Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Puesta en marcha de un sistema de posicionamiento mecánico de antenas

Grado en Ingeniería Mecánica

ALUMNO: Jesús Bermúdez Lara

DIRECTORES: José María Núñez Ortuño

CURSO ACADÉMICO: 2023-2024

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Puesta en marcha de un sistema de posicionamiento mecánico de antenas

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universidade Vigo

RESUMEN

El presente proyecto consiste en el desarrollo de una aplicación web diseñada para el control de una antena mediante una unidad de control y el software *Rotctld*. La esencia del proyecto estaba en integrar las funcionalidades de *Rotctld* en una App Web, permitiendo a usuarios controlar una antena a través de una interfaz gráfica. Este esfuerzo se ha estructurado en varias fases.

Inicialmente, se llevó a cabo una evaluación de las capacidades de *Rotctld* para identificar las funcionalidades esenciales para la aplicación web y asegurar que esta pudiera aprovechar el potencial de la unidad de control. Posteriormente, el foco del proyecto fue desarrollar el *back-end*, empleando Python para crear un programa que actuara como intermediario entre *Rotctld* y la aplicación web.

La fachada de la aplicación web se elaboró utilizando HTML y JavaScript, enfatizando la facilidad de uso en el control de la antena. Flask y Flask-SocketIO fueron instrumentales para lograr enlaces de comunicación en tiempo real entre el *front-end* y el *back-end*, asegurando interacciones de usuario suaves y "responsive".

El proyecto finalizó con una fase de pruebas que validaron el correcto funcionamiento del programa desarrollado para controlar la antena asegurando una buena integración entre el software y el hardware.

PALABRAS CLAVE

Rotctld, Flask, Control de Antenas, rotor, *front-end*, *back-end*, web.

AGRADECIMIENTOS

Quisiera expresar en este apartado mi más sincera gratitud a mis padres, cuyo apoyo incondicional y ánimo han sido mi luz guía a lo largo de este viaje. Su fe en mis capacidades me ha motivado para seguir siempre mis ambiciones sin descanso. A mis amigos, gracias por ser mis pilares de fortaleza, por las risas que han hecho mi estancia en la Escuela Naval Militar una experiencia verdaderamente memorable. El Centro Universitario de la Defensa ha sido el terreno fértil para mi crecimiento, proporcionando recursos, oportunidades y un ambiente que fomenta el aprendizaje. Estaré inmensamente agradecido por este refugio académico tan enriquecedor. Por último, mi más profundo agradecimiento a mi tutor, cuya orientación y experiencia fueron fundamentales para dar forma a este proyecto. Sus percepciones y comentarios constructivos fueron invaluables, y estoy profundamente agradecido por la sabiduría que me han impartido. Este logro no es únicamente mío, sino un testimonio del apoyo colectivo y la orientación que he recibido de cada una de las partes. Gracias por ser parte de este increíble viaje.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN Y OBJETIVOS

En este capítulo se exponen los objetivos planteados para la realización de este proyecto, posteriormente se ofrece una visión general de los distintos bloques que lo componen, definiendo los conceptos más relevantes para la comprensión de la presente memoria y relatando a su vez una breve contextualización y evolución a lo largo de la historia.

1.1 Contexto

Los sistemas de rotores han sido fundamentales para el avance de la tecnología de antenas, han evolucionado significativamente desde su inicio, en paralelo con la evolución de las antenas mismas. El concepto de la antena se remonta a la década de 1830 con los experimentos de Michael Faraday que demostraban el acoplamiento de la electricidad y el magnetismo, sentando las bases para entender la recepción y transmisión de ondas electromagnéticas. Este trabajo temprano presagiaba el desarrollo de la antena de bucle, marcando el amanecer del diseño práctico de antenas. A finales del siglo XIX y principios del XX, pioneros como Heinrich Hertz y Guglielmo Marconi avanzaron aún más en el campo, siendo la comunicación inalámbrica transatlántica de Marconi en 1901 un momento seminal para la tecnología de antenas. [1]

A medida que las antenas evolucionaban, también lo hacía la necesidad de un control preciso sobre su orientación, lo que llevó al desarrollo de sistemas de rotores. Estos sistemas, que cambian la dirección de una antena direccional dentro del plano horizontal, se hicieron populares en varias aplicaciones, incluyendo la radioafición y las comunicaciones militares. Demostraron ser soluciones rentables para optimizar la recepción de señales desde múltiples direcciones sin la necesidad de usar varias antenas. A mediados del siglo XX, empresas como Alliance Manufacturing Co. y Astatic Corporation se convirtieron en fabricantes importantes de sistemas de rotores para radio y televisión, destacando la creciente importancia comercial y doméstica de estos dispositivos. [2]

Hoy en día, los sistemas de rotores continúan evolucionando, con avances en tecnología que ofrecen un control más sofisticado e integración con sistemas de seguimiento digitales, facilitando más que nunca la alineación de antenas con satélites o fuentes de transmisión.

Otro elemento clave que compone este proyecto, son las Unidades de Control de Antenas (ACUs por sus siglas en inglés). Estos dispositivos son fundamentales en el ámbito de las telecomunicaciones y los sistemas de comunicación por satélite, permiten el posicionamiento y control preciso de las antenas. Estas unidades son cruciales para optimizar la transmisión y recepción de señales de radio, asegurando que las antenas puedan rastrear y comunicarse con precisión con satélites u otras fuentes de comunicación remotas. El papel de una ACU es gestionar los ángulos de azimut (horizontal) y elevación (vertical) de la antena, manteniendo la mejor alineación posible con una fuente de señal objetivo, como un satélite de radiodifusión o un transmisor de estación terrestre.

La historia de las Unidades de Control de Antenas se remonta a los primeros días de los sistemas de comunicación por satélite y radar. Inicialmente, el control de las antenas era principalmente manual o involucraba mecanismos rudimentarios, con operadores haciendo ajustes basados en la fuerza de la señal observada u horarios predefinidos. Sin embargo, a medida que la industria satelital creció, especialmente con el advenimiento de la era espacial en la década de 1950 y el lanzamiento subsiguiente del Sputnik, el primer satélite artificial de la Tierra, en 1957, se hizo evidente la necesidad de sistemas de control más sofisticados y automatizados. Esto llevó al desarrollo de las primeras ACUs automatizadas, que podían manejar cálculos complejos y ajustes en tiempo real, mejorando significativamente la fiabilidad de la comunicación y la calidad de la señal.[3]

La evolución de las ACUs ha estado estrechamente vinculada a los avances en tecnología informática, procesamiento digital de señales y ciencia de materiales. Las ACUs modernas son dispositivos altamente sofisticados, capaces de gestionar el posicionamiento de antenas con extrema precisión. A menudo incorporan características como GPS para la precisión de ubicación, procesadores de señal digital para algoritmos de control en tiempo real y conectividad de red para monitoreo y control remotos. La transición de sistemas de control mecánicos a digitales ha permitido avances significativos en las capacidades de comunicación, apoyando una amplia gama de aplicaciones desde la radiodifusión televisiva hasta la comunicación en el espacio profundo.

Por otro lado, un concepto crítico en este proyecto es el software de control de antenas. El software de control de antenas juega un papel fundamental en la industria de las telecomunicaciones, particularmente en las comunicaciones por satélite, al proporcionar la interfaz y los algoritmos necesarios para el control y manejo precisos de los sistemas de antenas. Este tipo de software es responsable de dirigir el posicionamiento de las antenas para asegurar una transmisión y recepción óptimas de la señal. Abarca una amplia gama de funcionalidades, incluyendo el cálculo de órbitas de satélites, la precisión en el apuntado de antenas, el control de seguimiento y la ejecución de horarios operativos predefinidos.

Los orígenes del software de control de antenas coinciden con la aparición de la tecnología informática a mediados del siglo XX. Las primeras versiones de este software surgieron como solución a la creciente complejidad de gestionar las comunicaciones por satélite, las cuales requerían un control preciso sobre el posicionamiento de las antenas para mantener conexiones fiables con satélites en rápido movimiento. Inicialmente, estas soluciones de software eran rudimentarias, desarrolladas para sistemas específicos y a menudo carecían de la versatilidad y eficiencia de las aplicaciones modernas. [3]

El lanzamiento del Sputnik en 1957 por la Unión Soviética marcó un momento crucial en la historia de la exploración espacial y necesitó el rápido avance de tecnologías capaces de rastrear y comunicarse con satélites. Esta era vio el desarrollo de los primeros sistemas de control de antenas informatizados, que sentaron las bases para el software de control de antenas contemporáneo. Estos primeros sistemas eran típicamente a medida, diseñados para configuraciones únicas de antenas y operaban en computadoras centrales que eran tanto costosas como físicamente imponentes.

Con el avance de la tecnología informática, particularmente con la llegada de los microprocesadores en la década de 1970 y el posterior desarrollo de computadoras personales, el software de control de antenas evolucionó significativamente. Estos avances permitieron algoritmos más sofisticados, interfaces de usuario amigables y la integración de datos de seguimiento en tiempo real, mejorando significativamente la eficiencia y precisión de las operaciones de control de antenas.

El software de control de antenas moderno se caracteriza por su alto grado de automatización, precisión y flexibilidad, soportando una amplia gama de tipos de antenas y protocolos de comunicación. Es capaz de integrarse con sistemas de posicionamiento global (GPS) para mejorar la precisión de ubicación, utilizar el procesamiento de señales digitales para ajustes en tiempo real y ofrecer capacidades de monitoreo y control remotos.[4]

1.2 Motivación

El presente trabajo se centra en el desarrollo de software de control de antenas con una interfaz visual que mejore significativamente la facilidad y eficiencia en la gestión y operación de sistemas de antenas. El trabajo debe combinar algoritmos de software con interfaces gráficas amigables, ofreciendo una solución tanto para usuarios novatos como para expertos. La razón detrás de este proyecto comprende varios aspectos clave en la operación y gestión de antenas:

Una interfaz visual reduce las complejidades del control de antenas, presentando al usuario controles intuitivos como deslizadores, diales y pantallas gráficas lo cual reduce la curva de aprendizaje y permite a un operador gestionar las posiciones y configuraciones de las antenas de manera más natural, haciendo ajustes basados en señales visuales en lugar de complejas entradas numéricas o instrucciones de línea de comandos.

Un software de control visual permite proporcionar retroalimentación en tiempo real sobre el estado de la antena, incluyendo su posición actual, la fuerza de la señal y los parámetros operativos. Esta representación visual inmediata ayuda a los usuarios a tomar decisiones informadas rápidamente, optimizando la alineación y el rendimiento de la antena sin necesidad de extensas pruebas y errores.

Con el control por software, es posible automatizar tareas rutinarias, como mover la antena a posiciones predefinidas en momentos específicos. Esta capacidad es especialmente valiosa en aplicaciones como el seguimiento de satélites o la radioastronomía, donde las antenas siguen trayectorias preestablecidas a través del cielo. Las interfaces visuales facilitan la configuración de estas rutinas de automatización.

Una ventaja significativa del control basado en software con una interfaz visual es la capacidad de gestionar antenas de forma remota. Esto resulta particularmente beneficioso para instalaciones de difícil acceso físico, como antenas en azoteas o ubicadas en entornos hostiles. El control centralizado de múltiples antenas desde una única interfaz también se simplifica, permitiendo una gestión eficiente de arreglos complejos de antenas.

Una solución software permite la personalización para satisfacer las necesidades evolutivas del usuario. Conforme aparecen nuevas características o algoritmos, se pueden incorporar en la interfaz visual, proporcionando mejoras continuas y adaptaciones a los requisitos cambiantes.

Teniendo en cuenta lo expuesto, se pretende desarrollar una herramienta que mejore significativamente la accesibilidad a la operación de la antena del laboratorio de investigación del CUD.

1.3 Objetivos

Este proyecto se embarca en la tarea de diseñar y desarrollar un software que gestione el funcionamiento de un sistema mecánico de posicionamiento de antenas. El enfoque principal de este sistema se centrará en controlar dos aspectos fundamentales: la dirección horizontal (azimut) y la dirección vertical (elevación) de las antenas. A través de este proyecto, se pretende alcanzar los siguientes objetivos principales:

Crear un software avanzado capaz de interactuar de manera eficiente con los rotores que controlan los movimientos de azimut y elevación de la antena. Este software deberá facilitar la emisión de órdenes precisas para ajustar la posición de la antena mejorando su funcionamiento general.

Implementar una interfaz de usuario sencilla y accesible, desarrollada como una página web. Este enfoque permitirá a cualquier usuario interactuar con el sistema de manera visual, utilizando botones e indicadores para el control de la antena, mejorando significativamente la usabilidad y accesibilidad del sistema en comparación con métodos basados en la línea de comandos.

Se deberá realizar una calibración y pruebas de direccionamiento para asegurar la precisión en los movimientos de azimut y elevación. Este paso será crucial para garantizar el correcto funcionamiento de la antena.

La aplicación web se desarrollará en lenguajes accesibles como HTML y Python, buscando no solo facilitar la implementación inicial y la calibración del sistema sino también proporcionar una plataforma sólida para futuras mejoras y desarrollos.

Al alcanzar estos objetivos, se aspira a ofrecer una solución integral que mejore el control y gestión actual de la antena desarrollando un sistema más accesible y fácil de usar para una amplia gama de usuarios, independientemente de su experiencia técnica.

2 ESTADO DEL ARTE

En este apartado se pretende desarrollar de manera general la situación actual de los diversos bloques que componen el proyecto. Inicialmente se expondrán las definiciones de los conceptos más relevantes del proyecto, a continuación, se relatará un contexto histórico para conocer los principales avances en los programas para diseño de software y los software de control de rotores actuales, posteriormente se hablará de los programas de desarrollo para aplicaciones web, los *framework* actuales más populares y las distintas funcionalidades que ofrece cada uno.

2.1 Sistemas de Posicionamiento Mecánico de Antenas

2.1.1 Contextualización

La comunicación por satélite es uno de los campos de la ciencia y tecnología con más rápido crecimiento. Resulta la opción más útil para la transmisión de información a nivel global. La antena juega un papel importante en la transmisión y recepción de señales. La calidad de la señal recibida (amplitud o intensidad) depende de factores como la ubicación relativa del satélite, la posición de la antena y los parámetros de la antena.

Las antenas de la Red de Espacio Profundo (Deep Space Network) de la NASA se comunican con las naves espaciales enviando comandos (enlace ascendente) y recibiendo información de las naves espaciales (enlace descendente). Para asegurar el seguimiento continuo durante la rotación de la Tierra, las antenas se ubican en tres sitios: Goldstone (California), Madrid (España) y Canberra (Australia). Las frecuencias de señal son de entre 8.5 GHz (banda X) y 32 GHz (banda Ka). El tamaño de las antenas varía entre 34 m o 70 m. Toda la estructura de la antena rota sobre una vía circular (pista de acimut) con respecto al eje vertical (o de acimut). Para la frecuencia de la banda Ka, la precisión de seguimiento requerida es del orden de 1 mdeg. Este requisito demuestra el papel tan crítico que tienen los sistemas de control y posicionamiento de antenas.

Para su uso en misiones de órbita elíptica en el espacio profundo y futuras misiones a Marte, la Agencia Espacial Europea (ESA) ha construido estaciones terrestres de espacio profundo de 35 m. Las antenas están diseñadas para frecuencias de hasta 35 GHz y una precisión de apuntamiento de 6 mdeg. La primera antena (que se mueve en los ejes de azimut y elevación) ha sido instalada en Australia y ha demostrado su cumplimiento con las especificaciones. La segunda antena está en construcción en España. La antena de diámetro 35 m, incorpora un pedestal de movimiento completo con un sistema de guía de onda de haz.

En Effelsberg se encuentra uno de los telescopios completamente orientables más grandes del mundo, 100 metros en diámetro. Es operado por el Instituto Max Planck de Radioastronomía en Bonn, Alemania, en longitudes de onda que van desde aproximadamente 3.5 mm hasta 90 cm. El telescopio se

utiliza para observar púlsares, cúmulos de gas y polvo fríos, los lugares de formación de estrellas, chorros de materia emitidos por agujeros negros y los núcleos de galaxias lejanas. Su construcción de acero pesa 3,200 toneladas. Su velocidad de acimut es de 0.5 grados/s, y de 0.25 grados/s en elevación. [5]

2.1.2 Definición

Las estructuras expuestas en el apartado anterior son todas manipuladas con sistemas de control de antenas. Un Sistema de Control de Antena (ACS) es un sistema que se utiliza para controlar la posición y configuración de una antena de estación terrestre con el fin de optimizar su rendimiento para una aplicación específica. Las antenas se utilizan para transmitir o recibir señales electromagnéticas, y la posición de la antena puede tener un efecto significativo en su rendimiento. La Unidad de Recepción de Seguimiento (TRU) es la parte del sistema encargada de recibir señales de un satélite y proporcionar entradas de seguimiento al Sistema de Control de Antena.

2.1.3 Ventajas de los sistemas de posicionamiento

Los sistemas de posicionamiento y control nos permiten operar equipos de enormes dimensiones con una precisión que de otro modo sería imposible como, por ejemplo, orientar antenas para captar señales de radio débiles; controlar estas antenas de manera manual sería imposible. El ser humano por sí solo no podría proporcionar la potencia requerida para la carga o la velocidad; los motores proporcionan la potencia, y los sistemas de control regulan la posición y la velocidad.

Los sistemas de control se construyen principalmente por 4 motivos:

- Amplificación de la potencia
- Control remoto
- Conveniencia en el formato de entrada
- Compensación por perturbaciones

Por ejemplo, una antena radar que se posiciona mediante los comandos de un interfaz, requiere una energía mínima de entrada ya que se trata solamente de presionar un botón, pero es necesario una energía de salida enorme para poder mover la antena. Un sistema de control puede producir la amplificación de potencia necesaria.

En otros casos, los sistemas de control y posicionamiento nos permiten operar estas estructuras cuando están en localizaciones inaccesibles o peligrosas como es el caso de una antena que brindará servicios de seguimiento, telemetría y telecomandos (TT&C - Telemetry, Tracking and Command) y de descarga de datos de ciencia de misiones satelitales cuya instalación se realizará en la Antártida. [6]

Otra ventaja de un sistema de control es la capacidad de compensar perturbaciones. Normalmente, controlamos variables tales como la temperatura en sistemas térmicos, la posición y la velocidad en sistemas mecánicos, y el voltaje, la corriente o la frecuencia en sistemas eléctricos. El sistema debe ser capaz de proporcionar la salida correcta incluso con una perturbación. Por ejemplo, consideremos un sistema de antena que apunta en una dirección ordenada, si el viento desplaza la antena de dicha posición, o si el ruido ingresa internamente, el sistema debe ser capaz de detectar la perturbación y corregir la posición de la antena. La entrada del sistema seguirá siendo la misma, por tanto, el sistema mismo es el que debe medir la cantidad que la perturbación ha reposicionado la antena y luego devolver la antena a la posición comandada por la entrada. [7]

2.1.4 Funcionamiento de un sistema de posicionamiento

Un sistema de control de posición convierte una orden de entrada de posicionamiento en una respuesta de salida de posición. En este apartado, se verá en detalle un sistema de control de posición de acimut de antena que podría usarse para posicionar una antena de telescopio de radio.

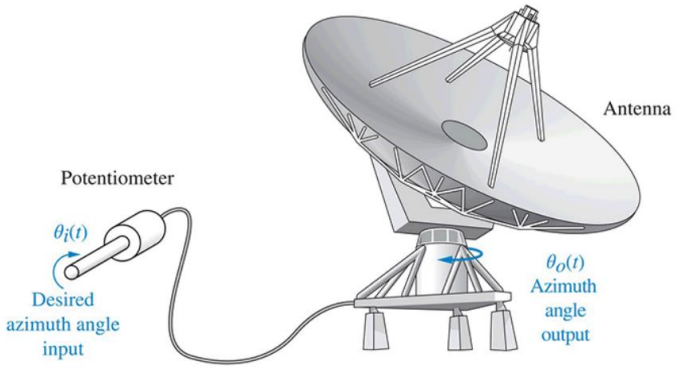

Figura 2-1 Esquema de un sistema de control de posicionamiento de antena [7]

El propósito final de este sistema es hacer que el ángulo de azimut de salida de la antena, θo(t), siga el ángulo de entrada de un potenciómetro, θi(t). La orden de entrada es un desplazamiento angular. El potenciómetro convierte el desplazamiento angular en un voltaje. De manera similar, el desplazamiento angular de salida se convierte en un voltaje por el potenciómetro en la ruta de retroalimentación. Los amplificadores de señal y potencia aumentan la diferencia entre los voltajes de entrada y salida. Esta señal de actuación amplificada impulsa el sistema.

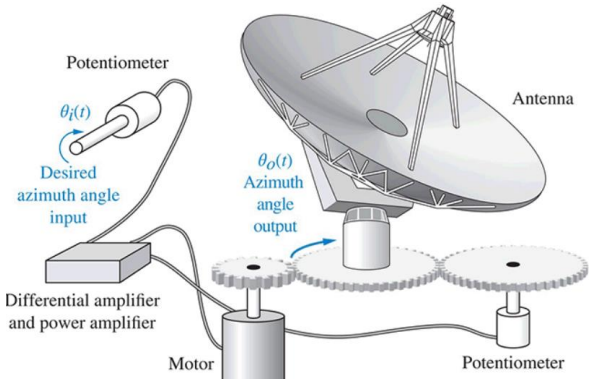

Figura 2-2 Esquema con engranajes de un sistema de control de posicionamiento de antena [7]

El sistema normalmente opera para llevar el error a cero. Cuando la entrada y la salida coinciden, el error será cero y el motor no gira, por tanto, el motor solo se mueve cuando la salida y la entrada no coinciden. Cuanto mayor sea la diferencia entre la entrada y la salida, mayor será el voltaje de entrada y más rápido girará el motor. Sin embargo, una mayor velocidad de giro a veces puede llevar a una sobre corrección o una respuesta transitoria que consiste en oscilaciones amortiguadas alrededor del valor en estado estacionario si la ganancia es alta.

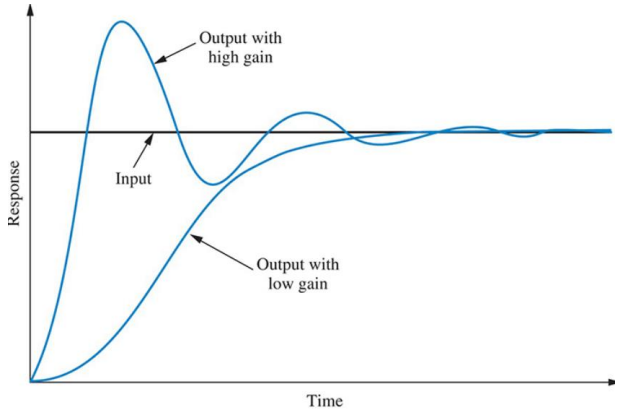

Figura 2-3 Respuesta de un sistema de control de posicionamiento [7]

Para resolver problemas como el anterior se utiliza un controlador con una respuesta dinámica, como un filtro eléctrico, junto con un amplificador. Con este tipo de controlador, es posible diseñar tanto la respuesta transitoria requerida como la precisión en estado estacionario requerida sin tener que depender de un amplificador de ganancia. Sin embargo, el controlador ahora es más complejo. El filtro en este caso se llama compensador. [7]

2.1.5 Componentes

Los sistemas de control de antenas tienen una amplia gama de aplicaciones, incluyendo telecomunicaciones, radiodifusión de radio y televisión, radar y comunicaciones por satélite. Los componentes específicos de un Sistema de Control de Antena pueden variar dependiendo del tipo de antena y aplicación, pero generalmente incluyen sensores, unidades de control, rotores y software de control que se explicarán en detalle a continuación. [4]

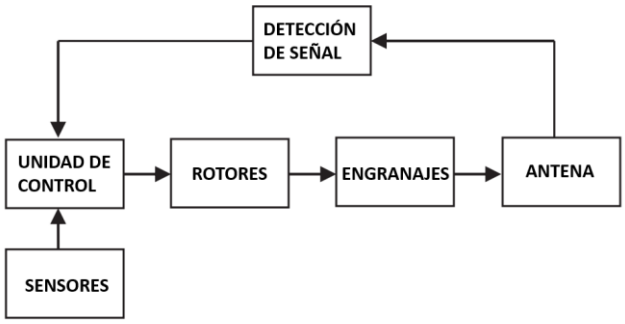

Figura 2-4 Esquema Sistema de control de antena [AP]

Sensores:

Los sistemas de posicionamiento de antenas cuentan con una gran cantidad de sensores, y dentro de un mismo equipo se utilizan varios tipos, entre los más destacados: sensores geomagnéticos, giróscopos de fibra óptica, sensores de tasa de viraje, codificadores ópticos, receptores GPS, potenciómetros, GPS-giro, brújula electrónica, magnetómetro, acelerómetro e inclinómetro. El ángulo de giro de la antena, por ejemplo, se mide usando un potenciómetro, la rotación de la antena en azimut a una tasa muy rápida se detecta mediante un giróscopo. Los ángulos de balanceo y cabeceo de la antena receptora se calculan usando el giroscopio. La posición y orientación de la antena, en términos de latitud, longitud y altitud, se detectan utilizando un receptor GPS. Para determinar la dirección de una antena con respecto al polo norte-sur se utiliza una brújula electrónica y para medir las inclinaciones de la antena en las direcciones X e Y se utiliza un inclinómetro. [3]

Unidad de control:

Es la parte principal del sistema. Lee datos analógicos de sensores o detectores de señal, los convierte en forma digital, los procesa y genera los comandos. Los comandos de salida de esta unidad son procesados previamente por el algoritmo de control, es decir, el software. Los controladores utilizados en la unidad de control suelen ser PC’s, microcontroladores, microprocesadores, DSPs (Procesadores de Señal Digital) y PLCs (Controladores Lógicos Programables). El PC suele ser la opción más recurrente ya que es fácil de programar y tiene un procesamiento de alta velocidad. Además, la capacidad de modificación de un programa y su implementación es la ventaja que presentan estos equipos. Estos utilizan algoritmos modificados y mejoras sin tener que cambiar el hardware. Por otro lado, la ventaja que presentan los microcontroladores y microprocesadores frente a los PC’s es el coste ya que es mucho menor en estos últimos.

Rotores:

Los motores utilizados para controlar los ejes de la antena son los motores paso a paso, motores de corriente continua (DC), motores de corriente continua con engranajes, motores de corriente continua sin escobillas (BLDC), motores de corriente alterna sin escobillas (AC) y servomotores de DC. El motor paso a paso es el más utilizado en el posicionamiento de antenas. Los motores síncronos son bastante caros y dan lugar a un sistema algo más complejo. Los motores de corriente continua ofrecen un comportamiento más lineal y por ello se han utilizado tradicionalmente en sistemas de seguimiento. La corta vida útil y el sobrecalentamiento del bobinado del inducido en los motores de DC limitan las aplicaciones repetitivas de alta velocidad. Los motores BLDC tienen ventajas como un alto torque para una respuesta dinámica de alta velocidad, alta eficiencia, larga vida útil, bajo ruido y mayor velocidad son también muy adecuados para aplicaciones de seguimiento de antenas, sin embargo, presentan un gran costo y dificultades en la reparación. En función del tipo de motor seleccionado, se utilizará un controlador u otro.

Sistema de engranajes:

El sistema de engranajes es uno de los factores clave en el sistema de seguimiento de antenas. La precisión, el tamaño y el torque del motor del sistema dependen del sistema de engranajes. Generalmente, los engranajes se clasifican como rectos, helicoidales, de doble hélice, sinfín y cónicos. El engranaje recto es más económico, pero produce ruido en el sistema a alta velocidad en comparación con los engranajes helicoidales. Los engranajes helicoidales minimizan la holgura del sistema. El uso de engranajes de reducción de holgura antibloqueo reduce el tamaño y el torque requerido de cada motor e incrementa la precisión de posicionamiento. El uso de la técnica de doble accionamiento reduce el efecto de la holgura. El costo de los engranajes depende del tamaño de la rueda dentada, la caja de engranajes y el material. Las cadenas de doble accionamiento con mecanismo de sesgo de torque reducen el ángulo de holgura. Para un motor paso a paso con un ángulo de paso de 1. 8º por paso y una relación de engranajes de 10:1, la precisión de posicionamiento es de 0. 18º en modo de paso completo y 0.09º en modo de medio paso del motor.

Software de control:

El software de control de antenas es un software especializado utilizado para gestionar y controlar la orientación, la precisión de apuntamiento y el seguimiento de antenas. Este software es esencial para sistemas de comunicación, telescopios, estaciones terrestres de satélite e instalaciones de radar, entre otras aplicaciones. Asegura que las antenas estén posicionadas con precisión para recibir o transmitir señales electromagnéticas y permite el control de las antenas de manera remota, esta es la mayor ventaja, sobre todo cuando la antena se ubica en lugares inaccesibles. Actualmente la mayoría de los software de control de antenas se centran en el seguimiento de satélites, sin embargo, sí que existen varios programas que permiten el control total de los rotores de manera independiente.

2.2 Componentes de los Sistemas de Posicionamiento de Antenas

2.2.1 Modelos de rotores para el control de antenas

Existen principalmente dos categorías de sistemas de rotor: el sistema azimutal y el sistema elevación/azimut. La principal diferencia entre ambos es que el primero solo permite el movimiento en un grado de libertad, paralelo al horizonte, mientras que el segundo cuenta con dos rotores que le permiten una rotación de 360° en la dirección acimutal y la elevación en el eje vertical. Para la radio por satélite, por ejemplo, se necesita un rotor de antena de este tipo debido a la procedencia de la señal, el cielo. Entre los modelos más vendidos en el mercado podemos destacar:

- Rotor Yaesu con controlador, este dispositivo ronda los 400 euros y tiene capacidad de carga hasta los 300 kg y un ángulo de rotación de 450 grados. El precio sin controlador baja hasta los 300 euros. [8]

Figura 2-5 Controlador Yaesu G-450, G-1000, G-2800 [8]

- Rotor Alfa Spid RAK, este presenta varias modalidades en función del voltaje que se requiera, pudiendo llegar hasta los 24 voltios y permitiendo un torque de 366 Nm. Existe la opción de adquirirlo con solo movimiento azimutal o con movimiento azimutal y vertical. Su precio ronda los 1200 euros. [9]

Figura 2-6 Controlador AlfaSpid RAK [9]

- Rotor de antena AR-500X, su precio está entorno a los 250 euros y se trata de un rotor de antena ligera para antenas pequeñas de hasta 30 kg, rotación de 360 ° con tope mecánico. Cuenta con una unidad de control para el funcionamiento de la fuente de alimentación 240 V con indicador de dirección. [10]

Figura 2-7 Controlador de antena AR-500X [10]

- Hygain HAM-IV Hygain DCU-3X, es la opción más popular para antenas con gran superficie. Cuenta con rodamientos de bolitas dobles o triples para la estabilidad máxima y tiene frenos electromecánicos que bloquean el eje del engranaje, previniendo que la antena de la vuelta en vientos fuertes. Al rotar la antena, el freno se abre automáticamente y se cierra una vez alcanzada la posición. Tiene un coste de 1700 euros. [11]

Figura 2-8 Rotor azimutal DCU-3X de Hygain [11]

- Rotor SPX el/az 01: se trata del rotor con el que se va a interactuar en el presente proyecto. Es el rotor que mueve la antena ubicada en el laboratorio de investigación del CUD. El SPX EL/AZ 01 es un rotor de antena ligero y preciso diseñado para sistemas de seguimiento de satélites, ofreciendo control a través de 4+4 cables y una precisión de 1 grado. Está equipado con un motor de corriente continua capaz de manejar antenas grandes de VHF/UHF, una placa de montaje en mástil opcional para montaje en superficies planas, y una unidad de control que cuenta con una amplia pantalla digital. Incluye un controlador digital ROT2 con interfaz USB, soportando una fuente de alimentación de 12V – 18V DC/3~6A. El precio de este rotor se encuentra entorno a los 940 euros.

Figura 2-9 Rotor SPX EL/AZ 01 [12]

2.2.2 Tipos de controladores

Las unidades de control de rotores rigen la dirección y el ángulo de orientación de la antena por lo que son una pieza fundamental en el ámbito de la radio y las telecomunicaciones, la precisión y flexibilidad que ofrecen los rotores de antena son críticos para lograr una recepción y transmisión de señales óptima. Este apartado profundiza en las unidades de control más populares para rotores de antena, explorando sus avances tecnológicos, interfaces de usuario y compatibilidad con diversos sistemas de antenas.

