Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

SISTEMA DE DIRECCIÓN ELÉCTRICA EN VEHICULOS MEDIANTE TRANSMISIÓN INALÁMBRICA

Autor:

Seisdedos Caballero, Gonzalo

Tutor(es):

Herrero de Lucas, Luis Carlos

Departamento de Tecnología Electrónica

Valladolid, Julio 2025

Agradecimientos

A mi familia, por apoyarme en todos los momentos durante estos años. Sin ellos no hubiese sido posible continuar hasta el final de este largo recorrido.

A mi grupo de amigos, por sus consejos, su paciencia y su ayuda sobretodo en la etapa final de este proyecto.

A Icíar, gracias por aportarme la confianza que en algunos momentos me ha faltado.

Gracias a Luis Carlos por permitirme desarrollar este TFG a su lado.

Resumen

El sistema de dirección de un automóvil es conocido por todo el mundo y necesario en todos los automóviles. Sin embargo, y a pesar del desarrollo tecnológico, es un sistema que no ha sufrido cambios muy significantes en los últimos años. Las nuevas líneas de desarrollo apuestan por un nuevo sistema llamado “Steer by wire” en el que la columna de dirección se sustituye por una serie de cables.

En este proyecto trataremos de dar un paso más allá y eliminar de forma completa cualquier tipo de conexión entre el volante y el eje delantero donde se encuentra la transmisión

Palabras clave

Sistema de dirección, microcontrolador ESP32, protocolo de comunicación ESP-NOW, motor lineal, encoder

Abstract

The steering system of an automobile is widely known and essential in all vehicles. However, despite technological advancements, it is a system that has not undergone significant changes in recent years. New development trends are moving toward a system known as “Steer by Wire,” in which the steering column is replaced by a series of cables.

In this project, we aim to take it a step further and completely eliminate any physical connection between the steering wheel and the front axle where the transmission is located.

Key Words

Steering system, microcontroller ESP32, ESP NOW, lineal motor, encoder

Índice de Contenido

Índice de Figuras

Índice de tablas

ÍNDICE DE ECUACIONES

SIGLAS Y ACRÓNIMOS

3D: Tres dimensiones ("*Three Dimensions*")

AES: Estándar de Cifrado Avanzado ("*Advanced Encryption Standard*")

ALU: Unidad Aritmético-Lógica ("*Arithmetic Logic Unit*")

ATX: Tecnología Avanzada Extendida ("*Advanced Technology eXtended*")

BA: Distancia entre los puntos A y B

BLE: Bluethooth de baja energía ("*Bluetooth Low Energy*")

CLK: Señal de Reloj ("*Clock*")

DC: Corriente Continua ("*Direct Current*")

GND: Conexión a Tierra ("*Ground*")

IDE: Entorno de Desarrollo Integrado

IEC: Comisión Electrotécnica Internacional ("*International Electrotechnical Commission")*

IMU: Unidad de Medición Inercial ("*Inertial Measurement Unit*")

L: Longitud del tirante de dirección

LED: Diodo Emisor de Luz ("*Light Emitting Diode*")

MAC: Control de Acceso al Medio ("*Media Access Control*")

Modo AP: Modo Punto de Acceso ("*Access Point Mode*")

Modo STA: Modo Estación ("*Station Mode*")

PWM: Modulación por Ancho de Pulso ("*Pulse Width Modulation*")

RAM: Memoria de Acceso Aleatorio ("*Random Access Memory*")

RPM: Revoluciones por minuto

TTL: Lógica Transistor Transistor ("*Transistor-Transistor Logic*")

UART: transmisor-receptor asíncrono universal ("*Universal Asynchronous Receiver-Transmitter*")

Vcc: Alimentación del dispositivo

Capítulo 1:

Introducción, motivación y objetivos

CAPÍTULO 1. Introducción, motivaciones y objetivos

1.1 Introducción

En la actualidad la industria del automóvil es una de las más influyentes e importantes a nivel mundial. Países como Alemania dependen fuertemente de esta industria con la presencia de empresas tan prestigiosas como Audi, Porsche, BMW o Volkswagen. Estas compañías cada día apuestan más por la introducción de electrónica en ámbitos tanto de seguridad como de comodidad. Con el paso de los años, los automóviles presentan más elementos electrónicos como pantallas de multientretenimiento, sistemas de conexión con Smartphone además de los nuevos servicios de seguridad que, en su mayoría, están basados en la electrónica.

Hablando de sistemas en la carrocería, actualmente todos los vehículos contienen sensores de posición que ayudan al aparcamiento o cámaras de seguridad que incluso son capaces de grabar a las personas que causen algún desperfecto al coche. Se ha de destacar toda la electrónica que se encarga del correcto funcionamiento de sistemas tan críticos como el motor. Elementos como sensores de temperatura, de presión, de velocidad o de oxígeno son esenciales para su correcto funcionamiento.

En este proyecto nos centraremos en el sistema de dirección del automóvil y trataremos de realizarlo de forma electrónica con el objetivo de eliminar la columna de dirección por completo. Esto puede solucionar problemas de desajuste en las ruedas, implementar mejoras en la conducción y sobre todo dar una mayor flexibilidad a la hora de diseño del coche debido a que, al suprimir un elemento que cruza desde el eje delantero del coche hasta el volante, se obtiene un mayor espacio para el resto de los componentes del vehículo.

Los primeros intentos de eliminar esta columna de dirección fueron por parte de Infinity en su modelo Q50. En este modelo se elimina parcialmente la columna de dirección, sin embargo, sigue existiendo conexión entre el volante y el eje delantero como se muestra en la *figura 1*.

***Figura 1**: Sistema de dirección del Infinity Q50*

El desarrollo de estas direcciones ha continuado hasta la actualidad donde fabricantes como Toyota implementan un nuevo sistema que se hace llamar "*Steer by Wire*" en el que la conexión se realiza únicamente a través de cables (*figura [2]*).

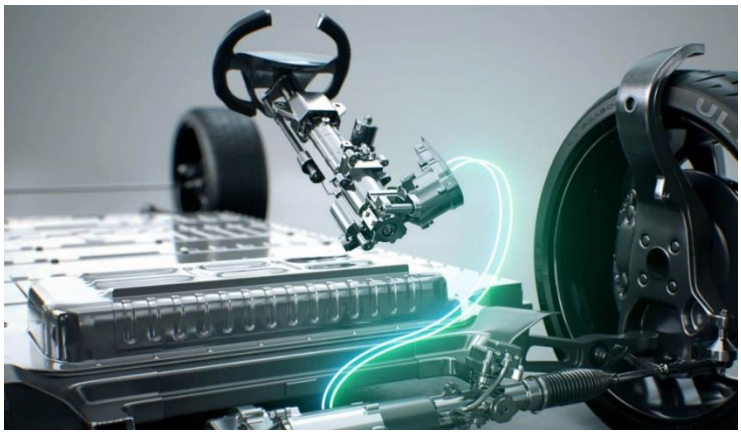


***Figura 2**: Steer by Wire implementado por Toyota*

Está línea de investigación y desarrollo está siendo implementada por otros fabricantes como Peugeot.
Sin embargo, en este TFG se hará una conexión completamente inalámbrica, que se espera que sea la nueva tendencia a pesar de que todavía no haya sido implementado por casi ningún fabricante

1.2 Motivaciones

La motivación de este trabajo de fin de grado surge de varias inquietudes personales, así como de mi interés en la electrónica y en la forma en que puede mejorar nuestras vidas.
A todo esto, se suma las ganas de intentar aportar un sistema novedoso que sea útil para distintas empresas en el sector del automóvil y el propósito de adquirir nuevos conocimientos que me puedan ayudar a la hora de continuar con mi carrera

En definitiva, la principal motivación es aprender mediante el desarrollo de un proyecto mecatrónico de forma que pueda ampliar mis conocimientos en las disciplinas que involucra la mecatrónica.

1.3 Objetivos

El objetivo de este TFG es la realización en físico del sistema de dirección a escala. En este proyecto, los encargados de girar las ruedas serán unos motores lineales que se moverán en base a la información que reciban del volante.

Como todo proyecto, todo comienza con una etapa de análisis en donde trataremos de determinar la mejor solución posible. Igualmente se mostrarán todas las posibilidades. En la figura 3 se muestra un esquema de desarrollo:

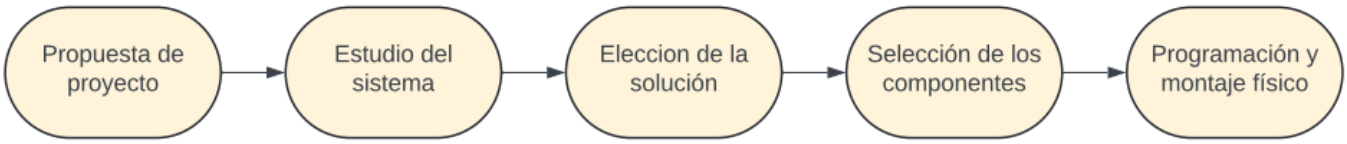


***Figura 3**: Esquema de desarrollo del proyecto*

El objetivo principal es la realización a escala del sistema de dirección, pero, existen otra serie de objetivos secundarios que han sido necesarios para el correcto desarrollo del proyecto:

- Entender conceptos clave: Ha sido necesario entender todos los conceptos tanto de mecánica como de electrónica o programación para el correcto desarrollo de todo el proyecto
- Establecimiento de hitos: El proyecto se ha desarrollado a través de hitos, siendo necesaria la finalización de uno para el comienzo del siguiente. De esta forma se ha adquirido una velocidad de desarrollo constante.
- Uso de softwares: Se ha necesitado de distintos softwares como Arduino IDE o Autodesk Inventor para la realización del sistema.
- Diseño 3D: Ha sido necesario para la realización de piezas en modelado 3D
- Lectura y comprensión de datasheet: Para la elección de los componentes electrónicos elegidos se ha necesitado leer toda la información de los datasheet para comprobar si el elemento era compatible con el sistema. Toda la información relacionada con

voltajes de alimentación, corrientes necesarias y procesamiento de datos viene reflejada en los datasheet.

- Verificación del funcionamiento: Una vez ensamblado, se ha procedido a la verificación del funcionamiento del sistema de forma que podamos verificar si cumple las especificaciones iniciales
- Comunicación inalámbrica: Se ha requerido la comprensión total del concepto de comunicación inalámbrica, así como todos los elementos relacionados con este proceso como las direcciones MAC, el emisor y receptor entre otros.

Ya hemos mencionado que el proyecto se trata del montaje de un sistema mecatrónico por lo que es importante explicar en qué consiste esta disciplina.

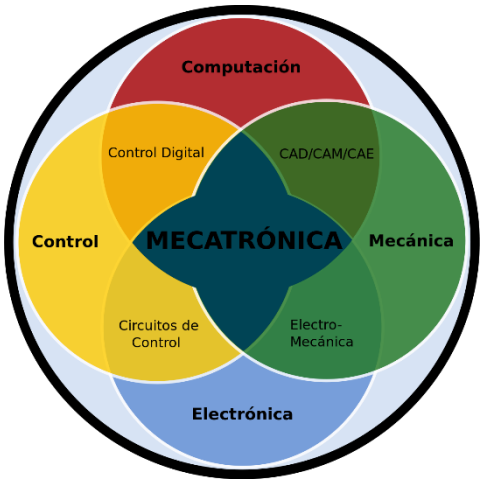


***Figura 4**: Esquema de la disciplina Mecatrónica*

El término mecatrónica surge en el siglo en 1969 por parte la empresa japonesa "Yaskawa Electric Co". La ingeniería mecatrónica es la disciplina que integra conocimientos de electrónica, control, mecánica y computación. La mecatrónica se encarga de solventar problemas como:

- Automatización de procesos
- Mejoras en consumos energéticos
- Desarrollo de nuevos productos
- Unión de la mecánica y electrónica en proyectos

1.4 Fundamentos previos

Para el correcto desarrollo de este trabajo se ha requerido de una sólida formación en campos como mecánica, electrónica o comunicaciones industriales. A continuación, se muestran las bases académicas en las que se fundamenta este proyecto.

- **Expresión gráfica en la ingeniería**: Imprescindible para entender todos los planos de componentes, así como para la elaboración de los planos de las piezas diseñadas
- **Matemáticas**: Como en cualquier proyecto de ingeniería es básico tener un buen control sobre las matemáticas para poder resolver los distintos problemas que puedan surgir
- **Mecánica para máquinas y mecanismos**: Relacionada con la mecánica clásica, es fundamental para poder entender el funcionamiento del sistema de dirección desarrollado
- **Fundamentos de electrónica**: Necesaria para poder tener un buen control sobre los componentes electrónicos, así como la comprensión de las gráficas de los datasheet
- **Electrónica de potencia**: Fundamental para el diseño de PWM, así como el uso de semiconductores y convertidores.
- **Comunicaciones industriales**: Necesario para entender el protocolo de comunicación entre microprocesadores, así como conceptos de direcciones MAC.
- **Mecatrónica**: Me ha aportado la formación necesaria para el ensamblaje en físico de un proyecto funcional.
- **Electrónica Digital y Analógica**: En estas asignaturas se nos ha aportado los conocimientos necesarios para la comprensión de conceptos como los bits, así como los cálculos sobre tensiones e intensidades en circuitos electrónicos.

Capítulo 2:

Conceptos previos

CAPÍTULO 2. Conceptos previos

Previo a comenzar a describir de forma completa el sistema desarrollado, se cree imprescindible hablar de los conceptos básicos sobre la dirección en automóviles, así como la historia de este sistema.

2.1 ¿Qué es el sistema de dirección de un coche?

Se trata de uno de los sistemas más básicos en un automóvil. Su misión es dirigir la orientación de las ruedas en función a lo que dicte el volante para obtener una determinada trayectoria.

Las direcciones tradicionales están formadas por una serie de elementos fundamentales que son:

- **Volante**: Dirigido por el conductor. A través de él se da la orientación deseada a las ruedas. Va conectado de forma directa a la columna de dirección.
- **Columna de dirección**: Formada por un árbol articulado conectado al mecanismo de dirección, se encarga de transmitir el giro del volante al mecanismo de dirección del vehículo. Con el paso de los años se ha ido mejorando, implementando funciones de seguridad como las columnas retractiles que en caso de impacto son capaces de doblarse por varios puntos evitando así un posible impacto con el conductor.

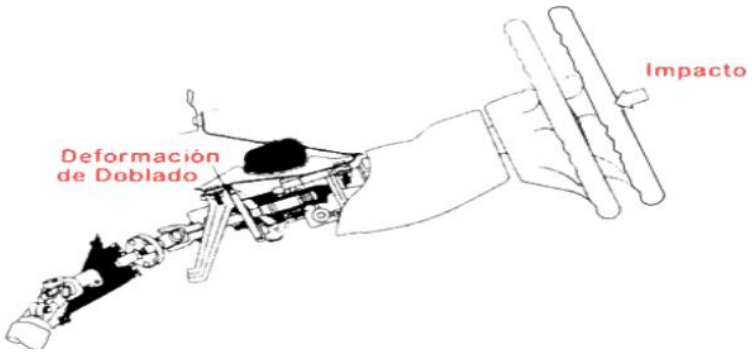


***Figura 5**: Columna de dirección retráctil*

- **Mecanismo de dirección**: Es el encargado de transmitir la orientación a las ruedas a través del árbol que conforma la columna de dirección. Existen distintos tipos de mecanismos de dirección que han ido evolucionando a lo largo de los años:
  - **Dirección mecánica**: Trabaja con la fuerza que aplica el conductor. El mecanismo de dirección puede ser de muy distintos tipos, aunque en este tipo de dirección lo más común es un mecanismo de piñón cremallera. Sin embargo, existen otros mecanismos como el de tornillo sinfín y rodillo, tornillo sinfín y tuerca o la dirección asistida por cremallera. Se muestran algunos de estos tipos en la *figura 6,7 y 8*

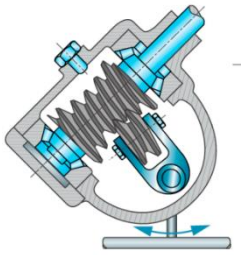


***Figura 6**: Tornillo sinfín cilíndrico*

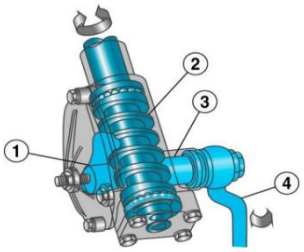


***Figura 7**: Tornillo sinfín y dedo*

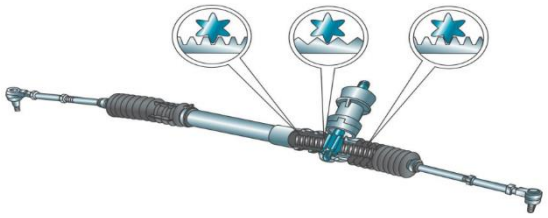


***Figura 8**: Dirección piñón cremallera*

  - **Dirección hidráulica**: En este caso, a la fuerza aplicada por el conductor se suma la fuerza que realiza un fluido. Mediante el uso de una bomba, se recircula un fluido de forma que ayude a la transmisión del movimiento a las ruedas. Es una de las más eficientes y por ello

de las más usadas. Su esquema se muestra en la *figura 9.*

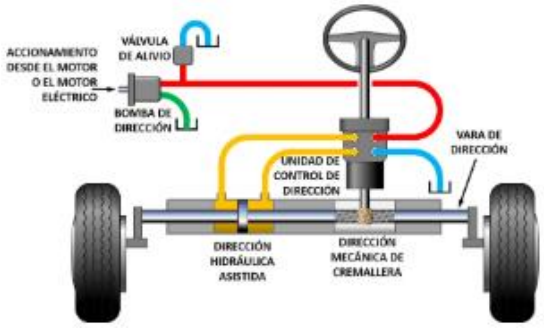


***Figura 9**: Esquema de funcionamiento de una dirección hidráulica*

- **Dirección electrohidráulica:** El principio de este tipo de dirección es el mismo que en la dirección hidráulica con la diferencia de que la presión que se necesita para hacer circular el fluido proviene del propio motor.
- **Dirección electromecánica:** Este tipo de dirección es la evolución de la dirección mecánica con cremallera con la diferencia de que, en este tipo, el encargado de darle la asistencia al movimiento a través de la cremallera es un motor eléctrico.

❖ **Tirantes de dirección:** Son los encargados de transmitir la orientación que las ruedas han de adquirir desde el mecanismo de dirección hasta las propias ruedas. Existen infinidad de mecanismos de tirantes diferentes para cada proyecto. Se trata de un mecanismo crucial para el sistema de dirección por lo que requiere un análisis del movimiento muy exhaustivo. La trirantería de un automóvil está formada principalmente por las bielas o palancas de mando, la barra de mando, las manguetas y las rótulas que permiten la rotación.

El mecanismo de tirantes diseñado para este proyecto se presenta en los siguientes capítulos.

Estos son los elementos que se consideran básicos para el diseño de una correcta dirección. El conjunto de todos ellos forma el *Sistema de dirección*.

2.2 Historia y evolución de la dirección

Al contrario de lo que se puede pensar, los primeros sistemas de dirección no surgen ligados al primer automóvil. La invención del primer automóvil se atribuye a Carl Benz en el año 1886, cuando realizó la patente del "vehículo motorizado con motor de gasolina". Sin embargo, los primeros sistemas de dirección surgen en el año 1605 a través de los ejes orientables. Estos permitían a los animales orientar los carros de los que tiraban hacia un sentido u otro (*figura 10*).

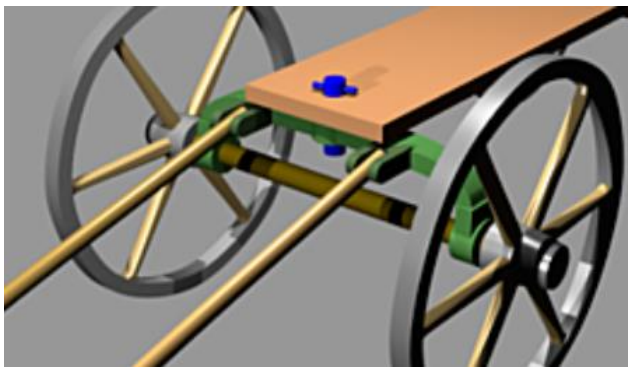


***Figura 10**: Sistema de dirección de un carro en 1605*

En este tipo de direcciones lo que se hacía era girar de forma conjunta todo el eje delantero del carro, permitiendo así la orientación de este.
Durante los siguientes siglos este sistema se fue mejorando, aumentando la calidad de los materiales y pensando formas más ingeniosas de conseguir el giro deseado.

El desarrollo de la dirección está muy ligado a la máquina de vapor. Inventada en 1765 por James Watt (patentada en 1769) ha sido uno de los mayores descubrimientos en la historia, pero su aplicación a la automoción se realiza en el año 1769 por Nicolas-Joseph Cugnot que consiguió crear el primer vehículo autopropulsado de la historia (figura 11)

***Figura 11**: Vehículo autopropulsado por Nicolas-Joseph Cugnot*

Es en este punto cuando surge la necesidad de mejorar el sistema de dirección anteriormente mostrado de forma que fuese aplicable al sistema creado por Cugnot. En 1817 George Lenkesperger presenta los planos para una dirección en los que las ruedas giran independientes al eje delantero (a diferencia de la dirección de 1605) permitiendo un giro más suave y fluido (figura 12)

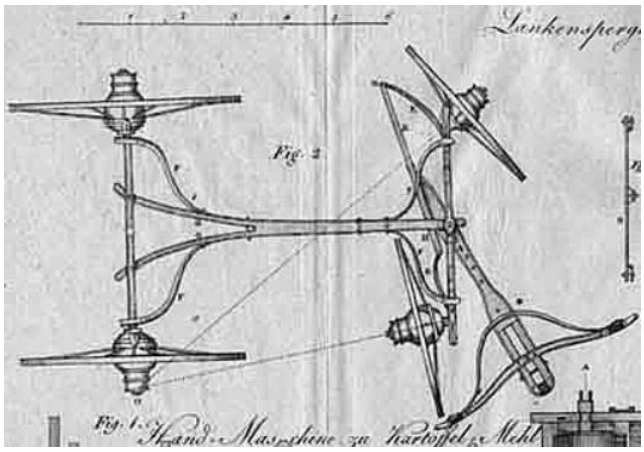


***Figura 12**: Planos de la dirección diseñada por Lenkesperger*

En estos planos se puede ver ya una dirección más similar a la que conocemos hoy en día, en donde el movimiento de orientación es realizado por las ruedas y no por el eje delantero. No obstante, en la práctica resultaba un sistema que perdía adherencia al suelo haciéndolo así poco fiable y seguro.

La geometría se mejoró en 1818 mediante el principio de Ackerman, que se sigue usando en la actualidad. Este principio nos dice que los ejes de giro de las ruedas deben de coincidir en un mismo punto llamado “Centro instantáneo de giro”.

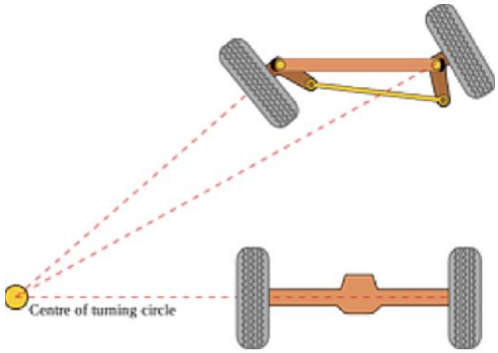


***Figura 13**: Principio de Ackerman*

Este principio se sigue cumpliendo en las direcciones actuales. Este tipo de direcciones se fue aplicando a los distintos vehículos, primero a vapor y más tarde a los vehículos de gasolina a lo largo de los años, pero el siguiente gran cambio surgió en 1885 por parte de Benz, presentando la dirección basada en un sistema de piñón cremallera.

***Figura 14**: Sistema piñón cremallera de Benz*

Estos sistemas han ido evolucionando a lo largo de los años dando lugar a las direcciones que conocemos hoy en día, primero con la aparición de direcciones de bolas circulantes en 1928, pasando por las direcciones hidráulicas hasta llegar la implementación de la gestión electrónica en la dirección por parte de Honda en 1991.

***Figura 15**: Dirección de bolas circulantes*

***Figura 16**: Implementación de electrónica en dirección, Honda 1991*

2.3 Presencia de sistemas de dirección en sistemas tecnológicos actuales

A pesar de que el término "Sistema de dirección" está ligado de forma inconfundible con la industria de la automoción se trata de un mecanismo que aparece en muchos más diseños de los que conocemos.

La tendencia actual es automatizar todo aquel proceso que sea posible, siendo necesario el uso de robots o de sistemas capaces de realizar una acción de forma autónoma. Muchos de estos sistemas son capaces de desplazarse por distintos entornos precisando de la capacidad de orientarse en una dirección u otra. Robots de exploración como la misión "Rover" necesita de un buen sistema de dirección, capaz de poder orientar de forma completa todo el sistema en cualquier dirección posible

***Figura 17**: Detalle de la dirección de la misión "Rover"*

La dirección que se desarrolla en este TFG sería también válida para proyectos de este tipo.

Capítulo 3:

Descripción del sistema y componentes electrónicos usados

CAPÍTULO 3. Descripción del sistema y componentes electrónicos usados

Como se ha explicado anteriormente, el objetivo de este proyecto es obtener como resultado un sistema de dirección en el que, sin hacer uso de la columna de dirección, seamos capaces de orientar las ruedas en un sentido u otro.

El sistema planteado consta de 2 partes fundamentales: un volante cuya posición queremos registrar y unos actuadores capaces de darnos una respuesta ante un ángulo de giro en el volante. Sin embargo, no usaremos ningún elemento de conexión entre el volante y los actuadores, en su lugar, usaremos 2 microprocesadores ESP32 capaces realizar el envío y la recepción de datos. Por ello, nuestro sistema consta de 3 etapas fundamentales:

1. Registro del ángulo de giro en el volante
2. Envío y recepción de la información entre microprocesadores
3. Acción de los actuadores

A continuación, y separado por las etapas anteriormente nombradas, describiremos la función de cada una de ellas, así como los componentes que se han usado y la razón de su uso.

3.1 Registro del ángulo de giro en el volante

El objetivo de esta etapa es ser capaz de leer la posición del volante en tiempo real, de forma que si una persona rota el volante un cierto ángulo se tenga la información durante el proceso y podamos ir enviando esa información a los actuadores.

3.1.1 Ángulo de giro en el volante y de orientación en las ruedas

Llamamos ángulo de giro en el volante al giro que produce un usuario en el volante durante la conducción. De forma habitual, los volantes tienen un ángulo máximo de rotación en una vuelta y media, siendo esta medida equivalente a 540 grados tanto en giro horario como en giro antihorario. Cuando el volante esté en esta posición, las ruedas han de llegar a su ángulo de giro máximo (cumpliendo el principio de Ackerman mencionado en el apartado 2.2).

Por su parte, el ángulo de orientación en las ruedas hace referencia al ángulo que existe entre la rueda en una posición determinada y en la posición inicial. Este ángulo suele tener su máximo entre los 40 y 45 grados en los coches

convencionales actuales. Sin embargo, en este proyecto el ángulo máximo se ha fijado en 50 grados pudiendo ampliarse más en caso de que se quiera usar para otra aplicación desligada de la automoción.

En las *figuras 18 y 19* se muestra en rojo los ángulos a los que nos referimos

***Figura 18**: ángulo de rotación en el volante*

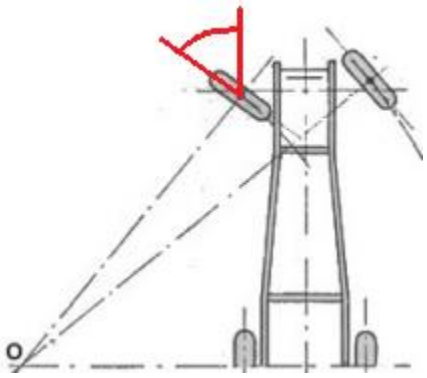


***Figura 19**: Ángulo de orientación en las ruedas*

De esta forma, hemos de establecer una relación entre el ángulo que se gira en el volante y el ángulo de orientación. Por ello realizamos la siguiente operación:

$$\frac{Ángulo\ de\ giro\ en\ el\ volante}{Ángulo\ de\ orientación\ en\ las\ ruedas} = \frac{540\ (grados)}{50\ (grados)} = 10.8\ grados \quad [1]$$

Por cada 10.8 grados de rotación que tome el volante, el ángulo de orientación de las ruedas debe de incrementarse en 1 grado. Para una mejor visualización, pasamos esta relación a Excel para obtener la siguiente gráfica:

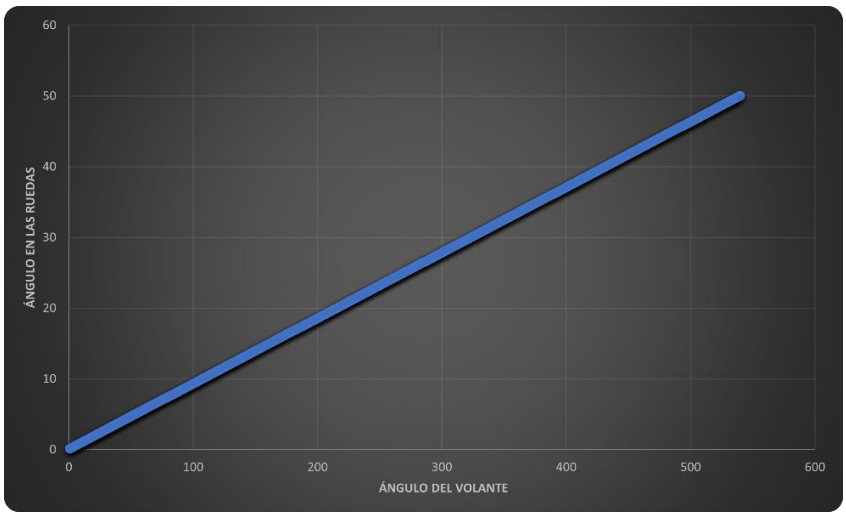


***Figura 20**: Gráfica de relación entre ángulos de giro y orientación*

La recta que sigue es: $$y(ángulo\ ruedas) = x(ángulo\ volante) * \frac{50}{540}$$ [2]

3.1.2 Estudio de las soluciones para registro del ángulo de giro del volante

1) Uso de una "Unidad de medición inercia" (IMU)

¿Qué es una IMU?

Se trata de un dispositivo electrónico capaz de dar información acerca de la velocidad, aceleración, fuerzas y orientación. Es un sistema muy utilizado en aeronáutica debido a que proporciona información sobre la situación en la que se encuentra la aeronave dando los valores de cabeceo, alabeo y la guiñada (los conocidos ángulos llamados Roll, Pitch y Yaw)

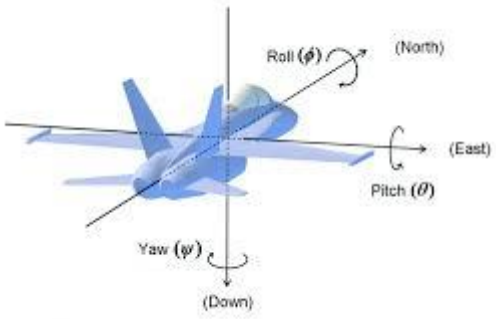


***Figura 21**: Ángulos roll, pitch y yaw en una aeronave*

Siendo el dispositivo de unas dimensiones no excesivas, se consideró su uso en este proyecto de forma que fuésemos capaces de integrarlo dentro del propio volante de forma que, al rotar el volante, este dispositivo también rotase y nos diese su nueva orientación. El dispositivo que se estudió fue el siguiente:

SINAT-485

El Sinat-485 se trata de una IMU de pequeñas dimensiones, que permite el registro de variables como orientación en 3 ejes, aceleraciones y velocidades.

***Figura 22**: Imagen del Sinat-485*

Este equipo posee un pinout bastante sencillo con únicamente 4 cables a conectar, dos de ellos dedicados a la alimentación (Vcc y GND) y los otros dos dedicados a la transferencia de datos.

Los cables dedicados a la circulación de datos hacen uso del protocolo RS485 a diferencia de ESP32 que utiliza TTL/UART.

Incompatibilidad entre UART y RS485:

TTL es un tipo de señal lógica que varía entre 0 y 3.3 voltios (en el caso de ESP32). El tipo de comunicación de la que hace uso el ESP es la UART (se explicará de forma más detallada en siguientes apartados), un tipo de

comunicación asíncrona que usa únicamente 2 cables (RX y TX) siendo el método de conexión punto a punto.

Por su parte, RS485 hace uso de una conexión diferencial de entre -7 hasta 12 voltios. Este tipo de comunicación está orientada a entornos industriales a diferencia de UART que se usa en sensores y comunicaciones cortas

UART	RS485
Señal lógica (0-3.3V)	Señal diferencial (-7V a 12V)
2 cables (RX/TX + GND)	2-3 CABLES (A/B +GND (opcional)
Uso en sensores y comunicación corta	Uso en entornos industriales

***Tabla 1**: Tabla comparación TTL y RS485*

Por ello, en caso de haber usado este componente, hubiera sido necesario usar un elemento capaz de cambiar los datos de RS485 a TTL de forma que el ESP pueda hacer uso de ellos.

Junto a esto, las IMUs son equipos en los que la rotación máxima en un eje es de 360 grados, reiniciando la posición a 0 grados cuando se supera una vuelta completa, por lo que no es compatible con nuestro proyecto debido a que hemos de ser capaces de registrar hasta 540 grados. Aunque existan IMUs capaces de registrar ángulos acumulativos, se decidió descartar esta opción en busca de una mejor solución

Cabe destacar que el procesamiento de los datos que estos sistemas precisan es bastante más complejo que otras opciones, por lo que se descartó su uso

Se trató de buscar otras IMUs que fuesen acordes al proyecto como la MPU-9250, pero al igual que el SINAT-485, presenta algunas características que no hacen su uso adecuado para este proyecto

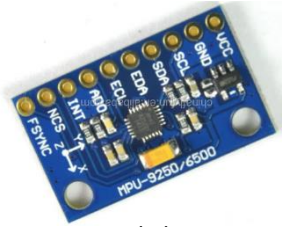


***Figura 23**: Imagen de la MPU-9250*

2) Potenciómetro de DFRobot "DFR0058"

El siguiente componente planteado fue el DFR0058, un tipo de potenciómetro que nos permite registrar hasta 3600 grados. Con esta característica solventamos el problema planteado por las IMUs de no poder registrar más allá de 360 grados. Este componente presenta un pinout muy sencillo con únicamente 3 pines:

Cable rojo: Alimentación

Cable azul: Señal con los datos

Cable negro: GND

***Figura 24**: Potenciómetro DFR0058*

Se trata de un dispositivo con plena compatibilidad con ESP32 por lo que podemos realizar la lectura de datos sin problema. Haciendo un conexionado muy sencillo y un código en Arduino IDE somos capaces de registrar el ángulo. Sin embargo, la lectura del sensor no es muy precisa y el ángulo cambia de forma constante, aunque se deje el potenciómetro en una determinada posición. Por esto, y buscando una mayor precisión, se ha descartado esta opción.

Finalmente, viendo la respuesta del DFR0058 y atendiendo a una mayor precisión se decidió comenzar a buscar un encoder de forma que pudiésemos registrar el ángulo y tener una buena precisión llegando a la siguiente solución que ha sido elegida solución final al problema de registrar la posición.

3) Codificador Incremental Rotativo 38S6G5-B-G24N AB

Este componente es un encoder rotativo capaz de proporcionar la posición en la que se encuentra.

Principio de funcionamiento:

En nuestro caso, el encoder 38S6G5-B-G24N AB es de tipo rotativo incremental fotoeléctrico. Esto quiere decir que su funcionamiento se basa en un disco perforado a través del que circula un haz de luz. Cuando se produce el giro del

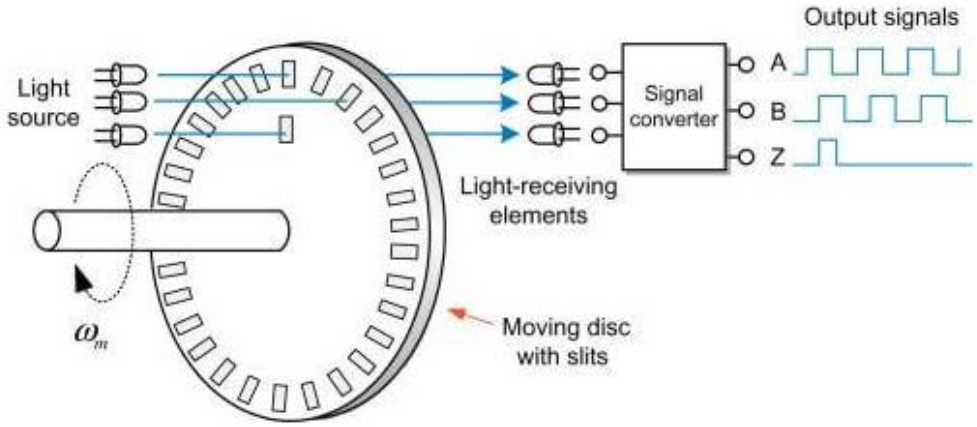


***Figura 25**: Principio de funcionamiento de codificador rotativo incremental*

disco el haz de luz es cortado generando así un pulso. De esta forma podemos obtener una señal cuadrada que contiene la información de la velocidad de giro y posición. El encoder escogido tiene una resolución de 1000 pulsos por revolución, lo que quiere decir que el disco perforado tiene 1000 perforaciones.

El dispositivo cuenta con un transistor a la salida de forma que este funciona como un interruptor que nos permite generar una onda cuadrada.

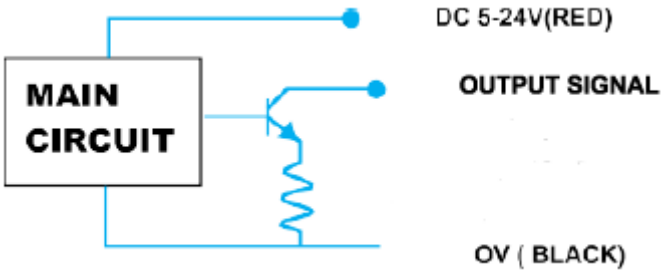


***Figura 25**: Esquema de funcionamiento de un encoder rotativo fotoelectrico*

Las etapas son las siguientes:

1) El circuito óptico interno detecta un cambio (por ejemplo, un agujero en el disco óptico). Se activa la base del transistor permitiendo que este conduzca y que el colector se conecte a 0V, por lo que la señal baja a 0V
2) El disco interrumpe el haz de luz por lo que el transistor se apaga. El conector queda abierto por lo que la resistencia PULL-UP (incluida en el

propio canal del ESP32) lleva la línea a +V creando así un flanco de subida.

Debido a esto, hemos de tener cuidado con el canal del ESP32 que usamos debido a que hay algunos que pueden no presentar resistencia PULL UP.

Pinout del 38S6G5-B-G24N AB

El pinout de nuestro encoder es el siguiente:

Color	Función
Negro	GND
Blanco	Datos A
Verde	Datos B
Rojo	alimentación

***Tabla 2**: Pinout del encoder 38S6G5-B-G24N AB*

Como se puede ver en la tabla anterior a diferencia de la opción del potenciómetro, disponemos de dos cables de datos. La función de esto es poder distinguir el giro horario del giro antihorario de forma que en caso de que el giro sea horario la señal en los datos A estará adelantada frente a la señal que circule por B. El proceso se muestra en la figura 27:

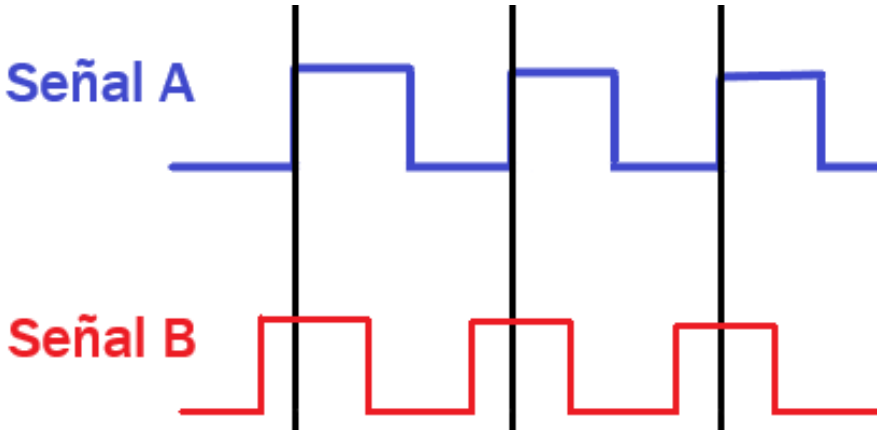


***Figura 27**: Señales en el encoder 38S6G5-B-G24N AB*

Como la señal A está adelantada respecto de B el giro ha de ser horario.

