INSTITUTO TECNOLÓGICO SUPERIOR
de la Región de los Llanos

TESIS PROFESIONAL

Nombre del proyecto: Prototipo para posicionamiento remoto de una antena de telecomunicaciones por medio de una conexión segura VPN

	Participantes (apellidos, nombre)	No. de control:
1	Leonardo Campos Vázquez	16B100264

Carrera: Ingeniería Mecatrónica

Semestre: 10

Asesor: Osbaldo Aragón Banderas
Gua

Celaya, Guanajuato a 5 de noviembre de 2021

ÍNDICE

Índice de Figuras

Índice de Tablas

DEDICATORIA

A mis padres: Miguel Ángel Campos Pineda y Margarita Vázquez Sandoval, por confiar y creer en mis expectativas, por los consejos, valores y principios que me han inculcado, y que con su valioso esfuerzo diario me han permitido sacar adelante mis estudios.

AGRADECIMIENTOS

Principalmente quiero agradecer a Dios por proporcionarme salud y protección, así como a mis padres que con su apoyo, dedicación, sacrificio, educación y amor me hicieron posible salir adelante, no solo en mis estudios y en el ámbito académico sino también en los demás aspectos de mi vida. También quiero agradecer de todo corazón a mi tía la Dra. Martina Vázquez Sandoval pues gracias a sus consejos, su cuidado, su afecto, su ayuda, su hogar, pero sobre todo a su gran corazón, se me pudo facilitar mi estadía durante los meses de mi residencia profesional y mi posterior realización de tesis profesional

Por último, pero no menos importante, quiero agradecer a los docentes pertenecientes a la carrera de ingeniería Mecatrónica del Instituto Tecnológico Superior de la Región de los Llanos, pues gracias a su vocación y enseñanza es que pude hacerme de los conocimientos y aptitudes concernientes a mi carrera. Quiero hacer especial mención del M.C. Jesús Leonel Arce Valdez, y del M.C. Osbaldo Aragón Banderas, pues durante mi permanencia en el instituto siempre me brindaron su apoyo, su atención y su guía desde el punto de vista académico basado en sus experiencias y conocimientos.

INTRODUCCIÓN

En los últimos años ha habido un aumento constante del uso de redes de telefonía móvil en el país, acorde con el Instituto Nacional de Estadística y Geografía (INEGI), en 2015 un 71.5 % de la población total del país contaba con un teléfono celular conectado a la red móvil, pero para 2019 el porcentaje aumentó a 75.1 %, sin embargo otras fuentes indican que para el año 2019 ya había 124 millones de usuarios de la red móvil registrados (Telconomia, 2021), es decir el 96% de la población total del país. Por lo anterior es correcto afirmar que la demanda de la red de telefonía celular ha crecido significativamente en los últimos 5 años, tendencia que no parece que vaya a cambiar en el corto plazo. Esto revela una necesidad de las empresas del sector de telecomunicaciones de aumentar la cobertura de su red para satisfacer este aumento de la demanda del servicio que ha sobrepasado a su infraestructura actual, las consecuencias de no adaptar ésta última a tales circunstancias desembocan en la inestabilidad e intermitencia en el desempeño del servicio y la pérdida de posibles clientes al ser las empresas incapaces de proporcionarles el mismo. Así pues, para aumentar la cobertura y la calidad de la red de telefonía móvil se necesita realizar un ajuste de posición de los parámetros ‘Tilt’ y ‘Azimut’ de las antenas que proveen cobertura, con respecto a un satélite de comunicación. Con este fin es que son empleados los sistemas de inclinación remota (RET) que pueden aumentar la cobertura de la estación base al mismo tiempo que reducen la interferencia de esta.

Los sistemas de inclinación remota (RET) son una herramienta que permite reposicionar los movimientos o parámetros ‘Tilt’ (inclinación) y ‘Azimut’ (Rotación/orientación) de una o varias antenas de telecomunicaciones desde una ubicación remota. En este proyecto se propone el diseño e implementación de un sistema de este tipo, en el que un usuario pueda acceder desde internet a través de un túnel VPN y con las credenciales correctas a un servidor ubicado en una red privada, dicho servidor ejecutará y entregará a cada cliente que se conecte a él una aplicación web con la cual dicho usuario podrá modificar en tiempo real las magnitudes de los movimientos ‘Tilt’ y ‘Azimut’ de un modelo miniatura de una

antena de telecomunicaciones. Debido a que el servidor VPN solicita unas credenciales determinadas para permitir a los usuarios acceder a la red privada, es como se tiene una interfaz de conexión entre el operador y el sistema bajo un esquema de seguridad confiable.

En el proyecto se puede observar el diseño mecánico realizado del sistema, así como el proceso de creación del servidor VPN en una Raspberry Pi y la creación de un servidor web, así como la aplicación web de control en una tarjeta NodeMCU que sirve de interfaz entre el sistema y el usuario.

En breve se presenta la problemática, los objetivos, para el desarrollo del proyecto basado en el estándar de sistemas de inclinación remota (RET). Se define el problema del cual surge la necesidad de implementar este tipo de tecnología en la infraestructura de telecomunicaciones de nuestro país, haciendo especial hincapié en la necesidad de un sistema de este tipo en la red de cobertura móvil nacional para poder hacer frente a la alta demanda de usuarios cada año más grande.

Se pone en contexto al lector acerca de los diversos conceptos relacionados con las formas de recepción y emisión de las antenas de telecomunicaciones, así como la manera en que distribuyen su cobertura dependiendo de diversos factores, tales como diseño, dimensiones, orientación, energía, etc., con el fin de que comprenda la importancia e influencia que tiene el proyecto para mejorar la infraestructura existente. También en este apartado se explican brevemente las herramientas y tecnologías en materia de software y hardware involucradas en el desarrollo del proyecto para que más adelante en el apartado de ´Procedimiento’ sea más sencillo reconocer algunos conceptos y términos técnicos. Para concluir se expusieron los resultados obtenidos del proyecto, midiendo los parámetros relevantes que contribuyen al cumplimiento del objetivo general del mismo, esto es el tiempo transcurrido entre que el cliente realiza una petición hasta que el servidor la recibe (tanto desde una red local como desde una externa).

OBJETIVOS

Objetivo general

Diseño e implementación de un prototipo de un sistema de posicionamiento remoto basado en el control en tiempo real de las dos magnitudes de movimiento de una antena de telecomunicaciones, Azimut y Tilt (Orientación e inclinación), a petición y requerimiento de un usuario determinado conectado a través de una conexión segura tipo VPN.

Objetivos específicos

- Realización del diseño de piezas y del ensamblaje mecánico virtual en un software de dibujo 3D y 2D para la realización de un modelo en miniatura que represente a una antena de telecomunicaciones. Dicho modelo debe ser capaz de reproducir los movimientos de inclinación (Tilt) y orientación/rotación (Azimut) de la antena.
- Realización de un listado de los componentes electrónicos necesarios para efectuar el control de movimiento del modelo y la construcción del mismo, así como concretar su correspondiente adquisición
- Creación y configuración de un servidor ubicado en una red privada utilizando una tarjeta NodeMCU ESP32, con el fin de alojar en él una aplicación web desarrollada para el propio proyecto desde la cual el usuario pueda manipular los movimientos de ‘Tilt y Azimut del modelo de la antena
- Implementación de una puerta de acceso hacia el servidor web ubicado en determinada red local, utilizando un servidor VPN en una Raspberry Pi 3.
- Programación de un Arduino UNO para que sea utilizado como drive de los servomotores que permiten el movimiento de la maqueta de la antena, a partir de los comandos provenientes del servidor (NodeMCU) por medio del protocolo I2C
- Impresión de las partes mecánicas que componen al modelo de la antena previamente diseñado, utilizando para ello la técnica de manufactura de impresión 3D.

- Ensamblaje de las piezas mecánicas del modelo de la antena de telecomunicaciones, así como la integración de sus motores y demás partes electrónicas adquiridas anteriormente, necesarias para el desarrollo del sistema de posicionamiento.
- Puesta en marcha del servidor que alberga la aplicación web de control del Tilt y Azimut y del servidor VPN en la Raspberry Pi 3 para que el prototipo esté listo para las pruebas

JUSTIFICACIÓN

La telefonía móvil es una de las tecnologías e infraestructuras de telecomunicaciones más importantes en la vida diaria de millones de personas en México y el mundo. Permite conectar a distintos individuos en los ámbitos social, académico, profesional, laboral y administrativo a través de largas distancias, de allí que dicha tecnología tuviera el alcance y el desarrollo que posee hoy. Prueba de lo anterior es que en la actualidad hay 7,950 millones de dispositivos conectados a la red de telefonía celular proporcionada por diversas compañías alrededor del mundo. Este número es mayor incluso al número de personas que viven hoy en día, sin embargo, es el 67% del porcentaje total de la población mundial que es usuario de al menos un teléfono móvil conectado a la red. Hablando específicamente de nuestro país México, según una estadística recopilada por Hootsuite y We Are Social en formato de infografía en enero de 2021, el número de dispositivos suscritos a un servicio de red celular en México va alrededor de los 115.4 millones lo que representa al 89.1% de la población nacional. Aunado a esto, acorde con Telconomia los ingresos de la telefonía móvil para solo el año 2019 arrojaron una cifra de $294,441 millones de pesos mexicanos en contraste con la cifra de $248,384 millones MXN del año 2016. Se puede observar que el mercado móvil ha aumentado en 3 años un 18.5% desde el punto de vista de los ingresos cuantificados. Esta expansión se traduce como un incremento en la demanda de este tipo de servicios, así como del aumento de potenciales clientes que por limitaciones técnicas más que económicas no pueden acceder a los mismos. Debido a todos los factores discutidos, el número de torres en el país en las que se suministra el acceso a la red móvil ha aumentado a 29 mil 320. Sin embargo, para aumentar la cobertura demandada se necesita realizar un ajuste en la posición de la antena en aras de alcanzar una orientación óptima con respecto a su correspondiente satélite de comunicación. Para dicha tarea suelen utilizarse los sistemas de inclinación remota (RET) que pueden aumentar la cobertura de la estación base a la vez que se reduce la interferencia. Los sistemas (RET) presentan otras ventajas como ofrecer una alta precisión y amplia gama de ajustes

(Wensheng, Yi, & Yanming, 2010). Para lograr la calibración mencionada se necesita posicionar la antena en base a dos magnitudes y/o movimientos: la elevación (tilt) y la orientación (azimut). La elevación corresponde al ángulo de elevación formado entre la dirección de onda radiada y la horizontal de la base terrena, el azimut corresponde al ángulo de apuntamiento horizontal de una antena.

PLANTEAMIENTO DEL PROBLEMA

En el último lustro el uso de redes de telefonía móvil se ha incrementado en México tal como puede ser observado en los datos proporcionados por el Instituto Nacional de Estadística y Geografía (INEGI), los cuales indican que en 2015 un 71.5 % de la población total del país contaba con un teléfono celular conectado a la red móvil, mismo indicador presentó en 2019 un valor del 75.1 %, sin embargo, otros medios como Telconomia estiman que la cantidad de usuarios es en realidad de hasta 124 millones de personas, lo que equivaldría al 96% de la población total del país en 2019. Aun tomando en cuenta estas discrepancias numéricas en los datos, la conclusión es la misma, la demanda de la red de telefonía celular ha crecido en los últimos años, y no parece que esta tendencia vaya a cambiar en el corto plazo, Esto revela una necesidad por parte del sector de telecomunicaciones de aumentar la cobertura de su red y optimizar el ajuste de la misma, ya que la alta demanda del servicio provoca que la infraestructura actual se vea sobrepasada, ocasionando en el peor de los casos inestabilidad e intermitencia en el desempeño del servicio. Así pues, para aumentar la cobertura y la calidad de la red de telefonía móvil, se necesita realizar un ajuste de posición de los parámetros ‘Tilt’ y ‘Azimut’ de las antenas de telefonía móvil con respecto a un satélite de comunicación, Con este fin es que son empleados los sistemas de inclinación remota (RET) que pueden aumentar la cobertura de la estación base al mismo tiempo que reducen la interferencia de esta. El ajuste de los movimientos ‘Tilt’ y ‘Azimut’ puede ser realizado en campo o de forma remota, este último modo presenta la ventaja de poder manipular varios equipos al mismo tiempo, adaptándose a las fluctuaciones de la demanda de cobertura dentro de determinada área durante el día. Dicho control se realiza a través de una aplicación ejecutada dentro de la red local de cada antena a ser manipulada. Para el acceso a dicha aplicación se debe considerar un sistema de seguridad para la propia red con el fin de que usuarios no autorizados no puedan alterar el funcionamiento de la antena. Debido a ello y ante la necesidad de asegurar un medio para acceder a las redes locales de cada antena de forma remota es que surge el concepto de VPN

CAPÍTULO I: FUNDAMENTO TEÓRICO

1.1 Antena

El Institute of Electrical and Electronics Engineers (IEEE) define una antena como aquel dispositivo parte de un sistema transmisor o receptor diseñado específicamente para radiar o recibir ondas electromagnéticas, su invención se asocia a Alexandr Stepánovich Popov, un ingeniero ruso que entorno al año 1897 descubrió el funcionamiento de la antena mientras experimentaba con las teorías de Hertz y el aumento de sensibilidad del aparato cohesor al unirlo a un hilo conductor suspendido en un cometa. Si bien sus formas son variadas, todas las antenas tienen en común el ser una región de transición entre una zona donde existe una onda electromagnética guiada y una donde haya una onda en el espacio libre a la que puede además asignar un carácter direccional. La representación de la onda guiada se realiza por voltajes y corrientes (hilos conductores y líneas de transmisión) o por campos (guías de onda); en el espacio libre, mediante campos (Aznar, Roca, Casals, Robert, & Boris, 1998).

La misión de la antena es radiar la potencia que se le suministra con las características de direccionalidad adecuadas a la aplicación. Por ejemplo, en radiodifusión o comunicaciones móviles se querrá radiar sobre la zona de cobertura de forma omnidireccional, mientras que en radiocomunicaciones fijas interesará que las antenas sean direccionales. En general, cada aplicación impondrá unos requisitos sobre la zona del espacio en la que se desee concentrar la energía. Asimismo, para poder extraer información se ha de ser capaz de captar en algún punto del espacio la onda radiada, absorber energía de esa onda y entregarla al receptor. Existen, pues, dos misiones básicas de una antena: transmitir y recibir, imponiendo cada aplicación condiciones particulares sobre la direccionalidad de la misma, niveles de potencia que debe soportar, frecuencia de trabajo y otros parámetros que se trataran más adelante. Esta diversidad de situaciones da origen a un gran número de tipos de antenas.

1.3 Emisión y recepción

El origen de las ondas electromagnéticas se basa en el hecho de que toda carga eléctrica en movimiento emite energía en forma de onda electromagnética, siendo la frecuencia de esta onda la misma que la del movimiento de la carga (Huidobro, 2013).

Un campo electromagnético se caracteriza por su frecuencia o longitud de onda y su intensidad (potencia), así como por la polarización (variación con el tiempo de la dirección de la intensidad de campo en un punto determinado del espacio) y la modulación empleada. Tanto la Intensidad de campo eléctrico (E) como la Intensidad de campo magnético (H) son magnitudes vectoriales, función de la posición y del instante, que se relacionan con las fuerzas electrostáticas y electromagnéticas y se miden en voltios/metro y amperios/metro (o su equivalente en teslas) (Huidobro, 2013). Para su medición se emplea un sensor (sonda) apropiado capaz de detectar ambos campos y reflejar el valor de su intensidad en la escala del aparato de medición, que se puede relacionar con la densidad de potencia en el punto mediante una fórmula matemática. No toda la potencia que se entrega a una antena se irradia, pues parte de ella se convierte en calor y se disipa.

A la hora de realizar un estudio acerca de las antenas, sus diferentes tipos y el modo en el que consiguen la propagación o recepción de las ondas electromagnéticas, es conveniente presentar el concepto de "campo cercano" y de "campo lejano".

Figura 1.1 Antena de telefonía móvil

1.4 Campo cercano y campo lejano

Campo cercano hace referencia al patrón de campo que está cerca de la antena, y el término campo lejano (también conocido como zona de Fraunhofer) se refiere al patrón de campo que está a gran distancia (Huidobro, 2013). Durante la mitad del ciclo, la potencia se irradia desde una antena, en donde parte de la potencia se guarda temporalmente en el campo cercano. Durante la segunda mitad del ciclo, la potencia que está en el campo cercano regresa a la antena (Huidobro, 2013). Esta acción es similar a la forma en que un inductor guarda y suelta energía. Por tanto, el campo cercano se llama a veces campo de inducción. La potencia que alcanza el campo lejano continúa irradiando lejos y nunca regresa a la antena (Huidobro, 2013). Por tanto, el campo lejano se llama campo de radiación. La potencia de radiación, por lo general, es la más importante de las dos; por consiguiente, los patrones de radiación de la antena, por lo regular se dan para el campo lejano.

1.5 Diagrama de radiación

Es una representación gráfica de las propiedades de radiación de una antena, en función de las distintas direcciones del espacio a una distancia fija. Para ello se utiliza un sistema de coordenadas esféricas, con la antena situada en el origen y manteniendo constante la distancia, se expresa el campo eléctrico en función de las variables angulares (θ, ϕ) (Aznar, Roca, Casals, Robert, & Boris, 1998).

1.6 Alcance

Según en el modo en que trabaje la antena, su alcance será uno u otro. Si la antena trabaja en modo direccional su alcance será mucho mayor que si lo hace en modo omnidireccional ya que, en este caso, concentra toda su potencia en un rango menor. Definimos zona, como la región donde se encuentran todos los usuarios. Esta zona se divide en dos subzonas:

1) Subzona Broadcast. Esta zona se corresponde con el rango de alcance de la antena en modo omnidireccional (Mansilla, Brandau, & Morineau, 2007).

2) Subzona Beamforming. Esta zona está dividida en "n" beams. Un beam se define como el rango de alcance de la antena en modo direccional para un cierto ángulo

de apertura (Mansilla, Brandau, & Morineau, 2007). Según el ángulo de apertura que se utilice habrá más o menos beams.

Cabe comentar que pese a que en el dibujo la cobertura direccional (beams) es de forma triangular, en realidad la cobertura es un lóbulo redondeado donde existe una distancia máxima.

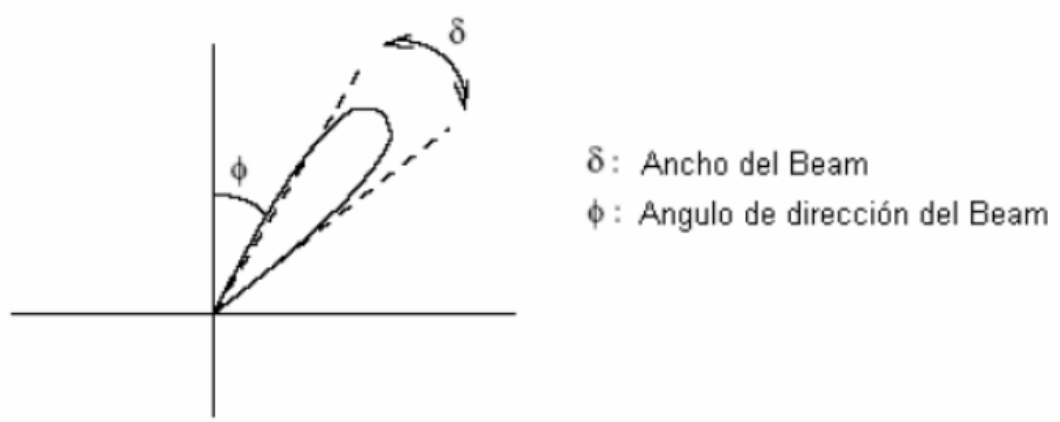

Figura 1.2 Lóbulo de radiación de una antena

El lóbulo de radiación se puede configurar en base a dos ángulos, el azimut (orientación horizontal) y la inclinación de la antena.

1.7 Parámetros de una antena

1.7.1 Ancho de banda

Es el margen de frecuencias en el cual los parámetros de la antena cumplen unas determinadas características (Aznar, Roca, Casals, Robert, & Boris, 1998).

1.7.2 Directividad

Es la relación entre la densidad de potencia radiada en la dirección de máxima radiación, a una cierta distancia R, y la potencia total radiada dividida por el área de la esfera de radio R. La directividad se puede calcular a partir del diagrama de radiación. La ganancia de una antena es igual a la directividad multiplicada por la eficiencia. La relación entre la densidad de potencia radiada por la antena en la

dirección útil y la que radia por el lóbulo trasero se conoce como relación delante/detrás (forward/backward) y es un importante parámetro de diseño de la antena en lo relativo a interferencias (Aznar, Roca, Casals, Robert, & Boris, 1998).

El ángulo que hace referencia el diagrama de radiación del lóbulo principal en el plano horizontal de la antena se denomina "azimut", que para el diagrama de radiación vertical se denomina "ángulo de elevación", que se diseña para concentrar el máximo de radiación para aquellos ángulos por debajo de la horizontal, que es donde se agrupan los usuarios, ya que las antenas se colocan en cotas elevadas para alcanzar una mayor cobertura (Aznar, Roca, Casals, Robert, & Boris, 1998).

1.7.3 Ganancia

Es la relación entre la densidad de potencia radiada en la dirección del máximo a una distancia R y la potencia total entregada a la antena dividida por el área de una esfera de radio R. La eficiencia es la relación entre la ganancia y la directividad, que coincide con la relación entre la potencia total radiada y la potencia entregada a la antena (Aznar, Roca, Casals, Robert, & Boris, 1998).

1.7.4 Rendimiento

Es la relación entre la potencia de radiación y la potencia total aplicada a la antena, en la cual se toma en cuenta, además de la potencia de radiación, la potencia de pérdida (Aznar, Roca, Casals, Robert, & Boris, 1998).

1.7.5 Impedancia

Una antena se tendrá que conectar a un transmisor (o a un receptor) y deberá radiar (recibir) el máximo de potencia posible con un mínimo de perdidas. Se deberá adaptar el transmisor o receptor a la antena para una máxima transferencia de potencia, que se suele hacer a través de una línea de transmisión. Esta línea también influirá en la adaptación, debiéndose considerar entre otros, su impedancia característica y atenuación (Aznar, Roca, Casals, Robert, & Boris, 1998).

La impedancia característica (Z0) es un parámetro que depende de parámetros primarios; de la relación longitud-diámetro del material del conductor y de la frecuencia de trabajo, mientras que la impedancia de entrada es el parámetro circuital de la antena (relación del voltaje de entrada a la corriente de entrada).

1.7.6 Anchura de haz

Es un parámetro de radiación, ligado al diagrama de radiación. Se puede definir el ancho de haz a -3 dB, que es el intervalo angular en el que la densidad de potencia radiada es igual a la mitad de la máxima. También se puede definir el ancho de haz entre ceros, que es el intervalo angular del haz principal del diagrama de radiación, entre los dos ceros adyacentes al máximo (Aznar, Roca, Casals, Robert, & Boris, 1998).

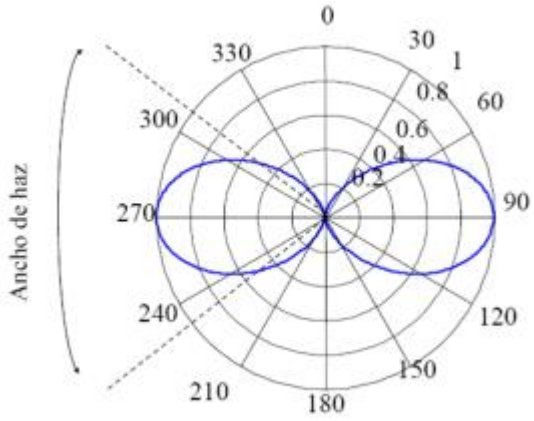

Figura 1.3 Ancho de haz de un diagrama de radiación

1.7.7 Polarización

La polarización electromagnética en una determinada dirección, es la figura geométrica que traza el extremo del vector campo eléctrico a una cierta distancia de la antena, al variar el tiempo. La polarización puede ser lineal, circular y elíptica. La polarización lineal puede tomar distintas orientaciones (horizontal, vertical, +45º, -45º) (Aznar, Roca, Casals, Robert, & Boris, 1998).

Las polarizaciones circular o elíptica pueden ser a derechas o izquierdas (dextrógiras o levógiras), según el sentido de giro del campo (observado alejándose desde la antena).

Se llama diagrama copolar al diagrama de radiación con la polarización deseada, y diagrama contrapolar (crosspolar, en inglés) al diagrama de radiación con la polarización contraria.

1.8 Tipos de antenas

Existe una gran diversidad de tipos de antena dependiendo del uso al que van a ser destinadas. En unos casos deben expandir en lo posible la potencia radiada, es decir, no deben ser directivas (ejemplo: una emisora de radio o una estación base de teléfonos móviles), otras veces deben serlo para canalizar la potencia y no interferir a otros servicios (antenas entre estaciones de radio enlaces).

El tamaño de las antenas está relacionado con la longitud de onda (λ) de la señal de radiofrecuencia transmitida o recibida, debiendo ser, en general, un múltiplo o submúltiplo exacto de esta longitud de onda y es por eso que, a medida que se van utilizando frecuencias mayores, las antenas disminuyen su tamaño (Huidobro, 2013). Si las dimensiones de la antena son mucho más pequeñas que la longitud de onda, las antenas se denominan elementales. La longitud de las antenas resonantes (cuando se anula su reactancia de entrada) es un múltiplo entero de la semilongitud de onda.

1.8.1 Antena isotrópica

La antena isotrópica es una antena hipotética sin pérdida de potencia irradiada y que tiene intensidad de radiación igual en todas direcciones. Sirve de base de referencia para evaluar la directividad (Huidobro, 2013).

1.8.2 Antenas de hilo

Las antenas de hilo están formadas por hilos conductores, eléctricamente delgados, cuyo diámetro es muchas veces menor a la longitud de onda de la onda electromagnética a radiar/recibir. Se modelan como un conductor de sección

infinitesimal. Pueden estar formadas por hilos rectos (dipolos, rombos), espirales (circular, cuadrada o cualquier forma arbitraria) y hélices. En la figura 1.4 se muestran algunos tipos ampliamente empleados en radiocomunicaciones (Huidobro, 2013).

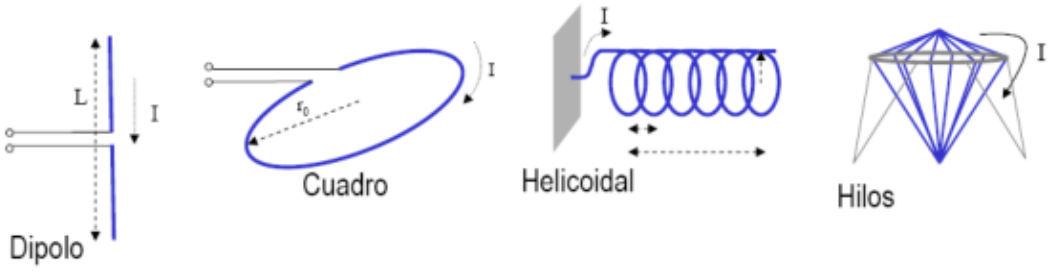

Figura 1.4 Antenas de hilo

El tipo más común son las antenas de dipolo. Esta clase de antena es la más sencilla de todas. El dipolo de media onda o antena de Hertz –el tipo más común– consiste en un hilo conductor de media longitud de onda a la frecuencia de trabajo, cortado por la mitad, en cuyo centro se coloca un generador o una línea de transmisión (Huidobro, 2013). Suelen estar fabricados de aluminio o cobre.

