pretende realizar dicha medida para un enlace simulado concreto durante una ventana de AOS preestablecida. Así pues, se necesitará conocer tanto la cantidad de paquetes máximos que se pueden transmitir durante el enlace, así como los parámetros necesarios a introducir en la herramienta GNU Radio



para una correcta simulación. Entre estos parámetros, la potencia de transmisión utilizada junto con las ganancias de las antenas, es decir, la PIRE y la ganancia de la antena receptora son esenciales para una simulación certera ya que las pérdidas del canal las introducirá la herramienta gr-leo.

4.4.1.- Algoritmo PER

A continuación, se explica el algoritmo empleado para realizar la prueba PER desarrollada como ejemplo para evaluar la eficacia de subsistemas como pueden ser los módems desarrollados explicados en el apartado anterior.

Esta prueba ha sido implementada sobre los nodos CSP de transmisión y recepción explicados en el apartado 4.1. Los algoritmos PER seguidos para el nodo de transmisión y recepción se muestran en las Figuras 4.12 y 4.13 respectivamente.

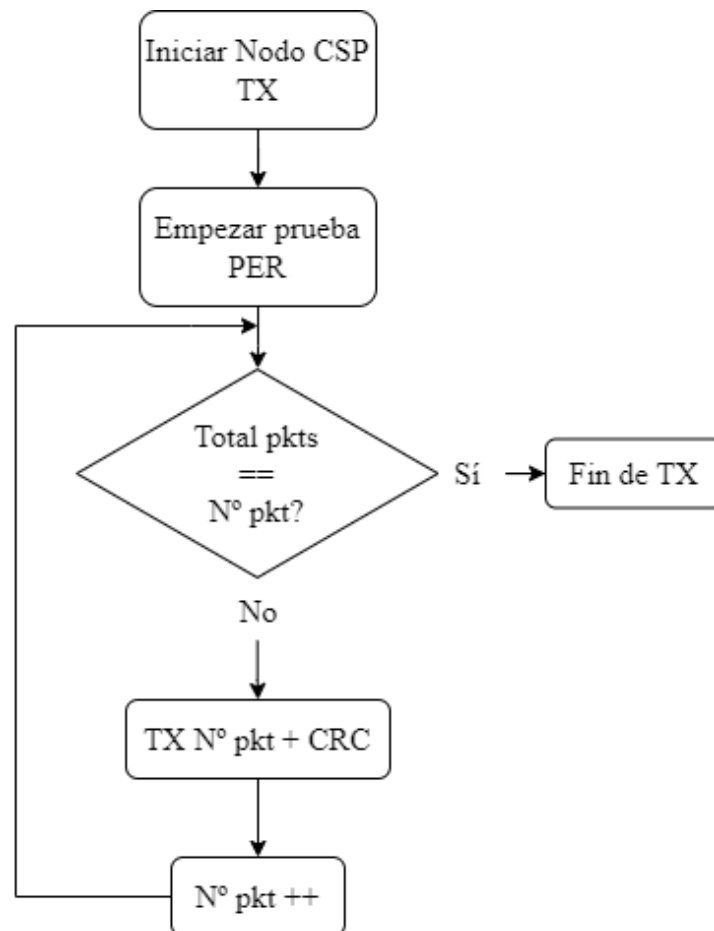


Figura 4.12.- Diagrama de flujo del algoritmo empleado en la transmisión de la prueba PER.

Analizando el algoritmo seguido para el nodo de transmisión, en primer lugar, se inicializa el nodo CSP siguiendo los pasos previos descritos en el apartado 4.1, donde se



inicializan el propio protocolo CSP, la interfaz de ZeroMQ, la tabla de rutas y la propia tarea de enrutamiento. Además, se ha de especificar en un comienzo el número de paquetes a transmitir para la prueba y acto seguido comienza la prueba PER. Seguidamente, se entra en el bucle de transmisión, del cual no se saldrá hasta haber acabado de transmitir todos los paquetes especificados previamente. Para la transmisión de cada paquete, su contenido consistirá en el número de paquete transmitido junto con un código CRC (Cyclic Redundancy Check) de 32 bits para analizar la correcta transmisión en el nodo de recepción. El último paso antes de volver al inicio del bucle será incrementar el número de paquete transmitido. Por último, cuando se alcance el número de paquetes a transmitir, finalizará el programa.

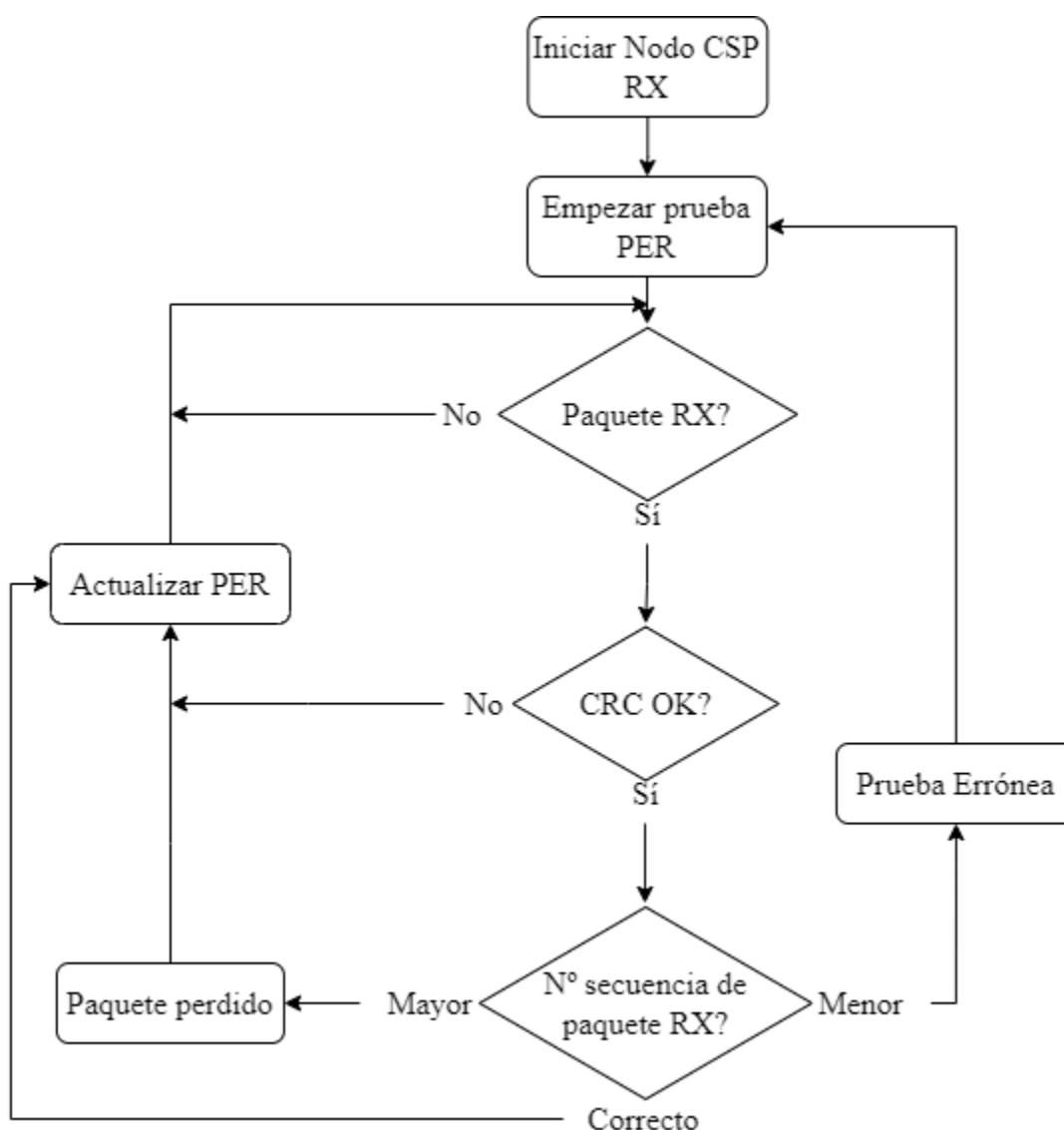


Figura 4.13.- Diagrama de flujo del algoritmo empleado en la recepción de la prueba PER.



Cabe mencionar que se ha de dejar un tiempo entre transmisión y transmisión para el correcto funcionamiento del programa. Además, nótese que el código de Python respectivo se encuentra en el Anexo C.

```

Correct packet!

Connection: source=30:42, dest=1:20
Expected packet nº 57
Rx packet nº 53, len=7
Data=000039a044511a
Correct packet!
^C

*****
*****[PER Results]*****
*****

- Total sent packets: 100
- Total received packets: 53
- Total missed packets: 47
- Total wrong packets: 7
- PER: 54.0 %

... Exiting now ...

Exception ignored in: <module 'threading' from '/usr/lib/python3.8/threading.py'>
Traceback (most recent call last):
  File "/usr/lib/python3.8/threading.py", line 1388, in _shutdown
    lock.acquire()
  File "rx_node_PER.py", line 115, in signal_handler
    PER_results()
  File "rx_node_PER.py", line 139, in PER_results
    exit(0)
  File "/usr/lib/python3.8/sitebuiltins.py", line 26, in __call__
    raise SystemExit(code)
SystemExit: 0
libcsp@alen-HP-Laptop-15s-fq2xxx:/libcsp/PER$

```

Figura 4.14.- Captura de pantalla del resultado final de la ejecución de una prueba PER.

Por otro lado, el primer paso del algoritmo empleado para el nodo receptor implicado en la prueba PER, es inicializar el nodo CSP, nuevamente, siguiendo los pasos descritos en el apartado 4.1. Seguidamente comienza la prueba PER y el nodo comienza a recibir paquetes. Cada vez que recibe un paquete, la primera comprobación que realiza es sobre los 4 bytes del CRC. Si este código no está correcto, actualizara los datos PER incrementando en una unidad los paquetes erróneos. En cambio, si es correcto comprueba a continuación el número de paquete recibido en el contenido del paquete. Este número será comparado con el número esperado resultando en tres opciones: que sea mayor, que sea correcto o que sea menor. En caso de que el número de paquete recibido sea mayor que el esperado se asumirá que se ha perdido el número de paquetes resultante a la diferencia entre el número de paquete recibido y el esperado. Por otro lado, si el número de paquete recibido coincide con el esperado, se actualiza el número del siguiente de paquete esperado. Como última opción, si el número de paquete recibido es menor que el esperado, se considerará una prueba errónea que hará que forzará a reiniciar la prueba, ya que se presupone que ha habido algún problema en la ejecución del programa o en la



coordinación del inicio de la prueba. Por último, se imprimirán por consola los resultados finales de la prueba PER al finalizar el programa de manera manual, tras haber finalizado, por su parte, el programa de transmisión.

Por último, en la Figura 4.14 se muestra un ejemplo de los resultados de una prueba PER impresos en la consola tras su finalización.

4.4.2.- Ventana de AOS y paquetes totales

A continuación, se detalla el método de cálculo necesario para conocer el número de paquetes que se pueden transmitir. Para ello se utilizará un ejemplo partiendo del uso del modem desarrollado de modulación GMSK del apartado 4.3.1. Además, los datos del enlace utilizado para este ejemplo serán los mismos que aparecen en la Figura 4.8, es decir, los del satélite Lume 1.

En primer lugar, se calcula la ventana de tiempo en segundos durante la cual se podrá establecer conexión con el CubeSat. Con ayuda del GPredict, se puede obtener el inicio y el final de la ventana que aparecen en la Figura 3.19. De esta forma, el inicio de la ventana corresponderá con la fecha y hora del 9 de abril del 2023 a las 21 horas 25 minutos y 50 segundos, y el final de la ventana se establece a las 21 horas 32 minutos y 30 segundos del mismo día. Restando la fecha final de la inicial se obtiene un total de 400 segundos.

Por otro lado, el modem GMSK ha sido diseñado para una tasa de bits de 4800 bps, lo que durante una ventana de 400 segundos equivale a un total de 1,920,000 bits transmitidos durante el enlace.

Además, se utiliza para este ejemplo como contenido de sus paquetes, un preámbulo y postámbulo de 50 y 10 bytes respectivamente, y como tamaño de carga útil un total de 304 bytes. Este último se debe a que, el tamaño máximo de los búferes de la red CSP es de 300, a lo que se le suman los 4 bytes de cabecera. De esta forma, junto con la palabra ASM de sincronización de 4 bytes y la longitud del paquete de 3 bytes de codificación Golay, se suma un total de 371 bytes por paquete, lo que equivale a un total de 2,968 bits por paquete transmitido.

Finalmente, dividiendo el total de bits transmitibles durante la ventana entre el número de bits por paquetes, se obtiene un total de 646 paquetes que se pueden transmitir durante la ventana de AOS.

4.4.3.- Potencia de señal en GNU Radio

Para conocer la potencia que se transmitirá en la simulación de canal se muestra a continuación, por medio de un ejemplo, como conocer la potencia de las muestras con las que trabaja GNU Radio.

El ejemplo es el del diagrama de bloques que se muestra en la Figura 4.15. En este, se generan muestras de tipo float complejo de 32 bits a partir del bloque Signal Source que genera un tono de frecuencia de 100 KHz y una amplitud de 1 V, y, seguidamente se



conecta un bloque Throttle. A continuación, el flujo de muestras pasa por el bloque Multiply Const, que multiplicara las muestras por el valor $10^{*(tx_gain/20)}$ como conversión de dB a unidades naturales, donde tx_gain es una variable de rango que representa la ganancia que se les añade a las muestras de la señal en dB. En este punto el flujo de muestras se divide en dos ramas, una que va a parar a un bloque QT GUI Frequency Sink, encargado de representar la señal en el dominio de la frecuencia, y otra que se encargará de calcular y mostrar el valor de la potencia de la señal. Para este último paso se calcula primero el módulo al cuadrado de las muestras y se hace la media de las últimas 50,000 muestras para seguidamente convertir a dBW al multiplicar por 10 y hacer el logaritmo en base 10 del resultado. Esto se lleva a cabo por medio de los bloques Complex to Mag², Moving Average y Log10. Por último, se muestra el resultado obtenido por medio del bloque QT GUI Number Sink.

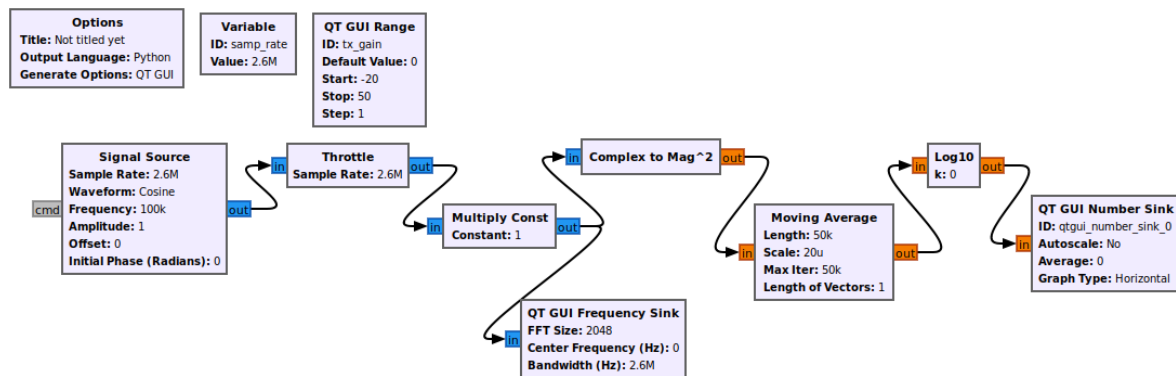


Figura 4.15.- Diagrama de bloques utilizado para la comprensión de la potencia de señal en GNU Radio.