- microHAM Advanced Rotor Controller. También conocido como ARCO, es un controlador de rotores diseñado para operar de manera fiable con prácticamente cualquier rotor. ARCO se caracteriza por su diseño. Tiene la capacidad de modificar automáticamente las velocidades, aumentando o disminuyendo gradualmente para reducir el estrés inercial en la caja de engranajes del rotor, la antena y la torre, y así prolongar la vida útil del equipo. El controlador ofrece una variedad de interfaces para el control remoto. Cuenta con una interfaz gráfica de usuario clara y fácil de usar en una pantalla táctil de 7 pulgadas para controlar las direcciones de azimut o elevación y las muestra en un mapa del globo con el usuario en el centro. Este dispositivo permite establecer rápidamente direcciones a través de la función Touch'n Turn simplemente tocando la dirección deseada en el mapa. Los métodos de control adicionales incluyen botones táctiles en el panel frontal, un botón de Punto y Disparo, y un teclado de entrada de dirección, disponibles tanto virtualmente en la pantalla táctil como a través de una versión de hardware conectada al puerto USB host en el ARCO. El precio de este dispositivo ronda los 800 euros. [13]

Figura 2-10 Controlador ARCO [16]

- 4O3A Rotor Genius Control Unit. Este dispositivo trabaja con casi todos los rotores disponibles en el mercado. La electrónica es compatible con motores de CC y CA y en el rango de 3 a 48 V y con un consumo de corriente de hasta 15 A. Una unidad puede controlar dos motores independientes simultáneamente. El Rotator Genius viene con su propio sensor de azimut magnético calibrado. Este se monta en el brazo de la antena direccional y se conecta a la unidad de control con un cable Ethernet Cat5. De esta manera, la dirección absoluta de la antena siempre se conoce con una precisión de 1 grado angular. Dependiendo de la ubicación, también se puede compensar la desviación de la brújula entre la dirección norte verdadera y magnética. El precio de esta unidad es de unos 900 euros. [14]

Figura 2-11 Rotator Genius Control Unit [17]

- Spid Control unit Azi/Ele. Este dispositivo es un controlador diseñado para los rotores RAS y BIG-RAS, mencionados en el apartado anterior, que presenta una pantalla LED de 7 segmentos. Es una unidad de control precisa (0. 5º) y fiable para rotores de antenas, ofreciendo una interfaz amigable para la manipulación del azimut y la elevación de antenas. Esta unidad es especialmente conocida por su compatibilidad con modelos específicos de rotores, satisfaciendo las necesidades de aficionados a la radio y profesionales que requieren un control preciso y fiable sobre sus sistemas de antena. La inclusión de una interfaz USB facilita la conectividad fácil e integración con otros dispositivos, mejorando su funcionalidad y versatilidad en diversos montajes. Este dispositivo es el que se encuentra el laboratorio y con el que se llevaran a cabo las pruebas de precisión una vez desarrollado el software de control. [15]

Figura 2-12 Spid Control Unit Azi/Ele [18]

2.3 Software de control de rotores

Existen numerosos software de control del movimiento de antenas, la mayoría enfocados al seguimiento de satélites. A continuación, se exponen los más relevantes del mercado, así como una breve descripción de cada uno.

AIO Contest Logging Software by WD8KNC: Software de registro de concursos de Windows con múltiples funciones como control de rotor de antena, clúster DX, control CAT, telegrafía y control de voz, admite muchos modos de datos, búsqueda en QRZ, impresión de QSL y etiquetas, mapeo y más. [16]

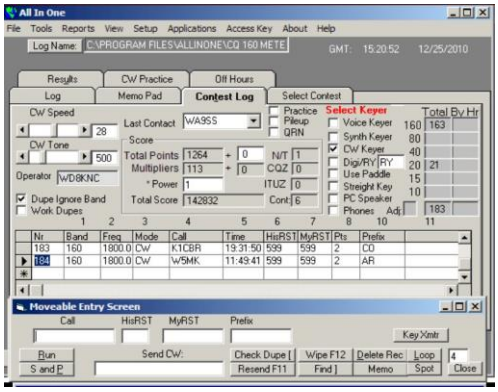

Figura 2-13 Contest Logging Software [20]

GJTracker: GJTracker es una aplicación de software diseñada para el seguimiento de satélites, específicamente adaptada para aficionados a la radio y a los satélites. Se enfoca en proporcionar información de seguimiento precisa para satélites en órbita terrestre, permitiendo a los usuarios alinear sus sistemas de antenas para una recepción y transmisión óptimas.[17]

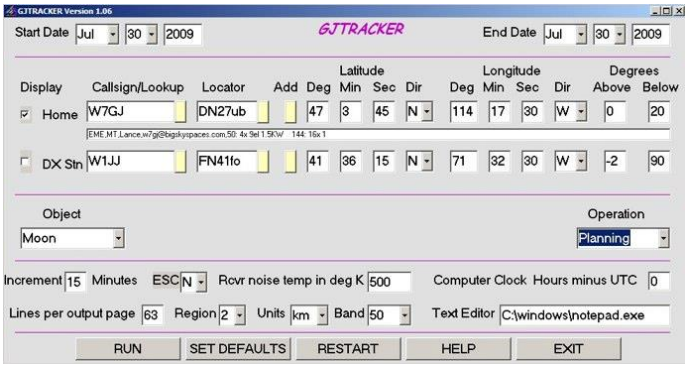

Figura 2-14 GJTracker [21]

LP-Rotor: software gratuito e independiente diseñado para el control de rotores que se utiliza en Windows. Su propósito es permitir a los radioaficionados y a otros usuarios controlar la dirección de sus antenas con precisión, facilitando así las comunicaciones o el seguimiento de satélites y otros objetos celestes.[18]

Figura 2-15 LP Rotor [22]

N1MM Rotor Control: software gratuito popular utilizado por operadores de radioaficionados. Tiene la capacidad de controlar hasta 16 rotores a la vez y ofrece funciones la verificación de duplicados, puntuación y soporte para una amplia gama de rotores. La característica más destacada de este programa es que mejora las capacidades del conjunto de herramientas al permitir a los operadores controlen sus rotores directamente desde el software. [19]

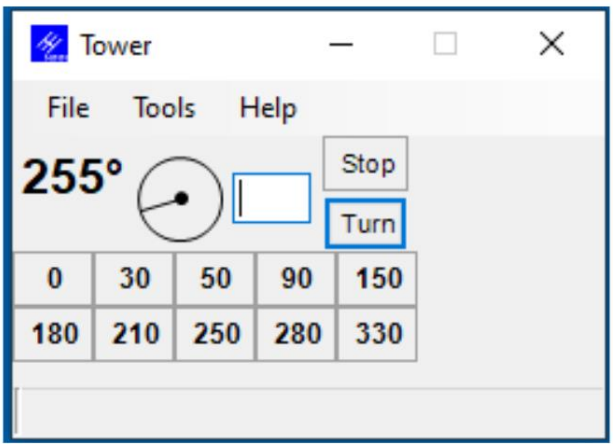

Figura 2-16 N1MM Rotor Control [24]

PstRotator: este controlador presenta una amplia gama de rotores controlables de uso común: Prosistel, Green Heron, HyGain, RC1 MDS, Rotor-EZ Idiom Press, Yaesu Az y muchos más. PstRotator incluye seguimiento automático para comunicaciones por satélite o EME, utilizando programas como WSJT, VQLog, TACLog, Z-Track, GJTracker, EME System, Trak_SM, WXtrack, WinOrbit, Orbitron, Nova for Windows, Sat_Explorer, SatPC32, Win-Test. PstRotator admite control remoto a través de TCP/IP. [20]

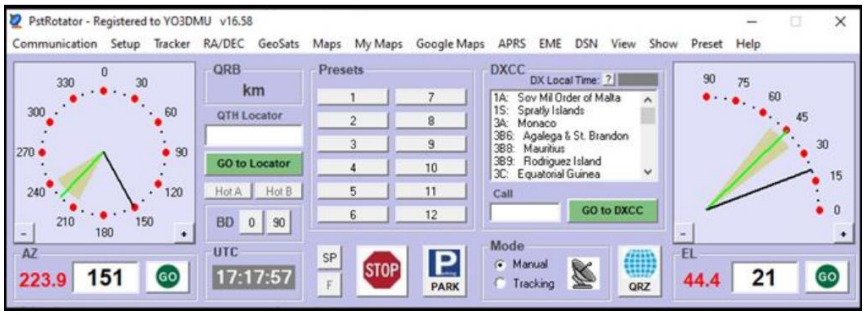

Figura 2-17 PstRotator Interface [25]

RemoteRotator: aplicación de línea de comandos que permite hacer disponibles en la red los rotores de antena de azimut/elevación. Está disponible para Linux, Windows y MacOS y está escrita en el lenguaje de programación Go. RemoteRotator es utilizado por operadores de radioaficionados para controlar rotores de antena tanto localmente como de manera remota, a través de una interfaz web simple, lo que permite su operación desde cualquier dispositivo (PC, tableta o teléfono móvil).[21]

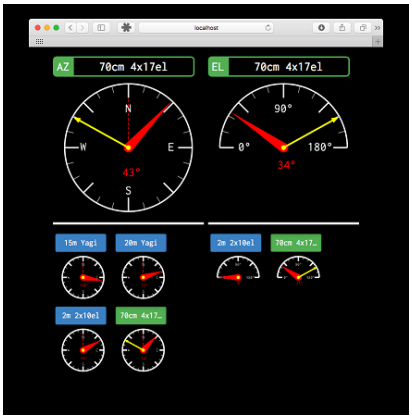

Figura 2-18 RemoteRotator [26]

RotorCraft: software de control de rotor de antena para Windows y Linux. Fue desarrollado como un proyecto complementario a un controlador de rotor basado en Arduino utilizado para una antena hexbeam. Este software ofrece una interfaz sencilla para el control de rotores, dirigida a ser una solución de código abierto y fácil de extender a otros tipos de rotores y protocolos. Su principal característica es una interfaz de usuario estándar que permite seleccionar la dirección objetivo con un clic y control manual del motor del rotor. [22]

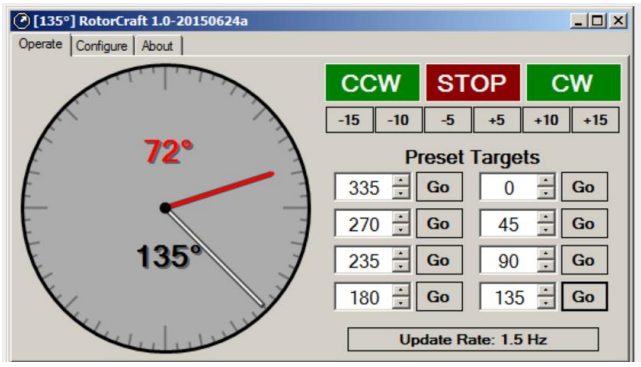

Figura 2-19 RotorCraft [27]

Satellite Tracking con WinRotor: software diseñado para el control de rotores de antenas, facilitando la orientación de antenas para radioaficionados y aplicaciones de seguimiento de satélites. Con versiones compatibles desde Windows 3.1 a sistemas más modernos de 32 bits, WinRotor se presenta como una solución versátil para la operación de antenas.[23]

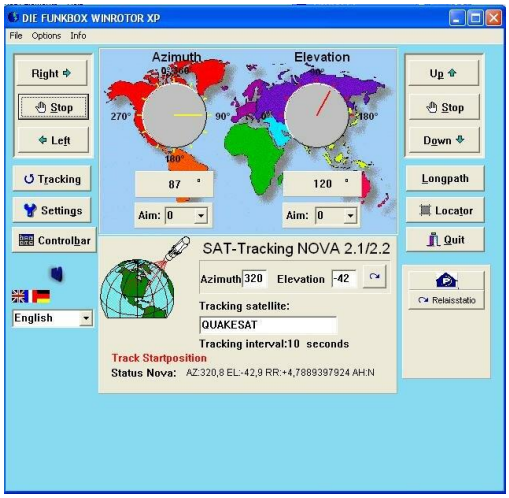

Figura 2-20 WinRotor Software Interface [29]

SimpleSat Rotor Controller: controlador de rotor para antenas, que utiliza el conjunto de comandos Yaesu GS-232 para controlar un rotor Yaesu G-5500 en azimut y elevación. Está diseñado para ser compatible con el programa de seguimiento SatPC32 y debería funcionar con cualquier programa de seguimiento que soporte el formato Yaesu GS-232.[24]

ARSVCOM: este programa presenta una interfaz gráfica amigable que te permite controlar tus antenas de manera sencilla. Incluye un gran extra: un puerto COM virtual que permite emular varios modelos de controladores de rotor como Yaesu, Create, Prosistel “D”, entre otros. De esta manera, cualquier programa de terceros (por ejemplo, HRD) detectará un modelo de rotor (Yaesu GS232A) a través de este Puerto COM Virtual y los comandos serán enrutados hacia la interfaz real (ARS o Prosistel).

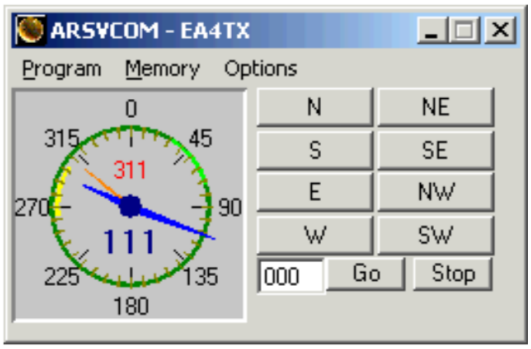

Figura 2-21 ARSVCOM software [31]

Rotctl (Hamlib): programa utilizado para controlar rotores de antenas en el ámbito de la radioafición. Permite enviar comandos desde la línea de comandos o en modo interactivo al simular el envío de comandos a un rotor "dummy", y está diseñado para trabajar con una amplia gama de modelos de rotores. Soporta opciones para seleccionar el modelo del rotor, especificar el dispositivo de conexión, ajustar la velocidad de la conexión serial entre otras. Rotctl es parte de Hamlib, una biblioteca de software en nivel beta que proporciona funciones básicas bien soportadas para controlar equipos de radio. [25]

Como se puede apreciar, la mayoría de los programas de control de rotores están enfocados al control de antenas y seguimiento de satélites diseñador para ajustar automáticamente la orientación de la antena y mantener la mejor conexión posible. El programa web que se desarrollará en este proyecto trabajará con el último controlador mencionado, Rotctl ya que este programa se ejecuta desde un terminal y carece de un interfaz de usuario. Más adelante se desarrollará en profundidad el funcionamiento de rotctl.

2.4 Desarrollo de Interfaces

2.4.1 Concepto y evolución del GUI

Una interfaz gráfica de usuario (GUI, por sus siglas en inglés) es una interfaz digital en la que un usuario interactúa con componentes gráficos como iconos, botones y menús. En una GUI, los elementos visuales mostrados en la interfaz de usuario transmiten información relevante para el usuario, así como acciones que pueden realizar.[26]

Hoy en día, es difícil imaginar las computadoras sin GUIs. Sin embargo, hubo un tiempo en el que ni siquiera existía el cursor de ratón. A continuación, se profundizará sobre la evolución del interfaz de usuario a lo largo de los años.

En los primeros días de la informática, la interacción del usuario era principalmente basada en texto a través de interfaces de línea de comandos (CLI). Los usuarios tenían que escribir comandos específicos para realizar tareas, lo que requería memorización y familiaridad con una sintaxis compleja. La CLI era eficiente para usuarios experimentados, pero representaba una barrera significativa de entrada para los novatos.[27]

La revolución de las GUI's comenzó en los años 70 y 80 con pioneros como Xerox PARC y las computadoras Lisa y Macintosh de Apple. Las GUI's introdujeron un cambio de paradigma al representar información y acciones gráficamente, usando iconos, ventanas y dispositivos de señalización como el ratón. Esto hizo que las computadoras fueran más accesibles para un público más amplio, ya que los usuarios ahora podían interactuar con la tecnología simplemente haciendo clic en iconos y arrastrando archivos.

Microsoft Windows, introducido a finales de los años 80, jugó un papel fundamental en la popularización de las GUIs. Windows trajo consigo la multitarea, mejores gráficos y un extenso

ecosistema de aplicaciones a las computadoras personales. La rivalidad entre Microsoft Windows y el Mac OS de Apple contribuyó a la rápida refinación e innovación de las GUIs.[28]

La introducción de pantallas táctiles, especialmente con el iPhone de Apple en 2007, marcó otro hito significativo. Las pantallas táctiles extendieron el concepto de GUI a dispositivos móviles, haciendo la interacción aún más intuitiva. Hoy en día, los smartphones, tablets e incluso algunos portátiles dependen en gran medida de las GUIs táctiles, proporcionando experiencias de usuario fluidas.

2.4.2 Software de diseño web

El mundo del diseño gráfico es dinámico y cambiante y conocer el arsenal de herramientas a disposición de los diseñadores es crucial para definir los límites de su creatividad y productividad. A medida que la industria continúa evolucionando a un ritmo vertiginoso, mantenerse al día con las herramientas de software de diseño gráfico más populares se ha vuelto indispensable para profesionales y aficionados por igual. A continuación, se presenta una visión general de las opciones de software principales que están moldeando el paisaje del diseño gráfico hoy destacando las herramientas que han ganado amplio reconocimiento por sus características robustas, versatilidad y capacidad para dar vida a visiones creativas.

- Adobe Dreamweaver. Herramienta integral de desarrollo web que ofrece una solución potente y flexible para el diseño, la codificación y la gestión de sitios web y aplicaciones web. Como parte de la suite Adobe Creative Cloud, Dreamweaver proporciona una interfaz visualmente rica que atiende tanto a desarrolladores novatos como profesionales al combinar herramientas de diseño visual con capacidades robustas de edición de código. Los usuarios pueden trabajar en un entorno "Lo Que Ves Es Lo Que Obtienes" (WYSIWYG), lo que permite la manipulación directa de elementos del sitio web en una vista de diseño, o pueden sumergirse en la vista de código para escribir HTML, CSS, JavaScript y otros estándares web manualmente. Dreamweaver soporta diseño responsive, lo que permite a los diseñadores crear páginas web que se adaptan sin problemas a diferentes tamaños de pantalla y dispositivos. También ofrece características de vista previa en tiempo real, sugerencias de código para acelerar la codificación e integración con otros productos de Adobe, mejorando la eficiencia del flujo de trabajo. Con su característica de FTP, los usuarios pueden subir fácilmente sus sitios web al servidor, haciendo de Dreamweaver una opción versátil para proyectos de desarrollo web que requieren una combinación de edición visual y profundidad de codificación. [29]

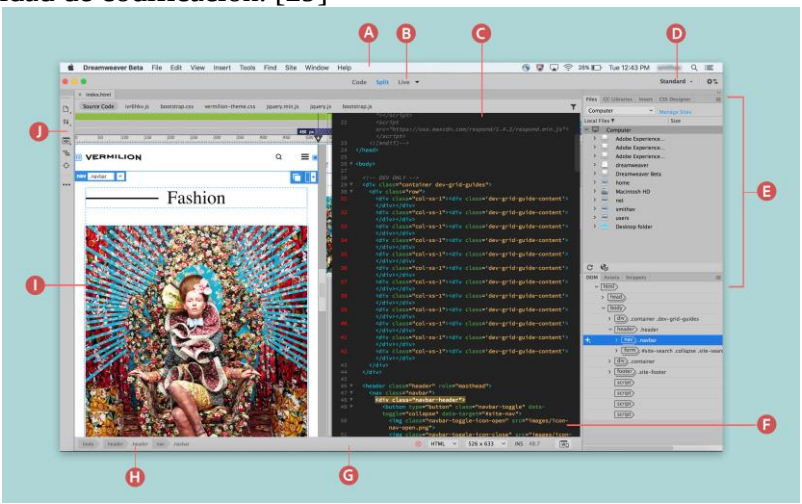

Figura 2-22 Adobe Dreamweaver Workspace [36]

- Webflow. Esta herramienta redefine los límites entre el diseño y el desarrollo web, ofreciendo una plataforma de diseño visual que genera HTML, CSS y JavaScript limpios y conformes con los estándares W3C. Permite a los diseñadores construir visualmente sitios web con vista previa en tiempo real y sin necesidad de codificación. Webflow es tanto una herramienta de diseño como una plataforma de alojamiento, proporcionando una transición sin fisuras del diseño al sitio web en vivo. Incluye capacidades de CMS y comercio electrónico, haciendo posible diseñar, construir y lanzar sitios web dinámicos y complejos. La interfaz de Webflow puede ser más compleja que las herramientas de diseño tradicionales, dada su extensa funcionalidad, pero sigue siendo una de las herramientas más poderosas para los diseñadores que quieren tomar el control total del proceso de desarrollo web sin un profundo conocimiento de codificación. [30]

Figura 2-23 Webflow Workspace [30]

2.4.3 IDE's

Para el desarrollo de la aplicación web, además de los software anteriormente expuestos, podemos considerar también crear el interfaz mediante un IDE, para ello debemos conocer primero el concepto de IDE para posteriormente elegir el más conveniente.

Un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) es una aplicación de software que proporciona facilidades completas a los programadores de computadoras para el desarrollo de software. Un IDE típicamente incluye un editor de código fuente, herramientas de automatización de compilación y un depurador. Algunos IDEs también ofrecen características adicionales como control de versiones, fragmentos de código y herramientas para la gestión de proyectos. El propósito principal de un IDE es agilizar el proceso de desarrollo combinando las actividades comunes de escribir, probar y depurar software en una única interfaz amigable para el usuario, aumentando así la productividad y simplificando el flujo de trabajo de desarrollo. [31]

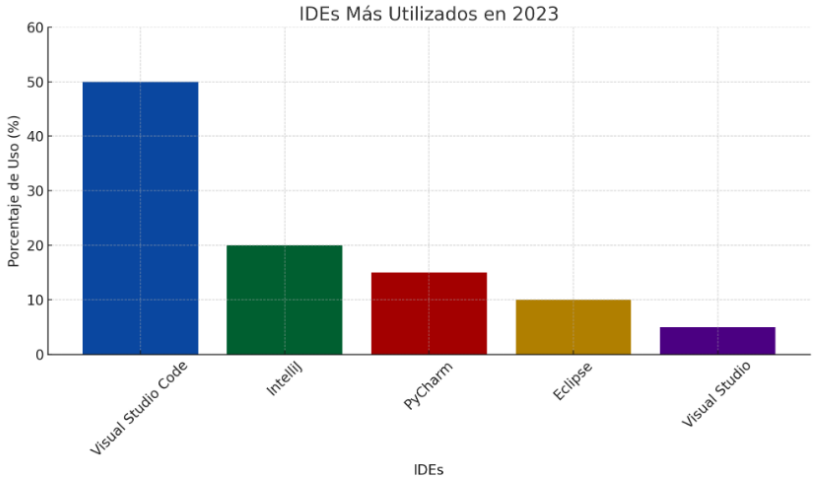

Tabla 2-1 IDE's más usados [AP]

- Visual Studio Code, desarrollado por Microsoft, es un potente IDE de código abierto que ha ganado una inmensa popularidad entre los desarrolladores por su rendimiento, flexibilidad y extensa biblioteca de extensiones. Soporta numerosos lenguajes de programación, incluidos HTML, CSS y JavaScript, que son fundamentales para el desarrollo de un *front-end*. La característica IntelliSense de VS Code ofrece complementos inteligentes basados en tipos de variables, definiciones de funciones y módulos importados. El IDE también se integra con Git y otros proveedores de SCM (Supply Chain Management), haciendo que el control de versiones sea fluido.

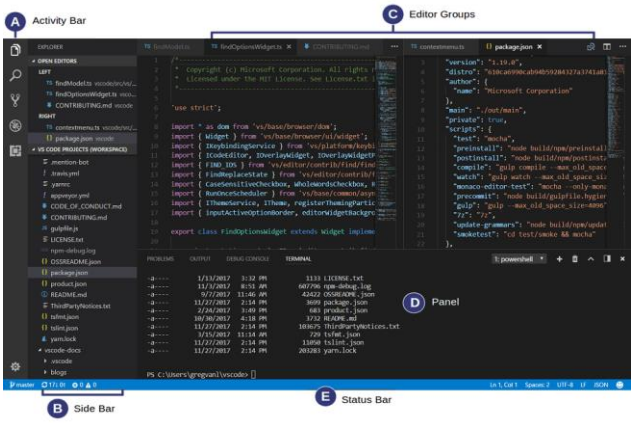

Figura 2-24 VS Studio Workspace [32]

- WebStorm, de JetBrains, es un IDE comercial específicamente diseñado para el desarrollo moderno de JavaScript. Proporciona asistencia avanzada para codificación en JavaScript, TypeScript, HTML, CSS y marcos modernos como React, Angular y Vue.js. Las características clave de WebStorm incluyen herramientas potentes de navegación y refactorización, análisis de código en tiempo real y un depurador integrado. Su integración con sistemas de control de versiones populares, ejecutores de tareas y herramientas de construcción lo convierte en una solución integral para desarrolladores *front-end*.

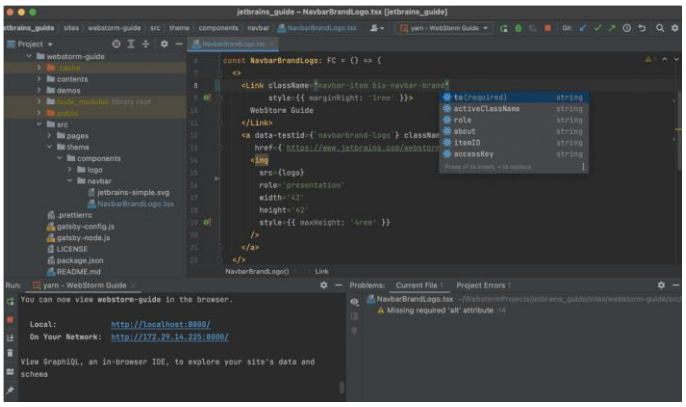

Figura 2-25 WebStorm Workspace [33]

- Atom, desarrollado por GitHub, es un editor de texto de código abierto que puede transformarse en un IDE con la adición de paquetes. Soporta de forma nativa HTML, CSS, JavaScript y Node.js, y ofrece características como autocompletado inteligente, un navegador de sistema de archivos y múltiples paneles. El paquete de Atom IDE añade características adicionales como un depurador, análisis estático más profundo y más, lo que lo convierte en una opción versátil para el desarrollo *front-end.* Su integración sin problemas con GitHub y Git hace que la colaboración y el control de versiones sean sencillos.

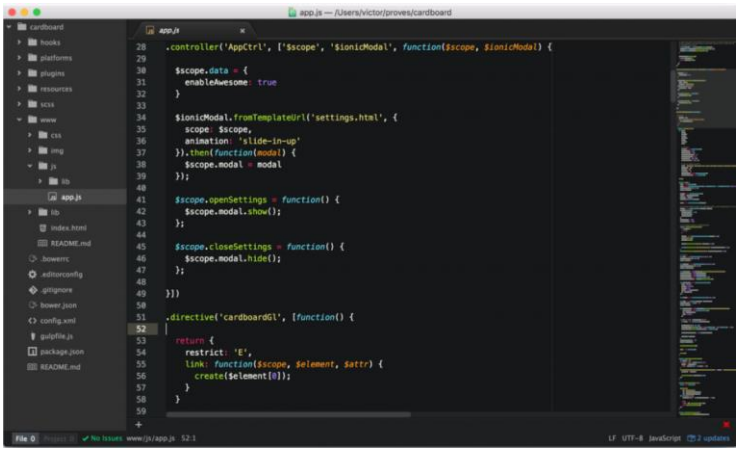

Figura 2-26 Atom Workspace [34]

- Brackets es un editor de texto de código abierto diseñado específicamente para diseñadores web y desarrolladores *front-end*. Ofrece una interfaz limpia y poderosas características de edición como la vista previa en vivo, que permite a los desarrolladores ver cambios en tiempo real en el navegador, y soporte para preprocesadores. El enfoque de Brackets en herramientas visuales y soporte para preprocesadores lo hace particularmente útil para la edición de CSS, HTML y JavaScript. Su naturaleza ligera y la capacidad de extender su funcionalidad a través de extensiones lo hacen un favorito entre los desarrolladores web.

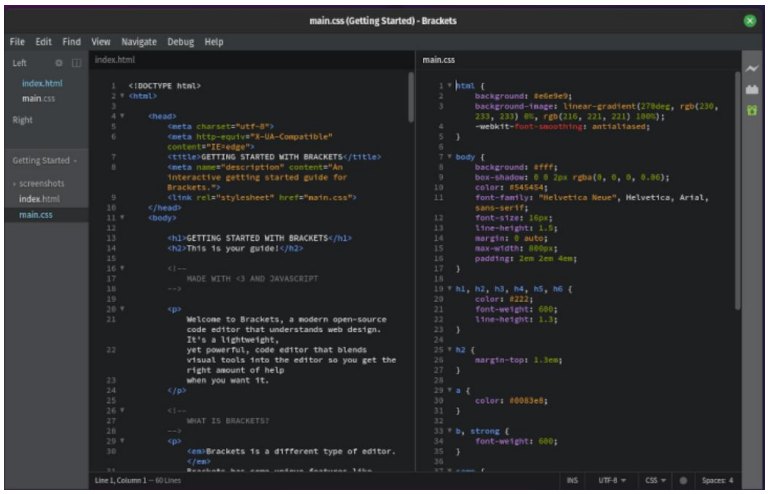

Figura 2-27 Brackets Workspace [35]

- Tkinter. Tkinter es un enlace de Python al toolkit de GUI Tk, una interfaz estándar que viene incluido con Python. Resulta una opción popular para crear interfaces gráficas de usuario (GUI) con Python debido a su simplicidad y al hecho de que se incluye con la distribución estándar de Python.
  Tkinter se construye sobre Tk, un toolkit de GUI utilizado por el lenguaje Tcl. La combinación de Tcl y Tk ofrece una manera de crear aplicaciones multiplataforma que funcionan en Windows, macOS y Linux. Esta librería de Python permite a los desarrolladores crear GUIs simples pero funcionales con una cantidad mínima de código, aprovechando las capacidades de Tk. Tkinter actúa como una capa entre Python y Tk, permitiendo que las aplicaciones de Python implementen elementos de GUI como ventanas, botones, campos de texto y más. Al crear un interfaz con Tkinter, la aplicación de Python se comunica con Tk, que a su vez maneja la representación y el manejo de eventos de los componentes GUI en la pantalla del usuario.
  Toda aplicación de Tkinter tiene un bucle de evento principal. Este bucle es esencial porque mantiene la aplicación funcionando y receptiva a acciones del usuario, como clics y presiones de teclas. Cuando el bucle está en funcionamiento, continúa verificando eventos y actualiza el interfaz en consecuencia.

Figura 2-28 TKinter Logo [36]

2.4.4 Framework

Para el desarrollo del software de control también es necesario conocer lo que es un *framework*. Un marco o *framework* en el desarrollo de software es una base o estructura sobre la cual los desarrolladores pueden construir para crear aplicaciones. Proporciona un conjunto de código preescrito, bibliotecas y

herramientas organizadas de manera que permite a los programadores desarrollar proyectos de manera más eficiente mediante la reutilización de código para tareas comunes, siguiendo pautas específicas y manteniendo un enfoque estructurado. Los marcos pueden ser específicos para lenguajes de programación o tecnologías y, a menudo, tienen como objetivo simplificar el desarrollo de aplicaciones resolviendo problemas comunes, acelerando así el proceso de desarrollo y asegurando la consistencia y las mejores prácticas. Los marcos se diferencian de las bibliotecas en que dictan la arquitectura y el flujo de su aplicación, a menudo a través de una "inversión de control", donde el marco llama a su código en lugar de que su código llame al marco (lo que es más típico en las bibliotecas)[37]. A continuación, se exponen los *framework* más populares entre usuarios actualmente:

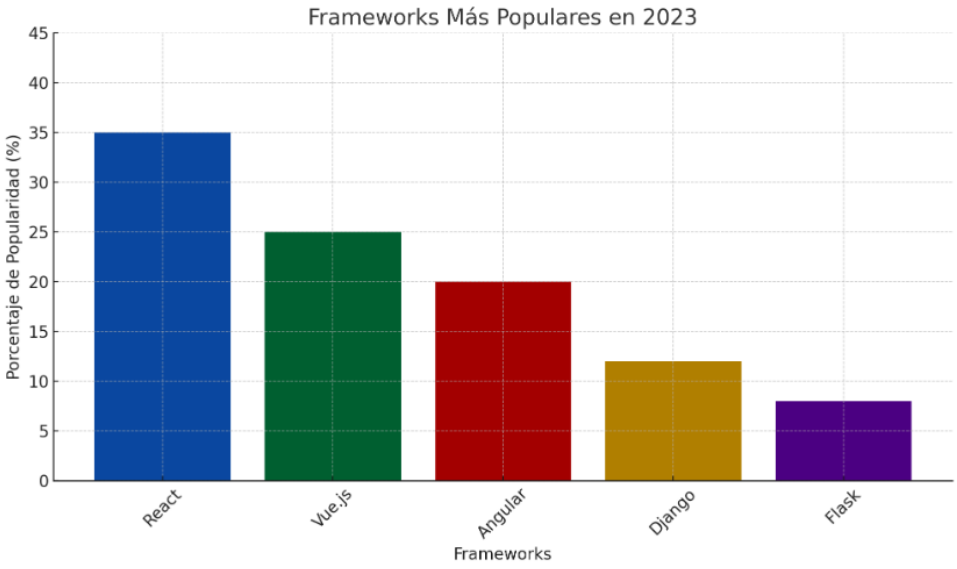

Tabla 2-2 Frameworks más populares [AP]

- Angular (JavaScript): Desarrollado por Google, Angular es una plataforma y marco de trabajo para construir aplicaciones de cliente de una sola página utilizando HTML y TypeScript. Angular ofrece una solución integral con herramientas para todo, desde construir la interfaz de usuario hasta probar y desplegar tu aplicación. Promueve la organización del código y respalda las mejores prácticas a través de su inyección de dependencias, utilidades para el testing unitario de componentes y una arquitectura modular.

