Universidad de Valladolid

Escuela Técnica Superior de Ingenieros de Telecomunicación

TRABAJO FIN DE GRADO

Grado en Ingeniería de Tecnologías de Telecomunicación

Adaptación de la estación terrena GS-UVa para su integración en red de seguimiento de satélites

Autor:

D. Samuel Adrián Peña

Tutor:

Dr. Ramón de la Rosa Steinz

Valladolid, 2 de julio de 2018

DESCRIPCIÓN DEL TRABAJO FIN DE GRADO

TITULO: Adaptación de la estación terrena GS-UVa para su integración en red de seguimiento de satélites

AUTOR: D. Samuel Adrián Peña

TUTOR: Dr. Ramón de la Rosa Steinz

DEPARTAMENTO: Teoría de la Señal y Comunicaciones e Ingeniería Telemática

MIEMBROS DEL TRIBUNAL

PRESIDENTE: Dr. Alonso Alonso Alonso

VOCAL: Dr. Juan Pablo de Castro Fernández

SECRETARIO: Dr. Ramón de la Rosa Steinz

SUPLENTE: Dr. Javier Aguiar Pérez

SUPLENTE: Dr. Jaime Gómez Gil

FECHA: 2 de Julio de 2018

CALIFICACIÓN:

Resumen

Este proyecto trata del desarrollo de una aplicación cliente, para la conexión de la estación terrena GS-UVa a la red de seguimiento de satélites SatNOGS. Además del desarrollo de la aplicación cliente, también se ha desarrollado un prototipo para el control de la polarización, de las antenas de VHF y UHF. La aplicación desarrollada, es una aplicación gráfica, que permite conectarse a la red SatNOGS, obteniendo las planificaciones de seguimientos y transmisión de datos recopilados, durante el seguimiento del satélite, a la red. Además de la conexión con la red, permite visualizar, de forma fácil, la información del satélite en seguimiento, los pases planificados, así como el estado de la radio y rotores. El desarrollo del cliente de la red, permite la interactuación de la estación terrena, no basada en dispositivos SDR, con la red SatNOGS concebida para el uso con dispositivos SDR.

Palabras clave: satélite, radio, seguimiento, datos, estación terrena, investigación, redes.

Abstract

This Project consist in developing a client application to connect the GS-UVa ground station to the SatNOGS satellite tracking network. In addition to the development of the client application, this project has developed a prototype to manage the polarization control, both for the VHF and the UHF antennas. The developed Graphic User Interface allows the connection to the SatNOGS network, retrieving the observation tasks and posting the acquired data to the network. This graphic interface allows to easily visualize the tracking satellite information, the scheduled observation tasks, and the radio and rotator status. The developed application allows the connection of a broad range of radio models to the SatNOGS network, not only SDR-based models, as initially conceived in the SatNOGS project.

Keywords: satellite, radio, tracking, data, ground station, investigation, network.

A mis padres y mi hermano, por el apoyo y ayuda que me han dado en todo momento, así como todos los recursos que me han facilitado para hacer todo más fácil.

Al resto de mi familia, por apoyarme con esta titulación y animarme a su realización, así como en el resto de mis proyectos personales.

A Ramón, por ser el tutor de este proyecto, al cual doy las gracias y agradezco que sea el impulsor y coordinador de la estación terrena, la cual ha puesto a disposición de los alumnos desde el mismo momento del comienzo del montaje.

A aquellos compañeros de titulación con los que he compartido muy buenos momentos y hemos realizado trabajos y cosas juntos.

A todos aquellos que durante este tiempo me han ayudado y apoyado en mis propósitos.

Índice de contenidos

Índice de figuras

1 Introducción

1.1 Introducción

El afán investigador e inquietudes de estudio del espacio, hace que los organismos y entidades científicas desarrollen satélites, con el objetivo principal de recolección de datos del espacio exterior, para el uso en labores de investigación. En el espacio, se han puesto en órbita multitud de satélites;, estos satélites son catalogados por su órbita descrita, fabricación y forma geométrica. Uno de los tipos de satélites utilizados, con fines de investigación, son los denominados CubeSats [1]. Este tipo de satélite está caracterizado principalmente por su formato de fabricación. Siguen una geometría en forma de cubo, de ahí el nombre que les denomina. La propuesta del formato de este tipo de satélites fue propuesta en en el año 1999 por los profesores Dr. Jordi Puig-Suari, de la *California Polytechnic State University*, y Dr. Bob Twiggs de la *Standford University's Space System Development Lab* [1].

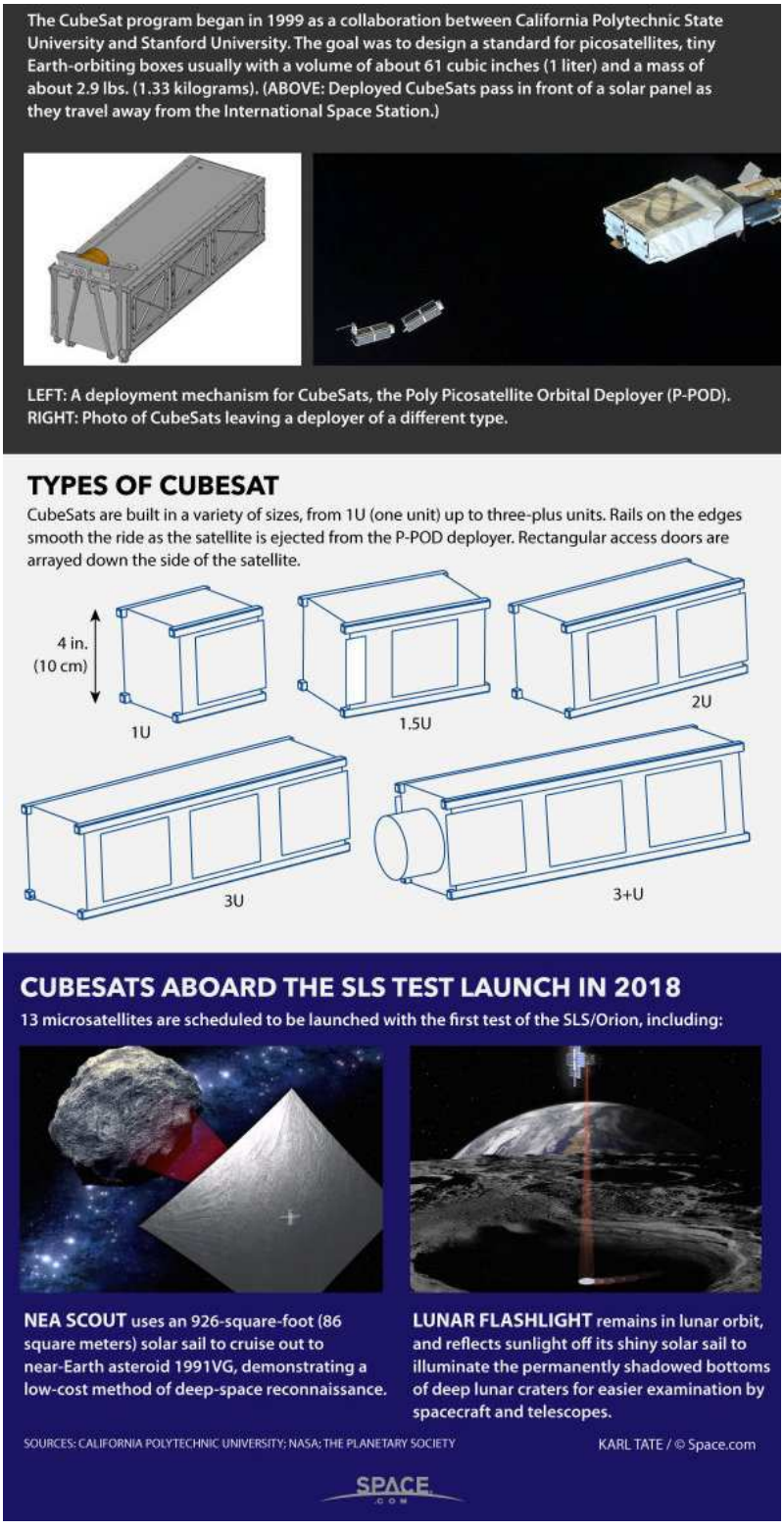


Figura 1. Tipos de *CubeSat* [1]

Otra característica, que determina el tipo de satélite, es el tipo de órbita descrita. Existen varios tipos de órbita, pero la más interesante para este tipo de elementos espaciales es la LEO (*Low Earth Orbit*), de tipo no geoestacionario. El hecho de no ser geoestacionario, implica que sea necesario el despliegue de multiples estaciones de seguimiento, repartidas por la superficie de la tierra, con el objetivo de cubrir todo el espacio, y recabar todos los datos enviados por los satélites.

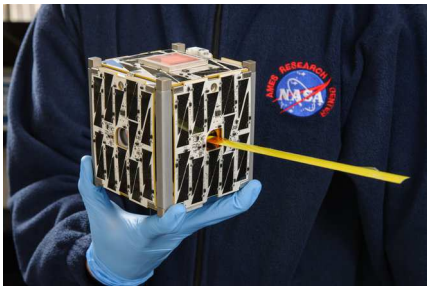


Figura 2. *cubesat* terminado

El despliegue de estaciones terrenas para el seguimiento de satélites es muy costoso si lo hace cada una de las instituciones propietarias de los satélites. Haría falta el alquiler o compra del emplazamiento, donde se instalarían los equipos para el seguimiento, además del gasto de compra de todos los equipos necesarios para cada localización. Esto es una solución muy costosa, y poco viable, a la cual hay que sumar el coste de fabricación y lanzamiento del satélite.

Este problema se puede solventar mediante el uso de estaciones terrenas ya implantadas de agencias espaciales, universidades, aficionados... Los elementos necesarios para llevar a cabo esta solución pasan por conectar las estaciones terrenas a una red de seguimiento. Para ello es necesario conectar los elementos de la estación con un PC que reciba las órdenes y parámetros de seguimiento, y una vez realizada la observación, transmita los datos recopilados al servidor para su utilización por el resto de usuarios. Con esta solución los costes disminuyen practicamente a la construcción y lanzamiento del satélite, y a la posible implantación de una única estación terrena de seguimiento y control.

El cliente de control de las estaciones debe calcular la posición del satélite en el espacio, además de calcular el azimuth y elevación, donde tienen que estar dirigidas las antenas de seguimiento, según la posición donde tenga la ubicación la estación. Para el cálculo de las posiciones, velocidad, altitud y resto de parámetros de los satélites, se utilizan los denominados TLE "Two-Line Elements", proporcionados por NORAD *North American Aerospace Defense Command*, consistente en las perturbaciones generales de los satélites [2]. Otro parámetro necesario, para la observación de satélites, es la frecuencia de los enlaces de bajada; la red debe proporcionar este dato, junto con el tipo de modulación, para la correcta demodulación y decodificación. La frecuencia de seguimiento dada por la red, es únicamente la frecuencia de referencia; a esta hay que aplicarle el efecto Doppler de cuerpos en movimiento. El efecto Doppler es la "variación de la frecuencia en función de la velocidad de movimiento de la fuente y/o receptor" [3].

Figura 3. Seguimiento del primer lanzamiento de lanzador Vega VV01, con el satélite Xatcobeo a bordo, en febrero de 2012, desde el control de la estación terrena

La Universidad de Valladolid, en concreto, en la Escuela Técnica Superior de Ingenieros de Telecomunicación, tiene instalada una estación terrena para el seguimiento y recepción de datos satelitales, de características detalladas anteriormente. El Proyecto de montaje de la estación terrena, de la Universidad de Valladolid (GS-UVa), comenzó en el año 2009, con la propuesta de formar parte de una novedosa red de seguimiento de satélites, respaldada por la Agencia Espacial Europea (ESA). A mediados del año 2011, tras todo el montaje, proyectos y permisos necesarios, el despliegue de la estación concluyó, comenzando a operar en la red *Global Education Network for Satellite Operations*, que en aquella época se encontraba en desarrollo [4].

La conexion y aportación de datos a la red GENSO fue positiva desde el primer momento. En los próxmos puntos, repasaremos las características, y situación se la red GENSO, así como de otras redes de seguimiento de satélites.

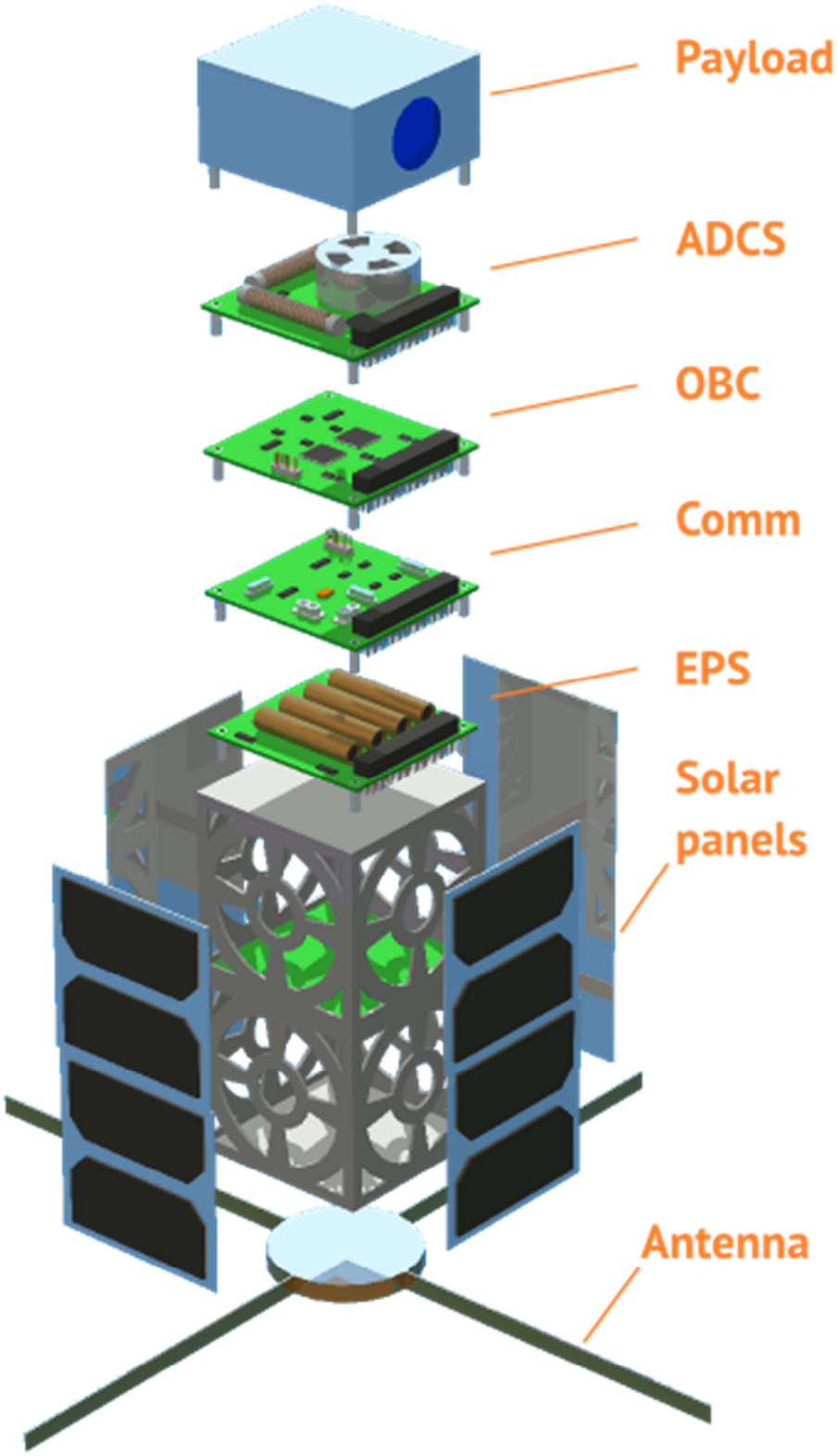


Figura 4. Despiece de un CubeSat [1]

1.2 Motivación

La motivación de este proyecto es la consecución de puesta al día de la estación GS-UVa, que en la actualidad se encuentra desactualizada y con un uso por debajo de sus posibilidades. La Universidad, como entidad de generación de conocimiento, necesita la recopilación de los datos obtenidos por los nanosatélites para la mejora de colaboración con otros centros de investigación. La consecución de una gran base de datos de información obtenida por las estaciones terrenas, de uso libre y abierto, pueda servir como base a nuevos proyectos investigadores o proyectos en curso, que puedan necesitar este tipo de información. Con frecuencia, se ponen en órbita *cubesats* que necesitan el soporte de este tipo de redes.

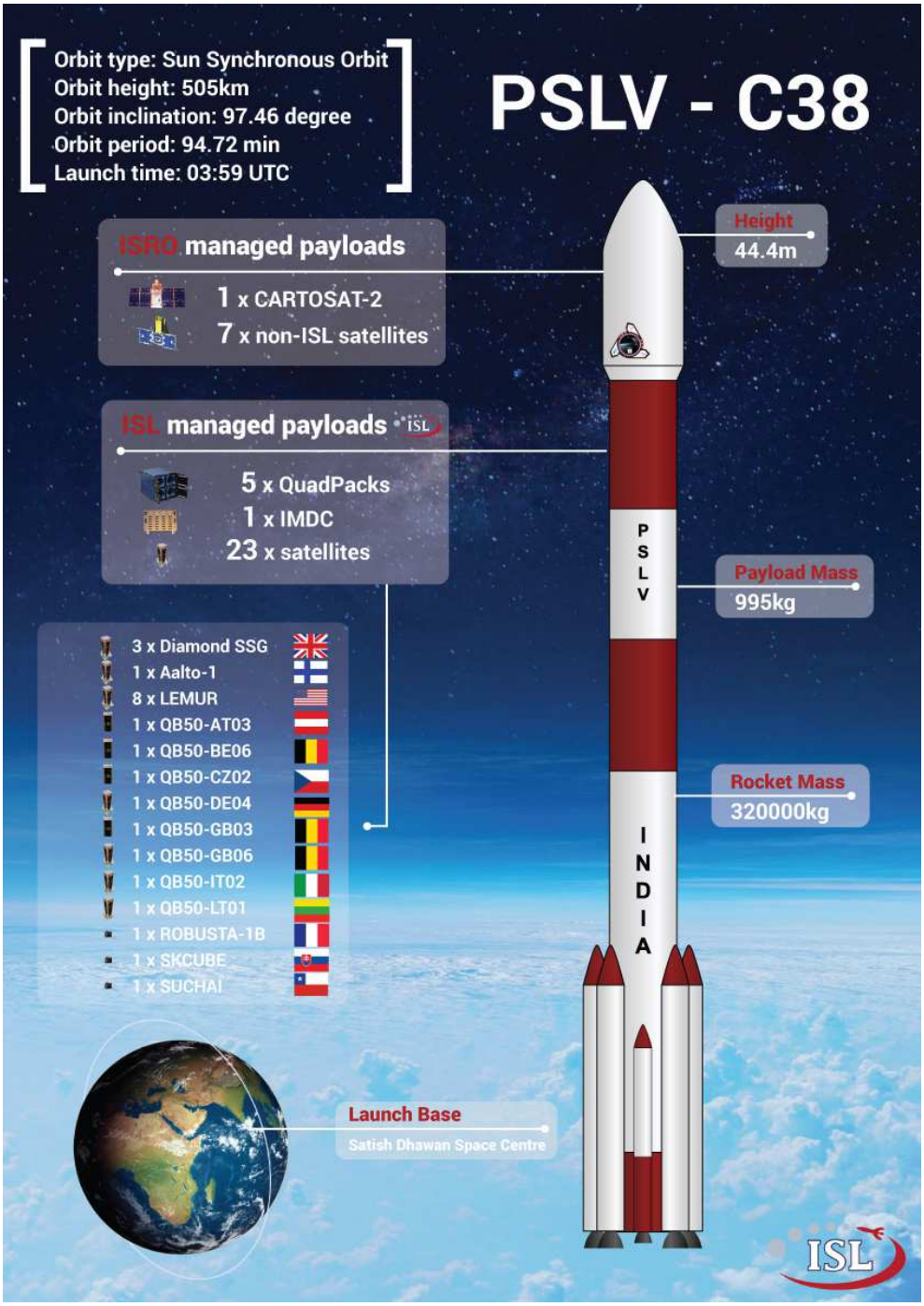


Figura 5. Misión PSLV-C38 con 23 CubeSats a bordo [5]

1.3 Objetivos

El objetivo buscado en este proyecto es la adaptación, puesta al día y conexión a una red de seguimiento de satélites abierta y libre. La red elegida es

SatNOGS, promovida y mantenida por la *Libre Space Foundation*, en conjunto con desarroladores voluntarios.

Los objetivos necesarios, para llevar a cabo el objetivo principal de conexión a la red, están supeditados al desarrollo de un software cliente específico, que realizará las siguientes actividades:

- **Solicitar y recibir la lista de tareas con los satélites a observar.** El software tiene que solicitar al servidor de la red la lista de satélites planificados, para su seguimiento por la estación. La planificación del seguimiento puede haber sido encargada por los propietarios de los satélites, o por cualquier usuario de la red.
- **Seguimiento de satélites.** El sistema desarrollado deberá controlar el hardware necesario para el correcto seguimiento del satélite. Principalmente, como elementos de hardware, se controlarán los rotores, encargados de dirigir las antenas hacia el objetivo en seguimiento, la radio, encargada de recibir los datos de radiofrecuencia, donde es necesario asignar y variar la frecuencia del enlace de bajada de los satélites en fuención del efecto Doppler [3].
- **Publicación de los datos obtenidos.** El cliente, una vez finalizado el seguimiento del satélite, transmitirá los datos recopilados por la estación a la red de seguimiento para la libre disposición de los usuarios.

2 Estado de la técnica

En el capítulo anterior se estableció como objetivo el desarrollo de un cliente para una red de seguimiento de satélites. En este capítulo, revisaremos el estado de las redes de seguimiento de satélites, para determiner la que mejor se adapte a nuestro proyecto y objetivos.

2.1 GENSO

La red *Global Education Network for Satellite Operations* fue una red promovida y mantenida principalmente por la Agencia Espacial Europea, en colaboración con la agencia estadounidense NASA, la agencia espacial japonesa JAXA y varias universidades a nivel mundial.

La Universidad de Valladolid, estuvo involucrada en el proyecto, realizando pruebas de software de seguimiento al igual que hicieron otras universidades y colaboradores. El proyecto fue desarrollado en el lenguaje de programación JAVA.

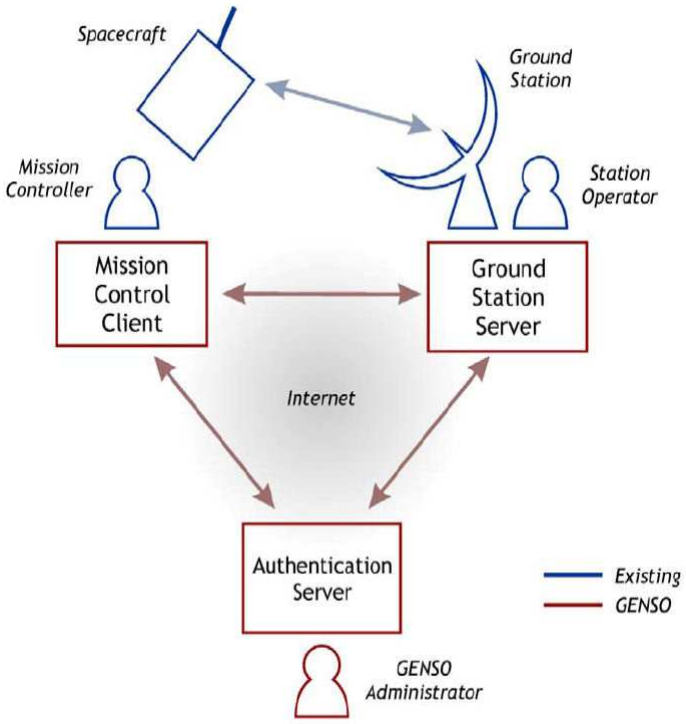


Figura 6. Elementos de la red GENSO [6]

El sistema GENSO se compone de tres elementos; podemos ver los elementos y cómo se relacionan en la Figura 6. Estos elementos son: un servidor de autenticación de usuarios (AUS), el cliente de control de misión (MCC) y el servidor de estación terrena (GSS) [6].

- **AUS.** El servidor de autenticación se encarga de validar a los usuarios en el sistema. Además, recibe las peticiones de seguimiento de satélites, encargadas por el MCC, y una vez hecho el seguimiento, transmite los datos recopilados al MCC que realizó la petición. Por otra parte, se encarga de asignar y planificar los seguimientos de satélites a los GSS, de los cuales recibirá los datos obtenidos durante los seguimientos. Este elemento de la red es gestionado por el administrador de GENSO [6].
- **MCC.** El cliente de control de misión permite realizar peticiones de seguimiento de satélites. La petición de seguimiento es enviada al AUS, el cual coordinará el seguimiento del satélite con los GSS. Recibe los datos obtenidos durante los seguimientos, correspondientes a las peticiones realizadas. [6].
- **GSS.** El servidor de estación terrena se encarga de recibir la planficación de seguimiento por parte del AUS. Controla los rotores, la radio y el TNC para la correcta adquisición de los datos durante el seguimiento. Una vez finalizado el pase del satélite en seguimiento, transfiere los datos recibidos al AUS [6].

El estado actual del proyecto GENSO es de descatalogado. Las últimas versiones publicadas del software fueron versiones alfa y beta, sin liberación del código fuente. El objetivo del proyecto, además de la consecución de una red de seguimiento de satélites, buscaba ser un proyecto libre y abierto. La planificación de liberación del código estaba prevista a partir de una versión estable, en funcionamiento y libre de errores [7].

2.2 RAISIN

Raisin es un proyecto de red de seguimiento de satélites, desarrollado y promovido por estudiantes de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid.

El proyecto fue iniciado en el año 2014 como proyecto fin de carrera, con el objetivo de continuar la labor de adquisición de datos por parte de GS-UVa, tras la descatalogación del proyecto GENSO. Tras el final del proyecto inicial, siguió en desarrollo sucesivo mediante becas de colaboración. El lenguaje de programación con el que se desarrolló el proyecto fue Python. Desde el comienzo, se penso en un proyecto abierto y libre.