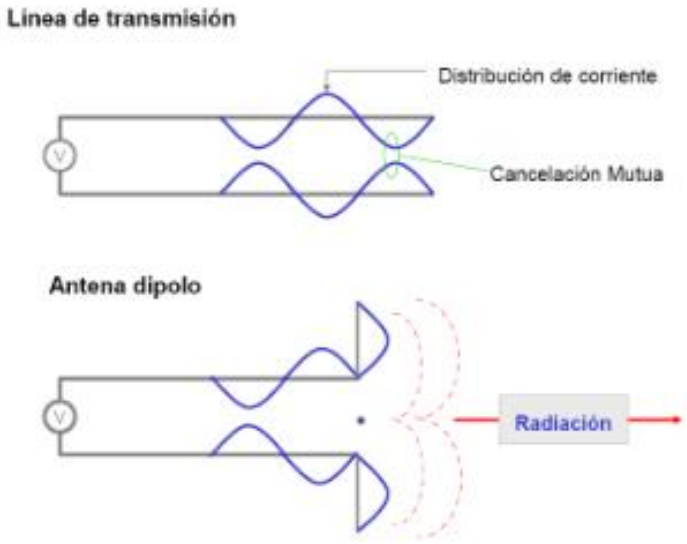

Figura 1.5 Antena de dipolo

1.8.3 Antenas Yagi – Uda

Una aplicación práctica de este tipo de antenas es para la recepción de señales de televisión en la banda de UHF, ya que poseen una gran directividad, cuanto mayor sea el número de elementos pasivos (parásitos) que incorpore, su ganancia será más adecuada para recibir el nivel de señal suficiente para que pueda ser amplificado sin problemas (Huidobro, 2013).

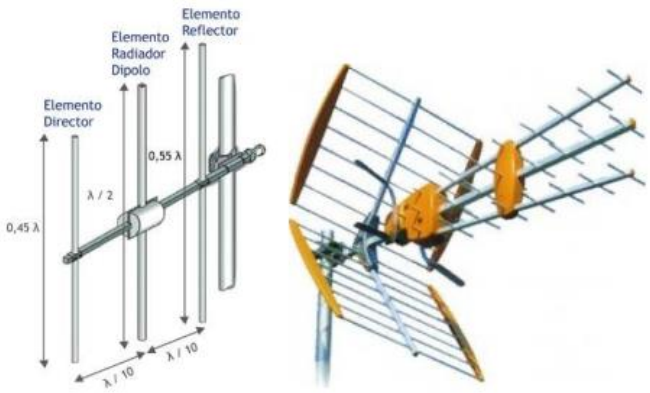

Figura 1.6 Antena Yagi - Uda

1.8.4 Antenas planas

Las antenas planas (microstrip) están formadas por un agrupamiento plano de radiadores (parches) y un circuito que distribuye la señal entre ellos. Su diseño se adecua de forma que la estructura disipe la potencia en forma de radiación. Ambos, parches y circuito, se fabrican utilizando técnicas de fotograbado sobre un sustrato dieléctrico laminado en cobre por ambas superficies. Al ser una tecnología plana, facilita su integración con el resto del sistema, favoreciendo la reducción del tamaño y peso global. Presentan la desventaja de su estrecho ancho de banda, pero actualmente existen numerosos métodos para solventar este inconveniente (Huidobro, 2013).

Figura 1.7 Antena placa

1.8.5 Antenas reflectoras

En el caso de una antena reflectora receptora, su funcionamiento se basa en la reflexión de las ondas electromagnéticas, por la cual las ondas que inciden paralelamente al eje principal se reflejan y van a parar a un punto denominado foco que está centrado en el paraboloide (Huidobro, 2013). En cambio, si se trata de una antena reflectora emisora, las ondas que emanan del foco (dispositivo de emisión) se ven reflejadas y salen en dirección paralela al eje de la antena (Huidobro, 2013).

Figura 1.8 Antena parabólica

1.9 RET

Es un sistema de inclinación eléctrica remota que se utiliza principalmente para monitorear y controlar dispositivos de línea de antena (ALD) como la unidad de control remoto (RCU) y el amplificador montado en la torre (TMA). También se trata de un estándar producido por Antena Interface Standards Group para garantizar la compatibilidad e interoperabilidad de los dispositivos ALD y la infraestructura de control (Wensheng, Yi, & Yanming, 2010). El sistema puede aumentar la cobertura de la estación base de manera efectiva al tiempo que reduce la interferencia en la

región adyacente. RET también tiene ventajas como tiempo real, alta precisión y una amplia gama de ajustes. Para ello debe ajustarse la dirección de la antena en base a dos coordenadas, el ángulo de inclinación de la antena con respecto al eje z y el ángulo de azimut, que se corresponde al ángulo entre la antena y el eje x.

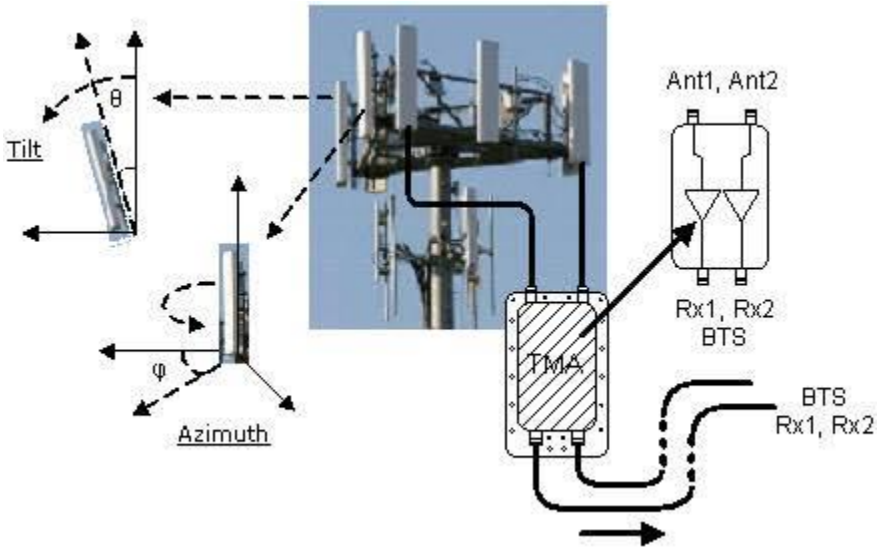

Figura 1.9 Sistema RET

1.10 Virtual Private Network (VPN)

Virtual Private Network (VPN) se trata de una conexión segura, confiable y lógica que se crea a través de una red pública como lo es el Internet. Permite a un administrador crear una red local entre múltiples equipos de cómputo en varios segmentos de red (Crist & Keijser, 2015). CISCO define una VPN como una conexión encriptada entre redes privadas a través de una red pública, destacando el hecho de que es una conexión virtual más allá del ámbito físico, extendiendo una red privada a través de una red pública o compartida (Laet & G.Schauwers, 2005). Permitiendo de esta forma que una computadora envíe y/o reciba datos de manera segura a través de una red pública o compartida, no importando si está directamente conectada a la red privada.

Algunos autores suelen compararla con un túnel que permite enlazar diferentes redes privadas a través de una conexión de internet. Siendo la imagen típica de una VPN la mostrada en la figura 1.10

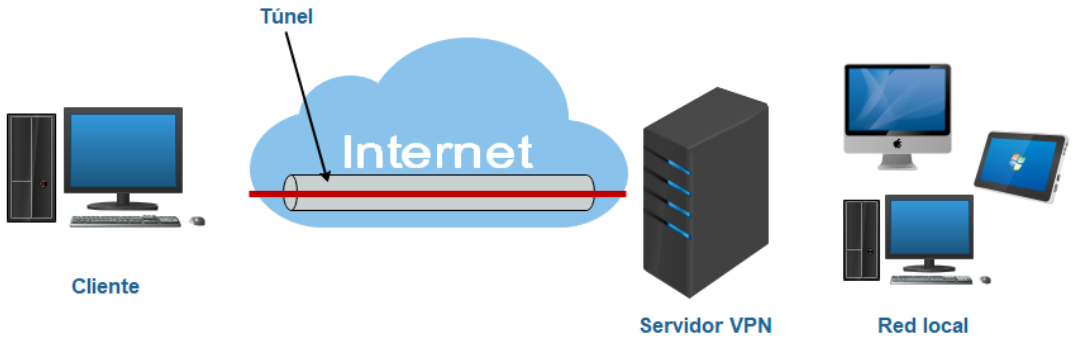

Figura 1.10 Diagrama de conexión VPN

Entre sus principales aplicaciones se pueden listar:

- Enlazar cajeros automáticos de forma segura a los sistemas bancarios.
- Navegar por Internet usando una red pública sin exponer información personal.
- Conectar oficinas remotas a grandes centros de datos, propiciando así el trabajo desde casa.
- Conexiones entre equipos mediante Wireless para un monitoreo a distancia.

1.11 Open VPN

Es una herramienta de conectividad basada en software libre que ofrece conectividad punto a punto con validación jerárquica de usuarios y host conectados remotamente (OpenVPN INC, 2020). Fue creado por James Yonan en el año 2002 con el objetivo de ofrecer una herramienta multiplataforma para simplificar la configuración de una conexión VPN, frente a otras más antiguas y difíciles como IPsec, haciéndola más accesible para gente inexperta en este tipo de tecnologías.

1.12 Raspberry Pi 3 B

Se trata de una computadora de placa reducida de bajo costo desarrollada en el Reino Unido por la fundación Raspberry Pi, presenta conectividad WiFi y Bluetooth. Este microordenador presenta las siguientes especificaciones:

- CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- RAM: 1GB LPDDR2 SDRAM
- Wi-Fi + Bluetooth: 2.4GHz y 5GHz IEEE 802.11.b/g/n/ac, Bluetooth 4.2, BLE
- Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps)
- GPIO de 40 pines
- Entrada HDMI
- 4 puertos USB 2.0
- Puerto CSI para conectar una cámara.
- Puerto DSI para conectar una pantalla táctil
- Salida de audio estéreo y vídeo compuesto
- Micro-SD
- Power-over-Ethernet (PoE)

Figura 1.11 Raspberry Pi 3 Modelo B

1.13 Sistema de nombres de dominio (DNS)

Es un servidor de hardware o software que implementa un servicio de red para proveer respuestas a las consultas en un servicio de directorio. Traduce un identificador basado en texto a una identificación numérica o componente de direccionamiento interno de sistema. Este servicio es realizado por el servidor en

respuesta a una petición de protocolo de servicio. La función más importante de los servidores DNS es la traducción (resolución) de los nombres de dominios y nombres de host identificables por los humanos (google.com, facebook.com, etc.), en sus direcciones numéricas del Protocolo de Internet (IP) correspondientes, el segundo principal espacio de nombres del Internet, que es usado para identificar y localizar a las computadoras y recursos en Internet (NO-IP, 2020).

1.14 NO-IP

Es un proveedor de DNS dinámico para servicios libres o de paga. No-IP ofrece servicios de DNS, correo electrónico, monitoreo de red y certificados SSL. El producto principal de No-IP son los servicios dinámicos de DNS ("DDNS"). Los servicios DNS dinámicos básicos que utilizan un dominio propiedad de No-IP son de uso gratuito siempre que la cuenta permanezca activa. Un servicio actualizado para usar un nombre de dominio propio cuesta alrededor de $ 25 dólares por año. Las direcciones IP dinámicas son comunes en las cuentas de banda ancha de cable residencial o DSL. El servicio gratuito permite a los usuarios configurar entre uno y tres nombres de host en un dominio proporcionado por No-IP. El nombre de host se resolverá en la dirección IP actual de la computadora de ese usuario. No-IP también proporciona un cliente de software para Windows, OS X y Linux que se puede ejecutar en la computadora que tiene la dirección dinámica (NO-IP, 2020).

Un nombre de host DNS dinámico está vinculado a la dirección IP dinámica del usuario. Cada vez que cambia la IP, un cliente DNS dinámico enviará una actualización a No-IP con la dirección IP actual y luego No-IP propaga el cambio de DNS a Internet en segundos (NO-IP, 2020).

Para facilitar las actualizaciones de direcciones IP, No-IP tiene un protocolo abierto que permite a los desarrolladores de software y fabricantes de hardware comunicarse a través de HTTP para notificarles sobre un cambio de dirección IP (NO-IP, 2020).

1.15 NodeMCU

El NodeMCU es una plataforma de IoT de código abierto basada en el ESP32, ofrece la posibilidad de realizar proyectos de domótica, control inteligente, y remoto a través de redes inalámbricas pudiendo funcionar como punto de acceso (AP) o estación de conexión (STA). Al ser de código abierto y tener una arquitectura similar a la de una tarjeta Arduino convencional, este dispositivo se puede programar utilizando el Arduino IDE haciendo solo unos pequeños ajustes al entorno de trabajo como añadir el kernel compatible de todas las tarjetas con firmware de NodeMCU ESP32, y las librerías pertinentes que, dicho sea de paso, son las mismas que poseen las tarjetas Arduino.

Ofrece la posibilidad de manejar eventos en una API para aplicaciones de red, lo cual facilita a los desarrolladores al hacer correr sus aplicaciones y código en un dispositivo tan compacto como este al trabajar en estilo Nodejs, posibilitando una mayor velocidad en el proceso de desarrollo de aplicaciones orientadas al IoT.

Además de todas las ventajas ya relatadas, es importante mencionar que este kit de desarrollo es de bajo costo lo que la convierte en una muy buena opción de desarrollo debido a las características que ofrece que lo convierten en un módulo potente y versátil, tomando en cuenta el tamaño tan pequeño que posee.

Este kit de desarrollo también dispone de varias de las características de hardware que posee una tarjeta de Arduino UNO, posee distintos puertos de entrada y salida de propósito general (GPIO), salidas PWM, un convertidor analógico digital, comunicación serial, entre otras características (NodeMcu Team, 2015).

1.16 Websockets

Websockets es una tecnología avanzada que hace posible abrir una sesión de comunicación interactiva entre el navegador del cliente y un servidor. Con esta API, se pueden enviar mensajes a un servidor y recibir respuestas controladas por eventos sin tener que consultar constantemente al servidor para una respuesta. Esta nueva forma de protocolo de comunicación posibilita la interacción en tiempo

real entre servidor y cliente, además que ahorra tiempo y recursos del servidor puesto que no tiene que abrir el canal de comunicación cada vez que se realiza una solicitud por parte del cliente o una respuesta proveniente del propio servidor, optimizando el tiempo y velocidad de respuesta y recursos de procesamiento. Por esta razón, es que este nuevo protocolo de interacción es ideal para aplicaciones web, pues éstas requieren de respuestas inmediatas en tiempo real donde los cambios acontecidos se deben hacer presentes tanto en el front-end como en el back-end. Esto no era posible utilizando el protocolo http, que requiere que la página web sea recargada ante cualquier cambio o evento generado

Para tener una conexión activa y bidireccional entre el servidor y el navegador, es necesario contar con los siguientes elementos:

WebSocket del cliente (navegador). Es quien establece la conexión inicial con el servidor.

WebSocket del servidor. Acepta las conexiones e inicia el intercambio de mensajes.

La conexión entre ambos se mantiene activa mientras la pestaña del navegador continúe abierta. Tanto el cliente como el servidor pueden cerrar la conexión en cualquier momento.

A pesar de constituir una de las mejoras agregadas en HTML5 en lo referente al cliente, para poder establecer la conexión y gestionar el envío y la recepción de mensajes es necesario programar los WebSockets con JavaScript (Mozila Developers Network, 2018).

1.17 JavaScript

JavaScript es un lenguaje de programación que permite realizar actividades complejas en una página web como mostrar actualizaciones de contenido en el momento, interactuar con mapas, animaciones gráficas 2D/3D, etc. Es la tercera

capa del pastel de los estándares en las tecnologías para la web, dos de las cuales son HTML y CSS.

HTML es un lenguaje de marcado que usa la estructura para dar un sentido al contenido web, definiendo párrafos, cabeceras, tablas, imágenes y videos en la página.

CSS es un lenguaje de reglas en cascada que se usa para aplicar estilo al contenido en HTML, por ejemplo, colocando colores de fondo, fuentes y marginando el contenido en múltiples columnas.

JavaScript: Es un lenguaje de programación que permite crear contenido nuevo y dinámico, controlar archivos de multimedia, crear imágenes animadas y muchas otras cosas más

La palabra dinámico se refiere a la habilidad para actualizar lo visualizado de una página/app para mostrar contenido diferente bajo diversas circunstancias, requiriendo nuevo contenido a generar. Como ejemplos están subir archivos desde una base de datos, donde el lado del cliente de JavaScript genera dinámicamente nuevo contenido dentro del navegador del cliente, por ejemplo, creando una nueva tabla en código HTML, insertando datos que son llamados desde un servidor propio, para luego visualizarlo en la tabla de la página web y mostrárselo al usuario. Una página web sin ninguna actualización de contenido dinámico es llamada como estática pues solo muestra el mismo contenido todo el tiempo.

JavaScript es aplicado a una página en HTML de una manera similar al CSS. Donde el CSS usa el elemento <link> para aplicar hojas de estilos externas y la etiqueta <style> es un elemento para aplicar hojas de estilos de manera interna al HTML, JavaScript solamente necesita de un solo componente en HTML, que es el elemento <script> (Mozilla Developers Network, 2015).

1.18 Archivos JSON

Con la creciente popularidad de los servicios Web, XML se ha convertido prácticamente de facto en el estándar para la transmisión de datos. Pero se necesita transmitir a través de Internet muchos más bytes de información para realizar una

tarea que se podría llevar a cabo con un flujo de información mucho más pequeño. Así se han desarrollado nuevas formas de compresión XML e incluso, nuevos formatos XML completos, tales como Binary XML (XML binario). Todas estas soluciones funcionan ampliando o añadiéndose a XML, conviniendo los aspectos de compatibilidad descendente en un asunto a tener en cuenta. Douglas Crockford, un experimentado ingeniero de software, propuso un nuevo formato de datos construido sobre JavaScript llamado JSON, JavaScript Object Notation (notación de objetos JavaScript). JSON es un formato de datos muy ligero basado en un subconjunto de la sintaxis de JavaScript: literales de matrices y objetos. Como usa la sintaxis JavaScript, las definiciones JSON pueden incluirse dentro de archivos JavaScript y acceder a ellas sin ningún análisis adicional como los necesarios con lenguajes basados en XML (Universidad de Alicante, 2008).

Literales de matriz

Estos elementos se especifican utilizando corchetes ([ ]) para encerrar listas de valores delimitados por comas de JavaScript (lo que puede significar cadenas, números, valores booleanos o valores null tales como:

```
var aNombre = ["Jaime", "Pepe", "Alfonso"];
alert (aNombre [0] ) ; //muestra "Jaime"
```

Literales del objeto

Las literales de objeto se utilizan para almacenar información en parejas nombre-valor para crear un objeto, Un literal de objeto se define mediante llaves ({ }). Dentro de estas, se pueden colocar cualquier número de parejas nombre-valor, definida mediante una cadena, un símbolo de dos puntos y el valor. Cada pareja nombre-valor deben estar separadas por coma (Universidad de Alicante, 2008).

```
var oPersona = {"nombre":"Robert", "edad":30, "hijos":true };
alert(oPersona.nombre);
```

Sintaxis JSON

La sintaxis de JSON realmente no es nada más que la mezcla de literales de objeto y matrices para almacenar datos. JSON representa solamente datos, no incluye el concepto de variables, asignaciones o igualdades (Universidad de Alicante, 2008).

Este código:

```
var oPersona3 =
[{"nombre":"Robert", "edad":30, "hijos":["Jaime","Pepe","Alfonso"] },
{"nombre":"Maria", "edad":36, "hijos":["Hijo Maria","Hijo2 Maria"] }];
```

Quedaría:

```
[{"nombre":"Robert", "edad":30, "hijos":["Jaime","Pepe","Alfonso"] },
{"nombre":"Maria", "edad":36, "hijos":["Hijo Maria","Hijo2 Maria"] }]
```

Si se recupera esta información y se almacena en una variable 'sJSON', se dispondrá de una cadena de información, no de un objeto. Para transformarla en un objeto, se utiliza la función eval ():

```
var sJSON =
'[{"nombre":"Robert", "edad":30, "hijos":["Jaime","Pepe","Alfonso"] },
{"nombre":"Maria", "edad":36, "hijos":["Hijo Maria","Hijo2 Maria"] }]';

var oPersona4    = eval("(" + sJSON + ")");

alert(oPersona4[1].edad);
```

1.19 API REST

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad, pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST (BBVA, 2016).

Características acordes con la instrucción proporcionada por BBVA, 2016:

- Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla.
- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).
- Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.
- Interfaz uniforme: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
- Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
- Visibilidad, fiabilidad y escalabilidad. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.
- La API REST siempre es independiente del tipo de plataformas o lenguajes: siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es

que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

1.20 SolidWorks

Es un software CAD para modelado mecánico en 2D y 3D, desarrollado en la actualidad por SolidWorks Corp., una filial de Dassault Systèmes, S.A. Este programa permite modelar piezas y conjuntos, además de extraer de ellos tanto planos técnicos como otro tipo de información necesaria para la producción, como la obtenida de la simulación y los estudios de movimiento (SolidWorks, 2020). De esta manera, SolidWorks proporciona una gran variedad de herramientas, necesarias para abordar los problemas más complejos del diseño mecánico y de manufactura, con el suficiente nivel de detalle para conseguir el mejor resultado para la fabricación del producto final. En este software se puede realizar la simulación de todo el diseño mecánico de un producto para detectar los posibles problemas y corregirlos.

Figura 1.12 Logo de Solidworks

1.21 Manufactura Aditiva

Es el nombre técnico que engloba todas las tecnologías de impresión 3D, se trata de la fabricación de objetos tridimensionales por aportación de material en vez de sustracción, como es el caso de tornos y fresadoras CNC. En impresión 3D, partiendo de un archivo digital (modelo 3D) realizado en algún software CAD de diseño mecánico, se utilizan diferentes procesos aditivos en los que se aplican

capas sucesivas de material para crear un objeto tangible (Ortega, 2017). Una impresora 3D es la encargada de efectuar la labor de manufactura, para ello necesita de un archivo en formato STL y algunas configuraciones previas concernientes a la velocidad de trabajo, la temperatura de operación del extrusor y la cantidad de relleno de la pieza para comenzar con el proceso, por lo regular los fabricantes de impresoras 3D proporcionan los programas necesarios para realizar dicha configuración, necesitando solamente el archivo STL del objeto a fabricar.

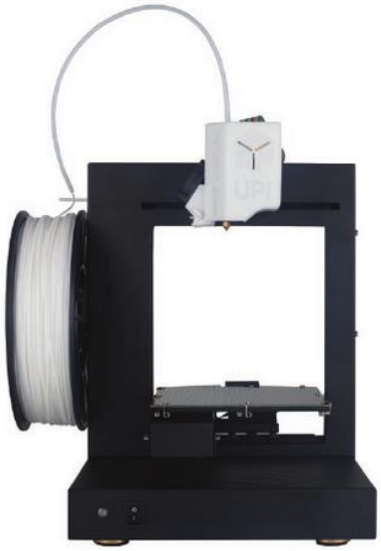

Figura 1.13 Ejemplo de impresora 3D

1.22 Formato STL

Una vez se realiza el diseño de una pieza en cualquiera de los diferentes programas CAD existentes (AutoCAD, SolidWorks, Catia, etc.) se dispondrá de un archivo con un formato específico según el software de diseño utilizado. Dicho archivo se debe exportar a formato STL el cual es un estándar para los diferentes programas de fabricación 3D.

El formato STL fue desarrollado por 3D Systems como formato nativo para el proceso de fabricación aditiva mediante estereolitografía, aunque se ha implementado como el formato estándar a todas las tecnologías de impresión 3D (Ortega, 2017). STL significa Standard Tesselation Language – o lenguaje estándar

de teselado. Este formato de representación describe únicamente la superficie de un objeto tridimensional, no soporta otros muchos atributos comunes al diseño CAD como pueden ser el color, el mapeado de texturas en la superficie, densidades de estructuras internas, etc. El STL utiliza una combinación de triangulación y subdivisión para describir superficies. Es decir, cualquier modelo guardado en formato STL está compuesto exclusivamente de polígonos triangulares.

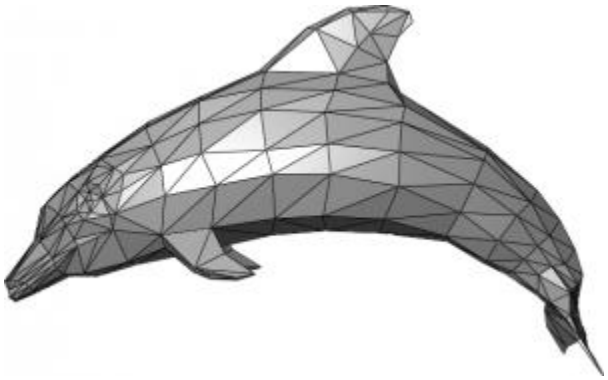

Figura 1.14 Pieza en formato STL

CAPÍTULO II: PROCEDIMIENTO

2.1 Metodología

En base al objetivo general y a los objetivos específicos del proyecto, se realizó el diagrama presentado en la figura 2.1. En dicho esquema se trató de representar la estructura técnica de la cual se compone el proyecto. La forma correcta de interpretar este diagrama es entender que en el extremo izquierdo del mismo se ubica la capa del usuario, mientras que en el extremo derecho está lo que se quiere controlar con el sistema presentado en esta tesis, es decir la orientación de una antena de telecomunicaciones. Lo que une a ambos extremos es la capa máquina o capa técnica que como se verá más adelante, involucra diversas disciplinas como la manufactura, la electrónica, el control, redes y programación web. En otras palabras, es en esta capa intermedia donde operan los componentes que hacen que la funcionalidad y la operatividad del sistema sea posible.

En esta capa intermedia descrita en el párrafo anterior, se tiene en primer lugar una Raspberry Pi que tiene la tarea de crear un servidor VPN que permita a equipos de cómputo conectados a Internet u otras redes públicas, acceder a la propia red local en la que se encuentra la Raspberry si poseen estos las credenciales y/o permisos necesarios. En esta red local además de estar conectada la Raspberry, también se halla un microcontrolador NodeMCU que crea un servidor web en el que se ejecuta cierta aplicación que permite al usuario tomar el control de las magnitudes 'Azimut' y 'Tilt' de la antena por medio de su interfaz. En base a la información que le llega al servidor por parte del cliente al interactuar con la aplicación antes mencionada, utilizando el protocolo de comunicación I2C, el primero es capaz de transmitir estos datos al driver controlador de los servomotores encargados de realizar cada movimiento. La energía necesaria para operar los motores se obtiene de una fuente conmutada de 5 V y 3 A. Cada servomotor se conecta a las salidas del driver y de esta manera una persona conectada a internet puede acceder a la red local por medio del servidor VPN creado y configurado en la Raspberry Pi, sin embargo, como ya se indicó líneas atrás para ello necesita de las credenciales necesarias, por lo

que se tiene una operación a distancia del sistema de posicionamiento y además de una forma segura.

