Escuela Politécnica Superior de Jaén
Máster Universitario en Ingeniería Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

DISEÑO DE UNA CONTROLADORA DE BAJO COSTE PARA MOTORES PASO A PASO

Alumno/a: **García Rodríguez, Pablo Manuel**

Tutor/a: Prof. D. Alejandro Sánchez García
Dpto.: Ingeniería Electrónica y Automática

Julio, 2020

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Ingeniería Electrónica y Automática

Don Alejando Sánchez García , tutor del Trabajo de Fin de Máster titulado: Diseño de una controladora de bajo coste para motores paso a paso, que presenta Pablo M. García Rodríguez, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Julio de 2020

El alumno:

GARCIA RODRIGUEZ PABLO MANUEL - 77360840H

Firmado digitalmente por GARCIA RODRIGUEZ PABLO MANUEL - 77360840H
Fecha: 2020.07.06 11:49:56 +02'00'

Pablo M. García Rodríguez

El tutor:

SANCHEZ GARCIA ALEJANDRO - 77321466C

Firmado digitalmente por SANCHEZ GARCIA ALEJANDRO - 77321466C
Fecha: 2020.07.06 11:31:21 +02'00'

Alejandro Sánchez García

A Alejandro, por guiarme en este camino.
A mi familia, por su apoyo incondicional.
A mis amigos del máster, E · I nos llevó al final del camino.

Índice general

Índice de figuras

Índice de tablas

Resumen

Este Trabajo de Fin de Máster trata sobre el diseño de una controladora para motores paso a paso. Se busca la realización de una controladora que sea adaptable a cualquier tipo de sistema que implemente estos motores y que permita una amplia configuración para los movimientos y ordenes para estos, como por ejemplo, movimientos a partir de distancias de desplazamiento o a partir de pasos del propio motor. Para ello, se va a hacer uso de herramientas Open Source, de la impresión 3D y del hardware electrónico necesario. Además, se perseguirá el objetivo de que la controladora se pueda utilizar en sistemas de automatización reales a partir de la implementación de protocolos de comunicación y de un Servidor web.

Abstract

This Master's thesis is about the design of a stepper motor controller. The aim is to make a controller that can be adapted to any type of system that implements these motors and that allows a wide configuration for the movements and orders for these, such as movements from distances of displacement or from steps of the motor itself. For this purpose, Open Source tools, 3D printing and the necessary electronic hardware will be used. In addition, the objective will be pursued that the controller can be used in real automation systems from the implementation of communication protocols and a Web Server.

Capítulo 1

Introducción

1.1. Motivación

Un motor paso a paso se caracteriza por su movimiento angular a partir de pulsos eléctricos en sus distintas bobinas. Esto quiere decir, que cuando se excita/n la/s bobina/s necesaria/s, el motor realiza un movimiento en la dirección correspondiente. Este tipo de movimiento intermitente, a priori, permite saber en qué posición se encuentra el motor, pues conociendo el número de pulsos enviados al motor, se sabrá el número de desplazamientos, llamados pasos, que habrá realizado. Sin embargo, en la práctica esto no es siempre así, ya que cuando el motor se encuentra una resistencia mecánica, ya sea externa, o debida a las aceleraciones de su movimiento, se puede ver imposibilitado este movimiento, haciendo así que la posición no cambie a pesar de que se le ha dado el pulso eléctrico correspondiente. Esto se conoce como la pérdida de un paso. Cuanto esta resistencia al movimiento se mantiene, se puede perder gran cantidad de pasos, por lo que, en un movimiento largo se puede llegar a tener una gran pérdida de la posición del eje del motor. Con esto, se entiende que un motor paso a paso funciona por si solo en lazo abierto, ya que solo se le envía información para su movimiento, pero luego no se recibe si este desplazamiento ha sido realizado de forma correcta.

Para evitar la pérdida de pasos, y por lo tanto, la pérdida de precisión y localización del movimiento, se puede acoplar al motor paso a paso un encoder y un microcontrolador o un microprocesador programado para la lectura de la información aportada por este elemento para su posterior retroalimentación al movimiento del motor, es decir, para cerrar el lazo del control del motor.

En el mercado existen distintas controladoras comerciales, como la serie JXC73/83 de SMC o como la seria AZ de Oriental motor, estudiadas en los próximos apartados de este documento, que permiten este control de motores paso a paso pero que debido a su complejidad y uso presentan unos precios superiores a los quinientos euros [2], por lo que he aquí la motivación de este trabajo de realizar una controladora, que tenga las características necesarias para el control en lazo cerrado de un motor paso a paso, pero con un bajo coste utilizando materiales

accesibles, ayudándose de la impresión 3D y de los recursos Open Source necesarios.

1.2. Objetivos

El objetivo principal de este trabajo es implementar una controladora para motores paso a paso en lazo cerrado, que permita seguir trayectorias definidas por el usuario. Para lograrlo, se proponen las siguientes metas a cumplir.

- Estudiar el funcionamiento de los motores paso a paso.
- Estudiar el funcionamiento de un encoder rotacional.
- Analizar distintos modelos de controladores del mercado actual.
- Elección de los distintos elementos que consta la controladora
- Programar la aplicación, desarrollando todos los ficheros necesarios para el correcto funcionamiento de la controladora.
- Realizar diseños CAD de las piezas necesarias.
- Estudiar posibles formas de comunicación con una Raspberry pi, como por ejemplo, un servidor web y protocolos de comunicación.
- Realizar pruebas y comprobaciones de funcionamiento.

Capítulo 2

Definición del problema

2.1. Motores paso a paso

Un motor paso a paso es un dispositivo electromagnético que se caracteriza por su movimiento angular proporcional al número de pulsos recibidos en sus bobinas y cuya velocidad será equivalente a la frecuencia de los mismos. Tienen un diseño efectivo y de bajo coste además de una alta fiabilidad. Por si solos trabajan en lazo abierto, es decir, el movimiento se realiza sin ningún tipo de retroalimentación.

2.1.1. Tipos de motores paso a paso

Motores paso a paso de reluctancia variable (V.R.)

Los motores de reluctancia variable están formados por un rotor y un estator con un número diferente de dientes. Al no existir un imán permanente, el rotor gira libremente sin par de retención [3]. El estator suele ser de forma cilíndrica con tres devanados desfasados 120° conectados en estrella, conectando el común a la borna positiva y alimentando las bobinas siguiendo una secuencia consecutiva [4].

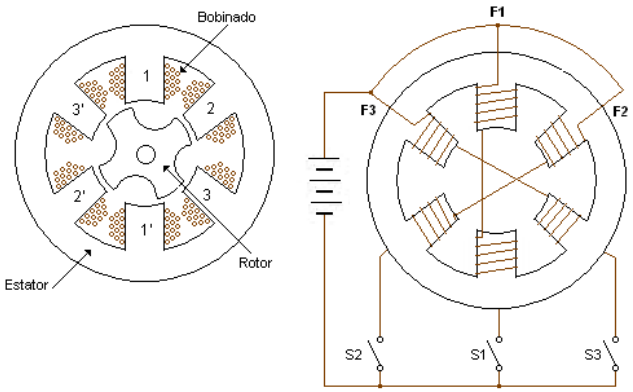


Figura 2.1: Sección de motor paso a paso de reluctancia variable.

Fuente: [5]

Motores paso a paso de imanes permanentes (P.M.)

Los motores de imanes permanentes son los más utilizados en robótica. En su forma más simple está formado por un rotor con magnetizado radial y un estator similar a los V.R. [3]. Existen dos tipos:

- **Unipolares** [4].
  Normalmente con cinco o seis cables en la salida según el conexionado interno. Se suelen utilizar cuatro para recibir los pulsos que indican la secuencia y duración de los pasos y los restantes para la alimentación del motor. Existen tres secuencias de manejo posibles para estos motores.

  1. Secuencia normal: se avanza un paso con la excitación de dos bobinas, consiguiendo así un par elevado.
  2. Secuencia Wave drive o de paso completo: el eje se posiciona en la bobina activa, consiguiendo un movimiento más suave pero de menor par.
  3. Secuencia de medio paso: se combinan las secuencias anteriores, activando primero dos bobinas y posteriormente solo una de forma consecutiva obteniéndose un paso más corto.

- **Bipolares** [4].
  Suelen disponer de cuatro cables, y necesitan un cambio en la dirección del flujo de la corriente en las bobinas en la secuencia apropiada para realizar el movimiento, por lo que se requiere de un puente en H por cada bobina, haciendo su tarjeta controladora más compleja y cara. La secuencia de alimentación necesaria se muestra en la tabla 2.1.

Paso	A+	A-	B+	B-
1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	+V
4	-V	+V	+V	-V

Tabla 2.1: Secuencia de alimentación de las bobinas para un motor paso a paso de P.M. bipolar.

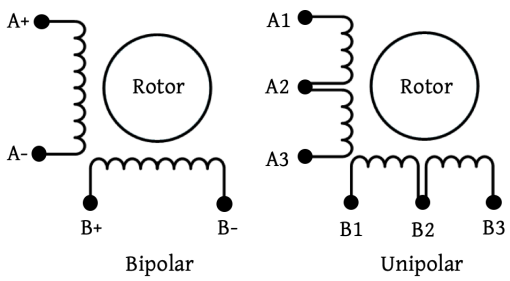


Figura 2.2: Esquema de los motores de P.M. Unipolar y Bipolar.

Motores paso a paso híbridos

Los motores híbridos se basan en una combinación del funcionamiento de los otros dos tipos. Consiste en un estator dentado y un rotor de tres partes de apilado simple. Este rotor contiene dos piezas de polos separados por un imán permanente, con los dientes opuestos desplazados medio paso [3].

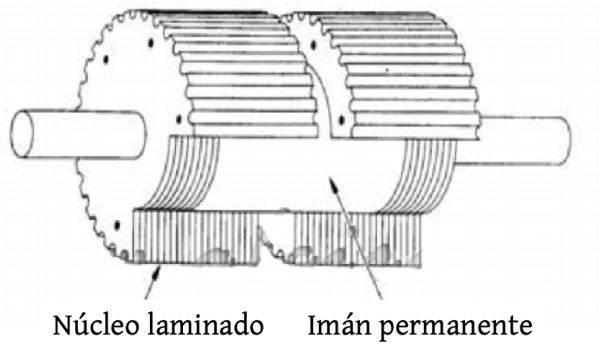


Figura 2.3: Rotor de un motor paso a paso híbrido.

Fuente: [5]

2.2. Encoder

Un encoder es un dispositivo cuyo parámetro principal es la posición. Existen encoders de rotación, que miden la rotación de un eje, y lineales que miden distancia recorrida. Para ambos casos, la medida puede ser absoluta o incremental. Para cerrar el lazo a un motor paso a paso se utiliza un encoder rotacional, que se basa en un disco, unido al eje de rotación, fabricado con una codificación de partes transparentes y opacas que no dejan pasar la luz emitida por una fuente, normalmente emisores infrarrojos. A medida que el disco gira, la luz que atraviesa las partes transparentes es recibida por un fotorreceptor generando pulsos digitales que permite conocer el giro producido.

2.2.1. Tipos de encoder rotacionales

Incremental

Este tipo de encoder tiene una posición estratégica desde la cuál siempre comienza la cuenta. Determina la posición a partir de cuentas incrementales, es decir, la posición actual se obtiene comparándola con la última posición registrada por el sensor.

Los encoders de cuadratura son un tipo de encoder incremental, que utiliza dos sensores ópticos con un desfase de un cuarto de ranura, generando así dos señales de pulsos desfasadas 90°, es decir, dos señales en cuadratura. Estas señales reciben, comúnmente, los nombres de A y B, y con ellas se pueden conocer valores de posición, velocidad y dirección de rotación. Algunos encoders más complejos tienen una tercera señal, Z, con una posición fija para utilizar como referencia. Normalmente, se establece que el eje está girando en sentido horario si la señal A adelanta a la señal B, es decir, si la señal A tiene el valor lógico 1 antes que la señal B [6].

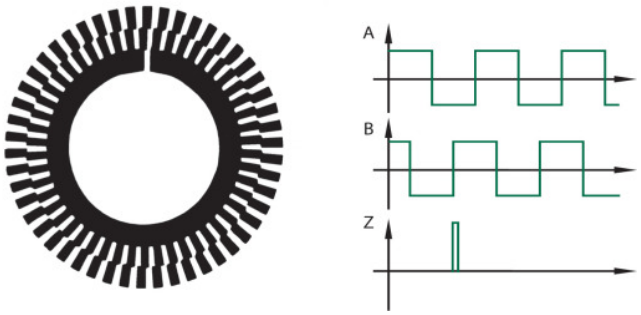


Figura 2.4: Encoder incremental y sus señales A, B y Z.

Fuente: [7]

Absoluto

Obtiene una posición absoluta a partir de la información recibida. El disco tiene un código único para cada posición por lo que es posible conocer la posición exacta aún cuando el sistema ha sufrido una pérdida del suministro de energía, por lo tanto, a diferencia de los incrementales, no es necesario tener una posición de referencia.

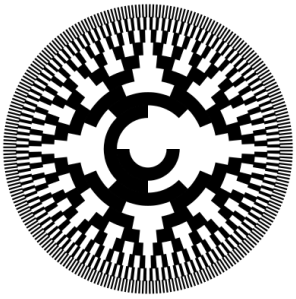


Figura 2.5: Disco de encoder absoluto codificado en 10 bits.

Fuente: https://i.stack.imgur.com/I1IyM.png

Monovuelta y Multivuelta

Los monovuelta dividen una revolución completa en un determinado número de pasos y, al terminar la revolución, se repiten los valores. Mientras tanto, los multivuelta son capcaces de registrar el número de revoluciones además de la posición angular [6].

2.2.2. Tipos de codificación para un encoder rotacional incremental

Existen tres tipos de codificaciones, X1, X2 y X4. La diferencia entre ellas es la forma de contabilizar los pulsos de la señales de salida, teniendo gran influencia en la resolución del encoder [8].

La codificación X1 consiste en contar los cambios a pulso alto o a pulso bajo de la señal A y compararlos con el estado de la señal B. Si la señal A llega a valor alto antes que B, entonces se dice que se mueve en sentido horario, mientras que si es B la primera, entonces irá en sentido antihorario.

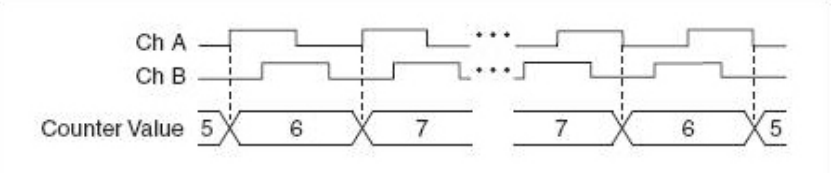


Figura 2.6: Codificación X1 para un encoder.

Fuente: [8]

La codificación X2 utiliza los cambios a pulso alto y bajo de la señal A. Doblando así el número de pulsos contabilizados y, por lo tanto, la resolución del encoder.

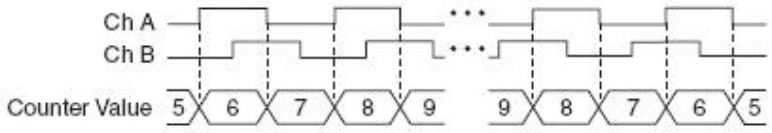


Figura 2.7: Codificación X2 para un encoder.

Fuente: [8]

La codificación X4 utiliza los cambios de pulso alto y bajo, como la X2, pero de las dos señales, no solo de A, por lo tanto, se multiplica por cuatro tanto el número de pulsos contabilizados como la resolución del encoder con respecto a la codificación X1.

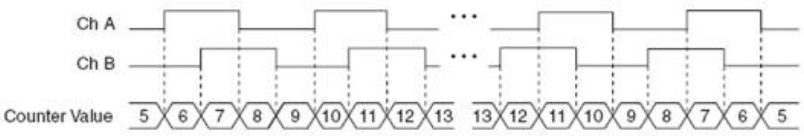


Figura 2.8: Codificación X4 para un encoder.

Fuente: [8]

2.3. Protocolos de comunicación

Un protocolo de comunicación está formado a partir de una serie de reglas y formatos de mensajes preestablecidos para que la comunicación emisor/receptor sea posible. Estas reglas definen cómo deben hacerse las comunicaciones de las redes, incluyendo tiempos, secuencia y revisión y corrección de errores.

Los tres elementos clave de la comunicación son la **sintaxis**, que es el formato de los mensajes (datos y comandos), la **semántica** o el significado de los comandos y la **secuencia y temporización** de las acciones dadas por los comandos [9].

Las arquitecturas estandarizadas para la comunicación son el modelo de interconexión de sistemas abiertos (**OSI**) y el modelo **TCP/IP** (Transmission Control Protocol/Internet Protocol).

- Modelo OSI: Fue desarrollado por la Organización Internacional de Normalización (ISO) y es un modelo de referencia formado por siente capas: Física, Enlace de datos, Red, Transporte, Sesión, Presentación y Aplicación.
- Modelo TCP/IP: Desarrollado por la Agencia de Investigación de Defensa de Estados Unidos (DARPA), es el estándar utilizado en Internet. Está formado por cuatro capas: Física o de Enlace a la Red, Internet o IP, Transporte o TCP y Aplicación.

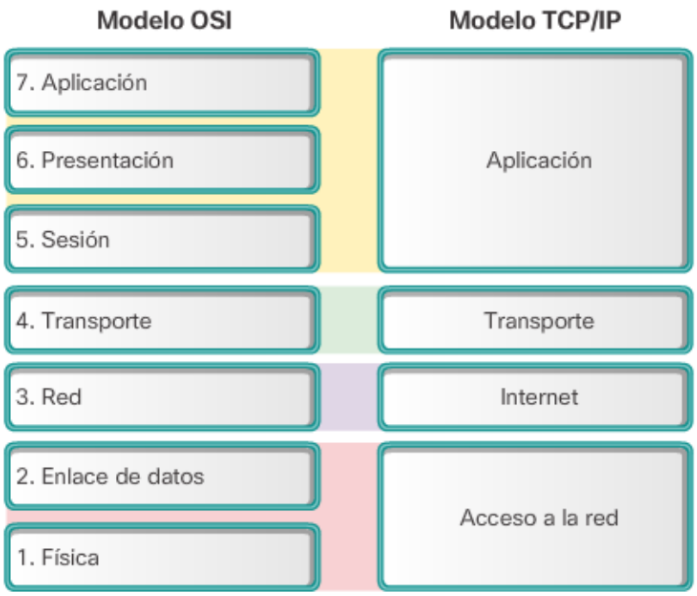


Figura 2.9: Comparación de capas entre los modelos OSI y TCP/IP.