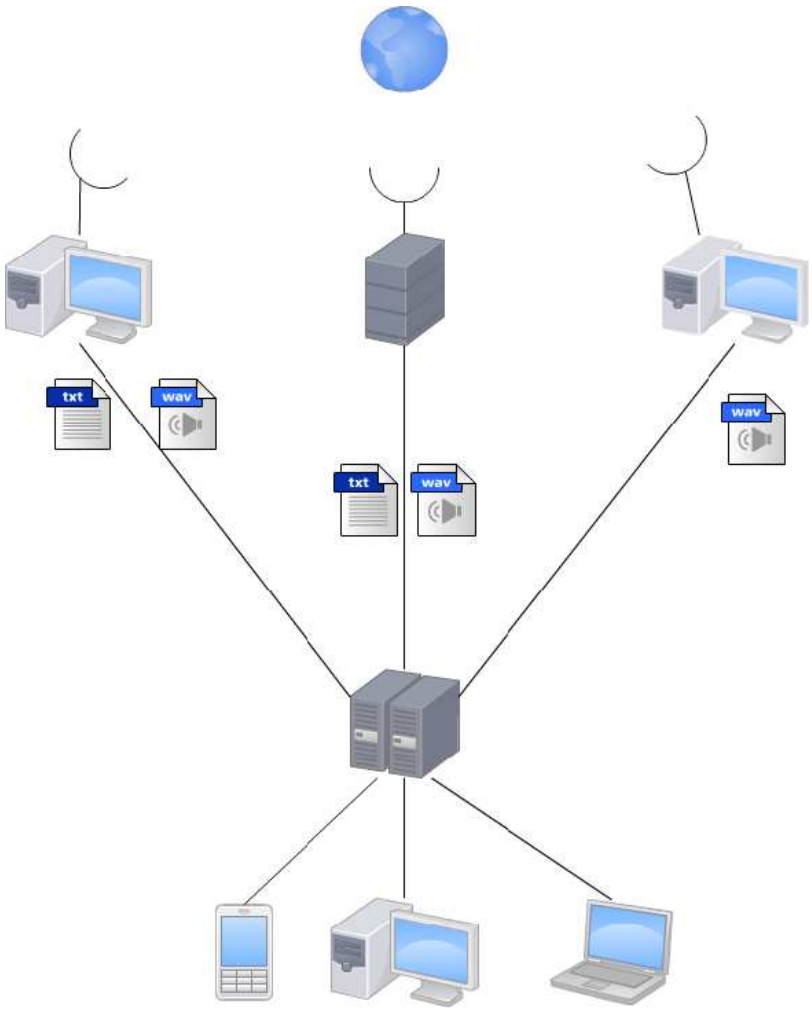


Figura 7. Esquema básico de red del proyecto Raisin [8]

En la Figura 7 podemos ver los elementos que componen la red de Raisin y que describimos a continuación [8]:

- **Servidor.** El servidor es el núcleo de la red de seguimiento Raisin. Este elemento de la red se encarga de realizar la planificación de observaciones, recopilación y confección de la base de datos de información. Desde el servidor se realizan las peticiones de seguimiento de satélites para cada una de las estaciones terrenas conectadas a la red. Dispone de un algoritmo de planificación para evitar solapamientos, en una misma estación, de los seguimientos, tomando como elementos de calidad el tiempo del pase y la elevación máxima, entre otros factores. Envía a las aplicaciones cliente los datos necesarios para la realización del seguimiento, como son los TLE, frecuencias, etc. Terminada la observación, recopila los datos enviados desde las estaciones terrenas para su publicación en la base de datos de recopilación de datos de observaciones.
- **Cliente.** El cliente, controla la estación terrena, recopila datos y los transfiere al servidor. Mediante los datos de seguimiento, recibidos por parte del servidor, controla los elementos hardware, como los rotores, radio y TNC. Además, graba el audio recibido de la radio, por si la TNC no es capaz de decodificar los datos, o existe la posibilidad de analizar con otras herramientas los datos recibidos. Una vez finalizado el pase de satélite, transmite los datos de audio y los datos decodificados por el TNC al servidor de Raisin para su procesamiento.

En la actualidad el proyecto ha sido descatalogado, en virtud de la utilización y contribución en otros proyectos en estado más avanzado y con un número de colaboradores y usuarios significativo. Este proyecto, a pesar de estar ideado como proyecto de código libre y abierto, nunca fue más allá del ambito de GS-UVa. Paralelamente a este proyecto, se desarrolló la red de seguimiento SatNOGS.

2.3 SatNOGS

La red SatNOGS es una red de seguimiento de satélites abierta y libre. Depende y es mantenida por la *Libre Space Foundation*, con la colaboración de la comunidad de desarrolladores voluntarios. El proyecto comenzó en abril de 2014, como parte de un proyecto realizado en el *Hackaton* (concurso de programación), *NASA Space App Challenge*, promovido por la NASA en Atenas, Grecia [9].

El objetivo del proyecto se centra en la observación y recepción de señales satelitales, en particular de satélites de baja órbita (LEO).

Las principales bandas de funcionamiento de radiofrecuencia son las bandas VHF y UHF.

Todos los elementos que componen el proyecto, software y hardware, buscan ser de código abierto, libre y de bajo coste para el usuario que quiera desplegar una estación de seguimiento. El proyecto ha sido desarrollodo con el lenguaje de programación *Python*, haciendo uso del soporte *Django* [10]. Todos los elementos del proyecto, como es el software, esquemas y resto de documentación necesaria, ha sido publicado en repositorios libres, para el uso y contribución al desarrollo por parte de los usuarios que lo deseen.

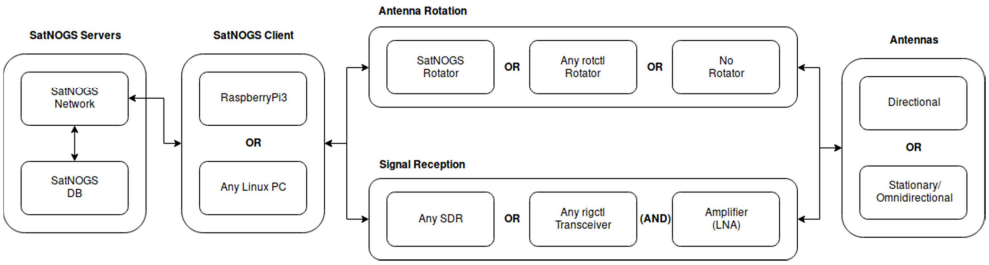


Figura 8. Esquema de elementos de SatNOGS [9]

El sistema se compone de cuatro elementos principales, tal y como podemos ver en la Figura 8: el servidor de la red, la base de datos de satélites, el cliente de control de las estaciones terrenas y el hardware [9].

- **SatNOGS Network.** Servidor de gestión y planificación de las estaciones terrenas y sus usuarios. Permite la programación de observaciones a satélites y la recopilación de los datos recibidos

durante el pase de los satélites. Proporciona la información necesaria a los clientes para el seguimiento de los satélites. Se comunica con la base de datos de SatNOGS para la obtención de los datos de los satélites, como son los TLE y los datos de los transpondedores. Tiene posibilidad de mostrar, de las observaciones, los datos decodificados, el audio grabado durante el pase, y el espectrograma obtenido si el seguimiento se ha realizado con un dispositivo SDR (*Software Defined Radio*). El sistema dispone de un mecanismo de calificación de las observaciones, normalmente asignado por el que solicita el seguimiento. Los calificativos disponibles, son:"bueno", "malo", "fallo". Con este sistema, los usuarios que consulten los datos pueden filtrar las observaciones en función del éxito. Proporciona una API REST para la conexión con el cliente y otras aplicaciones. Una API REST es una forma de intercambio de información entre dos aplicaciones mediante el protocolo HTTP, utilizado por páginas web.

- SatNOGS DB. Base de datos de satélites. Reúne información sobre los satélites, como el estado de operación, los transpondedores disponibles, el modo de operación y los TLE actualizados de NORAD. Estos datos son necesarios para el control de los rotores, de la radio y del TNC para la decodificación. Está enlazado con SatNOGS Network para la consulta de las observaciones disponibles obtenidas con estaciones terrenas conectadas a la red. Proporciona una API REST para la connexion con otras aplicaciones.
- SatNOGS Client. Cliente de control del hardware de las estaciones terranas. Se comunica con SatNOGS Network mediante API REST para la obtención de los trabajos de seguimiento y envío de los datos recopilados. El software proporcionado por el proyecto de la red está diseñado para controlar dispositivos SDR y cualquier tipo de rotor con soporte en HamLib. La red también es compatible con otros receptores comerciales no SDR para el control de la frecuencia de recepción, pero será necesario otro tipo de software que haga de cliente. En el caso de control de un SDR como receptor es necesario

diponer de las bibliotecas de GNURadio, el cliente útiliza esta biblioteca para el control y decodificación de los datos, mediante scripts de GNURadio, preparados para la realización de estos trabajos. La aplicación original, no contempla la conexión ni decodificación de datos a través de un TNC.

- **SatNOGS Ground Station.** Este elemento se compone del hardware de la estación terrena. Desarrollados y liberados mediante licencias de Hardware abierto que comprenden todos los elementos necesarios para el montaje de una estación, para el seguimiento, de bajo coste. De todos los elementos, que abarca este apartado, el SDR es el único elemento púramente comercial, ya que el resto de elementos, son materiales básicos que necesitan tranformación. Los componentes pueden ser sustituidos por componentes comerciales, como radio, rotores, controlador de rotores,...

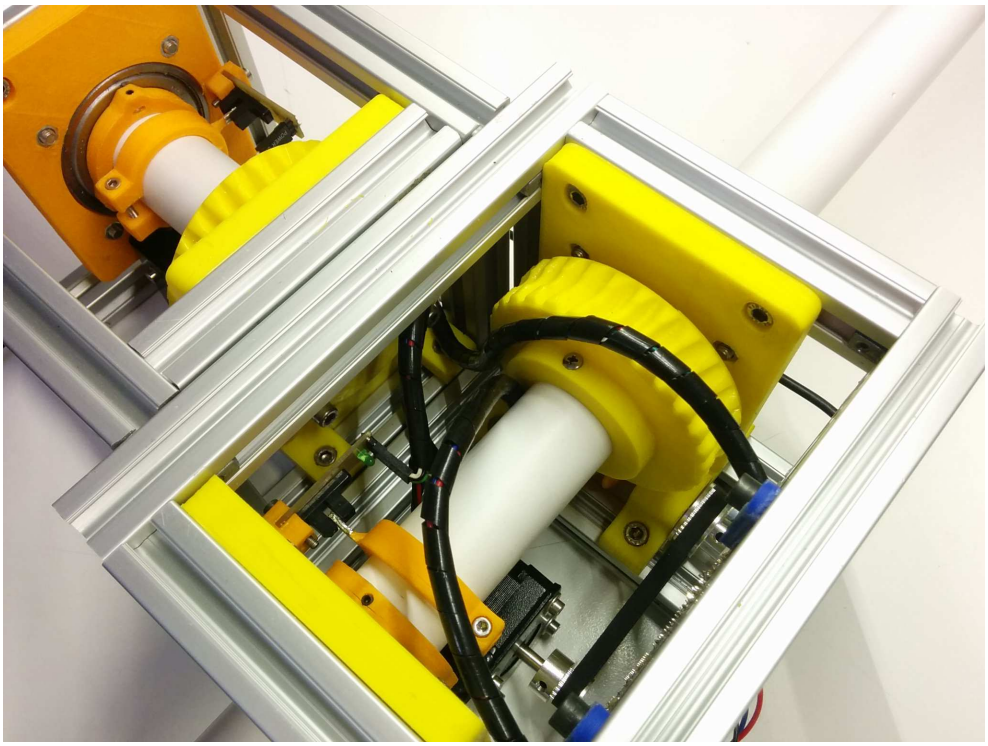


Figura 9. Rotor autofabricado del proyecto SatNOGS

En la actualidad este proyecto está bastante avanzado en despliegue y dispone de un amplio número de usuarios. Dispone de 136 estaciones registradas con recepción de datos en el momento de la escritura de este documento. Además no es obligatorio el uso de SatNOGS *client* pudiendo desarrollarse otros clientes, adaptados al hardware de cada estación participante.

El trabajo que se realizará será un cliente, para la conexion de GS-UVa a esta red de seguimiento de satélites. Se ha optado la adhesion a esta red por el estado en el que se encuentra, con un número significante de usuarios y estaciones en activo, además del constant desarrollo que se realiza en el software y hardware de la red de seguimiento.

Figura 10. Antena autoconstruida del proyecto SatNOGS

3 Métodos y materiales a utilizar

Para el desarrollo del proyecto serán necesarios varios elementos. Estos elementos están compuestos por parte hardware y software. Los elementos de hardaware que se decribirán a continuación, son los elementos que forman parte de la estación terrena, y la placa microcontroladora Arduino. Los elementos software que se describirán son, el lenguaje de programación a utilizar, bibliotecas de funciones, y resto de elementos necesarios para la conexion y puesta en marcha, de la estación con la red de seguimiento SatNOGS.

3.1 Hardware

3.1.1 Estación

La estación de seguimiento, de la Universidad de Valladolid, se compone de varios elementos, que se describen a continuación. Estos elementos son la radio, antenas, rotores, pre amplificadores, TNC, PC de control,...

En la Figura 11, podemos observar el rack de comunicaciones que alberga todos los equipos de la estación, y en la Figura 12, podemos observar el sistema radiante instalado en la azotea de la Escuela Técnica Superior de Ingenieros de Telecomunicación, en la Universidad de Valladolid.

Figura 11. Rack con los equipos de la estación terrena

Figura 12. Sistema radiante de la estación GS UVa

3.1.1.1. Radio

Figura 13. Transceptor ICOM IC-910

El dispositivo utilizado, en la estación, para la recepción y transmision de información, por radiofrecuencia, es un transceptor de radioaficionado. El transceptor es un ICOM, modelo IC-910, que opera en las bandas de frecuencia de 144 – 146 MHz (banda de 2 metros), de 432 – 440 MHz (banda de 70 centímetros) y 1200 MHz. El transceptor permite la operar en varios modos de funcionamiento, función repetidor, función *Split Frequency* (otro modo para repetidores), modo Satélite (permite introducir manualmente la variación de frecuencia por Doppler). También permite el control de *squelch* (silenciador), potencia de salida, ganancia del microfono, transmisión de datos a 9600 bps (por un puerto especial de datos).

Este dispositivo, se conecta a un PC, para su control en modo remoto o automático, a través de un puerto de serie. Para poder llevar a cabo esta conexión, es necesario el uso de un conversor de nivel. En GS-UVa, se utiliza el conversor ICOM CT-17, capaz de soportar hasta cuatro equipos del fabricante ICOM.

En la Figura 14, se puede observar una ilustración con la características principales del equipo [11].

•General

•Frequency coverage : (Unit: MHz)

Version	144 MHz	430(440) MHz	1200 MHz*1
U.S.A.	Tx: 144.0–148.0 Rx: 136.0–174.0*2	Tx: 430.0–450.0 Rx: 420.0–480.0*3	Tx: 1240.0–1300.0 Rx: 1240.0–1320.0*4
Europe	144.0–146.0	430.0–440.0	1240.0–1300.0
Australia	144.0–148.0	430.0–450.0	1240.0–1300.0
Sweden	144.0–146.0	432.0–438.0	1240.0–1300.0
Italy	144.0–146.0	430.0–434.0 435.0–438.0	1240.0–1245.0 1270.0–1298.0

*1 Optional UX-910
*2 Guaranteed range is 144.0–148.0 MHz.
*3 Guaranteed range is 430.0–450.0 MHz.
*4 Guaranteed range is 1240.0–1300.0 MHz.

•Mode : USB, LSB, CW, FM, FM-N*
*Not available in 1200 MHz
•No. of memory Ch. : 212 (99 regular, 6 scan edges, 1 calls for each band) plus 10 satellite memories)
•Antenna connector : SO-239 (50 Ω; VHF)
Type-N (50 Ω; UHF)
•Usable temp. range : –10˚C to +60˚C; +14˚F to +140˚F
•Frequency stability : Less than ±3 ppm (–10 to 60˚C; +14 to +140˚F)
•Frequency resolution : 1 Hz minimum
•Power supply : 13.8 V DC ±15% (negative ground)
•Current drain (at 13.8 V DC; approx.):
Transmit Max. power 23.0 A
Receive Standby 2.0 A (3.0 A; UX-910)
Max. audio 2.5 A (3.5 A; UX-910)
•Dimensions : 241(W)×94(H)×239(D) mm
(projections not included) 9½(W)×3 11/16(H)×9 13/32(D) in
•Weight (approx.) : 4.5 kg; 10 lb (5.35 kg; 11 lb 13 oz w/UX-910)
•ACC 1 connector : 8-pin DIN connector
•CI-V connector : 2-conductor 3.5 (d) mm (⅛")
•DATA connectors : 6-pin mini DIN × 2 (for MAIN and SUB)

•Transmitter

•Output power (continuously adjustable):
144 MHz 5–100 W
430(440) MHz 5–75 W
1200 MHz 1–10 W (optional UX-910)
•Modulation system :
SSB Balanced modulation
FM Variable reactance modulation
•Spurious emission :
144/430(440) MHz More than 60 dB
1200 MHz More than 50 dB
•Carrier suppression : More than 40 dB
•Unwanted sideband suppression : More than 40 dB
•Microphone connector : 8-pin connector (600 Ω)
•KEY connector : 3-conductor 3.5(d) mm (⅛")

•Receiver

•Receive system :
VHF SSB, CW Single conversion superheterodyne
FM Double conversion superheterodyne
UHF SSB, CW Double conversion superheterodyne
FM Triple conversion superheterodyne
•Intermediate frequencies: (Unit: MHz)

		MAIN BAND			SUB BAND		
		1st	2nd	3rd	1st	2nd	3rd
144 MHz	SSB	10.8500	—	—	10.9500	—	—
	CW	10.8491	—	—	10.9491	—	—
	FM	10.8500	0.455	—	10.9500	0.455	—
430(440) MHz	SSB	71.2500	10.8500	—	71.3500	10.9500	—
	CW	71.2491	10.8491	—	71.3491	10.9491	—
	FM	71.2500	10.8500	0.455	71.3500	10.9500	0.455
1200 MHz	SSB	243.8500	10.8500	—	243.9500	10.9500	—
	CW	243.8491	10.8491	—	243.9491	10.9491	—
	FM	243.8500	10.8500	0.455	243.9500	10.9500	0.455

•Sensitivity :
SSB, CW (10 dB S/N) Less than 0.11 µV
FM (12 dB SINAD) Less than 0.18 µV
•Squelch sensitivity (threshold):
SSB, CW Less than 1.0 µV
FM Less than 0.18 µV
•Selectivity :
SSB, CW More than 2.3 kHz/–6 dB
Less than 4.2 kHz/–60 dB*
FM More than 15.0 kHz/–6 dB
Less than 30.0 kHz/–60 dB*
FM-N More than 6.0 kHz/–6 dB
Less than 18.0 kHz/–36 dB
CW-N More than 0.5 kHz/–6 dB
(w/FL-132 or FL-133) Less than 1.34 kHz/–60 dB*
*Except 1200 MHz band
•Spurious and image rejection ratio:
144/430(440) MHz More than 60 dB
1200 MHz More than 50 dB
•AF output power (at 13.8 V DC) : More than 2.0 W at 10% distortion with an 8 Ω load
•RIT variable range :
144/430(440) MHz ±1.0 kHz (SSB, CW)
±5.0 kHz (FM)
1200 MHz ±2.0 kHz (SSB, CW)
±10.0 kHz (FM)
•IF SHIFT variable range : More than ±1.2 kHz
•PHONES connector : 3-conductor 6.35(d) mm (¼")
•Ext. SP connectors : 2-conductor 3.5 (d) mm (⅛") /8 Ω × 2 (for MAIN and SUB)

All stated specifications are typical and subject to change without notice or obligation.

Figura 14. Especificaciones generales ICOM IC-910

3.1.1.2. Antenas

Las antenas, son un elemento fundamental para la captación y emisión de ondas electromagnéticas. Las características de las antenas, hacen que la

elección, entre el tipo de antena no sea menospreciada, dependiendo del uso que se va a realizar con la antena. En una aplicación de seguimiento de satélites, es necesario la utlización de antenas, con alta ganacia, polaridad circular, y es deseable que sea lo más directiva posible.

En la estación de seguimiento de satélites GS-UVa, se encuentra instalada una agrupación de antenas, compuesta por dos antenas Yagi de polarización circular, para la banda de 2 metros, otra agrupación de dos antenas Yagi de polarización circular, para la banda de 70 cm, y una antena parabólica G3RUH para la banda de 2400 MHz.

3.1.1.2.1. Antena M2 2MCP14

El modelo de antena 2MCP14, es una antena diseñada para el trabajo en el rango de frecuencia de la banda de 2 metros, comprendido en el espectro entre 143 y 148 MHz. La ganancia que posee esta antena, es de 12,34 dBic, y permite una potencia máxima de 1,5 kW. Está compuesta por dos dipolos plegados, con dos reflectores y diez elementos directores, distribuido la mitad en polarización vertical y la otra mitad en polarización horizontal, permitiendo una polarización resultante de tipo circular [12].

Este modelo de antena, en origen, la polarización circular viene predefinida, no pudiendo cambiar el sentido de la polarización. El fabricante comercializa la actualización *"Polarization Switch"*, consistente en cambiar unos pocos componentes de la antena original, dotándola de control del sentido de la polarización. El control de la polarización se realiza mediante alimentación eléctrica, con una tensión de 12 Vcc, la polarización es conmutada. En la Figura 15 y Figura 16, podemos observar una ilustración de la antena utilizada, y el diagrama de radiación.

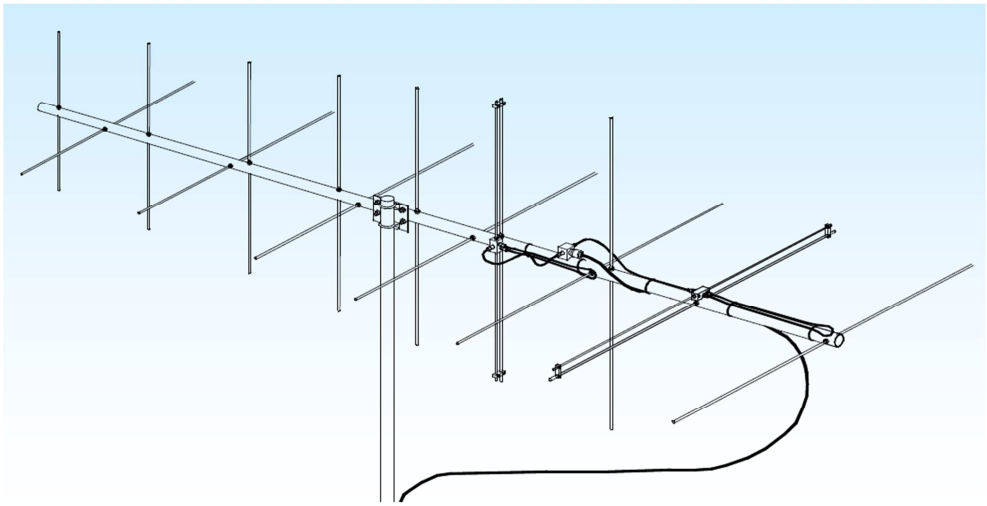


Figura 15. Antena 2MCP14

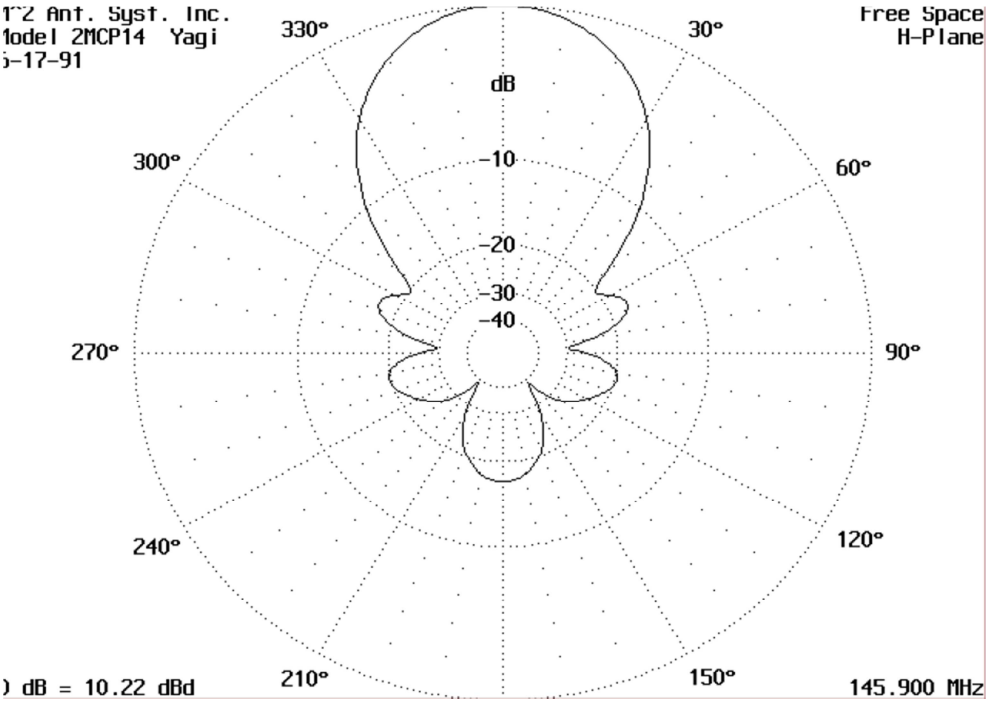


Figura 16. Diagrama de radiación de la antena 2MCP14

3.1.1.2.2. Antena 436CP30

El modelo de antena 436CP30, es una antena diseñada para el trabajo en el rango de frecuencia de la banda de 70 centímetros, comprendido en el espectro

entre 432 y 440 MHz. La ganancia que posee esta antena, es de 15,50 dBic, y permite una potencia máxima de 600 W. Está compuesta por dos dipolos plegados, con dos reflectores y doce elementos directores, distribuido la mitad en polarización vertical y la otra mitad en polarización horizontal, permitiendo una polarización resultante de tipo circular [13].

Este modelo de antena, en origen, la polarización circular viene predefinida, no pudiendo cambiar el sentido de la polarización. El fabricante comercializa la actualización *"Polarization Switch"*, consistente en cambiar unos pocos componentes de la antena original, dotándola de control del sentido de la polarización. El control de la polarización se realiza mediante alimentación eléctrica, con una tensión de 12 Vcc, la polarización es conmutada. En la Figura 17, podemos observar una ilustración de la antena utilizada.

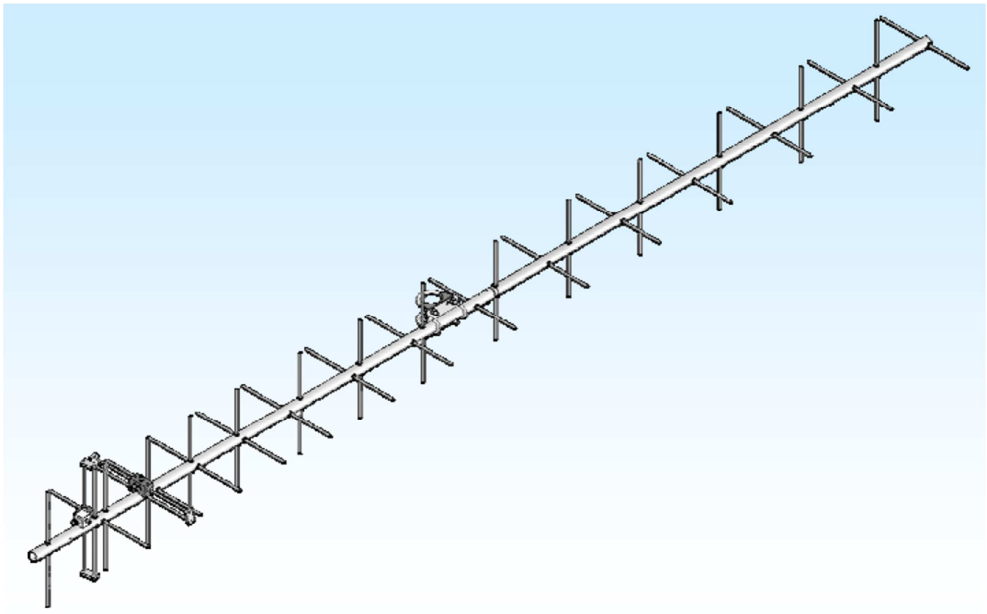


Figura 17. Antena 436CP30

3.1.1.3. Rotores

El sistema radiantes utilizado en la estación, es un sistema muy directivo y una ganacia aceptable, ideal para la recepción de las débiles señales de los satélites. Para poder realizar el seguimiento, y captar las señales, es necesario

orientar las antenas hacia la posición del satélite en seguimiento. La manera de conseguir la orientación necesaria, es mediante un sistema de dos rotores, que controlen la posición de azimuth y la posición de elevación. En GS-UVa, se ha utilizado, los siguientes componentes:

- Rotor de azimuth OR2800PX (Figura 18).
- Rotor de elevación MT1000 (Figura 19).