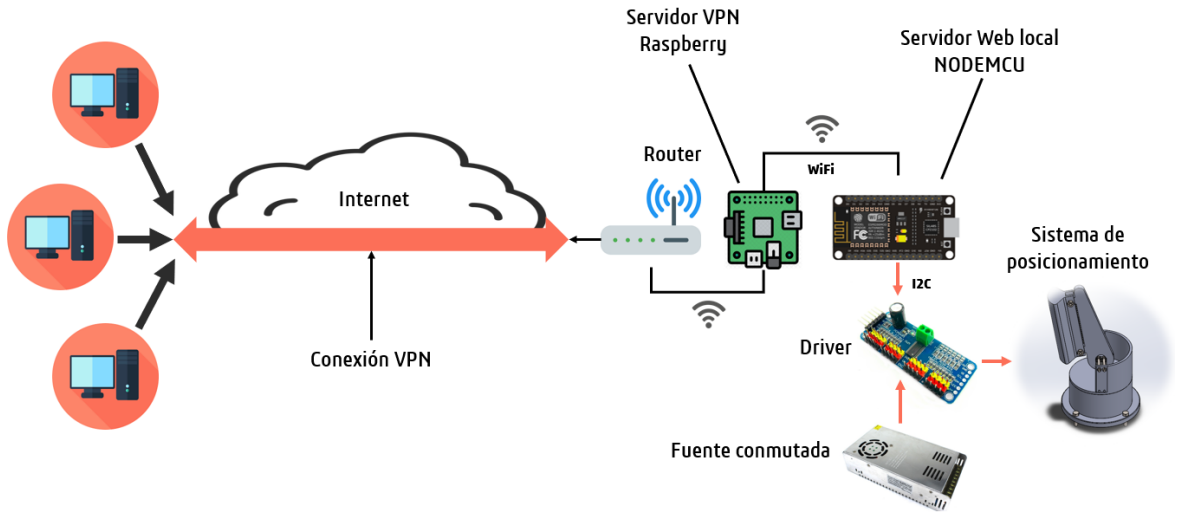

Figura 2.1 Diagrama esquemático del prototipo del sistema

2.2 Actividades

Este apartado será breve pues solo se quiere dejar en claro las actividades realizadas para concretar el proyecto, al mencionarlos de manera superficial en el orden en que fueron efectuadas, sin embargo, dichas actividades se describen con un mayor detalle técnico en los siguientes apartados del presente capítulo.

2.2.1 Planteamiento de la solución y selección de componentes

En esta actividad se definió la estructura del proyecto, en el sentido de qué tecnología, herramientas y componentes se utilizarían para concretarlo, cumpliendo con los objetivos propuestos en base a la justificación y planteamiento del problema dados. De esta actividad fue que surgió el diagrama esquemático del prototipo del sistema, presentado en la Figura 2.1 en esta misma página, con lo que se puede decir que la metodología del proyecto también fue definida en esta etapa.

2.2.2 Diseño del modelo/maqueta de la antena del sistema

Basándose en la guía elaborada en la actividad interior, y teniendo la ruta clara de lo que el proyecto requiere para llevarse a cabo, se comenzó a diseñar en

Solidworks la estructura mecánica de una maqueta que representa a una antena de telecomunicaciones. Es esta maqueta o modelo sobre la cual se implementó el sistema propuesto en el presente documento.

2.2.3 Programación de la aplicación web, y creación y configuración del servidor web/local

Se programó una aplicación web para ser ejecutada o 'servida' por la tarjeta Node MCU, la cual funge como servidor del sistema, por lo que se tuvo que hacer su debida configuración y programación tomando en cuenta su condición de servidor. De esta manera, cada vez que un usuario/cliente desee hacer uso de la aplicación web para poder controlar remotamente las magnitudes 'Azimut' y 'Tilt' de la antena, solo debe conectarse a la red del servidor e introducir la dirección IP del mismo en el navegador de su preferencia. De esta manera se conectará con la aplicación web y el servidor estará a la escucha de sus peticiones

2.2.4 Creación y configuración de una conexión túnel VPN usando una Raspberry Pi 3.0

En una Raspberry PI 3.0 se creó un servidor VPN para enrutar la conexión de los clientes que quieran acceder a la red privada local del servidor Node MCU desde una red pública, como lo puede ser la internet. El servidor VPN se configuró de tal manera que solo permita acceso al túnel de conexión a la red privada del Node MCU si el cliente en cuestión posee las credenciales pertinentes, con ello se garantiza que no exista intrusión de un usuario no autorizado para hacer uso de la aplicación web

2.2.5 Programación de un Arduino UNO para ser utilizado como el drive que controla los servomotores de la maqueta de la antena

Se utilizó una tarjeta Arduino UNO para que cumpla la función de controlador de servomotores. Dicho Arduino recibe los comandos por parte del servidor Node MCU utilizando el protocolo de comunicación serial I2C.

2.2.6 Fabricación de las piezas de la maqueta por adición de material caliente, y ensamblaje de la misma

Se imprimieron las piezas correspondientes a la maqueta de la antena en una impresora 3D y posteriormente se ensamblaron siguiendo las propias indicaciones del diseño realizado, colocando al mismo tiempo los servomotores en su respectivo sitio. También se cablearon los actuadores al drive controlador, y a su fuente de energía.

2.2.7 Puesta en marcha del servidor local y por consiguiente de la aplicación web y el servidor VPN

Esta fue la última actividad antes de iniciar con las pruebas pertinentes del sistema. En esta etapa puesto que la programación de la aplicación web y la configuración de los servidores web local y VPN ya fue realizada, solo se debe poner en marcha el hardware correspondiente que soporta a los servidores. Es decir, se energizaron tanto la Raspberry como el NodeMCU, y se hicieron las conexiones respectivas a la red.

2.3 Lista de materiales

A continuación, se muestra una lista del material requerido para la elaboración del sistema de acuerdo con la arquitectura del mismo. Posterior a ello se describe brevemente la justificación de la elección de cada componente y la función que desempeña en el prototipo. Hay que aclarar que en la presente lista no se tomaron en cuenta herramientas ni equipo de desarrollo a excepción de aquellos que no se tuvieran, como la Raspberry o el Node MCU, de allí en fuera, equipo de desarrollo como la impresora 3D, o la computadora en donde se desarrolló el software y el diseño mecánico no fueron tomados en cuenta en esta lista

- Servomotor MG995 con disco metálico
- Fuente conmutada de 5V 3ª
- Paquete de cables jumper 20 cm H-M
- Rollo de filamento PLA
- Tornillería
- Rasberry Pi 3.0
- Fuente de alimentación Raspberry Pi 3
- Micro SD 32 GB
- NodeMCU ESP 8266

A continuación, se realiza una justificación del material a utilizar, resaltando su importancia en la elaboración del prototipo.

El servomotor MG995 tiene un rango de operación de entre 4.8 V y 6.6 V, un torque máximo de 11 Kg-cm y un ángulo de rotación de 180°, es ideal para proyectos donde se desean movimientos precisos. Estos motores permiten al sistema de posicionamiento moverse en sus diferentes ángulos de interés, es así que se necesita uno de estos para ajustar el azimut de la antena y dos más para poder ajustar la elevación (Tilt), se plantea una unidad extra de este modelo de servomotor para tenerlo como refacción. Para poder controlar estos motores se necesita de un driver, el cual proporcione la magnitud de voltaje y corriente requerida, así como la señal de control, en este caso se contempla la utilización de una tarjeta Arduino

UNO a modo de driver. Para poder utilizar el driver con algún microcontrolador se necesita de un protocolo de comunicación en serie tipo I2C. El driver necesita de una fuente externa que permita suministrar de energía a los motores, para ello se contempla utilizar una fuente conmutada con una salida de 5 V y corriente de 3 A.

Se tiene contemplado utilizar un microcontrolador NodeMCU para poder crear un servidor web en el que se ejecutará una aplicación web desde donde se podrán modificar las magnitudes ‘tilt’ y ‘azimut’ del modelo de la antena. Dicho servidor estará dentro de una red privada conectada a un túnel VPN configurado y creado en la Raspberry PI, para que así cualquier cliente con las credenciales requeridas pueda acceder a la aplicación web desde cualquier red pública a través del servidor VPN. El NodeMCU, además de ser el servidor que aloja a la aplicación web, también es un microcontrolador, así pues, cuando el cliente realiza una petición al servidor a través de la aplicación, el NodeMCU procesará dicha petición y será capaz de comunicarse con el driver por medio del protocolo I2C para así enviar las señales de control requeridas por los motores, en base a la solicitud realizada por el cliente

Los cables jumper son utilizados para realizar las conexiones correspondientes entre los servomotores, el driver, la fuente y el NodeMCU. Estos son del tipo Hembra-Macho debido a que la conexión de los servomotores es del tipo macho y las terminales del NodeMCU son del tipo hembra. La fuente de alimentación tiene unas borneras en las cuales se pueden conectar cables tipo macho para alimentar a los motores.

El rollo de filamento de material PLA, es utilizado para realizar la impresión 3D de todas las partes del prototipo. Los tornillos especificados son utilizados para realizar el ensamble de las partes del prototipo y los servomotores son del tipo M4, M3 y ¼ de pulgada.

Finalmente se tiene la Raspberry, el cual es el dispositivo que se encargará de crear un servidor VPN para que desde una red externa sea posible acceder al servidor web generado con el NodeMCU en cierta red privada y de esta manera se pueda modificar la posición de la antena de forma remota desde cualquier lugar con acceso

a internet. La fuente de alimentación para la Raspberry, así como la tarjeta SD son componentes esenciales para su correcto funcionamiento, pues la primera suministra la energía necesaria por el miniordenador y la segunda es su medio de almacenamiento para el sistema operativo.

2.4 Software

2.4.1 Implementación de un servidor web local a través de un NodeMCU

En la Figura 2.2, se muestra una captura del código del sketch principal, como se puede observar, el código de este apartado está muy simplificado, esto es gracias a que la mayoría de la programación se hizo en ficheros separados en función de la acción que realizan cada uno de ellos. Estos ficheros y dependencias se incluyeron simplemente en la cabecera del programa principal.

```
#include <ESP8266WiFi.h>
#include <ESPAsyncWebServer.h>
#include <FS.h>
#include <WebSocketsServer.h>
#include <ArduinoJson.h>
#include <SoftwareSerial.h>
SoftwareSerial P1(D3,D4);
#include "config.h"
#include "API.hpp"
#include "WebSockets.hpp"
#include "set_connection.hpp"
#include "server.hpp"
#include "set_AWS.hpp"


void setup(void)
{
  Serial.begin(115200);
  P1.begin(115200);
  SPIFFS.begin();

  Connect_STA(true);

  InitServer();
  InitWebSockets();


}
void loop(void)
{
```

Figura 2.2 Contenido del programa principal

Las primeras siete dependencias incluidas son ficheros 'header' correspondientes a las librerías usadas en el proyecto. Estas librerías posibilitan la conexión 'Wi Fi' por parte del NodeMCU, permiten crear un servidor asíncrono con uso de 'web sockets' para la comunicación cliente/servidor sin interrupciones, crear y gestionar archivos Json y facilitar el manejo de ficheros tipo JavaScript y html en formato comprimido desde la memoria flash del dispositivo NodeMCU en lugar de hacerlo por medio de la memoria dinámica del mismo cada vez que el cliente solicite hacer uso de estos.

Enseguida de la inclusión de las librerías se incluyeron las dependencias escritas específicamente para este programa, aunque claro con ciertas modificaciones pueden ser reutilizadas para alguna aplicación diferente. La primera de ellas fue 'config.h', la cual se muestra en la Figura 2.3

```
const char* ssid ="INFINITUM1843";
const char* password = "1378FettGG";
```

Figura 2.3 Dependencia 'config.h'

Como se puede ver, simplemente contiene la información del punto de acceso al que se desea conectar el NodeMCU y donde, por tanto, se hará el servidor local. Estos datos pueden ser modificados en cualquier momento de ser necesario.

El fichero 'set_connection.hpp' al igual que el anterior mostrado (Figura 2.3), es de cuestión crítica en el programa, pues en este se incluye el 'setup' y preferencias de la conexión a la red, se puede observar el código en la siguiente captura (Figura 2.4).

```
IPAddress ip(192,168,1,200);
IPAddress gateway(192,168,1,254);
IPAddress subnet(255,255,255,0);
void Connect_STA(bool useStaticIP = false)
{
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid,password);
  if(useStaticIP) WiFi.config(ip,gateway,subnet);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(100);
  }
}

void ConnectWiFi_AP(bool useStaticIP = false)
{
  WiFi.mode(WIFI_AP);
  while(!WiFi.softAP(ssid, password))
  {
    delay(100);
  }
  if(useStaticIP) WiFi.softAPConfig(ip, gateway, subnet);
}
```

Figura 2.4 Dependencia 'set_connection.hpp'

En el código de la anterior imagen, se tienen dos funciones en 'set_connection.h', una para usar el NodeMCU en modo AP y otra en STA, el que interesa en este proyecto es el modo STA. La función 'Connect_STA' requiere un argumento de tipo booleano, si este se coloca como 'False', el punto de acceso proporcionará una diferente dirección IP cada vez que el NodeMCU se conecte a él. En el caso que se coloque como 'true', se usará una ip estática definida por el desarrollador, así como un 'subnet' preestablecido y un 'gateway' correspondiente al punto de conexión. Esta última opción es la que interesa, pues es preferible tener una única dirección ip constante para conectarse desde una red externa, por esta razón en el programa principal (Figura 2.2) dentro del 'setup' se colocó el argumento 'true' dentro de la función 'Connect_STA'.

El fichero 'server.hpp' también es básico en aplicaciones como esta, pues es en esta dependencia donde se configuraron los detalles de inicialización del servidor local. Si se observa la Figura 2.5, se puede ver que se define el puerto 80 para abrir un servidor asíncrono, es decir que admita múltiples clientes simultáneamente. En

la función 'InitServer' se especificó que cada vez que un cliente nuevo ingrese a la red, el servidor entregue a través del navegador el archivo 'index.html' es decir una página web para que el cliente interactúe con el servidor a través de ella. La realización de esta página se comentará más adelante.

```
AsyncWebServer server(80);

void InitServer()
{
  server.serveStatic("/", SPIFFS, "/").setDefaultFile("index.html");

  server.onNotFound([](AsyncWebServerRequest *request) {
     request->send(400, "text/plain", "Not found");
  });

  server.begin();
  //Serial.println("HTTP server started");
}
```

Figura 2.5 Dependencia 'server.hpp'

En caso de que el cliente realice una solicitud de información que el servidor no pueda proporcionar o responder, se enviará un código de error http 404 con el texto 'Not found'.

Finalmente, se inicializa el servidor utilizando una función proporcionada por la librería 'ESPAsynWebServer'. En el programa principal (Figura 2.2) simplemente se llamó a la función 'InitServer' después de haber realizado la conexión STA con el punto de acceso seleccionado.

2.4.2 Implementación de web sockets con archivos Json en el servidor web local

Se dijo que se usaron Web Sockets para el envío de datos cliente/servidor y servidor/cliente, esto con el fin de evitar el uso de formularios, el manejo de 'endpoints' y de recargar la página cada vez que se hiciera una petición. Para ello se usó la librería 'WebSocketsServer', y una vez que se incluyó ésta en el sketch principal, se creó la dependencia 'set_AWS' (Figura 2.6 y 2.7) para establecer las

configuraciones y funciones a usar para el manejo de web sockets en el servidor asíncrono.

```
void onWsEvent(AsyncWebSocket * server, AsyncWebSocketClient * client, AwsEventType type, void * arg, uint8_t *data, size_t len){
  if(type == WS_EVT_CONNECT){
    //Serial.printf("ws[%s][%u] connect\n", server->url(), client->id());
    client->printf("Hello Client %u :)", client->id());
    client->ping();
  } else if(type == WS_EVT_DISCONNECT){
    //Serial.printf("ws[%s][%u] disconnect: %u\n", server->url(), client->id());
  } else if(type == WS_EVT_ERROR){
    //Serial.printf("ws[%s][%u] error(%u): %s\n", server->url(), client->id(), *((uint16_t*)arg), (char*)data);
  } else if(type == WS_EVT_PONG){
    //Serial.printf("ws[%s][%u] pong[%u]: %s\n", server->url(), client->id(), len, (len)?(char*)data:"");
  } else if(type == WS_EVT_DATA){
    AwsFrameInfo * info = (AwsFrameInfo*)arg;
    String msg = "";
    if(info->final && info->index == 0 && info->len == len){
      if(info->opcode == WS_TEXT){
        for(size_t i=0; i < info->len; i++) {
          msg += (char) data[i];
        }
      } else {
        char buff[3];
        for(size_t i=0; i < info->len; i++) {
          sprintf(buff, "%02x ", (uint8_t) data[i]);
          msg += buff ;
        }
```

Figura 2.6 Dependencia 'set_AWS.hpp' parte 1

```
            if(info->opcode == WS_TEXT)
            ProcessRequest(client, msg);

         } else {
            //message is comprised of multiple frames or the frame is split into multiple packets
            if(info->opcode == WS_TEXT){
               for(size_t i=0; i < len; i++) {
                  msg += (char) data[i];
               }
            } else {
               char buff[3];
               for(size_t i=0; i < len; i++) {
                  sprintf(buff, "%02x ", (uint8_t) data[i]);
                  msg += buff ;
               }
            }
            Serial.printf("%s\n",msg.c_str());

            if((info->index + len) == info->len){
               if(info->final){
                  if(info->message_opcode == WS_TEXT)
                  ProcessRequest(client, msg);
               }
            }
         }
      }
   }

   void InitWebSockets()
   {
      ws.onEvent(onWsEvent);
      server.addHandler(&ws);
     // Serial.println("WebSocket server started");
   }
```

Figura 2.7 Dependencia 'set_AWS.hpp' parte 2

Varias partes del código de esta dependencia se usaron únicamente para indicar por medio de mensajes en la terminal serial del Arduino IDE, los procesos y estados de conexión o error y pérdida de conexión que ocurrieran, así como la conexión al servidor de nuevos clientes, sin embargo, el no imprimir esta información no afecta la funcionalidad del proyecto por lo que una vez que se tuvo la certeza de que todo funciona como debiese, se colocaron en comentarios esas partes del código. En cuanto al resto del código, éste se encarga de 'escuchar' las peticiones del cliente, y también de devolver las respuestas del servidor apoyándose para ello en un fichero JavaScript llamado 'main.js' del cual se hablará más adelante.

Esta dependencia (set_AWS) a diferencia de las demás mostradas, no se recomienda su modificación, puesto que incluye los procedimientos necesarios para el posterior procesamiento de las peticiones del cliente, este procedimiento siempre será el mismo pues este código no se encarga de procesar las peticiones como ya se indicó, solo las recibe y envía las respuestas del servidor. Así pues, este código tal cual como está es completamente reutilizable en otros proyectos.

Ya que se mencionó el procesamiento de las peticiones del cliente, es momento de mencionar al fichero ‘WebSockets.hpp’, el cual se muestra en la Figura 2.8.

```
AsyncWebSocket ws("/ws");

void ProcessRequest(AsyncWebSocketClient * client, String request){
  //Serial.println(request);
  StaticJsonDocument<200> doc;
  DeserializationError error = deserializeJson(doc, request);
  if (error) {return;}

  String command = doc["command"];
  if(command == "setEje")
    setEje(doc["id"], (bool)doc["status"]);
  else if(command == "setAngle")
    setAngle(doc["id"], (int)doc["angle"]);
  else if(command == "setDefault")
    setDefault(doc["id"]);
}
void updateState(String input, bool value)
{
  //Alguna actualizacion del servidor al cliente deberá ser escrita aqui
}
```

Figura 2.8 Dependencia ‘WebSockets.hpp’

Se creó el objeto ‘ws’ referente a la clase de Web Sockets asíncronos, y se definió la función ‘ProcessRequest’, los argumentos de esta función son el identificador del cliente en conjunto con su petición. Estos argumentos los proporciona la dependencia set_AWS.hpp (Figura 2.7) una vez que recibió la petición.

Dentro de la función ‘ProcessRequest’ se toma el archivo Json enviado por ‘main.js’ acorde a la petición del cliente. Se usan archivos Json puesto que es una forma sencilla y muy eficiente de estandarizar la comunicación cliente/servidor, ya que en

caso que se envíe un archivo Json del cliente al servidor, este es convertido a una cadena antes de enviarlo, posteriormente como se ve en las primeras dos líneas de código, dentro de la función 'ProcessRequest' dicha cadena es 'des-serializada' para su procesamiento, este procesamiento viene dado a través de una serie de pruebas 'if' que tienen el propósito de identificar el tipo de petición recibida por medio del comando contenido dentro de estos archivos Json. Para esta aplicación se tienen tres comandos diferentes; 'setEje', 'setAngle' y 'setDefault', dependiendo de este es la función que se ejecutará, colocando de argumentos para los primeros dos casos, el 'id' y el dato ligado a ese identificador, mientras que para el último comando solo se solicita el argumento 'id'. Hasta abajo se puede ver una función con el nombre de 'updateState', esta no hace nada en este proyecto en particular, pero puede escribirse código en ella para enviar información del servidor al cliente, sin necesidad de que el cliente haga una solicitud propiamente, es decir puede usarse para proporcionarle al usuario información o actualizaciones de estado.

2.4.3 Implementación de una API rest en el servidor web local

Volviendo al tema de los comandos tomados del archivo Json, las funciones que relacionan a cada comando se escribieron en un fichero distinto llamado 'API.hpp' (Figura 2.10), es decir que todas las peticiones del cliente son identificadas, procesadas y atendidas, sobre esta API REST, usando ficheros Json como ya se ha repetido en varias ocasiones. Con lo anterior, cada petición 'http' puede ser traducida y comprendida con palabras simples relacionadas con la tarea a realizar, por ejemplo, el comando setEje ejecuta la función del mismo nombre, la cual se encarga de activar o desactivar la opción de manipular el movimiento del eje seleccionado del sistema de reorientación de la antena, es decir si se trata del azimut o del tilt, mientras que la función 'setAngle' que se ejecuta bajo el comando del mismo nombre, cumple la tarea de reposicionar cada magnitud según el ángulo especificado por el usuario en la página web. Por último, la función 'setDefault' coloca la posición inicial a cada magnitud, que es de 80 grados para el eje azimut y 90 grados para el eje tilt (siempre y cuando la opción de poder cambiar la posición de ambos esté habilitada).

```
void setEje(String id, bool state)
{
   if(id == "I1"){
      P1.print("As");
      P1.println(state);
   } else {
   P1.print("Ts");
   P1.println(state);
   }
}

void setAngle(String id, int angle)
{
   if(id == "1"){
      P1.print("Aa");
      P1.println(angle);
   } else {
   P1.print("Ta");
   P1.println(angle);
   }
}

void setDefault(String id)
{
   P1.println("90");
}
```

Figura 2.9 Dependencia 'API.hpp'

Al prestar atención a la Figura 2.9 que muestra el código de la API del proyecto, se notará cómo cada función, envía usando un puerto serial virtual definido desde el sketch principal como 'P1' para los pines D3 y D4 para RX y TX respectivamente, cierta información en una cadena de texto, la cadena al menos en las operaciones de 'setEje' y 'setAngle' se conforma de dos partes, la primera es un identificador de dos caracteres, y la segunda parte es el dato que representa los valores a querer ser modificados por el usuario para cada uno de los movimientos de la antena, para la tercera función de 'setDefault' solo se envía un numero constante. Esto es importante puesto que el driver (en este caso el Arduino UNO Board) que controlará los servomotores de la antena necesita saber qué movimiento va a ser modificado (Azimut o Tilt), así como de qué tipo de modificación se trata, es decir si se variará su posición o su estado de encendido/apagado tras cada petición del usuario. Así, el servidor al reconocer que es lo que está pidiendo el cliente, enviará la cadena de

texto correspondiente a través de la comunicación serial a la tarjeta Arduino UNO haciendo uso del modelo 'maestro-esclavo'. Debido a lo anterior dicho, es claro que la tarjeta Arduino que hace el papel de esclavo y driver de los actuadores de la antena, cuenta con su propio programa, esto no será explicado en este apartado, sin embargo, si lo será en el siguiente referente al armado e implementación del sistema.

2.4.4 Creación de la aplicación web que se servirá desde el servidor

Aprovechando que ya se ha hecho mención sobre la página web de manera más frecuente, es tiempo de mostrar como luce visualmente la interfaz gráfica, así como el código html de la misma. En la Figura 2.10, se muestra la interfaz de la aplicación web, mientras que en la Figura 2.11 está el código html que le corresponde

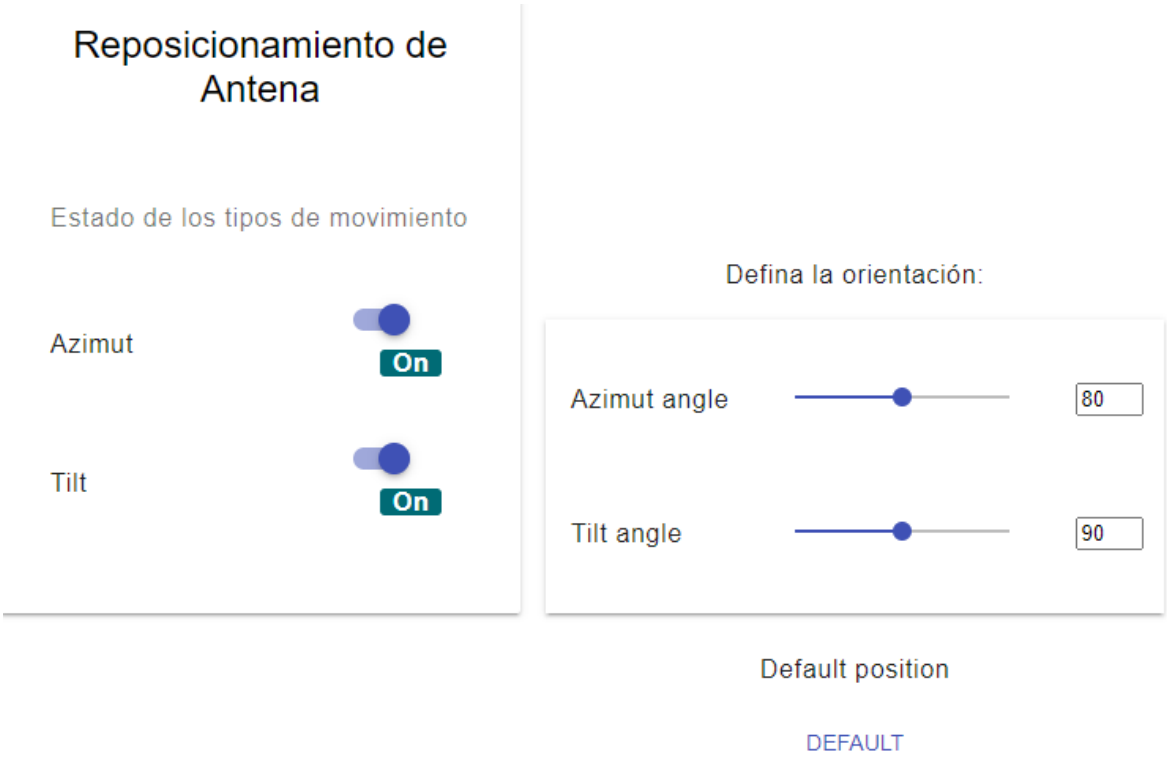

Figura 2.10 Interfaz de la página web

Como se puede observar, la interfaz es muy sencilla pero práctica, en el lado izquierdo se tiene la opción de habilitar o deshabilitar la opción de poder variar la posición de los motores, esto quiere decir que en caso de que, por ejemplo, el

‘switch’ del eje Azimut se encuentre apagado, no se podrá modificar la orientación del mismo, aun cuando el ‘slider’ correspondiente en el lado derecho sea manipulado.