En la Figura 4.16 se muestra la ventana gráfica de la ejecución del diagrama de bloques recién explicado. Analizando esta figura, en primer lugar, se observa que a la variable tx_gain se le asigna el valor de 10, representando una ganancia de 10 dB a la señal. Por otro lado, aparecen tanto el resultado de la potencia en dBW, así como la representación de la señal en el dominio de la frecuencia. En este último se diferencia un pico a 100 KHz propia del tono generado. Ha de diferenciarse que, mientras que el valor de pico de la delta de esta representación se encuentra cercano a los 0 dB, este no es el valor de la potencia ya que este último se corresponde a 10 dBW, lo que cuadra tras sumar 10 dB de ganancia a una señal de 1 W de potencia.

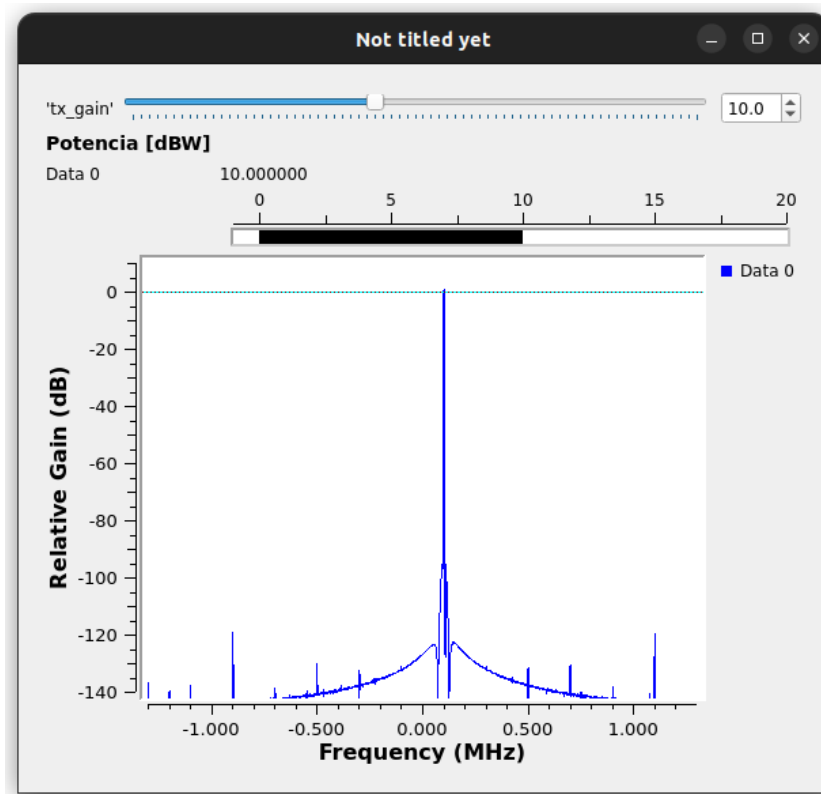


Figura 4.16.- Captura del resultado de la ejecución del diagrama de bloques de la Figura 4.18.



5.- PRUEBAS REALIZADAS

En este capítulo se detallan las pruebas que han sido realizadas para evaluar la eficacia de la herramienta desarrollada en este proyecto. Estas pruebas se diferencian en dos principalmente: balance de enlace y evaluación PER.

5.1.- Balance de enlace

Para las pruebas realizadas respecto al balance de enlace, se pretende comparar los resultados de un balance de enlace realizado de manera sencilla por medio del script de Python del Anexo A frente a los que ofrece la herramienta gr-leo. De esta forma, se presenta en las Tablas 5.1-5.4 los valores utilizados para ambos. Por otro lado, en la Figura 5.1 se muestra el diagrama de bloques empleado para obtener los resultados por parte del módulo gr-leo.

Tabla 5.1.- Datos del enlace ascendente utilizados para el cálculo del balance de enlace.

ENLACE ASCENDENTE	VALOR	UNIDAD
FRECUENCIA	437.06	MHz
PÉRDIDAS POLARIZACIÓN	0	dB
PÉRDIDAS POR APUNTAMIENTO	0	dB
MODULACIÓN	GMSK (BT = 0.35)	N/A
FEC	Cód. Convolutivos (R=1/2, K=7) + Reed-Solomon (255,223)	N/A
TASA DE DATOS	4800	bps
ANCHO DE BANDA	6480	Hz
MARGEN DEL SISTEMA	3	dB

Tabla 5.2.- Datos del enlace descendente utilizados para el cálculo del balance de enlace.

ENLACE DESCENDENTE	VALOR	UNIDAD
FRECUENCIA	437.06	MHz
PÉRDIDAS POLARIZACIÓN	0	dB
PÉRDIDAS POR APUNTAMIENTO	0.5	dB



MODULACIÓN	GMSK (BT = 0.35)	N/A
FEC	Cód. Convolucionales (R=1/2, K=7) + Reed-Solomon (255,223)	N/A
TASA DE DATOS	4800	bps
ANCHO DE BANDA	6480	Hz
MARGEN DEL SISTEMA	3	dB

Tabla 5.3.- Datos de la estación terrena utilizada para el cálculo del balance de enlace.

ESTACIÓN TERRENA	VALOR	UNIDAD
POTENCIA DISPONIBLE DE TX	19	dBW
PÉRDIDAS DE LÍNEA DE TRANSMISIÓN	5	dB
OTRAS PÉRDIDAS	1	dB
GANANCIA DE ANTENA	14.95	dBi
TEMPERATURA EFECTIVA DE RUIDO DEL RECEPTOR	638.348	K

Tabla 5.4.- Datos del CubeSat Lume 1 utilizados para el cálculo del balance de enlace.

SATÉLITE	VALOR	UNIDAD
POTENCIA DISPONIBLE DE TX	0	dBW
PÉRDIDAS DE LÍNEA DE TRANSMISIÓN	0.2	dB
OTRAS PÉRDIDAS	0.3	dB
GANANCIA DE ANTENA	0	dBi
TEMPERATURA EFECTIVA DE RUIDO DEL RECEPTOR	234	K

En este caso, siendo necesaria únicamente la obtención del fichero CSV del balance de enlace proporcionado por gr-leo, el diagrama de bloques diseñado no incluye un modem de comunicaciones ni ninguna fuente de datos externa. En cambio, partiendo de un bloque Random Source y pasando por el bloque propio de GNU Radio GMSK Mod, se generan simplemente unas muestras aleatorias para poder ejecutar la simulación, aunque en este caso no influyan para nada en el propio balance de enlace. Nótese igualmente que se ha completado debidamente con los valores de las tablas los bloques necesarios.



Por último, la ventana de observación o AOS que se ha utilizado para estas pruebas es la del enlace con el CubeSat Lume 1 desde el 09-03-2023 a las 21:25:50 hasta el 09-03-2023 a las 21:32:30 obtenido por medio de la herramienta GPredict.

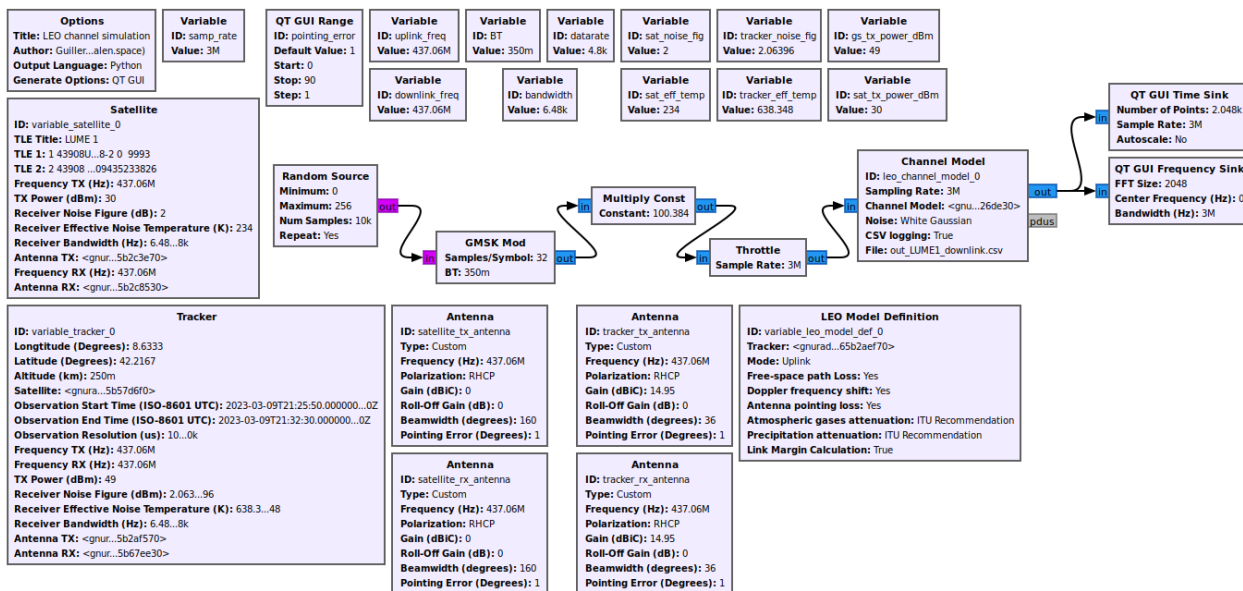


Figura 5.1.- Diagrama de bloques empleado para obtener el balance de enlace del CubeSat Lume 1 con la estación terrena.

5.1.1.- Resultados

En la Tabla 5.5 se muestra el resultado del balance de enlace calculado por medio del script de Python del Anexo A.

Tabla 5.5.- Resultados del balance de enlace entre una estación terrena y el CubeSat Lume 1.

BALANCE DE ENLACE	ASCENDENTE	DESCENDENTE	UNIDAD
FRECUENCIA	437.06	437.06	MHz
ELEVACIÓN	10	10	°
DISTANCIA MÁXIMA	1604.55	1604.55	km
PÉRDIDAS DE ESPACIO LIBRE	149.37	149.37	dB
PÉRDIDAS POR PRECIPITACIÓN	0	0	dB
PÉRDIDAS POR GASES ATMOSFÉRICOS	0	0	dB



PÉRDIDAS POR POLARIZACIÓN	0	0	dB
PÉRDIDAS POR APUNTAMIENTO	0	0.5	dB
OTRAS PÉRDIDAS	0	0	dB
PÉRDIDAS TOTALES DE LA TRAYECTORIA	149.37	149.87	dB
PIRE	27.95	-0.5	dBW
NIVEL RECIBIDO ISOTRÓPICO	-121.42	-150.37	dBW
FIGURA DE MÉRITO G/T	-23.69	-13.1	dB
POTENCIA DE SEÑAL RECIBIDA	-121.42	-135.42	dBW
POTENCIA DE RUIDO RECIBIDA	-166.79	-162.43	dBW
RELACION SEÑAL-RUIDO GAUSSIANO (SN0)	83.49	65.13	dBHz
RELACIÓN SEÑAL-RUIDO EN EL RECEPTOR (SNR)	42.38	24.02	dB
RELACION SEÑAL-RUIDO POR BIT (EB/N0)	43.68	25.32	dB

En las Figuras 5.2-5.9 se muestran los resultados del enlace ascendente procesados en función del tiempo de observación en segundos, obtenidos mediante el archivo CSV generado por medio de la herramienta gr-leo.

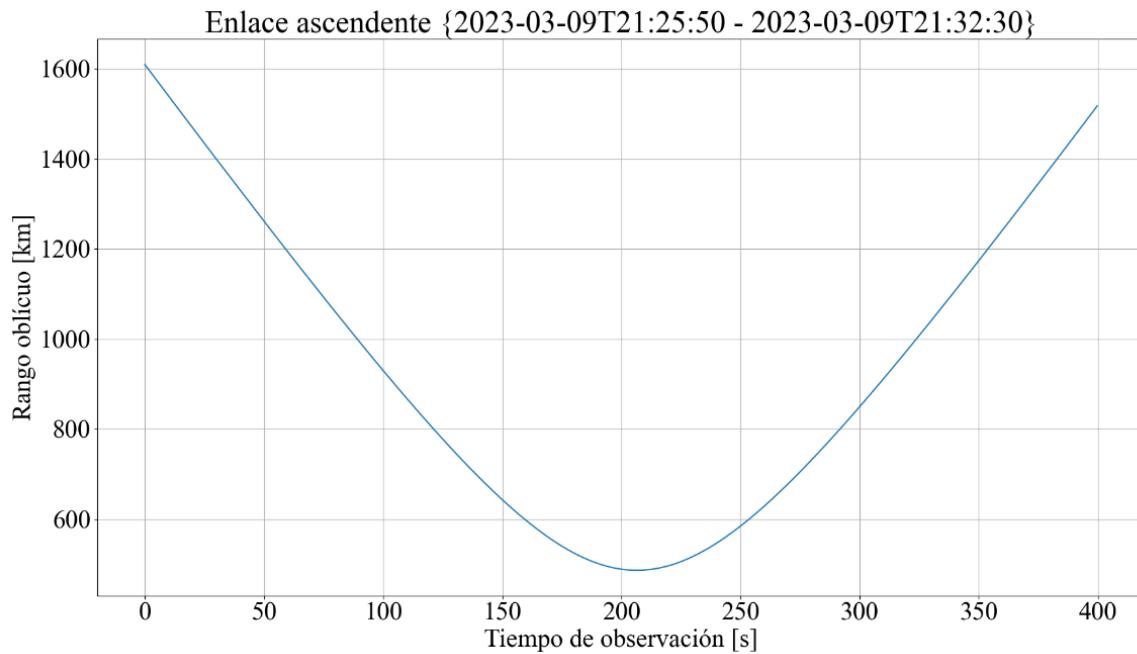


Figura 5.2.- Resultado del rango oblicuo del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

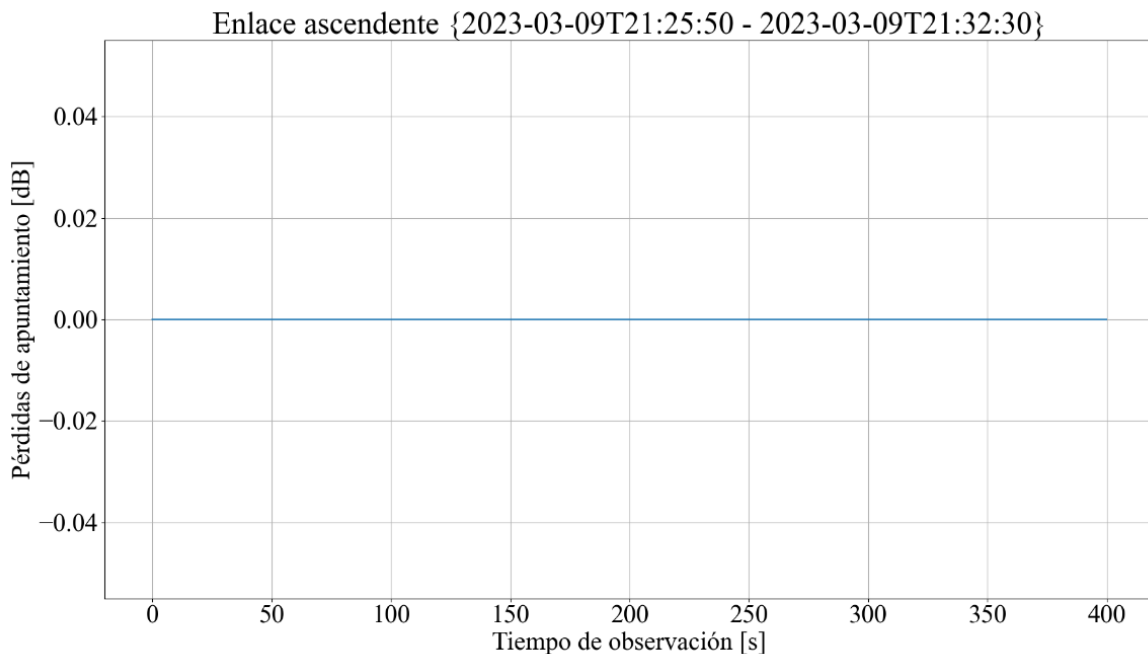


Figura 5.3.- Resultado de las pérdidas de apuntamiento del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