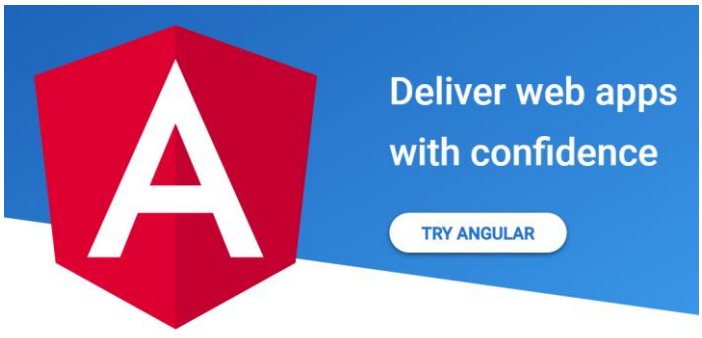

Figura 2-29 Logo Framework Angular [38]

- Vue.js (JavaScript): Vue.js es un marco de trabajo progresivo de JavaScript utilizado para construir interfaces de usuario y aplicaciones de una sola página (SPA, por sus siglas en inglés). Está diseñado para ser adoptado incrementalmente. La biblioteca central se centra solo en la capa de vista, lo que facilita su aprendizaje e integración con otras bibliotecas o proyectos existentes. Vue también es perfectamente capaz de potenciar aplicaciones de una sola página sofisticadas cuando se utiliza en combinación con herramientas modernas y bibliotecas de apoyo.

Figura 2-30 vue.js Framework [39]

- Express.js (Node.js): Express.js, o simplemente Express, es un marco de trabajo para aplicaciones web de *back-end* para Node.js, lanzado como software libre y de código abierto bajo la Licencia MIT. Está diseñado para construir aplicaciones web y APIs. Ha sido denominado el marco de servidor estándar de facto para Node.js. La filosofía de Express es proporcionar herramientas pequeñas y robustas para servidores HTTP, lo que lo convierte en una gran solución para aplicaciones de una sola página, sitios web, híbridos o APIs HTTP públicas.

Figura 2-31 Node.js Logo [40]

- Bootstrap: este es un marco de trabajo de *front-end* libre y de código abierto para desarrollar proyectos web "responsive" y orientados a dispositivos móviles. Ofrece una amplia gama de herramientas de CSS y JavaScript para crear componentes de interfaz de usuario como formularios, botones y navegaciones, asegurando una funcionalidad consistente en todos los dispositivos y tamaños de pantalla. Originalmente desarrollado por Twitter, Bootstrap simplifica el diseño web con sus componentes listos para usar, convirtiéndolo en uno de los favoritos entre los desarrolladores por su facilidad de uso, adaptabilidad y sólido apoyo comunitario.

Figura 2-32 Bootstrap Logo [41]

2.4.5 Python Flask y Flask SocketIO

Python Flask es un *framework* de aplicación web ligero y flexible que permite a los desarrolladores construir aplicaciones web escalables y fáciles de mantener. Opera como un *microframework*, lo que

significa que proporciona lo esencial para construir aplicaciones web sin imponer un requisito específico de herramientas o bibliotecas.

Flask funciona permitiendo a los desarrolladores configurar rutas y vistas dentro de la aplicación, las cuales son funciones de Python asociadas con URLs específicas. Cuando un usuario accede a una URL, Flask la empareja con la vista correspondiente y ejecuta la función, generando una página web como respuesta. Esto hace que Flask sea altamente adaptable, ya que los desarrolladores pueden integrarlo fácilmente con otras bibliotecas y herramientas de Python para tareas como interacción con bases de datos, validación de formularios y autenticación. La simplicidad de Flask, combinada con sus poderosas capacidades para el desarrollo de aplicaciones web, lo convierte en una opción clave para el desarrollo de la aplicación web de este proyecto.

Por otro lado, es importante definir o conocer la extensión de Flask-SocketIO, encargado de manejar la comunicación en tiempo real entre el cliente y el servidor utilizando WebSockets. El *framework* de Flask lleva integrado Socket.IO la cual es una biblioteca que permite la comunicación en tiempo real, bidireccional y basada en eventos entre clientes web y servidores. Esto mejora significativamente las capacidades de Flask, permitiendo el desarrollo de aplicaciones que requieren interacciones en vivo, como aplicaciones de chat, notificaciones en vivo y plataformas colaborativas, o en el caso del presente proyecto, la obtención en tiempo real de la posición de una antena o el envío de órdenes a esta.

Funcionamiento de Flask-SocketIO

Flask-SocketIO es una extensión de Flask que añade soporte para comunicación WebSocket, la cual mantiene una conexión persistente entre el cliente y el servidor a diferencia del ciclo tradicional de solicitud/respuesta HTTP, donde cada solicitud inicia una nueva conexión. A continuación, se expande sobre el funcionamiento de esta biblioteca.

Inicialización: Para usar Flask-SocketIO, se inicializa envolviendo el objeto de aplicación Flask. Esto permite que Flask-SocketIO maneje conexiones WebSocket junto con rutas Flask regulares.

Manejo de Eventos: Flask-SocketIO funciona definiendo y activando eventos. El desarrollador puede definir eventos personalizados que están asociados con funciones (evento-handlers). Cuando un evento específico es emitido por el cliente, el manejador de eventos correspondiente en el servidor es llamado.

Salas y Namespaces: Flask-SocketIO permite dividir conexiones en salas y namespaces, permitiendo un control más granular sobre qué clientes reciben mensajes específicos. Esto es particularmente útil para aplicaciones donde se necesita dirigir mensajes a subconjuntos de usuarios.

Integración con Aplicaciones Flask: Flask-SocketIO se integra sin problemas con Flask, permitiendo al desarrollador combinar el manejo de rutas HTTP regulares con el manejo de eventos WebSocket dentro de la misma aplicación.

Comunicación Cliente-Servidor:

Del Cliente al Servidor: El cliente (en este proyecto es un navegador web) utiliza una biblioteca cliente de Socket.IO para establecer una conexión con el servidor y emitir eventos. Estos eventos pueden llevar datos, por lo general serán valores numéricos para establecer una posición de la antena.

Del Servidor al Cliente: El servidor puede emitir eventos a uno o más clientes conectados, actualizando o notificando en tiempo real. Flask-SocketIO facilita la transmisión a todos los clientes o la emisión a clientes específicos.

2.5 Hamlib y sus Librerías

2.5.1 HAMLIB

Hamlib es una biblioteca de software de código abierto diseñada para facilitar el control de radios y otros equipos de comunicaciones por aplicaciones de software. El nombre "Hamlib" es un acrónimo de "ham", que se refiere a los operadores de radioaficionados, y "lib", indicando que es una biblioteca. Hamlib presenta una serie de características que la convierte en una opción muy popular entre los radioaficionados.

Hamlib ofrece compatibilidad entre plataformas, funciona en varios sistemas operativos, incluyendo Windows, Linux y MacOS, permitiendo una amplia accesibilidad e integración con diferentes sistemas. Además, soporta una amplia gama de dispositivos de radioaficionados, incluyendo transceptores, receptores y rotores de varios fabricantes. Esto se logra a través de una capa de abstracción que proporciona una interfaz estandarizada para controlar estos dispositivos, independientemente del hardware subyacente.

Es una librería que está principalmente escrita en C por eficiencia y portabilidad, pero proporciona enlaces para otros lenguajes de programación como Python, Perl y Ruby. Esto la hace accesible para desarrolladores con diferentes antecedentes de programación e integra fácilmente con varios proyectos de software.

Cuenta con una API para abstraer la comunicación con el equipo de radio lo que simplifica el desarrollo de aplicaciones de software de radioaficionado. Los desarrolladores pueden centrarse en la funcionalidad de sus aplicaciones sin necesidad de lidiar con las especificidades del hardware.

Como proyecto de código abierto, Hamlib se beneficia de las contribuciones de una comunidad de desarrolladores y entusiastas de la radioafición. Este enfoque colaborativo ayuda a extender el soporte para nuevos dispositivos, corregir errores y mejorar la biblioteca con el tiempo.

Tiene una amplia gama de aplicaciones de radioaficionado, incluyendo software de comunicación de modos digitales, programas de seguimiento de satélites y software de control automático de estaciones, por nombrar algunos. [42], [43]

2.5.2 ROTCTL

Rotctl es un programa de línea de comandos que forma parte de la biblioteca Hamlib, diseñado específicamente para el control de rotores de antena.

Rotctl presenta numerosas funcionalidades entre las cuales destacan:

- Interfaz de control universal: "*rotctl"* proporciona una interfaz uniforme para controlar varios modelos y marcas de rotores de antena. Esta abstracción permite a los usuarios escribir scripts o comandos que funcionan en diferentes hardware sin necesidad de preocuparse por el conjunto de comandos específico de cada rotor.
- Operaciones desde la línea de comandos: es una herramienta apta para la automatización y se puede integrar fácilmente en scripts, permitiendo operaciones de rotación programadas o desencadenadas por eventos, integración con otro software de radioaficionado o escenarios de control remoto.
- Amplio Soporte de Hardware: a través de la biblioteca Hamlib, *rotctl* soporta una amplia gama de modelos de rotores de diferentes fabricantes. El proyecto Hamlib actualiza continuamente su lista de dispositivos compatibles a medida que se lanzan nuevos modelos y los contribuyentes añaden soporte para ellos.
- Características y Comandos: *rotctl* ofrece una variedad de comandos para el control del rotor, incluyendo:

  - Establecer y consultar la posición del rotor en acimut y elevación
  - Mover el rotor en una dirección específica
  - Detener el movimiento del rotor
  - Configurar y consultar ajustes y estados del rotor.
- Integración con Software de Radioaficionado: *rotctl* se utiliza a menudo en conjunto con otro software de radioaficionado para aplicaciones como seguimiento de satélites, control automático de estaciones y registro. Permite que estas aplicaciones controlen los rotores de antena sin problemas, proporcionando funcionalidad mejorada al equipo de radio.
- Scripting y Automatización: es posible automatizar la rotación de antenas a través de scripts que llaman comandos de *rotctl*, permitiendo operaciones complejas como el seguimiento de objetos en movimiento (por ejemplo, satélites) o alinear automáticamente las antenas para una recepción óptima de señales basada en horarios predefinidos o eventos.
- Control en Red: *rotctl* también se puede utilizar a través de una red, permitiendo a los operadores controlar rotores de antena desde ubicaciones remotas. Esta característica es particularmente útil para operadores de radioaficionados que gestionan estaciones remotas o para fines educativos donde estudiantes pueden controlar equipos desde diferentes ubicaciones.

2.5.3 ROTCTLD

Rotctld es un *Daemon* (programa informático que se ejecuta en segundo plano, sin control directo del usuario y que realiza tareas específicas de manera autónoma o en respuesta a ciertos eventos) incluido en la suite Hamlib, diseñado para proporcionar capacidades de servidor de red y controlar rotores de antena de manera remota. Este *daemon* actúa como un servidor que escucha comandos de control a través de una red, permitiendo a clientes conectarse y enviar comandos para ajustar la posición de un rotor de antena. Esta característica es especialmente útil para configuraciones de radioaficionados donde se desea la operación remota, permitiendo a operadores ajustar la orientación de sus antenas desde cualquier ubicación con acceso a la misma red.

Entre las principales funcionalidades que presenta *Rotctld* podemos destacar:

- Accesibilidad de Red: *Rotctld* permite el control de rotores de antena a través de una red, incluyendo redes locales (LAN) o Internet. Esta accesibilidad amplía el alcance de la gestión de antenas, facilitando la operatividad de estaciones remotas.
- Compatibilidad con Múltiples Rotores: Al igual que *rotctl*, *rotctld* soporta una amplia variedad de modelos de rotores a través de la biblioteca Hamlib. Esto significa que los usuarios pueden interactuar con diferentes hardware sin preocuparse por los protocolos de comunicación específicos o comandos requeridos por cada modelo.
- Modelo Cliente-Servidor: *Rotctld* opera en un modelo cliente-servidor, donde *Rotctld* actúa como el servidor, y varias aplicaciones cliente (incluyendo scripts personalizados u otro software habilitado por Hamlib) pueden conectarse para enviar comandos y órdenes de giro. Este modelo permite que múltiples clientes interactúen con el rotor, mientras que el servidor gestiona estas solicitudes de manera apropiada.
- Interfaz de Comando Estandarizada: Los clientes se comunican con *Rotctld* usando un conjunto estandarizado de comandos, que son, en esencia, los mismos que se usan con *rotctl* para el control directo. Esta consistencia simplifica el desarrollo de aplicaciones y scripts cliente.
- Integración con Otro Software: Muchas aplicaciones de software de radioaficionados que requieren funcionalidad de rotación de antenas pueden integrarse con *Rotctld* para controlar rotores. Esto incluye software de seguimiento de satélites, como *Gpredict*, sistemas automáticos de gestión de estaciones y software de competición*, proporcionando una experiencia operativa sin fisuras.

La operación y uso de *Rotctld* es bastante intuitiva y funciona a través de comandos. La puesta en marcha del *Daemon* se hace normalmente desde la línea de comandos con opciones que especifican el modelo del rotor, el puerto de comunicación (para conexiones seriales o USB al rotor) y el puerto de red para aceptar conexiones de clientes; los clientes se conectan a *Rotctld* usando protocolos como TCP/IP, y envían comandos para consultar o establecer la posición del rotor. El *daemon* procesa estos comandos y se comunica con el hardware del rotor; por último, el *Daemon* proporciona retroalimentación a los clientes sobre la ejecución de comandos y la posición actual del rotor, permitiendo bucles de control y ajustes basados en información en tiempo real.

2.6 GUI's de ROTCTL y ROTCTLD

El mundo de la radioafición se puede considerar muy colaborativo, la biblioteca Hamlib, específicamente sus herramientas rotctl y *Rotctld*, han fomentado un ecosistema de interfaces y aplicaciones desarrolladas por usuarios. A través de diversos foros, repositorios de GitHub y otras plataformas colaborativas, radioaficionados y desarrolladores han llevado las funcionalidades que ofrecen *rotctl* y *Rotctld* a nuevas alturas, creando una amplia gama de GUI's que atienden a diversas necesidades operativas y preferencias. Estos proyectos impulsados por la comunidad mejoran significativamente la accesibilidad y usabilidad del control de rotores de antenas. Este capítulo profundiza en el panorama actual de estos interfaces desarrollados por usuarios, mostrando la relación colaborativa entre desarrolladores de software y radioaficionados.

GitHub es un sitio web y servicio basado en la nube que permite a los desarrolladores almacenar, gestionar y compartir su código. Podemos considerar GitHub como una plataforma donde los desarrolladores pueden colaborar en proyectos, rastrear errores y solicitudes de características, así como alojar y revisar código en varios lenguajes de programación. Es ampliamente utilizado por desarrolladores individuales, equipos y empresas para trabajar en proyectos tanto pequeños como grandes, facilitando la colaboración y manteniendo un historial del desarrollo de un proyecto. Es en este repositorio donde encontramos gran parte de los *front-end* de *Rotctld* desarrollados hasta ahora.

PyRotor (dfannin) [44]

Pyrotor es una interfaz gráfica de usuario (GUI) desarrollada por el usuario de github *dfannin* para la gestión de rotores de antenas a través del marco de trabajo rotctl/hamlib, está diseñado especialmente para integrarse con el software controlador de rotor de antena K3NG [45]. Esta aplicación mejora la interacción del usuario ofreciendo una visualización similar a una brújula para el seguimiento del azimut, soporta actualizaciones periódicas mediante sondeos y facilita la gestión del daemon *Rotctld*, incluyendo su activación y control.

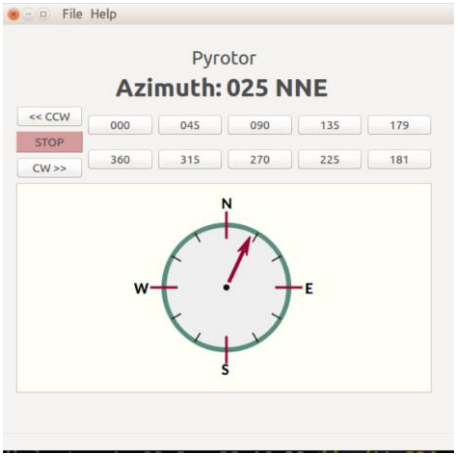

Figura 2-33 Pyrotor Dfanning [44]

Cuenta con un menú de configuración, junto con la capacidad de manejar archivos de configuración tanto para lectura como para escritura. Además, Pyrotor incorpora controles directos con botones para la gestión del rotor. Logra la comunicación con los controladores de rotor de antena utilizando el programa rotctl, que a su vez se interfaz con el daemon *Rotctld* para ejecutar tareas de control y monitoreo del sistema de rotor de antena.

Para el desarrollo de este GUI se utilizó QT designer y para la instalación del programa son necesarios los siguientes requisitos: Linux, Python 3.5 o mayor, pyqt5, libqt5, qt designer, hamlib 3.0 o mayor.

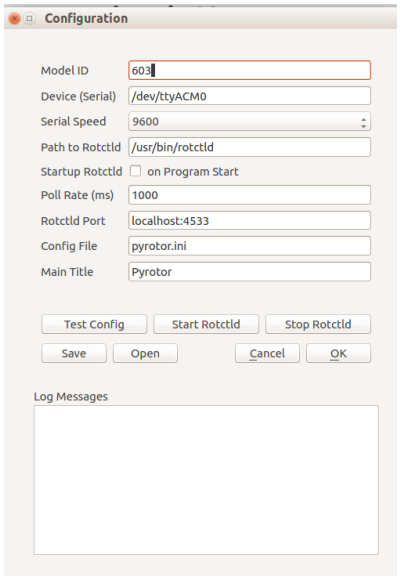

Figura 2-34 Menú Configuración de PyRotor [44]

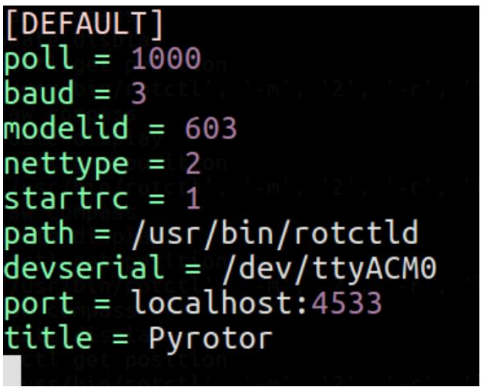

Figura 2-35 Conexión establecida con PyRotor [44]

***Rotctld*-web-gui (Mark Jessop)** [46]

Este programa desarrollado por el usuario de github *darksidelemm* es una interfaz gráfica de usuario (GUI) para controlar un rotor de antena a través del daemon *Rotctld* de la biblioteca Hamlib y la ventaja que presenta es que permite el control del rotor mediante dispositivos móviles al crear un servidor web. El programa está escrito en Python 2.7 y utiliza el marco web Flask junto con Flask-SocketIO para la comunicación en tiempo real entre la interfaz web y el servidor. A continuación, se detalla su funcionalidad:

1. Inicialización del Servidor Web: El script inicializa una aplicación Flask, configurándola para desarrollo con características como la recarga automática de plantillas. Flask-SocketIO se utiliza para habilitar la comunicación WebSocket, permitiendo actualizaciones en tiempo real sin necesidad de recargar la página web.
2. Variables Globales: Define variables globales para rastrear la posición actual y el punto de ajuste del rotor de antena en términos de azimut (dirección horizontal) y elevación (ángulo vertical).
3. Clase *ROTCTLD*: Se define una clase Python personalizada *ROTCTLD* para manejar la comunicación con el daemon *Rotctld*. Esto incluye abrir y cerrar la conexión, enviar comandos y analizar respuestas. La clase puede establecer y obtener el azimut y la elevación del rotor, detener el movimiento y manejar tiempos de espera en la comunicación.
4. Rutas de Flask: El script configura una ruta principal ("/") para servir la página índice de la interfaz web. Es aquí donde el usuario interactúa con la GUI.
5. Manejadores de SocketIO: Se definen manejadores de eventos para las comunicaciones de SocketIO, manejando acciones como actualizar el punto de ajuste del rotor, mover el rotor a una posición "home", detener el rotor y obtener la posición actual. Estas acciones son activadas por interacciones del usuario con la GUI web e involucran enviar comandos al rotor a través de la clase *ROTCTLD*.
6. Actualizaciones en Tiempo Real: La biblioteca Flask-SocketIO se utiliza para emitir eventos en tiempo real a la interfaz web, permitiendo la actualización dinámica de la posición y el punto de ajuste del rotor sin recargas de página.
7. Argumentos de Línea de Comandos: El script acepta argumentos de línea de comandos para especificar el puerto del servidor web y el nombre de host y puerto del servidor *Rotctld*, permitiendo una implementación flexible.
8. Flujo de Ejecución Principal: Al ejecutarse, el script intenta conectarse al daemon *Rotctld* usando el nombre de host y puerto proporcionados. Si tiene éxito, inicia la aplicación Flask y el servidor WebSocket, permitiendo a los usuarios controlar el rotor a través de la interfaz web. La conexión a *Rotctld* se cierra cuando la aplicación se sale.

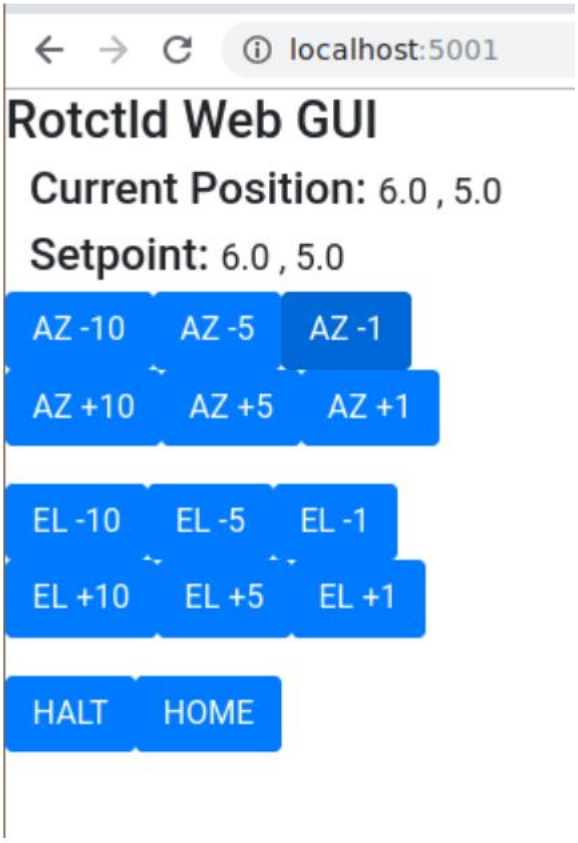

Figura 2-36 Captura RotctlWebGUI [46]

Rotctl-web-gui (Asgeir Bjorgan) [47]

Esta aplicación Flask es una bifurcación del trabajo de Mark Jessop que se desarrolló con la intención de añadir soporte para múltiples rotores y adaptarla para su uso en la estación radio del club LA1K. Se trata de una versión mejorada de interfaz gráfica de usuario (GUI) del programa anteriormente expuesto, utiliza Python 3, Flask para el marco web y Flask-SocketIO para la comunicación en tiempo real entre la interfaz web y el servidor. A continuación, se exponen sus funcionalidades, diferencias y mejoras:

Interfaz Web para Control de Rotor: Proporciona una interfaz web amigable que permite a los usuarios controlar rotores de antenas. Admite la configuración del azimut (dirección horizontal) y elevación (ángulo vertical) a través de una GUI web, reflejando los cambios en tiempo real sin necesidad de recargar la página.

Actualizaciones en Tiempo Real: Utiliza Flask-SocketIO para facilitar actualizaciones en tiempo real de la posición y el punto de ajuste del rotor hacia y desde la interfaz web, mejorando la interacción del usuario.

Configurable a través de un Archivo de Configuración: A diferencia de la versión anterior, este script se puede configurar a través de un argumento de archivo de configuración que apunta a un archivo de configuración. Este archivo especifica los detalles (por ejemplo, puerto, nombre, resolución) para cada rotor, permitiendo la gestión de múltiples rotores.

Diferencias Clave y Mejoras:

Soporte para Múltiples Rotores: El script incorpora funciones para manejar múltiples rotores, cada uno potencialmente con diferentes configuraciones (como puerto y resolución). Esto es una mejora significativa sobre la configuración de un solo rotor en la versión inicial.

Análisis del Archivo de Configuración: Introduce el uso de un archivo de configuración para especificar los detalles de cada rotor, incluyendo su puerto *Rotctld* y un parámetro de resolución opcional que indica el grado de giro de cada rotor.

Interfaz Web Dinámica: La interfaz web se ajusta dinámicamente basada en los rotores definidos en el archivo de configuración. Puede mostrar controles específicos para cada rotor, permitiendo a los usuarios seleccionar y controlar diferentes rotores desde la misma página web.

Establecimiento de acimut y elevación incrementales y absolutas: El script actualizado puede manejar actualizaciones de acimut y elevación tanto incrementales (relativas) como absolutas, proporcionando a los usuarios más flexibilidad en el control de los rotores.

Mejora del Manejo de Errores y Retroalimentación al Usuario: Aunque los métodos centrales para comunicarse con *Rotctld* (por ejemplo, set_azel, get_azel, halt) permanecen similares, el script actualizado probablemente incluye mejores mecanismos de manejo de errores y retroalimentación al usuario, considerando su configuración más compleja y funcionalidad.

Uso de ConfigParser para la Gestión de Configuración: Utiliza el módulo ConfigParser de Python para leer y analizar el archivo de configuración, mostrando un enfoque para externalizar los ajustes de configuración para un mejor mantenimiento.

Mejoras en el Enrutamiento de Flask: El script incluye rutas adicionales de Flask para acomodar la selección de diferentes rotores, apoyando aún más su capacidad multi-rotor.

Este script mejorado se basa en la versión anterior, pero introduce soporte para múltiples rotores, ajustes configurables a través de un archivo externo y una interfaz web más dinámica y amigable para el usuario. Estas mejoras lo hacen una solución más versátil y escalable para el control de rotores de antenas en configuraciones de radioaficionados.

Figura 2-37 Captura RotctlWebGUI BJorgan [47]

3 DESARROLLO DEL TFG

Descripción del apartado

En este apartado se desarrollan los pasos seguidos para la implementación del proyecto. El proyecto está divido en 2 bloques principales, que son el desarrollo del *front-end* de la aplicación y el *back-end*. En primer lugar, se explicará cómo obtener los software necesarios para la realización del proyecto, posteriormente el funcionamiento de *Rotctld*, de obligatoria compresión ya que el *back-end* a desarrollar debe interactuar con dicho programa.

3.1 Definiciones

La aplicación web desarrollada está compuesta, como la mayoría de las aplicaciones web, por dos componentes principales: el *front-end* y el *back-end*. Cada uno de estos componentes juega un papel crítico en el funcionamiento de una aplicación, juntos crean la experiencia de usuario. Para facilitar la compresión de los apartados de este capítulo se profundizará a continuación sobre estos dos conceptos.

3.1.1 Front-end

El *front-end*, también conocido como el lado del cliente, es la parte de la aplicación web con la que los usuarios interactúan directamente. Es todo lo que el usuario experimenta, desde el texto e imágenes hasta botones y deslizadores. El objetivo principal del *front-end* es presentar información de manera accesible y estéticamente agradable y capturar las entradas del usuario para ser procesadas por el *back-end*. En la aplicación desarrollada en este proyecto, el *front-end* está construido usando una combinación de lenguajes ampliamente conocidos:

HTML (Lenguaje de Marcado de Hipertexto): La columna vertebral de cualquier página web, HTML define la estructura y el diseño del contenido, organizándolo en elementos como párrafos, listas, enlaces y otros componentes.

CSS (Hojas de Estilo en Cascada): CSS es responsable del estilo de los elementos HTML. Dicta cómo deben aparecer los elementos del sitio web en términos de colores, fuentes, diseños y transiciones, mejorando el atractivo visual y la experiencia del usuario.

JavaScript: Este lenguaje de programación añade interactividad a las páginas web. JavaScript permite actualizaciones de contenido dinámico, mapas interactivos o gráficos animados haciendo que la aplicación web responda a las acciones del usuario.

Framework Bootstrap: este *framework* proporciona un conjunto integral de componentes de CSS, HTML y JavaScript, y permite diseñar rápidamente interfaces de usuario estéticamente agradables y altamente interactivos. Cuenta con un sistema de rejilla, componentes preconstruidos y *plugins* de

JavaScript que facilitan la creación de diseños consistentes en diferentes dispositivos y navegadores, reduciendo significativamente el tiempo de desarrollo.

3.1.2 Back-end

El *back-end*, o lado del servidor, es la potencia oculta de cualquier aplicación web. El cliente no interactúa directamente con el *back-end*, pero es esencial para la funcionalidad de la aplicación web. Consiste en un servidor, una aplicación y una base de datos. El *back-end* es responsable de almacenar, procesar y gestionar datos, asegurando que todo en el lado del cliente funcione. La lógica del *back-end* se ha escrito en el lenguaje de programación Python, el *framework* de Flask y Flask-SocketIO, elementos sobre los cuales se expandirá en el siguiente apartado. El *back-end* de la aplicación se ocupa de las siguientes funciones:

Almacenamiento y Gestión de Datos: El *back-end* maneja operaciones de base de datos, almacenando datos de posiciones predeterminadas del rotor y asegurando que los datos sean recuperados, actualizados y eliminados según sea necesario.

Lógica del Lado del Servidor: Esto incluye procesar comandos, realizar cálculos, generar resultados para enviar de vuelta al *front-end*, y manejar solicitudes API.

Integración de Interfaz de Programación de Aplicaciones (API): El *back-end* a menudo interactúa con servicios externos a través de APIs, buscando o enviando datos a otras aplicaciones o servicios como proveedores de datos de terceros, en este caso, *rotctld*.

Para el desarrollo del *back-end* se ha utilizado Python. Resultaba una gran opción para desarrollar la aplicación web gracias a su simplicidad y amplia biblioteca de soporte. Además, con la integración de Flask permite una rápida configuración de un servidor web ligero, mientras que Flask-SocketIO facilita la comunicación en tiempo real entre el cliente y el servidor, crucial para controlar dispositivos en tiempo real. Python, al ser compatible con diversas interfaces y sistemas, asegura una integración fluida con *rotctld* para la manipulación de antenas.

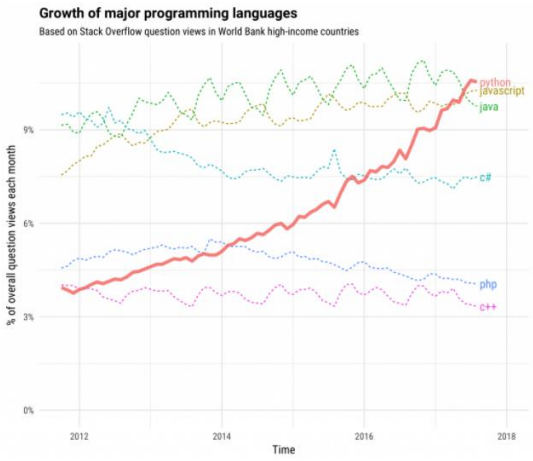

Tabla 3-1 Evolución del uso de Python con el tiempo [48]

3.2 Funcionamiento y Requisitos

3.2.1 Instalación de Visual Studio y Python

Para la realización de este proyecto se ha utilizado el entorno de desarrollo Visual Studio un editor de código fuente ligero pero eficaz que se ejecuta desde el escritorio y está disponible para Windows, macOS y Linux. Se ha utilizado este software debido a su compatibilidad integrada con JavaScript y HTML, y porque cuenta con un amplio ecosistema de extensiones para otros lenguajes, como, por

ejemplo, Python. En este apartado se muestra por pasos el proceso de instalación de este software y la extensión necesaria para poder escribir en Python.

Para Windows:

1. Descargar e Instalar Visual Studio Code desde el sitio web oficial.

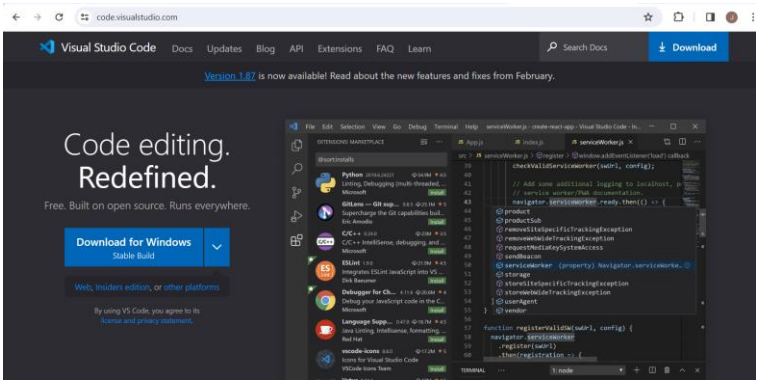

Figura 3-1 Instalación de Visual Studio Code [32]

2. Tras completar la descarga, ejecutar el instalador y seguir las indicaciones de instalación, una vez instalado, iniciar VS Code.
3. Instalar la Extensión de Python para Visual Studio Code. VS Code utiliza extensiones para soportar diferentes lenguajes y tecnologías. Para programar en Python, es necesario instalar la su extensión.
4. Instalar Python. Para ello es necesario ir al sitio web oficial de Python (Python.org) y descargar la última versión.
5. Verificar la Instalación: Abrir un terminal (Símbolo del sistema o PowerShell en Windows, Terminal en macOS y Linux) y escribe python --version. Se debería poder ver la versión de Python instalada.