Es importante indicar que el ‘slider’ del eje Azimut tiene un rango de 0° a 160°, mientras que el eje Tilt tiene un rango de 55° a 125°. En ambos ejes, los ‘sliders’ cuentan con una posición por defecto de 80° para el Azimut y 90° para el Tilt, este mismo valor se establecerá en ellos y por tanto en los ejes, si se presiona el botón inferior con la leyenda ‘Default’.

Se puede percatar cómo las tres acciones descritas; habilitar ejes, reorientar ejes y colocar los ejes a la posición por defecto, están relacionadas respectivamente con los comandos que soporta y procesa la API REST como se dijo páginas atrás, los cuales son; ‘setEje’, ‘setAngle’ y ‘setDefault’.

```
<!DOCTYPE html>
<html class="no-js" lang="">

<head>
    <title>Control de Antena</title>
    <meta charset="utf-8">
    <meta http-equiv="x-ua-compatible" content="ie=edge">

    <meta name="description" content="">
    <meta name="viewport" content="width=device-width, initial-scale=1">
</head>

<body>
    <link rel="stylesheet" href="css/main.css">
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
   <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/material-design-lite@1.3.0/dist/material.indigo-pink.min.css">
   <div id="bloque1">
     <div class="mdl-card mdl-shadow--2dp">
        <div class="mdl-card__title mdl-card--expand">
            <h2 class="mdl-card__title-text">Reposicionamiento de Antena</h2>
        </div>

        <div class="mdl-card__supporting-text">
            <h6>Estado de los tipos de movimiento</h6>
            <ul class="mdl-list">
                <li class="mdl-list__item">
                    <span class="mdl-list__item-primary-content">Azimut</span>
                    <span class="mdl-list__item-secondary-action">
                        <label class="mdl-switch mdl-js-switch mdl-js-ripple-effect">
                            <input id="state-azimut-I1" data-id="I1" type="checkbox" class="mdl-switch__input" checked
                            onchange="setEje(this.dataset.id, this.checked)"/>
                        </label>
                        <label id="eje-indicator-I1" class="label-big On-style">    On</label>
                        <!-- Nueva linea insertada debajo -->
                    </span>
                </li>
```

Figura 2.11 a) Código de la página web ‘index.html’ parte 1

```
            <li class="mdl-list__item">
                <span class="mdl-list__item-primary-content">Tilt</span>
                <span class="mdl-list__item-secondary-action">
                    <label class="mdl-switch mdl-js-switch mdl-js-ripple-effect">
                        <input id="state-tilt-I2" data-id="I2" type="checkbox" class="mdl-switch__input" checked
                        onchange="setEje(this.dataset.id, this.checked)"/>
                    </label>
                    <span class="mdl-list__item-secondary-action">
                        <label id="eje-indicator-I2" class="label-big On-style">On</label>
                    </span>
                </span>
            </li>
        </ul>
    </div>
 </div>
</div>
 <!-- Posicion por defecto -->
<div id="bloque2">
 <div class="mdl-list__item " id="">
    <div>
        <!-- probando -->
            <h6>Defina la orientación:</h6>
            <ul class="mdl-list mdl-shadow--2dp">
                <li class="mdl-list__item">
                    <span class="mdl-list__item-primary-content">Azimut angle</span>
                    <span class="mdl-list__item-secondary-action">
                        <div class="mdl-grid">
                            <div class="mdl-cell mdl-cell--10-col">
                                <input id="slider-angle-1" data-id="1" class="mdl-slider mdl-js-slider"
                                    type="range" min="0" max="160" value="80"
                                    onchange="setAngle(this.dataset.id, this.value);" >
                            </div>
                            <div class="mdl-cell mdl-cell--2-col">
                                <input id="slider-text-angle-1" data-id="1" style="width:35px;"
                                onchange="setAngle(this.dataset.id, this.value);" value="80"></input>
                            </div>
                        </div>
                    </span>
                </li>

          <li class="mdl-list__item">
                     <span class="mdl-list__item-primary-content">Tilt angle</span>
                     <span class="mdl-list__item-secondary-action">
                         <div class="mdl-grid">
                             <div class="mdl-cell mdl-cell--10-col">
                                 <input id="slider-angle-2" data-id="2" class="mdl-slider mdl-js-slider"
                                     type="range" min="0" max="90" value="90"
                                     onchange="setAngle(this.dataset.id, this.value);" >
                             </div>
                             <div class="mdl-cell mdl-cell--2-col">
                                 <input id="slider-text-angle-2" data-id="2" style="width:35px;"
                                 onchange="setAngle(this.dataset.id, this.value);" value="90"></input>
                             </div>
                         </div>
                     </span>
                 </li>
             </ul>
             <!-- probando -->
             <h6>Default position</h6>
             <button class="mdl-button mdl-js-button mdl-button--primary mdl-js-ripple-effect" style="width: 160px;"
                 data-id="default-pos" onclick="setDefault(this.dataset.id)">
                 DEFAULT
             </button>
         </div>
       </div>
     </div>
    </div>
</body>

<script type="text/javascript" src="./js/main.js"></script>
<script defer src="https://code.getmdl.io/1.3.0/material.min.js"></script>
</html>
```

Figura 2.11 b) Código de la página web 'index.html'

2.4.5 Diseño y creación del 'back-end' de la aplicación web en JavaScript

Sin embargo, esta página web no haría absolutamente nada y ni podría comunicarse con el servidor sin un fichero JavaScript que haga posible el back-end de operaciones, así pues, el fichero creado para esta tarea recibió el nombre de 'main.js', el cual se muestra en las figuras 2.12, 2.13, 2.14 y 2.15. Nótese como este fichero es llamado en la penúltima línea del código html de la Figura 2.11.

Por la Figura 2.12, se puede ver que se creó en el fichero 'main.js' una nueva conexión con el servidor, ligada a la locación dada al objeto 'ws' establecido en el fichero 'WebSockets.hpp', de esta manera se crea la conexión entre el archivo de JavaScript con el servidor para el envío de peticiones y recibimiento de respuestas por parte de este último

Las primeras tareas escritas son solo para notificar a través de la consola del navegador los procesos que se están llevando a cabo 'tras bastidores', como de que, si ocurrió un error de conexión, se cerró la conexión o si se recibió información del servidor.

```
var connection = new WebSocket('ws://' + location.hostname + '/ws', ['arduino']);

connection.onopen = function () {
   connection.send('Received from Client');
   console.log('Connected');
};

connection.onerror = function (error) {
  console.log('WebSocket Error', error);
};

connection.onmessage = function (e) {
    console.log('Received from server: ', e.data);
    processReceived(e.data);
};

connection.onclose = function () {
  console.log('WebSocket connection closed');
};

function processReceived(data)
{
   json = JSON.parse(data)
   if(json.command == 'updateState')
   {
      updateState(json.id, json.status);
   }
}
```

Figura 2.12 Fichero 'main.js' parte 1

Dentro de la función 'processReceived' se puede escribir cualquier acción dependiendo de la información recibida por el servidor y que a su vez se quiera mostrar en la página web.

En la Figura 2.13, se ve la función 'setEje', en esta se cambian los estilos y los 'labels' del indicador de estado ubicado en la interfaz web debajo de cada switch. Además, en un fichero Json se envía información como el id que indica qué eje es al que se le está cambiando el estado, el comando, que indica la operación solicitada por el cliente y el 'status' que indica el cambio efectuado en dicha operación.

De manera similar es la forma en que se maneja la función 'setAngle' (Figura 2.14), esta si bien no modifica los estilos en la interfaz, si genera cambios visibles como actualizar la posición del 'slider' recién manipulado, así como su respectivo cuadro de texto. También envía al servidor en un archivo Json el comando, que es 'setAngle', el id referente al eje a modificar su orientación, y el nuevo valor de orientación para que así el servidor tenga la suficiente información para ejecutar la acción solicitada en el front-end.

```
function setEje(id, status)
{
   console.log(id);
   if(status == true)
   {
        document.getElementById('eje-indicator-' +id).textContent = 'On';
        document.getElementById('eje-indicator-' +id).classList.add('On-style');
        document.getElementById('eje-indicator-' +id).classList.remove('Off-style');

   }
   if(status == false)
    {
      document.getElementById('eje-indicator-' +id).textContent = 'Off';
      document.getElementById('eje-indicator-' +id).classList.add('Off-style');
      document.getElementById('eje-indicator-' +id).classList.remove('On-style');
   }
   let data = {
      command : "setEje",
      id: id,
      status: status
   }

  let json = JSON.stringify(data);
  connection.send(json);

}
```

Figura 2.13 Fichero 'main.js' parte 2

Debajo de esa función se escribió la de 'setDefault', que coloca la posición de ambos ejes a 80 y 90 grados para los ejes Azimut y Tilt respectivamente (siempre y cuando ambos ejes tengan estado activo o de encendido), actualiza tales valores en los 'sliders' y cuadros de texto correspondientes, envía un fichero Json al servidor con el comando correspondiente, un 'id' constante (pues ambos ejes serán modificados) y un valor constante también (90 grados).

Por último, en la Figura 2.15 está la última función escrita en este fichero que simplemente actualiza los 'sliders' y la caja de texto que los acompaña, si es que el usuario los modifica desde la interfaz. Esta función es llamada dentro de la función 'setAngle'

```
function setAngle(id, angle)
{
   updateSliderText(id, angle);

   let data = {
      command : "setAngle",
      id: id,
      angle: angle
   }

  let json = JSON.stringify(data);
  connection.send(json);
}

function setDefault(id)
{
         document.getElementById('slider-angle-1').value = 80;
         document.getElementById('slider-text-angle-1').value = 80;
         document.getElementById('slider-angle-2').value = 90;
         document.getElementById('slider-text-angle-2').value = 90;
         let data = {
                command : "setDefault",
                id: "1",
                angle: "90"
        }

  let json = JSON.stringify(data);
  connection.send(json);
}
```

Figura 2.14 Fichero 'main.js' parte 3

```
function updateState(id, status)
{

}

function updateSliderText(id, value) {
   document.getElementById('slider-angle-' +id).value = value;
   document.getElementById('slider-text-angle-' +id).value = value;
  }
```

Figura 2.15 Fichero 'main.js' parte 4

2.4.6 Mejoras visuales para la aplicación web

Las hojas de estilo usadas para la página web fueron llamadas por CDN, así pues, el cliente al disponer de conexión a internet podrá bajar las hojas de estilo de manera automática cuando ingrese a la página web. La única hoja de estilo incluida dentro de la memoria del servidor, es decir que el cliente no se encarga de descargar y que el servidor la proporciona directamente, es 'main.css' (Figura 2.16). Este fichero contiene información sobre la forma en que deben lucir los cuadros indicadores de los estados On/Off de los ejes y de la disposición de los elementos en la página.

```
.label-big {
   border-radius: 3px;
    font-size: 16px;
    font-weight: 600;
    line-height: 2;
    padding: 0 8px;
    transition: opacity .2s linear;
    color: #fff
}

.On-style {
   background-color: #006b75;
}

.Off-style {
   background-color: #b60205;
}
/* FER */
#control_panel{
   text-align: center;
   float:center;
}
#bloque1{
   text-align: center;
   float:left;
}
#bloque2{
   padding-top: 124px;
   text-align: center;
   float: left;
}
```

Figura 2.16 Fichero 'main.css'

Los tres archivos presentados recientemente, que son; el index.html, main.js y main.css, se subieron directamente a la memoria flash del NodeMCU, así se ahorraron recursos de procesamiento y espacio de la memoria dinámica al tener y hacer uso de estos ficheros de forma permanente. Los ficheros mostrados al principio de extensión .hpp y .h, así como el sketch principal, se compilaron y subieron usando el entorno de Arduino IDE.

Tras toda la programación involucrada, solo queda poner en marcha el servidor, para lo cual solo basta energizar el NodeMCU, y contar con acceso a una conexión de internet, abrir el navegador y poner la dirección 192.168.1.200 para hacer uso de la aplicación web y controlar la orientación del modelo de antena presentado dentro de este mismo documento.

2.4.7 Programación del driver (Arduino UNO Board)

Una parte importante para la implementación del sistema fue la programación del driver, pues sin ello, aun teniendo el servidor VPN, el servidor del control de la antena y la antena misma, no se podría esperar que, teniendo los comandos indicados provenientes del cliente los actuadores realicen acción alguna. Así pues, en Arduino IDE se escribió un programa para esta tarea el cual se muestra en las figuras 2.17, 2.18 y 2.19.

```
#include <Servo.h>

Servo azimut;
Servo tilt1;
Servo tilt2;
String dato, r1, r2, angle1, angle2;
int tilt1a, tilt2a;
bool state1 = 1;
bool state2 = 1;
void setup() {
  azimut.attach(11);
  tilt1.attach(10);
  tilt2.attach(6);
  Serial.begin(115200);
}
```

Figura 2.17 Programación del driver parte 1

En la Figura 2.17 se percibe que únicamente se usó la librería 'Servo' que es indispensable para el control de los actuadores de la antena. Se declararon diversas variables útiles para llevar a cabo los procesos que más adelante se describen, y se definieron las salidas PWM para controlar cada servomotor, teniendo en uso tres salidas; D11 para el control del servomotor del eje Azimut, D10 para el servomotor derecho del eje Tilt y D6 para el servomotor izquierdo del mismo eje. Se inicializo el puerto serial a 115200 baudios puesto que a esa velocidad está configurado el puerto serial virtual del servidor, que hace el papel de maestro.

```
void loop() {

//Serial.flush();
String S = "" ;
if (Serial.available()>0)
{
  dato = Serial.readString ();
  delay(500);
  Serial.println(dato);
  Serial.flush();
  delay(100);
}
r1 = dato.substring(0,2);
r2 = dato.substring(2,3);

  if(r1 == "As") {
    //r2 = dato.substring(2,3);
      if (r2 == "0"){
        statel = 0;
      }else {
        statel = 1;
      }
  }
  if(r1 == "Ts") {
    //r2 = dato.substring(2,3);
    if (r2 == "0"){
      state2 = 0;
    }else {
      state2 = 1;
    }
  }
```

Figura 2.18 Programación del driver parte 2

Posteriormente en el 'void loop', se lee el dato proveniente del servidor originado a partir de una petición del cliente (recordar la estructura que presenta este dato, explicado en el apartado anterior) y se almacena en la variable 'dato', habiendo un

pequeño retardo para asegurarse de que se haya leído la cadena completamente, después dicha cadena es dividida en sus dos partes, la parte del identificador ('Id') que contiene la información de a qué eje se le hará alguna modificación así como el tipo de modificación que se hará y la segunda parte que contiene el valor de la propiedad a modificar. Ambas partes se guardan en 'r1' y 'r2' respectivamente.

Seguido de ello, se pregunta qué tipo de identificador es el que se acaba de recibir y en base a ello se efectúa una acción. Por ejemplo, el identificador 'As' significa que se debe cambiar el estado (encendido/apagado) del eje Azimut, mientras que 'Ts' significa que se deberá cambiar el del eje Tilt. Así pues, al recibir cualquiera de estos dos identificadores se tomará la segunda parte de la cadena de texto recibida, y para este caso solo podrá tener dos valores: 'false' o 'true'. Si el identificador fue 'As' la variable 'state1' cambiará según el valor booleano recibido, si el identificador es 'Ts' será la variable 'state2' la que se verá modificada. Con ello más adelante se sabrá si el usuario quiere tener activados o no cada uno de los ejes.

```
  if(r1 == "Aa" && state1 == 1){
    angle1 = dato.substring(2,5);
    azimut.write(angle1.toInt());
  }
  if(r1 == "Ta" && state2 == 1){
    angle2 = dato.substring(2,5);
    tilt1a = angle2.toInt();
    tilt2a = 180 - tilt1a;
    //Serial.println(tilt2a);
    tilt1.write(tilt1a);
    tilt2.write(tilt2a);
  }
  if(r1=="90" && state1 == 1){
    if(state2 == 1){
      azimut.write(80);
      tilt1.write(90);
      tilt2.write(90);
    } else {
      azimut.write(80);
    }
  } else if(r1=="90" && state2 == 1){
      tilt1.write(90);
      tilt2.write(90);
  }
}
```

Figura 2.19 Programación del driver parte 3

Si el usuario desea cambiar el ángulo de los ejes, el driver recibirá del servidor el identificador 'Aa' para Azimut o el 'Ta' para Tilt, en ambos casos se tomará la segunda parte de la cadena de texto para conocer cuál será el nuevo valor de la magnitud a modificar. Para el caso del Azimut este valor puede ser de 0 a 160 grados, y para el Tilt puede ser de 55 a 125 grados, aun así, serán valores enteros sin importar el caso.

Si el identificador resulta ser 'Aa', simplemente se corrobora con la variable 'state'1 que este eje esté activo (Figura 2.19) y en caso de estarlo, se toma el valor del ángulo que quiere el usuario que alcance este eje de la antena y se escribe por la salida de control PWM correspondiente. En cambio, si resulta que el identificador es 'Ta' se comprueba con la variable 'state2' que el eje se encuentre activado y se escribe en el servomotor con el nombre de 'Tilt1' el valor del ángulo que requiere el usuario cambiar pero al otro motor no se le puede colocar este valor directamente puesto que está colocado en contraposición a 'Tilt1', por lo que es necesario restarle al número 180 el valor del ángulo recibido, el resultado de esta sencilla operación ya puede ser escrito por la salida de control PWM correspondiente.

Por último, si se recibe por el puerto serial el número 90, se sabe que no se esperará una segunda parte de la información, debido a que este valor quiere decir que el usuario quiere colocar a la posición inicial los ejes de la antena, así pues, a los tres servomotores se les escribe un valor constante correspondiente al eje al que pertenecen (siempre y cuando los ejes se encuentren activos/habilitados), por lo que se corroboran las variables 'state1' y 'state2'. De esta manera, a los dos servomotores del movimiento Tilt se les coloca un valor de 90 grados y al único motor del movimiento Azimut se le coloca el valor de 80 grados.

2.5 Implementación del servidor VPN

Utilizando una Raspberry Pi 3 con el sistema operativo Raspbian se creó el servidor VPN de la siguiente manera:

1. Se configuró una dirección IP fija para la Raspberry para lo cual, se dio clic derecho en el icono de Wireless del panel superior del escritorio y se seleccionó la

opción Wireless & Wired Network Settings, en la ventana que apareció se asignó la dirección fija de la Raspberry, en este caso fue 192.168.1.204 (Figura 2.20) con una máscara de subred 24, en la opción Router se introdujo la dirección de la puerta de acceso del router utilizado para la conexión de internet, 192.168.1.254 y como servidor DNS se utilizó uno propiedad de Google con dirección 8.8.8.8. (Figura 2.21)

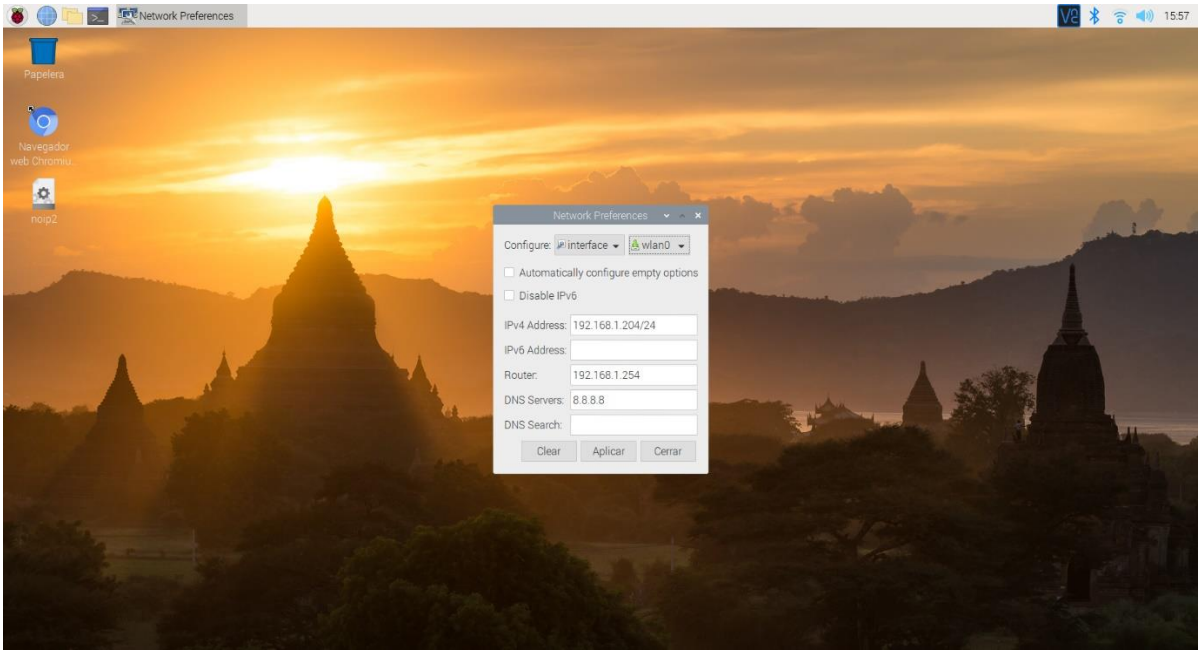

Figura 2.20 Configuración de la IP fija de la Raspberry

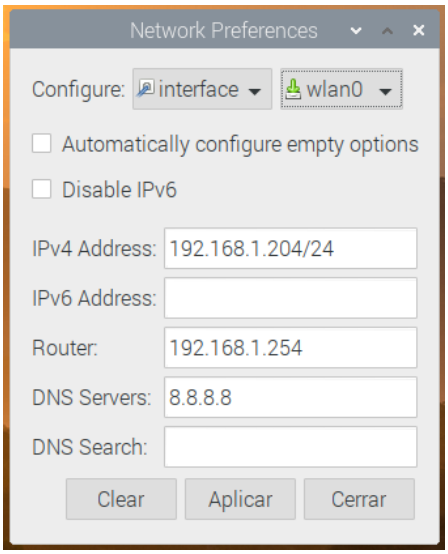

Figura 2.21 Configuración de la IP fija de la Raspberry en mayor detalle

2. Se procedió a establecer una dirección IP externa fija para poder conectarse al servidor VPN desde fuera de la conexión local de internet, para ello se utilizó un servicio de DNS dinámico como NO-IP. Este permite asignar un nombre de dominio a una dirección IP pública que cambia continuamente, permitiendo su conexión a esta. Para crear el servicio DNS en la página oficial de NO-IP se creó una nueva cuenta, con el nombre del servidor DNS, un correo electrónico y una contraseña. Luego de validar la dirección de correo electrónico se procedió a configurar el servidor DNS, para ello se le asignó el nombre de vongherx.ddns.net, el cual se corresponde a un servicio gratuito con vencimiento de un mes. A este nombre de dominio se le asignó la dirección pública del modem: 189.232.151.248. Tal como se muestra en la Figura 2.22

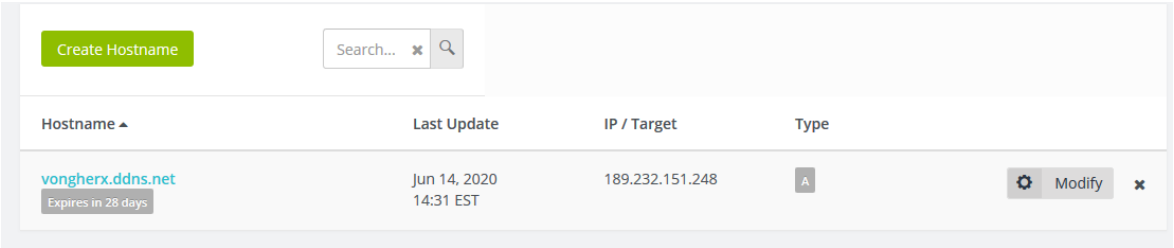

Figura 2.22 Creación del servidor DNS en NO-IP

Una vez realizada la configuración del nombre de dominio, se instaló el cliente No-IP en la Raspberry Pi, para ello en la terminal se escribieron los siguientes comandos:

- `wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz`
- `tar -zxvf noip-duc-linux.tar.gz`
- `cd noip-2.1.9-1`
- `sudo make install`

Los cuales se encargan de descargar y descomprimir los archivos de instalación necesarios para posteriormente instalar el cliente. Al teclear el último comando se abrió una interfaz en la cual se introdujeron los datos de validación de la cuenta de No-IP utilizada, tales como el correo electrónico y la contraseña, además de que se

seleccionó el nombre de dominio a utilizar por el cliente, que para este caso fue vongherx.ddns.net.

Una vez terminada la instalación del cliente se escribieron los siguientes comandos en la terminal:

- `sudo nano /etc/init.d/noip2`
- `sudo /usr/local/bin/noip2`
- `sudo chmod +x /etc/init.d/noip2`
- `sudo update-rc.d noip2 defaults`
- `sudo /usr/local/bin/noip2`

El primero de ellos crea un archivo con el nombre noip2 en el directorio /etc/init.d y lo abre utilizando el editor de texto nano, el segundo comando se debe escribir en el archivo de texto vacío y luego guardarlo. En la terminal se escriben las demás instrucciones, las cuales dan permisos al archivo y permiten al cliente No-IP arrancar automáticamente al encender la Raspberry.

3. El siguiente paso fue abrir los puertos del router para que redirija las peticiones del cliente al servidor VPN. Para realizar esto en el navegador se tecleó la dirección de la puerta de acceso del router, que para este caso fue: 192.168.1.254, y en el formulario se introdujo la contraseña del modem (Figura 2.24).

Para obtener la dirección de la puerta de acceso se tecleó el comando ipconfig en la consola de comandos de Windows (Figura 2.23).

```
C:\Users\luisz>ipconfig

Configuración IP de Windows


Adaptador de Ethernet Ethernet:

   Estado de los medios. . . . . . . . . . . : medios desconectados
   Sufijo DNS específico para la conexión. . : rga.ip

Adaptador de LAN inalámbrica Conexión de área local* 2:

   Estado de los medios. . . . . . . . . . . : medios desconectados
   Sufijo DNS específico para la conexión. . :

Adaptador de LAN inalámbrica Wi-Fi:

   Sufijo DNS específico para la conexión. . : rga.ip
   Vínculo: dirección IPv6 local. . . : fe80::f845:466:5c00:5c03%15
   Dirección IPv4. . . . . . . . . . . . . . : 192.168.1.71
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : 192.168.1.254

Adaptador de Ethernet Conexión de red Bluetooth:

   Estado de los medios. . . . . . . . . . . : medios desconectados
   Sufijo DNS específico para la conexión. . :
```

Figura 2.23 Obtención de la puerta de acceso del router

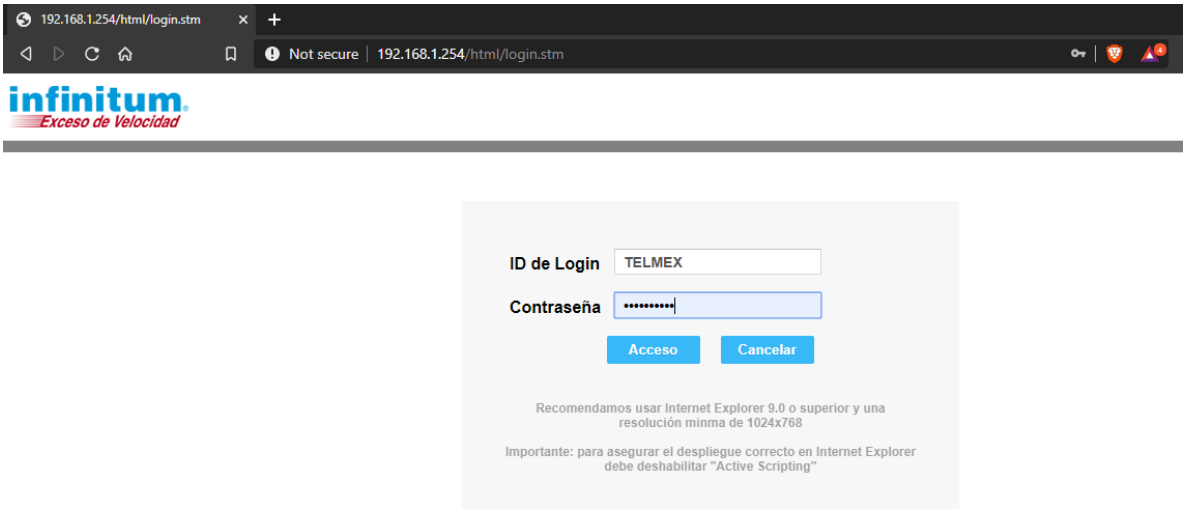

Figura 2.24 Acceso al modem al que está conectado la Raspberry

Una vez que se accedió al router, se abrió la pantalla por defecto del ‘Estado general del modem’, tal como se muestra en la Figura 2.25. Así pues, se dio clic en la opción NAT del panel de la izquierda, para poder cambiar la configuración de los puertos del modem.