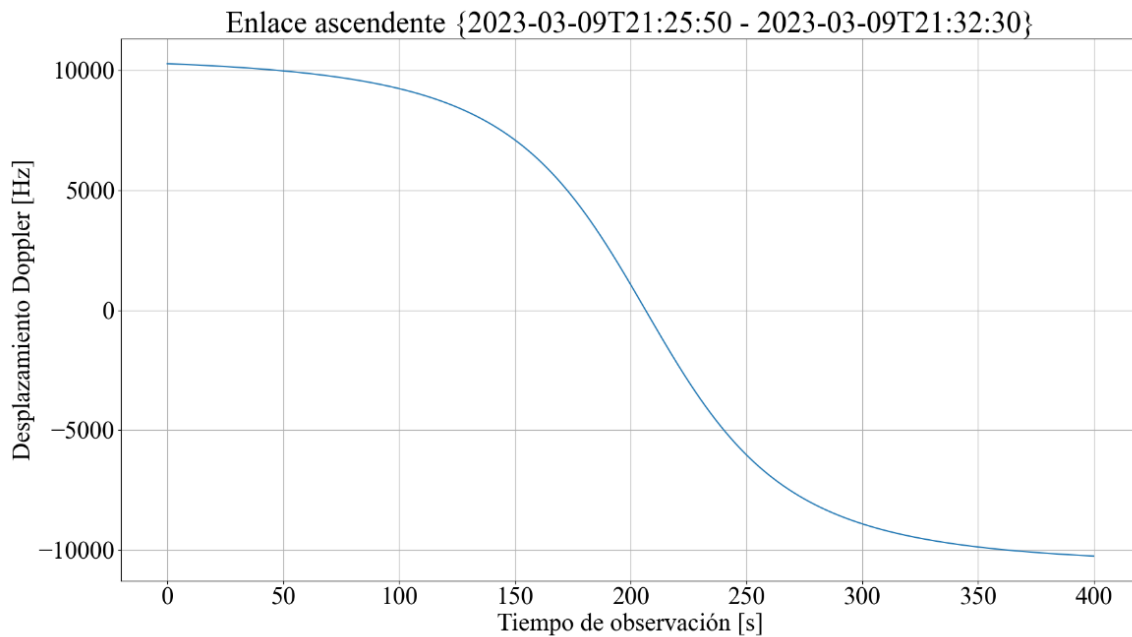


Figura 5.4.- Resultado del desplazamiento Doppler del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

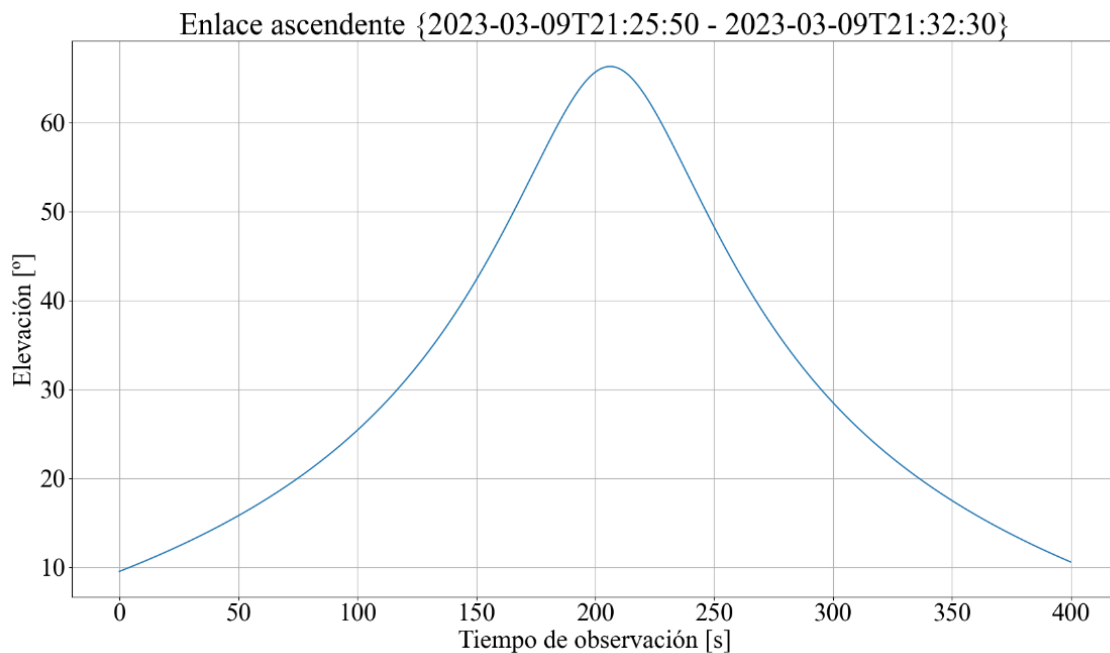


Figura 5.5.- Resultado de la elevación del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

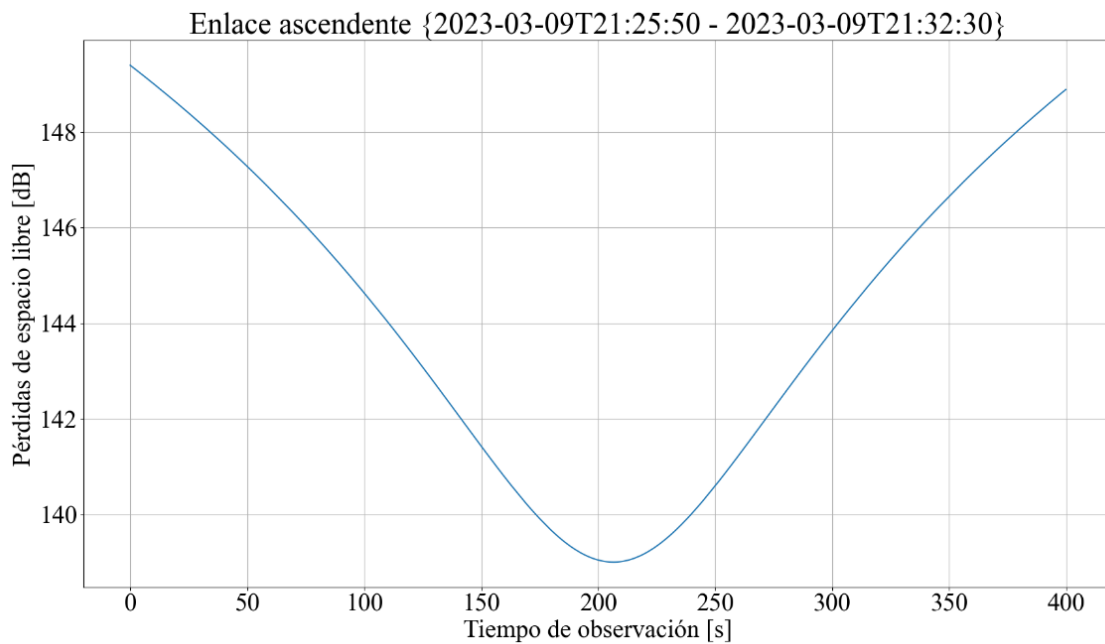


Figura 5.6.- Resultado de las pérdidas de espacio libre del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

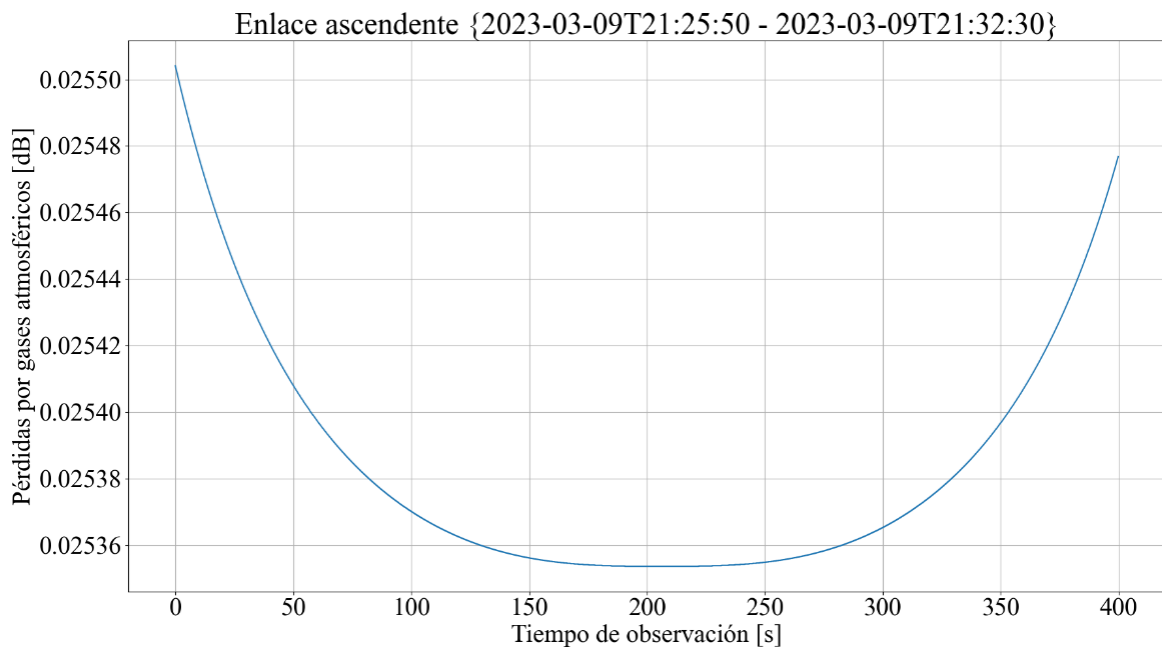


Figura 5.7.- Resultado de las pérdidas por gases atmosféricos del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

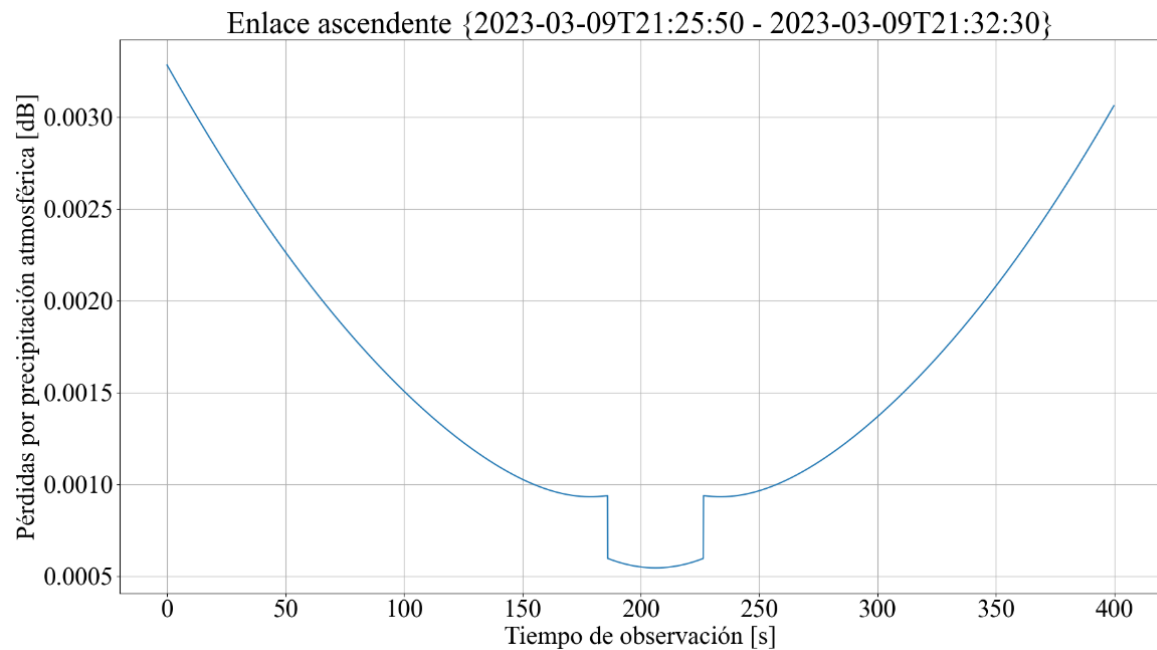


Figura 5.8.- Resultado de las pérdidas por precipitación atmosférica del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

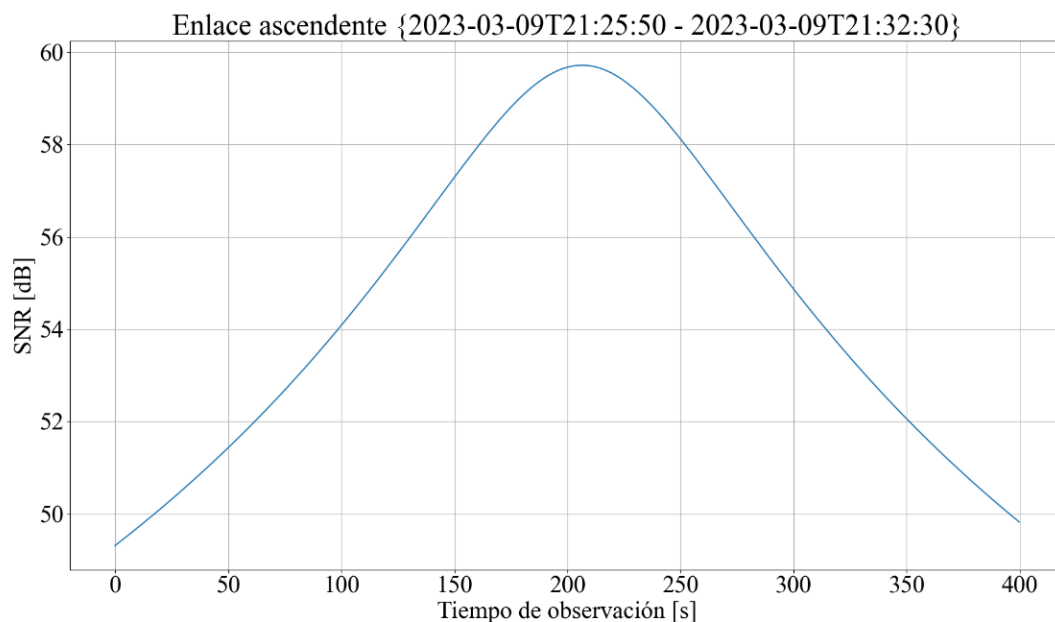


Figura 5.9.- Resultado de la SNR del enlace ascendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.



En las Figuras 5.10-5.17 se muestran los resultados del enlace ascendente procesados en función del tiempo de observación en segundos, obtenidos mediante el archivo CSV generado por medio de la herramienta gr-leo.

