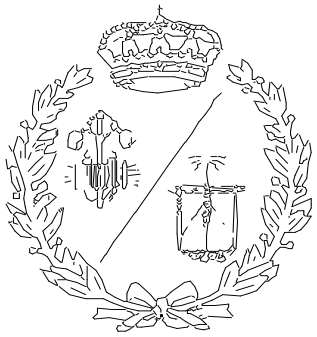
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Grado

Diseño e implementación de un sistema didáctico de laboratorio para las asignaturas del área de Automática. Control de posición.

(Design and implementation of a laboratory teaching system for subjects in the area of Automatic Control. Position control)

Para acceder al Título de

GRADUADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Eduardo Saiz Pérez

Junio – 2025

RESUMEN

Este proyecto aborda el diseño e implementación de un sistema didáctico para el estudio del control de posición mediante un controlador Proporcional-Integral-Derivativo (PID). El objetivo principal es la modelización de un sistema que regula el ángulo de salida de un motor, permitiendo al usuario visualizar cómo los parámetros K_p, K_i y K_d del controlador afectan la respuesta para alcanzar el ángulo deseado. Se busca facilitar una comprensión accesible y dinámica del funcionamiento del controlador PID y el ajuste de sus parámetros para lograr un sistema estable y preciso.

El desarrollo se estructura en una fundamentación teórica sobre los controladores PID y sus componentes, y una implementación práctica que comprende dos partes principales. Primero, una maqueta física portátil construida con un motor JGA25-371, una placa Arduino Mega 2560 REV3, un Arduino Motor Shield REV3 y una pantalla táctil TFT de 2,8 pulgadas para la interacción directa y ajuste de parámetros. Se detalla el montaje de estos componentes y la programación en lenguaje Arduino para el funcionamiento autónomo de la maqueta. Segundo, se desarrolla una aplicación de escritorio en Python con una interfaz gráfica de usuario, utilizando Spyder y Qt Designer. Esta aplicación permite la configuración remota de los parámetros del controlador PID, la visualización en tiempo real de la respuesta del sistema tanto en la maqueta como en el PC, y el registro de datos para su posterior análisis cuando la maqueta está conectada a un ordenador.

Finalmente, se presentan los resultados obtenidos, mostrando gráficamente cómo las variaciones en los parámetros del controlador PID influyen en la respuesta del sistema de control de posición. Se concluye que el sistema didáctico implementado, compuesto por la maqueta y la interfaz de PC, constituye una herramienta eficaz para la enseñanza y experimentación práctica de los principios de control PID.

ABSTRACT

This project addresses the design and implementation of a didactic system for the study of position control using a Proportional-Integral-Derivative (PID) controller. The main objective is to model a system that regulates the output angle of a motor, allowing the user to visualize how the K_p, K_i, and K_d parameters of the controller affect the response in achieving a desired angle. It aims to facilitate an accessible and dynamic understanding of PID controller operation and parameter adjustment to achieve a stable and precise system.

The development is structured into a theoretical foundation regarding PID controllers and their components, and a practical implementation comprising two main parts. First, a portable physical model was built using a JGA25-371 motor, an Arduino Mega 2560 REV3 board, an Arduino Motor Shield REV3, and a 2.8-inch TFT touch screen for direct interaction and parameter adjustment. The assembly of these components and programming in Arduino language for the autonomous operation of the model are detailed. Second, a desktop application was developed in Python with a graphical user interface (GUI), using Spyder and Qt Designer. This application allows remote configuration of the PID controller parameters, real-time visualization of the system's response on both the model and the PC, and data logging for subsequent analysis when the model is connected to a computer.

Finally, the obtained results are presented, graphically showing how variations in the PID controller parameters influence the position control system's response. It is concluded that the implemented didactic system, consisting of the model and the PC interface, constitutes an effective tool for teaching and practical experimentation with PID control principles.