Figura 18. Rotor OR2800PX

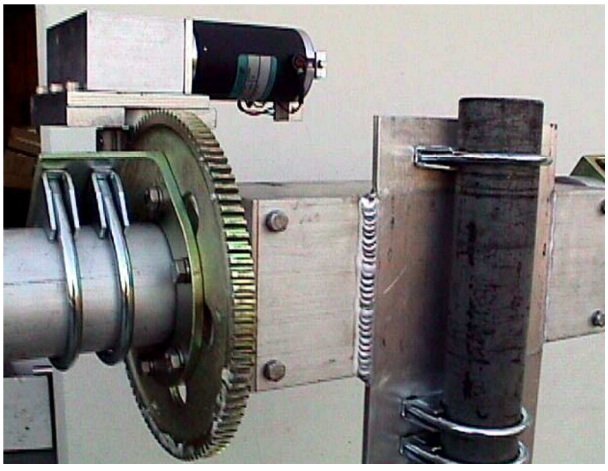


Figura 19. Rotor MT1000

3.1.1.4. Controlador de rotores RC2800PRK

El control de la posición de los rotores, se realiza con el controlador RC2800PRK de M2. Este controlador, permite la operación en modo manual, o en modo remoto o automático, mediante conexión a PC vía puerto de serie RS-232. Dispone de varias velocidades de movimiento y configuración. El modo de calibración, es muy útil, para los correctos ajustes de posicionamiento de los rotores. Es recomendable hacer calibraciones cada cierto tiempo, el sistema se va desajustando con el paso de tiempo, debido al peso, viento y resto de factores que puedan llevar al desajuste. En la Figura 20, podemos ver una ilustración del modelo utilizado en la estación terrena GS-UVa.

Figura 20. Controlador de rotores RC2800PRKX

3.1.1.5. Pre amplificadores

Las señales captadas por la antenas, son muy débiles, debido a la baja potencia de transmisión y la lejanía del satélite a miles de kilómetros. La débil señal captada, y la tirada de la línea de transmisión hasta la radio, hace que la señal aun se atenúe en mayor medida, pudiendo ocasionar la total pérdida de señal. Para solventar este inconveniente, en la estación se encuentran instalados pre amplificadores, situados inmediatamente después de la antena, aumentan el nivel de señal, así la atenuación, y el ruido que pueda introducirse en la línea de transmisión, le afecte poco.

La estación posee dos pre amplificadores, para los sistemas radiantes de las bandas de 2 metros y 70 centímetros, respectivamente. Ambos pre amplificadores han sido fabricados por el fabricante SSB.

3.1.1.5.1. SSB SP2000

El pre amplificador SP2000 está diseñado para la banda de operación de 2 metros (144 – 145 MHz). Dispone de una configuración de tres niveles de amplificación. Se debe seleccionar uno de los tres niveles, en función de las pérdidas producidas en el cable de la línea de transmisión. La selección del nivel se realiza mediante un potenciómetro, situado en el circuito impreso del pre amplificador, cubierto por la caja protectora de exterior. Además, este pre amplificador puede funcionar de manera autónoma o con un controlador de secuencia DCW-2004B. La potencia máxima admisible, en modo *bypass* (transmisión), es de 750 W, utilizando el controlador de secuencia [14]. La Figura 21, muestra una ilustración del pre amplificador utilizado para la rama de operación de 2 metros.

Figura 21. Pre amplificador SSB SP2000

3.1.1.5.2. SSB SP7000

El pre amplificador SP7000 está diseñado para la banda de operación de 70 centímetros (432 – 440 MHz). Dispone de una configuración de tres niveles de amplificación. Se debe seleccionar uno de los tres niveles, en función de las pérdidas producidas en el cable de la línea de transmisión. La selección del nivel se realiza mediante un potenciómetro, situado en el circuito impreso del pre amplificador, cubierto por la caja protectora de exterior. Además, este pre amplificador puede funcionar de manera autónoma o con un controlador de secuencia DCW-2004B. La potencia máxima admisible, en modo *bypass* (transmisión), es de 500 W, utilizando el controlador de secuencia [14]. La Figura 22, muestra una ilustración del pre amplificador utilizado para la rama de operación de 70 centímetros.

Figura 22. Pre amplificador SSB SP7000

3.1.1.6. Secuenciador SSB DCW-2004B

Los pre amplificadores, descritos en el punto anterior, necesitan ser alimentados eléctricamente para funcionar. La alimentación puede ser realizada directamente mediante línea eléctrica de 12 – 13,8V, modo de funcionamiento

autónomo, o mediante el secuenciador DCW-2004B. En la estación terrena GS-UVa, la alimentación y control de los pre amplificadores, se realiza con dos controladores de secuencia DCW-2004B. Este dispositivo, alimenta y controla los pre amplificadores a través de mismo cable coaxial, de la línea de transmisión. Realiza la conmutación a modo bypass, cuando la radio entra en modo de transmisión [14]. El secuenciador recibe señales de control de transmisión directamente de la radio, a través de uno de los puertos diponibles en el ICOM IC-910. El uso del secuenciador, permite utilizar las potencias descritas, en el punto anterior, para cada uno de los pre amplificadores, si no se utilizase el secuenciador, las potencias de transmisión máximas serían menores.

Figura 23. Secuenciador SSB DCW-2004B

3.1.1.7. Downconverter UEK-3000

El equipo de radio utilizado, como hemos visto en el punto de descripción de la radio, no posee soporte para la banda de 2400 MHz. Para la utilización de la banda de 2400 MHz, es necesario utilizar un conversor de frecuencia, que desplace la información desde alta frecuencia, a una frecuencia en la que opere la radio. Para poder realizar esta conversión, se ha utilizado un *downconverter* de SSB, en concreto el modelo UEK-3000, que convierte las señales RF de 2400 MHz a una frecuencia intermedia de 144 MHz, capaz de manejar por el equipo transceptor de GS-UVA [15].

Hasta el momento de escritura de este documento, esta rama de RF de la estación terrena, no ha sido utilizada, pero está planificado, para el futuro, la compra de un relé electrónico de conmutación coaxial, para que pueda ser controlada la conmutación entre la línea de 2 metros y la de 2400 MHz a través del PC. Una característica importante que posee el conversor, es la integración

de un pre amplificador de señal, con una ganacia típica de 30 dB. Para el funcionamiento de este dispositivo, es necesaria alimentación eléctrica de 12 – 13,8 V, y una corriente mínima de 150 mA [15].

La recepción y transmisión con la antena G3RUH, se realiza sintonizando la banda de 144 MHz, aunque realmente se esté operando la banda de 2400 MHz.

Figura 24. Downconverter SSB UEK-3000

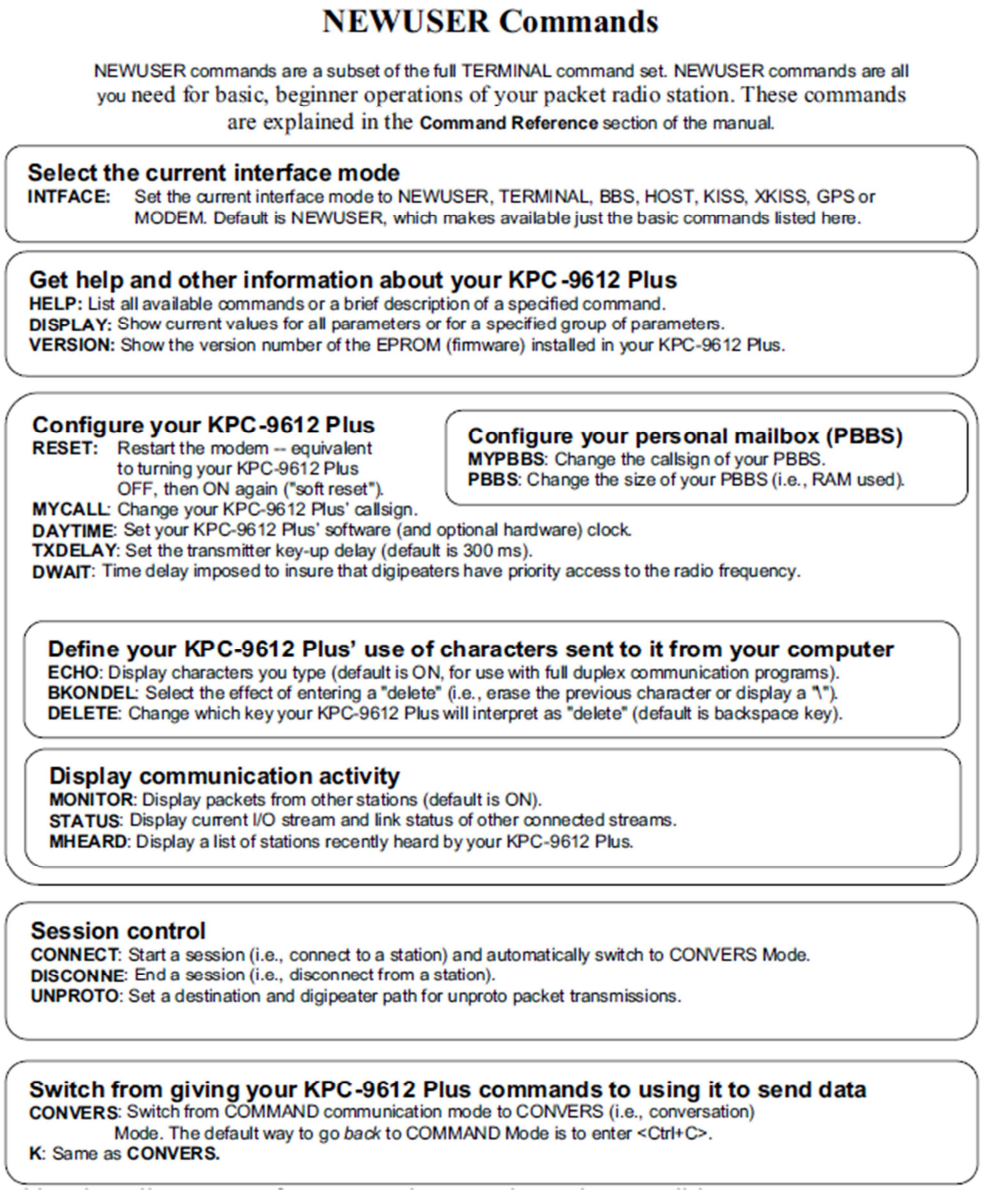
3.1.1.8. TNC

La estación terrena, tiene equipada un TNC (Terminal Node Controller), este dispositivo, es básicamente un modem que convierte las señales digitales a señales de audio, y viceversa. Estos módems, normalmente utilizan el protocolo AX.25 (Amateur X.25), basado en el popular protocolo X.25, pero con adaptaciones especiales al entorno radioaficionado. También suelen incorportar soporte para el protocolo KISS, para transmitir paquetes IP sobre AX.25, en conexión serie asíncrona.

El modelo de TNC instalado en la estación terrena GS-UVa es del fabricante Kantronics, y en concreto el modelo utilizado es el KPC-9612 Plus. Este modelo de TNC dispone de dos puertos de conexión con la radio. El

puerto 1 funciona a una tasa de símbolos de 1200 bps, con modulación AFSK (Audio FSK), con tonos 1300 Hz y 2100 Hz. El puerto 2, utiliza mayor complejidad, permite la transmisión digital a mayor tasa de símbolos, concretamente están soportadas las siguientes tasas: 4800, 9600, 19200 y 28400 bps. El esquema de modulación utilizado, para las señales de este puerto, es la modulación DFSK con filtro Gaussiano. Ambos puertos poseen detección externa de portadora y control de PTT (Push to Talk) para la transmisión [16]. En la , podemos observar una ilustración del modelo de TNC utilizado en la estación terrena.

Los comandos básicos de configuración del TNC KPC-9612 Plus, están recogidos en la Figura 25 [16].



NEWUSER Commands

NEWUSER commands are a subset of the full TERMINAL command set. NEWUSER commands are all you need for basic, beginner operations of your packet radio station. These commands are explained in the **Command Reference** section of the manual.

Select the current interface mode
INTFACE: Set the current interface mode to NEWUSER, TERMINAL, BBS, HOST, KISS, XKISS, GPS or MODEM. Default is NEWUSER, which makes available just the basic commands listed here.

Get help and other information about your KPC-9612 Plus
HELP: List all available commands or a brief description of a specified command.
DISPLAY: Show current values for all parameters or for a specified group of parameters.
VERSION: Show the version number of the EPROM (firmware) installed in your KPC-9612 Plus.

Configure your KPC-9612 Plus
RESET: Restart the modem -- equivalent to turning your KPC-9612 Plus OFF, then ON again ("soft reset").
MYCALL: Change your KPC-9612 Plus' callsign.
DAYTIME: Set your KPC-9612 Plus' software (and optional hardware) clock.
TXDELAY: Set the transmitter key-up delay (default is 300 ms).
DWAIT: Time delay imposed to insure that digipeaters have priority access to the radio frequency.

Configure your personal mailbox (PBBS)
MYPBBS: Change the callsign of your PBBS.
PBBS: Change the size of your PBBS (i.e., RAM used).

Define your KPC-9612 Plus' use of characters sent to it from your computer
ECHO: Display characters you type (default is ON, for use with full duplex communication programs).
BKONDEL: Select the effect of entering a "delete" (i.e., erase the previous character or display a "\").
DELETE: Change which key your KPC-9612 Plus will interpret as "delete" (default is backspace key).

Display communication activity
MONITOR: Display packets from other stations (default is ON).
STATUS: Display current I/O stream and link status of other connected streams.
MHEARD: Display a list of stations recently heard by your KPC-9612 Plus.

Session control
CONNECT: Start a session (i.e., connect to a station) and automatically switch to CONVERS Mode.
DISCONNE: End a session (i.e., disconnect from a station).
UNPROTO: Set a destination and digipeater path for unproto packet transmissions.

Switch from giving your KPC-9612 Plus commands to using it to send data
CONVERS: Switch from COMMAND communication mode to CONVERS (i.e., conversation) Mode. The default way to go *back* to COMMAND Mode is to enter <Ctrl+C>.
K: Same as **CONVERS.**

Figura 25. Comandos básicos de Kantronics KPC-9612 Plus

3.1.1.9. PC de control

La recepción de trabajos de seguimiento, y el control del hardware, se realiza mediante un PC. El PC instalado en la estación terrena, está equipado con un procesador de 64 bits, con cuatro núcleos del fabricante AMD. Tiene instalado 8 Gbytes de memoria RAM, y dos discos duros de 1 TByte, de tecnología SATA, con una configuración RAID en espejo. Además, para la conexión de los equipos mediante RS-232, tiene instalada una tarjeta PCI, que

provee a la maquina de los puertos de serie RS-232, necesarios para la interconexión de los equipos. La conexión a Internet se realiza mediante interfaz Ethernet de 100 mbps, conectado a la red de la Universidad de Valladolid, integrada en la red de investigación española RedIris.

3.1.2 Arduino

Arduino es una plataforma de dispositivos electrónicos *open source* de uso fácil, tanto en hardware como software. Esta plataforma proporciona placas microcontroladoras de desarrollo, que permiten el manejo de entradas y salidas, digitales y analógicas. El hardware utilizado en este tipo de placas, se basa en microcontroladores del fabricante Atmel, normalmente. Estas placas contienen todos los elementos necesarios para su funcionamiento, para que el usuario solo tenga que conectar sus dispositivos y grabe el programa en el controlador.

La comunicación con el microcontrolador para su programación, se realiza mediante RS 232. En el mercado existen multitud de placas (shields) con sensores, controladores de comunicaciones, GPS,... La plataforma Arduino, también proporciona las herramientas necesarias para el desarrollo de los programa, mediante un IDE (Interfaz de programación), con soporte en lenguaje C/C++, con particularidades para cada modelo de Arduino [17].

En la Figura 26, se muestra la ilustración del Arduino micro, que se utilizará en el proyecto.

Figura 26. Arduino micro

En la actualidad, la plataforma Arduino está muy extendida. Gracias a su versatilidad y bajo coste, miles de proyectos han sido desarrollados con

Arduino, los cuales muchos de ellos han sido publicados libremente por Internet como proyectos *open source*.

En este proyecto, se utilizará Arduino micro, junto con transistores MOSFET (shield) para el control de la polarización de las antenas, para las bandas de 2 metros y 70 centímetros. También es posible su utilización, para el control del relé de conmutación coaxial, para conmutar entre la banda de 2400 MHz y 144 MHz.

3.2 Software

El software, que se utilizará para el desarrollo del proyecto, se compone principalmente de bibliotecas de funciones de código abierto. Estas bibliotecas proporcionan las funciones necesarias para el control de la radio o lo rotores. Proporcionan los algoritmos necesarios para el cálculo de posiciones de los satélites, en un tiempo dado. Permiten controlar la grabación de audio, proveniente de la tarjeta de sonido, comunicaciones mediante HTTP y desarrollo del entorno gráfico.

3.2.1 Lenguajes de programación C y C++

El lenguaje de programación C fue creado a principios de los años 1970, por Dennis M. Ritchie, en los laboratorios AT&T Bell. El lenguaje es derivado de otro lenguaje de programación no tipado, el BCPL. En 1982 el lenguaje C fue estandarizado [18].

El lenguaje C dispone de varios tipos primitivos de datos, como pueden ser enteros, caracteres, flotantes y dobles. También dispone de estructuras de datos, enumeraciones y tipos definidos por las bibliotecas de C o por el usuario. A diferencia de otros lenguajes de programación de alto nivel, C permite gestionar la reserva, uso y liberación de la memoria, utilizada por el programa, mediante punteros a direcciones de memoria. Permite el desarrollo de controladores de dispositivos. El núcleo del Sistema Operativo Linux, está desarrollado en C.

El lenguaje de programación C++ fue creado por Bjarne Stroustrup, en los laboratorios AT&T Bell. El primer estándar de este lenguaje data del año 1997

[19]. El lenguaje está basado en su antecesor C, y son totalmente compatibles entre si. La compilación de ambos lenguajes es diferente, pero se puede indicar, durante el preprocesado, el tipo de bibliotecas que va a utilizar, si han sido compiladas en C o C++. Este lenguaje, como se ha explicado antes, hereda la mayoría de características de C, lo que hace que sea diferente es el soporte de clases, para programación orientada a objetos.

La librería estándar de C++, provee a los desarrolladores de un conjunto de clases que facilitan la labor de programación, como pueden ser la listas, mapas de valores, colas,... En lenguaje C, para crear una lista, se tenía que realizar mediante reservas de memoria y un índice, en un estructura de datos, con estas clases, el nivel de abstracción ha aumentado, siendo el proceso de manejo de memoria transparente para el desarrollador.

Los lenguajes de programación C y C++ continúan en pleno desarrollo, cada cierto tiempo se publica un nuevo estándar, recogiendo las novedades introducidas en los lenguajes.

Como se ha visto en apartados anteriores, GENSO utilizó Java para el desarrollo, SatNOGS y Raisin utilizaron Python. Estas elecciones, posiblemente, fueron realizadas por sus autores, por la facilidad de desarrollo de las aplicaciones, gracias al gran nivel de abstracción con respecto a C/C++. La elección de utilizar C/C++ es por la comodidad que dispongo para el desarrollo en estos lenguajes, al ser unos lenguajes que domino, sin necesidad de aprender otros lenguajes para llevar a cabo este proyecto. Además, SatNOGS, utiliza una API REST para sus comunicaciones, indenpendiente del lenguaje de programación utilizado.

3.2.2 HAMLIB

Hamlib es un proyecto, de bibliotecas de funciones, de código abierto, para el control de equipos de radioaficionado, permite el control de rotores y radio [20].

En la actualidad esta biblioteca soporta un buen número de equips de radio y rotores. Las implementaciones de cada modelo de dispositivo de encuentran en versiones alpha y beta, en la mayoría de los casos, a la espera de reportes de

funcionamiento de los usuarios. Dada esta situación de desarrollo y pruebas, no se descarta que existan errores en las bibliotecas de hamlib.

El proyecto fue creado por Frank Singleton y Stephane Fillod, quienes mantienen las bibliotecas y documentación, junto con la comunidad de desarrolladores y colaboradores. El motivo de este proyecto, está fundamentado en la búsqueda de una interfaz común de control de equipos de radioaficionado, sin necesidad de disponer de otras bibliotecas para cada dispositivo. Así, las labores de control que realiza la biblioteca, son transparentes al usuario, sin necesidad de conocer los protocolos de cada dispositivo, únicamente hay que indicar que dispositivo se va a controlar.

Las bibliotecas, se descomponen en dos, RIG contiene el conjunto de funciones necesarias pora el control de equipos de radio y ROT contiene el conjunto de funciones necesarias para el control de rotores.

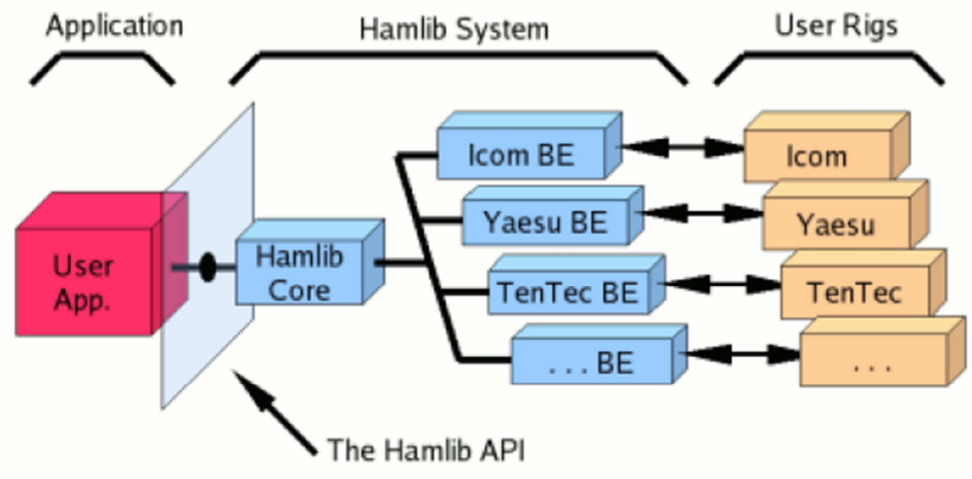


Figura 27. Capas de Hamlib

Las principales funciones de RIG a utilizar en el proyecto son:

- **rig_init** – función para inicializar el handle de la radio.
- **rig_open** – función de apertura del puerto serie configurado para la radio.
- **rig_close** – función de cierre del puerto serie utilizado por la radio.
- **rig_cleanup** – función de limpieza y liberación de memoria reservada por las funciones de control de la radio.

- **rig_set_freq** – función para el envío de la frecuencia a la radio.
- **rig_get_freq** – función de consulta de la frecuencia actual de la radio.
- **rig_set_mode** – función para el envío del tipo de modulación a la radio.
- **rig_set_mode** – función de consulta del tipo de modulación actual, utilizada, por la radio.
- **rig_set_vfo** – función para el envío del vfo a utilizar a la radio.
- **rig_get_vfo** – función de consulta del vfo utilizado en la radio.
- **rig_get_powerstat** – función de consulta de la potencia de transmisión actual de la radio.

Las principales funciones de ROT a utilizar en el proyecto son:

- **rot_init** – función para inicializar el handle de los rotores.
- **rot_open** – función de apertura del puerto serie configurado para los rotores.
- **rot_close** – función de cierre del puerto serie utilizado por los rotores.
- **rot_cleanup** – función de limpieza y liberación de memoria reservada por las funciones de control de rotores.
- **rot_set_position** – función para el envío de la posición de azimuth y elevación a los rotores.
- **rot_get_position** – función para la consulta de la posición actual de los rotores.
- **rot_reset** – función de reseteo de los rotores.

La utilización de esta biblioteca, tiene una gran ventaja, que es el nivel de abstracción que proporciona para el control de distintos dispositivos, sin tener que entrar a profundizar en los protocolos de comunicaciones de cada uno, ni control de los dispositivos serie asociados. Como desventaja, podemos incluir la fase de desarrollo del proyecto, en estado alpha y beta, que puede incluir errores de funcionamiento, no siendo estable. Aun teniendo estos inconvenientes, para los propósitos del proyecto son más que suficientes.

3.2.3 SatNOGS API

El proyecto SatNOGS es un proyecto libre y abierto, que proporciona datos de satélites. Coordina el seguimiento de satélites planificados, para cada una de las estaciones terrenas conectadas a la red. Se compone de base de datos, donde se guardan los datos de los satélites, servidor de la red, controla los seguimiento, recibe y guarda los datos de las observaciones, y el cliente, encargado de controlar la estación terrena.

La comunicación que realiza la red con los cliente, se efectua mediante una API REST, sobre el protocolo HTTP, con respuesta en formato JSON (*JavaScript Object Notation*). Es necesario disponer de un token de autentificación, que es enviado en las cabeceras de las peticiones, para la transmisión de datos y control de conexión de la estación con la red. Para la consulta de datos, no es necesario estar autentificado en la red [21].

En el momento de la escritura del presente documento, la API de SatNOGS, ofrece los siguientes recursos REST [22]:

- **jobs.** Proporciona los datos necesarios a los clientes para el correcto seguimiento de los satélites.
    - **id.** Indentificador de la observación a realizar.
    - **start.** Fecha y hora, en UTC, de comienzo del seguimiento.
    - **end.** Fecha y hora, en UTC, de final de seguimiento.
    - **ground_station.** Indentificador de la estación que debe realizar el seguimiento planificado.
    - **tle0.** Línea cero del conjunto de elementos TLE. Necesario para el cálculo de posición.
    - **tle1.** Línea uno del conjunto de elementos TLE. Necesario para el cálculo de posición.
    - **tle2.** Línea dos del conjunto de elementos TLE. Necesario para el cálculo de posición.
    - **frequency.** Frecuencia del enlace de bajada del satélite. Será necesario aplicar el efecto Doppler.

    - **mode.** Modo de funcionamiento/modulación del satélite.
    - **transmitter.** Indentificación del usuario que hace la petición de observación.
    - **baud.** Tasa de transferencia de símbolos para modulaciones digitales.
- **data.** Permite el envío de datos por parte de las estaciones terrenas. Permite la consulta de los datos recopilados de las estaciones terrenas.
    - **id.** Identificador de la observación.
    - **start.** Fecha y hora, en UTC, de comienzo del seguimiento.
    - **end.** Fecha y hora, en UTC, de final del seguimiento.
    - **ground_station.** Identificador de la estación que realizó el seguimiento.
    - **transmitter.** Identificador del usuario que hizo la petición de seguimiento.
    - **norad_cat_id.** Identificador del satélite proporcionado por NORAD.
    - **payload.** Fichero de audio grabado durante el seguimiento.
    - **waterfall.** Fichero de imagen con el espectograma del seguimiento.
    - **demoddata.** Datos demodulados (modulaciones digitales) de la observación.
    - **station_name.** Nombre de la estación que realizó el seguimiento.
    - **station_lat.** Latitud donde se encuentra situada la estación que realizó el seguimiento.
    - **station_lng.** Longitud donde se encuentra situada la estación que realizó el seguimiento.
    - **vetted_status.** Estado de la observación. Seleccionado por el usuario que hace la petición de observación. Sirve como filtro.
    - **client_version.** Versión del cliente de SatNOGS utilizado.