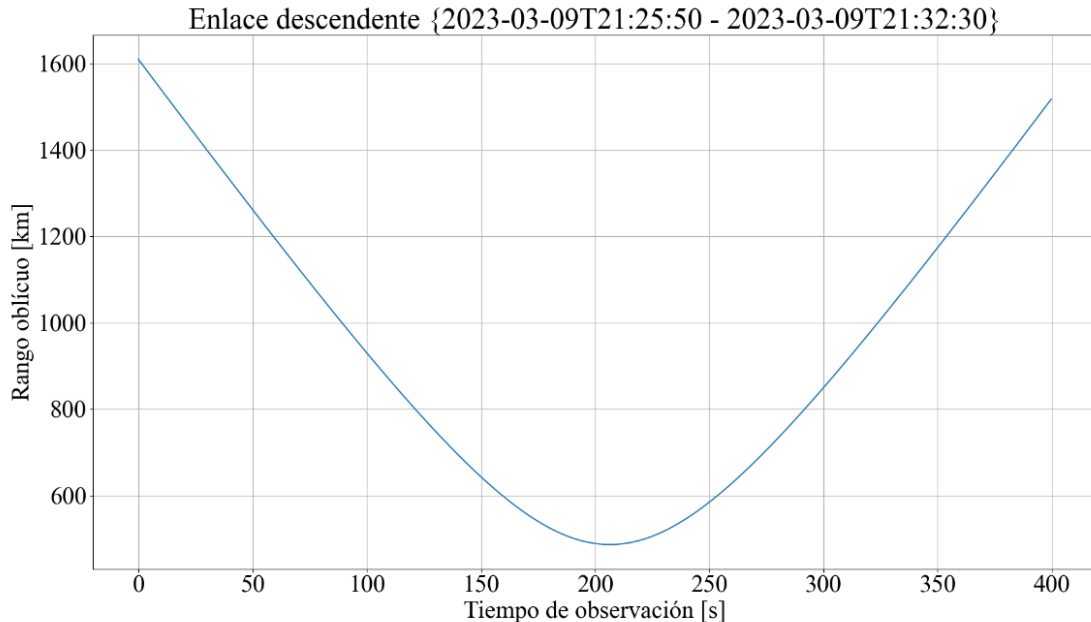


Figura 5.10.- Resultado del rango oblicuo del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

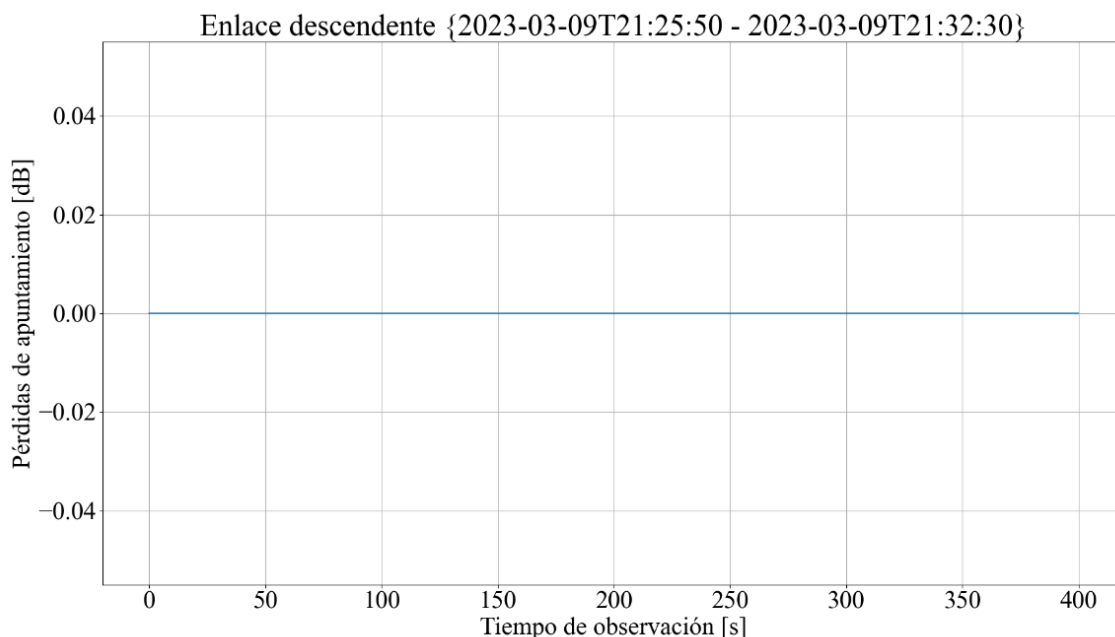


Figura 5.11.- Resultado de las pérdidas de apuntamiento del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

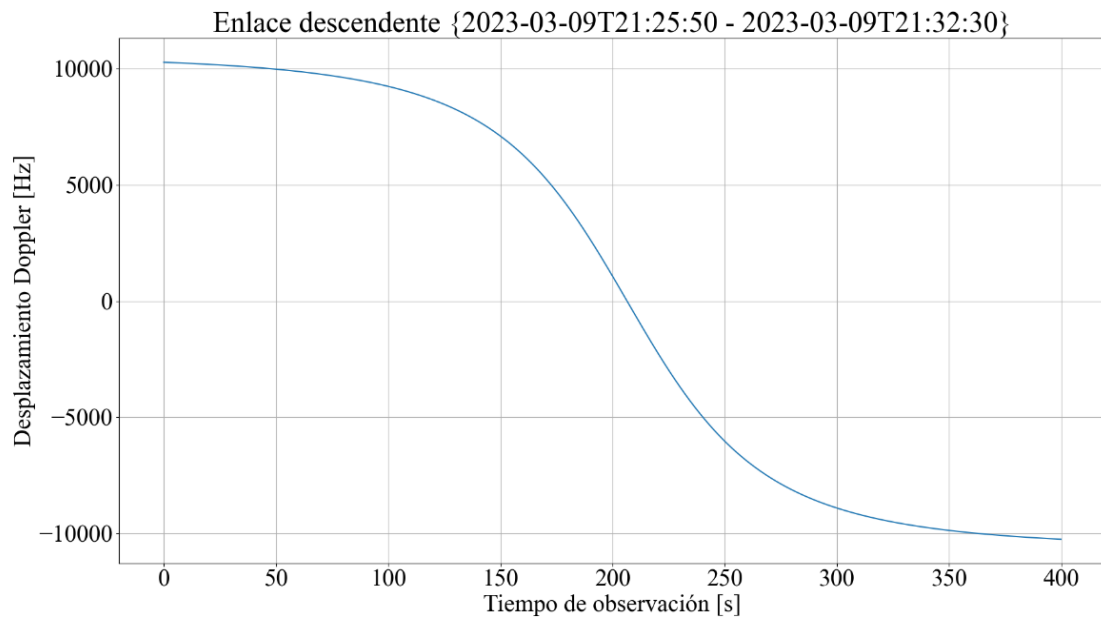


Figura 5.12.- Resultado del desplazamiento Doppler del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

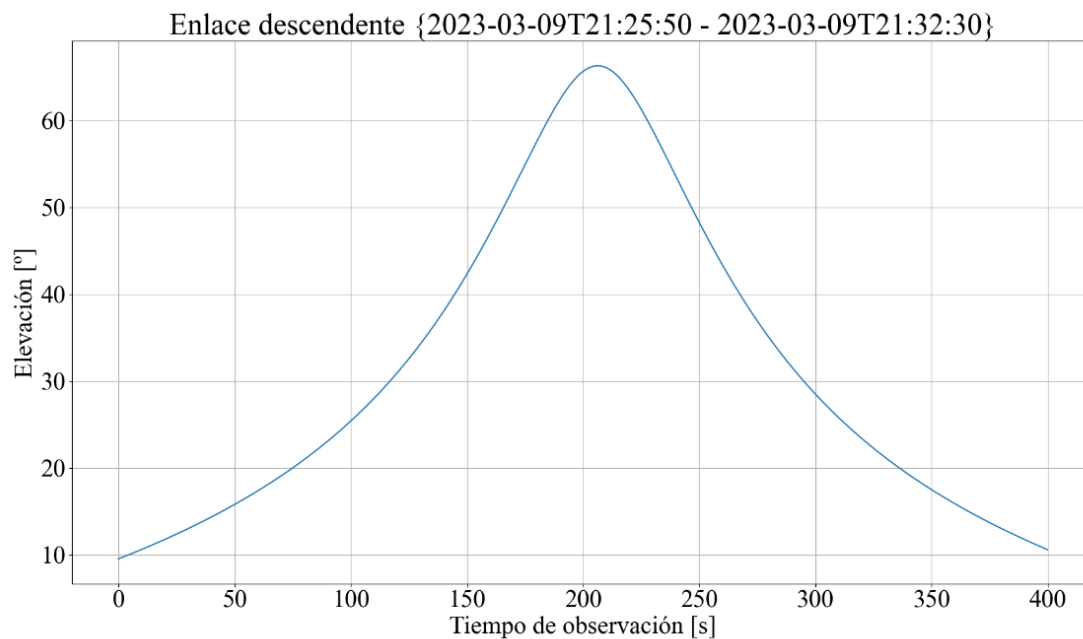


Figura 5.13.- Resultado de la elevación del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

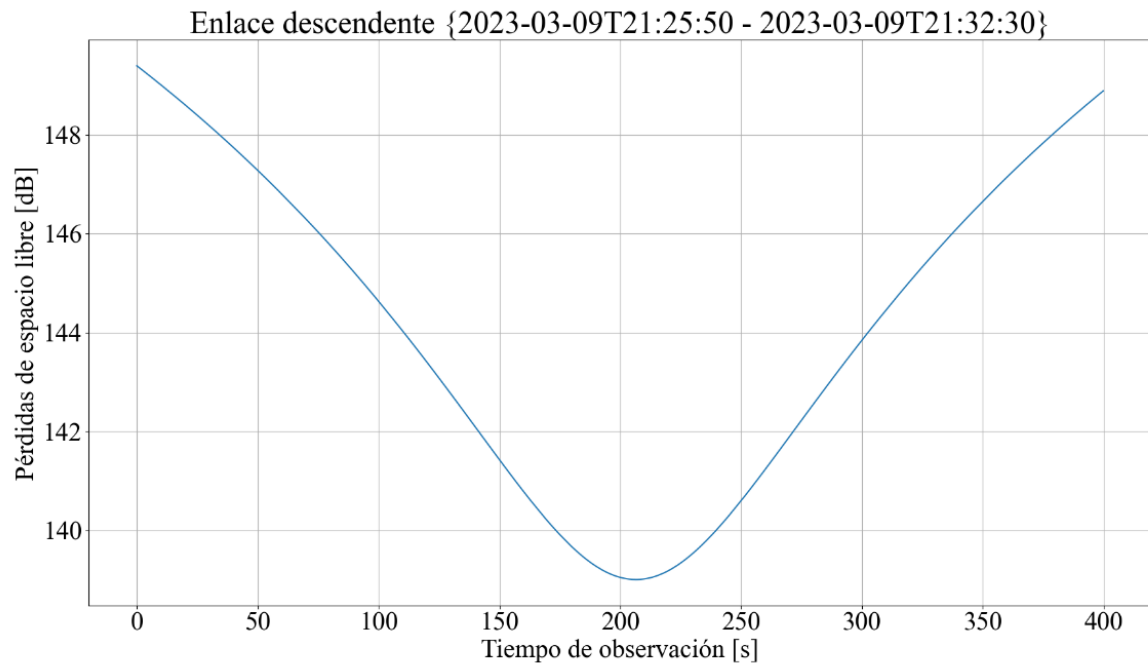


Figura 5.14.- Resultado de las pérdidas de espacio libre del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

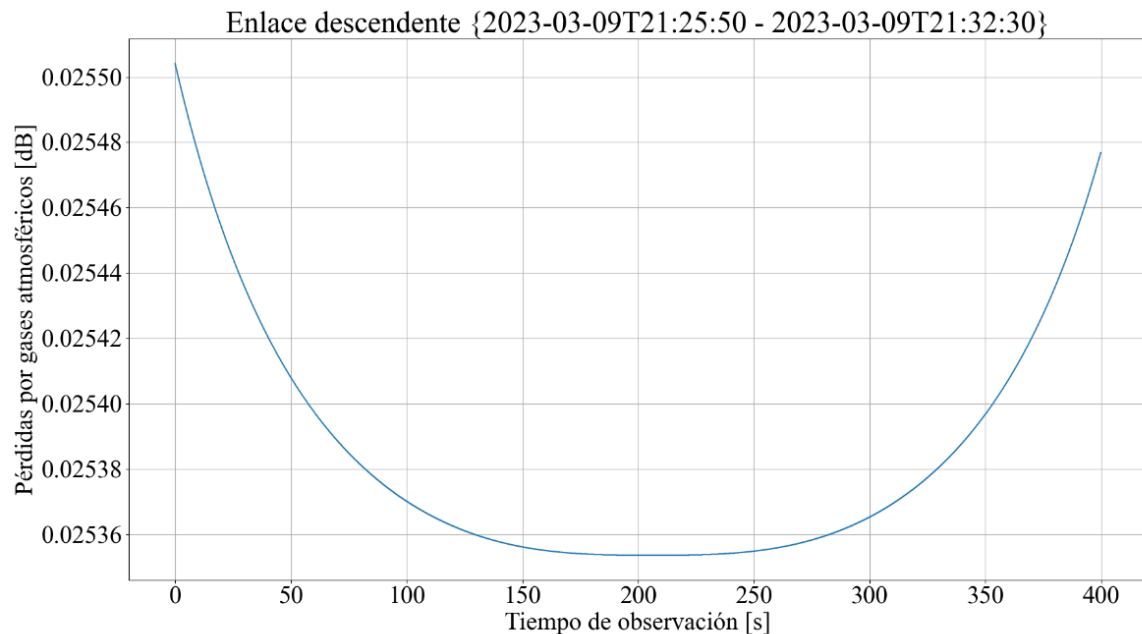


Figura 5.15.- Resultado de las pérdidas por gases atmosféricos del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

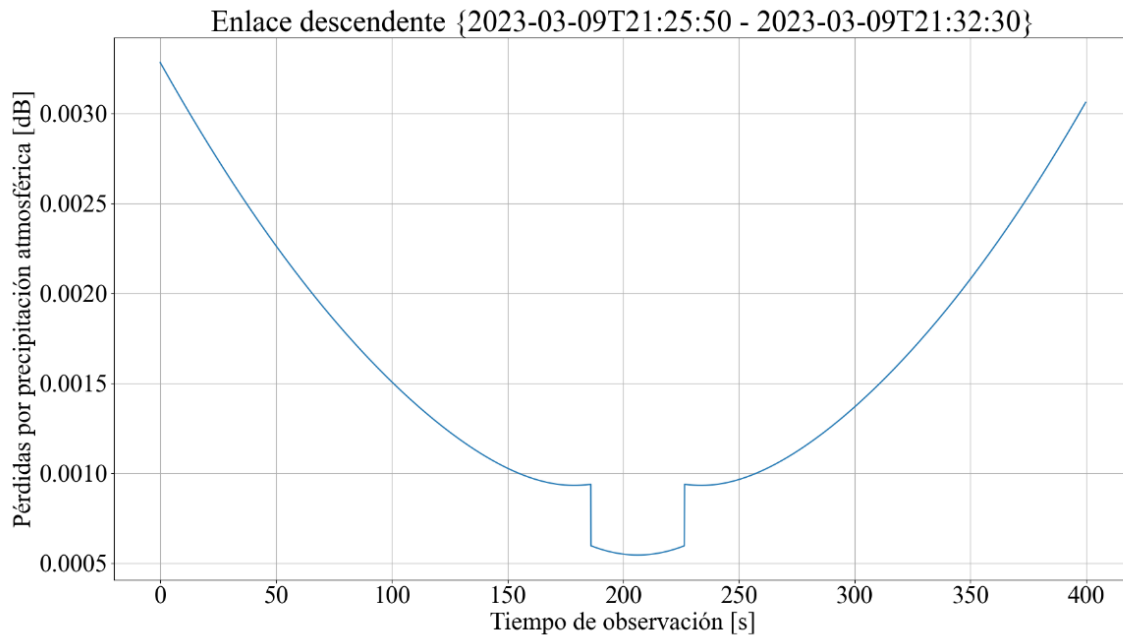


Figura 5.16.- Resultado de las pérdidas por precipitación atmosféricas del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