3.2.2 Instalación de Flask y Flask SocketIO

Para la ejecución del programa es necesario instalar la librería de Flask y Flask-SocketIO de Python. En este apartado se explica cómo es la instalación de dicha librería:

1. Configurar un Entorno Virtual. Un entorno virtual permite gestionar paquetes para un proyecto específico sin afectar el entorno global de Python. Para ello, debemos abrir un terminal o línea de comandos en el directorio del proyecto.
2. En el terminal mencionado, ejecutar el comando "`python3 -m venv venv`" en macOS/Linux o "`python -m venv venv`" en Windows. Este comando crea un entorno virtual llamado venv dentro del directorio del proyecto.
3. A continuación, se debe activar el entorno virtual, para ello:
- En macOS/Linux: `source venv/bin/activate`
- En Windows: `.\venv\Scripts\activate`
4. Instalar Flask con el entorno virtual activado, ahora se puede instalar Flask ejecutando el comando "`pip install Flask`". Este comando instala Flask y sus dependencias.
5. Instalar Flask-SocketIO. Flask-SocketIO mejora las aplicaciones Flask para soportar comunicaciones web en tiempo real entre el cliente y el servidor.
6. Instalar Flask-SocketIO, para ello es necesario ejecutar "`pip install Flask-SocketIO`".

Gracias a la instalación de Flask SocketIO será posible utilizar comunicaciones en tiempo real entre la aplicación web y el lado del servidor.

3.2.3 Funcionamiento de ROTCTLD

Rotctld es parte de la biblioteca Hamlib, que proporciona una interfaz estandarizada para controlar diversos equipos de radioaficionado (ham radio), incluidos antenas, rotores y transceptores. Específicamente, *Rotctld* se centra en el control de rotores de antena, permitiendo al usuario ajustar la dirección en la que su antena apunta a través de comandos de software, en lugar de girar la antena manualmente. A continuación, se expone un hay un desglose de cómo funciona *Rotctld*:

- Funcionalidad Principal

Rotctld se ejecuta como un *daemon* (servicio en segundo plano) en una computadora conectada al hardware del rotor. Esta configuración permite el control remoto del rotor a través de comandos de red.

El programa proporciona una interfaz de red (típicamente TCP/IP) a la que los clientes pueden conectarse para enviar comandos de control al rotor y consultar su estado. Esto significa que las aplicaciones o usuarios pueden controlar rotores desde diferentes dispositivos o a través de internet, sin estar directamente conectados al ordenador donde se ejecuta *rotctld*.

Rotctld utiliza un protocolo de comunicación basado en texto simple. Los clientes envían comandos en un formato estandarizado, y *Rotctld* responde con la información solicitada o el acuse de recibo del comando. Este protocolo incluye comandos para mover el rotor, detener el movimiento y consultar la posición actual.

- Interoperabilidad y Soporte

Rotctld soporta una amplia gama de modelos de rotores de varios fabricantes. El objetivo de Hamlib es abstraer las diferencias entre modelos, proporcionando una interfaz uniforme al usuario.

Gracias a su interfaz de red y comandos estandarizados, *Rotctld* puede integrarse fácilmente con otras aplicaciones, como software de registro, sistemas automáticos de gestión de estaciones o scripts personalizados para control de antenas.

- Casos de Uso Práctico:

En el seguimiento de satélites, *Rotctld* puede usarse para ajustar automáticamente la posición de la antena y seguir satélites a través del cielo.

Rotctld permite ajustar de forma remota la dirección de una antena cuando se opera la estación a través de internet, mejorando la flexibilidad de las operaciones de radioaficionado.

3.2.4 Instalación de Hamlib y Rotctld

Para la instalación de Hamlib hay que seguir un proceso sencillo que varía en función del sistema operativo, en este apartado se muestra una guía de los pasos a seguir en función del sistema operativo usado:

En linux:

1. Actualizar la lista de paquetes para asegurar tener la información más reciente del repositorio. Para ello, se abrirá un terminal y se ejecutará el siguiente comando:

```
sudo apt-get update
```

2. Instalar la biblioteca de hamlib y sus utilidades:

```
sudo apt-get install libhamlib2 libhamlib-utils
```

3. Comprobar la instalación, para ello se ejecutará el comando:

```
Rotctl –l
```

El cual devuelve la lista de rotores soportados por hamlib y *Rotctld*.

En Windows:

La instalación de Hamlib en Windows generalmente se realiza a través de binarios precompilados:

1. Descarga: Visitar la página de lanzamientos de Hamlib en GitHub o un repositorio de confianza para descargar la última versión de Hamlib para Windows.

2. Extraer: Extraer el archivo .zip descargado en un directorio de libre elección. Este directorio contendrá archivos ejecutables para rigctl, rotctl, *Rotctld*, entre otros.

3. Variable de Entorno (Opcional): Para hacer que las herramientas de Hamlib sean accesibles desde cualquier línea de comandos, agregar el directorio donde se extrajo Hamlib a la variable de entorno PATH del sistema.

Inicialización de una sesión de *Rotctld*:

Para iniciar una sesión o inicializar un servidor de *Rotctld* desde un dispositivo se deberá de ejecutar el siguiente comando en un terminal:

```
Rotctld -m <model_number> -t <TCP_port> -r <rotor_device> -T <IP_address>
```

Donde:

<model_number> es el modelo de rotor con el que vamos a interactuar

<TCP_port> es el puerto en el cual *Rotctld* escuchará por posibles conexiones, se debe de usar un puerto que no esté siendo usado por otros servicios, por defecto *Rotctld* escucha en el puerto 4533.

<rotor_device> es el archivo de dispositivo asociado con la interfaz del rotor (por ejemplo, /dev/ttyUSB0 para un rotor conectado por USB en Linux).

<IP_address> es la dirección IP en la cual *Rotctld* escuchará las conexiones. Si no se especifica una IP, *Rotctld* escuchará en todas las interfaces de red disponibles estableciendo la IP: 0.0.0.0

Ejemplo: `Rotctld -m 901 -t 4533 -T 0.0.0.0`

En este ejemplo, *Rotctld* está iniciando una sesión para enviar comandos al rotor 901 (ROT_MODEL_SPID_ROT2PROG) a través del puerto 4533 y mediante todas las interfaces de red (0.0.0.0), la mayoría de las veces no es necesario indicar el archivo de dispositivo asociado.

3.2.5 Comandos

A continuación, se expone una lista los comandos más relevantes de *Rotctld* junto con su sintaxis:

-m	selección del modelo de rotor
-T	este comando seguido de una dirección IP establece la IP a la que estará a la escucha *Rotctld*
-t	establece en qué puerto debe escuchar el programa
-l	muestra la lista de todos los rotores soportados por hamlib y *Rotctld*

-v	Nivel de verbose admitido, muestra el diagnóstico de la sesión actual funcionando, revelando datos como conexiones a la sesión y desde qué puertos
P ‘azimuth’ ‘elevation’	establece la posición al acimut y elevación ordenado. Ejemplo: P 130.00 45.00
p	obtiene la posición actual del rotor devolviendo los valores de acimut y elevación
S	comando de “stop” que detiene de manera inmediata el movimiento del rotor
K	comando que “aparca” el rotor
R	comando que reinicia la posición del rotor

Tabla 3-2 Tabla de comandos de *Rotctld* [AP]

3.2.6 Funcionalidades que Implementar y Requisitos

En este apartado se exponen las funcionalidades necesarias para la integración de *Rotctld* con una aplicación web que sea capaz de interactuar con el daemon incorporando todas sus funcionalidades de forma visual y “responsive”.

1. Introducción de valores absolutos para acimut elevación. *Rotctld* admite órdenes de posicionamiento a un acimut o elevación determinada. La aplicación web debe ser capaz de tomar estos valores y enviarlos al daemon de manera correcta.
2. Introducción de valores de manera incremental. Se debe añadir una funcionalidad mediante la cual la modificación de acimut o elevación sea incremental, es decir, de grado en grado o según el incremento que indique el cliente.
3. Detención del rotor. Se debe incorporar una funcionalidad que permita detener el movimiento del rotor en caso de emergencia enviando el comando ‘S’ al daemon.
4. Reinicio de la posición. Debe haber una función que permita reiniciar la posición del rotor a su estado original mediante el envío del comando ‘R’ al daemon.
5. Posiciones predeterminadas. Para conseguir un sitio web operativo con varias funcionalidades se buscará implementar un apartado con posiciones predeterminadas para el rotor, de manera que, al seleccionarlas, el rotor orientará a dicha posición. La lista de estas posiciones debe ser consistente entre sesiones, es decir, las posiciones se deben almacenar en el lado del servidor para estar siempre a disposición.
6. Teniendo en cuenta el anterior requisito anterior, debe existir la opción de añadir o eliminar posiciones de la lista, por tanto, se debe añadir un apartado con cuadros de texto para añadir nuevas posiciones con el acimut y elevación que introduzca el usuario.
7. Configuración de las conexiones. Puesto que *Rotctld* permite distintas conexiones en función de los puertos a los que estén conectados los rotores, debe haber una funcionalidad que permita añadir nuevos rotores junto con su puerto y que la aplicación logre establecer una conexión con cada uno.
8. Como último requisito se establece la incorporación de un “log” para ver las interacciones con el *daemon* de *Rotctld* en tiempo real y detectar cualquier tipo de error que surja.

Por otro lado, para la creación de la aplicación web y la implementación de todas estas funcionalidades, se fijan los siguientes requisitos:

1. Conocimiento de *Rotctld*: Entender cómo opera *Rotctld*, incluyendo las opciones de línea de comando, protocolo de red, y los comandos que soporta para controlar rotores.

2. Habilidades de Desarrollo *Back-end*: Proficiencia en un lenguaje de programación *back-end* (en este caso Python) para desarrollar una aplicación de servidor que pueda comunicarse con *Rotctld* y servir como intermediario entre este y el *front-end* web.
3. Habilidades de Desarrollo *front-end*: Familiaridad con HTML, CSS y JavaScript para crear la interfaz de usuario. Conocimiento de un *framework* o biblioteca de *front-end* (como React, Angular, o Vue.js) puede ayudar a construir una UI dinámica y responsiva.
4. Web Sockets: Para la comunicación en tiempo real entre la aplicación web y el servidor, es necesario conocimiento sobre WebSockets o AJAX. Esto permite que la aplicación web se actualice en tiempo real sin necesidad de refrescar la página.
5. Diseño de API: Diseñar una API (Interfaz de Programación de Aplicaciones) para el *back-end* que traduzca las solicitudes de la aplicación web en comandos de *Rotctld* y retorne el estado del rotor a la aplicación web.
6. Conocimientos de Despliegue: Familiaridad con el despliegue de aplicaciones web, que incluye elegir un servicio de hosting, configurar un nombre de dominio y entender cómo desplegar los componentes de *front-end* y *back-end*.
7. Base de Datos (Opcional): Debido a que la aplicación requiere almacenar configuraciones de usuario o presets de rotores, será necesario conocimiento sobre tecnologías de bases de datos.
8. Pruebas Multiplataforma: Asegurar que la aplicación web funcione en diferentes navegadores y dispositivos, ajustando el diseño y funcionalidad según sea necesario para compatibilidad y *responsiveness*.

3.3 Diseño

3.3.1 Diseño Conceptual Del front-end

Teniendo en cuenta las funcionalidades que se buscan incorporar en la aplicación web se llegó a un diseño basado en un sistema de rejillas proporcionado por Bootstrap. El diseño preliminar del *front-end* de la aplicación web fue el siguiente:

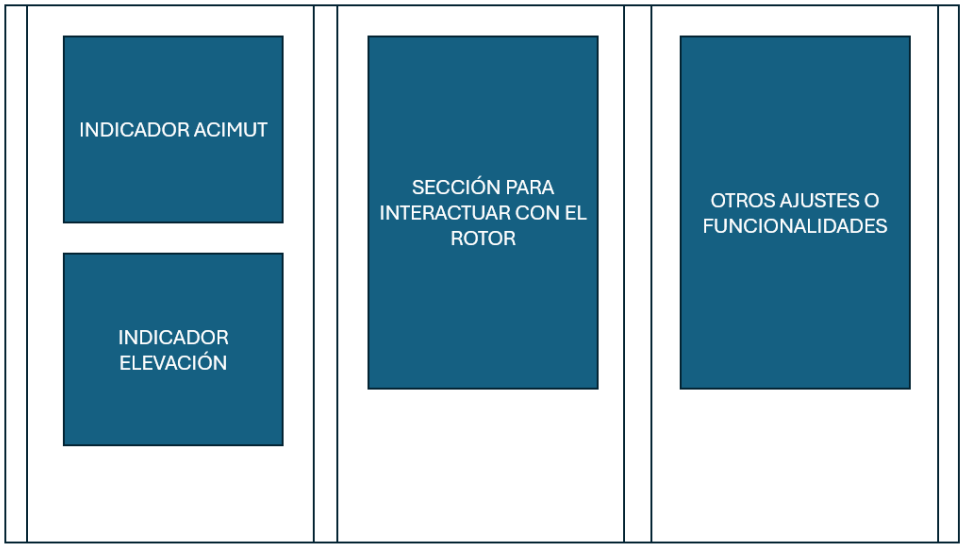

Figura 3-2 Diseño Preliminar de la App Web [AP]

Como se puede observar en la figura, se trata de un diseño basado en 3 columnas. En la primera columna se busca implementar dos indicadores, el superior mostrará la orientación y el segundo la elevación de la antena en tiempo real. En la segunda columna se implantará la sección para controlar los movimientos de la antena. En la parte superior de esta sección se añadirá el apartado para mover la antena a una orientación y elevación específica mientras que en la parte inferior se implementará un apartado para mover la antena de manera incremental. Por último, en la parte superior de la tercera columna se busca implementar un apartado mediante el cual se podrá elegir posiciones predeterminadas para la antena como norte, sur, este u oeste. Se reservará un espacio para la incorporación de otras funcionalidades que vayan surgiendo.

3.3.2 Elementos que incorporar en el front-end

Una vez fijada la estructura general, el siguiente paso es pasar a escribir el código HTML. Este paso resulta sencillo recurriendo al *framework* Bootstrap mencionado anteriormente en esta memoria ya que ofrece un sistema de rejillas preconstruido fácil de incorporar en aplicaciones web. A continuación, se presentan los pasos seguidos para el desarrollo del *front-end* de la aplicación web:

Para poder implementar el *framework* de Bootstrap hay que comprobar que está incluido en el proyecto. Para ello, es necesario agregar Bootstrap incluyendo su enlace CDN en la sección <head> del archivo HTML, esto se hace pegando el siguiente enlace en dicha sección:

```
<linkhref="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css
"rel="stylesheet"integrity="sha384T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dw
wykc2MPK8M2HN"crossorigin="anonymous">
```

Una vez incorporado el *framework*, es posible implementar el sistema de rejillas que proporciona Bootstrap. Para el diseño de la aplicación web a desarrollar, se han utilizado los siguientes elementos principales:

- Contenedores
- Columnas
- Botones
- Cuadros de texto
- Menú desplegable

Contenedor y columnas:

Los contenedores son el elemento de diseño más básico de Bootstrap y se utilizan para contener, rellenar y centrar el contenido dentro de ellos. Para implementar un sistema de rejillas basado en un contenedor formado por 3 columnas se ha utilizado la siguiente estructura proporcionada por Bootstrap:

```
<div class="container text-center">
  <div class="row align-items-start">
    <div class="col">
      One of three columns
    </div>
    <div class="col">
      One of three columns
    </div>
    <div class="col">
      One of three columns
    </div>
  </div>
</div>
```

Figura 3-3 Código para generar rejillas con Bootstrap [41]

Botones

Por otro lado, otro elemento importante incorporado desde Bootstrap por su estilo moderno y minimalista son los botones. El *framework* ofrece varios estilos de botones en función de la acción que realicen. Para incorporar este estilo de botones solo se pega el código ofrecido por Bootstrap en el archivo HTML.

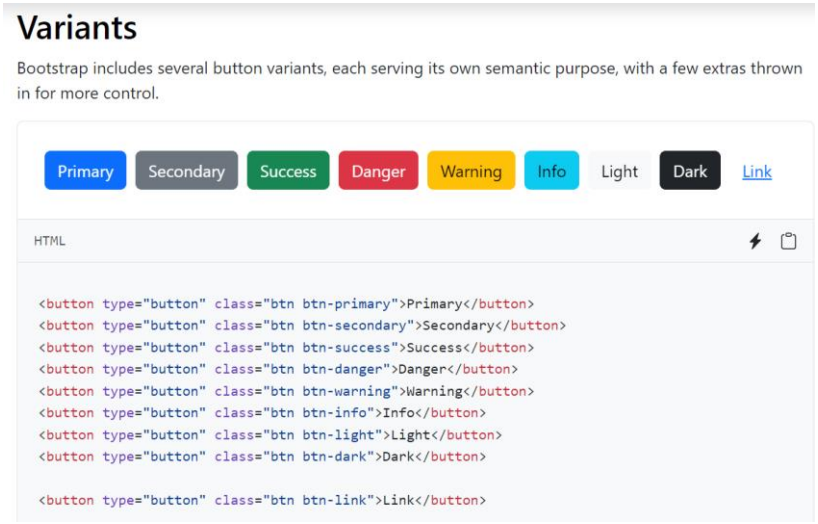

Figura 3-4 Estilos de botones de Bootstrap [41]

En la aplicación web desarrollada se han utilizado los siguientes estilos de botones:

Primary: se ha implementado para mover el rotor de manera incremental tanto en acimut como elevación.

Secondary: se ha utilizado en el botón que añade una posición predeterminada a la lista desplegable de posiciones.

Success: este estilo se utiliza para el botón que envía una posición determinada a la antena.

Danger: se usa en el botón “STOP” que detiene el movimiento del rotor.

Warning: se utiliza en el botón de “PARK” el cual aparca la antena en una posición.

Info: utilizado para el botón de “RESET” el cual reinicia la posición de la antena.

Figura 3-5 Botones Incorporados en la App Web [AP]

Cuadros de texto

Los cuadros de texto en Bootstrap están diseñados para crear formularios interactivos y estilizados que permiten a los usuarios introducir datos. En este caso se han utilizado los siguientes estilos:

Figura 3-6 Formato de Entradas de texto de Bootstrap [49]

Menú Desplegable

Los menús desplegables son superposiciones contextuales conmutables para mostrar listas de enlaces y otros elementos. Se vuelven interactivos con el plugin de JavaScript para desplegables incluido en Bootstrap. Este elemento se ha utilizado para implementar una función que permite seleccionar posiciones predeterminadas y ordenar a la antena moverse a dicha posición.

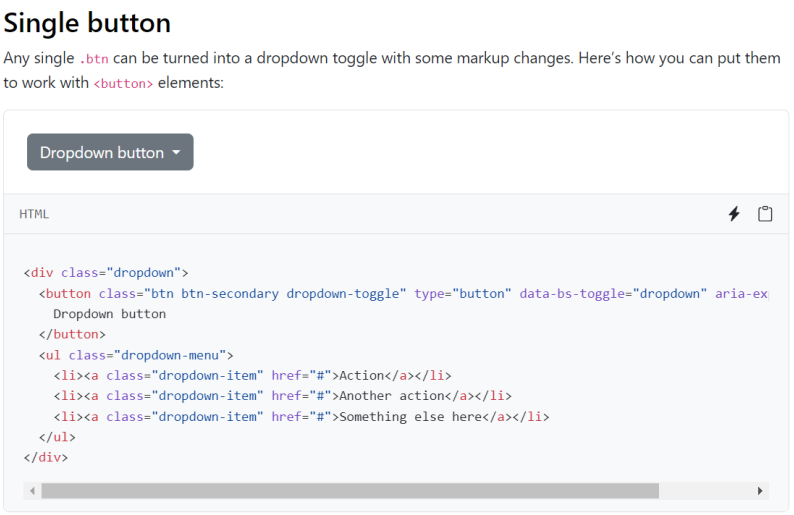

Figura 3-7 Formato y código del menú desplegable de Bootstrap [49]

Indicadores

Para la implementación de indicadores que muestran en tiempo real la orientación y elevación de la antena se ha recurrido a la página canvas-gauges.com. Canvas Gauges ofrece componentes basados en HTML5, minimalistas y de código abierto, diseñados para aplicaciones web. Estos componentes permiten la creación y personalización de varios tipos de medidores, como velocímetros, relojes y otros instrumentos de medición, directamente dentro de las páginas web. El sitio ofrece documentación, una guía para usuarios, documentación de la API para desarrolladores y ejemplos de uso para asistir en la implementación y personalización de estos medidores para desarrolladores y diseñadores web.

Existen dos maneras de implementar los indicadores de esta página en la aplicación web. El primer método es definiendo componentes de medidores en HTML y otra forma es utilizar la API de scripting para inyectar medidores en la página. Para este proyecto se ha utilizado el segundo método, inyectar los medidores mediante la API.

3.4 Interacción de Python con el *Rotctld*

La lógica empleada para algunas funciones del *back-end* de la aplicación web se ha obtenido del repositorio de Github de Bjorgan. El funcionamiento general de la aplicación sigue el siguiente esquema de bloques, posteriormente se explica cada función individualmente:

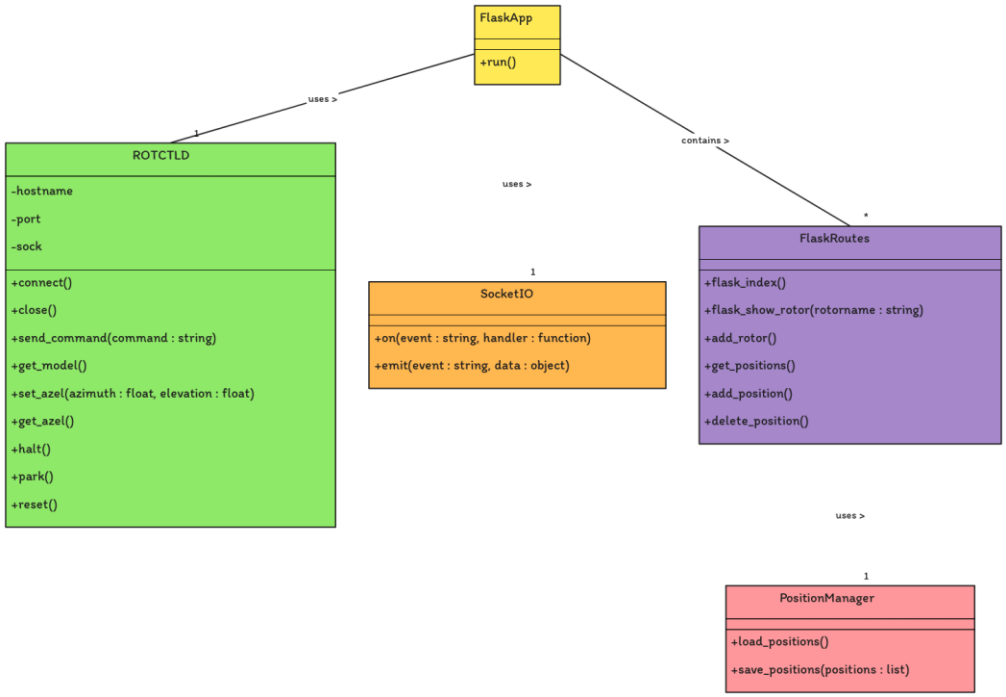

Figura 3-8 Diagrama de Bloques del funcionamiento de la App Web [AP]

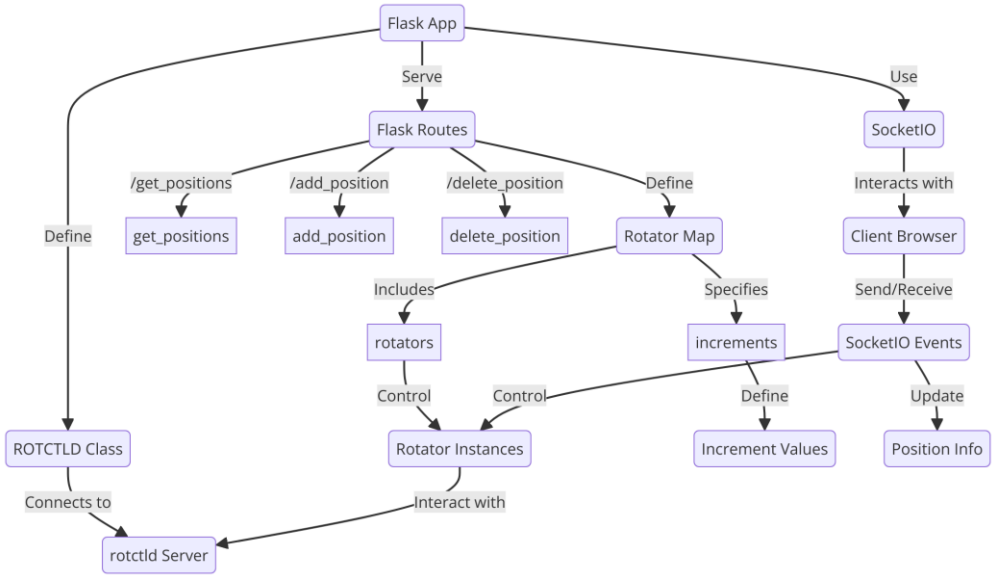

Figura 3-9 Diagrama de Flujo del funcionamiento general de las funciones [AP]

3.4.1 Conexión a Rotctld

El programa establece una conexión con una sesión de *Rotctld* usando la clase *ROTCTLD* personalizada definida dentro del script. Esta clase está diseñada para comunicarse con *Rotctld* a través de sockets TCP, permitiendo que la aplicación web envíe comandos y reciba respuestas de *Rotctld*. El proceso general de conexión es el siguiente:

Inicialización (método __init__):

Cuando se crea una instancia de la clase *ROTCTLD*, el constructor `__init__` inicializa varios atributos:

`self.sock`: Se crea un socket TCP usando `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`. Este socket se utilizará para comunicarse con la instancia de *Rotctld*.

`self.sock.settimeout(timeout)`: El tiempo de espera del socket se establece en el número especificado de segundos (predeterminado a 5 segundos). Este tiempo de espera afecta operaciones en el socket, como conectar y recibir datos, definiendo cuánto tiempo debería esperar el socket para que estas operaciones se completen antes de lanzar una excepción por tiempo de espera.

`self.hostname` y `self.port`: Estos atributos se establecen al nombre de host (dirección IP o nombre de dominio) y número de puerto donde *Rotctld* está en ejecución. Estos valores se pasan al constructor cuando se crea una instancia de *ROTCTLD*.

Conexión (función connect):

La función `connect` es la responsable de establecer una conexión TCP con la instancia de *Rotctld* usando el nombre de host y número de puerto almacenados en `self.hostname` y `self.port`.

Intenta conectar el socket a *Rotctld* llamando a `self.sock.connect((self.hostname, self.port))`. Si la conexión es exitosa, entonces procede a verificar la conexión intentando recuperar el modelo del rotor conectado a *Rotctld*.

Se llama al método `get_model` para enviar un comando a *Rotctld* y recibir su respuesta. Este paso sirve como una prueba simple para asegurar que la conexión es funcional y que *Rotctld* responde como se espera. El método "_" (guión bajo) se utiliza como el comando para recuperar la información del modelo de *Rotctld*.

Manejo de Errores:

Si ocurre algún error durante el intento de conexión (por ejemplo, *Rotctld* no se está ejecutando en el nombre de host y puerto especificados, o hay un problema de red), se captura una excepción `socket.error`.

Al capturar una excepción, se llama al método `close` para asegurar que el socket se cierre correctamente. En este caso la función devuelve `None` para indicar que la conexión no fue exitosa.

Conexión Exitosa:

Si el método `get_model` recupera con éxito la información del modelo de *Rotctld*, indica que la conexión está establecida y funcionando correctamente. El método entonces devuelve la información del modelo, señalando una conexión exitosa.

Cerrando la Conexión (Función close):

La función "`close`" llama a `self.sock.close()` para cerrar el socket TCP. Este método puede ser llamado para cerrar limpiamente la conexión a *Rotctld* cuando ya no sea necesaria o cuando el programa esté limpiando recursos.

```python
def __init__(self, hostname, port=4533, timeout=5):
    """ Open a connection to rotctld, and test it for validity """
    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sock.settimeout(timeout)

    self.hostname = hostname
    self.port = port

Codiumate: Options | Test this method
def connect(self):
    """Attempt to connect to a rotctld instance."""
    try:
        self.sock.connect((self.hostname, self.port))
        model = self.get_model()
        if model is None:
            self.close()
            return None
        return model
    except socket.error as e:
        self.close()
        return None

def close(self):
    self.sock.close()
```

Figura 3-10 Código de establecimiento de conexión [47]

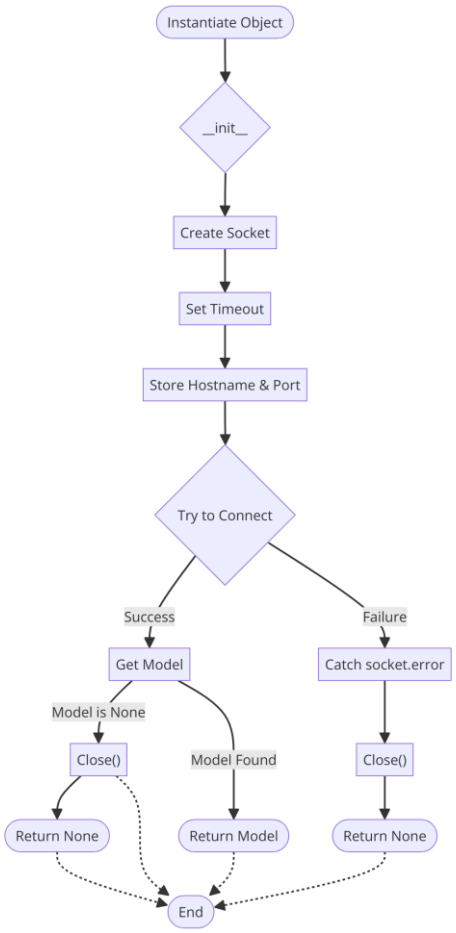

Figura 3-11 Diagrama de Flujo de la función "connect" [AP]

3.4.2 Envío de comandos

Para el envío de comandos a una instancia de *Rotctld* se utiliza la función "`send_command`". Esta función encapsula el proceso de enviar comandos a *Rotctld*, recibir y registrar respuestas, y proporcionar

retroalimentación al usuario a través de actualizaciones de la interfaz web. Es un componente crítico de la lógica de comunicación entre la aplicación web y *Rotctld*. En este apartado se detalla el funcionamiento de esta.

Envío del Comando

Preparar y Enviar el Comando: La función toma una cadena de comando como entrada. A esta cadena se le añade un carácter de nueva línea ('\n') para cumplir con el protocolo de *Rotctld*, que espera que los comandos terminen con una nueva línea. Esta cadena de comando se codifica a bytes usando el método `.encode()` y se envía a la instancia de *Rotctld* usando el método `self.sock.sendall()`. Este método asegura que todos los bytes del comando se envíen a través de la conexión TCP establecida con *Rotctld*.

Recibir y Manejar la Respuesta

Recibir la Respuesta: La función intenta recibir una respuesta de *Rotctld* usando `self.sock.recv(1024)`, que intenta leer hasta 1024 bytes del socket. Esta llamada es bloqueante, lo que significa que esperará a que los datos estén disponibles o hasta que se alcance el tiempo de espera del socket. Los bytes recibidos se decodifican a una cadena usando `.decode()`.

Manejo de Errores: Si ocurre un error mientras se reciben los datos (por ejemplo, problema de conexión, tiempo de espera), se captura una excepción `socket.error`. La función registra el error usando `logging.error` y establece `recv_msg` en una cadena vacía (' '), indicando que no se recibió una respuesta válida.

Registro del Evento

Registrando la Interacción: La función construye una cadena de registro que contiene la marca de tiempo actual, el comando enviado y la respuesta recibida. Esta funcionalidad se ha incorporado para fines de depuración y monitoreo de la interacción con *Rotctld*.

La función intenta emitir esta cadena de registro como un log_event a cualquier cliente web conectado vía SocketIO, proporcionando retroalimentación en tiempo real sobre la ejecución del comando. Si la emisión falla (por ejemplo, debido a que no se llama dentro del contexto de un evento de SocketIO), se captura y se ignora un RuntimeError.

Procesado de la Respuesta

Si se recibe una respuesta, la función verifica si comienza con la cadena 'RPRT', que es un prefijo estándar utilizado por *Rotctld* para indicar un informe de comando. Si está presente, la función divide la respuesta en espacios en blanco para extraer el código de informe, que sigue al prefijo 'RPRT'.

El código de informe (analizado como n en tu código) se convierte a un entero (`num`). Este código se utiliza para determinar el éxito o el fracaso del comando:

Si `num` es negativo, típicamente indica un error, y la función devuelve None.

Si `num` es no negativo, generalmente significa que el comando se ejecutó con éxito, y la función devuelve el código de informe numérico.