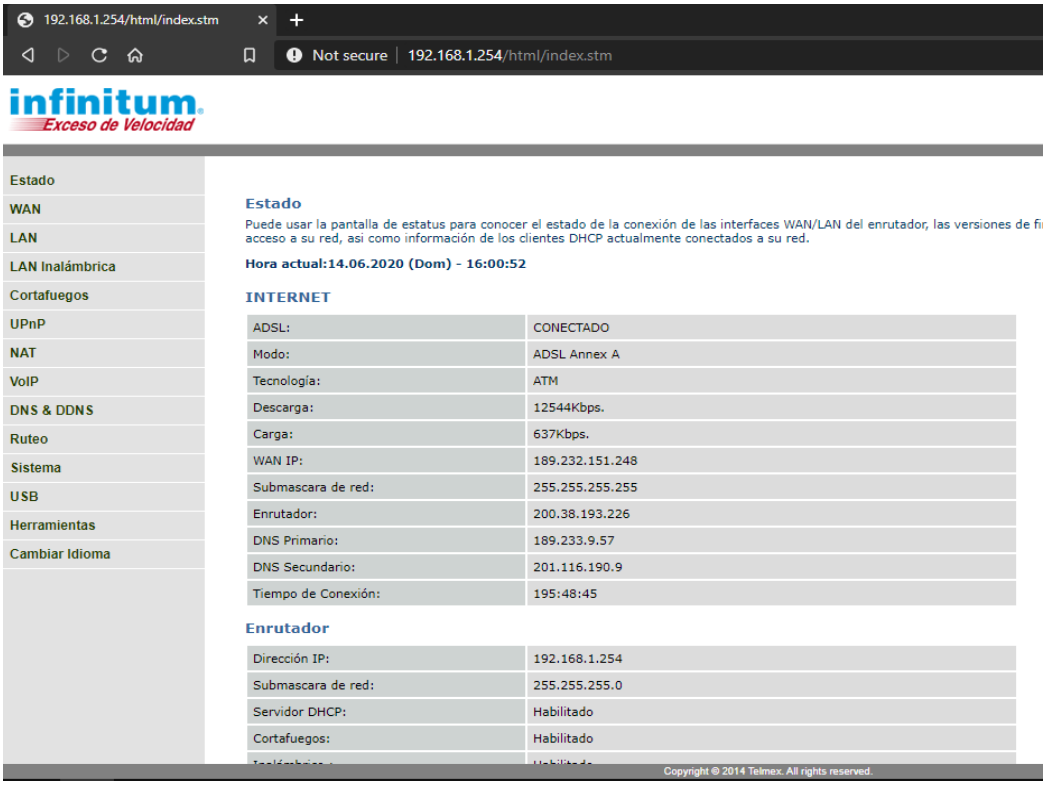

Figura 2.25 Pantalla del estado general del router

En NAT se accedió a la opción de 'Mapeo de puertos' para poder abrir uno nuevo que redireccionará al servidor VPN de la Raspberry. El puerto creado fue del tipo VPN cuya dirección IP es la misma que la dirección IP fija de la Raspberry, el tipo de protocolo es UDP, ya que este es el que se utiliza en la mayoría de los servicios de VPN, el puerto es el número 1194 que a su vez apuntará al puerto público 1194. Una vez habilitado el puerto se agregó a la configuración del router. En la Figura 2.26 se muestra un resumen del proceso.

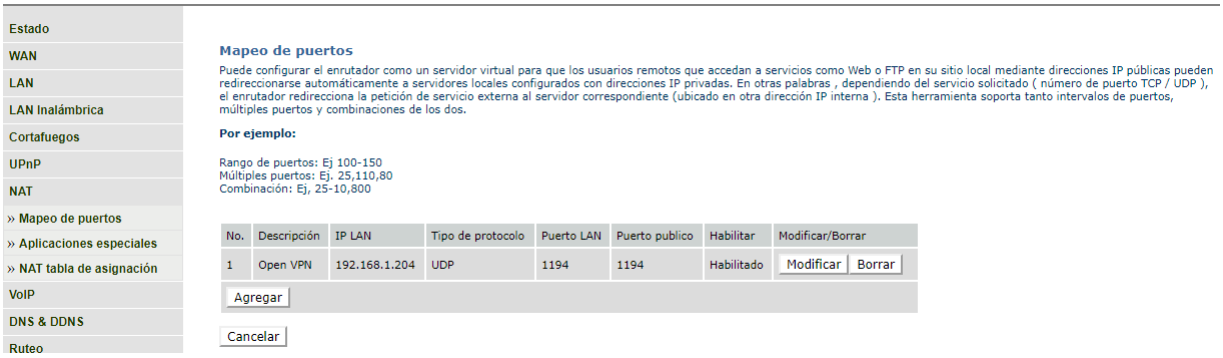

Figura 2.26 Apertura del puerto 1194 para el redireccionamiento al servidor VPN

4. El siguiente paso fue instalar y configurar el servidor OPENVPN, así que en la terminal de la raspberry se tecleó el siguiente comando: curl -L https://install.pivpn.io | bash. Inmediatamente se comenzaron a descargar los archivos de instalación del servidor y se ejecutó un instalador como el siguiente (Figura 2.27):

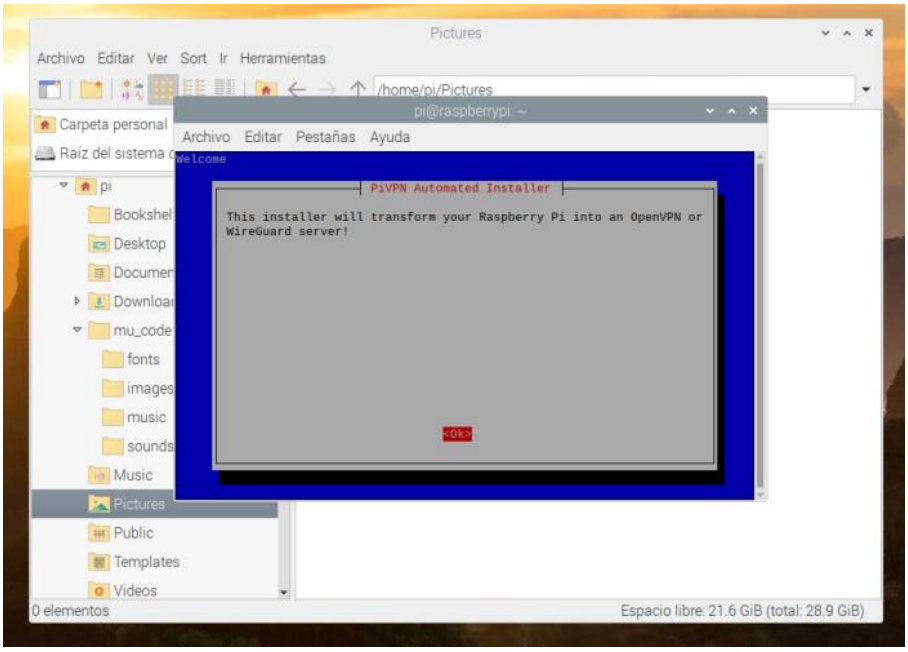

Figura 2.27 Inicio del proceso de instalación

La primera pantalla (Figura 2.27) pide el permiso para convertir la Raspberry en un servidor VPN, para lo cual se eligió OK.

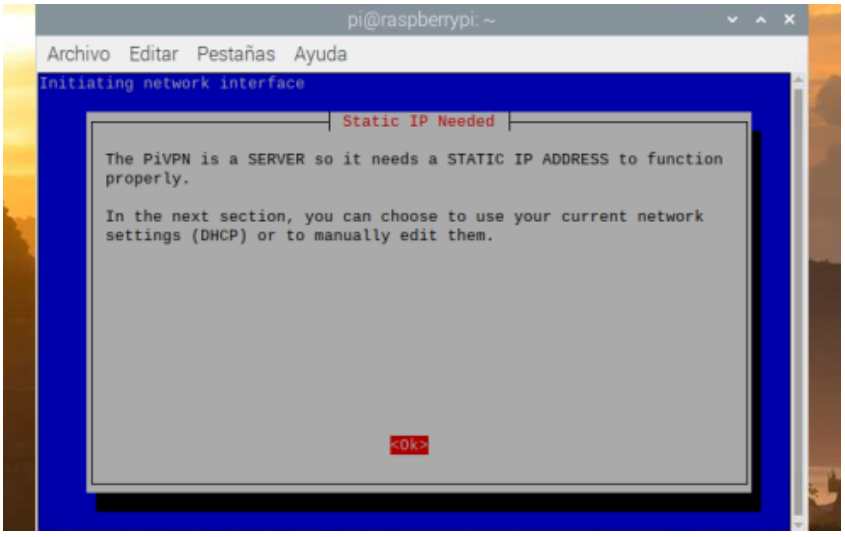

Figura 2.28 Utilización de la dirección IP de la Raspberry como IP fija

Posteriormente, se pidió la autorización para establecer la IP de la Raspberry como estática, tal como en la Figura 2.28 se presenta

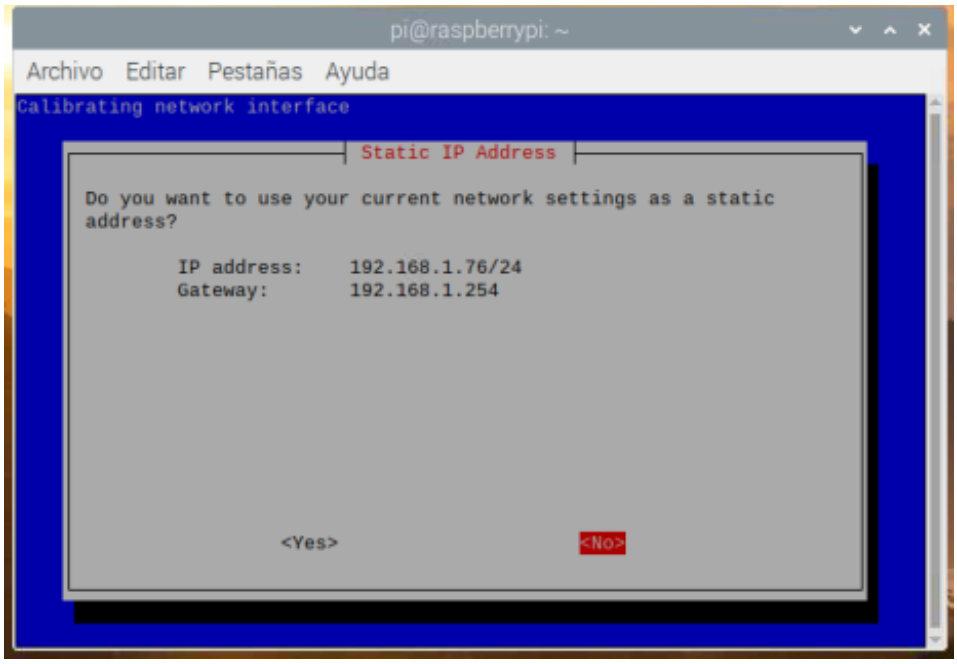

Figura 2.29 Establecimiento de la dirección IP del servidor y la puerta de acceso del router

Se estableció la dirección IP del servidor, así como dirección de la puerta de acceso del router y se seleccionó YES (Figura 2.29).

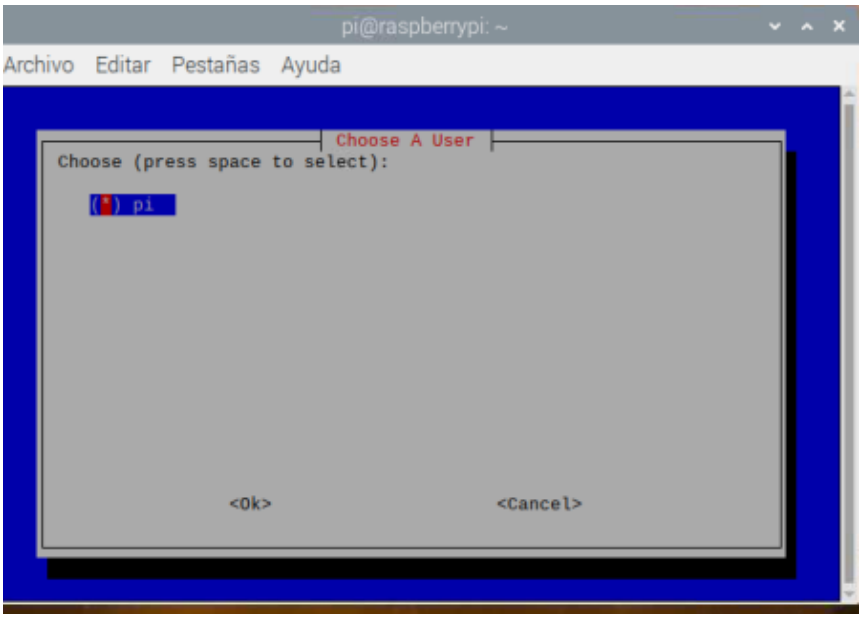

Figura 2.30 Usuario

Ahora se eligió el usuario en el que se instalará el servidor (Figura 2.30).

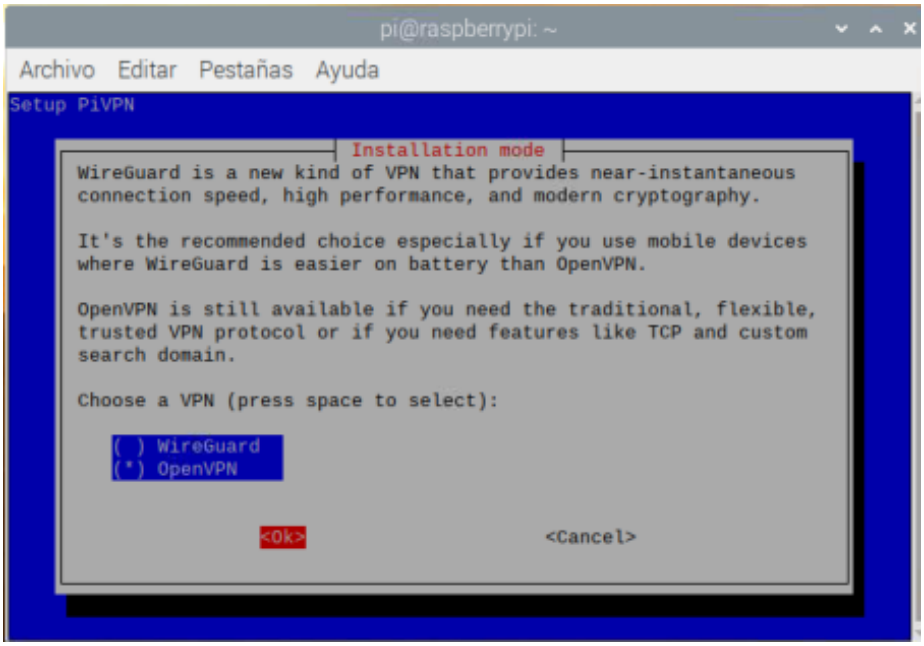

Figura 2.31 Tipo de instalación

En la pantalla presentada en la Figura 2.31 se eligió el tipo de instalación seleccionando la opción de OpenVPN y posteriormente se dio enter en la opción OK.

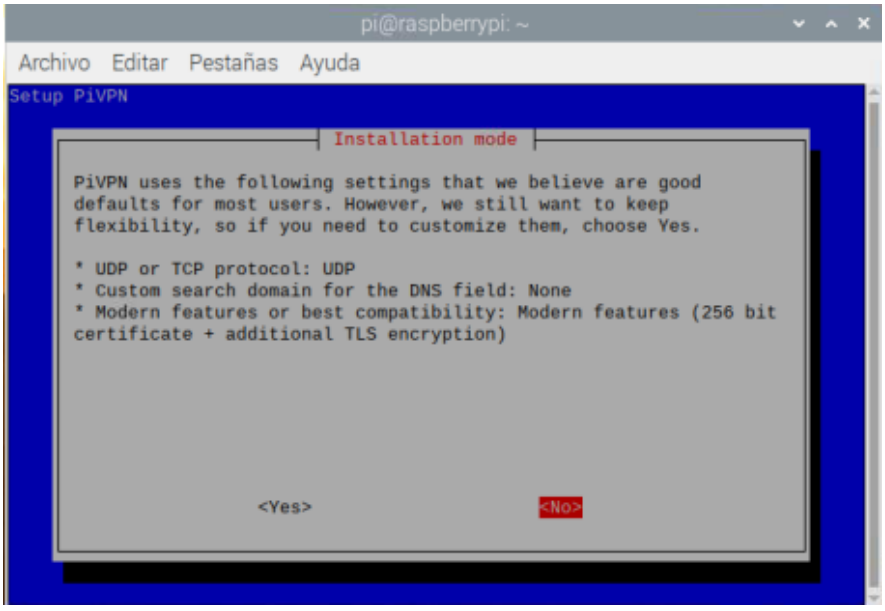

Figura 2.32 Tipo de protocolo

Para el protocolo de comunicación se eligió el 'UDP'. Para tales efectos se marcó la opción NO (Figura 2.32).

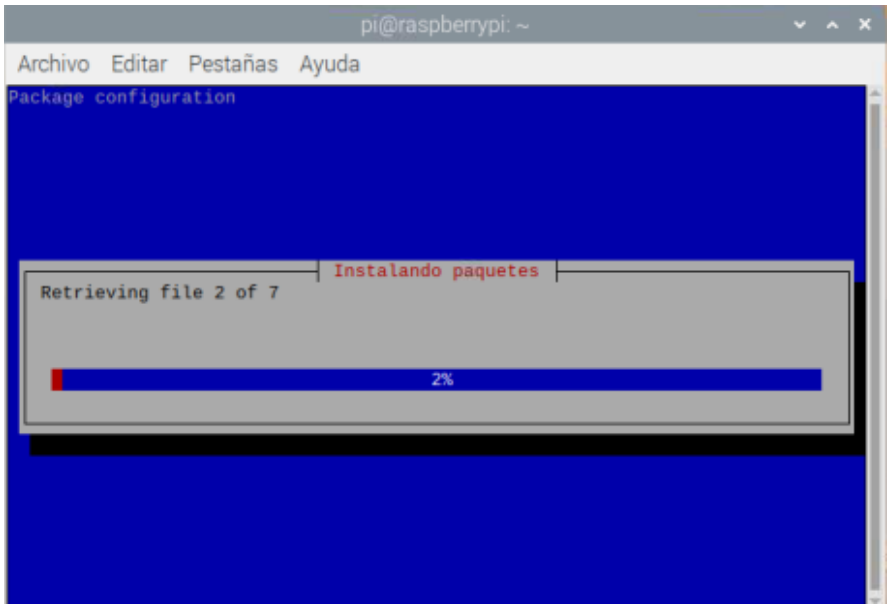

Figura 2.33 Proceso de instalación

Comienzo del proceso de instalación del servidor (Figura 2.33).

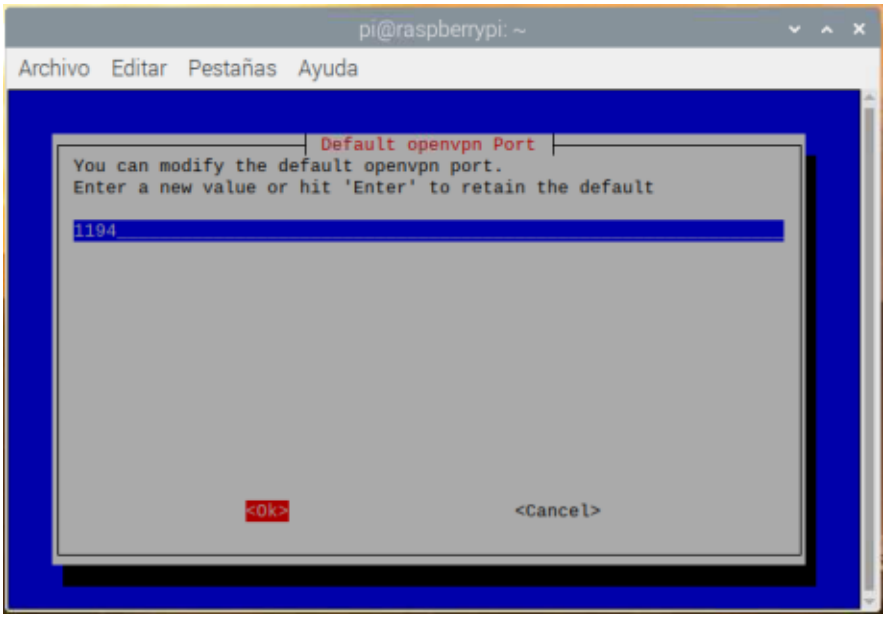

Figura 2.34 Puerto

El puerto que se seleccionó para el servidor fue el mismo que se configuró en el router en el apartado de mapeo de puertos (Figura 2.34).

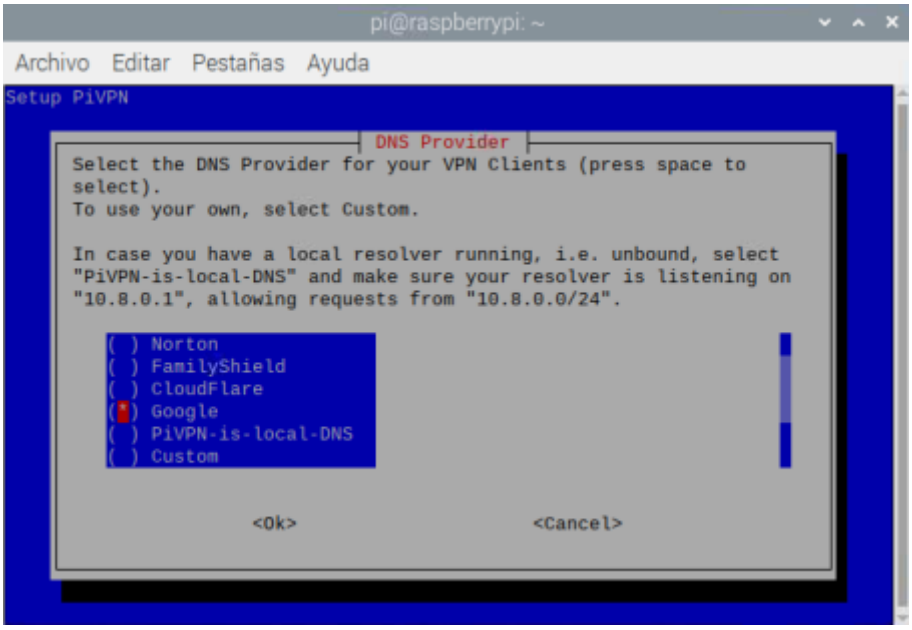

Figura 2.35 Proveedor DNS

Como proveedor de DNS se eligió el servicio proporcionado por Google (Figura 2.35), además de que esta configuración fue la que se estableció en la configuración de red de la Raspberry.

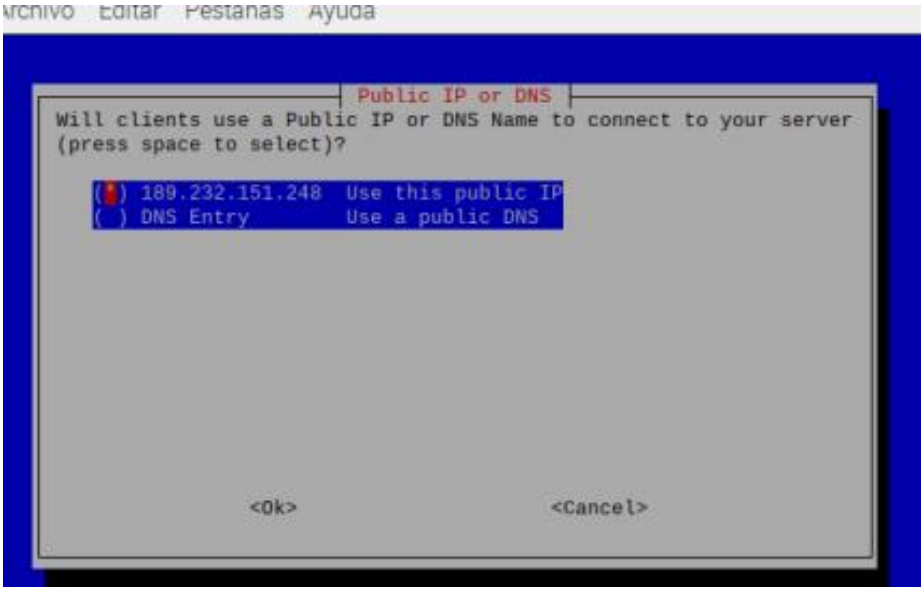

Figura 2.36 Servicio DNS

Se eligió el servicio de IP pública como parte de un servidor DNS bajo el nombre de dominio vongherx.ddns.net, para ello se marcó 'DNS Entry y se introdujo el nombre de dominio, tal y como se muestra en la Figura 2.36

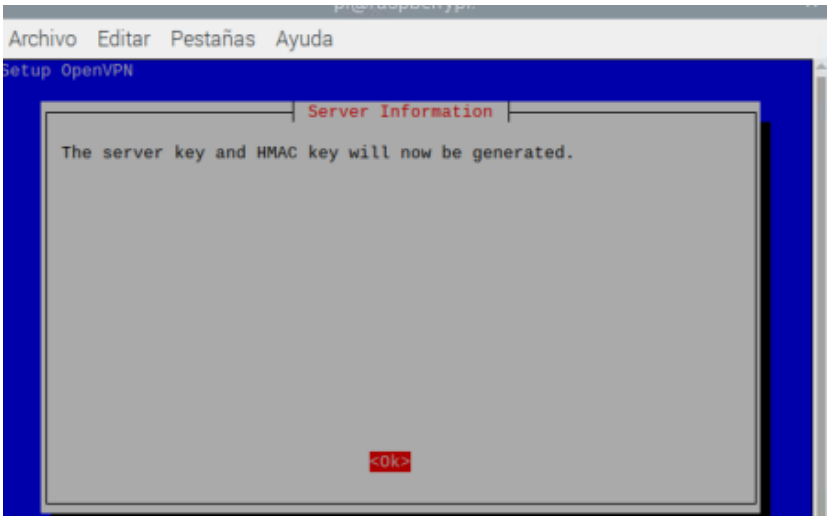

Figura 2.37 Generación del servidor

A continuación, se procedió a generar el servidor bajo la configuración establecida (Figura 2.37). Finalmente, se aceptaron las demás opciones restantes para poder actualizar la seguridad del servidor (Figura 2.38, Figura 2.39, Figura 2.40 y Figura 2.41).