Para poder comenzar con las lecturas de datos del dispositivo, hemos de alimentar el dispositivo a cinco voltios y conectar el pin de GND. Por su parte, los pines de datos han de conectarse al ESP32. Se han elegido los pines 18 y 19 del ESP emisor para los datos A y B respectivamente.

Pruebas de funcionamiento

Una vez conocido los fundamentos teóricos que permiten el correcto funcionamiento del equipo, vamos a comprobar que el comportamiento del dispositivo es correcto. Para ello, hemos conectado los pines del dispositivo a un osciloscopio para poder ver las señales que transmiten, usando los 2 canales del osciloscopio para poder ver las señales de datos tanto A como B. Se han realizado varias pruebas para ver el comportamiento en distintos casos.

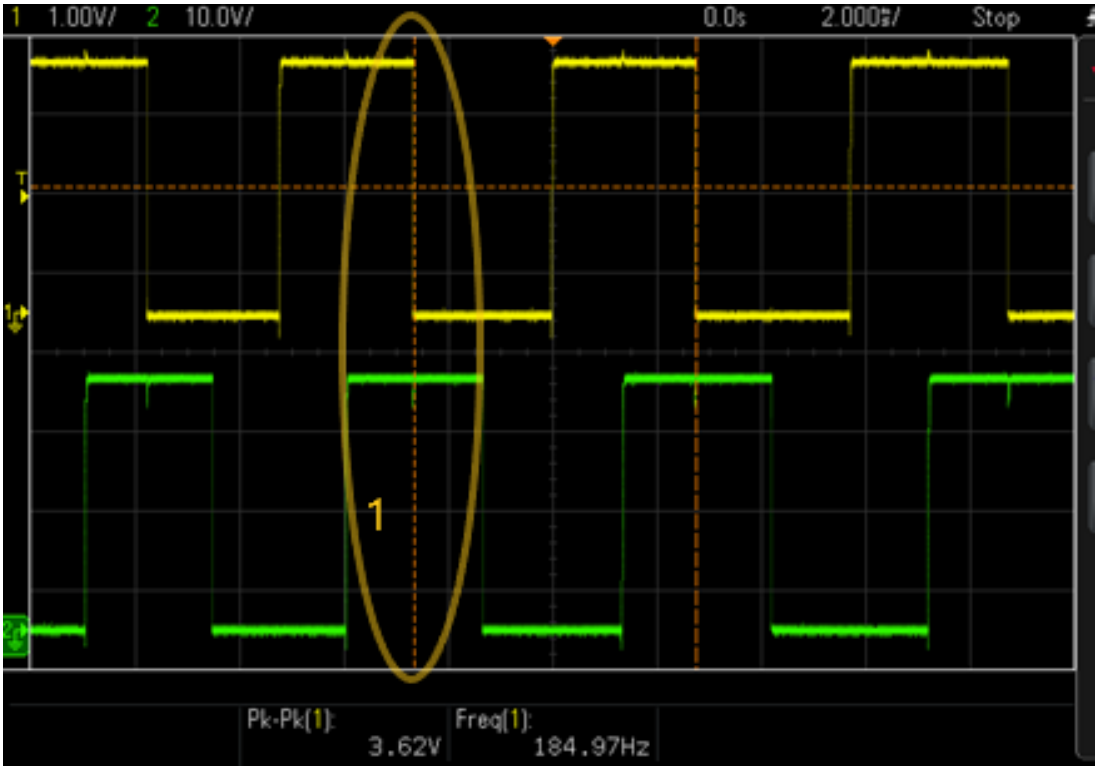


***Figura 28**: Captura de osciloscopio con giro horario*

Giro horario velocidad media: Para este caso, la visualización del osciloscopio es la mostrada en la figura [28].

En esta figura se ha delimitado con un óvalo 1 el momento en el que en la señal A se produce un flanco de bajada. En ese momento la señal B sigue en un nivel de tensión alto (lo que sería una 1 digital) y más tarde, se produce un flanco de bajada. Esto nos indica que el giro es horario.

Giro antihorario con velocidad media: En este caso el giro que se ha dado al vástago del encoder es el contrario que en el caso anterior. El osciloscopio ha mostrado lo que se ve en la figura [29]

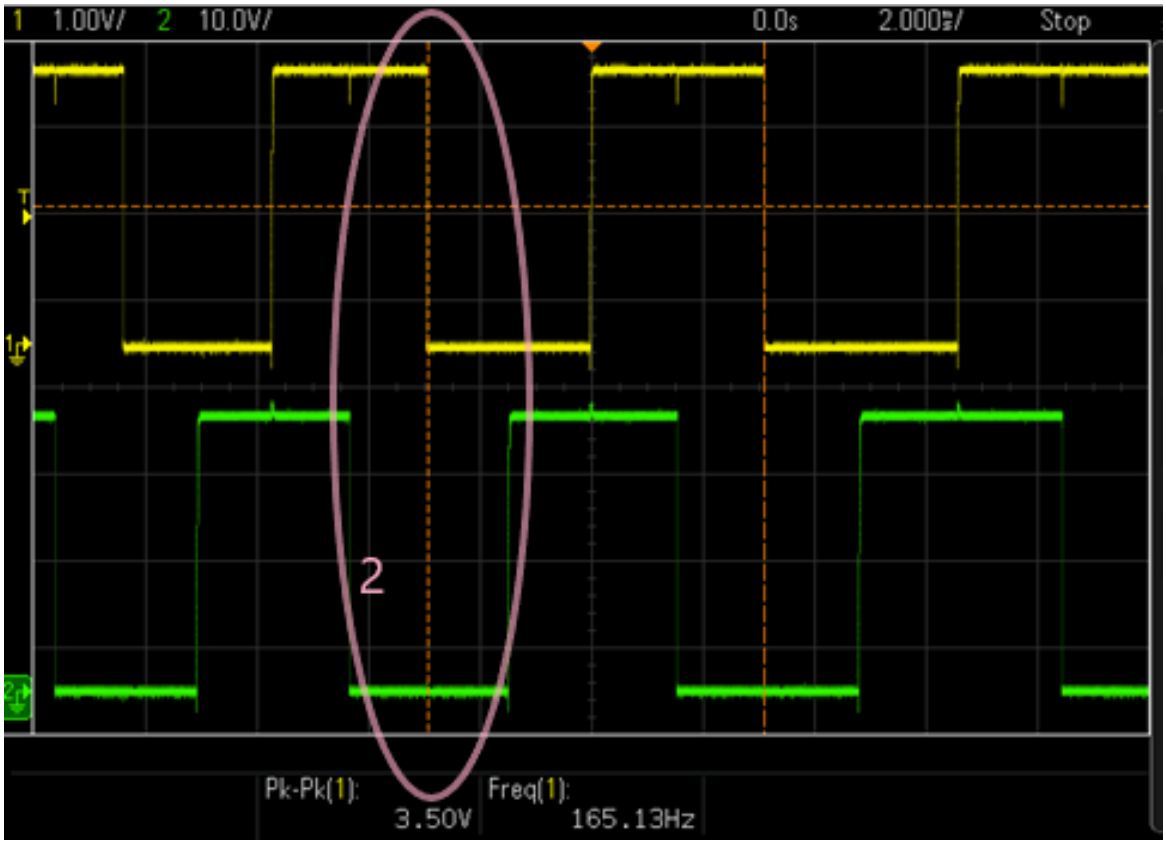


***Figura 29**: Captura de osciloscopio con giro antihorario*

Al igual que en el caso anterior, se ha rodeado (óvalo 2 en la figura [29]) el momento en el que la señal A realiza el flanco de bajada. A diferencia del caso anterior, en este momento la señal B se encuentra en un nivel de tensión bajo (a diferencia del giro horario) y el flanco de bajada se ha producido antes. Esto nos indica que el giro es antihorario.

Para poder obtener estas gráficas cabe decir que hay que mover el vástago del encoder, debido a que sino no se producirá ningún tipo de señal. La gráfica también dependerá de la velocidad con la que se mueva el vástago. En caso de que se produzca un giro de gran velocidad se obtendrán graficas más similares a la mostrada en la figura [30], donde se puede apreciar unas ondas con una frecuencia muy superior a las anteriormente mostradas

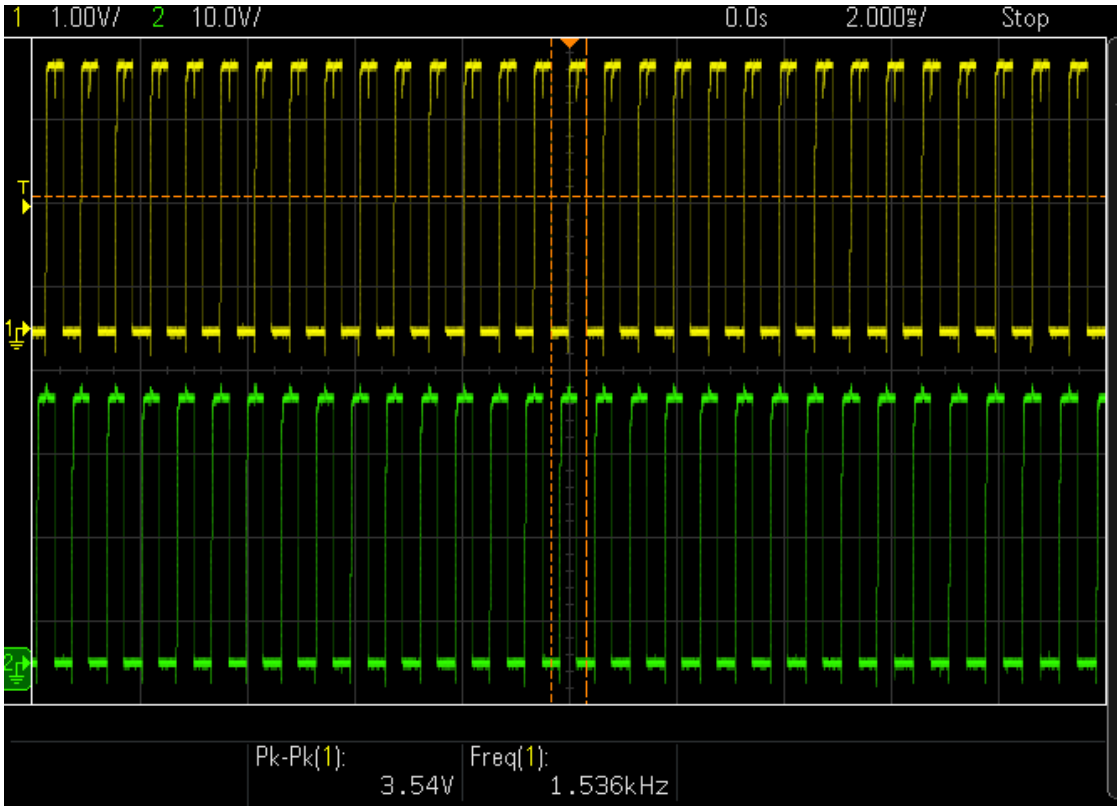


***Figura 30**: Captura de osciloscopio con velocidad alta*

Cableado y conexión al microcontrolador

Como se ha explicado anteriormente, hemos de conectar el dispositivo al microcontrolador para poder realizar el procesamiento de los datos, pero hemos de seleccionar de forma correcta el pin del microcontrolador para poder usar la resistencia pull up. Teniendo en cuenta esto, se ha decidido realizar el siguiente conexionado:

Color	Función	PIN DEL ESP32 (EMISOR)
Negro	GND	PIN GND
Blanco	Datos A	GPIO18
Verde	Datos B	GPIO19
Rojo	Vcc	5 V

***Tabla 3**: Conexión ESP emisor con Encoder*

Los datos procedentes de los cables A y B serán recibidos en los pines 18 y 19 del encoder para su posterior tratamiento. La recepción de los datos por parte del ESP no es un proceso complicado puesto que el microcontrolador se encarga de detectar únicamente los cambios de estado en las señales que recibe.

3.2 Envío y recepción de la información entre microprocesadores

Como se ha mencionado en anteriores apartados de este documento, se necesita un soporte para poder realizar una comunicación inalámbrica. La forma que se ha considerado adecuada para la realización de este proyecto está basada en el uso de 2 microprocesadores ESP32 y la comunicación ESP-NOW.

3.2.1 ¿Qué es ESP32?

ESP32 es un microcontrolador desarrollado por *Expressif Systems* muy usado en aplicaciones de IoT, automatización de procesos o control de motores entre otras aplicaciones.

Destaca por su bajo coste, su bajo consumo y sus buenas prestaciones. Permite el modelado de PWM, uso de funciones inalámbricas sin necesidad de estar conectado a WiFi y posibilidad de usar de forma simultánea funciones Bluetooth

Las características dependerán del modelo usado pues en base a esto obtenemos unas prestaciones u otras.

MODELO	CARACTERÍSTICAS PRINCIPAÑES
ESP32-WROOM-32	Versión estándar, WiFi y Bluetooth, 4 MB de Flash
ESP32-WROVER	Incluye PSRAM adicional (para aplicaciones más exigentes)
ESP32-S2	Solo WiFi, más seguro pero con un solo núcleo
ESP32-S3	WiFi + BLE 5.0, mayor rendimiento gráfico y de IA
ESP32-C3	Basado en RISC-V, más eficiente en energía, ideal para IoT

***Tabla 4**: Tabla con las características de distintos ESP32*

Usaremos la versión estándar de ESP, el ESP32-WROOM-32. Hemos de hablar un poco más en detalle de las características de estos controladores

Procesador

El ESP 32 cuenta con un procesador de doble núcleo basado en Tensilica Xtensa LX6, ofreciendo un gran rendimiento en tareas como el control en tiempo real, control de procesos o análisis de datos. El esquema de un procesador de este tipo es el que se muestra en la figura [31].

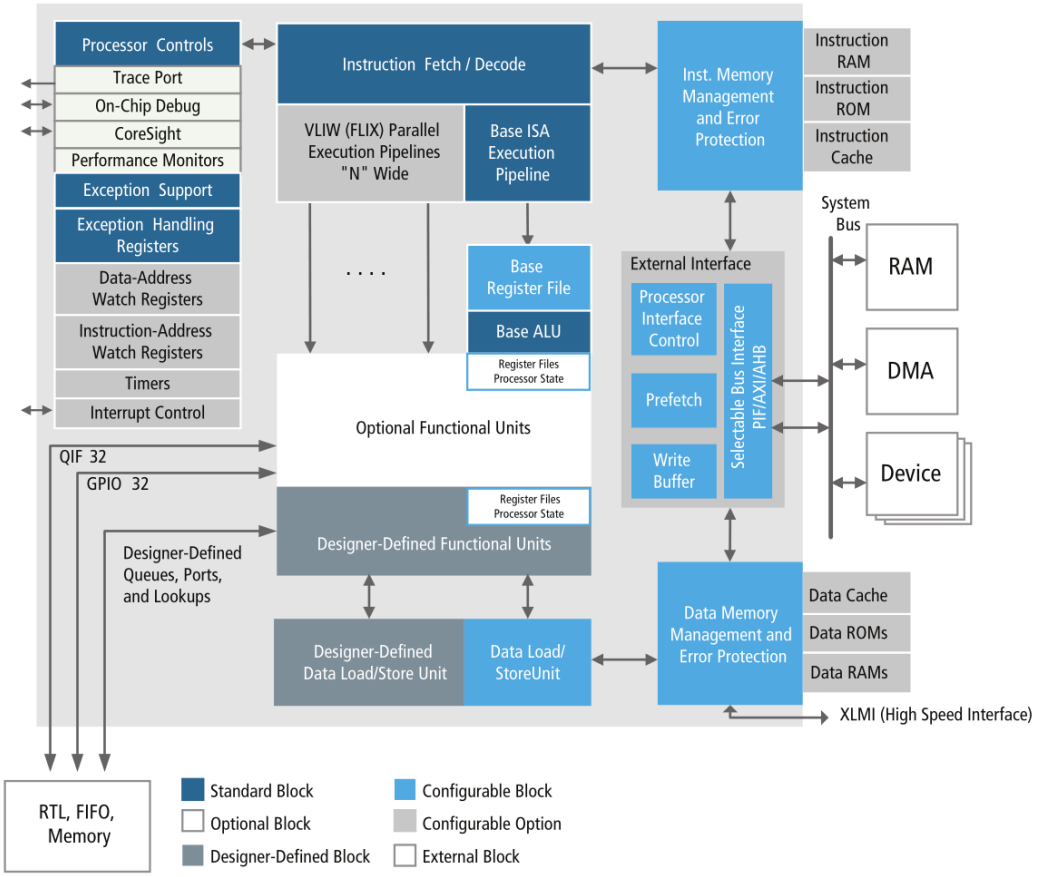


***Figura 31**: Esquema de arquitectura LX6*

En este diagrama (figura[31]) corresponde con la arquitectura LX6, no la de ESP 32 si bien las diferencias que presentan uno con otro no son significativas. Los bloques más importantes (y estándar en arquitectura LX6) de esta arquitectura se muestran en azul oscuro, siendo estos

- El ciclo de instrucción de 32 bits
- Un juego de 80 instrucciones básicas
- Una unidad aritmético-lógica de 32 bits (ALU)
- Soporte de excepciones de 32 bits

La arquitectura que implementa el ESP está basada en LX6 y cuenta con esos 4 bloques obligatorios.

Velocidad del reloj

El reloj es un componente esencial encargado de dictar el ritmo al que se ejecutan las operaciones (leer una instrucción, realizar una operación matemática o mover datos entre registros entre otras operaciones) en el

microcontrolador, funcionando como un metrónomo. El ESP32-WROOM-32 puede operar hasta a una velocidad de 240Mhz siendo esta velocidad de 240 millones de ciclos por segundo.

Memoria

El ESP32 cuenta con una memoria de RAM de hasta 520KB y una memoria flash de hasta 16 MB.

La *memoria RAM* (Random Access Memory) es una memoria volátil, lo que quiere decir que cuando se apaga o reinicia el ESP esta memoria se borra. Es usada para guardar variables o almacenar datos de ejecución de un proceso entre otras funciones. Se trata de una memoria muy rápida en procesos tanto de lectura como de escritura

La *memoria Flash,* a diferencia de la RAM, si que conserva datos aunque se apague el dispositivo (no es volátil) y es usada para almacenar el programa que se ejecuta. Es una memoria más lenta que la RAM y tiene ciclos limitados de escritura.

Conectividad

ESP32 es un controlador muy sólido en aspectos de conectividad inalámbrica.

- Incluye **conectividad Wi-Fi** compatible con el estándar IEEE 802.11b/g/n (encargado de regular como deben funcionar las redes inalámbricas Wi-Fi)
- Integra **Bluetooth v4.2** compatible con Bluetooth clásico y bluetooth low energy

Gracias a estas características el microcontrolador puede comunicarse de forma inalámbrica con una gran variedad de dispositivos, así como acceder a redes o intercambiar datos en tiempo real

Protocolo de comunicación

El protocolo de comunicación del que hace uso el ESP 32 es el UART. Se trata de un protocolo o conjunto de normas que rigen el intercambio de información entre 2 dispositivos. Hace uso de únicamente 2 hilos entre el transmisor y el receptor con el fin de transmitir y recibir información en las 2 direcciones. El transmisor de uno va conectado al receptor del otro (figura [32]).

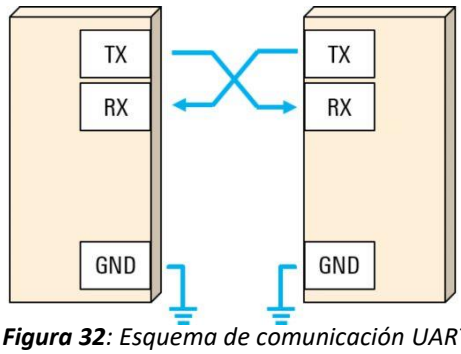


***Figura 32**: Esquema de comunicación UART*

La comunicación puede ser de 3 tipos:

- ***Simplex***: Los datos se envían en una sola dirección
- ***Semiduplex***: Cada extremo se comunica, pero solo 1 al mismo tiempo
- ***Dúplex completo***: Ambos extremos pueden transmitir de forma simultánea

Lo datos se transmiten en forma de tramas, cuyo formato se muestra a continuación:

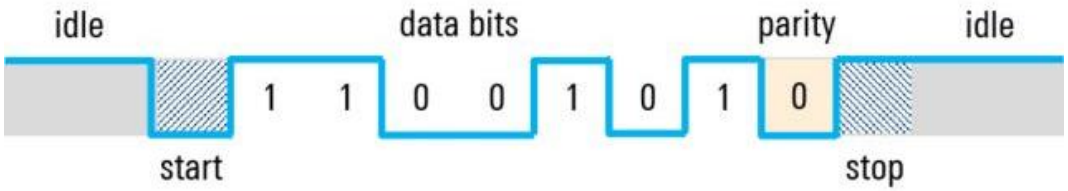


***Figura 33**: Ejemplo de trama UART*

Las tramas UART están formadas por bits de inicio, de parada, el conjunto de bits de datos y de forma opcional un bit de paridad.

Al tratarse de un tipo de comunicación asíncrona, el transmisor necesita marcar los datos que están llegando. De forma usual, el bit de inicio es una transmisión de estado alto a estado bajo seguido de forma inmediata de los bits de datos. Por el lado contrario, el bit de parada es o bien una transición al estado alto o de reposo o la permanencia en el estado alto por un tiempo de un bit adicional.

Los bits de datos son la información que se encuentra entre el bit de inicio y el bit de parada, siendo la longitud de estos mensajes de unos 7 u 8 bits

Por último, hablaremos del bit de paridad que se sitúa tras el bit de parada y su principal función es la detección de errores. En base al tipo de paridad tendrá un valor u otro.

- ❖ Paridad par: El número de unos que haya en la trama ha de ser par
- ❖ Paridad impar: El número de uno que haya en la trama ha de ser un número impar

Hemos de diferenciar 2 conceptos que se pueden llegar a confundir debido a que en muchas situaciones aparecen junto. Estos conceptos son TTL y UART.

UART es un tipo de comunicación (cuyas características se han definido anteriormente), mientras que TTL, por sus siglas "Transistor-Transistor-Logic", no es un protocolo de comunicación sino un tipo de señal lógica que usan muchos sistemas digitales. Cuando usamos UART, los pines TX (transmisión) y RX (recepción) hacen uso de señales TTL para comunicarse. Es decir, TTL define como se representan los datos a nivel eléctrico mientras que UART nos da la información de cómo se intercambian esos datos entre dispositivos

3.2.2 Pinout de ESP32-WROOM-32

Para la realización de este proyecto se ha decidido usar el modelo más general de ESP, el ESP32-WROOM-32. Este microcontrolador nos permite usar el protocolo de comunicación ESP NOW en el que se basará nuestra conexión inalámbrica. Este ESP pone a nuestra disposición el siguiente pinout:

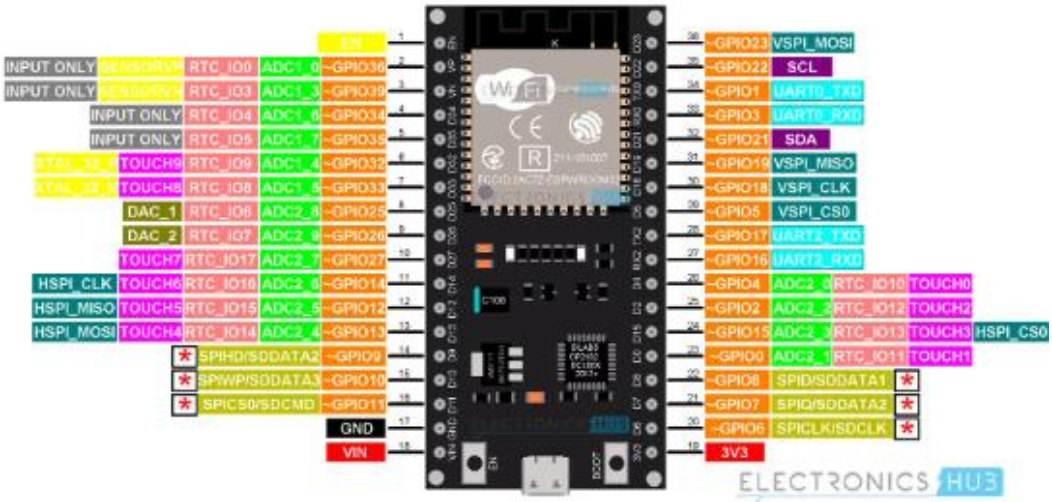


***Figura 34**: Pinout del ESP32-WROOM-32*

El ESP 32 cuenta con los siguientes pines:

- *18 canales de conversión analógico-digital (ADC):* Permiten la lectura de señales analógicas para su posterior conversión a digital.
- *10 GPIO de detección capacitiva*: Ideales para interfaces táctiles o botones capacitivos
- *3 interfaces UART*: Permite conectar múltiples dispositivos seriales

- *2 interfaces SPI:* Usadas en conexiones de dispositivos de alta velocidad como pantallas o sensores
- *2 interfaces I2C*
- *16 canales de salida PWM*
- *2 convertidores de digital a analógico (DAC):* Permite crear señales analógicas
- *2 interfaces 12S:* Útiles en procesamiento de audio

De los pines de propósito general, existen únicamente 4 que son de entrada (GPIO34,35,36 Y 39) y 8 con pull-up interno (GPIO14,16,17,18,19,21,22 y 23, necesario para el correcto funcionamiento de nuestro encoder).

3.2.3 ¿Por qué ESP32?

El microcontrolador ESP 32 presenta una gran cantidad de funcionalidades que lo convierte en un sistema idóneo para la realización de proyectos de electrónica. Se trata de un hardware muy intuitivo y fácil de aprender a usar, pero que si se estudia a fondo presenta funciones muy útiles que permiten realizar un gran proyecto.

Sin embargo, existen una gran cantidad de microcontroladores que ofrecen servicios similares al ESP como pueden ser el BasicX-24P, el BASIC Stamp (Parallax) o el STM32 de STMicroelectronics. Todos estos microcontroladores serían aptos para la realización del proyecto, pero se ha elegido ESP por las razones siguientes:

- Bajo coste: Aunque ninguno de los microcontroladores mencionados posee un coste muy elevado, el ESP 32 presenta el coste mas bajo.
- Entorno de programación: El ESP 32 permite la programación en el entorno Arduino IDE, un entorno tremendamente sencillo con un lenguaje de programación similar a C o C++
- Uso en distintas plataformas: ESP es compatible con Windows, MAC o Linux, a diferencia de otros microcontroladores que presentan limitaciones en este aspecto.

Además, ESP 32 destaca por su conectividad WI-FI y bluetooth integrada así como su capacidad para manejar grandes cantidades de información de distintos periféricos o sensores

3.2.4 ESP NOW

Se trata de un protocolo de comunicación de baja latencia basado en Peer to Peer (P2P).

Peer to peer

Se trata de un modelo de comunicación de redes en donde cada uno de los dispositivos llamado "peer" (también llamado par" puede actuar como cliente o como servidor. Se trata de un modelo "Punto a Punto" en donde en lugar de depender de un dispositivo central, el envío de datos se produce directamente entre los dos dispositivos interesados, funcionando como emisor y receptor (aunque estos roles no tienen la obligación de ser fijos).

Gracias a este tipo de comunicación se libera carga y la dependencia del nodo central como puede pasar en otros modos de comunicación. Es usado en situaciones donde se requiere una conexión directa y rápida como en sistemas distribuidos o transferencia de información

Características del protocolo ESP-NOW

Funciona de forma similar a una red de comunicación bluetooth donde los dispositivos pueden intercambiar datos directamente sin necesidad de acceso a un punto de red wifi.

Proporciona una comunicación rápida y eficiente debido a que no hace uso de TCP/IP reduciendo de esta forma la latencia y el consumo de energía. Permite enviar paquetes de hasta 250 bytes hasta una distancia de 200 metros. Se puede programar de forma que un microcontrolador funcione como maestro de un esclavo o de forma inversa teniendo en cuenta que está limitado a 10 clientes para el modo estación y como mucho 6 en modo combinado (en caso de que el dispositivo reciba información de varios nodos de forma simultánea).

No existen canales como tal, sino que los dispositivos se identifican por sus direcciones MAC por lo que es necesario sacar las direcciones de los dispositivos de forma previa a la programación.

El funcionamiento es el siguientes:

Cada ESP32 tiene una dirección MAC única por lo que el emisor debe de conocer esta dirección para poder enviar los datos. Una vez conocida esta dirección se establece una pareja ("peer") con esa MAC de forma que ya se puede comenzar el envío de datos. Existen distintos modos de comunicación en ESP NOW:

- *Unicast*: A un solo dispositivo a través de la MAC de este
- *Multicast*: A varios a la vez (varias MACs)
- *Broadcast*: A todos los dispositivos al alcance

Una vez establecida la conexión, podremos enviar datos del ESP emisor y leerlos en el receptor sin necesidad de establecer una conexión física.

AES-128

Por último, podemos activar la encriptación AES-128 de forma que aumentemos la seguridad en el envío de información. AES-128 hace referencia a "Advanced Encryption Standard", un algoritmo de cifrado simétrico (se usa una clave para cifrar y descifrar datos. El 128 hace referencia que la clave que se usa es de 128 bit (16 bytes). El funcionamiento es el siguiente:

AES trabaja en bloques de 128 bits de forma que cada vez que se cifran datos AES los divide en bloques de 16 bytes y las procesa realizando las siguientes etapas:

- *División y expansión*: Se comienza dividiendo el mensaje en bloques de bits que más tarde se expanden mediante el programa de claves de AES. También se añade una clave de cifrado llamada RoundKey
- *Sustitución*: El texto sin formato se reemplaza por el texto cifrado mediante la tabla Rijndael S-box
- *Desplazamiento*: Todas las filas menos la primer se desplazan una posición
- *Mezcla*: Las filas ya cifradas y desplazadas se vuelven a mezclar
- *RoundKey*: Mediante la RoundKey se vuelve a cifrar la información

Este proceso se repite varias veces en base al tipo de AES

3.2.5 Programación en Arduino IDE

Todos los componentes que se han explicado y se van a explicar van a estar controlados mediante un ESP, por lo que se considera necesario explicar cómo se realiza la programación de estos dispositivos.

La programación de estos dispositivos pasa primero por entender cómo funciona el entorno Arduino IDE (donde se han desarrollado todos los códigos). Arduino IDE permite la programación, compilación y posterior subida al microcontrolador de códigos siendo la diferencia con otros entornos de programación la forma en que se dividen los "*sketch*" (archivos con los códigos de programación). Estos se dividen en 2 partes, el "*setup()*" y el "*loop ()*":

- *Setup():* Se trata de una función que se llama una única vez en toda la ejecución del código y se usa para inicializar bibliotecas o funciones que solo se realizan una vez en todo el proceso.
- *Loop()*: Por su parte, la función loop se ejecuta de forma continua durante todo el proceso y, en la mayoría de los códigos, es donde se sitúa el código principal del programa.

Todos los códigos desarrollados se mostrarán en siguientes apartados

3.3 Acción de los actuadores

En este apartado describiremos los elementos encargados de realizar la acción mecánica, tanto los actuadores, así como los elementos asociados a estos.

3.3.1 Motor lineal

Los motores lineales son aquellos motores capaces de producir un movimiento a lo largo de un eje. En nuestro caso, necesitamos de un motor lineal para poder producir un ángulo en las ruedas (la geometría y mecánica se explicará en siguientes apartados). El motor elegido ha sido el siguiente (cabe destacar que el componente no posee ningún nombre específico). La imagen[35] muestra una foto del motor elegido.

***Figura 35**: Esquema de motor rotativo*

Las características esenciales del motor son las siguientes:

Voltaje de alimentación	12 V
Velocidad del vástago	60mm/s
Fuerza	100N
Longitud del vástago	100mm

***Tabla 5**: Características de motor lineal*

Estos motores son capaces de, en vez de dar movimiento rotativo, dar un movimiento lineal. En nuestro caso, usamos un actuador eléctrico con motor de imán permanente y escobillas. En este dispositivo usamos un motor DC para girar un tornillo interno de forma que la rosca convierte el giro en movimiento lineal. En nuestro caso la alimentación del dispositivo es de 12V. El control del sentido se realiza a través de la alimentación, de forma que se conectamos el positivo de la batería al cable negro y el negativo de la batería al cable rojo

conseguiremos que el motor avance y en caso de conectarlo de forma contraria el motor retrocederá

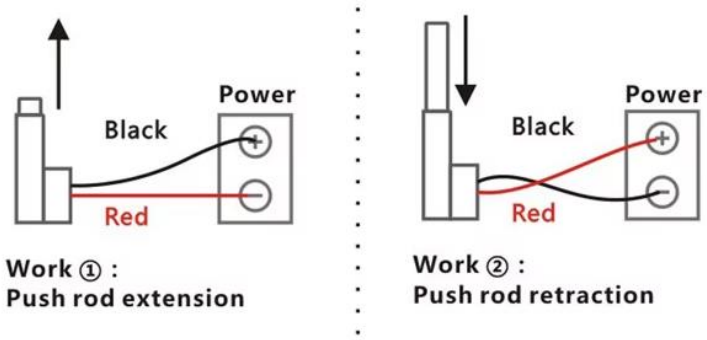


***Figura 36**: Principio de motor rotativo*

Para poder conmutar entre los dos tipos de movimientos se ha de usar un puente en H de forma que podamos cambiar el sentido de la alimentación.

3.3.2 Puente en H

Un puente H es un dispositivo electrónico que se usa mayoritariamente en el control del sentido de un motor DC. Está conformado por 4 interruptores (generalmente MOSFET o BJTs) que conmutan de forma que cambian la alimentación que se le da al motor, permitiendo así que gire en ambos sentidos. El esquema básico de un puente en H es el siguiente.

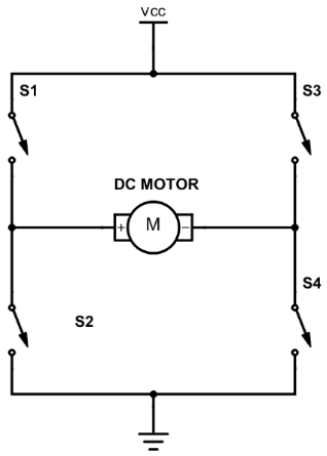


***Figura 37**: Principio de puente H*

En el caso de nuestro motor y siguiendo el esquema de puente H (y siendo el positivo del motor el cable rojo) de la figura [37], si los semiconductores 1 y 4 conducen el motor retrocedería, y si los semiconductores activos son el 3 y 2 el motor avanzaría. En nuestro caso, usaremos un puente H con 8 MOSFET IRF3205 como el que se muestra en la imagen

***Figura 38**: Imagen de puente H*

Se trata de un puente en H pensado para realizar el control de 2 motores, por lo que cuenta con 8 MOSFET. Los pines con los que cuenta son los siguientes:

PINOUT DEL PUENTE EN H			
IN		OUT	
GND	Tierra	POWER	Alimentación del motor (12V)
5V	Alimentación	GND	Tierra
PWM1	Control PWM para la velocidad del motor 1	MOTOR 1(PIN 1)	+/- del motor 1
DIR1	Control de la dirección del motor 1	MOTOR 1(PIN 2)	+/- del motor 1
PWM2	Control PWM para la velocidad del motor 2	MOTOR 2(PIN 1)	+/- del motor 2
DIR2	Control de la dirección del motor 2	MOTOR 2(PIN 2)	+/- del motor 2

***Tabla 6**: Pinout del puente en H*

Las conexiones que se realizan son las siguientes:

PUENTE H (PINES OUT)	MOTOR
GND	GND (FUENTE 12V)
POWER	12 V
MOTOR 1	CABLE ROJO
MOTOR 1	CABLE NEGRO

***Tabla 7**: Conexión del puente en H con el motor*

PUENTE H (PINES IN)	ESP 32(RECEPTOR)
GND	GND
5V	5V
DIR1	GPIO18
PWM1	GPIO5

***Tabla 8**: Conexión del puente en H con el ESP receptor*

Control PWM

El control PWM ("Pulse Width Modulation") es una técnica de control que nos permite controlar cuanta energía se suministra a un dispositivo y de esta forma controlar su reacción. En vez de cambiar el voltaje de forma continua, este método enciende y apaga muy rápidamente una señal digital, de forma que cuando esta señal sea 1 (ON) se entrega energía y cuando esté a 0 (OFF) no se entrega. Todo esto se regula a través del ciclo de servicio ("Duty Cycle") que es el tiempo que está encendido frente al de un ciclo completo.

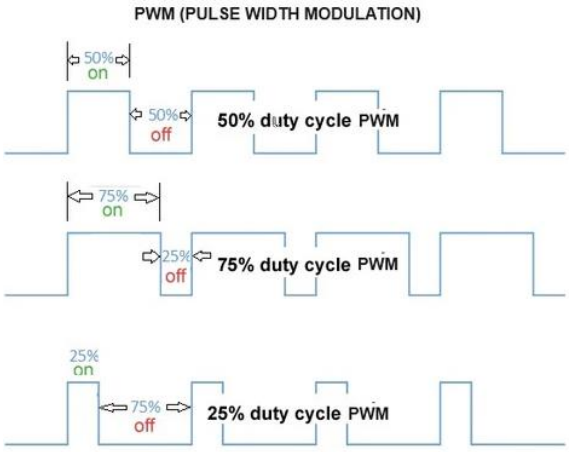


***Figura 39**: Principio de PWM*

De esta forma, si situamos el ciclo de servicio en un 100%, el motor avanzará a su velocidad máxima, si está en un 50% el motor avanzará a la mitad de su velocidad y si es del 0% el motor no se moverá. Cabe destacar también la frecuencia del PWM que hace referencia a la cantidad de veces que se realiza que se realiza el ciclo (tiempo en ON+tiempo en OFF) en un segundo

Se ha usado el control PWM que implementa ESP32 en donde el ciclo se servicio tiene una resolución de 8 bits por lo que, si se programa el ciclo de servicio con un valor de 255 el ciclo de servicio es máximo. De esta forma, y cambiando el valor de una variable en el código somos capaces de controlar el

ciclo de servicio al que funciona el motor. Uno de los canales a través del que podemos modular un PWM es el GPIO5 por lo que se ha elegido ese para realizar el control.

3.4 Otros componentes usados

A parte de todos los componentes mencionados, el proyecto requiere el uso de otros componentes para un correcto funcionamiento. Estos componentes se recopilan a continuación

3.4.1 -Motor rotativo

Para el diseño final se ha usado el kit0085 de Dfrobot, conformado por 2 motores rotativos, 2 ruedas y 2 adaptadores de encoder. Las características de estos componentes se describen a continuación.

***Figura 40**: Kit 0085 de DFRobot*

El motor que añade el kit es el llamado de un motor rotativo de corriente continua (DC) 28PA51G alimentado a 12 voltios. Es un motor silencioso y capaz de dar un elevado par, por lo que sus principales campos de aplicación son la robótica móvil, así como la automatización. Posee un codificador óptico que permite un elevado control sobre la posición y velocidad.