Si la respuesta no comienza con 'RPRT' o si no se requiere un manejo específico para el código de informe, se devuelve la cadena de respuesta decodificada completa (`recv_msg`). Esto permite que el código que llama procese la respuesta según sea necesario.

```python
def send_command(self, command):
    """Send a command to the connected rotctld instance,
    and return the return value."""
    self.sock.sendall((command + '\n').encode())
    try:
        recv_msg = self.sock.recv(1024).decode()
    except socket.error as e:
        logging.error(f"Error receiving message: {e}")
        recv_msg = ''

    timestring = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    text = f'{timestring} {command} --> {recv_msg}'
    try:
        emit('log_event', text, namespace='/update_status')
    except RuntimeError:
        pass

    if recv_msg and recv_msg.startswith('RPRT'):
        r, n = recv_msg.split()
        num = int(n)

        if num < 0:
            return None
        else:
            return num

    return recv_msg
```

Figura 3-12 Código de la función send_command [47]

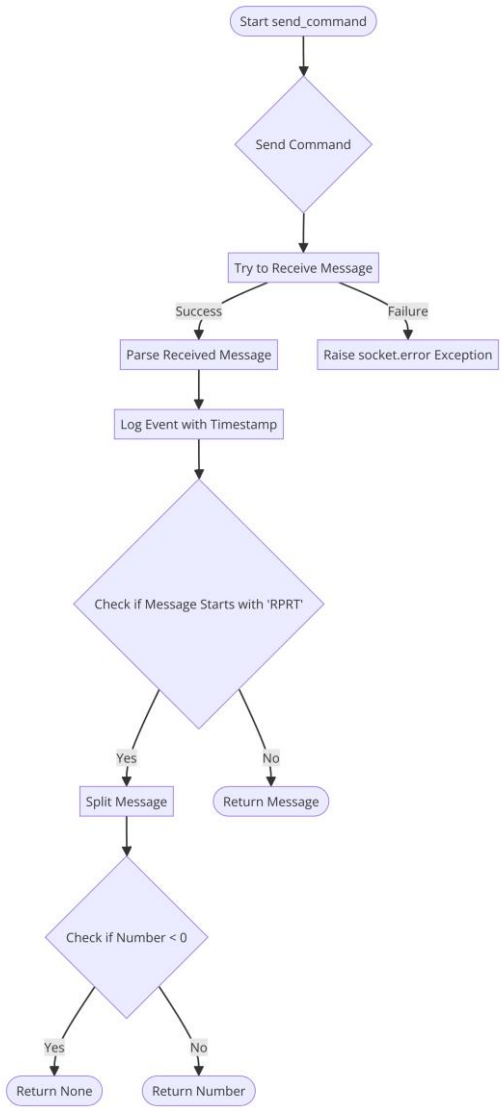

Figura 3-13 Diagrama de Flujo de la función "send_command" [AP]

3.4.3 Establecer una posición

Para establecer una posición de la antena se utiliza la función `set_azel` la cual está diseñada para comandar al rotador moverse a una posición específica de azimut y elevación enviando un comando formateado a *Rotctld* a través de la conexión TCP establecida. Así es como funciona:

Comprobación de una Entrada Correcta

Verificación de Elevación: En primer lugar, verifica el valor de elevación proporcionado para asegurar que esté dentro de un rango plausible. La elevación, por lo general, varía de 0 (horizonte) a 90 grados (directamente sobre la cabeza). Si la elevación proporcionada es mayor a 90 grados, se establece en 90. Si es menos de 0 grados, se corrige a 0. Esto previene el envío de valores de elevación inválidos a *Rotctld*.

Verificación de Azimut: De manera similar, para el azimut, la función asegura que el valor esté dentro del rango de 0 a 360 grados. Si el azimut proporcionado excede los 360 grados, se normaliza calculando el resto de su división por 360.

Formato y Envío del Comando

Preparación del Comando: Usando los valores de azimut y elevación saneados, la función formatea una cadena de comando de acuerdo con el protocolo de *Rotctld*. El comando comienza con P, seguido por los valores de azimut y elevación formateados a un decimal. Esta cadena se construye para cumplir con el formato que *Rotctld* espera.

Enviando el Comando: Esta cadena de comando se pasa luego al método `send_command` de la clase *ROTCTLD*. Esta función es la responsable del envío del comando a *Rotctld* a través de la conexión TCP, esperando una respuesta e interpretando la respuesta para determinar el éxito o fracaso del comando.

Manejo e Interpretación de la Respuesta

Interpretando la Respuesta: El método `send_command` devuelve una respuesta de *Rotctld*. Para comandos de establecimiento de posición como establecer azimut y elevación, *Rotctld* devuelve un código de informe (RPRT) seguido por un número. Un código de informe de 0 típicamente indica éxito, mientras que cualquier número negativo indica un error como se ha explicado en el apartado anterior.

Valor de Retorno: La función `set_azel` verifica el código de informe. Si es 0, indicando que el comando para establecer azimut y elevación ha sido exitoso, la función devuelve `True`. Si la respuesta indica fracaso (cualquier otro valor, típicamente negativo), o si el método `send_command` no devuelve 0, la función devuelve `False`.

```python
def set_azel(self, azimuth, elevation):
    """ Command rotator to a particular azimuth/elevation """
    # Sanity check inputs.
    if elevation > 90.0:
        elevation = 90.0
    elif elevation < 0.0:
        elevation = 0.0

    if azimuth > 360.0:
        azimuth = azimuth % 360.0

    command = "P %3.1f %2.1f" % (azimuth, elevation)
    response = self.send_command(command)
    # Since send_command returns an int for success/failure, compare directly
    if response == 0:  # Assuming 0 indicates success
        return True
    else:
        return False
```

Figura 3-14 Código de la función set_azel [56]

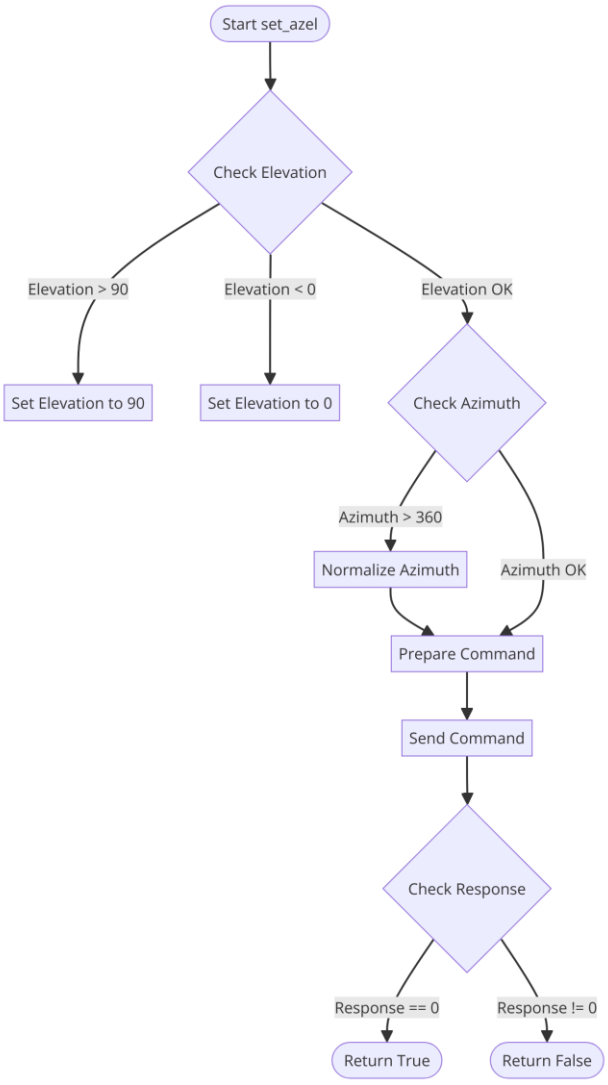

Figura 3-15 Diagrama de Flujo de la función "set_azel" [AP]

3.4.4 Obtención de Posición

Para monitorear y controlar la posición de la antena, se utiliza la función "`get_azel`" la cual recupera los valores actuales de azimut (az) y elevación (el) del rotor controlado por *Rotctld*. A continuación, se explica cómo funciona la función:

Envío del Comando

La función comienza enviando un comando a *Rotctld* para obtener la posición actual del rotador. Utiliza el comando `'p'`, que es un comando estándar en el protocolo de *Rotctld* para preguntar por los valores de azimut y elevación actuales.

Esto se hace llamando al método `self.send_command('p')`, que maneja la comunicación con *Rotctld*.

Manejando la Respuesta

El método `send_command` devuelve la respuesta de *Rotctld* como una cadena. Se espera que esta respuesta contenga los valores actuales de azimut y elevación, separados por caracteres de nueva línea.

La función primero verifica si la respuesta es `None` o una cadena vacía (' '). Si alguna de estas condiciones es verdadera, lanza un `ValueError` indicando que no se recibió respuesta, lo que sugiere un error de comunicación o protocolo.

Analizando la Respuesta

Asumiendo que se recibe una respuesta válida, la función procede a analizar los valores de azimut y elevación de ella:

La respuesta se limpia de espacios en blanco iniciales y finales usando `response.strip()`. Este paso sirve para asegurar que cualquier espacio en blanco extraño no afecte el proceso de análisis.

Luego divide la respuesta limpia en líneas usando split('\n'). En una respuesta exitosa de *Rotctld*, debería haber dos líneas: la primera línea contiene el valor de azimut, y la segunda línea contiene el valor de elevación.

La función verifica si la respuesta dividida tiene al menos dos líneas. Si no es así, lanza un `ValueError` indicando un formato de respuesta inesperado.

Conversión y Devolución de Valores

Si el formato de respuesta es correcto, la función convierte las cadenas que representan azimut y elevación a números de punto flotante usando `float()`. Esta conversión es necesaria porque los valores se reciben inicialmente como cadenas en la respuesta.

Los valores de azimut y elevación convertidos se devuelven entonces como una tupla (az, el). Esto permite al código acceder fácilmente a ambos valores para su posterior procesamiento o visualización.

Manejo de Errores

La función está diseñada para verificar la validez de la respuesta de *Rotctld* y el formato de los datos recibidos. Utiliza excepciones (ValueError) para señalar problemas como respuestas vacías, formatos inesperados o problemas en el análisis de los valores. Esto asegura que cualquier problema en la comunicación o la integridad de los datos se identifique rápidamente y pueda ser manejado adecuadamente por el código.

```python
def get_azel(self):
    """Poll rotctld for azimuth and elevation"""
    response = self.send_command('p')
    if response is None or response == '':
        raise ValueError("Received empty response from send_command.")
    try:
        lines = response.strip().split('\n')
        if len(lines) >= 2:
            az = float(lines[0])
            el = float(lines[1])
            return (az, el)
        else:
            raise ValueError(f"Unexpected response format: {response}")
    except ValueError as e:
        raise ValueError(f"Could not parse position from response: {response}. Error: {e}")
```

Figura 3-16 Código de la función "get_azel" [56]

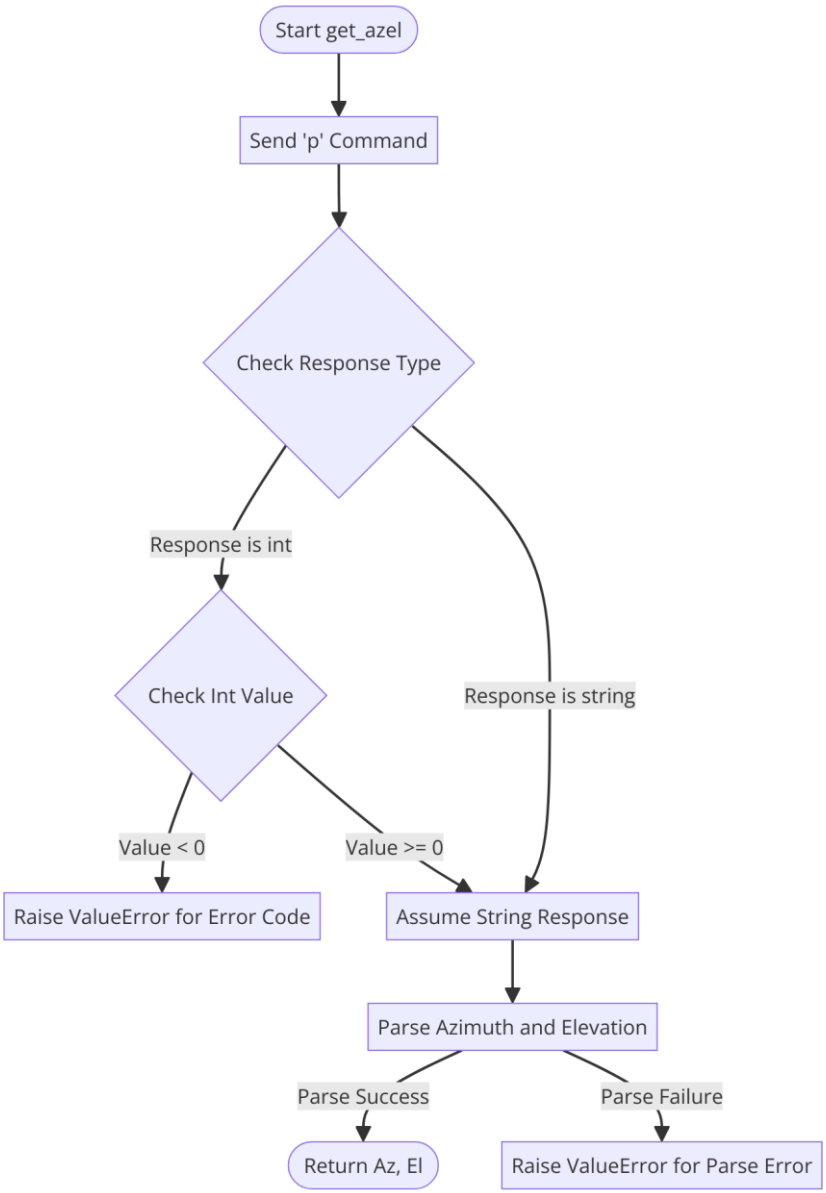

Figura 3-17 Diagrama de Flujo de la función "get_azel" [AP]

3.4.5 Detención del Movimiento

La función *`halt`* se encarga de detener inmediatamente el movimiento de un rotor si este está en movimiento. Consiste en enviar un comando para detener el movimiento si es necesario. El proceso es el siguiente:

Ejecución del Comando: Llama directamente al método `send_command` con el parámetro `'S'`. Entre los comandos de *Rotctld*, `'S'` representa "stop" (detener) o "halt" (parar). Este comando se envía al servidor *Rotctld*, instruyéndolo a detener inmediatamente cualquier movimiento en curso del rotador.

Comunicación con *Rotctld*: El método `send_command`, en el que se basa halt, maneja el proceso de formatear el comando, enviarlo a través de la conexión TCP a *Rotctld* y esperar una respuesta. Este método asegura que el comando `'S'` se transmita correctamente a *Rotctld*.

Manejo de la Respuesta de *Rotctld*: La función `halt` en sí no procesa la respuesta de *Rotctld* directamente, el método `send_command` que llama recibirá e interpretará cualquier respuesta. Esta respuesta puede incluir un código de estado o mensaje que indica si la operación de parada fue exitosa o no.

Manejo de Errores y Retroalimentación: Cualquier error o problema encontrado al enviar el comando o recibir la respuesta se maneja mediante el método `send_command`. Suponiendo que el comando sea recibido y entendido exitosamente por *Rotctld*, y el rotador conectado admita la detención, el movimiento del rotador debería cesar inmediatamente tras la ejecución de este comando.

```
def halt(self):
    """ Immediately halt rotator movement, if it support it """
    self.send_command('S')
```

Figura 3-18 Código de la función “halt” [56]

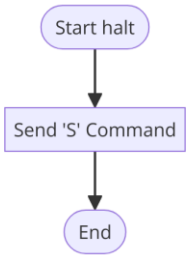

Figura 3-19 Diagrama de Flujo de la función “halt” [AP]

Las funciones “`park`” y “`reset`” funcionan de manera similar a “`halt`”, simplemente el comando enviado a traves de la función “`send_command`” es K para “park” y R para “`reset`”.

3.4.6 Función Limit

La función `limit` en el programa está diseñada para restringir un número dado (`num`) dentro de un rango específico definido por un valor mínimo (`minimum`) y máximo (`maximum`). Esta función asegura que el valor devuelto no exceda los límites definidos. Funciona de la siguiente manera:

`min(num, maximum)`: Esta parte de la función compara el `num` de entrada con el valor `maximum` permitido. Si `num` es mayor que `maximum`, esta expresión se evalúa a `maximum`, limitando efectivamente `num` al límite superior. Si `num` es menor o igual que `maximum`, simplemente devuelve `num`.

`max(resultado_de_min, minimum)`: El resultado del primer paso se compara entonces con el valor `minimum` permitido. Esto garantiza que el valor no esté por debajo del límite inferior definido. Si el resultado del paso 1 es menor que `minimum`, esta expresión se evalúa a `minimum`. Si es mayor o igual a `minimum`, devuelve el resultado del primer paso.

Valor de Devuelto: El resultado final es un valor que está garantizado estar dentro del rango [`minimum`, `maximum`]. Si el `num` de entrada ya está dentro de este rango, permanece sin cambios. Si está fuera del rango, se ajusta al valor límite más cercano (ya sea `minimum` o `maximum`).

Esta función resulta útil porque los valores de entrada deben mantenerse en un rango específico para evitar errores al establecer ángulos para un rotor donde la elevación debe caer en sus respectivos límites operacionales.

```
def limit(num, minimum, maximum):
        """Limits input 'num' between minimum and maximum values."""
        return max(min(num, maximum), minimum)
```

Figura 3-20 Código de la función 'limit' [56]

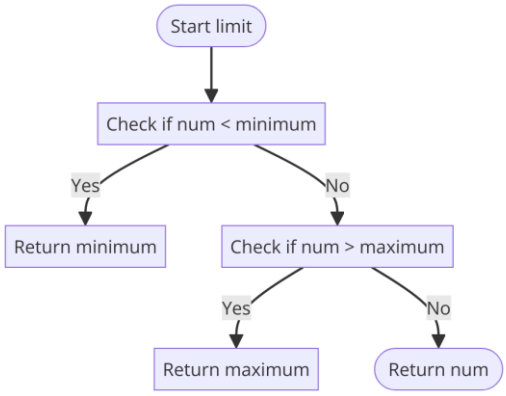

Figura 3-21 Diagrama de Flujo de la función "limit" [AP]

3.5 Rutas de Flask

Las rutas de Flask son un concepto fundamental en el marco web de Flask, se han utilizado para definir las URLs que implementa la aplicación y la lógica que debe ejecutarse cuando esas URLs son accedidas por un cliente. Las rutas actúan como el mapeo entre las solicitudes HTTP a funciones Python (a menudo referidas como funciones de vista) en la aplicación.

Cuando se define una ruta en Flask, se le está diciendo a la aplicación Flask qué función debe manejar las solicitudes para una URL específica. Esto se hace usando el decorador `@app.route()` proporcionado por Flask, que se aplica a una función Python en el código.

3.5.1 Función "Flask_Index"

La función flask_index es un "handler" de ruta en la aplicación web Flask, específicamente para la ruta URL raíz ("/"), es responsable de servir la página principal de la aplicación, ajustando dinámicamente su contenido basándose en los rotadores disponibles y el estado de la sesión actual. A continuación, se explica su funcionamiento:

Decoración de Ruta: El decorador `@app.route("/")` le indica a Flask que esta función debe ser llamada cuando se realiza una solicitud web a la URL raíz de aplicación, en este caso 127.0.0.1:5001.

La función `flask_index()` sirve la página índice principal de la aplicación web. Genera dinámicamente contenido para la página índice determinando qué rotador está actualmente seleccionado y el valor de incremento, entre otras cosas.

Primero crea una lista de nombres de rotadores convirtiendo las claves del diccionario `rotators` en una lista. El diccionario `rotators` contiene los rotadores conectados, identificados por sus nombres.

Luego determina el nombre del rotador actual a ser mostrado en la página. Lo hace intentando recuperar el `current_rotor_name` de la sesión del usuario. Si no lo encuentra, por defecto elige el primer nombre del rotador en la lista (si hay alguno disponible) o un mensaje de reserva "No hay rotor conectado" si no hay rotadores configurados.

Por otro lado, busca el valor de incremento para el nombre del rotador elegido en el diccionario `increments`. Si el nombre del rotador elegido no se encuentra en el diccionario, por defecto usa `DEFAULT_INCREMENT`, que es 1. El valor de incremento se usa para controlar cuánto cambia el azimut o la elevación del rotador con cada comando.

Por último, se realiza el renderizando la plantilla, la función renderiza la plantilla `index.html`, pasando el nombre del rotador elegido, su incremento y la lista de todos los nombres de los rotadores. Para esto, se utiliza la función `flask.render_template`, lo que permite que el contenido HTML se genere dinámicamente basado en las variables proporcionadas, permitiendo que el *front-end* muestre el estado actual y las opciones relacionadas con los rotadores.

En resumen, la función `flask_index` es responsable de servir la página principal de tu aplicación, ajustando dinámicamente su contenido basado en los rotadores disponibles y el estado de la sesión actual.

```python
@app.route("/")
def flask_index():
    """Render main index page"""
    rotator_names = list(rotators)
    chosen_rotor_name = session.get('current_rotor_name', rotator_names[0]) if rotator_names else 'No rotor connected'
    rotor_increment = increments[chosen_rotor_name] if chosen_rotor_name in increments else DEFAULT_INCREMENT
    return flask.render_template(
        "index.html",
        chosen_rotor_name=chosen_rotor_name,
        rotor_increment=rotor_increment,
        rotator_names=rotator_names,
    )
```

Figura 3-22 Código de la función flask_index [56]

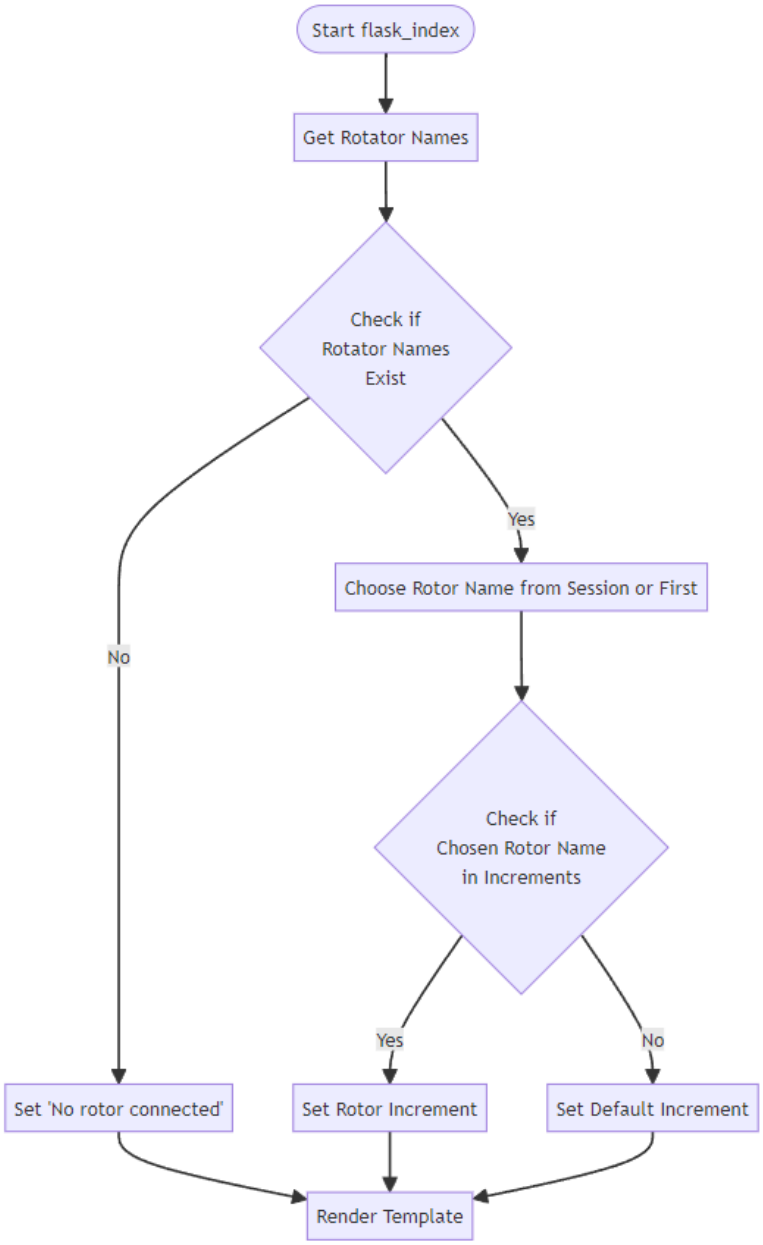

Figura 3-23 Diagrama de Flujo de la función "flask_index" [AP]

3.5.2 Función "Flask_Show_Rotor"

Esta función es responsable de mostrar en la página web un rotor específico identificado por `rotorname`. La función genera y sirve dinámicamente la página web para controlar o ver el estado del rotor específico, basado en el nombre de este, especificado en la URL. Aquí se desglosa su funcionalidad:

El decorador `@app.route("/<rotorname>")` especifica que esta función debe ser llamada cuando se accede a una URL que coincide con el patrón /<`rotorname`>, donde <`rotorname`> es un marcador de posición para el nombre del rotor. Esto permite que la función maneje solicitudes para rotores específicos de manera dinámica.

Para la obtención de la lista de rotores, `rotator_names = list(rotators)` crea una lista de todos los nombres de rotores actualmente registrados en la aplicación. Esta lista se deriva de las claves

del diccionario `rotators`, que contiene objetos o datos relacionados con cada rotor. Este diccionario se rellena al iniciar una instancia de *Rotctld* nueva.

En primer lugar, la función verifica si el `rotorname` solicitado existe en la lista de rotores. Si no es así (if rotorname not in rotator_names:), la función devuelve una página de error 404 usando `flask.render_template("404.html").` Este paso asegura que el cliente solo pueda solicitar nombres de rotores válidos.

Es importante obtener el valor del incremento del rotor, por tanto, si el rotor existe, recupera el valor de incremento asociado a este `(rotor_increment = increments[rotorname]).` Este valor se utiliza para controlar cuánto cambia la posición del rotor con cada comando de movimiento incremental.

Finalmente, la función devuelve una página HTML renderizada `("index.html")` usando la función `render_template` de Flask. La página renderizada se personaliza con el `rotorname` solicitado, su rotor_increment asociado y la lista de todos los `rotator_names`. Esto permite que la página web muestre información específica al rotor solicitado y proporcione una interfaz de usuario para interactuar con él.

```python
@app.route("/<rotorname>")
def flask_show_rotor(rotorname):
    rotator_names = list(rotators)
    if rotorname not in rotator_names:
        return flask.render_template("404.html")

    rotor_increment = increments[rotorname]

    return flask.render_template(
        "index.html",
        chosen_rotor_name=rotorname,
        rotor_increment=rotor_increment,
        rotator_names=rotator_names,
    )
```

Figura 3-24 Código de la función flask_show_rotor [56]

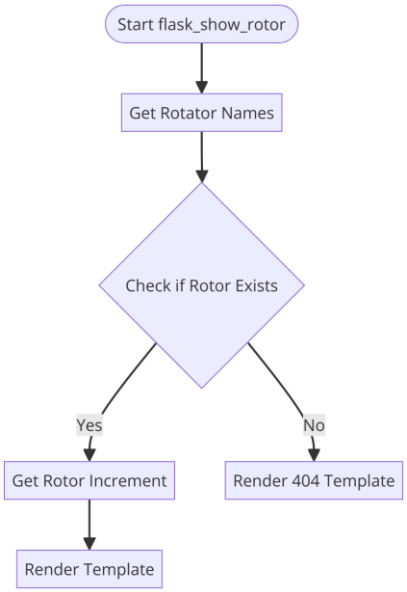

Figura 3-25 Diagrama de Flujo de la función “flask_show_rotor” [AP]

3.5.3 Función add_rotor

La función `add_rotor()` está diseñada para manejar solicitudes POST al endpoint /add_rotor de la aplicación. Esta función facilita la configuración dinámica de nuevos rotores permitiendo a los clientes o administradores añadirlos a través del interfaz web, sin necesidad de editar manualmente archivos de configuración. Aquí se explica a el funcionamiento:

La función comienza extrayendo datos JSON enviados en la solicitud POST a través de data = request.json. Estos datos contienen información sobre el rotor, específicamente su nombre y número de puerto y el incremento.

Luego, la función intenta abrir el archivo `rotors.conf` en modo de append ("a"), lo que permite añadir datos al final del archivo sin modificar su contenido existente. Si el archivo no existe, será creado.

Dentro del archivo, escribe la nueva configuración del rotor. Esto consiste en un encabezado de sección con el nombre del rotor `([{name}]\n)` seguido por una línea que especifica el puerto de control del rotor `(Rotctld_port={port}\n\n).` Los caracteres de nueva línea aseguran que cada entrada esté adecuadamente separada y formateada para su legibilidad y análisis.

Si la operación de actualización del archivo se realiza sin lanzar una excepción, la función devuelve una respuesta JSON con un mensaje indicando la adición exitosa de la configuración `({"message": "Rotor configuration added successfully"}).`

Si ocurre una excepción durante la operación del archivo (por ejemplo, debido a problemas de permisos o errores del sistema de archivos), la excepción se captura y el mensaje de error se imprime en la consola (print(e)). La función luego devuelve una respuesta JSON con un mensaje de error `({"message": "Failed to update configuration file"})` y un código de estado HTTP de 500, indicando un error del servidor.

```python
@app.route("/add_rotor", methods=["POST"])
def add_rotor():
    data = request.json
    name = data["name"]
    port = data["port"]

    # Update the rotors.conf file
    try:
        with open("rotors.conf", "a") as file:
            file.write(f"[{name}]\n")
            file.write(f"rotctld_port={port}\n\n")
        return jsonify({"message": "Rotor configuration added successfully"})
    except Exception as e:
        print(e)
        return jsonify({"message": "Failed to update configuration file"}), 500
```

Figura 3-26 Código de la función add_rotor [AP]

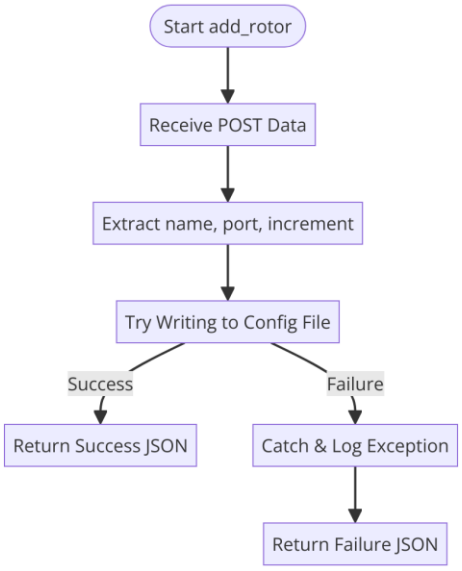

Figura 3-27 Diagrama de Flujo de la función “add_rotor” [AP]

3.6 SocketIO Handlers

Los manejadores de Socket.IO (handlers) en la aplicación actúan como oyentes de eventos y respondedores para la comunicación en tiempo real y bidireccional entre el cliente (navegador web) y el servidor. Esta comunicación se establece a través de una conexión WebSocket, que permite la transferencia instantánea de datos sin necesidad de refrescar la página. Socket.IO abstrae las complejidades de las conexiones WebSocket, proporcionando una API simple para la comunicación basada en eventos. En la aplicación desarrollada, los handlers de Socket.IO se utilizan para gestionar interacciones en tiempo real relacionadas con las posiciones de los rotores y los comandos de control.

3.6.1 Función client_connected

La función `client_connected` es un manejador de eventos de Socket.IO diseñado para gestionar las acciones que ocurren cuando un cliente establece una conexión con el servidor bajo el espacio de nombres `/update_status`. A continuación, un desglose detallado de su funcionalidad:

Esta función utiliza el espacio de nombres "/`update_status`". Los espacios de nombres en Socket.IO permiten segregar las comunicaciones a canales específicos, facilitando la organización de la lógica para diferentes partes de una aplicación.

Por otro lado, la función escucha el evento `client_connected`. Este evento debe ser emitido desde el lado del cliente cuando se conecta exitosamente al servidor usando Socket.IO.

Al conectarse un cliente, la función actualiza la sesión de Flask con el nombre del rotor actual. Esto se hace configurando `session['current_rotor_name']` al valor de data["rotator_key"], donde data es el payload JSON recibido con el evento. `Rotator_key` es un identificador único para el rotor con el que el cliente conectado desea interactuar.

Para la lectura de posición, la función llama a `read_position(data)`, una función que lee la posición actual del rotor especificado por `data["rotator_key"]`.

Finalmente, emite un `setpoint_event` de vuelta al cliente. Este evento lleva el setpoint actual para el rotor especificado, que se obtiene del diccionario `current_setpoints` usando

`data["rotator_key"]` como clave. La función emit envía estos datos de vuelta al cliente, actualizando la interfaz de usuario del cliente en consecuencia.

```python
@socketio.on("client_connected", namespace="/update_status")
def client_connected(data):
    session['current_rotor_name'] = data["rotator_key"]
    # display current position
    read_position(data)
    # and setpoint
    emit("setpoint_event", current_setpoints[data["rotator_key"]])
```

Figura 3-28 Código de la función client_connected [56]

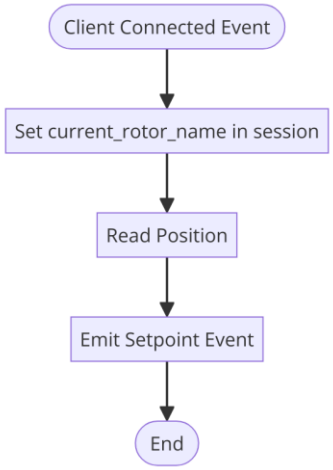

Figura 3-29 Diagrama de Flujo de la función "client_connected" [AP]

3.6.2 Función Update_Setpoint

La función `update_setpoint` es un "handler" que actualiza la posición, o "setpoint", de un rotador basándose en los datos recibidos del navegador web. Procesa los datos entrantes para ajustar los ángulos de azimut y elevación del rotador a un nuevo setpoint y luego ordena al rotador moverse a estos ángulos. El proceso se detalla a continuación:

En primer lugar, se extrae el `rotator_key` de los datos entrantes para identificar qué rotador se está controlando. La variable "motor" se utiliza para determinar si el ajuste es para el azimut o la elevación.

Si los datos entrantes incluyen un valor delta, el setpoint se ajustará en relación con la posición actual. Si es así, ajusta el setpoint correspondiente (azimut o elevación) añadiendo el valor delta, que es el incremento del rotor, al valor de la posición actual. Si no hay un valor delta, intenta establecer el setpoint a un valor específico proporcionado en el campo data["val"]. Si data["val"] no se puede convertir a un flotante (lo que lanzaría un `ValueError`), la función sale sin hacer ningún cambio.

