Universidad de Jaén

Escuela Politécnica Superior de Jaén

CONTROL DE PÉNDULO INVERTIDO MEDIANTE VOLANTE DE INERCIA

Autor: EDUARDO BERMÚDEZ CAÑABATE

Grado: GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL

Director: ELÍSABET ESTÉVEZ ESTÉVEZ
Departamento del director: Ingeniería de Sistemas y Automática

Fecha: 02/07/2024

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Doña ELÍSABET ESTÉVEZ ESTÉVEZ tutora y Don ALEJANDRO SÁNCHEZ GARCÍA cotutor, del Proyecto Fin de Carrera titulado: CONTROL DE UN PÉNDULO INVERTIDO MEDIANTE VOLANTE DE INERCIA que presenta EDUARDO BERMÚDEZ CAÑABATE, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, 02de Julio de 2024

El alumno	Tutora	Cotutor
BERMUDEZ CAÑABATE EDUARDO - 77374796J Firmado digitalmente por BERMUDEZ CAÑABATE EDUARDO - 77374796J Fecha: 2024.07.02 10:55:02 +02'00'	ESTEVEZ ESTEVEZ ELISABET - 72578245C Firmado digitalmente por ESTEVEZ ESTEVEZ ELISABET - 72578245C Fecha: 2024.07.02 11:00:39 +02'00'	SANCHEZ GARCIA ALEJANDRO - 77321466C Firmado digitalmente por SANCHEZ GARCIA ALEJANDRO - 77321466C Fecha: 2024.07.02 12:05:50 +02'00'

Resumen:

En el presente trabajo, se realiza el desarrollo de un prototipo de péndulo invertido controlado por volante de inercia junto con el control de este para mantener una referencia y el desarrollo de una interfaz para ordenador.

Para el caso del prototipo, se ha realizado un proceso teórico y experimental para desarrollar diferentes volantes con el fin de comparar los resultados, para el control del sistema se ha usado el microcontrolador conocido como ESP32. Este tambien es el encargado de gestionar las comunicaiones usando el protocolo Modbus TCP/IP.

Los actuadores del sistema son un motor de corriente continua y una controladora para el motor.

Para el caso de la interfaz de ordenador, se ha usado Processing para su desarrollo. Esta interfaz permite al usuario interactuar con la controladora, ya que permite recibir datos y enviar las constantes del PID que el usuario introduzca, todo esto usando Modbus.

Abstract:

In this work, the development of an inverted pendulum prototype controlled by a flywheel is carried out, along with its control to maintain a reference, and the development of a computer interface.

For the prototype, a theoretical and experimental process has been undertaken to develop different flywheels with the aim of comparing the results. The microcontroller known as ESP32 has been used for system control. It is also responsible for managing communications using the Modbus TCP/IP protocol.

The system's actuators are a direct current motor and a motor controller.

For the computer interface, Processing has been used for its development. This interface allows the user to interact with the prototype's controller, as it can receive data and send the PID constants that the user inputs, all using Modbus.

Índice de contenido

1. INTRODUCCIÓN 1
1.1. Motivación 1
1.2. ESTADO DEL ARTE 1
1.2.1. Péndulo Invertido sobre carro: 2
1.2.2. Péndulo de furuta o péndulo rotatorio invertido. 2
1.2.3. Transporte personal Segway 3
1.2.4. The Cubli 4
2. Objetivos 5
3. Descripción de los componentes del lazo de control 7
3.1. Controladora 8
3.2. Actuador 9
3.3. Sensor de inclinación. 11
4. Analisis del volante de inercia 13
4.1. Análisis teórico del sistema. 13
4.2. Volante con acople para eje liso: 15
4.3. Volante sin acople para eje liso, tamaño M: 17
4.4. Volante sin acople para eje liso, tamaño XL 19
4.5. Comparativa de los volantes de inercia 21
5. Estructura del proyecto 22
5.1. Base del péndulo 22
5.2. Estructura para la base del péndulo 24
6. Software de la controladora 28
6.1. freeRTOS 28
6.2. Comunicaciones controladora-ordenador 35
6.2.1. Código Wifi para la controladora 35
6.2.2. Código Modbus para la controladora 38
6.3. Control PID 45
7. Interfaz HMI 47
7.1. Guía de usuario de la HMI 47
7.2. Instalación librería “EasyModbus”. 50
7.3. Código Modbus para la HMI 51
8. Presupuesto y mediciones 55
9. Caso de uso 56
10. Conclusiones 58
Referencias 60

Índice de Figuras:

Figura 1: Péndulo invertido sobre carro. 2
Figura 2: Péndulo de Furuta 3
Figura 3: Transporte personal Segway 4
Figura 4: The Cubli 4
Figura 5: Lazo cerrado de control del sistema 7
Figura 6: ESP-WROOM-32 8
Figura 7: Motor cc con reductora de 70:1 y encoder. 9
Figura 8:Controladora MDV 2x2A DC 10
Figura 9: Sensor BNO055 12
Figura 10: Volante con acople para eje liso 15
Figura 11: Brida de eje rígido para ejes lisos de 6mm de diámetro. 16
Figura 12: Volante sin acoplador, tamaño M 17
Figura 13: Inercia del volante sin acoplador, tamaño M 18
Figura 14: Inercia del volante con todos los tornillos y tuercas 19
Figura 15: Inercia del volante sin acople, tamaño XL 20
Figura 16: Inercias del volante sin acople y todos los tornillos y tuercas, tamaño XL 20
Figura 17: Comparativa del peso e inercia de los distintos volantes sin acoplador 21
Figura 18: Base del péndulo 22
Figura 19:Montaje. A Vista del sistema. B vista de perfil 23
Figura 20: Montaje 2. A, Vista 3D del sistema. B vista de perfil 24
Figura 21: Estructura incompleta 25
Figura 22: Estructura completa. A vista 3D, B vista de perfil 25
Figura 23: Soporte sensor y controladora 26
Figura 24: Montaje completo del sistema. A vista 3D. B vista perfil derecho. C vista de planta. 26
Figura 25: Montaje completo del sistema en la realidad. 27
Figura 26: Núcleo de ejecución de la función void loop() 29
Figura 27: Definición de la tarea 30
Figura 28: Creación de la función y del bucle infinito para la nueva tarea 31
Figura 29: Apuntar la tarea a un núcleo en concreto 32
Figura 30: Visualizar el tamaño restante de la pila [14] 33
Figura 31: Código completo de freeRTOS 34
Figura 32: Definición parámetros Wifi 36
Figura 33: Configuración del ESP32 para conexión Wifi_STA 36
Figura 34: Reconexión en caso de pérdida de conexión Wifi 37
Figura 35: Función que se ejecuta en caso de perder la conexión 37
Figura 36:Esquema de Modbus TCP/IP. [17] 39
Figura 37: Librería Modbus para la controladora 39
Figura 38: Definición de los registros. 40
Figura 39: Función que crea los registros 41
Figura 40: Creación de los registros en función del área de memoria. [18]. 41
Figura 41: Definición de variables auxiliares en el núcleo 0 42
Figura 42: Flujograma, código núcleo 0 44
Figura 43: Código PID, ejecutándose en el núcleo 1 45
Figura 44: Interfaz HMI en modo edición 47
Figura 45: Ejemplo de la interfaz en modo edición 49

Figura 46: Interfaz en modo adquisición 49
Figura 47: Web de la librería en GitHub 50
Figura 48: Instrucción para cargar la librería en el código 51
Figura 49: Manual de usuario de la librería en GitHub 51
Figura 50: Creación de una nueva tarea. 52
Figura 51: Flujograma del hilo de Modbus en Processing 53
Figura 52: Tabla de presupuesto y mediciones del proyecto.[22], [23] 55
Figura 53:Respuesta del sistema en el mejor de los casos 56
Figura 54: QR video demostración 57

Índice de tablas:

Tabla 1: Comportamiento del motor en función de las señales para la controladora del motor [6] 11

1. INTRODUCCIÓN

1.1. Motivación

La amplia variedad de aplicaciones de un péndulo invertido controlado por volante de inercia, abarcan desde un barco hasta aviones y cohetes pasando por la industria médica, lo que lo convierten en un sistema tremendamente interesante. Por ejemplo, en los barcos se usa para mantener la embarcación adrizada, en los satélites y naves espaciales, se usa para controlar la orientación en el espacio. Lo que es crucial para poder orientar las cámaras, las antenas y demás dispositivos de una forma precisa y hacia objetivos específicos. Por otro lado, en la industria médica, su aplicación principal se ve reflejada en los exoesqueletos, ya que emplean el péndulo invertido controlado por volante de inercia para mantener el equilibrio y así poder proporcionar asistencia en el movimiento a personas con algún tipo de discapacidad motriz.

Además, este sistema, se puede usar para explicar conceptos de control y estabilización de sistemas dinámicos. En resumen, un péndulo invertido controlado por volante de inercia es un dispositivo extremadamente interesante para mantener una referencia constante en un plano del espacio.

Este documento abarcará todas las fases de diseño de un péndulo invertido, desde los cálculos teóricos y el diseño de la estructura y el volante de inercia, hasta las comunicaciones entre microcontrolador y el ordenador utilizando un protocolo industrial. Todo esto con el fin de sentar las bases y ofrecer conclusiones para futuros proyectos de características o requerimientos similares.

1.2. ESTADO DEL ARTE

Hoy por hoy las aplicaciones del péndulo invertido controlado se centran en la robótica para poder mantener el equilibrio, pero la evolución de los péndulos invertidos ha conllevado a que se desarrollen diferentes modelos de péndulos para satisfacer distintas características, a continuación, se exponen algunos ejemplos empezando por el péndulo invertido sobre carro.

1.2.1. Péndulo Invertido sobre carro:

El péndulo invertido sobre carro, que consiste en mantener la posición vertical de un péndulo montado en un vehículo que puede moverse hacia delante y hacia atrás tratando de mantener el péndulo en una posición vertical. Para realizar el control de este sistema deberemos medir el ángulo θ y la velocidad angular del péndulo θ. El objetivo es que θ=0, siendo la variable de control sobre el sistema *u* que es la fuerza que se aplica al vehículo, todo esto se puede ver en la Figura 1. [1]

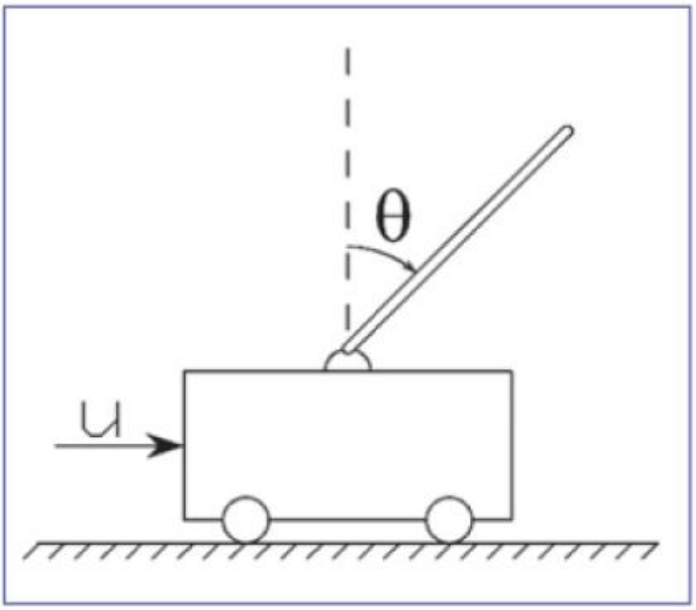

Figura 1: Péndulo invertido sobre carro.

1.2.2. Péndulo de furuta o péndulo rotatorio invertido.

Consta de un brazo que gira en el plano horizontal y que está unido a un péndulo que puede girar libremente en el plano vertical. Diseñado por *Katsuhisa Furuta*. El objetivo de este péndulo es el mantener el péndulo en equilibrio mediante la aceleración angular de la varilla que va unida a la base que gira. Las distintas medidas y relaciones físicas que afectan al péndulo mencionado anteriormente se pueden observar en la Figura 2

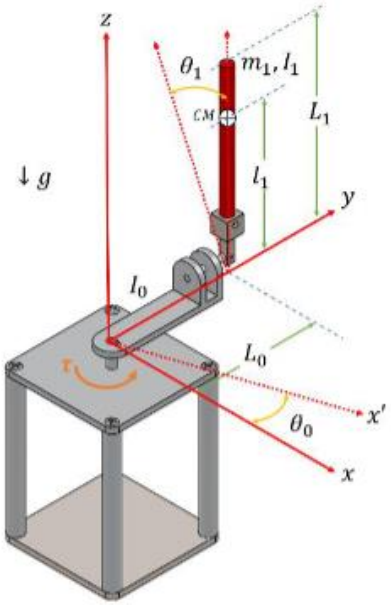

Figura 2: Péndulo de Furuta

1.2.3. Transporte personal Segway

Este sistema es un vehículo de transporte personal usado por policías en algunas ciudades, ver Figura 3. Su comportamiento dinámico se asemeja enormemente al de un péndulo invertido sobre carro. Pero con la complejidad añadida de que la persona que va encima es capaz de controlar la velocidad y la dirección del vehículo. El control de este sistema debe evitar que la persona se caiga y sea un sistema fácil de manejar[2]

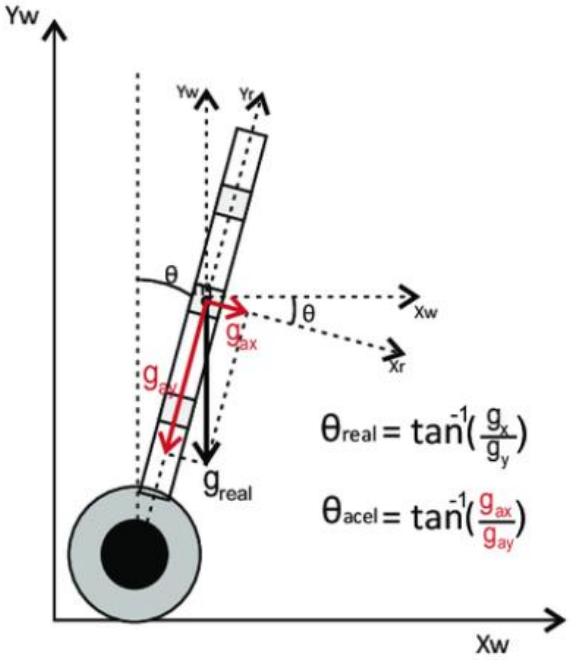

Figura 3: Transporte personal Segway

1.2.4. The Cubli

Diseñado por la ETH de Zürich, consiste en un cubo con tres ruedas de inercia montadas en tres caras del cubo, este sistema es capaz de mantener el equilibrio cuando está apoyado sobre uno de sus vértices. Esto es posible gracias a las ruedas de inercia, que mediante el giro consiguen aplicar un momento de inercia opuesto al que haría que el sistema perdiera el equilibrio, el sistema completo se puede ver en la figura que hay a continuación. [3]

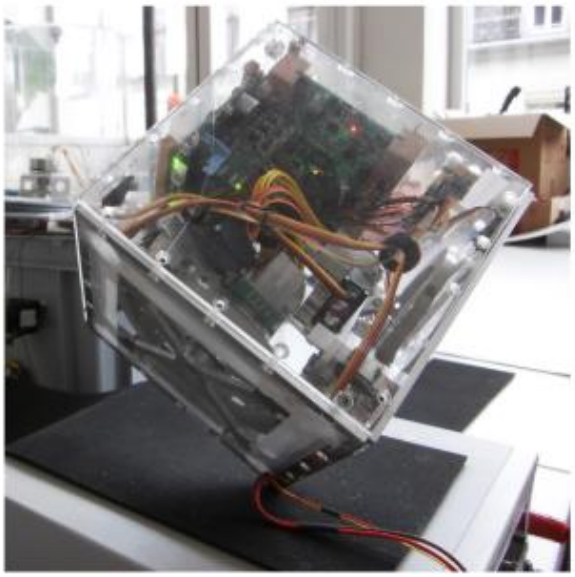

Figura 4: The Cubli

2. Objetivos

El objetivo principal de este proyecto es el de desarrollar, construir y controlar un péndulo invertido utilizando un volante de inercia. Además de desarrollará una Interfaz Hombre-Máquina (HMI) que permitirá la interacción y el control del sistema. Los pasos principales son los siguientes.

1) Identificar los componentes hardware.