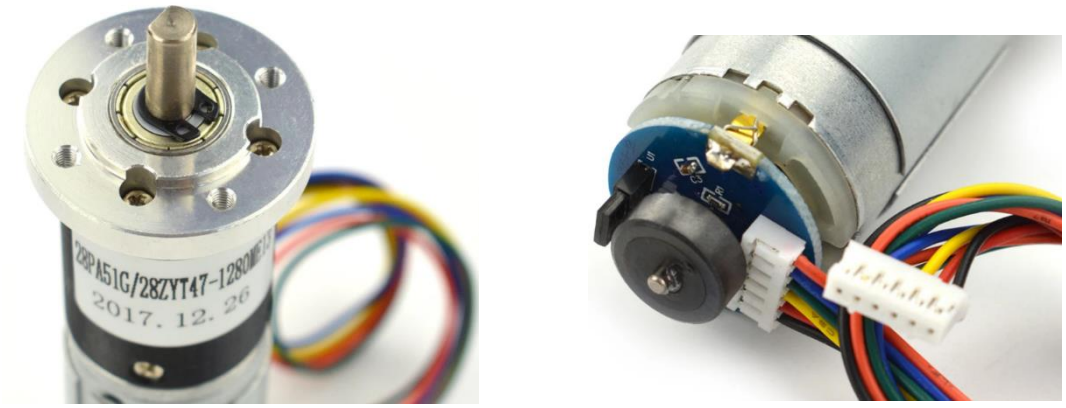


***Figura 41**: Kit 0085 de DFRobot*

Sus principales especificaciones se recogen en la siguiente tabla:

Motor 28PA51G	
Voltaje de trabajo	12 V
Velocidad sin carga (antes de la caja de engranajes)	8000 rpm
Relación de engranajes	51:1
Velocidad sin carga (después de la caja de engranajes)	146 rpm a 12 V
Corriente sin carga	0.23 A
Corriente de bloqueo	3.6 A
Par nominal	10 kg/cm
Codificador óptico	13PPR
Tipo de codificador	Hall de dos fases
Peso	270 g
Nivel de ruido	68Dba a 10 cm de distancia

***Tabla 9**: características motor rotativo*

El kit nos incluye un "Encoder adapter" cuyo objetivo es facilitarnos la conexión del motor con algún dispositivo de control como puede ser nuestro ESP32. Los cables y la función que tienen se muestran en la figura[42]

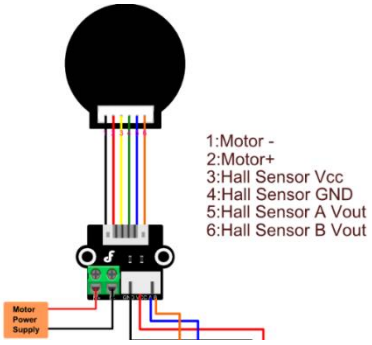


***Figura 42**: Conexión del motor rotativo*

Sin embargo, en nuestro caso, la conexión de alimentación al motor (en la figura[42] el "Motor Power Supply") se ha llevado al puente en H con la finalidad

de poder controlar la velocidad a la que gira el motor. De esta forma, el esquema de conexiones que obtenemos es el siguiente.

En este caso, y a través de la programación del ESP32 receptor podremos controlar tanto la velocidad de giro como el sentido.

Como se ha mencionado varias veces, el objetivo de este proyecto es comenzar con la migración de las transmisiones mecánicas hacia las transmisiones basadas en elementos electrónicos e informáticos. Sin embargo, para un proyecto de este nivel es imposible realizar el prototipo a escala real o con componentes industriales. Debido a esto se ha decidido usar un kit explicado anteriormente. De esta forma, y a través de la realización de este prototipo, podremos verificar si se trata de un proyecto viable a mayor escala.

3.4.3-Diodo LED

Un diodo Led (*Light Emitting Diode*) es un dispositivo electrónico capaz de emitir una cierta cantidad de luz cuando le atraviesa una corriente. Su estructura es simple pero muy ingeniosa. Su núcleo está formado por una pequeña ficha de un material semiconductor llamado "*die*" montado sobre una base conductora. Dicha ficha está protegida por una resina epoxi que protege el diodo y actúa como lente. El corazón del LED es el material semiconductor y condiciona el color del LED. Los más comunes son:

- Arseniuro de Galio (GaAs): Rojo e infrarrojo
- Fosfuro de Galio (GaP): Verde
- Nitruro de Galio (Gana): Verde, azul y blanco

El LED tiene dos terminales, el cátodo y el ánodo, y es imprescindible distinguirlas bien para poder usar el diodo de forma correcta. Generalmente el cátodo (negativo) es el terminal más corto a diferencia del ánodo (positivo).

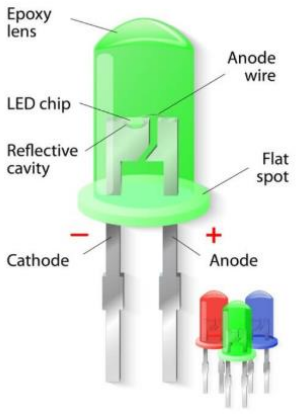


***Figura 43**: Esquema de un diodo LED*

3.4.4 Fuente de alimentación

Es indispensable alimentar todos los elementos que conforman el circuito, por lo que necesitaremos una fuente de tensión. Para la realización de pruebas (al comienzo del proyecto) se usaron fuentes de tensión como la de la figura [44] (disponibles en los laboratorios de la facultad).

***Figura 44**: Fuente de tensión del laboratorio*

Sin embargo, para poder acercarnos más a un caso real, hacia el final del proyecto se redireccionó hacia otros tipos de fuentes de tensión debido a que las anteriores son muy pesadas y difíciles de transportar.

La fuente que se decidió usar es una fuente que corresponde a un ordenador de sobremesa, en concreto es la fuente *PC CASE 500 ATX2.2 (P4) Modelo: LPK 12-25P5.*

***Figura 45**: Fuente de alimentación PC CASE*

Se trata de una fuente que cuenta con los siguientes pines y cables:

INPUTS

La fuente se alimenta con un cable IEC tradicional a una toma de corriente habitual de 220 Voltio y 50 hertzios.

***Figura 46**: Input de PC CASE*

OUTPUTS

Como salidas, podemos ver los diferentes conectores que hay:

- *1 cable ATX de 24 pines* → Va conectado a la placa base del PC. Por él circulan varios cables de datos, así como de alimentaciones. A pesar de existir cables de 12 y 5 voltios, estos se encuentran limitados por la funcionalidad de este cable y no son capaces de soportar altas corrientes.

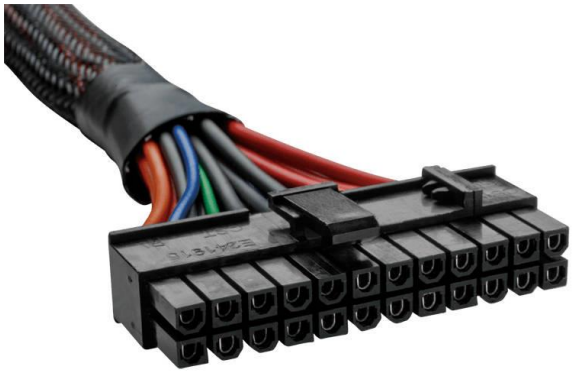


***Figura 47**: Cable ATX de de PC CASE*

Sin embargo, si conectamos la fuente a la tensión y pulsamos el botón de encendido la fuente no se encenderá a la primera. Esto es debido al cable verde dentro del conector ATX, llamado PS_ON y cuya funcionalidad es dar la orden de encendido. Para que la fuente

pueda comenzar a funcionar se ha de cortocircuitar el cable verde con el negro (GND).

- *2 cables SATA*→ Son usados para la conexión de discos duros y unidades ópticas SATA. Poseen alimentación, pero el acceso a estos cables es bastante complejo.

***Figura 48**: Cable Sata de de PC CASE*

- *2 cables Molex 4 pines (IDE):* Usados en discos duros, ventiladores y algunos accesorios.

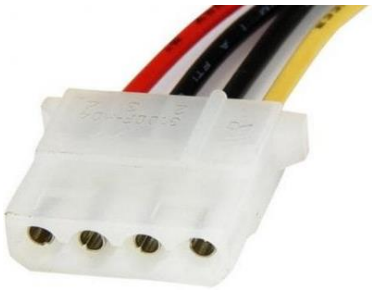


***Figura 49**: Cable Molex de de PC CASE*

Los pines que usaremos para alimentar el sistema son los de este cable. Esto es debido a que la principal función de los cables molex es la alimentación de diferentes dispositivos, de forma que no está tan limitado en corriente como otros conectores. El cable amarillo son 12 voltios, el rojo son 5 voltios y el negro es tierra (GND)

3.4.5- Protoboard

Las *protoboards* son un elemento muy usado en proyectos de electrónica. Nos permite realizar esquemas electrónicos sin la necesidad de soldar, permitiéndonos de esta forma probar un circuito antes de su realización final. Suelen están formadas por dos barras laterales y una zona central, siendo su uso habitualmente de alimentación y colocación de elementos respectivamente.

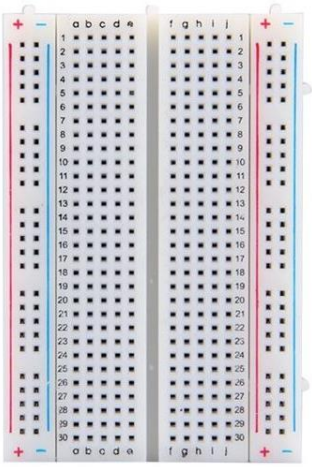


***Figura 50**: Imagen de PROTOBOARD*

De esta forma, en las barras laterales (de alimentación) todos los puntos de acceso que se encuentran en la línea vertical estarán a la misma tensión.

Por su parte, la zona central funciona de igual forma, pero a diferencia de las laterales, las zonas de tensión son comunes en líneas horizontales.

3.4.6 Pulsador

Un pulsador es un dispositivo que permite que, cuando el botón que posee el pulsador se cierre un circuito que en estado de reposo se encontraría abierto.

***Figura 51**: Imagen de pulsador*

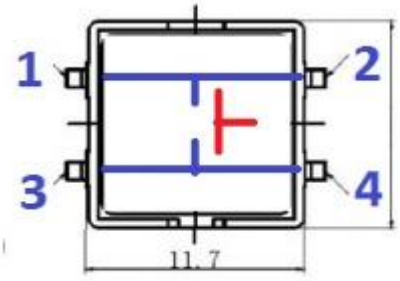


***Figura 52**: Esquema de pulsador*

Su funcionamiento es muy sencillo. Las patillas (1,2,3 y 4) se encuentran conectadas por pares, la 1 con la 2 y la 3 con la 4. Cuando el l botón no está

presionado (la parte roja de la figura [52]) estos pares de patillas no están conectados. Sin embargo, cuando el botón es pulsado estos pares de patillas de conectan entre sí. En nuestro caso se ha conectado la patilla 1 (también podría ser la 2) a los 3.3 voltios con los que puede alimentar el ESP y la patilla 2 al pin 2 configurado previamente como un pin con resistencia input. De esta forma y a través del código desarrollado (se explicará en el correspondiente apartado) podemos saber en que momento se ha pulsado el botón).

3.4.7 Placas para sostener el ESP

Para poder sujetar los ESP a una superficie y poder conectar los cables de forma correcta se han usado una serie de sujeciones ya diseñadas y bastante expandidas (figura [53]).

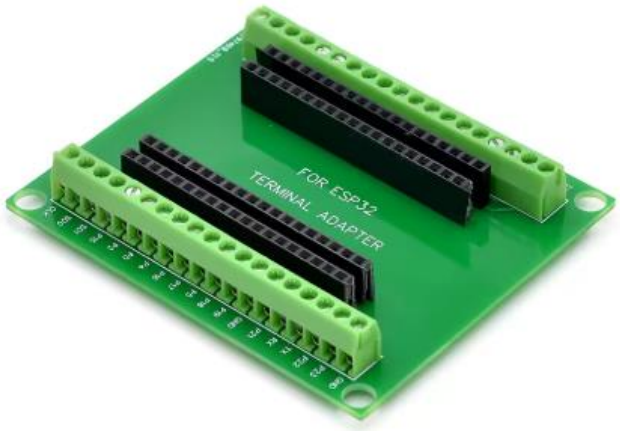


***Figura 53**: Imagen de sujeción de ESP32*

Esta placa permite la conexión al ESP a través de cables tanto hembra como macho.

3.4.8- Cables usados

Para todo el proyecto se han usado el mismo tipo de cables, los llamados cables jumper propios de los circuitos y conexiones rápidas en protoboard. Estos cables están formados por un hilo de cobre interno recubierto por una película de plástico (generalmente de PVC). Son capaces de soportar hasta 250 voltios y de conducir hasta 1 amperio. Pueden ser de 3 tipos:

- Macho-Macho→ Ambos extremos tienen pines metálicos
- Macho-Hembra→Un extremo tiene el pin y otro el receptáculo
- Hembra-Hembra→Ambos extremos tienen el receptáculo.

***Figura 54**: Imagen de cables usados*

3.5 Esquemas eléctricos

Con todos los elementos del proyecto ya presentados y explicados, pasamos a mostrar los esquemas eléctricos de cada parte con la finalidad de entender bien el conexionado del proyecto:

> **NOTA**: Al final de este documento se muestra los planos eléctricos completos

3.5.1 Esquema eléctrico del emisor

ENCODER:

El conexionado del encoder es el siguiente:

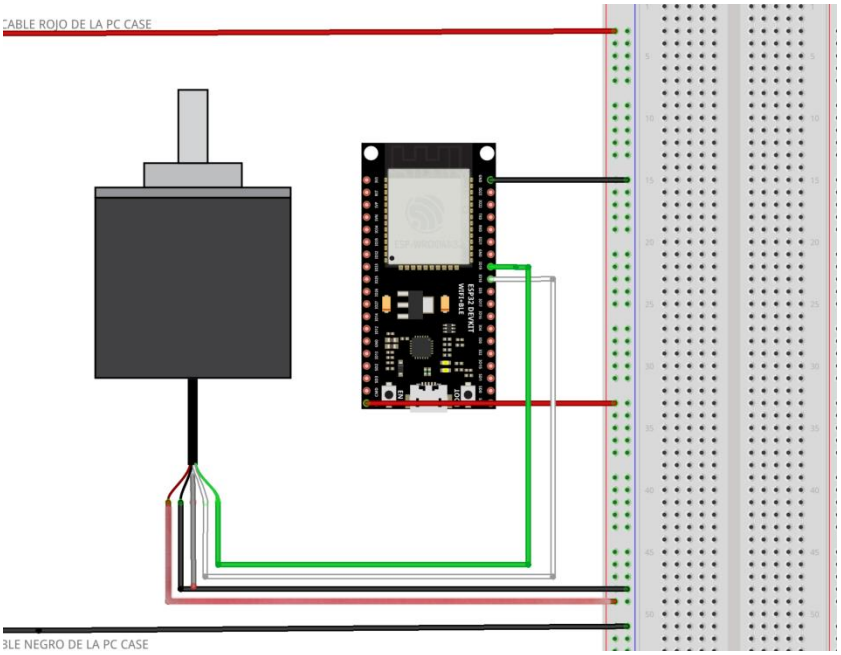


***Figura 55**: Esquema de conexión de ESP emisor*

Pulsador

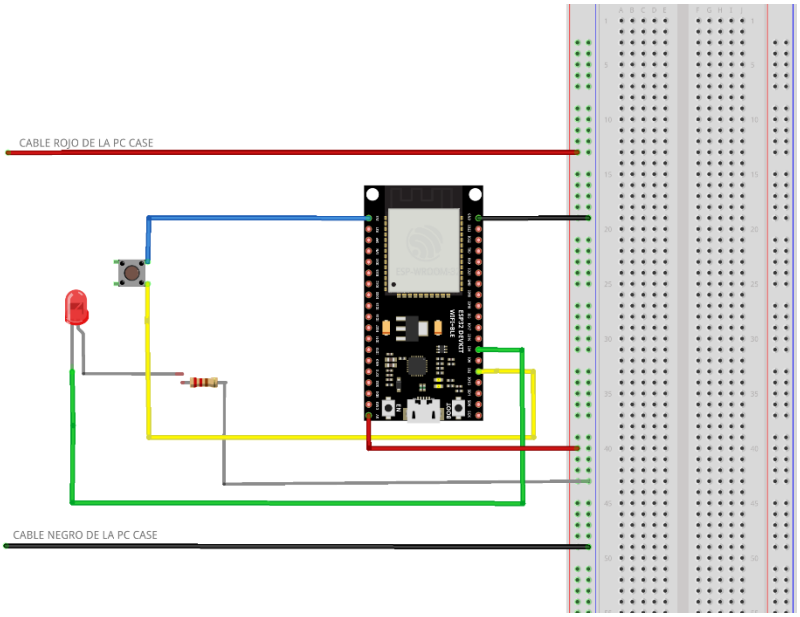


***Figura 56**: Esquema de conexión de ESP emisor con pulsador*

Esquema completo del emisor

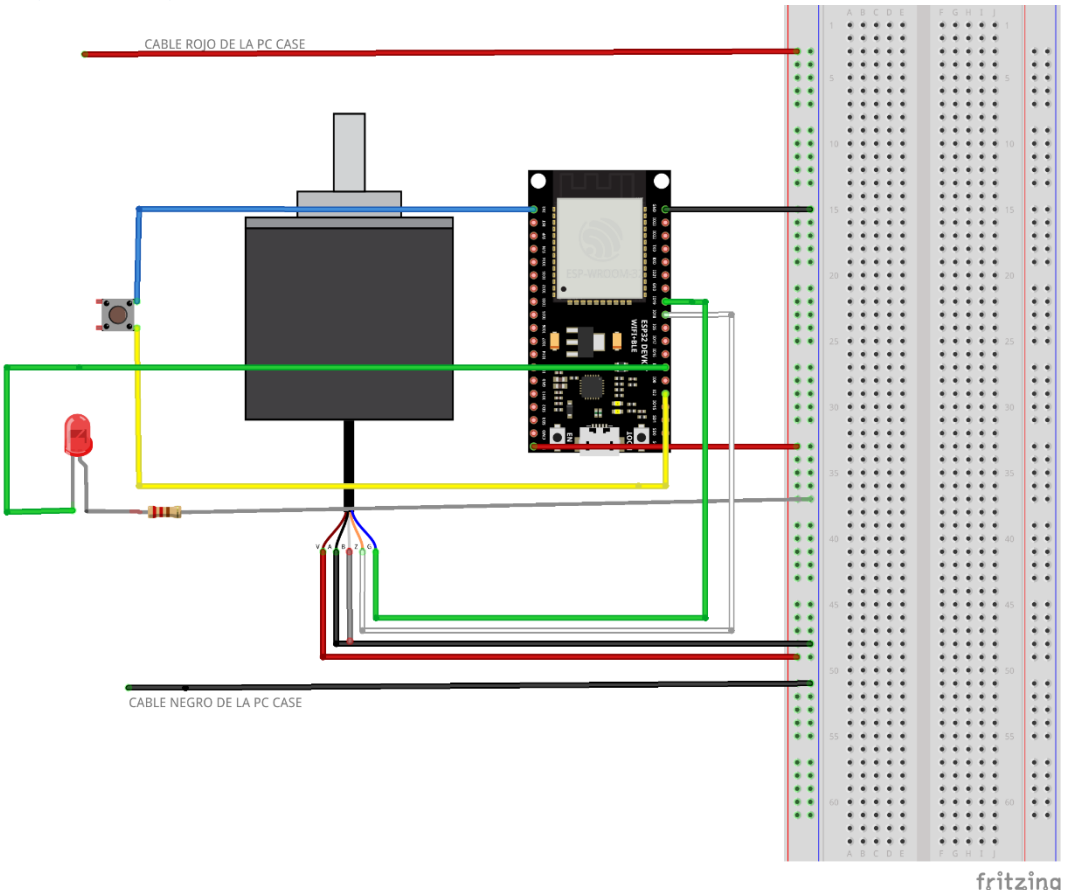


***Figura 57**: Esquema de conexión de ESP emisor completo*

3.5.2 Esquema eléctrico del receptor

Control del motor lineal:

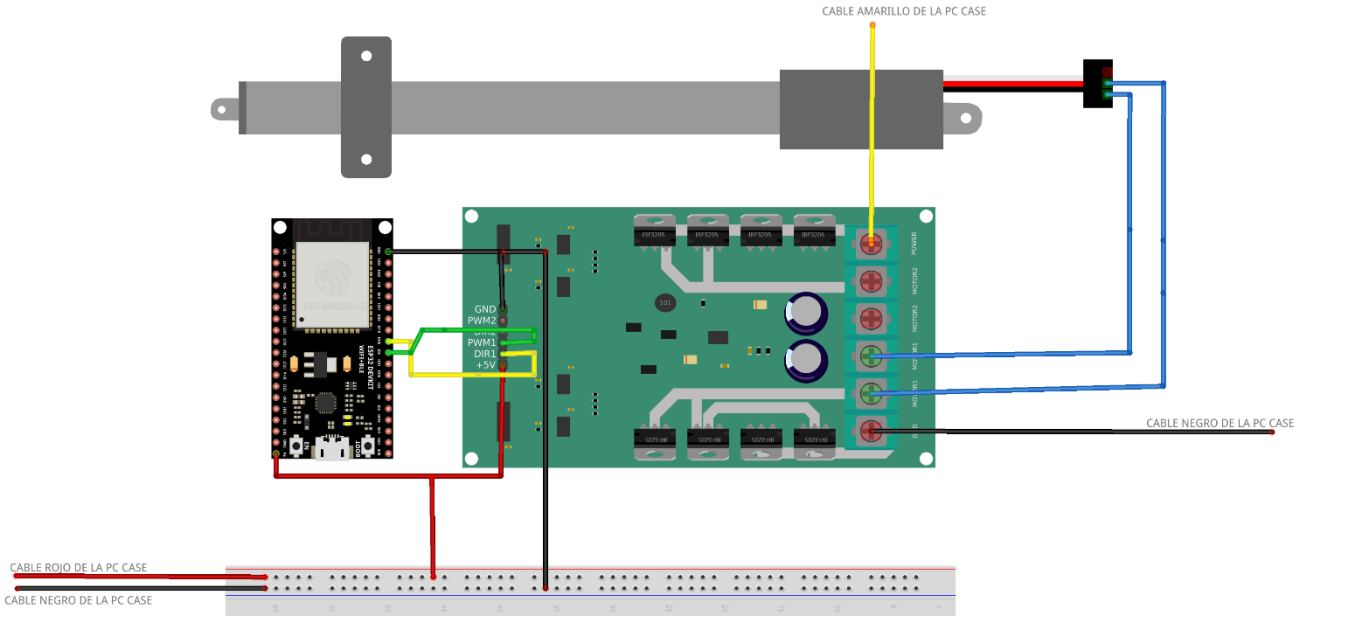


***Figura 58**: Esquema de conexión del motor lineal*

Control del motor rotativo:

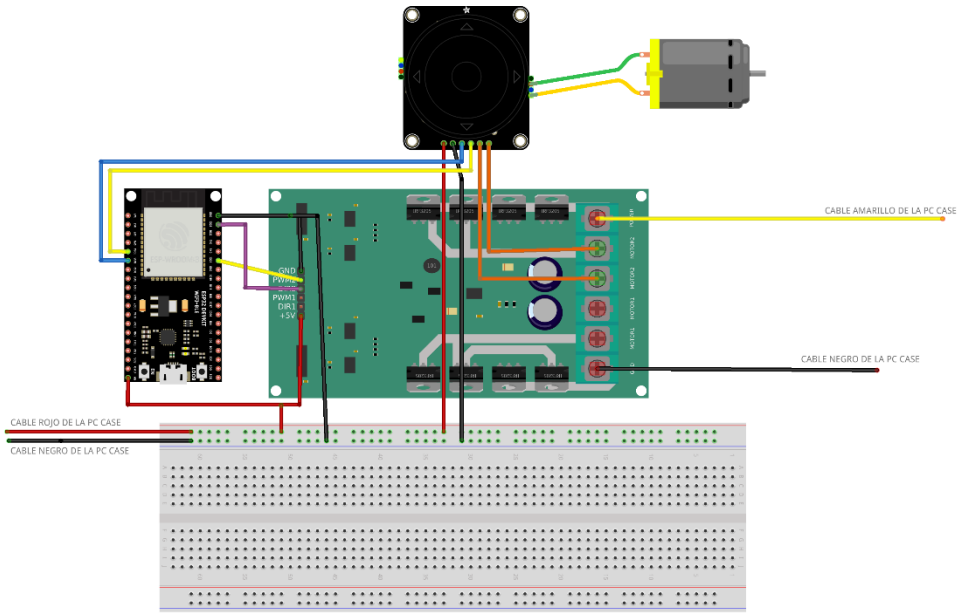


***Figura 59**: Esquema de conexión del motor rotativo*

Esquema global del receptor:

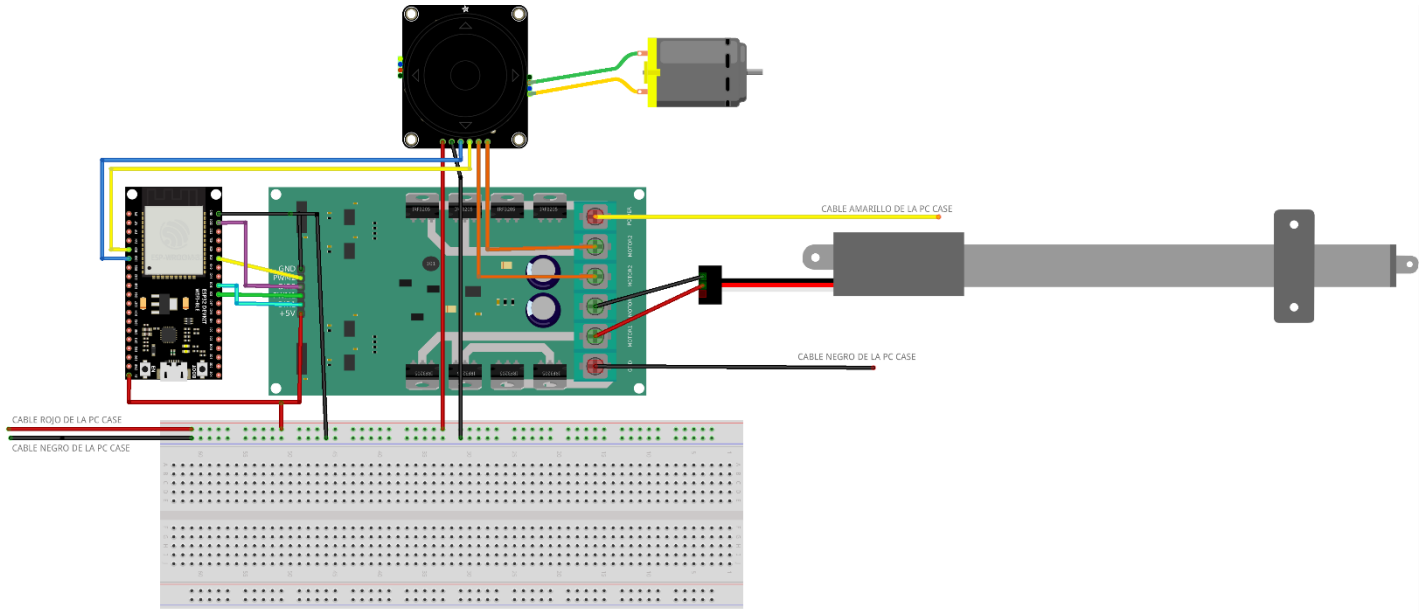


***Figura 60**: Esquema de conexión del ESP receptor*

Capítulo 4:

Cálculos geométricos y diseño de piezas en 3D

CAPÍTULO 4. Cálculos geométricos y diseño de piezas en 3D

El objetivo de este capítulo es la justificación del diseño mecánico que conforma la dirección propuesta. Previo a ello vamos a repasar la geometría de las direcciones actuales, así como las piezas que lo conforman.

4.1 Diseño tradicional

Habiendo comentado ya los elementos que conforman una dirección tradicional en el capítulo 2 "Conceptos Previos", no es necesario volver a explicar cada uno de ellos. Sin embargo, sí que explicaremos el ensamblaje de estos elementos para poder producir un ángulo de giro en las ruedas.

***Figura 61**: Esquema de dirección tradicional*

En la figura [61] se puede apreciar de forma muy esquemática los elementos principales de un sistema de dirección. El giro del volante implica un desplazamiento en la cremallera conectada al tirante de dirección, que, a causa del desplazamiento será capaz de producir un cierto ángulo de giro en las ruedas.

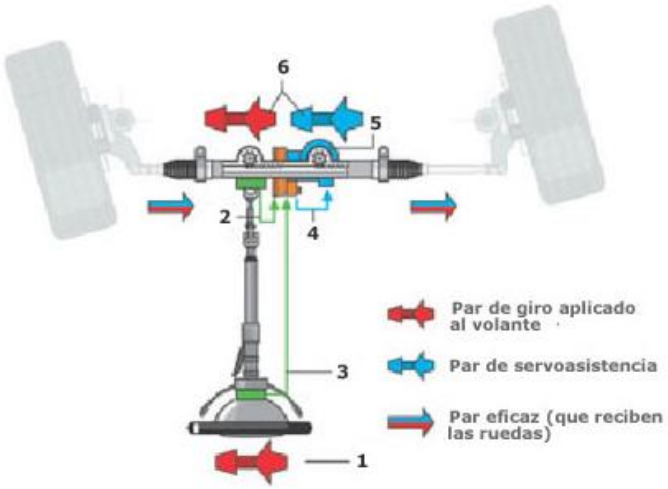


***Figura 62**: Esquema de dirección tradicional (vista cenital)*

Este ha sido el sistema tradicional y que implementan la mayoría de los vehículos en la actualidad. Sin embargo, la tendencia actual apunta hacia una transformación radical del sistema, con la adopción de tecnologías más avanzadas como la dirección eléctrica asistida y los sistemas Steer-by-wire.

4.2 Geometría de dirección propuesta

Uno de los puntos más complicados a realizar es la geometría para esta dirección. En nuestro caso, el desplazamiento lateral se realizará a través de un motor lineal (en vez del mecanismo piñón-cremallera) de forma que este desplazamiento suponga un ángulo de giro. Para ello se realizaron distintos diseños que se han ido evolucionando hasta dar paso al diseño final.

Los elementos con los que se comenzó a realizar los cálculos de la geometría de esta dirección son los siguientes:

- Rueda: Procedente del Kit0085 de Dfrobot.
- Sujeción del motor rotativo: Su función es sujetar el motor rotativo y permitir el diseño de la dirección de forma que tengamos una pieza cercana a la rueda sobre la que podamos actuar.
- Motor lineal: El actuador encargado de provocar un movimiento.

Ha sido necesario añadir otros elementos, pero se explicarán a lo largo del documento.

4.2.1 Primera aproximación al diseño final

Lo primero a diseñar es la geometría para conseguir la orientación de nuestra rueda. El primer diseño constaba de un sistema muy sencillo en el que, se acoplaba directamente el motor lineal a la sujeción del motor rotativo dando lugar al siguiente sistema

NOTA: Cabe destacar que la rueda es paralela a la sujeción del motor rotativo, por lo que si esta pieza rota, la rueda rotará con ella.

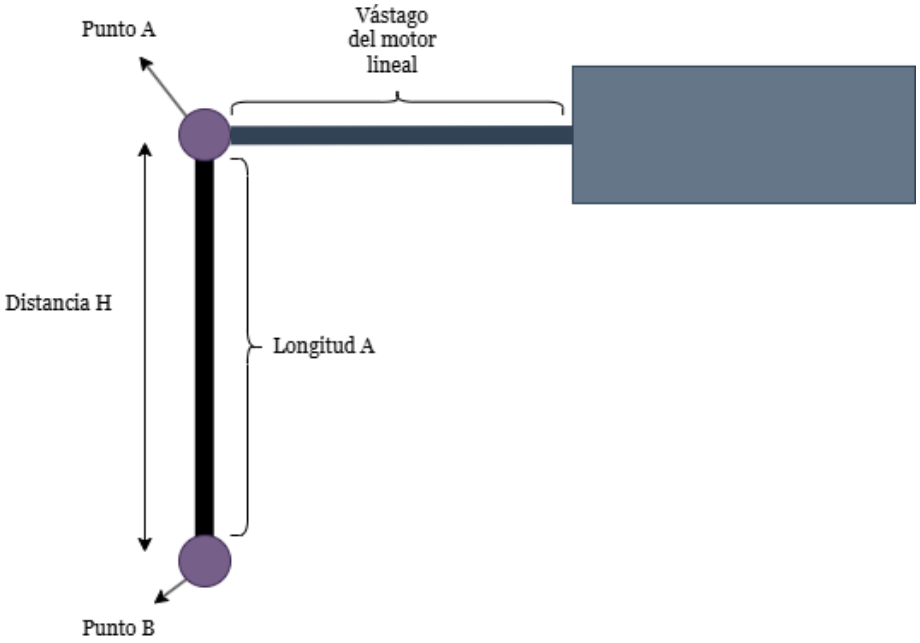


***Figura 63**: primera geometría propuesta*

El sistema está formado por los siguientes elementos:

- Punto A: Se trata de un punto en la sujeción del motor rotativo. Este punto permite la rotación sobre sí mismo y el desplazamiento eje horizontal
- Punto B: Se trata de un punto fijo que permite la rotación sobre él mismo, pero en ningún eje más (ni horizontal ni vertical)
- Longitud X: Se llama "Longitud X" a la distancia entre 2 puntos dentro de la sujeción del motor
- Vástago del motor lineal: Es el vástago del motor lineal capaz de avanzar y retroceder en el eje horizontal

La distancia H ha de ser constante, pues el motor lineal no se puede mover en el eje vertical.

Cálculos geométricos:

Como se muestra en la figura [63], el propio extremo del motor empujaría el soporte del motor rotativo en un sentido u otro de forma que se produjese un giro en la rueda de un ángulo γ. Estos casos se muestran en las figuras [64] [65].

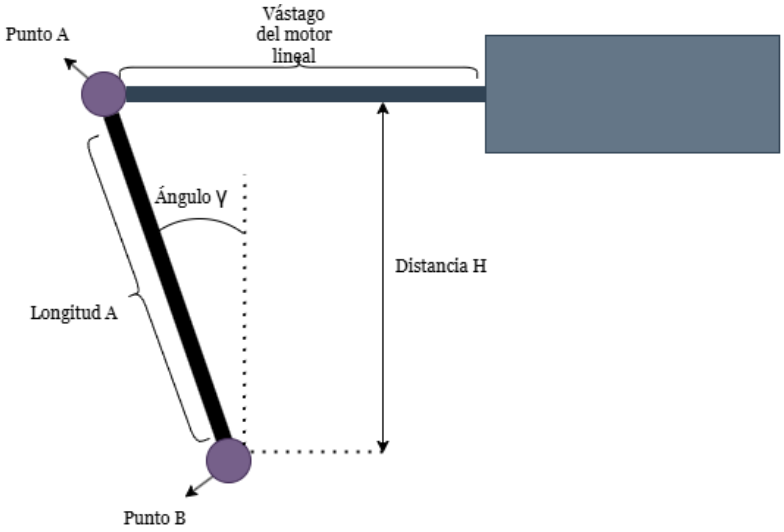
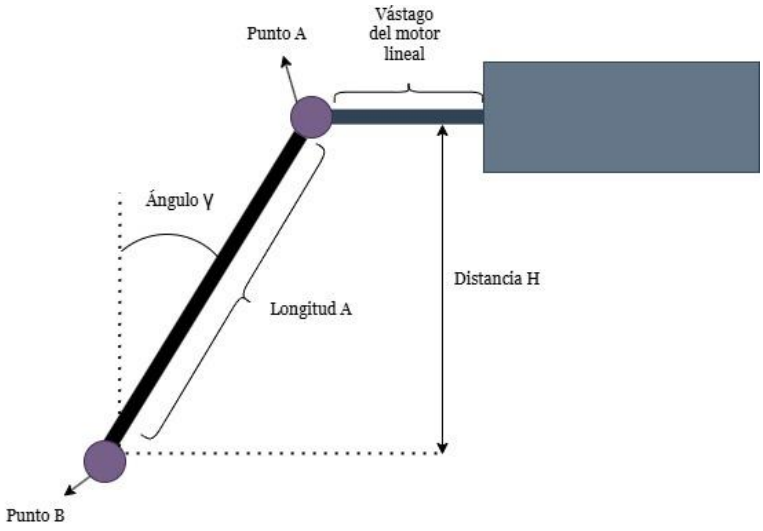


***Figura 64**: Primera geometría propuesta con giro*



Usaremos la siguiente nomenclatura:

- Distancia de B a la punta del vástago: *"BA"*
- Distancia H: "*H*"

Vamos a estudiar el sistema con 2 casos, en caso de que el sistema esté en la posición inicial (con γ igual a 0) y en caso de que γ sea distinto de 0:

Caso 1:

En caso de que el ángulo *"γ"* sea de 0 grados, la "*distancia H"* coincide con la "*Longitud A",* que a su vez coincide con la distancia *"BA"*

$$BA = H = Longitud\ A \qquad [3]$$

Caso 2:

Sin embargo, la geometría cambia en caso de que el ángulo γ sea distinto de 0. En ese caso, la "*distancia H"* sigue siendo la misma que en el caso inicial e igual a la "*longitud A"*. Pero la "distancia *BA"* en este caso no corresponde con la "*longitud A"*, si no a una longitud mayor (por el teorema de Pitágoras).

$$(BA)^2 = (Distancia\ H)^2 + (Incremento\ del\ vástago)^2 \qquad [4]$$

Por ello, esta solución necesita una *"longitud A"* variable que implicaría un par de puntos dentro de la sujeción del motor que fuesen capaz de aumentar y disminuir la distancia entre ellos. Se descartó esta opción debido a su complejidad de diseño.

4.2.2 Diseño final

Conociendo los problemas que surgen a la hora de realizar un diseño como el que se ha explicado en el apartado anterior, se buscó una solución que fuese menos compleja. Por ello, se decidió diseñar un tirante de dirección que fuese unido al soporte del motor rotativo y que permitiese el giro completo. De esta forma, el esquema del diseño final es el siguiente:

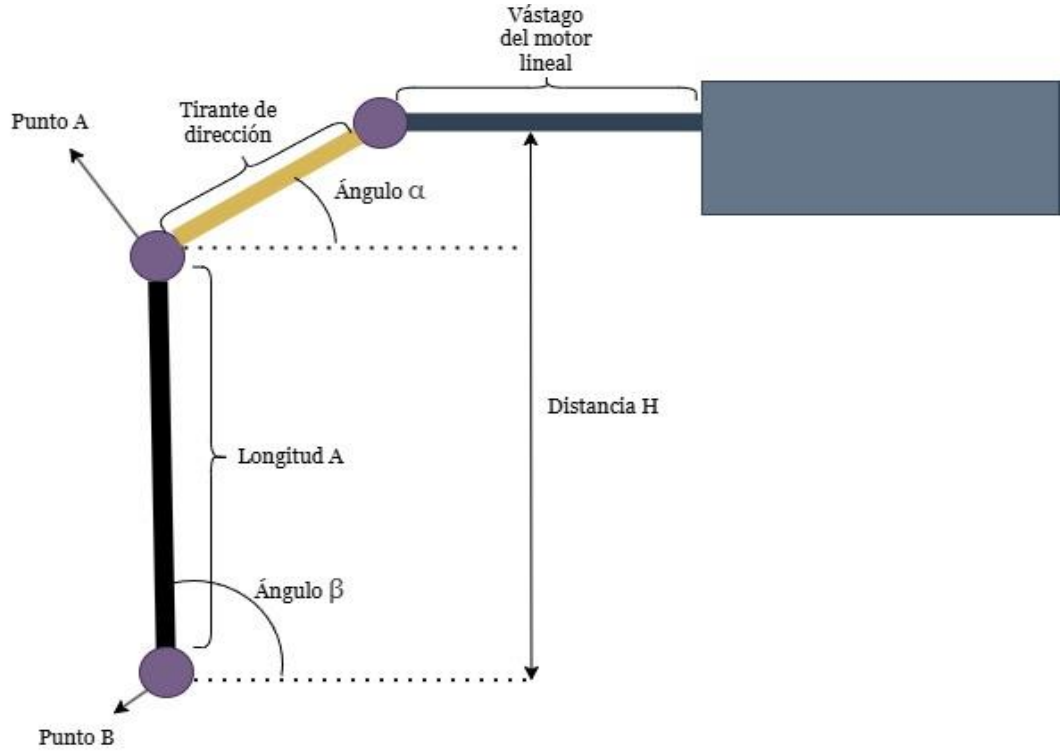


***Figura 65**: Geometría final propuesta*

En este nuevo diseño se mantienen los elementos del anterior diseño y se añaden los siguientes elementos:

- *Tirante de dirección*: Segmento de una *Longitud L (su valor se determinará más tarde)*. Se une de forma directa tanto al motor lineal como al *Punto A*.
- *Punto C:* Este punto es el punto de unión entre el motor lineal y el tirante. Permite el giro tanto sobre sí mismo como en el eje horizontal.
- *Ángulo* β: Es el ángulo que forma la sujeción del motor rotativo con la horizontal. *El valor inicial de este ángulo es de 90º*
- *Ángulo* α: Es el ángulo que forma el tirante con la horizontal

De forma similar al diseño anterior, la distancia H se ha de mantener todo el rato, pues el motor no se puede mover en el eje vertical. Teniendo en cuenta esto, obtendremos las ecuaciones del movimiento basándonos en 3 casos, el sistema en reposo (figura [65]), giro en sentido horario (figura [67]) y sentido antihorario en la siguiente.