El setpoint de azimut se ajusta para asegurar que permanezca dentro del rango de 0-360 grados, utilizando la operación de módulo. El setpoint de elevación se limita entre 0 y 180 grados para asegurar que permanezca dentro de un rango realista para el movimiento del rotador.

Posteriormente, la función envía los nuevos setpoints de azimut y elevación al rotador llamando a `rotators[rotator].set_azel(az, el)`, donde `rotators[rotator]` es una instancia de la clase *ROTCTLD* correspondiente al rotador especificado.

Finalmente, emite un mensaje `setpoint_event`, junto con el setpoint actualizado, para actualizar la visualización del cliente. Esto asegura que la interfaz de usuario refleje la nueva posición objetivo hacia la cual se está comandando mover al rotador.

```python
@socketio.on("update_setpoint", namespace="/update_status")
def update_setpoint(data):
    rotator = data["rotator_key"]
    motor = data["motor"]

    setpoint = current_setpoints[rotator]
    position = current_positions[rotator]

    # set new setpoints
    if "delta" in data:
        setpoint[motor] = position[motor] + data["delta"]
    else:
        try:
            setpoint[motor] = float(data["val"])
        except ValueError:
            # bogus input, just ignore it and quit
            return

    # limit azi and ele to 0-360 and 0-180 for setpoint display purposes,
    # though rotctld will take care of this automatically
    az = setpoint["azimuth"] = setpoint["azimuth"] % 360
    el = setpoint["elevation"] = limit(setpoint["elevation"], 0, 180.0)

    # set rotctld to current setpoint
    rotators[rotator].set_azel(az, el)

    # update client display
    emit("setpoint_event", setpoint, namespace="/update_status")
```

Figura 3-30 Código de la función update_setpoint [56]

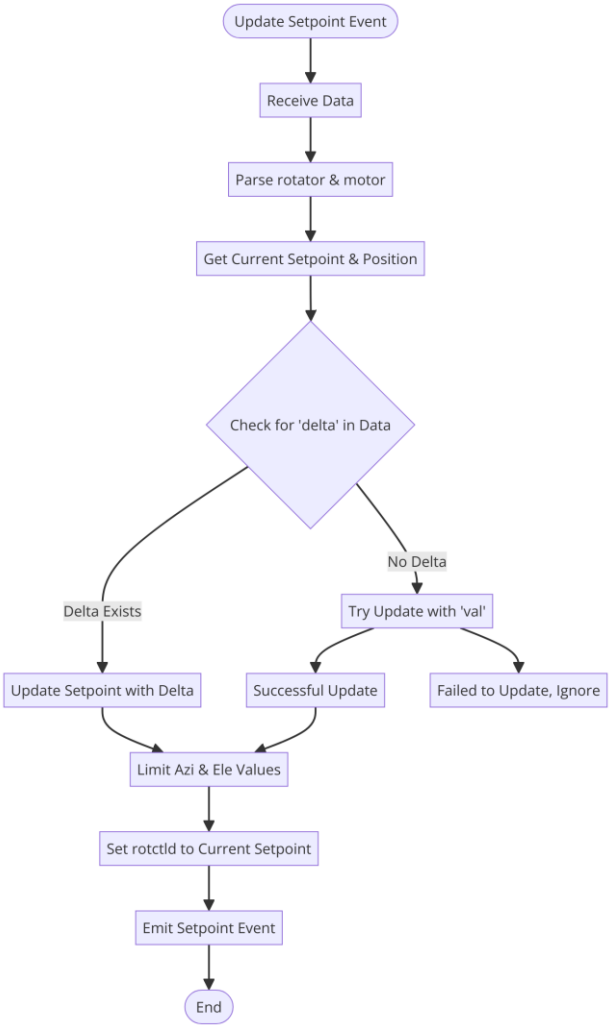

Figura 3-31 Diagrama de Flujo de la función “update_setpoint” [AP]

3.6.3 Función get_position

La función `read_position` es una parte clave del sistema de control y monitoreo en tiempo real del rotor, permite ver la posición actual del rotor a través de la interfaz web. La aplicación utiliza Flask para el servidor web y Flask-SocketIO para la comunicación en tiempo real entre el cliente. Aquí se expone lo que hace la función paso a paso:

Primero, se activa por un evento "`get_position`" enviado desde el lado del cliente a través de la conexión WebSocket (establecida por Flask-SocketIO). Los datos recibidos incluyen una clave ("rotator_key") que identifica de qué rotador se está solicitando la posición.

Posteriormente, recupera el azimut actual y la elevación del rotador especificado llamando al método `get_azel()`. Este método se comunica con el servicio *Rotctld* para obtener la posición actual del rotador.

La función verifica si el valor del azimut es `None`, lo que podría indicar un problema al obtener los datos de posición. Si el azimut es `None`, la función se cierra sin hacer nada más.

Si se reciben datos válidos de azimut y elevación, la función actualiza el diccionario `current_positions` para el rotador especificado, esto se hace mediante el llamamiento a `position.update({"azimuth": az, "elevation": el})`, lo cual modifica el diccionario para reflejar la posición actual del rotor.

Finalmente, la función emite un mensaje `position_event` de vuelta al cliente a través de la conexión WebSocket, que incluye los datos de posición actualizados, permitiendo que el lado cliente de la aplicación actualice la interfaz de usuario con la posición actual del rotador.

```python
@socketio.on("get_position", namespace="/update_status")
def read_position(data):
    rotator = data["rotator_key"]
    position = current_positions[rotator]

    (az, el) = rotators[rotator].get_azel()
    if az == None:
        return
    else:
        # ensure we are updating the dict and not replacing it!
        position.update({"azimuth": az, "elevation": el})

        # display current position
        emit("position_event", position)
```

Figura 3-32 Código de la función ‘get_position’ [56]

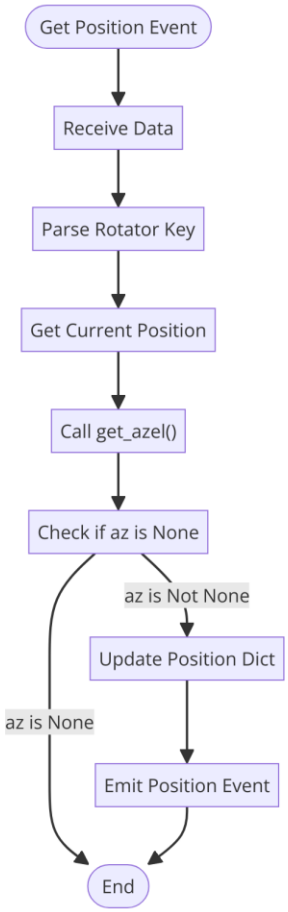

Figura 3-33 Diagrama de Flujo de la función "get_position" [AP]

3.6.4 Función halt, park y reset

La función `halt_rotator` es un "handler" que responde a las solicitudes de los clientes para detener inmediatamente el movimiento de un rotador. A continuación, se detalla su funcionamiento:

Escucha eventos de `halt_rotator` en la vía `/update_status`. Cuando un cliente emite un evento de `halt_rotator` y especifica un rotator_key, esta función se activa. La función extrae el `rotator_key` de los datos enviados con el evento.

Recupera la instancia *ROTCTLD* apropiada del diccionario de `rotators` usando el `rotator_key`. La función luego llama al método `halt()` en esta instancia. El método `halt()` envía comando "S" al servidor *Rotctld* para detener inmediatamente cualquier movimiento en curso del rotador.

Las funciones "`park`" y "`reset`" funcionan de la misma manera que "`halt`", la unica variación es el comando que se envía. En la función "`park`" se envía el comando 'K' y en el caso de la función "`reset`" se envía el comando 'R'.

```python
@socketio.on("halt_rotator", namespace="/update_status")
def halt_rotator(data):
    rotator = data["rotator_key"]
    rotators[rotator].halt()


Codiumate: Options | Test this function
@socketio.on("park_rotator", namespace="/update_status")
def park_rotator(data):
    rotator = data["rotator_key"]
    rotators[rotator].park()


Codiumate: Options | Test this function
@socketio.on("reset_rotator", namespace="/update_status")
def reset_rotator(data):
    rotator = data["rotator_key"]
    rotators[rotator].reset()
```

Figura 3-34 Código de las funciones halt, park y reset [AP]

3.6.5 Programa Principal

El programa principal está diseñado para ser el punto de entrada cuando se ejecuta el script. Configura el servidor web e inicializa conexiones a las sesiones de *Rotctld*. En este capítulo se desglosa en sus componentes clave:

En primer lugar, utiliza la biblioteca `argparse` para analizar los argumentos de la línea de comandos. Esto permite al usuario especificar el puerto en el que el servidor web Flask debería escuchar (`--listen_port`) y la ruta al archivo de configuración (`--config-file`) que contiene información sobre los rotadores (con valor predeterminado "rotors.conf").

Después utiliza el módulo `ConfigParser` para leer y analizar el archivo de configuración. Este archivo contiene configuraciones para cada rotador, incluyendo su nombre de host y puerto (`Rotctld_host` y `Rotctld_port`), y un incremento de paso (`increment`) para los movimientos.

Posteriormente, reitera sobre las secciones del archivo de configuración (cada una correspondiente a un rotador diferente). Para cada rotador, hace lo siguiente:

1. Recupera el nombre de host, puerto y configuración de incremento del archivo de configuración, aplicando valores predeterminados donde sea necesario.
2. Crea una instancia de la clase *ROTCTLD* para gestionar la comunicación con el servidor *Rotctld* para cada rotador. Intenta conectarse al servidor *Rotctld* usando el nombre de host y puerto.
3. Obtiene el azimut y elevación actuales para cada rotador e inicializa los diccionarios `current_setpoints` y `current_positions` con estos valores. Estos diccionarios rastrean la posición deseada (setpoint) y la posición actual de cada rotador.

A continuación,llama a `socketio.run()` para iniciar el servidor web Flask con integración de Socket.IO. Este servidor aloja la interfaz web para interactuar con los rotadores. El servidor escucha en la dirección IP 0.0.0.0 (haciéndolo accesible desde otras máquinas) y en el puerto especificado por el argumento `--listen_port`. La opción `allow_unsafe_werkzeug=True` se utiliza con fines de desarrollo para habilitar características como el depurador interactivo y la recarga automática.

Después de que el servidor web se detiene (típicamente presionando CTRL-C), itera sobre todos los rotadores y llama al método `close()` en cada instancia de *ROTCTLD* para cerrar la conexión a los servidores *Rotctld*.

```python
if __name__ == "__main__":
    import argparse
    from configparser import ConfigParser

    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-l",
        "--listen_port",
        default=5001,
        help="Port to run Web Server on. (Default: 5001)",
    )
    parser.add_argument(
        "--config-file",
        default="rotors.conf",
        help="Path to config file specifying rotor ports and names.",
    )
    args = parser.parse_args()

    # Parse config file.
    default_hostname = "localhost"
    config = ConfigParser()
    config.read(args.config_file)  # Changed to directly use args.config_file

    # Connect to rotctld instances specified in config file.
    for rotor in config.sections():
        hostname = config.get(rotor, "rotctld_host", fallback=default_hostname)
        port = int(config.get(rotor, "rotctld_port"))
        name = rotor

        # add rotor resolution to rotor increments
        DEFAULT_INCREMENT = 1
        resolution = float(config.get(rotor, "increment", fallback=DEFAULT_INCREMENT))
        increments[name] = resolution

        # connect to rotctld
        rotator = ROTCTLD(hostname=hostname, port=port)
        rotator.connect()
        rotators[name] = rotator

        (az, el) = rotators[name].get_azel()
        current_setpoints[name] = {"azimuth": az, "elevation": el}
        current_positions[name] = current_setpoints[name]

    # Run the Flask app, which will block until CTRL-C'd.
    socketio.run(app, host="0.0.0.0", port=args.listen_port, allow_unsafe_werkzeug=True)

    # Close the rotator connection.
    for rotator in rotators.values():
        rotator.close()
```

Figura 3-35 Código del programa principal [56]

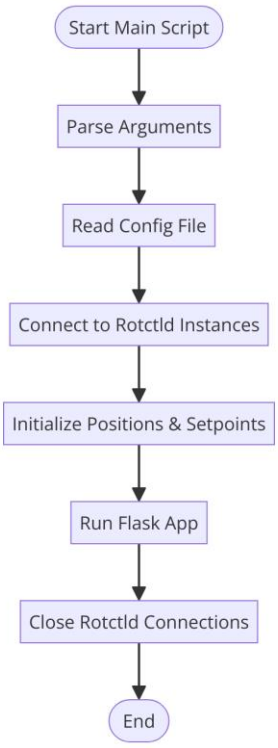

Figura 3-36 Diagrama de Flujo del programa principal [AP]

3.7 JavaScript

Javascript en la aplicación web se ha utilizado para complementar la lógica de los botones utilizados en la página principal.

3.7.1 Handler connect (JavaScript)

Esta parte de la aplicación web establece una conexión WebSocket al servidor (*back-end*) usando Socket.IO, la biblioteca utilizada para conseguir comunicación en tiempo real, bidireccional y basada en eventos entre clientes web y servidores. En este apartado se desglosa lo que está sucediendo:

Primero define el "namespace" `/update_status` para organizar la comunicación del socket. Al usar un espacio de nombres específico, se asegura de que la comunicación relacionada con la actualización del estado de la aplicación esté aislada de otras comunicaciones potenciales con el servidor.

Para la conexión al servidor Socket.IO se utiliza la función io.connect. Esta función toma una URL como argumento, que se construye usando partes de la ubicación de la ventana actual:

`location.protocol`: El protocolo utilizado por la página actual, ya sea http: o https:.

`document.domain`: El nombre de dominio del servidor al que conectar. Esto se utiliza para asegurar que la conexión se realice al mismo dominio desde el cual se cargó la página.

`location.port`: El número de puerto del servidor. Junto con el dominio, esto especifica la dirección de red del servidor.

`namespace`: El espacio de nombres definido anteriormente (/update_status). Esto especifica la ruta en el servidor que la conexión Socket.IO debe usar.

La URL construida sigue el formato http[s]://<dominio>:<puerto>/<espacio de nombres>, dirigiendo al cliente a conectarse al espacio de nombres especificado en el servidor que está sirviendo la página web.

Esta conexión permite que el lado del cliente envíe y reciba mensajes en tiempo real hacia y desde el servidor a través de la conexión WebSocket establecida. Los mensajes pueden incluir actualizaciones sobre el estado de la aplicación, como cambios en valores, comandos emitidos por los usuarios o cualquier otro dato en tiempo real que se necesite comunicar entre el cliente y el servidor. Las conexiones WebSocket son full-duplex, lo que significa que los datos pueden fluir en ambas direcciones simultáneamente, permitiendo una comunicación interactiva que puede actualizar la página web en tiempo real sin necesidad de refrescar la página.

```
// Use the 'update_status' namespace for all of our traffic
namespace = "/update_status";

// Connect to the Socket.IO server.
// The connection URL has the following format:
//     http[s]://<domain>:<port>[/<namespace>]
var socket = io.connect(
  location.protocol +
    "//" +
    document.domain +
    ":" +
    location.port +
    namespace
);
```

Figura 3-37 Código del handler 'connect' [56]

3.7.2 Handler "position_event"

Esta función sirve para actualizar la interfaz de usuario en tiempo real con los datos de posición actuales recibidos del servidor. Asegura que el usuario siempre tenga información actualizada sobre la posición del dispositivo controlado sin necesidad de refrescar manualmente la página o solicitar una actualización. Esta función escucha un evento llamado "`position_event`" del servidor a través de la conexión establecida con Socket.IO. Cuando se recibe el evento, ejecuta la función de callback proporcionada con el mensaje (msg) enviado por el servidor como su argumento.

Se espera que el mensaje del servidor (msg) contenga datos sobre el azimut (azimuth) y la elevación (elevation). Estos valores representan la información de posicionamiento actual.

A continuación, los valores de azimut y elevación se convierten a notación de punto fijo con un decimal usando el método `.toFixed(1)`. Esta conversión estandariza el formato de visualización y hace que los valores sean más fáciles de leer. Las cadenas resultantes se almacenan en las variables `_az` y `_el`, respectivamente.

Posteriormente, el método jQuery .html() se utiliza para actualizar el contenido HTML de elementos con IDs `az_pos` y `el_pos` con los valores de azimut (`_az`) y elevación (`_el`) formateados, respectivamente. Esto muestra visualmente los valores de posición actuales al usuario en la página web.

```
socket.on("position_event", function (msg) {
  var _az = msg.azimuth.toFixed(1);
  var _el = msg.elevation.toFixed(1);

  $("#az_pos").html(_az);
  $("#el_pos").html(_el);
  gaugeAz.value = _az;
  gaugeEl.value = _el;
});
```

Figura 3-38 Código del handler 'position_event' [56]

3.7.3 Handler setpoint_event

Esta función es un manejador de eventos que escucha mensajes de "`setpoint_event`" desde la conexión WebSocket establecida a través de Socket.IO. Es responsable de actualizar dinámicamente los puntos de ajuste mostrados de azimut y elevación en la página web cada vez que el servidor envía un mensaje de "`setpoint_event`". Esto permite retroalimentación en tiempo real al usuario sobre las posiciones objetivo para el azimut y la elevación del dispositivo controlado a través de la aplicación web. Cuando se recibe un mensaje de "`setpoint_event`", la función ejecuta las siguientes acciones:

Extrae los valores de azimut y elevación del mensaje recibido (`msg`). Ambos valores se formatean luego para tener un decimal usando el método `.toFixed(1)`. Este formateo asegura que la presentación de estos valores sea consistente y fácil de leer.

La función actualiza el contenido HTML de los siguientes elementos en la página web:

1. El elemento con el ID `az_target_pos` se actualiza para mostrar el valor de azimut formateado (_az).
2. El elemento con el ID `el_target_pos` se actualiza para mostrar el valor de elevación formateado (_el).

```
// Handle a setpoint update.
socket.on("setpoint_event", function (msg) {
  var _az = msg.azimuth.toFixed(1);
  var _el = msg.elevation.toFixed(1);

  $("#az_target_pos").html(_az);
  $("#el_target_pos").html(_el);
});
```

Figura 3-39 Código del handler setpoint_event [56]

3.7.4 Handler log_event

Esta función es otro manejador de eventos que escucha mensajes de "`log_event`" de la conexión WebSocket establecida a través de Socket.IO. La función actualiza el log de registro de la página web en tiempo real, agregando cada nuevo mensaje de registro en la parte superior del área de visualización cada vez que se recibe un mensaje de "`log_event`" a través de la conexión WebSocket. Esto permite a los usuarios monitorear actividades recientes o actualizaciones de estado de manera dinámica sin necesidad de refrescar la página. Cuando se recibe un mensaje de "`log_event`", se sigue el siguiente proceso:

Se actualiza la visualización del registro mostrando el mensaje recibido (`msg`) en el contenido HTML del elemento con el ID log. El mensaje se "antepone", anteponer el mensaje significa que la función añade la nueva entrada de registro al principio (arriba) del contenido del registro, haciendo que las entradas más recientes aparezcan primero.

```
// Handle a log line.
socket.on("log_event", function (msg) {
  $("#log").prepend(msg);
});
```

Figura 3-40 Código del handler log_event [47]

3.7.5 Función update_setpoint

Esta función está diseñada para comunicarse con el servidor a través de la conexión WebSocket establecida, con el objetivo de actualizar el setpoint del dispositivo incrementalmente. Aquí se desglosa su funcionamiento:

Primero se define una función global `updateSetpoint` que toma dos argumentos:

`motor_val`: Especifica el rotor que está siendo actualizado.

`delta_val`: Especifica el incremento (o decremento, si es negativo) por el cual actualizar el punto de ajuste del motor. El incremento es el paso del rotor específico.

A continuación, la función utiliza el método `socket.emit` para enviar un mensaje `update_setpoint` al servidor. El mensaje contiene tres propiedades:

- `delta`: El valor de paso por el cual el acimut debe de ajustarse. Esto se toma del argumento delta_val.
- `motor`: Especifica el motor que debe actualizarse. Esto se determina por el argumento motor_val.
- `rotator_key`: se utiliza un identificador para el dispositivo siendo controlado. Esto asegura que el comando sea dirigido al dispositivo correcto si múltiples dispositivos están siendo gestionados. Se utiliza el valor de `currentRotator`, que es una variable que mantiene el identificador del rotador actualmente seleccionado o activo.

```
//update setpoint with increment
updateSetpoint = function updateSetpoint(motor_val, delta_val) {
  socket.emit("update_setpoint", {
    delta: delta_val,
    motor: motor_val,
    rotator_key: currentRotator,
  });
};
```

Figura 3-41 Código del handler update_setpoint [47]

3.7.6 Función updateSetpointAbs

La función `updateSetpointAbs` se comunica con el servidor a través de una conexión WebSocket usando Socket.IO con el propósito de permitir el control preciso de la antena estableciendo su posición a un valor exacto, al contrario que los ajustes incrementales facilitados por la función `updateSetpoint` previamente descrita. Analizando en profundidad, esto es lo que hace:

En primer lugar, se define una función global `updateSetpointAbs` que acepta dos argumentos:

`motor_val`: Al igual que en la función anterior, esta función indica el rotor que está siendo actualizado.

`val`: Este es el valor absoluto al cual el setpoint del rotor debería ser actualizado.

Una vez más se utiliza el método `socket.emit`, por el cual la función envía un mensaje `update_setpoint` al servidor. La carga útil del mensaje es un objeto que contiene, además de todo lo anterior, el `rotator_key`: que identifica el dispositivo rotor específico que está siendo controlado. Por otro lado, la variable `currentRotator`, guarda el identificador del rotador siendo controlado.

```
//update setpoint with absolute rotor values
updateSetpointAbs = function updateSetpointAbs(motor_val, val) {
  socket.emit("update_setpoint", {
    val: val,
    motor: motor_val,
    rotator_key: currentRotator,
  });
};
```

Figura 3-42 Código del handler ‘updateSetpointAbs [47]

3.7.7 Función connect

Esta función establece un protocolo de comunicación inicial entre el cliente y el servidor a través de una conexión WebSocket, utilizando Socket.IO, su propósito principal es iniciar un diálogo con el servidor, señalando que el cliente ha conectado exitosamente y se está identificando con un dispositivo rotador específico. funciona de la siguiente manera:

La función escucha el evento "`connect`" en el socket. Este evento es disparado automáticamente por Socket.IO cuando el cliente establece con éxito una conexión WebSocket con el servidor. Al activarse el evento "`connect`", se ejecuta la función de callback. Dentro de esta función, el cliente (aplicación web) notifica al servidor que ha conectado exitosamente y está listo para comunicarse.

El cliente utiliza el método `socket.emit` para enviar un mensaje "`client_connected`" al servidor. Este mensaje incluye el objeto `{ rotator_key: currentRotator }` como su carga de datos. `Rotator_key` es un identificador para el dispositivo rotador actualmente seleccionado o activo con el cual el cliente pretende interactuar. Esta información ayuda al servidor a entender qué cliente (y dispositivo rotador asociado) acaba de conectarse y está listo para el intercambio de datos.

```
// Tell the server we are connected and ready for data.
socket.on("connect", function () {
  socket.emit("client_connected", { rotator_key: currentRotator });
});
```

Figura 3-43 Código del socket 'connect' [47]

3.7.8 Función get_position

Este fragmento de código se ha diseñado para solicitar continuamente la posición actual del dispositivo rotador especificado cada segundo, lo que permite actualizaciones dinámicas y en tiempo real. Aquí se detalla de cómo funciona:

El método `window.setInterval` de JavaScript se utiliza para ejecutar repetidamente una función especificada a intervalos de tiempo fijos, en este caso, cada 1000 milisegundos (o 1 segundo). El primer argumento es la función que se ejecutará, y el segundo argumento es el intervalo de tiempo en milisegundos. El primer argumento de `window.setInterval` es una función anónima. Esta función contiene el código que se ejecutará en cada intervalo. Dentro de la función anónima, se utiliza el método `socket.emit`. Este método envía un mensaje del cliente (tu aplicación web) al servidor a través de la conexión WebSocket. El mensaje se denomina "`get_position`", indicando al servidor que el cliente está solicitando la posición actual del rotador. El contenido del mensaje es un objeto que incluye el `rotator_key: { rotator_key: currentRotator }`.

El objetivo de la ejecución repetida de esta función es asegurar que el cliente (aplicación web) tenga información actualizada sobre la posición del rotador. Al hacer esto cada segundo, la aplicación puede proporcionar retroalimentación en tiempo real al usuario sobre la posición del rotador, mejorando la experiencia del usuario al mostrar un estado continuo y actual.

```
  // Request current rotator position every second.
  window.setInterval(function () {
    socket.emit("get_position", { rotator_key: currentRotator });
  }, 1000);
});
```

Figura 3-44 Código del handler get_position [47]

3.7.9 Función addnewPosition

La función `addNewPosition` se centra en la recuperación y validación de entradas. Asegura que los datos ingresados por el usuario para una nueva posición estén completos y correctos antes de proceder con cualquier paso subsiguiente para agregar realmente la posición. A continuación, se desglosa su funcionamiento:

Primero, recupera las entradas proporcionadas por el usuario de los elementos HTML con IDs `newPositionName`, `newPositionAzimuth` y `newPositionElevation`. Para el nombre, también elimina cualquier espacio en blanco al inicio o al final usando `.trim()` para asegurarse de que el nombre esté limpio y libre de espacios accidentales.

Los valores de azimut y elevación, inicialmente recibidos como cadenas, se convierten a números usando el constructor `Number().` Esta conversión es necesaria porque las operaciones aritméticas y las validaciones sobre estos valores requieren que estén en forma numérica. La función luego verifica tres condiciones para asegurar que las entradas son válidas:

1. Verifica si `positionName` no está vacío. Un nombre vacío significaría que el usuario no ha ingresado ningún nombre para la nueva posición, lo cual es requerido.
2. Utiliza `isNaN(positionAzimuth)` para verificar si el valor de azimut no es un número. Esto podría ocurrir si el usuario ingresa caracteres no numéricos o deja el campo vacío.
3. De manera similar, verifica si el valor de la elevación no es un número usando `isNaN(positionElevation)`.

Si cualquiera de estas verificaciones falla (indicando entradas faltantes o inválidas), la función muestra un mensaje de alerta al usuario pidiéndole que llene todos los campos con números válidos para azimut y elevación, y luego termina sin realizar ninguna acción adicional. Esta alerta sirve como retroalimentación inmediata al usuario de que su entrada no fue aceptada debido a falta de información o información incorrecta.

Tras una validación exitosa, la función envía una solicitud POST al servidor en el endpoint `/add_position` utilizando la API Fetch. La solicitud incluye:

- Una carga útil en JSON que contiene el nombre, azimut y elevación de la nueva posición.
- Los encabezados de la solicitud especifican que la carga es en formato JSON.

La función espera una respuesta del servidor. Al recibirla, la misma función convierte la respuesta a JSON y luego procede con la actualización del interfaz basándose en los datos recibidos. Si la operación es exitosa se registra en la consola.

El resultado final es que se crea una nueva opción y se añade al menú desplegable de selección de posiciones, usando el nombre proporcionado de la posición, azimut y elevación. Esto ocurre creando un nuevo objeto Option con el nombre de la posición como texto de visualización y un valor que combina el azimut y la elevación, y luego añadiendo esta opción al elemento select identificado por directionSelector.

Los campos de entrada para el nombre de la posición, azimut y elevación se limpian, preparando el formulario para otra entrada y proporcionando una interacción con el programa fluida.

Si la solicitud fetch encuentra un error (por ejemplo, si el servidor no está accesible o devuelve un estado de error), la función lo captura y lo registra en la consola, ayudando en la depuración e informando al usuario de cualquier problema en el proceso de envío de datos.

```
function addNewPosition() {
  var positionName = document.getElementById("newPositionName").value.trim();
  // Convert input values to numbers
  var positionAzimuth = Number(
    document.getElementById("newPositionAzimuth").value
  );
  var positionElevation = Number(
    document.getElementById("newPositionElevation").value
  );

  if (!positionName || isNaN(positionAzimuth) || isNaN(positionElevation)) {
    alert(
      "Please fill in all fields with valid numbers for azimuth and elevation."
    );
    return;
  }

  fetch("/add_position", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      name: positionName,
      // Ensure azimuth and elevation are sent as numbers
      azimuth: positionAzimuth,
      elevation: positionElevation,
    }),
  })
    .then((response) => response.json())
    .then((data) => {
      console.log(data.message);
      // Directly add the new option to the dropdown with numbers
      var select = document.getElementById("directionSelector");
      var newOption = new Option(
        positionName,
        `${positionAzimuth},${positionElevation}`
      );
      select.appendChild(newOption);

      // Clear the input fields after successful addition
      document.getElementById("newPositionName").value = "";
      document.getElementById("newPositionAzimuth").value = "";
      document.getElementById("newPositionElevation").value = "";
    })
    .catch((error) => console.error("Error:", error));
}
```

Figura 3-45 Código de la función addnewPosition [47]

3.7.10 Función deleteSelectedPosition

El propósito de esta función es eliminar una posición seleccionada del menú desplegable de la interfaz de usuario y para notificar al servidor sobre la eliminación.

En primer lugar, comienza obteniendo el elemento select (`directionSelector`) y recupera la opción actualmente seleccionada usando `select.options[select.selectedIndex]`.

Luego, envía una solicitud POST al servidor en el endpoint `/delete_position` con una carga útil JSON que contiene el nombre de la posición a ser eliminada. Esta acción tiene la intención de notificar al servidor para remover la posición seleccionada del registro.

Tras la eliminación correcta, suprime la opción seleccionada del menú desplegable usando `select.remove(select.selectedIndex)`. Si hay un error (por ejemplo, un problema con el servidor, problema de red), se registra el error en la consola.

```
function deleteSelectedPosition() {
  var select = document.getElementById("directionSelector");
  var selectedOption = select.options[select.selectedIndex];
  var positionName = selectedOption.text;

  // Notify the server to delete the position
  fetch("/delete_position", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ name: positionName }),
  })
    .then((response) => response.json())
    .then((data) => {
      console.log(data.message);
      // Remove the option from the dropdown
      select.remove(select.selectedIndex);
    })
    .catch((error) => console.error("Error:", error));
}
```

Figura 3-46 Código de la función deleteSelectedPosition[47]

3.7.11 Función DOMContentLoaded

Esta función es otro handler, especificamente para el evento DOMContentLoaded, que se dispara cuando el documento HTML ha sido completamente cargado y analizado, sin esperar a que las hojas de estilo, imágenes y subframes terminen de cargar. La función lo que hace es llenar el `dropdown directionSelector` con opciones basadas en datos recuperados del servidor, permitiendo a los usuarios seleccionar de una lista de posiciones predefinidas. El proceso que sigue específicamente es el siguiente:

Escucha el evento `DOMContentLoaded` en el documento. Una vez que este evento se dispara, indicando que el HTML está completamente analizado, la función se ejecuta. A continuación, se envía una solicitud fetch al endpoint /`get_positions` en el servidor. Esta solicitud se hace para recuperar la lista de posiciones predefinidas guardadas.

Si la operación fetch es exitosa y el servidor responde con un estado HTTP indicando éxito (response.ok es verdadero), se procesa la respuesta convirtiéndola de formato JSON a objetos de JavaScript. Si la respuesta no está OK (indicando un fallo), lanza un error, que luego es capturado por el bloque catch.

Tras recuperar exitosamente las posiciones, limpia las opciones existentes en el dropdown directionSelector para asegurarse de que está comenzando desde un estado limpio. Luego, para cada posición recibida del servidor, crea un nuevo objeto Option con el nombre de la posición como texto de visualización y un valor compuesto por los valores de azimut y elevación de la posición, separados por una coma. Cada nueva opción se añade al dropdown, haciendo estas posiciones seleccionables por el usuario.

Por otro lado, registra mensajes en la consola al inicio de la operación para indicar que la página ha cargado y está recuperando posiciones, y nuevamente después de cargar exitosamente y mostrar las posiciones en el dropdown. Si ocurre un error durante este proceso, el error también se registra en la consola.

```
document.addEventListener("DOMContentLoaded", function () {
  console.log("Page loaded. Fetching positions...");
  fetch("/get_positions")
    .then((response) => {
      if (!response.ok) {
        throw new Error("Network response was not ok");
      }
      return response.json();
    })
    .then((positions) => {
      var select = document.getElementById("directionSelector");
      select.innerHTML = ""; // Clear existing options first
      positions.forEach((pos) => {
        var option = new Option(pos.name, pos.azimuth + "," + pos.elevation);
        select.add(option);
      });
      console.log("Positions loaded successfully.");
    })
    .catch((error) => console.error("Error:", error));
});
```

Figura 3-47 Código para la carga de contenido [AP]

3.7.12 Función addRotorConfiguration

Esta función está diseñada para manejar la adición de nuevas configuraciones de rotor a la aplicación. Lleva a cabo varios pasos para asegurar que el usuario pueda agregar una nueva configuración especificando un nombre de rotor, puerto e incremento. Aquí se desglosa su funcionamiento:

La función comienza obteniendo los valores ingresados por el usuario en los campos de entrada para el nombre del rotor (`rotorName`), puerto (`rotorPort`) e incremento (`rotorIncrement`). Recorta estos valores para eliminar cualquier espacio en blanco inicial o final, asegurando que los datos enviados al servidor no incluyan espacios innecesarios. Antes de seguir con la solicitud fetch, verifica si todos los campos requeridos han sido completados por el usuario. Si algún campo está vacío (`!rotorName || !rotorPort || !rotorIncrement`), muestra una alerta al usuario pidiéndole que llene todos los campos. Esta es una forma básica de validación del lado del cliente para asegurar que el servidor reciba datos completos.