Los componentes requeridos son los siguientes, un microcontrolador *ESP-WROOM-32,* el sensor de orientación absoluta Adafruit *IMU de 9 - DOF fusión – BNO055,* el motor de corriente continua *Pololu*. modelo *GB 37 70:1* de 12V con encoder de 64 CPR y una reductora de 70:1. Por último la controladora *MDV 2x2A DC Motor Controller* del fabricante *DFROBOT.*

2) Realizar un proceso teórico y experimental para el diseño del volante de inercia.

Un análisis teórico del sistema permite identificar los parámetros principales que influyen en el diseño. Estos son, el par del motor y la inercia del volante. Basándose en este último se deberán diseñar varios volantes de inercia con diferentes características y prestaciones.

3) Desarrollar una estructura para el péndulo.

Diseñar y construir una estructura que albergue los componentes del sistema, como el motor, un microcontrolador, un sensor capaz de medir el ángulo de inclinación y el volante de inercia, esta estructura debe de ser lo más compacta posible. Este sistema va montado sobre un eje que permite su rotación en sentido horario y en sentido antihorario.

4) Desarrollar un programa de control y comunicación Modbus para el ESP32.

Implementar un programa que permita usar los dos núcleos del ESP32 mediante el sistema operativo en tiempo real conocido como *freeRTOS.* Un núcleo estará destinado al control del sistema mientras que el otro manejará las comunicaciones *Modbus.*

5) Permitir el control de distintos parámetros del sistema desde una aplicación.

Implementar una HMI para ordenador utilizando *Processing,* que a través del protocolo de comunicación *Modbus,* permita al usuario editar las constantes del PID y monitorear el ángulo del sistema.

3. Descripción de los componentes del lazo de control

El proyecto de estudio en este documento tiene el siguiente lazo cerrado de control, donde.

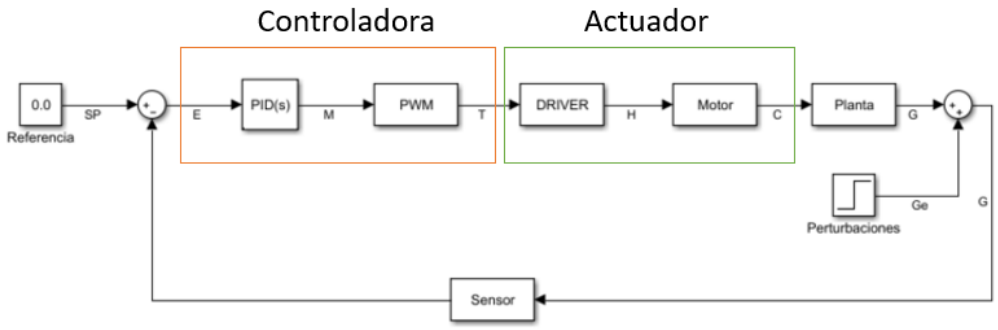

Figura 5: Lazo cerrado de control del sistema

Las variables significativas del sistema son:

- SP: "Set point", referencia de 0. 0º
- E: Error entre el set point y la medición del sensor, ambos en grados decimales.
- M: Indica el ciclo de trabajo de una señal PWM y el estado del pin (LOW/HIGH) en cargado de indicar la dirección de giro.
- T: Gracias al bloque PWM, se genera una señal PWM que admite el driver del motor,
- H: Proceso manipulado, el sentido de giro y la velocidad de un motor
- G: Variable controlada, ángulo de inclinación del sistema
- Ge: Perturbaciones a la salida, perturbaciones a la salida de la planta debidas a las condiciones en las que se encuentre el sistema, estas posibles perturbaciones afectan a la inclinación del sistema.

A continuación, se explican los componentes necesarios para el lazo de control.

3.1. Controladora

Para el control del sistema, se optó. por el microcontrolador ESP-WROOM-32, fabricado por *Espressif Systems.* Es un microcontrolador muy potente y cuenta con una amplia variedad de recursos. Dicho microcontrolador destaca entre otras cosas por:

- **Conectividad**: cuenta con Wifi a 2.4 GHz y Bluetooth, lo que facilita la comunicación inalámbrica
- **Procesamiento**: dispone de dos núcleos que operan a 214 MHz, proporcionando un rendimiento robusto
- **Relojes**: Incluye relojes internos que pueden habilitarse para interrupciones internas y relojes externos que pueden utilizarse en cualquiera de los 25 pines GPIO disponibles.
- **Conversores**: Dispone de dos conversores ADC1 y el ADC2, ambos de 12 bits y 18 canales. Sin embargo, si se utiliza Wifi, el ADC 2 no estará disponible.
- **Interfaz**: Ofrece 2 pines par I2C (Inter-Integrated Circuit), 4 pines para SPI (Serial Peripheral Interface), 2 pines para I2S (Inter-Integrated Circuit Sound) y 3 pines UART (universal asynchronous receiver / transmitter). Los pines GPIO que admiten los protocolos mencionados se pueden observar en la figura 6

Estas características, como se pueden ver en la figura 6, hacen del ESP32-WROOM-32 una elección adecuada para este proyecto, ya que proporciona la capacidad de procesamiento y características para gestionar un control continuo y comunicaciones inalámbricas. Como, por ejemplo, en este proyecto, se utiliza el protocolo Modbus TCP/IP para comunicaciones entre controladora y ordenador. [4]

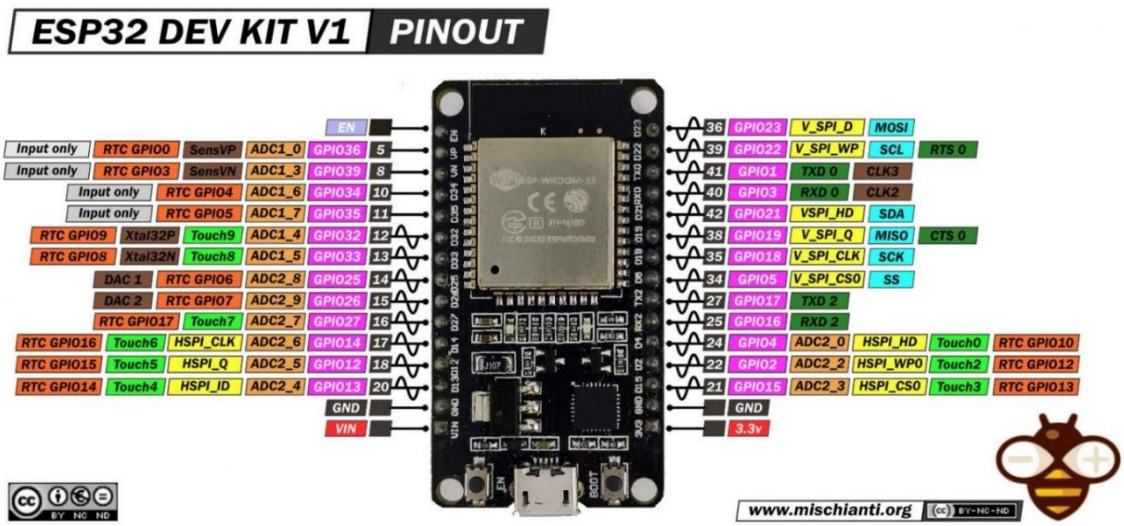

Figura 6: ESP-WROOM-32

3.2. Actuador

Motor cc:

Este motor de corriente continua con escobillas de 12V, del fabricante *Pololu* equipado con una caja reductora metálica de 70:1, proporciona el par necesario para este proyecto. Con un diámetro de 37 mm, una longitud de 70mm y un diámetro del eje motor a la salida de la caja reductora de 6mm y 16mm de largo, es el motor compacto ideal para el sistema.

Este motor alcanza una velocidad máxima de 150 rpm y un par de 27 kg*cm, además cuenta con un encoder de cuadratura integrado con una resolución de 64 pulsos por revolución del eje motor y 4480 pulsos por revolución del eje de salida de la caja reductora. El motor descrito previamente se puede ver la figura 7. [5]

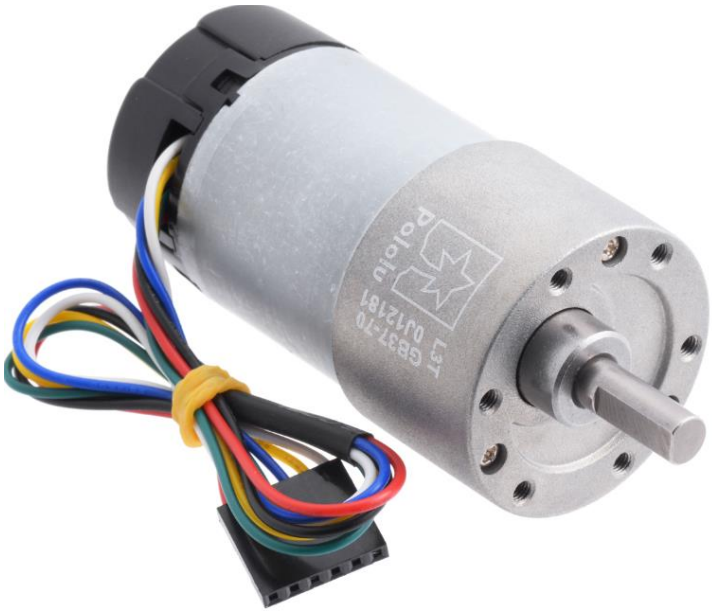

Figura 7: Motor cc con reductora de 70:1 y encoder.

Controladora del motor:

Para controlar la dirección y la velocidad del motor de corriente continua mencionado anteriormente, es necesaria una controladora. En este caso, se optó por la controladora MDV 2x2A DC del fabricante DFROBOT. Este driver utiliza del chip L298N, que permite controlar de uno a dos motores de forma independientemente, tanto en sentido como en velocidad. A continuación, se detallan algunas especificaciones del dispositivo.

Características del circuito de potencia:

- Voltaje de entrada del dispositivo Vs: 4.8~ 46V
- La corriente máxima de salida para los motores es de: 2A
- Disipación máxima de potencia: 25W

Características del circuito de control:

- Nivel alto: 2.3V<= Vin<= Vss
- Nivel bajo: -0.3V <= Vin<= 1.5V
- Temperaturas de funcionamiento: -25ºC ~ +180ºC
- Tipo de controlador, controlador de soble puente en H de alta potencia

Dimensiones y peso:

- Las dimensiones del módulo son, 47mm x 53mm
- El módulo pesa unos 29g

A continuación, se adjunta una figura del módulo con las conexiones y una descripción detallada.[6]

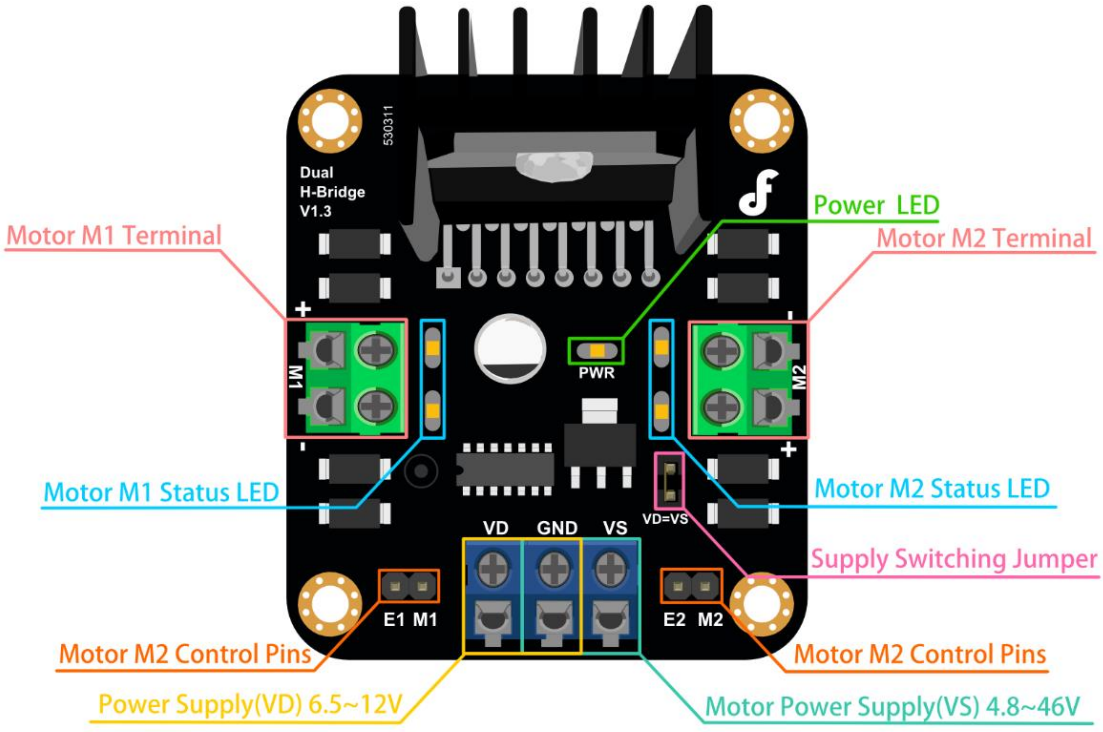

Figura 8:Controladora MDV 2x2A DC

Terminales del motor:

- Polaridad: Los terminales en los que se conecta el motor, cuentan con los signos “+” y “-” dibujados para representar la polaridad del motor.

Alimentación del módulo:

- VD: Alimentación de 6.5V ~ 12V
- VS: Alimentación de los motores de 4.8V ~ 46V
- GND: Tierra común del circuito de potencia y el circuito de control.

Pines de control del motor:

- E1, E2: pines para controlar la velocidad del motor, usando una señal PWM
- M1, M2: pines para la dirección del motor, si M2= 0, gira en sentido horario y cuando M2= 1, gira en sentido antihorario.

En la tabla 1, se puede observar el funcionamiento del motor ante diferentes combinaciones de señales.

E	M	Motor
LOW	LOW/HIGH	STOP
HIGH	HIGH	Gira en sentido antihorario
HIGH	LOW	Gira en sentido horario
PWM	LOW/HIGH	Velocidad en función de la señal PWM

Tabla 1: Comportamiento del motor en función de las señales para la controladora del motor [6]

Nota: 0=LOW, 1= HIGH y PWM= 0 ~ 255.

3.3. Sensor de inclinación.

El sensor de medida inercial de 9 ejes *Adafruit IMU de 9 - DOF fusión – BNO055*, ver figura 9 presenta las características adecuadas para medir el ángulo de inclinación de la estructura ya que cuenta con un acelerómetro, un giroscopio y un magnetómetro y con algoritmos de fusión de los sensores lo que proporciona una salida estable de orientación.

Este sensor combina los sensores mencionados con un procesador ARM Cortex -M0 de alta velocidad, para manejar todos los datos y cumplir con los requisitos de tiempo real. Los datos proporcionados por este sensor se obtienen en vectores de tamaño tres, ya que los valores se registran para cada uno de los ejes del espacio.