Caso 1: Sistema en reposo

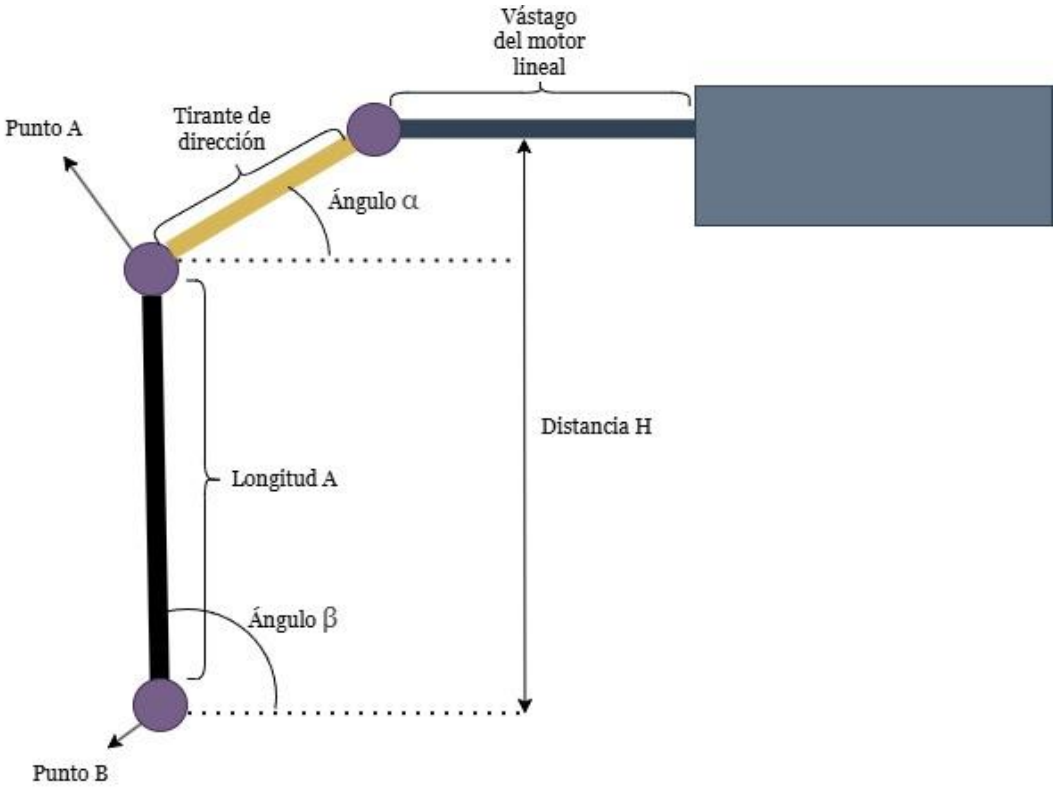


***Figura 66**: Geometría final propuesta*

Calculamos la *"distancia H"* (teniendo en cuenta que beta vale 90º en reposo):

$$H = (Longitud\ A) + L * sen(\alpha) \qquad [5]$$

siendo "L" la longitud del tirante de dirección

<u>*Caso 2*</u>: Giro en sentido horario

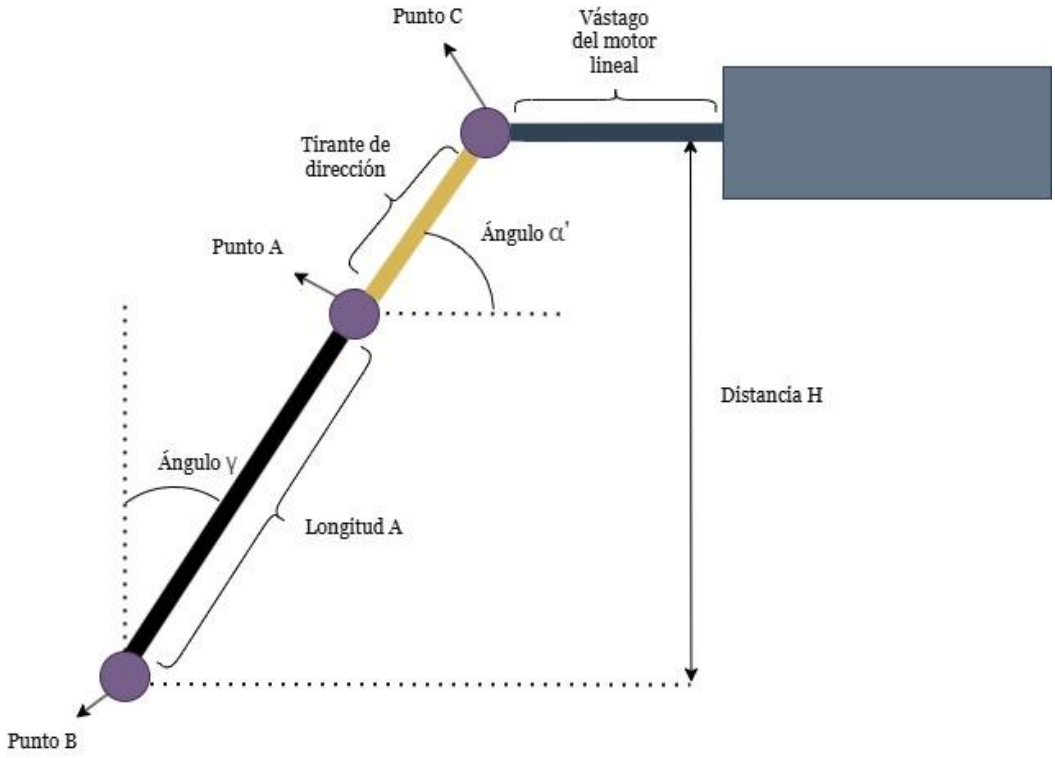


***Figura 67**: Geometria final propuesta con giro*

Al igual que en el primer caso, calculamos la distancia H:

$$H = (Longitud\ A) * \cos(\gamma) + L * \mathrm{sen}(\alpha') \qquad [6]$$

siendo "L" la longitud del tirante de dirección

Como podemos observar en la figura [67] el ángulo α ha cambiado de valor, llamándose ahora α′. Debemos calcular la ecuación que nos proporciona el valor de este nuevo ángulo. Para ello hemos de tener en cuenta que la altura H ha de ser constante, por lo que podemos usar la ecuación [] que hemos *obtenido en el caso 1.*

$$H(caso\ 1) = H(caso\ 2) \qquad [7]$$

$$(Longitud\ A) + L * sen(\alpha) = (Longitud\ A) * cos(\gamma) + L * sen(\alpha')$$

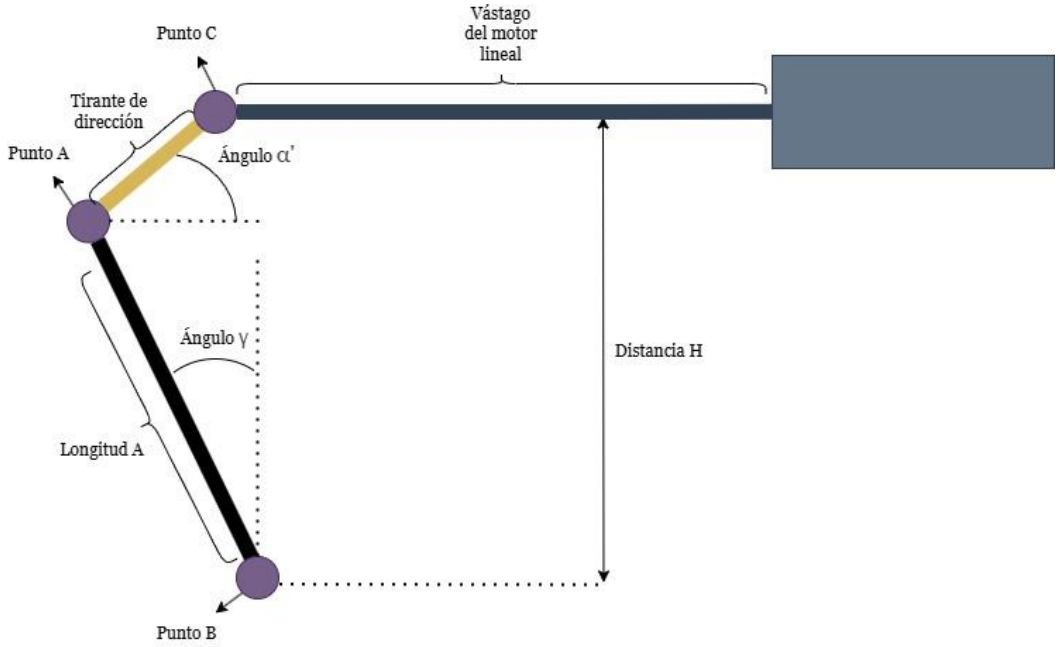
Despejando de esta ecuación obtenemos:

$$sen(\alpha') = \frac{(Longitud\ A) + L * sen(\alpha) - (Longitud\ A) * cos(\gamma)}{L} \qquad [8]$$

Siendo

- *Longitud A*: Distancia entre los 2 puntos de la sujeción
- *L*: Longitud del tirante de dirección
- *Alfa(α)*: Ángulo inicial del tirante de dirección con la horizontal
- *Alfa(α')*: Ángulo en un determinado momento del tirante de dirección con la horizontal

Caso 3: Giro en sentido antihorario



Al igual que anteriormente, vamos a obtener el valor de la distancia H:

$$H = (Longitud\ A) * \cos(\gamma) + L * \mathrm{sen}(\alpha') \qquad [9]$$

Como era de esperar, obtenemos la misma ecuación que si el giro fuese antihorario. Usando esta ecuación y siguiendo el mismo procedimiento que en el caso 2 ($H(caso1) = H(caso3)$) llegamos a la expresión del $\mathrm{sen}(\alpha')$:

$$sen(\alpha') = \frac{(Longitud\ A) + L * sen(\alpha) - (Longitud\ A) * cos(\gamma)}{L} \qquad [10]$$

Es la misma fórmula que obtenemos para el caso 2.

Por ello podemos concluir lo siguiente:

La distancia H siempre ha a tener el mismo valor, que viene dado por cualquiera de estas 2 fórmulas:

$$H = (Longitud\ A) + L * sen(\alpha) \quad [11]$$

$$H = (Longitud\ A) * \cos(\gamma) + \mathrm{L} * \mathrm{sen}(\alpha')$$

El ángulo α' se obtiene de la siguiente fórmula.

$$sen(\alpha') = \frac{(Longitud\ A) + L * sen(\alpha) - (Longitud\ A) * cos(\gamma)}{L} \quad [12]$$

Por último, hemos de determinar el recorrido que ha de realizar el vástago del motor lineal. Para ello podemos usar cualquiera entre los casos 2 y 3, donde podemos realizar el siguiente razonamiento para saber el desplazamiento

Lo que se desplaza el punto C en el eje X es igual que lo que se desplaza el punto A en el eje X. Por ello, el desplazamiento lateral seguirá la siguiente fórmula:

$$X = (Longitud\ A) * sen(\gamma) \quad [13]$$

El ángulo gama será el ángulo de giro de la rueda, por lo que su rango va desde 0 hasta 50 grados. El resto de los valores dependerán de este ángulo.

4.2.3 Establecimiento de las condiciones iniciales

Para crear una geometría que tenga el comportamiento que deseamos hemos de dar unos valores a las longitudes (la longitud A y la longitud del tirante) y del ángulo α. Tras probar distintas posibilidades, se ha decidido que los valores iniciales sean los siguientes.

Ángulo alfa (α)	10º
Longitud A	6 cm
Longitud del tirante	5.7 cm

***Tabla 10**: Condiciones iniciales de la geometría*

Para comprobar que estos valores cumplen con todas las condiciones, se ha decidido crear una tabla con todos estos valores y su evolución en base al ángulo gama. Cabe recalcar que, como se ha mencionado en anteriores apartados el ángulo de giro de la rueda debe alcanzar los 50º, por lo que el ángulo gama variará entre 0º y 50º

Ángulo gamma (γ)	Evolución de α'	Distancia H (con la fórmula inicial)	Distancia H (con la formula con ángulo)
0	10	6,989794613	6,989794613
1	10,00932755	6,989794613	6,989794613
2	10,03730896	6,989794613	6,989794613
3	10,08394056	6,989794613	6,989794613
4	10,14921625	6,989794613	6,989794613
5	10,23312761	6,989794613	6,989794613
6	10,33566399	6,989794613	6,989794613
7	10,45681263	6,989794613	6,989794613
8	10,59655885	6,989794613	6,989794613
9	10,75488621	6,989794613	6,989794613
10	10,93177673	6,989794613	6,989794613
11	11,12721116	6,989794613	6,989794613
12	11,34116924	6,989794613	6,989794613
13	11,57363	6,989794613	6,989794613
14	11,82457211	6,989794613	6,989794613
15	12,09397424	6,989794613	6,989794613
16	12,38181545	6,989794613	6,989794613
17	12,68807561	6,989794613	6,989794613
18	13,01273585	6,989794613	6,989794613
19	13,35577908	6,989794613	6,989794613
20	13,71719046	6,989794613	6,989794613
21	14,09695801	6,989794613	6,989794613
22	14,49507312	6,989794613	6,989794613
23	14,91153126	6,989794613	6,989794613
24	15,3463326	6,989794613	6,989794613
25	15,79948271	6,989794613	6,989794613
26	16,27099335	6,989794613	6,989794613
27	16,76088325	6,989794613	6,989794613
28	17,26917895	6,989794613	6,989794613
29	17,79591577	6,989794613	6,989794613
30	18,34113873	6,989794613	6,989794613
31	18,90490364	6,989794613	6,989794613
32	19,48727824	6,989794613	6,989794613
33	20,0883434	6,989794613	6,989794613
34	20,70819446	6,989794613	6,989794613
35	21,34694266	6,989794613	6,989794613

36	22,00471674	6,989794613	6,989794613
37	22,68166462	6,989794613	6,989794613
38	23,3779553	6,989794613	6,989794613
39	24,09378094	6,989794613	6,989794613
40	24,82935914	6,989794613	6,989794613
41	25,58493551	6,989794613	6,989794613
42	26,36078649	6,989794613	6,989794613
43	27,15722255	6,989794613	6,989794613
44	27,97459175	6,989794613	6,989794613
45	28,81328378	6,989794613	6,989794613
46	29,67373453	6,989794613	6,989794613
47	30,55643133	6,989794613	6,989794613
48	31,46191889	6,989794613	6,989794613
49	32,39080617	6,989794613	6,989794613
50	33,34377436	6,989794613	6,989794613

***Tabla 11**: Evolución de las variables de la geometría*

Podemos apreciar que la distancia H es constante en toda la tabla para todos los valores del ángulo γ.

Por su parte, el desplazamiento lateral varía entre 0 y 4.59 cm en el caso de que la rueda tenga que girar 50º. Siendo nuestro vástago del motor de 10cm podemos cubrir sin problema todo el recorrido. Por ello, la posición inicial ha de ser de 4.59 centímetros del vástago salido.

4.3 Piezas desarrolladas y ensamblaje 3D

Una vez realizado el análisis del movimiento y determinadas las longitudes, así como los ángulos, ya conocemos las piezas que hemos de desarrollar. Para el desarrollo de dichas piezas se ha usado el software *"Autodesk Inventor"*.

4.3.1 Autodesk Inventor. Primeros pasos

Autodesk Inventor es un software desarrollado por Autodesk, que permite el diseño y ensamblaje de mecanismos de piezas en 3D. A su vez también permite la creación de los planos de las piezas que se desarrollan en él y la visualización de movimiento entre piezas ensambladas.

Autodesk Inventor permite realizar proyectos de 2 tipos: diseño de piezas o de ensamblajes.

El diseño de piezas está centrado en la creación de piezas en 3D, a través de dibujos sobre planos y extrusiones.

El diseño de ensamblajes por su parte consiste en la creación de mecanismos entre las distintas piezas que se han diseñado para realizar un determinado movimiento. Todo esto se consigue a través de restricciones.

Diseño de Piezas

Cuando abrimos un proyecto en Autodesk inventor relativo al diseño de piezas, nos encontramos una pantalla similar a la siguiente:

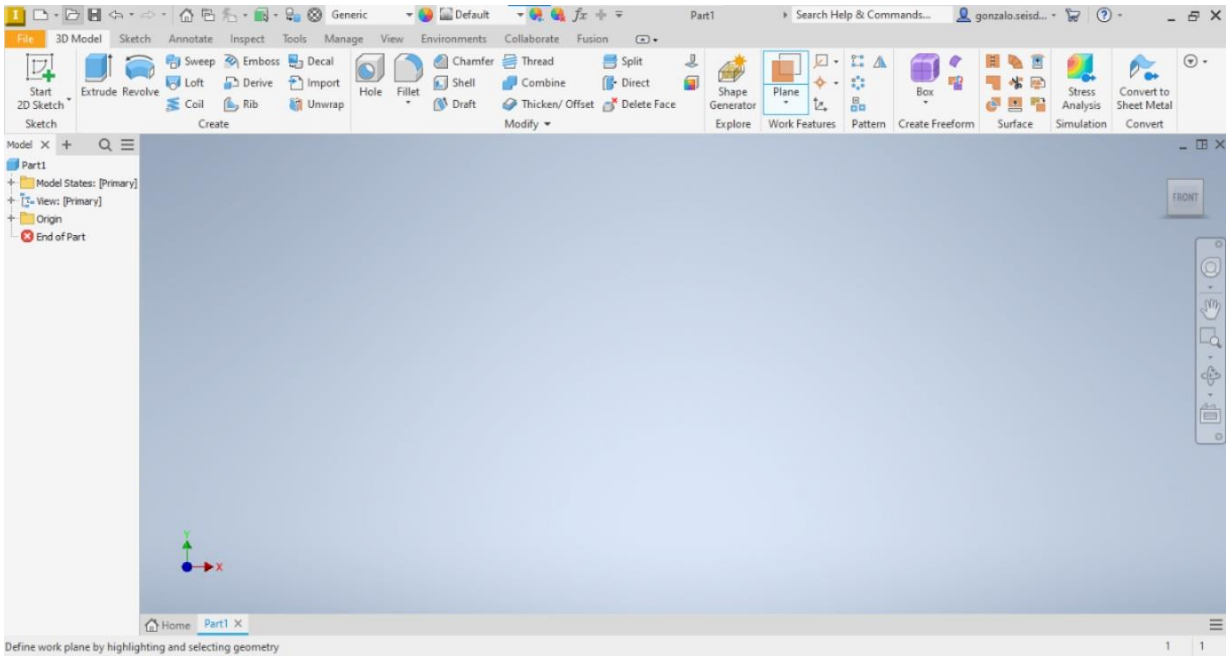
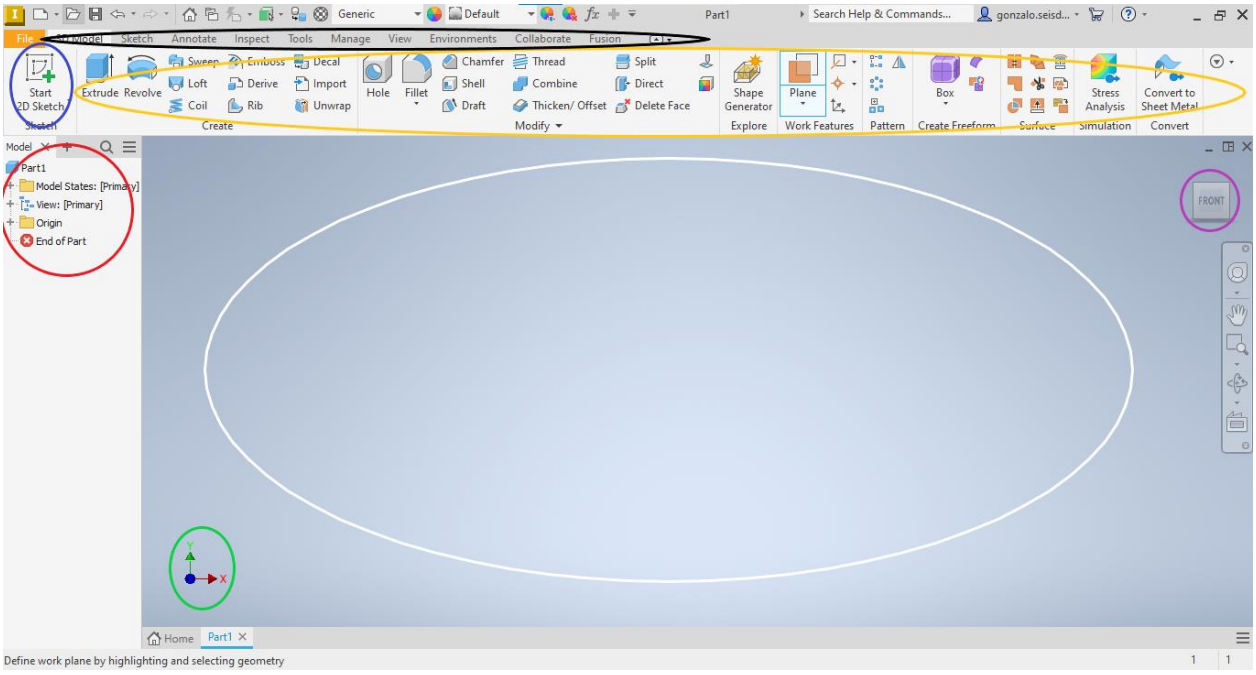


***Figura 68**: Pantalla principal de autodesk inventor*



En ella no aparece ningún dibujo debido a que no se ha diseñado nada todavía, pero previo a comenzar con el diseño vamos a hablar de los distintos elementos que aparecen en esta pantalla. Para explicar que realiza cada opción, vamos a señalarlas primer en la captura.

- En Blanco, y en el centro de la pantalla se encuentra el conocido como *"Viewport"* o *"Workspace"*. Es el espacio donde se podrá visualizar el modelo 3D que se está diseñando.
- En verde nos encontramos la *triada de ejes*, que nos indica sobre que eje estamos trabajando y sirve para conocer la orientación de la pieza
- En rojo se ha marcado el *Árbol de modelo.* Este es uno de los elementos más importantes en toda la pantalla. En este índice se muestran todos los *sketches* que hemos hecho (un *sketch* es el dibujo que se ha realizado sobre uno de los planos, ya sea el XY, YZ o el XZ, o se haya creado sobre una superficie de la pieza). También nos indica las diferentes extrusiones que se han realizado sobre dichos *sketches* y que han dado lugar a la pieza 3D. Nos permite realizar cambios sobre los distintos elementos de la pieza cambiando así la forma de la pieza final sin tener que diseñarla entera de nuevo (como podría ser el hecho de cambiar la altura de una pieza). En caso de tratarse de un ensamblaje, el árbol de modelo nos mostraría las distintas piezas que conforman el ensamblaje, así como las restricciones que existen entre estas.
- En azul se ha marcado la opción *"Start 2D sketch"*. Esta es la opción que nos da acceso al diseño de una pieza sobre un plano 2D.
- En morado está marcado el Cubo de vistas o *Viewcube,* que nos permite ver la pieza desde todas sus caras, así como desde otros puntos. Esto nos ayuda a hacernos una idea de como serán los planos de la pieza.
- En **negro** se sitúa la cinta de operaciones, que agrupa las distintas operaciones que se pueden realizar por grupos (*Sketch, Create, Modify…)*
- En naranja tenemos las distintas operaciones que nos permite la pestaña 3D Model. Entre estas operaciones está la extrusión (que permite dar volumen a un boceto 2D), la revolución o la opción de hacer agujeros entre muchas otras.

4.3.2 Piezas diseñadas y función de cada una de ellas

Para realizar el diseño mecánico de este sistema hemos primero de explicar que piezas hemos de diseñar.

Para poder realizar el ensamblaje final de todo el sistema hemos de diseñar todas las piezas que conforman el sistema en la realidad. Ello significa que no

solo se han de diseñar las piezas 3D que se van a fabricar si no aquellos elementos que se han comprado y forman parte del proyecto.

Motor rotativo:

Es el motor que viene con el Kit0085 de DFRobot. Tiene las siguientes dimensiones (todos estos planos se mostrarán de forma completa en los anexos).

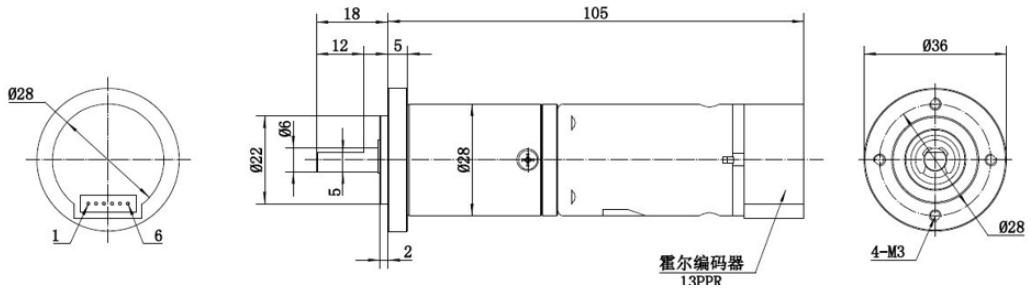


***Figura 69**: Plano principal de motor rotativo*

Se trata de un componente con un peso de 236.7 gramos, es el encargado de transmitir el movimiento a las ruedas. Se ha diseñado en 3D obteniendo el siguiente modelo:

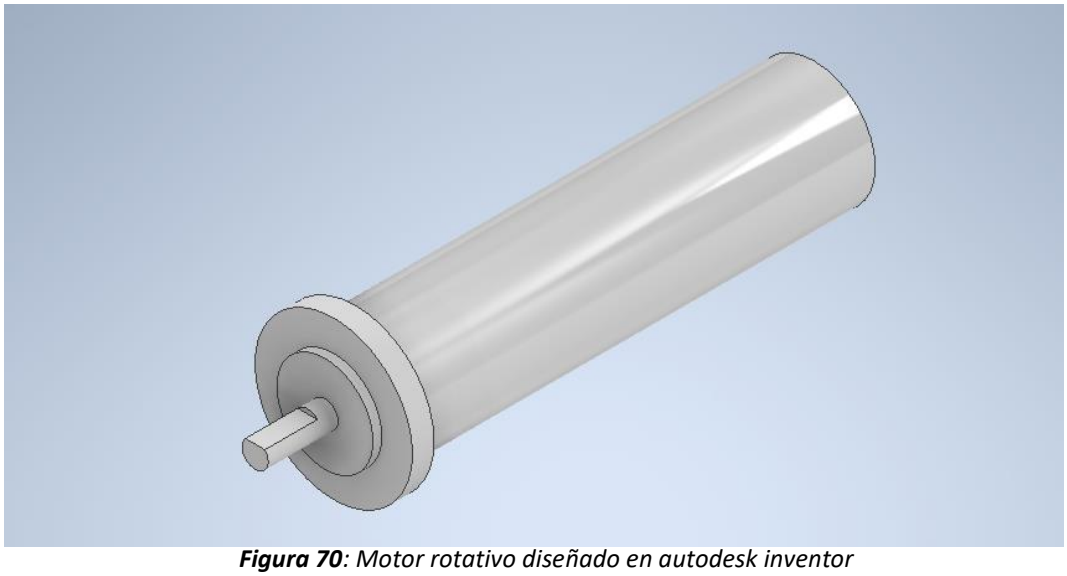


***Figura 70**: Motor rotativo diseñado en autodesk inventor*

Diseñado con las medidas reales, nos ayudará a saber cómo será el aspecto final de todo el proyecto

Ruedas:

También forman parte del Kit0085 de DFRobot. Tienen un diámetro exterior de 13,4 cm y una serie de perforaciones gracias a las que no se trata de una rueda muy pesada. Su peso es de 292.6 gramos y vienen acompañados de una serie de adaptadores para que se puedan acoplar al motor rotativo. El diseño de esta pieza ha resultado de la siguiente manera:

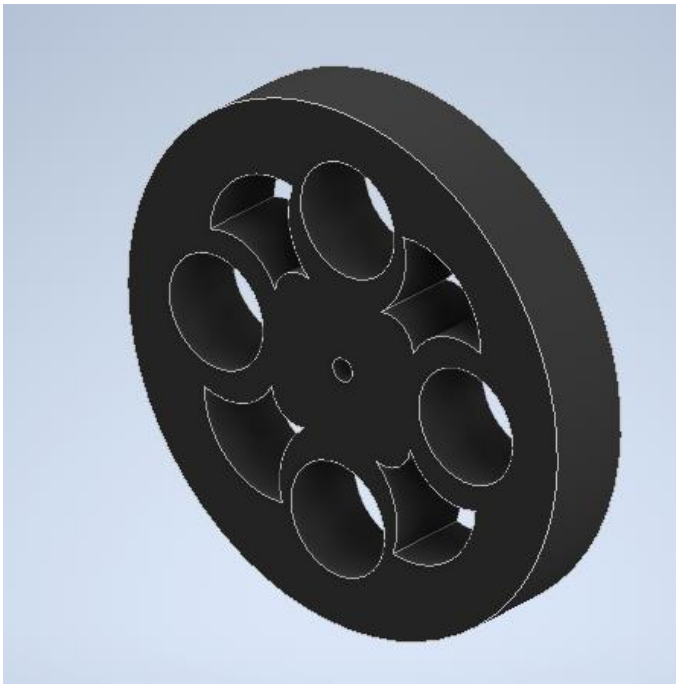


***Figura 71**: Rueda diseñado en autodesk inventor*

Motor lineal

Es el encargado de transmitir el movimiento de traslación en el eje horizontal. Dispone de una reductora y un vástago que avanza o retrocede en base a la alimentación. El plano real se muestra a continuación, y seguido a él, se muestra el diseño 3D:

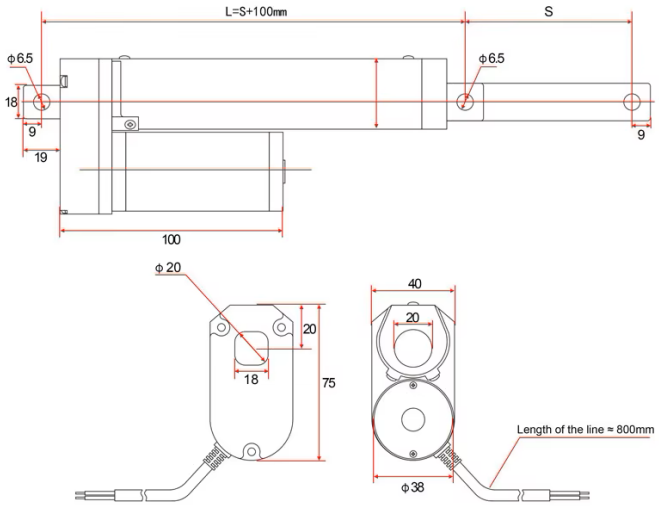


***Figura 72**: Plano de motor lineal*

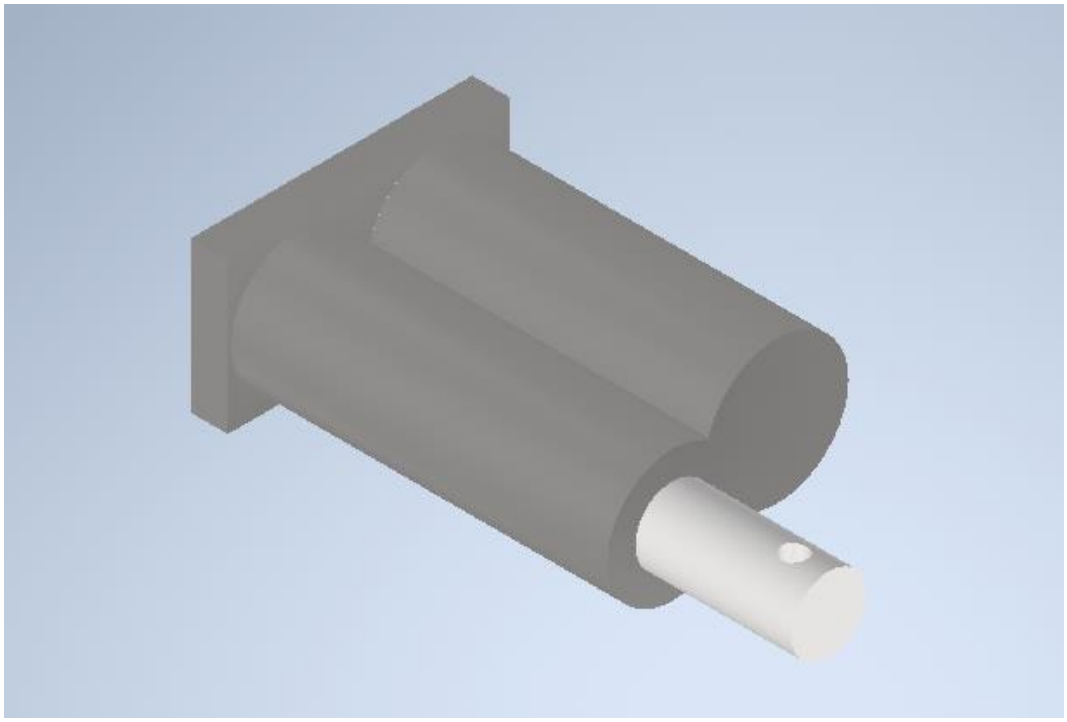


***Figura 73(1/2)**: Motor línea diseñado en autodesk inventor*

A continuación, se muestran las piezas de diseño propio y que han sido necesarias para la construcción del prototipo.

Sujeción del motor rotativo

Es la pieza encargada de soportar el peso del motor rotativo, así como sobre la que se ha diseñado el "punto A" explicado en la mecánica. El aspecto de esta pieza es el siguiente:

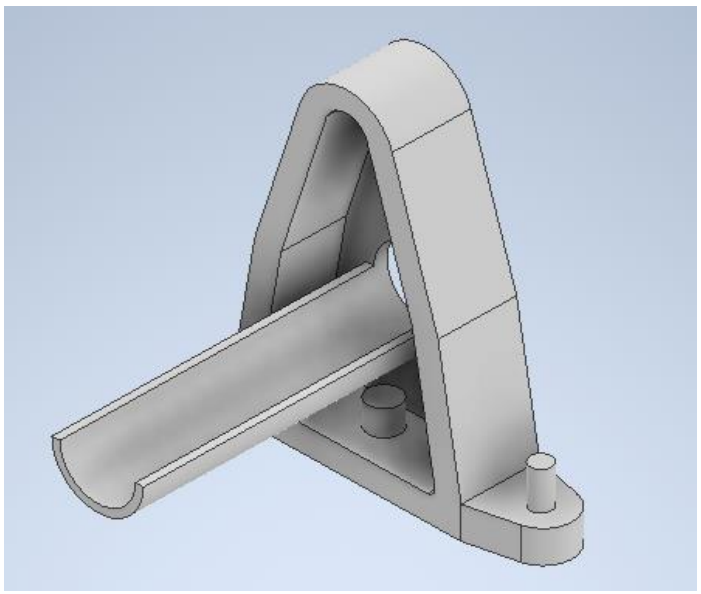
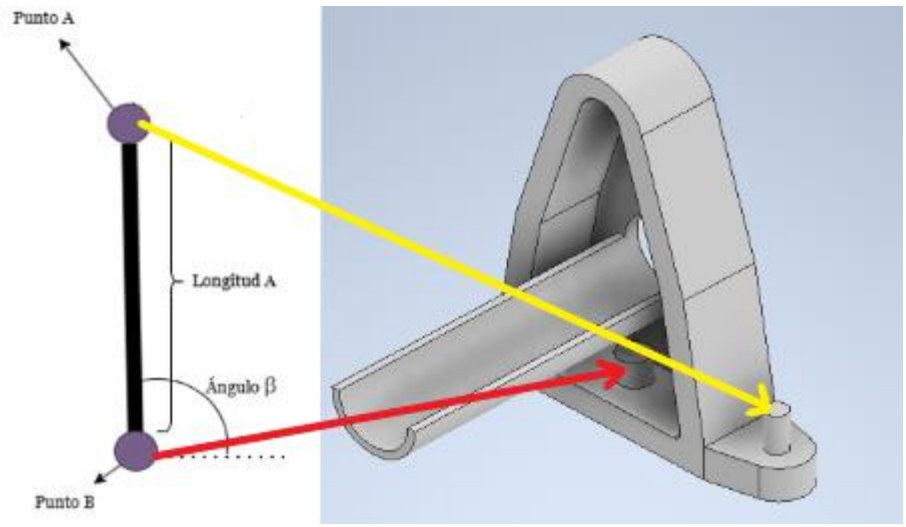


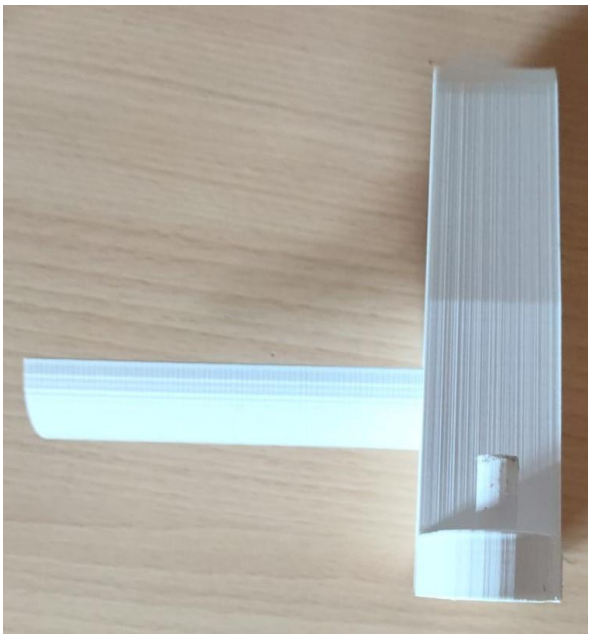
***Figura 73(2/2)**: Sujeción motor rotativo diseñada en autodesk inventor*

Podemos observar un saliente en forma de media circunferencia sobre la que se situará el motor rotativo. Debajo de esta, se observa un pequeño sobresalto en la base que corresponderá con el "punto B". A la derecha hay un pequeño pivote ("punto A"). Estos 2 puntos tienen el suficiente espacio para poder poner un rodamiento en cada uno de ellos de forma que el movimiento sobre ellos mismos sea más sencillo. A continuación, se muestra una relación entre la pieza y la mecánica desarrollada:



Se recuerda que la distancia entre estos puntos es de 6 centímetros. El "punto B" está fijo y es capaz de rotar sobre sí mismo mientras que el A aparte de rotar sobre sí mismo se desplazará en el eje horizontal. El resultado final de esta pieza es el siguiente:

***Figura 74**: Sujeción motor rotativo impresa en 3D, extraído de [elaboración propia]*



Tirante de dirección

Ya se ha explicado su necesidad anteriormente. Se ha diseñado un tirante con una longitud entre los centros de los rodamientos de 5.7 centímetros. El aspecto antes de impresión fue el siguiente:

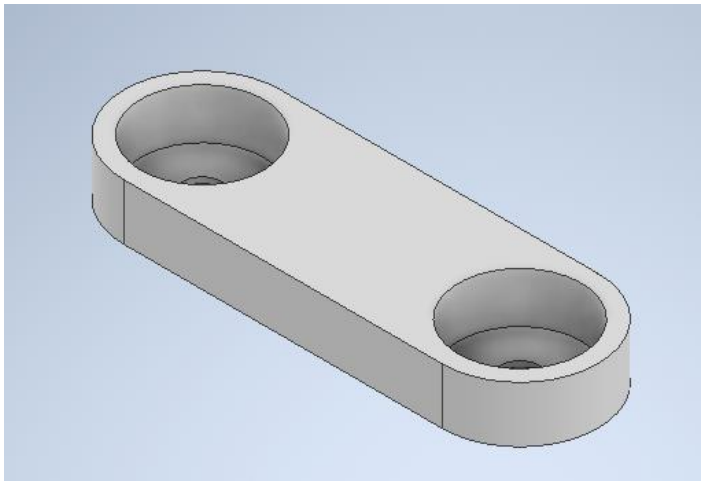


***Figura 75**: Tirante de dirección diseñado en 3D*

Una vez impresa, y con los rodamientos encajados se obtuvo la siguiente pieza:

***Figura 76**: Tirante de dirección impreso en 3D*

Sujeción de motor lineal

Debido a la fuera que puede llegar a hacer el motor lineal, es necesario anclarlo a la mesa de forma que evitemos su movimiento. Con esto conseguimos transmitir de forma completa el movimiento del vástago a la sujeción del motor rotativo. Se han diseñado dos piezas diferentes, una para la parte que comprende tanto el tubo que contiene el vástago, así como la reductora y otra únicamente para el tubo y el vástago. Estas piezas contienen una serie de agujeros que nos permiten anclarlas a una superficie con una serie de tornillos

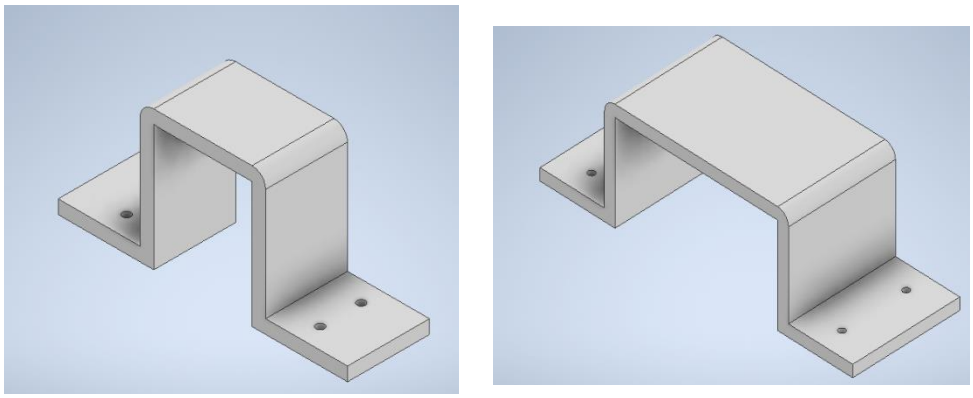


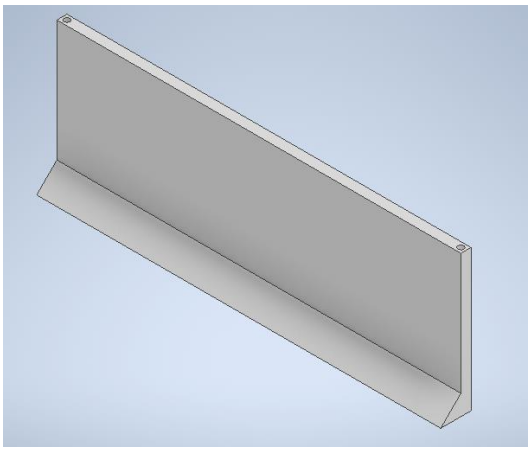
***Figura 77**: Sujeción de motor lineal diseñado en 3D*

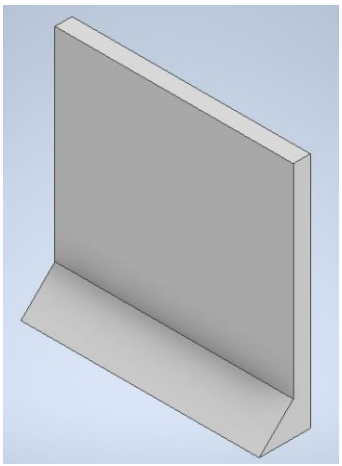
Caja de componentes

Con el fin de obtener un sistema más ordenado y menos susceptible a errores como la desconexión de cables, se ha diseñado una caja en la que se situarán todos los componentes relativos al receptor (puente en H, ESP receptor…). Esto no se ha diseñado para el emisor con el objetivo de poder mostrar una parte de cableado. La caja esta formada por 4 paredes y una tapa en la que aparecen con relieve tanto mi nombre como el nombre de este proyecto.