Si todos los campos han sido rellenados, se envía una solicitud POST al endpoint `/add_rotor` utilizando la API Fetch. La solicitud incluye un cuerpo JSON que contiene los valores de nombre, puerto e incremento del rotor. Esta es una operación asíncrona que envía los datos de configuración al servidor para ser procesados y potencialmente añadidos a las configuraciones del sistema.

Al recibir una respuesta exitosa del servidor (es decir, el servidor ha podido procesar la solicitud y añadir la nueva configuración de rotor), se alerta al usuario con un mensaje contenido en la respuesta. Este mensaje confirma la adición correcta de la nueva configuración del rotor.

Después de una operación exitosa, limpia los campos de entrada (`rotorName`, `rotorPort`, `rotorIncrement`) para permitir la adición fácil de más configuraciones sin la necesidad de borrar manualmente cada campo.

Si la solicitud falla en algún momento (por ejemplo, error del servidor, problema de red), captura el error y alerta al usuario de que la operación para añadir una configuración de rotor falló. Además, registra el error en la consola para fines de depuración.

```
function addRotorConfiguration() {
  var rotorName = document.getElementById("rotorName").value.trim();
  var rotorPort = document.getElementById("rotorPort").value.trim();
  var rotorIncrement = document.getElementById("rotorIncrement").value.trim(); // New line

  if (!rotorName || !rotorPort || !rotorIncrement) { // Updated condition
    alert("Please fill in all fields.");
    return;
  }

  fetch("/add_rotor", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      name: rotorName,
      port: rotorPort,
      increment: rotorIncrement, // Added increment
    }),
  })
    .then((response) => response.json())
    .then((data) => {
      alert(data.message);
      document.getElementById("rotorName").value = "";
      document.getElementById("rotorPort").value = "";
      document.getElementById("rotorIncrement").value = ""; // Clear the new input field
    })
    .catch((error) => {
      console.error("Error:", error);
      alert("Failed to add rotor configuration.");
    });
}
```

Figura 3-48 Código de la función addRotorConfiguration [AP]

3.8 *Front-end*

Para el desarrollo del *front-end* de la aplicación web se irá columna por columna explicando los contenidos de cada una, así como su desarrollo.

3.8.1 Primera Columna del Interfaz

En esta primera sección se define la primera columna del interfaz de usuario diseñada para ver la posición de la antena. En este capítulo se detallada de cada elemento y su propósito:

Div Wrapper: Actúa como un contenedor para todo el contenido de esta parte de tu aplicación web, asegurando que todos los elementos anidados estén agrupados juntos para propósitos de estilo o de diseño.

Title Box Div: Contiene elementos que definen la sección de encabezado o título de tu aplicación. Este apartado cuenta con dos elementos dentro:

1. Dos etiquetas img con atributos de fuente apuntando a "static/cud.png" y "static/enm.png" respectivamente, probablemente utilizadas para mostrar logotipos o imágenes relevantes para la aplicación en ambos lados del título.
2. Un elemento h1 con la clase "display-6" que muestra el título de la app: "APLICACIÓN DE CONTROL DE ANTENA". La clase probablemente se refiere a un estilo específico definido en tu CSS.

Div Container: Un contenedor genérico que se usa para aplicar un diseño o estilo específicos a sus elementos hijos. Dentro de este contenedor encontramos:

- Un div de clase "column", con el fin de mostrar los contenidos de manera vertical. Esta columna incluye:

- Un encabezado h1 que indica "Indicador de Azimut y Elevación", señalando que esta sección de la aplicación está dedicada a mostrar estos dos valores clave relacionados con el posicionamiento de la antena.
- Un hr (regla horizontal) para separar visualmente el título del contenido debajo.
- Dos grupos de divs organizados en una estructura de tipo "row", cada uno conteniendo dos divisiones de col (columna). Estos se utilizan para mostrar las etiquetas "Azimut:" y "Elevación:", y sus posiciones correspondientes que serán dinámicamente rellenadas con JavaScript con los valores de la orientación y elevación con ids "az_pos" y "el_pos".
- Dos elementos tipo “canvas” con ids "gauge-az" y "gauge-el", los cuales llaman a las representaciones gráficas que son los indicadores o diales, para indicar visualmente el azimut y la elevación actuales de la antena.
- Un div en la parte inferior dentro de la columna, estilizado como "current-rotor-box", contiene dos elementos span. El primero muestra el texto estático "Rotor Conectado: ", y el segundo muestra dinámicamente el nombre del rotor conectado a través del renderizado de plantillas (indicado por {{ chosen_rotor_name }}).

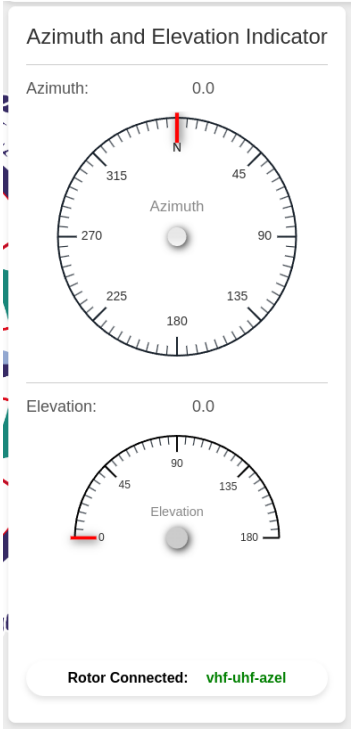

Figura 3-49 Primera columna del interfaz de la App Web [AP]

3.8.2 Indicadores

Esta parte del código inicializa y configura los dos indicadores radiales utilizando la clase `RadialGauge`, pertenecientes a la biblioteca de indicadores de Canvas Gauges. Ambos indicadores sirven como indicadores visuales dinámicos que se actualizan en tiempo real para reflejar cambios en los ángulos de azimut y elevación. Estos indicadores mejoran la interfaz de usuario proporcionando una representación intuitiva e inmediata de la orientación del rotor, facilitando al usuario monitorear y ajustar configuraciones según sea necesario. Aquí hay un desglose de las características de cada indicador:

Indicador de Azimut (gaugeAz)

- Muestra el ángulo de azimut actual (0° a 360°), que representa la dirección de la brújula.
- El objetivo de renderizado es un elemento canvas con ID gauge-az.
- Tiene un rango de 0 a 360 grados, con marcas mayores para cada dirección cardinal e incrementos de 45 grados, más marcas menores.
- El indicador es circular (360°), comenzando desde el norte (ángulo de 180°).
- Los estilos visuales incluyen colores para el plato, las marcas, los números y la aguja. La aguja está diseñada para ser una línea delgada, coloreada de rojo.
- La animación de la aguja es lineal para un movimiento suave.
- Detalles estéticos como configuraciones de borde y sombra están establecidos para el atractivo visual.

Indicador de Elevación (gaugeEl)

- Muestra el ángulo de elevación actual (0° a 180°), indicando cómo de alto está apuntando el dispositivo por encima del horizonte.
- El objetivo de renderizado es un elemento canvas con ID gauge-el.
- Tiene un rango semicircular de 0 a 180 grados, con marcas mayores cada 45 grados.
- Similar al indicador de azimut, tiene elementos visuales estilizados para legibilidad y estética.
- La aguja del indicador de elevación se mueve sobre un semicírculo, reflejando valores de elevación desde el horizonte hasta directamente sobre la cabeza.

```
$(document).ready(function () {
var gaugeAz = new RadialGauge({
  renderTo: "gauge-az",
  title: "Azimuth",
  minValue: 0,
  maxValue: 360,
  majorTicks: ["N", "45", "90", "135", "180", "225", "270", "315", "N"],
  minorTicks: 9,
  ticksAngle: 360,
  startAngle: 180,
  strokeTicks: true,
  highlights: true,
  colorPlate: "#fff",
  colorMajorTicks: "#00000",
  colorMinorTicks: "#17202A",
  colorNumbers: "#333",
  colorNeedle: "rgba(255, 0, 0, 1)",
  colorNeedleEnd: "rgba(255, 0, 0, 1)",
  valueBox: false,
  valueTextShadow: false,
  colorCircleInner: "#fff",
  colorNeedleCircleOuter: "#ccc",
  needleCircleSize: 10,
  needleCircleOuter: false,
  animationRule: "linear",
  needleType: "line",
  needleStart: 75,
  needleEnd: 99,
  needleWidth: 3,
  borders: true,
  borderInnerWidth: 0,
  borderMiddleWidth: 0,
  borderOuterWidth: 0,
  colorBorderOuter: "#ccc",
  colorBorderOuterEnd: "#ccc",
  colorNeedleShadowDown: "#222",
  borderShadowWidth: 0,
  animationDuration: 10,
}).draw();
```

Figura 3-50 Código del indicador de Acimut [AP]

```
var gaugeEl = new RadialGauge({
  renderTo: "gauge-el",
  title: "Elevation",
  minValue: 0,
  maxValue: 180,
  majorTicks: [0, 45, 90, 135, 180],
  minorTicks: 9,
  ticksAngle: 180,
  startAngle: 90,
  strokeTicks: true,
  highlights: false,
  colorPlate: "#fff",
  colorMajorTicks: "#000000",
  colorMinorTicks: "#17202A",
  colorNumbers: "#333",
  colorNeedle: "rgba(255, 0, 0, 1)",
  colorNeedleEnd: "rgba(255, 0, 0, 1)",
  valueBox: false,
  valueTextShadow: false,
  valueInt: 3,
  valueDec: 0,
  colorCircleInner: "#fff",
  colorNeedleCircleOuter: "#ccc",
  needleCircleSize: 10,
  needleCircleInner: false,
  needleCircleOuter: true,
  needleType: "line",
  needleStart: 75,
  needleEnd: 99,
  needleWidth: 3,
  borders: true,
  borderInnerWidth: 0,
  borderMiddleWidth: 0,
  borderOuterWidth: 0,
  colorBorderInner: "#aaa",
  colorBorderOuter: "#ccc",
  colorNeedleShadowDown: "#222",
  borderShadowWidth: 0,
  animationDuration: 10,
}).draw();
```

Figura 3-51 Código del indicador de elevación [AP]

3.8.3 Segunda columna del interfaz

Esta segunda sección proporciona el interfaz para el control del Azimut y la Elevación de la antena, permite también gestionar las configuraciones del rotor y ejecutar comandos inmediatos como detener, aparcar o reiniciar el rotor. Analizando de arriba a abajo:

Establecimiento del Azimut y Elevación: Esta parte permite al usuario establecer manualmente los ángulos de Azimut y Elevación para la antena en valor absoluto, es decir, la orientación que se introduzca es a la que irá la antena, para ello, cuenta con dos campos de entrada donde el usuario puede escribir los ángulos deseados. En la parte derecha están los botones correspondientes de "Establecer" para aplicar los ángulos ingresados. En la parte inferior, encontramos los botones de incremento/decremento para ambos, Azimut y Elevación, permitiendo ajustes finos por incrementos predefinidos (-incremento o +incremento).

Debajo de los campos de entrada, encontramos los botones para rotar la antena en pequeños incrementos:

1. Rotación de Azimut en Sentido Antihorario y Horario: se han usado un par de botones con íconos de flecha izquierda y derecha para que el usuario pueda rotar el ángulo de Azimut en cualquiera de las direcciones según el paso establecido por el incremento.

A la derecha, encontramos botones similares que permiten aumentar o disminuir el ángulo de Elevación.

A continuación, en la parte inferior se han implementado los controles de emergencia: tres botones para acciones inmediatas:

STOP: Para detener cualquier movimiento en curso del rotador.

PARK: Para mover la antena a una posición 'aparcada' predefinida, generalmente para protección.

RESET: Para reiniciar el sistema del rotor.

Finalmente, en la parte inferior de esta columna se ha integrado la configuración del rotor, esta área proporciona apartados de “input” para añadir una nueva configuración de rotor especificando un nombre único de rotor y el puerto de control correspondiente además del incremento de este, si es conocido. Esto permite la adición dinámica de rotores al sistema sin necesidad de cambiar directamente la base de código del software. Un botón etiquetado "Ingresar" activa la adición de la nueva configuración de rotor.

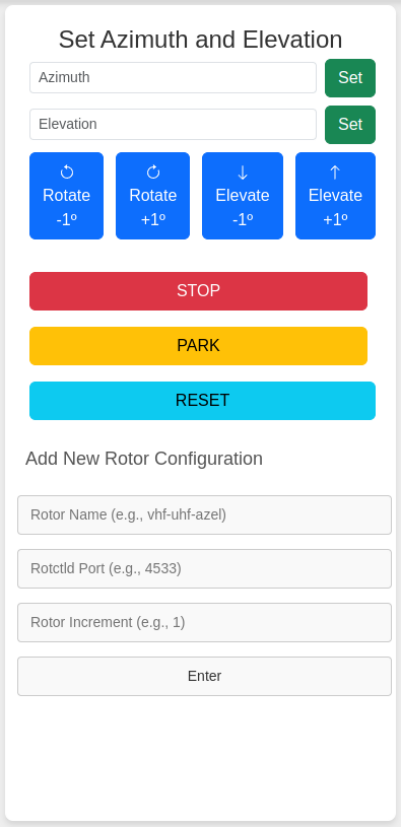

Figura 3-52 Segunda columna del interfaz de la App Web [AP]

3.8.4 Lógica e Interacción de los botones

Cuando un botón en esta sección es presionado, se llaman funciones específicas de JavaScript para interactuar con el lado del servidor, es decir, el programa de Python que gestiona el sistema de posicionamiento de la antena. Cada botón está asociado con una acción única que desencadena un cambio en el estado o configuración de la antena. En este apartado se desglosa el proceso que se sigue al presionar cada botón:

Botones de Establecer para Azimut y Elevación: Al presionar uno de estos botones, se llama a la función updateSetpointAbs con dos argumentos: el motor ('azimut' o 'elevación') y el valor ingresado en el campo de entrada correspondiente. Esta función emite un evento de Socket.IO al servidor con la posición absoluta deseada para el azimut o la elevación, solicitando que la antena se mueva a este nuevo punto.

Botones de Rotar y Elevar (-1º/+1º): Estos botones llaman a la función updateSetpoint con argumentos especificados para la dirección y los valores de incremento. Para el ajuste de acimut, se pasa el argumento 'acimut' junto con -incremento o +incremento dependiendo de si se desea rotar en sentido horario o antihorario. La misma lógica se aplica para el ajuste de elevación con el parámetro del motor 'elevación'. Esta función, al igual que la anterior, emite un evento de Socket.IO al servidor, indicando un ajuste relativo a la posición actual de la antena.

Botones STOP, PARK, RESET: Presionar cualquiera de estos botones invoca sus respectivas funciones (haltRotator, parkRotator, resetRotator). Cada función emite un evento de Socket.IO específico que no requiere parámetros adicionales de la interfaz de usuario. Estos eventos le dicen al servidor que detenga inmediatamente cualquier movimiento en curso mediante haltRotator, que mueva la antena a una posición 'aparcada' segura mediante parkRotator o que reinicie el sistema de la antena para limpiar errores o reinicializar el sistema resetRotator.

Botón de Añadir Nueva Configuración de Rotor: Al presionar este botón se activa la función addRotorConfiguration, que recopila los valores ingresados en los campos de entrada "Nombre del Rotor" y "Puerto *Rotctld*". Luego envía una solicitud POST a un endpoint especificado (/add_rotor), incluyendo el nombre del rotor y el puerto en el cuerpo de la solicitud. Esta solicitud es manejada por el servidor mediante la función anteriormente expuesta “add_rotor” para añadir una nueva configuración de rotor al sistema, permitiéndole controlar rotores adicionales según lo especificado.

3.8.5 Tercera Columna del Interfaz

La tercera sección de la aplicación web está dedicada a gestionar posiciones predeterminadas para un sistema controlador de antena, así como monitorear y limpiar los registros de comandos relacionados con el funcionamiento del daemon. Aquí tienes un desglose de su funcionalidad:

La parte superior de esta columna proporciona una interfaz de usuario para seleccionar y aplicar posiciones de azimut y elevación predefinidas a la antena. Esto se facilita mediante un menú desplegable (elemento select con id="directionSelector") para elegir una posición y un botón "Go" para aplicar la posición seleccionada.

Se ha implementado esta función de manera que el usuario puede añadir nuevas posiciones predeterminadas ingresando el nombre de esta posición, un valor de azimut (0-360 grados) y otro de elevación (0-180 grados) en los campos de texto indicados para ello. Mediante el botón "Add Position" se activa la función addNewPosition() a través de la cual se recopilan los valores de entrada, se validan y se envía una solicitud al servidor para añadir la nueva posición. Si todo este proceso se realiza con éxito, la nueva posición se añade al menú desplegable para su selección futura.

Por otro lado, se ha implementado un botón "Delete Selected Position" que permite al usuario eliminar la posición actualmente seleccionada del menú desplegable. Al clicar en este botón, se activa la función deleteSelectedPosition(), la cual envía una solicitud al servidor para eliminar la posición seleccionada basándose en su nombre. Una vez eliminada, también se elimina del menú desplegable.

Por último, se ha implementado un log (elemento div estilizado con id="commandLog") que muestra información de registro sobre las operaciones o comandos ejecutados por la aplicación. Esto puede incluir acciones tomadas sobre las posiciones de la antena, como añadir o eliminar posiciones, o los resultados de mover la antena a una nueva posición. También se ha implementado un botón para limpiar el contenido del registro de comandos ($('#log').html('')). Esto permite al usuario restablecer fácilmente el área del registro sin tener que borrar el texto manualmente.

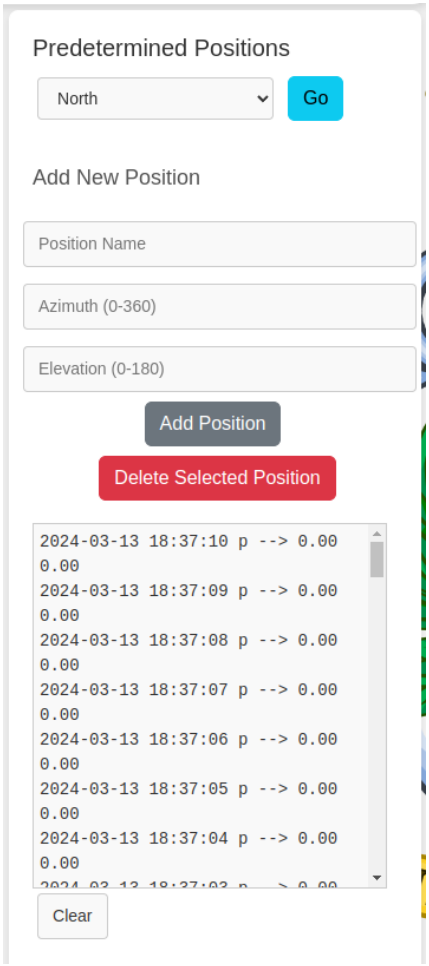

Figura 3-53 Tercera columna del interfaz de la App Web [AP]

4 RESULTADOS / VALIDACIÓN / PRUEBA

En este apartado se procede a exponer el resultado final del programa, así como su proceso de ejecución y funcionamiento general con la antena instalada en el edificio de investigación del CUD.

4.1 Instalación y ejecución

El proceso de instalación del programa desarrollado es simple y se puede hacer siguiendo los siguientes pasos:

1. Descompresión del archivo zip en el que viene, haciendo esto obtendremos una carpeta llamada “Proyecto Mirapamar”.

Figura 4-38 Archivos del programa en formato zip [AP]

2. Abrir la carpeta llamada “Proyecto Mirapamar” en la cual podremos identificar los siguientes 3 archivos: positions.json, proyectomirapamar.py, rotors.conf y las carpetas static y templates.

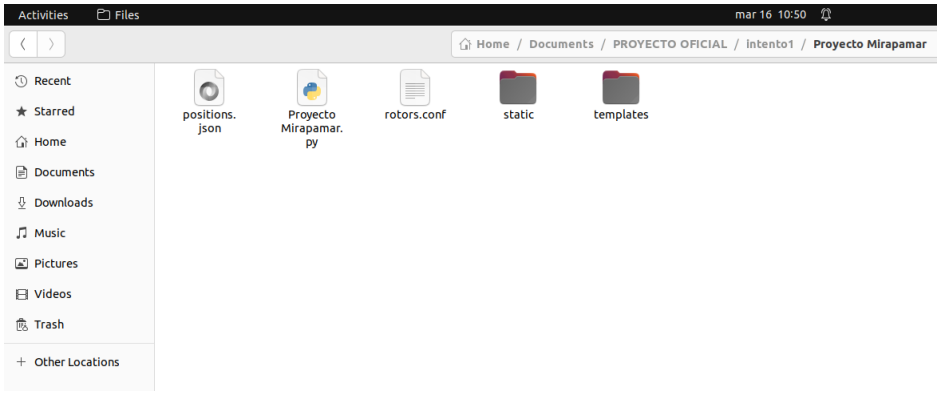

Figura 4-39 Archivos que componen la aplicación web [AP]

3. Para la usar el programa será necesario estar enlazado a la instancia de *Rotctld* que esté siendo ejecutada. Para ello se ejecutará el siguiente comando en el servidor del laboratorio de investigación del CUD: '*Rotctld* -m 2 -vvvv'

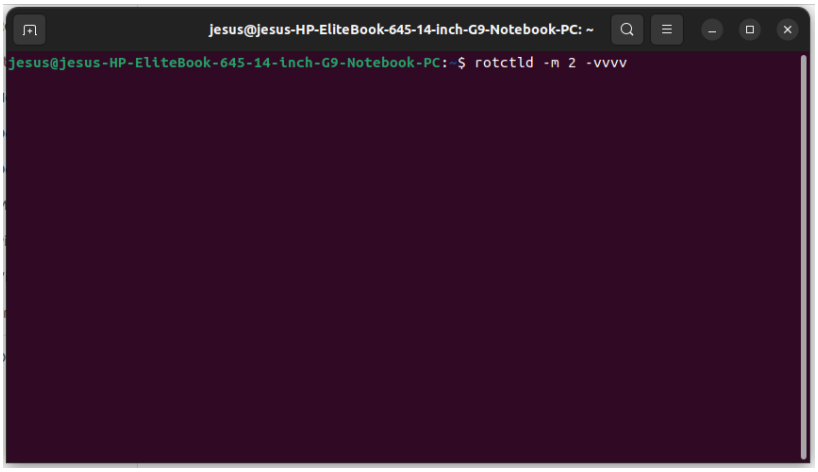

Figura 4-40 Ejecución del Daemon de Rotctld [AP]

4. Una vez arrancada la instancia de *Rotctld* simplemente habrá que hacer click derecho en el archivo llamado "proyectomirapamar.py" y presionar en "ejecutar como un programa".

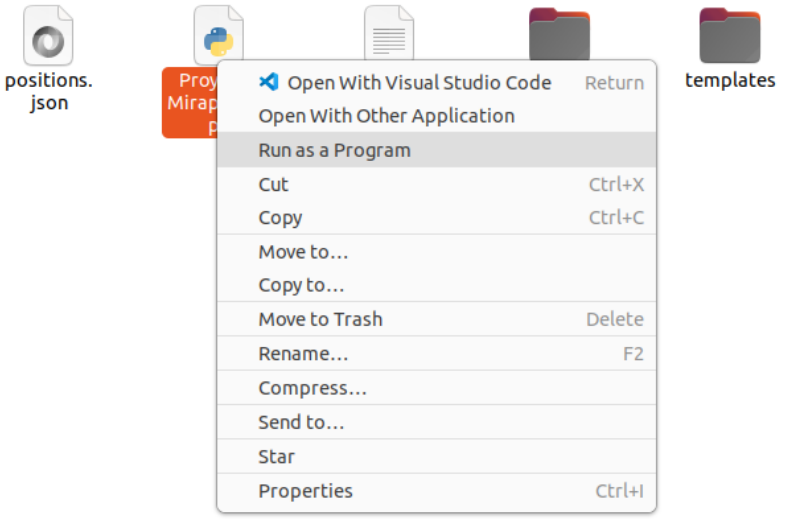

Figura 4-41 Ejecución del archivo ProyectoMirapamar.py [AP]

5. Una vez hecho este se abrirá un terminal con el siguiente formato:

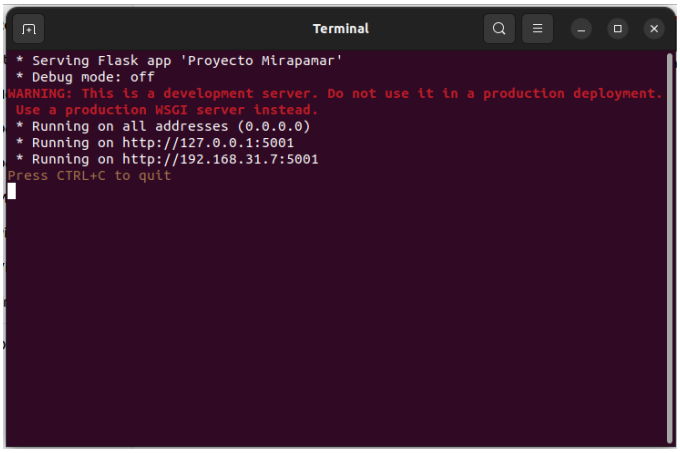

Figura 4-42 Servidor Web en ejecución [AP]

6. Se deberá de hacer click en el primer enlace que aparece si se quiere acceder desde ese terminal o copiar y pegar la dirección inferior en el dispositivo móvil desde el cual se quiera acceder.

Tras seguir estos pasos se abrirá una pestaña en el navegador con la aplicación web en ejecución lista para recibir los comandos por parte del cliente e interactuar con la antena.

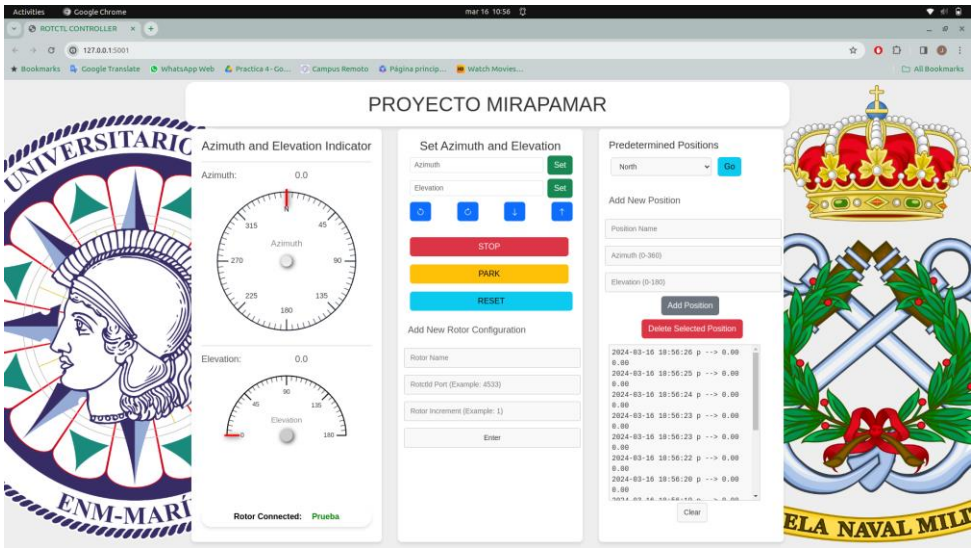

Figura 4-43 Interfaz completo de la aplicación web [AP]

La figura 4-6 es el panel de interacción de la antena a través del cual podremos indicar que se ubique a un acimut especifico, que cambie su posición incrementalmente o que se detenga durante una operación. Por otro lado, podremos elegir una posición predeterminada desde la tercera columna y modificar el nombre de la antena con la que se interactúa, así como el puerto en el que se encuentra y el paso de sus rotores en la parte inferior de la segunda columna.

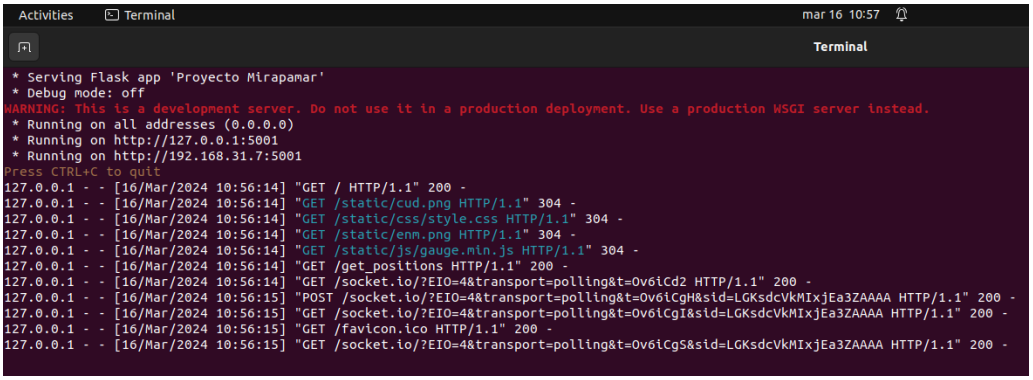

Figura 4-44 Terminal del servidor web [AP]

En la figura 4-7 se tiene la sesión por parte del servidor siendo ejecutada mostrando cualquier mensaje de error en caso de que lo hubiese.

Las posiciones predeterminadas se guardan en el archivo ‘positions.json’ el cual tiene le siguiente formato:

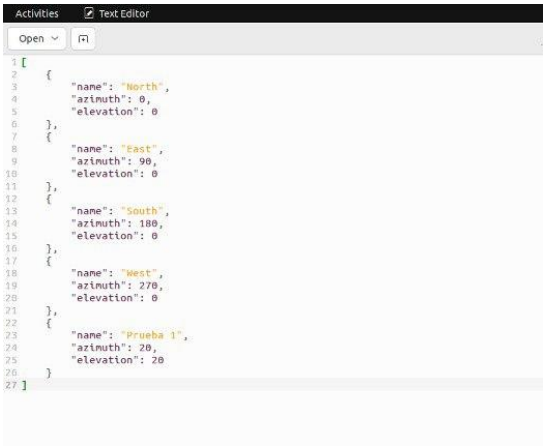

Figura 4-45 Archivo positions.json [AP]

Y el archivo rotors.conf contiene la información necesaria para enlazar con la instancia de *Rotctld* siendo ejecutada, el archivo tiene el siguiente formato:

Figura 4-46 Archivo rotors.conf [AP]

En el primer apartado se deberá especificar la dirección IP a la que se quiere enlazar, en el segundo apartado se introducirá el puerto que, por lo general, será 4533 ya que es el que usa *Rotctld* por defecto, y finalmente el incremento o paso con el que cuentan los rotores, en el caso de la antena del laboratorio del CUD es 0.5º.

4.2 Comprobación de respuesta a los comandos

En este apartado se detalla el proceso seguido para la comprobación de que el envío de comandos, su recepción y ejecución es correcta.

Inicialmente se ha ejecutado una prueba de movimiento de la antena con el terminal de *Rotctld* para que no hubiese fallo en el lado del Daemon y de la unidad de control ROT 2 PROG, la cual se debe encontrar en modo 'A' ya que es así como el controlador responderá a comandos del software de control que se ejecuta en un ordenador conectado.

Figura 4-47 Controlador ROT 2 PROG en modo A

Para esto se han enviado los siguientes comandos al terminal: 'P 10 0' y 'P 350 0'. Se han enviado estos comandos precisamente para comprobar el correcto movimiento de la antena tanto en sentido horario como en sentido antihorario puesto que en un primer momento estaba orientada al Norte. La respuesta por parte de la antena ha sido satisfactoria.

Posteriormente, se ha comprobado el correcto enlace de la aplicación web con el Daemon de *Rotctld*. Para ello, se ha especificado la dirección IP en la que se encuentra el servidor ejecutando el Daemon (192.168.17.50) y su respectivo puerto (4355) en el archivo rotors.conf, también se ha indicado el paso del rotor en el apartado de 'increments'.

Tras enlazar correctamente con el Daemon y posteriormente con el controlador, se ha iniciado el script de Python conforme a los pasos mencionados en el apartado anterior y a continuación se han enviado los siguientes comandos desde el menú desplegable de la aplicación web:

- Al enviar el comando North, desde la App web, la antena se dispone en orientación norte como se observa en la siguiente figura:

Figura 4-48 Antena en orientación Norte

- Posteriormente se ejecutan los comandos Este, Oeste y Sur en ese orden y se comprobó que la antena obedecía los posicionándose en las direcciones indicadas según los comandos de manera correcta.

