FDISat: Nodo de adquisición de imágenes de satélite de la FDI
FDISat: FDI's Satellite Imagery Acquisition Node

Trabajo de Fin de Grado
Curso 2022–2023

Autor
César Falcón Gómez

Directores
Juan Carlos Fabero Jiménez
Jose Luis Vázquez Poletti

Grado en Ingeniería en Computadores
Facultad de Informática
Universidad Complutense de Madrid

FDISat: Nodo de adquisición de imágenes de satélite de la FDI
FDISat: FDI's Satellite Imagery Acquisition Node

Trabajo de Fin de Grado en Ingeniería en Computadores

Autor
César Falcón Gómez

Directores
Juan Carlos Fabero Jiménez
Jose Luis Vázquez Poletti

Convocatoria: *Junio 2023*

Grado en Ingeniería en Computadores
Facultad de Informática
Universidad Complutense de Madrid

16 de Mayo de 2023

Dedicatoria

A Irene por ayudarme a retomar la carrera, por animarme cuando más lo necesitaba, por soportarme los agobios y sobre todo por celebrar cada pasito como si fuera suyo.
A mis padres Ana y Jose que aun siendo muy jóvenes siempre he sido su prioridad y siempre han hecho todo lo posible para que sea feliz.
Os quiero.

Agradecimientos

A mis directores de TFG que me han ayudado en todo momento, resolviendo dudas y dándome ideas.

También a mis compañeros de facultad por haberme ayudado en cada momento tanto dentro como fuera de clase, cada ratito que hemos tenido y cada risa que ha hecho que esto sea una de las etapas más bonitas de mi vida: Epi, Pelli, Díaz, Javi, Barbas, Jorge y Patín, gracias.

A mi familia que pese a todas las dificultades han hecho todo lo posible para que sea feliz y tenga una vida maravillosa. Os quiero.

A mi novia Irene, la que me animó a volver a la universidad, la que me ha enseñado que con esfuerzo y con ganas todo es posible, la que me ha demostrado que la paciencia es un don y que con amor todo se puede superar.

Y no puedo olvidarme de mí, que sin mi esfuerzo y trabajo no hubiera sido posible.

Sin todos estos ingredientes este final tan bonito de carrera nunca llegaría. Gracias a todos.

Resumen

FDISat: Nodo de adquisición de imágenes de satélite de la FDI

A lo largo del día son varios satélites meteorológicos los que pasan en una órbita baja terrestre a una altura de 850km sobre de la Facultad de Informática y de nuestra posición. De todos ellos, hay cuatro (NOAA19, NOAA18, NOAA15 Y METEOR-M2) que emiten en la banda de los 137MHz y son de libre captación ya que las comunicaciones no están cifradas. En la actualidad hay equipo asequible para realizar la captación y decodificación de las imágenes, así como un sinfín de aplicaciones de software libre y gratuitas. Con todo esto, este trabajo de fin de grado se basará en un sistema que se active cuando vaya a haber un pase, de cualquiera de los satélites NOAA19, NOAA18 y NOAA15, por encima de nuestra Facultad de Informática, capture todo el pase, lo descodifique y que a través de una página web de forma remota pueda mostrarse en las pantallas informativas de la facultad, todo ello en un sistema de pocos recursos como es una Raspberry Pi Model 3B.

Palabras clave

NOAA, Raspberry, RaspiNOAA, Satélites, Meteorología, Espectrograma, SDR

Abstract

FDISat: FDI's Satellite Imagery Acquisition Node

Throughout the day several meteorological satellites pass in a low earth orbit at an altitude of 850km above the Faculty of Informatics and our position. Of all of them, there are four (NOAA19, NOAA18, NOAA15 and METEOR-M2) that broadcast in the 137MHz band and are freely available since communications are not encrypted. At present, there is affordable equipment to capture and decode the images, as well as an endless number of free and open source software applications. With all this, this final degree work will be based on a system that is activated when there will be a pass of any of the satellites NOAA19, NOAA18 and NOAA15, above our Faculty of Computer Science, capture the entire pass, decode it and through a web page remotely can be displayed on the information screens of the faculty, all in a system of few resources such as a Raspberry Pi Model 3B.

Keywords

NOAA, Raspberry, RaspiNOAA, Satellites, Meteorology, Spectrogram, SDR

Índice

Índice de figuras

Capítulo 1

Introducción

"Tenemos que fallar aquí abajo, para no fallar allí arriba"
— Neil Armstrong

En astronomía un satélite es un objeto que orbita alrededor de otro objeto de mayor tamaño. Existen dos tipos de satélites: los naturales y los artificiales. Los satélites naturales los podemos encontrar en el universo, en nuestro sistema solar, y el más cercano y mejor conocido es la Luna que orbita la tierra. Por otro lado, tenemos los satélites artificiales que han sido creados por el ser humano para varios fines:

- Para enviar y recibir comunicaciones masivas como: TV, Internet...
- Para meteorología.
- Con fines militares.
- Para investigación científica.

El primer satélite artificial creado por el hombre fue lanzado por Rusia el 4 de octubre de 1957. Bautizado como Sputnik-1 (compañero de viaje), se trataba de un dispositivo muy simple: una bola de aluminio, con cuatro antenas y baterías.

Los satélites artificiales, como hemos dicho, orbitan alrededor de la tierra, y según la órbita que hagan pueden ser de dos tipos diferentes: geoestacionarios y polares.

- **Geoestacionarios:** son los que siguen una órbita de este a oeste por encima del ecuador. Siguen la dirección y velocidad de la rotación de la tierra.
- **Polares:** estos al contrario que los geoestacionarios viajan de polo a polo.

Ya conocemos el uso que se le puede dar al satélite y cómo se mueve, pero no sabemos a qué altura. Cuando se ponen en órbita son transportados hasta la altura deseada a través de un lanzamiento espacial, el cual cuando alcanza la altura deseada abandona el artefacto para siempre. Son 3 las órbitas más importantes:

- **Órbita baja terrestre:** están entre 700Km y 1.400Km de altura.

- **Órbita media terrestre:** estos se ubican entre los 9.000Km y los 20.000Km de altura.
- **Órbita alta terrestre:** es una órbita geocéntrica (qué tiene como centro de la órbita a la tierra) con una altura totalmente superior a los 35.786 kilómetros.

Con esta breve información ya podemos hacernos una idea de la cantidad de satélites que vuelan sobre nosotros cada día, cada uno de ellos con diferentes motivos. Ya que la mayoría de ellos son de uso privado las señales que emiten están cifradas y obviamente es ilegal descifrarlas sin el consentimiento de dicha empresa. Por suerte aún sigue habiendo satélites que no tienen las señales cifradas y son de libre captación: NOAA19, NOAA18, NOAA15 y METEOR-M2. Estos satélites son sobre todo usados con fines meteorológicos, ya que pasan capturando en una órbita polar (a unos 850KM sobre la altura del mar) imágenes meteorológicas de superficie terrestre y atmósfera (1).

Es por ello por lo que hay mucha gente interesada en captar estas imágenes que emiten de forma constante, ya sea por *hobby* o fines científicos. Y por esto mismo hay mucho software con el que hacerlo, mucho hardware que se puede usar y esto para una persona que quiere iniciarse en el maravilloso mundo de "La Captación de Señales De Satélites"puede llegar a ser tedioso, por eso este trabajo de fin de grado. Un sistema de fácil uso, sin mayores complicaciones y con la oportunidad además de poder visualizar estas señales de forma remota.

1.1. Motivación

Como hemos comentado en la introducción es mucho el software y el hardware que existe para la captación de las señales de los satélites y muchas las personas interesadas que sin conocimiento alguno quieren iniciarse. Por ello la mayor motivación de este proyecto es que con pocos recursos y casi sin ningún conocimiento puedan montar un sistema que capture, decodifique y guarde las imágenes emitidas para luego poder verlas, de forma que todo sea automático y que no requiera de casi (o nada) de ningún tipo de mantenimiento.

Otra de las motivaciones de este sistema es que pese a que existe mucho software, este se distinga por su sencillez y sobre todo por ser un sistema al que se puede acceder de forma remota a través de una web y que pueda ser expuesto en las pantallas informativas de la facultad a través de la red local de la universidad, con una interfaz cuidada y sencilla para el usuario final, al igual que una parte de administración que permite al administrador del sistema cambiar los ajustes de la aplicación a través de la web sin ni siquiera tener acceso a la Raspberry donde se tendrá la aplicación corriendo en segundo plano.

1.2. Requisitos

Los objetivos de este trabajo de fin de grado son los siguientes:

- **Escasos recursos:** ejecutar un sistema completo en un dispositivo con pocos recursos como es una Raspberry Pi Model 3B (Core 2 Quad a 1.2Ghz y 1GB de RAM)
- **Facilidad de uso:** un sistema que sea fácil de usar, que con una breve configuración inicial el usuario pueda capturar señales y con un mantenimiento casi nulo.
- **Accesible de forma remota:** que el sistema sea accesible a través de una página web para poder mostrar lo capturado de forma remota a través de una web, al igual que para modificar su configuración.
- **Atractivo para el usuario:** que la web que se muestre de forma remota sea fácil de visualizar, agradable a la vista e invite al usuario a que se pare a verla.

1.3. Plan de trabajo

El plan de trabajo utilizado para el desarrollo de este proyecto se puede definir en 4 etapas que se han ejecutado en el orden que se exponen:

- **Etapa 1:** obtener información acerca del uso y de cómo funcionan cada uno de los satélites. También en esta etapa se ha probado y estudiado en profundidad el software en el qué usaremos como base para el desarrollo: RaspiNOAA V2.
- **Etapa 2:** en esta etapa se ha diseñado sobre el papel cómo iba a ser la web, qué apartados y que funcionalidades iba a tener, así como la arquitectura y requisitos necesarios.
- **Etapa 3:** es la etapa de la programación, en esta etapa (la de mayor duración de las 4) se ha ido programando cada una de las funcionalidades que el sistema iba a tener
- **Etapa 4:** parte de pruebas, una vez creada una funcionalidad, se prueba y en caso de encontrar errores en la comprobación del correcto funcionamiento de las funcionalidades se solventan y se vuelve a probar de nuevo.

Durante cada una de las etapas se han ido programando reuniones con mis directores de proyecto con el fin de solventar dudas y de obtener posibles mejoras.

Capítulo 2

Introduction

"We need to fail down here so we don't fail up there."
— Neil Armstrong

In astronomy, a satellite is an object that orbits around a larger object. There are two types of satellites: natural and artificial. Natural satellites can be found in the universe, in our solar system, and the closest and best known is the Moon orbiting the Earth. On the other hand we have artificial satellites that have been created by humans for various purposes:

- To send and receive massive communications such as: TV, Internet...
- Meteorology.
- Military purposes.
- Scientific research.

The first man-made artificial satellite was launched by Russia on October 4, 1957. Named Sputnik-1 (traveling companion), it was a very simple device: an aluminum ball with four antennas and batteries.

Artificial satellites, as we have said, orbit around the earth, and depending on the orbit they make, they can be of two different types: geostationary and polar.

- **Geostationary:** these are those that follow an east-west orbit above the equator. They follow the direction and speed of the earth's rotation.
- **Polars:** these, unlike the geostationary ones, travel from pole to pole.

We already know the use that can be given to the satellite and how it moves, but we do not know at what height. When they are put into orbit they are transported to the desired altitude through a space launch, which when it reaches the desired altitude abandons the device forever. There are 3 major orbits:

- **Low Earth orbit:** they are between 700Km and 1,400Km high.

- **Mid-Earth orbit:** these are located between 9,000 km and 20,000 km in altitude.
- **High terrestrial orbit:** is a geocentric orbit (which has the earth as the center of the orbit) with an altitude totally higher than 35,786 kilometers.

With this brief information we can already get an idea of the amount of satellites that fly over us every day, each of them with different motives. Since most of them are for private use, the signals they emit are encrypted and it is obviously illegal to decrypt them without the consent of the company. Fortunately there are still satellites that do not have encrypted signals and are freely available: NOAA19, NOAA18, NOAA15 and METEOR-M2. These satellites are mainly used for meteorological purposes, since they spend capturing in a polar orbit (about 850KM above sea level) meteorological images of the Earth's surface and atmosphere (1).

That is why there are many people interested in capturing these images that emit constantly, either for hobby or scientific purposes. And for this reason there is a lot of software to do it, a lot of hardware that can be used and this for a person who wants to start in the wonderful world of "Satellite Signals Captureçan become tedious, that is why this final degree work. An easy to use system, without major complications and with the opportunity to visualize these signals remotely.

2.1. Motivation

As we mentioned in the introduction there is a lot of software and hardware that exists for the capture of satellite signals and many interested people without any knowledge want to get started. Therefore, the main motivation of this project is that with few resources and almost no knowledge they can assemble a system that captures, decodes and saves the emitted images for later viewing, so that everything is automatic and does not require almost (or none) of any maintenance.

Another of the motivations of this system is that although there is a lot of software, this is distinguished by its simplicity and above all for being a system that can be accessed remotely via a web and can be displayed on the information screens of the faculty through the local network of the university, with a neat and simple interface for the end user, as well as an administration part that allows the system administrator to change the settings of the application through the web without even having access to the Raspberry where the application will be running in the background.

2.2. Requirements

The objectives of this final thesis are the following:

- **Low resources:** running a complete system on a device with few resources such as a Raspberry Pi Model 3B (Core 2 Quad at 1.2Ghz and 1GB of RAM)
- **Ease of use:** a system that is easy to use, that with a brief initial setup the user can capture signals and with almost zero maintenance.
- **Remotely accessible:** the system should be accessible through a web page to be able to display the captured data remotely through a web page, as well as to modify its configuration.
- **Attractive for the user:** that the web that is displayed remotely is easy to visualize, pleasing to the eye and invites the user to stop and look at it.

2.3. Workplan

The work plan used for the development of this project can be defined in 4 stages that have been executed in the order shown below:

- **Stage 1:** obtain information about the use and how each of the satellites work. Also at this stage we have tested and studied in depth the software on which we will use as a basis for development: RaspiNOAA V2.
- **Stage 2:** in this stage we designed on paper what the website was going to look like, which sections and functionalities it was going to have, as well as the architecture and necessary requirements.
- **Stage 3:** this is the programming stage, in this stage (the longest of the 4 stages) each of the functionalities that the system will have has been programmed.
- **Stage 4:** testing part, once a functionality is created, it is tested and in case of finding errors in the verification of the correct operation of the functionalities, they are solved and tested again.

During each of the stages, meetings have been scheduled with my project managers in order to solve doubts and obtain possible improvements.

Capítulo 3

Estado del arte

En este capítulo se va a explicar en qué estado se encuentran actualmente los satélites y otros programas y el porqué es necesario un software monolítico como el que presentamos.

3.1. Satélites NOAA

La Administración Nacional Oceánica y Atmosférica (NOAA) tiene una larga trayectoria en la operación de satélites geoestacionarios (GOES) y satélites ambientales en órbita polar (POES).

Durante más de 50 años, estos satélites han sido fundamentales para los pronósticos meteorológicos y climáticos, así como para recopilar datos ambientales utilizados en diversas aplicaciones.

Uno de los satélites mas importantes es el NOAA-15, lanzado en 1998 como parte de la serie TIROS de la NASA. A pesar de que su vida útil originalmente se estimaba en dos años, continúa en funcionamiento y es ampliamente utilizado por estaciones de recepción civiles. Sin embargo, también ha tenido problemas operativos debido a una pérdida de lubricante en su motor de exploración para el sensor de imágenes AVHRR. Desde 2020, el satélite NOAA-15 opera como satélite AM secundario para NOAA-20.

El satélite NOAA-18, lanzado en 2005, cuenta con instrumentos similares a su contemporáneo NOAA-17 y cubre la otra mitad del día. Aunque algunos de sus instrumentos hayan fallado, como los giroscopios láser MIMU y las imágenes AVHRR, sigue operativo a día de hoy.

NOAA-19, también conocido como NOAA-N-Prime, fue lanzado en 2009. Una vez puesto en orbita tuvo fallos en sus instrumentos debido a sus problemas de fabricación en 2003. Desde 2020, todos los instrumentos a bordo del satélite funcionan correctamente (2).

3.2. Software para captación

Hoy en día son varias las distintas opciones que tenemos sobre Linux para la captación de señales. En el mundo que nos rodea son muchos los aficionados a la captación de radio y que por ello utilizan software SDR. Solo nos basta hacer una pequeña búsqueda en internet para darnos cuenta la cantidad de programas y software libre que se puede usar.