***Figura 78**: Caja de componentes diseñado en 3D, extraído de [elaboración propia]*

El diseño 3D de las paredes es el siguiente:





La caja ya ensamblada es la siguiente:

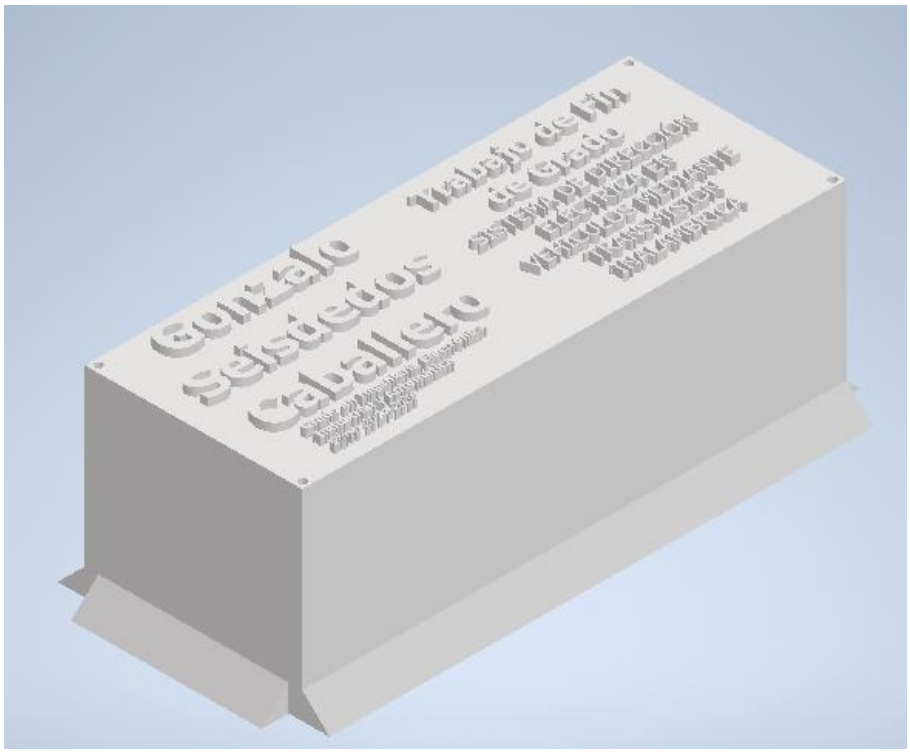


***Figura 78**: Caja de componentes diseñado en 3D*

El espacio útil de dentro tiene unas dimensiones de 7 centímetros de ancho por 20 de largo. La altura es de 7 centímetros teniendo un espacio útil de almacenaje de 980 centímetros cúbicos

NOTA: TODOS LOS PLANOS RELATIVOS A ESTAS PIEZAS SE ENCUENTRAN EN EL APARTADO DE ANEXOS SITUADO AL FINAL DEL DOCUMENTO

4.3.3 Ensamblaje final

Con todas las piezas ya diseñadas, podemos comenzar con el proceso de ensamblaje en Autodesk. Para ello se diseñó una mesa para tener una superficie sobre la que trabajar. El ensamblaje en Autodesk Inventor se realiza a través de restricciones, por lo que hemos de hablar de estas primero.

Restricciones

Las restricciones son operaciones que se pueden realizar a distintas piezas entre sí para restringir su movimiento. Para poder realizar una restricción entre

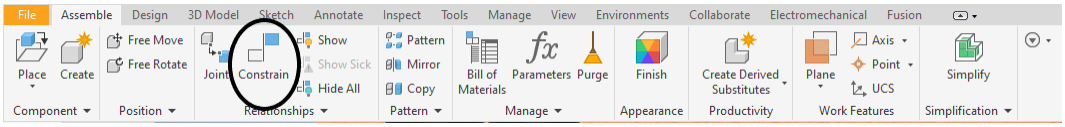


***Figura 79**: Opción de restricciones en autodesk Inventor*

2 piezas hemos de situarnos en la pestaña *Assemble* y en las opciones que se nos da hacer click en "*Constrain*"

Una vez hemos clicado en esa pestaña se nos abre el siguiente cuadro, donde podemos elegir entre distintas opciones

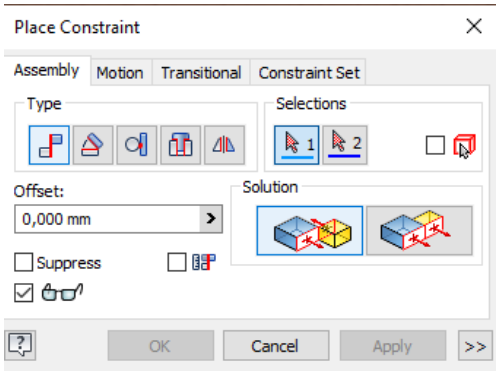


***Figura 80**: Tipos de restricciones en autodesk Inventor*

Podremos elegir entre 2 tipos de restricciones entre piezas en la opción de "*Solution*". La primera de ellas nos permite enfrentar dos piezas entre sí cara contra cara, mientras que la segunda nos permite que las 2 caras de las dos piezas se apoyen sobre un mismo plano siendo sus normales paralelas.

De esta forma, y en un mismo archivo de ensamblaje podemos restringir el movimiento del conjunto.

Ensamblaje final

Lo primero que hacemos para comenzar con el ensamblaje es situar la mesa que hemos diseñado. Esta se ha realizado para tener una superficie de referencia sobre la que trabajar.

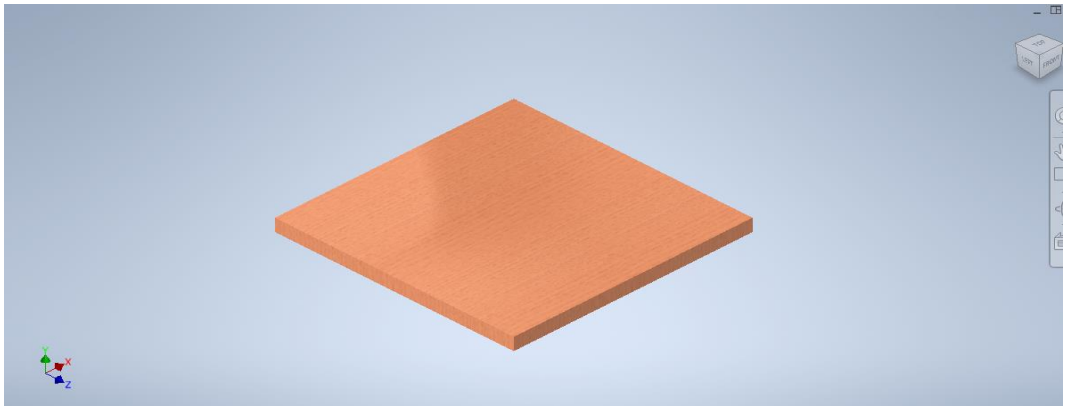


***Figura 81**: Proceso de ensamblaje (1/5)*

Sobre esta mesa se situarán todas las piezas que conforman el sistema. Tras esto, situamos el motor lineal con una restricción de primer tipo con la mesa.

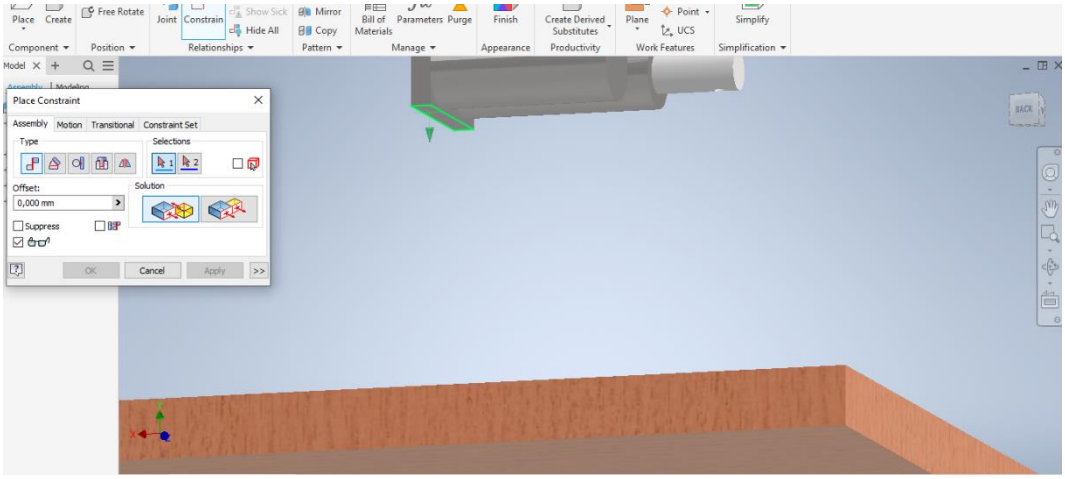
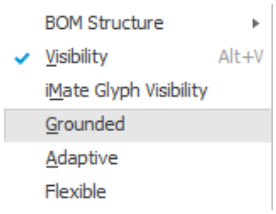


***Figura 81**: Proceso de ensamblaje (2/5)*

Con esta restricción conseguimos que el motor este sujeto a la mesa. A mayores, podemos fijar este objeto para que no se pueda mover de la posición en la que se encuentra con la opción "*grounded*".



Seguido de esto situamos el tirante de dirección y dentro de este los rodamientos. Al igual que existen restricciones entre caras, también las existen entre ejes. De esta forma podemos aplicar una restricción de ejes entre el punto central de los extremos del tirante y el punto central del rodamiento.

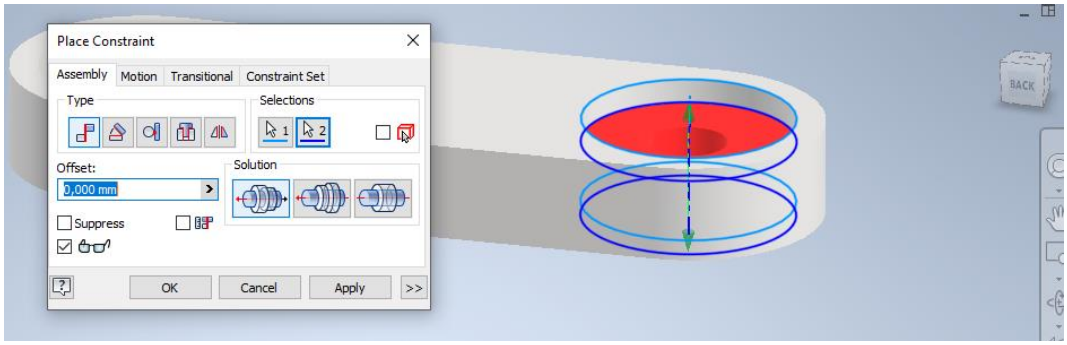


***Figura 81**: Proceso de ensamblaje (3/5)*

De igual forma, a esta forma podemos juntar este punto con el punto central del extremo del vástago, obteniendo el siguiente conjunto

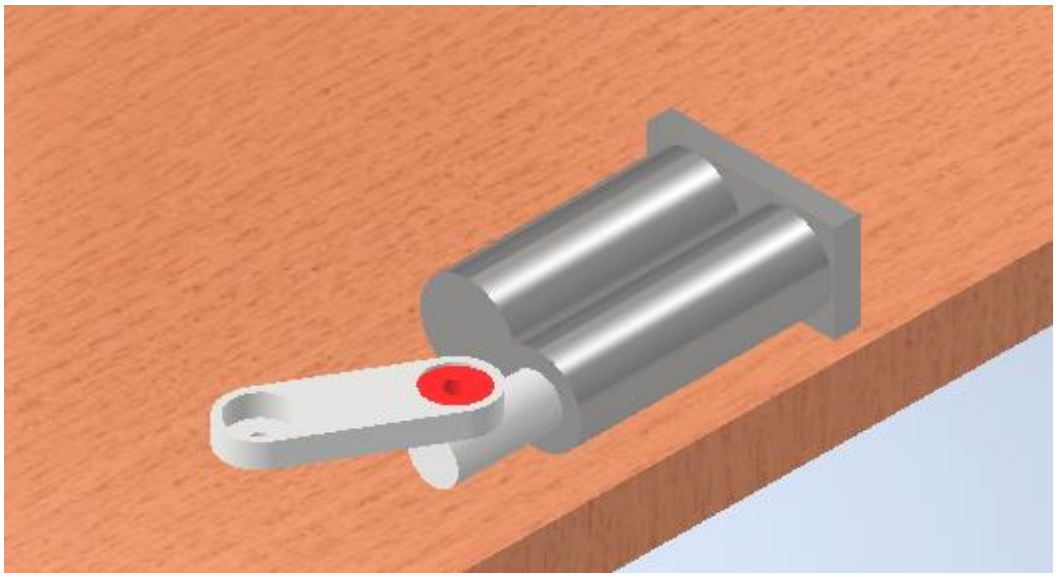


***Figura 81**: Proceso de ensamblaje (4/5)*

Por último, y siguiendo el mismo procedimiento de restricciones podemos obtener el modelo final diseñado.

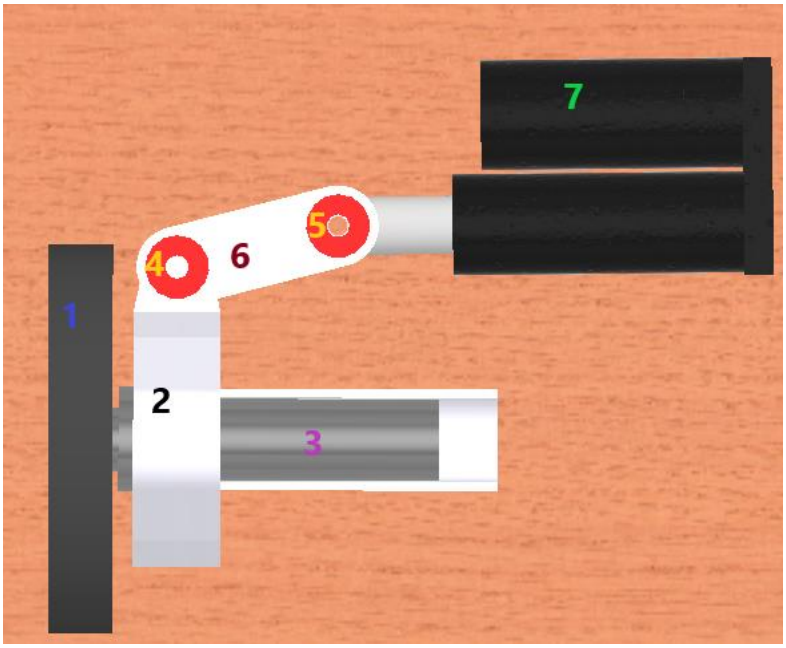


***Figura 81**: Proceso de ensamblaje (5/5)*

En la figura:

(1) Rueda del kit0085
(2) Sujeción del motor lineal
(3) Motor rotativo
(4) Rodamiento
(5) Rodamiento
(6) Tirante de dirección
(7) Motor lineal

El aspecto global del sistema es el siguiente:

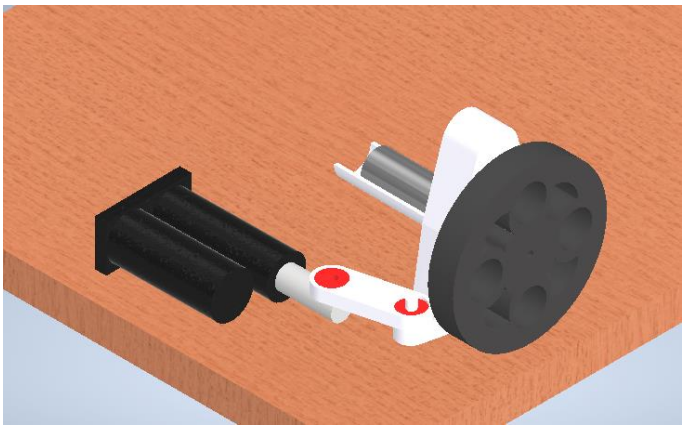


***Figura 82**: Ensamblaje final*

Una vez con las piezas situadas veremos si tanto las distancias como los ángulos que se han calculado son posible de realizar. Como se ha establecido en el apartado 4.2.3 el ángulo que debe de existir entre el tirante y el vástago del motor lineal ha de ser de 10º, la longitud entre los puntos A y B es de 6 y la longitud entre los centros de los extremos del tirante de 5.7centímetro. Esto lo podemos medir con la herramienta "*Measure*".

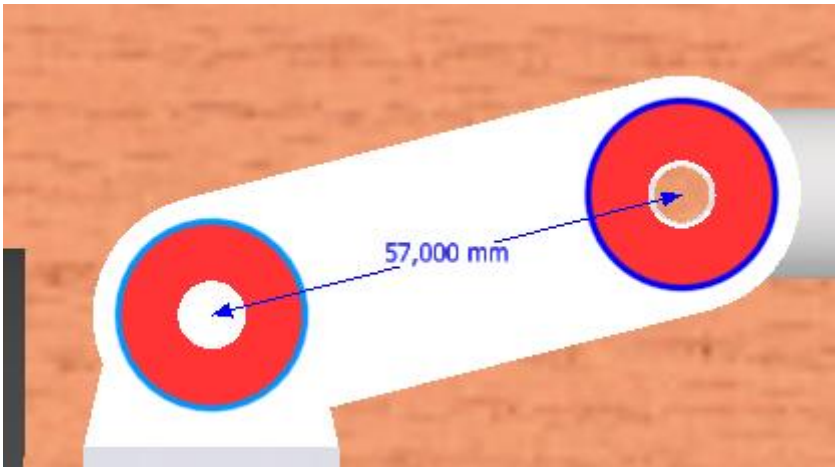


***Figura 83**: Comprobación de distancias (1/3)*

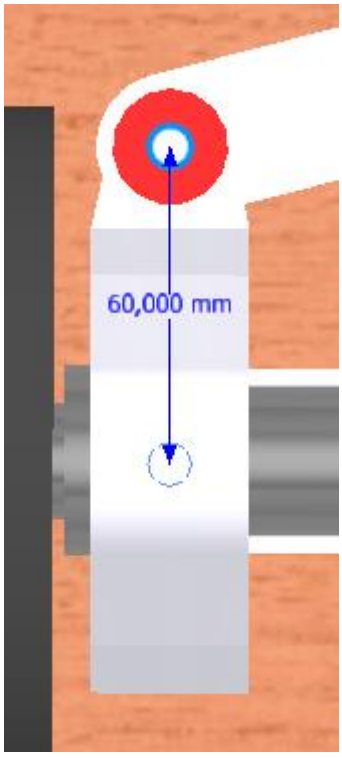


***Figura 83**: Comprobación de distancias (2/3)*

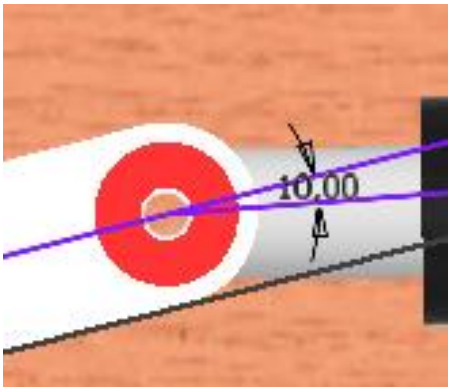


***Figura 83**: Comprobación de distancias (3/3)*

Con estas distancias, podremos probar distintas situaciones para comprobar que el diseño de la mecánica es viable

4.3.4 Pruebas del diseño

El sistema en reposo es el mostrado anteriormente. Probaremos dos casos, primero si el vástago avanza y luego si el vástago retrocede. Recordemos que la posición inicial del sistema en con el vástago salido unos 4.59 centímetros y ha de llegar hasta 4.59 centímetros más (siendo su longitud total de 10 centímetros).

Vástago avanzando:

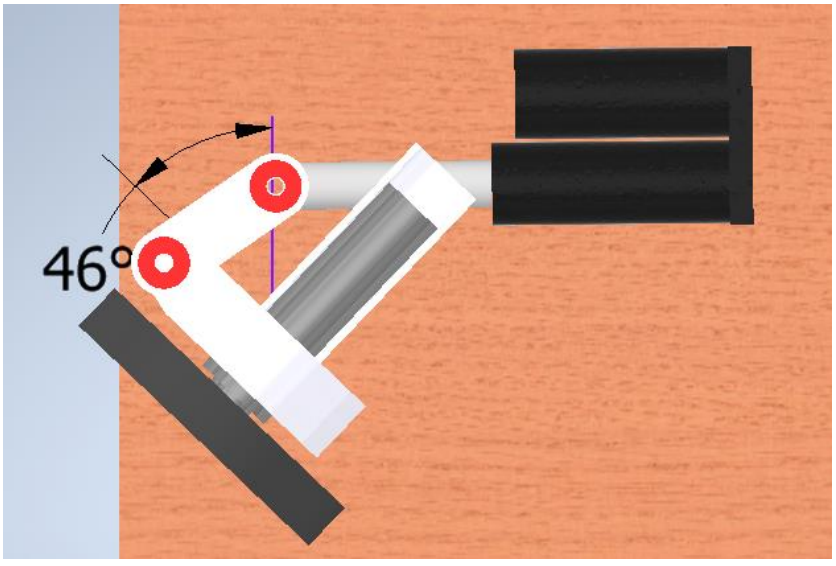


***Figura 84**: Giro antihorario*

Cuando avanza el vástago la forma del sistema es la siguiente. El ángulo de giro en este caso puede ser superior a los 45º pudiendo llegar hasta los 50º (a pesar de que en la figura se muestre 46º).

Vástago retrocediendo:

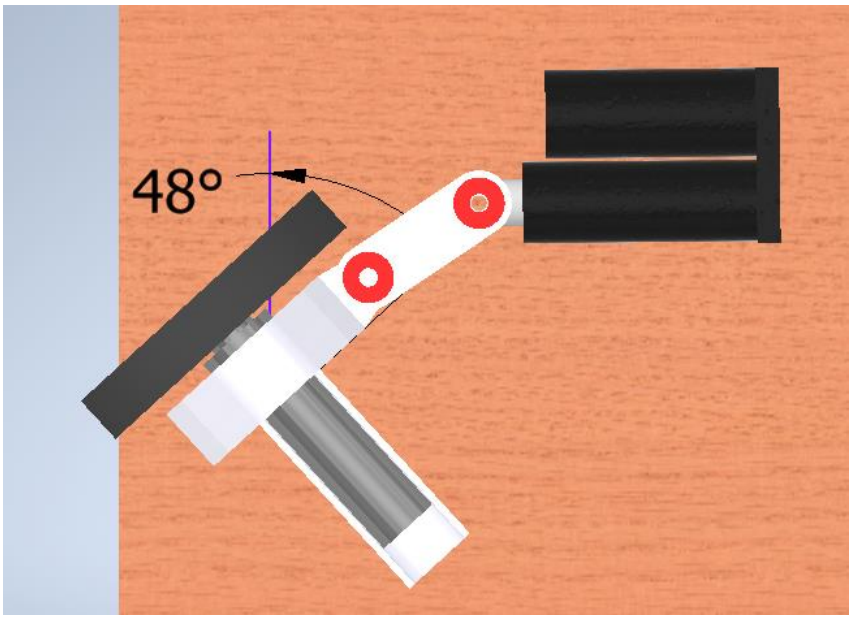


***Figura 85**: Giro horario*

En este caso, el ángulo máximo también es de 50º (aunque la captura muestre un ángulo de 48º)

Capítulo 5:

Programación y explicación de códigos

Capítulo 5: Programación y explicación de códigos

El objetivo de este capítulo es la explicación de los códigos que se han cargado en los ESP 32, como se han desarrollado, así como su funcionamiento. Como se ha explicado en anteriores apartados el protocolo de comunicación usado es el ESP-NOW a través del cual podemos enviar información entre los dispositivos sin necesidad de que estén conectados de forma física. De esta forma, cada ESP tendrá asignado un rol. Un ESP funcionará como emisor y otro como receptor.

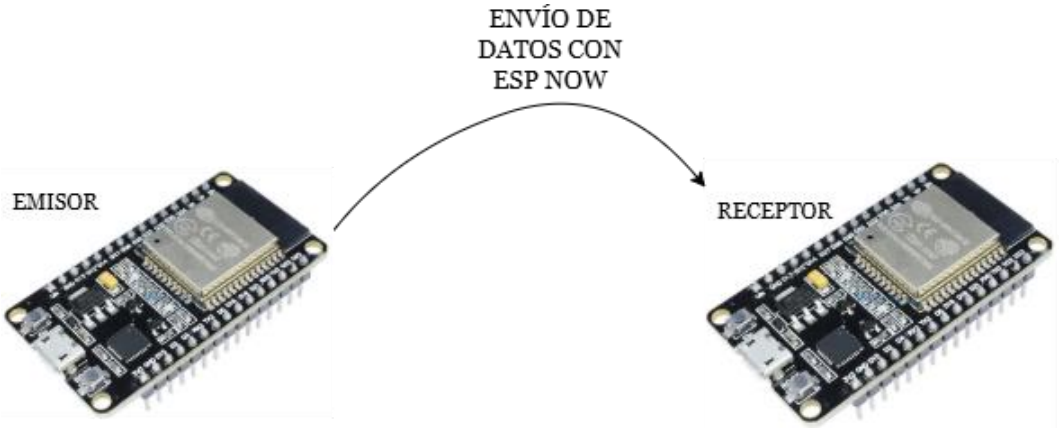


***Figura 86**: Esquema de funcionamiento del protocolo*

5.1 Funciones del emisor y del receptor

Cada ESP tiene una serie de funciones a realizar, y siempre de la misma forma. Las funciones simplificadas de cada uno de ellos se muestran a continuación:

- EMISOR
  1) Recibe la información del encoder
  2) Inicia la comunicación
  3) Cifrado AES 128
  4) Envío de la información

***Figura 87**: Diagrama de flujo del código emisor*

- RECEPTOR
  1) Recepción
  2) Descifrado de AES 128
  3) Interpretación de ángulo
  4) Control de motor rotativo
  5) Control de motor lineal

***Figura 88**: Diagrama de flujo del código receptor*

> **NOTA**: En los diagramas se muestra el bloque "Procesamiento de datos". Se explicará más adelantes como y que se realiza en ese

Una vez explicadas de forma muy breve el objetivo de cada ESP, vamos a explicar de forma muy detallada el funcionamiento de cada código desarrollado.

5.2 Código en el emisor

Se trata del código que va cargado en el ESP emisor. Cada código se programa en el entorno Arduino IDE. Mas tarde, se ha de compilar el código para saber si se ha desarrollado de forma correcta para poder más tarde subirlo al propio ESP. En el proceso de carga del código se muestra una pantalla similar a la mostrada en la figura [89].

***Figura 89**: Pantalla carga de Arduino IDE*

Esta pantalla muestra información muy importante: si el chip se ha detectado correctamente, el proceso de escritura en la memoria flash y la finalización del proceso.

Comenzaremos comentando ya todo el código

5.2.1 Librerías utilizadas

Todo código suele comenzar con la importación de una serie de librerías. Estas son las encargadas de proveer un conjunto de herramientas ya desarrolladas de forma que se puedan implementar en el código. Gracias a ellas, disponemos de datos o funciones sin necesidad de desarrollarlas en nuestro programa, mejorando así la eficiencia y legibilidad del código.

Las librerías que han sido añadidas en este código son las siguientes (figura [90])

```
#include <WiFi.h>
#include <esp_now.h>
#include <mbedtls/aes.h>
```

***Figura 90**: Librerías código emisor*

"#include <WiFi.h>": Se trata de una librería que se añade para proporcionar al ESP las funciones necesarias para que se pueda conectar a redes Wi-Fi. Sin embargo, en nuestro proyecto no deseamos conectarnos a ninguna red Wi-Fi por lo que, ¿por qué es necesaria esta librería?

El protocolo ESP-NOW desarrollado por *Expressif* nos permite realizar la comunicación entre 2 dispositivos sin necesidad de que estos dos estén conectados. Sin embargo, existen diversas razones por la que se necesita esta librería Wi-Fi:

- ESP-NOW hace uso del hardware de Wi-Fi haciendo uso del chip Wi-Fi del ESP32
- La configuración del modo Wi-Fi sigue siendo necesaria. Se ha de configurar el ESP en base al modo en el que deseamos que funcione(modo estación, modo AP, canal, etc)

La librería Wi-Fi nos permite realizar estas acciones.

#include <esp_now.h>: Gracias a esta librería podemos hacer uso del protocolo ESP-NOW. Gracias a esta librería podemos realizar acciones como inicializar el protocolo y la comunicación, añadir dispositivos, enviar datos o recibirlos y notificar envíos. Todas estas funciones se han implementado en el código que más adelante veremos.

#include <mbedtls/aes.h>: Es usada para incluir la funcionalidad del algoritmo de encriptación AES128 ("*Advanced Encryption Standard*"). Nos permite

trabajar con claves de 128, 192 o 256 bits. Esta librería es imprescindible para poder realizar la encriptación de forma correcta.

5.2.2 Definiciones y configuraciones

Previo a comenzar con la redacción de tanto el Setup() como el Loop(), hemos de asignar los pines del ESP para la lectura del encoder así como configurar otras funcionalidades. Estas líneas de código se muestran en la figura [91]

Lo primero de todo, y para poder usar el encoder de forma correcta, hemos de definir una serie de pines del ESP que nos permitan la lectura de los datos. Esto lo realizamos definiendo 2 de los pines del ESP, en este caso el pin 18 y el 19.

```
#define ENCODER_PIN_A  18
#define ENCODER_PIN_B  19
#define PPR 1000
#define CPR (PPR * 2)

int botonPin = 2;
int ledPin = 4;          // LED en GPIO 4

int estadoBoton = 0;
int ultimoEstado = 1;
int envioActivo = 0;
```

***Figura 91**: Definiciones del código de ESP emisor*

"*#define ENCODER_PIN_A 18*": En este caso, a este pin se conectará el cable de datos A (cable blanco)

"*#define ENCODER_PIN_B 19*": Se conectará el cable de datos B (cable verde)

El resto de los cables del encoder corresponden a la alimentación, por lo que no se trata en el código.

Seguido a la definición de estos pines, nos encontramos con la definición de los pulsos por revolución del encoder (PPR) y de ciclos por revolución (CPR).

Pulsos por revolución del encoder ("PPR): Este concepto se refiere al número de pulsos eléctricos generados por el encoder en una revolución completa. Esta característica es propia del encoder y proporcionado por el fabricante, por lo que, en nuestro caso tendrá un valor de 1000. Sin embargo, nuestro encoder es un encoder de cuadratura por lo que se generan 1000 pulsos por A y otros 1000 por B, por lo que el número de pulsos totales será de 2000 ("*#define PPR 1000*2").*

A continuación, se declaran los pines en los que se situarán un led (que nos indicará si se está produciendo la comunicación) y un botón (en caso de que se pulse, se cortará o iniciará de nuevo la comunicación). Junto a esto se hace la declaración de

unas variables de estado, que nos informarán del estado en el que se encuentra tanto el sistema como el botón.

Las siguientes líneas de código tienen conexión directa con el protocolo ESP-NOW. Primero de todo, se declara la dirección MAC del receptor.

```
uint8_t direccion_receptor[] = {0x3C, 0x8A, 0x1F, 0x9B, 0x92, 0x10};
```

***Figura 92**: Definiciones de la MAC del receptor*

Con un pequeño código podemos acceder a la MAC del receptor. Tras esto, añadimos al código del emisor la dirección MAC del receptor. Esto se muestra en la línea de código mostrada en la figura [92].

Tras esto, definimos la clave AES que se ha decidido poner para este proyecto. Por ello, el receptor ha de tener esta clave para poder leer los datos que se envíen.

```
uint8_t aes_key[16] = {
    0x01, 0x02, 0x03, 0x04,
    0x05, 0x06, 0x07, 0x08,
    0x09, 0x0A, 0x0B, 0x0C,
    0x0D, 0x0E, 0x0F, 0x10
};
```

5.2.3 Volátiles y funciones auxiliares

Tras la declaración de los elementos asociados al protocolo ESP-NOW podemos comenzar con la creación de funciones y la declaración de datos.

En primer lugar, nos encontramos con la declaración de dos datos de tipo volátil (figura [93]).

```
volatile int position = 0;
volatile int lastStateA = 0;
```

***Figura 93**: Definición de datos volátiles en el receptor*

Los datos "*volátiles*" son aquellos que son susceptibles de cambio en cualquier momento, fuera del control del programa principal. Son muy útiles cuando se trabaja con interrupciones que, como se verá más adelante, será la forma en que trabaje nuestro programa. Tras esto comienza la declaración de las distintas funciones que se usarán a lo largo del programa.

La función "***void IRAM_ATTR encoderISR()***" es la encargada de gestionar las señales del encoder. El término ISR (procedente del nombre encoderISR) hace referencia a "*Interrupt Service Routine*" informándonos así que es una función que hace uso de interrupciones. Por su parte, "IRAM_ATTR" nos indica que la función se debe almacenarse en la IRAM (memoria interna de la RAM) debido a que se ejecutará más rápido.

```
void IRAM_ATTR encoderISR() {
    int stateA = digitalRead(ENCODER_PIN_A);
    int stateB = digitalRead(ENCODER_PIN_B);

    if (stateA != lastStateA) {
        if (stateA == stateB) {
            position++;
        } else {
            position--;
        }
    }
    lastStateA = stateA;
}
```

***Figura 94**: Código para lectura del encoder*

La función comienza con la lectura de los estados A y B procedentes de los pines 18 y 19. La rutina se activa cuando detecta un cambio en ENCODER_PIN_A (correspondiente con el pin 18 del ESP), esto se realiza comparando el estado de A actual con el estado que se leyó la última vez que se almacena en una variable llamada "*lastStateA*". En caso de que exista un cambio en este valor, se realiza una nueva comparación, que es si el estado de A es igual al estado de B. En caso de que sean igual, se aumenta el valor de la posición (lo que nos indica que el giro es horario). En caso de que los estados sean diferentes el valor de la posición se disminuye (nos indica que el giro es antihorario).

La siguiente función con la que nos encontramos es "***void envíoDatos()***" (figura[95])

```
void EnvioDatos(const uint8_t *mac, esp_now_send_status_t status) {
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Envio exitoso" : "Envio fallido");
}
```

***Figura 95**: Función de envío de datos*

Esta función es un callback de envío en ESP-NOW, y su objetivo es informarnos si el envío de información se ha realizado de forma correcta. Se llama callback a aquellas funciones que no se llaman directamente en el código, si no que se ejecutan automáticamente cuando ocurre un determinado evento. En nuestro caso, cuando el ESP termine de enviar un mensaje se llama directamente a

esta función. Como se puede observar, esta función toma los siguientes argumentos:

- "*Const unit8_t *mac*": Es un puntero a la dirección MAC del receptor
- *"esp_now_send_status_t status"*: Indica si el envío ha sido correcto o no

En caso de que la variable "*status*" tome el valor "ESP_NOW_SEND_SUCCESS" se imprimirá por pantalla la frase "Envío exitoso". En caso de que la variable "*status*" tome otro valor, se imprimirá "envío fallido" haciéndonos saber que el envío no se ha podido realizar de forma correcta.

La última función previa al setup() es la función "**Cifrar AES** ()". El propósito de esta función es cifrar un bloque de 16 bytes usando AES 128 en modo ECB.

```
void cifrarAES(uint8_t* data, size_t len) {
    mbedtls_aes_context aes;
    mbedtls_aes_init(&aes);
    mbedtls_aes_setkey_enc(&aes, aes_key, 128);
    mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, data, data);
    mbedtls_aes_free(&aes);
}
```

***Figura 96**: Función de cifrado AES*

La función toma como argumentos un punto que apunta a los datos a cifrar ("data") y el tamaño de estos datos. Tras esto, se crea una estructura que guarda toda la información que se necesita para realizar el cifrado ("*mbedtls_aes_context aes*"). Una vez creada la estructura se inicia el AES (*mbedtls_aes_init(&aes)*). Una vez iniciado, se asigna la clave de cifrado y se indica que se usaran 128 bits ("*mbedtls_aes_setkey_enc(&aes, aes_key, 128)*"). Una vez hecho todo esto, se procede con el cifrado en ECB ("*mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, data, data)*"), donde el primer data hace referencia a los datos sin cifrar y el segundo "*data*" hace referencia a los datos cifrados. Este método solo cifra datos de 16 bytes, por lo que, si la longitud de los datos a cifrar no es esa, el resultado no será correcto. Por último, se libera la memoria del contexto AES.

Una vez explicado todo esto, podemos comenzar con la explicación del setup y loop.