    - **client_metadata.** Metadatos comunicados por el cliente de SatNOGS.
- **observations.** Es un duplicado de data.
- **stations.** Permite el envío y consulta de datos de la estaciones terrenas, registradas en la red.
    - **id.** Identificador de la estación terrena.
    - **name.** Nombre de la estación.
    - **altitude.** Altitud en la que se ubica la estación.
    - **lat.** Latitud en la que se ubica la estación.
    - **lng.** Longitud en la que ubica la estación.
    - **qthlocator.** Localización de la estación en la tabla de localizaciones.
    - **location.** Localización de la estación.
    - **antenna.** Agrupación de valores (*array*) de las antenas disponibles en la estación.
    - **created.** Fecha de creación de la estación en la red.
    - **last_seen.** Última conexión del cliente con la red.
    - **status.** Estado de conexión del cliente en la red.
    - **observations.** Observaciones o seguimientos realizado y/o planificados en la estación.
    - **description.** Descripción proporcionada por el propietario sobre la estación terrena, de seguimiento de satélites.

La ventaja de proporcionar una API REST, es la interacción de programas, desarrollados en cualquier tipo de lenguaje de programación, sin necesidad de estar sujeto al lenguaje de programación base utilizado en la red de SatNOGS. El inconveniente que tiene, es que hay que utilizar bibliotecas que permitan el uso y manejo del protocolo HTTPS, y la formulación correcta de las peticiones y respuestas.

3.2.4 SGP4/SDP4

Para el seguimiento de objetos en el espacio, es necesario disponer de un conjunto de elementos de perturbaciones. Este conjunto de elemento es mantenido y actualizado por NORAD. Los datos proporcionados por NORAD, son periódicamente afinado, para la disposición razonable de predicción. El conjunto de elementos está publicado para ser consultado por todos los usuarios que lo necesite.

La obtención de la posición y velocidad del objeto, en el espacio, es necesario de disponer de herramientas que permitan obtener la predicción de los datos, a partir de los TLE.

El modelo a utilizar es el SGP4 (*Simplified Perturbations Models 4*), desarrollado por Ken Cranford en 1970. Este modelo es válido para objetos cercanos a la Tierra. La obtención de este modelo, resulta de la simplificación de la teoría analítica más extensa de Lane y Cranford (1969), que utiliza la solución de Brouwer (1959) para el modelo gravitacional y la función de densidad de potencia del modelo atmosférico [2].

El modelo SDP4, es un modelo similar a SGP4, con la particularidad de tener cálculos para objetos en espacio profundo [2].

La biblioteca de funciones que vamos a utilizar, está escrita en lenguaje de programación C. En esta biblioteca se encuentran todas las funciones necesarias, y la implementación de los modelos SGP4 y SDP4, para el cálculo de predicciones.

Los autores de la biblioteca, tanto en la creación como aportaciones son, Dr. TS Kelso, Neoklis Kyriazis, Hari Nair y Alexandru Csete.

La utilización de este modelo en el proyecto, viene caracterizada por la amplia utilización en otros proyectos, esto es para nosotros una ventaja, al disponer de documentación necesaria para su utilización. Pueden existir algún inconveniente, en la fiabilidad del cáculo de predicciones, pero los errores son los suficientemente pequeños para considerarlos despreciables.

3.2.5 LIBCURL

LibCURL es una biblioteca de funciones multiprotocolo, proporciona transferencia de ficheros. Los protocolos soportados por la biblioteca son: DICT, FILE, FTP, FTPS, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAP2, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, Telnet y TFTP. Soporta certificados SSL y es multiplataforma, soportando múltiples sistemas operativos [23].

La biblioteca está escrita en el lenguaje de programación C, publicada como proyecto de código abierto y libre. Dispone de soporte en multitud de lenguajes de programación, este hecho hace que sea una librería muy extendida y en continuo desarrollo.

En el proyecto se utilizará la biblioteca con soporte del protocolo HTTPS, para la comunicación del cliente con el servidor de SatNOGS. El intercambio de información entre ambos elementos, resulta más fácil a través de esta biblioteca. Permite la realización de peticiones HTTP de tipo GET, POST, PUT, DELETE, HEAD y OPTIONS, necesarias y requeridas por SatNOGS. Permite añadir cabeceras personalizada en las peticiones HTTP/HTTPS, en este caso necesarias para la autenticación de la estación en la red de SatNOGS.

Esta biblioteca fue creada en 1996 por Daniel Stenberg. En la actualidad, es mantenida por el propio Stenberg y otros colaboradores.

La ventaja de esta biblioteca es el nivel de abstracción e integración que nos proporciona, sobre el protocolo HTTP seguro. Como inconvenientes, podemos destacar que debido al constante desarrollo, hay formas, funciones y métodos de implementaciones de la biblioteca que quedan descatalogados, obligando en un futuro a modificar la aplicación desarrollada.

3.2.6 PortAudio

PortAudio es una biblioteca de funciones multiplataforma, de código abierto, que proporciona las funciones necesarias para el control de los dispositivos de entrada y salida de audio. Permite la escritura de programa en C o C++, compilables en varios Sistemas Operativos, como Windows, Linux,

Macintosh. Está pensado para el uso en sistemas, de software de audio, multiplataforma [24].

El funcionamiento de PortAudio está provisto mediante una API sencilla, con funciones para la grabación y/o reproducción de audio. El funcionamiento se basa en funciones de *callback* e interfaces bloqueantes de lectura/escritura (ficheros, dispositivos de audio,...).

El proyecto PortAudio y se API fueron propuestas realizadas por Ross Bencina y codiseñado por Phil Burk. El proyecto recibe contribuciones de desarrolladores voluntarios [24].

La utilización esta biblioteca en el proyecto, será para la captura de audio necesaria para SatNOGS, recogida durante el tiempo del pase del satélite en seguimiento.

La ventaja que proporciona esta biblioteca, es el manejo fácil de dispositivos de audio, sin entrar en profundidad en funciones de manejo, nativas, del sistema de sonido de Linux. Como inconveniente, realmente no he encontrado inconvenientes a esta biblioteca, para los propósitos del proyecto.

3.2.7 LibSNDFile

La biblioteca de funciones LibSNDFile provee de un conjunto de funciones de lectura y escritura, de ficheros, que contengan muestras de audio, a través de una interfaz estándar. Es una bibliotea de código abierto, distribuida bajo los términos de la licencia "*GNU Lesser General Public License*" (LGPL) [25].

Esta biblioteca fue escrita para compilar y funcionar en Sistemas Operativos Linux, pero en la actualidad también es posible su utilización en Unix y Mac OS X. Existen ficheros binarios precompilados para el Sistema Operativo Windows de 32 y 64 bits [25].

Las características de esta biblioteca son:

- Lectura y escritura de ficheros, con soporte en multitud de formatos.
- Disposición de una API simple, elegante y sencilla de utilizar.
- Utilización en Linux, Unix, MacOS, Win32 y otros Sistemas Operativos.

- Conversión en directo de formato de audio.
- Opción de normalización, en la lectura de datos de tipo coma flotante, en ficheros de tipo de datos entero.
- Apertura de ficheros en modo lectura/escritura.
- Escritura de la cabecera de los ficheros, sin cerrar el fichero (solo disponible en el modo de escritura o lectura/escritura).
- Consulta a la biblioteca de todos los formatos que son soportados, con devolución de información en formato texto, con la descripción de cada uno de los formatos.
- No incluye soporte para MPEG Layer 3 (MP3), debido a un problema de patentes.

El autor de esta biblioteca es Erik de Castro Lopo, apasionado del tratamiento y procesado digital de señal, que publicó la primera versión oficial el 15 de Febrero de 1999. Este proyecto es el resultado de la contribución realizada al código fuente de wavplay, de Linux. El gran interés del autor en DSP, hizo continuar el desarrollo de esta biblioteca de funciones. Actualmente se distribuye como software libre y abierto, bajo las condiciones de la licencia LGPL [25].

En el proyecto, esta biblioteca será utilizada para guardado de los ficheros de audio, en formato Vorbis OGG. Este tipo de codificación de audio es utilizada en la red de SatNOGS, por lo tanto es necesario disponer de herramientas que permitan la codificación del audio en este formato, para la publicación de los resultados.

Las ventajas que nos proporciona esta biblioteca son el manejo fácil de ficheros de audio, tanto para lectura como escritura, así como la codificación y decodificación en varios formatos de audio, como es Vorbis OGG, necesario en el proyecto. Como inconvenientes, no soporta MPEG Layer 3 (MP3), formato de audio muy extendido, que actualmente no se utiliza en el proyecto, pero podría utilizarse en el futuro.

3.2.8 GTKmm

GTK+ o GIMP Toolkit, es un conjunto de herramientas multiplataforma para la creación de interfaces gráficas de usuario (GUI). GTK+ ofrece un completo conjunto de widgets, con utilidad desde pequeñas aplicaciones hasta las más complejas suites de aplicaciones [26]. En la Figura 28, se puede observar un ejemplo visual de widgets incuídos en la biblioteca GTK+.

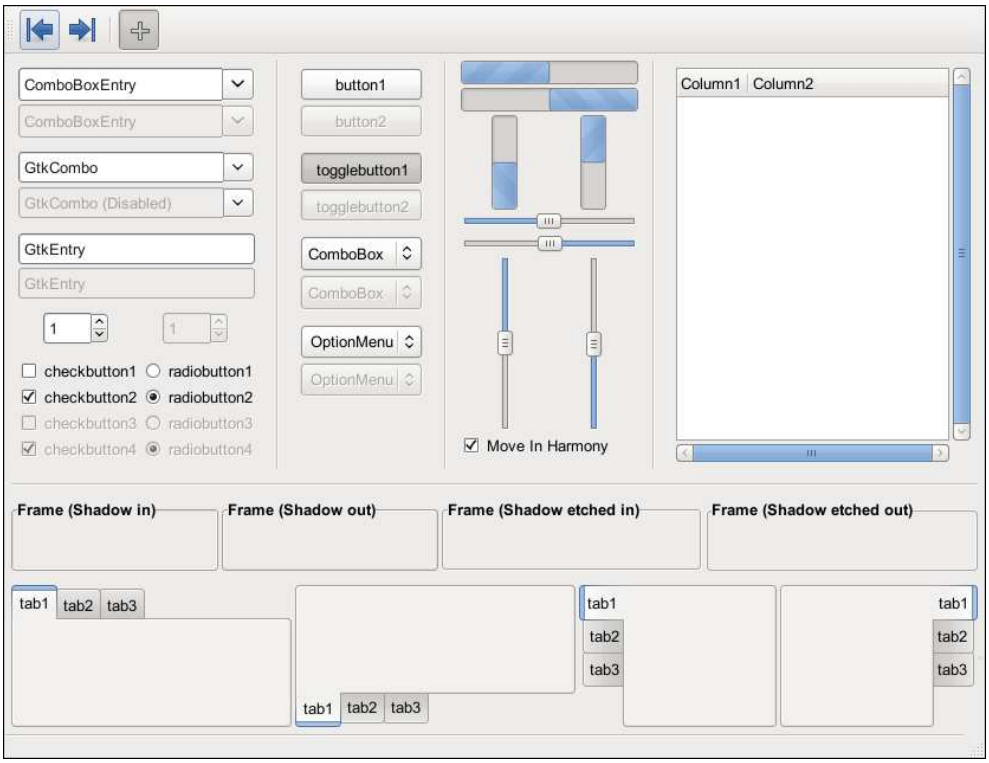


Figura 28. Widgets GTK+

GTK+ está escrita en lenguaje C, pero ha diseñado, desde la base, para soportar un amplio conjunto de lenguaje de programación, lo limitándose únicamente a C o C++. La utilización de GTK+ con lenguajes como Perl o Python, proveen a los desarrolladores de un método rápido de desarrollo de aplicaciones gráficas [26].

GTK+ es un proyecto de código abierto y libre, forma parte del poyecto GNU. La distribución es realizada bajo las condiciones de la licencia LGPL,

que permite la utilización de la biblioteca en todos los desarrollos, incluido el software propietario, sin necesidad de pago de tasas o cánones de utilización [26].

Existen otras bibliotecas para el desarrollo de aplicaciones gráficas, además de GTK+, estas son Qt y WxWidgets. La decisión de utilizar GTK+ es debido al desarrollo y actualización constante de la biblioteca, además del tipo de licencia utilizada. Qt utiliza una doble licencia, la cual no provee de uso libre de la biblioteca, en el caso de software propietario o desarrollos para el Sistema Operativo Windows, obligando al pago de una licencia. En cambio, WxWidgets no dispone del problema de Qt, pero su actualización es más lenta que el resto de bibliotecas.

En el momento de escritura de este documento, GTK+ es utilizada en un gran número de programas, la cifra es superios a las 60.000 aplicaciones, estas cifras arrojan un número positivo del buen soporte que dispone la biblioteca.

El núcleo de GTK+ está mantenido, actualmente, por personas asociadas a grandes empresas como Red Hat, Google, Novell, Codethink, Lanedo GmbH, Canonical (desarrollador de Ubuntu) y Enless Mobile. La coordinación y manenimiento general de la biblioteca es llevada a cabo por la *GNOME Foundation*.

Gtkmm es la interfaz C++ oficial de GTK+ Destacan las funciones *callback* de tipo seguro (en su utilización con varios threads), y un exhaustivo conjunto de widgets, que son fácilmente extensibles mediate herencia de clases, propia de la programación orientada a objetos. Permite el desarrollo de interfaces mediante la programación en C++, o utilizando el diseñador de interfaces Glade, que genera un fichero XML, posteriormente interpretado por la biblioteca a través de Gtk::Builder. Dispone de una documentación extensa, con referencias a la API y tutorial de desarrollo con múltiples ejemplos [27].

Características de gtkmm:

- Utilización de herencia de clases para la extensión de widgets personalizados.
- Manejadores de señales de tipo seguro, en C++ estándar.
- Polimorfismo.

- Utilización de la biblioteca de C++ estándar, incluyendo cadenas, contenedores e iteradores.
- Internacionalización completa con UTF-8.
- Gestión completa de memoria C++.
    - Composición de objetos.
    - Liberación automática de widgets reservados dinámicamente (recolector de basura).
- Uso completo de los espacios de nombres de C++.
- No contine macros.
- Multi plataforma: Linux, FreeBSD, NetBSD, Solaris, Win32, MacOSX y otros.
- Software libre, sin coste para desarrolladores de software propietario.
- Discutido, diseñado e implentado por el público.

Gtkmm, al igual que la biblioteca base GTK+, se distribuye con licencia LGPL. Es utilizada por un amplio conjunto de aplicaciones, como por ejemplo gparted (gestor gráfico de particiones para Linux).

Para la creación de la interfaz gráfica de usuario, en el proyecto, se utilizará GTK+, mediante gtkmm.

3.2.9 GooCanvasmm

GooCanvas es una biblioteca de funciones para porporcionar funciones de canvas a GTK+. Esta biblioteca utiliza, para el widget de canvas, la biblioteca de dibujo en 2 dimensiones Cairo.

La biblioteca es de código libre y abierto, liberada bajo licencia LGPL. Está mantenida por la GNOME Foundation. Está escrita en lenguaje de programación C, pero tiene soporte para varios lenguajes de programación.

Características de GooCanvas [28]:

- Separación opcional de modelo/vista.
- Uso de interfaces para elementos y vistas.

- Elementos básicos – rectángulos, elipses, polígonos, líneas, texto, imágenes, rejillas, grupos,...
- Elemento tabla como plantilla para otros elementos (similar al widget GtkTable).
- Widgets de GTK+ incrustados.
- Soporte de cascada de estilos (CSS).
- Transformaciones para todos los elementos – rotación, escalado,...
- Gestión de eventos – botones, movimientos,...
- Imprimible.
- Scrolling.
- Zooming.
- Configuración de visibilidad de los elementos.
- Escalable.
- Soporte de varias unidades de medida – píxeles, puntos, pulgadas, milímetros.
- Animaciones simples.

GooCanvasmm, al igual que la biblioteca base GooCanvas, se distribuye bajo los términos de la licencia LGPL. Es el soporte oficial de la biblioteca para el lenguaje C++.

En el proyecto se espera la utilización, de esta biblioteca, para el dibujado de gráficos.

4 Requisitos previos para el funcionamiento del software

El sistema utilizado, para el funcionamiento de la aplicación de control de la estación terrena, es el Sistema Operativo Linux Mint versión 18.2 Cinnamon.

El software cliente desarrollado, tiene dependencias con varias bibliotecas, a continuación se realiza un listado de bibliotecas utilizadas, y la versión de las mismas. Además, se explica la instalación de las bibliotecas en Linux Mint.

Listado de bibliotecas utilizadas:

- libcurl3 – versión 7.47.0
- libcurl3-gnutls – versión 7.47.0
- libportaudio2 – versión 19+svn20140130-1build1
- libsndfile1 – versión 1.0.25-10ubuntu0.16.04.1
- libgtkmm-3.0-1v5 – versión 3.18.0-1
- libgoocanvasmm-2.0-6 – versión 1.90.11-0ubuntu1
- libhamlib2 – versión 1.2.15.3-3.1build1
- libhamlib-utils – versión 1.2.15.3-3.1build1

Listado de bibliotecas de desarrollo utilizadas:

- libcurl4-openssl-dev – versión 7.47.0
- libportaudiocpp0 – versión 19+svn20140130-1build1
- portaudio19-dev - versión 19+svn20140130-1build1
- libsndfile1-dev – versión 1.0.25-10ubuntu0.16.04.1
- libgtkmm-3.0-dev – versión 3.18.0-1
- libgoocanvasmm-2.0-dev – versión 1.90.11-0ubuntu1
- libhamlib-dev – versión 1.2.15.3-3.1build1

Las instalación de las bibliotecas necesarias, se he realizado con la herramienta apt-get. En el intérprete de comandos se debe introducir *apt-get install [nombre_paquete/s]*, con permisos de superusuario y sustituyendo *[nombre_paquete/s]* por el nombre del paquete o paquetes a instalar.

5 Desarrollo de la aplicación cliente

En las próximas líneas se van a describir y detallar el desarrollo de los módulos y componentes que forman la aplicación cliente. Estos componentes, se distribuyen en grupos, en función de las funcionalidades que aportan al sistema.

En el anexo 2 se pueden ver las principales estructuras de datos utlizadas en el desarrollo de la aplicación.

5.1 Comunicación con SatNOGS

La comunicación con los servidores de la red de SatNOGS, se realiza únicamente, para la obtención de los datos de satélites a seguir, y para la transmisión de los datos recopilados.

5.1.1 Petición de seguimiento de satélite

La petición con el servidor de SatNOGS, para recibir la lista de seguimiento, se realiza mediante API REST, sobre protocolo HTTP seguro. Es necesario hacer una petición, de tipo HTTP GET, con el ID de estación como parámetro (https://network.satnogs.org/api/jobs/?ground_station=[ID_Estación]).
Además, es necesario enviar una cabecera personalizada, en la petición, con el token de autenticación, "Authorization: Token [id del token]".

Para la realización de la petición, ha sido empleada la biblioteca libCURL. El siguiente código muestra la configuración y petición:

```
CURL * curl; /* Manejador de CURL. */
CURLcode res; /* Variable con el código de resultado devuelto por CURL. */
std:string datos; /* Variable de tipo string, para guardar los datos de la petición. */
struct curl_slist * headers = NULL; /* Variable con la lista de cabeceras personalizadas. */

curl = curl_easy_init(); /* Función de inicialización de CURL. */
```

```
if(curl)
{
headers = curl_slist_append(headers, "Authorization: Token [Token ID]"); /* Cabecera de autenticación. */
curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers); /* Añade las cabeceras a CURL. */
curl_easy_setopt(curl, CURLOPT_URL, https://network.satnogs.org/api/jobs/?ground_station=[ID_ESTACIÓN]); /* Añade la URL de petición a CURL. */
curl_easy_setopt(curl, CURLOPT_HTTPGET, 1); /* Configura CURL para realizar petición HTTP GET. */
curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, curlJobsReceiverCallback); /* Configura en CURL, la función de callback para la recepción de los datos de la petición. */
curl_esay_setopt(curl, CURLOPT_WRITEDATA, (void *)&datos); /* Argumento, de tipo puntero vacío, que se pasa a la función de callback, al ser llamada. */
res = curl_easy_perform(curl); /* Realiza la petición HTTP */
curl_easy_cleanup(curl); /* Libera la memoria utilizada por CURL */
curl_slist_free_all(headers); /* Libera la memoria utilizada por la lista de cabeceras personalizadas. */
}
```

Los datos devueltos, por la red de SatNOGS, están en formato JSON, para la decodificación de los datos, se ha utilizado la biblioteca nxJSON. Los datos que devuelve la petición, se han detallado en el apartado de *Métodos y materiales a utilizar*.

El siguiente código es el utilizado para la decodificación de los datos JSON:

```
const nx_json * json = nx_json_parse((char *)datos.c_str(), 0); /* Crea el objeto nx_json y decodifica los datos, pasados por la cadena de caracteres datos. */
```

```
if(json)
{
    for(int i=0; i<json->lenght; i++) /* Recorre todos los datos JSON. */
    {
        const nx_json * ítem = nx_json_item(json, i); /* Obtiene el ítem JSON, con índice i. */
        satnogs_job_t *job = (satnogs_job_t *)malloc(sizeof(satnogs_job_t)); /* Puntero, con reserva de memoria, para el trabajo de seguimiento del satélite. */
        job->id = nx_json_get(ítem, "id")->int_value); /* Obtiene un dato de tipo entero, para el índice id. */
        job->start = nx_json_get(ítem, "start")->text_value); /* Obtiene un dato de tipo texto, para el índice start. */
        ... /* Más líneas de obtención de datos. */
        job->baud = nx_json_get(ítem, "baud")->dbl_value); /* obtiene un dato de tipo doble, para el índice baud. */
    }
    ... /* Más líneas, para la programación del  seguimiento. */
}
```

El último paso, tras la obtención de todos los datos necesarios, es la programación del seguimiento. Para la programación de tareas, ha sido necesario crear una clase Scheduler, que detallaremos su funcionamiento más adelante. En el siguiente código, se muestra la creación de la tarea y como se añade al programador de tareas.

```
time_t time_start = time(0); /* Variable time_t, que guarda los segundos de timestamp, para el comienzo de la tarea. */
time_start = str_to_time(job->start); /* Asigna el valor de comienzo, de la tarea de seguimiento, a time_t. */
task_t task; /* Variable de tipo estructura de datos task_t. */
task.id = strdup(buffer); /* ID de la tarea a realizar. */
```

```
task.name = strdup(job->tle0); /* Nombre de la tarea. */
task.interval = 0; /* Establece el valor 0, para indicar que no es una tarea de tipo intervalo. */
task.interval_value = 0; /* Establece el intervalo de ejecución, solo para tareas de tipo invervalo. */
task.func = tracking_satellite; /* Nombre de la función de callback, que ejecutará la tarea. */
task.args = (void *)job; /* Variable de tipo puntero vacío, con el argumento a pasar, en la llamada de la función callback. */
scheduler->in(time_start, task); /* Añade la tarea al scheduler, especificando el tiempo de inicio y la estructura de datos, con la tarea. */
```

5.1.2 Transmisión de datos a SatNOGS

La transmisión de datos a SatNOGS, al igual que se hace con la petición de datos, se realiza mediante API REST, sobre HTTP. Para el envío de los datos, se hace uso de la petición HTTP PUT, en el cual se envían los parámetros, fuera de la URL.

La transmisión de datos, que realizaremos a SatNOGS, son básicamente el audio grabado, durante el pase, y datos obtenidos, si están disponibles.

La petición HTTP, es muy similar al caso anterior, en el siguiente código se muestran únicamente el nuevo código y modificaciones.

```
struct curl_httppost * formpost = NULL; /* Puntero de tipo estructura, al formulario POST. */
struct curl_httppost * lastptr = NULL; /* Punterio para la finalización del formulario POST. */


curl_formadd(&formpost, &lastptr, CURLFORM_COPYNAME, "payload", CURLFORM_FILE, audio_filename, CURLFORM_END); /* Añade al formulario, el campo payload, y carga el fichero, al ser un campo de tipo file. */
```

```
curl_easy_setopt(curl, CURLOPT_HTTPPOST, formpost); /* Añade el formulario POST a CURL. */
curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, "PUT"); /* Establece el método PUT. */
```

5.2 Seguimiento de satélites

Para el seguimiento de los satélites, es necesario disponer de las herramientas adecuadas, para las predicciones de los datos de posición y velocidad. Utilizaremos el modelo matemático SGP4. También es necesario, disponer de los elementos hardware necesarios para la recepción de las señales, así como de las herramientas software, que permitan controlar el hardware. Por último, la recogida de datos se efectua mediante la grabación de audio, obtenido de la radio durante el seguimiento.

Las funciones para el seguimiento son, tracking_init, encargada de inicializar los elementos hardware para el seguimiento, y tracking_satellite, que ejecuta la tarea de seguimiento.

La inicialización de los elementos hardware se realiza mediante el siguiente código:

```
Radio * radio; /* Variable del objeto Radio. */
Rotator * rotator /* Variable del objeto Rotator, para el control de rotores. */
radio = new Radio(RIG_MODEL_IC910, "/dev/ttyS4"); /* Crea un nuevo objeto Radio, y establece el modelo de radio a utilizar, así como el puerto de comunicación. */
rotator = new Rotator(ROT_MODEL_RC2000, "/dev/ttyS0"); /* Crea un nuevo objeto Rotator, y establece el modelo de rotores a utilizar, así como el puerto de comunicación. */
```

5.2.1 Predicción de datos del satélite

En las líneas de código, que se detallan más abajo, se obtiene los datos de prección a partir de los elementos TLE. Antes de comenzar las predicciones, y una vez cargado los datos, a partir de los TLE, es necesario realizar una inicialización. Inicialmente, no se realizaba esta inicialización, y lo datos obtenidos de predicciones eran erróneos, observando el código de la aplicación GPredict, de Alexandru Csete, se podía ver que realiza una inicialización del satélite, antes de realizar la predicción. Añadida esta inicialización a la aplicación, el problema de errores en la predicción desapareció.