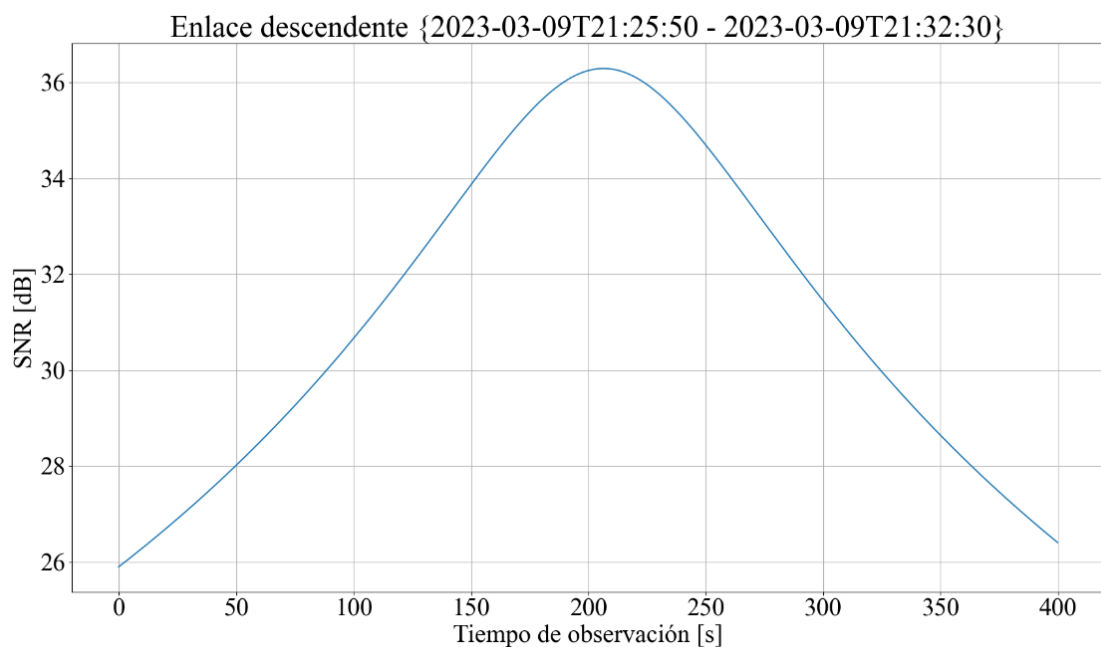


Figura 5.17.- Resultado de la SNR del enlace descendente obtenido a partir del archivo CSV generado por medio del módulo OOT gr-leo durante la ventana de tiempo desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.



5.1.2.- Análisis de los resultados

Comparando los resultados obtenidos por ambos métodos destacan las siguientes conclusiones.

En primer lugar, nótese que hay parámetros del balance de enlace realizado mediante el script de Python que la herramienta gr-leo no incluye como son la figura de mérito G/T, la energía de bit sobre ruido EbN0 o la potencia isotrópica recibida. Estos son parámetros importantes ya que pueden dar una idea de las características de rendimiento del enlace y del diseño del sistema de comunicaciones. No obstante, son parámetros que pueden obtenerse fácilmente desde los parámetros facilitados por la herramienta de GNU Radio mediante unos cálculos adicionales sencillos. Por otro lado, mientras que el balance de enlace realizado en Python ofrece sólo resultados para el peor de los casos, esto es, para la mayor distancia del enlace, el resultado del módulo OOT recoge los resultados del balance para cada momento de tiempo de la ventana de AOS. Esto puede suponer una ventaja en un análisis o planificación de misión ofreciendo más información pudiendo por ejemplo conocer la velocidad con la que mejora la SNR de la señal durante la ventana de observación seleccionada.

Otro punto destacable de los resultados de gr-leo es la inclusión de medida del efecto Doppler que sufre la señal con respecto al paso del tiempo en la duración de la observación. Esto puede suponer una gran ventaja, nuevamente para la planificación de misiones. Además, incluye un cálculo de pérdidas tanto de precipitaciones como de gases atmosféricos más preciso que el simplificado incluido en el script de Python. No obstante, cabe destacar, por parte del balance de enlace simplificado, su rápida obtención frente a la ejecución de la simulación por parte del módulo para GNU Radio. Esto también hace posible la iteración de diferentes configuraciones para un mismo análisis permitiendo la comparación del uso de diferentes valores en un tiempo mucho más reducido y de forma más sencilla sin necesidad de un post procesado de los datos.

Analizando los valores resultantes de la tabla en comparación con los correspondientes de las gráficas generadas a partir del documento CSV, destaca lo siguiente. Destacan similitudes en cuanto a los resultados obtenidos en cuanto al rango oblicuo, las pérdidas de espacio libre y la elevación, todos ellos comparados en el punto de inicio o final de observación ya que coincide con el momento sobre el cual ha sido calculado el balance de enlace del script de Python. Por otro lado, se encuentran diferencias en cuanto a los valores obtenidos en las pérdidas por gases y precipitaciones atmosféricas como ya se advirtió anteriormente. Además, el resultado de la SNR calculada por medio del balance de la herramienta gr-leo es unos 3 dB superior al restante. No obstante, esto último se debe a la inclusión de un margen de sistema propio de este tipo de balances de enlace como método de seguridad.



5.2.- Evaluación PER

Las pruebas de evaluación PER han sido realizadas para evaluar los módems mostrados y explicados en el capítulo 4 bajo las mismas condiciones de una ventana de AOS. Para esto, han sido utilizados también los scripts de Python explicados también en el capítulo 4 actuando como cliente y servidor para representar la estación terrena y CubeSat respectivamente, en el enlace ascendente y viceversa durante el enlace descendente.

En primer lugar, la ventana de observación o AOS que se ha utilizado para estas pruebas es la misma que se ha utilizado en el apartado anterior, esto es, la del enlace para el CubeSat Lume 1 desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

Nótese que, para una correcta y rigurosa evaluación de estas pruebas, se han seleccionado diferentes configuraciones basadas en potencia de transmisión, velocidad de bits, codificación empleada y longitud de paquetes.

En cuanto a la potencia de transmisión se ha tenido en cuenta la potencia entregada al receptor tras la ganancia de la antena receptora, esto es, la potencia de señal recibida. Ha sido este valor resultante el que se ha utilizado como variable de ganancia en GNU Radio previo al canal y sobre la cual se simula su paso por el mismo. Estos valores han sido escogidos en, primer lugar, de manera que sean similares al balance de enlace anteriormente analizado y reducido a continuación para observar el efecto de una transmisión con potencias menores.

Respecto a la velocidad de bits utilizada para las pruebas, se han seleccionado tasas de 4.8 y 9.6 kbps propias de enlaces de UHF. Adicionalmente, empleando únicamente el módem de Satlab, se han realizado unas pocas pruebas con tasas de velocidad de 128 y 512 kbps, propias de un enlace en banda S, para el cual está destinado a funcionar dicho módem. No obstante, estas últimas pruebas han sido reducidas en número por una serie de razones que se explican a continuación. Al optar a una tasa de bits tan elevada, la cantidad de paquetes que se pueden transmitir en la ventana de observación utilizada es muy elevada. Esto, unido al hecho de que ha de dejarse un pequeño tiempo de procesamiento entre cada transmisión de paquete del orden de entre 0.05 y 1 segundos (en función del bitrate empleado) para no saturar la red CSP, resulta en unos tiempos totales de simulación que pueden llegar a unas pocas horas. Por esto, y por falta de tiempo, se han limitado estas últimas pruebas.

Por otro lado, el tamaño de paquetes a transmitir seleccionado es de 255 bytes para todas las diferentes configuraciones. Este tamaño ha sido seleccionado de forma que no sea demasiado grande para que la probabilidad de que haya un error en un solo paquete sea menor, pero no demasiado pequeño para que el número de paquetes a transmitir no sea demasiado grande y, por tanto, el tiempo de simulación resulte aceptable.

Por último, el contenido de la payload de los paquetes utilizados ha sido adaptado en tamaño para poder transmitir siempre los paquetes con 255 bytes en función de la codificación y el formato de la trama utilizados con cada módem. Además, este contenido



ha consistido siempre en 3 bytes cuyo contenido representa el número de paquete transmitido, seguido del número de bytes de relleno con el valor 0xAA necesarios para completar los 255 bytes de longitud de paquete, teniendo en cuenta que los últimos 4 bytes de este contenido serán un código CRC32 como se advierte del algoritmo utilizado para el cálculo de la PER.

5.2.1.- Resultados

En primer lugar, cabe mencionar que, para una rigurosidad de los resultados mostrados a continuación, cada uno de ellos ha sido obtenido como el promedio de cinco repeticiones de la misma prueba.

Tabla 5.6.- Resultados de las pruebas PER realizadas del modem de Satlab para diferentes escenarios del enlace ascendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

PIRE [dBW]	GRX [dB]	G _{GNU Radio} [dB]	R _b [bps]	Codificación	Longitud Paquete [bytes]	N.º de paquetes	PER [%]
24	0	24	4800	CRC+RS+CC+Randm	255	941	0.00
12	0	12	4800	CRC+RS+CC+Randm	255	941	0.00
6	0	6	4800	CRC+RS+CC+Randm	255	941	0.00
3	0	3	4800	CRC+RS+CC+Randm	255	941	0.00
0	0	0	4800	CRC+RS+CC+Randm	255	941	15.19
24	0	24	9600	CRC+RS+CC+Randm	255	1882	0.00
12	0	12	9600	CRC+RS+CC+Randm	255	1882	0.00
6	0	6	9600	CRC+RS+CC+Randm	255	1882	0.12
3	0	3	9600	CRC+RS+CC+Randm	255	1882	16.02
0	0	0	9600	CRC+RS+CC+Randm	255	1882	45.38
24	0	24	4800	CRC+RS+Randm	255	941	0.00
12	0	12	4800	CRC+RS+Randm	255	941	0.00
6	0	6	4800	CRC+RS+Randm	255	941	1.68
3	0	3	4800	CRC+RS+Randm	255	941	29.85
0	0	0	4800	CRC+RS+Randm	255	941	55.37
24	0	24	9600	CRC+RS+Randm	255	1882	0.00
12	0	12	9600	CRC+RS+Randm	255	1882	0.00
6	0	6	9600	CRC+RS+Randm	255	1882	29.91
3	0	3	9600	CRC+RS+Randm	255	1882	57.05
0	0	0	9600	CRC+RS+Randm	255	1882	80.21
24	0	24	4800	-	255	941	0.00
12	0	12	4800	-	255	941	0.00
6	0	6	4800	-	255	941	43.91
3	0	3	4800	-	255	941	68.67
0	0	0	4800	-	255	941	95.32
24	0	24	9600	-	255	1882	0.00



12	0	12	9600	-	255	1882	15.42
6	0	6	9600	-	255	1882	69.06
3	0	3	9600	-	255	1882	95.04
0	0	0	9600	-	255	1882	100.00

Tabla 5.7.- Resultados de las pruebas PER realizadas del modem de Satlab para diferentes escenarios del enlace descendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

PIRE [dBW]	GRX [dB]	G _{GNU Radio} [dB]	R _b [bps]	Codificación	Long. Paquete [bytes]	N.º de paquetes	PER [%]
0	15	15	4800	CRC+RS+CC+Randm	255	941	0.00
-3	15	12	4800	CRC+RS+CC+Randm	255	941	0.00
-6	15	9	4800	CRC+RS+CC+Randm	255	941	0.00
-9	15	6	4800	CRC+RS+CC+Randm	255	941	1.63
-12	15	3	4800	CRC+RS+CC+Randm	255	941	30.51
0	15	15	9600	CRC+RS+CC+Randm	255	1882	0.00
-3	15	12	9600	CRC+RS+CC+Randm	255	1882	0.00
-6	15	9	9600	CRC+RS+CC+Randm	255	1882	1.59
-9	15	6	9600	CRC+RS+CC+Randm	255	1882	31.06
-12	15	3	9600	CRC+RS+CC+Randm	255	1882	56.83
0	15	15	4800	CRC+RS+Randm	255	941	0.00
-3	15	12	4800	CRC+RS+Randm	255	941	0.43
-6	15	9	4800	CRC+RS+Randm	255	941	12.54
-9	15	6	4800	CRC+RS+Randm	255	941	42.62
-12	15	3	4800	CRC+RS+Randm	255	941	66.70
0	15	15	9600	CRC+RS+Randm	255	1882	0.47
-3	15	12	9600	CRC+RS+Randm	255	1882	13.54
-6	15	9	9600	CRC+RS+Randm	255	1882	43.29
-9	15	6	9600	CRC+RS+Randm	255	1882	68.16
-12	15	3	9600	CRC+RS+Randm	255	1882	94.39
0	15	15	4800	-	255	941	2.83
-3	15	12	4800	-	255	941	28.98
-6	15	9	4800	-	255	941	54.87
-9	15	6	4800	-	255	941	80.09
-12	15	3	4800	-	255	941	99.97
0	15	15	9600	-	255	1882	30.10
-3	15	12	9600	-	255	1882	56.28
-6	15	9	9600	-	255	1882	81.49
-9	15	6	9600	-	255	1882	99.89
-12	15	3	9600	-	255	1882	100.00



Tabla 5.8.- Resultados de las pruebas PER realizadas del modem GMSK para diferentes escenarios del enlace ascendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

PIRE [dBW]	GRX [dB]	GGNU Radio [dB]	R _b [bps]	Codificación	Long. Paquete [bytes]	N.º de paquetes	PER [%]
24	0	24	4800	Longitud Golay	255	941	0.00
12	0	12	4800	Longitud Golay	255	941	0.00
6	0	6	4800	Longitud Golay	255	941	0.00
3	0	3	4800	Longitud Golay	255	941	0.00
0	0	0	4800	Longitud Golay	255	941	2.10
-3	0	-3	4800	Longitud Golay	255	941	28.50

Tabla 5.9.- Resultados de las pruebas PER realizadas del modem GMSK para diferentes escenarios del enlace descendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

PIRE [dBW]	GRX [dB]	GGNU Radio [dB]	R _b [bps]	Codificación	Long. Paquete [bytes]	N.º de paquetes	PER [%]
0	15	15	4800	Longitud Golay	255	941	0.00
-3	15	12	4800	Longitud Golay	255	941	0.00
-6	15	9	4800	Longitud Golay	255	941	0.00
-9	15	6	4800	Longitud Golay	255	941	0.28
-12	15	3	4800	Longitud Golay	255	941	11.94
-15	15	0	4800	Longitud Golay	255	941	41.82

Tabla 5.10.- Resultados de las pruebas PER realizadas del modem de Satlab en banda S para diferentes escenarios del enlace ascendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

PIRE [dBW]	GRX [dB]	GGNU Radio [dB]	R _b [bps]	Codificación	Long. Paquete [bytes]	N.º de paquetes	PER [%]
40	0	40	128000	CRC+RS+CC+Randm	255	25098	0.00
32	0	40	128000	CRC+RS+CC+Randm	255	25098	49.05
24	0	32	128000	CRC+RS+CC+Randm	255	25098	100.00
40	0	40	512000	CRC+RS+CC+Randm	255	100392	82.41
32	0	40	512000	CRC+RS+CC+Randm	255	100392	100.00
24	0	32	512000	CRC+RS+CC+Randm	255	100392	100.00



Tabla 5.11.- Resultados de las pruebas PER realizadas del modem de Satlab en banda S para diferentes escenarios del enlace descendente del Lume 1 en una ventana de AOS desde el 09/03/2023 a las 21:25:50 hasta el 09/03/2023 a las 21:32:30.