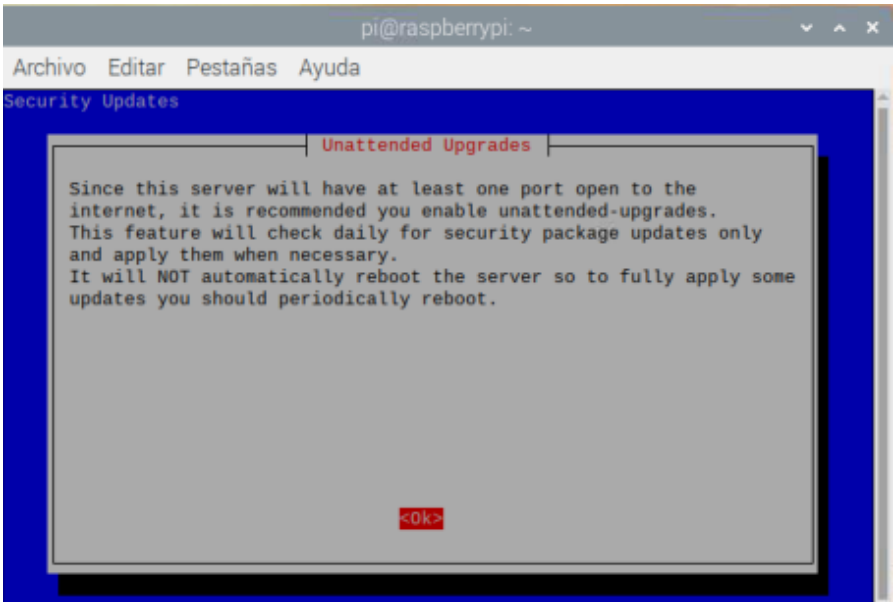

Figura 2.38 Aceptar actualización

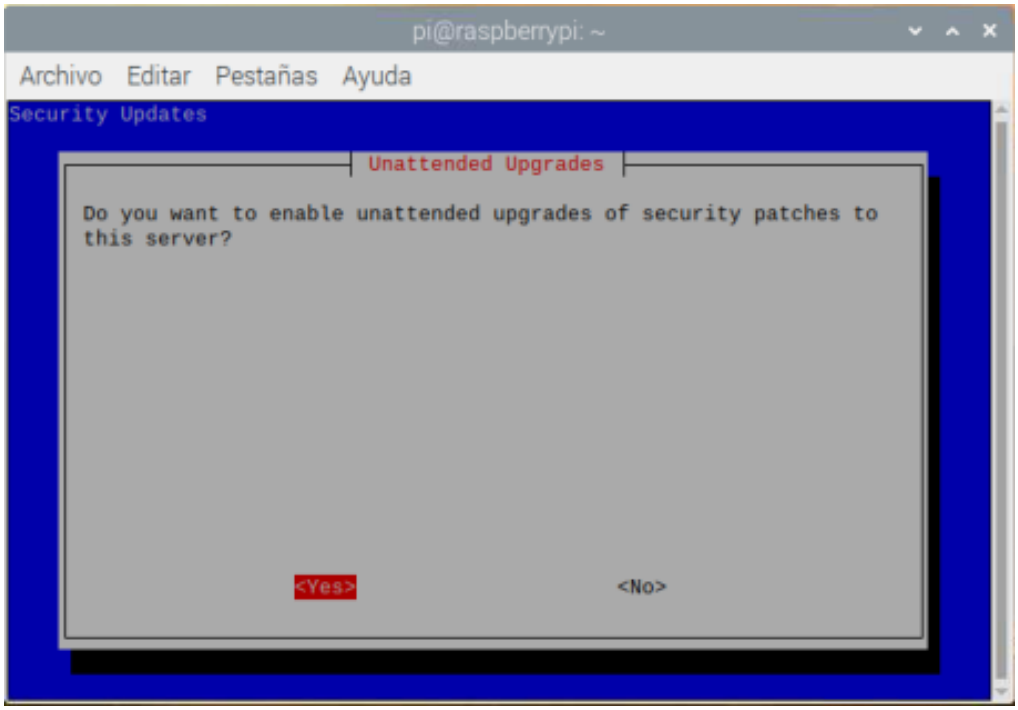

Figura 2.39 Actualizaciones de seguridad

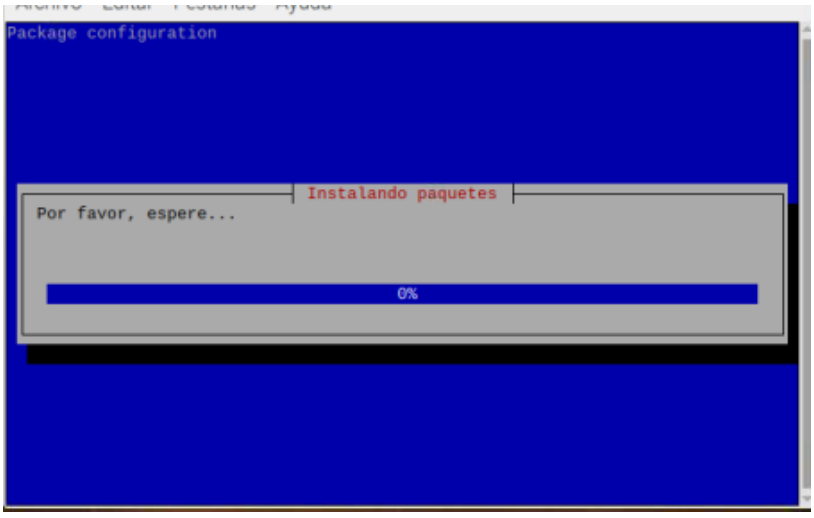

Figura 2.40 Instalando paquetes de actualización

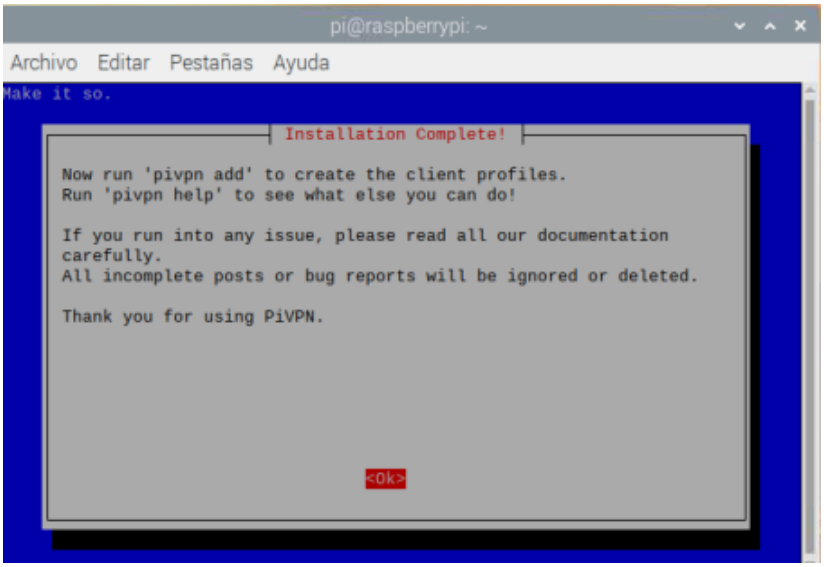

Figura 2.41 Actualización terminada

De esta manera terminó la instalación del servidor, por lo que solo restó crear las credenciales de los clientes. Para lo que se tecleo el comando mostrado en la Figura 2.42

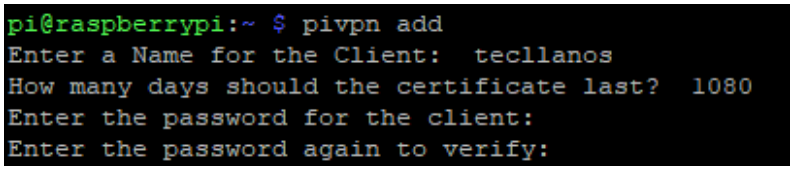

Figura 2.42 Creación de credenciales para un cliente

El nombre que se eligió para identificar al primer cliente que se conectará al servidor fue “tecllanos”, se especificó una duración para el certificado de 1080 días, al igual que una contraseña para establecer la conexión (Figura 2.43). A través de este certificado un cliente puede utilizar un servicio de VPN para conectarse a la red local en la que se encuentra la Raspberri Pi e interactuar con un servidor web creado a partir de una NodeMCU.

```
pi@raspberrypi:~ $ pivpn add
Enter a Name for the Client:  tecllanos
How many days should the certificate last?  1080
Enter the password for the client:
Enter the password again to verify:
spawn ./easyrsa build-client-full tecllanos

Note: using Easy-RSA configuration from: /etc/openvpn/easy-rsa/vars
Using SSL: openssl OpenSSL 1.1.1d  10 Sep 2019
Generating an EC private key
writing new private key to '/etc/openvpn/easy-rsa/pki/easy-rsa-1481.goKC44/tmp.UtXYjx'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
Using configuration from /etc/openvpn/easy-rsa/pki/easy-rsa-1481.goKC44/tmp.IaxevW
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'tecllanos'
Certificate is to be certified until May 30 20:28:11 2023 GMT (1080 days)

Write out database with 1 new entries
Data Base Updated

Client's cert found: tecllanos.crt
Client's Private Key found: tecllanos.key
CA public Key found: ca.crt
tls Private Key found: ta.key


================================================================
Done! tecllanos.ovpn successfully created!
tecllanos.ovpn was copied to:
  /home/pi/ovpns
for easy transfer. Please use this profile only on one
device and create additional profiles for other devices.
================================================================
```

Figura 2.43 Credencial creada correctamente

Para poder acceder a la red en la que se ejecuta el servidor VPN, se utilizó la aplicación móvil OpenVPNConnect, la cual es un cliente VPN para acceder a una red por medio de un archivo de credencial. Esta aplicación se descargó desde Google Play, al abrir la aplicación se importó el archivo de credencial correspondiente al realizado con los comandos mostrados de las figuras 2.42 y 2.43. Tal archivo se encuentra alojado en la carpeta ovpn del directorio home de la Raspberry y se pasó a la tarjeta SD del teléfono celular.

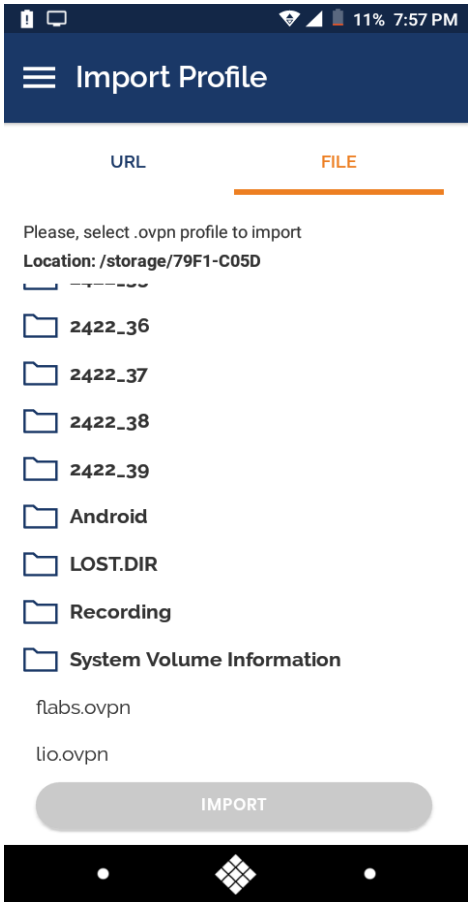

Figura 2.44 Archivo de credencial en la tarjeta SD

Tal como se observa en la Figura 2.44, el archivo de credencial fue importado a la aplicación y una vez que este fue aceptado, se mostró el servidor DNS al que se realizará la conexión, así como el nombre del cliente (Figura 2.45).

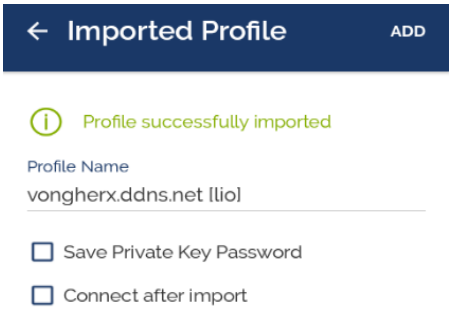

Figura 2.45 Perfil creado

Al dar clic sobre el perfil, se abrió una ventana para introducir la contraseña del cliente, Figura 2.46.

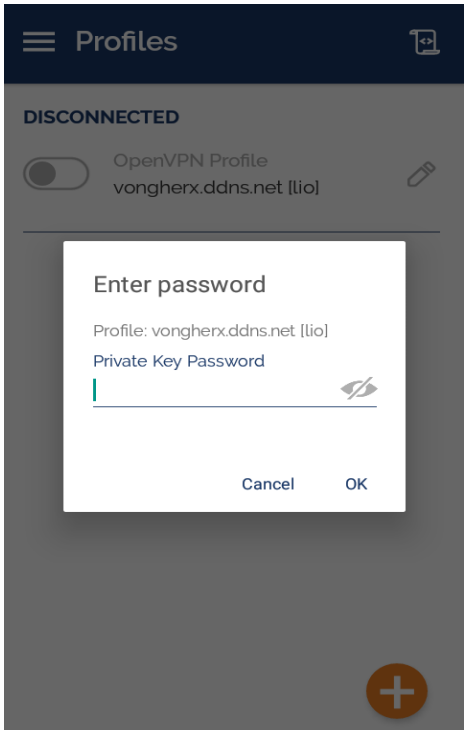

Figura 2.46 Contraseña del cliente

Una vez que se autentificó la información de la credencial y la contraseña del cliente, se estableció la conexión a la red local de la Raspberry por medio del servidor VPN gestionado por la misma, tal como se muestra en la Figura 2.47.

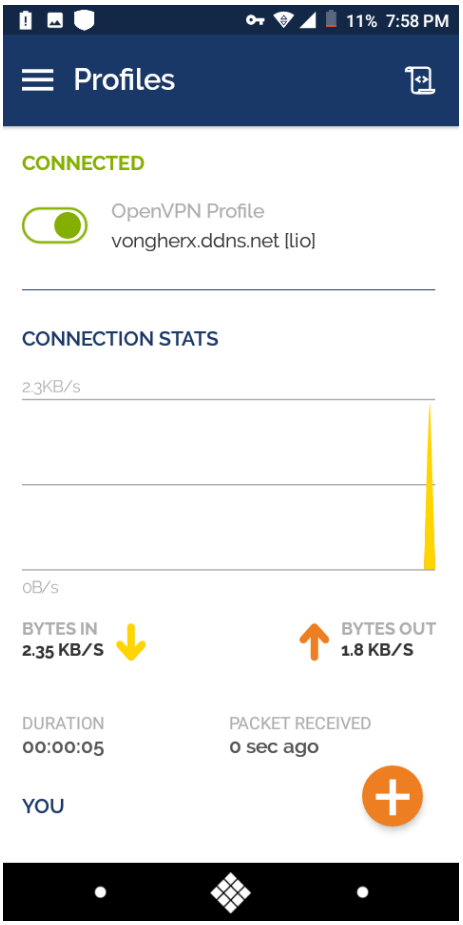

Figura 2.47 Acceso a la red VPN

Finalmente, para realizar un monitoreo de los clientes conectados a la VPN, se utilizó el comando para la consola de la Raspberry mostrado en la Figura 2.48

```
pi@raspberrypi:~ $ pivpn -c
: NOTE : The output below is NOT real-time!
:      : It may be off by a few minutes.

::: Client Status List :::
Name          Remote IP              Virtual IP      Bytes Received      Bytes Sent      Connected Since
tecllanos     189.155.64.11:59646    10.8.0.3        29KiB               44KiB           Jun 14 2020 - 15:31:00
```

Figura 2.48 Lista de usuario conectados

2.6 Construcción de la maqueta de la antena

2.6.1 Diseño del modelo mecánico

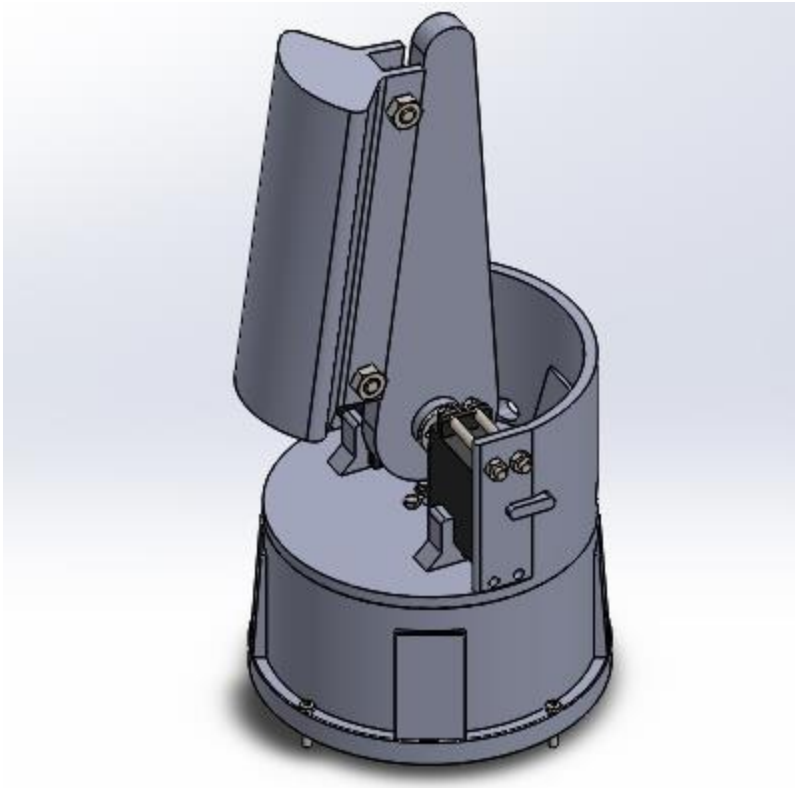

Figura 2.49 Diseño 3D del modelo mecánico de la antena a controlar por el sistema

En la figura anterior (Figura 2.49) se muestra el modelo de la antena del sistema de posicionamiento. Dicho modelo se diseñó utilizando el software SolidWorks, como se puede observar es un mecanismo de dos grados de libertad que consta de una base (Figura 2.50) la cual fija el dispositivo por medio de unos tornillos de ¼ de pulgada, dentro de la base se encuentra fijo un servomotor que permite modificar el ángulo de azimut de la antena en un rango de 180°.

En la Figura 2.51 se pueden observar distintas vistas de la base del mecanismo, así como las dimensiones de cada una de sus partes. Tal como se puede observar, se tiene una ranura en la cual se puede ajustar el servomotor, algunos orificios para la salida de los cables y unos nervios que distribuyen la carga que soporta la base evitando deformaciones en el material.

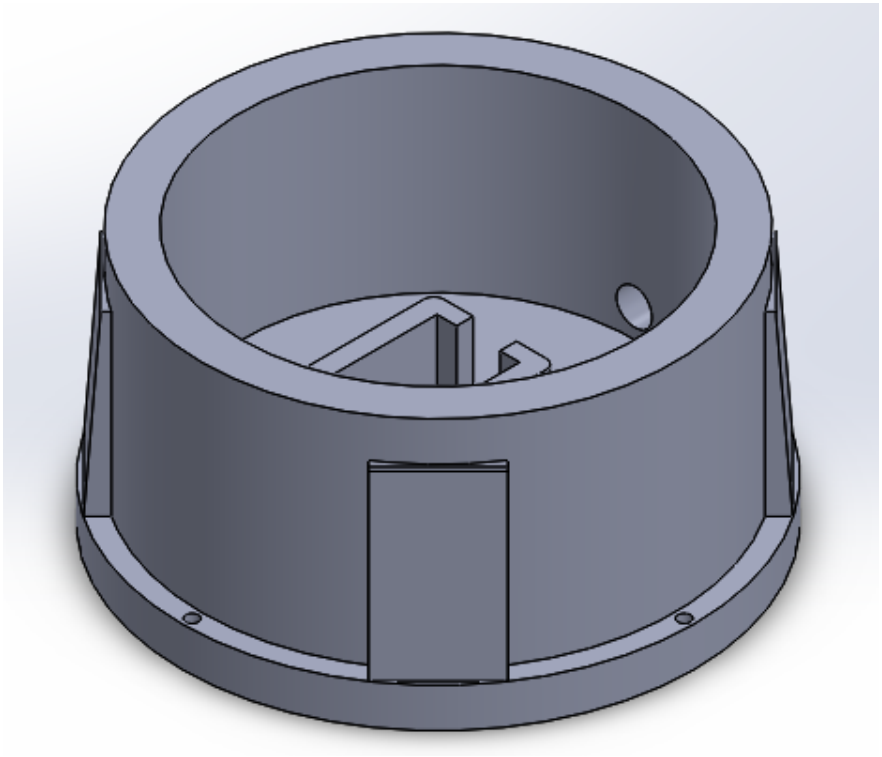

Figura 2.50 Vista isométrica de Base fija/bancada del sistema

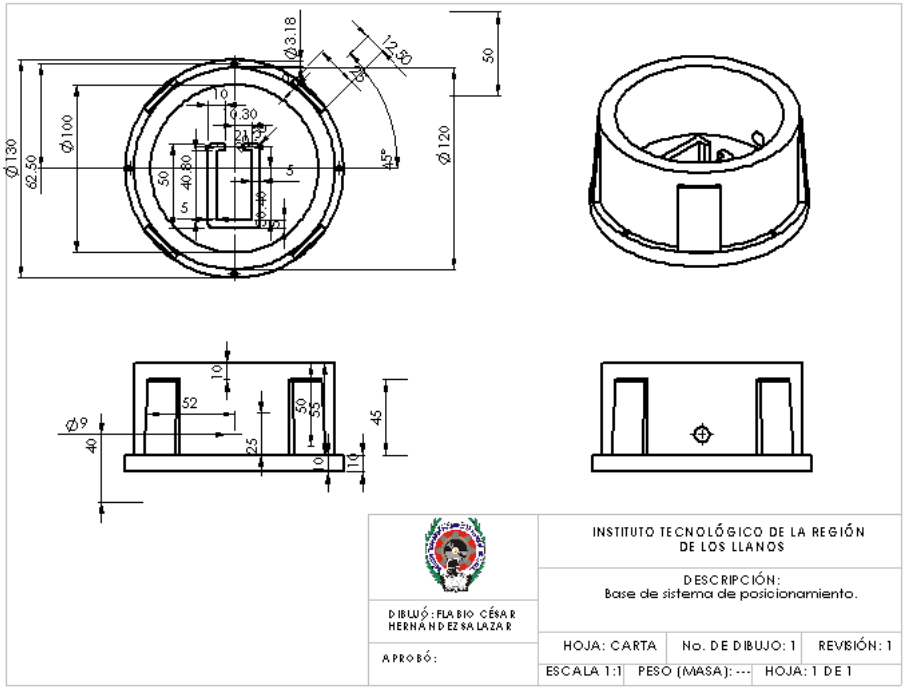

Figura 2.51 Plano de la base fija

El eje del servomotor ubicado en la base se conecta a una rueda móvil por medio de tornillos M3 del número 13 enroscados en 4 orificios ubicados en el centro de la pieza. La superficie inferior de la rueda móvil se apoya en la base fija permitiendo un giro uniforme en cualquier dirección. Además, en esta superficie móvil se colocan a los costados los servomotores que cambian la elevación (Tilt) de la antena, estos se fijan utilizando tornillos M4 del número 40. Como se puede observar en la Figura 2.52, se dispone de un orificio para poder rutear los cables de los servomotores y que estos no se enreden, además de una base de apoyo para que los servomotores no se muevan por acción de la inercia al momento de efectuarse un movimiento del azimut.

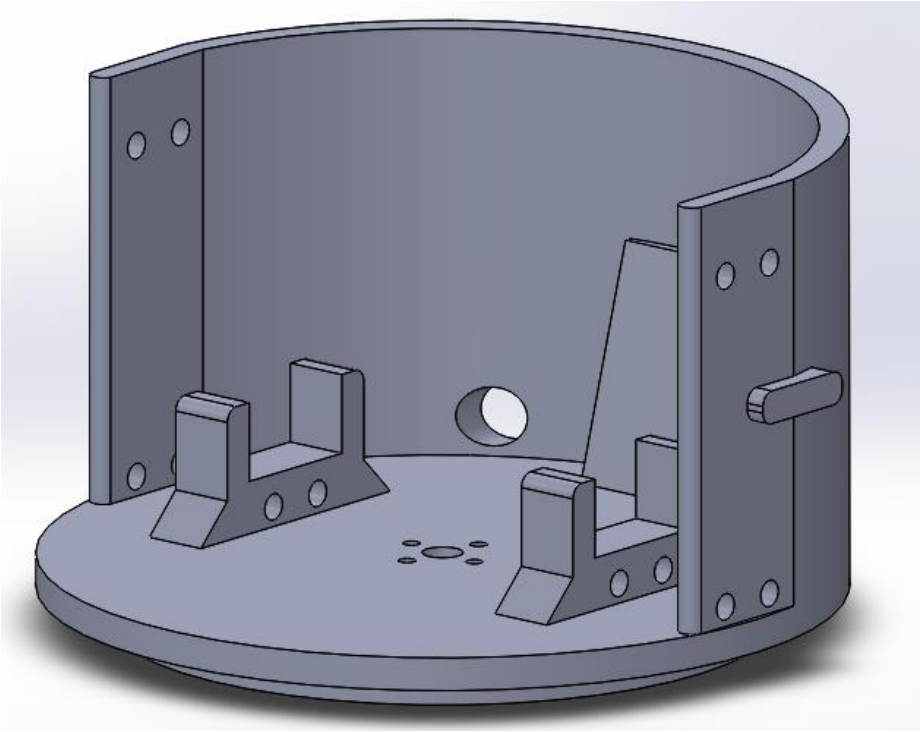

Figura 2.52 Superficie móvil para el cambio de la magnitud azimut

En la Figura 2.53 se observa el plano general de la pieza utilizada como superficie o base móvil, destacando las dimensiones de cada una de las partes que la componen.