Tras haber analizado alguno de ellos, en nuestro caso hemos analizado 3 de ellos, siempre hemos llegado a la misma conclusión: los programas SDR por si solos no bastan.

En nuestro caso, para la captación de señales de NOAA lo primero que tenemos que hacer es saber la hora y el día a la que van a pasar sobre nuestra localización. Para ello no hay una herramienta en concreto, pero sí que hay varias webs que introduciendo nuestra longitud y latitud nos puede decir los siguientes pases.

Con esto en mente ya sabemos en qué momento va a pasar, ahora solo nos falta un programa SDR corriendo en Linux (como es nuestro caso). No son pocas las opciones que tenemos, de hecho, para alguien que sea nuevo se puede abrir un abanico muy grande de posibilidades cada uno con sus pros y contras. Nos vamos a centrar en 2 programas que son de los más conocidos como son GQRX y CubicSDR. Ambos programas son de licencia abierta y es muy fácil encontrarlos en la web para descargarlos, al igual que también hay infinidad de tutoriales. En ambos programas al abrirlos podemos ver las interfaces son muy parecidas:

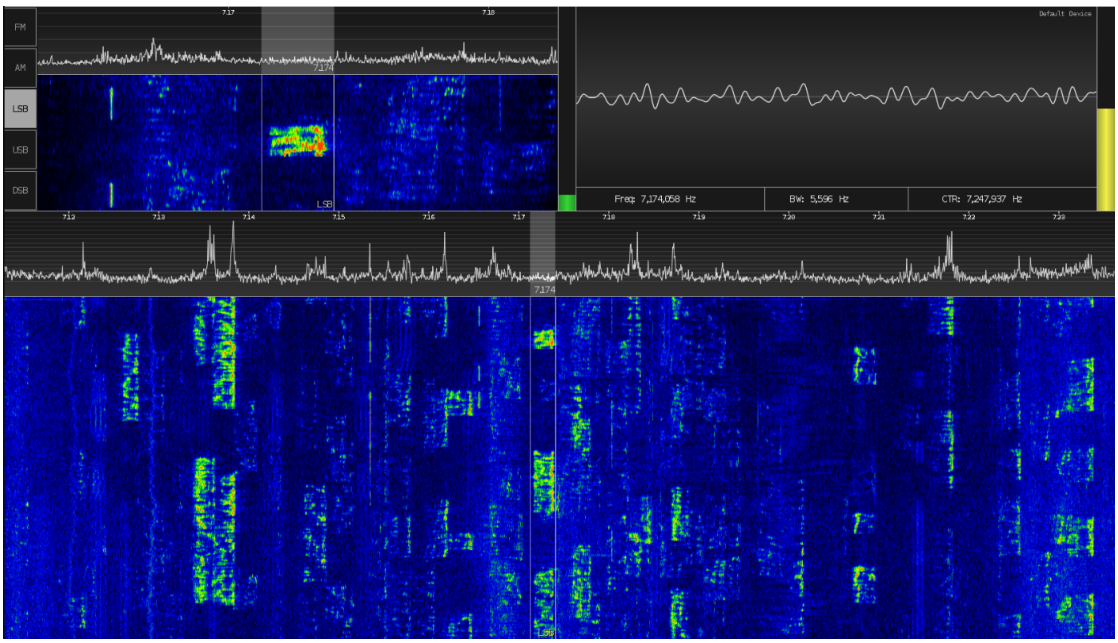

Figura 3.1: Interfaz del programa CubicSDR

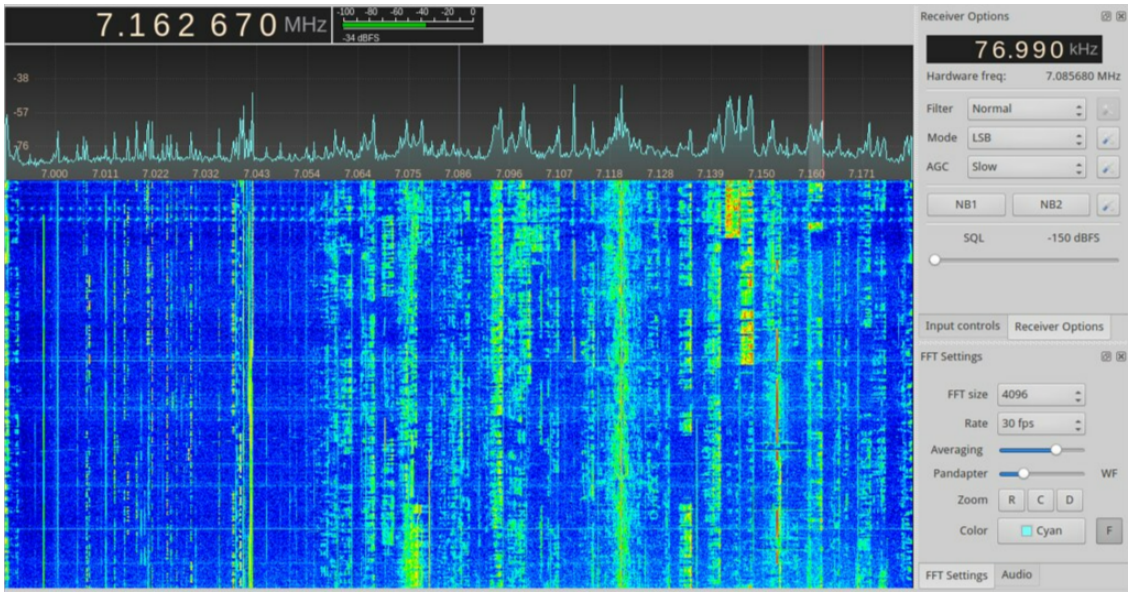

Figura 3.2: Interfaz del programa GQRX

En ambas podemos ver en tiempo real las señales de radio. Para ello tenemos que elegir la frecuencia a la que queremos escuchar y tan simple como ir grabando la señal en un .WAV durante el tiempo que dure el pase.

Ya hemos sabido a qué hora pasaba y hemos puesto nuestro programa SDR a escuchar, pero ahora nos hace falta otro software para la decodificación del .WAV creado. En concreto vamos a hablar de WXtoImg ya que posiblemente sea el mejor software para decodificación para señales de satélites meteorológicos como los NOAA. Es un software libre, aunque hoy en día no es fácil instalarlo en Linux ya que hay muchas librerías que son necesarias y que no se instalan de forma de automática. Tampoco tiene una interfaz gráfica por lo que para hacer uso de él es necesario la línea de comandos, es decir, un comando para cada una de las decodificaciones que queramos hacer ya que el sistema nos ofrece distintas capas para añadir (3)(4)(5).

Con el fin de simplificar este proceso, nace RaspiNOAA, el software en el que está basado nuestro proyecto y que es un software monolítico. Ya tiene preinstaladas las herramientas necesarias tanto para la escucha como para la decodificación. Además, tiene la particularidad de que el software se puede programar para que cuando haya un pase se active, lo que hace que no esté todo el tiempo capturando la señalas hasta que pase. Por otro lado, también es muy importante la BBDD que gestiona ya que gracias a esto podemos hacer una web para que toda la información capturada sea visualizada de forma remota y también que de forma remota se pueda configurar RaspiNOAA.

Capítulo 4

Hardware

En esta sección vamos a hablar del hardware necesario para llevar a cabo la captación de señales de satélites. Cabe mencionar que este hardware junto a otras herramientas se puede utilizar también para la captación de señales de radio en diferentes frecuencias y/o para la captación de televisión como es el TDT.

4.1. Raspberry Pi

Raspberry Pi es una placa de microcomputador, es de pequeñas dimensiones a la cual se le pueden dar multitud de usos como veremos más adelante. La Raspberry Pi apareció en febrero de 2012. La primera unidad tenía 256 MB de RAM y un procesador a 700 MHz, tenía el característico conector de 26 pines GPIO y salida de vídeo por HDMI o RCA además de un conector de 3.5mm para el audio, el primer modelo carecía de puerto ethernet.

La fundación Raspberry Pi quería hacer llegar la informática a todos los usuarios, con esta sencilla placa, pero suficiente para hacer las tareas más comunes de cualquier ordenador. También se ha introducido para la enseñanza de informática en escuelas.

El microcomputador Raspberry Pi ha pasado por varias versiones de mejora, aunque sigue manteniendo un precio bastante competitivo para el rendimiento que ofrece (aproximadamente 35-40 euros) Para funcionar utiliza un sistema operativo, el cual se instala en una microSD que actúa de memoria secundaria en la que se carga el sistema operativo. El sistema operativo más comúnmente usado es Raspbian qué está basado en Debian y en especial para aprovechar el hardware de este microcomputador. Además de Raspbian, multitud de sistemas han aparecido que hacen de la Raspberry Pi un microcomputador para diferentes fines, centros multimedia, emuladores de máquinas recreativas, etc.

En nuestro caso hemos elegido unas Raspberry Pi Model 3B (año de lanzamiento 2016). Esto se debe a que hemos elegido el modelo más bajo posible o con menos recursos para que el sistema funcione de manera estable y que tiene las siguientes características (las más relevantes para este proyecto) (6):

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 Wi-Fi LAN y Bluetooth Baja Energía (BLE)
- 100 Base Ethernet
- 4 Puertos USB version 2.0
- HDMI

Figura 4.1: Imagen de una Rasberry Pi 3 Model B

4.2. Receptor RTL-SDR

Un RTL-SDR es un receptor de radio de bajo costo que convierte señales analógicas en señales digitales utilizando el chip Realtek RTL2832u. Esto posibilita el procesamiento y análisis de las señales mediante software especializado. Estos dispositivos se emplean en diversas aplicaciones, como la captación de señales de televisión y radio, detección de radar y exploración de frecuencias no autorizadas.

El chip Realtek RTL2832U, desarrollado y fabricado por Realtek Semiconductor Corporation en Taiwán, constituye la base del funcionamiento del dispositivo RTL-SDR. Fue utilizado por primera vez en 2007 por un grupo de apasionados de la radioafición y la tecnología de radio definida por software (SDR). En aquel momento, llevaron a cabo una modificación no oficial del chip para su empleo en aplicaciones de SDR.
En concreto el que hemos utilizado para este proyecto pertenece a la empresa Nooelec en su versión NESDR SMArt SDR v5 y tiene un coste aproximado de unos 40€. Hemos utilizado este dispositivo porque es uno de los más comunes del mercado y encontrarlo a la venta es bastante fácil en distintos sitios web o en tiendas especializadas. Sus principales características son:

- Demodulador RTL2832U
- Interfaz USB
- Sintonizador R820T2/R860
- TCXO[1] de ruido de fase ultra bajo de 0,5 PPM
- Regulador de voltaje compatible con RF
- Entrada de antena SMA hembra
- Construido en aluminio para una mejor disipación del calor

También comentar que este receptor viene con una antena con el que se captarán las señales, cabe destacar que la antena que viene con el receptor es una antena básica (como la que se aprecia en la imagen de abajo) la cual ha dado unos resultados bastantes negativos ya que, en el centro de Madrid, donde se ha probado la antena apenas captaba un mínimo de señal que emitían los satélites a su paso debido a la cantidad de ruido e interferencias que producen el resto de ondas de radio que hay en el centro de la ciudad (8).

[1] "El TCXO es un oscilador de cristal de cuarzo que reduce la cantidad de cambio en la frecuencia de oscilación debido a los cambios de temperatura ambiente a través de un circuito de compensación de temperatura adicional (7)."

Figura 4.2: Bundle de Nooelec usado en el proyecto.

4.3. Cliente

Por último y no menos importante, nos hace falta un cliente. El objetivo de este cliente que puede ser un pc ya sea en su versión portátil, sobremesa o netbook, es la de conectarse a la Raspberry y así poder mostrar la información que vaya captando el sistema de forma remota. Cabe destacar que este equipo no requiere de grandes recursos por lo que por ejemplo se podría utilizar un equipo antiguo. Los únicos requisitos obligatorios son: conexión a internet, ya sea Wifi o Ethernet, que tenga también salida de vídeo al que conectar una pantalla, teclado y ratón.

Capítulo 5

Herramientas, tecnologías y protocolo SSH

En este capítulo vamos a describir todas las herramientas y tecnologías usadas durante el desarrollo de este proyecto. Todo ello bajo el sistema operativo Ubuntu.

5.1. Herramientas

Empezaremos describiendo las herramientas necesarias que se han utilizado durante todo el desarrollo del proyecto.

5.1.1. Visual Studio Code

Visual Studio Code es un editor de código fuente ligero pero eficaz que se ejecuta en el escritorio desarrollado por Microsoft. Este editor ha sido elegido por su gran facilidad de uso, ya que a lo largo de la carrera se ha fomentado su uso y por la cantidad de extensiones que tiene, que hace que programar sea más agradable.

5.1.2. Apache

Apache es un servidor web que se encarga de almacenar, procesar y servir las páginas web. Se distribuye bajo una licencia de código abierto.

5.1.3. SQLite

SQLite es una de las bases de datos relacionales más conocidas. Básicamente, funciona como un servidor propio e independiente, ya que el Sistema de Gerencia de Base de Datos se puede ejecutar en la misma instancia eliminando así las consultas y procesos separados. La biblioteca SQLite se genera y almacena directamente en el archivo de la base de datos.

5.1.4. Editor de texto y navegador

Aquí no se ha tenido ninguna preferencia sobre ninguno de ellos, se ha utilizado el editor de texto que viene por defecto y el navegador Chrome en su última versión. Cabe destacar que se podría haber utilizado cualquier otro editor/navegador.

5.2. Tecnologías

A continuación, se van a describir las tecnologías usadas en este proyecto para el desarrollo de la web.

5.2.1. HTML

HTML es el lenguaje con el que se define el contenido de las páginas web. Básicamente se trata de un conjunto de etiquetas que sirven para definir el texto y otros elementos que compondrán una página web, como imágenes, listas, vídeos, etc.

5.2.2. CSS

CSS *(Cascade Style Sheets)* qué significa: "hojas de estilo en cascada". Básicamente, es un lenguaje que define el diseño y la presentación de las páginas web, es decir, cómo se verá una página web cuando un usuario la visite. Funciona con el lenguaje HTML que maneja el contenido básico de la página.

5.2.3. PHP

PHP está enfocado principalmente a la programación de *scripts* del lado del servidor, por lo que se puede hacer cualquier cosa que pueda hacer otro programa CGI, como recopilar datos de formularios, generar páginas con contenidos dinámicos, o enviar y recibir *cookies*. Aunque PHP puede hacer mucho más, se ha elegido principalmente por su gran compatibilidad con el protocolo SSH que se explicará más adelante.

5.2.4. JavaScript

JavaScript es un lenguaje de programación que se utiliza para hacer páginas web interactivas. Las funciones de JavaScript pueden mejorar la experiencia del usuario de un sitio web y por ese motivo se ha utilizado también esta tecnología.

5.2.5. SQL

SQL es un acrónimo en inglés para *Structured Query Language*. Un Lenguaje de Consulta Estructurado. Un tipo de lenguaje de programación que te permite manipular y descargar datos de una base de datos.

5.3. Protocolo SSH

Por último, vamos a explicar el protocolo más usado en este proyecto y que sin cuya existencia de este hubiera complicado mucho el desarrollo de la web.
SSH, o *Secure Shell*, es un protocolo de red que nos permite conectarnos de forma segura a otras máquinas utilizando la línea de comandos. Con SSH, podemos establecer conexiones seguras con servidores a través de la red.

La principal ventaja de SSH es su seguridad. Todos los datos que se envían a través de SSH están encriptados, lo que significa que nadie más puede leer la información mientras viaja por la red. Esto garantiza que nuestras comunicaciones sean confidenciales y evita que alguien pueda robar información o contraseñas de acceso a los servidores.
Además de la seguridad, SSH también se utiliza para transferir archivos entre máquinas. Por ejemplo, el comando SCP utiliza SSH para realizar transferencias seguras de archivos. y que tendrá un papel muy importante es nuestro desarrollo (9).

Capítulo 6

FDISat

6.1. RaspiNOAA

La base de nuestro proyecto es RaspiNOAA. RaspiNOAA es un software libre de código abierto creado por usuarios que querían un software monolítico, es decir querían unificar todos esos programas que por separado eran complicados y tediosos para construir al contrario uno simple y funcional. El sistema operativo sobre el que se construye RaspiNOAA es Raspbian, basado en Debian. Aunque la idea es muy buena y tiene unos resultados sorprendentes, RaspiNOAA simplemente captura, decodifica y guarda las imágenes de los pases. Aquí llega nuestra primera parte a solventar: conocer al cien por cien cómo está construido RaspiNOAA para aprovechar todas esas cualidades que tiene. Lo primero que hacemos es descargarnos una imagen de arranque que se graba en la microSD. Esto lo descargaremos de la página oficial y usaremos el programa Etcher, que es un programa que nos creará todas las particiones necesarias en la microSD y volcará la imagen que nos hemos descargado previamente para crear RaspiNOAA.