El código para la carga de datos del satélite a partir de un conjunto TLE es el siguiente:

```
char tle_str[3][80]; /* Doble array de tipo char, el primer elemento del array determina la línea del TLE, el segundo elemento del array contiene el valor de la línea del TLE. */
double pass_time; /* Variable de tipo doble, con el valor del tiempo en función del calendario Juliano, para la predicción de los datos. */
strcpy(tle_str[0], job->tle0); /* Copia la línea cero del TLE. */
strcpy(tle_str[1], job->tle1); /* Copia la línea uno del TLE. */
strcpy(tle_str[2], job->tle2); /* Copia la línea dos del TLE. */
qth.lat = 41.6621; /* Asigna el valor de la latitud local, a la estructura del lugar donde se realiza la observación. */
qth.lon = -4.7055; /* Asigna el valor de la longitud local, a la estructura del lugar donde se realiza la observación. */
qth.alt = 710/1000; /* Asigna el valor de la altitud local, en kilómetros, a la estructura del lugar donde se realiza la observación. */
pass_time = get_current_daynum(); /* Asigna el tiempo actual, en calendario Juliano, a la variable. */
select_ephemeris(&sat); /* Selecciona la efeméride, para el objeto satélite, pasado por puntero. */
sat_data_init_sat(&sat, &qth); /* Inicializa valores, de la estructura satélite, a partir de los datos que ya dispone y de la estructura del qth. */
```

```
sat.aos = find_aos(&sat, &qth, pass_time, 0.0); /* Realiza un barrido de tiempo, para la obtención de la fecha y hora de comienzo de cobertura del satélite. */
sat.los = find_los(&sat, &qth, pass_time, 0.0); /* Realiza un barrido de tiempo, para la obtención de la fecha y hora de pérdida de cobertura del satélite. */
predict_calc(&sat, &qth, pass_time); /* Realiza la predicción para sat, en función de qth y pass_time, el resultado se actualiza en sat. */
```

5.2.2 Inicialización de la estructura de datos sat_t

Como se ha comentado en el subapartado anterior, es necesario, una vez cargados los tle a la estructura sat_t, realizar la inicialización antes de realizar una predicción. En el siguiente código se puede ver el proceso de inicialización, a través de la función sat_data_init_sat.

```
geodetic_t obs_geodetic; /* Estructura de datos geodésicos del observador. */
obs_set_t obs_set; /* Estructura de datos del observador. */
geodetic_t sat_geodetic; /* Estructura de datos geodésicos del satélite. */
doublé jul_utc, age; /* Variables de datos de tipo doble, para la fecha UTC en Juliano, y la edad del satélite, respectivamente. */

/* Conversión y asignación de la fecha epoch del TLE a la estructura sat_t. */
jul_utc = Julian_Date_of_Epoch(sat->tle.epoch);
sat->jul_epoch = jul_utc;
/* Inicialización de la estructura geodetic_t para el observador */
obs_geodetic.lon = qth->lon * de2ra; /* Convierte la longitud de grados a radianes. */
obs_geodetic.lat = qth->lat * de2ra; /* Convierte la latitud de grados a radianes. */
```

```
obs_geodetic.alt = qth->alt/1000.0;
obs_geodetic.theta = 0;
/* Aplicamos los modelos SGP4 o SDP4, según el tipo de satélite. */
if(sat->flags & DEEP_SPACE_EPHEM_FLAG)
{
    SDP4(sat, 0.0); /* Aplica el modelo SDP4, descrito en [2]. */
}
else
{
    SGP4(sat,0.0); /* Aplica el modelo SGP4, descrito en [2]. */
}
/* Convierte la posición a km, y la velocidad a km/s. */
Convert_Sat_State(&sat->pos, &sat->vel);
/* Obtiene la velocidad y posición del satélite. */
Magnitude(&sat->vel);
sat->velo = sat->vel.w;
Calculate_Obs(jul_utc, &sat->pos, &sat->vel, &obs_geodetic, &obs_set);
CalculateLatLonAlt(jul_utc, &sat->pos, &sat_geodetic);

/* Ajusta el valor de la longitud del satélite al rango entre –pi y pi. */
while(sat_geodetic.lon < -pi)
{
    sat_geodetic.lon += twopi;
}
while(sat_geodetic.lon > pi)
{
    sat_geodetic.lon -= twopi;
}
/* Asignación de los valores calculados a la estructura sat_t. */
sat->az = Degrees(obs_set.az); /* Azmimuth en grados. */
sat->el = Degrees(obs_set.el); /* Elevación en grados. */
sat->range = obs_set.range;
```

```
    sat->range_rate = obs_set.range_rate.
    sat->ssplat = Degrees(sat_geodetic.lat); /* Latitud del satélite en grados.
*/
    sat->ssplon = Degrees(sat_geodetic.lon); /* Longitud del satélite en
grados. */
    sat->alt = sat_geodetic.alt
    sat->ma = Degrees(sat->phase); /* Fase en grados, para la perturbación
media. */
    sat->ma *= 256.0/360.0; /* Perturbación media. */
    sat->footprint = 2.0 * xkmper * acos(xkmper/sat->pos.w); /* Área de
cobertura. */
    age = 0.0;
    sat->orbit = (long) floor((sat->tle.xno * xmnpda/twopi + age * sat-
>tle.bstar * ae) * age + sat->tle.xmo/twopi) + sat->tle.revnum -1 ; /*
Cálculo y asignación del número de órbita, del satélite. */
    sat->otype = get_orbit_type(sat); /* Tipo de órbita del satélite. */
```

5.2.3 Cálculo de la frecuencia, aplicando el efecto Doppler

Los satélites son cuerpos en movimiento, la velocidad a la que se trasladan es del orden de kilómetros por segundo. El hecho de que un satélite sea un cuerpo en movimiento, hace que la frecuencia con la que transmite, no sea la misma en el receptor. La frecuencia, que debe sintonizar el observador, debe ser el resultado de aplicar el efecto Doppler [3].

Durante el desarrollo, inicialmente, se aplicó la fórmula que viene dada en [3], cuando el transmisor está en movimiento, Esta fórmula, siempre daba como resultado la misma frecuencia de Doppler, y no se capturaba bien la señal de los satélite.

Para solventar este problema, en caso del seguimiento de señales de satélites, no hay que utilizar la velocidad, sino la tasa a la cual el rango de inclinación del satélite cambia. Tras aplicar una nueva fórmula, con esta tasa, la frecuencia resultante es la correcta, obteniendo una mejor calidad de señal en recepción, al recibir en la frecuencia correcta.

El siguiente fragmento de código es el utilizado para calcular la frecuencia, aplicando el efecto Doppler:

```
int freq = (1 – (sat->range_rate/299792.458)) * job->frequency; /*
Frecuencia aplicando Doppler. */
```

Siendo 299792.458 la constante de la velocidad de la luz, expresada en kilómetros.

5.2.4 Control de rotores y radio

A partir de los datos calculados, mediante el cálculo de predicción de los parámetros del satélite, de la frecuencia aplicada el efecto Doppler, y resto de datos de interés, obtenidos de SatNOGS, podemos controlar los rotores y la radio.

Para el control de estos dispositivos Hardware, se ha hecho uso de la biblioteca HamLib. Las funciones de HamLib, necesarias para la aplicación, se han reunido en las clases Radio y Rotator, simplificando al máximo el uso de la biblioteca. Más adelante, se detallarán las funciones utilizadas en las clases mencionadas anteriormente.

A continuación, se detalla el código utilizado, de las clases Radio y Rotator, para el control.

```
radio->set_vfo(RIG_VFO_A); /* Establece en la radio, el Oscilador
Variable de Frecuencia (VFO) de trabajo, en este caso el VFO A. */
radio->set_freq((freq_t)freq); /* Establece en la radio, la frecuencia de
sintonización, para el VFO de trabajo establecido en la línea anterior. */
rotator->set_position(sat.az, sat.el); /* Envía la posición de elevación y
azimuth al controlador de rotores, para que se orienten en la posición. */
```

5.3 Adquisición de datos

El objetivo, durante la observación de un satélite, es la adquisión de los datos que son envíados vía radio, por el satélite. La aplicación de control, desarrollada para la estación terrena GS-UVa, dispone de una conexión de

audio, entre el PC y la radio, permitiendo la grabación de audio por parte del PC.

Los datos adquiridos, durante un seguimiento, son transferidos a la red de SatNOGS, explicado anteriormente.

5.3.1 Grabación de audio

Para la grabación de audio vamos a utilizar dos bibliotecas de funciones, PortAudio, que proporciona las funciones necesarias para la adquisición de audio, a través de la tarjeta de sonido, y la biblioteca libSNDFile que nos provee de las funciones necesarias para la escritura de las muestras de audio, grabadas, en un fichero de audio, con la codificación especificada, en este caso Vorbis OGG.

El código para la inicialización, inicio de grabación y finalización es el siguiente:

```
SNDFILE * outfile; /* Puntero para manejar el fichero de audio de salida. */
PaStreamParameters inputParameters; /* Estructura de datos, para la especificación de los parámetros de adquisión de audio. */
SF_INFO * sfinfo; /* Estructura de datos, con la información del fichero de audio a guardar/abrir. */
PaStream * stream; /* Puntero del manejador de la adquisición de audio. */
PaError retval  = 0; /* Variable que almacena el código de error devuelto por PortAudio, inicializada a cero. */
sfinfo.channels = 2; /* Especifica el número de canales de audio, para el fichero de salida. En este caso 2 canales de audio. */
sfinfo.samplerate = 44100; /* Especifica la tasa de muestreo de audio, para el fichero de salida. En este caso 44100 Hz. */
sfinfo.format = SF_FORMAT_OGG | SF_FORMAT_VORBIS: /* Especifica el códec, con el que se codifica el fichero de audio. En este caso VORBIS OGG. */
```

```
    /* Apertura del fichero de audio, en modo escritura y con la información
del fichero de audio, mediante sfinfo. */
    if(!(outfile=sf_open(audio_file, SFM_WRITE, &sfinfo)){
        printf("Not able to open output file %s.\n", audio_file);
        sf_perror(NULL); /* Devuelve el error. */
    }

    retval = Pa_Initialize(); /* Inicializa la biblioteca PortAudio. */
    inputParameters.device = Pa_GetDefaultInputDevice(); /* Selecciona el
disposiivo de entrada por defecto. */
    inputParameters.channelCount = 2; /* Selecciona 2 canales de audio de
entrada. */
    inputParameters.sampleFormat = paFloat32; /* Determina el formato de
muestreo, en este caso en coma flotante de 32 bits. */
    inputParameters.suggestLatency =
Pa_GetDeviceInfo(inputParameters.device)->defaultLowOutputLatency; /*
Sugiere una latencia baja, por defecto. */
    inputParameters.hostApiSpecificStreamInfo = NULL; /* Selecciona una
API específica, es este caso no se utiliza una API especifica. */
    retval = Pa_OpenStream(&stream, &inputParameters, NULL, 44100,
4096, paClipOff, paLibsndfileCb, outfile); /* Comienza la adquisión de audio, a
44100 Hz, sin Clipping. */

    retval = Pa_StopStream(stream); /* Detiene la adquisición de audio. */
    retval = Pa_CloseStream(stream); /* Cierra y libera el dispositivo de
audio de entrada. */
    sf_close(outfile); /* Cierra el fichero de audio, donde se ha guardado las
muestras de audio codificadas. */
    Pa_Terminate(); /* Libera los recursos de PortAudio. */
```

Las muestras, de la adquisición de audio, se reciben a través de una función de callback, el siguiente código, muestra la función de callback utilizada para la recepción de la muestras de audio, y escritura en el fichero.

```
int paLibsndfileCb(const void * inputBuffer, void * outputBuffer, unsigned
long framesPerBuffer, const PaStreamCallbackTimeInfo * timeInfo,
paStreamCallbackFlags statusFlags, void * userData)
{
        float * in = (float *)inputBuffer; /* Buffer de entrada */
        long writecount = 0;
        SNDFILE * outfile = (SNDFILE *)userData;
        writecount = sf_write_float(outfile, in, framesPerBuffer * 2); /*
Escritura en el fichero de audio. */
        /* Final de fichero. */
        if(writecount <= 0) {
                printf("No se puede escribir en el fichero.\n");
                return paComplete; /* Gabación completada. */
        }
        return paContinue: /* Continua la grabación. */
}
```

5.4 Programador de tareas

Los pases de satélites, se producen en instantes de tiempo determinados. La aplicación obtiene desde SatNOGS la planificación de satélites a seguir. Los pases, de cada satélite, son necesarios programarlos, para que se ponga en funcionamiento las rutinas de seguimiento. Para que se pueda poner en funcionamiento, en un instante de tiempo determinado, es necesario disponer de un programador de tareas. El programador de tareas, debe ser capaz de ejecutar las tareas programadas, y consumir el menor número de recursos, durante el tiempo de espera.

El programador de tareas se compone de un multimapa (std::multimap) de datos, que almacena dos tipos de datos, time_t que es el instante de tiempo en el que se ejecutará la tarea, y task_t con los datos de la tarea a ejecutar.

La clase, tras su inicialización, ejecuta un thread (hilo de ejecución), únicamente destinado al planificador. Este thread comprueba la primera tarea a realizar, dentro del multimapa. Tras la comprobación, pone el thread a dormir, para que no consuma recursos del procesador, hasta que tenga que ejecutar la tarea.

El método de inserción de tarea, añade un nuevo elemento al multimapa, con la nueva tarea a ejecutar, además despierta al thread de gestión de tareas

(mediante una variable condicional), para así comprobar de nuevo, la primera tarea a ejecutar. Tras esta comprobación, el thread vuelve al estado de dormir, hasta la ejecución de la primera tarea a ejecutar.

5.4.1 Constructor e inicializador

```
    pthread_mutex_init(&lock, NULL); /* Inicializa el mutex de bloqueo multithread. */
    pthread_mutex_init(&sleep_mtx, NULL); /* Inicializa el mutex, utilizado por la variable condicional. */
    pthread_cond_init(&cv, NULL); /* Inicializa la variable condicional. */
    interrupted = 0; /* Variable de estado de interrupción, por la variable condicional, incialmente 0. */
    finalized = 0; /* Variable de estado de finalización del bucle principal. */
    pthread_create(&schedule_thread, NULL, Scheduler::scheduleThread, (void *)this); /* Crea el thread de gestión de tareas, pasa como parámetros la propia clase. */
```

5.4.2 Métodos de programación de nuevas tareas

```
    /* Método para añadir una tarea, con el parámetro de instante de tiempo en tipo time_t. */
    void Scheduler::in(time_t time, task_t task)
    {
        task.interval = 0; /* Establece que la tarea no es de tipo intervalo. */
        add_task(time, task); /* Método para añadir la tarea al multimapa. */
    }

    /* Metodo para añadir una tarea, con el parámetro de los segundos, que tienen que pasar para la ejecución de la tarea. */
    void Scheduler::in(int seconds, task_t task)
    {
        time_t time; /* Variable para almacenar el instante de tiempo. */
```

```
        time = clack_now(); /* Obtiene el instante de tiempo actual. */
        time += seconds; /* Suma los segundos al instante de tiempo actual.
*/
        in(time, task); /* Llama al método par añadir la tarea. */
    }

    /* Método para añadir una tarea de tipo intervalo, con intervalo el número
de segundos entre cada ejecución. */
    void Scheduler::interval(int seconds, task_t task)
    {
        task.interval = 1; /* Configura la tarea como tarea de tipo intevalo. */
        task.interval_value = seconds; /* Asigna el intervalo de tiempo de
cada ejecución. */
        add_task(clock_now, task); /* Método para añadir la tarea al
multimapa, con el instante de ejecución, el instante actual (para la primera
ejecución). */
    }
```

5.4.3 Gestor de tareas

```
    pthread_mutex_lock(&lock); /* Bloquea la ejecución, para otros thread.
*/
    std::multimap<time_t, task_t>::iterator end_task_to_run; /* Variable,
de tipo iterador, que almacena la posición de iteración de la última tarea a
ejecutar. */
    std::multimap<time_t, task_t>::iterator curTask; /* Variable de iteración,
para el recorrido del multimapa. */
    end_task_to_run = tasks.upper_bound(clock_now()); /* Obtiene el
iterador, de la última tarea a ejecutar, para el instante actual. */
    /* Comprueba que el multimapa no esté vacío. */
    while(!tasks.empty())
    {
```

```
        curTask = task.begin(); /* Obtiene el iterador de la primera posición,
del multimapa. */
        /* Comprueba, dentro de la posición actual del multimapa, el primer
elemento, para ver si el instante de tiempo es menor o igual al actual. Si se
cumple la condición, procesa la tarea. */
        if((*curTask).first <= clock_now())
        {
                task_t task = (*curTask).second; /* Obtiene el segundo
elemento (task_t), del elemento actual del multimapa. */
                ptrhead_t thread: /* Crea una variable de thread. */
                pthread_create(&thread, NULL, task-func, (void*)task.args);
/* Crea un nuevo thread, y ejecuta la tarea con los argumentos. */
                /* Comprueba si la tarea es de tipo intervalo, si lo es, vuelve a
añadir la tarea al planificador. */
                if(task.interval == 1){
                        pthread_mutex_unlock(&lock); /* Desbloquea la
ejecución para otro threads. */
                        add_task(clock_now() + task.interval_value, task); /*
Añade la tarea. */
                        pthread_mutex_unlock(&lock); /* Bloquea la ejecución
para otro thread. */
                }
                task.erase(curTask); /* Elimina la tarea actual del planificador.
*/
        } else {
                break; /* Sale del bucle. */
        }
    }
```

5.4.4 Thread principal del gestor de tareas

```
Scheduler * scheduler = (Scheduler *)args; /* Carga el objeto Scheduler, a partir del argumento pasado al crear el thread. */
/* Bucle de ejecucución, se ejecuta infinitamente, hasta que la variable finalized tiene valor 1. */
while (!scheduler->finalized){
    /* Comprueba si el multimapa está vacío, si está vacío pone al thread a dormir, en otro caso pone el thread a dormir, hasta la ejecucuión de la primera tarea. */
    if(scheduler->task.empty()){
        scheduler->sleep();
    } else {
        time_t time_of_first_task = (*scheduler->task.begin()).first;
        scheduler->sleep_until(time_of_first_task);
    }
    Scheduler->manage_tasks();
}
return (void*)NULL; /* Retorna un puntero vacío y nulo. */
```

5.4.5 Control de la variable de condición

```
/* Método para dormir el thread, durante los segundos que se especifiquen. */
void Scheduler::sleep_for(int seconds)
{
    struct timespec sleep_time; /* Estructura para el temporizador de tiempo. */
    pthread_mutex_lock(&sleep_mtx); /* Bloqueo. Mutex asociado a la variable de condición. */
    sleep_time.tv_sec = time(NULL) + seconds; /* Añade los segundos, al instante de tiempo actual en segundos. */
```

```
        sleep_time.tv_nsec = 0; /* Nanosegundos a esperar, en este caso cero.
*/
        pthread_cond_timedwait(&cv, &sleep_mtx, &sleep_time); /* Pone en
funcionamiento la variable de condición, se desactiva mediante señal o
vencimiento del temporizador. */
        interrupted = 0; /* Establece el estado de interrupción a cero. */
        pthread_mutex_unlock(&sleep_mtx); /* Debloqueo. Mutex asociado a
la variable de condición. */
    }

    /* Método para dormir el thread, hasta el instante de tiempo especificado.
*/
    void Scheduler::sleep_until(time_t time_s)
    {
        int seconds; /* Variable para almacenar la diferencia de segundos. */
        struct timespec sleep_time; /* Estructura para el temporizador de
tiempo. */
        pthread_mutex_lock(&sleep_mtx); /* Bloqueo. Mutex asociado a la
variable de condición. */
        seconds = time_s - clock_now(); /* Realiza la diferencia de segundos
entre el instante de tiempo que debe despertar y el instante actual, en UTC. */
        sleep_time.tv_sec = time(NULL) + seconds; /* Añade los segundos, al
instante de tiempo actual en segundos. */
        sleep_time.tv_nsec = 0; /* Nanosegundos a esperar, en este caso cero.
*/
        pthread_cond_timedwait(&cv, &sleep_mtx, &sleep_time); /* Pone en
funcionamiento la variable de condición, se desactiva mediante señal o
vencimiento del temporizador. */
        interrupted = 0; /* Establece el estado de interrupción a cero. */
        pthread_mutex_unlock(&sleep_mtx); /* Debloqueo. Mutex asociado a
la variable de condición. */
    }
```

```
    /* Método para dormir el thread, hasta recibir la señal de despertar. */
    void Scheduler::sleep ()
    {
        pthread_mutex_lock(&sleep_mtx); /* Bloqueo. Mutex asociado a la
variable de condición. */
        pthread_cond_wait(&cv, &sleep_mtx); /* Pone en funcionamiento la
variable de condición, se desactiva mediante. */
        interrupted = 0; /* Establece el estado de interrupción a cero. */
        pthread_mutex_unlock(&sleep_mtx); /* Debloqueo. Mutex asociado a
la variable de condición. */
    }

    /* Método para interrumpir y desactivar la variable de condición. */
    void Scheduler::interrupt()
    {
        pthread_mutex_lock(&sleep_mtx); /* Bloqueo. Mutex asociado a la
variable de condición. */
        interrupted = 1; /* Establece la variable de interrupción a uno. */
        pthread_cond_signal(&cv); /* Genera una señal de desbloqueo para la
variable de condición. */
        pthread_mutex_unlock(&sleep_mtx); /* Debloqueo. Mutex asociado a
la variable de condición. */
```

5.4.6 Método para añadir la tarea al multimapa

```
    void Scheduler::add_task(time_t time, task_t task)
    {
        pthread_mutex_lock(&lock); /* Bloquea la ejecución para otros
threads. */
```

```
        tasks.insert(std::pair<time_t, task_t>(time, task)); /* Añade el
instante de tiempo y la tarea al multimapa, mediante un tipo pair, de la
biblioteca estándar de C++. */
        interrupt(); /* Genera una interrupción, para que el thread de gestión
de tareas despierte. */
        pthread_mutex_unlock(&lock); /* Desbloquea la ejecución para otros
threads. */
    }
```

5.5 Interfaz gráfica de usuario (GUI)

La aplicación desarrollada, para la estación terrena GS-UVa, dispone de una interfaz gráfica de usuario. Esta interfaz gráfica, muestra datos del satélite en seguimiento, en formato texto, y un plano polar de posición. También muestra información sobre el estado de la radio y de los rotores, mostrando la frecuencia de operación, modo; y posiciones de azimuth y elevación. También dispone de un listado, donde se muestra las tareas de seguimiento programadas.

Para la generación de la interfaz gráfica, se han utilizado las librerías GTK+ y Goocanvas, con los conectores gtkmm y goocanvasmm, respectivamente.

La documentación necesaria para el desarrollo de la interfaz gráfica, con gtkmm y goocanvasmm, se puede encontrar en [29] y [30].

5.5.1 Dialogo de aviso de necesidad de permisos de superusuario

La aplicación, para su funcionamiento, necesita disponer de permisos de superusuario. Sin estos permisos, el sistema no nos permite acceder al hardware del sistema, como son los puertos de comunicación serie, necesarios para el control de radio y rotores.

Al inicio de la aplicación, se comprueba que se haya iniciado con permisos de superusuario (UID = 0), si no es así, se muestra un mensaje de error y la aplicación se cierra. En la , podemos ver el dialogo de error.

El Widget utilizado, es MessageDialog, que permite generar mensajes emergentes, con botones.

```
/* Comprueba el UID de ejecución del programa, si no es cero, lanza un aviso y cierra la aplicación. */
if(getuid != 0)
{
    Gtk::MessageDialog dialog("Root privileges are required for running this program", false, Gtk::MESSAGE_ERROR, Gtk::BUTTONS_OK); /* Genera el mensaje emergente, de tipo error, botón "OK", y únicamente el título. */
    dialog.set_secondary_text("GS UVa Manager require root privileges for manage COM serial (RS-232) ports."); /* Establece el mensaje que se muestra dentro del dialogo. */
    dialog.run() /* Muestra el mensaje emergente. */
    exit(0); /* Finaliza la aplicación con código cero. */
}
```

5.5.2 Inicio de la aplicación y ventana principal

La aplicación dispone de una única ventana principal de operaciones, en esta ventana se muestra un menú contextual, un toolbar y el resto de Widgets de información.

En la Figura 29, podemos observar el aspecto que tiene la ventana principal de la aplicación.

Figura 29. Mensaje emergente de error por privilegios de usuario

Código de inicio de la aplicación:

```
Gtk::Main kit(argc, argv, false); /* Inicializa la biblioteca GTK+, se le pasan los argumentos de entrada, y se desactiva la utilización de la configuración regional de idioma (LOCALE). */
Win_ GSManager main_ window; /* Variable del objeto de la ventana principal. */
Gtk::Main::run(main_ window); /* Lanza la interfaz gráfica y muestra como punto de partida, la ventana principal. */
```

La clase Win_GSManager, es una clase heradada del Widget Gtk::Window. A continuación se muestra el código, para configurar el título, añadir otros wigets a la ventana y otras configuraciones de interés.

```
this->set_ title("GS UVA Manager"); /* Configura el título de la ventana. */
this->set_ default_ size(775,500); /* Configura el tamaño de la ventana por defecto, 775x500 pixels. */

/* Bloque de prueba, por si falla, captura la excepción. */
try
{
    this->set_ default_ icon_ name("icon"); /* Selecciona el nombre, del icono, por defecto. */
}
catch(Glib::Exception &e)
{
```

```
    /* Código a ejecutar, en caso de excepción. */
}

this->add(widget_var); /* Añade un widget a la ventana. */
this->show_all_children(); /* Mestra todos los widgets hijos. */
```

5.5.3 Cajas de contenido – VBox y HBox

Los widgets, que se añaden a las ventanas, necesitan disponer de un contenedor, que asegure el orden, almacenamiento y posición. Para esta labor, existen cajas "Box", que cumplen estas características. Además, existen widgets de cajas, que facilitan la labor de ordenación, para añadir widgets en orden vertical u horizontal, estos widgets son VBox y HBox, que diponen los widgets hijos en orden vertical y horizontal, respectivamente.

El modo de funcionamiento de ambos widgets son similares, lo único que cambia es la gestión interna de ordenación.

En el siguiente código, se muestran los métodos básicos de uso de uno de estos widgets.

```
Gtk::HBox * hbox_main = Gtk::manage(new HBox()); /* Crea un nuevo objeto HBox, con el control de memoria dinámica por la aplicación, utilizará el recolector de basura. */
hbox_main->pack_start(widget_var, Gtk::PACK_SHRINK); /* Añade el widget hijo a la caja, y en el empaquetado, en la caja, se contrae al tamaño del widget hijo. */
hbox_main->pack_start(widget_var2, TRUE, TRUE, 0); /* Añade el widget hijo a la caja, permite que se expanda (segundo argumento) y se rellene (tercer argumento), y el padding sea cero (cuarto argumento). */
```

5.5.4 Cambios en la interfaz gráfica, desde otros thread

Dependiendo el tipo de tareas que realice la aplicación, puede ser necesario, actualizar el contenido de algún widget, desde un thread diferente al principal que creó la interfaz gráfica.

Los cambios y actualizaciones de la interfaz gráfica no están permitidos, desde otros threads diferentes, al thread que creó la interfaz gráfica. Esta característica de las interfaces gráficas son un inconveniente, cuando existe la necesidad de realizar estos cambios o actualizaciones.

La biblioteca GTK+, y a través de gtkmm, dispone de una clase "Dispatcher", que permite recibir señales de otros threads, para la actualización de la interfaz gráfica. Así, los cambios que sean necesarios realizar, son siempre hechos con el thread principal, que creó la interfaz gráfica.

En la aplicación de gestión de la estación terrena, es necesario utilizar este "Dispatcher", para realizar cambios, desde el thread de seguimiento de satélites. Los datos calculados por el thread de seguimiento, deben ser actualizados en los widgets correspondientes a la muestra de información, del satélite en seguimiento.

El siguiente código muestra como se inicia y llama al "Dispatcher", así como la función de callback, para la ejecución de la actualización de la interfaz gráfica.

```
Glib::Dispatcher m_Dispatcher_UpdateSatInfo; /* Variable del
Dispatcher. */
M_Disparcher_UpdateSatInfo.connect(sigc::mem_fun(*this,
&Win_GSManager::on_notification_update_sat_info)); /* Configura la
función de callback, que será llamada cuando se envíe una señal al Dispatcher.
*/


/* Función de callback, llamada por el Dispatcher. */
void Win_GSManager::on_notification_update_sat_info()
{
    sat_info->update(); /* Invoca al método update(), del widget
personalizado SingleSatInfo. */
```

```
        polar_view->update(); /* Invoca al método update(), del widget personalizado PolarView. */
    }

    /* Lanzamiento de la señal al Dispatcher, desde otro thread, distinto al principal. */
    main_window->m_Dispatcher_UpdateSatInfo.emit();
```

5.5.5 Temporizadores

En ocasiones, pueder ser necesario, actualizar la información de algún widget, cada cierto tiempo. Para poder realizar estas actualizaciones, son necesarios temporizadores, que ejecuten un bloque de código al vencer el temporizador.

La biblioteca GTK+, a través de gtkmm, dispone de un temporizador, que nos ayudará a ejecutar código, para la actualización de widgets de la interfaz gráfica.

En la aplicación, se utilizan los temporizadores, para actualizar los widgets de estado de radio y rotores.