Fuente: [10]

2.4. Controladoras en el mercado

Para controlar un motor paso a paso en lazo cerrado con un encoder, ya sea incremental o absoluto, es necesario el uso de una controladora que permita interpretar los datos de esta retroalimentación. Actualmente existen distintas controladoras comerciales como la serie JXC73/83 de SMC que permite el control de una máquina de cuatro ejes con motores paso a paso con sus conexiones pertinentes y las de sus encoders. La diferencia entre los modelos 73 y 83 son las especificaciones de las entradas y salidas, siendo la 73 referente a NPN y la 83 a PNP. Esta serie permite la utilización de 512 pasos en su modo normal y 2048 en su modo extendido. Tiene puerto USB para la conexión a un ordenador, dos conectores de I/O para la conexión a un PLC de 24 VDC y dos conexiones para la alimentación de los motores a 24 VDC. A parte, tiene su propia interfaz gráfica de usuario (GUI) para el control de los distintos motores [11].

Figura 2.10: Serie JXC73/83 de SMC.

Fuente: [11]

SMC también tiene en su catálogo otras controladoras más sencillas que permiten el control de un único motor como es el caso de la serie LECPA. Esta serie tiene los modelos AN y AP para las entradas NPN y PNP respectivamente. Se puede conectar a un ordenador o a una consola de programación mediante Ethernet y a un PLC con alimentación de 24 VDC para sus señales y permite la comuniciación en serie con RS485 mediante protocolo Modbus [12].

Figura 2.11: Serie LECPA de SMC.

Fuente: [12]

Otra opción en el mercado actual es la serie AZ de Control motor. Esta serie tiene una disponibilidad de cuatro formatos según la comunicación; controlador compatible con EtherNET/IP, controlador de datos almacenados en red, controlador de entrada de pulsos o entrada de pulsos con RS-485 para la monitorización. En cualquier caso, tiene funciones de protección internas, como la de eliminación de par de emergencia y dispone de alimentación en 100-120 VAC monofásico o 200-240 VAC en monofásico y trifásico. A parte dispone de software propio para el control, MEXE02, de descarga gratuita [13].

Figura 2.12: Serie AZ de Oriental motor.

Fuente: [13]

2.5. Necesidades

El objetivo principal de una controladora es conseguir un control de movimiento del motor paso a paso en lazo cerrado, para así tener un conocimiento continuo de su posición y movimiento de forma precisa, gracias a la información aportada por el encoder.

A parte, la controladora debe permitir la comunicación con el exterior, como por ejemplo con un PLC, a partir de protocolos de comunicación para así tener una utilidad real en la industria o en alguna actividad que se quiera automatizar. Por ello, la controladora a diseñar debe de disponer de una potencia de cálculo suficiente para interpretar los datos de entrada y ser capaz de realizar las comunicaciones pertinentes. Dichas comunicaciones deben de seguir algún estándar, como Modbus basado en TCP, que permita una comunicación simple, rápida y suficiente.

Por último, si el motor a controlar se trata de un motor bipolar será necesario disponer también de los puentes en H necesarios para las bobinas a parte del encoder y de los sensores pertinentes para el sistema en cuestión.

Capítulo 3

Descripción de la solución

3.1. Motor paso a paso

Para la realización del proyecto es necesario utilizar un motor a paso para controlar. En el mercado existe una gran amplia variedad, desde motores de baja potencia y bajo coste, hasta motores más robustos y potencia. Para las pruebas de control se ha utilizado un motor de imanes permanentes bipolar que se puede adquirir en Amazon por un precio de 13,99 €. Aún así, existen páginas como BricoGeek con un catálogo amplio de estos motores, llegando a haber modelos de bajo precio y coste hasta modelos de 140 Ncm de par de retención con un precio aproximado de 50 euro. En RS existe una mayor variedad de modelos llegando a casos de cerca de 200 Ncm de par de retención y 140 €

Figura 3.1: Motor paso a paso utilizado.

Las características dadas por el fabricante son las mostradas en las siguientes tablas.

Carac.	Valor
Tipo	Motor bipolar
Ángulo de paso	1,8°
Pasos por rev.	200
Par de retención	26 Ncm
I_{nom}	0,4 A
V_{nom}	12 V
R_{fase}	30 Ω
L	37 mH +/- 20 % (1 kHz)

Tabla 3.1: Características del motor paso a paso utilizado.

Carac.	Valor
Longitud del cable	≈ 300 mm
Tamaño del marco	42x42 mm
Longitud del cuerpo	34 mm
Diámetro del eje	5 mm
Longitud del eje delantero	20 mm
Longitud D-cut	15 mm
Nº de contactos	4
Longitud del plomo	300 mm
Peso	230 g

Tabla 3.2: Características físicas del motor paso a paso utilizado.

Color	Borna
Negro	A+
Verde	A-
Rojo	B+
Azul	B-

Tabla 3.3: Disposión de las bobinas del motor paso a paso utilizado.

3.1.1. StepStick A4988

Para la alimentación de las bobinas del motor se necesita un puente en H para emitir los pulsos de forma correcta. Para ello, se debe utilizar un driver para motores paso a paso. En el mercado existe una gran variedad de ellos, algunos ejemplos son los modelos A4988, DRV8825, TB6600, TMC2208 y TMC2130. Los dos primeros modelos son los más utilizados para el control de este tipo de motores en aplicaciones menos industriales, como pueden ser las impresoras 3D de usuario. El modelo DRV8825 tiene unas características un poco superiores al A4988, pero este último tiene un precio menor, por lo que es el elegido. Un pack de cinco A4988 con disipador se puede comprar en Amazon por 9,99 €.

El esquema de bloques funcional, dado por el fabricante, en el que se muestran los distintos elementos que conforman al driver, y los pines de este se muestra en la figura 3.2.

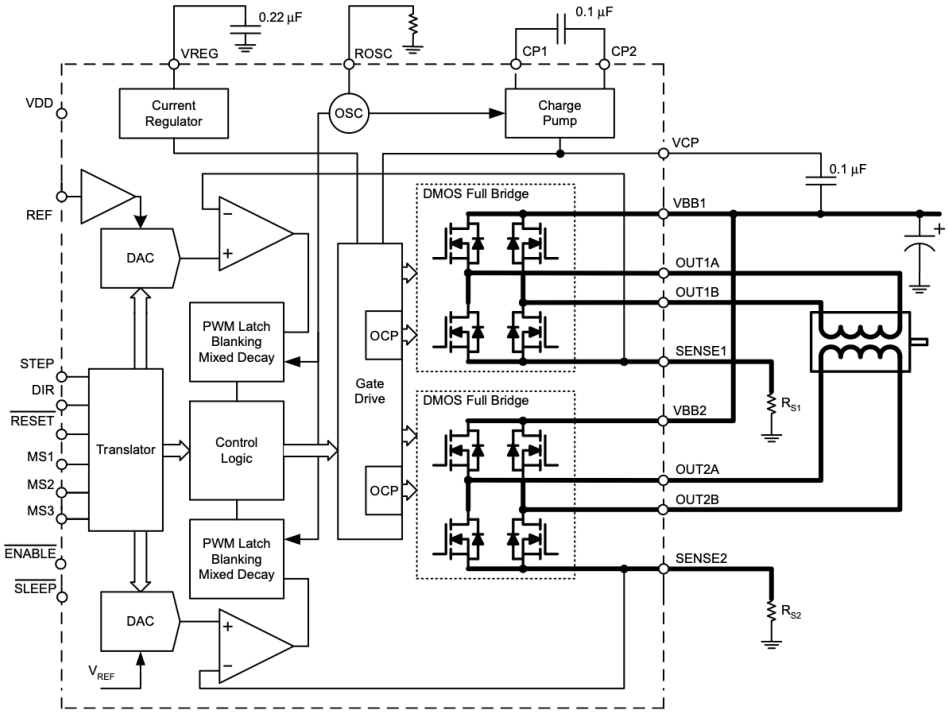


Figura 3.2: Esquema de bloques funcional del driver A4988.

Fuente: [14]

En cuanto al conexionado exterior de los pines del driver con el motor y la alimentación es el mostrado en la figura 3.3.

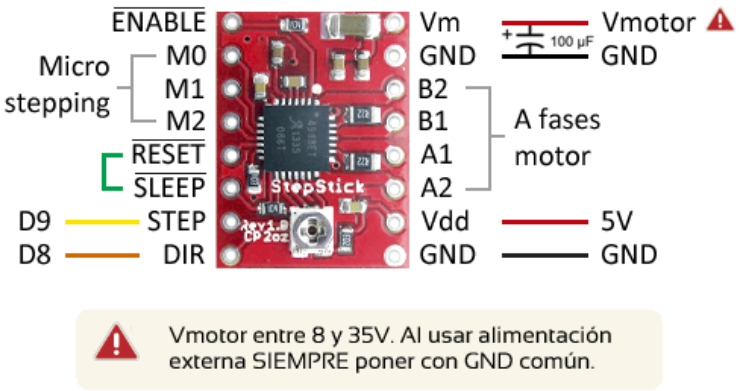


Figura 3.3: Esquema de conexionado del driver A4988.

Fuente: Luis LLamas Blog

Las características de funcionamiento de cada uno de los pines son las siguientes [14].

- $\overline{\text{ENABLE}}$: Esta entrada enciende o apaga todas las salidas FET. Cuando se encuentra en un valor lógico alto, las salidas se desactivan. Las entradas STEP, DIR y MSx, quedan activas independientemente del valor de este pin.

- MSx (M0, M1 y M2): Estos pines hacen referencia a la resolución para micropasos, si no se activan, el motor se moverá con los pasos por defecto. La lógica que siguen estos pines se muestra en la siguiente tabla.

M1	M2	M3	Res. de micropaso
0	0	0	Paso completo
1	0	0	1/2 paso
0	1	0	1/4 paso
1	1	0	1/8 paso
1	1	1	1/16 paso

Tabla 3.4: Disposión de las bobinas del motor paso a paso utilizado.

- $\overline{\text{RESET}}$: Cierra todas las puertas FET y todas las entradas por STEP son ignoradas hasta que se activa en valor alto.

- $\overline{\text{SLEEP}}$: Para minimizar el consumo energético cuando el motor no está en uso, este pin desactiva gran parte de la circuitería interna, incluidas las salidas FET, el regulador de corriente y el multiplicador de tensión. Un pulso bajo activa el modo reposo, mientras que un pulso alto permite la operación normal.

- STEP: Una transición de bajo a alto produce el movimiento de un paso en el motor.

- DIR: Determina la dirección de giro del motor. Un cambio en esta entrada no afecta al cambio de estado en el pin STEP.

- Vm: La alimentación para el motor, 12 VDC en el caso del motor utilizado.

- GND: Neutro.

- B2, B1, A1, A2: Salidas para las bobinas A y B del motor bipolar.

- Vdd: Alimentación del dirver a 5 VDC.

Calibrado del driver

Para el correcto funcionamiento del motor en conjunto con el driver es necesario limitar la corriente que circulará desde este último hasta las bobinas del motor. Una mala regulación puede dañar gravemente el motor. Para esta calibración se debe limitar la intensidad al 70 % del valor nominal del motor con el potenciómetro que dispone el driver,

$$I_{fijada} = 0,7 \cdot I_{nom} = 0,7 \cdot 0,4 = 0,28 \text{ A} \tag{3.1}$$

La fórmula que indica la tensión de alimentación del driver, según fabricante, es la siguiente.

$$I_{max} = \frac{V_{ref}}{8 \cdot R(\text{k}\Omega)} \tag{3.2}$$

donde I_{max} es la intensidad fijada calculada y V_{ref} la tensión a calcular. El valor de resistencia, en kΩ, es el de la resistencia del driver, que en este caso es de 100 Ω. Por ello, la tensión máxima de alimentación necesaria para este caso tiene el siguiente valor.

$$V_{ref} = I_{max} \cdot 8 \cdot 0,1 = 0,28 \cdot 8 \cdot 0,1 = 0,224 \text{ V} \tag{3.3}$$

Un vez obtenidos los valores límites, se deben seguir los siguientes pasos para realizar la calibración.

- Alimentar el driver esta el motor desconectado.
- Medir tensión entre el potenciómetro y neutro.
- Ajustar el potenciómetro hasta que la tensión sea la deseada.
- Quitar la alimentación.
- Conectar el motor, midiendo corriente en una de sus bobinas.
- Alimentar el driver de nuevo y terminar de ajustar el potenciómetro con la intensidad.
- Retirar la alimentación y realizar el conexionado definitivo.

Terminado este proceso, el driver esta completamente listo para ser utilizado.

3.2. Encoder Incremental

El encoder incremental utilizado se puede encontrar en Amazon por 16,99 €. De estos elementos existen bastantes posibilidades en el mercado, por ejemplo, en RS existen modelos que incluyen las fase Z de posición pero con precios superiores a los 200 €. También existe la posibilidad de compra de un motor que lleve anclado el encoder y compartan eje de rotación. Este es uno de estos casos con un precio aproximado de 370 €.

Figura 3.4: Encoder utilizado.

El dispositivo utilizado no dispone de fase Z, por lo que será necesario implementar un punto de referencia en el sistema, bien con un pulsador, o bien con un final de carrera. Las salidas son de colector abierto NPN, lo que le permite trabajar en lógica de 3,3 V sin riesgo de dañar la placa. Las características dadas por el fabricante se muestran en las siguientes tablas.

Carac.	Valor
Pulsos por revolución	600
Alimentación	5-24 VDC
Vel. Máxima	6000 rpm
Frec. de respuesta	20 kHz

Tabla 3.5: Características del encoder incremental utilizado.

Carcac.	Valor
Dimensiones	39x35,5 mm
Diámetro del eje	6 mm
Altura del eje	13 mm
Longitud de los alambres	1880 mm

Tabla 3.6: Características físicas del encoder utilizado.

Color	Cable
Verde	Fase A
Blanco	Fase B
Rojo	+Vcc
Negro	GND

Tabla 3.7: Disposión de los cables del encoder utilizado.

Para poder realizar las medidas se debe acoplar el encoder al motor paso a paso. Para ello, se ha diseñado e impreso en 3D un juego de engranajes, además de una fijación entre los dos dispositivos para que estos estén anclados y siempre a la misma distancia el uno del otro. La

cara superior de la armadura del motor se encuentra a menor altura que la del encoder, por lo que es necesario salvar esta diferencia con la fijación. Además, para poder ajustar la distancia entre los elementos, mejorar el agarre entre los dos engranajes y encontrar la posición de fijación óptima los agujeros para los tornillos del motor están alargados. Los planos de los diseños se encuentran en el anexo de planos y en la figura 3.5 se muestra el diseño montado.

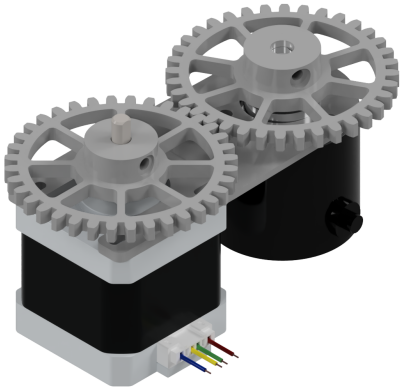


Figura 3.5: Acople y engranajes para el motor paso a paso y el encoder.

3.3. Raspberry Pi 3B+

Una Raspberry Pi es una placa computadora (SBC) de bajo coste. Tiene un tamaño aproximado de una tarjeta de crédito y fue desarrollado en el Reino Unido por la Fundación Raspberry Pi apoyada por la Universidad de Cambridge. El modelo utilizado es la última versión de la serie 3, la Raspberry Pi 3B+, con un precio de 33,12 € en RS. Las especificaciones de esta placa dadas por el fabricante se muestran en la tabla 3.8.

Figura 3.6: Raspberry Pi 3 B+.

Fuente: [15]

Especificación	Valor
Procesador	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4 GHz
Memoria	2,4 GHz and 5 GHz IEEE 802,11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE Gigabit Ethernet over USB 2.0 (velocidad máxima 300 Mbps) 4 × puertos USB 2.0
Acceso	40 pines GPIO
Video y Sonido	1 x HDMI MIPI DSI display port MIPI CSI camera port Salida estéreo de 4 polos y puerto de vídeo compuesto
Alimentación	5 V/2,5 A DC por conexión micro USB 5 VDC por pin GPIO Power over Ethernet (PoE)

Tabla 3.8: Especificaciones de la Raspberry Pi 3B+.

Fuente: [15]

3.3.1. Configuración

Para inciar la Raspberry Pi es necesario tener una micro SD con un Sistema Operativo instalado. La RPi soporta distintas distribuciones de GNU/Linux, pero la escogida ha sido Raspberry Pi OS, anteriormente conocido como Raspbian. Para ello, es necesario un software que permita escribir archivos .iso en la tarjeta de memoria. Un ejemplo de este tipo de software es balenaEtcher, desarrollado por balena y con licencia Apache License 2.0.

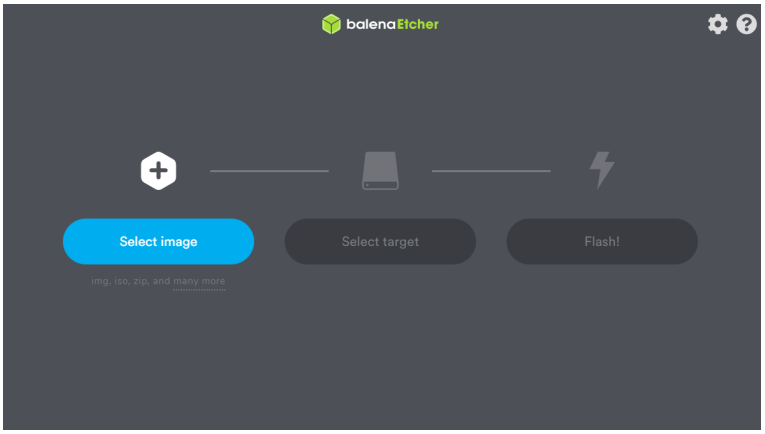


Figura 3.7: Interfaz gráfica de balenaEtcher.

Una vez grabado el SO en la micro SD, esta se conecta a la placa, junto un teclado, un ratón y un monitor y se termina de preparar el Sistema Operativo.

Para una mayor comodidad en el desarrollo, la Raspberry Pi se puede controlar de forma remota a través de una conexión SSH, pero para esto es necesario permitir el acceso por esta vía con anterioridad en las preferencias del sistema.

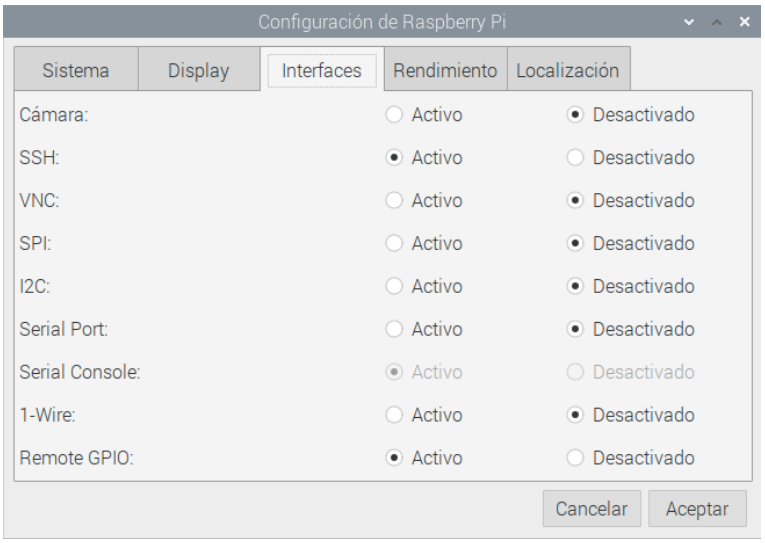


Figura 3.8: Configuración RPi SSH.