PIRE [dBW]	GRX [dB]	G _{GNU Radio} [dB]	R _b [bps]	Codificación	Long. Paquete [bytes]	N.º de paquetes	PER [%]
6	20	26	128000	CRC+RS+CC+Randm	255	25098	21.35
3	20	23	128000	CRC+RS+CC+Randm	255	25098	49.55
0	20	20	128000	CRC+RS+CC+Randm	255	25098	72.83
6	20	26	512000	CRC+RS+CC+Randm	255	100392	100.00
3	20	23	512000	CRC+RS+CC+Randm	255	100392	100.00
0	20	20	512000	CRC+RS+CC+Randm	255	100392	100.00

5.2.2.- Análisis de los resultados

Con relación a todos los resultados obtenidos, existe una relación directa entre la potencia transmitida utilizada en la simulación frente a la PER obtenida. Esta relación mejora el resultado de la PER según esta potencia incrementa en valor haciendo que la SNR de la señal recibida sea mayor y, por tanto, la probabilidad de error en la señal sea menor. Esto es muy notable comparando sobre todo los enlaces ascendentes con los descendentes, resultando los descendentes en unos peores valores de PER dado que para el enlace descendente se dispone generalmente de una menor potencia de transmisión que desde la estación terrena. Otro aspecto remarcable es el hecho de que los resultados de PER empeoran también directamente proporcionales a la velocidad de bits utilizada. Esto se debe a que, según aumenta la velocidad de bits, el ancho de banda de la señal también lo hace proporcionalmente, lo que resulta en una potencia de ruido mayor, y, por tanto, una menor SNR en el receptor. Por último, en cuanto a la relación de todos los resultados se refiere, el empleo de técnicas de detección y corrección de errores FEC juega un papel muy importante en los enlaces resultando en una mejora notable en los resultados para todos los escenarios.

Por otro lado, comparando ambos módems, se observa un comportamiento significativamente mejor por parte del modem diseñado a partir del diseño previo de Alén Space respecto al modem comercial de Satlab incluso empleando todas las técnicas de detección y corrección de errores disponibles. Esto se debe principalmente a un mejor diseño y optimización del modem GMSK basado en el de Alén Space, donde, por medio únicamente de una codificación de la cabecera para asegurar una correcta decodificación del cuerpo del paquete, junto con una optimización de parámetros de modulación y demodulación de la señal, se consigue un comportamiento muy eficiente.

No obstante, el hecho de que el modem GMSK ofrezca un mejor rendimiento no lo convierte necesariamente en el mejor. Es importante tener en cuenta que el modem de



Satlab es mucho más flexible, ofreciendo la posibilidad de incluir diferentes técnicas de codificación e incluso de emplear el bitrate deseado.

Por último, respecto a las pruebas realizadas con el módem de Satlab en banda S, el análisis sería el mismo a lo establecido previamente. Al transmitir con una potencia mayor, la PER mejora, aunque, al transmitir con un bitrate tan elevado, el suelo de ruido sube, dificultando la correcta demodulación de la señal al obtenerse una SNR menor.



6.- CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

A continuación, se presentan las conclusiones del desarrollo de este proyecto. En primer lugar, se presentan unas conclusiones generales junto con unos casos de uso que se le pueden dar a esta herramienta, y, finalmente, las líneas de trabajo futuras que ampliarían este proyecto.

6.1.- Conclusiones

En este proyecto se ha desarrollado una herramienta que posibilita la simulación de las comunicaciones propias de misiones de satélites de órbita baja terrestre. Para ello se ha hecho uso de la herramienta GNU Radio junto con la red CSP, asistidas por otras tecnologías como son Docker y GPredict. El resultado ha sido una herramienta cuya funcionalidad ha sido probada evaluando la PER de dos módems, uno comercial y otro implementado a partir del diseño previo por parte de la empresa Alén Space basado en el transceptor AX100 de GomSpace. A partir de dicha evaluación se ha podido mostrar el mejor rendimiento del segundo frente al comercial gracias a un diseño y optimización exhaustivos. No obstante, el modem comercial ofrece una mayor flexibilidad en cuanto a las opciones de configuración de codificación y bitrates que ofrece.

A continuación, se muestran algunos casos interesantes para los cuales se puede emplear la herramienta desarrollada.

- **Planificación de misiones.** Esta herramienta puede servir como una eficiente herramienta de balance de enlace que permite a los usuarios estimar parámetros de comunicación esenciales para el éxito de un enlace de comunicación entre un satélite y la Tierra. Esto incluye el cálculo de la relación señal a ruido, determinando el rango máximo oblicuo que puede estar presente en el enlace, y estimando la deriva máxima de frecuencia que el canal de comunicación puede encontrar.
- **Simulación y evaluación de TX/RX.** También puede servir para diversos propósitos, incluyendo experimentación o investigación, relacionados con los aspectos de transmisión y recepción del enlace de comunicación entre satélites de órbita baja y Tierra. Ya sea que se quiera explorar nuevas técnicas de comunicación, evaluar el rendimiento de un sistema de comunicación u obtener información sobre el funcionamiento del enlace de comunicación, esta herramienta puede ser un recurso valioso.
- **Caracterización del rendimiento de módems.** Se ofrece la flexibilidad de incorporar módulos externos, como módems y decodificadores, que pueden ser evaluados y caracterizados en escenarios específicos. Esto puede ser



particularmente útil para optimizar el diseño de un sistema e identificar posibles problemas antes de implementarlo en un entorno real.

- **Validación de TLE y/o corrección de frecuencia.** La simulación orbital proporcionada está basada en un conjunto de TLE como parámetro para modelar con precisión la posición y el movimiento del satélite. Esta característica permite a los usuarios validar los datos de TLE y verificar su precisión en la predicción de la posición y trayectoria del satélite. Además, se facilita la prueba y validación de correctores de efecto Doppler utilizando sus capacidades de simulación de canal.
- **Aplicación docente y de aprendizaje.** Por medio de la posibilidad de experimentación e investigación que ofrece la herramienta en cuanto a su modo de empleo, es fantástica para la aplicación docente y el aprendizaje. La posibilidad de creación de diferentes bloques de comunicaciones y poder simular su uso directamente en órbita ofrece un gran valor para poder desarrollar en un entorno seguro y con bajos costes.

6.2.- Líneas de trabajo futuras

Las líneas de trabajo futuras más destacadas con el propósito de ampliar la utilidad de esta herramienta y aportar una mayor facilidad de uso son las que aparecen descritas a continuación:

- **Pruebas de sistemas hardware.** Sería de interés realizar pruebas y facilitar la evaluación de sistemas de comunicaciones hardware por medio de elementos de hardware para SDR como pueden ser los dispositivos USRP de Ettus Research [43]. De esta forma sería sencillo simular la puesta en vuelo de sistemas como TTCs que irían a bordo de un CubeSat proporcionando una útil herramienta a los equipos de testeo de este tipo de subsistemas.
- **Pruebas de corrección Doppler.** Sería igualmente de interés realizar pruebas donde se verifique el funcionamiento de sistemas que garanticen una corrección de la desviación Doppler.
- **Interfaz más amigable al usuario.** Por último, una interfaz de usuario haría más llevadera la interacción con la herramienta facilitando su uso. También se podría acompañar con pruebas de evaluación predefinidas para casos comunes concretos.



REFERENCIAS

- [1] Página web de Alén Space.
<https://alen.space/es/inicio/> [Junio de 2023]
- [2] *Applying HOL/PBL to Prepare Undergraduate Students into Graduate Level Studies in the Field of Aerospace Engineering Using the Puerto Rico CubeSat Project Initiative - Scientific Figure on ResearchGate.*
https://www.researchgate.net/figure/Standard-2U-CubeSat-diagram-CubeSats-are-small-scale-satellites-composed-of-several_fig1_331595385 [Junio de 2023]
- [3] Recomendación P.619-3 de la UIT. “*Propagation data required for the evaluation of interference between stations in space and those on the surface of the Earth*”. (12/2017)
- [4] Recomendación P.676-11 de la UIT, Anexo 1. “*Attenuation by atmospheric gases*”. (09/2016)
- [5] Recomendación P.618-13 de la UIT. “*Propagation data and prediction methods required for the design of Earth-space telecommunication systems*”. (12/2017)
- [6] Recomendación P.837-7 de la UIT. “*Características de la precipitación para establecer modelos de propagación*”. (06/2017)
- [7] *The Range and Horizon Plane Simulation for Ground Stations of Low Earth Orbiting (LEO) Satellites - Scientific Figure on ResearchGate.*
https://www.researchgate.net/figure/Ground-station-geometry_fig2_220099290 [Junio de 2023]
- [8] Edición de 2020 del Reglamento de Radiocomunicaciones de la UIT
- [9] Hoja de características del AX100.
<https://gomspace.com/UserFiles/Subsystems/datasheet/gs-ds-nanocom-ax100-33.pdf>
[Junio 2023]
- [10] BOE-A-2022-10757 Ley 11/2022, de 28 de Junio, General de Telecomunicaciones.
- [11] Manual de atenuación de Cable Coaxial.
http://rfelektronik.se/manuals/Datasheets/Coaxial_Cable_Attenuation_Chart.pdf
[Junio de 2023]



- [12] Hoja de características de la antena Wimo X-Quad.
http://www.radiomanual.info/schemi/ACC_antenna/Wimo_X-Quad_user_OZ1BXM.pdf [Junio de 2023]
- [13] Página web de ISISpace.
<https://www.isispace.nl/product/cubesat-antenna-system-1u-3u/> [Junio de 2023]
- [14] Página web de GPredict.
<http://gpredict.oz9aec.net> [Junio de 2023]
- [15] Página web de Docker.
<https://www.docker.com> [Junio de 2023]
- [16] Aplicaciones y Sistemas Distribuidos. Apuntes del Tema 4 de la asignatura del segundo semestre de 1º de Máster en Ingeniería de Telecomunicaciones, 2022. José Luis Díaz de Arriba.
- [17] Manual para el comando build de Docker.
[https:// docs.docker.com /engine/reference /commandline/build/](https://docs.docker.com/engine/reference/commandline/build/) [Junio de 2023]
- [18] Manual para el comando run de Docker.
<https://docs.docker.com/engine/reference/run/> [Junio de 2023]
- [19] Página web de GNU Radio.
<https://www.gnuradio.org> [Junio de 2023]
- [20] Página web de Ubuntu LTS
<https://releases.ubuntu.com/jammy/> [Junio de 2023]
- [21] Repositorio GitHub de GNU Radio.
<https://github.com/gnuradio/gnuradio> [Junio de 2023]
- [22] Manual de GNU Radio y C++ API Reference.
<https://www.gnuradio.org/doc/doxygen/index.html> [Junio de 2023]
- [23] Manual de uso de Polimorphic Types en GNU Radio.
[https://wiki.gnuradio.org/index.php/Polymorphic_Types_\(PMTs\)](https://wiki.gnuradio.org/index.php/Polymorphic_Types_(PMTs)) [Junio de 2023]
- [24] Repositorio Gitlab de la herramienta gr-leo.
<https://gitlab.com/librespacefoundation/gr-leo> [Junio de 2023]



- [25] Apartado de problemas del repositorio Gitlab de la herramienta gr-leo.
<https://gitlab.com/librespacefoundation/gr-leo/-/issues/45> [Junio de 2023]
- [26] *Radiowave Propagation in Satellite Communications*, escrito por Louis J. Ippolito Jr.
- [27] Repositorio GitHub de la librería SGP4.
<https://github.com/dnwrnr/sgp4> [Junio de 2023]
- [28] "AMSAT-IARU Link Model".
<http://www.amsatuk.me.uk/iaru/spreadsheet.htm> [Junio 2023]
- [29] ARRL Antenna Book.
https://www.qsl.net/sp9hzx/pdf2/arrl_antenna_book_21st_ed.pdf [Junio de 2023]
- [30] Lume 1.
<https://en.wikipedia.org/wiki/Lume-1> [Junio de 2023]
- [31] Conjunto de TLE proporcionado por Celestrak.
<http://celestrak.org/NORAD/elements/cubesat.txt> [Junio de 2023]
- [32] Página web de GomSpace.
<https://gomspace.com/home.aspx> [Junio de 2023]
- [33] Repositorio GitHub de LibCSP.
<https://github.com/libcsp/libcsp> [Junio de 2023]
- [34] Página de CSP en Wikipedia.
https://en.wikipedia.org/wiki/Cubesat_Space_Protocol [Junio de 2023]
- [35] Repositorio GitHub de la versión 1.6 de LibCSP. The basics of CSP.
<https://github.com/libcsp/libcsp/blob/v1.6/doc/basic.rst> [Junio de 2023]
- [36] Manual de GomSpace CubeSat Space Protocol (CSP)
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUK EwjL3YTRvNb_AhVQUKQEhdVDBooQFnoECBQQAQ&url=https%3A%2F%2Fbytebucket.org%2Fbbruner0%2Falbertasat-on-board-computer%2Fwiki%2F1.%2520Resources%2F1.1.%2520DataSheets%2FCSP%2FGS-CSP-1.1.pdf%3Frev%3D316ebd49bed49fdbb1d74efdeab74430e7cc726a&usg=AOvVaw2n0ibHqJThJcQFJ9ZHj0tF&opi=89978449 [Junio de 2023]



- [37] Manual de portación de versiones de módulos OOT en GNU Radio.
https://wiki.gnuradio.org/index.php/GNU_Radio_3.10_OOT_Module_Porting_Guide
[Junio de 2023]
- [38] Manual de uso de la herramienta gr-modtool de GNU Radio.
https://wiki.gnuradio.org/index.php?title=Creating_C%2B%2B_OOT_with_gr-modtool [Junio de 2023]
- [39] GS Kit de Alén Space.
<https://products.alen.space/products/ground-station-for-uhf-vhf-s-band/> [Junio de 2023]
- [40] Recommendation for Space Data System Standards. TM Synchronization and Channel Coding. CCSDS.
<https://public.ccsds.org/Pubs/131x0b4.pdf> [Junio de 2023]
- [41] Módem para GNU Radio de Satlab.
<https://www.satlab.com/software/gr-satlab/> [Junio de 2023]
- [42] Página web de Satlab.
<https://www.satlab.com> [Junio de 2023]
- [43] Página web de Ettus Research.
<https://www.ettus.com> [Junio de 2023]



ANEXO A. BALANCE DE ENLACE PYTHON

A continuación, se muestra el código de Python utilizado para los balances de enlace calculados en este proyecto.

```

from scipy.constants import k, c, degree #degree = pi/180 para conversion a rads
from math import log10, sqrt, sin, cos, atan, pi, exp, e
from tabulate import tabulate
import pandas as pd

to_dB = lambda x: 10*log10(x)
to_nu = lambda x: 10**(x/10)

# --- Orbit characterization ---
def slant_range(parameters):
    """
    Reference for slant range calculation:
    ["The Range and Horizon Plane Simulation for Ground Stations of Low Earth Orbiting (LEO)
    Satellites"]
    (https://www.researchgate.net/publication/220099290\_The\_Range\_and\_Horizon\_Plane\_Simulation\_for\_Ground\_Stations\_of\_Low\_Earth\_Orbiting\_LEO\_Satellites)
    """
    Re = 6_371_010 # Earth radius (meters)
    apogee = parameters['Apogee'] # Furthest orbital point from earth
    perigee = parameters['Perigee'] # Closest orbital point to earth
    gs_elevation = parameters['Elevation']

    # Max altitude of the orbit
    h = max(apogee, perigee)

    # Slant range calculation
    slant_range = Re * ( sqrt( ((h + Re)/Re)**2 - cos(gs_elevation)**2 ) - sin(gs_elevation) )

    return slant_range

# --- Transmitter characterization ---
def eirp(parameters):
    tx_eirp = sum(parameters.values())
    return tx_eirp