INDICE DE CONTENIDO

INDICE DE ESQUEMAS

INDICE DE TABLAS

INDICE DE FIGURAS

INDICE DE CODIGO

1 MEMORIA

1.1 INTRODUCCIÓN

Para poder iniciar este proyecto correctamente resulta indispensable comenzar repasando brevemente qué son las siglas PID incluidas en el propio título del documento. Recordando lo estudiado, PID viene de controlador proporcional-integral-derivativo. Este tipo de controlador nos permite, de forma resumida, obtener la respuesta deseada y mantenerla en el tiempo mediante un proceso de feedback constante. Resulta verdaderamente interesante puesto que permite controlar infinidad de variables y ajustarlas a demanda en función de las necesidades requeridas. A pesar de su sencillez muestra excelentes resultados sin necesidad de conocer al detalle la planta a controlar.

El proceso operativo de un controlador de este tipo pasa por calcular el error como la diferencia entre el valor de la variable medida y el valor deseado, para posteriormente minimizar dicho error manipulando la entrada del proceso. Cada uno de los tres términos que componen el PID se encarga de controlar el error actual, la acumulación de errores pasados y la velocidad de cambio del error respectivamente. Es la suma de dichos términos y la elección adecuada de los parámetros que los conforman lo que consigue que el controlador tenga una respuesta óptima.

1.2 OBJETIVO

Se requiere la modelización de un sistema de control PID que regule el ángulo de salida de un motor y permita la visualización de cómo afectan cada uno de los parámetros del controlador a la obtención de la respuesta deseada.

Esta maqueta didáctica permitirá al usuario fijar un ángulo deseado para la posición del motor y jugar con las variables proporcional, derivativa e integral del motor para corregir el error del sistema y llegar al ángulo fijado inicialmente. Se pretende que el usuario entienda el funcionamiento del controlador de una forma accesible y dinámica siendo capaz de fijar los parámetros correctamente para conseguir un sistema estable y con el mínimo error.

Para conseguir esto se va a realizar una primera implementación mediante Arduino con diferentes componentes que va a permitir el transporte de la maqueta a cualquier lugar y una segunda que va a consistir en el desarrollo de una aplicación de escritorio en Python con una interfaz gráfica de usuario (GUI) todo ello permite la configuración remota de los parámetros del controlador PID, la visualización en tiempo real de la respuesta del sistema en Arduino y

PC y el registro de datos para su posterior análisis cuando la maqueta esté conectada a un PC.

2 METODOLOGÍA

Propongo dividir el proceso en tres partes, una teórica en la que se fundamente el trabajo realizado, otra donde se verá el hardware y el firmware de Arduino para la construcción de la maqueta y una última parte donde se desarrollará el software para el control y registro de datos cuando se conecte la maqueta al PC.

2.1 TEORÍA

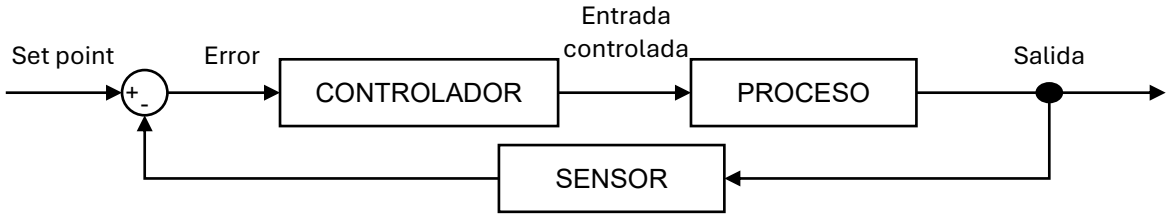
Afrontaremos en este apartado la parte teórica que nos ha llevado hasta este proyecto y que nos ayudará a entender tanto la elaboración de la maqueta como el funcionamiento de esta.

2.1.1 CONTROLADOR PID

Para entender cómo funciona un controlador PID partiremos de un sistema básico sin control que podríamos modelar como podemos ver en el Esquema 1 [8].