Una vez concedido el acceso es necesario conocer cual es la dirección IP del dispositivo, para esto, es necesario conectar la RPi por Ethernet a un ordenador, y dentro de la terminal de la Raspberry Pi escribir el comando `ifconfig`.

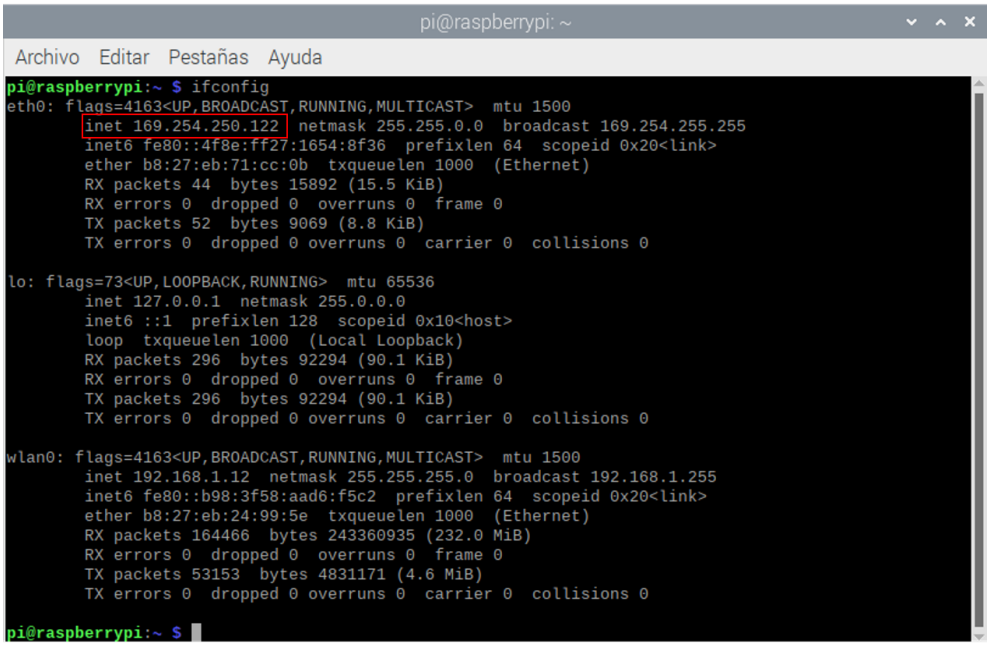


Figura 3.9: IP de la Raspberry Pi para la comunicación SSH.

En la figura 3.9, se observa como para la RPi utilizada la IP en cuestion es **169.254.250.122**.

3.4. Modbus

El protocolo de comunicación Modbus es una estructura de mensajería desarrollado por Modicon en el año 1979. Establece comunicaciones Maestro-Esclavo, producidas en pares, en las cuales, el maestro inicia la interacción y debe esperar la respuesta del esclavo. De forma habitual, el maestro es una interfaz humano-máquina (HMI) o un sistema SCADA y el esclavo un sensor, un Controlador Lógico Programable (PLC) o un Controlador de Automatización Programable (PAC) [1].

Los tres formados estándares de Modbus son TCP, Unidad Terminal Remota (RTU) y ASCII. En el caso de TCP en vez de hablar de comunicaciones Maestro-Esclavo se hace referencia a comunicaciones Cliente-Servidor.

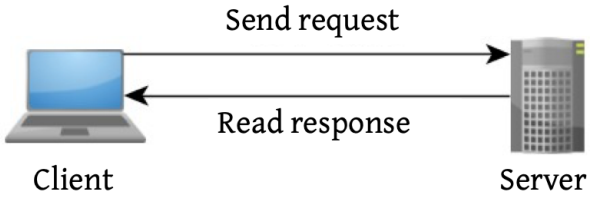


Figura 3.10: Comunicación Cliente-Servidor.

Fuente: [1]

El medio físico utilizado para TCP es Ethernet con RJ45 y para RTU RS232, RS422 o RS485 [16]. La Raspberry Pi 3B+ dispone de conexión a Ethernet, por lo tanto, se puede utilizar este puerto para así aplicar este protocolo basado en TCP. Si se quisiera utilizar Modbus RTU, por ejemplo, sería necesario la adquisición de una placa HAT (Hardware Attached on Top) preparada con para la conexión por RS485.

Para Modbus TCP, la Unidad de Datos de Aplicación (ADU), consiste en en el Encabezado de Protocolo de Aplicación Modbus (MBAP) combinado con la Unidad de Datos de Protocolo (PDU) de Modbus [1]. En la tabla 3.9 se muestra este formato.

Transaction	Protocol	Length	Unit ID	Modbus PDU

Tabla 3.9: ADU de Modbus TCP.

El primer identificador de transacción es valioso en una red en la cual se soportan múltiples solicitudes de forma simultánea, ya que con este término el cliente puede igualar la solicitudes con las respuestas aunque lleguen en un orden diferente al enviado. El campo de protocolo es normalmente cero, pero se puede modificar para ampliar el protocolo. El término de longitud

indica la propia longitud del resto del paquete. El identificador de unidad generalmente no se utiliza en los dispositivos TCP/IP, pero puede ser usado en un entorno en el que se permita la conexión con nuevas redes TCP/IP. Por último, la ADU incluye una PDU limitada a 253 bytes para el protocolo estándar [1].

Los datos disponibles en Modbus son almacenados en uno de los cuatro bloques de memoria disponibles. Cada tipo de bloque permite un tipo de dato y permite un acceso al Maestro/-Cliente y al Esclavo/Servidor distinto. Estos bloques se muestran en la tabla 3.10.

Bloque de Memoria	Tamaño	Acc. Cliente	Acc. Servidor
Bobinas (Coils)	1 bit	Lec./Esc.	Lec./Esc.
Entradas Discretas (Discrete Inputs)	1 bit	Solo Lec.	Lec./Esc.
Registros de Retención (Holding Registers)	16 bits	Lec./Esc.	Lec./Esc.
Registros de Entrada (Input Registers)	16 bits	Solo Lec.	Lec./Esc.

Tabla 3.10: Bloques del Modelo de Datos de Modbus [1].

Capítulo 4

Software desarrollado

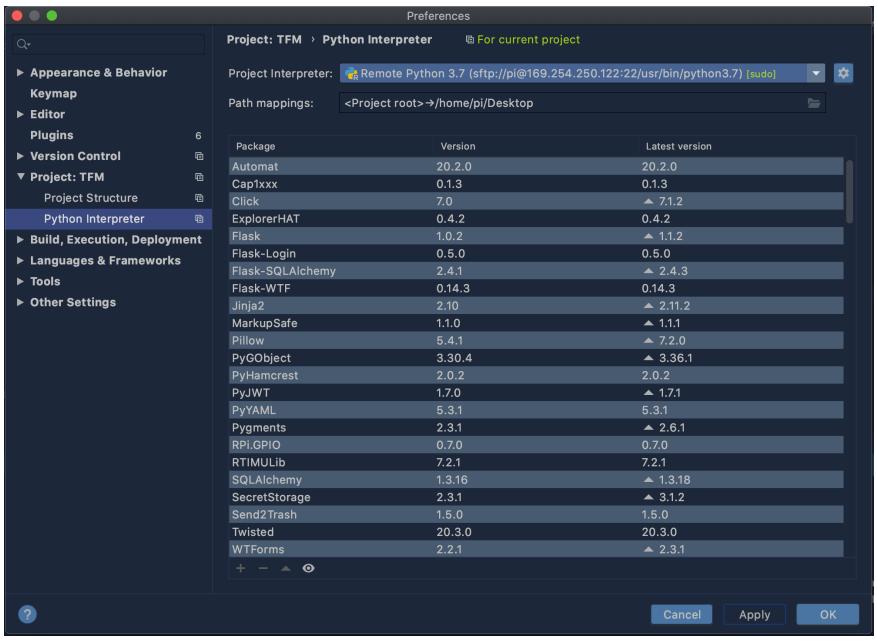
4.1. Python y PyCharm

El lenguaje de programación escogido para el desarrollo es Python en su versión 3.7 (fig. 4.1). Se trata de un lenguaje de programación interpretado, interactivo y orientado a objetos. Incorpora modulos, excepciones, escritura dinámica, tipos de datos dinámicos de muy alto nivel y clases. Combina una gran potencia con una sintaxis clara y fácil de interpretar. Es portable y funciona en muchas variedades de Unix, incluidos Linux y macOS además de Windows [17].

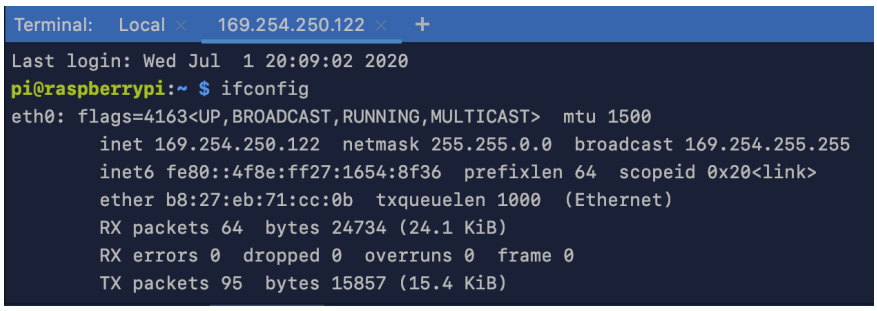
Figura 4.1: Logo de Python.

Existen muchos Entornos de Desarrollo Integrados (IDEs) capaces de soportar Python, pero el escogido es PyCharm (fig. 4.2), desarrollado de forma específica para este lenguaje. Este IDE permite tener la conexión SSH para control remoto de la RPi en su zona inferior dedicada a terminales y consolas. Además permite seleccionar un intérprete del lenguaje personalizado, dando la posibilidad de que este sea la propia Raspberry Pi a través del Protocolo de Transferencia de Archivos de SSH (SFTP).

Figura 4.2: Logo de PyCharm.



(a) Intérprete SFTP en PyCharm.



(b) Control remoto por SSH en PyCharm.

Figura 4.3: Conexión SSH e Intérprete SFTP en PyCharm.

4.2. Estructura

El esquema estructural del software desarrollado es el que se muestra en la figura 4.4. En ella se pueden ver los distintos módulos por los que está formado y la comunicación entre ellos además de los directorios y archivos contenidos.

El módulo estructural **Movement** es el principal del sistema. En él se contienen los archivos que definen las ordenes para el movimiento del motor. El módulo **Flask** define el servidor web desarrollado, mientras que **Modbus** es el módulo desarrollado para la implementación de este protocolo de comunicación.

YAML no es un módulo desarrollado en Python, si no que es un directorio contenedor de los distintos archivos, escritos en el formato que su nombre indica, que guardan la información necesaria para el correcto funcionamiento de los distintos módulos, además del archivo

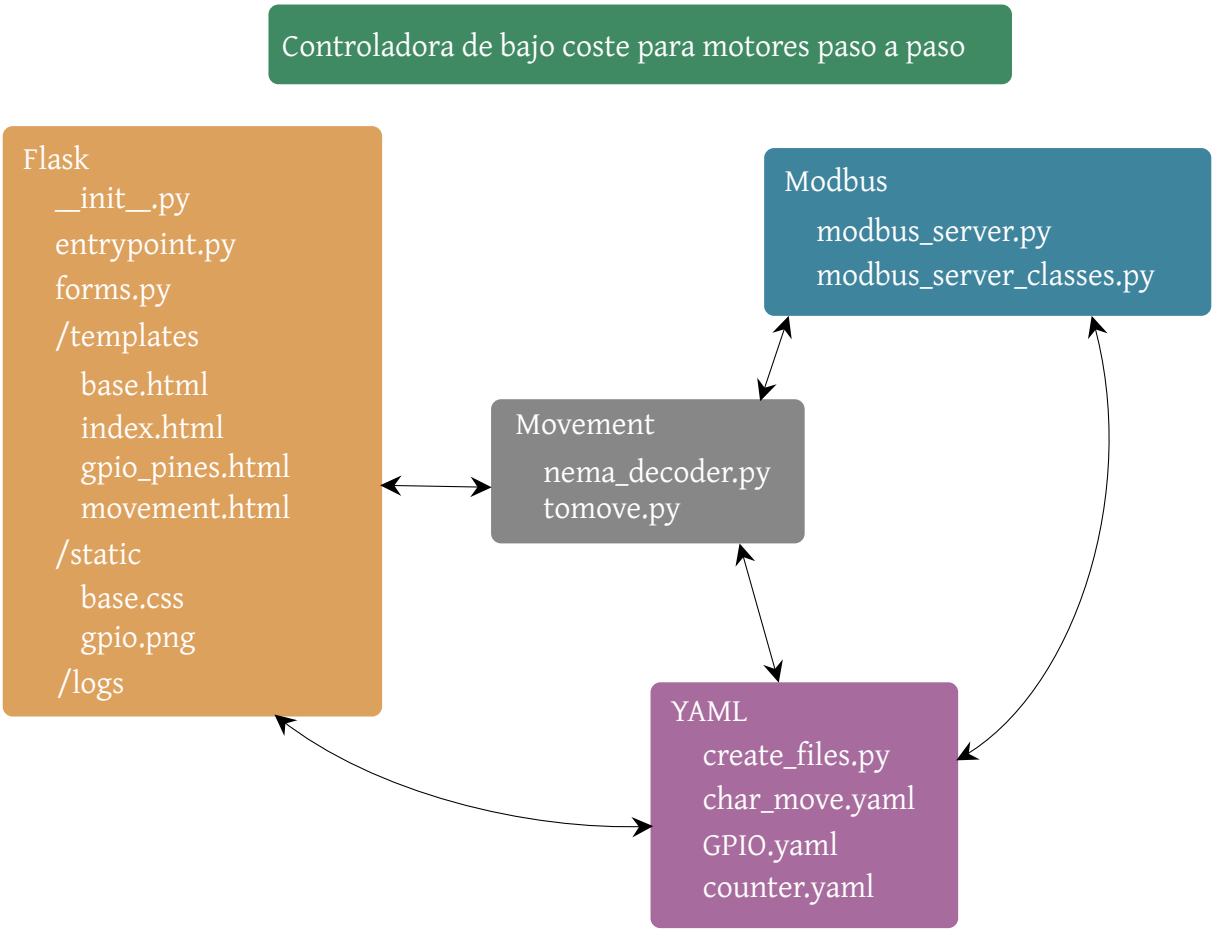


Figura 4.4: Estructura del software desarrollado.

create_files.py. Como se puede observar en la figura, todos los módulos se comunican con este directorio, pues estos archivos guardan en memoria cuáles son los pines de entrada y salidas de la placa a utilizar en el archivo **GPIO.yaml**, las características del movimiento a realizar en **char_move.yaml** y el estado del contador para el encoder en **counter.yaml**, permitiendo con este realizar un movimiento absoluto o incremental.

Todo el Software desarrollado tiene una copia en un repositorio alojado en GitLab. Para poder obtener solo es necesario clonar el siguiente enlace.

`https://gitlab.com/Pablo96ENG/TFM.git`

En el siguiente apartado se describen los distintos módulos de Python que permiten el desarrollo del software siguiente esta estructura definida.

4.3. Módulos utilizados

En Python, un módulo es un fichero contenedor de código del lenguaje, con la extensión .py, en el cuál se declaran variables, funciones, clases, etc, simplificando así el código a generar.

Existen gran cantidad de módulos, Open Source, con multitud de aplicaciones, con creadores ajenos a la Python Software Foundation. En los siguientes apartados se presentan cuáles son necesarias para el desarrollo de este proyecto.

4.3.1. RPi.GPIO

Esta libreria permite el control de los pines GPIO (General Purpose Input Output) de la Raspberry. Estos pines pueden ser utilizados como entradas o salidas digitales a la placa con lógica de 3,3 VDC. En la figura 4.5 se muestra cuáles son estos pines y cuál es su enumeración. Está puede tener dos referencias distintas, la numeración BCM y la BOARD. La BCM hace referencia al número GPIO asignado al pin, mientras que la BOARD al número del pin físico en la placa.

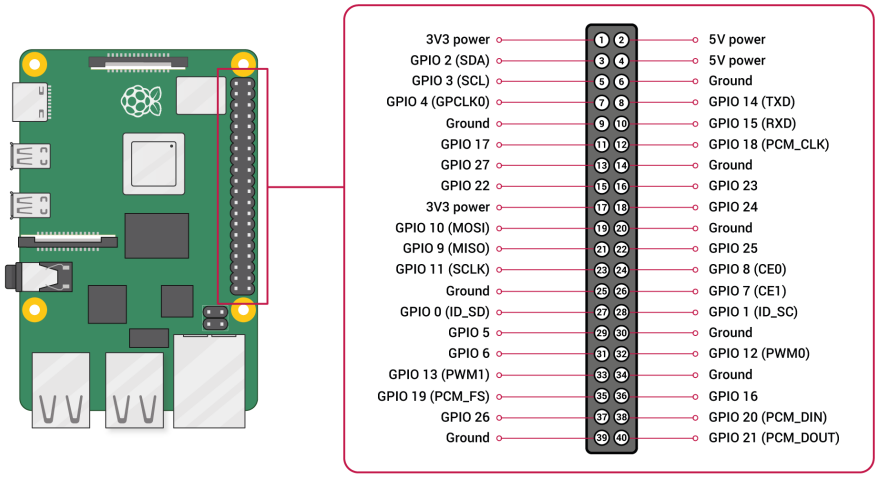


Figura 4.5: Pines GPIO de la placa.

Fuente: https://www.raspberrypi.org/documentation/usage/gpio/

Para instalar el modulo es necesario utilizar el comando `pip3 install RPi.GPIO` en la terminal.