```



```

# --- Friis formula ---

def rain_loss_fun(parameters):
    """
    - Rain loss reference: https://www.itu.int/dms\_pubrec/itu-r/rec/p/R-REC-P.618-13-201712-I!!PDF-S.pdf
    https://www.itu.int/dms\_pubrec/itu-r/rec/p/R-REC-P.837-7-201706-I!!PDF-S.pdf
    https://www.itu.int/rec/R-REC-P.837-7-201706-I/en (mapa rain rate)
    https://www.itu.int/dms\_pubrec/itu-r/rec/p/R-REC-P.838-3-200503-I!!PDF-E.pdf
    https://www.itu.int/dms\_pubrec/itu-r/rec/p/R-REC-P.839-4-201309-I!!PDF-S.pdf (determine rain height)
    """

    if isinstance(parameters['Rain Losses'], dict):
        lat = 42.2406 # [degree] Vigo latitude
        h_s = 0.2 # [km] GS height
        R_e = 8_500 # [km] effective radius of the Earth (8 500 km).
        theta = parameters['Slant Range']['Elevation'] # [rads] elevation
        # Step 1: Determine rain height h_r
        h_0 = 1.408 # [km] https://www.itu.int/dms\_pubrec/itu-r/rec/p/R-REC-P.839-4-201309-I!!PDF-S.pdf
        h_r = h_0 + 0.36 # [km]
        # Step 2: For elevation > 5° (always pretty much) compute the slant-path length L_s else use other formula
        if parameters['Slant Range']['Elevation'] > 5*degree:
            L_s = (h_r - h_s)/sin(theta) # [km]
        else:
            L_s = 2*(h_r - h_s)/(((sin(theta))**2+(2*(h_r - h_s)/R_e)**(1/2)+sin(theta))
        # Step 3: Calculate the horizontal projection, L_g, of the slant-path length
        L_g = L_s*cos(theta) # [km]
        # Step 4: Obtain the rainfall rate, R0.01, exceeded for 0.01% of an average year from Recommendation ITU-R P.837 (mapa rain rate)
        R001 = parameters['Rain Losses']['Rain rate'] #Obtained by looking at the map approximately
        if R001 == 0:
            rain_loss_ret = 0
            return rain_loss_ret

        # Step 5: Obtain the specific attenuation, gamma_R, using the frequency-dependent coefficients given in Recommendation ITU-R P.838
        # and the rainfall rate, R0.01, determined from Step 4
        k_h = parameters['Rain Losses']['K_h']
        alpha_h = parameters['Rain Losses']['Alpha_h']
        k_v = parameters['Rain Losses']['K_v']
        alpha_v = parameters['Rain Losses']['Alpha_v']

```



```

R = parameters['Rain Losses']['Rain rate'] # Rain rate in mm/h
tau = 45 * degree # 45 for circular polarization
k = (k_h + k_v + (k_h - k_v) * cos(theta)**2 * cos(2*tau)) / 2
if k == 0:
    alpha = 0
else:
    alpha = (k_h * alpha_h + k_v * alpha_v + (k_h * alpha_h - k_v * alpha_v) *
cos(theta)**2 * cos(2*tau)) / (2*k)
gamma_R = k * (R**alpha) # Rain attenuation in dB/km
# Step 6: Calculate the horizontal reduction factor, r0.01, for 0.01% of the time
f = parameters['Frequency']
r001 = 1/(1+0.78*sqrt(gamma_R*L_g/f)-(0.38*(1-exp(-2*L_g))))
# Step 7: Calculate the vertical adjustment factor, v0.01, for 0.01% of the time
dzeta = atan((h_r-h_s)/(L_g*r001))/degree
if dzeta>theta:
    L_r = L_g*r001/cos(theta)
else:
    L_r = (h_r - h_s)/sin(theta)
if abs(lat) > 36:
    chi = 0
else:
    chi = 36 - abs(lat)
v001 = 1/(1+sqrt(sin(theta))*(31*(1-exp(-(theta/(chi+1)))))*sqrt(L_r*gamma_R)/f**2-0.45))
# Step 8: The effective path length is
L_e = L_r*v001
# Step 9: The predicted attenuation exceeded for 0.01% of an average year is obtained from
A001 = gamma_R*L_r # [dB]
# Step 10: The estimated attenuation to be exceeded for other percentages of an average
year, in the range
# 0.001% to 5%, is determined from the attenuation to be exceeded for 0.01% for an average
year
# Not necessary ...
rain_loss_ret = A001
else:
    rain_loss_ret = parameters['Rain Losses']
return rain_loss_ret

def gases_loss_fun(parameters):
    """
    - Gasses loss reference: https://www.itu.int/dms\_pubrec/itu-r/rec/p/R-REC-P.676-11-201609-I!!PDF-E.pdf
    """

```



```

# For an elevation angle, phi, between 5° and 90°, the path attenuation is obtained using the
# cosecant law

phi = parameters['Slant Range']['Elevation'] # [rads] elevation
f = parameters['Frequency']
p = 1_020 # hPa (presion atmosférica de Vigo)
rp = (p + e)/1013.25
gamma0 = parameters['Gases'] # specific attenuation for dry air (aproximation from figure 5)
gammaw = 0 # specific attenuation for moist (aproximation from figure 5)
t1 = 4.64/(1+0.066*rp**(-2.3))*exp(-((f-59.7)/(2.87+12.4*exp(-7.9*rp)))**2)
t2 = 0.14*exp(2.12*rp)/((f-118.75)**2+(0.031*exp(2.2*rp)))
t3 = 0.0114/(1+0.14*rp**(-2.6))*f*(-0.0247+0.0001*f+1.61e-6*f**2)/(1-0.0169*f+4.1e-5*f**2+3.2e-
7*f**3)

sigmaw = 1.013/(1+exp(-8.6*(rp-0.57)))
h0 = 6.1/(1+0.17*rp**(-1.1))*(1+t1+t2+t3)
hw = 1.66*(1+(1.39*sigmaw)/((f-22.235)**2+2.56*sigmaw)+(3.37*sigmaw)/((f-
183.31)**2+4.69*sigmaw)+(1.58*sigmaw)/((f-325.1)**2+2.89*sigmaw))
A0 = h0 * gamma0 # attenuation for dry air
Aw = hw * gammaw # attenuation for moist air

# cosecant law
A = (A0 + Aw)/sin(phi)
gases_loss_ret = A
return gases_loss_ret

def other_loss_fun(parameters):
    if isinstance(parameters['Other Losses'], dict):
        other_losses_ret = sum(parameters['Other Losses'].values())
    else:
        other_losses_ret = parameters['Other Losses']
    return other_losses_ret

def free_space_loss_fun(parameters):
    """
    - Free Space Path Loss reference: https://en.wikipedia.org/wiki/Friis\_transmission\_equation
    """
    # For generic budget: slant range depends on gs_elevation
    if isinstance(parameters['Slant Range'], dict):
        slant_range_val = slant_range(parameters['Slant Range'])
    # For ISL budget: the slant range is constant
    else:
        slant_range_val = parameters['Slant Range']

```



```

    freq = parameters['Frequency']
    free_space_loss = 20 * log10(4 * pi * slant_range_val * freq / c)
    return free_space_loss

def friis_formula(parameters):
    """
    Here we sum up the signal loss due to pointing, atmospheric and ionospheric effects, rain,
    polarization, free space propagation...
    """
    # Free Space Loss
    free_space_loss = free_space_loss_fun(parameters)
    # Polarization Loss
    polarization_loss = parameters['Polarization Loss']
    # Pointing Loss
    pointing_loss = parameters['Pointing Loss']
    # Rain Loss
    rain_loss = rain_loss_fun(parameters)
    # Gasses Loss
    gases_loss = gases_loss_fun(parameters)
    # Other Losses
    other_loss = other_loss_fun(parameters)
    # Complete Path Loss
    total_rf_loss = polarization_loss + pointing_loss + free_space_loss + rain_loss + other_loss +
    gases_loss
    return total_rf_loss

def isotropic_rx_signal_level(tx_parameters, friis_parameters):
    isotropic_rx_signal_level = eirp(tx_parameters) - friis_formula(friis_parameters)
    return isotropic_rx_signal_level # dBW

# --- Receiver characterization ---
def rx_gain(parameters):
    G = parameters['Antenna gain']
    return G

def rx_g_t(parameters):
    G = rx_gain(parameters)
    G_T = G - to_dB(parameters['Effective Noise Temperature'])
    return G_T

# --- SNR and Eb/N0 Calculations ---

```



```
"""
```

```
In this case, we use the bandwidth of the signal for SNR calculation, as we are measuring power over that bandwidth
```

```
"""
```

```
def snr(tx_parameters, rx_parameters, friis_parameters, bandwidth):
    G_T = rx_g_t(rx_parameters)
    isotropic_rx_signal_level_dB = isotropic_rx_signal_level(tx_parameters, friis_parameters)
    SNR = (isotropic_rx_signal_level_dB + G_T - to_dB(k) - to_dB(bandwidth)) # dB
    return SNR
```

```
def eb_n0(tx_parameters, rx_parameters, friis_parameters, bitrate):
    G_T = rx_g_t(rx_parameters)
    isotropic_rx_signal_level_dB = isotropic_rx_signal_level(tx_parameters, friis_parameters)
    EbN0 = (isotropic_rx_signal_level_dB + G_T - to_dB(k) - to_dB(bitrate)) # dB
    return EbN0
```

```
def sn0(tx_parameters, rx_parameters, friis_parameters):
    G_T = rx_g_t(rx_parameters)
    isotropic_rx_signal_level_dB = isotropic_rx_signal_level(tx_parameters, friis_parameters)
    SN0 = (isotropic_rx_signal_level_dB + G_T - to_dB(k)) # dB
    return SN0
```

```
def noise_power(rx_parameters, bandwidth):
    rx_noise_temp = rx_parameters['Effective Noise Temperature']
    noise_power = to_dB(k * rx_noise_temp * bandwidth) # dBW
    return noise_power
```

```
def received_power(tx_parameters, rx_parameters, friis_parameters):
    G = rx_gain(rx_parameters)
    isotropic_rx_signal_level_dB = isotropic_rx_signal_level(tx_parameters, friis_parameters)
    received_power = (isotropic_rx_signal_level_dB + G) # dBW
    return received_power
```

```
def bitrate_from_ebn0(tx_parameters, rx_parameters, friis_parameters, ebn0):
    G_T = rx_g_t(rx_parameters)
    isotropic_rx_signal_level_dB = isotropic_rx_signal_level(tx_parameters, friis_parameters)
    bitrate = to_nu(isotropic_rx_signal_level_dB + G_T - to_dB(k) - ebn0)
    return bitrate
```

```
# --- Final Results ---
```

```
def calculate_all_results(name, link):
```



```

results = {
    'Link': name,
    'Operating Freq [MHz]': link['friis_parameters']['Frequency']/1e6,
    'Elevation [°]': link['friis_parameters']['Slant Range']['Elevation'] / degree,
    'Slant Range [km]': slant_range(link['friis_parameters']['Slant Range']) / 1000,
    'Free Space Loss [dB]': free_space_loss_fun(link['friis_parameters']),
    'Rain Loss [dB]': rain_loss_fun(link['friis_parameters']),
    'Gasses Loss [dB]': gases_loss_fun(link['friis_parameters']),
    'Pol Losses [dB]': link['friis_parameters']['Polarization Loss'],
    'Pointing Losses [dB]': link['friis_parameters']['Pointing Loss'],
    'Other Losses [dB]': other_loss_fun(link['friis_parameters']),
    'Total Path Att [dB]': friis_formula(link['friis_parameters']),
    'EIRP [dBW]': eirp(link['tx_parameters']),
    'Isotropic Rx Level [dBW]': isotropic_rx_signal_level(link['tx_parameters'],
link['friis_parameters']),
    'G/T [dB]': rx_g_t(link['rx_parameters']),
    'S [dBW]': received_power(link['tx_parameters'], link['rx_parameters'],
link['friis_parameters']),
    'N [dBW]': noise_power(link['rx_parameters'], link['bandwidth']),
    'SN0 [dBHz]': sn0(link['tx_parameters'], link['rx_parameters'],
link['friis_parameters']),
    'SNR [dB]': snr(link['tx_parameters'], link['rx_parameters'], link['friis_parameters'],
link['bandwidth']) - link['System Margin'],
    'Eb/N0 [dB]': eb_n0(link['tx_parameters'], link['rx_parameters'],
link['friis_parameters'], link['bitrate']) - link['System Margin'],
}

return results

def print_results(joint_dict):
    df = pd.DataFrame(joint_dict)
    df = df.T
    print(tabulate(df, tablefmt='rounded_grid', showindex="always"))

```




ANEXO B. DOCKER

A continuación, se muestra el Dockerfile utilizado para el desarrollo de GNU Radio.

```
FROM ubuntu:22.04
LABEL maintainer="Guillermo Lena - guillermo.lena@alen.space"
# You use this mode when you need zero interaction while installing or upgrading the system via
apt.
ENV DEBIAN_FRONTEND=noninteractive
ENV LC_ALL=C.UTF-8
ENV LANG=C.UTF-8
# Use GCC-11
ENV CC=/usr/bin/gcc-11
ENV CXX=/usr/bin/g++-11
RUN apt-get update && apt-get install -y \
    xterm \
    cmake \
    software-properties-common \
    git
RUN apt-get update && apt-get install -y \
    libusb-1.0-0-dev \
    libssl-dev \
    liborc-dev \
    libzmq5 \
    libzmq3-dev \
    swig \
    python3 \
    python3-pip \
    vim \
    nano \
    gcc-10 g++-10 \
    libboost-all-dev \
    libcppunit-dev \
    liblog4cpp5-dev \
    python3-pygccxml \
    pybind11-dev
RUN update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-10 100 \
    --slave /usr/bin/g++ g++ /usr/bin/g++-10 --slave /usr/bin/gcov gcov /usr/bin/gcov-10
# else it will output an error about Gtk namespace not found
RUN apt-get install -y gir1.2-gtk-3.0
# to have add-apt-repository available
```



```

RUN apt-get install -y software-properties-common
# create user gnuaradio with sudo (and password gnuradio)
RUN apt-get install -y sudo
RUN useradd --create-home --shell /bin/bash -G sudo gnuradio
RUN echo 'gnuradio:gnuradio' | chpasswd
# I create a dir at home which I'll use to persist after the container is closed (need to change
it's ownership)
RUN mkdir /home/gnuradio/persistent-folder && chown gnuradio /home/gnuradio/persistent-folder
RUN add-apt-repository -y ppa:gnuradio/gnuradio-releases
RUN apt-get update && apt-get install -y gnuradio
RUN sudo ldconfig
# installing other packages needed for downloading and installing OOT modules
RUN apt-get install -y gnuradio-dev
# Download UHD images
RUN apt-get install -y uhd-host
RUN /usr/lib/uhd/uhd_images_downloader.py
RUN pip install psutil
RUN pip install psutil configargparse
# Install limesuite dependencies
RUN add-apt-repository -y ppa:myriadrf/drivers
RUN apt-get update && apt-get install -y limesuite
# of course, nothing useful can be done without vim
RUN apt-get install -y vim
# Fix dependency issue as noted on 'InstallingGR' wiki page for 3.10
RUN pip install packaging
# install gr-satellites
RUN git clone https://github.com/daniestevez/gr-satellites.git
WORKDIR /gr-satellites/
RUN mkdir build
WORKDIR /gr-satellites/build/
RUN cmake -DCMAKE_INSTALL_PREFIX=/usr/local .. && make && make install && ldconfig
WORKDIR /gr-satellites/grc/
RUN mkdir build
WORKDIR /gr-satellites/grc/build/
RUN cmake -DCMAKE_INSTALL_PREFIX=/usr/local .. && make && make install && ldconfig
WORKDIR /
# --- gr-leo --- cannot use git clone for gr-leo bc there is an edit that needs to be done upon
tracker.h adding using namespace libsgp4; after the includes
# this has already been edited in the tar.xz file
COPY ./gr-leo.tar.xz /
RUN tar -xf gr-leo.tar.xz