A continuación, se comprobó el movimiento de la antena en función del envío de coordenadas en valor absoluto y en valor incremental resultando un éxito ambas operaciones. Por otro lado, observando el intercambio de información de la sesión siendo ejecutada en el terminal vemos que todos los trámites se dan sin error:

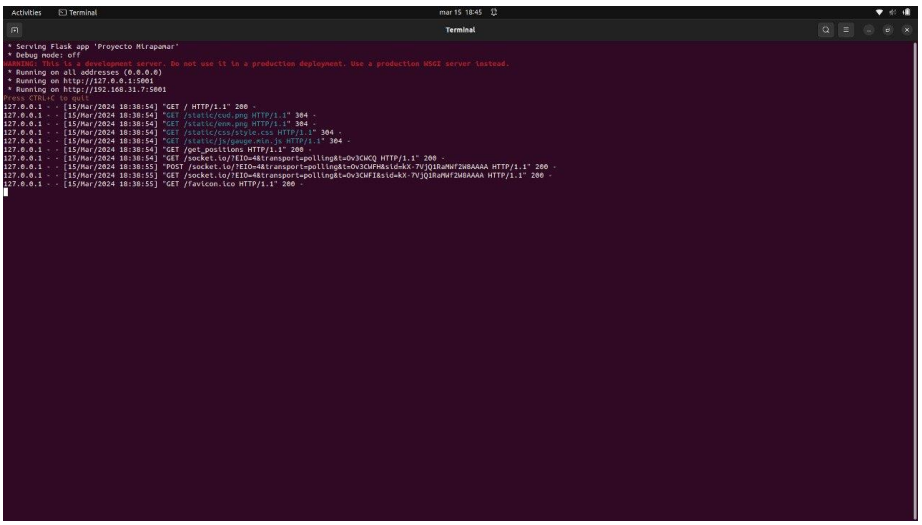

Figura 4-49 Sesión en ejecución en el lado del servidor

Más adelante, se comprobó la correcta ejecución de los comandos de movimiento en elevación enviados desde la aplicación web resultado una vez más en un éxito.

4.3 Calibración de la antena

Para la calibración de la antena se ha aprovechado la orientación hacia el norte que tiene el edificio del laboratorio de investigación. Se ha alineado la barra horizontal que sostiene las antenas con el borde del edificio y posteriormente se ha fijado esta orientación como 270º. Después se ordenó un giro de 90º en sentido horario quedando orientada al norte y una vez en esta posición se reinició la configuración del controlador acorde al manual siguiendo los siguientes pasos:

1. Apagar el controlador
2. Encenderlo presionando el botón F

De esta forma la antena queda orientada al norte y en su display se puede visualizar acimut 0º y elevación 0º.

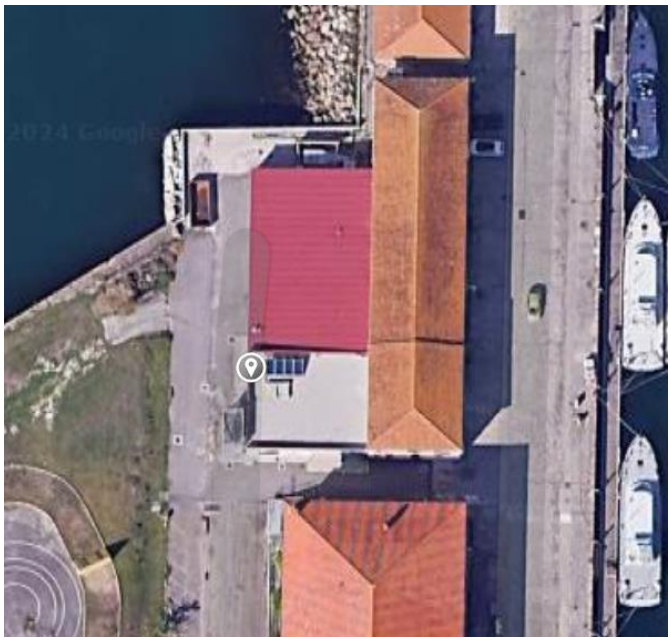

Figura 4-50 Vista aérea del edificio de investigación del CUD

Una vez calibrada la antena en acimut y elevación se volvió a probar la ejecución de comandos para comprobar el funcionamiento correcto del programa. Se enviaron los comandos debidos para ubicar la antena en las distintas posiciones cardinales, esta respondió en todos los casos de manera correcta.

4.4 Manual Básico de uso

A continuación, se expone un manual básico de uso para la correcta manipulación del programa desarrollado. Puesto que el programa enlaza automáticamente con la antena que se desea controlar, este manual parte de la base que se ha instalado correctamente, se ha ejecutado conforme al apartado 4.1 de este capítulo y el usuario se encuentra en la pantalla principal listo para interactuar con la antena. Recorriendo la ventana del programa de izquierda a derecha:

Indicadores de acimut y elevación: en esta sección se encuentran dos indicadores visuales que muestran en tiempo real la orientación y elevación de la antena. Estos se actualizan automáticamente conforme la antena cambia de posición.

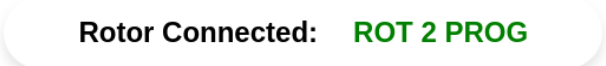

Figura 4-51 Indicador de controlador conectado

Debajo de los indicadores de acimut y elevación se encuentra una ventana que indica el controlador al que se está conectado tal y como se ve en la figura 4-14.

Modificación de la posición de la antena: esto se podrá hacer de 3 formas distintas, mediante la orden de un valor de acimut o elevación específico, mediante los botones de movimiento incremental o desde la ventana desplegable de posiciones predeterminadas.

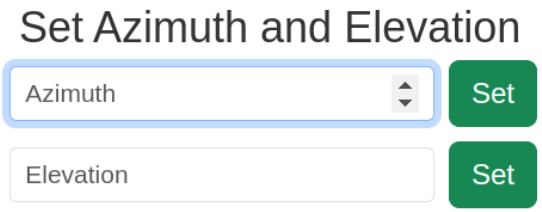

Figura 4-52 Modificación de la posición por valor absoluto

Para mover la antena a una orientación específica se deberá de ingresar el valor de acimut o elevación en los cuadros de texto indicados para ello y presionar el botón “set”.

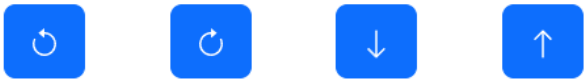

Figura 4-53 Modificación de la posición por valor incremental

Para modificar la posición de la antena según el paso del rotor (este se especifica en el archivo rotors.conf) se presionará el primer botón azul para moverla en sentido antihorario, el segundo para moverla en sentido horario. Para modificar la elevación se presionará el tercer botón para mover la antena hacia arriba y el cuarto botón para moverla hacia abajo según el paso del rotor.

Figura 4-54 Menú desplegable de posiciones predeterminadas

Para orientar la antena a una posición predeterminada, se elegirá la deseada en el menú de la figura 4-17 y se presionará el botón “Go”.

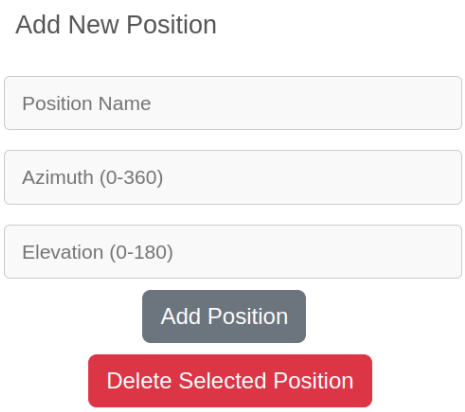

Figura 4-55 Sección para añadir nueva posición predeterminada

Para añadir una nueva posición al menú desplegable se introducirá el nombre, el acimut y elevación indicado en el menú de la figura 4-18 y se presionará el botón “Add Position”. Para eliminar una posición del menú desplegable se seleccionará esta y se presionará el botón “Delete Position”.

Figura 4-56 Botones de STOP, PARK y RESET

El botón “STOP” detiene cualquier movimiento siendo ejecutado de la antena.

El botón “PARK” devuelve la antena a la posición norte y 0º de elevación.

El botón “RESET” reinicia la antena si el controlador tiene soporte para ello.

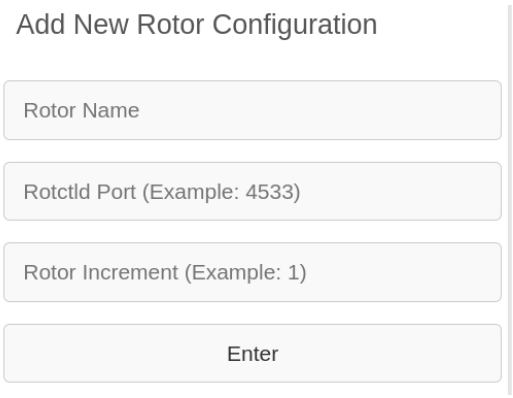

Figura 4-57 Sección para añadir nuevos controladores

Para añadir un controlador nuevo o modificar el actual, se indicará en la sección de la figura 4-18 el nombre, el puerto (por lo general será 4533) y el incremento o paso que soporta este rotor.

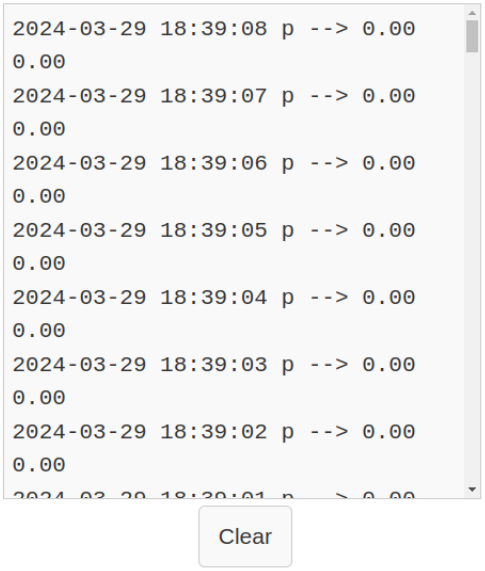

Figura 4-58 Log de errores

El log de errores muestra la comunicación con el *daemon* de *rotctld*. El botón "Clear" borra todos los mensajes dejando espacio para más.

Para interactuar con una antena desde un dispositivo móvil o Tablet se deberá acceder desde el navegador de esta a la dirección IP inferior que se indica al arrancar el programa. Esto nos llevará a la siguiente página la cual funciona de la misma manera que la versión de escritorio:

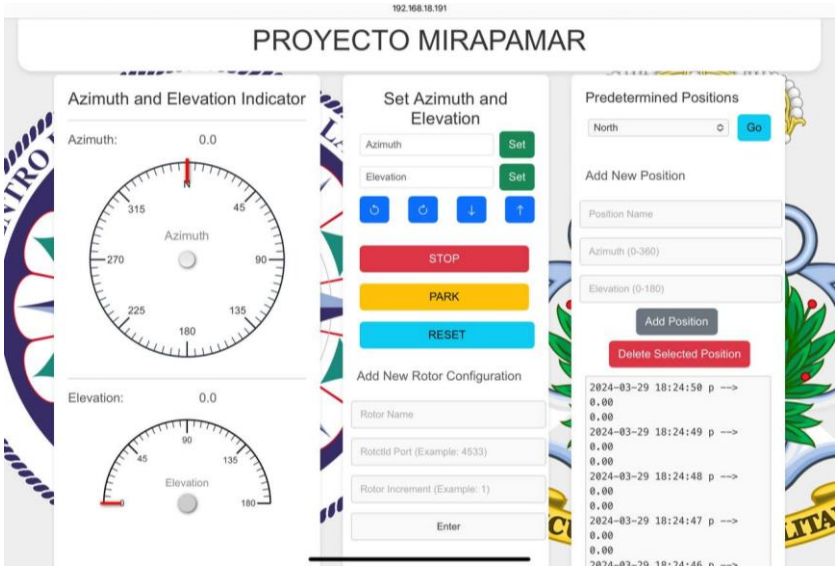

Figura 4-59 Aplicación Web en una Tablet

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Resultados Alcanzados

El trabajo propuesto consistía en la creación de un software destinado a gestionar y visualizar el funcionamiento de un sistema mecánico encargado de posicionar antenas, abarcando tanto la dirección horizontal (azimut) como la vertical (elevación). Dicho software se debía encargar de emitir las órdenes necesarias para ajustar los movimientos de los rotores responsables del azimut y la elevación. La interfaz para este sistema de control debía consistir en una sencilla página web. Tras completar el desarrollo del software, se debía efectuar una calibración junto con varias pruebas de direccionamiento, lo que permitirá evaluar las capacidades finales del sistema.

En cuanto al primer objetivo fijado, la creación de un software capaz de gestionar el movimiento en elevación y acimut de la antena del laboratorio de investigación se conseguido mediante el desarrollo de una aplicación web que permite interactuar con *Rotctld* de manera visual consiguiendo un control de la antena mediante elementos como botones o indicadores, dejando atrás la línea de comandos con la que había que interactuar previamente y mejorando la accesibilidad al control de la antena.

Por otro lado, se ha llevado a cabo una calibración de la orientación de la antena, tanto en acimut como en elevación para asegurar su efectividad que garantiza la correcta integración del software desarrollado con el hardware existente del laboratorio del CUD.

Por último, al crear una aplicación web con el lado del cliente desarrollado en html y el lado del servidor en Python se facilita el posible desarrollo de mejoras del programa ya que tanto html como Python son excelentes lenguajes para principiantes debido a su sintaxis simple y legible, que se asemeja mucho al lenguaje natural. Esto facilita que los recién llegados comprendan los conceptos de programación y comiencen a codificar rápidamente.

Tras la realización completa del proyecto se puede afirmar que además de cumplir con los objetivos establecidos al inicio del proyecto y de esta memoria, se ha conseguido lo siguiente:

- Desarrollo de una interfaz web amigable para cualquier usuario independientemente de su formación técnica o su conocimiento sobre los sistemas de posicionamiento de antenas.
- Se ha conseguido un control preciso de azimut y elevación de la antena mediante pruebas de calibración, asegurando una recepción óptima de la señal de la antena.
- Se ha logrado un movimiento eficiente de la antena pudiendo visualizar en tiempo real la posición de esta minimizando el tiempo de respuesta y en consecuencia el consumo de energía.

5.2 Líneas Futuras

En este apartado se plantean una serie de líneas futuras a seguir de cara a continuar con la implementación de mejoras en la aplicación web.

En primer lugar, se plantea explorar la adquisición de una unidad de control de antena mejorada. Aunque la actual unidad de control Spid Azi/Ele con USB ha sido de gran utilidad, ofreciendo un control fiable tanto de la azimuth como de la elevación, el panorama tecnológico está evolucionando rápidamente. Una unidad de control superior podría ofrecer características avanzadas tales como posicionamiento más preciso, tiempos de respuesta más rápidos e interfaces de usuario mejoradas, potencialmente integrándose sin problemas con el software existente para una experiencia aún más fluida. Tal actualización no solo reforzaría el rendimiento del sistema, sino que también abriría nuevas posibilidades en la gestión de antenas, desde patrones de movimiento más complejos hasta ajustes ambientales en tiempo real. Se propone la adquisición del controlador microHAM Advanced Rotor Controller que permite establecer rápidamente direcciones a través de la función Touch'n Turn simplemente tocando la dirección deseada en el mapa del display con el que cuenta.

También se plantea la integración de nuevas funcionalidades como tecnologías de seguimiento y mecanismos de control adaptativo. La adición de seguimiento dotaría al sistema con la capacidad de seguir automáticamente las trayectorias de satélites o señales terrestres, asegurando una recepción óptima de señal mediante el ajuste dinámico del acimut y la elevación de la antena en tiempo real. Esto se podría llevar a cabo implementando funcionalidades de trabajos de fin de grado de años anteriores [50]. Estos avances prometen elevar significativamente la experiencia del usuario, ofreciendo un sistema de control más autónomo, robusto y eficiente. Otra posible funcionalidad que resultaría práctica sería establecer orientaciones de la antena mediante apuntamiento sobre un mapa desde la aplicación web, para ello se incorporaría una nueva ventana o sección que cuente con un mapa con posibilidad de hacer *zoom* y mediante el click en un lugar específico la antena se orientaría a dicho punto realizando los debidos cálculos.

Otra línea futura que seguir y que supondría una mejora significativa es la capacidad de acceder a la aplicación web desde diferentes redes, liberándose de la actual limitación de accesibilidad en la misma red. Este desarrollo implicaría implementar capacidades de acceso remoto seguras, permitiendo al usuario conectarse a la aplicación desde cualquier parte del mundo con una conexión a internet. Incorporando medidas de seguridad avanzadas, como conexiones cifradas, autenticación de usuarios y posiblemente una configuración de red privada virtual (VPN), se podría asegurar una experiencia de acceso remoto segura y confiable. Este avance no solo aumentaría enormemente la versatilidad y conveniencia del software, sino que también abriría nuevas posibilidades para la monitorización remota, el control y el análisis de datos, ampliando significativamente el alcance y la utilidad de la aplicación en diversos casos de uso.

Por último, una posible línea futura es elevar la accesibilidad móvil del software de control, aunque en su estado actual la aplicación web es accesible desde dispositivos móviles, se podrían implementar una serie de mejoras en la interfaz de usuario (UI) y la experiencia de usuario (UX) diseñadas específicamente para estas plataformas. Esto involucrará optimizar el diseño para pantallas más pequeñas, simplificar la navegación para asegurar que las características clave sean fácilmente accesibles con solo unos pocos toques y mejorar la capacidad de respuesta al tacto para una interacción más fluida. Además, al incorporar elementos de diseño adaptativo, el software proporcionará una experiencia de visualización óptima en una amplia gama de dispositivos, desde smartphones hasta tablets. Estas mejoras tienen como objetivo hacer que la versión móvil no solo sea más accesible sino también excepcionalmente amigable para el usuario, garantizando al usuario el control y gestión de la antena de manera eficaz mientras están en movimiento, sin comprometer la funcionalidad o el rendimiento.

6 BIBLIOGRAFÍA

En esta sección figurarán todas las referencias, sean recursos web, libros, artículos, etc., incluyendo la información de autores, título de la obra, nombre de la publicación, año, edición y enlace más fecha de último acceso en el caso de referencias a recursos online.

[1] «The Antenna History Page». Accedido: 28 de marzo de 2024. [En línea]. Disponible en: https://www.antenna-theory.com/intro/history.php

[2] K. B. Cafe RF, «A New Antenna Rotor, January 1960 Electronics World». Accedido: 28 de marzo de 2024. [En línea]. Disponible en: https://www.rfcafe.com/references/electronics-world/antenna-rotor-electronics-world-january-1960.htm

[3] A. A. Mulla y P. N. Vasambekar, «Overview on the development and applications of antenna control systems», *Annual Reviews in Control*, vol. 41, pp. 47-57, ene. 2016, doi: 10.1016/j.arcontrol.2016.04.012.

[4] «Antenna Control Systems - Antenna Technologies, Communications & Power Industries (CPI)». Accedido: 15 de febrero de 2024. [En línea]. Disponible en: https://www.cpii.com/product.cfm/15/134

[5] W. Gawronski, *Modeling and Control of Antennas and Telescopes*. en Mechanical Engineering Series. Boston, MA: Springer US, 2008. doi: 10.1007/978-0-387-78793-0.

[6] «CONAE avanza en la instalación de dos antenas en la Antártida», Argentina.gob.ar. Accedido: 19 de febrero de 2024. [En línea]. Disponible en: https://www.argentina.gob.ar/noticias/conae-avanza-en-la-instalacion-de-dos-antenas-en-la-antartida

[7] N. S. Nise, *Control System Engineering*, 8.ª ed. Wiley-Interscience, 2019.

[8] «Rotores Yaesu con controlador, G-450, G-1000, G-2800». Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://www.wimo.com/es/yaesu-antenna-rotors-with-controller

[9] «Rotador de satélites horizontal/vertical RAS». Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://www.wimo.com/es/ras

[10] «Rotador de antena luminosa AR-500X». Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://www.wimo.com/es/ar-500x

[11] «Hygain HAM-IV Hygain Rotores de antena con DCU-3X». Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://www.wimo.com/es/ham-iv-1

[12] «SPID ROTOR SPX EL/AZ 01 936.54 €|Envío gratuito», Astroradio. Accedido: 28 de marzo de 2024. [En línea]. Disponible en: https://www.astroradio.com/p/rotor_spx_elaz_01/

[13] microHAM s.r.o, «ARCO - Smart Antenna Rotator Controller». Accedido: 21 de febrero de 2024. [En línea]. Disponible en: http://www.microham.com/contents/en-us/d50_ARCO.html

[14] «4O3A Rotor Genius Control Unit». Accedido: 21 de febrero de 2024. [En línea]. Disponible en: https://www.wimo.com/en/4o3a-rotator-genius

[15] «Spid Control unit Azi/Ele w. USB». Accedido: 21 de febrero de 2024. [En línea]. Disponible en: https://www.wimo.com/en/25089-rot2u
[16] «AIO All in One by WD8KNC», The DXZone Amateur Radio Internet Guide. Accedido: 8 de febrero de 2024. [En línea]. Disponible en: https://www.dxzone.com/dx9745/aio-all-in-one-by-wd8knc.html
[17] «W7GJ SOFTWARE DOWNLOADS, including GJTRACKER Moontracking Computer Program and CALL3.TXT». Accedido: 8 de febrero de 2024. [En línea]. Disponible en: http://www.bigskyspaces.com/w7gj/tracker.htm
[18] «LP-Rotor Help». Accedido: 8 de febrero de 2024. [En línea]. Disponible en: http://www.telepostinc.com/LP-Rotor-Html/
[19] «Interfacing – N1MM Logger Plus». Accedido: 8 de febrero de 2024. [En línea]. Disponible en: https://n1mmwp.hamdocs.com/setup/interfacing/
[20] «PstRotator - Software for Antenna Rotators». Accedido: 8 de febrero de 2024. [En línea]. Disponible en: https://www.pstrotator.com/
[21] «remoteRotator – EA4TX.com». Accedido: 8 de febrero de 2024. [En línea]. Disponible en: https://ea4tx.com/products-page/ars-usb/ars-remote/remoterotator/
[22] «RotorCraft Version 1.0». Accedido: 8 de febrero de 2024. [En línea]. Disponible en: https://www.kk5jy.net/RotorCraft-v1/
[23] «WinRotor Software». Accedido: 8 de febrero de 2024. [En línea]. Disponible en: https://www.qsl.net/dh0gmr/winrotore.htm
[24] «SimpleSat Rotor Controller - The DXZone.com». Accedido: 8 de febrero de 2024. [En línea]. Disponible en: https://www.dxzone.com/dx34542/simplesat-rotor-controller.html
[25] «Ubuntu Manpage: rotctl - control antenna rotators». Accedido: 8 de febrero de 2024. [En línea]. Disponible en: https://manpages.ubuntu.com/manpages/trusty/man1/rotctl.1.html
[26] «What Is GUI? Graphical User Interfaces, Explained». Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://blog.hubspot.com/website/what-is-gui
[27] «There and Back Again: The Evolution of the Graphical User Interface – Xojo Programming Blog». Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://blog.xojo.com/2014/07/29/there-and-back-again-the-evolution-of-the-graphical-user-interface/
[28] «Graphical User Interface History», KASS. Accedido: 7 de febrero de 2024. [En línea]. Disponible en: https://kartsci.org/kocomu/computer-history/graphical-user-interface-history/
[29] «Dreamweaver gratuito | Descargar la versión completa de Adobe Dreamweaver». Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://www.adobe.com/es/products/dreamweaver/free-trial-download.html
[30] «Webflow: Create a custom website | Visual website builder». Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://webflow.com/?utm_source=google&utm_medium=search&utm_campaign=SS-GoogleSearch-Brand-Tier2&utm_term=aud-445214972577:kwd-11668981_webflow_e_351375596855__&gad_source=1&gclid=CjwKCAiAuNGuBhAkEiwAGId4aj_JD3uv99vRJ9XuUVfoScqShl820vzL49y2534Lni1hQQEUboMoABoCqioQAvD_BwE
[31] «What is an IDE?» Accedido: 21 de febrero de 2024. [En línea]. Disponible en: https://www.redhat.com/en/topics/middleware/what-is-ide
[32] «Visual Studio Code - Code Editing. Redefined». Accedido: 16 de marzo de 2024. [En línea]. Disponible en: https://code.visualstudio.com/
[33] «WebStorm: The JavaScript and TypeScript IDE, by JetBrains», JetBrains. Accedido: 16 de marzo de 2024. [En línea]. Disponible en: https://www.jetbrains.com/webstorm/promo/
[34] «A hackable text editor for the 21st Century», Atom. Accedido: 16 de marzo de 2024. [En línea]. Disponible en: https://atom-editor.cc
[35] «A modern, open source code editor that understands web design», Brackets. Accedido: 16 de marzo de 2024. [En línea]. Disponible en: http://brackets-cont.github.io

[36] «tkinter — Interface de Python para Tcl/Tk», Python documentation. Accedido: 16 de marzo de 2024. [En línea]. Disponible en: https://docs.python.org/3/library/tkinter.html
[37] «Framework: qué es, para qué sirve y algunos ejemplos», UNIR FP. Accedido: 21 de febrero de 2024. [En línea]. Disponible en: https://unirfp.unir.net/revista/ingenieria-y-tecnologia/framework/
[38] «Angular». Accedido: 16 de marzo de 2024. [En línea]. Disponible en: https://angular.io/
[39] «Vue.js». Accedido: 16 de marzo de 2024. [En línea]. Disponible en: https://vuejs.org/
[40] «Express - Node.js web application framework». Accedido: 16 de marzo de 2024. [En línea]. Disponible en: https://expressjs.com/
[41] M. O. contributors Jacob Thornton, and Bootstrap, «Bootstrap». Accedido: 16 de marzo de 2024. [En línea]. Disponible en: https://getbootstrap.com/
[42] «Hamlib/hamlib.github.io». Ham radio control library development, 28 de junio de 2022. Accedido: 28 de febrero de 2024. [En línea]. Disponible en: https://github.com/Hamlib/hamlib.github.io
[43] «Hamlib/Hamlib». Ham radio control library development, 26 de febrero de 2024. Accedido: 28 de febrero de 2024. [En línea]. Disponible en: https://github.com/Hamlib/Hamlib
[44] D. Fannin, «dfannin/pyrotor». 16 de enero de 2022. Accedido: 4 de marzo de 2024. [En línea]. Disponible en: https://github.com/dfannin/pyrotor
[45] «The K3NG Arduino Rotator Controller», Radio Artisan. Accedido: 4 de marzo de 2024. [En línea]. Disponible en: https://blog.radioartisan.com/arduino_rotator_controller/
[46] «GitHub - darksidelemm/rotctld-web-gui: A very basic Web GUI to drive a rotctld rotator.» Accedido: 4 de marzo de 2024. [En línea]. Disponible en: https://github.com/darksidelemm/rotctld-web-gui
[47] A. Bjorgan, «bjorgan/rotctld-web-gui». 9 de enero de 2023. Accedido: 4 de marzo de 2024. [En línea]. Disponible en: https://github.com/bjorgan/rotctld-web-gui
[48] executradmin, «Python, el lenguaje con mayor crecimiento», Executrain. Accedido: 3 de abril de 2024. [En línea]. Disponible en: https://executrain.com.mx/python-el-lenguaje-con-mayor-crecimiento/
[49] M. O. contributors Jacob Thornton, and Bootstrap, «Get started with Bootstrap». Accedido: 16 de marzo de 2024. [En línea]. Disponible en: https://getbootstrap.com/docs/5.3/getting-started/introduction/
[50] Á. L. Cuquerella y J. M. N. Ortuño, « Desarrollo de una plataforma mecánica de posicionamiento de antenas para el seguimiento automático de satélites »

ANEXO I: IMPLICACIONES SOCIALES, Y/O ECONÓMICAS, Y/O AMBIENTALES

En la actualidad la tecnología y la comunicación desempeñan roles fundamentales en el desarrollo y progreso de la sociedad, resulta crucial analizar las implicaciones sociales, económicas y ambientales de los avances en el campo de las telecomunicaciones. Este apartado aborda cómo el trabajo desarrollado plantea posibles maneras de afectar a la sociedad, impulsar el crecimiento económico y plantear retos y oportunidades para la sostenibilidad ambiental. A través de esta exploración, se busca comprender el impacto no solo de este proyecto en particular sino de la ingeniería de telecomunicaciones en la vida cotidiana y en el futuro del planeta.

En el ámbito social este proyecto ha contribuido a la mejora de las comunicaciones, al optimizar el posicionamiento de las antenas, además contribuirá significativamente a mejorar la calidad y fiabilidad del seguimiento de buques en la zona puesto que la función de la antena en sí será actuar como un radar pasivo. Esto es especialmente relevante esta zona de Galicia donde el puerto de Marín es uno de los más relevantes en cuanto a tráfico marítimo. Esta implementación podría aumentar significativamente la seguridad en el puerto, permitiendo una detección más efectiva de embarcaciones y objetos, lo cual resulta importante para prevenir accidentes, contrabando y otras actividades ilícitas. Además, supondría una mejora en la seguridad y eficiencia del puerto teniendo efectos positivos en la economía local, fomentando el turismo y el comercio y mejorando la calidad de vida de la población local al reducir potenciales riesgos asociados con el tráfico marítimo.

Con relación a las implicaciones medioambientales, el principal factor a tener en cuenta es el impacto que puede tener una antena en la ría de Pontevedra. La integración visual de antenas y otras estructuras de telecomunicaciones en entornos tanto naturales como urbanos es un tema de creciente importancia, especialmente cuando se consideran los impactos visuales que estas pueden tener en paisajes valiosos o en áreas con significado cultural o histórico. Es importante tratar de preservar la calidad del entorno para residentes y visitantes, y de mantener o aumentar el valor de la propiedad en áreas circundantes. Para llevar a cabo proyectos de este tipo, se deben tomar medidas como realizar un análisis visual previo para entender cómo la antena se percibirá desde diferentes puntos de vista, posteriormente ejecutar un diseño y una colocación estratégica y finalmente realizar la instalación con camuflaje y disimulo para no sólo reducir el impacto visual sino también para interferir lo mínimo posible con la fauna local, especialmente las aves.

La optimización del sistema de control de antenas trae otros beneficios como la mejora en la eficiencia energética. Al reducir el tiempo y la energía necesarios para ajustar y mantener las antenas en su orientación óptima, se disminuye el consumo general de electricidad. Esta reducción en el uso de energía no solo contribuye a disminuir la huella de carbono de las operaciones de telecomunicaciones, sino que también alinea a la industria con objetivos más amplios de sostenibilidad y protección ambiental. En el contexto global donde la lucha contra el cambio climático es cada vez más urgente debido a la actuación del ser humano en el planeta, la implementación de tecnologías que promuevan el ahorro energético representa un paso crucial hacia la minimización del impacto ambiental de sectores tecnológicamente intensivos.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

En el desarrollo e implementación de los sistemas de posicionamiento y control mecánico de antenas, nos encontramos ante un panorama complejo donde las innovaciones tecnológicas ofrecen nuevas oportunidades significativas para mejorar la comunicación y la conectividad a nivel mundial. Sin embargo, estos avances también plantean importantes cuestiones éticas y sociales que deben ser cuidadosamente estudiadas. Desde la privacidad y vigilancia hasta el impacto ambiental y la equidad en el acceso a la información, cada aspecto de este proyecto conlleva responsabilidades y desafíos. En este apartado, se proponen una serie de cuestiones a explorar, reflexionando sobre cómo la tecnología afecta a la sociedad y al individuo, y destacando la importancia de un desarrollo tecnológico responsable. Este enfoque permite no solo comprender mejor las implicaciones del trabajo desarrollado, sino también contribuir de manera positiva a un futuro donde la tecnología sirva al bienestar común.

El potencial de los sistemas de control de antenas para ser utilizados en actividades de vigilancia plantea importantes cuestiones éticas. La capacidad de rastrear y recopilar datos de comunicaciones resulta instrumental en esfuerzos de seguridad nacional o en investigaciones criminales. Sin embargo, siempre existe el riesgo de abuso, donde la vigilancia se extiende más allá de objetivos legítimos, pudiendo llegar a invadir la privacidad personal sin el debido proceso legal. Es crucial establecer límites claros y regulaciones que protejan los derechos a la privacidad, equilibrando la seguridad con las libertades civiles.

Otro aspecto que considerar es que, en aplicaciones críticas, como las comunicaciones de emergencia, la seguridad y la fiabilidad del sistema de posicionamiento de antenas son de suma importancia. Un fallo podría tener consecuencias graves, desde la interrupción de servicios vitales hasta la pérdida de vidas. Es éticamente necesario implementar rigurosos estándares de seguridad y realizar pruebas exhaustivas para asegurar la fiabilidad del sistema, así como desarrollar protocolos claros para la respuesta en caso de fallos.

El uso de tecnologías de posicionamiento en contextos militares o en la escalada de conflictos presenta dilemas éticos significativos. Mientras que la tecnología puede ser crucial para la defensa nacional, su aplicación en conflictos armados plantea preguntas sobre la contribución a la violencia y el sufrimiento humano. Resulta importante reflexionar sobre las implicaciones de estas aplicaciones y considerar medidas para limitar el uso indebido de la tecnología en escenarios de conflicto.

Otro factor importante es el efecto que puede tener la implementación de sistemas de posicionamiento de antenas en comunidades locales. Mientras que la mejora en la conectividad puede ofrecer beneficios económicos y sociales, los proyectos también pueden enfrentar resistencia debido a preocupaciones sobre la salud, el impacto visual, o la utilización del suelo. Dialogar con las comunidades afectadas, comprender sus preocupaciones, y trabajar hacia soluciones mutuamente beneficiosas es fundamental para un desarrollo tecnológico responsable.

La rápida evolución de la tecnología plantea constantes desafíos y oportunidades éticas. El desarrollo futuro de sistemas de posicionamiento de antenas puede abrir nuevas aplicaciones y mejorar la eficiencia y accesibilidad. Sin embargo, también puede generar nuevos dilemas éticos, especialmente en lo que respecta a la privacidad, seguridad, y equidad. Para un progreso sostenible y responsable es clave mantener un compromiso con la reflexión ética continua y la adaptación a los cambiantes contextos tecnológicos y sociales.

ANEXO III: CÓDIGO EMPLEADO

El código empleado se ha subido a la siguiente carpeta de Google Drive:

https://drive.google.com/file/d/1M5CjpflpUWGNCv3MCtzX7skvFFmpF69M/view?usp=drive_link