Una vez cargado todo el sistema RaspiNOAA en nuestra microSD ha llegado el momento de encenderlo. Lo primero que veremos será el escritorio, ahí ya nos damos cuenta de que es un sistema que está creado por y para la captación de señales de satélites.

En el escritorio vemos que hay un *widget* en la parte derecha que aporta información sobre el sistema, carga de trabajo, temperatura y horario entre otros. Esto es Conky (10), un *widget* muy liviano y usado en los sistemas basados en Linux. Este mismo *widget* tiene también un tema personalizado, en el que podemos ver información de las zonas diurnas y nocturnas de nuestro planeta respecto a la posición del sol y alguna información relevante a nuestra localización. También vemos anclados en la barra de tareas algunos programas, que, aunque los mencionemos aquí a modo informativo, no se han utilizado para nuestro sistema:

- **HamClock:** es una aplicación estilo quiosco que proporciona información meteorológica espacial en tiempo real, modelos de propagación de radio, eventos operativos y otra información especialmente útil para los radioaficionados.

- **GPredict:** es un programa para el seguimiento en tiempo real y predicción de órbitas para satélites (11).

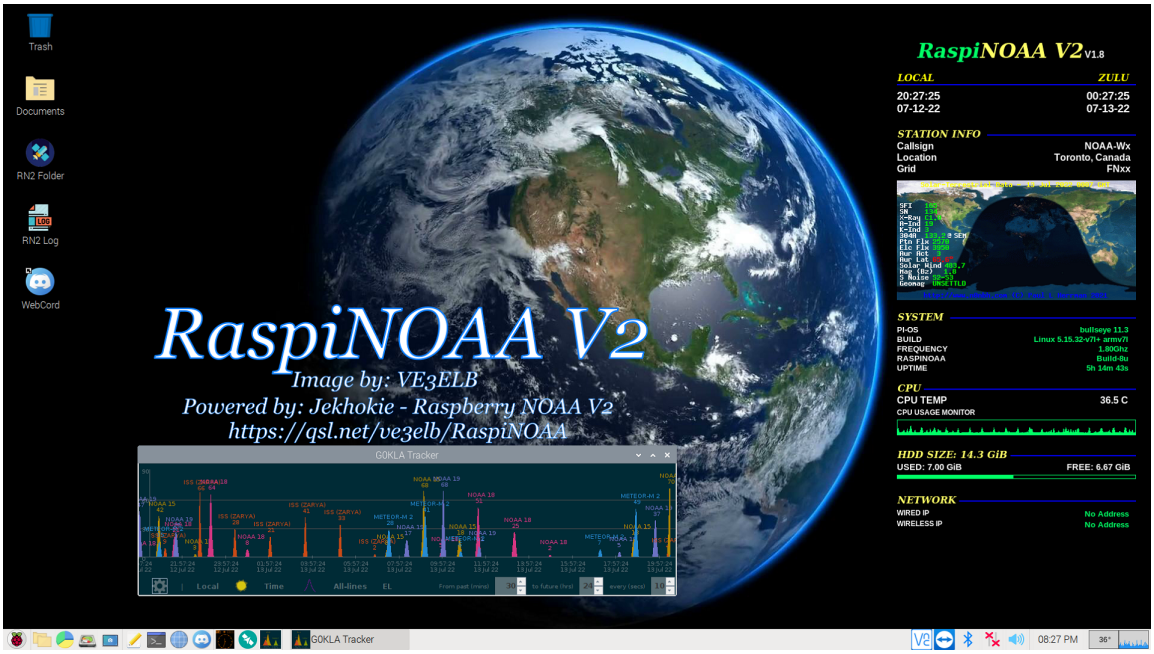

Figura 6.1: Escritorio de RaspiNOAA

Con todo esto, vamos a ver cómo funciona realmente este software monolítico. Mirando en el limitado manual que dejaron los creadores, y siguiendo unos pasos de configuración previos, que más tarde recopilaremos en el Apéndice A, existe una carpeta en la ruta `/home/pi/raspberry-noaa-v2/`. En esta carpeta encontramos unas subcarpetas de gran importancia que se irán explicando en la siguiente sección. Nos vamos a la ruta `/home/pi/raspberry-noaa-v2/config` y abrimos el fichero `settings.yml`. En este fichero encontramos la configuración de la que se alimentará el sistema y que tenemos que modificar para que funcione de forma correcta.

Una vez relleno, el manual indica que volvamos a la ruta padre: `/home/pi/raspberry-noaa-v2/` y ejecutemos un *scripts* llamado: `install_and_upgrade.sh` que realiza las comprobaciones necesarias de que todo esté instalado, de que el archivo modificado previamente tenga el formato correcto y a partir de ahí carga el sistema. Ya tenemos el sistema operativo. Solo enchufar el RTL-SDR, colocar la antena y a esperar los pases (12)(13).

6.2. Funcionalidades y desarrollo

Con todo esto preparado y funcionando, llega el momento de crear la web. Tras analizar en profundidad todo RaspiNOAA nos damos cuenta de que lo más importante son dos elementos: la base de datos que maneja es SQLite3 y necesitamos también una tecnología que nos permita transferir archivos entre RaspiNOAA y el cliente, de forma segura. Con estos dos requisitos se ha optado por realizar el desarrollo en su mayor parte con PHP y JavaScript.

6.2.1. Inicio

El objetivo de esta funcionalidad, que se encuentra en el fichero: `index.php` en la raíz del proyecto, se divide en 3 zonas, a parte de la barra superior de navegación.

El primero de ellos es la parte izquierda de la pantalla, que se desarrolla en `cargaFotosDiapositivas.php` donde se mostrarán las imágenes capturadas por el sistema del último pase que se haya capturado. Primero se debe saber que RaspiNOAA tiene una base de datos en el archivo de ruta: `/home/pi/raspberry-noaa-v2/db/panel.db` que contiene 2 tablas: `decoded_passes y predict_passes`, en nuestro caso nos interesa `decoded_passes` que es donde se almacena información relativa acerca de todos los pases que han sido capturados, en concreto nos interesa el último y para ello ejecutamos la query: `SELECT * FROM decoded_passes ORDER BY pass_start DESC LIMIT 1` que nos devolverá todos los campos de la última captura realizada.

Con todos estos campos que hemos extraído de la BBDD y sabiendo que todas las imágenes que se decodifican se guardar en la ruta: `srv/images` (esté parámetro se puede modificar, se explicará más adelante) ejecutamos el comando `ssh2_scp_recv` para traernos las nuevas imágenes creadas por RaspiNOAA.

Una vez recibidas las imágenes, y con los datos extraídos de la BBDD ya podemos cargar las imágenes que nos interesan a través del campo `file_path` con el comando `glob` de PHP. Ahora con JavaScript creamos una presentación de las fotografías que cada 10 segundos se vayan mostrando cada una de ellas.

```php
<?php

    $config = parse_ini_file("./config.ini");

    $connection = ssh2_connect($config['ip'], $config['puerto']);
    if (ssh2_auth_password($connection, $config['username'], $config['password'])) {
        $stream = ssh2_scp_recv($connection, '/home/pi/raspberry-noaa-v2/db/panel.db', './database/panel.db');

        $bbdd = "./database/panel.db";
        $db = new SQLite3($bbdd);
        $stmt = $db->prepare('SELECT * FROM decoded_passes ORDER BY pass_start DESC LIMIT 1');
        $result = $stmt->execute();

        $row = $result->fetchArray(SQLITE3_ASSOC);
        chdir("./images/");
        $stream = ssh2_scp_recv($connection, '/srv/images/'.$row['file_path']."-polar-direction.png", $row['file_path'].'-HVCT.jpg');
        $stream = ssh2_scp_recv($connection, '/srv/images/'.$row['file_path'].'-HVCT.jpg', $row['file_path'].'-HVCT.jpg');
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-histogram.jpg", $row['file_path']."-histogram.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-HVCT-precip.jpg", $row['file_path']."-HVCT-precip.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-HVCT.jpg", $row['file_path']."-HVCT.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-MCIR-precip.jpg", $row['file_path']."-MCIR-precip.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-MCIR.jpg", $row['file_path']."-MCIR.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-MSA-precip.jpg", $row['file_path']."-MSA-precip.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-MSA.jpg", $row['file_path']."-MSA.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-NO.jpg", $row['file_path']."-NO.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-pristine.jpg", $row['file_path']."-pristine.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-sea.jpg", $row['file_path']."-sea.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-MCIR.jpg", $row['file_path']."-MCIR.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-therm.jpg", $row['file_path']."-therm.jpg");
        $stream = ssh2_scp_recv($connection, "/srv/images/".$row['file_path']."-ZA.jpg", $row['file_path']."-ZA.jpg");
    }

    $imagenes = false;

    echo "<div class = 'corner1' id = 'corner1'>";
    foreach (glob($row['file_path']."-*") as $nombre_fichero) {
        echo "<img class='slide' src=./images/".$nombre_fichero.">";
        $imagenes = true;
    }

    if (!$imagenes){
        echo "<div class = 'corner1' id = 'corner1'>";
        echo "<img class='slide'>";
    }
    echo "</div>";

    chdir("..");
    $db->close();
    ssh2_disconnect($connection);

?>
```

Figura 6.2: Código de `cargaFotosDiapositivas.php`

```
function showDivs(n) {

    var i;
    var x = document.getElementsByClassName("slide");
    if (n > x.length) {slideIndex = 1}
    if (n < 1) {slideIndex = x.length}
    for (i = 0; i < x.length; i++) {
        x[i].style.display = "none";
    }
    x[slideIndex-1].style.display = "block";

    if (x[slideIndex-1].src.includes("NOAA-19")){

        if (!document.getElementById("imgInfoSatellite")){
            var img = document.createElement("img");
            img.src = "./info/NOAA19.png";
            img.id = "imgInfoSatellite";
            img.className = "imgInfoSatellite";

            document.getElementById("corner2Image").appendChild(img);
        } else {
            var img = document.getElementById("imgInfoSatellite");
            img.src = "./info/NOAA19.png";
        }

    } else if (x[slideIndex-1].src.includes("NOAA-18")){

        if (!document.getElementById("imgInfoSatellite")){
            var img = document.createElement("img");
            img.src = "./info/NOAA18.jpg";
            img.id = "imgInfoSatellite";
            img.className = "imgInfoSatellite";

            document.getElementById("corner2Image").appendChild(img);
        } else {
            var img = document.getElementById("imgInfoSatellite");
            img.src = "./info/NOAA18.jpg";
        }
    }
}
```

Figura 6.3: Código JS que crea la presentación de imágenes.

La segunda zona de la web es la esquina superior derecha. Aquí podemos ver una pequeña tabla con algunos campos y una imagen de un satélite. Esta es la información del último pase capturado que se relaciona con la parte izquierda de la pantalla. No nos extenderemos demasiado en esta parte ya que únicamente obtiene de la BBDD la información del último pase como lo explicamos arriba, se le da formato a la información y se muestra por pantalla. Esta parte de `index.php` está desarrollada en el fichero `tablaPaseActual.php`.

```php
<?php

    $config = parse_ini_file("./config.ini");

    $connection = ssh2_connect($config['ip'], $config['puerto']);
    if (ssh2_auth_password($connection, $config['username'], $config['password'])) {
        $stream = ssh2_scp_recv($connection, '/home/pi/raspberry-noaa-v2/db/panel.db', './database/panel.db');
    }

    $bbdd = "./database/panel.db";
    $db = new SQLite3($bbdd);
    $actual = floor(microtime(true));
    $stmt = $db->prepare('SELECT * FROM predict_passes WHERE pass_end < :actual ORDER BY pass_end DESC LIMIT 1');
    $stmt->bindValue(':actual', $actual, SQLITE3_INTEGER);
    $result = $stmt->execute();
    $row = $result->fetchArray(SQLITE3_ASSOC);

    echo "<table class = 'tablePase'>
    <tr class='cabecerasActual'>
        <td>Satélite</td>
        <td>Comienzo</td>
        <td>Final</td>
        <td>Elevación</td>
        <td>Azimut</td>
        <td>Dirección</td>
    </tr>
    <tr>";
        echo "<td>".$row["sat_name"]."</td>";
        echo "<td>".date("d/m/Y H:i:s", $row["pass_start"])."</td>";
        echo "<td>".date("d/m/Y H:i:s", $row["pass_end"])."</td>";
        echo "<td>".$row["max_elev"]."</td>";
        echo "<td>".$row["azimuth_at_max"]."</td>";
        if ($row["direction"] == 'Southbound'){
            echo "<td>Sur</td>";
        } else {
            echo "<td>Norte</td>";
        }
    echo "</tr>
    </table>";

    $db->close();
    ssh2_disconnect($connection);

?>
```

Figura 6.4: Código de `tablaPaseActual.php`

La última y tercera zona es la esquina inferior derecha. Aquí tenemos un contador decreciente con el formato HORAS:MINUTOS:SEGUNDOS que es el tiempo restante que queda hasta el siguiente pase y una tabla con la información del siguiente satélite. Se explica en orden de carga. Lo que primero carga es la información del siguiente pase programado. Esta parte se desarrolla en el fichero `tablaPaseSiguiente.php`, básicamente el funcionamiento es muy similar a la otra tabla descrita anteriormente solo que en vez de realizar la query sobre la tabla `decoded_passes` se realiza sobre la tabla `predict_passes` y con la query: `SELECT * FROM predict_passes WHERE pass_start >:actual LIMIT 1` con esta sentencia obtenemos la información relevante solo y únicamente del siguiente pase. A continuación, se le da formato de tabla y se muestra.

```php
<?php

    $config = parse_ini_file("./config.ini");

    $connection = ssh2_connect($config['ip'], $config['puerto']);
    if (ssh2_auth_password($connection, $config['username'], $config['password'])) {
        $stream = ssh2_scp_recv($connection, '/home/pi/raspberry-noaa-v2/db/panel.db', './database/panel.db');
    }

    $bbdd = "./database/panel.db";
    $db = new SQLite3($bbdd);
    $actual = floor(microtime(true));
    $stmt = $db->prepare('SELECT * FROM predict_passes WHERE pass_start > :actual LIMIT 1');
    $stmt->bindValue(':actual', $actual, SQLITE3_INTEGER);
    $result = $stmt->execute();
    $row = $result->fetchArray(SQLITE3_ASSOC);
    echo "<table class = 'tablePaseSiguiente'>
    <tr class='cabecerasSiguiente'>
        <td>Satélite</td>
        <td>Comienzo</td>
        <td>Final</td>
        <td>Elevación</td>
        <td>Azimut</td>
        <td>Dirección</td>
    </tr>
    <tr>";
        echo "<td>".$row["sat_name"]."</td>";
        echo "<td id = 'horaInicial'>".date("d/m/Y H:i:s", $row["pass_start"])."</td>";
        echo "<td id = 'horaFinal' >".date("d/m/Y H:i:s", $row["pass_end"])."</td>";
        echo "<td>".$row["max_elev"]."</td>";
        echo "<td>".$row["azimuth_at_max"]."</td>";
        if ($row["direction"] == 'Southbound'){
            echo "<td>Sur</td>";
        } else {
            echo "<td>Norte</td>";
        }
    echo "</tr>
    </table>";

    $db->close();
    ssh2_disconnect($connection);

?>
```

Figura 6.5: Código de `tablaSiguientePase.php`

El contador se desarrolla en concreto directamente sobre `index.php` y es en JavaScript. La idea es sencilla, es a partir de la tabla que hemos cargado anteriormente obtener la hora a la que va a pasar e ir actualizando el contador decrementándolo cada segundo. En la figura 6.6, se muestra el código que realiza dicha función.

```javascript
function updateCountdown() {

    const NOW = new Date();
    const DURATION = reordenaFecha() - NOW;
    const REMAINING_HOURS = Math.floor((DURATION % MILLISECONDS_OF_A_DAY) / MILLISECONDS_OF_A_HOUR);
    const REMAINING_MINUTES = Math.floor((DURATION % MILLISECONDS_OF_A_HOUR) / MILLISECONDS_OF_A_MINUTE);
    const REMAINING_SECONDS = Math.floor((DURATION % MILLISECONDS_OF_A_MINUTE) / MILLISECONDS_OF_A_SECOND);

    if (REMAINING_HOURS < 10){
        SPAN_HOURS.textContent = '0' + REMAINING_HOURS;
    } else {
        SPAN_HOURS.textContent = REMAINING_HOURS;
    }
    if (REMAINING_MINUTES < 10){
        SPAN_MINUTES.textContent = '0' + REMAINING_MINUTES;
    } else {
        SPAN_MINUTES.textContent = REMAINING_MINUTES;
    }
    if (REMAINING_SECONDS < 10){
        SPAN_SECONDS.textContent = '0' + REMAINING_SECONDS;
    } else {
        SPAN_SECONDS.textContent = REMAINING_SECONDS;
    }

    if (REMAINING_HOURS == 0 && REMAINING_MINUTES == 0 && REMAINING_SECONDS == 0){
        window.location.href = "/functions/capturaPase.php";
    }

}

updateCountdown();
setInterval(updateCountdown, MILLISECONDS_OF_A_SECOND);
```

Figura 6.6: Código JavaScript que desarrolla el contador regresivo.