```



```
# RUN git clone https://gitlab.com/librespacefoundation/gr-leo
WORKDIR /gr-leo/
RUN mkdir build
WORKDIR /gr-leo/build/
RUN cmake -DCMAKE_INSTALL_PREFIX=/usr/local .. && make && make install && ldconfig
WORKDIR /gr-leo/grc/
RUN mkdir build
WORKDIR /gr-leo/grc/build/
RUN cmake -DCMAKE_INSTALL_PREFIX=/usr/local .. && make && make install && ldconfig
WORKDIR /

# gr-satlab
COPY ./gr-satlab-v20230207.tar.xz /
RUN tar -xf gr-satlab-v20230207.tar.xz
WORKDIR /gr-satlab-v20230207/
RUN mkdir build
WORKDIR /gr-satlab-v20230207/build/
RUN cmake .. && make && make install && ldconfig
WORKDIR /gr-satlab-v20230207/grc/
RUN mkdir build
WORKDIR /gr-satlab-v20230207/grc/build/
RUN cmake .. && make && make install && ldconfig
WORKDIR /

# gr-satmisc
COPY ./gr-satmisc.tar.xz /
RUN tar -xf gr-satmisc.tar.xz
WORKDIR /gr-satmisc
RUN mkdir build
WORKDIR /gr-satmisc/build/
RUN cmake .. && make && make install && ldconfig
WORKDIR /

# LimeSuite
RUN apt-get update && apt-get install -y limesuite liblimesuite-dev limesuite-udev limesuite-images
# gr-limesdr daniestevez
RUN git clone https://github.com/chrisjohgorman/gr-limesdr.git
WORKDIR /gr-limesdr
RUN mkdir build
WORKDIR /gr-limesdr/build
RUN cmake .. && make && make install && ldconfig
WORKDIR /

# install Visual Code for embedded python blocks editing
```



```
RUN apt update && apt install software-properties-common apt-transport-https wget -y
RUN wget -q https://packages.microsoft.com/keys/microsoft.asc -O- | apt-key add -
RUN add-apt-repository "deb [arch=amd64] https://packages.microsoft.com/repos/vscode stable main"
RUN apt install -y code
# Install gr-satnogs
RUN apt update && apt install -y \
    libboost-dev \
    libitpp-dev \
    libboost-date-time-dev \
    libboost-filesystem-dev \
    libboost-program-options-dev \
    libboost-system-dev \
    libboost-thread-dev \
    libboost-regex-dev \
    libboost-test-dev \
    swig \
    cmake \
    build-essential \
    pkg-config \
    gnuradio-dev \
    libconfig++-dev \
    libgmp-dev \
    liborc-0.4-0 \
    liborc-0.4-dev \
    liborc-0.4-dev-bin \
    nlohmann-json3-dev \
    libpng++-dev \
    libvorbis-dev \
    git \
    libhamlib-dev \
    libhamlib++-dev \
    libgsl-dev
RUN git clone https://gitlab.com/librespacefoundation/satnogs/gr-satnogs.git
WORKDIR /gr-satnogs
RUN mkdir build
WORKDIR /gr-satnogs/build
RUN cmake .. && make -j $(nproc --all) && make install && ldconfig
USER gnuradio
RUN echo gnuradio | sudo -S mkdir /home/gnuradio/hier_blocks
RUN echo gnuradio | sudo -S ldconfig
```



```

WORKDIR /home/gnuradio
ENV PYTHONPATH "${PYTHONPATH}:/usr/local/lib/python3/dist-packages"
CMD bash

```

El script utilizado para construir el contenedor es el siguiente.

```

#!/usr/bin/env bash
docker build -t ubuntu:gnuradio-releases-3.10 .

```

El script utilizado para ejecutar el contenedor es el siguiente.

```

#!/usr/bin/env bash
docker run --net=host \
    --env="DISPLAY" \
    --volume="$HOME/.Xauthority:/root/.Xauthority:rw" \
    --privileged -e DISPLAY=$DISPLAY \
    -v ./persistent-folder:/home/gnuradio/persistent-folder \
    -v /dev/bus/usb:/dev/bus/usb \
    -v /tmp/.X11-unix:/tmp/.X11-unix \
    -v /dev/bus/usb:/dev/bus/usb \
    --rm -ti ubuntu:gnuradio-releases-3.10 \
    Bash

```

Por otro lado, el Dockerfile utilizado para la implementación de la librería LibCSP es el que se muestra a continuación.

```

FROM ubuntu:20.04
LABEL maintainer="Guillermo Lena - guillermo.lena@alen.space"
ENV DEBIAN_FRONTEND=noninteractive
ENV LC_ALL=C.UTF-8
ENV LANG=C.UTF-8
RUN apt-get update && apt-get install -y \
    xterm \
    cmake \
    software-properties-common \
    git \
    libusb-1.0-0-dev \
    liborc-dev \
    libzmq5 \
    libzmq3-dev \
    python3 \
    vim \

```



```

nano\
gcc-10 g++-10 \
tmux \
pkg-config \
libsocketcan-dev \
python-is-python3 \
libpython3-dev
RUN apt-get install -y iproute2
# create user libcsp with sudo (and password libcsp)
RUN apt-get install -y sudo
RUN useradd --create-home --shell /bin/bash -G sudo libcsp
RUN echo 'libcsp:libcsp' | chpasswd
WORKDIR /
RUN git clone https://github.com/libcsp/libcsp.git && cd libcsp && git checkout v1.6
WORKDIR /libcsp
RUN python3 waf configure build --enable-python3-bindings --enable-can-socketcan \
    --enable-if-zmqhub --with-driver-usart=linux --enable-examples --enable-shlib \
    --with-os=posix
RUN mkdir tests && chown libcsp tests
RUN mkdir PER && chown libcsp PER
USER libcsp
WORKDIR /
CMD bash

```

El script utilizado para construir el contenedor es el siguiente.

```

#!/usr/bin/env bash
docker build -t ubuntu:libcsp .

```

Por último, el script utilizado para ejecutar el contenedor es el siguiente.

```

#!/usr/bin/env bash
docker run --net=host \
    --env="DISPLAY" \
    --volume="$HOME/.Xauthority:/root/.Xauthority:rw" \
    --privileged -e DISPLAY=$DISPLAY \
    -v /tmp/.X11-unix:/tmp/.X11-unix \
    -v /dev/bus/usb:/dev/bus/usb \
    -v $PWD/tests:/libcsp/tests \
    -v $PWD/PER:/libcsp/PER \
    --rm -ti ubuntu:libcsp \
    bash

```



ANEXO C. NODO TX/RX PYTHON

A continuación, se muestra el script de Python desarrollado para actuar como nodo receptor implementando la librería LibCSP.

```
import sys
import signal
import zlib
import threading
import libcsp_py3 as libcsp

def csp_server():
    """
    Function that executes main CSP server task
    """
    print("\n*****[Starting RX]*****\n")

    # Initialize variables
    global wrong_packets
    global pkts_sent
    global rx_counter
    global expected_rx_sequence_number
    global missed_pkts

    pkts_sent = int(sys.argv[1])
    rx_counter = 0
    expected_rx_sequence_number = 1
    wrong_packets = 0
    missed_pkts = 0 # Initialize and create libcsp socket
    sock = libcsp.socket() # Bind libcsp socket to any port (will listens in every port)
    libcsp.bind(sock, libcsp.CSP_ANY) # Set socket to listen with a max backlog queue incoming
of 5 connections
    libcsp.listen(sock, 5)

    while True:
        # Wait for incoming connection
        conn = libcsp.accept(sock, libcsp.CSP_MAX_TIMEOUT)

        if not conn:
            # will skip this while iteration if conn == false so it stays waiting for a connection
            continue

        # Print debugging for the made connection (comment for better terminal performance)
        # print("\nConnection: source=%i:%i, dest=%i:%i" % ( libcsp.conn_src(conn),
        #                                             libcsp.conn_sport(conn),
        #                                             libcsp.conn_dst(conn),
        #                                             libcsp.conn_dport(conn)))

        # Read packet on the connection with a timeout of 1000
        packet = libcsp.read(conn, 1000)

        # Check if packet has any content, else skip loop iteration
        if packet is None:
            continue # At this point a valid packet has been received
        rx_counter += 1 # Get length and data from received packet
        length = libcsp.packet_get_length(packet)
        data = libcsp.packet_get_data(packet)

        # Print received packet debugging (comment for better terminal performance)
        # print("Expected packet nº ", expected_rx_sequence_number)
        # print("Rx packet nº "+str(rx_counter)+", len="+str(length)+"\nRx
Data="+''.join('{:02x}'.format(x) for x in data))
```



```

# -- PER Algorithm --
# Checks content as an expected integer representing sequence number of sent packet

# Satlab: CRC + RS + CC
rx_data = data[0:71]
rx_data_num = data[0:3]
rx_crc_data = data[71:75]

# # Satlab: CRC + RS
# rx_data = data[0:196]
# rx_data_num = data[0:3]
# rx_crc_data = data[196:200]

# # Satlab: -
# rx_data = data[0:232]
# rx_data_num = data[0:3]
# rx_crc_data = data[232:236]

# # GMSK
# rx_data = data[0:180]
# rx_data_num = data[0:3]
# rx_crc_data = data[180:184]

rx_pkt_number = int.from_bytes(rx_data_num, byteorder='big')
rx_crc = int.from_bytes(rx_crc_data, byteorder='big')
# print("\nOnly Data="+''.join('{:02x}'.format(x) for x in rx_data))
# print("\nCRC Data="+''.join('{:02x}'.format(x) for x in rx_crc_data))

# first of all check CRC32
if zlib.crc32(rx_data) == rx_crc:
    # Compare received integer with expected sequence number
    # if received sequence number equal to expected sequence number
    if rx_pkt_number == expected_rx_sequence_number:
        # increment next expected sequence number
        print("Correct packet!")
        expected_rx_sequence_number += 1          # else if received sequence number is
greater than the expected sequence number
    elif rx_pkt_number > expected_rx_sequence_number:
        # calculate the increment difference and update the wrong packets count with that
number
        print("Incorrect packet!")
        missed_pkts += (rx_pkt_number - expected_rx_sequence_number)
        expected_rx_sequence_number = rx_pkt_number + 1
    # else (received sequence number is smaller than the expected sequence number)
    else:
        pass
else:
    print("Incorrect packet!")
    wrong_packets += 1
    expected_rx_sequence_number += 1          # Free buffer memory
    libcsp.buffer_free(packet)

def signal_handler(sig, frame):
    PER_results()

def PER_results():
    global wrong_packets
    global pkts_sent
    global rx_counter
    global expected_rx_sequence_number
    global missed_pkts

    missed_last_pkts = (pkts_sent - expected_rx_sequence_number) + 1

```




```

# +1 because that following expected pkt never arrived
missed_pkts += missed_last_pkts

PER = ((missed_pkts+wrong_packets)/pkts_sent)*100

print("\n\n*****")
print("*****[PER Results]*****")
print("*****")
print("\n\t- Total sent packets: "+str(pkts_sent))
print("\t- Total received packets: "+str(rx_counter))
print("\t- Total missed packets: "+str(missed_pkts))
print("\t- Total wrong packets: "+str(wrong_packets))
print("\t- PER: "+str(PER)+" %")    print("\n\n ... Exiting now ... \n\n")
exit(0)signal.signal(signal.SIGINT, signal_handler)

if __name__ == "__main__":
    # Init libcsp node (address, hostname, model, revision, buffers, buffer_data_size)
    libcsp.init(1, "test_service", "bindings", "1.2.3", 10, 500)    # Init ZMQ interface (ZMQHUB)
    libcsp.zmqhub_init(1, "localhost") # own address for zmq

    # Set libcsp routing table
    """
    Syntax for routing table:
    - dest_addr/mask(5=5 bits) ZMQHUB next_hop_addr, 0/0 ZMQHUB
    """
    libcsp.rtable_load("0/0 ZMQHUB")    # Start libcsp router task
    libcsp.route_start_task()    # Print debug info, hostname, model, revision and finally routes
table
    print("\nHostname: %s" % libcsp.get_hostname())
    print("Model: %s" % libcsp.get_model())
    print("Revision: %s" % libcsp.get_revision())    # Print routing table
    print("\nRoutes:")
    libcsp.print_routes()    # start CSP server main function as a thread
    t = threading.Thread(target=csp_server)
    t.start()

```

Por otro lado, se muestra el script de Python desarrollado para actuar como nodo transmisor implementando la librería LibCSP.

```

"""
This script acts as a transmitting node on a CSP network.

It first initializes all the required libcsp parameters and then sends N packets with a time step
of 1 second.
"""

import sys
import time
import zlib
import libcsp_py3 as libcsp

if __name__ == "__main__":
    # Init libcsp variables

    priority = 1

    src_address = 30

    dst_address = 1

    dst_port = 20

```



```

src_port = 20
N_pkts = int(sys.argv[1])
pkt_sent = 1
content = 170
# padding = 293* content.to_bytes(1, byteorder='big') # GMSK modem
# padding = 206* content.to_bytes(1, byteorder='big') # satlab modem
padding = (75-4-3) *content.to_bytes(1, byteorder='big') # Satlab: CRC + RS + CC
# padding = (200-4-3) *content.to_bytes(1, byteorder='big') # Satlab: CRC + RS
# padding = (236-4-3) *content.to_bytes(1, byteorder='big') # Satlab: -
# padding = (184-4-3) *content.to_bytes(1, byteorder='big') # GMSK
# Init libcsp node (address, hostname, model, revision, buffers, buffer_data_size)
libcsp.init(src_address, "host", "model", "1.2.3", 10, 500)
# Init ZMQ interface (ZMQHUB)
libcsp.zmqhub_init(src_address, "localhost") # own address for zmq
# Set libcsp routing table
"""
Syntax for routing table:
    - dest_addr/mask(5=5 bits) ZMQHUB next_hop_addr, 0/0 ZMQHUB
"""
libcsp.rtable_load("1/5 ZMQHUB 6, 0/0 ZMQHUB")
# Start libcsp router task
libcsp.route_start_task()
time.sleep(0.2) # allow router task startup
# Print routing table
print("\nRoutes:")
libcsp.print_routes()
# Print some debugging
print("\nNumber of packets to be transmitted: ", N_pkts)
print("\n*****[Starting TX]*****\n")
# Start PER packet transmission loop
while pkt_sent<=N_pkts:
    # Set packet data
    pkt_data = pkt_sent.to_bytes(3, byteorder='big') + padding
    crc = zlib.crc32(pkt_data).to_bytes(4, byteorder='big')
    pkt_data_crc = pkt_data + crc
    # Make libcsp connection
    conn = libcsp.connect(priority, dst_address, dst_port, 1000, libcsp.CSP_0_NONE)
    # (prio, dest, dport, timeout, opts)
    # Get buffer memory ready with packet data size
    pkt = libcsp.buffer_get(len(pkt_data_crc))

```



```
# Print packet to be transmitted
print("\nTx packet nº "+str(pkt_sent)+", len="+str(len(pkt_data_crc))+
"\nData="+''.join('{:02x}'.format(x) for x in pkt_data_crc))

# Set packet data into buffer
libcsp.packet_set_data(pkt, bytearray(pkt_data_crc))

# Send data in buffer through the libcsp connection
libcsp.send(conn, pkt, 1000) # (conn_capsule, packet_capsule, timeout)

# Free buffer memory
libcsp.buffer_free(pkt)

# Update number of packet sent
pkt_sent+=1

# Provide some processing time to avoid CSP net congestion
time.sleep(0.3)

print("\n\n*****")
print("\n      PER transmission finished !\n")
print("*****")
time.sleep(3)
```