5.2.4 *SetUp ()* del emisor

En este bloque se sitúan todas las acciones que se realizan una única vez al inicio del programa del ESP32.

```
void setup() {
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);

    pinMode(ENCODER_PIN_A, INPUT_PULLUP);
    pinMode(ENCODER_PIN_B, INPUT_PULLUP);
    pinMode(botonPin, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);

    digitalWrite(ledPin, LOW);  // LED apagado al inicio

    lastStateA = digitalRead(ENCODER_PIN_A);
    attachInterrupt(digitalPinToInterrupt(ENCODER_PIN_A), encoderISR, CHANGE);

    if (esp_now_init() != ESP_OK) {
        Serial.println("Error al inicializar ESP-NOW");
        return;
    }

    esp_now_register_send_cb(EnvioDatos);

    esp_now_peer_info_t peerInfo = {};
    memcpy(peerInfo.peer_addr, direccion_receptor, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;

    if (esp_now_add_peer(&peerInfo) != ESP_OK) {
        Serial.println("Error al registrar peer");
    }
}
```

***Figura 97**: SetUp() en el emisor*

Lo primero de todo, como en cualquier código de este estilo, es iniciar la comunicación serial y dar un valor a los baudios. Los baudios representan la velocidad de transmisión de datos, y en nuestro caso tiene un valor de 115200. Tras esto, se inicia el modo Wi-Fi como estación (*WIFI_STA*), necesario para poder usar ESP-NOW.

Como se ha explicado anteriormente, las entradas de encoder necesitan de una resistencia pull-up, por lo que configuramos esos pines con entrada pull-up en las 2 entradas procedentes del encoder. También se declara el pin asociado al led con resistencia input de forma que evitamos realizar un cortocircuito (igual se realiza con el botón). Mediante el uso de "*digitalWrite*" podemos iniciar el led apagado. Además de esto, para poder usar las interrupciones, se ha de guardar el estado inicial del pin A para poder comparar cambios dentro de la ISR. Una vez hecho esto, se asocia la función "encoderISR" a las interrupciones del pin A. Una vez configuradas las interrupciones del encoder, pasamos a configurar el protocolo ESP-NOW.

Lo primero de todo es comprobar si el protocolo se ha iniciado de forma correcta. Para ello hacemos uso de un "IF" de forma que si no se ha podido configurar de forma correcta se imprime "Error al inicializar el ESP-NOW". Si falla se sale del setup () para evitar errores. Con el protocolo ya iniciado de forma correcta, se procede a llamar a la función envioDatos (que es una función callback) cada vez que se realice un envío a través de ESP NOW.

Una vez creado este callback que nos indica si el envío se realiza de forma exitosa, creamos una estructura llamada "*peerInfo*" donde se almacenarán los datos del receptor (se llama así debido al modelo de comunicación "Peer to Peer"). Se añade la dirección MAC del receptor a esta estructura para saber donde queremos enviar los datos. Con la dirección ya almacenada, indicamos que el canal que se ha de usar ha de ser el mismo canal Wi-Fi ("*channel=0*") y que no se use la encriptación de ESP debido a que ya hemos usado AES. Por último, se añade el Peer receptor a la lista de peers con los que este dispositivo puede conectarse y en caso de que falle se imprime un error.

5.2.5 *Loop ()* del emisor

El código loop () es el que se ejecuta de forma repetida en el ESP emisor. El código comienza comprobando el estado del botón y si este ha sido pulsado.

```
void loop() {
    estadoBoton = digitalRead(botonPin);

    if (estadoBoton != ultimoEstado) {
        delay(50);  // antirrebote básico
        if (estadoBoton == LOW) {
            envioActivo = !envioActivo;

            // Ahora el LED sigue el estado inverso del envío
            if (envioActivo) {
                digitalWrite(ledPin, LOW);   // envío activo -> LED apagado
                Serial.println("Envio ACTIVADO");
            } else {
                digitalWrite(ledPin, HIGH);  // envío inactivo -> LED encendido
                Serial.println("Envio DETENIDO");
            }
        }
    }

    ultimoEstado = estadoBoton;
```

***Figura 98**: Loop() en el emisor*

Comenzamos asignando un valor a la variable "*estadoBoton*", que tomará el valor que haya en el pin del botón. En caso de que se detecte un cambio ("*estadoBoton != ultimoEstado*"; la variable *ultimoEstado* se ha inicializado anteriormente con un valor de 1) se entra dentro de un "*if*" en donde primero se comprueba que el cambio de estado fue porque el botón fue presionado (if

(estadoBoton == LOW) de forma que se comprueba que el botón ha sido presionado y se ha soltado) y una vez dentro de este if se cambia el estado de la variable "envioActivo", en caso de que estuviese a 1 se pasa a 0 y viceversa (envioActivo=!envioActivo). Por último, en caso de que el envío esté desactivado se enciende el LED y se imprimirá por el monitor serial "Envío DETENIDO".

```
    if (envioActivo) {
        int pos_snapshot;
        noInterrupts();
        pos_snapshot = position;
        interrupts();

        float angle = (pos_snapshot / (float)CPR) * 360.0;

        uint8_t buffer[16] = {0};
        memcpy(buffer, &angle, sizeof(angle));

        cifrarAES(buffer, 16);

        esp_now_send(direccion_receptor, buffer, 16);
        Serial.print("Ángulo enviado: ");
        Serial.println(angle);
    }

    delay(100);
}
```

***Figura 99**: Envío de datos en el emisor*

Para entender este código hay que tener muy claro la función de la variable

"position". La variable position es una variable compartida con la función "encoderISR" que se ejecuta cada vez que se detecta un cambio en el encoder, variando así el valor de la variable position. Por ello, primero se desactivan las interrupciones para que mientras se guarda la variable "position" en otra variable llamada "pos_snapshot" el valor de "position" no cambie. Una vez guardado este valor, las interrupciones se vuelven a activar.

Con el valor ya guardado, podemos proceder a calculas el valor del ángulo en base a la posición. Para ello se usa la siguiente fórmula:

$$Ángulo = \frac{pos\_snapshot}{PPR} * 360º \quad [14]$$

La variable pos_snapshot es de tipo int al igual que PPR, por lo que, al hacer la división, el ángulo se truncaría y no obtendríamos el valor exacto.

$$\frac{Dato\ int}{Dato\ int} = Resultado\ int \qquad \frac{Dato\ int}{Dato\ float} = Resultado\ float$$

Un ejemplo de este comportamiento sería el siguiente:

Supongamos que "pos_snapchot" tiene un valor de 3 y nuestro PPR de 4. Si los dos datos fuesen de tipo int la operación sería la siguiente:

$$\frac{3}{4} = 0\ (a\ pesar\ de\ que\ el\ valor\ sea\ 0.75)$$

En caso de que el valor de PPR sea de tipo float, el resultado sería de 0.75 (el número exacto). Por ello, se define PPR como un float para forzar al programa a que el valor de "Ángulo" sea de tipo float y el resultado sí que tenga decimales.

Una vez calculado el valor del ángulo, creamos un buffer de 16 bytes (debido a que nuestro cifrado AES cifra para 16 bytes) y almacenamos los 4 bytes del ángulo en este buffer. Tras esto, llamamos a la función "*cifrarAES*" para cifrar todos los datos. Por último, se envían los 16 bytes cifrados del buffer al receptor y se realiza una espera de 100 milisegundos para ejecutar de nuevo el código.

5.2.6 Diagrama de funcionamiento

Para visualizar mejor el funcionamiento del código se muestra a continuación el diagrama de funcionamiento del emisor:

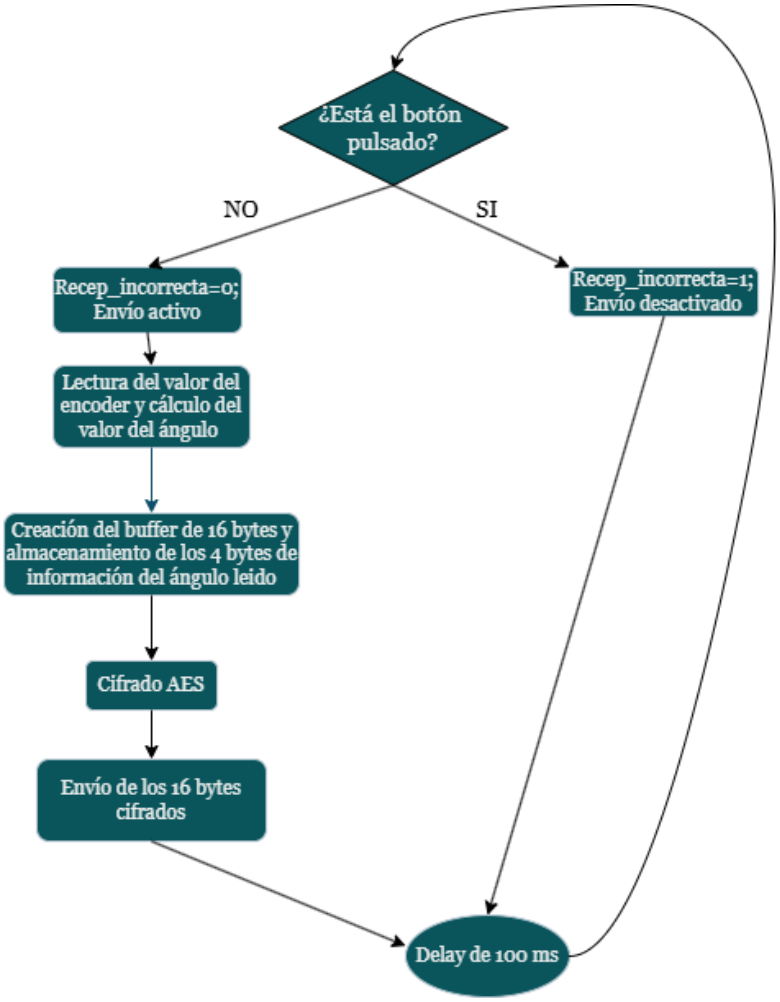


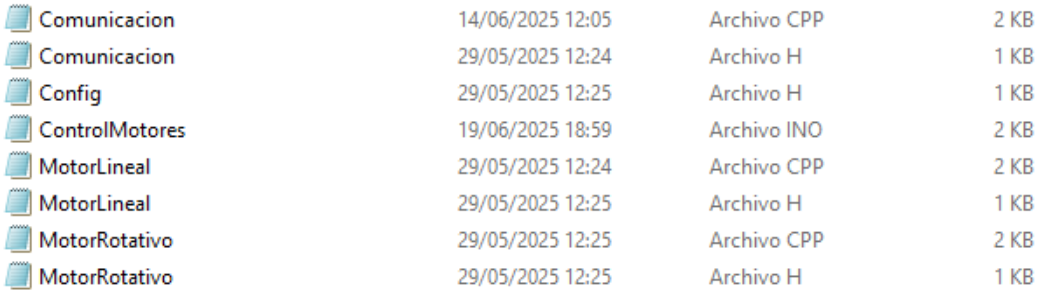
***Figura 100**: Diagrama de flujo en el emisor completo*

5.3 Código en el receptor

En este apartado nos dedicaremos únicamente a explicar cómo funciona y que conforma el código que se carga en el ESP receptor.

5.3.1 Organización y estructura de archivos

Al tratarse de un código bastante complejo y con muchas líneas de programación, se decidió que la mejor manera para poder organizar todo el código era el uso de módulos funcionales. Por ello disponemos de un código principal (llamado ControlMotores.ino) y una serie de módulos que completan este código principal (distribuidos en archivo .cpp y .h).



Comunicacion	14/06/2025 12:05	Archivo CPP	2 KB
Comunicacion	29/05/2025 12:24	Archivo H	1 KB
Config	29/05/2025 12:25	Archivo H	1 KB
ControlMotores	19/06/2025 18:59	Archivo INO	2 KB
MotorLineal	29/05/2025 12:24	Archivo CPP	2 KB
MotorLineal	29/05/2025 12:25	Archivo H	1 KB
MotorRotativo	29/05/2025 12:25	Archivo CPP	2 KB
MotorRotativo	29/05/2025 12:25	Archivo H	1 KB

***Figura 101**: Archivos del código del receptor*

A continuación, se comenzará a explicar que contiene cada código y como se ha programado.

5.3.2 Módulo funcional de comunicación (Comunicación.cpp y Comunicación.h)

5.3.2.1 Comunicación.h

El código comienza con "*#pragma once*" que indica al compilador que este archivo solo se incluya una vez durante la compilación.

```
#pragma once
#include <esp_now.h>

void RecibirDatos(const esp_now_recv_info_t *info, const uint8_t *datos, int len);
```

***Figura 102**: Código de Comunicación.h*

Tras esto comenzamos con la inclusión de librerías.

"#include <esp_now.h>": Usada también en el código del emisor, permite hacer uso de funciones propias del protocolo ESP-NOW.

En este caso, el código solo incluye dicha librería. Con esto ya hecho se declara la función "void recibirDatos". Esto se realiza con la finalidad de que otros archivos (en nuestro caso ControlMotores.ino) pueda hacer uso de dicha función (que se implementa en el Comunicación.cpp)

5.3.2.2 Comunicación.cpp

En este archivo es donde de implementan todas las funciones relacionadas con la comunicación (como la anteriormente nombrada "*RecibirDatos*")

```
#include <mbedtls/aes.h>
#include "Config.h"
#include <esp_now.h>
#include <WiFi.h>
```

***Figura 103**: Librerías de comunicación.cpp*

Las librerías que se implementan en este archivo son:

"#include <WiFi.h>": Usada en el código del emisor y explicada con mas detalle en el apartado dedicado a él. Proporciona al ESP las funcionalidades Wi-Fi.

"#include <mbedtls/aes.h>": Es la última librería en común al emisor. Es usada para poder realizar la encriptación en AES 128.

"include<Config.h>": Se incluye debido a que en este documento (que se explicará mas tarde) se declaran e inician algunas variables que son necesarias para la implementación de las funciones declaradas en este archivo

También se usa "*esp_now.h"*, ya explicada anteriormente.

A continuación, se declaran las variables recep_incorrecta (que nos indica si la recepción de los datos se ha realizado de forma correcta o por el contrario existe algún error), última_recepción (que es el último instante de tiempo donde se ha recibido un dato) y las variables que guardan el ángulo del volante tanto actual como de la anterior pasada.

```
volatile int recep_incorrecta = 0;
unsigned long ultimaRecepcion = 0;
float angulo_volante = 0;
float angulo_anterior = 0;
```

***Figura 104**: Variables de comunicación.cpp*

Se introduce la clave del cifrado AES que ha de coincidir con la implementada en el emisor. Junto a esto, se declara la función "descifrarAES".

```
uint8_t aes_key[16] = {
  0x01, 0x02, 0x03, 0x04,
  0x05, 0x06, 0x07, 0x08,
  0x09, 0x0A, 0x0B, 0x0C,
  0x0D, 0x0E, 0x0F, 0x10
};
```

***Figura 105**: Clave AES en comunicación.cpp*

La primera función, es ***descifrarAES*** y es la encargada de la desencriptación del mensaje.

```
void descifrarAES(uint8_t* data, size_t len) {
  mbedtls_aes_context aes;
  mbedtls_aes_init(&aes);
  mbedtls_aes_setkey_dec(&aes, aes_key, 128);
  mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_DECRYPT, data, data);
  mbedtls_aes_free(&aes);
}
```

***Figura 106**: Función descifrar en comunicación.cpp*

Esta función recibe como argumentos un puntero a un arreglo de bits. El número de bytes que contiene ese arreglo es lo que nos indica la variable "len", que debe de ser de 16 bytes (para ser compatible con AES). Tras esto se crea el contexto de AES ("") y una vez creado se inicia ("*mbedtls_aes_init(&aes)*"). Estando inicializada la estructura del contenido se carga la clave de descifrado ("*mbedtls_aes_setkey_dec(&aes, aes_key, 128)*"). AES key es un arreglo de 16 bytes y el 128 indica que se usa AES 128. Tras esto, se realiza ya el descifrado de los datos que se han recibido a través del proceso de descifrado en modo ECB ("*mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_DECRYPT, data, data)*"). Por último, hemos de liberar los recursos. De esta forma ya se ha producido el descifrado de los datos. El diagrama de flujo de esta función es muy sencillo y se muestra a continuación:

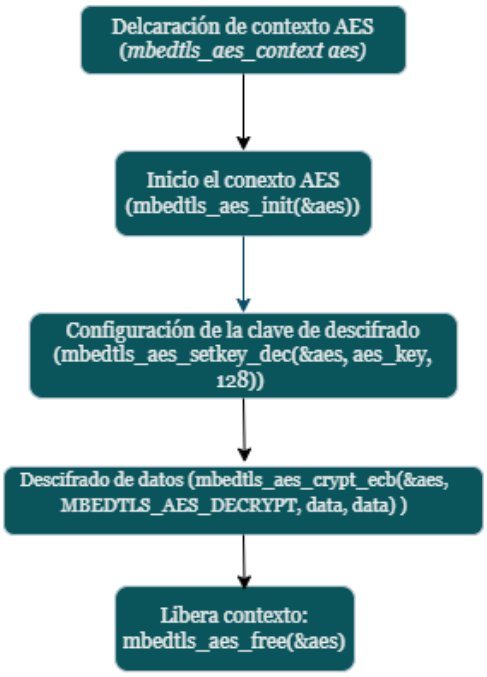


***Figura 107**: Diagrama de flujo de descifrar AES*

Por último, se implementa la función ***recibirDatos***:

```
void RecibirDatos(const esp_now_recv_info_t *info, const uint8_t *datos, int len) {
  if (len == 16) {
    uint8_t decrypted[16];
    memcpy(decrypted, datos, 16);
    descifrarAES(decrypted, 16);
    memcpy(&angulo_volante, decrypted, sizeof(float));

    recep_incorrecta = 0;
    ultimaRecepcion = millis();

    Serial.print("Ángulo recibido: ");
    Serial.println(angulo_volante);
  } else {
    recep_incorrecta = 1;
    Serial.println("Error: datos recibidos con LONGITUD INCORRECTA");
  }
}
```

***Figura 108**: Función Recibir Datos*

Esta es una función cuyo objetivo es mostrarnos los datos que se van recibiendo. La función recibe como argumentos la información del emisor (en este caso no se usa, pero es necesario para hacer funcionar de forma correcta el *callback* y es útil para futuras mejoras). Los otros dos argumentos son: un puntero al arreglo de bytes recibido y la longitud del arreglo, que ha de ser de 16 bytes).

La función continua con un *if* con el objetivo de diferenciar datos de distinta longitud. Si se recibiesen datos de otra longitud se generaría un problema de compatibilidad por lo que en caso de que la longitud no sea de 16 bytes la variable *recep_incorrecta* se pondría a 1 indicando que hay un problema con la recepción de los datos. En caso de que la longitud sea la adecuada se comienza con la recepción de los datos. Se crea un arreglo de datos llamado *decrypted* donde almacenaremos los datos descifrados (“*uint8_t decrypted[16]*”) *y se usa memcpy()* para copiar los 16 bytes de datos recibidos a decrypted. Tras esto, y haciendo uso de la función *descifrarAES* (previamente explicada) se procede a descifrar los datos que se encuentran en “decrypted” y se sobrescriben en este mismo arreglo. A continuación, ocurre una de las acciones más importante de todo el proyecto, que es la extracción e interpretación de los datos que se ha enviado por parte del emisor. Esto ocurre en la línea “*memcpy(&angulo_volante, decrypted, sizeof(float))*” de la siguiente forma. Se interpreta que los 4 primeros bytes de “*decrypted*” como un float (que es lo que ha enviado el emisor) y se copia ese float a la variable “*ángulo_volante*”.

Una vez tenemos ya en el receptor el ángulo del volante se procede a marcar la recepción de datos como correcta situando la variable “*recep_incorrecta*” a 0. Se guarda el tiempo actual a través de “millis()” en la variable *últimaRecepción(*en futuros apartados se tratará de forma más extensa millis()

y su uso). Por último, imprimimos por pantalla el valor del ángulo recibido. El diagrama de flujo de esta función es el siguiente:

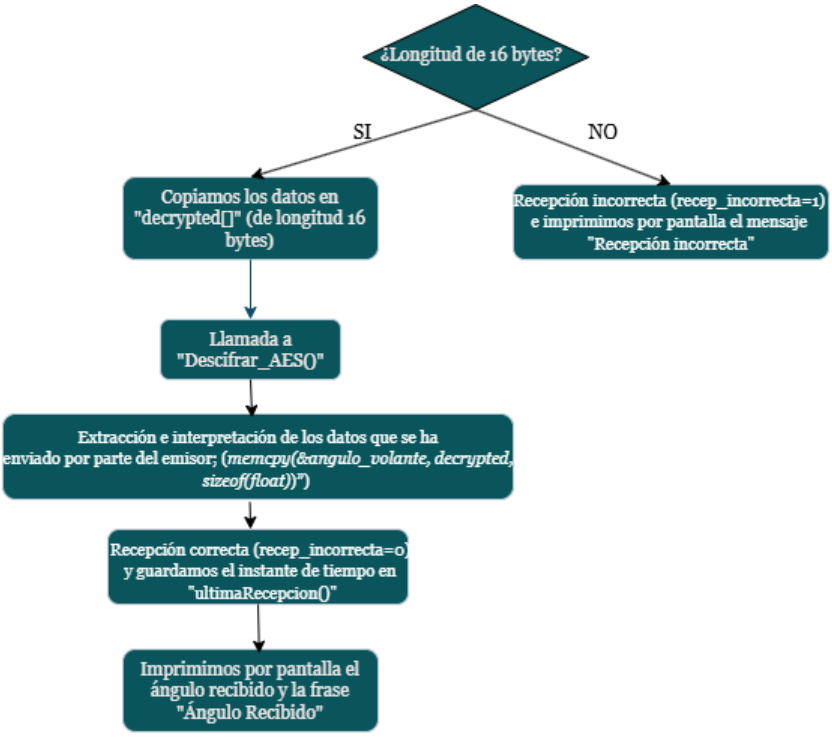


***Figura 109**: Diagrama de flujo de Recibir Datos*

5.3.3 Módulo "Config.h"

En este archivo se declaran los diferentes pines que se usarán del ESP receptor así como algunas variables de gran importancia.

```
#pragma once
#include <Arduino.h>
```

***Figura 110**: Librerías de Config.h*

Las librerías usadas son:

"*#include <Arduino.h>*": Se usa para acceder a las funciones de la biblioteca principal de Arduino (como *millis*(), *digitalWrite*()…).

También se usa *"#pragma once"*, cuyo uso ya ha sido explicado.

Con esto, definimos las siguientes variables

```
4   // Pines motores
5   #define M1_DIR_PIN 23
6   #define M1_PWM_PIN 21
7
8   #define M2_DIR_PIN 18
9   #define M2_PWM_PIN 5
10
11  // PWM
12  const int frequency = 5000;
13  const ledc_timer_bit_t resolution = LEDC_TIMER_8_BIT;
14  const ledc_timer_t timer_m1 = LEDC_TIMER_1;
15  const ledc_timer_t timer_m2 = LEDC_TIMER_0;
16
17  const ledc_channel_t PWM_Channel_M1 = LEDC_CHANNEL_1;
18  const ledc_channel_t PWM_Channel_M2 = LEDC_CHANNEL_0;
19
20  // Variables globales
21  extern volatile int recep_incorrecta;
22  extern unsigned long ultimaRecepcion;
23  extern float angulo_volante;
24  extern float angulo_anterior;
25  extern int dutyCycleM1;
26  extern int dutyCycleM2;
27  extern bool motorRotativoActivo;
```

***Figura 111**: Código en config.h*

Primero se definen los pines a loa que van conectados los cables relativos al control de tanto la dirección como el PWM de cada uno de los motores ("*#define* M1_DIR_PIN 23"...).

Con los pines de los motores ya definimos, pasamos a definir algunas variables y constantes que nos son útiles. La primera variable que de define es la frecuencia ("frequency") con un valor de 5000 hercios. Esta variable nos indica la cantidad de veces que se produce por segundo el ciclo ON/OFF de un PWM, en este caso 5000 veces por segundo. Se ha elegido este valor buscando un funcionamiento suave y silencioso debido a que, a frecuencias bajas, los motores DC pueden producir un "zumbido" que puede hacerse muy molesto

Lo siguiente que se hace es definir la **resolución** del propio PWM. Esto nos indica la cantidad de niveles puede tener un ciclo completo de la señal PWM. Esto nos influye directamente en el concepto de "ciclo de servicio" ("Duty Cycle"). De esta forma, si la resolución fuese de 1 bit solo dispondríamos de 2 niveles (1: encendido al 100%, 0: apagado). En caso de que aumentemos la resolución podremos obtener distintos niveles. En nuestro caso se ha decidido

que la resolución sea de 8 bits, por lo que tendremos 256 estados ("ledc_timer_bit_t resolution = LEDC_TIMER_8_BIT"). El ciclo de servicio seguirá la siguiente fórmula:

$$Duty\ cycle(\%) = \frac{(valor\ entre\ 0\ y\ 255)}{255} * 100 \quad [15]$$

Este porcentaje nos indicará el tiempo en un ciclo que el motor está encendido respecto al tiempo total del periodo.

Tras esto, configuraremos los temporizadores hardware que se encargan de controlar el PWM. Los temporizadores son los encargados de controlar la resolución y la frecuencia de los PWM (estos 2 parámetros se acaban de explicar). Se usa un temporizador para cada motor, de forma que uno no interfiera con el otro, evitando así errores en las señales PWM. ESP permite el uso de hasta 4 temporizadores para generar señales PWM pudiéndose usar un mismo temporizador para varios motores. Sin embargo, en nuestro caso, habiendo temporizadores de sobra se ha decidido usar un temporizador por motor. Teniendo ya definidos los temporizadores, hemos de configurar los canales a través de los que se controlará la salida de la señal PWM. Para cada motor se asigna un canal ("*PWM_Channel_M2 = LEDC_CHANNEL_0*"; "*PWM_Channel_M1 = LEDC_CHANNEL_1*"). Usando canales separados somos capaces de controlar cada motor con una señal PWM diferente.

Por último, se definen una serie de variables globales que se utilizarán a lo largo del código principal.

5.3.4 Módulo funcional de control del motor lineal (MotorLineal.cpp y MotorLineal.h)

5.3.4.1 MotorLineal.h

```
1   #pragma once
2   void motorLinealAdelante();
3   void motorLinealAtras();
4   void motorLinealSetVelocidad(int pwm);
5   void inicializarMotores();
```

***Figura 112**: Librería en MotorLineal.h*

Usa #pragma once, al igual que los códigos anteriores y en este archivo declara las siguientes funciones para que otros puedan hacer uso de ellas.

- void motorLinealAdelante();
- void motorlinealAtrás();
- void motorLinealSetVelocidad(int pwm);
- void inicializarMotores();

5.3.4.2 MotorLineal.cpp

Las librerías que se usan en este archivo on "config.h", "MotorLineal.h" y "driver/ledc.h".

```
1   #include "Config.h"
2   #include "MotorLineal.h"
3   #include "driver/ledc.h"
```

***Figura 113**: Librería en MotorLineal.cpp*

#include "driver/ledc.h": Se trata de la librería que nos da acceso al controlador LEDC (LED control) del ESP32. Este hardware es el encargado de generar las señales PWM encargadas del control de los motores. A través de esta librería podemos configurar una señal PWM por cualquiera de los 16 canales distintos que permiten el uso de esta herramienta. Para el control de la velocidad de tanto el motor lineal como el rotativo haremos uso de señales PWM.

Tras esto, comenzamos con la declaración de funciones relativas al motor lineal:

La función ***motorLinealAdelante*** pone los pines de dirección del motor lineal en HIGH para indicar este sentido

```
7   void motorLinealAdelante() {
8     digitalWrite(M2_DIR_PIN, HIGH);
9   }
```

***Figura 114**: Función motorLinealAdelante*

La función ***motorLinealAtras*** pone los pines de dirección del motor lineal en LOW para indicar este sentido

```
11  void motorLinealAtras() {
12    digitalWrite(M2_DIR_PIN, LOW);
13  }
```

***Figura 115**: Función motorLinealAtras*

La función ***motorLinealSetVelocidad***, que requiere que pongamos el valor del ciclo de servicio que deseamos y lo almacenará en la varia *PWM*

```
15  void motorLinealSetVelocidad(int pwm) {
16    ledc_set_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M2, pwm);
17    ledc_update_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M2);
18  }
```

***Figura 116**: Función motorLinealSetVelocidad*

La primera línea ("*ledc_set_duty (LEDC_HIGH_SPEED_MODE, PWM_Channel_M2, pwm)*") configura el PWM del canal especificado. LEDC_HIGH_SPEED_MOD nos indica que se usa el modo de alta velocidad, PWM_Channel_M2 el canal por el que se transmite el PWM y por último se sitúa el valor del ciclo de servicio.

Por último, a través de la función "**inicializarMotores**" preparamos al motor lineal para poder controlar el giro y la velocidad. Comenzamos declarando los pines de la dirección y el PWM del motor como pines de salida

```
20  void inicializarMotores() {
21    pinMode(M2_DIR_PIN, OUTPUT);
22    pinMode(M2_PWM_PIN, OUTPUT);
```

***Figura 117.1**: Función inicializarMotores()*

Tras esto, comenzamos con la configuración del PWM a través de la estructura "ledc_timer_config_t" usada para poder realizar de la configuración del temporizador del PWM para el motor rotativo (Motor 2). A continuación, explicamos los distintos elementos que conforman esta estructura.

```
24  ledc_timer_config_t timerConfig = {
25    .speed_mode = LEDC_HIGH_SPEED_MODE,
26    .duty_resolution = resolution,
27    .timer_num = timer_m2,
28    .freq_hz = frequency,
29    .clk_cfg = LEDC_AUTO_CLK
30  };
31  ledc_timer_config(&timerConfig);
```

***Figura 117.2**: Función inicializarMotores()*

Dentro de la estructura, comenzamos estableciendo el modo de operación del temporizador en el modo de alta velocidad (dentro de la estructura, el "*speed_mode*") de forma que obtenemos una mayor precisión ("*speed_mode=LEDC_HIGH_SPEED_MODE*"). *LEDC_HIGH_SPEED_MODE* usa un reloj más rápido y permite tener un mayor control de precisión en la señal PWM frente a *LEDC_LOW_SPEED_MODE.* Tras esto, hemos de definir la resolución del PWM. Esta resolución nos da los diferentes niveles de "Duty_cycle" que nos permitirá variar la velocidad en base al ciclo de servicio. En este caso, se ha definido la resolución en 8 bits (entre 0 y 255). La siguiente parte de la estructura del temporizador del PWM es a través de la cual podemos indicar cual de los 4 temporizadores que proporciona el ESP se usará para el ESP32. La penúltima parte de la estructura es a través de la cual definimos la frecuencia del PWM. De forma final, se elige la mejor fuente de reloj para la

frecuencia y resolución elegida (esta elección lo realiza el ESP 32 de forma automática).

El siguiente paso, es configurar el PWM que usará el temporizador anteriormente definido.

```
33      ledc_channel_config_t channelConfig = {
34        .gpio_num = M2_PWM_PIN,
35        .speed_mode = LEDC_HIGH_SPEED_MODE,
36        .channel = PWM_Channel_M2,
37        .intr_type = LEDC_INTR_DISABLE,
38        .timer_sel = timer_m2,
39        .duty = 0,
40        .hpoint = 0
41      };
42      ledc_channel_config(&channelConfig);
43    }
```

***Figura 117.3**: Función inicializarMotores()*

Aquí se configura el canal PWM que va a controlar el motor rotativo. En esta parte, se asignan los siguientes elementos:

"*gpio_num =M2_PWM_PIN*": Pin físico del PWM al que se conecta el motor

"Speed_mode": Debe de ser el mismo que el del temporizador

"*Channel=PWM_channel_M2*": Se define el camal PWM que controla el PIN

"timer_sel=timer_m2": Se indica el temporizador que se va a usar en este PWM (se ha definido anteriormente).

"*Duty_cycle*": Se define el ciclo de servicio para el control de la velocidad del motor

"Hpoint": Punto de activación del ciclo de servicio.

Tras esto, con el temporizador y el PWM configurado, podemos inicializar el canal y con ello poder transmitir el PWM:

5.3.5 Módulo funcional de control del motor rotativo (MotorRotativo.cpp y MotorRotativo.h)

5.3.5.1 MotorRotativo.h

En este caso, solo se implementa #pragma once, y el objetivo de este código (al igual que de los demás .h es que otros archivos hagan uso de estas funciones).

```
1   #pragma once
2   void iniciarMotorRotativo();
3   void frenarMotorRotativo();
4   void inicializarPWM();
5   void inicializarMotores();
```

***Figura 118**: Librerías de MotorRotativo.h*

5.3.5.1 MotorRotativo.cpp

Usa las librerías "Config.h", "MotorRotativo.h" y "driver/ledc.h" (ya explicadas anteriormente) El código comienza con la definición de una variable booleana y con el valor del PWM para el motor rotativo.

```
1   #include "Config.h"
2   #include "MotorRotativo.h"
3   #include "driver/ledc.h"
4
5   bool motorRotativoActivo = false;
6   int dutyCycleM1 = 244;
```

***Figura 119**: Librerías de MotorRotativo.cpp*

Tras esto, se comienza con la implementación de diferentes funciones.

La función ***iniciarMotorRotativo*** es la encargada de, como su propio nombre indica, comenzar con el movimiento del motor rotativo.

```
8   void iniciarMotorRotativo() {
9     if (motorRotativoActivo) return;
10
11    digitalWrite(M1_DIR_PIN, HIGH);
12    for (int duty = 0; duty <= dutyCycleM1; duty += 5) {
13      ledc_set_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1, duty);
14      ledc_update_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1);
15      delay(15);
16    }
17
18    motorRotativoActivo = true;
19  }
```

***Figura 120**: Función iniciaMotorRotativo*

Esto lo hace modificando los pines tanto de dirección como del PWM correspondiente al motor rotativo. Se pone en HIGH el pin de dirección, indicando al motor rotativo que gire en esa dirección y se inicia el movimiento con un ciclo de servicio de 200.

La siguiente función en aparecer es **frenarMotorRotativo().**

```
21  void frenarMotorRotativo() {
22    if (!motorRotativoActivo) return;
23
24    int actualDuty = ledc_get_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1);
25    for (int duty = actualDuty; duty >= 0; duty -= 5) {
26      ledc_set_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1, duty);
27      ledc_update_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1);
28      delay(15);
29    }
30
31    motorRotativoActivo = false;
32  }
```

***Figura 121**: Función FrenarMotorRotativo*

Primero se comprueba que el motor rotativo está activo mediante el uso de la variable booleana previamente definida (“*if (!motorRotativoActivo) return*”). Tras esto se obtiene el valor del PWM actual mediante la línea de código “*actualDuty = ledc_get_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1)*”. Tras esto, y mediante el uso de un *for* se va reduciendo de forma progresiva el PWM hasta llegar a 0 y que el motor se pare. Finalmente se marca el estado del motor como inactivo.

Por último nos encontramos con la función “**inicializarPWM()** cuya estructura es exactamente la misma que la de “**inicializarMotores**” del motor lineal, cambiando los parámetros propios de cada motor. Se ha decidido llamarlas así de forma que se puedan distinguir claramente.

```
34  void inicializarPWM() {
35    pinMode(M1_DIR_PIN, OUTPUT);
36    pinMode(M1_PWM_PIN, OUTPUT);
37
38    ledc_timer_config_t timerConfig = {
39      .speed_mode = LEDC_HIGH_SPEED_MODE,
40      .duty_resolution = resolution,
41      .timer_num = timer_m1,
42      .freq_hz = frequency,
43      .clk_cfg = LEDC_AUTO_CLK
44    };
45    ledc_timer_config(&timerConfig);
46
47    ledc_channel_config_t channelConfig = {
48      .gpio_num = M1_PWM_PIN,
49      .speed_mode = LEDC_HIGH_SPEED_MODE,
50      .channel = PWM_Channel_M1,
51      .intr_type = LEDC_INTR_DISABLE,
52      .timer_sel = timer_m1,
53      .duty = 0,
54      .hpoint = 0
55    };
56    ledc_channel_config(&channelConfig);
57  }
```

***Figura 122**: Función InicializarPWM*

5.3.6 ControlMotores.ino

En este apartado nos centraremos en comentar el SetUp() del código principal debido a que se ha considerado que el Loop() es mejor explicarlo junto a las funcionalidades del equipo.

En este caso, las librerías que se incluyen son bastantes más que en ningún otro código comentado. Esto es debido a que este archivo ha se usar todas las funciones declaradas en los módulos auxiliares.

```
#include <WiFi.h>
#include <esp_now.h>
#include "driver/ledc.h"

#include "Config.h"
#include "MotorRotativo.h"
#include "MotorLineal.h"
#include "Comunicacion.h"
```

***Figura 123**: Variable declaradas en ControlMotores.ino*

Con todas las librerías ya declaradas, podemos observar cómo se declara una variable de tipo const unsigned long llamada *TIMEOUT_RECEPTION.* Esta variable representa en milisegundos el tiempo máximo que el receptor puede estar sin recibir un dato antes de que la recepción pase a ser incorrecta.

5.3.6.1 SetUp ()

El *SetUp*() comienza estableciendo la comunicación serial en 115200 baudios. Tras esto se llama a las funciones *inicializarPWM*() (usada para poder controlar la dirección y el PWM del motor rotativo), *inicializarMotores* (tiene la misma función que InicializarPWM pero para el motor lineal) y por último se llama a iniciarMotorRotativo (para que el motor comience a girar),

```
void setup() {
  Serial.begin(115200);

  inicializarPWM();
  inicializarMotores();
  iniciarMotorRotativo();

  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error al inicializar ESP-NOW");
    return;
  }

  esp_now_register_recv_cb(RecibirDatos);
}
```

***Figura 124**: SetUp() en ControlMotores.ino*

Tras esto, se configura el ESP como modo estación (“STA”),de esta forma el ESP no actúa como un punto de acceso (no crea su propia red Wi-Fi) y busca conectarse a una red.

El bloque inicializa el bloque ESP-NOW y en caso de que no sea posible iniciar el protocolo se imprime un mensaje de error. Por último, se registra una función de callback que se ejecuta de forma automática cada vez que se reciben datos.

Capítulo 6:

Comportamiento y funcionalidades del sistema

Capítulo 6: Comportamiento y funcionalidades del sistema

El objetivo de este capítulo es comentar el funcionamiento del sistema así como las diferentes funcionalidades que se han desarrollado

6.1 Funcionamiento del sistema. *"Loop()"* en el receptor

Todo el control del sistema se realiza en el "*loop()*" del receptor. Dentro de este se han desarrollado diferentes acciones a realizar en base a distintas situaciones que se han considerado potencialmente peligrosas.

Recepción incorrecta

Una de las partes más sensibles de todo este proyecto es la transmisión de información. Al ser una transmisión inalámbrica es posible que surjan diferentes situaciones que pongan en riesgo el funcionamiento completo del sistema. Por ello, se han diseñado una serie de cortafuegos tratando de evitar las situaciones de peligro. Para ello se ha usado la variable "*recep_incorrecta*" en la cual reflejaremos el estado del protocolo de comunicación. Si esta variable pasa a 1 será porque se ha dado alguna de las siguientes situaciones:

- Tiempo de recepción elevado: Existe la posibilidad de que no se reciban datos debido a interferencias o a un fallo en el cableado del emisor (como una desconexión por un golpe). Por ello, en caso de que no se reciba información en un periodo determinado, la variable de recepción pasará a uno.

```
if (millis() - ultimaRecepcion > TIMEOUT_RECEPCION) {
  recep_incorrecta = 1;
  Serial.println("Error: NO se reciben datos");
}
```

***Figura 125**: Código de detección de tiempo muerto*

"*Millis()*" nos da el tiempo que lleva el ESP ejecutando un código y La variable "*ultima_recepción*" almacena el último instante de tiempo. Por ello, haciendo la diferencia entre dos valores obtenemos el intervalo de tiempo entre las recepciones. En caso de que este dato sea mayor a una constante definida anteriormente llamada "*TIMEOUT_RECEPTION*" que tiene un valor de 100 ms la variable "*recep_incorrecta*" pasará a uno. Seguidamente, realizamos una comprobación del estado completo del sistema para saber si hemos de arrancar el sistema o si se ha de frenar. Para ello, volvemos a hacer uso de la variable "recep_incorrecta" y de "motorRotativoActivo". En caso de que la recepción sea incorrecta y el motor rotativo esté activo, frenamos el sistema. En caso de que la recepción sea correcta pero el sistema se encuentre parado, arrancamos el motor rotativo (figura[126])

```
if (recep_incorrecta == 1 && motorRotativoActivo) {
  frenarMotorRotativo();
}

if (recep_incorrecta == 0 && !motorRotativoActivo) {
  iniciarMotorRotativo();
}
```

***Figura 126**: Código de arranque y parada del motor rotativo*

- Datos recibidos de longitud incorrecta: En caso de que los datos que se reciban no sean de la longitud adecuada, se pondrá la variable "*recep_incorrecta*" a 1 y se imprimirá por pantalla "*Error: Datos recibidos con LONGITUD INCORRECTA*". Este proceso lo lleva a cabo la función recibir datos del archivo comunicación cpp

Establecimiento de la velocidad

La velocidad de los motores se controla a través de PWM codificado en 8 bits de forma que 255 equivaldría a una velocidad máxima. La velocidad del motor rotativo es constante (salvo en los momentos de arranque y frenado). A diferencia de este, se ha decidido dar un PWM al motor lineal que pueda variar en base a la situación que se dé.