Este módulo permite modificar el valor de las salidas, así como leer las entradas o esperar a que exista un cambio en alguna de estas. Algunos de los comandos utilizados son los mostrados a continuación.

```
1 import RPi.GPIO as GPIO # Importa el modulo como GPIO
2
3 GPIO.setmode(GPIO.BCM) # Selecciona la enumeracion para los pines
4 GPIO.setwarnings(False) # Desactiva ciertas alarmas que ocasionan problemas
5 GPIO.setup(pin, GPIO.OUT) # Indica que el pin es una salida
6 GPIO.setup(pin, GPIO.IN) # Indica que el pin es una entrada
```

```
GPIO.cleanup(pin) # Elimina la configuracion del pin
GPIO.output(pin, GPIO.HIGH) # Pone el pin de salida en valor alto
GPIO.output(pin, GPIO.LOW) # Pone el pin de salida en valor bajo
```

4.3.2. Pigpio

Este módulo permite, como el anterior, tener un control sobre los pines GPIO de la Raspberry Pi, pero además incluye ciertas clases predefinidas para el control de dispositivos. Para instalarlo es necesario descargar una copia de su repositorio de GitHub con los siguientes comandos en la terminal.

```
wget https://github.com/joan2937/pigpio/archive/master.zip
unzip master.zip
cd pigpio-master
make
sudo make install
```

Una vez instalada, es necesario iniciar el daemon de pigpio con `sudo pigpiod` . Un daemon es un tipo de proceso informático no interactivo, es decir, que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario. Tras esto se pueden utilizar las clases del modulo, algunas de estas son necesarias descargarlas de forma a parte y guardarlas en el directorio del proyecto. Este es el caso del objeto utilizado, que define un encoder rotacional incremental en codificación X4, es decir, teniendo en cuenta el estado de las dos fases A y B del mismo. Esta clase tiene el nombre de **rotary_encoder** y se puede descargar de la página oficial de Pigpio. Algunos comandos importantes de este módulo son los siguientes.

```
import pigpio # Importa el modulo principal
from rotary_encoder import decoder  # Importa la clase decoder del modulo

pi = pigpio.pi() # Inicializa la rpi como la placa para el control de los puertos de la
    libreria
encoder = decoder(pi, FaseA, FaseB, callback) # Crea la clase. Se introduce la rpi, los
    pines para las fases y una funcion callback que modifique el valor del contador

encoder.cancel() # Cierra la clase decoder
pi.stop() # Cierra el control de los pines GPIO
```

La clase **decoder** utiliza los distintos comandos de Pigpio que tienen la misma función que los enseñados en el módulo RPi.GPIO. Hay que tener en cuenta que este segundo módulo solo utiliza los pines en formato BCM. Estos son, con la nomeclatura de este módulo, los siguientes.

```
pi.set_mode(pin, pigpio.INPUT) # Indica que el pin es una entrada
pi.set_pull_up_down(pin, pigpio.PUD_UP) # Activa una resistencia pull up interna para el
    pin
```

```
3 pi.callback(pin, pigpio.EITHER_EDGE, callback) # Metodo de la rpi que permite activar la
      funcion callback con cualquier cambio de estado (EITHER_EDGE) del pin
```

4.3.3. Flask

Flask es un MicroFramework para el desarrollo de aplicaciones web. Un Framework es una colección de programas que trabajan en conjunto para cumplir una tarea, facilitar la programación y hacerla más eficiente. Que Flask se defina como un MicroFramework no significa que toda la aplicación web se pueda agrupar en un único archivo de Python, si no que trata de mantener el núcleo de la aplicación simple pero extensible [18]. Para instalarlo se utiliza `pip3 install Flask`.

Por defecto, Flask no incluye una capa de abstracción de base de datos, formularios o elementos que puedan ser creados a partir de librerías ya existentes. Si no que soporta extensiones que permiten añadir las funcionalidades necesarias a la aplicación como si fueran implementadas por si mismo. Esto permite incluir solo la información necesaria para su uso en concreto, liberando así memoria y capacidad de la aplicación. Estas extensiones permiten, entre otras cosas, la integración de una base de datos, formularios de validación y la utilización de distintas tecnologías abiertas de autentificación. Por ello, a pesar de ser un MicroFramework está preparado para la producción.

Figura 4.6: Logo de Flask.

Fuente: [18]

Para realizar formularios, se puede utilizar la extensión **wtforms** y crear una clase heredad del objeto **FlaskForm** de **flask_wtf**. Esta extensión permite incluir en el código aquellos campos (Submit, Integer, Boolean, TextArea) que sean necesarios para la aplicación además de las distintas validaciones que se necesiten en cada uno de estos campos.

Para interactuar con HTML5, se pueden escribir las plantillas combinándolas con Jinja2, que permite la inclusión de código escrito en Python en estas. Es un formato de escritura que se caracteriza por el uso de los símbolos de llave y porcentaje. Un ejemplo de este formato de escritura sería el siguiente.

```
1 <title>{% block title %}{% endblock %}</title>
2 <ul>
```

```
{% for user in users %}
  <li><a href="{{ user.url }}" > {{ user.username }}</a></li>
{% endfor %}
</ul>
```

4.3.4. Pymodbus

Pymodbus es un módulo que permite una completa implementación del protocolo Modbus en Python para todas las versiones superiores a la 2.7. Permite la creación de tanto Clientes como Servidores para el protocolo tanto síncronos como asíncronos. La diferencia entre estas opciones es que un método síncrono se espera en el punto de invocación hasta que se reciba una respuesta (resultado o error), mientras que un método asíncrono continua con la ejecución y la respuesta se recibirá en una función predefinida como callback.

Para instalar el módulo hay que realizar una copia del repositorio en GitHub, que descargará todas las dependencias necesarias para la versión de Python utilizada.

```
git clone git://github.com/bashwork/pymodbus.git
cd pymodbus
python setup.py install
```

Las características de los Clientes creados con este módulo son las siguientes [19].

- Protocolo de lectura/escritura completo para todos los tipos de datos.
- Incluye la mayor parte del protocolo ampliado (diagnóstico/archivos/pipes/configuración/información).
- TCP, UDP, Serial ASCII, Serial RTU y Serial Binario.
- Versiones asíncrona, con las librerías twisted, tornado y asyncio, y síncrona.

Las de los Servidores son las expuestas a continuación [19].

- Puede funcionar como un Servidor Modbus totalmente implementado.
- TCP, UDP, Serial ASCII, Serial RTU y Serial Binario.
- Versión asíncrona con twisted y síncrona.

Un ejemplo de código utilizado para la creación de un Cliente síncrono con Modbus TCP es el siguiente.

```
from pymodbus.client.sync import ModbusTcpClient

client = ModbusTcpClient('127.0.0.1')
client.write_coil(1, True)
result = client.read_coils(1,1)
print(result.bits[0])
client.close()
```

4.4. Movimiento del motor

El módulo **Movement** es el encargado de definir las distintas clases y funciones que permiten desarrollar un movimiento en el motor paso a paso. Está compuesto por dos archivos, **nema_decoder.py** y **tomove.py**. El primero contiene dos clases, **nema** y **decoder**, siendo la primera la que define el motor y la segunda el encoder. Como esta sección se centra en el movimiento del motor solo se explica la referente a este. El segundo archivo contiene las funciones **move** y **home**, la primera es la encargada de ordenar al motor el movimiento predefinido y la segunda de buscar la posición de referencia del sistema.

4.4.1. Clase nema

En esta clase se define en sus distintos métodos los pines GPIO ha utilizar para el movimiento del motor, y cómo debe hacerse dicho movimiento. En la figura 4.7 se muestra un esquema con los distintos métodos de la clase.

- **Método constructor**: Este método define cuales son las variables de entrada a la hora de crear al objeto, siendo para esta clase cuatro, **pinDIR**, **pinSTEP**, **pinRESET** y **pinENDSTOP**. Estos son los números de los pines, en formato **BCM** para dichas Entradas/Salidas que son definidas como tal.
- **change_STEP**: Sirve para cambiar el pin referente a la Salida STEP para el A4988.
- **change_DIR**: Cambia el pin para la Salida DIR del A4988.
- **change_RESET**: Modifica el valor de la Salida RESET hacia el A4988.
- **change_ENDSTOP**: Cambia el pin asociado a la Entrada del final de carrera de referencia.
- **delete**: Elimina todos los pines GPIO.
- **simple_move**: Ordena un movimiento sin retroalimentación, es decir, en lazo abierto. Sus entradas son la dirección de movimiento, 0 para giro en sentido antihorario y 1 para

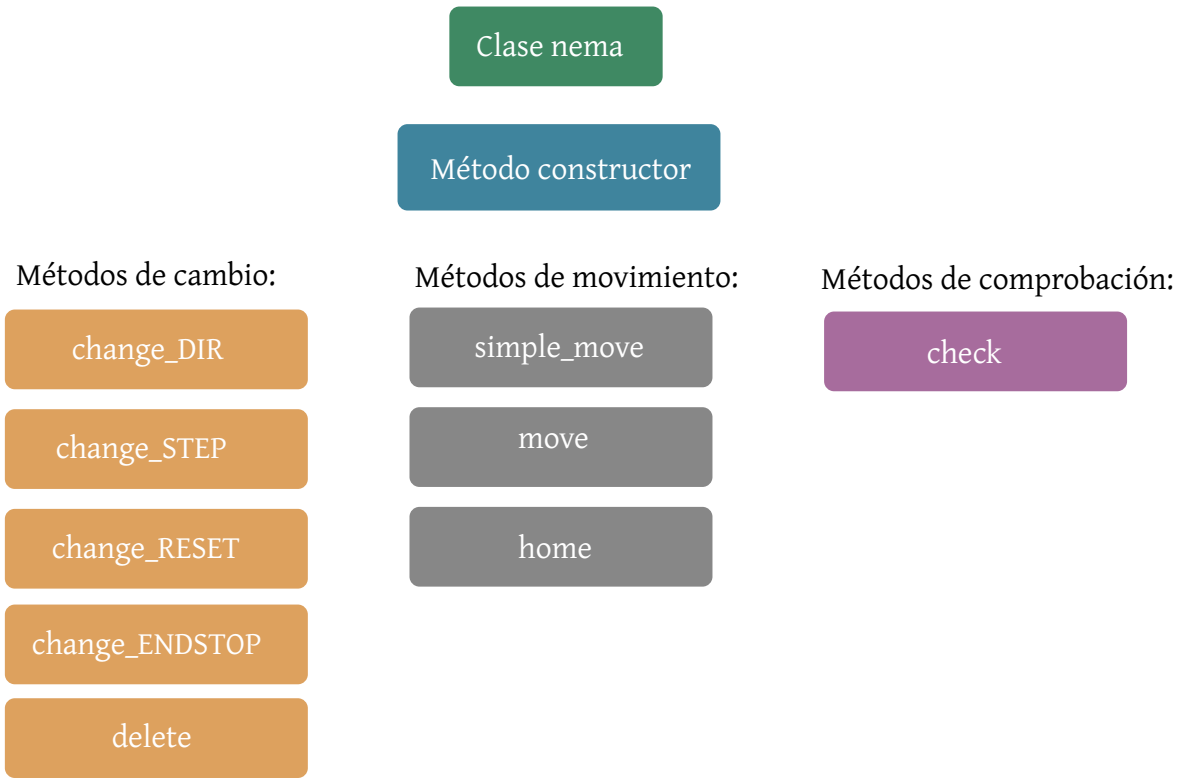


Figura 4.7: Clase nema.

giro en sentido horario; el número de pasos a mover y el tiempo de delay para la transición entre valor alto y bajo.

- **check**: Método utilizado para comprobar el estado del contador a partir de un estado de referencia.

- **move**: Ordena el movimiento hacia una nueva posición en lazo cerrado. Tiene múltiples entradas. La primera es el modo de movimiento, 0 para absoluto, 1 para incremental; la segunda el modo de desplazamiento, 1 si el valor a introducir será una distancia en mm, o 0 si será el número de pasaos a dar. La tercera entrada es el desplazamiento; la cuarta el contador del encoder; la quinta el delay en **milisegundos** (ms) entre estados alto y bajo de los pulsos del movimiento; la sexta cambia el sentido de giro con un valor alto; la séptima es la relación de movimiento que existe entre el encoder y el motor, 0 si estos giran en el mismo sentido o 1 si giran en sentido contrario y por último, si se activa el modo de desplazamiento en distancia, se introduce la relación pasos/mm que tiene el sistema.

- **home**: Este es el último método de la clase. Ordena el movimiento hacia la posición de referencia hasta que el final de carrera da un pulso alto.

4.4.2. Función home

Está función está definida en el archivo **tomove.py**. Sus entradas son los directorios en los que se encuentran los archivos **GPIO.yaml** y **counter.yaml**. Es la encargada de leer cuáles son los pines guardados y dar dichos valores como entrada a la clase **nema**. Además da el directorio del contador a su clase para posteriormente cargar el valor y reiniciarlo una vez se llegue a la posición de referencia.

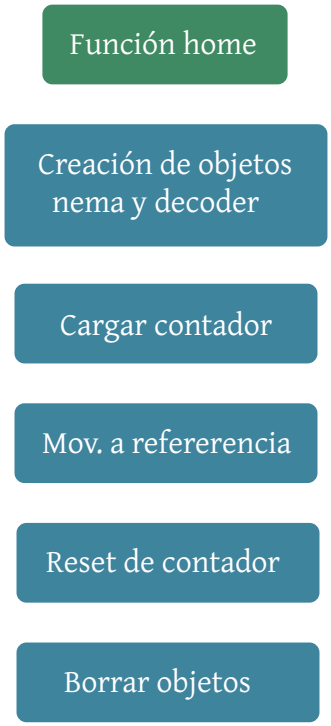


Figura 4.8: Función home.

4.4.3. Función move

Está definida en el archivo **tomove.py**. Sus entradas son los directorios contenedores de los archivos **GPIO.yaml**, **char_move.yaml** y **counter.yaml**. Como la función anterior, el directorio del contador es dado a la clase referente al contador mientras que los otros dos sirven para cargar sus datos internos. Se vuelve a crear el objeto **nema** y además se crea la clase **decoder** para el encoder cuyas entradas variarán según el valor definido para el sentido. Se hace uso del método **move** de la clase del motor con la que se ordena el movimiento en cuestión.

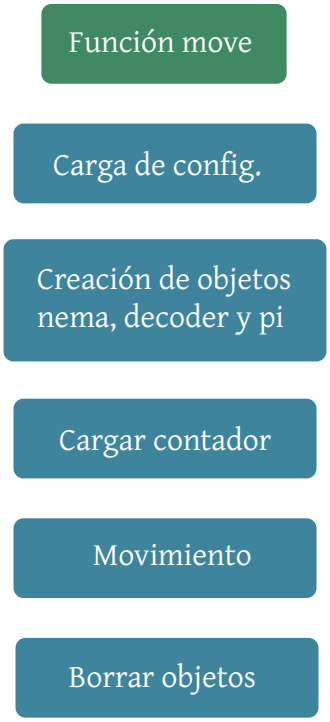


Figura 4.9: Función move

4.5. Funcionamiento del encoder

Para el correcto del funcionamiento son necesarias dos clases y un archivo. La clase principal es **decoder**, contenida en el archivo **nema_decoder.py** que es la clase perteneciente al módulo Pigpio para un encoder incremental en codificación X4. La segunda clase es **enc_counter**, encontrada en el archivo **tomove.py** que define el contador necesario para el control de movimiento. Por último, el archivo es **counter.yaml**, donde se almacena el valor del contador.

4.5.1. Clase decoder

Esta clase está interpreta el movimiento del encoder y llama al contador para aumentar o disminuir según sea este movimiento. Está formada por tres métodos.

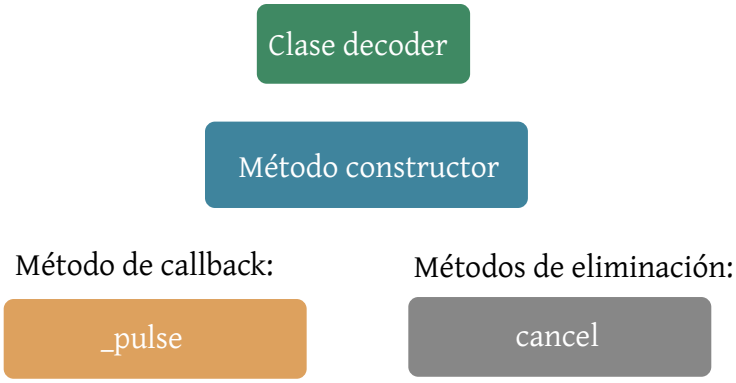


Figura 4.10: Clase decoder.

- El **método constructor** toma como entradas la clase **pi** del módulo Pigpio predefinida, los pines para la fase A y B del encoder y una función callback que modifique el valor del contador.
- **_pulse**: Utilizada como un callback propio dentro de la propia clase, tiene como entrada el pin de la fase a estudiar y el valor de este. Es utilizado en el método constructor y para el nivel se le introduce el valor pigpio.EITHER_EDGE, activándolo así cuando la Entrada correspondiente sufre un cambio de estado para la codificación X4. El método comprueba cuál ha sido el pin y su valor que lo ha llamado y lo compara con el estado de la otra Entrada, definiendo así si el contador debe aumentar o disminuir.
- **cancel**: Método que elimina los pines asociados a las fases del encoder.

4.5.2. Clase enc_counter

Esta clase define el contador necesario para la retroalimentación del movimiento.

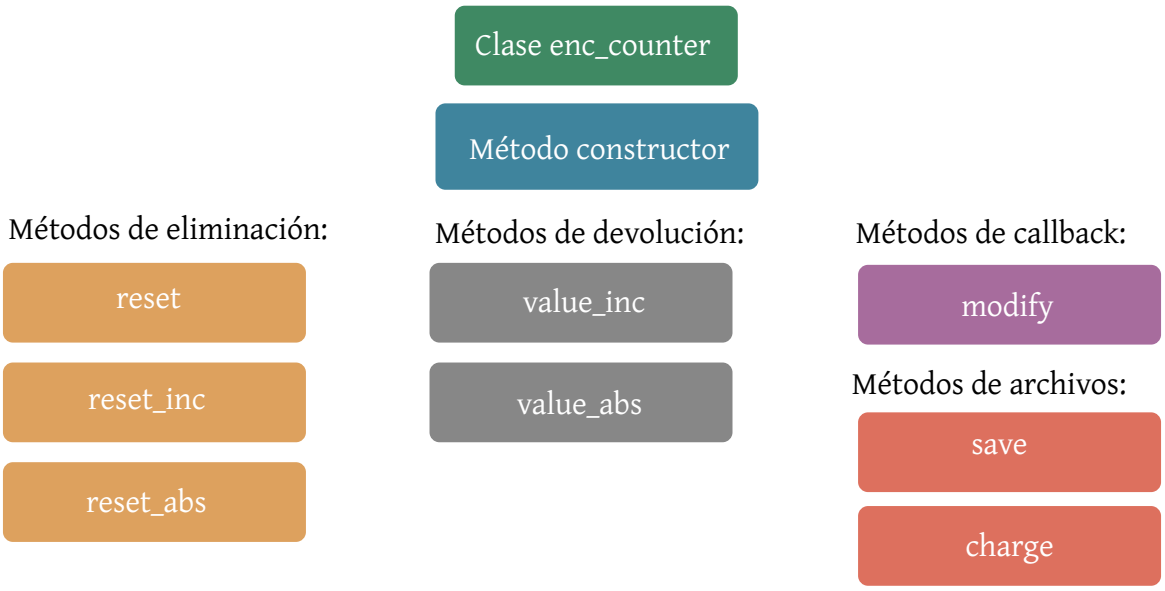


Figura 4.11: Clase enc_counter.