- Un vector de aceleraciones
- Un vector de velocidad angular
- Medición de campos magnéticos
- Obtención del ángulo de inclinación en los ejes del espacio

Junto con lo mencionado anteriormente, la facilidad de uso del sensor, gracias a la comunicación I2C y a la librería del sensor, que permite obtener los datos deseados con muy pocas líneas de código, lo convierten en el sensor ideal para este proyecto. [7]

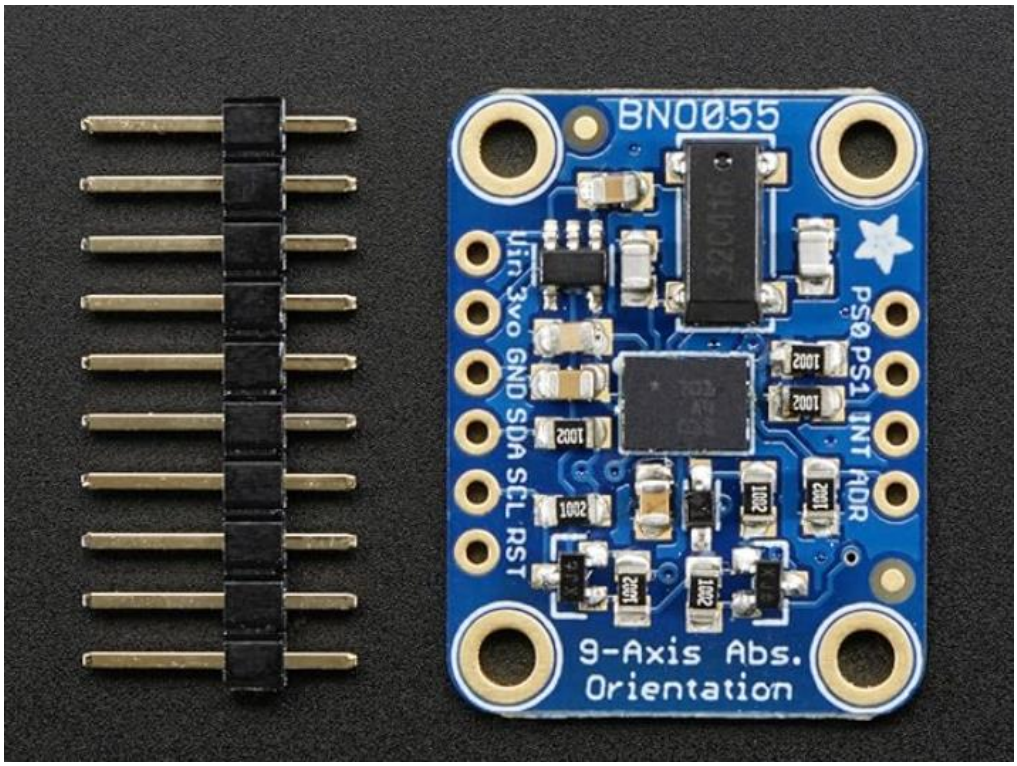

Figura 9: Sensor BNO055

4. Analisis del volante de inercia

Este apartado es fundamental en el documento, ya que abarca el desarrollo de la pieza clave del proyecto: el volante de inercia. Se expondrán diferentes versiones de volantes de inercia junto con sus características más significativas. Este apartado, comenzará con un análisis teórico del sistema para definir los parámetros más relevantes.

4.1. Análisis teórico del sistema.

La primera pregunta que surge es ¿Cómo mover un sistema de forma lineal?, la respuesta es bien sencilla, aplicando una fuerza. Según la segunda ley de Newton, esta fuerza tendrá la siguiente expresión.

$$a = \frac{F}{m}$$

Ecuación 1 [8]

Donde a es la aceleración, F es la fuerza y m la masa del sistema

Sin embargo, si el sistema debe girar en torno a un eje para mantener el equilibrio, debemos aplicar un momento o par que generará sobre el mismo una aceleración angular, en este caso la ecuación quedaría de la siguiente forma.

$$\boldsymbol{\alpha = \frac{M}{I}}$$

Ecuación 2 [9]

Donde α es la aceleración angular, M es el par o momento de inercia e I es la inercia del sistema.

Un péndulo invertido controlado por volante de inercia funciona de la siguiente manera. Cuando el sistema comienza a inclinarse se debe de aplicar el par necesario para devolverlo a su punto de equilibrio. Para esto, se usan motores conectados a volantes de inercia. Cuando un motor genera un par que acelera o frena el volante de inercia se genera un par de valor contrario sobre el motor y sobre el resto del sistema.

Este es un sistema bastante complejo, por lo que únicamente se aplicará un volante de inercia para mantener el equilibrio en un plano del espacio. Con lo

mencionado anteriormente, la ecuación del sistema sería la siguiente, donde $M_{volante}$ es el momento de inercia exclusivamente del volante y $M_{sistema}$ es el momento de inercia del sistema:

$$M_{volante} = M_{sistema}$$

Ecuación 3

$$\alpha_{Volante} * I_{volante} = \alpha_{sistema} * I_{sistema}$$

Ecuación 4 [10]

Esta ecuación, permite extraer varias conclusiones partiendo de que la inercia del sistema en un parámetro que desconocemos.

1) La aceleración angular del volante depende directamente del motor, ya que este es el encargado de aplicar el par necesario al volante en un sentido u en otro.
2) La inercia del volante es el parámetro de diseño, ya que, con un mismo motor, en función de la inercia, se consigue una respuesta u otra del sistema.
3) La aceleración angular del sistema es un parámetro que indica lo reactivo o lo rápido que es nuestro sistema ante perturbaciones. El objetivo principal es maximizar este término para tener un sistema reactivo.

Con todo esto procedemos al diseño del volante de inercia.

Proceso de diseño del volante de inercia

El primer paso es seleccionar la forma adecuada para el volante.

Como se ha mencionado anteriormente, con un mismo motor en este sistema, se puede variar la dinámica de este en función de la inercia del volante. Utilizando *Autodesk Inventor* podemos diseñar volantes de inercia y obtener la inercia.

Se ha de definir la forma del volante, utilizando las fórmulas de los momentos de inercia en función de las masas y de los radios [11], de un circulo macizo y de un anillo se obtienen los siguientes resultados:

$$I_{circulo\ macizo} = \frac{1}{2} * m * r^2$$

Ecuación 5

$$I_{anillo} = m * (r_2 - r_1)^2$$

Ecuación 6

Donde r_2 es el radio exterior y $r_1$es el radio interior.

Para obtener la misma inercia para ambos sistemas, se necesitaría más masa en el caso del circulo macizo que en el anillo. Esto es perjudicial para la inercia del sistema ($I_{sistema}$) ya que, al aumentar la masa, también aumenta la inercia del sistema lo cual provocaría una disminución en la aceleración angular del sistema($\alpha_{sistema}$).

Por otro lado, si el volante de inercia tuviese forma de anillo, se reduciría el peso del volante y la inercia del sistema, además de aumentar la inercia del volante. Considerando estos factores se ha decidido diseñar un volante con forma de anillo.

A continuación, se exponen las 3 versiones más significativas de los volantes de inercia, junto con sus criterios de diseño, las ventajas y desventajas de cada versión.

4.2. Volante con acople para eje liso:

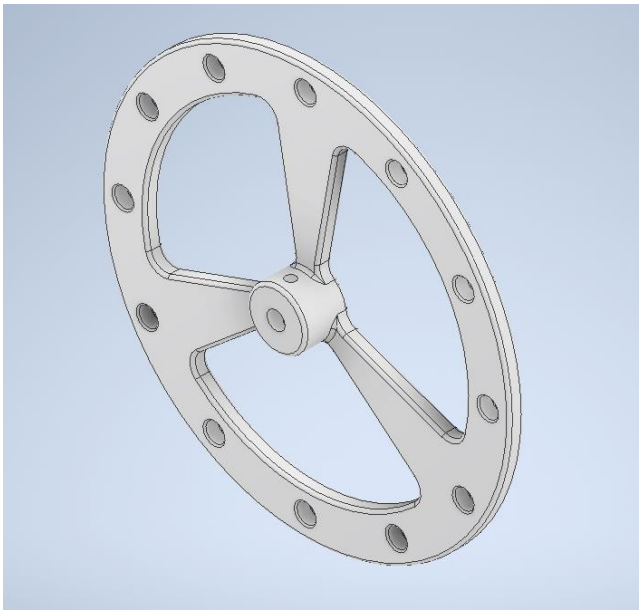

Figura 10: Volante con acople para eje liso

En la figura 10, se observa la primera versión del volante de inercia, que sirvió como toma de contacto para el diseño de los volantes, este diseño se realizó con los siguientes objetivos.

1) **Variabilidad de inercia:** Se diseñaron agujeros exteriores para albergar tornillos de métrica 6, permitiendo así variar la inercia del volante según las tuercas añadidas. Esto facilitaba las pruebas, ya que la inercia aportada por las tuercas era significativamente mayor que el peso añadido al volante.
2) **Simplicidad de montaje:** Del volante sale un cilindro en que se insertaba el eje del motor. Se perforó un agujero a este cilindro con el objetivo de albergar un tornillo prisionero para fijar el volante al eje del motor.

Sin embargo, este diseño se descartó completamente tras varias pruebas ya que fallaba el tornillo prisionero y el agujero donde se encontraba se rompía con el uso esto se veía en la respuesta del volante, ya que cambios bruscos en la aceleración o en el sentido provocaban que este patinase sobre el eje del motor y tardase un tiempo en igualar la nueva dirección y velocidad.

Tras estas conclusiones se ha tenido que rediseñar la zona de contacto entre el volante de inercia y el motor. Para ello se usó el mundo los coches teledirigidos de competición como inspiración, una práctica muy habitual en este mundo es el uso de bridas de eje rígido [22]para montar las ruedas en ejes lisos de motores, similar a nuestro caso. Se optó por un tipo de brida de eje rígido que se adapta al motor, como se muestra en la figura 11.

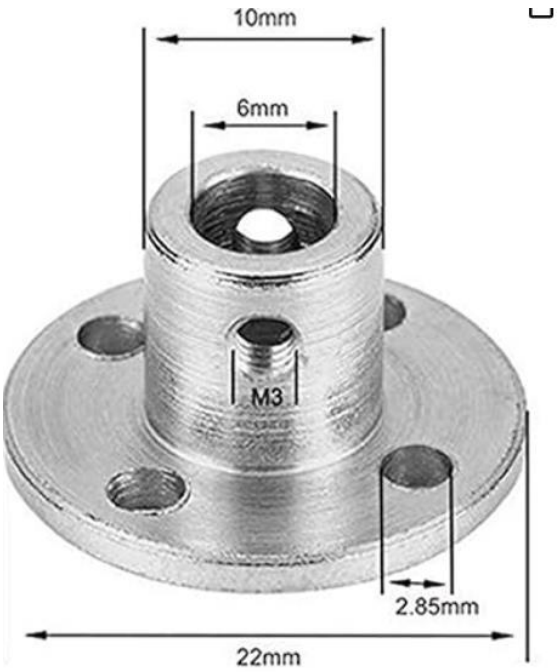

Figura 11: Brida de eje rígido para ejes lisos de 6mm de diámetro.

Utilizando este modelo de brida de eje rígido como inspiración, se ha diseñado un nuevo volante que permitiese atornillarse a esta brida, que se fija en el motor gracias a dos tornillos prisioneros independientes.

4.3. Volante sin acople para eje liso, tamaño M:

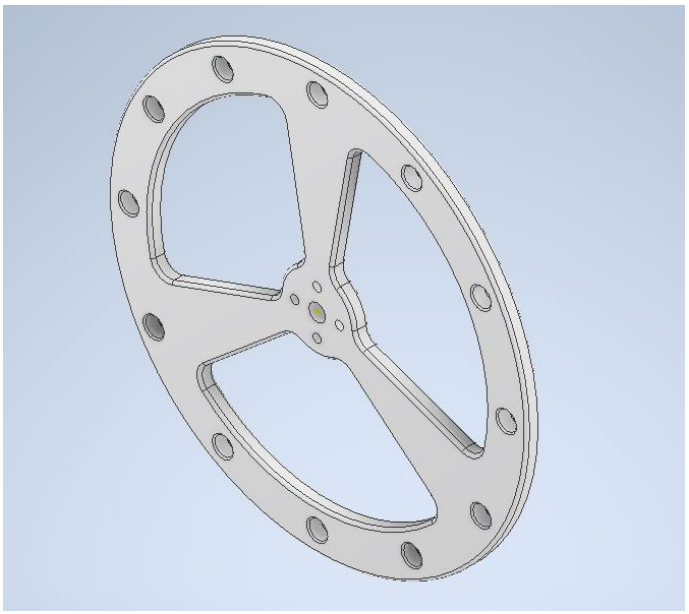

Figura 12: Volante sin acoplador, tamaño M

En la figura 12 se puede observar la nueva versión del volante de inercia la cuál mantiene los mismos radios del anillo que la versión 1, los cuales son. Diámetro exterior de 140mm y diámetro interior de 110mm. Sin embargo, se tuvo que aumentar el diámetro de zona interior a 27mm, esto con el objetivo de poder albergar los tornillos que fijan el volante a la brida y asegurar que la zona sea resistente. Este diseño cumplía los siguientes objetivos.

1) **Agarre y fiabilidad:** Gracias a la incorporación de la brida rígida, el volante y el motor funcionan completamente a la par creando un sistema más robusto y cohesionado.
2) **Variabilidad de inercia:** Se diseñaron agujeros exteriores para albergar tornillos de métrica 6, permitiendo así variar la inercia del volante según las tuercas añadidas. Esto facilitaba las pruebas, ya que la inercia aportada por las tuercas era significativamente mayor que el peso añadido al volante.

La inercia de este volante sin tornillos ni tuercas es el siguiente *Ixx=85,324 kg*mm^2*, ver figura 13.

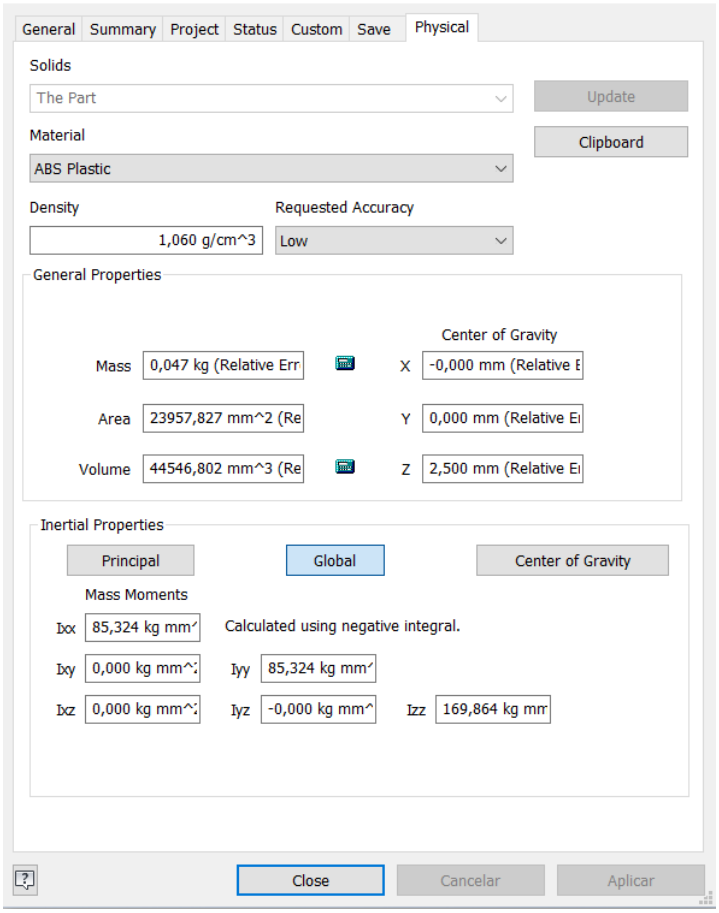

Figura 13: Inercia del volante sin acoplador, tamaño M

El mismo volante, pero con dos tuercas por tornillo tiene una inercia diferente, de *507,583 kg*mm^2,* como se muestra en la figura 14. Esta es inercia es la máxima que se puede obtener con este volante

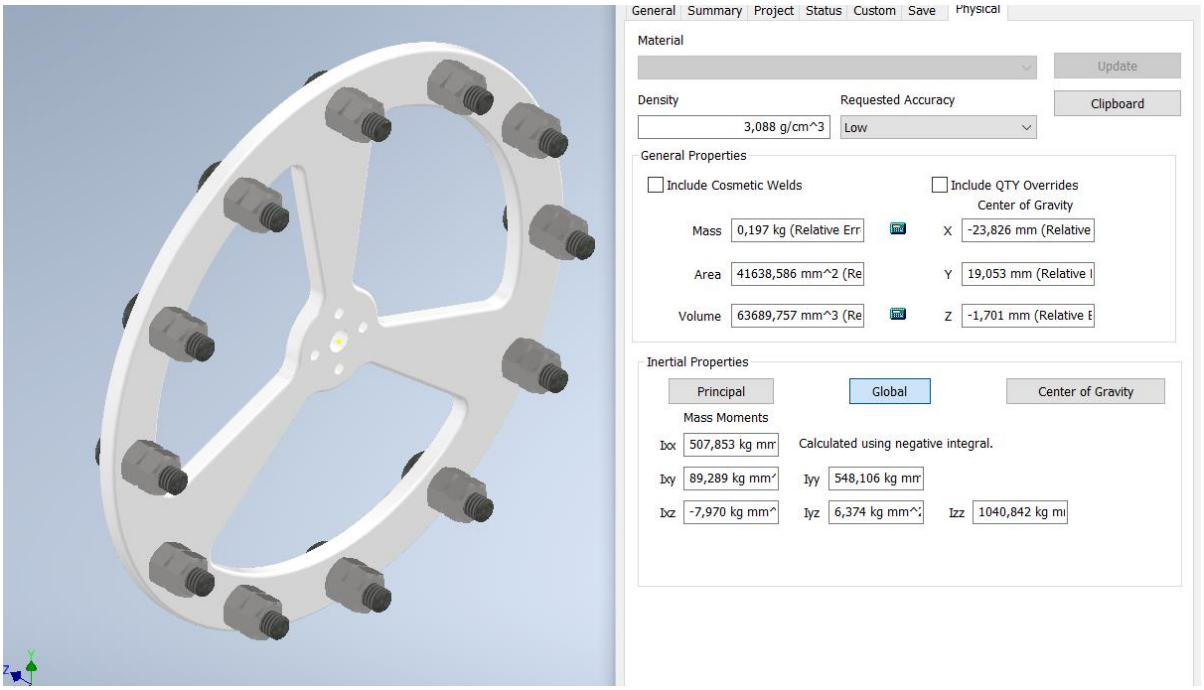

Figura 14: Inercia del volante con todos los tornillos y tuercas

4.4. Volante sin acople para eje liso, tamaño XL

La versión 3 es idéntica a la versión 2, pero con una diferencia crucial: los radios del volante de inercia han sido maximizados Estos son los radios más grandes que admite el volante dentro de la estructura en la que se encuentra. El radio exterior ahora es de 170 mm de diámetro y el radio interior de 140mm de diámetro. Este cambio afecta directamente a la inercia del sistema, que pasa de *85,324 kg*mm^2* a *130,351kg*mm^2*, reflejándose en una mejora sustancial en la inercia del sistema. La masa del volante, sin embargo, no se incrementa significativamente. Estos dos valores se pueden ver en la figura 15.