Una vez que el contador llega a 00:00:00 la página se auto redirige a la pantalla a `capturaPase.php` que es una ventana cortina con el único uso de mostrar un contador a pantalla completa de 35 minutos que es la media que tarda RaspiNOAA en capturar el pase. Cuando este contador llega a 00:00:00 se redirige a `index.php` para cargar de nuevo todo.

Figura 6.7: Cuenta regresiva cuando se captura un pase.

Para finalizar, la página web está programada para que cuando el administrador pulse la tecla énterén el teclado, se ponga a pantalla completa. Cuando se entra en este modo la barra de navegación superior se oculta con el fin de tener una web lo más limpia posible para el usuario final.

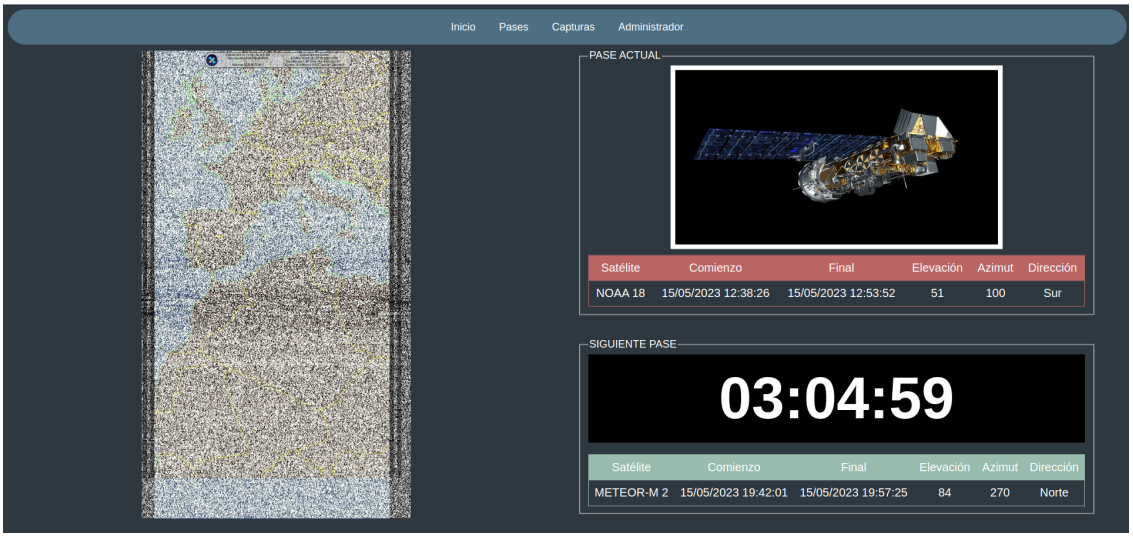

Figura 6.8: Vista de `index.php`

6.2.2. Capturas

Desarrollado en `captures.php` es una pantalla para el administrador. La funcionalidad reside en que aquí podremos ver todos los pases capturados por RaspiNOAA. Podemos ver en la pantalla, se ha escogido con el fin de tener una estética agradable y aprovechando que RaspiNOAA crea un mapa del pase, un *grid* con dicha información para después elegir un pase y poder ver todas las capturas en pantalla completa. En caso de que no se haya generado el mapa del pase, se cogerá como miniatura la primera imagen descodificada.
Con un `glob` obtenemos todas las imágenes y las agrupamos.
Con cada uno de los pases generamos un enlace a la página `galeriaFotos.php` el cual le pasaremos por parámetro de URL que facilita PHP el nombre del pase para cargar todas las imágenes que hemos ido almacenando. Para así poder ver todas las foto y si pulsamos sobre ellas se abrirán en nueva pestaña en el navegador.

```php
<?php
chdir("./images/");
$control = null;
foreach (glob("*") as $nombre_fichero) {

    $substrings = explode("-",$nombre_fichero);
    $name = $substrings[0] . "-" . $substrings[1] . "-" . $substrings[2] . "-" . $substrings[3];

    if ($control != $name){
        $polar = glob($name."-polar-direction.png");
        if (count($polar) == 0){
            $control = $name;
            $nombre_aux = glob($name."-HVCT.jpg");
            echo "<div class='packImagenes'>
            <div style>
            <a href='galeriaFotos.php?name=".$name."'><img src='./images/".$nombre_aux[0]."' width='100%'></a>
                <span>".$name."</span>
            </div>
            </div>";
        } else {
            $control = $name;
            echo "<div class='packImagenes'>
            <div style>
            <a href='galeriaFotos.php?name=".$name."'><img src='./images/".$polar[0]."' width='100%'></a>
                <span>".$name."</span>
            </div>
            </div>";
        }
    }
}
chdir("..");
?>
```

Figura 6.9: Código principal de `captures.php`

```php
<?php
chdir("./images/");
$name = $_GET['name'];
foreach (glob($name."*") as $nombre_fichero) {
    echo "<div class='packImagenes'>
            <div>
            <a href=./images/".$nombre_fichero." target='_blank'><img src='./images/".$nombre_fichero."' width='100%'></a>
            <span>".$nombre_fichero."</span>
            </div>
          </div>";
}
chrir("..");
?>
```

Figura 6.10: Código principal de `galeriaFotos.php`.

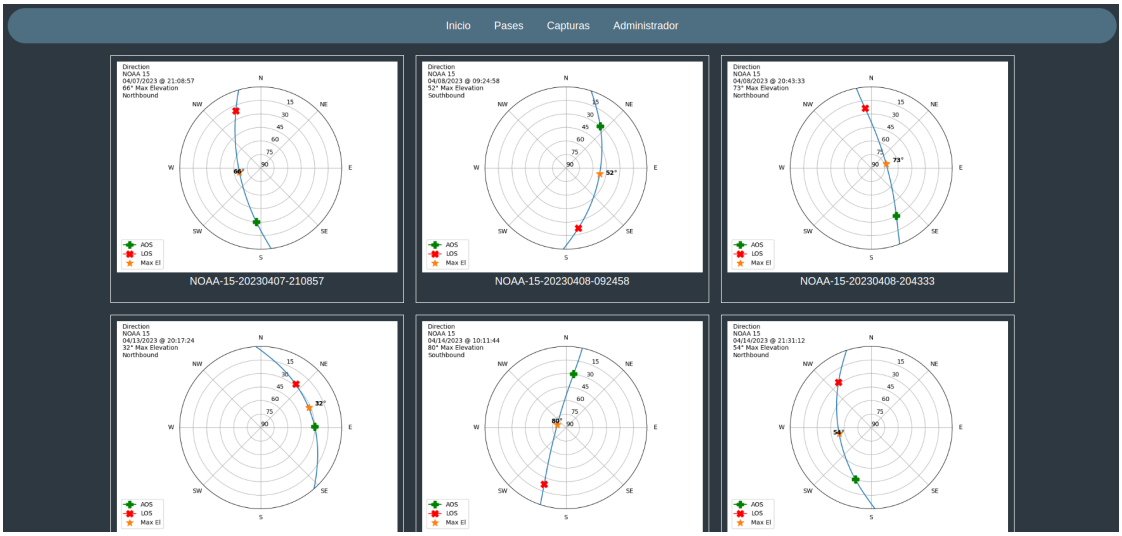

Figura 6.11: Vista de `captures.php`

6.2.3. Pases

Esta parte de la web es simplemente informativa. Desarrollada en `pases.php` únicamente nos conectamos por SSH a RaspiNOAA para obtener la BBDD y posteriormente acceder a la tabla `predict_passes` y con la query: `SELECT * FROM predict_passes` obtenemos todos los registros de todos los pases que están programados y con PHP separamos los pases programados agrupándolos por días.

```php
<?php

    $config = parse_ini_file("./config.ini");

    $connection = ssh2_connect($config['ip'], $config['puerto']);
    if (ssh2_auth_password($connection, $config['username'], $config['password'])) {
        $stream = ssh2_scp_recv($connection, '/home/pi/raspberry-noaa-v2/db/panel.db', './database/panel.db');
    }

    $bbdd = "./database/panel.db";
    $db = new SQLite3($bbdd);
    $sql = "SELECT * FROM predict_passes ORDER BY pass_start";
    $results = $db->query($sql);

?>
```

Figura 6.12: Conexión y obtención de la BBDD de RaspiNOAA

```php
<div style = "display: flex; justify-content: center; text-align: center; width: 100%">
<table class = "tablePases">
    <tr class="rowHeaders">
        <td>Satélite</td>
        <td>Comienzo</td>
        <td>Final</td>
        <td>Elevación</td>
        <td>Azimut</td>
        <td>Dirección</td>
    </tr>
    <?php
        $fecha = null;
        while($row = $results->fetchArray(SQLITE3_ASSOC) ) {
            if ($row["pass_end"] >= floor(microtime(true))){
                if ($fecha == null) {
                    $fecha = date("d/m/Y", $row["pass_start"]);
                    echo "<tr><td colspan='6'class='rowDate'>".$fecha."</td></tr>";
                }
                if (date("d/m/Y", $row["pass_start"]) > $fecha){
                    $fecha = date("d/m/Y", $row["pass_start"]);
                    echo "<tr><td colspan='6'class='rowDate'>".$fecha."</td></tr>";
                }
                echo "<tr>";
                    echo "<td>".$row["sat_name"]."</td>";
                    echo "<td>".date("d/m/Y H:i:s", $row["pass_start"])."</td>";
                    echo "<td>".date("d/m/Y H:i:s", $row["pass_end"])."</td>";
                    echo "<td>".$row["max_elev"]."</td>";
                    echo "<td>".$row["azimuth_at_max"]."</td>";
                    if ($row["direction"] == 'Southbound'){
                        echo "<td>Sur</td>";
                    } else {
                        echo "<td>Norte</td>";
                    }
                echo "<tr>";
            }
        }
    ?>
</table>
</div>
```

Figura 6.13: Formato y extracción de datos de la tabla principal de `passes.php`.

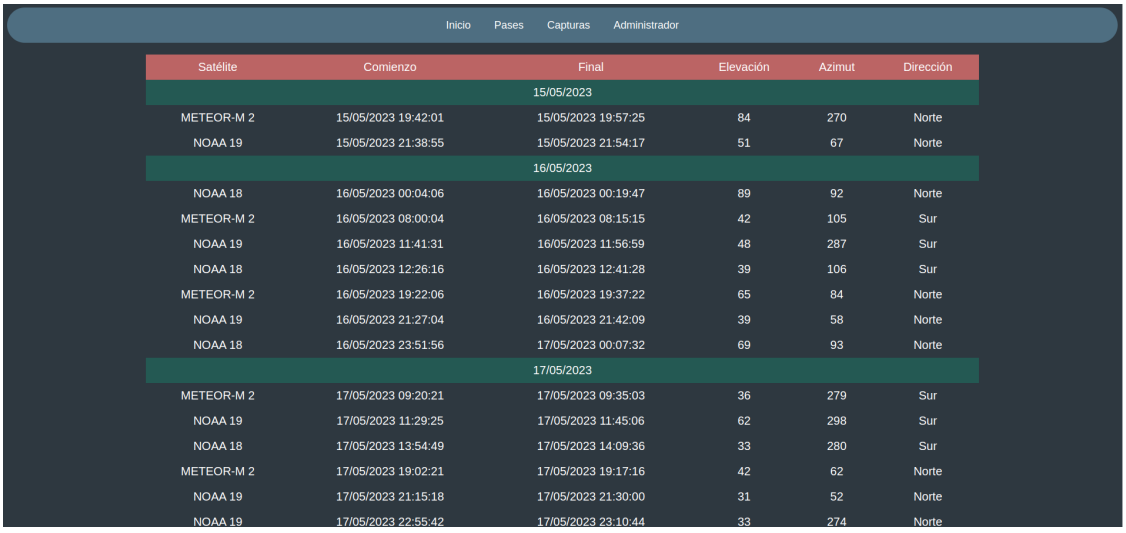

Figura 6.14: Vista de `passes.php`

6.2.4. Administrador

Para explicar esta parte, es importante explicar que RaspiNOAA está basado en *scripts*, son muchos los que contiene, pero hay uno en concreto que es el más importante: `install_and_update.sh` que es el encargado de comprobar `settings.yml`, comprobar los requisitos del sistema y cargar todos los demás. El punto negativo de este *scripts* es que su ejecución puede llegar a demorarse de 5 a 15 minutos. Una vez finalizado este *script* y tener el sistema completamente operativo para empezar a capturar pases, cabe destacar que genera un fichero `.noaa-v2.conf` con toda la configuración y del cual el resto de los *scripts* se apoyan.

A raíz de este conocimiento tenemos esta parte de la web. Es una parte exclusivamente orientada al administrador del sistema y una de las características más fuertes de este sistema que ninguno otro software posee hoy en día.

Para dotar al sistema de un mayor control se ha divido esta sección en dos. Por un lado tenemos la opción de **Guardado rápido** que se desarrolla en `adminGuardadoRapido.php`, `cargaRapidaConfiguracion.php`, `guardaRapidoConfiguracion.php`. Y por otro lado tenemos **Guardado completo** que se desarrollan en los ficheros `adminGuardadoCompleto.php`, `cargaCompletaConfiguracion.php`, `guardaCompletoConfiguracion.php`, `adminGuardadoCompletoLog.php`.

Ambas opciones están basadas en la misma interfaz de uso:

- **COLUMNA 1:** contiene los nombres de las variables de configuración.
- **COLUMNA 2:** contiene el valor que toma esa variable en el momento de carga.
- **COLUMNA 3:** muestra una explicación de la función que realiza esa variable.

El funcionamiento de **Guardado Rápido** es simple: se obtiene el archivo `.noaa-v2.conf` a través de SSH a RaspiNOAA, se lee y se carga en la tabla. A la hora de ir importando los valores del fichero, se hace que la columna 2 sea `contenteditable`, así el administrador puede modificar directamente el valor sobre la tabla. También carga la columna 3 que se carga de otro fichero `comentarios_noaa.txt` donde cada línea corresponde con una variable.
Hasta aquí ya tenemos el archivo de configuración leído, pasado a una tabla y además editable. Ahora falta guardarlo y enviarlo a RaspiNOAA. Para ello pulsamos sobre el botón abajo a la derecha que poner guardar configuración.

```php
<?php

    $config_file = file_get_contents("./config.ini");
    $config = parse_ini_string($config_file);

    $connection = ssh2_connect($config['ip'], $config['puerto']);
    if (ssh2_auth_password($connection, $config['username'], $config['password'])) {
        ssh2_scp_recv($connection, '/home/pi/.noaa-v2.conf', './config/.noaa-v2.conf');
    }

    $fp = fopen("./config/.noaa-v2.conf", "r");
    $fp2= fopen("./config/comentarios_noaa.txt", "r");
    while (!feof($fp)){
        $row = fgets($fp);
        if ($row != "\n"){
            $data = explode("=",$row);
            echo "<tr>";
            echo "<td style='font-weight: bold;'>".$data[0]."</td>";
            echo "<td><span id='config' contenteditable='true' style = 'font-size: 15px'>".$data[1]."</span></td>";
            echo "<td style = 'color: #ECA869; font-size: 15px; font-style: italic;'>".fgets($fp2)."</td>";
            echo "</tr>";
        }
    }

    fclose($fp);
    fclose($fp2);
    ssh2_disconnect($connection);

?>
```

Figura 6.15: Código de `cargaRapidaConfiguracion.php`.

```javascript
function saveFast(){
    tabla = document.getElementById("configuracion").rows;
    datos = [];
    for (var i = 0; i < tabla.length; i++) {
        fila = tabla[i].children;
        datosFila = [];
        for (var j = 0; j < fila.length-1; j++) {
            datosFila.push(fila[j].innerText);
        }
        datos.push(datosFila);
        console.log(datos);
    }
    $.ajax({
        url: './functions/guardaRapidoConfiguracion.php',
        type: 'POST',
        contentType: 'application/json',
        data: JSON.stringify(datos),
        dataType: 'json',
        success: window.alert("La configuración se ha guardado correctamente.")
    });
}
```

Figura 6.16: Código de JavaScript del botón y llamada Ajax.

```php
<?php

    $json = file_get_contents('php://input');
    $objeto = json_decode($json);
    $datos = json_decode($json, true);

    $config = array();

    chdir("..");
    $config_file = file_get_contents("./config.ini");
    $config = parse_ini_string($config_file);

    chdir("./functions");
    $fp = fopen(".noaa-v2.conf","a+");
    foreach($datos as $fila){
        fwrite($fp,$fila[0]."=".$fila[1]."\n");
    }
    fclose($fp);

    $connection = ssh2_connect($config['ip'], $config['puerto']);
    if (ssh2_auth_password($connection, $config['username'], $config['password'])) {
        ssh2_scp_send($connection, '.noaa-v2.conf', '/home/pi/.noaa-v2.conf');
    }

    unlink(".noaa-v2.conf");
    chdir("..");
    ssh2_disconnect($connection);
?>
```

Figura 6.17: Código de `guardaRapidoConfiguracion.php`.