- **Método constructor**: Solo tiene como entrada el directorio en el que se encuentra el archivo **counter.yaml**. Al crear el objeto se inician las variables internas de los contadores absolutos e incremental en cero.
- **reset**: Reinicia el valor de las variables del contador incremental y absoluto.
- **reset_inc**: Reinicia el valor de la variable del contador incremental.
- **reset_abs**: Reinicia el valor de la variable del contador absoluto.
- **modify**: Este método modifica el valor de los contadores en función de la entrada (± 1). Se puede utilizar como función callback que modifique dichas variables.
- **value_inc**: Devuelve el valor del contador incremental.
- **value_abs**: Devuelve el valor del contador absoluto.
- **save**: Guarda el valor de los dos contadores en el archivo **counter.yaml**. Si este archivo no existe lo crea.
- **charge**: Carga los valores guardados en el archivo de memoria **counter.yaml** en las variables de cada uno de los contadores.

4.6. Servidor web

El servidor web está basado en el MicroFramework Flask y se puede acceder a él de forma remota a partir de la IP de la Raspberry Pi (160.254.250.122) y el puerto 5000. Está conformado por distintos archivos. **__init__.py** indica cuál es la carpeta principal de la aplicación para las posterior importación de los módulos creados; **entrypoint.py** es el archivo principal. En el se definen todas las páginas de la aplicación, las rutas y las funciones internas en cada de ellas; **forms.py** define los distintos formularios que componen el servidor. Estos sirven para moverse por sus distintas páginas y para la ejecución de funciones; El directorio **templates** contiene los archivos HTML combinados con Jinja2. Flask por defecto busca estos archivos en un directorio llamado de tal forma, aunque se puede modificar a criterio del creador; La carpeta **static** guarda los archivos estáticos de la aplicación web, estos son las imágenes y las hojas de estilo en CSS. Como con las plantillas, Flask busca este tipo de archivos por defecto en el directorio estático; Por último, el directorio contenedor **logs** almacenada los distintos archivos .log que almacenan la información de lo ocurrido en el servidor en el periodo que este ha estado encendido. Los nombres de estos archivos son la fecha y hora en la que han sido creados. Cada vez que se enciende el servidor se crea un nuevo archivo.

4.6.1. Archivo principal

El servidor web tiene un total de cuatro páginas. Estas se definen con el decorador app.route(), al cuál hay que darle de entrada un string que contenga el nombre de la página e indicarle los métodos GET y POST si la página requiere de un formulario. Estos son métodos de envío de información a los formularios. GET los envía de forma visible al navegador web, y se puede ver la URL, mientras que POST envía los datos ocultos.

Tras crear una página con el decorador, se define la función que ocurrirá dentro de la página correspondiente. Las páginas creadas son las siguientes.

- **Página principal**: Su dirección es la IP de la RPi junto con el puerto 5000. En esta se define la función **index** con un formulario con tres botones. El primer de ellos,sirve para llamar a la **/home**, el segundo a la página **/gpio_pines** y el último a la página **/Movement**.
- **/gpio_pines**: Esta página es la utilizada para modificar los pines asignados a cada una de las variables para el movimiento del motor y las fases del encoder. Cambia información con el archivo **GPIO.yaml**, si este no existe, lo crea.
- **/Movement**: Esta página se comunica con el archivo **char_move.yaml** creándolo si no existiera. Permite modificar las variables para el movimiento y se guardan dando la orden para que este ocurra.
- **/home**: Esta página ordena el movimiento del motor hasta la posición de referencia en la que se encontrará el final de carrera.

4.6.2. Formularios

Para el completo funcionamiento del servidor son necesarios un total de tres formularios. Estos se crean con forma de clases heredadas de la clase **FlaskForm** del módulo **flask_form**.

- **FirstForm**: Este es el formulario de la página principal. Se definen los tres botones como campos tipo **SubmitField** que invocan cada una de las distintas páginas.
- **FormMov**: Formulario utilizado para la definición de los parámetros del movimiento y para lanzar la orden de este. Para las variables del modo de movimiento, modo de desplazamiento, sentido y relación de movimiento entre motor y enconder se utilizan los campos **BooleanField**, ya que si no se marcan el resultado será un cero, mientras que si se marcan será un uno. Para las variables de desplazamiento, delay y relación pasos/mm se utiliza el campo **IntegerField** y son obligatorios. **Si no está activado el modo desplazamiento hay que introducir un cero para el campo de relación pasos/mm**. Finalmente, el botón que indica la orden es un campo **SubmitField**.

- **FormGPIO**: El último de los formularios, sirve para modificar los valores de los pines asignados en la placa para las variables del movimiento. Todos los campos son **IntegerField** salvo por el botón de confirmación que es un **SubmitField**.

4.6.3. Templates

En total hay cuatro de estos archivos, el primero, **base.html** sirve como plantilla para el resto de ellos; **index.html** es la página principal del servidor; **gpio_pines** la página con la que se pueden modificar los pines GPIO, en ella se encuentra la figura 4.5 como referencia y se muestra cuáles son los valores actualmente guardados, y por último, el archivo **movement.html** es el encargado de guardar los parámetros que definen el movimiento y dar la orden de iniciarlo.

```
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <title>{% block title %}{% endblock %}</title>
6      <link rel="stylesheet" href="{{ url_for('static', filename='base.css?v=23') }}">
7  </head>
8  <body>
9      {% block content %}{% endblock %}
10 </body>
11 </html>
```

4.6.4. Static

Este directorio sirve para guardar los archivos de tipo estático, es decir, las hojas de estilo en CSS y las imágenes. Para el desarrollo de la aplicación solo es necesario el uso de una hoja de estilo, **base.css**. En ella se definen los formatos, colores, fuentes y todo lo referente a la estética del servidor web. En las siguientes imágenes se muestran las distintas páginas con su estética.

Figura 4.12: Página principal del servidor web.

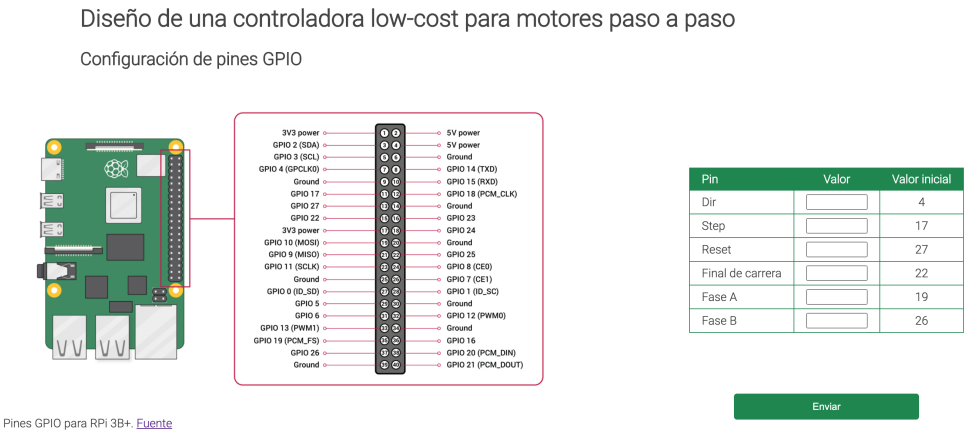


Figura 4.13: Página de edición de los pines GPIO del servidor web.

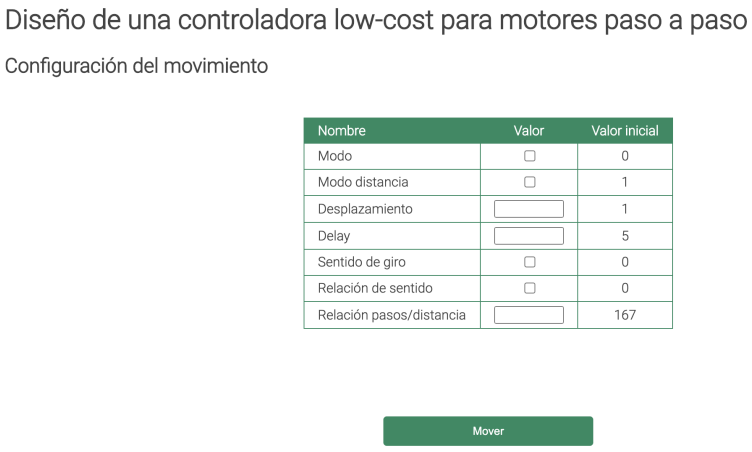


Figura 4.14: Página de configuración de movimiento del servidor web.

En la siguiente imagen se muestra cómo la acción de poner el cursor encima de un botón hace que este cambie su estética.

Figura 4.15: Cambio de color en botones con hover del servidor web.

4.7. Protocolo Modbus

La creación del Servidor Modbus se realiza gracias a la implementación del módulo Pymodbus. El Servidor utilizado es tipo asíncrono para así poder seguir ejecutando sin tener que esperar una respuesta por parte del Cliente. Para esto, se han creado dos archivos. **modbus_server.py** contiene dos clases, la función **set_slaves** y el código necesario para iniciar el Servidor Modbus. Las clases son **Home** y **Move** que simplemente sirven para almacenar un valor lógico sobre el estado de estas acciones; cuando este valor es uno se ha dado la orden por Modbus para activar la función con el mismo nombre del módulo Movement. El segundo archivo para el Servidor Modbus es **modbus_server_classes.py**, en el cual se definen una serie de clases que sirven para los distintos bloques de datos del protocolo.

El código necesario para la iniciación del servidor es el siguiente.

4.7.1. Función set_slaves

Esta función sirve para definir el diccionario **slaves**. Este diccionario contiene el contexto de los distintos esclavos creados. Para ello, es necesario importar las clases del segundo archivo creado para el módulo Modbus. Con la función se crean un total de tres unidades de esclavos que se encuentran tabulados en el Manual de Usuario.

- Esclavo **0x001**: Este esclavo se define un total de seis Bloques de Memoria de Registros de Retención (Holding Registers) empezando a enumerar por el registro cero. Se utiliza para almacenar los pines GPIO de la placa que se van a asociar al motor y al encoder.

- Esclavo **0x002**: Definido para cinco Bloques de Memoria de Registros de Retención enumerados desde el cero. Estos Registros se utilizan para la información del movimiento que se debe guardar en el archivo **char_move.yaml**.
- Esclavo **0x003**: A diferencia de los dos anteriores, este Esclavo utiliza dos Bloques de Memoria de Bobinas (Coils) con las que se dan las ordenes para el movimiento guardado o para el movimiento a la posición de referencia.

4.7.2. Clases del servidor

Se necesitan un total de tres clases para el Servidor Modbus. Las tres son clases creadas a partir de la herencia de las clases **ModbusSequentialDataBlock** y **ModbusSparseDataBlock** ya implementadas en el módulo Pymodbus. Se han añadido una serie de funciones en el método **setValues** de cada una de estas.

- Clase **GPIO_DataBlock**: Se utiliza para los Registros de Retención referentes a los pines GPIO. Hereda de la clase **ModbusSequentialDataBlock** y sirve para comunicarse con el archivo **GPIO.yaml** para almacenar los datos de los nuevos pines. Si el archivo no existe lo crea con unos valores por defecto.
- Clase **CharMove_DataBlock**: Utilizada para los Registros de Retención de las características del movimiento. Hereda de la clase **ModbusSequentialDataBlock**. Se comunica con el archivo **char_move.yaml** y si este no existe lo crea con todos los valores en cero.
- Clase **Move_DataBlock**: Es la utilizada para las ordenes de movimiento. Hereda de la clase **ModbusSparseDataBlock**, ya que solo se utilizan dos registros concretos que modifican el valor almacenado en los objetos **Home** y **Move** expuestos con anterioridad. Pone dichos valores en uno cuando se ordena el movimiento, y en cero cuando no.

Cabe la posibilidad de que por algún fallo, o por que el usuario lo ha decidido, los archivos YAML de memoria no existan. Para esto, el archivo **create_files.py** define las funciones necesarias para la creación de estos archivos con unos valores por defecto expuestos en el Manual de Usuario. El contenido de este archivo es el siguiente.

Capítulo 5

Pruebas de funcionamiento

Como pruebas del funcionamiento de la controladora se va a realizar una secuencia de movimientos, en la cuál se tenga que hacer uso de los dos modos de desplazamiento, por distancia y por pasos, y de los dos modos del encoder, incremental y absoluto. El sistema con el que se realiza la prueba está dispuesto de tal forma que el encoder gira en sentido contrario al motor, por lo tanto, la opción de relación de sentido deberá de tener un valor alto, al igual que que el modo de movimiento y el modo de desplazamiento si el caso lo requiere.

En primer lugar se va a probar a ir a la posición de referencia, para hacer la prueba la entrada del final de carrera se activa con un pulsador. En esta posición el encoder debe de resetearse. Los movimientos han sido grabados y subidos a Google Drive. Hay dos animaciones, una que muestra la terminal y otra que muestra el movimiento del motor. Se puede acceder a ellas con los enlaces remarcados en color azul o a partir de los siguientes códigos QR.

(a) Animación de la terminal

(b) Animación del motor

Figura 5.1: Pruebas para el desplazamiento a la posición de referencia.

El siguiente movimiento es con un desplazamiento de distancia. Se ha tomado el caso de husillo con un paso de 1,2 mm, por lo que la relación pasos/mm es de 167. En el Manual de Usuario se explica como obtener este tipo de relaciones. Se realiza un movimiento positivo de 2 mm en el eje del husillo, por lo tanto, el movimiento del motor debe ser en sentido opuesto

a la búsqueda de la posición de referencia. Se puede consultar la terminal y el motor.

(a) Animación de la terminal

(b) Animación del motor

Figura 5.2: Prueba de desplazamiento de 2 mm.

La siguiente posición es, en posición absoluta, 600 pasos. Debe de seguir siendo un movimiento de avance, por lo que debe tener el mismo sentido que el caso anterior. Esto se puede ver consultando la terminal y el movimiento del motor.

(a) Animación de la terminal

(b) Animación del motor

Figura 5.3: Prueba de desplazamiento a 600 pasos en absoluto.

Como el movimiento anterior fue con el encoder en contador absoluto, si se vuelve a ordenar el movimiento a la misma posición absoluta el motor debe permanecer quieto tal y como se muestra por terminal y con el motor.

(a) Animación de la terminal

(b) Animación del motor

Figura 5.4: Prueba del contador absoluto en la posición de 600 pasos.

Si se ordena el desplazamiento hasta la posición referente a 300 pasos desde la posición de referencia, el movimiento debe de ser negativo, pues se está en una posición más adelantada. Esto implica, que en la terminal se debe mostrar como el contador se reduce, mientras que el motor gira en sentido contrario a los casos anteriores.

(a) Animación de la terminal

(b) Animación del motor

Figura 5.5: Prueba de desplazamiento negativo hasta la posición absoluta de 300 pasos.

Activando el movimiento incremental, se avanza 200 pasos desde la posición actual. En la terminal se debe mostrar el contador incremental y el desplazamiento del motor debe ser en sentido contrario al anterior.

(a) Animación de la terminal

(b) Animación del motor

Figura 5.6: Prueba de desplazamiento incremental de 200 pasos sobre la posición actual.

En primer lugar se estaba en la posición 300 absoluta, por lo tanto, al moverse 200 pasos de forma incremental se ha avanzado a la posición 500 en absoluto, por lo tanto, si se ordena el desplazamiento a esta posición el motor debe estar quieto y en la terminal no mostrar cambio.

(a) Animación de la terminal

(b) Animación del motor

Figura 5.7: Prueba de posición absoluta tras movimiento incremental.

Finalmente, se prueba desde la última posición desplazada a volver a la posición de referencia. El motor no debe de parar de moverse hasta que se active el final de carrera. Se puede ver este desplazamiento en la terminal y en el desplazamiento del motor.

(a) Animación de la terminal

(b) Animación del motor

Figura 5.8: Prueba de movimiento final a la posición de referencia.

Como se puede observar en todas las animaciones, los movimientos concuerdan con como se debería de comportar la controladora. En este enlace y en el siguiente código QR se puede ver la secuencia de movimientos del motor completa.

Figura 5.9: Secuencia de movimientos completa de las pruebas de la controladora.

Capítulo 6

Manual de Usuario

Índice general

6.1. Contenido de los archivos de memoria

La controladora está formada por tres módulos escritos en Python y un directorio de memoria. Este directorio contiene tres archivos en formato YAML, **GPIO.yaml**, **char_move.ymal** y **counter.yaml**, que se utilizan para la configuración de la controladora. El archivo **GPIO.yaml** contiene cuales son los pines GPIO utilizados, en formato BCM, para el control del motor paso a paso y los pines para la lectura de las dos fases del encoder incremental; **char_move.yaml** contiene los parámetros necesarios para el movimiento del motor paso a paso y por último, **counter.yaml** guarda la información de los contadores del enconder para las dos posibilidades de movimiento, incremental y absoluto.

El formato escogido para estos archivos, YAML, se debe a que está diseñado para ser legible y se inspira en los diccionarios de Python entre otros, por lo que lo hace ideal para guardar la información de estos. Al recuperar su información interna se genera directamente un diccio-

nario y solo es necesario saber el nombre del campo a buscar en él para obtener sus valores asociados.

6.2. Valores por defecto

Cuando los archivos de memoria son creados por primera vez o vuelven a ser creados por las funciones internas de la controladora tras ser eliminados por el propio usuario, se generan con unos valores por defecto. En las tablas 6.1, 6.2 y 6.6 se muestran cuales son estos.

Pin	Valor por defecto
pinDIR	4
pinSTEP	17
pinRESET	27
pinENDSTOP	22
FaseA	19
FaseB	26

Tabla 6.1: Valores por defecto de los pines GPIO.

Carac.	Valor por defecto
Mode	False
Dis_mode	False
Displacement	0
Delay	0
Sense	False
Motion	False
Rel	0

Tabla 6.2: Valores por defecto de las características del movimiento.

Contador	Valor por defecto
Incremental	0
Absoluto	0

Tabla 6.3: Valores por defecto de los contadores.

6.3. Configuración del movimiento

Para configurar el movimiento, ya sea a través del Servidor web, del Servidor Modbus o editando el propio archivo de memoria se deben modificar los mismo campos. Al final, todos

los métodos de comunicación con el motor hacen uso del archivo **char_move.yaml**, por lo que los campos a configurar son los mismos.