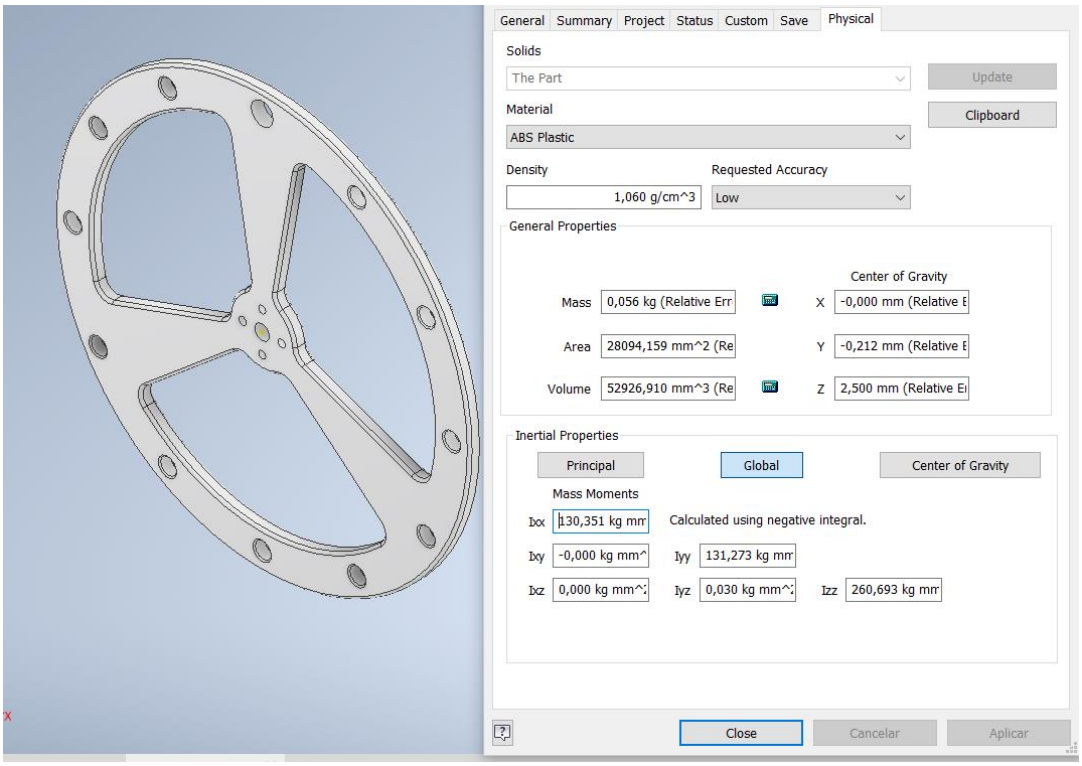

Figura 15: Inercia del volante sin acople, tamaño XL

El mismo volante, pero con dos tuercas por tornillo tiene una inercia diferente, como se muestra en la figura 16. Esta es la inercia es la máxima que se puede obtener para el volante de inercia y su valor es de *589,593 kg*mm^2* frente a la inercia de *507,583 kg*mm^2* de la versión 2.

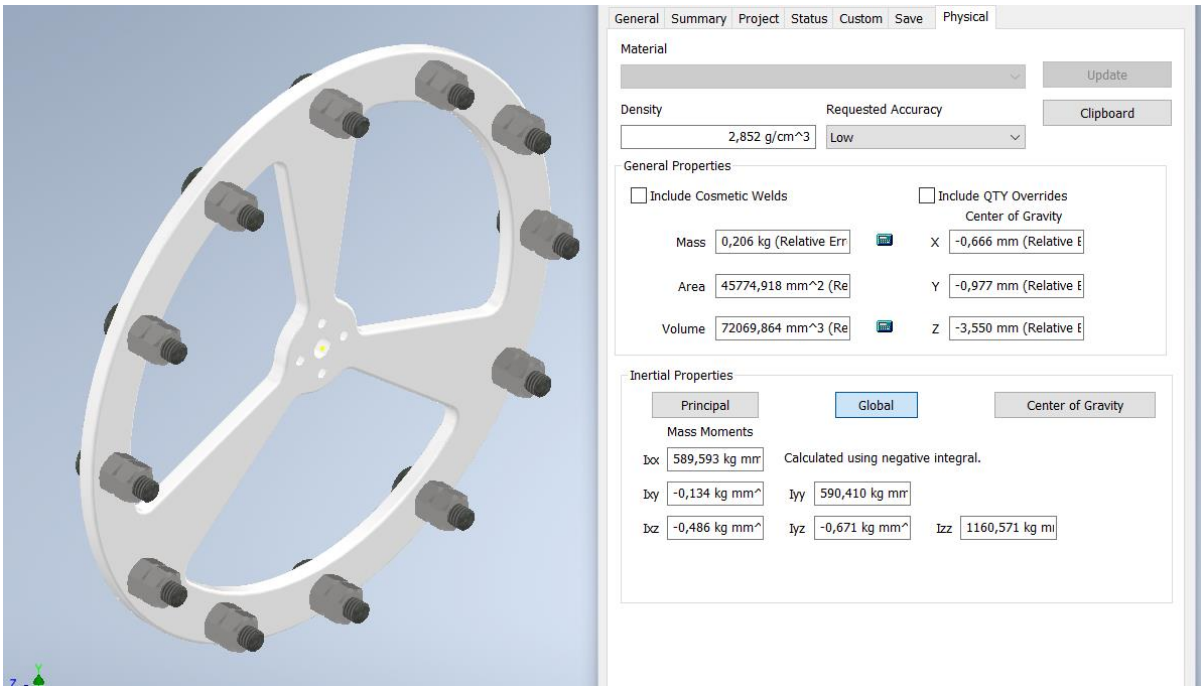

Figura 16: Inercias del volante sin acople y todos los tornillos y tuercas, tamaño XL

Esta versión será la utilizada para el montaje final del proyecto.

4.5. Comparativa de los volantes de inercia

A continuación, se adjunta una gráfica a modo de comparativa y de resumen, comparando las características de la versión 2 con y sin tornillos y las características de la versión 3 con y sin tornillos

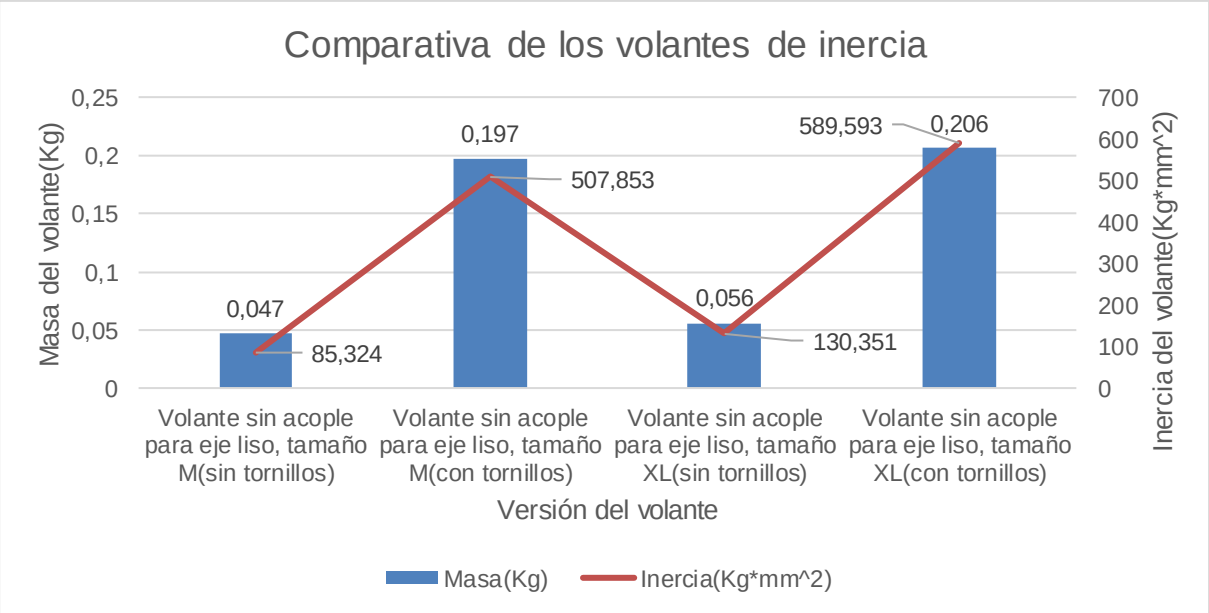

Figura 17: Comparativa del peso e inercia de los distintos volantes sin acoplador

5. Estructura del proyecto

Para la construcción de un péndulo invertido es necesario contar con los siguientes componentes, una base inestable para colocar el motor, el volante de inercia y el soporte para el ESP32 y el sensor. Una estructura que le permita a la base girar en sentido horario y en sentido antihorario, con todas estas necesidades se ha diseñado una base con las siguientes características.

5.1. Base del péndulo

En la figura 18, se puede observar la base donde se colocan el motor y el volante de inercia, tiene una forma similar a un triángulo evitando así que el sistema sea estable por naturaleza. El triángulo dispone de 6 agujeros para atornillar la base al motor, estos tornillos deben de ser de métrica 3 y de longitud 12mm. Además, cuenta con un agujero central de diámetro 6mm por el que pasa el eje del motor, por último, cuenta con tres agujeros exteriores de métrica 6 para pasar varillas roscadas, con el fin de hacer un sistema sólido al unir las demás partes.

Figura 18: Base del péndulo

De forma que, juntando la base, con el motor, la brida y el volante de inercia se obtiene la siguiente forma, como se muestra en la figura 19. (Es importante que en la realidad se apliquen los tornillos pertinentes para unir los componentes)

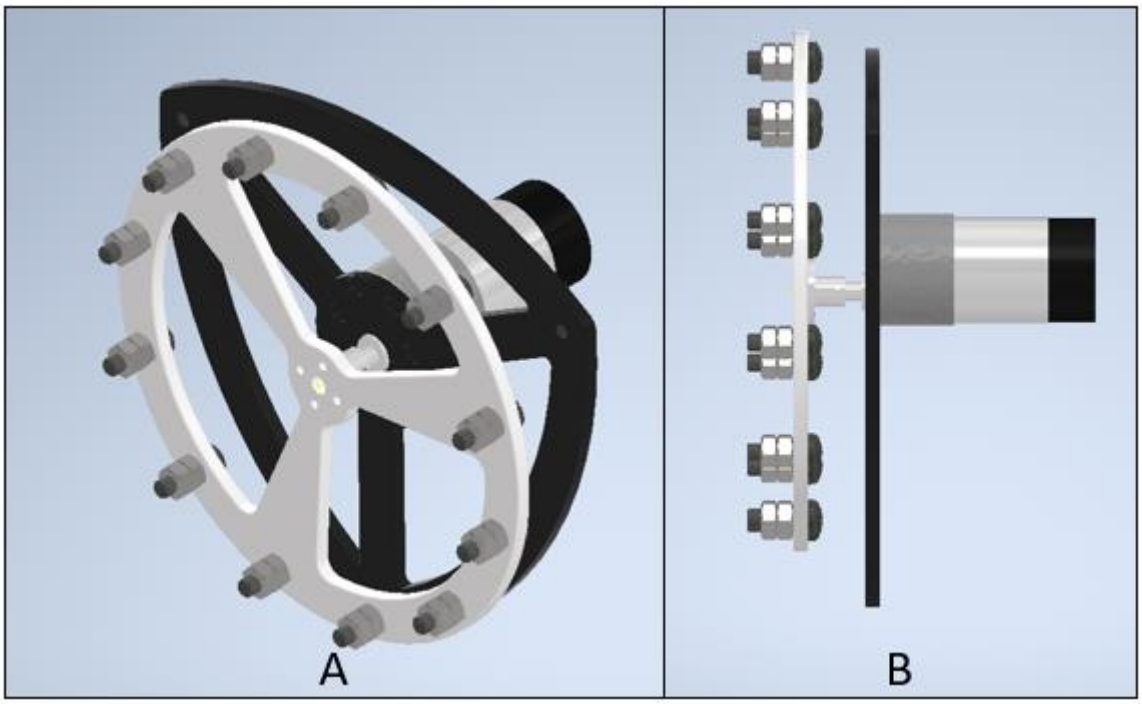

Figura 19:Montaje. A Vista del sistema. B vista de perfil

Como se puede observar en la figura de arriba, el sistema no sería capaz de mantenerse vertical debido a la falta de apoyos, por eso añadimos una base al final y usamos las varillas roscadas de métrica 6 y longitud 160mm para hacer del conjunto una pieza única, como se puede ver en la figura 20.

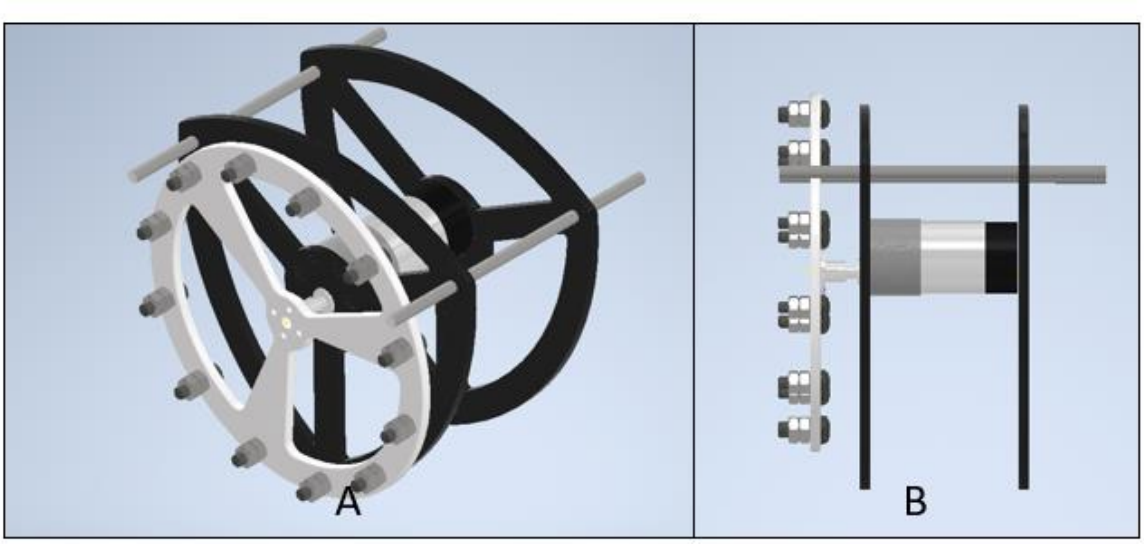

Figura 20: Montaje 2. A, Vista 3D del sistema. B vista de perfil

Como se puede observar en la figura de arriba, hay un agujero en la base que queda libre, por este agujero pasará un eje liso de 6mm de diámetro y 100 mm de longitud, el cual permitirá al sistema rotar en torno a este eje.