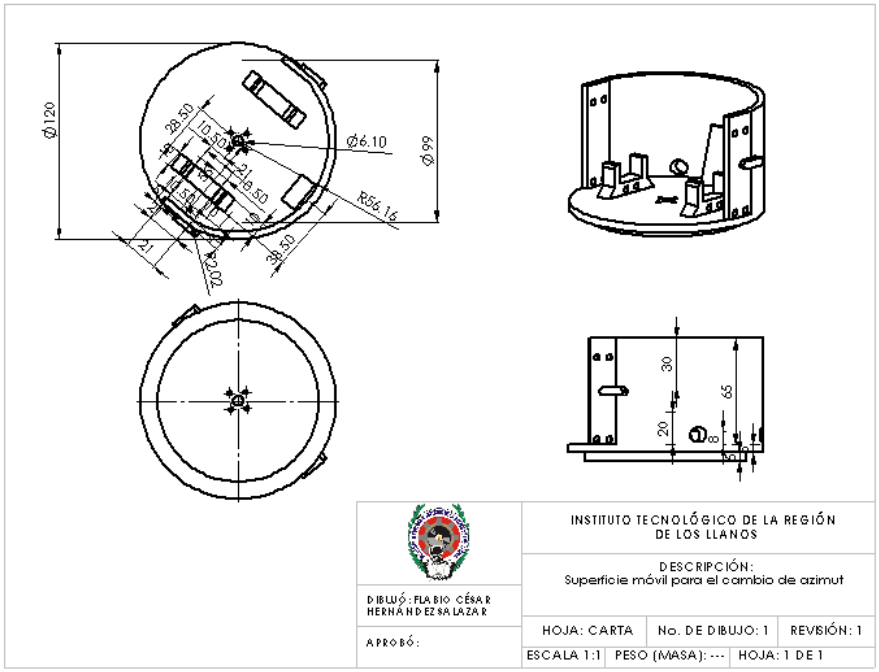

Figura 2.53 Plano de la superficie móvil

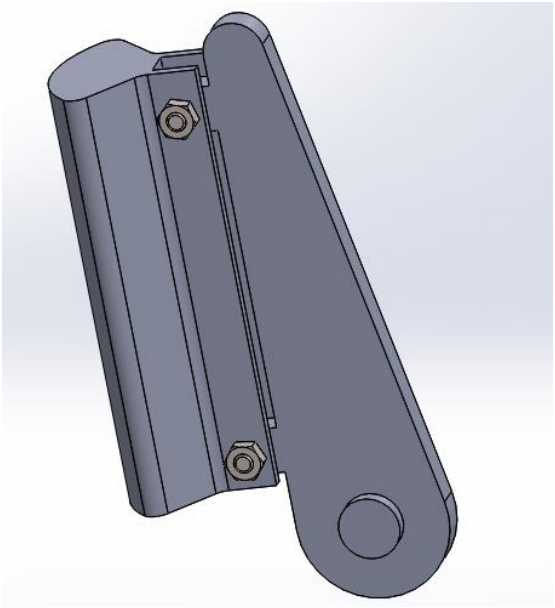

Figura 2.54 Vista lateral del Brazo sujetador, el cual es afectado por el cambio de la magnitud Tilt

Finalmente, el brazo (Figura 2.54) se une a los ejes de los servomotores ubicados a los costados de la base giratoria para modificar el ángulo de elevación de la antena. Esta última pieza se sujeta a la base móvil por medio de tornillos de ¼ de

pulgada. De esta manera se tiene el mecanismo para poder cambiar la elevación (Tilt) y la orientación (Azimut) de la antena.

En la Figura 2.55 se muestra el plano del brazo del modelo de la antena, mientras que en la Figura 2.56 se muestra también un plano, pero del modelo completo de la antena ya con todas sus partes ensambladas

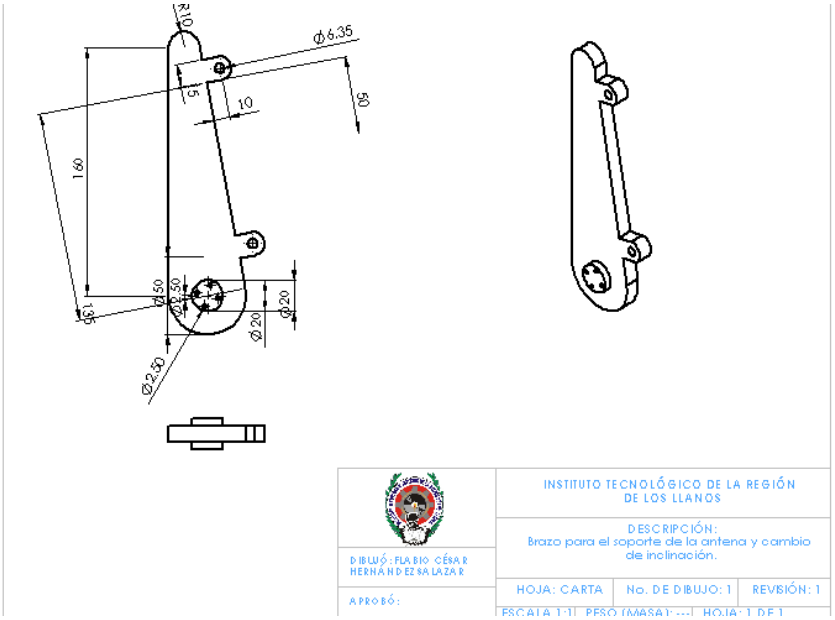

Figura 2.55 Plano del brazo sujetador

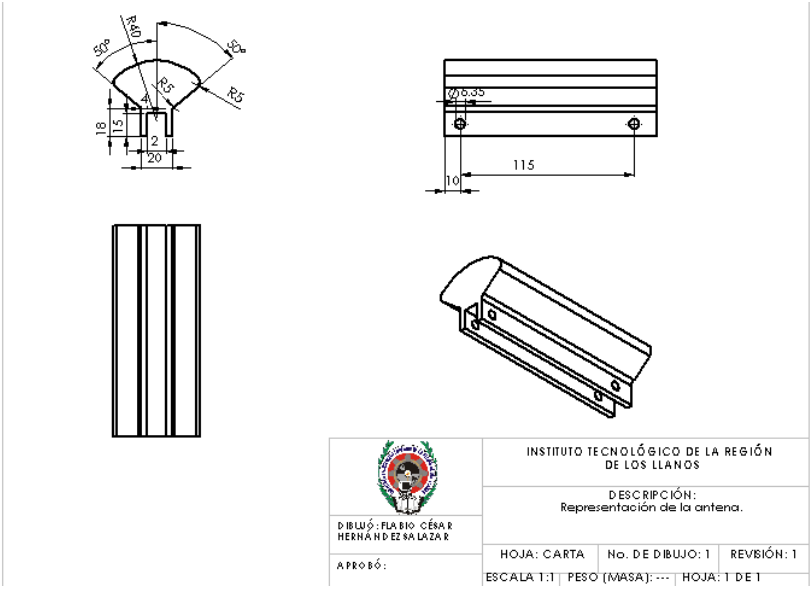

Figura 2.56 Plano de la antena

2.6.2 Obtención de archivos STL

El primer paso para fabricar por impresión 3D las piezas que conforman el modelo de la antena del sistema de posicionamiento es el de obtener los archivos STL de cada una de estas. Pata tal propósito, en Solidworks tras abrir el archivo de dibujo de cada una de las piezas, se dio clic en la opción 'Archivo' y luego en 'Guardar como', tal como se observa en la Figura 2.57. Posteriormente en la ventana que apareció se seleccionó el formato STL y luego se guardó con el mismo nombre que el de la pieza, Figura 2.58.

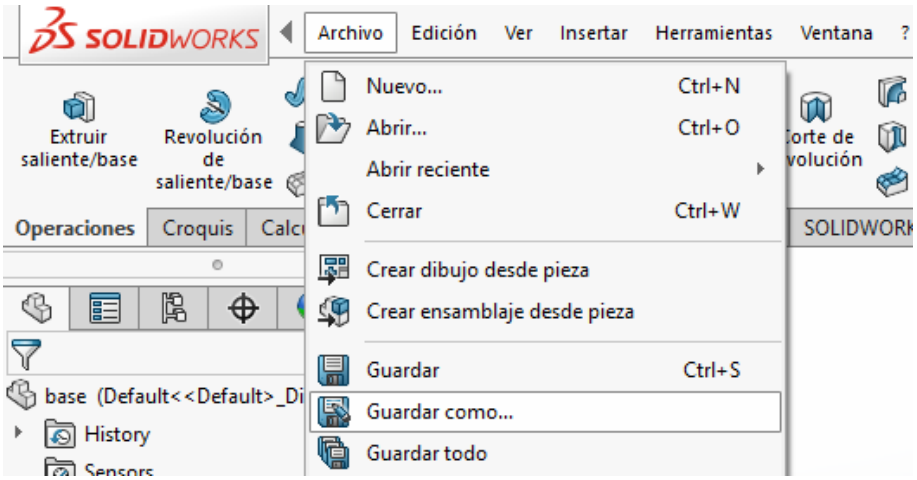

Figura 2.57 Ubicación de Archivo/Guardar como

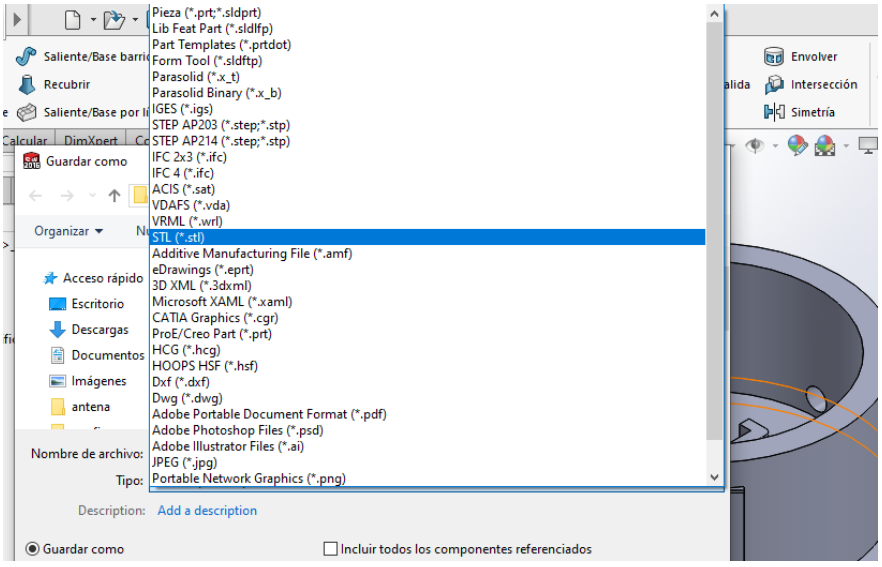

Figura 2.58 Elección del formato y guardado

El mismo procedimiento se aplicó a cada una de las piezas del sistema de posicionamiento, obteniendo al final cuatro archivos correspondientes (Figura 2.59) a las cuatro piezas del modelo de la antena.

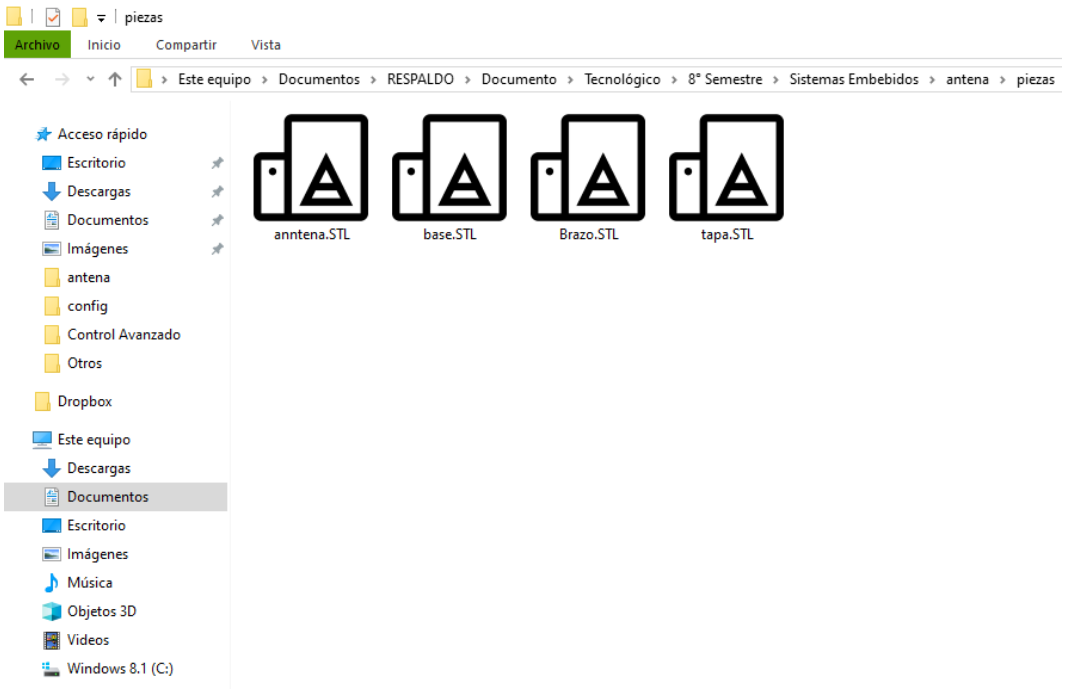

Figura 2.59 Archivos STL de cada una de las piezas

2.6.3 Impresión de las piezas

Se utilizó una impresora 3D marca Dremel con filamento PLA de color blanco. La impresora se conectó a la computadora por medio de un cable USB y utilizando el software Dremel 3D se cargó la pieza en formato STL a la impresora, para ello se dio clic en la opción 'cargar' y en la ventana que apareció se seleccionó el archivo de la pieza a imprimir, justo como se muestra en la Figura 2.60.

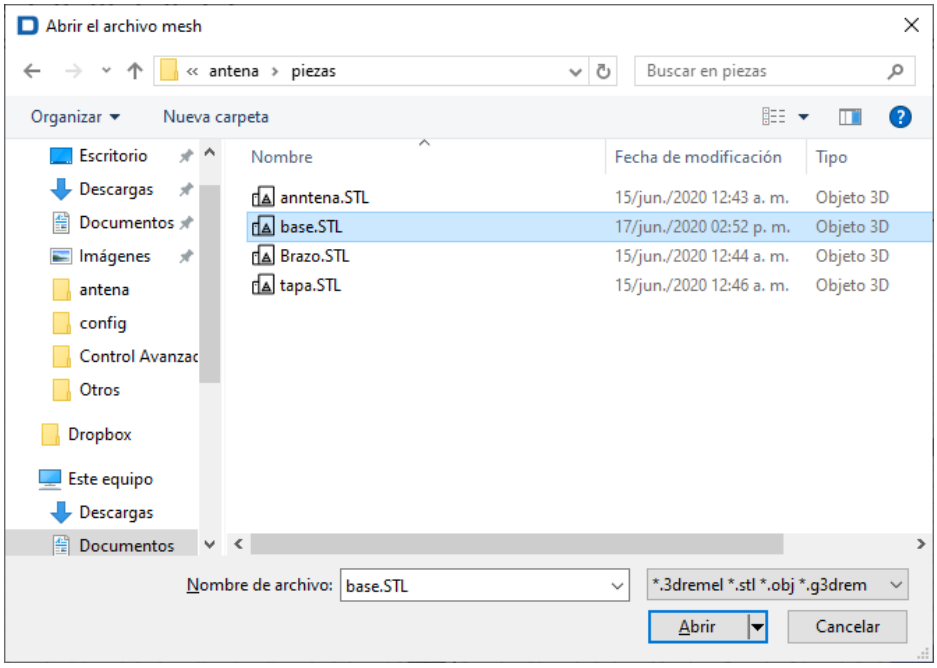

Figura 2.60 Pieza a imprimir

Utilizando los controles ‘Mover’ y ‘Girar’, se posicionó la pieza dentro del espacio de trabajo de la impresora, con el fin de que la impresión se realice de la forma correcta. Tal como se muestra en la Figura 2.61.

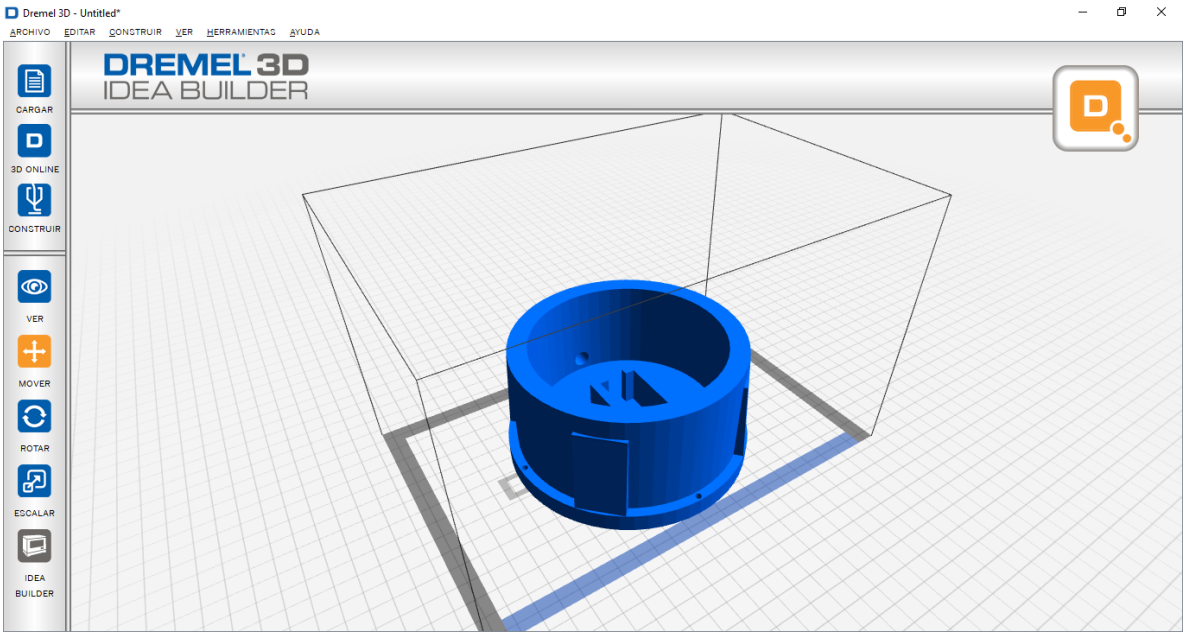

Figura 2.61 Ubicación de la pieza ‘base’ en el espacio de trabajo de la impresora

El siguiente paso fue realizar la conexión de la impresora 3D con la computadora, por lo que se dio clic en 'Construir' y luego en la opción 'Conectar' (Figura 2.62). Una vez hecha la conexión, se procedió a construir el modelo tras hacer clic en el botón 'Construir'.

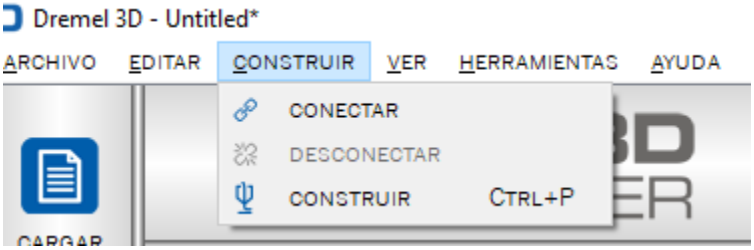

Figura 2.62 Opciones 'Conectar' y 'Construir'

De esta manera se abrió la ventana de la Figura 2.63 mostrando las opciones de configuración del proceso de adición de material de la impresora 3D. El archivo de construcción de la pieza se almacenó en una memoria SD que posteriormente se introdujo a la impresora, la calidad de la impresión se estableció en un valor estándar y la densidad de llenado fue del 25 %, con un patrón del tipo línea, la temperatura del extrusor se estableció a 220 °C.

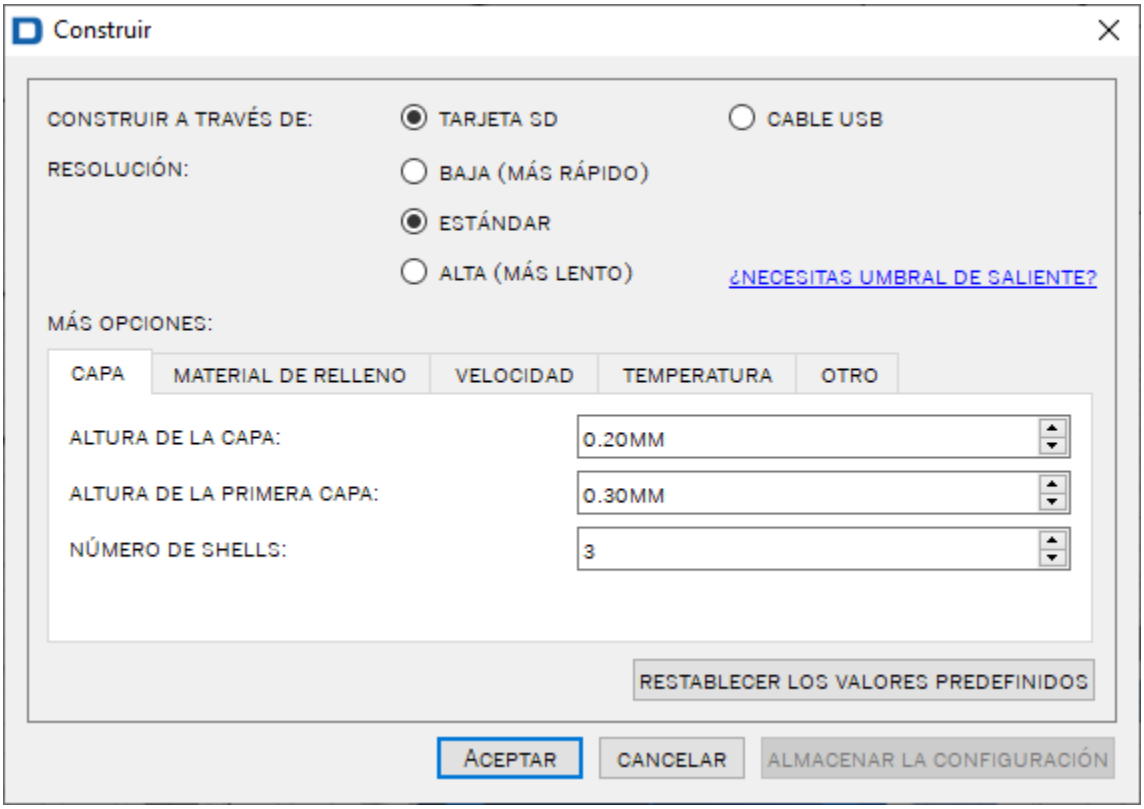

Figura 2.63 Configuración de la impresión

Una vez terminada la configuración se dio clic en 'Aceptar' y el archivo de construcción se almacenó en la memoria SD utilizada, cabe destacar que este proceso se realizó con cada una de las piezas. Una vez que se introdujo la memoria SD en la impresora se seleccionó la pieza a imprimir y se dio clic a la opción 'Inicio', de esta manera comenzó el proceso de impresión de las piezas.

2.7 Integración del sistema

Una vez que se imprimieron todas las piezas se procedió a armar el sistema de posicionamiento de la siguiente manera:

1.- En la base se introdujo uno de los servomotores MG995 justo en la ranura dedicada a ello, el cable del servomotor se sacó al exterior utilizando la perforación de la base y el disco de unión se atornilló a las cuatro ranuras de la pieza 'tapa'.

2.- Ahora, utilizando un Arduino Uno se movió el servomotor de la base a una posición de 90 ° y en esta posición se colocó el disco de unión de la tapa al eje del servomotor y con un tornillo se unieron tapa y base. Para comprobar una correcta unión se movió el servomotor a diferentes posiciones, comprobando que la tapa fuese capaz de girar en cualquier dirección.

3.- En la tapa móvil se posicionaron los dos servomotores restantes en sus ranuras correspondientes y por medio de tornillería se ajustaron a la tapa, de tal manera que al girar la tapa los servomotores no se desajustaran. Los cables de estos motores se sacaron de la tapa haciendo uso de las ranuras en la base de esta, es así que al girar la tapa los cables no se enreden.

4.- Los servomotores de la tapa se colocaron en la posición de 90 ° con la ayuda de una tarjeta Arduino Uno. Acto seguido, en el brazo se atornillaron los discos de unión de los demás servomotores y con mucho cuidado se introdujo el eje de cada servomotor en el orificio de cada disco. Ahora se hicieron algunas pruebas de movimiento para verificar que la posición tilt se mueve de forma correcta, cabe resaltar que los ángulos de movimiento que se introducen a un servomotor son el complemento del otro, pues está colocado en sentido contrario al otro.

5.- Ahora, la antena se unió al brazo por medio de tornillería y se procedió a conectar cada servomotor en conjunto con el driver, el driver se conectó a una fuente de alimentación conmutada y se realizaron las conexiones necesarias entre el NodeMCU y el driver.

6.- Finalmente, se encendió el servidor web además de la Raspberry Pi, y utilizando las credenciales de un usuario se realizó la conexión con el servidor VPN, logrando así acceder al servidor web desde una red externa, de esta manera se pudo cambiar la ubicación de la antena en torno al azimut y su inclinación con respecto al eje Z, por medio de unos controles del tipo Slider, desde una ubicación remota sin acceso directa a la red local en la que se aloja el servidor web.

Figura 2.64 Sistema de posicionamiento armado, diferentes vistas

En la siguiente página en la Figura 2.65 se muestran las conexiones eléctricas que se realizaron entre los componentes a fin de ofrecer un mayor panorama de la composición del prototipo.

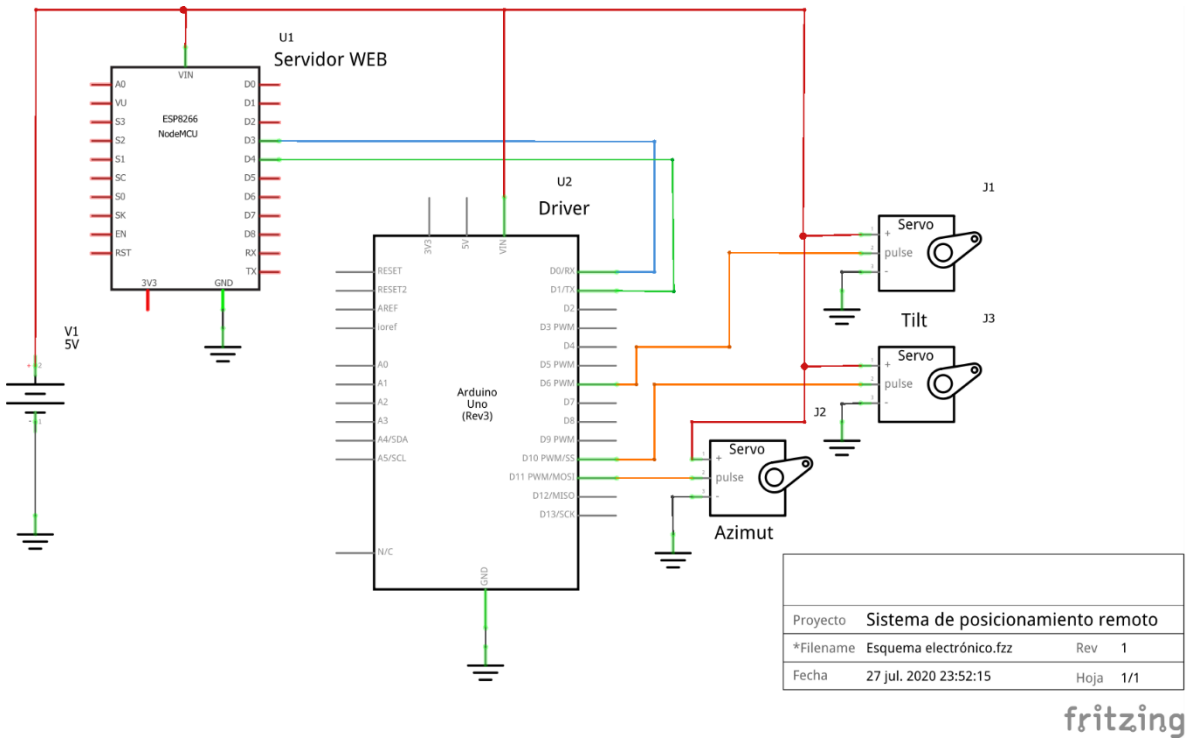

Figura 2.65 Diagrama esquemático del proyecto

2.8 Costo del desarrollo del prototipo

En la siguiente tabla se muestra la cotización del material cotizado necesario para la elaboración del prototipo, tomando como base la arquitectura del proyecto previamente establecida con la realización del esquema de la figura 2.1. En la mencionada tabla (Tabla 2.2) se especifica el nombre, cantidad, costo, proveedor y costo de envío del material.