- **Mode**: Hace referencia al modo de movimiento, 1 para incremental, 0 para absoluto.
- **Dis_mode**: Modo de desplazamiento, 1 para introducir el desplazamiento en unidades de distancia, 0 para introducir el número de pasos a desplazar.
- **Displacement**: Desplazamiento a realizar. Si la controladora se encuentra en modo de desplazamiento por distancia se debe introducir en **mm** y en caso contrario en pasos.
- **Delay**: Tiempo de espera entre pulso alto y pulso bajo. Se debe introducir en **ms**.
- **Sense**: Activar este modo hace implica que se ha cambiado el sentido de giro del sistema. Es decir, un desplazamiento positivo al activarlo sería negativo.
- **Motion**: Relación del sentido de giro entre el motor paso a paso y el encoder. Un valor alto significa que giran en sentido opuesto, mientras que un valor bajo significa que giran en el mismo sentido.
- **Rel**: Relación pasos/mm del sistema. Tiene que ser un valor entero y estar siempre implementado, a pesar de que la controladora se encuentre en modo de desplazamiento por pasos. En cualquier caso se recomienda introducir siempre el valor de la relación real del sistema.

6.4. Implementación de distancias

En la configuración de movimiento se puede escoger el tipo de desplazamiento. Este puede ser por **pasos** o por **mm**. Para activar el desplazamiento en unidades de distancia hay que indicar un valor alto dicho modo, si se mantiene un valor bajo, el desplazamiento deberá ser dado en pasos a mover.

Si se activa el desplazamiento en distancia, es necesario indicar cuál es la relación del sistema en pasos/mm. A continuación se exponen dos ejemplos de cómo poder calcular esta relación.

6.4.1. Husillo acoplado al eje del motor

Para ver cuál es la relación pasos/mm de este sistema basta con saber cuál es el paso que tiene el husillo y el número de pasos por vuelta del motor, dividirlos y coger rendondear a un valor entero. El paso define la distancia desplazada por vuelta, es decir, si el husillo tiene un

paso de 1,2 mm, quiere decir, que se avanza 1,2 mm en una vuelta del mismo. Si el motor tiene 200 pasos/vuelta, como el utilizado para este trabajo, esta relación será la siguiente.

$$r = \frac{200 \text{ pasos/vuelta}}{1,2 \text{ mm/vuelta}} = 166,67 \approx 167 \text{ pasos/mm} \tag{6.1}$$

6.4.2. Polea acoplada al eje del motor

El uso de una polea acoplada a un motor paso a paso que realice el desplazamiento a partir de una correa dentada es bastante usual, sobre todo en las impresoras 3D. Para ver cuál es la relación pasos/mm de este sistema es necesario saber el número de dientes de la polea, la distancia desplazada por diente y el número de pasos del motor por vuelta. Utilizando a modo de ejemplo, una polea GT2, que tiene un desplazamiento de 2 mm por diente, y de 20 dientes con un motor de 200 pasos por vuelta, la relación será la siguiente.

$$r = \frac{200 \text{ pasos/vuelta}}{20 \text{ diente/vuelta} \cdot 2 \text{ mm/diente}} = 5 \text{ pasos/mm} \tag{6.2}$$

6.5. Servidor web

Para lanzar el servidor por terminal es necesario indicar cuál es el archivo principal de la aplicación para posteriormente hacer que esta arranque. Para ello es necesario desplazarse por terminal hasta la carpeta del módulo Flask y ejecutar los siguientes comandos en orden.

```
export FLASK_APP="entrypoint.py"
flask run --host 0.0.0.0
```

Esto hace que Flask interprete el archivo **entrypoint.py** como su archivo principal, es decir, donde están creadas todas las páginas y funciones, y que se encienda el servidor en local. Por defecto, el puerto de este servidor será el **5000**, por lo que para acceder a este desde el navegador de un ordenador externo es necesario estar conectado a la misma red que la RPi e introducir como dirección la ip de la Raspberry Pi seguido de este puerto. Para este caso en concreto sería la dirección **160.254.250.122:5000**.

Si se desea modificar los pines utilizados para el control de Entradas y Salidas desde la página del servidor web, es necesario introducir un valor a cada uno de ellos. De lo contrario, saltará un mensaje de error por pantalla indicando que es necesario. Si no se desea modificar todos los pines basta con copiar el valor de la columna de la derecha.

Pin	Valor	Valor inicial
Dir	4	4
Step	15	17
Reset	12	27
Final de carrera	22	22
Fase A	19	19
Fase B	26	26

Enviar

Figura 6.1: Ejemplo de formulario de cambio de pines bien rellenado.

En la página de configuración de movimiento, los campos valores obligatorios son los referentes al desplazamiento deseado, delay y relación pasos/mm; el resto de campos, al ser de tipo booleano, se toman como valor bajo si no son activas. Estos tres campos mencionados son obligatorios debido a que son campos de números enteros y Flask no permite que el campo esté vacío, sin embargo, para el caso del campo de relación pasos/mm, aunque se rellene, si no se activa el modo de desplazamiento en distancia se obviará cualquier valor que se introduzca. Aún así, se recomienda introducir el valor real de esta relación para el sistema en cualquier caso.

Nombre	Valor	Valor inicial
Modo	☑	0
Modo distancia	☐	1
Desplazamiento	200	1
Delay	5	5
Sentido de giro	☐	0
Relación de sentido	☐	0
Relación pasos/distancia	167	167

Mover

Figura 6.2: Ejemplo de formulario de movimiento bien rellenado.

Figura 6.3: Ejemplo de error por no rellenar no un campo obligatorio.

6.6. Servidor Modbus

El Servidor Modbus es del tipo asíncrono, es decir, no necesita esperar a un respuesta por parte del Cliente para poder continuar con la ejecución. Tiene tres Unidades Esclavas internas para cada tipo de información necesaria, pines, características de movimiento y las ordenes. Las Unidades 0x001 y 0x002 utilizan Bloques de Memoria de Registros de Retención y son para los pines GPIO y para las características del movimiento respectivamente. La Unidad 0x003 utiliza Bloques de Memoria de Bobina para la orden de movimiento a preconfigurado o para el movimiento a la posición de referencia. En las siguientes tablas se muestran cuales son la relaciones entre los Registros/Bobinas de cada Unidad Esclava.

Pin	Registro
DIR	0
STEP	1
RESET	2
ENSTOP	3
Fase A	4
Fase B	5

Tabla 6.4: Pines asociados a cada Registro de Rentención de la Unidad Esclava 1 (0x001).

Carac.	Registro
Modo de mov.	0
Modo de despl.	1
Desplazamiento	2
Delay	3
Rel. de giro motor/encoder	4
Sentido de giro	5
Relación pasos/mm del sis.	6

Tabla 6.5: Características del movimiento asociadas a cada Registro de Retención de la Unidad Esclava 2 (0x002).

Función	Bobina
Home	0
Move	1

Tabla 6.6: Función asociada a cada bobina de la Unidad Esclava 3 (0x003).

Capítulo 7

Conclusiones

7.1. Discusión de resultados

Del funcionamiento y comportamiento de la controladora tras su desarrollo se pueden sacra las siguientes conclusiones.

- La controladora permite de forma efectiva la configuración de movimientos por distancias o por pasos. Esto le da gran versatilidad para el desarrollo de distintos movimientos. Además la forma en la que esta configurado el modo de desplazamiento por distancia le otorga una gran adaptabilidad, pues permite personalizar este tipo de movimiento con el sistema, así, si este tiene que cambiar la herramienta de movimiento (husillo, correa, etc) se puede adaptar a este cambio, estando también la posibilidad de utilizar la misma controladora para distintos sistemas en diferentes instantes de tiempo y solo tener que cambiar un campo para que se adapte.

- A pesar de utilizar un encoder rotacional de tipo incremental, la implementación de un archivo de memoria en la controladora permite utilizar este encoder como si se tratara de un encoder absoluto. El contador del movimiento se va guardando en dicho archivo conociéndose así la posición absoluta en la que se encuentra el sistema, ya se haya hecho el desplazamiento en modo incremental o en modo absoluto. Hay que incluir que la disponibilidad de estos dos modos de movimiento aumenta la adaptabilidad y posibilidades de configuración de la controladora.

- La controladora también dispone de distintos modos de configuración para el acople motor/encoder o para el sentido de giro del sistema. Esto está enfocado en la adaptabilidad de la controladora, pues se busca poder controlar cualquier tipo de sistema, y que además, si este sufre algún cambio en su cadena movimiento, la propia controladora pueda asumirlos sin necesidad de una modificación del software.

- Para la comunicación del sistema y control de motor se han creado dos posibilidades, un Servidor web y a través del protocolo de comunicación Modbus. Ambos métodos son

completamente funcionales y permite las mismas configuraciones y modificaciones al sistema. Así se puede controlar un motor desde un ordenador que esté comunicado con la controladora o se puede hacer este control a partir del uso de un PLC que tenga implementado este protocolo y se pueda comunicar con los registros y bobinas correspondientes del sistema, permitiendo así la inclusión de esta controladora en la automatización de sistemas reales.

7.2. Futuros trabajos

Como futuros trabajos o futuras líneas para profundizar más en el desarrollo de una controladora para motores paso a paso se hacen las siguientes propuestas.

- Implementar el uso de micropasos para aumentar la resolución disponible del motor. Estos permiten un aumento notable de los pasos por vuelta del motor. Este aumento depende en gran parte de las posibilidades que ofrezca el driver utilizado. Por ejemplo, para el A4988 se puede llegar a multiplicar por 16, mientras que en algunos drivers más avanzados se puede llegar a múltiplos de 32 e incluso 64.
- Realizar de una PCB HAT que permita concentrar todos los componentes electrónicos necesarios encima de la Raspberry Pi y no sea necesario el uso de placas externas.
- Crear una carcasa para la controladora modificable y adaptable según lo requiera el sistema al que este destinado el uso.
- Probar el comportamiento de la controladora con distintos motores, drivers y encoders comerciales.
- Probar el funcionamiento del sistema con distintos acoples entre el motor y el encoder, como por ejemplo, una polea con correa.

Bibliografía

[1] NI. Información detallada sobre el protocolo modbus. https://www.ni.com/es-es/innovations/white-papers/14/the-modbus-protocol-in-depth.html. Accedido el 30 de Junio. VII, 22, 23

[2] Oriental motor. Stepper motors. https://www.orientalmotor.eu/Products/Stepper_motors/, 2020. Accedido el 21 de Junio de 2020. 1

[3] Steve Jennings. Motores paso a paso. *Informador Técnico*, 65:47–58, 2002. 3, 4, 5

[4] Ingeniería Mecafenix. Motor paso a paso ¿qué es y como funciona? https://www.ingmecafenix.com/electricidad-industrial/motor-paso-a-paso/, 2017. Accedido el 29 de Junio. 3, 4

[5] Manuel Delgado Crepo. Motores paso a paso. http://manueldelgadocrespo.blogspot.com/p/motores-paso-paso.html, 2019. Accedido el 29 de Junio. 3, 5

[6] Ingeniería Mecafenix. Encoder, ¿cómo funciona? y sus tipos. https://www.ingmecafenix.com/automatizacion/encoder/, 2017. Accedido el 30 de Junio. 6

[7] Jean Pollefliet. 18 - current -, angular position -, speed transducers. In Jean Pollefliet, editor, *Power Electronics*, pages 18.1 – 18.34. Academic Press, 2018. 6

[8] Danielle Collins. Faq: What do x1, x2 and x4 position encoding mean for incremental encoders? https://www.motioncontroltips.com/faq-what-do-x1-x2-and-x4-position-encoding-mean-for-incremental-encoders/, 2015. Accedido el 25 de Marzo. 7

[9] Gabriel Tolosa. Protocolos y modelo osi. Recuperado de http://www.tyr.unlu.edu.ar/TYR-publica/02-Protocolosy-OSI.pdf, 2014. 8

[10] Adriana Torres. Comparación entre el modelo osi y el modelo tcp/ip. https://interpolados.wordpress.com/2017/03/01/comparacion-entre-el-modelo-osi-y-el-modelo-tcpip/, 2017. Accedido el 30 de Junio. 8

[11] SMC. SMC JXC73/83 Series Operation Manual. https://static.smc.eu/binaries/content/assets/smc_global/product-documentation/operation-manuals/en/om_jxc73-87_parallel-io_en-b.pdf, Febrero 2019. Accedido el 26 de Marzo de 2020. 9

[12] SMC. Controlador del motor paso a paso (modelo de entreada de pulsos). https://static.smc.eu/pdf/LECPA_ES.pdf. Accedido el 26 de Marzo de 2020. 9, 10

[13] Oriental motor. Az series closed loop stepper motor drivers (ac input). https://www.orientalmotor.com/stepper-motors/closed-loop-drivers-az-ac.html, 2018. Accedido el 21 de Junio de 2020. 10

[14] LLC Allegro MicroSystems. A4988 datasheet - dmos microstepping driver with translator and overcurrent protection. https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf. Accedido el 30 de Junio. 15

[15] Raspberry Pi Foundation. Raspberry pi 3 model b+. https://static.raspberrypi.org/files/product-briefs/200206+Raspberry+Pi+3+Model+B+plus+Product+Brief+PRINT&DIGITAL.pdf. Accedido el 1 de Julio. 19, 20

[16] Área de Sistemas y Automática. Redes de comunicaciones industriales. buses de campo y redes lan industriales. 22

[17] Python Software Foundation. General python faq. https://docs.python.org/3/faq/general.html. Accedido el 1 de Julio. 25

[18] Pallets. Flask, foreword. https://flask.palletsprojects.com/en/1.1.x/foreword/. Accedido el 2 de Julio. 30

[19] Sanjay. Pymodbus - a python modbus stack. https://pymodbus.readthedocs.io/en/latest/readme.html. Accedido el 1 de Julio. 31

[20] The Modbus Organization. Modbus faq: About the protocol. http://www.modbus.org/faq.php. Accedido el 30 de Junio.

Anexos

Anexo A

Código fuente desarrollado completo

En este anexo se incluye el código fuente, escrito en Python, de cada una de las clases, funciones y páginas escritas.

A.1. Clase nema

```
1  import RPi.GPIO as GPIO
2  import pigpio
3  import time
4
5  class nema:
6
7      def __init__(self, pinDIR, pinSTEP, pinRESET, pinENDSTOP):
8          """pinDIR, pinSTEP y pinRESET are the GPIO pins in BCM format for A4899 inputs.
               pinENDSTOP
9          is the pin for the home endstop."""
10         self.pinDIR = pinDIR
11         self.pinSTEP = pinSTEP
12         self.pinRESET = pinRESET
13         self.pinENDSTOP = pinENDSTOP
14
15         self._HOMEDELAY = 0.01
16         self._STEPS = 200 # steps/rev
17
18         GPIO.setmode(GPIO.BCM)
19         GPIO.setwarnings(False)
20         GPIO.setup(pinDIR, GPIO.OUT)
21         GPIO.setup(pinSTEP, GPIO.OUT)
22         GPIO.setup(pinRESET, GPIO.OUT)
23         GPIO.setup(pinENDSTOP, GPIO.IN)
24
25     def change_STEP(self, pinNSTEP):
26         """Change the GPIO for STEP"""
27         GPIO.cleanup(self.pinSTEP)
```

```
28          self.pinSTEP = pinNSTEP
29          GPIO.setup(self.pinSTEP, GPIO.OUT)
30
31      def change_DIR(self, pinNDIR):
32          """Change the GPIO for DIR"""
33          GPIO.cleanup(self.pinDIR)
34          self.pinDIR = pinNDIR
35          GPIO.setup(self.pinDIR, GPIO.OUT)
36
37      def change_RESET(self, pinNRESET):
38          """Change the GPIO for RESET"""
39          GPIO.cleanup(self.pinRESET)
40          self.pinRESET = pinNRESET
41          GPIO.setup(self.pinRESET, GPIO.OUT)
42
43      def change_ENDSTOP(self, pinNENDSTOP):
44          """Change the GPIO for ENDSTOP"""
45          GPIO.cleanup(self.pinENDSTOP)
46          self.pinENDSTOP = pinNENDSTOP
47          GPIO.setup(self.pinENDSTOP, GPIO.OUT)
48
49      def delete(self):
50          """Clean up all GPIO"""
51          GPIO.cleanup()
52
53      def simple_move(self, dir, steps, delay):
54          """Move the stepper the quantity of steps given. No closed loop.
55          - dir has value 0 for turn left and 1 for turn right.
56          - steps the qty of steps to move.
57          - delay is the time in seconds between steps (speed)."""
58          if dir:
59              GPIO.output(self.pinDIR, GPIO.HIGH)
60          else:
61              GPIO.output(self.pinDIR, GPIO.LOW)
62
63          GPIO.output(self.pinRESET, GPIO.HIGH)
64
65          #steps = rev * self._STEPS
66
67          for i in range(int(steps)):
68              GPIO.output(self.pinSTEP, GPIO.HIGH)
69              print("high")
70              time.sleep(delay)
71              GPIO.output(self.pinSTEP, GPIO.LOW)
72              print("low")
73              time.sleep(delay)
74              print("Motor = {}".format(i))
```

```
75
76          GPIO.output(self.pinDIR, GPIO.LOW)
77          GPIO.output(self.pinRESET, GPIO.LOW)
78
79      def check(self, posA, posB, init_state):
80          """Check if the Movement must continue or not.
81          - posA is the current position
82          - posB is the position to move
83          - init_state 1 if posA was bigger on initial state, 0 if not"""
84          if init_state*(posA <= posB) or (not init_state)*(posA >= posB):
85              return 0
86          else:
87              return 1
88
89      def move(self, mode, dis_mode, posB, counter, delay, sense, motion, rel=None):
90          """Move from current position to a new one. Closed loop. Every call to the method
                moves one step.
91          - mode 0 for absolute mov. 1 for incremental
92          - dis_mode 0 for steps move, 1 for distance move
93          - posB is the position or the distance to move
94          - counter the encoder counter
95          - delay is the half period
96          - sense is the rotational sense of Movement
97          - motion is the sense of move between encoder and stepper
98          - rel is the relation between steps and distance of the system"""
99
100         if mode:
101             counter.reset_inc()
102             posA = counter.value_inc()
103         else:
104             posA = counter.value_abs()
105
106         if dis_mode:
107             posB = posB * rel
108
109         cond1 = (posB > posA and motion) and (not sense)
110         cond2 = (posB < posA and not motion) and (not sense)
111         if cond1 or cond2:
112             # dir = 1
113             GPIO.output(self.pinDIR, GPIO.HIGH)
114         else:
115             # dir = 0
116             GPIO.output(self.pinDIR, GPIO.LOW)
117
118         GPIO.output(self.pinRESET, GPIO.HIGH)
119         print(posA, posB)
120         init_state = posA > posB
```

```
121         move = self.check(posA, posB, init_state)
122 
123         while move != 0:
124             GPIO.output(self.pinSTEP, GPIO.HIGH)
125             print("high")
126             time.sleep(delay)
127             GPIO.output(self.pinSTEP, GPIO.LOW)
128             print("low")
129             time.sleep(delay)
130 
131             if mode:
132                 nposA = counter.value_inc()
133             else:
134                 nposA = counter.value_abs()
135 
136             print("Encoder = {}".format(nposA))
137             move = self.check(nposA, posB, init_state)
138             print(move)
139 
140         counter.save()
141         GPIO.output(self.pinDIR, GPIO.LOW)
142         GPIO.output(self.pinRESET, GPIO.LOW)
143 
144     def home(self):
145         """Move the system to home position. Waits to ENDSTOP."""
146         GPIO.output(self.pinDIR, GPIO.LOW)
147         GPIO.output(self.pinRESET, GPIO.HIGH)
148 
149         print("Moving to home position...\n")
150         while not GPIO.input(self.pinENDSTOP):
151             GPIO.output(self.pinSTEP, GPIO.HIGH)
152             time.sleep(self._HOMEDELAY)
153             GPIO.output(self.pinSTEP, GPIO.LOW)
154             time.sleep(self._HOMEDELAY)
155 
156         print("Done.\n")
157         GPIO.output(self.pinRESET, GPIO.LOW)
```