5.2. Estructura para la base del péndulo

Se ha diseñado la siguiente estructura, usando un listón de madera de 400x53x34mm el cuál esta atornillado a una tabla de madera de 400x200x18mm mediante 4 tornillos de avellanado plano para madera de métrica 6 y 60mm de longitud Al listón se le ha hecho un agujero de 6 mm de diámetro en el centro y a una altura de 22mm desde donde apoya el listón, por este agujero pasará el eje liso. Como se puede ver en la figura 21.

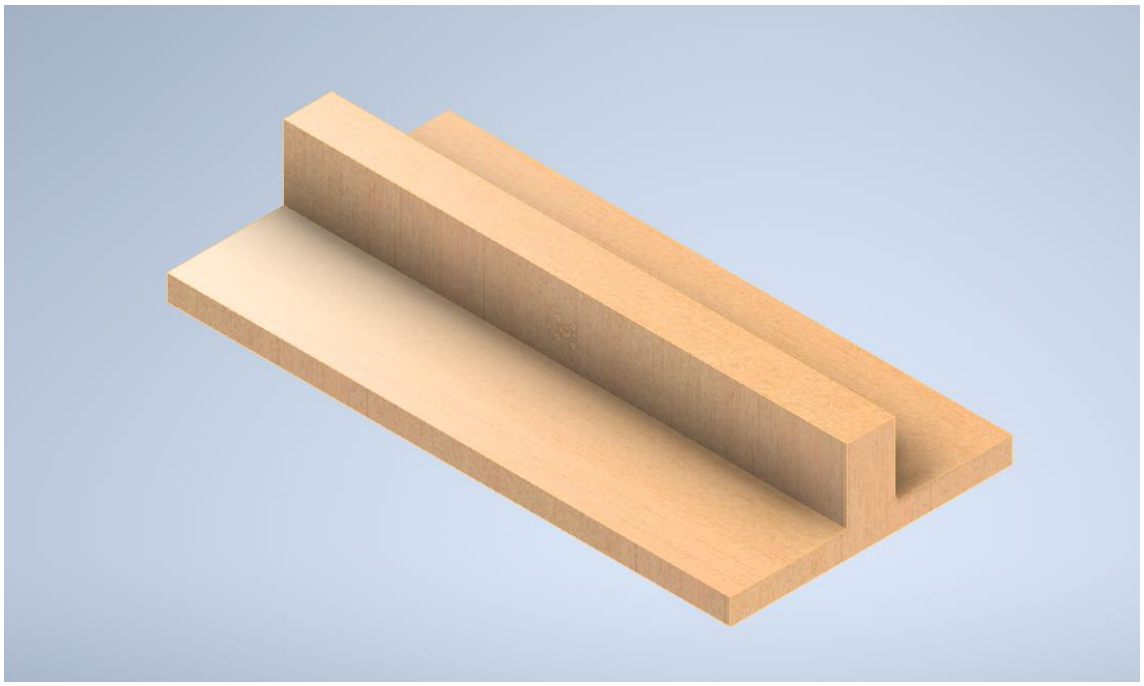

Figura 21: Estructura incompleta

Es crucial que el eje se mantenga fijo en su posición para así evitar posibles fallos debido al diseño de la estructura. Aprovechando que se dispone de 4 bridas de eje rígido (una se usa para fijar el volante al motor), se emplean 2 bridas para mantener el eje liso en su posición. EL montaje resultante se puede ver en la figura 22 Para atornillar las bridas a la madera, se necesitan 8 tornillos de cabeza plana avellanada para madera de métrica 3 y longitud 20mm.

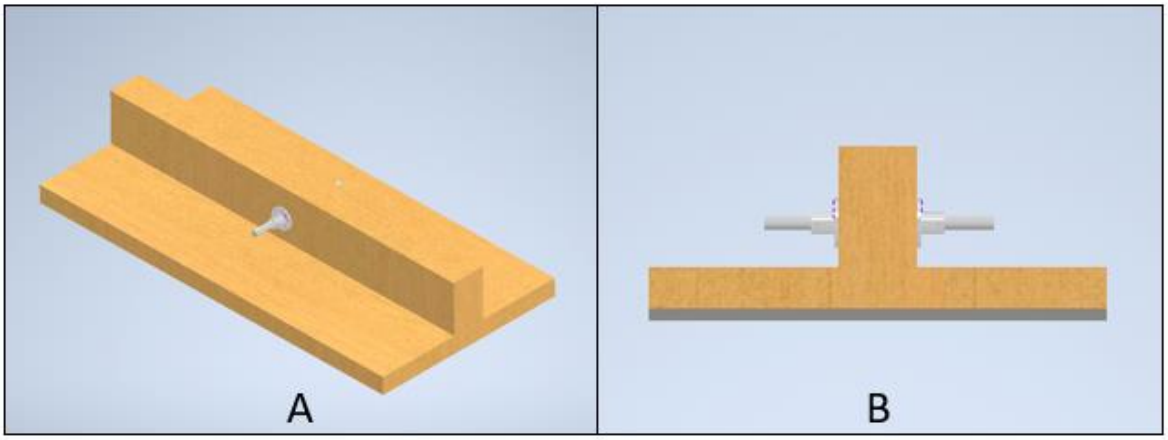

Figura 22: Estructura completa. A vista 3D, B vista de perfil

Por último, se diseña una base, como se puede ver en la figura 23, donde colocar la placa protoboard en la que están el ESP32 y el sensor.

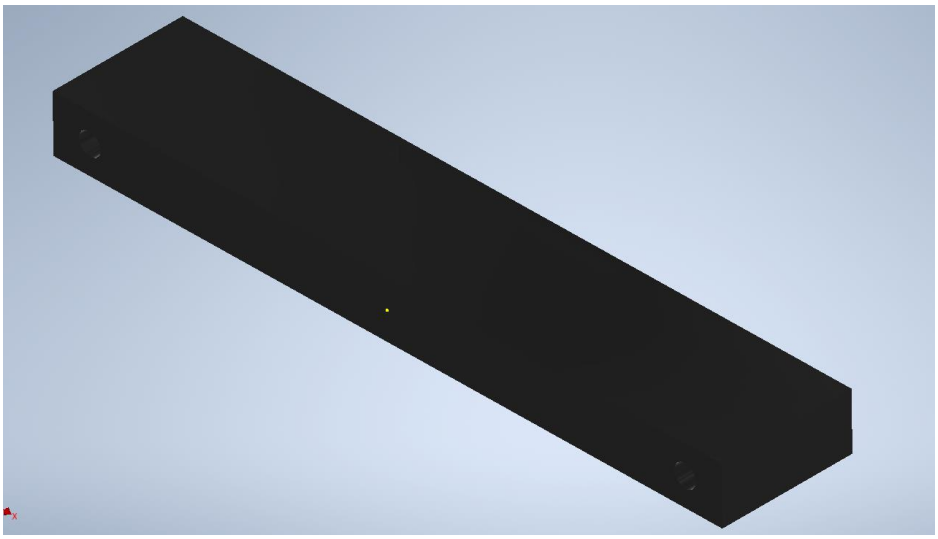

Figura 23: Soporte sensor y controladora

Con todo esto el montaje final en *Autodesk Inventor* quedaría de la siguiente forma, ver figura a continuación. (Es importante que en la realidad se apliquen los tornillos y tuercas pertinentes para unir los componentes)

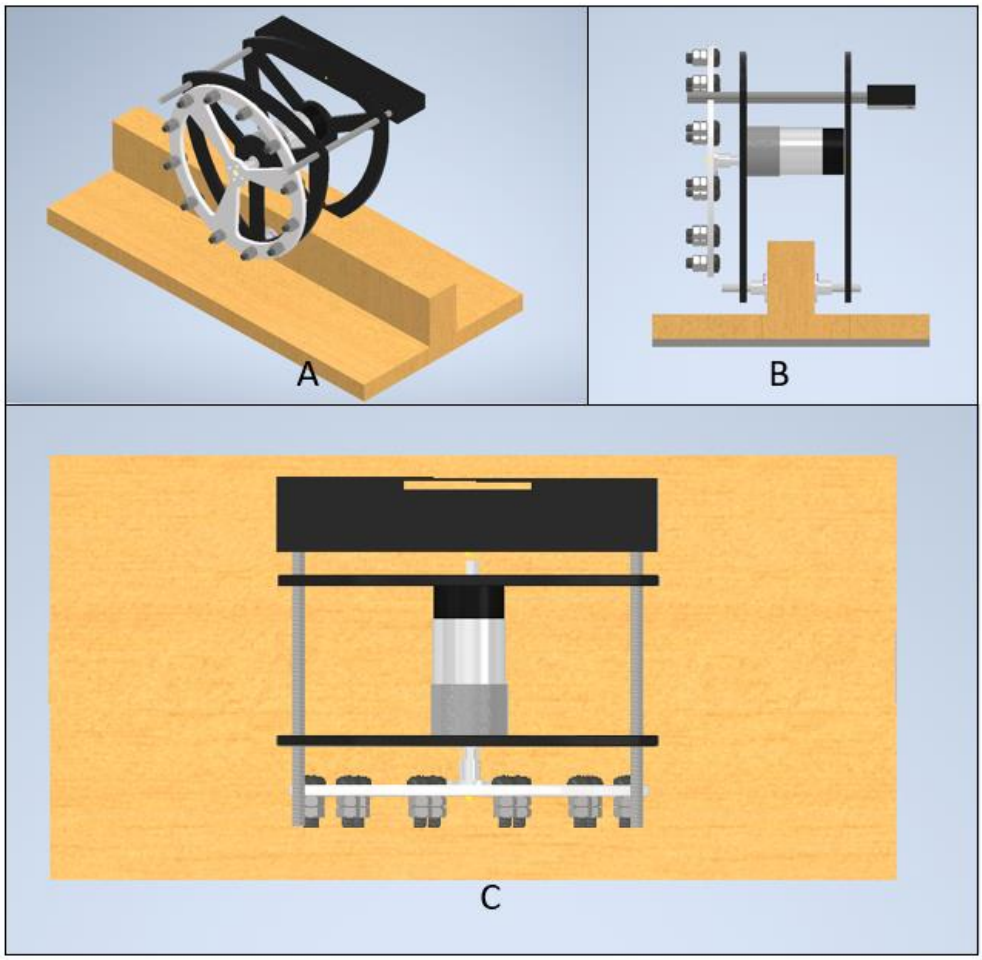

Figura 24: Montaje completo del sistema. A vista 3D. B vista perfil derecho. C vista de planta.

Y este es el resultado en la realidad, ver figura a continuación, con todas las tuercas y tornillos necesarios:

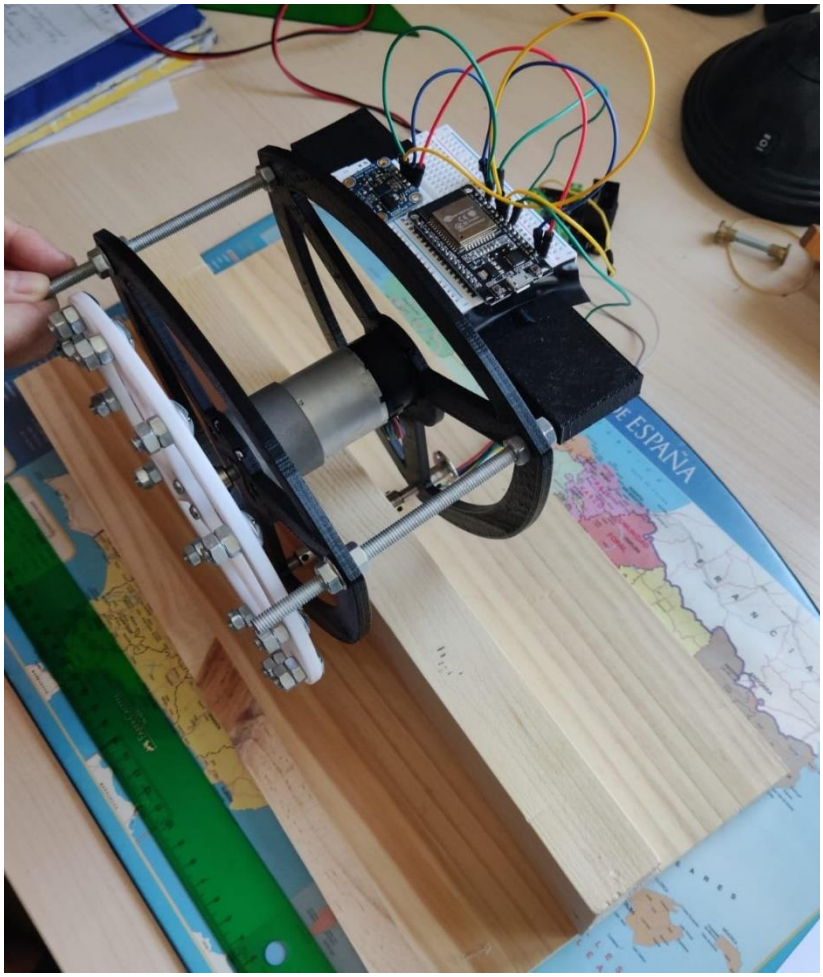

Figura 25: Montaje completo del sistema en la realidad.

6. Software de la controladora

Los requisitos principales que debe cumplir el ESP32(la controladora) son, el control continuo del sistema y la comunicación con otros dispositivos. Para cumplir estos requisitos de la mejor forma posible y aprovechar al máximo las capacidades del ESP32, se opta por utilizar *freeRTOS*, una práctica muy común en sistemas *IoT*. A continuación, se explicará el *freeRTOS* y los códigos que se ejecutan en cada núcleo.

Se definen las tareas de cada núcleo

- **Núcleo 0:** Encargado de la gestión Wifi y del protocolo de comunicación Modbus.
  Estas tareas se ejecutan en el núcleo 0.
- **Núcleo 1**: Encargado del control del dispositivo.
  Esta tarea se ejecuta en el núcleo 1.

6.1. freeRTOS

6.1.1. Introducción al freeRTOS

FreeRTOS es un sistema operativo en tiempo real kernel que, usado en dispositivos embebidos, el cual permite asignar tareas a cada núcleo del microcontrolador. En este caso el ESP32 dispone únicamente de dos núcleos lo cual es suficiente para las necesidades de control y comunicación. [12]

Lo normal en los ESP32 es que el *freeRTOS* ya venga instalado y listo para usarse por lo que no hace falta añadir ninguna librería.