Tabla 2.1 Lista de materiales para el prototipo

Concepto	Cantidad	Costo unitario	Proveedor	Costo de Envío	Costo subtotal
Servomotor MG995 con disco metálico	4	$ 125.00	MVELECTRNICA	$ 100.00	$ 500.00
Fuente conmutada de 5V 3A	1	$ 115.00	RANMEX	$ 90.00	$ 115.00
Paquete de cables jumper 20 cm H-M	1	$ 74.00	RANMEX	$ 93.00	$ 74.00
Paquete de cable jumper 20 cm M-M	1	$ 74.00	RANMEX	$ 93.00	$ 74.00
Rollo de filamento PLA	1	$ 650.00	MC3DMEXICO	$ 0.00	$ 650.00
Tornillería	1	$ 100.00	LOCAL	$ 0.00	$ 100.00
Rasberry Pi B 3	1	$ 1400.00	COJO2093534	$ 0.00	$ 1400.00
Fuente de alimentación Raspberry Pi 3	1	$ 290.00	LURKAN	$ 0.00	$ 290.00
Micro SD 32 GB	1	$ 129.00	ADATA	$ 90.00	$ 129.00
NodeMCU ESP 8266	1	$ 170.00	COSISMO	$ 90.00	$ 170.00
Subtotal:				$ 556.00	$ 3,502.00
Total:					**$ 4,058.00**

CAPÍTULO III: RESULTADOS

3.1 Sistema terminado e integrado

3.1.1 Modelo de la antena

Tras la metodología y el procedimiento descrito en el capítulo anterior, se obtuvo como producto final la maqueta/modelo de la antena de telecomunicaciones. Dicha maqueta posee dos grados de libertad, correspondientes a los movimientos 'Tilt' y 'Azimut'. El movimiento 'Azimut' es accionado por un solo servomotor y el movimiento 'Tilt' se acciona por dos servomotores, tal como se deja ver en la Figura 3.1

Figura 3.1 Vista superior del modelo de la antena

3.1.2 Montaje e integración del servidor con el modelo de la antena

En la Figura 3.2 se tiene una fotografía con el NodeMCU (el servidor que aloja a la aplicación web para controlar remotamente las magnitudes 'Tilt' y 'Azimut del modelo de la antena) conectado al Arduino UNO (que mediante comunicación serial con el NodeMCU hace el papel de drive para controlar el movmiento de los servomotores que accionan al modelo de la antena, en función de las peticiones del

usuario). En la misma figura se aprecian a los dos componentes en completo funcionamiento, energizados y comunicandose entre sí

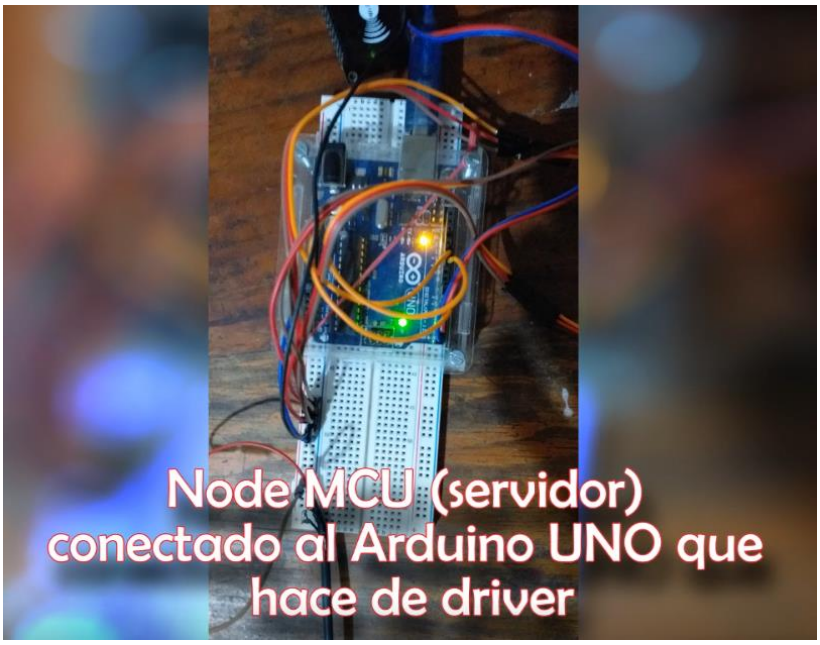

Figura 3.2 NodeMCU conectado al Arduino UNO

3.1.3 Visualización de la aplicación web ejecutada en el servidor

Lo mostrado hasta ahora, compete a la parte física o de 'hardware' del sistema. Se ha presentado el resultado obtenido del diseño y construcción del modelo de la antena, y la integración del arduino UNO y el NodeMCU a nivel electrónico. Sin embargo, en este apartado se mostrarán unas cuantas capturas de pantalla de la interfaz de la aplicación web en pleno funcionamiento, así como una serie de otras imágenes que comprueban el funcionamiento del servidor VPN.

Así pues, en las figuras 3.3 y 3.4 se tienen capturas de pantalla de la aplicación web, siendo servida por el servidor en el navegador web de un smartphone.

En las figuras 3.5, 3.6 y 3.7 se pueden ver de igual forma capturas de pantalla del mismo dispositivo donde se ejecutó la aplicación web, sin embargo, dichas imágenes demuestran que dicho dispositivo esta conectado a la aplicación a través de la conexión VPN proveída por el servidor configurado previamente para este sistema.

Figura 3.3 Interfaz de la aplicación web parte 1

Figura 3.4 Interfaz de la aplicación web parte 2

Figura 3.5 Conexión al túnel VPN como cliente 'Leonardo' parte 1

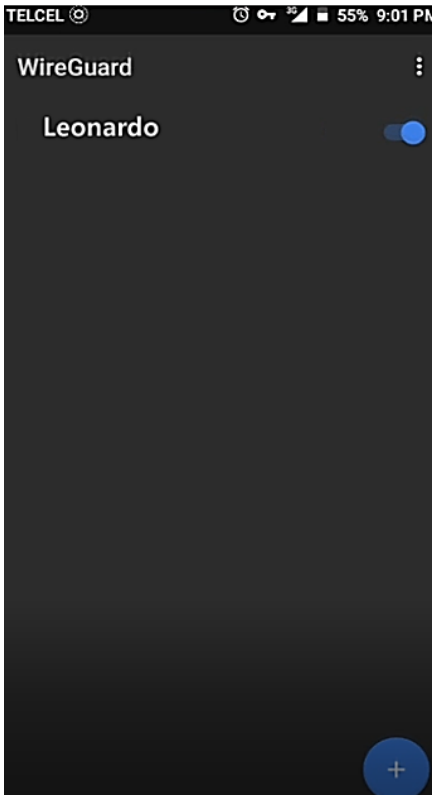

Figura 3.6 Conexión al túnel VPN como cliente 'Leonardo' parte 2

Nótese que tras colocar el switch a modo 'On', es decir tras conectarse al túnel VPN (Figura 3.6), aparece un icono en forma de llave en la barra de estado del dispositivo, mismo que se aprecia en la Figura 3.6. Esto indica que, bajo las credenciales correspondientes, se ha conectado el dispositivo de forma efectiva al servidor VPN por lo que todas las solicitudes del dispositivo (así como las respuestas a estas) se rutearán a través del túnel VPN hacia la red donde fue configurado el servidor VPN en la Raspberry, misma red donde se ubica el servidor que provee la aplicación web. En la Figura 3.7 se presenta una captura donde se está accediendo a la aplicación web, pero conectado al servidor VPN desde una red ajena a donde se ubica el servidor. Para comprobar lo anterior dicho, observe cómo en la barra de estado del dispositivo en las figuras 3.3 y 3.4, donde la aplicación web se utiliza en la misma red del servidor, el modo de conexión del dispositivo es por Wi-Fi, sin embargo, en la Figura 3.7 la barra de estado muestra que el medio de conexión es por datos móviles, lo que significa que el teléfono móvil está conectado a la aplicación por una red externa gracias al túnel VPN.

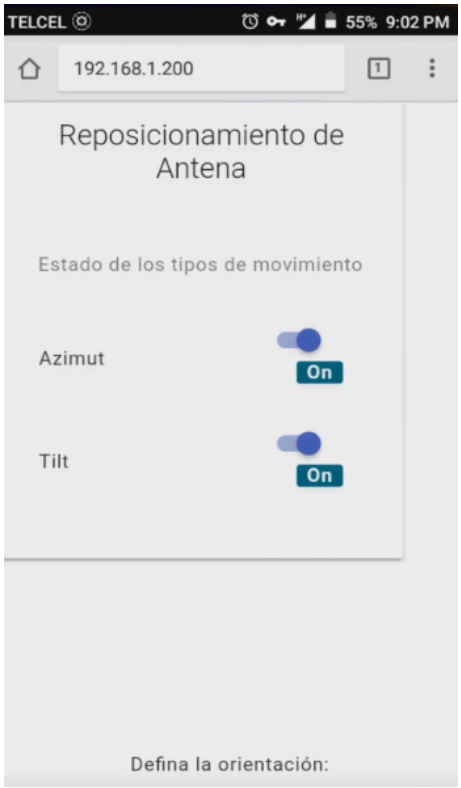

Figura 3.7 Uso de la aplicación web desde una red externa

3.2 Pruebas de latencia

En la sección 3.1 del presente capítulo se presentaron resultados superficiales del sistema, cuestiones acerca de cómo quedó la construcción del modelo de la antena. La ejecución de la aplicación web, el uso del servidor VPN, etc. No obstante, el parámetro de especial interés que permite realizar la afirmación de si los objetivos del sistema se cumplieron o no, es la latencia entre el cliente y el servidor. Es decir, el tiempo que transcurre entre que un usuario realiza una petición de ajuste de cualquiera de los ejes de orientación de la antena (Azimut y Tilt) a través de la interfaz de la página web, hasta que el servidor, o sea el Node MCU, recibe la petición y la procesa. Uno de los objetivos que se busca es que el cambio o ajuste de los ejes de orientación sea efectuado en tiempo real, tanto desde la red local como desde una red externa. El hecho de que se requiera una respuesta en tiempo real no quiere decir que instantáneamente el NodeMCU pueda procesar la petición tan pronto como el usuario la solicitó, pero sí que el tiempo de duración de este proceso sea tan pequeño que llegue a ser lo suficientemente imperceptible para verlo como algo instantáneo. Para conocer este tiempo de respuesta o latencia, se hizo una leve modificación al código del servidor del NodeMCU que se explicará en breve.

Para esta explicación de la obtención de uno de los parámetros de los resultados, se hará referencia constantemente a las secciones 2.4.2 y 2.4.5 donde se detalla el funcionamiento de los Websockets, ficheros Json y el funcionamiento del Backend del lado del cliente en JavaScript respectivamente. En JavaScript hay un método llamado performance.now(), el cual devuelve el tiempo de origen, es decir el tiempo trascurrido desde que un usuario cargó la página web de la interfaz de control, pues durante ese proceso creó el contexto de ejecución del fichero JavaScript. El tiempo lo devuelve en milisegundos con una precisión de 5 microsegundos, su sintaxis es la siguiente:

```
t = performance.now()
```

Así pues, como se observa en la Figura 3.8, esta función se llamó en todas las funciones que se ejecutan cuando el usuario interactúa a través de la interfaz web,

ya sea si mueve los switches, los sliders, o si presiona el botón ‘Default’. En cualquier caso, se almacena en una variable llamada ‘start’ el tiempo en milisegundos cuando se realizó determinada petición y se envía como un dato más dentro del archivo Json que contiene la demás información de la petición del cliente, algo que en las secciones 2.4.3 y 2.4.5 ya se había comentado

```
function setAngle(id, angle)
{
   var start = performance.now();
   updateSliderText(id, angle);

   let data = {
      command : "setAngle",
      id: id,
      angle: angle,
      time: start
   }

  let json = JSON.stringify(data);
  connection.send(json);
}

function setDefault(id)
{
         var start = performance.now();
         document.getElementById('slider-angle-1').value = 80;
         document.getElementById('slider-text-angle-1').value = 80;
         document.getElementById('slider-angle-2').value = 90;
         document.getElementById('slider-text-angle-2').value = 90;
         let data = {
                command : "setDefault",
                id: "1",
                angle: "90",
                time: start
        }

  let json = JSON.stringify(data);
  connection.send(json);
}
```

Figura 3.8 Backend del cliente

Ahora toca situarse en la programación del servidor, donde en la API del mismo, se escribió una función para enviarle información al cliente (Figura 3.9), es decir establecer una comunicación servidor-cliente. Esta función es llamada cada vez que el servidor recibe una petición del cliente y que corresponda con las peticiones planteadas en la misma API, como se ve en la figura 3.10. En pocas palabras, el servidor enviara un mensaje al Backend del cliente, de que ha recibido su petición, esto se conoce con el nombre de ‘eco’.

```
void updateState()
{
  String response;
  StaticJsonDocument<300> doc;
  doc["time"] = "tiempo";
  serializeJson(doc, response);
  ws.textAll(response);
 /*
```

Figura 3.9 Función que envía mensaje de recibido al cliente

```
void ProcessRequest(AsyncWebSocketClient * client, String request){
  Serial.println(request);
  StaticJsonDocument<200> doc;
  DeserializationError error = deserializeJson(doc, request);
  if (error) {return;}

  String command = doc["command"];
  if(command == "setEje") {
    updateState();
    setEje(doc["id"], (bool)doc["status"]);
    }
  else if(command == "setAngle"){
    updateState();
    setAngle(doc["id"], (int)doc["angle"]);}
  else if(command == "setDefault"){
    updateState();
    setDefault(doc["id"]);}
}
```

Figura 3.10 Llamada de la función 'updateState' tras cada petición recibida del cliente

Este mensaje se puede considerar como una simple verificación de que el servidor recibió la petición.

Volviendo al backend del cliente, en la Figura 3.11 se ve cómo se definió una variable 'end' que almacena el tiempo que devuelve la función performance.now(). Esta vez la función fue llamada, para esa variable, una única vez, es decir cuando se reciben mensajes del servidor como se ve en la Figura 3.11. Con ello se tiene el tiempo exacto en el que el servidor contestó al cliente de que ha recibido su petición.

```
connection.onmessage = function (e) {
    var end = performance.now()
    console.log('Received from server: ', e.data);
    //processReceived(e.data);
    let data1 = {
      time: end
    }
    let json = JSON.stringify(data1);
    connection.send(json);
};
```

Figura 3.11 Toma y envío del tiempo exacto de cuando se recibió el mensaje del servidor

Entonces, momento de repasar. El cliente interactúa con los elementos de la interfaz web, al hacerlo envía peticiones al servidor correspondientes con el elemento que esté manipulando. Al momento que el usuario realiza esto, se toma el tiempo transcurrido en milisegundos desde que éste cargó por primera vez la página. El servidor recibe la petición y devuelve al cliente un aviso de 'recibido', el backend detecta el mensaje del servidor, y guarda el tiempo en que sucedió este evento en la variable 'end'. Si ambas variables de tiempo (start y end) son enviadas al servidor y se imprimen en el monitor serial para conocer su valor (Figura 3.12), solo queda restar ambos tiempos y se tiene el tiempo de respuesta o latencia de la comunicación cliente-servidor con una precisión de ±5 microsegundos.

```
%V%{"command":"setEje","id":"I1","status":false,"time":4275542.475000001}
{"time":4275632.32}
```

Figura 3.12 Impresión de los tiempos de envío y recepción de la petición del cliente

En la Tabla 3.1 se muestran las 10 pruebas realizadas con este método para un cliente conectado dentro de la red local, en la misma tabla se presenta el promedio del tiempo de latencia basado en las diez pruebas efectuadas. De manera similar, en la Tabla 3.2 se tienen diez pruebas de latencia de comunicación con el servidor salvo que estas se realizaron para un cliente conectado desde una red externa a través de la conexión segura VPN. Con el fin de que el lector pueda comparar fácilmente los datos obtenidos en las pruebas de latencia bajo las dos condiciones de conexión al servidor creado en el NodeMCU, es que en la Figura 3.13 se presenta

una gráfica que sintetiza la información presentada en las tablas 3.1 y 3.2 de manera visual.

Tabla 3.1 Pruebas de latencia para un cliente conectado en la red local

	Latencia (ms)
Prueba 1	89.845
Prueba 2	61.835
Prueba 3	97.79
Prueba 4	258.985
Prueba 5	20.685
Prueba 6	9.29
Prueba 7	55.32
Prueba 8	44.975
Prueba 9	80.37
Prueba 10	57.565
Promedio	77.366 ms

Tabla 3.2 Pruebas de latencia para un cliente conectado desde una red externa

	Latencia (ms)
Prueba 1	198.715
Prueba 2	113.945
Prueba 3	131.35
Prueba 4	148.4
Prueba 5	190.655
Prueba 6	110.14
Prueba 7	514.545
Prueba 8	109.485
Prueba 9	134.75
Prueba 10	128.055
Promedio	178.004 ms

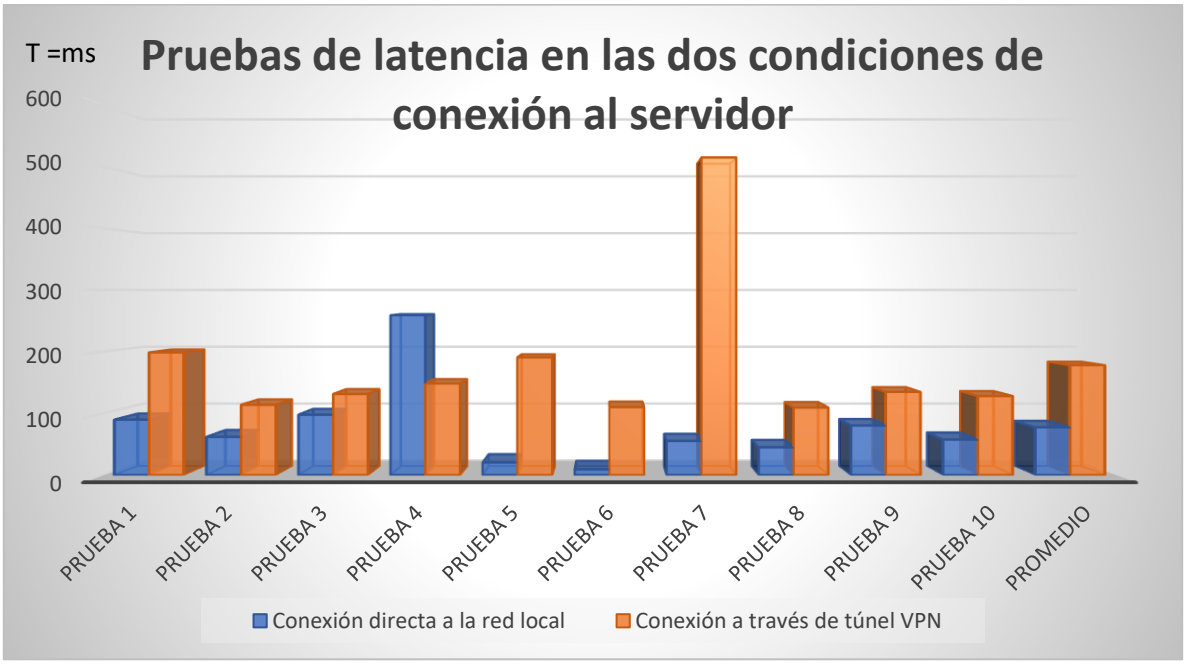

Figura 3.13 Gráfica comparativa de los resultados de las pruebas de latencia

Como se puede ver, las distintas pruebas de latencia entre cliente-servidor cuando el cliente está conectado en la propia red local del servidor, devolvieron valores bajos de latencia, salvo un pico alto donde se llegó a tener hasta 258 ms. Sin embargo, a pesar de ello, el promedio entre las 10 pruebas fue de 77.366 ms, lo cual refleja muy buenos resultados de latencia, no obstante, el sistema de reposicionamiento de la antena está pensado para ser usado de manera remota, es decir que muy rara vez un cliente se conectará desde la misma red local del servidor para controlar los ejes azimut y tilt de la antena, por lo que los resultados más relevantes, son los de la Tabla 3.2. Se observa cómo la latencia promedio es poco más del doble que en los resultados de las pruebas mostrados en la Tabla 3.1. Esto es algo que se esperaba puesto que la conexión a través de un túnel, es decir de una conexión VPN para acceder a una red local desde una red externa, añade más tiempo en la latencia debido al redireccionamiento. Aun así, el resultado de las pruebas dio 178 ms de latencia como promedio, lo cual aun así no está mal para cumplir el objetivo de control en tiempo real de los ejes de la antena. Es importante indicar que, con una conexión a internet rápida y estable por parte del cliente, se pueden conseguir tiempos de hasta 100 ms de latencia.

CAPÍTULO IV: CONCLUSIÓN Y RECOMENDACIONES

Por medio de este prototipo de sistema se garantiza que un operador pueda modificar en tiempo real de manera segura y remota a través de una aplicación sencilla, la inclinación y el azimut de una antena de telecomunicaciones para aumentar la cobertura de red de dicha antena.

Al realizar el proyecto se pudo observar que las redes privadas virtuales (VPN) son una herramienta muy poderosa en la ingeniería pues permite extender una red local a lo largo del internet, logrando que se pueda acceder a interfaces de monitores e instrumentación de forma remota bajo un esquema de seguridad que es producto de una conexión punto a punto entre un servidor y un cliente VPN. De esta manera una persona en otra parte del mundo puede observar el funcionamiento de un proceso y manipularlo de forma segura sin la posibilidad de la irrupción de terceros mal intencionados.

Los objetivos iniciales del proyecto se cumplieron de forma satisfactoria, ya que mediante una conexión VPN fue posible enlazarse a la interfaz de control del sistema, representada por medio de un servidor web local implementado con la tarjeta NodeMCU desde una conexión remota, garantizando así el posicionamiento adecuado de la antena a distancia y de forma segura, pues al estar alojado el servidor en una red local, no corre el riesgo de sufrir ataques de terceros a lo largo de internet. Por medio de este sistema se garantiza que un operador, ubicado en un centro de operaciones, pueda modificar la inclinación y el azimut de una antena de telecomunicaciones para aumentar la cobertura de red de dicha antena de forma remota y segura.

Se puede aumentar la robustez del sistema propuesto en este proyecto, mediante la incorporación de actuadores de mayor precisión o bien utilizando un driver dedicado como el PCA9685, pues, aunque al inicio se planteó utilizar éste, por las condiciones en las que se realizó el proyecto, se optó por implementar una tarjeta Arduino Uno como driver para controlar los motores. Otro aspecto a mejorar es el servidor web, que puede implementarse en las nuevas tarjetas ESP32, cuya velocidad de procesamiento permite reducir la latencia a la hora de realizar

peticiones al servidor, permitiendo incluso tener más dispositivos clientes conectados al mismo tiempo.

CAPÍTULO V: BIBLIOGRAFÍA

Aznar, Á. C., Roca, L. J., Casals, J. M., Robert, J. R., & Boris, S. B. (1998). *Antenas.* Barcelona: Edicions UPC.

BBVA. (23 de Marzo de 2016). *BBVA API Market*. Recuperado el 27 de Junio de 2020, de BBVA API Market: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

Crist, E. F., & Keijser, J. J. (2015). *Mastering OpenVPN.* Birmingham: Packt Publishing.

Hootsuite, We Are Social (2020). Mobile Users vs Mobile Connections [infografía]. Datareportal. https://datareportal.com/reports/digital-2020-global-digital-overview

Hootsuite, We Are Social (2021). Mexico [infografía]. Datareportal. https://datareportal.com/reports/digital-2021-mexico

Huidobro, J. M. (2013). Antenas de telecomunicaciones. *Revista digital de ACTA*, 1 - 11.

Instituto Nacional de Estadística y Geografía. (2019). *Encuesta Nacional sobre Disponibilidad y Uso de Tecnologías de la Información en los Hogares (ENDUTIH) 2019*. Recuperado el 13 de Mayo de 2020, de inegi.org.mx: https://www.inegi.org.mx/programas/dutih/2019/

Laet, G. D., & G.Schauwers. (2005). *Network security fundamentals.* Cisco press.

Mansilla, C. G., Brandau, E. M., & Morineau, N. F. (2007). Antenas inteligentes y su desempeño en redes wireless. *Sintesis Tecnológica, 3*(2), 97-109. doi:10.4206/sint.tecnol.2007.v3n2-05

Mozila Developers Network. (2018). *MDN Web Docs*. Recuperado el 27 de Junio de 2020, de MDN Web Docs: https://developer.mozilla.org/es/docs/Web/API/WebSockets_API

Mozilla Developers Network. (2015). *MDN Web Docs*. Recuperado el 27 de Junio de 2020, de MDN Web Docs: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Qu%C3%A9_es_JavaScript

NodeMcu Team. (2015). *NodeMcu*. Recuperado el 27 de Junio de 2020, de NodeMcu: https://www.nodemcu.com/index_en.html

NO-IP. (25 de Junio de 2020). *Dynamic DNS*. Recuperado el 26 de Junio de 2020, de noip.com: https://www.noip.com/remote-access

NO-IP. (25 de Junio de 2020). *What is DNS?* Obtenido de noip.com: https://www.noip.com/what-is-dns

OpenVPN INC. (2020). *VPN Software Solutions & Services for Business*. Recuperado el 25 de Junio de 2020, de openvpn.net: https://openvpn.net/

Ortega, A. J. (2017). *Fabricación digital.* Madrid: Ministerio de Educación de España.

Raspberry Pi Foundation. (2020). *Raspberry Pi 3 Model B*. Recuperado el 25 de Junio de 2020, de raspberrypi.org: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

SolidWorks. (2020). *Software de diseño CAD 3D.* Recuperado el 26 de Junio de 2020, de solidworks.com: https://www.solidworks.com/es

Telconomia. (28 de Febrero de 2021). *México: mercado móvil al 4T20*. Obtenido de telconomia.com: https://telconomia.com/mexico-mercado-movil-al-4t20-y-resultados-de-2020/

Travieso, T. B. (2016). *Sistema de posicionamiento autónomo y teleguiado de antenas, para receptores de señales de radiofrecuencia con dispositivos de bajo coste SDR, enfocados en la observación de cuerpos celestes y otras fuentes de radio.* Cádiz: Universidad de Cádiz.

Universidad de Alicante. (5 de noviembre de 2008). *Universidad de Alicante.* Recuperado el 27 de Junio de 2020, de Universidad de Alicante: https://si.ua.es/es/documentacion/mootools/documentos/pdf/json.pdf

Wensheng, L., Yi, L., & Yanming, Z. (2010). The Design of RET Control System Based on AISG2.0. *IEEE*. doi: 10.1109/CISE.2010.5676920