A.2. Función home

```
1 import pigpio
2 import yaml
3 import os
4 
5 def home(GPIO_filepath, counter_filepath):
```

```
6       with open(GPIO_filepath, 'r+') as f:
7           GPIO = yaml.load(f, Loader=yaml.FullLoader)
8
9       from .nema_decoder import nema
10
11      decoder_counter = enc_counter(counter_filepath)
12
13      nema = nema(GPIO["pinDIR"], GPIO["pinSTEP"], GPIO["pinRESET"], GPIO["pinENDSTOP"])
14      decoder_counter.charge()
15      nema.home()
16      decoder_counter.reset()
17      decoder_counter.save()
18
19      nema.delete()
```

A.3. Función move

```
1   import pigpio
2   import yaml
3   import os
4
5   def move(GPIO_filepath, char_move_filepath, counter_filepath):
6       with open(char_move_filepath, 'r+') as f:
7           char_move = yaml.load(f, Loader=yaml.FullLoader)
8
9       with open(GPIO_filepath, 'r+') as f:
10          GPIO = yaml.load(f, Loader=yaml.FullLoader)
11
12      from .nema_decoder import nema, decoder
13
14      decoder_counter = enc_counter(counter_filepath)
15
16      print("Initialazing...\n")
17      pi = pigpio.pi()
18      nema = nema(GPIO["pinDIR"], GPIO["pinSTEP"], GPIO["pinRESET"], GPIO["pinENDSTOP"])
19
20      decoder_counter.charge()
21
22      if char_move["Sense"]:
23          encoder = decoder(pi, GPIO["FaseB"], GPIO["FaseA"], decoder_counter.modify)  #
                Change phases
24      else:
25          encoder = decoder(pi, GPIO["FaseA"], GPIO["FaseB"], decoder_counter.modify)
26
```

```
27      nema.move(char_move["Mode"], char_move["Dis_mode"], char_move["Displacement"],
            decoder_counter,
28                  char_move["Delay"] / 1000, char_move["Sense"], char_move["Motion"],
                        char_move["Rel"])
29
30      print("Bye bye.")
31      encoder.cancel()
32      nema.delete()
33      pi.stop()
```

A.4. Clase decoder

```
1   import pigpio
2
3   class decoder:
4      """Class to decode mechanical rotary encoder pulses."""
5
6      def __init__(self, pi, gpioA, gpioB, callback):
7         """Instantiate the class with the pi and gpios connected to
8         rotary encoder contacts A and B.  The common contact
9         should be connected to ground.  The callback is
10        called when the rotary encoder is turned.  It takes
11        one parameter which is +1 for clockwise and -1 for
12        counterclockwise."""
13
14        self.pi = pi
15        self.gpioA = gpioA
16        self.gpioB = gpioB
17        self.callback = callback
18
19        self.levA = 0
20        self.levB = 0
21
22        self.lastGpio = None
23
24        self.pi.set_mode(gpioA, pigpio.INPUT)
25        self.pi.set_mode(gpioB, pigpio.INPUT)
26
27        self.pi.set_pull_up_down(gpioA, pigpio.PUD_UP)
28        self.pi.set_pull_up_down(gpioB, pigpio.PUD_UP)
29
30        self.cbA = self.pi.callback(gpioA, pigpio.EITHER_EDGE, self._pulse)
31        self.cbB = self.pi.callback(gpioB, pigpio.EITHER_EDGE, self._pulse)
32
33     def _pulse(self, gpio, level):
```

```
34        """Decode the rotary encoder pulse.
35
36                     +---------+         +---------+      0
37                     |         |         |         |
38           A         |         |         |         |
39                     |         |         |         |
40           +---------+         +---------+         +----- 1
41
42               +---------+         +---------+            0
43               |         |         |         |
44           B   |         |         |         |
45               |         |         |         |
46           ----+         +---------+         +---------+  1
47        """
48
49        if gpio == self.gpioA:
50           self.levA = level
51        else:
52           self.levB = level
53
54        if gpio != self.lastGpio: # debounce
55           self.lastGpio = gpio
56
57           if   gpio == self.gpioA and level == 1:
58              if self.levB == 1:
59                 self.callback(1)
60           elif gpio == self.gpioB and level == 1:
61              if self.levA == 1:
62                 self.callback(-1)
63
64     def cancel(self):
65        """ Cancel the rotary encoder decoder.
66        """
67
68        self.cbA.cancel()
69        self.cbB.cancel()
```

A.5. Clase enc_counter

```
1  import os
2  import yaml
3
4  class enc_counter:
5      def __init__(self, counter_filepath):
6          self.abs_counter = 0
```

```
7           self.inc_counter = 0
8           self.filepath = counter_filepath
9
10      def reset(self):
11          self.abs_counter = 0
12          self.inc_counter = 0
13
14      def reset_inc(self):
15          self.inc_counter = 0
16
17      def reset_abs(self):
18          self.abs_counter = 0
19
20      def modify(self, way):
21          self.abs_counter += way
22          self.inc_counter += way
23
24      def value_inc(self):
25          return self.inc_counter
26
27      def value_abs(self):
28          return self.abs_counter
29
30      def save(self):
31          if (not os.path.isfile(self.filepath)
32                  or not os.stat(self.filepath).st_size):
33              content = {"Absolute": 0,
34                         "Incremental": 0
35                         }
36              with open(self.filepath, 'w+') as f:
37                  yaml.dump(content, f)
38
39          with open(self.filepath, 'r+') as f:
40              content = yaml.load(f, Loader=yaml.FullLoader)
41              f.seek(0)  #
42              f.truncate()  # Erase content to rewrite
43              content["Absolute"] = self.abs_counter
44              content["Incremental"] = self.inc_counter
45              yaml.dump(content, f)
46
47      def charge(self):
48          if (not os.path.isfile(self.filepath)
49                  or not os.stat(self.filepath).st_size):
50              content = {"Absolute": 0,
51                         "Incremental": 0
52                         }
53              with open(self.filepath, 'w+') as f:
```

```
54                 yaml.dump(content, f)
55
56         with open(self.filepath, 'r+') as f:
57             content = yaml.load(f, Loader=yaml.FullLoader)
58             self.abs_counter = content["Absolute"]
59             self.inc_counter = content["Incremental"]
```

A.6. Archivo principal del servidor web

```
1  """Paquetes para Flask"""
2  from flask import Flask, render_template, url_for, redirect, request
3  from .forms import FormMov, FirstForm, FormGPIO
4  import logging
5  import shlex
6  import subprocess
7  import collections
8  import yaml
9  import os
10 import datetime
11
12 """Paquetes para movimiento"""
13 from Movement.tomove import move as tomove
14 from Movement.tomove import home as tohome
15 from YAML.create_files import create_files
16
17 app = Flask(__name__)
18 app.config[
19     "SECRET_KEY"] = "28
        dc8b38aad7999fb7655549d01a41bf9f7b88051f609f5579e09de7b68f5bda40352e0e0fdb3504622 5f9d1fd82bea63
        "
20
21 date = datetime.datetime.now()
22 date = str(date)
23 date = date.replace(" ", "_")
24 date = date[:-7]
25 filename = "logs/" + date + ".log"
26 f = open(filename, 'x')
27 f.close()
28
29 LOG_FORMAT = "[%(levelname)s] %(asctime)s - %(message)s"
30 logging.basicConfig(level=logging.DEBUG,
31                     format=LOG_FORMAT,
32                     )
33 log = logging.getLogger()
34 fh = logging.FileHandler(filename)
```

```
35 log.addHandler(fh)
36
37
38 @app.route("/", methods=["GET", "POST"])
39 def index():
40     form = FirstForm(request.form)
41     if form.validate_on_submit():
42         if form.gpio_pines.data:
43             return redirect(url_for("gpio_pines"))
44         elif form.movement.data:
45             return redirect(url_for("move"))
46         elif form.home.data:
47             return redirect(url_for("home"))
48
49     return render_template("index.html", form=form)
50
51
52 @app.route("/gpio_pines", methods=["GET", "POST"])
53 def gpio_pines():
54     form = FormGPIO(request.form)
55     error = None
56     if form.validate_on_submit():
57         pins = [form.dir.data,
58                 form.step.data,
59                 form.reset.data,
60                 form.endstop.data,
61                 form.FaseA.data,
62                 form.FaseB.data]
63         # Check if there's a duplicated value
64         check = [x for x, y in collections.Counter(pins).items() if y > 1]
65         if check:
66             error = f"Hay␣valores␣repetidos"
67         else:
68             GPIO_filepath = "../YAML/GPIO.yaml"
69             create_files()
70
71             with open(GPIO_filepath, 'r+') as f:
72                 content = yaml.load(f, Loader=yaml.FullLoader)
73                 f.seek(0)  #
74                 f.truncate()  # Erase content to rewrite
75
76                 content["pinDIR"] = form.dir.data
77                 content["pinSTEP"] = form.step.data
78                 content["pinRESET"] = form.reset.data
79                 content["pinENDSTOP"] = form.endstop.data
80                 content["FaseA"] = form.FaseA.data
81                 content["FaseB"] = form.FaseB.data
```

```
82                 yaml.dump(content, f)
83     with open("../YAML/GPIO.yaml", 'r') as f:
84         init_content = yaml.load(f, Loader=yaml.FullLoader)
85 
86     return render_template("gpio_pines.html", form=form, error=error, content=
           init_content)
87 
88 
89 @app.route("/Movement", methods=["GET", "POST"])
90 def move():
91     form = FormMov(request.form)
92     if form.validate_on_submit():
93         # Check if pigpiod is working, if not start it
94         if subprocess.run(shlex.split("pigs t")).returncode == 255:  # returncode == 255
               if offline
95             subprocess.run(shlex.split("sudo pigpiod"))
96             app.logger.info("Iniciar pigpio daemon.")
97         else:
98             app.logger.info("pigpio daemon ya esta encendida.")
99 
100        app.logger.info("Moviendo...")
101 
102        GPIO_filepath = "../YAML/GPIO.yaml"
103        char_move_filepath = "../YAML/char_move.yaml"
104        counter_filepath = "../YAML/counter.yaml"
105        create_files()
106 
107        with open(char_move_filepath, 'r+') as f:
108            content = yaml.load(f, Loader=yaml.FullLoader)
109            f.seek(0)  #
110            f.truncate()  # Erase content to rewrite
111 
112            content["Mode"] = form.mode.data
113            content["Dis_mode"] = form.dis_mode.data
114            content["Displacement"] = form.displacement.data
115            content["Delay"] = form.delay.data
116            content["Sense"] = form.sense.data
117            content["Motion"] = form.motion.data
118            content["Rel"] = form.rel.data
119            yaml.dump(content, f)
120 
121        tomove(GPIO_filepath, char_move_filepath, counter_filepath)
122        app.logger.info("Movimiento finalizado.")
123        return redirect(url_for("index"))
124 
125    with open("../YAML/char_move.yaml", 'r') as f:
126        init_content = yaml.load(f, Loader=yaml.FullLoader)
```

```
127
128     return render_template("movement.html", form=form, content=init_content)
129
130
131 @app.route("/home")
132 def home():
133     if subprocess.run(shlex.split("pigs t")).returncode == 255:  # returncode == 255 if
            offline
134         subprocess.run(shlex.split("sudo pigpiod"))
135         app.logger.info("Iniciar pigpio daemon.")
136     else:
137         app.logger.info("pigpio daemon ya esta encendida.")
138
139     GPIO_filepath = "../YAML/GPIO.yaml"
140     counter_filepath = "../YAML/counter.yaml"
141     create_files()
142
143     app.logger.info("Moviendo a Home...")
144
145     tohome(GPIO_filepath, counter_filepath)
146
147     return redirect(url_for("index"))
```

A.7. Formularios del servidor web

```
1  from flask_wtf import FlaskForm
2  from wtforms import SubmitField, IntegerField, BooleanField
3  from wtforms.validators import DataRequired
4
5
6  class FirstForm(FlaskForm):
7
8      gpio_pines = SubmitField("Configuracion de pines GPIO")
9      movement = SubmitField("Configuracion de movimiento")
10     home = SubmitField("Home")
11
12
13 class FormMov(FlaskForm):
14     mode = BooleanField("Modo")
15     dis_mode = BooleanField("Modo distancia")
16     displacement = IntegerField("Desplazamiento", validators=[DataRequired("Es necesario
           indicar el desplazamiento")])
17     delay = IntegerField("Delay", validators=[DataRequired("Es necesario indicar un delay
           ")])
18     sense = BooleanField("Sentido de giro")
```

```
19      motion = BooleanField("Relación de sentido")
20      rel = IntegerField("Relación pasos/distancia")
21      submit = SubmitField("Mover")
22
23
24  class FormGPIO(FlaskForm):
25      dir = IntegerField("Dir", validators=[DataRequired("Es necesario indicar todos los
            pines")])
26      step = IntegerField("Step", validators=[DataRequired("Es necesario indicar todos los
            pines")])
27      reset = IntegerField("Reset", validators=[DataRequired("Es necesario indicar todos
            los pines")])
28      endstop = IntegerField("Final de carrera", validators=[DataRequired("Es necesario
            indicar todos los pines")])
29
30      FaseA = IntegerField("Fase A", validators=[DataRequired("Es necesario indicar todos
            los pines")])
31      FaseB = IntegerField("Fase B", validators=[DataRequired("Es necesario indicar todos
            los pines")])
32
33      submit = SubmitField("Enviar")
```

A.8. Templates del servidor web

- **index.html**

```
1   {% extends "base.html" %}
2
3   {% block title %}TITULO TFM{% endblock %}
4
5   {% block content %}
6       <h1>Diseño de una controladora low-cost para motores paso a paso</h1>
7       <h2>Trabajo de Fin del Máster de Ingeniería Industrial</h2>
8       <h3>Pablo M. García Rodríguez</h3>
9       <form method="POST" action="/" novalidate class="form1">
10          {{ form.csrf_token() }}
11          <div>
12              {{ form.home(class_="boton1") }}
13          </div>
14          <div>
15              {{ form.gpio_pines(class_="boton1") }}
16          </div>
17          <div>
18              {{ form.movement(class_="boton1") }}
19          </div>
20      </form>
```

```
21 {% endblock %}
```

- **gpio_pines.html**

```
1  {% extends "base.html" %}
2
3  {% block title %}Selección de pines GPIO{% endblock %}
4
5  {% block content %}
6      <h1>Diseño de una controladora low-cost para motores paso a paso</h1>
7      <h2>Configuración de pines GPIO</h2>
8      {% if error %}
9          <p style="color: red;"><strong>Error: </strong> {{ error }}
10     {% endif %}
11     <figure>
12         <img src="../static/gpio.png" height="444.375" width="774">
13         <figcaption>Pines GPIO para RPi 3B+. <a href="https://www.raspberrypi.org/
               documentation/usage/gpio/">Fuente</a></figcaption>
14     </figure>
15     <form action="{{ url_for('gpio_pines') }}" method="POST" novalidate class="
           form2">
16         {{ form.csrf_token() }}
17         <table class="table1">
18         <tr>
19             <th class="td_izq">Pin</th>
20             <th>Valor</th>
21             <th>Valor inicial</th>
22         </tr>
23         <tr>
24             <td class="td_izq">{{ form.dir.label }}</td>
25             <td>{{ form.dir(size=10) }}</td>
26             <td>{{ content["pinDIR"] }}</td>
27         </tr>
28         <tr>
29             <td class="td_izq">{{ form.step.label }}</td>
30             <td>{{ form.step(size=10) }}</td>
31             <td>{{ content["pinSTEP"] }}</td>
32         </tr>
33         <tr>
34             <td class="td_izq">{{ form.reset.label }}</td>
35             <td>{{ form.reset(size=10) }}</td>
36             <td>{{ content["pinRESET"] }}</td>
37         </tr>
38         <tr>
39             <td class="td_izq">{{ form.endstop.label }}</td>
40             <td>{{ form.endstop(size=10) }}</td>
41             <td>{{ content["pinENDSTOP"] }}</td>
```

```
42          </tr>
43          <tr>
44              <td class="td_izq">{{ form.FaseA.label }}</td>
45              <td>{{ form.FaseA(size=10) }}</td>
46              <td>{{ content["FaseA"] }}</td>
47          </tr>
48          <tr>
49              <td class="td_izq">{{ form.FaseB.label }}</td>
50              <td>{{ form.FaseB(size=10) }}</td>
51              <td>{{ content["FaseB"] }}</td>
52          </tr>
53          </table>
54          {% for error in form.dir.errors %}
55                  <span style="color:␣red;">{{ error }}</span>
56          {% endfor %}
57          {% for error in form.step.errors %}
58                  <span style="color:␣red;">{{ error }}</span>
59          {% endfor %}
60          {% for error in form.reset.errors %}
61                  <span style="color:␣red;">{{ error }}</span>
62          {% endfor %}
63          {% for error in form.endstop.errors %}
64                  <span style="color:␣red;">{{ error }}</span>
65          {% endfor %}
66          {% for error in form.FaseA.errors %}
67                  <span style="color:␣red;">{{ error }}</span>
68          {% endfor %}
69          {% for error in form.FaseB.errors %}
70                  <span style="color:␣red;">{{ error }}</span>
71          {% endfor %}
72          <br>
73          <div>
74              {{ form.submit(class_="boton3") }}
75          </div>
76      </form>
77  {% endblock %}
```