6.1.2. Configurar freeRTOS

PASO 1: El primer paso consiste en abrir *VScode*, descargar la extensión de *PlatformIO*, seleccionar la placa *DOIT ESP32 DEVKIT V1* y abrir un programa usando el entorno de Arduino. Una vez ya se tenga esto, se debe de comprobar en que núcleo se está ejecutando la función “void loop()”, como se muestra en la figura 26.

```
#include <Arduino.h>

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

}

void loop(){

  Serial.print("Loop ejecutandose en el core: ");
  Serial.println(String(xPortGetCoreID()));

}
```

Figura 26: Núcleo de ejecución de la función void loop()

Esta instrucción obtendrá en que núcleo se está ejecutando e imprimirá por el puerto serie el número del núcleo que la ejecuta. Por defecto, el "loop", se ejecuta en el núcleo 1, por lo que se muestra lo siguiente por el puerto serie. [13]

Loop ejecutándose en el core: 1

PASO 2: Una vez completado el paso 1, se estará listo para empezar con el *freeRTOS*. Lo siguiente es declarar la tarea que ejecutará junto con la función que se va a ejecutar en la tarea, ver figura 27.

```
#include <Arduino.h>

TaskHandle_t Task0;//Declaramos una nueva tarea
void loop2(void *parameter );
// Funcion que se ejecuta en la Task0

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

}

void loop(){

  Serial.print("Loop ejecutandose en el core: ");
  Serial.println(String(xPortGetCoreID()));

}
void loop2(void *parameter){

}
```

Figura 27: Definición de la tarea

Paso 3: Dentro de la función de la tarea es necesario crear un bucle infinito para esta se ejecute constantemente. Como se puede ver en la figura 28.

```
#include <Arduino.h>

TaskHandle_t Task0;//Declaramos una nueva tarea
void loop2(void *parameter );
// Funcion que se ejecuta en la Task0

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

}

void loop(){

  Serial.print("Loop ejecutandose en el core: ");
  Serial.println(String(xPortGetCoreID()));

}
void loop2(void *parameter){
  for( ;; ){
    //Añadir el codigo que se va a repetir de manera
indefinida
  }

}
```

Figura 28: Creación de la función y del bucle infinito para la nueva tarea

Paso 4: En este punto las tareas apuntan al mismo núcleo, por lo tanto, se debe configurar la *Task0* para que apunte al núcleo cero, mientras que el *loop ()* se ejecutará en el núcleo 1. Como se puede observar en la figura 29.

```
#include <Arduino.h>

TaskHandle_t Task0;//Declaramos una nueva tarea
void loop2(void *parameter );
// Funcion que se ejecuta en la Task0


void setup(){
  Serial.begin(115200);
   xTaskCreatePinnedToCore(
    loop2,/*Función que  se ejecuta en la tarea*/
    "Task_0",/*Nombre de la tarea*/
    4096,/*Tamaño de la pila de la tarea*/
    NULL,
    1,/*Prioridad de la tarea, siendo 0 la menor*/
    &Task0,
    0/*Asignamos la tarea al núcleo 0*/
  );

}

void loop(){

  Serial.print("Loop ejecutandose en el core: ");
  Serial.println(String(xPortGetCoreID()));

}
void loop2(void *parameter){
  for( ;; ){
    //Añadir el codigo que se va a repetir de manera
indefinida
  }

}
```

Figura 29: Apuntar la tarea a un núcleo en concreto

El tamaño de la pila debe ajustarse experimentalmente, en este caso el tamaño de la pila que se puede ver en la figura superior, es más que suficiente para el código que se ejecutará y que se explicará en el siguiente apartado. Para comprobar la cantidad de pila que no está siendo utilizada, se puede usar la instrucción que se muestra en la figura 30 El tamaño máximo de la pila es de 10000. [13]

```
UBaseType_t water_mark=uxTaskGetStackHighWaterMark(NULL
);
    Serial.println(water_mark);
```

Figura 30: Visualizar el tamaño restante de la pila [14]

Paso 5: Por último, se deben hacer unas series de modificaciones para dejar el código lo más claro y eficiente posible. De forma que quedaría como en la figura 31

```
#include <Arduino.h>

TaskHandle_t Task0;//Declaramos una nueva tarea
void loop2(void *parameter );
// Funcion que se ejecuta en la Task0


void setup(){
  Serial.begin(115200);
   xTaskCreatePinnedToCore(
    loop2,/*Función que  se ejecuta en la tarea*/
    "Task_0",/*Nombre de la tarea*/
    4096,/*Tamaño de la pila de la tarea*/
    NULL,
    1,/*Prioridad de la tarea, siendo 0 la menor*/
    &Task0,
    0/*Asignamos la tarea al núcleo 0*/
  );

}

void loop(){

  Serial.print("Loop ejecutandose en el core: ");
  Serial.println(String(xPortGetCoreID()));
  for( ;; ){
    //Añadir el codigo que se va a repetir de manera
indefinida

    vTaskDelay(pdMS_TO_TICKS(1));
  }

}
void loop2(void *parameter){
  Serial.print("Loop2 ejecutandose en el core: ");
  Serial.println(String(xPortGetCoreID()));
  for( ;; ){
    //Añadir el codigo que se va a repetir de manera
indefinida

    vTaskDelay(pdMS_TO_TICKS(1));
  }

}
```

Figura 31: Código completo de freeRTOS

Con estos ajustes, cada vez que se ejecute el programa, se mostrará por el puerto serie que núcleo está ejecutando cada tarea. Es importante añadir dentro de cada bucle infinito la siguiente instrucción:

vTaskDelay (pdMS_TO_TICKS (1)); [15]

Esta instrucción es de vital importancia para el *freeRTOS*, ya que actúa como una pausa que permite al sistema operativo comprobar si hay más tareas que deben de ejecutarse en el mismo núcleo, en cuanto haga la pausa y compruebe que no ha mas tareas a ejecutar, retomara la tarea que estaba realizando. Este proceso lo hace casi instantáneo ya que esta comprobación requiere de muy pocos ciclos de reloj.

La instrucción mencionada con anterioridad convierte en valor de un milisegundo a pulsos del reloj del sistema, proporcionando la pausa más pequeña posible admitida por *freeRTOS*. Esto permite una espera extremadamente precisa, asegurando un funcionamiento eficiente del sistema.

6.2. Comunicaciones controladora-ordenador

Como se mencionó anteriormente, en este núcleo se ejecutará la tarea encargada de la gestión de la Wifi y del protocolo de comunicación Modbus. Primero se explicará el código encargado de la gestión Wifi y después se explicará el código para el protocolo Modbus.

6.2.1. Código Wifi para la controladora

El primer paso será configurar el ESP32 para que sea capaz de conectarse a una Wifi y, en el caso de que pierda la conexión, que intente conectarse de nuevo. Para ello se deberán de añadir las líneas que se ven en las figuras 32, 33, 34 y 35.

```
#include <Arduino.h>
#include <WiFi.h>

const char* ssid= "Wifi_name";
const char* password= "Wifi_passwaord";

TaskHandle_t Task0;//Declaramos una nueva tarea
void loop2(void *parameter );
// Funcion que se ejecuta en la Task0
void reconnectWiFi();
//Funcion que se ejecuta cuando perdemos la conexion
wifi
```

Figura 32: Definición parámetros Wifi

```
void setup(){
  Serial.begin(115200);
   xTaskCreatePinnedToCore(
    loop2,/*Función que  se ejecuta en la tarea*/
    "Task_0",/*Nombre de la tarea*/
    4096,/*Tamaño de la pila de la tarea*/
    NULL,
    1,/*Prioridad de la tarea, siendo 0 la menor*/
    &Task0,
    0/*Asignamos la tarea al núcleo 0*/
  );

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);//Iniciamos la conexión
  Serial.print("Conectando a:\t");
  Serial.println(ssid);

  // Esperar a que nos conectemos
  while (WiFi.status() != WL_CONNECTED)
 //Esperar a que nos conectemos
  {
    delay(200);//Nos conectamos a la red
  Serial.print('.');
  }

  // Mostrar mensaje de exito y dirección IP asignada
  Serial.println();
  Serial.print("Conectado a:\t");
  Serial.println(WiFi.SSID());
  Serial.print("IP address:\t");
  Serial.println(WiFi.localIP());

}
```

Figura 33: Configuración del ESP32 para conexión Wifi_STA

El valor de la IP es muy importante, ya que nos hará falta para la aplicación en Processing

```
void loop2(void *parameter){
  Serial.print(
"Comunicaciones ejecutandose en el core: ");
  Serial.println(String(xPortGetCoreID()));
  for( ;; ){
    if (WiFi.status() != WL_CONNECTED) {
//En el caso de perdida de conexion, nos conectamos a
la wifi
      reconnectWiFi();
    }
    vTaskDelay(pdMS_TO_TICKS(1));
  }

}
```

Figura 34: Reconexión en caso de pérdida de conexión Wifi

Y definimos la siguiente función.

```
void reconnectWiFi(){
  Serial.println(
"Perdida de conexión, intentando reconectar...");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
      vTaskDelay(pdMS_TO_TICKS(200));
      // Tambien podemos usar: delay(pdMS_TO_TICKS(20
0));
      Serial.print(".");
    }
    Serial.println();
    Serial.println("Conexión establecida");
}
```

Figura 35: Función que se ejecuta en caso de perder la conexión

Tras esta configuración, el ESP32 será capaz de conectarse la red wifi deseada y en el caso de perder la conexión, se conectará automáticamente [16] Durante este proceso de reconexión, el ESP32 no ejecutará el resto del código asignado a este núcleo, evitando así posibles conflictos con el protocolo Modbus.

6.2.2. Código Modbus para la controladora

¿Qué es Modbus?:

Modbus es un protocolo de comunicación de uso libre que puede ser implementado tanto por fabricante como por usuarios individuales. Fue diseñado en 1970 por Modicon como una forma de conectar PLCs (Programmable Logic Controller) utilizando el concepto de maestro/esclavo. Hoy en día fabricantes como Siemens, Allen-Bradley, Mitsubishi y Omron, permiten que sus PLCs utilicen este protocolo, ya que resuelve el problema de comunicación entre dispositivos de diferentes fabricantes, siempre que estos dispositivos sean compatibles con Modbus.

Modbus cuenta con varias áreas de memoria

- **Discrete input:** Almacena valores de entrada digitales (LOW o HIGH).
- **Coil output:** Almacena valores de salida digitales (LOW o HIGH).
- **Input register:** Almacena valores de entrada analógicos, como los valores de sensores de temperatura, humedad, posición, etc.
- **Holding register.** Almacena valores de salida analógicos, que pueden ser usados para la velocidad de un motor, la temperatura, etc. [17].

Tipos de Modbus.

- **Modbus RTU (Remote Terminal Unit):** Es el tipo más utilizado ya que emplea código binario y CRC (Cyclic Redundancy Check) para detección de errores, usa el concepto maestro-esclavo. Dentro de este tipo podemos encontrar más variantes en función de la conexión, estas pueden ser: RS232, RS422 y RS485.
- **Modbus TCP/IP (Transmission Control / Internet Protocol):** Este protocolo usa la arquitectura servidor-cliente, lo que permite que varios dispositivos actúen como servidores como clientes. Para comunicar los dispositivos, se pueden usar cables de Ethernet o una red Wifi, y cada dispositivo deberá de tener una IP única. Como se puede ver en el esquema que se ve en la figura 36. [17].

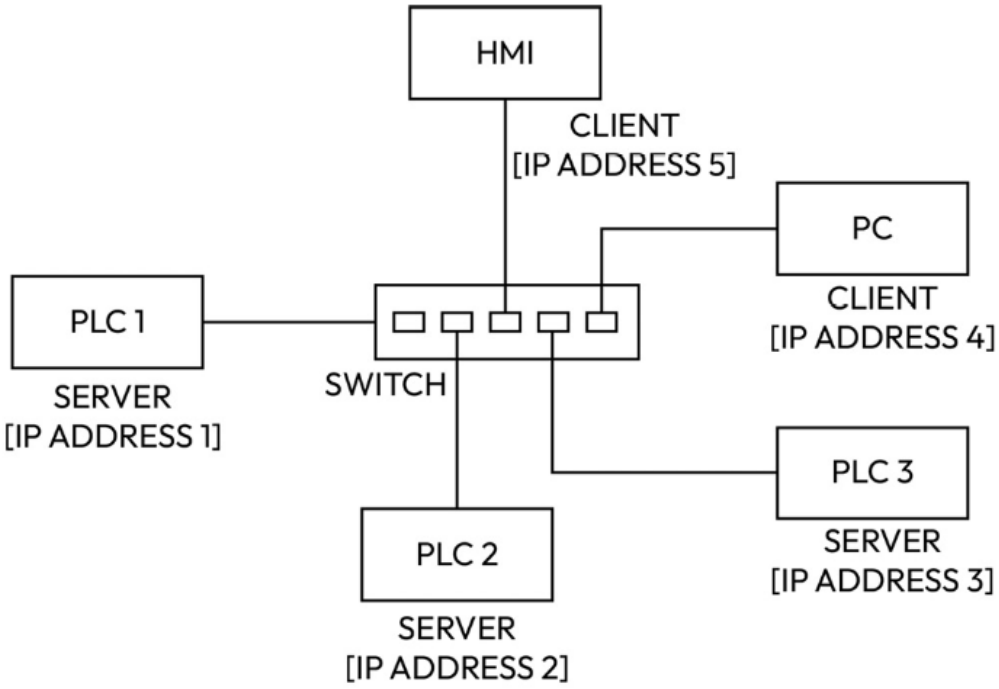

Figura 36:Esquema de Modbus TCP/IP. [17]

ESP32y Modbus

El objetivo de este código es poder procesar la información que le llega desde el cliente (el ordenador) y configurar el ESP32 como servidor, para así poder apagar o encender el motor y editar las constantes del PID.

Paso 1: Para poder usar Modbus en el ESP32, es necesario añadir la librería que se muestra en la figura 37. [18]

```
#include <ModbusIP_ESP8266.h>
```

Figura 37: Librería Modbus para la controladora

Paso 2: Una vez instalada la librería, de debe crear el objeto de la librería y definir los registros que vamos a usar, tal y como se puede ver en la figura 38.

```
ModbusIP mb;//Objeto de la librería
const int func_mode=0;// Registro 0, de tipo coil
const int sensor_value=0;// REgistro 0 de tipo Ireg
const int kp_value=0;// Registro 0 de tipo Hreg
const int kp_factor=1;// Registro 1 de tipo Hreg
const int ki_value=2;// Registro 2 de tipo Hreg
const int ki_factor=3;// Registro 3 de tipo Hreg
const int kd_value=4;// Registro 4 de tipo Hreg
const int kd_factor=5;// Registro 5 de tipo Hreg
```

Figura 38: Definición de los registros.

Como se puede ver en la figura anterior, hay registros con el nombre “xx_value” y “xx_factor”. Esto se debe a que la librería de Modbus admite únicamente números enteros de hasta 16 bits. Si se desea escribir un número con decimales desde la aplicación en el ordenador, es necesario tener un registro con el número multiplicado por un factor (1,10,100, 1000, …) para eliminar los decimales y otro registro con ese factor. De esta forma, en el ESP32, al dividir el número entre el factor, se obtiene el número original que introdujo el usuario en el ordenador. Esto se calcula en la HMI.

El registro “func_mode” es el encargado de apagar o encender el sistema. Si “func_mode” es 0, el motor y el PID se apagarán permitiendo escribir en los registros de tipo Hreg y editar las constantes del PID. Pero si “func_mode” es 1, no se podrán editar los registros de tipo Hreg y se activará el motor y el PID.

En el registro “sensor_value”, se guarda el valor del sensor para poder leerlo en el ordenador.

Paso 3: Dentro de la función “void setup ()” y al final de la misma, se debe añadir la siguiente función (ver figura 39), que llama a la función que crea los registros (ver figura 40).

```
setup_modbus();
```

Figura 39: Función que crea los registros

```
void setup_modbus(){
  mb.server();//Usamos el esp32 como servidor
  mb.addCoil(func_mode);
//Creamos el registro 0 de tipo Coil
  mb.Coil(func_mode,0);
// Ponemos a 0 el registro 0 de tipo COil
  mb.addIreg(sensor_value);
//Creamos el registro 0 de tipo Input Regsiter
  mb.addHreg(kp_value);
//Creamos el registro 0 de tipo Holding Register
  mb.addHreg(kp_factor);
//Creamos el registro 1 de tipo Holding Register
  mb.addHreg(ki_value);
//Creamos el registro 2 de tipo Holding Register
  mb.addHreg(ki_factor);
//Creamos el registro 3 de tipo Holding Register
  mb.addHreg(kd_value);
 //Creamos el registro 4 de tipo Holding Register
  mb.addHreg(kd_factor);
//Creamos el registro 5 de tipo Holding Register
}
```

Figura 40: Creación de los registros en función del área de memoria. [18].

Ahora se está en disposición de leer y escribir en los registros.

En función del tipo de registro que se desee leer, en el ESP32, se deben utilizar algunas de las siguientes instrucciones:

X=mb.Coil(Num_registro);

X= mb.Ireg(Num_registro);

X= mb.Hreg(Num_registro);

En función del registro que se desee escribir, en el ESP32, se debe utilizar alguna de las siguientes instrucciones. Es importante tener cuidado con el tipo de dato que admite cada área de memoria:

mb.Ireg(Num_registro,valor);

mb.Coil(Num_registro,valor);

mb.Hreg(Num_registro,valor);

Paso 4: Para optimizar al máximo código del núcleo 0, es necesario minimizar la cantidad de veces que se lee o se escribe en un registro. Para ello deben definirse unas variables auxiliares con las que se comprobará el valor del número sin decimales. Estas variables se definirán antes del bucle infinito que se ejecuta en el "loop2" y deberán ser de tipo "static" para mantener su valor entre ejecuciones. Las variables definidas se pueden ver en la figura 41.

```
void loop2(void *parameter){
  Serial.print("Modbus ejecutandose en el core: ");
  Serial.println(String(xPortGetCoreID()));

  static int kp_changed=0;
  static int ki_changed=0;
  static int kd_changed=0;
  static int kp_changed_current=0;
  static int ki_changed_current=0;
  static int kd_changed_current=0;
  boolean on_off_2=false;

  for( ;; ){
```

Figura 41: Definición de variables auxiliares en el núcleo 0

Estas variables auxiliares permiten almacenar temporalmente los valores leídos de los registros Modbus, evitando así múltiples accesos innecesarios a los registros

dentro del bucle infinito, mejorando de esta manera la eficiencia y el rendimiento del sistema.