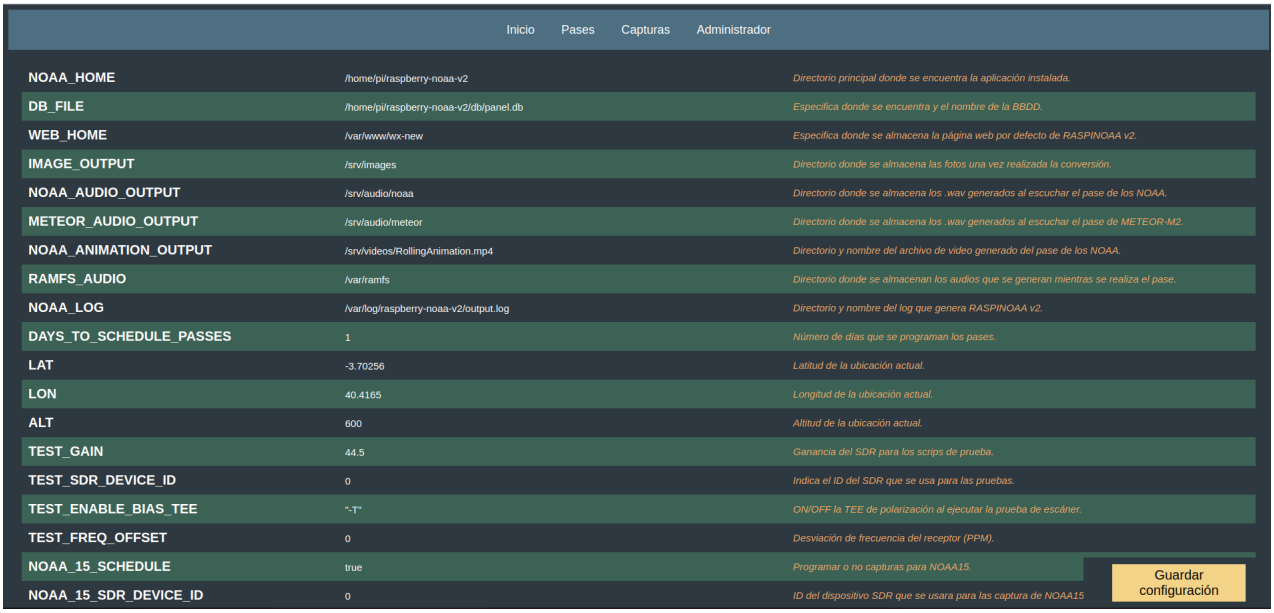

Figura 6.18: Vista de administrador para Guardado Rápido.

Ahora pasamos a la opción de **Guardado completo**. Se basa en la misma del anterior, pero con algunas diferencias: la configuración la cargamos desde el archivo `settings.yml`, la pasamos a una tabla editable, y del archivo `comentarios_settings.txt` cargamos las explicaciones de cada una de las variables, las guardamos y a continuación ejecutamos de forma remota el *script* `install_and_upgrade.sh`.

```php
<?php

    $config_file = file_get_contents("./config.ini");
    $config = parse_ini_string($config_file);

    $connection = ssh2_connect($config['ip'], $config['puerto']);
    if (ssh2_auth_password($connection, $config['username'], $config['password'])) {
        ssh2_scp_recv($connection, '/home/pi/raspberry-noaa-v2/config/settings.yml', './config/settings.yml');
    }

    $fp = fopen("./config/settings.yml", "r");
    $fp2= fopen("./config/comentarios_settings.txt", "r");
    while (!feof($fp) && !feof($fp2)){
        $row = fgets($fp);
        if (!str_contains($row, "#") && ($row != "\n") && (!str_contains($row, "---"))){
            $data = explode(": ",$row);
            echo "<tr>";
            echo "<td style='font-weight: bold;'>".$data[0]."</td>";
            echo "<td><span id='config' contenteditable='true' style = 'font-size: 15px'>".$data[1]."</span></td>";
            echo "<td style = 'color: #ECA869; font-size: 15px; font-style: italic;'>".fgets($fp2)."</td>";
            echo "</tr>";
        }
    }

    fclose($fp);
    fclose($fp2);
    ssh2_disconnect($connection);

?>
```

Figura 6.19: Código de `cargaCompletaConfiguracion.php`.

```javascript
<script>

    function save(){
        tabla = document.getElementById("configuracion").rows;
        datos = [];
        for (var i = 0; i < tabla.length; i++) {
            fila = tabla[i].children;
            datosFila = [];
            for (var j = 0; j < fila.length-1; j++) {
                datosFila.push(fila[j].innerText);
            }
            datos.push(datosFila);
            console.log(datosFila);
        }
        $.ajax({
            url: './functions/guardaCompletoConfiguracion.php',
            type: 'POST',
            contentType: 'application/json',
            data: JSON.stringify(datos),
            dataType: 'json',
            success: window.location.href = "adminGuardadoCompletoLog.php"
        });
    }

</script>
```

Figura 6.20: Código de JavaScript del botón y llamada Ajax.

```php
<?php

    $json = file_get_contents('php://input');
    $objeto = json_decode($json);
    $datos = json_decode($json, true);

    $config = array();

    chdir("..");
    $config_file = file_get_contents("./config.ini");
    $config = parse_ini_string($config_file);

    chdir("./functions");
    $fp = fopen("settings.yml","a+");
    fwrite($fp,"---\n");
    foreach($datos as $fila){
        fwrite($fp,$fila[0].": ".$fila[1]."\n");
    }
    fwrite($fp,"...\n");
    fclose($fp);

    $connection = ssh2_connect($config['ip'], $config['puerto']);
    $stream = NULL;
    if (ssh2_auth_password($connection, $config['username'], $config['password'])) {
        ssh2_scp_send($connection, 'settings.yml', '/home/pi/raspberry-noaa-v2/config/settings.yml');

        $stream = ssh2_exec($connection, 'cd ./raspberry-noaa-v2/; ./install_and_upgrade.sh > output.txt 2>&1');
    }

    unlink("settings.yml");
    chdir("..");
    ssh2_disconnect($connection);
?>
```

Figura 6.21: Código de `guardaCompletoConfiguracion.php`.

Figura 6.22: Vista de administrador para Guardado Completo.

Una vez realizado este proceso la página nos redirige a la página `adminGuardadoCompletoLog.php` que cada 10 segundos nos va mostrando la salida del *script* que hemos ejecutado de forma remota. Para poder hacer este proceso es necesario redirigir la salida del *script* ejecutado a un fichero que se crea llamado `output.txt` qué se irá leyendo cada 10 segundos, tiempo que la página toma para recargarse de forma automática. Cabe destacar que debido a la naturaleza de como está diseñado la redirección, no se le puede dar un formato al texto ya que no escribe saltos de línea, lo que hace que se vea menos estético aunque sigue cumpliendo su función: informar al usuario cuando acaba y si ha sido exitosa o fallida la ejecución.

```php
<?php
    $config = array();
    $config_file = file_get_contents("./config.ini");
    $config = parse_ini_string($config_file);

    $connection = ssh2_connect($config['ip'], $config['puerto']);
    if (ssh2_auth_password($connection, $config['username'], $config['password'])) {
        $stream = ssh2_exec($connection, "tail ./raspberry-noaa-v2/output.txt");
        stream_set_blocking($stream, true);
        $stream_out = ssh2_fetch_stream($stream, SSH2_STREAM_STDIO);
        $processStatus = stream_get_contents($stream_out);
        echo $processStatus;
    }
?>
```

Figura 6.23: Código de `adminGuardadoCompletoLog.php`.

Figura 6.24: Vista de la pantalla de salida de la ejecución.

Capítulo 7

Conclusiones y Trabajo Futuro

Como experiencia propia, no olvidaré mi primera reunión con mis tutores cuando me presentaron el trabajo de fin de grado. Me hablaron de satélites, ondas de radio, frecuencias, receptores, antenas, software para la decodificación, páginas web donde podía encontrar toda la información... En resumen, un cúmulo de información que para alguien que no está familiarizado con este tema ni que tenga un conocimiento sobre informática, telecomunicaciones o similar, puede llegar a ser muy tedioso.

Este proyecto trata de quitar todas esas barreras, trata de evitar una cantidad de información ingente que de primeras puede quitar las ganas de profundizar en el tema e incluso para alguien que lo intente. Por eso creo que hacía falta este proyecto: bajo coste, fácil instalación, fácil configuración, sin mantenimiento apenas y con una página web para que los resultados se puedan ver de forma remota.

Respecto a la web para visualizar de forma remota, puede ser un gran atractivo para cuando se utilice en las pantallas informativas de nuestra facultad (aunque se puede utilizar en cualquier otro entorno), llamando así la atención del personal y alumnos al poder ver en tiempo real la meteorología, e incluso quién sabe, llamarles la atención para involucrarse en este mundo de radioaficionados.

Como trabajo a futuro hay mucho margen de mejora aún, algunas de las ideas que pueden aportar un mayor valor al sistema son los siguientes:

Scripts: nueva creación de varios *scripts*, por ejemplo, un *scripts* inicial que compruebe todos los requisitos para que la web pueda funcionar de forma correcta y con un solo comando instale todo lo faltante y la web.

Meteor-M2: También se puede trabajar más sobre la captación, decodificación de este satélite ya que utiliza un protocolo distinto a los NOAA, lo que hace que todo el proceso requiera de mayores recursos y por ese motivo se ha dejado de lado en nuestro sistema (14).

Web: los tiempos avanzan, y lo que hoy es una web agradable, mañana deja de serlo, por lo que cambiar el aspecto de la web siempre es un trabajo a futuro. Tampoco nos podemos olvidar que siempre se pueden desarrollar nuevas funcionalidades

Capítulo 8

Conclusions and Future Work

As my own experience, I will never forget my first meeting with my tutors when I was presented with my final degree project. They told me about satellites, radio waves, frequencies, receivers, antennas, software for decoding, web pages where I could find all the information... In short, an accumulation of information that for someone who is not familiar with this subject or who has no knowledge of computers, telecommunications or similar, can become very tedious.
This project tries to remove all those barriers, it tries to avoid a huge amount of information that at first can take away the desire to delve into the subject and even for someone who tries. That's why I think this project was needed: low cost, easy installation, easy configuration, with almost no maintenance and with a web page so that the results can be viewed remotely.
Regarding the web for remote viewing, it can be a great attraction when used in the information screens of our faculty (although it can be used in any other environment), thus attracting the attention of staff and students to see the weather in real time, and who knows, even draw their attention to get involved in this world of amateur radio.
As future work there is still much room for improvement, some of the ideas that can bring more value to the system are the following:

Scripts: several scripts can be created, for example an initial script that checks all the requirements so that the web can work correctly and with a single command installs everything missing and the web.

Meteor-M2: more work can be done on the capture and decoding of this satellite since it uses a different protocol than the NOAA, which makes the whole process requires more resources and for that reason it has been left aside in our system (14).

Web: times move on, and what today is a nice web, tomorrow it is not, so changing the look of the web is always a future work. Nor can we forget that new functionalities can always be developed.

Bibliografía

[1] "Órbitas de los satélites @ONLINE." https://es.wikipedia.org/wiki/orbitas_de_satelites_artificiales.

[2] "Historia de los noaa @ONLINE." https://www.nesdis.noaa.gov/current-satellite-missions/history-of-noaa-satellites.

[3] "Gqrx @ONLINE." https://gqrx.dk/.

[4] "Cubicsdr @ONLINE." https://cubicsdr.com/.

[5] "Wxtoimg @ONLINE." https://wxtoimgrestored.xyz/.

[6] "Raspberry pi 3 model b @ONLINE." https://www.raspberrypi.com/products/raspberry-pi-3-model-b/.

[7] "Qué es un txco y para que sirve @ONLINE." http://www.sumerscience.com/info/what-is-a-temperature-compensated-crystal-osci-25519136.html.

[8] "Nooelec @ONLINE." https://www.nooelec.com/store/nesdr-smart.html.

[9] "Ssh que es y para qué sirve @ONLINE." https://www.arsys.es/blog/ssh.

[10] "Conky @ONLINE." https://drive.google.com/file/d/102sh3wL6gnH9uKj3F04b3F5JItPgEwqj/view.

[11] "Gpredict @ONLINE." https://drive.google.com/file/d/1-cmcT_r8Ug0HYN9iA603nlkyMDIi823c/view.

[12] "Raspinoaa @ONLINE." https://qsl.net/ve3elb/RaspiNOAA/.

[13] "Manual raspinoaa @ONLINE." https://drive.google.com/file/d/10AQjXmIC8PAUrak-_h4gfxi_hdD8h1o7/view.

[14] "Manual daemonmeteor @ONLINE." https://drive.google.com/file/d/10OFipRYGWIgxFTUnqXa-QcrrDQYSOvlm/view.

Apéndice A

Manual de uso: RaspiNOAA desde cero

A.1. Descarga y grabado de imagen en microSD

Se recomienda tener una microSD de al menos 16GB de memoria y de clase A1 o A2.

1. Accedemos a la página oficial de RaspiNOAA V2: `https://qsl.net/ve3elb/RaspiNOAA/` y pulsamos sobre donde pone descargar.

Figura A.1: Página web oficial de RaspiNOAA.

2. Una vez pulsado ese botón nos redirige a una carpeta compartida en Google Drive. Ahí descargamos el archivo que se encuentra marcado en la imagen. Puede que tarde unos minutos en descargarse dependiendo de la conexión.

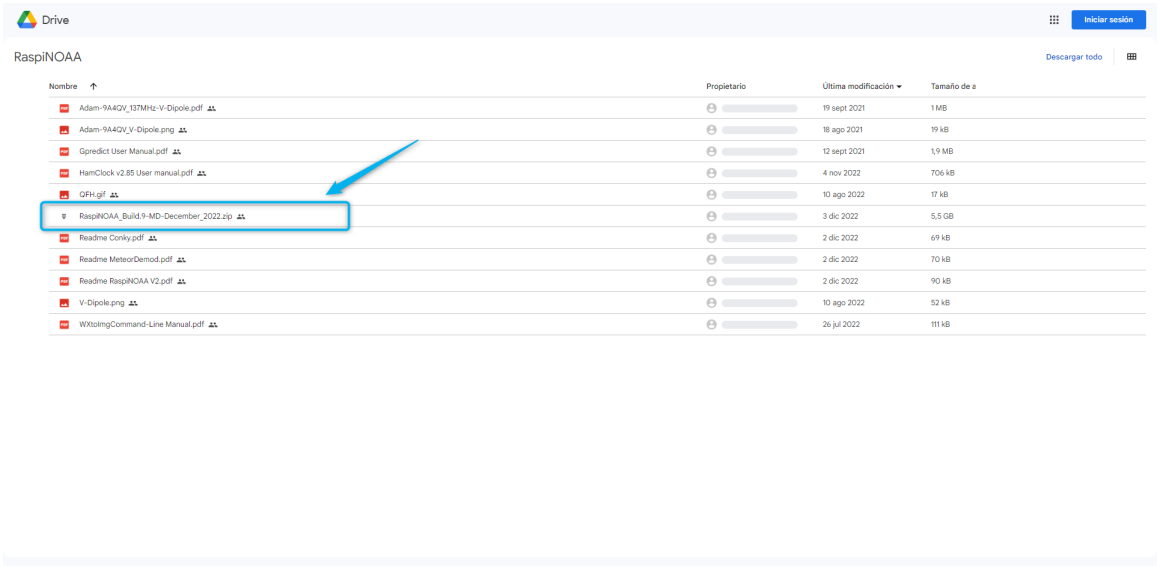

Figura A.2: Carpeta compartida de Google Drive de RaspiNOAA.