Se cree necesario recordar la geometría del sistema (figura [127]) para poder explicar de forma adecuada la elección de la velocidad adecuada.

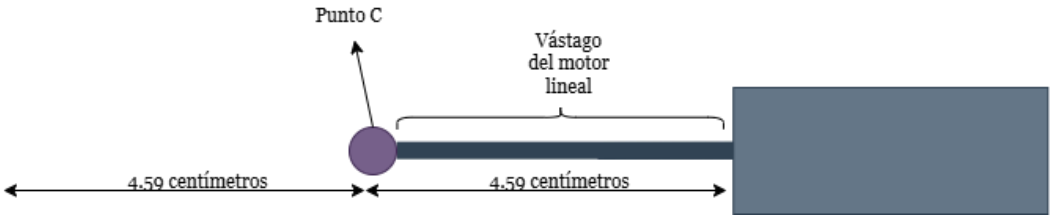


***Figura 127**: Imagen movimiento del Punto A*

La posición inicial del sistema es a 4.59 centímetros de la posición del motor en la que el vástago está completamente recogido. Por ello, para alcanzar un ángulo de 50 grados (ya sea para un giro hacia la derecha o hacia la izquierda) el vástago se ha de mover 4.59 centímetros.

El tiempo de girar un volante de forma completa (recorrer los 540º) es de 2.5 segundos en una situación normal. Es decir, nuestro sistema ha de ser capaz de recorrer 4.59 centímetros en 2.5 segundos.

$$Velocidad = \frac{espacio}{tiempo} = \frac{4.59\ centímetros}{2.5\ segundos} = 1.836 cm/s \quad [16]$$

Nuestra velocidad máxima es de 6 cm/s equivalente a un PWM (codificado en 8 bits) de 255.

$$\frac{6\ cm/s}{1.836\ cm/s} = \frac{255}{X} \rightarrow X = 78 \quad [17]$$

El PWM en una situación normal debería de ser de 78, pero se ha incrementado a 100 de forma que tengamos una respuesta más rápida todavía. Sin embargo, el poder controlar la velocidad del motor lineal nos da la posibilidad de diferenciar situaciones de emergencia.

En caso de que se detecte un cambio de dirección muy brusco se entiende que el conductor ha detectado un problema en la vía necesita una respuesta rápida por parte de la dirección para esquivar una potencial situación de riesgo. En ese caso, podremos aumentar la velocidad de los motores de forma que el ángulo en las ruedas aumente de una forma más rápida. Por ello, en caso de que en 100 milisegundos (una pasada al código del receptor) se detecte que el cambio en el volante es de más de 65º la velocidad del motor lineal aumentará hasta un PWM de 180 (una velocidad de 4.2 cm/s). En este caso, en poco más de un segundo se podría alcanzar un ángulo en las ruedas de 50º

```
if (abs(angulo_volante - angulo_anterior) < 65) {
dutyCycleM2=120;
}
else{
    dutyCycleM2=180;
}
```

***Figura 128**: Variación de la velocidad del motor lineal*

Control del motor lineal

Para poder controlar el motor lineal, lo primero que hemos de comprobar es que el proceso de comunicación se está realizando de forma correcta. Pare ello comprobamos la variable "*recep_incorrecta*" (figura[129]).

```
if (recep_incorrecta == 0) {
  if (angulo_volante > angulo_anterior + umbral) {
    motorLinealAdelante();
    motorLinealSetVelocidad(dutyCycleM2);S
  } else if (angulo_volante < angulo_anterior - umbral) {
    motorLinealAtras();
    motorLinealSetVelocidad(dutyCycleM2);
  } else {
    motorLinealSetVelocidad(0);
  }
}

angulo_anterior = angulo_volante;
delay(100);
```

***Figura 129**: Control motor lineal en base al ángulo del volante*

Una vez comprobamos la variable de la recepción, comenzamos con la lectura del ángulo del volante. La posición del volante en un determinado momento se almacena en la variable "*angulo_volante*". Una vez se acaba el barrido del programa, se guarda ese valor en la variable "*angulo_volante*" que necesitará ser usado.

En caso de que "*angulo_volante*" sea mayor de "*angulo_anterior*" el motor lineal avanzará y damos una velocidad de "*dutyCycleM2*". Sin embargo, en la condición "*if*" aparece una constante llamada "*umbral*" que no hemos tratado. Esto se debe a que nuestro encoder (el encargado de registral el ángulo) es muy sensible, por lo que hasta posibles vibraciones pueden causar variaciones en el ángulo. Por ello, usando una variable umbral vamos a ser capaces de inhibir estas variaciones de ángulo. Esta variable umbral se ha definido en 0.5º de forma que no afecte de forma significativa al funcionamiento global

6.2 Diagrama de funcionamiento

Al igual que en funciones anteriores, se muestra el diagrama de funcionamiento del código que se ejecuta en el receptor. Primero se mostrará la lógica de la variable "*recep_incorrecta*" y luego la del receptor global

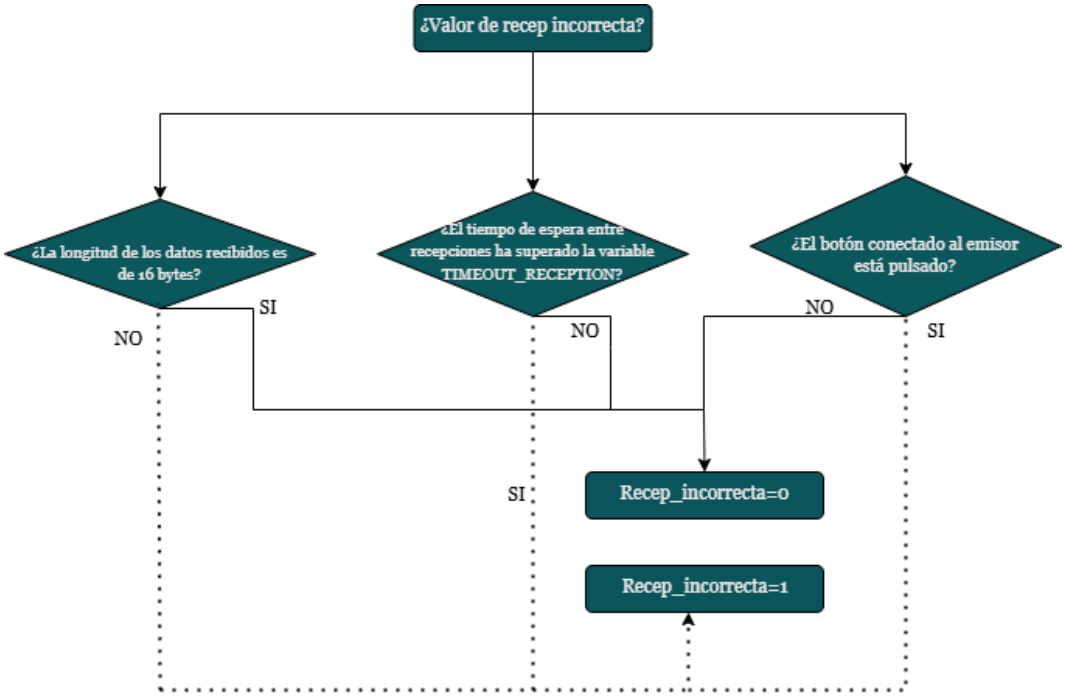


***Figura 130**: Diagrama de flujo de la variable "recep_incorrecta"*

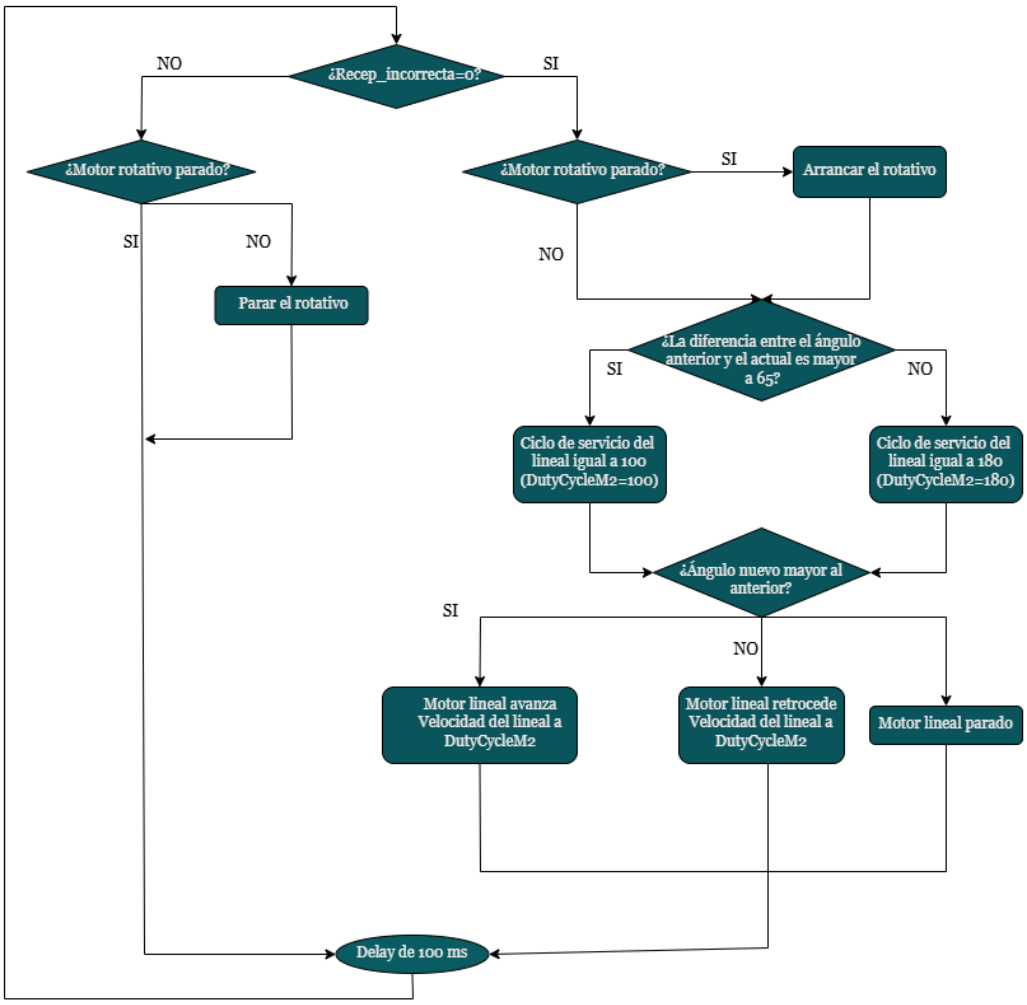


***Figura 131**: Diagrama de flujo del receptor*

Capítulo 7:

Montaje del sistema y pruebas de funcionamiento

Capítulo 7: Montaje del sistema y pruebas de funcionamiento

El objetivo de este capítulo es mostrar el proceso de ensamblado y montaje del sistema, así como el resultado final obtenido y las pruebas realizadas para comprobar el funcionamiento

7.1 Proceso de ensamblaje y pruebas

Para la realización completa del sistema, de lo primero que se necesita es una superficie sobre la que trabajar. En este caso se decidió que una tabla de un grosor de aproximadamente un centímetro con 2 milímetros era la mejor opción. Esto era debido a la facilidad de pintar sobre ella para realizar bocetos, así como la facilidad de taladrado. El proceso que se siguió fue el siguientes. Se comenzó realizando un taladro cerca del lateral de la tabla, correspondiente al "*Punto B*". Este es el punto que debe de permitir la rotación sobre si mismo pero que no se mueve en ningún sentido, ya sea lateral o vertical.

Con el taladro ya realizado se dibujó sobre madera tratando de tener la mayor precisión posible el esquema de funcionamiento, de forma que encontremos los puntos "A" y "C".

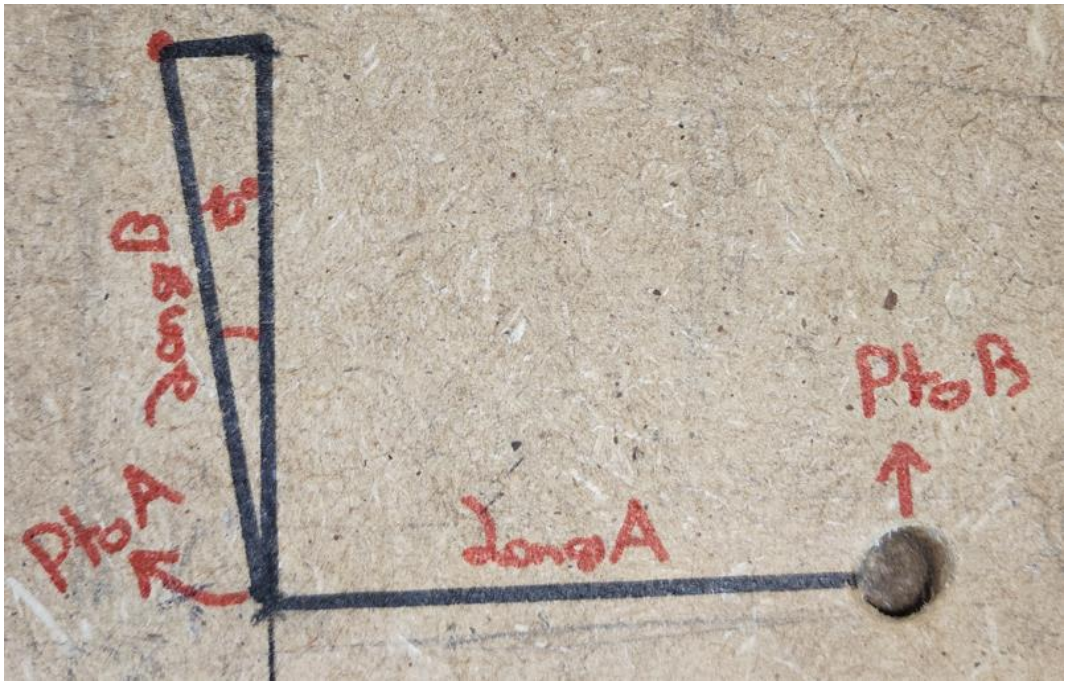


***Figura 132**: Puntos perforados en la base del prototipo*

Con estas medidas ya tomadas, lo siguiente que se hizo fue buscar la posición adecuada del motor lineal. Para fijar el motor, hemos de tener en cuenta que en la posición inicial (ángulo de las ruedas a 0º) el vástago ha de estar 4,59 centímetros avanzado. Teniendo esto en cuenta, se realizaron unas marcas donde se situaría las distintas sujeciones.

***Figura 133**: Puntos de la sujeción del motor lineal*

Con el motor ya fijo, se añadieron el resto de los elementos (a excepción de la caja del receptor) para comprobar si el sistema funcionaba de forma correcta. A pesar de la cantidad de cables y el desorden que supone una primera prueba, esta fue todo un éxito cumpliéndose todo el comportamiento esperado a la perfección. Sin embargo la apariencia del sistema no es la más adecuada principalmente por la cantidad de cables usados (que en algún punto pueden producir un corto circuito) (figura[134])

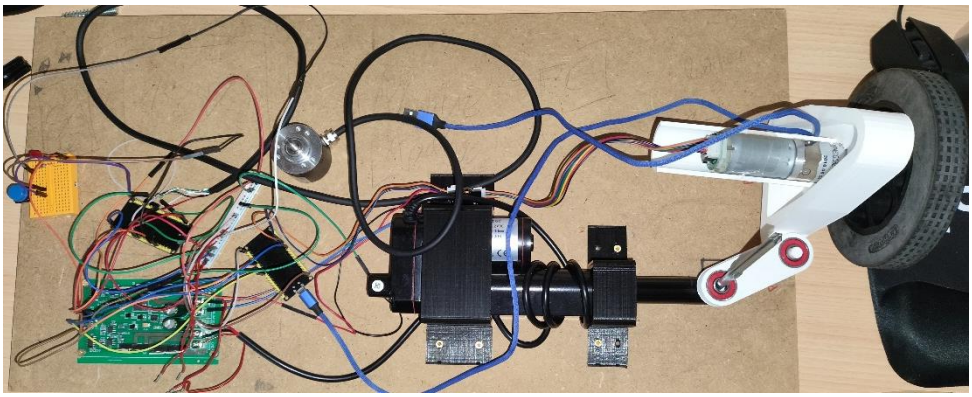


***Figura 134**: Primer montaje del sistema*

En esta primera prueba lo que se comprobó es el correcto funcionamiento del sistema en todos sus casos. Se realizó una prueba para cada funcionalidad, siguiendo una "*checklist*" de forma que se comprobaban una a una todas las funcionalidades. Dicha lista se muestra a continuación.

Arranque y parada	✓
Protocolo de comunicación	✓
Prueba fallo en comunicación	✓
Prueba fallo en alimentación	✓
Prueba de giro	✓
Prueba cambio de velocidad	✓

***Tabla 12**: CheckList de las primeras pruebas*

Una vez se comprobó que todo el sistema era funcional y correcto se añadió la caja para tener más orden en todo el sistema. Se obtuvo un sistema como el mostrado en la *figura [136]*.

Con el sistema ya montado, probamos de nuevo a hacer pruebas de giro, obteniendo los siguientes resultados:

Giro horario

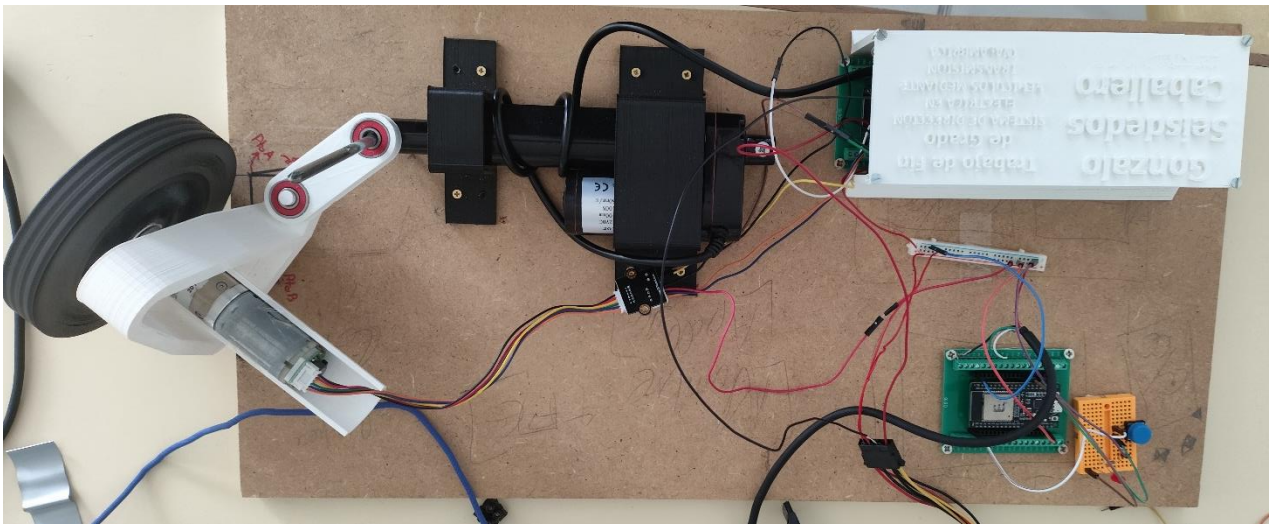


***Figura 135**: Prototipo con ángulo de giro horario*

Giro antihorario

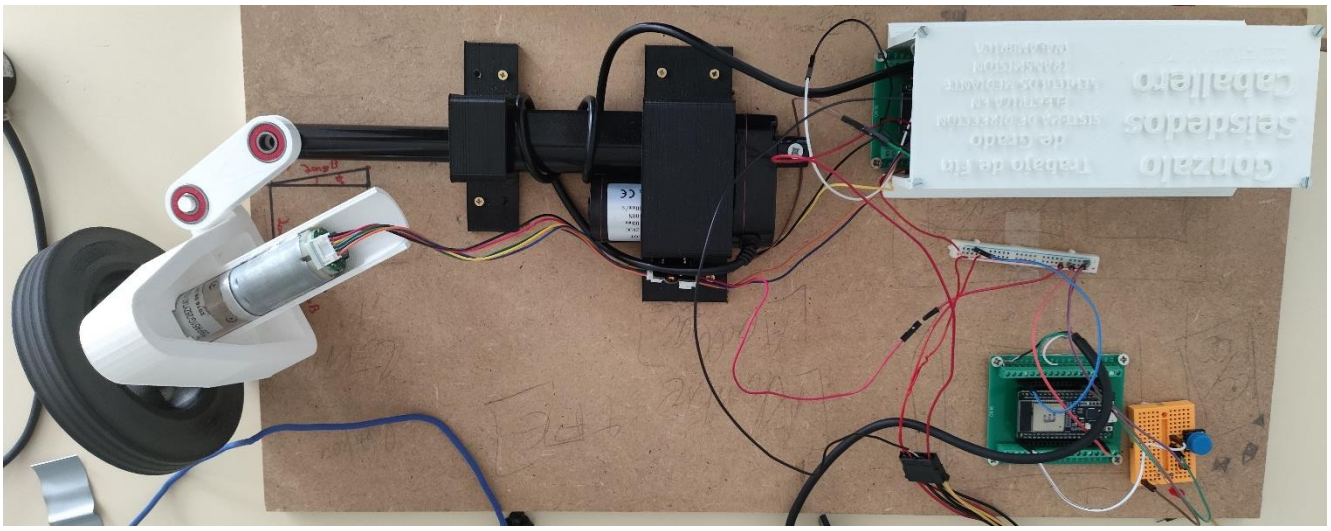


***Figura 136**: Prototipo con ángulo de giro antihorario*

7.2 Funcionamiento conectado a PC

Hemos comprobado todo el proyecto de forma empírica, pero, Arduino IDE nos permite lanzar los códigos con los microcontroladores conectados al PC de forma que podamos ver aquellos mensajes que hemos decidido imprimir por pantalla. De igual forma, y para comprobar los códigos se volvieron a realizar las mismas pruebas que antes, pero con el PC conectado.

Pulsando el botón

En caso de que se haya pulsado el botón, el ESP emisor deberá enviar un mensaje diciendo "Envío detenido", el motor rotativo pararse y el lineal no seguir órdenes del encoder.

```
Envio exitoso
Ángulo enviado: 86.76
Envio exitoso
Ángulo enviado: 86.76
Envio exitoso
Ángulo enviado: 86.76
Envio exitoso
Envio DETENIDO
```

***Figura 137**: Serial monitor del emisor con botón pulsado*

Por su parte, el receptor mostrará el ángulo que ha recibido.

```
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
```

***Figura 138**: Serial monitor del receptor con botón pulsado*

Desconexión de alimentación del ESP Emisor

En este caso lo que haremos será quitar el pin de alimentación del ESP emisor de forma que no sea capaz de enviar ningún tipo de dato. Como hemos dejado de alimentar, no podemos leer lo que el emisor imprime por pantalla, pero sí lo que hace el receptor. En este caso, como el emisor no envía nada el receptor está sin recibir datos durante un tiempo superior a la variable ya comentada "*TIMEOUT_RECEPTION*" por lo que entiende que hay un fallo en el protocolo de comunicación y para el motor rotativo.

```
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
Error: NO se reciben datos
```

***Figura 139**: Serial monitor del emisor, con desconexión*

Funcionamiento normal

Sin crear ningún fallo y con el botón no pulsado, el funcionamiento es el siguientes:

Podemos ver como el ángulo que se envía es el mismo que el que recibe el receptor y de forma visual se comprueba que los motores avanzan.

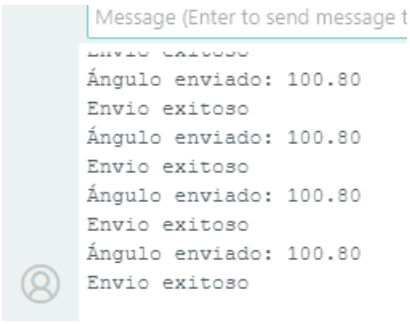


***Figura 140**: Serial monitor del emisor, con comportamiento normal*

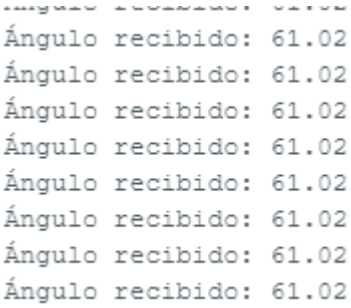


***Figura 141**: Serial monitor del receptor, con comportamiento normal*

Capítulo 8:

Presupuesto, conclusiones y líneas de mejora

CAPÍTULO 8: Presupuesto, conclusiones y líneas de mejora

El objetivo de este capítulo es mostrar el presupuesto que se ha usado para la realización del proyecto, así como las conclusiones una vez finalizado y sus líneas futuras o de mejora.

8.1 Presupuesto

El presupuesto que se ha usado para este proyecto es el mostrado en la siguiente tabla:

Componente				Precio	Cantidad	TOTAL
Nombre	Número de referencia	Modelo	Fabricante			
Motor rotativo	-	85	Dfrobot	98,93 €	1	98,93 €
Puente en H	F23108636	-	Funduino	21.14 €	1	21,14 €
Motor lineal	-	-	CNMAWAY	18,29 €	1	18,29 €
Microcontrolador ESP 32	ESP32-WROOM-32D-N4	WROOM32	EXPRESSIF	4,69 €	2	9,38 €
Sujeción ESP 32	-	-	UICPAL	2 €	2	4,00 €
Encoder	38S6G5-B-G24N	1000P/R	CICK	8,89 €	1	8,89 €
Cables (30 unidades)	-	-	-	2.23 €	1	2,23 €
Protoboards	-	-	-	1.48 €	1	1,48 €
Filamento PARA IMPRESIÓN 3D	-	-	Anycubic	5 €	1	5,00 €
Pulsador		TZT 25	TZT	1,31	1	1,31
					TOTAL	120,07

***Tabla 13**: Presupuesto del equipo*

El presupuesto total de todo el equipo ha sido de 120,07 euros.

8.2 Conclusiones

El objetivo de este proyecto era tratar de implementar una dirección de un coche sin uso de la columna de dirección. Como diseñar una dirección de un coche real era prácticamente imposible para un proyecto de fin de grado, se decidió realizar una maqueta con aquellas partes que serían esenciales en la realización física de una dirección real. Con esta idea, hemos de valorar el proyecto.

El registro del volante ha sido todo un acierto. Somos capaces de registrar todos los cambios que se producen en él con una exactitud de 0.01 grados. Su única

parte negativa es la gran sensibilidad que tiene y debido a la cual ha sido necesario introducir algunos umbrales para evitar que esa gran precisión influyese en nuestro código.

El protocolo de comunicación también se ha implementado de manera exitosa, pudiendo enviar y recibir datos sin problema. De igual forma, hemos sido capaces de cifrar todos los datos enviados, ser capaces de detectar fallos en el envío y saber actuar contra estos.

La actuación en el receptor funciona de forma correcta con una gran eficiencia y con una geometría que nos permite generar hasta un ángulo de 50º en las ruedas con un tiempo de alrededor de un segundo. Actualmente los sistemas de dirección permiten un giro de hasta 45 grados, por lo que con nuestro sistema se ha aumentado en 5, permitiendo así a los coches crear ángulos de giro más cerrados.

Por ello, se puede decir que es un sistema completamente funcional, preparado contra fallos y con una gran efectividad, por lo que se espera que las direcciones en los próximos años evolucionen hacia este tipo de dirección.

Sin embargo, la investigación de estos tipos de direcciones no implica que una vaya a desaparecer, si no que seguramente haya una etapa de transición donde ambas se utilicen y se complementen.

8.3 Aplicación de nuestro sistema

A pesar de que este estudio está pensado a vehículos particulares, este prototipo sería aplicable a robots de pequeña escala que tuviesen la necesidad de orientarse en diferentes direcciones. Ejemplos de estos equipos son:

Knightscope K7: Se trata de un robot de seguridad autónoma diseñado para patrullar y realizar labores de vigilancia sin necesidad de intervención humana. Se trata de un equipo capaz de desplazarse a través de ruedas por lo que requiere un sistema de dirección para poder cambiar la orientación y rumbo.

***Figura 142**: Knightscope K7*

Clearpath Robotics – Husky UGV: Se trata de un vehículo terrestre no tripulado diseñado para aplicaciones de investigación en entornos exteriores exigentes. Como se puede apreciar en la figura[144], se trata de un sistema conformado por 2 ejes en los que se sitúan las ruedas. El equipo hace uso de un sistema de dirección similar al propuesto en este TFG para su orientación y desplazamiento.

***Figura 143**: Clearpath Robotics – Husky UGV*

Estos son solo algunos ejemplos de aplicación del diseño desarrollado.

8.4 Líneas futuras y de mejora

La principal mejora que se podría introducir a un sistema como el planteado es un control sobre el ángulo de giro y otro sobre la posición del motor lineal.

Existen motores que son capaces de registrar su posición y saber en que punto se encuentran en cada instante, permitiendo así a los diseñadores de código controlar la velocidad de los motores en base a la posición que se quiera llegar.

***Figura 144**: Motor lineal con control de posición*

Motores como el mostrado en la figura presentan además de los pines de alimentación otros de salida que informan sobre la posición del motor. Con esto se podía implementar una nueva funcionalidad del código en la que nos basásemos en esta posición para que el vástago del motor lineal avanzase o retrocediese.

De igual forma, se podría implementar un control sobre el ángulo de y a través de un control PID dar una orden de actuación a los receptores. El vástago del encoder se situaría en el "*Punto B*" y con el giro de este punto sobre sí mismo podríamos determinar el ángulo que se ha girado y en base a este crear actuaciones en los receptores.

Finalmente, al tener control directo sobre las ruedas podríamos controlar de forma directa la dirección del vehículo, podríamos ser capaces de variar el sentido del vehículo de forma directa, creando así un sistema de dirección autónoma. Para ello se podrían implementar sensores de distancia y de localización y en base a estos dar una orden de acción a los motores lineales.

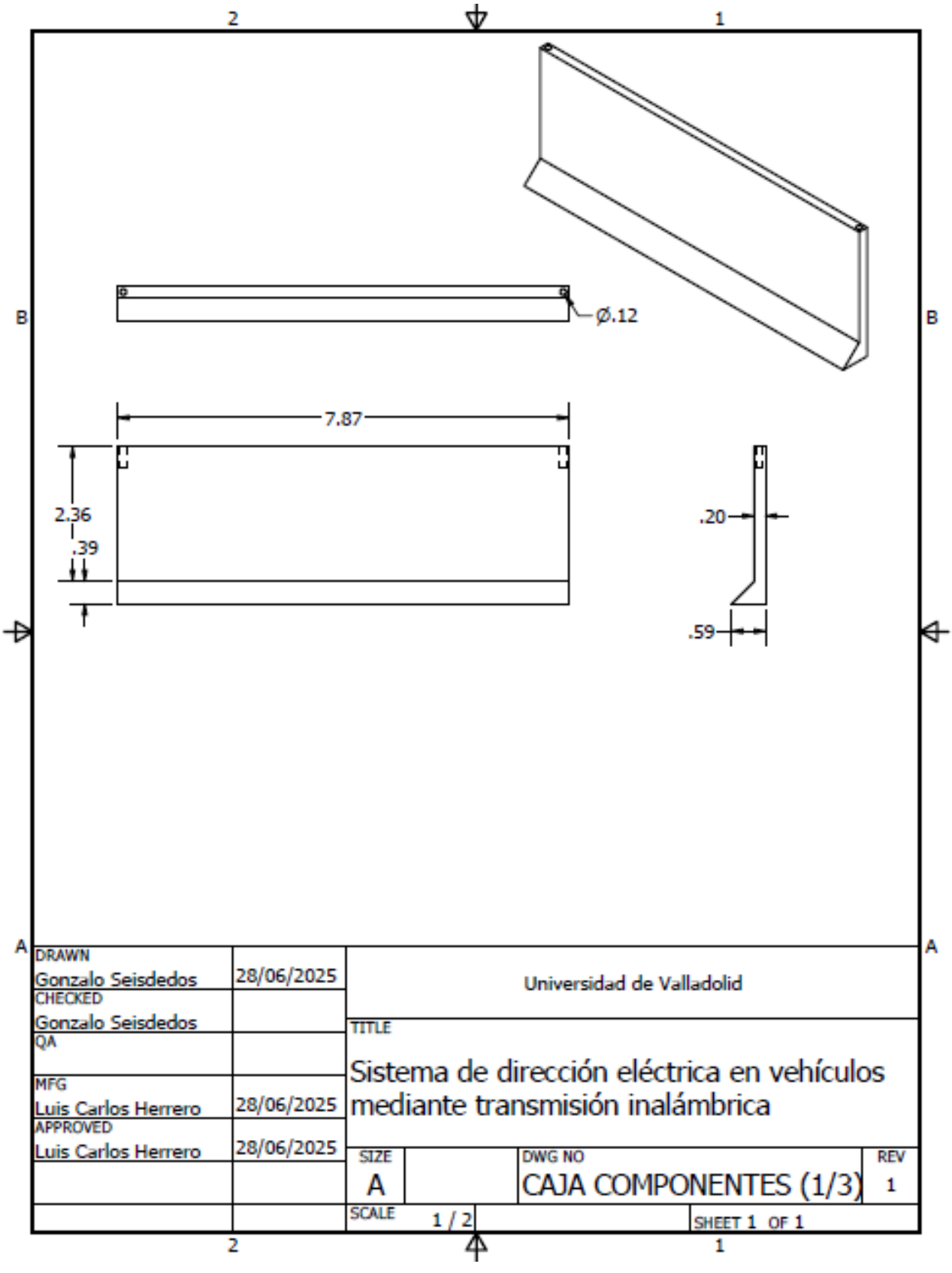
***Figura 145**: Sensor de distancia usado en automoción*

Bibliografía

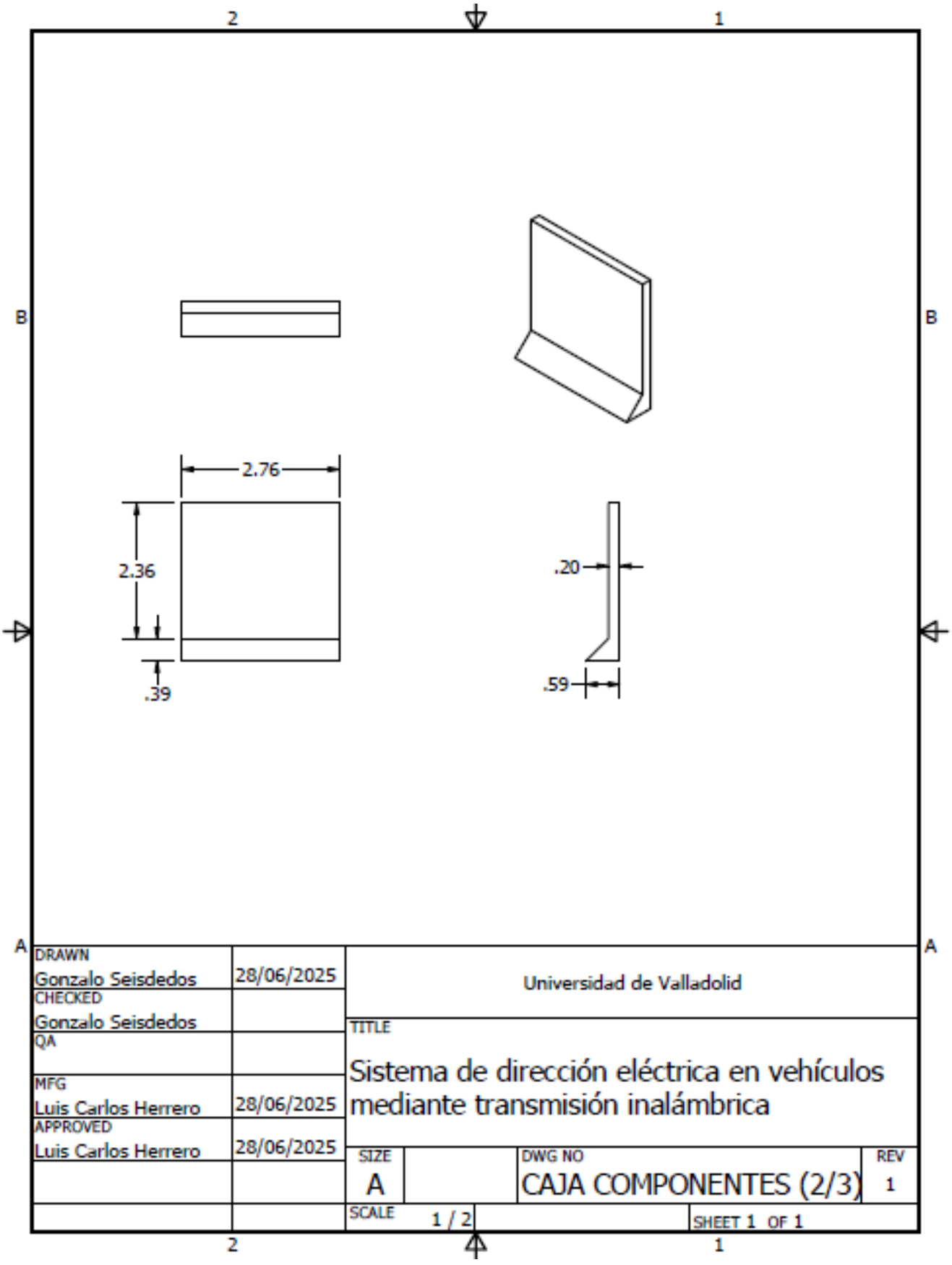
- Alabajos, Inma. «Partes del sistema de dirección en el coche y sus funciones», 29 de enero de 2024. https://www.rodesrecambios.es/blog/mecanica/sistema-de-direccion/general-sistema-de-direccion/partes-composicion/.
- Createc 3D. «Guía para usar Cura. Primeros pasos rápidos.» Createc3d (blog), 12 de mayo de 2021. https://createc3d.com/blog/usar-cura/.
- «¿Cuál es la diferencia entre RS-485 y UART? - Conocimiento». Accedido 23 de marzo de 2025. https://www.matching-ic.net/info/what-is-the-difference-between-rs-485-and-uart-98850683.html.
- «ESP-NOW Wireless Communication Protocol | Espressif Systems». Accedido 30 de marzo de 2025. https://www.espressif.com/en/solutions/low-power-solutions/esp-now.
- Gonzalez martinez, Roberto. SISTEMAS DE DIRECCIÓN EN VEHICULOS, s. f.
- «Help», s. f. https://help.autodesk.com/view/INVNTOR/2022/ESP/?guid=GUID-C2452393-D245-49DA-AFBC-9E67830ECEEF.
- HeTPro-Tutoriales. «Arduino Serial, ejemplos y funcionamiento». Accedido 3 de abril de 2025. https://hetpro-store.com/TUTORIALES/arduino-serial/.
- «Ingeniería mecatrónica». En Wikipedia, la enciclopedia libre, 10 de abril de 2025. https://es.wikipedia.org/w/index.php?title=Ingenier%C3%ADa_mecatr%C3%B3nica&oldid=166743943.
- International, Rohde & Schwarz. «Qué es UART». Accedido 1 de mayo de 2025. https://www.rohde-schwarz.com/es/productos/test-y-medida/essentials-test-equipment/digital-oscilloscopes/que-es-uart_254524.html.
- Llamas, Luis. «Configuración Básica de ESP-NOW con ESP32». Luis Llamas. Accedido 16 de marzo de 2025. https://www.luisllamas.es/como-usar-esp-now-en-esp32/.
- «Qué es y cómo usar AES-128 en ESP32». Luis Llamas. Accedido 10 de mayo de 2025. https://www.luisllamas.es/como-usar-aes128-en-esp32/.
- «Mecánica - Weebly Site», 5 de junio de 2025. https://direccionysuspension.weebly.com/mecanica.html.