Paso 5: Dentro del bucle infinito, se debe comprobar el valor de los registros periódicamente, antes de esto, es de vital importancia añadir la función

mb.task()

Esta función gestiona todas las solicitudes de lectura y escritura de los registros, es una función proporcionada por la librería. A continuación, se muestra con un flujograma cómo debería quedar el código que se ejecuta en el núcleo 0, encargado de cambiar el estado del motor y editar las constantes del PID (ver figura 42)

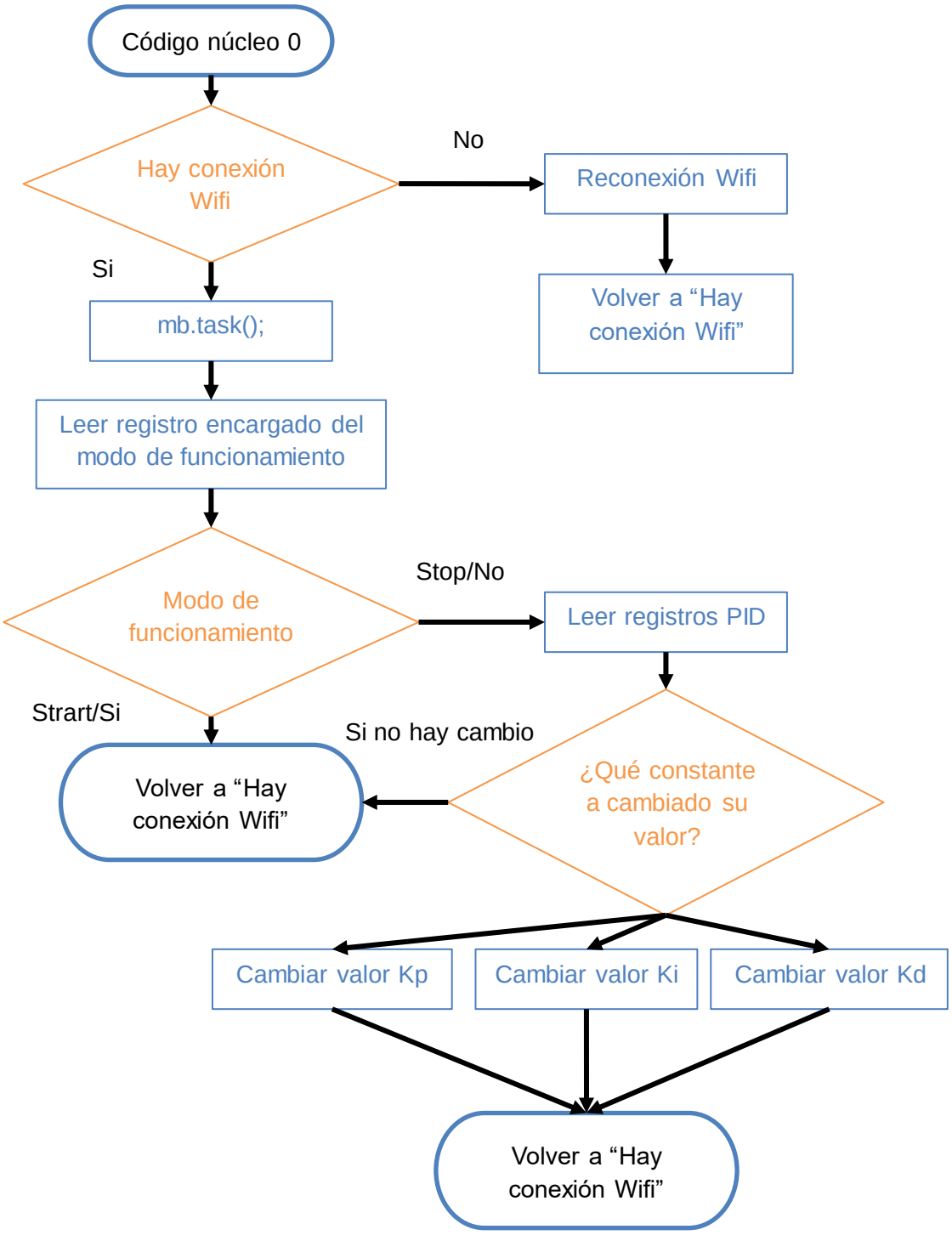

Figura 42: Flujograma, código núcleo 0

Con estos pasos, el ESP32 será capaz de conectarse a la red, gestionar solicitudes Modbus [18] y controlar el motor y las constantes del PID de manera eficiente. La implementación de estas funciones garantiza una comunicación robusta y un control preciso del sistema.

6.3. Control PID

Para la estrategia de control de la planta, se ha programado un PID, como se puede ver figura 43, que se ejecuta en el núcleo 1. Este PID recibe como argumento de entrada el valor del sensor y como salida varía el ciclo de trabajo de una señal PWM de resolución 8 bits y una frecuencia de 20000Hz. Además de variar el ciclo de trabajo también varía el signo, lo que indica al actuador si girar en sentido horario o antihorario.

```
void update_PID(double sens_value){
  //Definimos las variables locales y le ponemos static pa
ra no perder el valor
  static double error=0.0;
  static double error_old=0.0;
  static double proporcional=0.0;
  static double integral=0.0;
  static double derivativo=0.0;
  static double dt_PID;
  static unsigned long temp_old_PID=0;
  static unsigned long temp_new_PID=0;
  static boolean direc=0;

  temp_new_PID=micros();

  dt_PID=(temp_new_PID-temp_old_PID)/pasar_a_seg;
  error=refe-sens_value;
  //PID, la accion integral y derivativa por el tiempo por
el tiempo transcurrido entre que se ejecuta y la ejecucion
anterior
  proporcional=kp*error;
  integral +=ki*error*dt_PID;
  //Limitamos la acción integral, entre los valores maimos
con signo de la señal pwm, para que no crezca infinito. El
signo indica la dirección;
  integral = constrain(integral, -255, 255);

  derivativo=kd*(error-error_old)/dt_PID;

  duty_cycle=proporcional+integral+derivativo;

  error_old=error;

  temp_old_PID=temp_new_PID;
  direc = (duty_cycle >= 0) ? 0 : 1;
//Ajustar esto en caso de error
  duty_cycle = constrain(abs(duty_cycle), 0, 255);
//Constrain, limita el valor entre 0 y 255

  digitalWrite(pin_dir,direc);
  ledcWrite(pwmChannel,duty_cycle);

}
```

Figura 43: Código PID, ejecutándose en el núcleo 1

Para conseguir la resolución de 8 bits y frecuencia de 20 kHz, mencionados con anterioridad, se debe asignar un timer de la controladora a un pin y configurarlo para la frecuencia deseada, esto se logra incluyendo en el “void setup ()” las siguientes instrucciones.

```
ledcSetup(timer,freq_desada,resolución);

ledcAttachPin(pin_deseado,timer);
```

7. Interfaz HMI

Este apartado abarca el desarrollo de la interfaz hombre máquina (HMI) utilizando el software conocido como *Processing* [19]. La finalidad de esta interfaz es permitir al usuario controlar diferentes parámetros del sistema, facilitando así la implementación de una estrategia de control basada en un controlador PID.

Objetivos de la interfaz HIMI:

- **Ajuste PID:** Permitir al usuario editar la acción proporcional (Kp), la acción integral (Ki) y la acción derivativa (Kd) de forma independiente.
- **Visualización en Tiempo Real:** Ofrecer una visualización de los datos del sensor con un tiempo de muestreo de 10ms, que es el máximo que admite el ESP32 sin saturarse.
- **Interacción intuitiva:** Facilitar una interfaz intuitiva y fácil de usar, que permita a usuarios de diferentes niveles interactuar con el sistema de una forma efectiva.

7.1. Guía de usuario de la HMI

Modo edición.

Figura 44: Interfaz HMI en modo edición

En la figura 44, se ven dos grandes apartados principales de la interfaz, la zona de control y la zona de muestreo.

- Zona de Control

En esta sección, el usuario puede editar los valores de las constantes del PID. Para ello, debe seguir estos pasos:

1. **Editar Constantes PID:** Hacer clic en los rectángulos correspondientes a las constantes que se deseé modificar.
2. **Ingresar Valor:** Escribir el nuevo valor utilizando el teclado. Este valor está limitado a 6 caracteres, incluido el punto decimal. Esto significa que, para los números decimales, solo se pueden ingresar hasta 5 dígitos (como máximo) y el punto decimal.
3. **Confirmar Cambio:** Hacer clic fuera del rectángulo después de ingresar el valor. El nuevo número se enviará automáticamente al controlador ESP32.

Botón "Start":

Este botón permite iniciar el lazo de control en el ESP32 y cambiar al modo adquisición.

- Zona de display:

En esta sección se mostrarán los valores muestreados del sensor, en total se muestrean 1,2 segundos. Si el usuario pone el ratón sobre el display, puede ver el valor exacto del sensor.

Un ejemplo de la interfaz en modo edición se puede ver a continuación.

Figura 45: Ejemplo de la interfaz en modo edición

Modo adquisición:

Figura 46: Interfaz en modo adquisición

En la figura 46, se puede observar la interfaz en modo adquisición Durante este modo no se pueden editar las constantes del PID, en su lugar se muestrea los valores del sensor cada 10ms Si se pone el ratón sobre el display, se mostrará el valor exacto del sensor.

Botón “STOP”

Si se desea pasar a modo edición, se deberá hacer clic en el botón “STOP”

7.2. Instalación librería “EasyModbus”.

Para poder usar el protocolo de comunicación Modbus TCP en Processing, hay que instalarse una librería externa, a continuación, se explica cómo instalarla.

Lo primero es buscar en Google “easymodbustcp java”.

Hacer clic en el enlace de GitHub [20]. Una vez se haga clic en el enlace, aparecerá la página de GitHub como se puede ver en la figura 47.

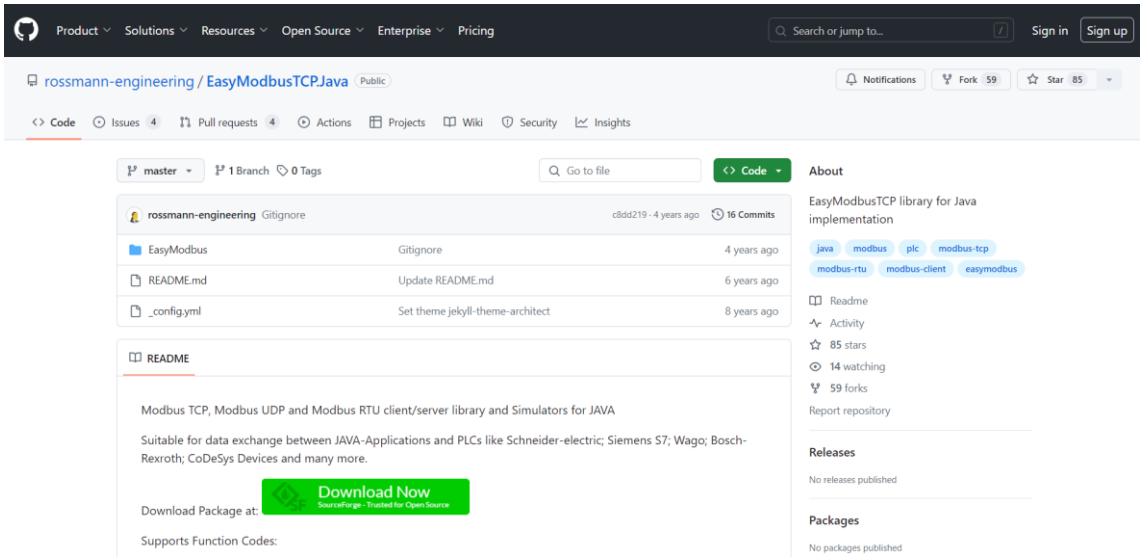

Figura 47: Web de la librería en GitHub

Al hacer clic en el rectángulo verde que indica “Download Now” (ver figura 47) se abrirá una página donde se descarga automáticamente un archivo con nombre “EasyModbusJava.jar”, este archivo se deberá mover a la carpeta “lib” que se encuentra dentro de la carpeta de instalación de *Processing* por defecto esta carpeta se llama” processing-4.3” y está ubicada donde se instalase *Processing*

Con estos pasos la librería ya estará disponible para ser utilizada en *Processing,*

Para incluir la librería en el código hay que introducir la siguiente instrucción:

```
import de.re.easymodbus.modbusclient.*;
```

Figura 48: Instrucción para cargar la librería en el código

Para ver todas las instrucciones [21] y la forma en la que se utiliza, hay que hacer clic donde pone “EasyModbus”. A la izquierda de esto se encuentra la imagen de una carpeta, como se muestra la figura 47. Una vez hecho clic en el enlace, se debe de buscarla carpeta que contiene los comandos de la librería. Esta ruta y el archivo que alberga los comandos se muestra en la figura 49.

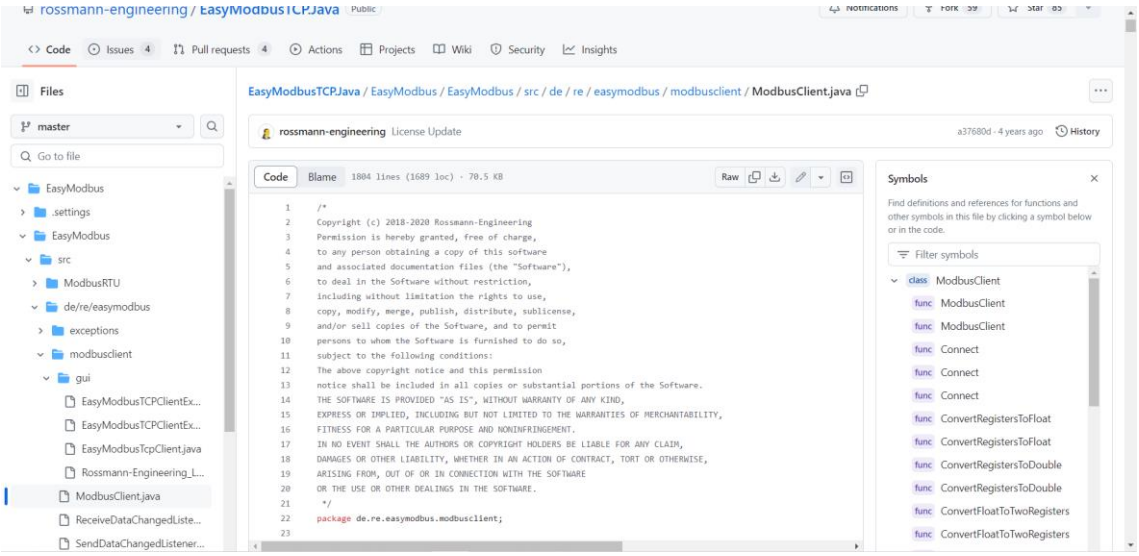

Figura 49: Manual de usuario de la librería en GitHub

7.3. Código Modbus para la HMI

Igual que en el ESP32, se programa usando los dos núcleos para maximizar las prestaciones. En la HMI, la filosofía no va a ser diferente. Para programar la interfaz se han implementado dos hilos de ejecución, un hilo se encarga de las comunicaciones ordenador-controladora y otro hilo se dedica de dibujar la interfaz.

Para poder programar a dos hilos, lo primero es asignar una tarea a un nuevo hilo de ejecución. Esto se realiza dentro de la función “void setup ()”, de la siguiente manera.

```
thread("Modbus");// Asignamos la tarea "Modbus" a un nuevo hilo de ejecución
```

Figura 50: Creación de una nueva tarea.