En el siguiente código, se muestra la inicialización del temporizador, así como la función de callback, que ejecutará el código tras vencer el temporizador.

```
    sigc::connection con = Glib::signal_timeout().connect(sigc::mem_fun(*this, &Win_GSManager::on_timeout_update_radio_info), 5000); /* Configura el timeout, para que se ejecute cada cinco segundos, y establece a la función de callback que debe llamar al vencer el temporizador. */

    /* Función de callback llamada al vencer el temporizador. */
    bool Win_GSManager::on:timeout_update_radio_info()
    {
        radio_info->update(); /* Invoca el método update, del widget personalizado RadioInfo. */
```

```
        rotator_info->update(); /* Invoca el método update, del widget personalizado RotatorInfo. */
        job_view->update(); /* Invoca el método update, del widget personalizado JobView. */
        return true; /* Necesario devolver true, para que el temporizador se reinicie. */
    }
```

5.5.6 Creación de un menú contextual

Muchas aplicaciones que se desarrollan, disponen de un menú contextual, en la parte superior de la aplicación, que disponen de herramientas para el uso en la aplicación.

La biblioteca de creación de interfaces gráficas, dispone de los widgets necesarios, para la creación y administración de un menú contextual, estos widget son Menubar, Menu y MenuItem.

En la aplicación, se ha creado un menú contextual, que servirá para colocar las acciones más importantes a realizar. Podemos ver en la Figura 30, el aspecto del menú contextual de la aplicación.

Figura 30. Ejemplo de menú contextual hecho con GTK+

El código necesario para la creación de un menú contextual es el siguiente:

```
    Gtk::Menubar menubar_main; /* Variable para la barra de menús. */
    Gtk::Menu * menu; /* Variable para el widget Menú. */
    Gtk::MenuItem * menu_item; /* Variable para los elementos que incluye el menú. */

    menu = manage(new Gtk::Menu()); /* Crea un nuevo objeto del widget Menu. */
```

```
menu_item = manage(new Gtk::MenuItem()); /* Crea un nuevo objeto, ítem, para el menú. */
menu_item->set_label("Nuevo"); /* Configura la etiqueta del elemento. */
menu_item->signal_activate().connect(sigc::mem:fun(*this, &Win_GSManager::on_activate)); /* Configura la señal de activación del elemento, y el callback asociado. */
menu->append(*menú_item); /* Añade el elemento al menú, no puede ser un puntero. */

menu_item = manage(new Gtk::MenuItem()); /* Crea un nuevo objeto ítem, para la barra de menús. */
menu_item->set_label("Archivo"); /* Configura la etiqueta del elemento. */
menú_item->set_submenu(*menú); /* Configura el submenú asociado al elemento, no puede ser puntero. */
menubar_main.append(*menú_item); /* Añade el elemento a la barra de menús, no puede ser puntero. */
```

5.5.7 Barra de herramientas

Las barras de herramientas, son un elemento muy útil, basado en iconos. Estas barras de herramientas, permite disponer de elementos elementos rápidos, para acciones rápidas.

La biblioteca gráfica, dispone de widgets para la realización de barras de herramientas, mediante los widgets HBox, Toolbar, ToolButton, Image.

La aplicación de gestión de la estación GS UVa, dispone de una barra de herramientas, con un botón para el control de la polarización de las antenas. En un futuro, es posible que disponga de más acciones, como la conmutación coaxial, entre la antena de 144 – 146 MHz y la antena de 2400 MHz.

En la Figura 31, podemos observar un ejemplo de barra de herramientas, realizada con GTK+.

Figura 31. Barra de herramientas, realizada con GTK+

El código para la creación de la barra de herramientas es el siguiente:

```
Gtk::HBox hbox_toolbar; /* Variable para el widget HBox principal, del toolbar. */
Gtk::Image * image; /* Variable para las imágenes del toolbar. */
Gtk::ToolButton * toolbutton; /* Variable para los botones del toolbar. */
Gtk::ToolBar toolbar_main; /* Variable para el toolbar. */

hbox_toolbar.pack_start(toolbar_main); /* Añade la barra de herramientas al contenedor HBox. */

/* Creación de un botón de acción para el toolbar. */
image = manage(new Gtk::Image(Gtk::Stock::NEW, Gtk::ICON_SIZE_BUTTON)); /* Crea la imagen del botón. */
Glib::ustring str_temp = "Action"; /* Variable de cadena de texto, para el texto auxiliar del botón. */
toolbutton = manage(new Gtk::ToolButton(*image, str_temp)); /* Crea un botón para el toolbox, con la imagen (no puntero) y el texto auxiliar. */
toolbar_main.append(*toolbutton); /* Añade el botón a la barra, sin puntero. */
toolbutton->set_tooltip_text("Texto de ayuda"); /* Añade un texto de ayuda al botón. */
toolbutton->set_sensitive(true); /* Activa el botón, por defecto está activo. */

/* Añadido de un separador en la barra de herramientas. */
toolbar_main.append(*(Gtk::manage(new Gtk::SeparatorToolItem))); /* Añade el separado a la barra de herramientas. */
```

5.5.8 Widget SingleSatInfo

El widget SingleSatInfo, es un widget personalizado, creado únicamente para la aplicación. Este widget, muestra información sobre el satélite en seguimiento.

Entre los datos que nos muestra, los datos de alta importancia son el azimuth, elevación, rango de inclinación, próximo LOS (Pérdida de cobertura), pérdidas de señal, retardo de la señal, velocidad a la que se desplaza, cobertura, longitud y latitud.

Para la realización de este widget, se ha hecho a partir de la herencia del widget VBox. Se le ha añadido un widget Label, para la cabecera y un widget Grid para almacenar los datos. En el Grid, se han añadido widgets Label, para escribir la información de cada uno de los ítem que se muestran en pantalla.

Los datos necesarios para mostrar sobre el satélite en seguimiento, son obtenidos mediante un puntero a una estructura sat_t, otro puntero a una estructura qth_t y un array de caracteres, para el título del satélite. La actualización de los datos se realiza mediante el método update().

En la Figura 32, podemos observar los datos sobre el satélite en seguimiento.

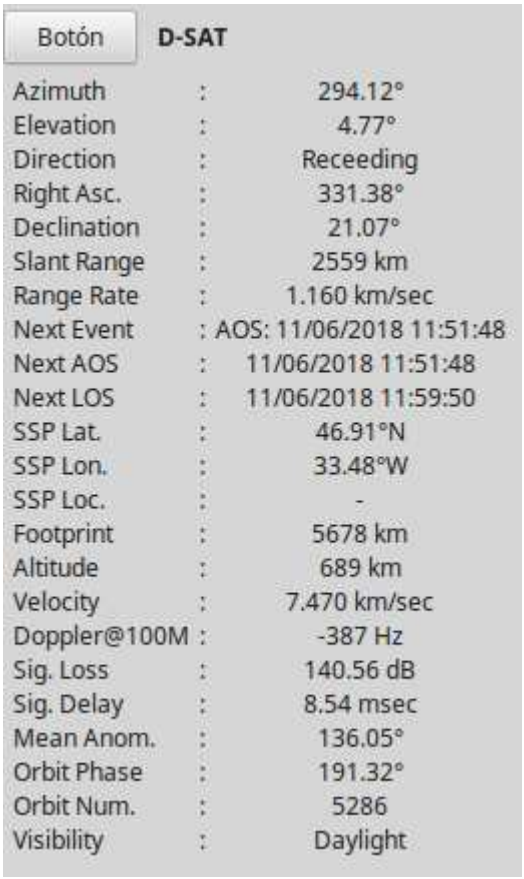


Figura 32. Widget SingleSatInfo

A continuación, se muestra el código de creación e inicialización del widget.

```
Gtk::Label * header; /* Etiqueta para el título del satélite. */
Gtk::Label * labels[SINGLE_SAT_FIELDS_NUMBER]; /* Array de etiquetas para almacenar y mostrar la información del satélite en seguimiento. La dimensión del array, viene determinada por SINGLE_SAT_FIELDS_NUMBER. */
Gtk::Grid * table; /* Tabla donde se situan las etiquetas que muestran la información. */
qth_t * qth; /* Puntero a la estructura de datos de las coordenadas del observador. */
sat_t * sat; /* Puntero a la estructura de datos de información del satélite.*/
```

```
    char * sat_name; /* Puntero a un array de caracteres que dispone del
nombre del satélite. */
    Gtk::Label * label1; /* Widget Label, para el título de los campos de
información. */
    Gtk::Label * label2; /* Widget Label, para el valor de los campos de
información. */

    /* Header Label */
    header = Gtk::manage(new Gtk::Label()); /* Nuevo objeto Label, para el
título del satélite. */
    header->set_markup("<b>No tracking satellite</b>"); /* Configura el
contenido de la etiqueta, mediante etiquetado (HTML). En este caso el valor
en negrita. */
    header->set_xalign(0.0f); /* Alineación horizontal. */
    header->set_yaling(0.5f); /* Alineación vertical. */
    pack_start(*header, true, true, 10); /* Añade la etiqueta al contenedo, con
un padding de valor 10. */

    /* Grid table satellite information. */
    table = Gtk::manage(new Gtk::Grid()); /* Nueva tabla. */
    table->set_border_width(5); /* Configura el ancho del borde. */
    table->set_row_spacing(0); /* Configura el espaciado entre filas. */
    table->set_column_spacing(5); /* Configura el espaciado entre columnas.
*/

    /* Create and add information labels. */
    for(int i = 0; i<SINGLE_SAT_FIELD_NUMBER; i++)
    {
        label1 = Gtk::manage(new
Gtk::Label(SINGLE_SAT_FIELD_TITLE[i])); /* Crea una nueva instancia
de Label, con el valor, el título del campo i. */
    label1->set_xalign(1.0f); /* Alineación horizontal. */
```

```
    label1->set_yalign(0.5f); /* Alineación vertical. */
    table->attach(*label1, 0, i, 1, 1); /* Añade la etiqueta a la tabla, a la
columna 1, fila i. */

    label2 = Gtk::manage(new Gtk::Label("-")); /* Crea una nueva instancia de
Label, con el valor '-'. */
    label2->set_xalign(0.0f); /* Alineación horizontal. */
    label2->set_yalign(0.5f); /* Alineación vertical. */
    table->attach(*label2, 2, i, 1, 1); /* Añade la etiqueta a la tabla, a la
columna 3, fila i. */
    labels[i] = label2; /* Asigna la etiqueta al array de etiquetas, en la posición
i. */

    label1 = Gtk::manage(new Gtk::Label(":")); /* Crea una nueva instancia de
Label, con el valor ':'. */
    table->attach(*label1, 1, i, 1, 1); /* Añade la etiqueta a la tabla, a la
columna 2, fila i. */
    }

    pack_start(*table, Gtk::PACK_SHRINK); /* Añade la tabla, con su
contenido, al contenedor. */
```

5.5.9 Widget RadioInfo

El widget personalizado RadioInfo, obtiene y muestra información del estado de la radio. Los datos que podemos visualizar es el VFO (Oscilador variable de frecuencia) seleccionado, si está disponible, el modo de operación (Modulación) y la frecuencia de sintonización en MHz.

El widget es un derivado, mediante herencia, de un contenedor VBox. Utiliza los widgets Label y Grid.

En la Figura 33, podemos ver un ejemplo de información mostrada por el widget.

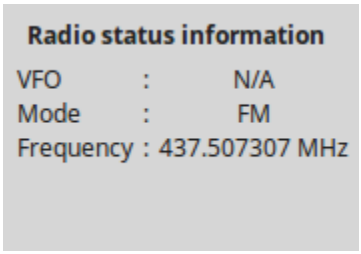


Figura 33. Widget RadioInfo

En las siguientes líneas se muestra el código de creación e inicialización del widget.

```
Gtk::Label * header; /* Etiqueta para mostrar el título del widget. */
Gtk::Label * labels[RADIO_FIELD_NUMBER]; /* Array de etiquetas, para mostrar la información de estado de la radio. */
Gtk::Grid * table; /* Tabla para la muestra de información. */

/* Header Label */
header = gtk::manage(new Gtk::Label()); /* Crea la etiqueta. */
header->set_markup("<b>Radio status information</b>"); /* Configura el valor de la etiqueta con lenguaje de marcado, en este caso se muestra el título en negrita. */
header->set_xalign(0.0f); /* Configura la alineación horizontal. */
header->set_yalign(0.5f); /* Configura la alineación vertical. */
pack_start(*header, true, true, 10); /* Añade la etiqueta al contenedor, con un padding de valor 10. */

/* Grid table for radio information. */
table = Gtk::manage(new Gtk::Grid()); /* Nuevo objeto Grid. */
table->set_border_width(5); /* Configura el ancho de borde. */
table->set_row_spacing(0); /* Configura el espaciado de filas. */
table->set_column_spacing(5); /* Configura el espaciado de columnas. */

/* Create and add information labels to grid. */
```

```
for(int i = 0; i < RADIO_FIELD_NUMBER; i++)
{
    label1 = Gtk::manage(new Label(RADIO_FIELD_TITLE[i])); /* Crea una nueva etiqueta, con el título del campo de información i. */
    label1->set_xalign(1.0f); /* Configura la alineación horizontal. */
    label1->set_yalign(0.5f); /* Configura la alineación vertical. */
    table->attach(*label1, 0, i, 1, 1); /* Añade la etiqueta a la tabla, en la columna 1 y fila i. */

    label2 = Gtk::manage(new Label("-")); /* Crea una nueva etiqueta, con el texto '-'. */
    label2->set_xalign(0.0f); /* Configura la alineación horizontal. */
    label2->set_yalign(0.5f); /* Configura la alineación vertical. */
    table->attach(*label2, 2, i, 1, 1); /* Añade la etiqueta a la tabla, en la columna 3 y fila i. */
    labels[i] = label2; /* Asigna la etiqueta al array de etiquetas, en la posición i. */

    label1 = Gtk::manage(new Label(":")); /* Crea una nueva etiqueta, con el texto ':'. */
    table->attach(*label1, 1, i, 1, 1); /* Añade la etiqueta a la tabla, en la columna 2 y fila i. */
}

pack_start(*table, Gtk::PACK_SHRINK); /* Añade la tabla, con todo el contenido, al contenedor. */
```

Para la obtención de los datos, de estado, de la radio, se hacen uso de los siguientes métodos de la clase Radio:

- get_vfo(); /* Obtiene el valor del VFO en uso actual. */
- get_mode(); /* Obtiene el valor del modo de utilización (Modulación), actual de la radio. */

- get_freq(); /* Obtiene el valor actual, para el VFO actual, de la frecuencia de sintonización en Hz. */

5.5.10 Widget RotatorInfo

El widget personalizado RotatorInfo, obtiene y muestra información del estado de los rotores. Los datos que podemos visualizar son el azimuth y la elevación, en grados.

El widget es un derivado, mediante herencia, de un contenedor VBox. Utiliza los widgets Label y Grid.

En la Figura 34, podemos ver un ejemplo de información mostrada por el widget.

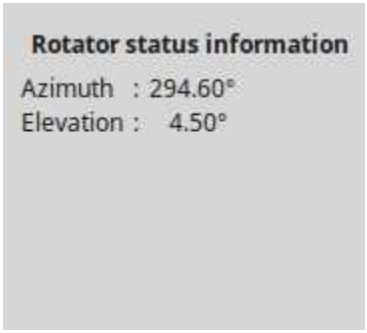


Figura 34. Widget RotatorInfo

En las siguientes líneas se muestra el código de creación e inicialización del widget.

```
Gtk::Label * header; /* Etiqueta para mostrar el título del widget. */
Gtk::Label * labels[ROTATOR_FIELD_NUMBER]; /* Array de etiquetas, para mostrar la información de estado de los rotores. */
Gtk::Grid * table; /* Tabla para la muestra de información. */

/* Header Label */
header = gtk::manage(new Gtk::Label()); /* Crea la etiqueta. */
header->set_markup("<b>Rotator status information</b>"); /* Configura el valor de la etiqueta con lenguaje de marcado, en este caso se muestra el título en negrita. */
```

```
    header->set_xalign(0.0f); /* Configura la alineación horizontal. */
    header->set_yalign(0.5f); /* Configura la alineación vertical. */
    pack_start(*header, true, true, 10); /* Añade la etiqueta al contenedor,
con un padding de valor 10. */

    /* Grid table for rotator information. */
    table = Gtk::manage(new Gtk::Grid()); /* Nuevo objeto Grid. */
    table->set_border_width(5); /* Configura el ancho de borde. */
    table->set_row_spacing(0); /* Configura el espaciado de filas. */
    table->set_column_spacing(5); /* Configura el espaciado de columnas. */

    /* Create and add information labels to grid. */
    for(int i = 0; i < ROTATOR_FIELD_NUMBER; i++)
    {
        label1 = Gtk::manage(new Label(ROTATOR_FIELD_TITLE[i])); /*
Crea una nueva etiqueta, con el título del campo de información i. */
        label1->set_xalign(1.0f); /* Configura la alineación horizontal. */
        label1->set_yalign(0.5f); /* Configura la alineación vertical. */
        table->attach(*label1, 0, i, 1, 1); /* Añade la etiqueta a la tabla, en la
columna 1 y fila i. */

        label2 = Gtk::manage(new Label("-")); /* Crea una nueva etiqueta, con
el texto '-'. */
        label2->set_xalign(0.0f); /* Configura la alineación horizontal. */
        label2->set_yalign(0.5f); /* Configura la alineación vertical. */
        table->attach(*label2, 2, i, 1, 1); /* Añade la etiqueta a la tabla, en la
columna 3 y fila i. */
    labels[i] = label2; /* Asigna la etiqueta al array de etiquetas, en la posición
i. */

        label1 = Gtk::manage(new Label(":")); /* Crea una nueva etiqueta, con
el texto ':'. */
```

```
        table->attach(*label1, 1, i, 1, 1); /* Añade la etiqueta a la tabla, en la
columna 2 y fila i. */
    }

    pack_start(*table, Gtk::PACK_SHRINK); /* Añade la tabla, con todo el
contenido, al contenedor. */
```

Para la obtención de los datos, de estado, de la radio, se hacen uso de los siguientes métodos de la clase Rotator:

- get_position(); /* Obtiene los valores actuales, del azimuth y elevación. */

5.5.11 Widget PolarView

El widget PolarView, es un derivado del widget Canvas, de la biblioteca Goocanvas. Este widget dibuja un plano polar, Norte – Sur – Este – Oeste, con curvas de elevación, entre cero y 90^{o}.

El objetivo de este widget, es mostrar la posición en el plano polar, del satélite en seguimiento, para que el usuario de la estación conozca en todo momento la posición del satélite, de manera visual.

En la Figura 35, podemos observar el plano polar, con un satélite en seguimiento.

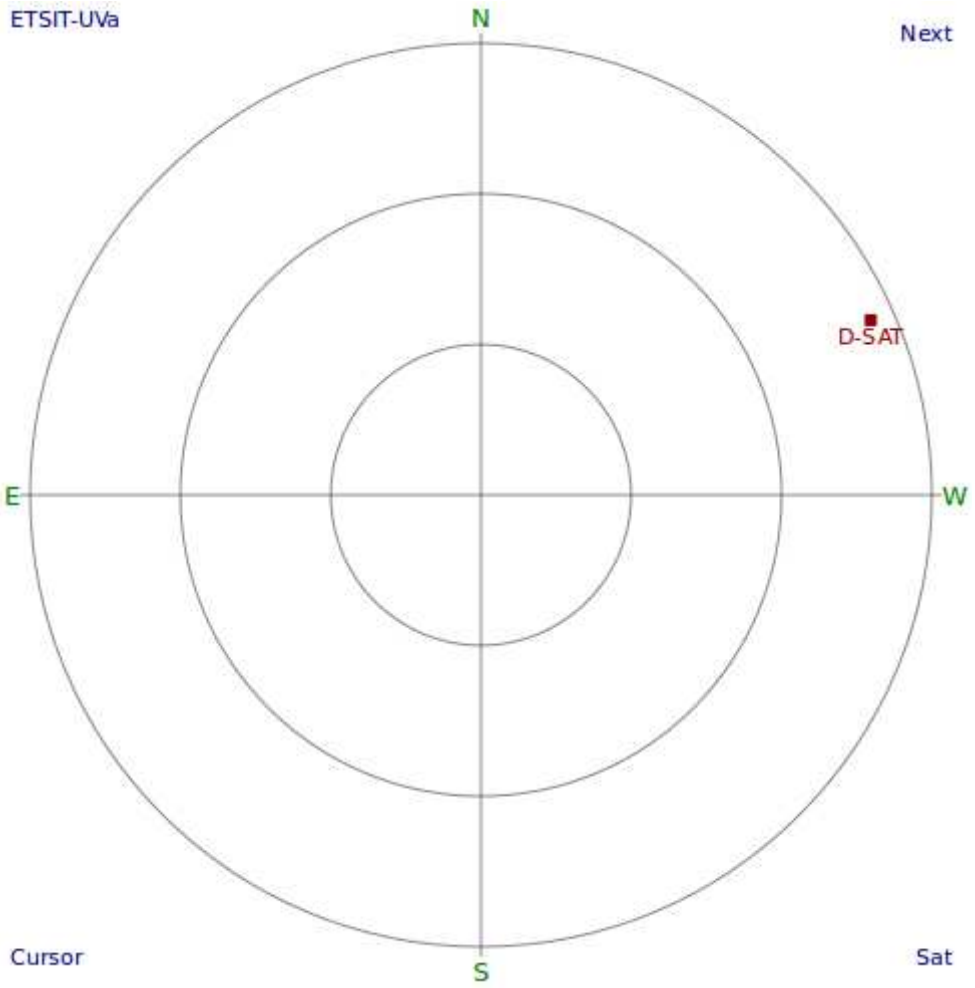


Figura 35. Widget PolarView

El código para la creación e inicialización del widget es el siguiente:

```
Glib::RefPtr<Goocanvas::RectModel> bgd; /* Cuadrado de fondo. */
Glib::RefPtr<Goocanvas::EllipseModel> C00, C30, C60;      /* Círculos de elevación para 0, 30 y 60 grados. */
Glib::RefPtr<Goocanvas::PolylineModel> hl, vl;      /* líneas horizontales y verticales. */
Glib::RefPtr<Goocanvas::TextModel> N, S, E, W;  /*  Etiqueta Norte, Sur, Este y Oeste. */
Glib::RefPtr<Goocanvas::TextModel> locnam; /* Nombre de la localización. */
```

```
    Glib::RefPtr<Goocanvas::TextModel> curs;   /* Texto del cursor en
seguimiento. */
    Glib::RefPtr<Goocanvas::TextModel> next;   /* Texto del próximo evento
*/
    Glib::RefPtr<Goocanvas::TextModel> sel;    /* Texto de información sobre
el satélite actual en seguimiento. */

    Glib::RefPtr<Goocanvas::RectModel> marker; /* Marcador del satélite en
el plano. */
    Glib::RefPtr<Goocanvas::TextModel> label; /* Etiqueta que acompaña al
marcador del satélite en el plano. */

    qth_t * qth; /* Puntero a la estructura qth_t con la localización del
observador. */
    sat_t * sat; /* Puntero a la estructura sat_t, del satélite en seguimiento.
*/
    polar_view_swap_t swap; /* Estructura para el intercambio de los ejes
polares. */

    /* Constructor */
    Glib::RefPtr<Goocanvas::GroupModel> root; /* Modelo de tipo grupo
principal. */
    qth = NULL; /* Establecemos un valor nulo, para el puntero a la
estructura qth_t. */
    sat = NULL; /* Establecemos un valor nulo, para el puntero a la
estructura sat_t. */
    swap = POLAR_VIEW_NWSE; /* Configura el plano, para una
visualización de los ejes normal. */
    set_size_request(POLV_DEFAULT_SIZE, POLV_DEFAULT_SIZE);
/* Configura el tamaño del Widget al tamaño por defecto, configurado en las
definiciones. */
```

```
    set_bounds(0, 0, POLV_DEFAULT_SIZE, POLV_DEFAULT_SIZE); /*
Configura las esquinas del widget. */
    show(); /* Muestra el widget. */
    root = create_canvas_model(); /* Llama al método de creación del
modelo. */
    set_root_item_model(root); /* Selecciona el modelo de elemento,
principal, utilizado por Canvas. */

    /* Método create_canvas_model() */
    Glib::RefPtr<Goocanvas::GroupModel> root; /* Variable del modelo de
grupo. */
    float x,y; /* Posición x e y. */
    uint32_t col; /*  */
    Goocanvas::AnchorType anch = Goocanvas::ANCHOR_CENTER; /* Tipo
de ancho. */

    root = Goocanvas::GroupModel::créate(): /* Creación del modelo de
Grupo. */

    /* Variables de dimensiones. */
    size = POLV_DEFAULT_SIZE; /* Tamaño. */
    r = (size/2) - POLV_DEFAULT_MARGIN; /* Radio. */
    cx = POLV_DEFAULT_SIZE / 2; /* Coordenada x del centro. */
    cy = POLV_DEFAULT_SIZE / 2; /* Coordenada y del centro. */

    /* Rectángulo de fondo. */
    bgd = Goocanvas::RectModel::créate(0.0, 0.0, POLV_DEFAULT_SIZE);
    bgd->set_property("fill-color-rgba", 0XFFFFFFFF);
    bgd->set_property("stroke-color-rgba", 0XFFFFFFFF);
    root->add_child(bgd); /* Añade el rectángulo al grupo. */

    /* Círculo de elevación 0 grados. */
```

```
    C00 = Goocanvas:EllipseModel::créate(cx, cy, r, r);
    C00->set_property("line-width", 1.0);
    C00->set_property("stroke-color-rgba", 0X0F0F0F7F);
    root->add_child(C00); /* Añade el cículo al grupo. */

    /* Círculo de elevación 30 grados. */
    C30 = Goocanvas:EllipseModel::créate(cx, cy, 0.667 * r, 0.667 * r);
    C30->set_property("line-width", 1.0);
    C30->set_property("stroke-color-rgba", 0X0F0F0F7F);
    root->add_child(C30); /* Añade el cículo al grupo. */

    /* Círculo de elevación 60 grados. */
    C60 = Goocanvas:EllipseModel::créate(cx, cy, 0.333 * r, 0.333 * r);
    C60->set_property("line-width", 1.0);
    C60->set_property("stroke-color-rgba", 0X0F0F0F7F);
    root->add_child(C60); /* Añade el cículo al grupo. */

    /* Línea horizontal. */
    hl = Goocanvas::PolylineModel::créate(cx - r –POLV_LINE_EXTRA, cy,
cx + r + POLV_LINE_EXTRA, cy);
    hl->set_property("line-width", 1.0);
    hl->set_property("stroke-color-rgba", 0X0F0F0F7F);
    root->add_child(hl); /* Añade la línea al grupo. */

    /* Línea vertical. */
    vl = Goocanvas::PolylineModel::créate(cx, cy - r –POLV_LINE_EXTRA,
cx, cy + r + POLV_LINE_EXTRA);
    vl->set_property("line-width", 1.0);
    vl->set_property("stroke-color-rgba", 0X0F0F0F7F);
    root->add_child(vl); /* Añade la línea al grupo. */

    /* Etiqueta N (Norte). */
```

```
    azel_to_xy(0.0, 0.0, &x, &y); /* Conversor de azimuth/elevación a xy. */
    correct_pole_coor(POLAR_VIEW_POLE_N, &x, &y, &anch); /* Corrije
coordenadas polares. */
    N = Goocanvas::TextModel::create("N", x, y, -1, anch);
    N->property_font() = "Sans 10";
    N->set_property("fill-color-rgba", 0X007F00FF);
    root->add_child(N); /* Añade la etiqueta al grupo. */

    /* Etiqueta S (Sur). */
    azel_to_xy(180.0, 0.0, &x, &y); /* Conversor de azimuth/elevación a xy.
*/
    correct_pole_coor(POLAR_VIEW_POLE_S, &x, &y, &anch); /* Corrije
coordenadas polares. */
    S = Goocanvas::TextModel::create("S", x, y, -1, anch);
    S->property_font() = "Sans 10";
    S->set_property("fill-color-rgba", 0X007F00FF);
    root->add_child(S); /* Añade la etiqueta al grupo. */

    /* Etiqueta E (Este). */
    azel_to_xy(90.0, 0.0, &x, &y); /* Conversor de azimuth/elevación a xy. */
    correct_pole_coor(POLAR_VIEW_POLE_E, &x, &y, &anch); /* Corrije
coordenadas polares. */
    E = Goocanvas::TextModel::create("E", x, y, -1, anch);
    E->property_font() = "Sans 10";
    E->set_property("fill-color-rgba", 0X007F00FF);
    root->add_child(E); /* Añade la etiqueta al grupo. */

    /* Etiqueta W (Oeste). */
    azel_to_xy(270.0, 0.0, &x, &y); /* Conversor de azimuth/elevación a xy.
*/
    correct_pole_coor(POLAR_VIEW_POLE_W, &x, &y, &anch); /*
Corrije coordenadas polares. */
```

```
    W = Goocanvas::TextModel::create("W", x, y, -1, anch);
    W->property_font() = "Sans 10";
    W->set_property("fill-color-rgba", 0X007F00FF);
    root->add_child(W); /* Añade la etiqueta al grupo. */

    /* Texto de cursor. */
    curs = Goocanvas::TextModel::create("Cursor", cx – r – 2 *
POLV_LINE_EXTRA, cy + r + POLV_LINE_EXTRA, -1,
Goocanvas::ANCHOR_W);
    curs->property_font() = "Sans 8";
    curs->set_property("fill-color-rgba", 0X00007FFF);
    root->add_child(curs); /* Añade la etiqueta al grupo. */

    /* Texto de localización. */
    locnam = Goocanvas::TextModel::create("ETSIT-UVa", cx – r – 2 *
POLV_LINE_EXTRA, cy - r - POLV_LINE_EXTRA, -1,
Goocanvas::ANCHOR_SW);
    locnam->property_font() = "Sans 8";
    locnam->set_property("fill-color-rgba", 0X00007FFF);
    locnam->add_child(curs); /* Añade la etiqueta al grupo. */

    /* Texto de próximo evento. */
    next = Goocanvas::TextModel::create("Next", cx + r + 2 *
POLV_LINE_EXTRA, cy - r - POLV_LINE_EXTRA, -1,
Goocanvas::ANCHOR_E);
    next->property_font() = "Sans 8";
    next->set_property("fill-color-rgba", 0X00007FFF);
    root->add_child(next); /* Añade la etiqueta al grupo. */

    /* Texto de satélite seleccionado. */
```

```
    sel = Goocanvas::TextModel::create("Sat", cx + r + 2 *
POLV_LINE_EXTRA, cy + r + POLV_LINE_EXTRA, -1,
Goocanvas::ANCHOR_E);
    sel->property_font() = "Sans 8";
    sel->set_property("fill-color-rgba", 0X00007FFF);
    root->add_child(sel); /* Añade la etiqueta al grupo. */

    /* Marcador y texto del satélite en seguimiento. */
    marker = Goocanvas::TextModel::create(cx – MARKER_SIZE_HALF, cy
– MARKER_SIZE_HALF, 2 * MARKER_SIZE_HALF, 2 *
MARKER_SIZE_HALF);
    marker->set_property("fill-color-rgba", 0X8F0000FF);
    marker->set_property("stroke-color-rgba", 0X8F0000FF);
    marker->set_property("visibility", Goocanvas::ITEM_INVISIBLE); /*
Oculta el objeto. */
    root->add_child(marker); /* Añade el rectángulo al grupo. */
    label = Goocanvas::TextModel::create("SAT", cx, cy + 2, -1,
Goocanvas::ANCHOR_NORTH);
    label->property_font() = "Sans 8";
    label->set_property("fill-color-rgba", 0X00007FFF);
    label->set_property("visibility", Goocanvas::ITEM_INVISIBLE); /*
Oculta el objeto. */
    root->add_child(label); /* Añade la etiqueta al grupo. */

    return root; /* Retorna el grupo. */
```