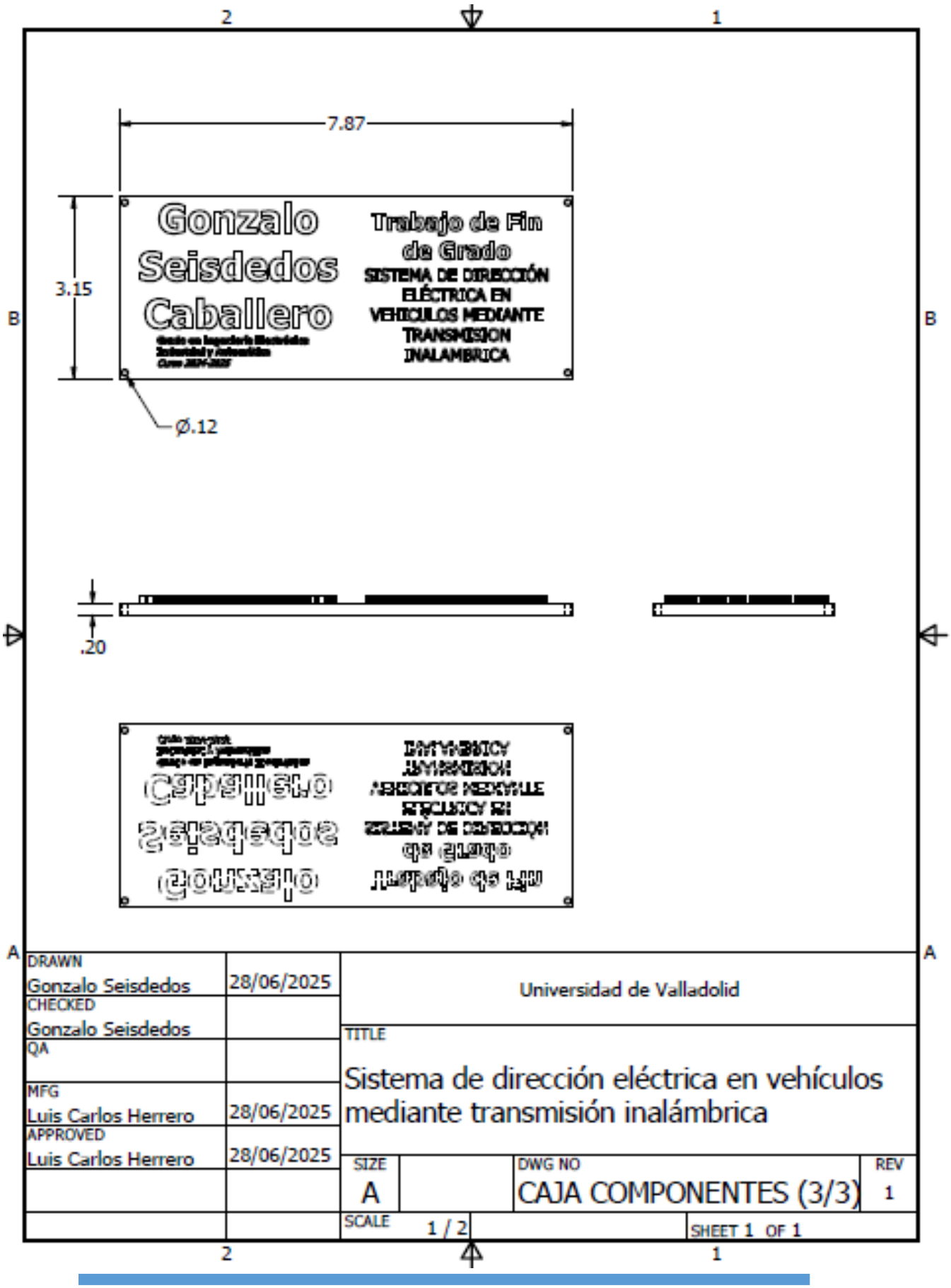
- «Protocolo P2P - GINZO TECHNOLOGIES», 19 de febrero de 2021. https://ginzo.tech/protocolo-p2p/.
- «Pruebas hidráulicas de los sistemas de dirección de los vehículos de carretera | Webtec». Accedido 20 de mayo de 2025. https://es.webtec.com/news/2022/05/04/pruebas-hidrulicas-de-los-sistemas-de-direccin-de-los-vehculos-de-carretera.
- Scribd. «Sistema de Dirección Mecánica | PDF | Direccion | Eje». Accedido 30 de junio de 2025. https://es.scribd.com/document/520881335/SISTEMA-DE-DIRECCION-MECANICA.
- Security, Panda. «¿Qué es el cifrado AES? Una guía sobre el Advanced Encryption Standard». Panda Security Mediacenter, 27 de julio de 2023. https://www.pandasecurity.com/es/mediacenter/cifrado-aes-guia/.
- Timetoast Timelines. «LINEA DE TIEMPO SISTEMA DE DIRECCIÓN EN VEHÍCULOS Timeline.» Accedido 12 de marzo de 2025. https://www.timetoast.com/timelines/linea-de-tiempo-sistema-de-direccion-en-vehiculos.
- «CIFRADO AES» Accedido 22 de mayo de 2025. https://whitestack.com/es/blog/cifrado-aes/.

ANEXOS

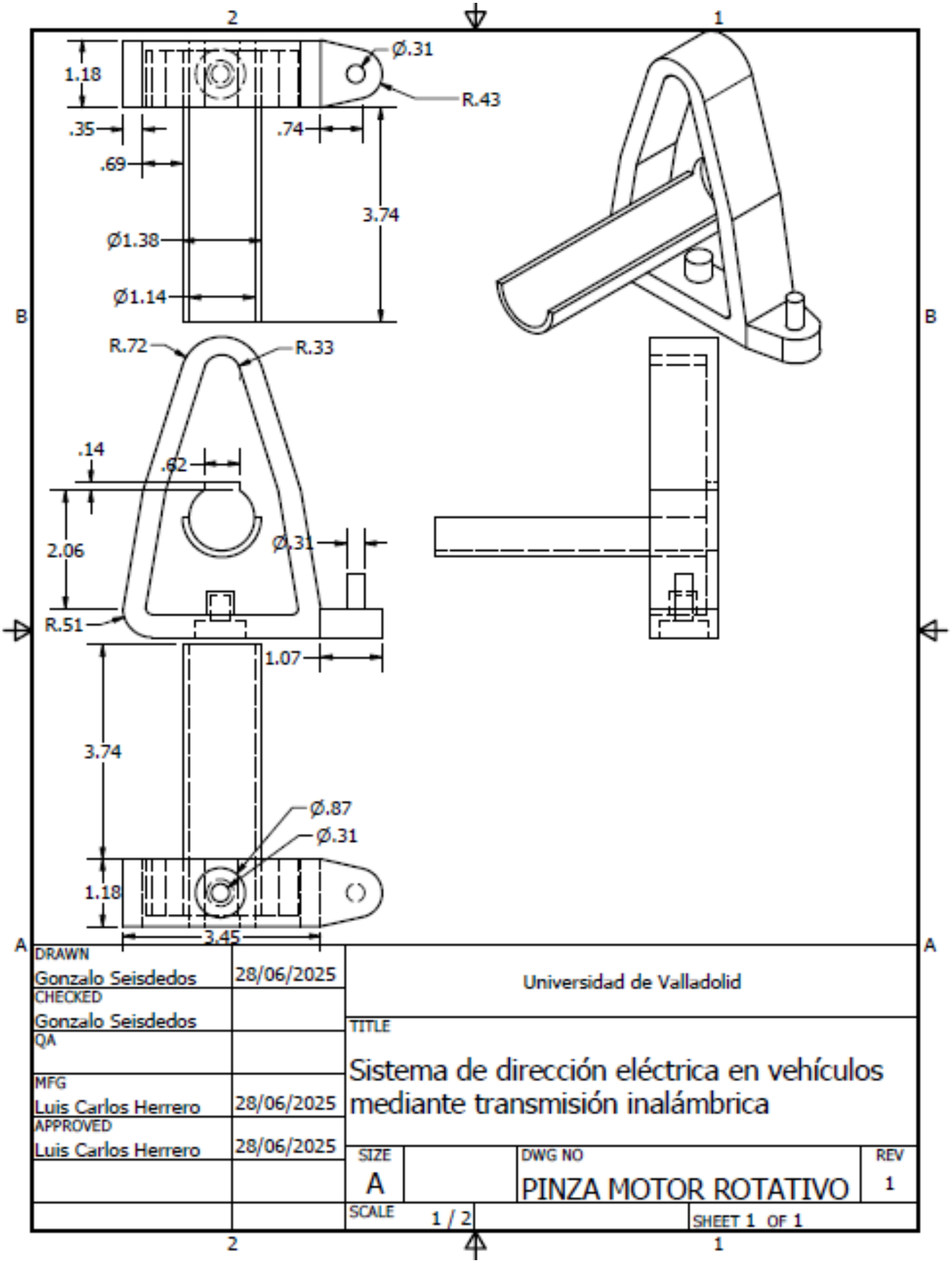


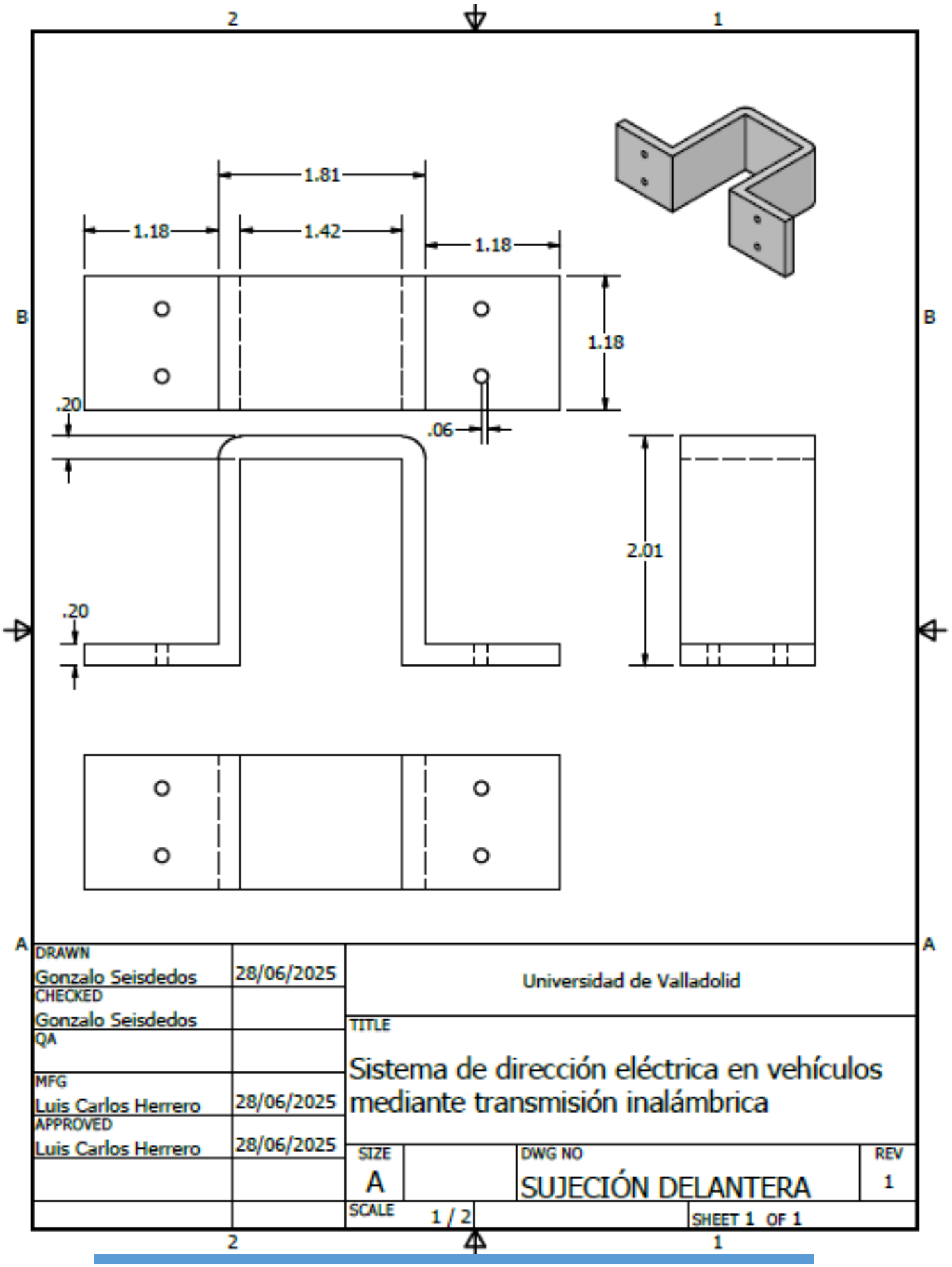
DRAWN Gonzalo Seisdedos	28/06/2025	Universidad de Valladolid	
CHECKED Gonzalo Seisdedos		TITLE Sistema de dirección eléctrica en vehículos mediante transmisión inalámbrica	
QA			
MFG Luis Carlos Herrero	28/06/2025		
APPROVED Luis Carlos Herrero	28/06/2025		
		SIZE A	DWG NO CAJA COMPONENTES (1/3) REV 1
		SCALE 1 / 2	SHEET 1 OF 1


2
1
B
B
2.76
2.36
.39
.20
.59
A
A
DRAWN
Gonzalo Seisdedos
28/06/2025
CHECKED
Gonzalo Seisdedos
QA
MFG
Luis Carlos Herrero
28/06/2025
APPROVED
Luis Carlos Herrero
28/06/2025
Universidad de Valladolid
TITLE
Sistema de dirección eléctrica en vehículos mediante transmisión inalámbrica
SIZE
A
DWG NO
CAJA COMPONENTES (2/3)
REV
1
SCALE
1 / 2
SHEET 1 OF 1
2
1

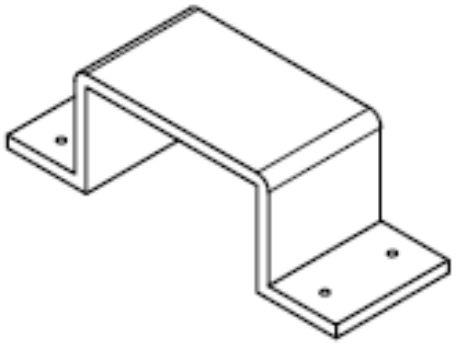


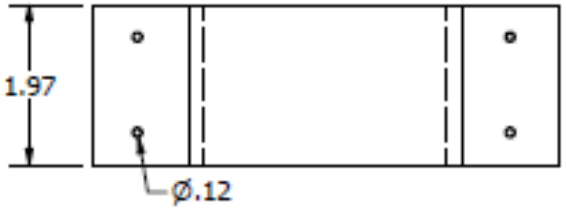
DRAWN Gonzalo Seisdedos	28/06/2025	Universidad de Valladolid	
CHECKED Gonzalo Seisdedos			
QA		TITLE: Sistema de dirección eléctrica en vehículos mediante transmisión inalámbrica	
MFG Luis Carlos Herrero	28/06/2025		
APPROVED Luis Carlos Herrero	28/06/2025		
		SIZE A	DWG NO CAJA COMPONENTES (3/3) — REV 1
		SCALE 1 / 2	SHEET 1 OF 1

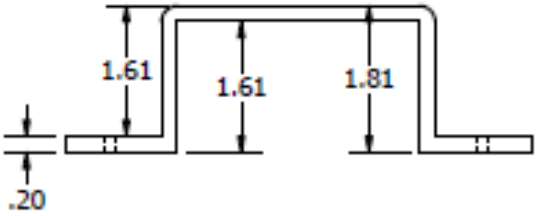

2
1
1.18
Ø.31
R.43
.35
.74
.69
3.74
Ø1.38
Ø1.14
R.72
R.33
.14
.62
2.06
Ø.31
R.51
1.07
3.74
Ø.87
Ø.31
1.18
3.45
B
A
DRAWN
Gonzalo Seisdedos
28/06/2025
CHECKED
Gonzalo Seisdedos
QA
MFG
Luis Carlos Herrero
28/06/2025
APPROVED
Luis Carlos Herrero
28/06/2025
Universidad de Valladolid
TITLE
Sistema de dirección eléctrica en vehículos mediante transmisión inalámbrica
SIZE
A
DWG NO
PINZA MOTOR ROTATIVO
REV
1
SCALE
1 / 2
SHEET 1 OF 1

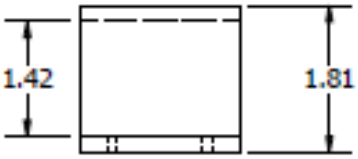


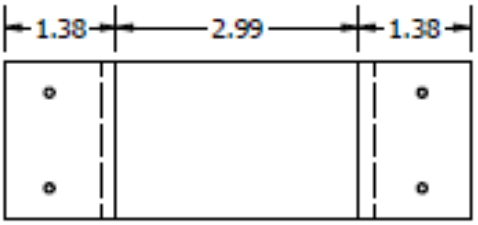
DRAWN Gonzalo Seisdedos	28/06/2025	Universidad de Valladolid
CHECKED Gonzalo Seisdedos		TITLE: Sistema de dirección eléctrica en vehículos mediante transmisión inalámbrica
QA		
MFG Luis Carlos Herrero	28/06/2025	
APPROVED Luis Carlos Herrero	28/06/2025	
SIZE A	DWG NO SUJECIÓN DELANTERA	REV 1
SCALE 1 / 2		SHEET 1 OF 1



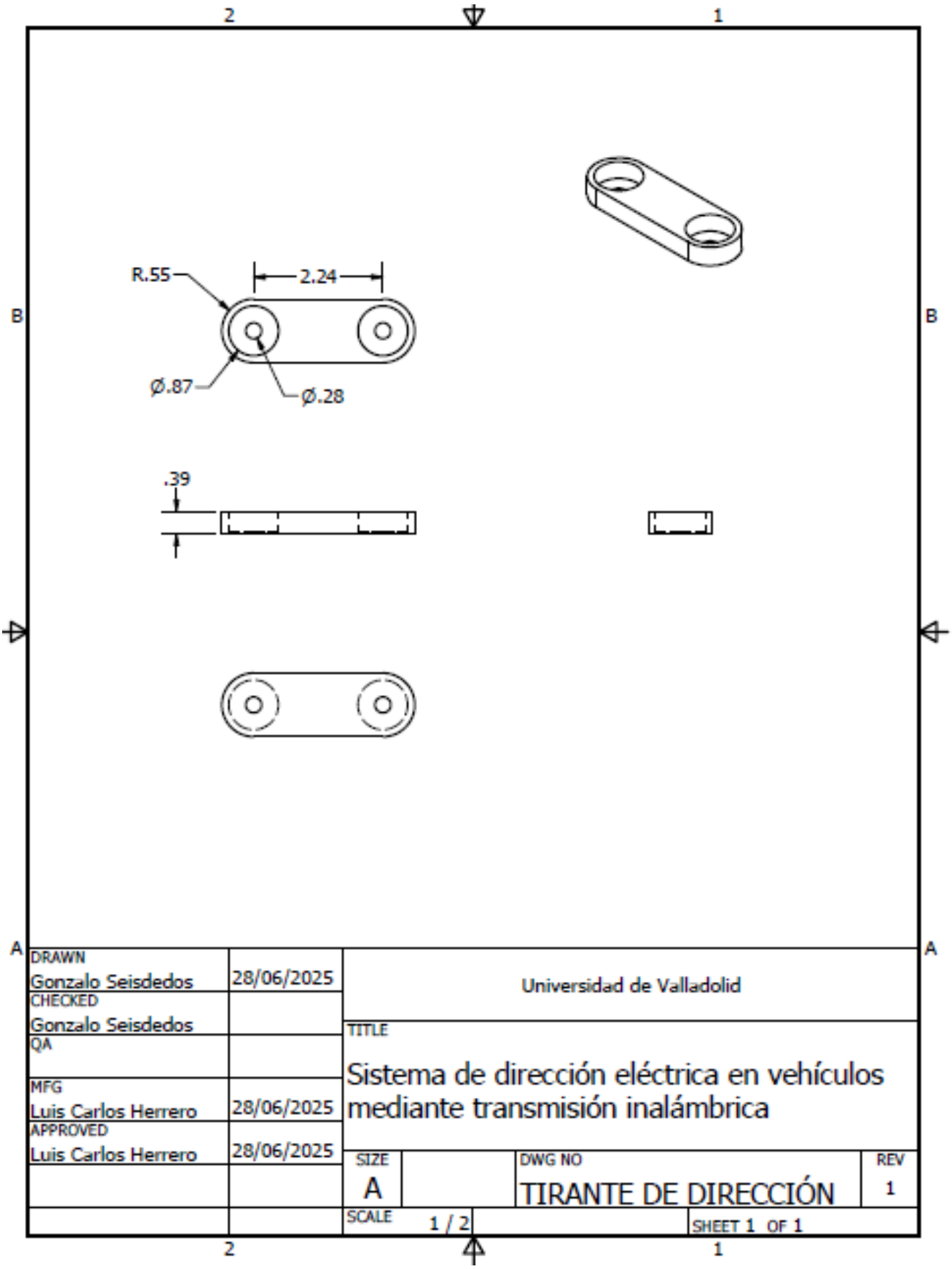








DRAWN Gonzalo Seisdedos	28/06/2025	Universidad de Valladolid
CHECKED Gonzalo Seisdedos		TITLE: Sistema de dirección eléctrica en vehículos mediante transmisión inalámbrica
QA		
MFG Luis Carlos Herrero	28/06/2025	
APPROVED Luis Carlos Herrero	28/06/2025	
SIZE A	DWG NO SUJECIÓN TRASERA	REV 1
SCALE 1 / 2	SHEET 1 OF 1	



DRAWN Gonzalo Seisdedos	28/06/2025	Universidad de Valladolid	
CHECKED Gonzalo Seisdedos		TITLE	
QA		Sistema de dirección eléctrica en vehículos mediante transmisión inalámbrica	
MFG Luis Carlos Herrero	28/06/2025		
APPROVED Luis Carlos Herrero	28/06/2025		
		SIZE A	DWG NO TIRANTE DE DIRECCIÓN
		SCALE 1 / 2	SHEET 1 OF 1

SHARVI ELECTRONICS
LET'S MAKE IT

DIGITAL OPTICAL ROTARY ENCODER

Technical Details:

Item		Diameter 38mm shaft 6mm type Incremental rotary encoder
Resolution (P/R)		100, 360, 400, 600, 1000 PPR
Input Phase		AB Phase
Output Phase		NPN Open Collector
	Supply Voltage	5-24 VDC
Electrical	Current Consumption	Max , 40mA
	Response Frequency	Max , 100 Khz
	Allowable Revolution	Max , 3000 rev /min
Mechanical	Starting torque	Max , 20 gf.cm (0.002N.,m)
	Rotor Interia	Max , 15 g.cm2 (1,5* 10 -6 kg.m2)
	Shaft Loading	Radial : Max 2kgf, Axial : Max .1kgf
	Mechanical Speed	Max .5000 rev / min (*1)
Environmental	Ambient Temperature	-10~ 70 (at non- freezing status) , Stronger :- 25 ~ 85
	Ambient Humidity	35 ~85% RH , Stronger : 35~ 90 % RH
	Protection	Ip52 (IEG Standard)
	Vibration	1.5 mm amplitude at frequency of 10-55 Hz in each Of X,Y,Z direction for 2 hour.
	Shock	Max. 40G
Unit Weight		Approx :180g
Cable		2.0m (the cable length can be customized)
Approval		CE ROHS
(*1)		Mechanical speed > Allowable revolution , Please take allowable speed as standard when use

Wire Colour	OC VP OP 3-CHANNEL	TTL / HTL 6- CHANNEL	PIN 9-POLE	EXPLANATION
RED	VCC	VCC	1	Supply Voltage
BLACK	0 V	0 V	4	Common Port
GREEN	A	A	5	Signal wire
WHITE	B	B	3	Signal wire
YELLOW	Z	Z	8	Signal wire
BROWN	–	-A	6	Signal wire
GRAY	–	-B	7	Signal wire
ORANGE	–	-Z	2	Signal wire
SHIELD	SHIELD	SHIELD	9	

Output:

AB two-phase quadrature output rectangular pulse, the circuit output is **NPN** open collector output **type.**

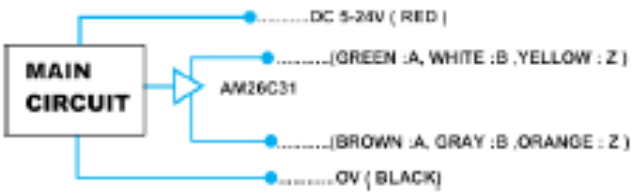
This type can be output with internal pull -up resistor available in Arduino, microcontroller s or PLC, such asAtmega, pic, 51 or microcontroller Mitsubi shi PLC.

If Internal Pullup is not available, then you need to Pull-up Output Channel A & B with resistors supplied with product. i.e. resister between Green and RED, White and RED wire.

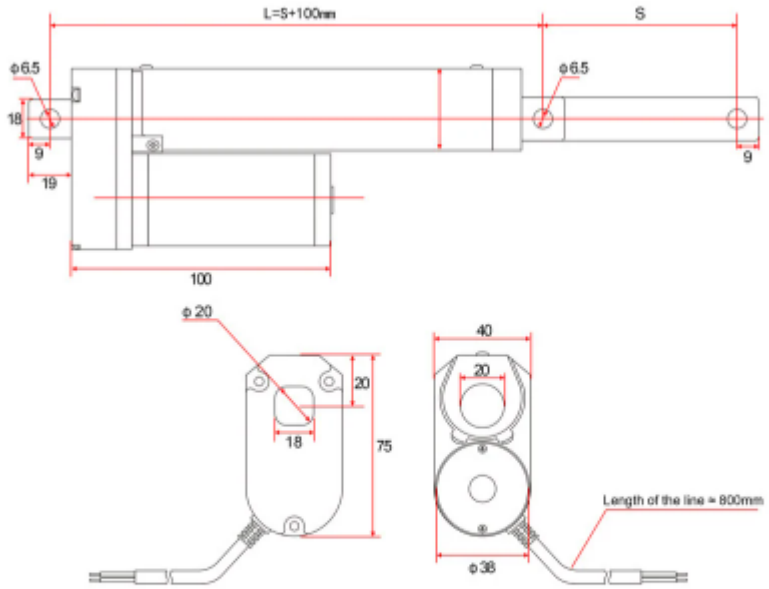
OC NPN OPEN COLLECTOR

OC VOLTAGE OUTPUT

ILD : LONG DRIVER OUTPUTS 5V
LDH : LONG DRIVER OUTPUTS 5-24V



Tamaño:

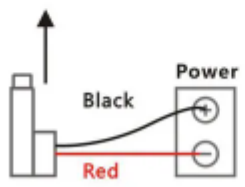


L= Distancia mínima de instalación
S= Carrera efectiva

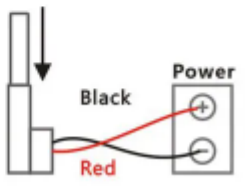
Introducción del tamaño:
Longitud de carrera: significa la longitud de movimiento del actuador lineal.
Longitud retraída: significa la longitud cuando el actuador lineal está 100% completamente cerrado.
Longitud extendida: significa la longitud cuando el actuador lineal está 100% completamente abierto.

Introducción de la longitud de la carrera:
Carrera de 50 mm, longitud retraída = 155 mm, longitud extendida = 205 mm
Carrera de 100 mm, longitud retraída = 205 mm, longitud extendida = 305 mm
Carrera de 150 mm, longitud retraída = 255 mm, longitud extendida = 405 mm
Carrera de 200 mm, longitud retraída = 305 mm, longitud extendida = 505 mm
Carrera de 250 mm, longitud retraída = 355 mm, longitud extendida = 605 mm
Carrera de 300 mm, longitud retraída = 405 mm, longitud extendida = 705 mm
... ...

Stroke	30mm/50mm/100mm/150mm/200mm/250mm/300mm/350mm/400mm/450mm/500mm							
Voltage	DC12V/24V							
No-load speed	90mm/s	60mm/s	35mm/s	20mm/s	15mm/s	10mm/s	6mm/s	4mm/s
Max force	50N	100N	300N	500N	700N	1000N	1200N	1500N



Work ① :
Push rod extension



Work ② :
Push rod retraction

Código Emisor:

25.05.26_Emisor.ino

```
#include <WiFi.h>
#include <esp_now.h>
#include <mbedtls/aes.h>

#define ENCODER_PIN_A  18
#define ENCODER_PIN_B  19
#define PPR 1000
#define CPR (PPR * 2)

int botonPin = 2;
int ledPin = 4;          // LED en GPIO 4

int estadoBoton = 0;
int ultimoEstado = 1;
int envioActivo = 0;

uint8_t direccion_receptor[] = {0x3C, 0x8A, 0x1F, 0x9B, 0x92, 0x10};

uint8_t aes_key[16] = {
    0x01, 0x02, 0x03, 0x04,
    0x05, 0x06, 0x07, 0x08,
    0x09, 0x0A, 0x0B, 0x0C,
    0x0D, 0x0E, 0x0F, 0x10
};

volatile int position = 0;
volatile int lastStateA = 0;

void IRAM_ATTR encoderISR() {
    int stateA = digitalRead(ENCODER_PIN_A);
    int stateB = digitalRead(ENCODER_PIN_B);

    if (stateA != lastStateA) {
        if (stateA == stateB) {
            position++;
        } else {
            position--;
        }
    }
    lastStateA = stateA;
}

```

```
43  void EnvioDatos(const uint8_t *mac, esp_now_send_status_t status) {
44      Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Envio exitoso" : "Envio fallido");
45  }
46
47  void cifrarAES(uint8_t* data, size_t len) {
48      mbedtls_aes_context aes;
49      mbedtls_aes_init(&aes);
50      mbedtls_aes_setkey_enc(&aes, aes_key, 128);
51      mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, data, data);
52      mbedtls_aes_free(&aes);
53  }
54
55  void setup() {
56      Serial.begin(115200);
57      WiFi.mode(WIFI_STA);
58
59      pinMode(ENCODER_PIN_A, INPUT_PULLUP);
60      pinMode(ENCODER_PIN_B, INPUT_PULLUP);
61      pinMode(botonPin, INPUT_PULLUP);
62      pinMode(ledPin, OUTPUT);
63
64      digitalWrite(ledPin, LOW);  // LED apagado al inicio
65
66      lastStateA = digitalRead(ENCODER_PIN_A);
67      attachInterrupt(digitalPinToInterrupt(ENCODER_PIN_A), encoderISR, CHANGE);
68
69      if (esp_now_init() != ESP_OK) {
70          Serial.println("Error al inicializar ESP-NOW");
71          return;
72      }
73
74      esp_now_register_send_cb(EnvioDatos);
75
76      esp_now_peer_info_t peerInfo = {};
77      memcpy(peerInfo.peer_addr, direccion_receptor, 6);
78      peerInfo.channel = 0;
79      peerInfo.encrypt = false;
80
81      if (esp_now_add_peer(&peerInfo) != ESP_OK) {
82          Serial.println("Error al registrar peer");
83      }
84  }
```

```
86  void loop() {
87      estadoBoton = digitalRead(botonPin);
88
89      if (estadoBoton != ultimoEstado) {
90          delay(50);  // antirrebote básico
91          if (estadoBoton == LOW) {
92              envioActivo = !envioActivo;
93
94              // Ahora el LED sigue el estado inverso del envío
95              if (envioActivo) {
96                  digitalWrite(ledPin, LOW);   // envío activo -> LED apagado
97                  Serial.println("Envio ACTIVADO");
98              } else {
99                  digitalWrite(ledPin, HIGH);  // envío inactivo -> LED encendido
100                 Serial.println("Envio DETENIDO");
101             }
102         }
103     }
104
105     ultimoEstado = estadoBoton;
106
107     if (envioActivo) {
108         int pos_snapshot;
109         noInterrupts();
110         pos_snapshot = position;
111         interrupts();
112
113         float angle = (pos_snapshot / (float)CPR) * 360.0;
114
115         uint8_t buffer[16] = {0};
116         memcpy(buffer, &angle, sizeof(angle));
117
118         cifrarAES(buffer, 16);
119
120         esp_now_send(direccion_receptor, buffer, 16);
121         Serial.print("Ángulo enviado: ");
122         Serial.println(angle);
123     }
124
125     delay(100);
126 }
```

Comunicación.cpp

```
#include <mbedtls/aes.h>
#include "Config.h"
#include <esp_now.h>
#include <WiFi.h>

volatile int recep_incorrecta = 0;
unsigned long ultimaRecepcion = 0;
float angulo_volante = 0;
float angulo_anterior = 0;

uint8_t aes_key[16] = {
  0x01, 0x02, 0x03, 0x04,
  0x05, 0x06, 0x07, 0x08,
  0x09, 0x0A, 0x0B, 0x0C,
  0x0D, 0x0E, 0x0F, 0x10
};

void descifrarAES(uint8_t* data, size_t len) {
  mbedtls_aes_context aes;
  mbedtls_aes_init(&aes);
  mbedtls_aes_setkey_dec(&aes, aes_key, 128);
  mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_DECRYPT, data, data);
  mbedtls_aes_free(&aes);
}

void RecibirDatos(const esp_now_recv_info_t *info, const uint8_t *datos, int len) {
  if (len == 16) {
    uint8_t decrypted[16];
    memcpy(decrypted, datos, 16);
    descifrarAES(decrypted, 16);
    memcpy(&angulo_volante, decrypted, sizeof(float));

    recep_incorrecta = 0;
    ultimaRecepcion = millis();

    Serial.print("Ángulo recibido: ");
    Serial.println(angulo_volante);
  } else {
    recep_incorrecta = 1;
    Serial.println("Error: datos recibidos con LONGITUD INCORRECTA");
  }
}
```

Comunicación.h:

```
#pragma once
#include <esp_now.h>

void RecibirDatos(const esp_now_recv_info_t *info, const uint8_t *datos, int len);

```

Config.h:

```
#pragma once
#include <Arduino.h>

// Pines motores
#define M1_DIR_PIN 23
#define M1_PWM_PIN 21

#define M2_DIR_PIN 18
#define M2_PWM_PIN 5

// PWM
const int frequency = 5000;
const ledc_timer_bit_t resolution = LEDC_TIMER_8_BIT;
const ledc_timer_t timer_m1 = LEDC_TIMER_1;
const ledc_timer_t timer_m2 = LEDC_TIMER_0;

const ledc_channel_t PWM_Channel_M1 = LEDC_CHANNEL_1;
const ledc_channel_t PWM_Channel_M2 = LEDC_CHANNEL_0;

// Variables globales
extern volatile int recep_incorrecta;
extern unsigned long ultimaRecepcion;
extern float angulo_volante;
extern float angulo_anterior;
extern int dutyCycleM1;
extern int dutyCycleM2;
extern bool motorRotativoActivo;
```

MotorLineal.cpp:

```cpp
#include "Config.h"
#include "MotorLineal.h"
#include "driver/ledc.h"

int dutyCycleM2 = 244;

void motorLinealAdelante() {
  digitalWrite(M2_DIR_PIN, HIGH);
}

void motorLinealAtras() {
  digitalWrite(M2_DIR_PIN, LOW);
}

void motorLinealSetVelocidad(int pwm) {
  ledc_set_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M2, pwm);
  ledc_update_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M2);
}

void inicializarMotores() {
  pinMode(M2_DIR_PIN, OUTPUT);
  pinMode(M2_PWM_PIN, OUTPUT);

  ledc_timer_config_t timerConfig = {
    .speed_mode = LEDC_HIGH_SPEED_MODE,
    .duty_resolution = resolution,
    .timer_num = timer_m2,
    .freq_hz = frequency,
    .clk_cfg = LEDC_AUTO_CLK
  };
  ledc_timer_config(&timerConfig);

  ledc_channel_config_t channelConfig = {
    .gpio_num = M2_PWM_PIN,
    .speed_mode = LEDC_HIGH_SPEED_MODE,
    .channel = PWM_Channel_M2,
    .intr_type = LEDC_INTR_DISABLE,
    .timer_sel = timer_m2,
    .duty = 0,
    .hpoint = 0
  };
  ledc_channel_config(&channelConfig);
}
```

MotorLineal.h:

```
1   #pragma once
2   void motorLinealAdelante();
3   void motorLinealAtras();
4   void motorLinealSetVelocidad(int pwm);
5   void inicializarMotores();
6
```

MotorRotativo.cpp:

```cpp
#include "Config.h"
#include "MotorRotativo.h"
#include "driver/ledc.h"

bool motorRotativoActivo = false;
int dutyCycleM1 = 244;

void iniciarMotorRotativo() {
  if (motorRotativoActivo) return;

  digitalWrite(M1_DIR_PIN, HIGH);
  for (int duty = 0; duty <= dutyCycleM1; duty += 5) {
    ledc_set_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1, duty);
    ledc_update_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1);
    delay(15);
  }

  motorRotativoActivo = true;
}

void frenarMotorRotativo() {
  if (!motorRotativoActivo) return;

   int actualDuty = ledc_get_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1);
   for (int duty = actualDuty; duty >= 0; duty -= 5) {
     ledc_set_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1, duty);
     ledc_update_duty(LEDC_HIGH_SPEED_MODE, PWM_Channel_M1);
     delay(15);
   }

   motorRotativoActivo = false;
}

void inicializarPWM() {
  pinMode(M1_DIR_PIN, OUTPUT);
  pinMode(M1_PWM_PIN, OUTPUT);

  ledc_timer_config_t timerConfig = {
    .speed_mode = LEDC_HIGH_SPEED_MODE,
    .duty_resolution = resolution,
    .timer_num = timer_m1,
    .freq_hz = frequency,
    .clk_cfg = LEDC_AUTO_CLK
  };
  ledc_timer_config(&timerConfig);
```

```
46
47      ledc_channel_config_t channelConfig = {
48          .gpio_num = M1_PWM_PIN,
49          .speed_mode = LEDC_HIGH_SPEED_MODE,
50          .channel = PWM_Channel_M1,
51          .intr_type = LEDC_INTR_DISABLE,
52          .timer_sel = timer_m1,
53          .duty = 0,
54          .hpoint = 0
55      };
56      ledc_channel_config(&channelConfig);
57  }
```

MotorRotativo.h

```
1   #pragma once
2   void iniciarMotorRotativo();
3   void frenarMotorRotativo();
4   void inicializarPWM();
5   void inicializarMotores();
6
```

ControlMotores.ino:

```
#include <WiFi.h>
#include <esp_now.h>
#include "driver/ledc.h"

#include "Config.h"
#include "MotorRotativo.h"
#include "MotorLineal.h"
#include "Comunicacion.h"
const unsigned long TIMEOUT_RECEPCION = 200;
void setup() {
  Serial.begin(115200);

  inicializarPWM();
  inicializarMotores();
  iniciarMotorRotativo();

  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error al inicializar ESP-NOW");
    return;
  }

  esp_now_register_recv_cb(RecibirDatos);
}

void loop() {
  const float umbral = 0.5;

  if (millis() - ultimaRecepcion > TIMEOUT_RECEPCION) {
    recep_incorrecta = 1;
    Serial.println("Error: NO se reciben datos");
  }

  if (recep_incorrecta == 1 && motorRotativoActivo) {
    frenarMotorRotativo();
  }

  if (recep_incorrecta == 0 && !motorRotativoActivo) {
    iniciarMotorRotativo();
  }


  if (abs(angulo_volante - angulo_anterior) < 65) {
  dutyCycleM2=100;
  }
  else{
      dutyCycleM2=180;
```

```
48    }
49    if (recep_incorrecta == 0) {
50      if (angulo_volante > angulo_anterior + umbral) {
51        motorLinealAdelante();
52        motorLinealSetVelocidad(dutyCycleM2);
53      } else if (angulo_volante < angulo_anterior - umbral) {
54        motorLinealAtras();
55        motorLinealSetVelocidad(dutyCycleM2);
56      } else {
57        motorLinealSetVelocidad(0);
58      }
59    }
60
61    angulo_anterior = angulo_volante;
62    delay(100);
63  }
```