Finalmente, hay que definir una función que no reciba ningún argumento de entrada y que contenga un bucle infinito con el código que se desea ejecutar indefinidamente en el otro núcleo, de forma que el código quedaría como en el siguiente flujograma (ver figura 51):

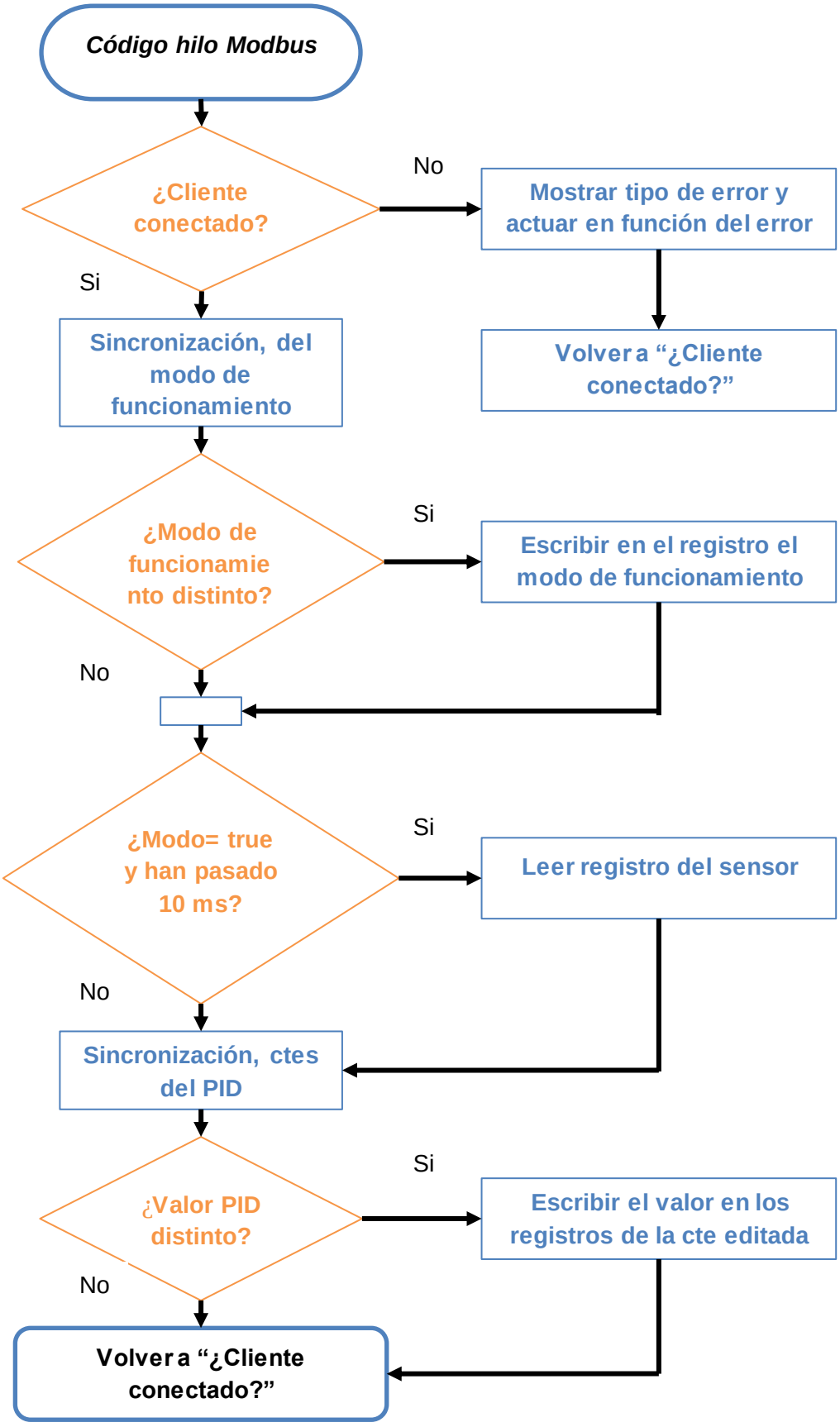

Figura 51: Flujograma del hilo de Modbus en Processing

Para este código, se usan variables sincronizadas conocidas como “synchronized” [24]. Aunque este tipo de variables no es obligatorio para programar a múltiples hilos, es una práctica muy recomendada para hacer un código muy robusto y minimizar los posibles errores. Se recomienda el uso de estas cuando varios hilos comparten una misma variable, con el fin de evitar errores de concurrencia.

8. Presupuesto y mediciones

En la figura 52, se muestra un cuadro con las cantidades necesarias de cada material, el precio por unidad y el total del proyecto.

Capítulo 1: Péndulo invertido				
Nombre	Unidades	Cantidad	Precio por unidad	Total
ESP-WROOM-32	ud	1	8,99 €	8,99 €
Adafruit 9-DOF-Sensor de orientación absoluta IMU de 9-DOF fusión-BNO055	ud	1	74,99 €	74,99 €
Motor cc, 12V, con reductora de 70:1 y encoder. Fabricante Pololu	ud	1	48,63 €	48,63 €
MDV 2x2A DC	ud	1	12,84 €	12,84 €
Varilla roscada de M6 y1m de longitud	ud	1	0,64 €	0,64 €
Conector acoplamiento de brida, D 6mm, accesorio acoplador de guía rígida, pack de 4 unidades	ud	1	11,99 €	11,99 €
Eje liso de D 6mm, longitud 100mm acero inoxidable, pack de 2 unidades	ud	1	7,99 €	7,99 €
Tablero pino C/N A=80 b=20 C=1,8 cm	ud	1	7,49 €	7,49 €
Liston cepillado abeto 24x56x900mm	ud	1	6,99 €	6,99 €
Base péndulo, impresión 3D en ABS	ud	2	5,45 €	10,90 €
Disco inercia, impresión 3D en ABS	ud	1	9,55 €	9,55 €
Soporte sensor, impresión 3D en ABS	ud	1	5,45 €	5,45 €
6 Tornillos,cabeza plana avellanada para madera, M6 y longitud 60mm	ud	1	1,99 €	1,99 €
45 Tornillos, cabeza plana avellanda, M 3, longitud 20mm	ud	1	1,79 €	1,79 €
10 Tornillos, cabeza redonda, M 6 y longitud 16 mm	ud	2	2,79 €	5,58 €
20 Tornillos, cabeza cilíndrica, M 3 y longitud 12 mm	ud	1	2,99 €	2,99 €
Horas de ingeniero	h	120	60,00 €	7.200,00 €
			Total Proyecto:	7.418,80 €

Figura 52: Tabla de presupuesto y mediciones del proyecto.[22], [23]

9. Caso de uso

A continuación, se muestra con un ejemplo la respuesta del sistema en un uso normal. En este caso se puede observar como el sistema oscila entre los límites de funcionamiento con una cierta frecuencia constante, no obstante, no es capaz de mantener la referencia de 0 grados.

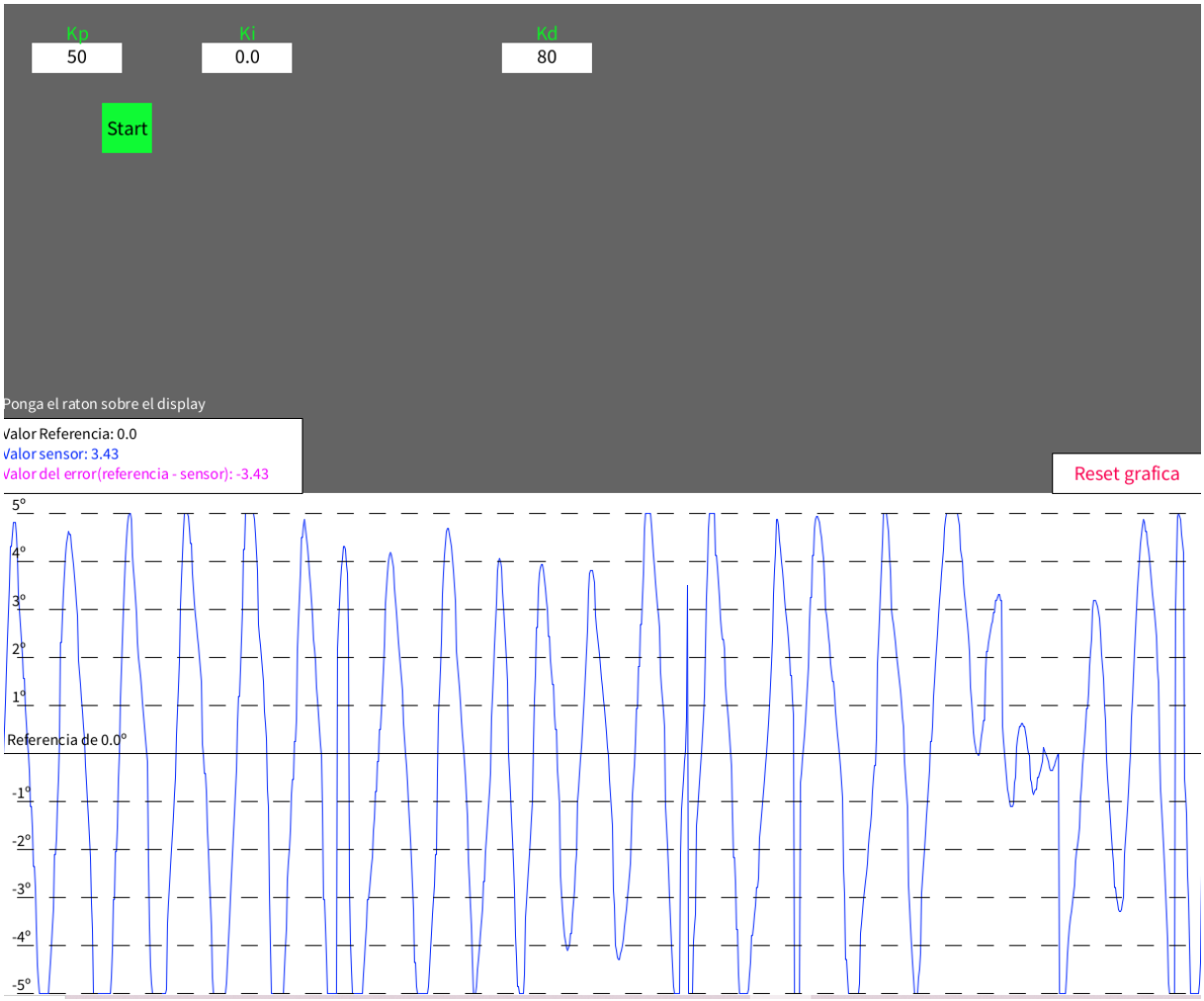

Figura 53:Respuesta del sistema en el mejor de los casos

En el QR que se adjunta a continuación se puede ver un vídeo del sistema funcionando.

Figura 54: QR video demostración

10. Conclusiones

Tras una gran cantidad de ensayos experimentales, se llega a las siguientes conclusiones.

Aunque no se consigue mantener la referencia de 0. 0º mencionada anteriormente, se obtiene otro tipo de respuesta. Gracias a un controlador tipo PD, el sistema es capaz oscilar entre ±5º respecto a la horizontal. Es decir, el sistema, cuando está libre la gravedad lo llevará a -5º o a 5º. Cuando el sistema se encuentre en algunas de estas posiciones es capaz de aplicar el par necesario para cambiar de posición, como se muestra en la figura 53

Esta es la mejor respuesta a la que se ha podido llegar experimentalmente. Como se puede apreciar en la figura anterior, hay una cierta frecuencia de oscilación entre los dos márgenes máximos de operación, que como se ha mencionado anteriormente, es entre -5º a +5º respecto a la horizontal.

La controladora del motor es un punto que se podría mejorar, buscando una controladora que permita un frenado en seco del motor, esto sin duda mejoraría la respuesta del sistema. La controladora que se usa para este sistema no tiene un frenado en seco; si no que al poner a 0 el ciclo de trabajo esta frena el volante en gran medida, pero se puede observar como aún le queda una cierta velocidad angular al volante.

Por otro lado, el volante de inercia es todo un éxito, ya que, gracias a él, el sistema puede oscilar entre un amplio margen de grados, sin embargo, esto no ocurre con otras versiones del volante en las cuales, no es suficiente para controlar el sistema o el margen de grados de funcionamiento se reduce considerablemente. Además, el diseño de la estructura que aguanta la base del péndulo cumple con sus funciones.

El sensor, aunque es muy fácil usarlo, presenta un gran problema relacionado con su calibración. Ya que antes de usarlo hay que hacer una secuencia de movimientos para calibrarlo, pero no siempre se obtienen las mismas mediciones, lo que introduce un error. A diferencia de otros sensores similares, el sensor utilizado en este proyecto no permite un bypass para poder forzar unos valores de calibración y así garantizar siempre una misma medida.

Dado que el sistema se ha programado para que sea compacto y sin necesidad de cables para transmitir y recibir la información, sería ideal que, para un futuro, se diseñe una PCB en la que se pueda conectar el sensor al ESP32 y este a una batería y así evitar tener que alimentar el ESP32 con un cable USB.

Referencias

[1] Julio César, Cortés Ríos,*" El algoritmo de sintonización simple de controladores difusos (ASSCD*)", 2017, ISBN: 607-749-045-8

[2] "Segway-Wikipedia, la enciclopedia libre".

URL: https://es.wikipedia.org/wiki/Segway

[3] Mohanarajah Gajamohan; Michael Muehlebach; Tobias Widmer; Raffaello D'Andrea, *"The Cubli: A Reaction Wheel Based 3D Inverted Pendulum",* 2013 European Control Conference, doi: 10.23919/ECC.2013.6669562

[4] ESP-WROOM-32 *Datasheet:* Accessed: Feb,2024.

URL:https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[5] Pololu. 70:1 Metal Gearmotor 37Dx70L mm. Accessed: Jun 2024.

URL: https://www.pololu.com/product/4754

[6] DFROBOT. "L298 Dual H-Bridge Motor Driver". Accessed: Jun, 2024.

URL: https://wiki.dfrobot.com/MD1.3_2A_Dual_Motor_Controller_SKU_DRI0002

[7] Adafruit Industries, "Adafruit_BNO055*"* Accessed: Jun, 2024,

URL: https://github.com/adafruit/Adafruit_BNO055

[8] "Leyes de Newton- Wikipedia, la enciclopedia libre"

URL: https://es.wikipedia.org/wiki/Leyes_de_Newton

[9] "Momento de inercia – Wikipedia, la enciclopedia libre"

URL: https://es.wikipedia.org/wiki/Momento_de_inercia

[10] Tom Stanton, "Can Reaction Wheels control a Drone" Accessed Jun,2024

URL: https://www.youtube.com/watch?v=4kfBEaTncjI

[11] Bedford, Anthony.; Fowler, Wallace L, "*Mecánica para ingeniería: estática*" 5ª. Edición, ISBN: 607-442-876-X

[12] Daniel Carrasco, "FREERTOS en ESP32/ESP8266 (multitarea)" Accessed: Jun, 2024

URL: https://www.electrosoftcloud.com/freertos-en-esp32-esp8266-multi-tarea/

[13] IoTicos, "ESP32 Tutorial al Rtos- DUAL CORE" Accessed: Jun,2024.

URL: https://www.youtube.com/watch?v=l3LR_ljKEPQ

[14] ESPRESSIF, "FreeRTOS (IDF)", Accessed: Jun, 2024

URL:https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/freertos_idf.html

[15] Thierry Vaira, "Activité: ESP32 + FreeRTOS", Accessed: Jun, 2024

URL: http://tvaira.free.fr/esp32/esp32-freertos.html

[16] Luis Llamas, "Como conectar un ESP8266 o ESP32 a una red Wifi modo STA", Accessed Jun, 2024.

URL: https://www.luisllamas.es/como-conectar-un-esp8266-a-una-red-wifi-modo-sta/

[17] Akande; Olushola," *Industrial automation from scratch a hands-on guide for using sensors, actuators, PLCS, HMIs, and SCADA to automate industrial processes",* 1ª edición, 2023, ISBN: 1-80056-690-5

[18] emelianov, "modbus-esp8266", Accessed: Jun, 2024.

URL:https://github.com/emelianov/modbus-esp8266/blob/master/examples/TCP-ESP/IP-server-Led/IP-server-Led.ino

[19] Processing, "Processing", Accessed: Jun, 2024

URL: https://processing.org/

[20] rossmann-engineering, “EasyModbusTCP.Java”, Accessed: Jun, 2024.

URL: Enlace

[21] Talos Electrónico, “Desarrollo de interfaz gráfica (GUI) para trabajar con MODBUS TCP/IP a través de Java y Visual Basic”. Accessed: Jun, 2024

URL: https://www.youtube.com/watch?v=bZP7PCzkl5c

[22] Amazon, “Brida 6 mm” Accessed, Jun, 2024

URL: https://www.amazon.es/dp/B0833NTD9M/ref=twister_B0833SCFL5?_encoding=UTF8&th=1

[23] Amazon, “Eje liso 6mm de diámetro” Accessed: Jun, 2024.

URL: Enlace

[24] Stack**overflow**, “What is the use of "private final Object" locking in java multithreading?”, Accessed, Jun, 2024.

URL: https://stackoverflow.com/questions/19419702/what-is-the-use-of-private-final-object-locking-in-java-multithreading