3. Mientras se descarga, vamos a descargar Etcher que es el software que nos ayudará a grabar la imagen en la microSD: `https://etcher.balena.io/#download-etcher`. Se recomienda la versión *portable*.

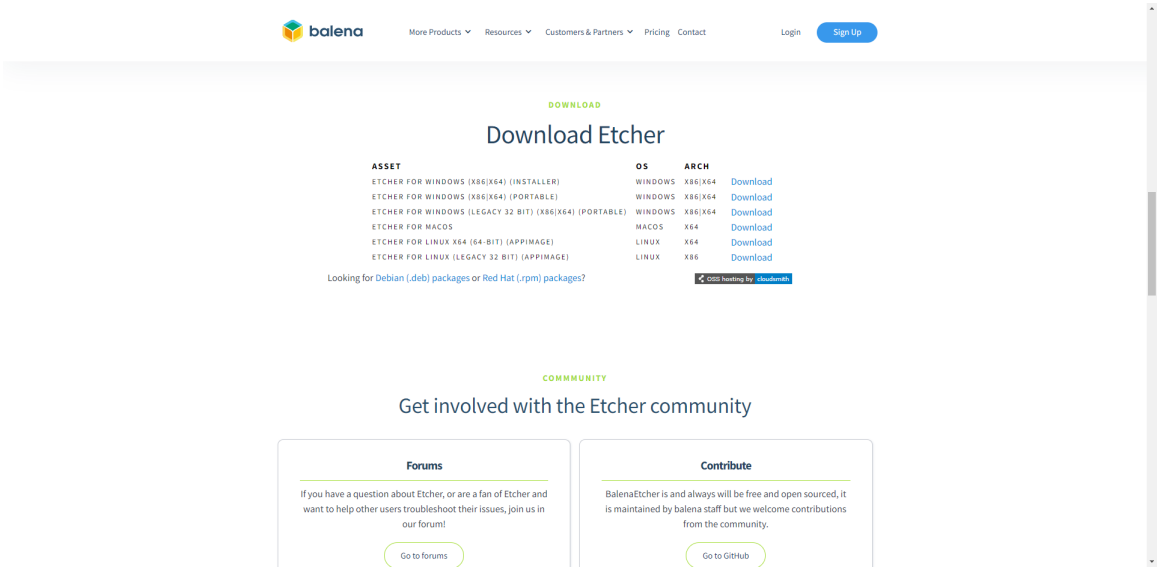

Figura A.3: Página web oficial de Etcher.

4. Cuando se haya descargado la imagen, la descomprimimos. Una vez descomprimida la imagen, introducimos la tarjeta microSD en nuestro PC y abrimos Etcher. Seleccionamos la ruta donde hemos descomprimido nuestra imagen, seleccionamos la microSD y pulsamos sobre flash. El programa nos avisará cuando haya acabado.

5. Ya tenemos listo la imagen en un nuestra microSD.

A.2. Primer arranque y configuración básica

1. Ahora que ya tenemos la imagen grabada en nuestra microSD, la introducimos en la Raspberry y la encendemos. Hay que tener en cuenta que el primer arranque puede llevar mas tiempo de lo habitual. Una vez que el sistema haya arrancado nos aparecerá el escritorio. En los iconos de la barra de tareas pulsamos sobre *Terminal* para abrir un `shell` (terminal de comandos en Linux)

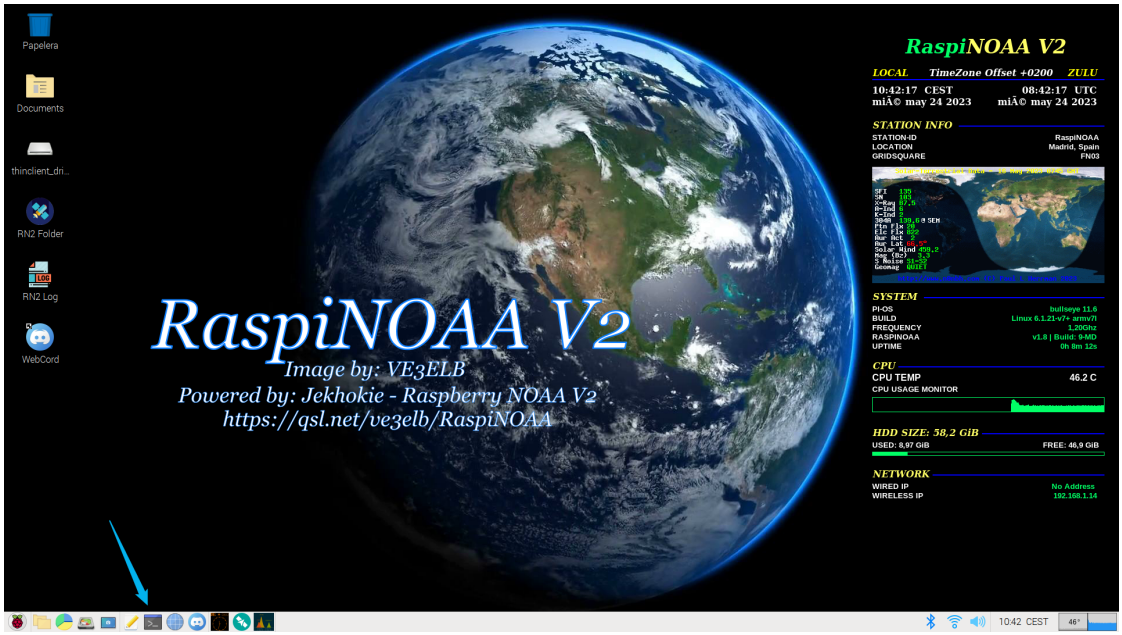

Figura A.4: Escritorio de RaspiNOAA.

2. Escribimos el comando `sudo raspi-config` y pulsamos intro. Nos aparecerá la siguiente pantalla.

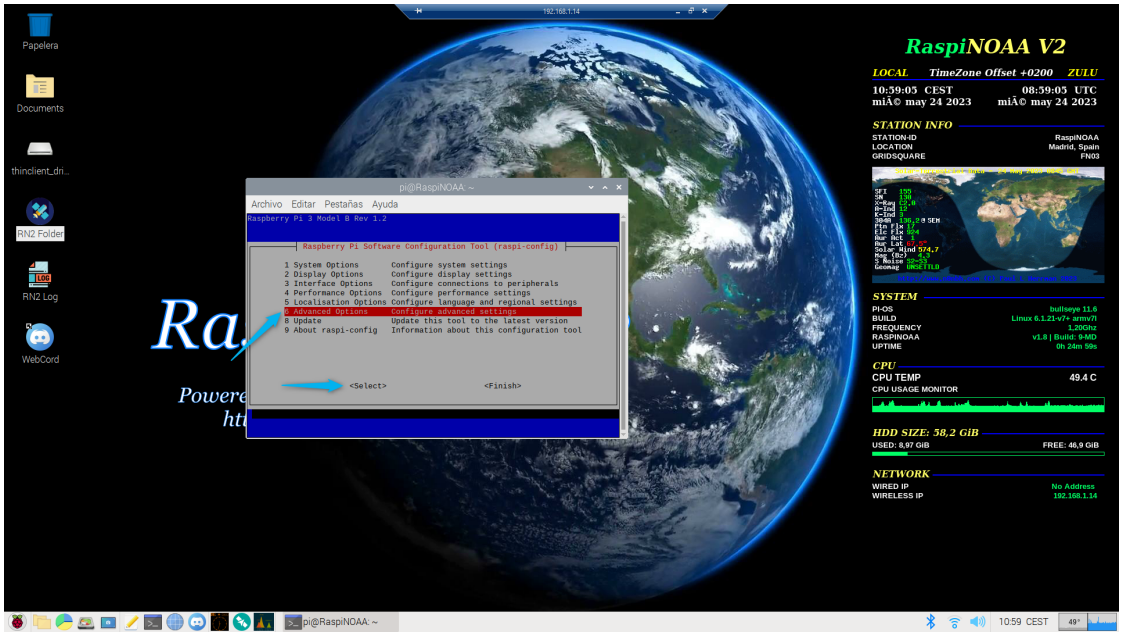

Figura A.5: Terminal con la configuración de RaspiNOAA.

3. En esta pantalla elegiremos la opción 6 *Advance Options* y pulsamos *Select* y avanzaremos a la siguiente pantalla donde seleccionaremos la opción A1 *Expand Filesystem* y otra vez *Select*.

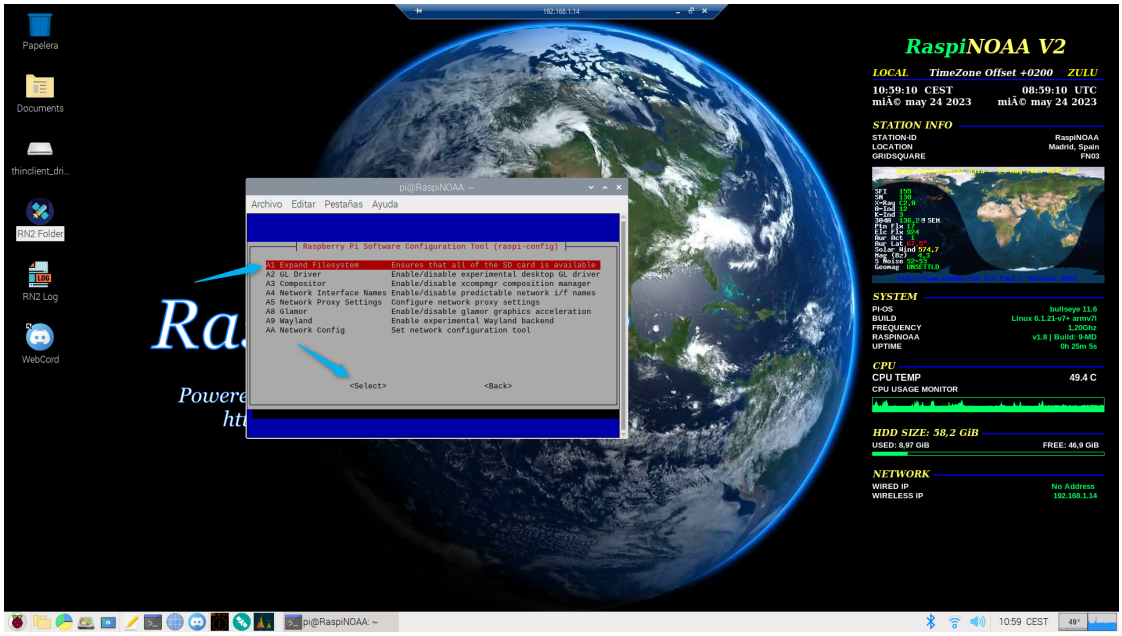

Figura A.6: Configuración *Expand Filesystem*.

4. Empezará a realizar operaciones, una vez que finalice cerramos la terminal y reiniciamos el sistema.

5. Una vez reiniciado el sistema volveremos a estar en el escritorio. Pulsamos en el botón de inicio de la barra de tareas, el qué está mas a la izquierda. Pulsamos sobre preferencias y a continuación sobre Configuración de Raspberry Pi.

Figura A.7: Captura de menú inicio de RaspiNOAA.

6. Se nos abrirá una ventana de configuración de Raspberry. Tenemos que pulsar sobre Interfaces y activar las casillas de SSH y VNC. Este paso es muy importante ya que SSH sirve para que nuestra web se pueda conectar de forma remota y VNC para conectarse de forma remota y poder controlar la Raspberry.

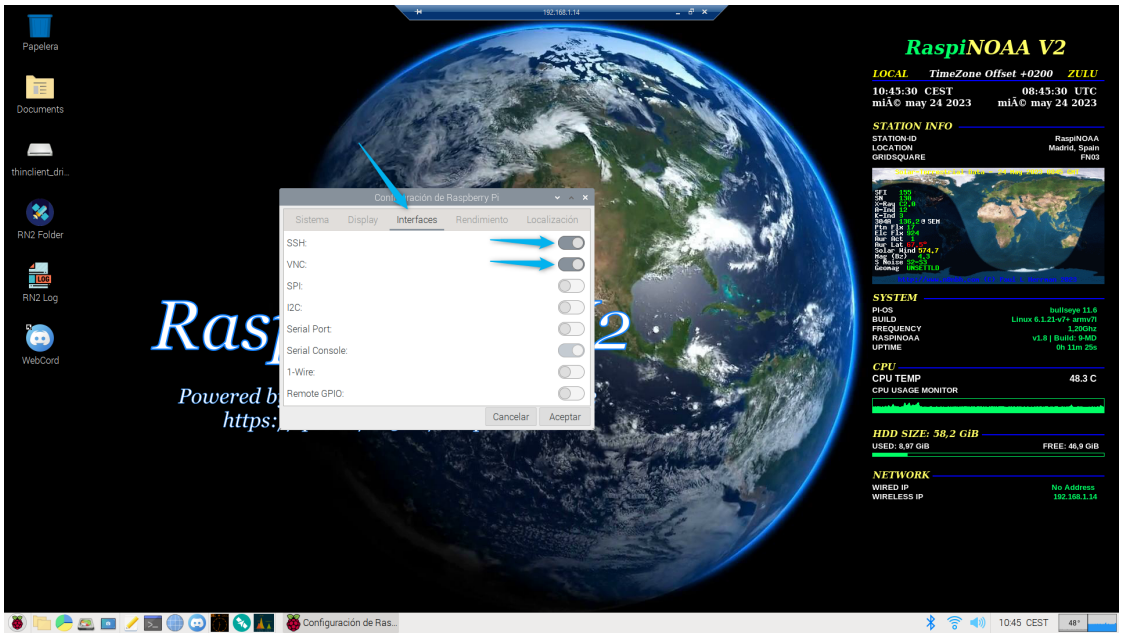

Figura A.8: Captura de configuración de interfaces.

7. Ahora nos vamos a la pestaña Localización y pulsamos sobre Configurar Local. Una vez pulsado se nos abrirá una ventana como la de la siguiente figura y debemos elegir el idioma deseado, el país donde nos encontremos y lo mas imparte poner como Conjunto de Caracteres: UTF-8 o de lo contrario fallarán los *scripts* que usaremos mas adelante. Pulsamos en aceptar.

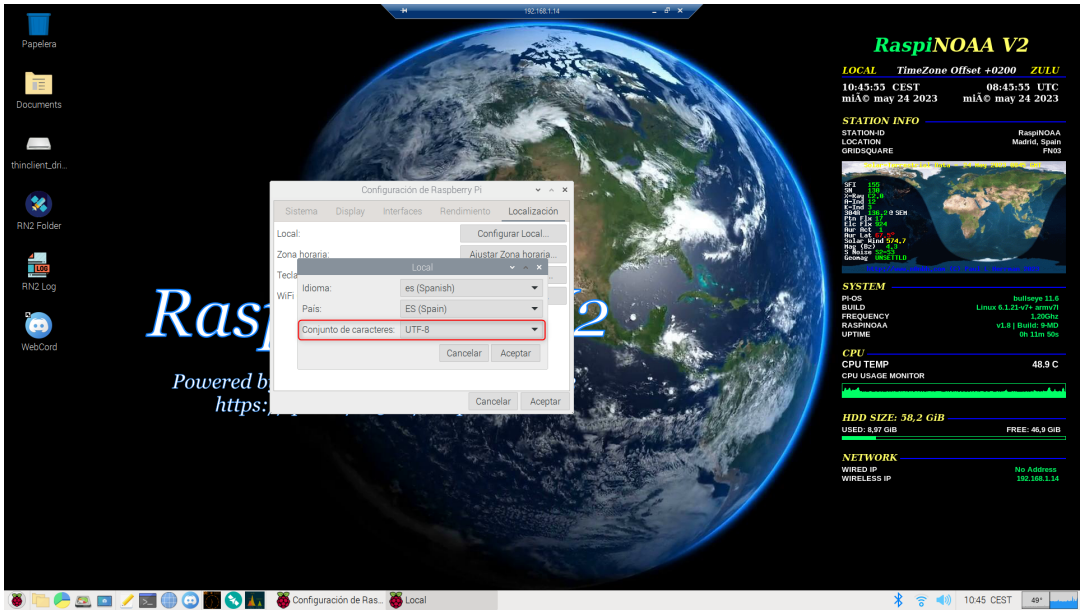

Figura A.9: Captura de configuración de localización.