- **movement.html**

```
1   {% extends "base.html" %}
2
3   {% block title %}TITULO TFM{% endblock %}
4
5   {% block content %}
6       <h1>Diseño de una controladora low-cost para motores paso a paso</h1>
7       <h2>Configuración del movimiento</h2>
8       <form action="{{␣url_for('move')␣}}" method="POST" novalidate class="form2">
```

```
9           {{ form.csrf_token() }}
10          <table class="table">
11          <tr>
12              <th class="td_izq">Nombre</th>
13              <th>Valor</th>
14              <th>Valor inicial</th>
15          </tr>
16          <tr>
17              <td class="td_izq">{{ form.mode.label }}</td>
18              <td>{{ form.mode(size=10) }}</td>
19              <td>{{ content["Mode"] }}</td>
20          </tr>
21          <tr>
22              <td class="td_izq">{{ form.dis_mode.label }}</td>
23              <td>{{ form.dis_mode(size=10) }}</td>
24              <td>{{ content["Dis_mode"] }}</td>
25          </tr>
26          <tr>
27              <td class="td_izq">{{ form.displacement.label }}</td>
28              <td>{{ form.displacement(size=10) }}</td>
29              <td>{{ content["Displacement"] }}</td>
30
31          </tr>
32          <tr>
33              <td class="td_izq">{{ form.delay.label }}</td>
34              <td>{{ form.delay(size=10) }}</td>
35              <td>{{ content["Delay"] }}</td>
36
37          </tr>
38          <tr>
39              <td class="td_izq">{{ form.sense.label }}</td>
40              <td>{{ form.sense(size=10) }}</td>
41              <td>{{ content["Sense"] }}</td>
42          </tr>
43          <tr>
44              <td class="td_izq">{{ form.motion.label }}</td>
45              <td>{{ form.motion(size=10) }}</td>
46              <td>{{ content["Motion"] }}</td>
47          </tr>
48          <tr>
49              <td class="td_izq">{{ form.rel.label }}</td>
50              <td>{{ form.rel(size=10) }}</td>
51              <td>{{ content["Rel"] }}</td>
52          </tr>
53          </table>
54          {% for error in form.mode.errors %}
55                  <span style="color:␣red;">{{ error }}</span>
```

```
56          {% endfor %}
57          {% for error in form.dis_mode.errors %}
58                  <span style="color: red;">{{ error }}</span>
59          {% endfor %}
60          {% for error in form.displacement.errors %}
61                  <span style="color: red;">{{ error }}</span>
62          {% endfor %}
63          {% for error in form.delay.errors %}
64                  <span style="color: red;">{{ error }}</span>
65          {% endfor %}
66          {% for error in form.sense.errors %}
67                  <span style="color: red;">{{ error }}</span>
68          {% endfor %}
69          {% for error in form.motion.errors %}
70                  <span style="color: red;">{{ error }}</span>
71          {% endfor %}
72          {% for error in form.rel.errors %}
73                  <span style="color: red;">{{ error }}</span>
74          {% endfor %}
75          <br>
76          <div>
77              {{ form.submit(class_="boton2") }}
78          </div>
79      </form>
80  {% endblock %}
```

A.9. Inicio de servidor Modbus

```
1  from pymodbus.server.asynchronous import StartTcpServer
2  from pymodbus.server.asynchronous import ModbusDeviceIdentification
3  from pymodbus.datastore import ModbusServerContext
4
5  import logging
6  import datetime
7
8  date = datetime.datetime.now()
9  date = str(date)
10 date = date.replace(" ", "_")
11 date = date[:-7]
12 filename = "modbus_logs/" + date + ".log"
13 f = open(filename, 'x')
14 f.close()
15
16 LOG_FORMAT = ("%(asctime)-15s %(threadName)-15s"
17           "%(levelname)-8s %(module)-15s:%(lineno)-8s %(message)s")
```

```
logging.basicConfig(level=logging.DEBUG,
                    format=LOG_FORMAT)
log = logging.getLogger()
fh = logging.FileHandler(filename)
log.addHandler(fh)

context = ModbusServerContext(slaves=set_slaves(), single=False)

identity = ModbusDeviceIdentification()
identity.VendorName = "Pablo M. Garcia Rodriguez"

StartTcpServer(context, indentity=identity, address=("0.0.0.0", 2222))  # Puerto 2222, la
     rpi tiene que tenerlo superior a 1024
```

A.10. Función set_slaves

```
from pymodbus.datastore import ModbusSlaveContext

from Modbus.modbus_server_classes import CharMove_DataBlock, GPIO_DataBlock,
    Move_DataBlock

def set_slaves():
        """Set the different slaves, coils and registers.
    - Slave 0x001 - Holding registers:
        This slave is to set the GPIO pin in BCM format
        0 - pinDIR
        1 - pinSTEP
        2 - pinRESET
        3 - pinENDSTOP
        4 - FaseA
        5 - FaseB

    - Slave 0x002 - Holding registers:
        This slave is for the characteristics of the Movement
        0 - Mode -> 0 absolute, 1 incremental
        1 - Dis_mode -> 0 None, 1 activated
        2 - Displacement -> Steps to move
        3 - Delay -> Time in seconds
        4 - Sense -> 0 no change, 1 inverse
        5 - Motion -> Sense of mov. between encoder and stepper
            1 the same
            0 different
        6 - Rel -> Relation steps/mm

    - Slave 0x003 - Coil:
```

```
29          This slave is to move the stepper with the char. of Movement given with slaves 1
                and 2.
30          Once the char. of the Movement are setted
31          0 - Home -> 1 to move, 0 to not
32          1 - Movement
33      """
34      slaves = {
35          0x001: ModbusSlaveContext(hr=GPIO_DataBlock(0, [0]*0x006), zero_mode=True),
36          0x002: ModbusSlaveContext(hr=CharMove_DataBlock(0, [0]*0x005), zero_mode=True),
37          0x003: ModbusSlaveContext(co=Move_DataBlock({0x000: Home(), 0x001: Move()}),
                zero_mode=True)
38      }
39
40      return slaves
```

A.11. Clases del servidor

```
1   import os
2   import shlex
3   import subprocess
4
5   import yaml
6   from pymodbus.datastore import ModbusSequentialDataBlock, ModbusSparseDataBlock
7
8   from Movement.tomove import move, home
9   from YAML.create_files import modbus_create_files
10
11
12  class GPIO_DataBlock(ModbusSequentialDataBlock):
13      def setValues(self, address, value):
14
15          modbus_create_files()
16
17          with open("YAML/GPIO.yaml", 'r+') as f:
18              content = yaml.load(f, Loader=yaml.FullLoader)
19              f.seek(0)  #
20              f.truncate()  # Erase content to rewrite
21              if address == 0:
22                  change = "pinDIR"
23              elif address == 1:
24                  change = "pinSTEP"
25              elif address == 2:
26                  change = "pinRESET"
27              elif address == 3:
28                  change = "pinENDSTOP"
```

```
29              elif address == 4:
30                  change = "FaseA"
31              elif address == 5:
32                  change = "FaseB"
33
34              content[change] = value[0]
35              yaml.dump(content, f)
36
37          super(GPIO_DataBlock, self).setValues(address, value)
38
39
40  class CharMove_DataBlock(ModbusSequentialDataBlock):
41      def setValues(self, address, value):
42
43          modbus_create_files()
44
45          with open("YAML/char_move.yaml", 'r+') as f:
46              content = yaml.load(f, Loader=yaml.FullLoader)
47              f.seek(0)  #
48              f.truncate()  # Erase content to rewrite
49              if address == 0:
50                  change = "Mode"
51              elif address == 1:
52                  change = "Dis_mode"
53              elif address == 2:
54                  change = "Displacement"
55              elif address == 3:
56                  change = "Delay"
57              elif address == 4:
58                  change = "Motion"
59              elif address == 5:
60                  change = "Sense"
61              elif address == 6:
62                  change = "Rel"
63
64              content[change] = value[0]
65              yaml.dump(content, f)
66
67          super(CharMove_DataBlock, self).setValues(address, value)
68
69
70  class Move_DataBlock(ModbusSparseDataBlock):
71      def __init__(self, type):
72          self.type = type
73          values = {k: 0 for k in type.keys()}
74          super(Move_DataBlock, self).__init__(values)
75
```

```
76      def setValues(self, address, value):
77          if (not os.path.isfile("YAML/char_move.yaml")
78                  or not os.stat("YAML/char_move.yaml").st_size):
79              print("No están definidos los archivos de movimiento")
80          else:
81              if address == 1:
82                  self.type[address].value = value[0]
83                  if value[0]:
84                      if subprocess.run(shlex.split("pigs t")).returncode == 255:  #
                            returncode == 255 if offline
85                          subprocess.run(shlex.split("sudo pigpiod"))
86
87                      GPIO_filepath = "YAML/GPIO.yaml"
88                      char_move_filepath = "YAML/char_move.yaml"
89                      counter_filepath = "YAML/counter.yaml"
90
91                      move(GPIO_filepath, char_move_filepath, counter_filepath)
92
93              elif address == 0:
94                  self.type[address].value = value[0]
95                  if value[0]:
96                      if subprocess.run(shlex.split("pigs t")).returncode == 255:  #
                            returncode == 255 if offline
97                          subprocess.run(shlex.split("sudo pigpiod"))
98
99                      GPIO_filepath = "YAML/GPIO.yaml"
100                     counter_filepath = "YAML/counter.yaml"
101                     home(GPIO_filepath, counter_filepath)
102
103         super(Move_DataBlock, self).setValues(address, value)
```

A.12. Función create_files

```
1   import yaml
2   import os
3
4
5   def create_files():
6       GPIO_filepath = "../YAML/GPIO.yaml"
7       char_move_filepath = "../YAML/char_move.yaml"
8       counter_filepath = "../YAML/counter.yaml"
9
10      if (not os.path.isfile(GPIO_filepath)
11              or not os.stat(GPIO_filepath).st_size):
12          content = {"pinDIR": 4,
```

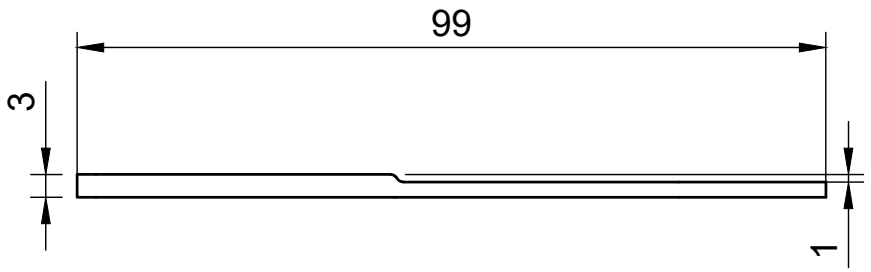
```
                    "pinSTEP": 17,
                    "pinRESET": 27,
                    "pinENDSTOP": 22,
                    "FaseA": 19,
                    "FaseB": 26
                    }  # Default pins
        with open(GPIO_filepath, 'w+') as f:
            yaml.dump(content, f)

    if (not os.path.isfile(char_move_filepath)
            or not os.stat(char_move_filepath).st_size):
        content = {"Mode": 0,
                    "Dis_mode": 0,
                    "Displacement": 0,
                    "Delay": 0,
                    "Sense": 0,
                    "Motion": 0,
                    "Rel": 0
                    }  # Default pins
        with open(char_move_filepath, 'w+') as f:
            yaml.dump(content, f)

    if (not os.path.isfile(counter_filepath)
            or not os.stat(counter_filepath).st_size):
        content = {"Absolute": 0,
                    "Incremental": 0
                    }
        with open(self.filepath, 'w+') as f:
            yaml.dump(content, f)

def modbus_create_files():
    GPIO_filepath = "YAML/GPIO.yaml"
    char_move_filepath = "YAML/char_move.yaml"
    counter_filepath = "YAML/counter.yaml"

    if (not os.path.isfile(GPIO_filepath)
            or not os.stat(GPIO_filepath).st_size):
        content = {"pinDIR": 4,
                    "pinSTEP": 17,
                    "pinRESET": 27,
                    "pinENDSTOP": 22,
                    "FaseA": 19,
                    "FaseB": 26
                    }  # Default pins
        with open(GPIO_filepath, 'w+') as f:
            yaml.dump(content, f)

```

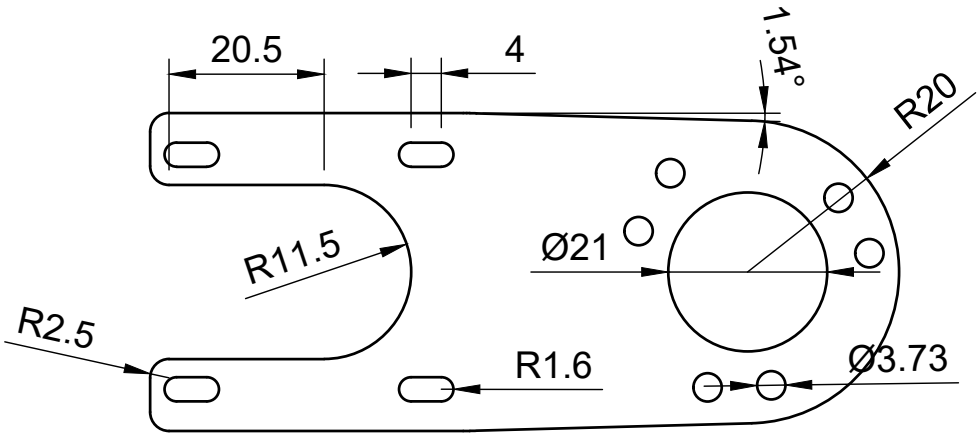
```
60      if (not os.path.isfile(char_move_filepath)
61              or not os.stat(char_move_filepath).st_size):
62          content = {"Mode": 0,
63                     "Displacement": 0,
64                     "Delay": 0,
65                     "Divider": 0,
66                     "Motion": 0
67                     }  # Default pins
68          with open(char_move_filepath, 'w+') as f:
69              yaml.dump(content, f)
70
71      if (not os.path.isfile(counter_filepath)
72              or not os.stat(counter_filepath).st_size):
73          content = {"Absolute": 0,
74                     "Incremental": 0
75                     }
76          with open(self.filepath, 'w+') as f:
77              yaml.dump(content, f)
```

Anexo B

Planos del acople del motor y el encoder

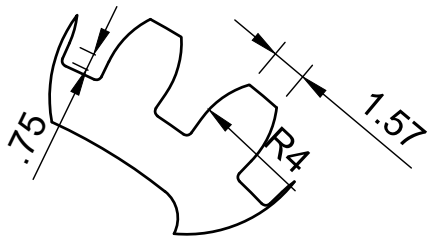
En este anexo se incluye el código fuente, escrito en Python, de cada una de las clases, funciones y páginas escritas.

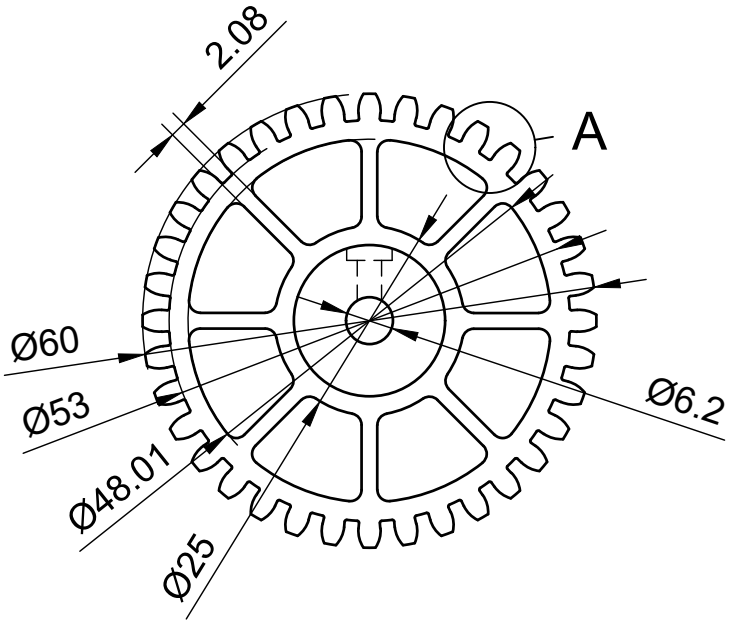


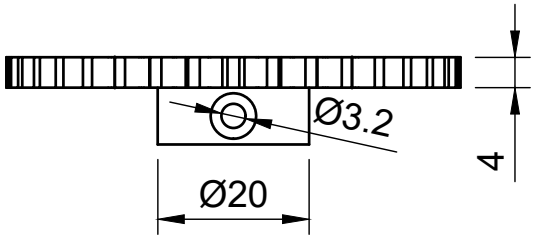


Universidad de Jaén	Diseñado por: Pablo M. García Rodríguez
	Tutor: Alejandro Sánchez García
	Trabajo de Fin de Máster: Controladora para motores paso a paso
	Título: Acople — Hoja: 1/3

A (3:1)

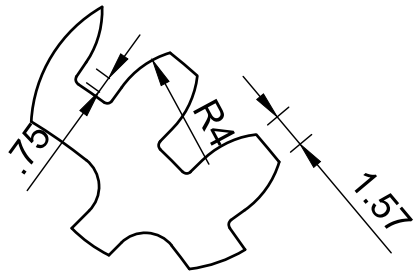


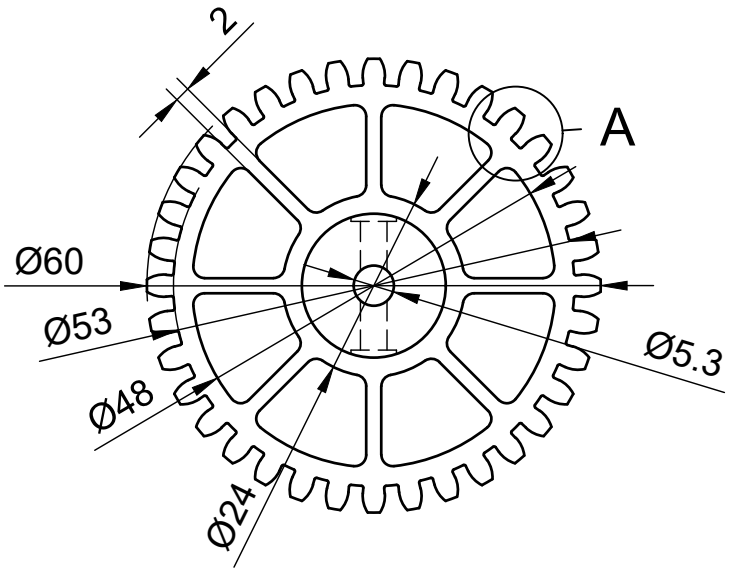


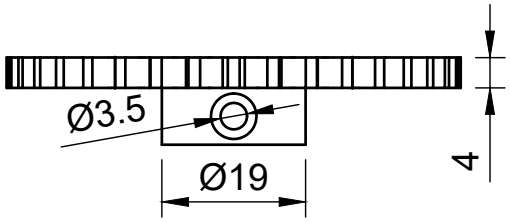


uja Universidad de Jaén	Diseñado por Pablo M. García Rodríguez	
	Tutor Alejandro Sánchez García	
	Trabajo de Fin de Máster Controladora para motores paso a paso	
	Título Engranaje del encoder	Hoja 2/3

A (3:1)







uja Universidad de Jaén	Diseñado por Pablo M. García Rodríguez	
	Tutor Alejandro Sánchez García	
	Trabajo de Fin de Máster Controladora para motores paso a paso	
	Título Engranaje del motor	Hoja 3/3