Código del método update_sat:

```
    float x,y; /* Variables de posición. */
    if(sat_name != NULL)
    {
        /* Si el puntero sat_name no es nulo, cambia el texto a la etiqueta del
satélite. */
```

```
        label->set_property("text", Glib::ustring(sat_name));
        sel->set_property("text", Glib::ustring(sat_name));
    }

    /* Si los punteros sat y sat_name no son nulos, traslada el marcador y
etiqueta por el plano polar. */
    if(sat != NULL && sat_name != NULL)
    {
        azel_to_xy(sat->az, sat->el, &x, &y); /* Convierte azimuth y
elevación a x e y. */
        marker->set_property("x", x - MARKER_SIZE_HALF);
        marker->set_property("y", y - MARKER_SIZE_HALF);
        label->set_property("x", x);
        label->set_property("y", y + 2);
        marker->set_property("visibility", Goocanvas::ITEM_VISIBLE); /*
Objeto visible. */
        label->set_property("visibility", Goocanvas::ITEM_VISIBLE); /*
Objeto visible. */
    }
```

5.5.12 Widget JobView

El widget JobView, es un listado, que muestra la lista de tareas, de seguimientos de satélites, programadas. El listado muestra el ID, nombre, hora de comienzo y final de pase, frecuencia base del enlace de bajada y modo de funcionamiento.

El widget utiliza los widgets TreeView, ListStore, ColumnRecord, Row y TreeModelColumn. Deriva de una herencia de la clase VBox, que será el contenedor del título del widget, y del listado de tareas de seguimiento.

En la Figura 36, podemos observar los datos mostrados por el widget, para la planificación del seguimiento de varios satélites.

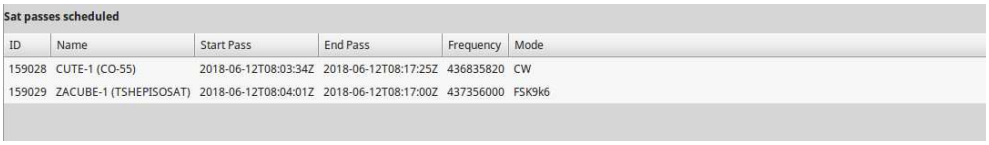


Figura 36. Widget JobView

El código del constructor e inicializador del widget es el siguiente:

```
/* Etiqueta de cabecera. */
m_header = Gtk::manage(new Gtk::Label()); /* Nuevo objeto Label. */
m_header->set_markup("<b>Sat passes scheduled</b>"); /* Configura el contenido de la etiqueta, mediante lenguaje de marcado para mostrarlo en negrita. */
m_header->set_xalign(0.0f); /* Configura la alineación horizontal. */
m_header->set_yalign(0.5f); /* Configura la alineación vertical. */
pack_start(*m_header, true, true, 10); /* Añade el objeto Label, al contenedor, con un padding de valor 10. */

/* Creación del modelo de columnas. */
m_model.add(m_col_id); /* Columna ID. */
m_model.add(m_col_name); /* Columna Nombre. */
m_model.add(m_col_start); /* Columna Comienzo del pase. */
m_model.add(m_col_end); /* Columna Fin del pase. */
m_model.add(m_col_freq); /* Columna Frecuencia del satélite. */
m_model.add(m_col_mode); /* Columna Modo de funcionamiento. */

m_refTreeModel = Gtk::ListStore::créate(m_model); /* Crea una lista. */

m_TreeView.set_model(m_refTreeModel); /* Configura el visor en árbol, como una lista. */

/* Añadido de columnas al visor. */
m_TreeView.append_column("ID", m_col_id);
m_TreeView.append_column("Name", m_col_name);
m_TreeView.append_column("Start Pass", m_col_start);
```

```
    m_TreeView.append_column("End Pass", m_col_end);
    m_TreeView.append_column_numeric("Frequency", m_col_freq, "%d");
    m_TreeView.append_column("Mode", m_col_mode);

    pack_start(m_TreeView, Gtk::PACK_SHRINK); /* Añade el visor al
contenedor. */
```

Código del método de actualización del listado:

```
    Gtk::TreeModel::Row row; /* Variable para filas. */
    std::multimap<time_t, task_t>::iterator curTask; /* Iterador del
multimapa. */

    m_refTreeModel->clear(); /* Limpia el listado, del visor. */

    /* Comprueba que el multimapa no esté vacío. */
    if(!scheduler->task.empty())
    {
        /* Recorre el multimapa. */
        for(curTask = scheduler->beginTasks(); curTask != scheduler-
>endTasks(); curTask++)
        {
                task_t currentTask = (*curTask).second; /* Obtiene la
estructura de la tarea, de la posición actual. */
                /* Comprueba que el ID de la tarea tenga una longitud superior
a doce caracteres. */
                if(strlen(currentTask.id) > 12)
                {
                        /* Compara los primeros 12 caracteres, para ver si la
tarea es una observación. */
                        if(strncmp(currentTask.id, "observation_", 12) == 0)
                        {
```

```
                                satnogs_job_t * job = (satnogs_job_t *)
currentTask.args; /* Carga los datos de seguimiento de satnogs. */
                                row = *(m_refTreeModel->append()); /*
Nueva fila. */
                                row[m_col_id] = job->id;
                                row[m_col_name] = Glib::ustring(job->tle0);
                                row[m_col_start] = Glib::ustring(job->start);
                                row[m_col_end] = Glib::ustring(job->end);
                                row[m_col_name] = job->frequency;
                                row[m_col_mode] = Glib::ustring(job->mode;
                        }
                }
        }
    }
```

6 Desarrollo del programa de Arduino para el control de la polarización

La estación terrena de la Universidad de Valladolid, dispone en su sistema radiante, un control de sentido de polarización. El control de sentido de polarización, se realiza mediante una señal de control eléctrica.

En la actualidad, la única manera de realizar el cambio de polarización, es de forma manual. Se ha desarrollado, con arduino y un transistor MOSFET, un control automático de cambio de polarización, así mediante comandos escritos en el puerto serie del PC, podemos cambiar el sentido de la polarización.

La placa Arduino utiliza ha sido el modelo Nano, que dispone de los puertos suficientes para nuestros propósitos, además de ocupar muy poco tamaño.

El código desarrollado y cargado en la placa Arduino es el siguiente:

```
/* Incluye la librería EEPROM, para el manejo de memoria. */
#include <EEPROM.h>

/* Variable de Direccionamiento. */
int direccion = 0;

/* Variable para lectura del valor en memoria. */
byte valor;

/* Pin de control de polarización. */
int pin_polarizacion = 2;

void setup()
{
  pinMode(pin_polarizacion, OUTPUT);
  /* Inicialización del puerto serie. */
  Serial.begin(9600);
  Serial.flush();
```

```
  /* Lectura del valor del control de polarización. */
  valor = EEPROM.read(direccion); /* Lectura en la dirección 0. */

  if(valor == 1)
  {
    digitalWrite(pin_polarizacion, HIGH);
  }
  else if(valor == 0)
  {
    digitalWrite(pin_polarizacion, LOW);
  }
}

void loop()
{
  String input = "";

  /* Comprueba si hay datos en el puerto serie. */
  while(Serial.available() > 0)
  {
    /* Lee un caracter del puerto serie. */
    input += (char) Serial.read(); /* Espera 5ms. */
    delay(5);
  }

  if(input == "on")
  {
    digitalWrite(pin_polarizacion, HIGH); /* Salida en alta. */
    EEPROM.write(direccion, 1); /* Escribe el estado en memoria. */
  }
  else if(input == "off")
```

```
    {
      digitalWrite(pin_polarizacion, LOW); /* Salida en baja. */
      EEPROM.write(direccion, 0); /* Escribe el estado en memoria. */
    }

  }
```

Los comandos admitidos por el controlador son *"on"*, para la activación de la polarización, y *"off"* para la polarización por defecto. Además, el estado del controlador, es guardado en la memoria EEPROM interna de la placa Arduino, para que en caso de apagado del controlador, mantenga la configuración.

7 Resultados

Una vez concluida la programación del cliente, y del control de polarización, se exponen las pruebas de funcionamiento del cliente con la red SatNOGS.

El primer paso para la comprobación, será entrar en la red de SatNOGS y seleccionar la estación terrena con ID 93 GS UVA, tras entrar en la página de la estación, se nos muestrar un listado de pases que pueden programarse para la estación, en la Figura 37, podemos observar algún pase posible para realizar el seguimiento.

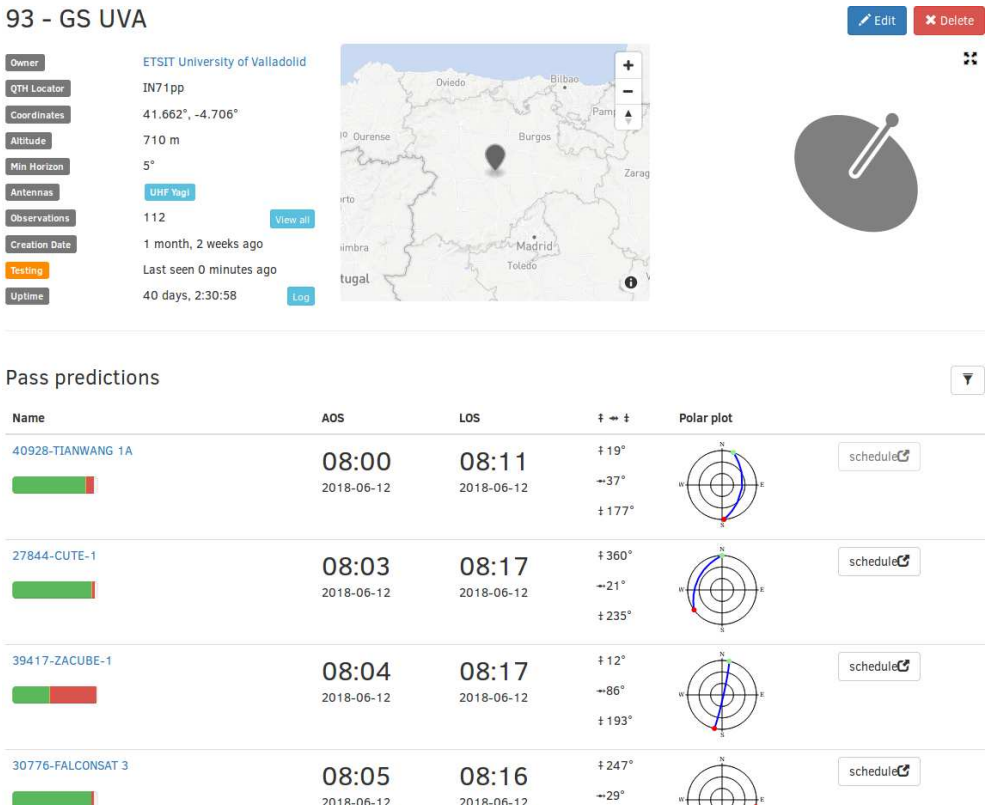


Figura 37. Página de la GS UVA en SatNOGS

La programación de una observación, se realiza mediante el botón schedule, que nos llevará a otra pantalla, donde podremos seleccionar el transpondedor, del satélite, del cual queremos recibir datos.

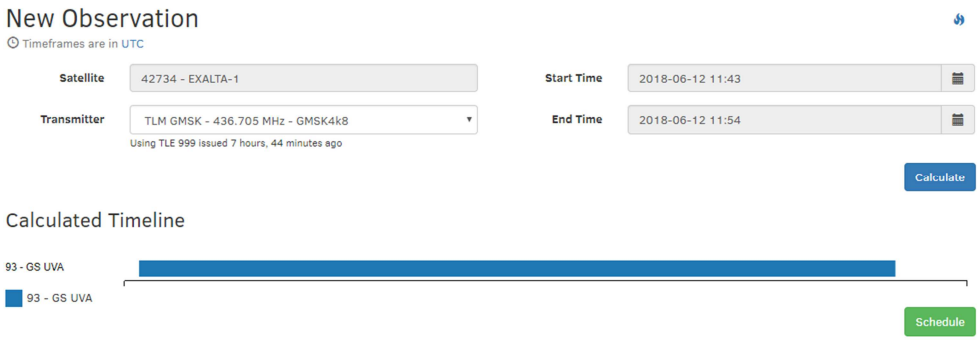


Figura 38. Programación del seguimiento y selección de transpondedor

Tras la programación de los seguimientos de satélites a realizar, en la aplicación cliente, desarrollada en este proyecto, podemos observar que en el listado de pases programados, observamos el listado de seguimientos programados.

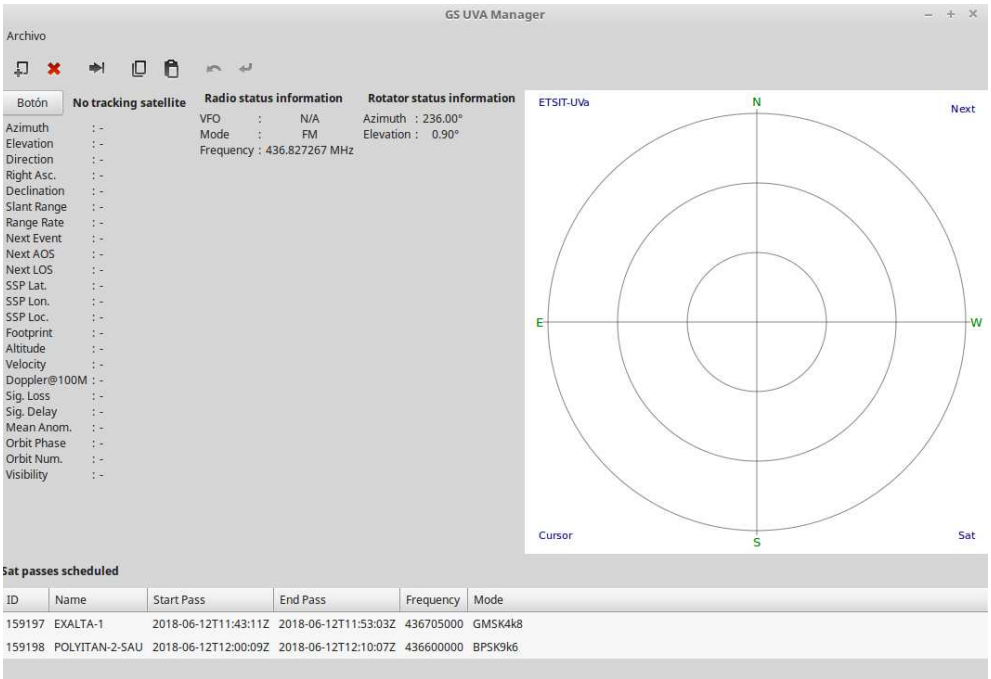


Figura 39. Cliente esperando al comienzo de un seguimiento

Durante el seguimiento del satélite, la aplicación desarrollada, nos muestra información sobre el satélite en seguimiento, así como el estado de la radio y rotores, para que el usuario esté informado en todo momento de la situación.

Mientras tanto, la aplicación también realiza la grabación del audio recibido, durante el seguimiento del satélite. En la , podemos ver que se está realizando el seguimiento del satélite POLYITAN-2-SAU, el cual, en el momento de la toma de imagen, se encontraba a azimuth 37,25 º y elevación 10,95 º, además de trasladarse a una velocidad de 7,699 km/s.

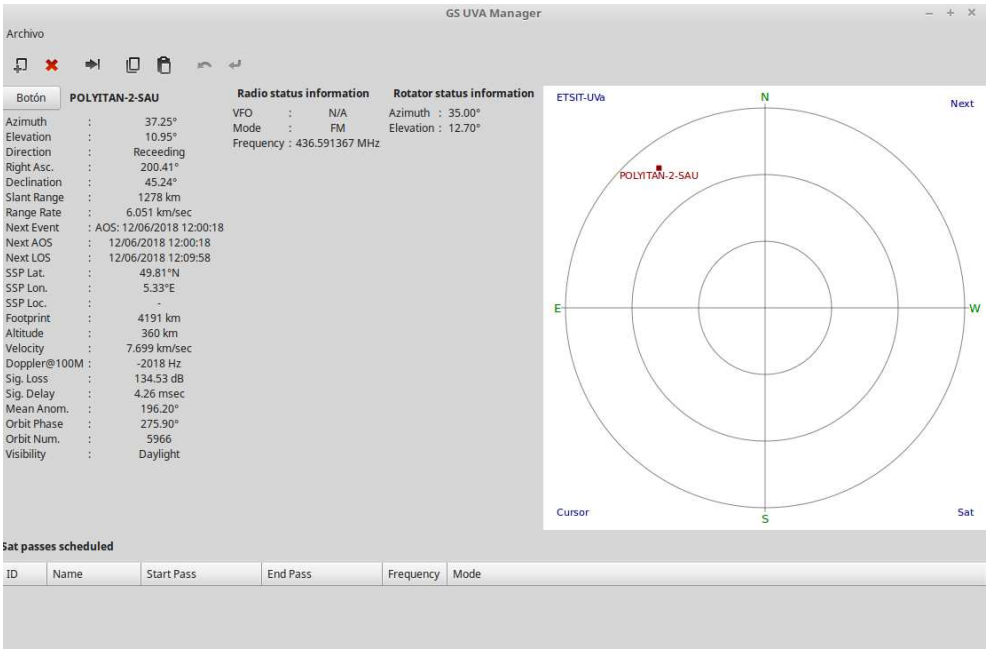


Figura 40. Cliente realizando un seguimiento

Una vez concluido el seguimiento, los datos son enviados a la red de SatNOGS, para el divulgamiento público. Podemos observar, en la página de la observación, los datos recopilados durante el pase, ejemplo claro es la audición del audio grabado.

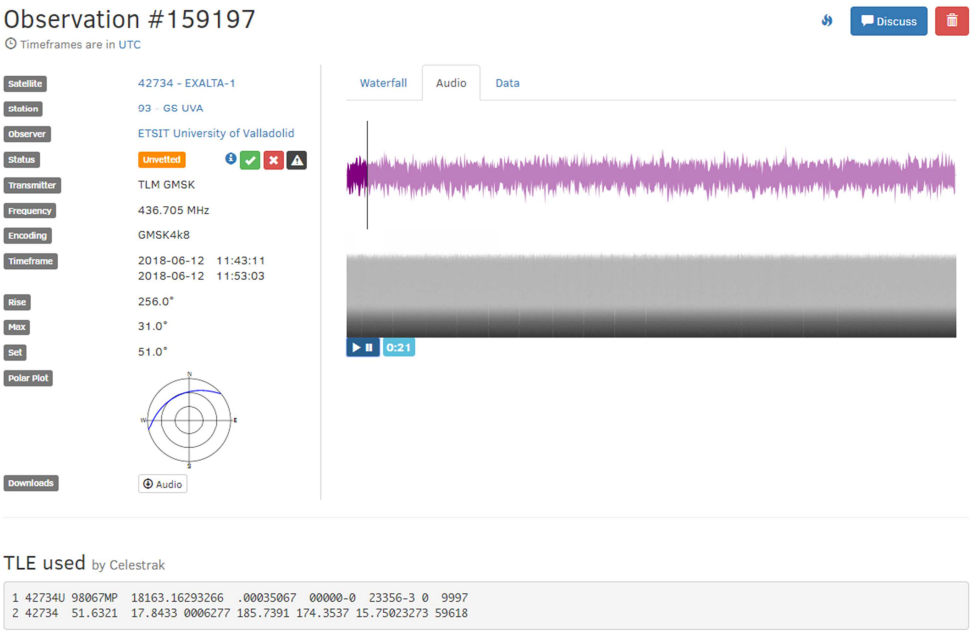


Figura 41. Datos de la observación, una vez concluida

Tras la revisión de los datos recopilados durante el pase, SatNOGS dispone de un sistema de filtrado, en función del estado de calidad del pase, en el que podemos catalogar una observación como buena, mala o fallida.

Figura 42. Observación con claros indicios de captura de datos

La red de SatNOGS, permite a cualquier usuario registrado en la red, planificar seguimientos de satélites en cualquier estación conectada. Hasta el momento, en la estación de seguimiento GS UVa, solo se han programado observaciones por parte del usuario propietario de la estación, pero no se descarta que en un futuro próximo, haya observaciones programadas por otros usuarios.

Tras la utilización de la aplicación, en múltiples seguimientos, podemos afirmar que los resultados obtenidos han sido positivos.

El segundo elemento, desarrollado en este proyecto, es el prototipo de control de la polarización de las antenas, para las bandas de VHF y UHF. El dispositivo resultante, es un simple controlador, que obtiene comandos de control por el puerto serie, activando o desactivando el pin de control. El controlador dispone de memoria EEPROM interna, para el almacenamiento del estado, así siempre, en caso de pérdida de energía eléctrica o desconexión, el estado de la polarización, queda almacenada y activada en el momento de inicio del dispositivo.

El pin digital de control del controlador, está conectado a un transistor MOSFET (IRF520N), que actuará como un interruptor, sustituyendo a un relé físico y convirtiéndolo en un relé de estado solido. A la salida del MOSFET, se conectará la señal eléctrica de cambio de polarización, proveniente de las antenas. En el prototipo, se ha simulado la conexión con el intercambiador de polarización, por un LED, haciendo más fácil la visualización de las acciones que realiza el dispositivo.

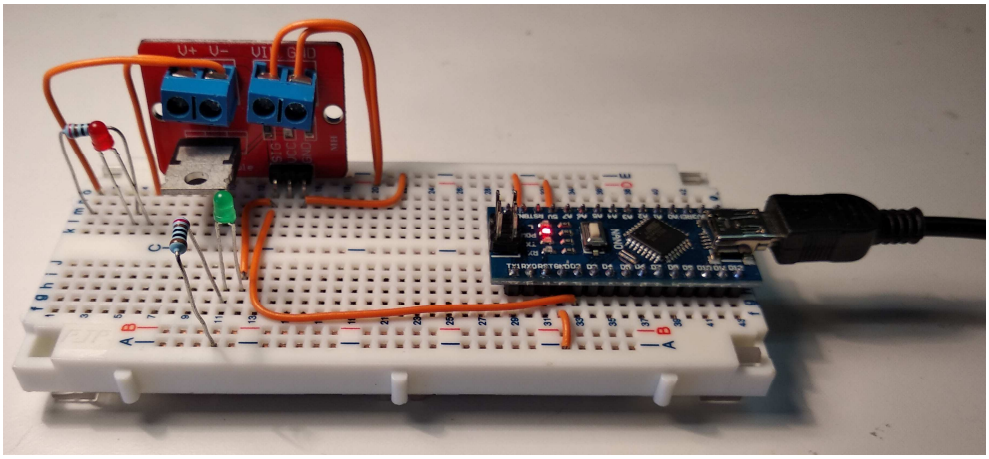


Figura 43. Prototipo resultante apagado

Los comandos, admitidos por el prototipo, enviados por el puerto serie, son *"on"* para la activación de la señal de control, que a su vez intercambiará el sentido de la polarización de las antenas, y "off" para la desactivación de la señal de control, que hará que el intercambiador de sentido de polarización se situe en la posición por defecto.

En la actualidad, el control de este dispositivo no se encuentra integrado en la aplicación cliente de la estación terrena GS-UVa desarrollada, pero se puede controlar mediante una consola de control de puertos serie (RS-232). También, se ha detectado un problema, durante las pruebas de funcionamiento sobre Linux Mint, instalado en la estación. El problema es el reinicio de la placa controladora Arduino, en el momento de la apertura de la conexión serie en el PC con el dispositivo, el reinicio del controlador provoca que no responda a los comandos de control enviados, no realizando la conmutación de polarización. Este error es producido por el cambio de estado de los bits DTR y HANG UP, del puerto serie, los cuales Arduino utiliza para el reinicio del controlador, para entrar en modo programación durante el inicio, a través del bootloader.

Para el correcto funcionamiento, es necesario, en la consola serie utilizada, descactivar el cambio de estado de los bits anteriormente mencionados.