8. Siguiendo en esa misma pestaña de Localización, pulsamos sobre Ajustar Zona Horaria y configuramos nuestra área y localización.

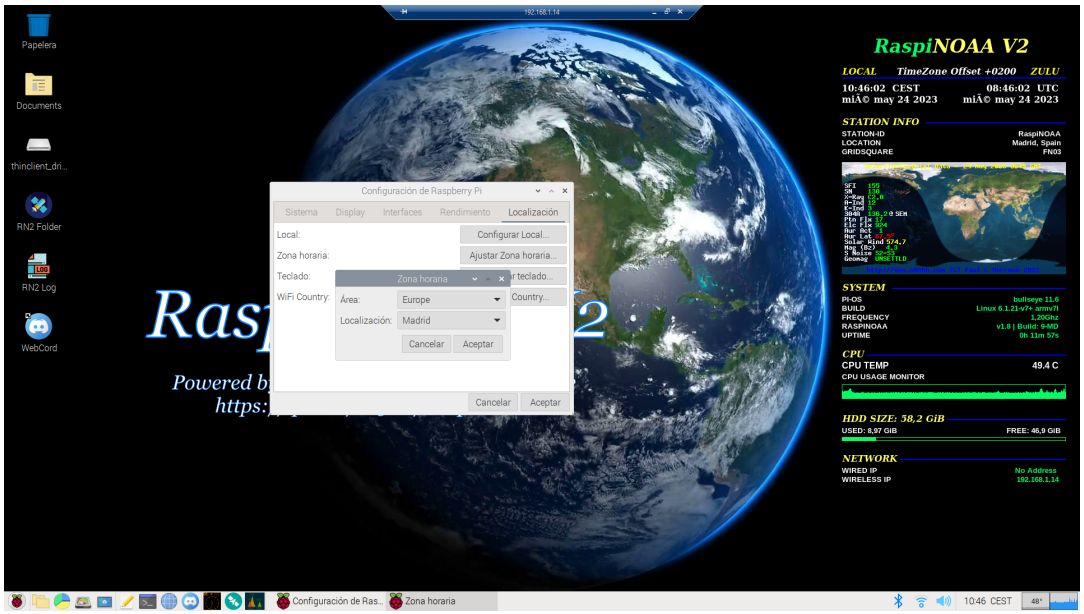

Figura A.10: Captura de configuración de zona horaria.

9. Reiniciamos el sistema.

A.3. Configuración RaspiNOAA

Con los pasos anteriores realizados vamos a proceder a configurar lo que verdaderamente nos importa: RaspiNOAA. Con los siguientes pasos vamos a darle una configuración para que nuestro sistema pueda empezar a capturar pases de forma autónoma.

1. Lo primero que haremos será acceder al sistema de ficheros. Para ello pulsamos el icono que pone RN2 Folder en el escritorio y pulsamos aceptar. En la barra de navegación de la izquierda, seleccionamos la carpeta `raspberry-noaa-v2` y luego dentro de esa carpeta abrimos `config`. Abrimos con doble click el archivo que se llama `settings.yml`.

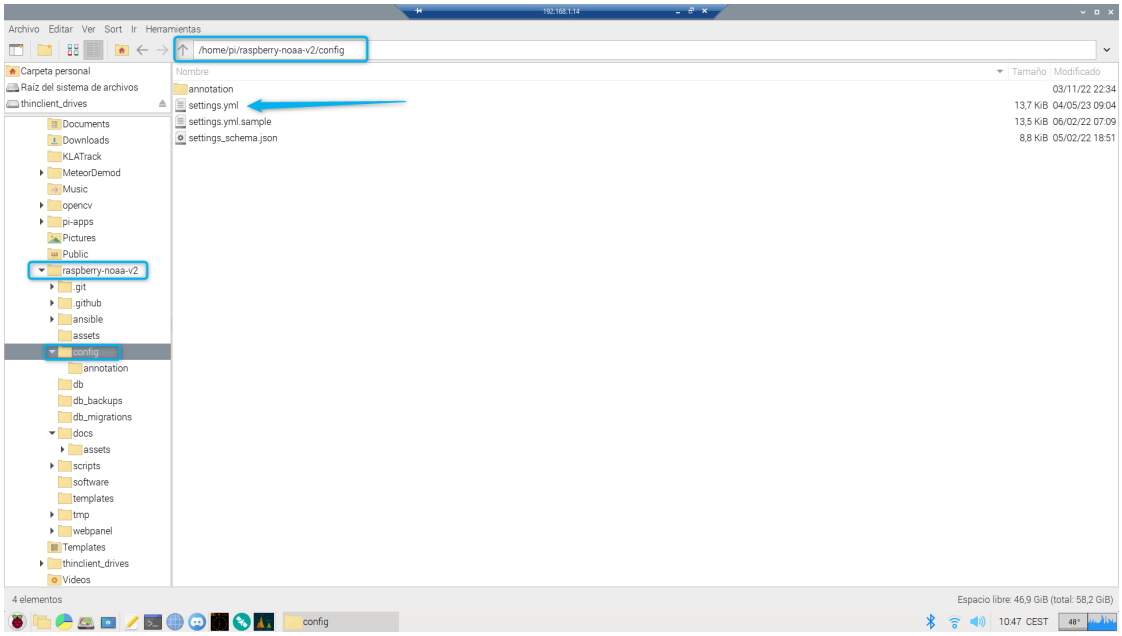

Figura A.11: Carpeta `config` de RaspiNOAA.

2. Vamos a modificar el fichero, para ello seguiremos las imágenes siguientes. Lo que no se indique en las imágenes no se modifica.

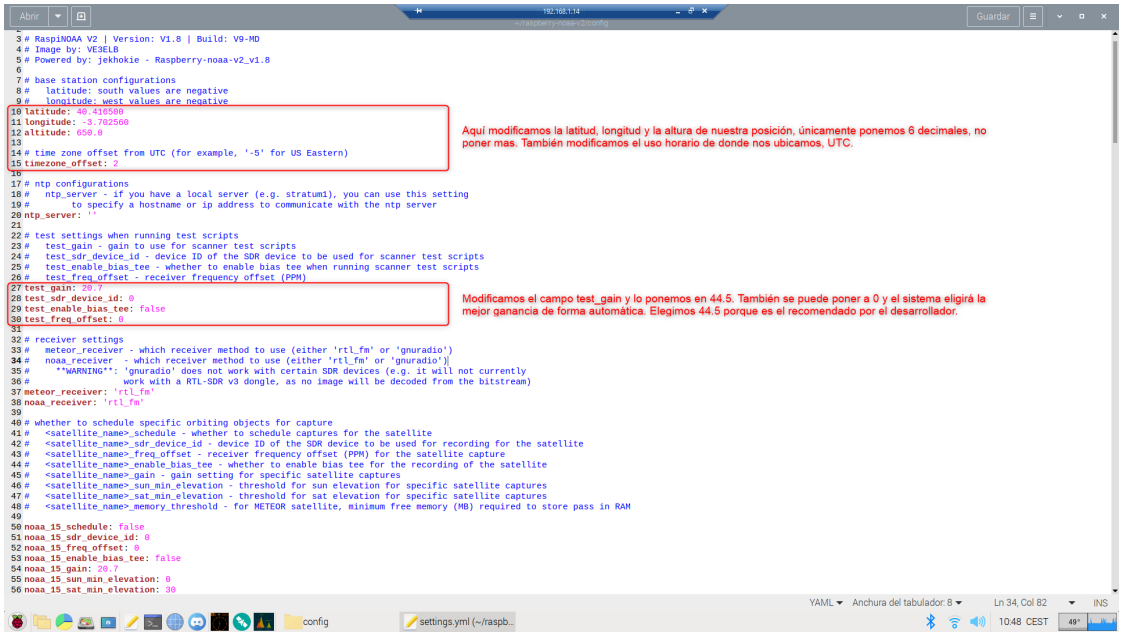

Figura A.12: Edición de configuración de localización y test de RaspiNOAA.

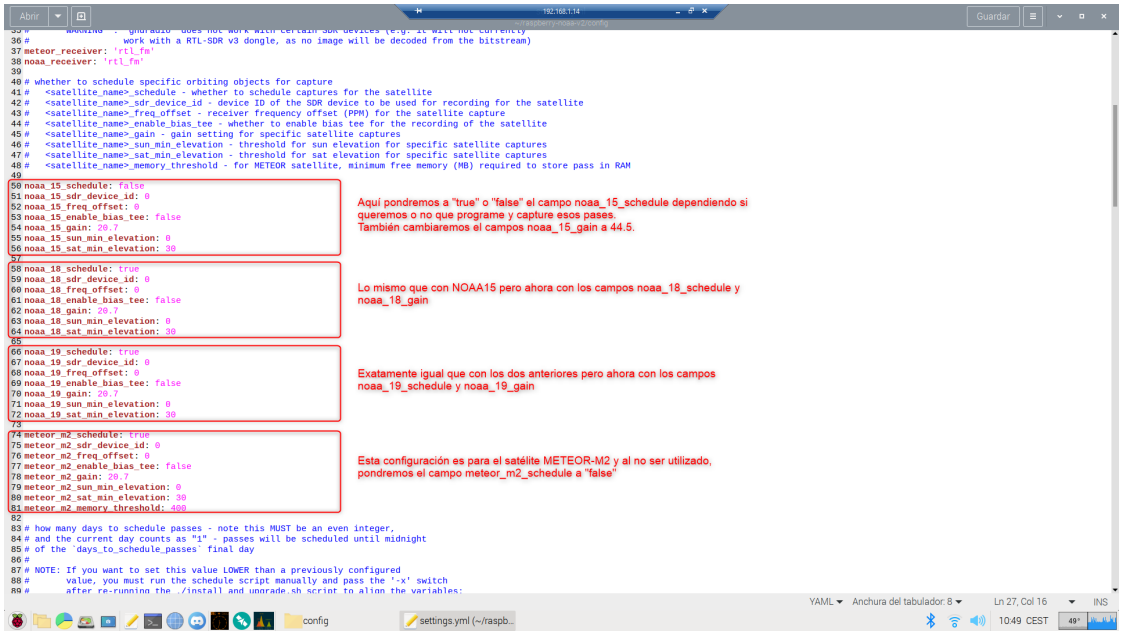

Figura A.13: Edición de los parámetros de satélites en RaspiNOAA.

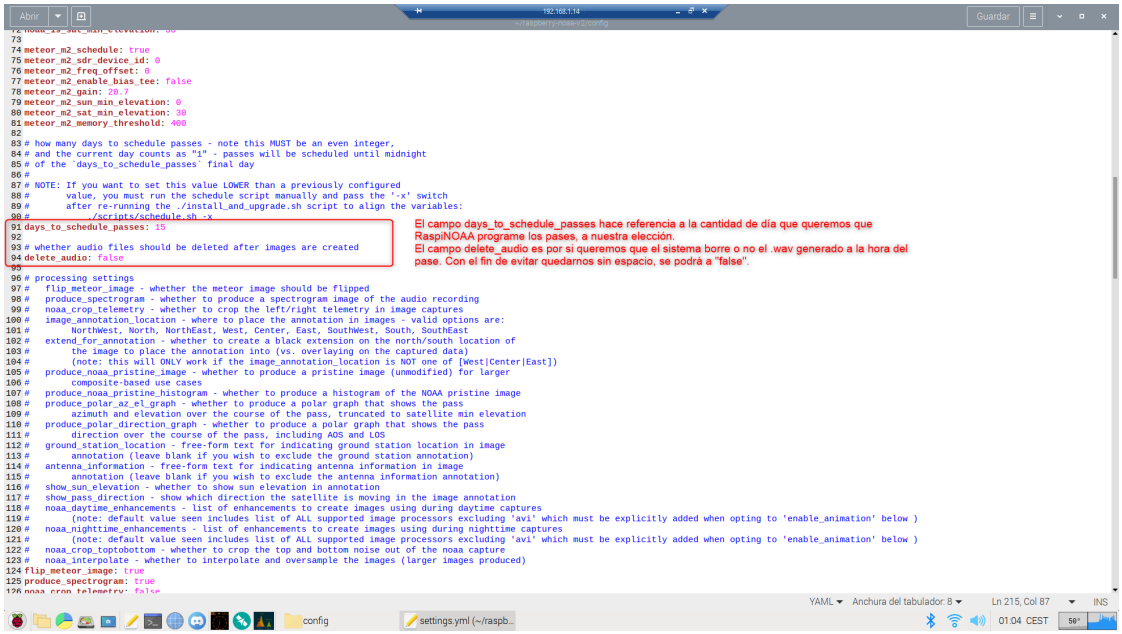

Figura A.14: Edición del guardado de pases y `.wav`.

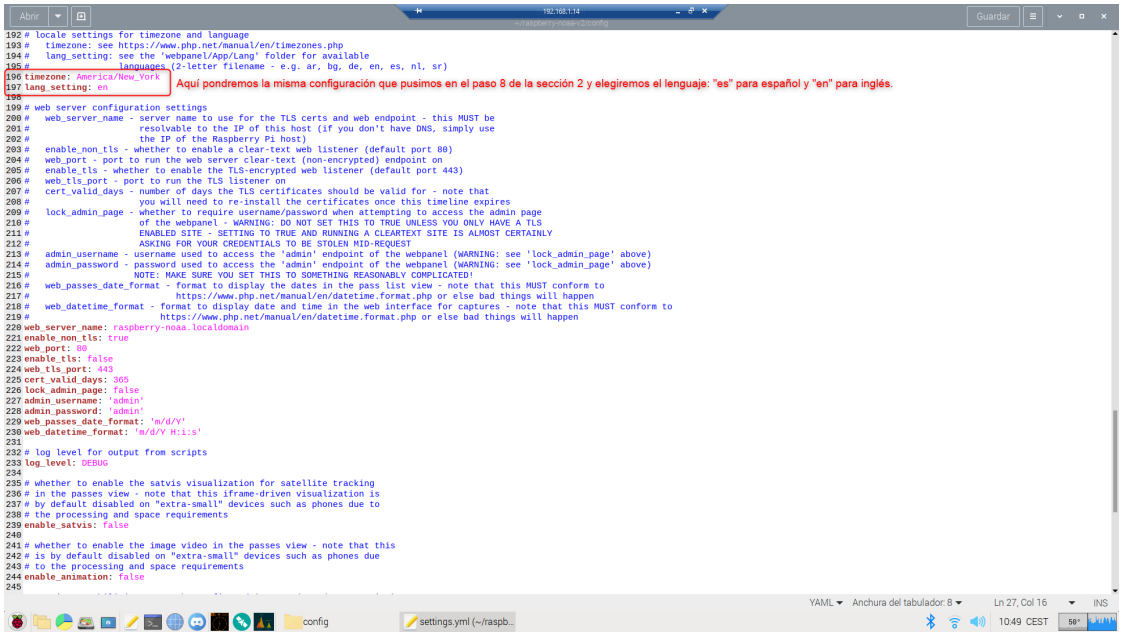

Figura A.15: Edición de los parámetros *timezone* y lenguaje.