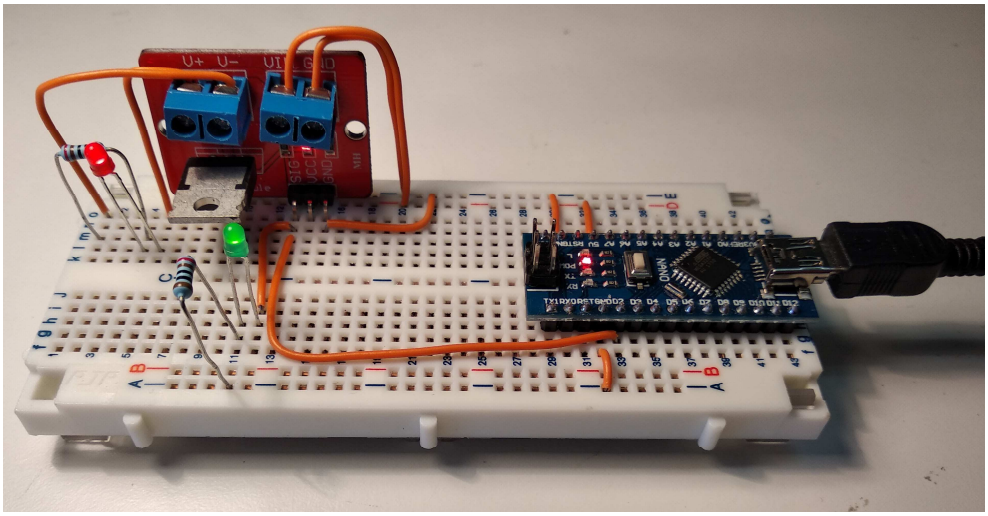


Figura 44. Prototipo de control de polarización encendido

En la Figura 44. Prototipo de control de polarización encendido, podemos observar dos LED, el LED rojo simula la carga que se conecta al transistor MOSFET, en este caso se conectaría la señal de control proveniente de las antenas. El LED verde, es un led de información de estado. El prototipo, una vez terminado el desarrollo final, debe ir guardado en una caja para electrónica, con LEDs de estado.

8 Conclusiones y líneas futuras

8.1 Conclusiones generales

Durante el desarrollo del proyecto, ha sido necesario abordar varios problemas iniciales, impuestos por la características necesarias para el correcto funcionamiento y proporción de datos a la red SatNOGS. El primer problema vino con la comunicación con la red, SatNOGS utiliza una API REST, compuesta por peticiones y respuestas HTTPS, con el formato de respuesta en JSON, además debe permitir el uso de cabeceras personalizadas, para poder realizar la autenticación en la red. Este problema fue solventado con la utilización de dos bibliotecas de funciones, libCURL para la comunicación HTTPS, y nxjson para la decodificación de los mensajes en formato JSON. Este tipo de bibliotecas son fáciles y sencillas de utilizar, con una documentación clara y concisa para su implementación.

El segundo problema que nos encontramos, tras el desarrollo de la comunicación con SatNOGS, fue la gestión de tareas de seguimiento. Era necesario disponer de un mecanismo de programación y ejecución de tareas, en función del instante de tiempo. Sin este elemento, podemos comunicarnos con la red, pero no podíamos realizar las tareas de seguimiento. Para solventar este problema, se creó una clase de programación de tareas, la cual escrita en C++, ayudó a programar las tareas de seguimiento, así como las peticiones del listado de seguimientos a realizar. Está basado en threads (ejecución multihilo), con una gestión de CPU que podemos categorizar como buena, ya que en el tiempo que tiene que esperar al comienzo de una tarea, el hilo gestor de tareas se pone en una situación de dormir. El hilo gestor puede ser despertado por una señal o por vencimiento de temporizador. Gracias a esta clase desarrollada, no se despilfarran recursos de CPU en bucles infinitos comprobando condiciones de cominezo de las tareas.

El tercer problema que se planteaba, era el cálculo de la posición de los elementos en seguiiento. Los cuerpos espaciales siguen órbitas descritas por un conjunto de elementos, llamados TLE. Este conjunto de elementos debe ser interpretado y aplicado un modelo matemático, para así obtener los datos de

predicción de posición y velocidad. El primer paso para solucionar este problema, fue la búsqueda de información sobre los elementos TLE y modelos matemáticos utilizados para obtener los datos deseados. Tras la lectura de documentación, se llegó a la conclusión que el modelo SGP4/SDP4 es un modelo que cumplía con las exigencias que necesitábamos. Además, este modelo matemático está bastante extendido en los programas de seguimiento de satélites. La biblioteca de funciones elegida, para nuestro propósito, que además estaba escrita en C, fue una realizada por varios autores, basado en el trabajo realizado en FORTRAN por el Dr. T.S. Kelso, y obtenido del proyecto GPredict, de Alexandru Csete.

Una vez solucionados los problemas anteriores, ya se podía realizar el seguimiento de los satélites que se planificasen para la estación, por parte de la red de SatNOGS. A pesar de estas soluciones, se planteaba otro problema a solucionar, la adquisión o grabación de audio en ficheros, con formato Vorbis OGG. Para solucionar el problema, se utilizaros dos bibliotecas, una para la adquisión de audio, y otra para la codificación y escritura del fichero de audio resultante. La búsqueda de bibliotecas, que cumpliesen estos requisitos, no fue fácil de encontrar. Muchas soluciones a la adquisición de audio, venían por la utilización de las funciones nativas de control de los dispositivos de audio, solución complicada y engorrosa, además solo funcional en una única plataforma, no permitían el uso del cliente en otro sistema operativo. Si la solución a la adqusición de las muestras de audio fue complicada, la búsqueda de soluciones para la codificación y guardado fue aun peor. Las soluciones se basaban en la utilización de un conjunto de bibliotecas de funciones, que solo manejaban ficheros de audio entre conversiones, esto provocaría la creación y eliminación de varios ficheros intermedios, además del uso de mayor número de bibliotecas, con el consiguiente aumento de la complejidad de la aplicación. Tras mucho buscar por Internet, apareción la biblioteca deseada, que permitía la escritura del fichero de audio, con la codificación Vorbis OGG, además la codificación y guardado se hace al vuelo, evitando ficheros intermedios y múlutiplés bibiliotecas, simplificando el uso y código del programa.

Solventados los problemas iniciales, la aplicación desarrollada, era totalmente funcional, pero no visualmente agradable. La aplicación funcionaba

únicamente mediante línea de comandos, sin una interfaz de usuario amigable en el que poder visualizar todos los datos necesarios de los seguimientos programados o del estado del hardware. El siguiente objetivo, tras cumplir los objetivos mínimos, fue la realización de una interfaz gráfica de usuario, en el cual se podrían ver la información fácilmente. Como hemos visto en apartados anteriores, la interfaz gráfica dispone de varios widgets, para mostrar la información del satélite en seguimiento, la lista de seguimientos programados, el estado de la radio y rotores, o un plano polar donde ver el movimiento del cuerpo espacial en seguimiento. El primer paso para el desarrollo de la interfaz, fue la elección de la biblioteca gráfica. En este caso se utilizó GTK+, a través de gtkmm que es el conector oficial de GTK+ para C++. La elección esta biblioteca, se describió en secciones anteriores, haciendo pequeñas comparativas con otras bibliotecas gráficas. Tras la elección de gtkmm para el desarrollo, vino la elección, prácticamente obligatoria, al ser un widget de GTK+, la biblioteca goocanvasmm, que proporciona los elementos necesarios para el dibujo del plano polar.

El desarrollo del entorno gráfico comenzó con el widget de información del satélite en seguimiento, tras el desarrollo del widget, se reestructuró el código anterior, para mejor comprensión, mediante la división en funciones. El inicio y final de grabación de audio se separó en dos funciones, las peticiones y subida de datos a SatNOGS se separaron también. Realizada la reestructuración y limpieza del código, se acopló la vista con el resto de la aplicación, así inicialmente se podría visualizar al menos los datos del satélite, obtenidos por la predicción en cada instante de tiempo.

La primera prueba, de la aplicación resultante con el único widget, se puso en marcha, para ello se hizo la programación de varios satélites, para así poder observar el correcto funcionamiento de la aplicación, como ocurría anteriormente. El resultado no fue así, tras finalizar el primer seguimiento, la aplicación se cerraba por error de segmentación. Este fallo, inicialmente, resultó raro, ya que la aplicación anteriormente funcionaba, y el widget mostraba los datos correctamente, siendo el único fallo al finalizar el seguimiento. Para la detección del fallo, se hizo uso de un depurador de Linux, GDB, disponible en

el paquete de utilizades de desarrollo, junto con GCC o G++ utilizados para la compilación y enlazado de la aplicación.

Tras muchas pruebas de funcionamiento, con el depurador, se observó que no siempre ocurría el error en el final del seguimiento del primer satélite, sino que ocurria en los finales de otros seguimientos o incluso durante algunos pases, a partir del segundo. Comprobando los datos, recopilados por el depurador, llegué a la conclusión, que el fallo se estaba produciendo en la grabación del sonido. Comprobé que todo estaba inicializado correctamente, y los parámetros que se pasaban entre funciones eran correctos y disponián de las reservas de memoria e inicializaciones correspondientes, la comprobación era correcta, y no había motivo alguno para que se produjese el error. En vista de que el error no se corregia, tras las comprobaciones, volví a juntar el código de inicio y final de grabación, en la función de seguimiento de satélites. Tras juntar el código en uno único, el error desapareción, volviendo a funcionar el programa con total normalidad. A día de hoy no sé a que se debía el error, y porque por separado no funciones, pero junto si funciona.

Tras el arreglo del error con la grabación, continué desarrollando los widgets de información de radio e información de rotores. Estos dos widgets, son muy sencillos, únicamente hacen unas llamadas a la biblioteca de HamLib, para obtener el estado del hardware. Con estos widgets, también aparecieron nuevos problemas. El widget de la radio, dejó de mostrar la información, mostrando en su defecto un modo no disponible, y una frecuencia cero. Este hecho no se producía siempre, solo en algunas ocasiones. Tras realizar comprobaciones, se hizo uso de la biblioteca de HamLib, con las propias herramientas de la biblioteca, para comprobar si el fallo estaba en el código del widget. Tras las comprobaciones, se obtuvo también un resultado negativo, siendo problema externo a HamLib, y por supuesto al software desarrollado. Comprobando el conexionado hardware, se detectó el error, que estaba producido por una conexión floja entre la interfaz serie de la radio y el PC. Se ajustó la conexión, volviendo el funcionamiento a la normalidad. En cambio en la información de los rotores, se notó que en ocasiones, la posición de azimuth y elevación, por separado, se mostraba con valor cero, nunca se producía el valor cero simultáneamente. Este fallo dio que pensar, ya que la biblioteca HamLib,

nos devuelve la posición de los dos ejes en conjunto, y no por separado. Se repasó la escritura del código, por si hubiese alguna variable que se modificase desde varias funciones. El funcionamiento del código era correcto. También se comprobó el conexionado del controlador de rotores con el PC, aunque a primera vista parecía raro y descartable, que en este caso, el error sea por conexión. Tras varias pruebas de funcionamiento, se detectó que el fallo, únicamente, ocurre caundo el controlador de rotores esta posicionándose, y rara vez lo hacía cuando estaba posicionado. Visto estos resultados en las comprobaciones, el error dejó de tener importancia, dejándolo de considerar como tal, ya que muy probablemente esté ocasionado por estar el dispositivo ocupado.

Otro elemento desarrollado en este proyecto, es un dispositivo de control de polarización, para las antenas de VHF y UHF. Este elemento, es un prototipo electrónico, compuesto por parte hardware y parte software. Está realizado con una placa, de la plataforma Arduino, en concreto el prototipo está hecho con Arduino Nano, aunque inicialmente, y en secciones anteriores se habló de Arduino Micro. Ambas placas son compatibles, y de prestaciones similares para el propósito planteado. Además de la plataforma Arduino, el prototipo se compone de un transistor MOSFET (IRF520N), que no es un transistor óptimo para la utilización con Arduino, ya que no se puede obtener todas las prestaciones, esto es debido a que para poder conseguir manejar la cantidad de amperios, que puede manejar el transistor, es necesario una tensión de entre puerta y fuente de nueve voltios, y Aduino solo saca cinco voltios por sus salidad digitales. Para este proyecto, el no poder manejar grandes cantidades de corriente no es un problema, ya que la corriente máxima que tiene que soportar el transistor es del orden de los miliamperios. Al ser la corriente muy pequeña, y el transistor viene preparado para la conexión con Arduino, se eligió este transistor, además por su bajo precio, sobre un euro.

La placa Arduino necesitaba ser programada, con un programa que realizase las funciones que se necesitasen para el controlador. La placa Arduino, sin programa cargado, no hace nada por si sola. Se programó un programa básico, que fue cargado y probado sobre Arduino Nano, tal y como se explicó en una sección anterior. El funcionamiento actual es mediante comunicación

por puerto de serie, recibiendo comandos para el control del MOSFET. El comando *"on"*, activa el MOSFET, el comando *"off"* desactiva el MOSFET, actuando el transistor MOSFET como interruptor.

8.2 Líneas futuras

El proyecto desarrollado, aunque sea totalmente funcional, para los objetivos propuestos, hay aspectos que se pueden mejorar o añadir nuevas mejoras.

Actualmente, no hay implementación en la aplicación cliente de SatNOGS para el control del dispositivo controlador de polaridad, si bien, es posible el control de este dispositivo con aplicaciones externas de control del puerto serie del PC.

En la actualidad, la aplicación cliente desarrollada, en sí, es totalmente funcional, planificando los pases, obteniendo los cálculos de predicción satelital, grabando el audio y transmitiendo los resultados a SatNOGS. Es posible, en líneas futuras, mejorar las características del software cliente desarrollado, como es la inclusión de nuevos widgets como puede ser un mapa con satélites, o añadir funcionalidad de conexión con la TNC, para la decodificación de datos de satélites. También puede ser interesante, añadir alguna funcionalidad para los momentos de reposo, durante la no planificación de seguimientos o esperas entre pases, como puede ser la orientación al Digipeater (repetidor digital) de URE Valladolid, con el que capturar paquetes APRS (Automatic Packet Report System), e incluso transmitir, mediante APRS a través del Digipeater, la posición de la estación terrena GS UVa. La transmisión de la posición puede indicar al resto de usuarios la posición física de la estación, así como el estado activo de la misma.

También sería necesario, en el futuro, añadir el control, desde la aplicación, al controlador de polarización, para poder seleccionar fácilmente la polarización a utilizar. Además, el prototipo desarrollado, sería conveniente convertirlo en dispositivo final, metido en una caja protectora. Otro dispositivo a realizar, puede ser el controlador de conmutación de antenas de 144-146 MHz con la antena de 2400 MHz, mediante la actualización del actual prototipo de control

de polarización. Para ello sería neceario añadir otro transistor de control eléctrico del relé coaxial, en este momento inexistente. Además sería necesaria la adaptación del código, para el control de un segundo transistor y cambio de los comandos de control.

9 Referencias

National Aeronautics and Space Administration, «CubeSat101 Basic
1] Concepts and Processes for First-Time CubeSat Developers,» Octubre 2017. [En línea]. Available: https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf. [Último acceso: 5 Junio 2018].

D. A. Vallado, P. Crawford, R. Hujsak y T. Kelso, «Revisiting
2] Spacetrack Report #3,» 2006. [En línea]. Available: http://celestrak.com/publications/AIAA/2006-6753/AIAA-2006-6753.pdf. [Último acceso: 05 Junio 2018].

P. Tipler, Física Moderna, Editorial Reverté, 1980.
3]

European Space Agency, «Global Educational Network for Satellite
4] Operations,» [En línea]. Available: http://esa.int/Education/Global_Educational_Network_for_Satellite_Operations. [Último acceso: 5 Junio 2018].

ISIS Space, [En línea]. Available: https://www.isispace.nl/isis-launches-
5] 23-cubesats-low-earth-orbit-pslv-c38/. [Último acceso: 24 junio 2018].

European Space Agency, «How GENSO Works,» [En línea]. Available:
6] http://www.esa.int/Education/How_GENSO_works. [Último acceso: 5 Junio 2018].

European Space Agency, «Genso Project Timeline,» [En línea].
7] Available: http://www.esa.int/Education/Project_timeline. [Último acceso: 5 Junio 2018].

J. C. Vicente Corral, Sistema de transferencia y control en una red de
8] estaciones terrenas para comunicaciones espaciales, Valladolid: Escuela Técnica Superior de Ingenieros de Telecomunicación. Universidad de Valladolid, 2014.

Libre Space Foundation, «SatNOGS Intro,» 4 Abril 2017. [En línea].

9] Available: https://wiki.satnogs.org/Intro. [Último acceso: 5 Junio 2018].

Libre Space Foundation, «SatNOGS Frequently Asked Questions,» [En
10] línea]. Available: https://satnogs.org/faq/. [Último acceso: 5 Junio 2018].

ICOM INC., «IC-910H Intruction Manual,» 2010. [En línea]. Available:
11] http://www.icom.co.jp/world/support/download/manual/pdf/IC-910H_9a.pdf. [Último acceso: 2 Junio 2018].

M2 Antenna Systems, Inc., «2MCP14 Manual,» [En línea]. Available:
12] http://www.m2inc.com/content/PDF%20MANUALS/2MANTS/2MCP14 MAN02-W.pdf. [Último acceso: 2 Junio 2018].

M2 Antenna Systems, Inc., «436CP30 Manual,» [En línea]. Available:
13] http://www.m2inc.com/content/PDF%20MANUALS/2MANTS/2MCP14 MAN02-W.pdf. [Último acceso: 2 Junio 2018].

SSB Electronic USA, «SP-6/SP-2000/SP-220/SP-7000 Super Amp
14] GaAsFET Series,» [En línea]. Available: http://www.tel.uva.es/personales/gs_uva/docs/SSB-Electronic_SP-6_SP-220_SP-2000_SP-7000_user.pdf. [Último acceso: 2 Junio 2018].

SSB-Electronic GmbH, «UEK-3000 Users Manual,» [En línea].
15] Available: http://www.ssb.de/download/Geraete/161219%203018%20UEK-3000%20ENG.pdf. [Último acceso: 2 Junio 2018].

Kantronics, «KPC-9612 Plus User's Guide,» [En línea]. Available:
16] https://www.kantronics.com/documents/KPC-9612PMX_Manual.pdf. [Último acceso: 2 Junio 2018].

Arduino, «Arduino Introduction,» [En línea]. Available:
17] https://www.arduino.cc/en/Guide/Introduction. [Último acceso: 2 Junio 2018].

D. M. Ritchie, «The Development of the C Language,» AT&T Bell
18] Laboratories, [En línea]. Available: https://heim.ifi.uio.no/inf2270/programmer/historien-om-C.pdf. [Último acceso: 28 Mayo 2018].

B. Stroustrup, *El lenguaje de programación C++.*

19]

Hamlib Github Community, «Ham Radio Control Libraries,» 4 Marzo
20] 2018. [En línea]. Available: https://github.com/Hamlib/Hamlib/wiki. [Último acceso: 28 Mayo 2018].

E. Kosmas, «API on SatNOGS Network,» Libre Space Foundation, 30
21] Octubre 2014. [En línea]. Available: https://satnogs.org/2014/10/api-on-satnogs-network/. [Último acceso: 28 Mayo 3018].

Libre Space Foundation, «SatNOGS Network API v 1,» [En línea].
22] Available: https://network.satnogs.org/api/. [Último acceso: 29 Mayo 2018].

«libcurl - the multiprotocol file transfer library,» [En línea]. Available:
23] https://curl.haxx.se/libcurl/. [Último acceso: 18 Mayo 2018].

PortAudio Community, «PortAudio Portable Cross-platform Audio
24] I/O,» [En línea]. Available: http://www.portaudio.com/. [Último acceso: 20 Mayo 2018].

E. d. Castro Lopo, «LibSNDFile,» [En línea]. Available:
25] http://www.mega-nerd.com/libsndfile/. [Último acceso: 25 Mayo 2018].

The GTK+ Team, «The GTK+ Project,» 2017. [En línea]. Available:
26] https://www.gtk.org/. [Último acceso: 20 Mayo 2018].

«gtkmm Interfaces C++ para GTK+ y GNOME,» [En línea].
27] Available: https://www.gtkmm.org/es/. [Último acceso: 22 Mayo 2018].

The GNOME Project, «GooCanvas Reference Manual,» 2014. [En
28] línea]. Available: https://developer.gnome.org/goocanvas/unstable/goocanvas-overview.html. [Último acceso: 3 Junio 2018].

GNOME Foundatio, «gtkmm API Documentation,» [En línea].
29] Available: https://developer.gnome.org/gtkmm/stable/namespaceGtk.html. [Último acceso: 8 Junio 2018].

GNOME Foundation, «goocanvasmm API Reference,» [En línea].
30] Available: https://developer.gnome.org/goocanvasmm/stable/namespaceGoocanvas.html. [Último acceso: 12 junio 2018].

10 Anexos

10.1 Anexo 1: Formato del conjunto TLE

A continuación se detalla el formato, con la descripción de cada campo, de un conjunto TLE.

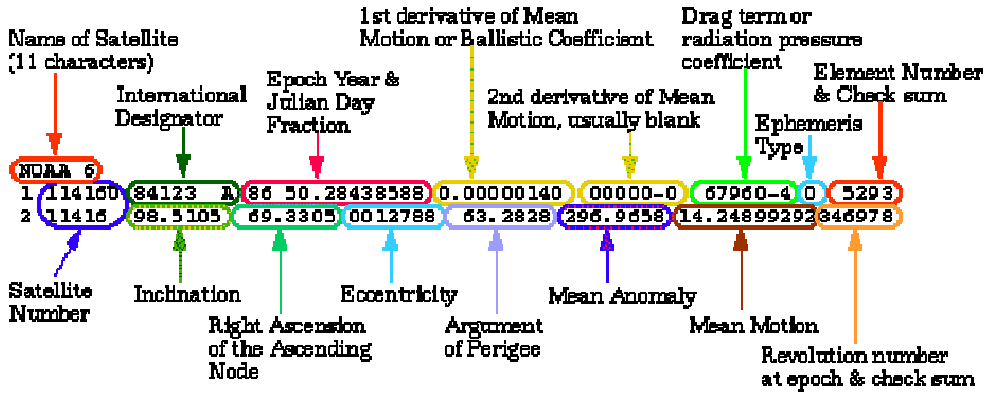


Figura 45. Formato TLE

10.2 Anexo 2: Principales estructuras de datos

10.2.1 Estructura de datos del satélite sat_t

La estructura sat_t se utiliza por los modelos matemáticos, de predición de datos satelitales, SGP4/SDP4.

```
typedef struct
{
    char * name; /* Nombre del satélite */
    char * nickname; /* Alias del satélite */
    char * website; /* Página web del satélite, si está disponible */
    tle_t tle; /* Estructura de datos tle_t */
    int flags; /* Banderas de estados */
    sgpsdp_static_t sgps; /* Datos de los modelos matemáticos SGP4/SDP4 */
    deep_static_t dps; /* Datos de espacio profundo de SDP4 */
    vector_t pos; /* Vector de posición */
    vector_t vel; /* Vector de velocidad */
    obs_set_t bearing; /* Conjunto de datos del observador */
    obs_astro_t astro; /* Conjunto de datos de observación del satélite */
    double jul_epoch; /* Fecha en Juliano desde epoch */
    double jul_utc; /* Fecha en Juliano UTC */
    double tsince; /* Instante de tiempo del cáclulo de predicción */
    double aos; /* Instante de tiempo de inicio de cobertura */
    double los; /* Instante de tiempo de fin de cobertura */
    double az; /* Azimuth */
    double el; /* Elevación */
    double range; /* Rango de elevación */
    double range_rate; /* Tasa de cambio del rango de elevación */
    double ra;
    double dec;
    double ssplat; /* Latitud del satélite */
```

```
    double ssplon; /* Longitud del satélite */
    double alt; /* Altitud del satélite */
    double velo; /* Velocidad del satélite */
    double ma; /* Anomalía media */
    double footprint; /* Área de cobertura */
    double phase; /* Fase */
    double meanmo;
    long orbit; /* Número de órbita */
    orbit_type_t otype; /* Tipo de órbita */
} sat_t;
```

10.2.2 Estructura de datos TLE tle_t

Esta estructura de datos guarda los datos referentes a un conjunto TLE. Los datos que se proporcionan en el conjunto TLE vienen desglosados en el anexo 1.

```
typedef struct
{
    double epoch; /* Fecha desde epoch */
    unsigned int epoch_year; /* Año desde epoch */
    unsigned int epoch_day; /* Día desde epoch */
    double epoch_fod;
    double xndt2o;
    double xndd6o;
    double bstar;
    double xincl;
    double xnodeo;
    double eo;
    double omegao;
    double xmo;
    double xno;
    int catnr; /* Número de catálogo */
```

```
    int elset; /* Grupo de elementos */
    int revnum; /* Número de revisión del TLE */
    char sat_name[25]; /* Nombre del satélite */
    char idesg[9];
    op_stat_t status; /* Estado del satélite */
    double xincl1;
    double xnodeo1;
    double omegao1;
} tle_t;
```

10.2.3 Estructura de datos de posición del observador qth_t

```
typedef struct
{
    char * name; /* Nombre del lugar del observador */
    char * loc; /* Localización del observador */
    char * desc; /* Descripción */
    double lat; /* Latitud del observador */
    double lon; /* Longitud del observador */
    int alt; /* Altitud del observador en kilómetros */
    char * qra; /* Nombre del observador */
    char * wx; /* Datos de estación meteorológica */
    void * data; /* Datos personalizados */
} qth_t;
```

10.2.4 Estructura de datos task_t

```
typedef struct
{
    char * id; /* Identificador de tarea */
    char * name; /* Nombre de tarea */
    int interval; /* Tipo de tarea */
```

```
        int inverval_value; /* Valor en segundos para tareas de tipo intervalo
*/
        void * (*func)(void *); /* Función de callback para ejecución de tarea
*/
        void * args; /* Argumentos para la función de callback */
    } task_t;
```

10.2.5 Estructura de datos del trabajo de seguimiento de satnogs

satnogs_job_t

```
    typedef struct
    {
        int id; /* Identificador de observación de SatNOGS */
        char * start; /* Fecha y hora de comienzo de la observación */
        char * end; /* Fecha y hora de finalización de la observación */
        int ground_station; /* Identificador de la estación terrena */
        char * tle0; /* Línea cero del TLE */
        char * tle1; /* Línea uno del TLE */
        char * tle2; /* Línea dos del TLE */
        int frequency; /* Frecuencia de operación del satélite, para recepción */
        char * mode; /* Modo de funcionamiento del satélite. Modulación */
        char * transmitter; /* Identificador del transpondedor utilizado */
        double baud; /* Tasa de símbolos utilizada en modulaciones digitales
*/
    } satnogs_job_t;
```

Nombre de archivo: Memoria final V1 revisado.docx
Directorio: C:\Users\Samuel\Google Drive\Proyecto Universidad
Plantilla: C:\Users\Samuel\Desktop\LaTex 7.dotx
Título: Adaptación de la estación terrena GS-UVa para su integración en red de seguimiento de satélites
Asunto:
Autor: Samuel Adrián Peña
Palabras clave: satélite, radio, seguimiento, datos, estación terrena, investigación, redes
Comentarios:
Fecha de creación: 24/06/2018 10:21:00
Cambio número: 2
Guardado el: 24/06/2018 10:21:00
Guardado por: Samuel
Tiempo de edición: 0 minutos
Impreso el: 24/06/2018 10:22:00
Última impresión completa
Número de páginas: 134
Número de palabras: 26.547 (aprox.)
Número de caracteres: 146.009 (aprox.)