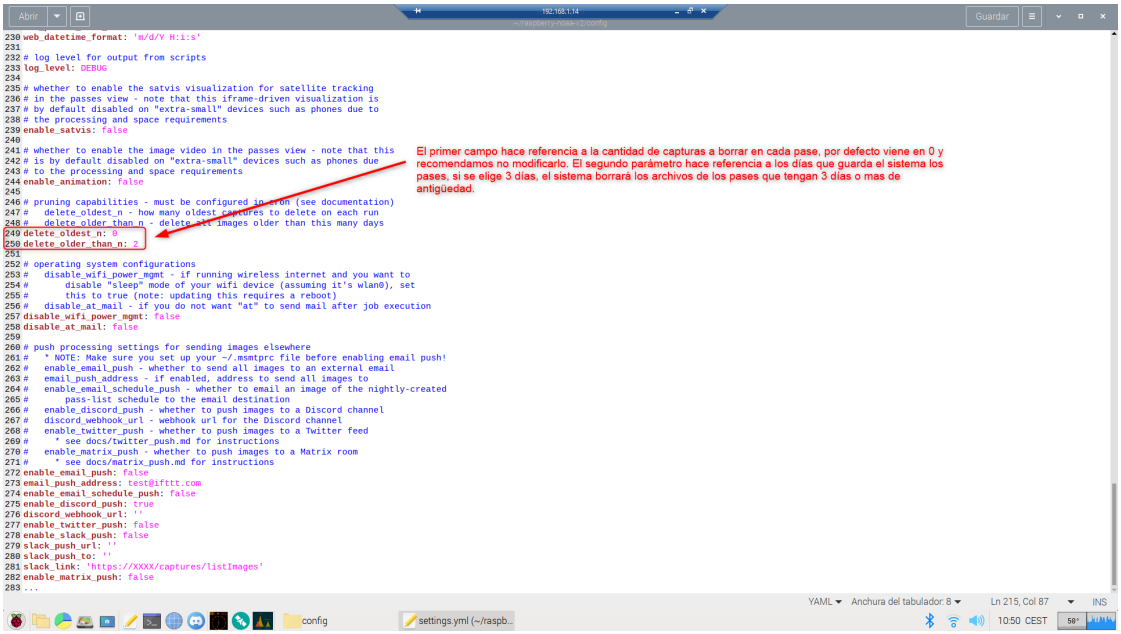

Figura A.16: Edición de borrado de pases capturados.

3. Guardamos los cambios realizados en el fichero y volvemos al escritorio.

4. Abrimos de nuevo un terminal y ejecutamos el comando `cd raspberry-noaa-v2/` y pulsamos intro, con esto indicaremos al `shell` el directorio en el que nos encontramos, y ahora seguido ejecutamos el comando `./install_and_upgrade.sh` y pulsamos intro. Con este último paso lo que estamos haciendo es aplicar los cambios realizados en el punto 2 a nuestro sistema y que configure todo. Este paso puede demorarse entre 15-20 minutos. Una vez que acabe cerramos la terminal y ya está nuestro sistema completamente configurado y listo para capturar pases.

Extra: En el paso 2 hemos configurado el número de días que queremos programar, pasado este tiempo es necesario volver a descargar los *TLE* con la información de los pases. Desde el escritorio abrimos una terminal y escribimos el comando `cd raspberry-noaa-v2/`, pulsamos intro y a continuación el comando `cd scripts` e intro. Por último escribimos el comando `./schedule.sh -t -x`. Este *script* lo que realiza es un borrado de los *TLE* antiguos y descarga los nuevos para los próximos días.

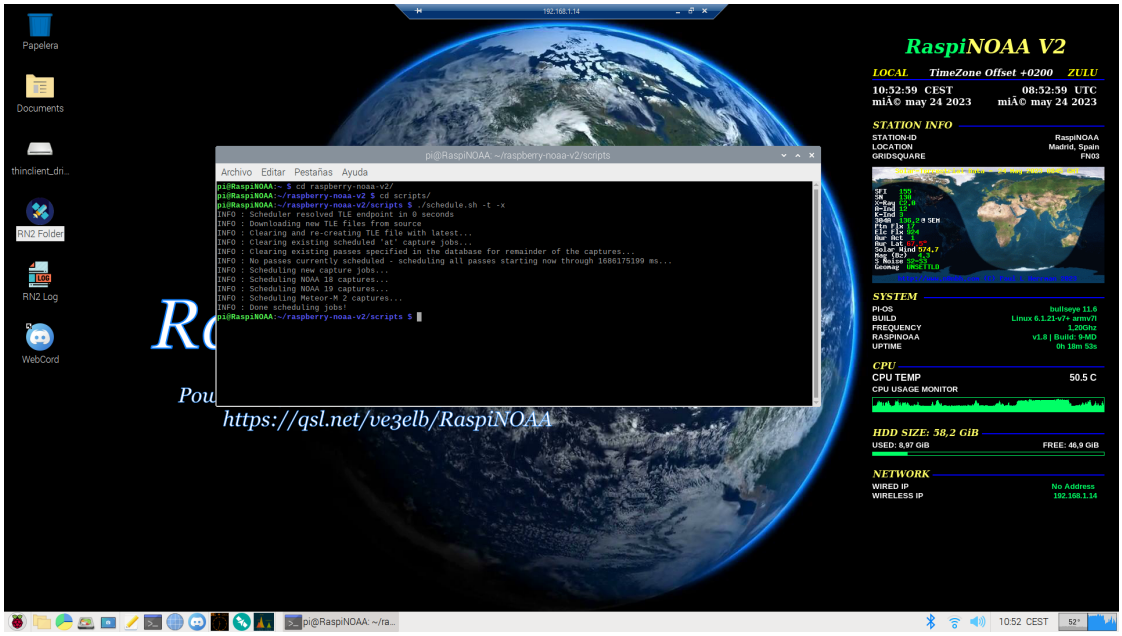

Figura A.17: Captura de *shell* para reprogramado de pases.

A.4. OPCIONAL: Configuración Conky

Este paso es opcional y sirve para configurar el *widget* Conky que tenemos en el escritorio.

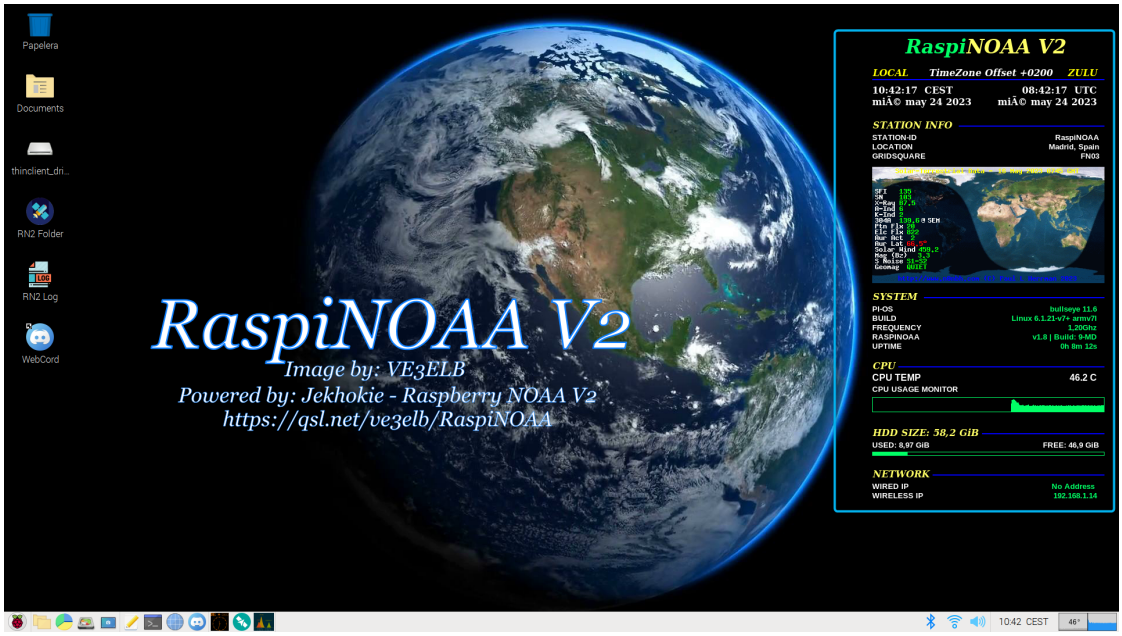

Figura A.18: Escritorio NOAA con Conky.

1. Abrimos el explorador de archivos pulsando sobre de `RN2 Folder` y nos dirigimos a la ruta `/home/pi/Conky`.

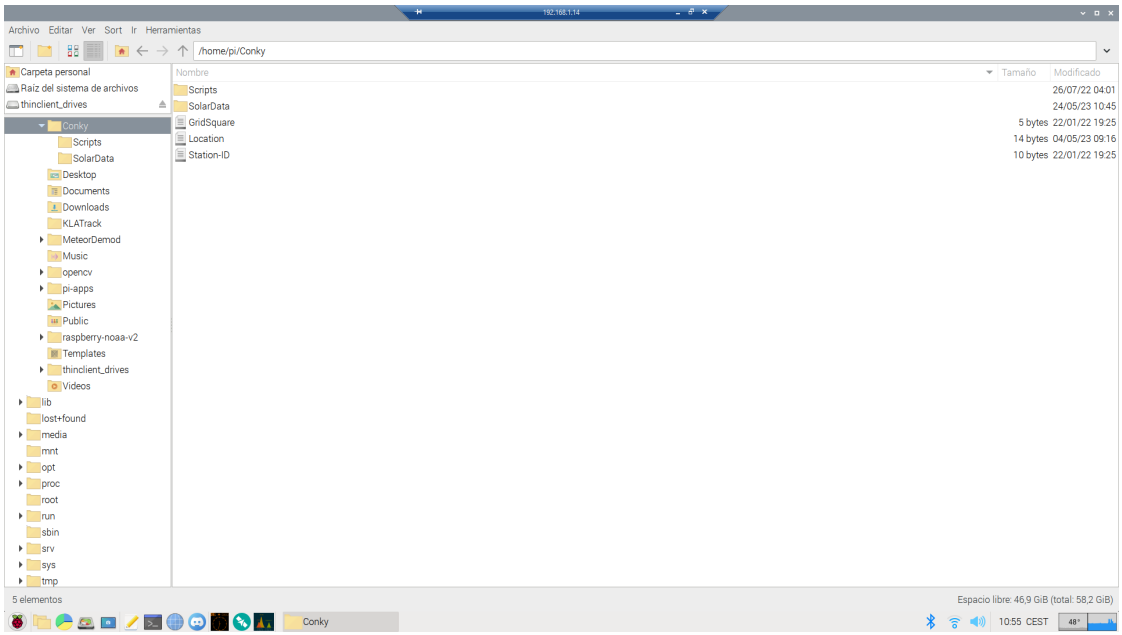

Figura A.19: Carpeta de configuración de Conky.

2. Abriremos el que se llama `Location` en el editor de textos y escribiremos nuestra localización para que se muestre en el *widget*.

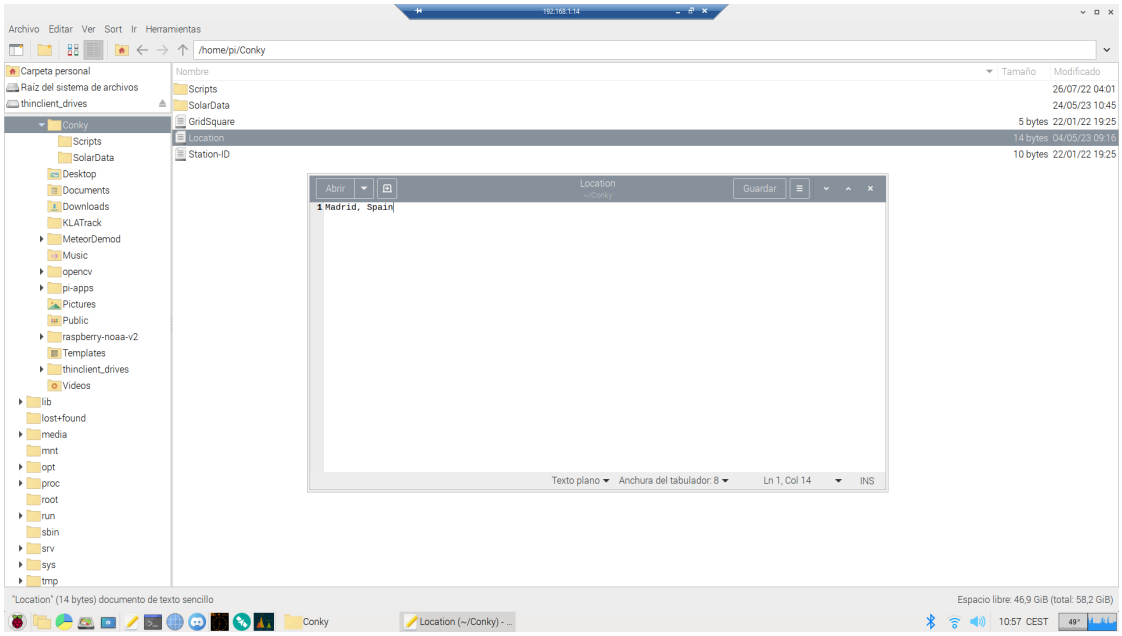

Figura A.20: Edición del fichero `Location` de Conky.

3. Ahora abrimos el que se llama `Station-ID` con el que le pondremos un nombre a nuestra estación.

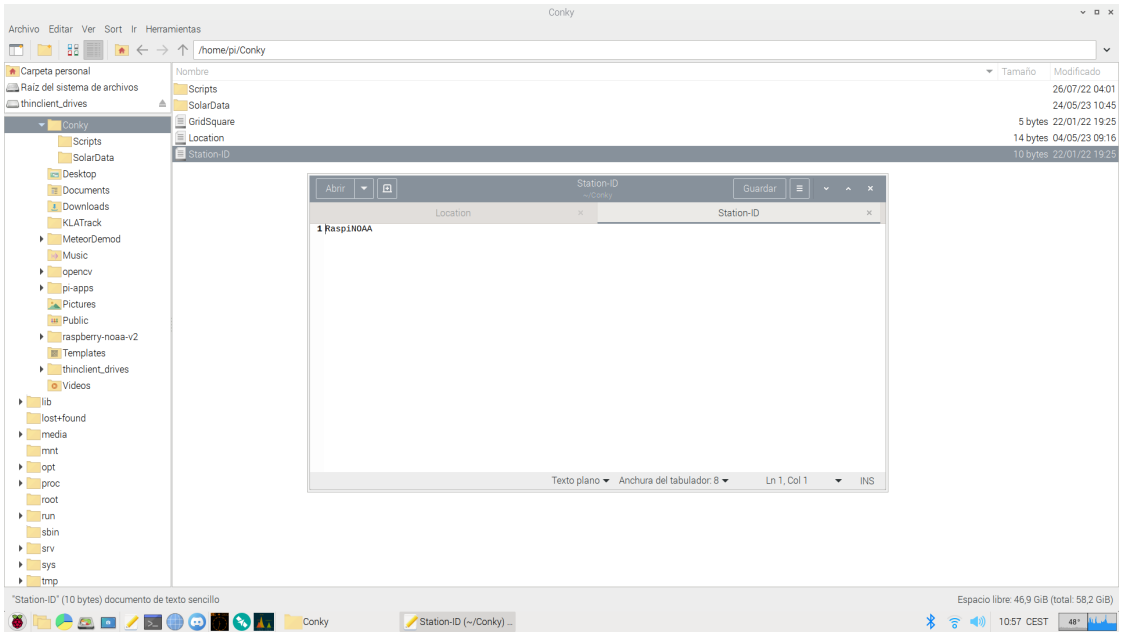

Figura A.21: Edición del fichero `Station-ID` de Conky.

4. Pasado unos minutos el *widget* se actualizará con la nueva información y ya tendremos actualizado Conky.

Apéndice B

Manual de uso: Instalación WEB

Esta es una sencilla guía de como instalar al web en nuestro equipo. Para seguir este manual es necesario tener un equipo qué tenga instalada cualquier distribución de Linux y además tener permisos de administrador.

OPCION Antes de comenzar es recomendodo abrir un terminal y ejecutar el comando `sudo apt-get update` con el fin de evitar errores mas adelante.

1. Instalamos **Apache2** para ello abrimos un terminal y ejecutamos el comando `sudo apt install -y apache2`.

2. Ahora instalamos **PHP** en su ultima versión. Abrimos un terminal y ejecutamos el comando `sudo apt-get install php8.1` (nos instalará la versión 8.1).

3. Para poder utilizar algunas funciones es necesario instalar una librería de apoyo para PHP llamada **libssh2-1** y como anteriormente hemos hecho, abrimos un terminal y ejecutamos el comando `sudo apt-get install libssh2-1`

4. Con todos estos pasos realizados solo nos queda copiar nuestra web en el directorio principal de Apache. En nuestro caso será `/var/www` aunque siempre podremos consultarlo en el archivo de configuración: `/etc/apache2/sites-enabled/000-default`.

5. Agregar la configuración de nuestro RaspiNOAA a la web. Para ello abrimos el archivo `config.ini` en la raíz de nuestra web. Lo abrimos con un editor de texto cualquiera y pondremos la IP, el puerto (por defecto siempre es 22), el usuario y la contraseña de nuestro RaspiNOAA, guardamos.

6. Comprobar que todo funciona abriendo un navegador y accediendo a la pagina **localhost** o a través de la ip **127.0.0.1**.