Esquema 1 Proceso

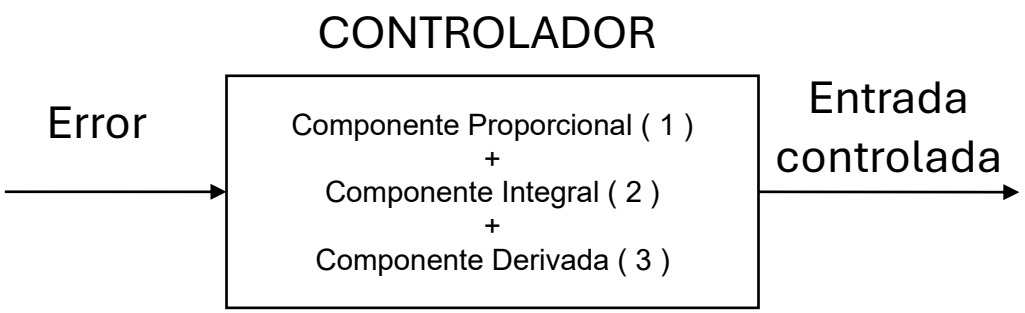
A continuación, se añade la acción del controlador y de un sensor que va a medir la salida. Además, se fija un “Set point” o valor deseado al que se quiere llegar quedando ahora el proceso como en el Esquema 2 [8].



Esquema 2 Diagrama de bloques

El error como ya se ha visto, será la diferencia entre el valor de salida del proceso medido por el sensor y el valor que se ha fijado como salida deseada o “Set point”. El proceso puede ser totalmente opaco, el sensor únicamente mide la salida, pero lo verdaderamente interesante de este sistema de control de lazo cerrado es cómo actúa el controlador sobre el error para ir proporcionando al proceso una entrada controlada [8].

En el Esquema 3 podemos ver como el controlador combina la acción de tres componentes con diferentes funciones (1) (2) (3).



Esquema 3 Controlador

$$K_p e(t) \quad (1)$$

$$K_i \int e(t)dt \quad (2)$$

$$K_d * \frac{de(t)}{dt} \quad (3)$$

Se puede observar que cada componente del controlador tiene su parámetro K_p, K_i y K_d respectivamente y es el ajuste correcto de dichos parámetros lo que permitirá obtener la respuesta deseada. Pero antes de ver cómo ajustar dichas constantes hay que analizar más detalladamente cada uno de los componentes del controlador [8]:

Componente proporcional

De la propia formulación de esta componente (4) podemos deducir que actúa directamente sobre el error actual siendo tanto mayor su efecto cuanto mayor sea el error y viceversa. Influye notablemente en la velocidad de respuesta del sistema de tal manera que cuanto menor sea el valor de la constante elegida más tiempo vamos a tardar en obtener la respuesta deseada o incluso no llegaremos si la constante es demasiado pequeña. Si por el contrario optamos por un valor demasiado elevado podemos oscilar tanto que nuestro sistema se vuelva inestable y no alcancemos tampoco la respuesta. Generalmente esta componente no elimina por completo el error a largo plazo puesto que, al ir disminuyendo el error, disminuye a su vez la capacidad de este término de modificar la entrada al sistema con lo que podríamos conseguir una salida estable pero ligeramente distinta de la deseada [8].

$$K_p e(t) = K_p e = K_p(v_{referencia} - v_{actual}) \quad (4)$$

Componente integral

Esta componente actúa sobre la integral del error a lo largo del tiempo dotando de “memoria” al controlador. Gráficamente equivaldría a la superficie encerrada bajo la curva del error en

función del tiempo, lo que discretamente podría equivaler al cálculo de dicha área mediante rectángulos o trapecios. En el caso de la maqueta se ha optado por una aproximación trapezoidal (5) considerando un periodo de muestreo determinado (st). Esto permitirá eliminar por completo el error a largo plazo, pero si la constante es demasiado pequeña el sistema tardará mucho en eliminarlo. Como inconveniente, esta componente tiene más tendencia a la inestabilidad que la componente proporcional [8].

$$\mathrm{K_i} \int \mathrm{e(t)dt} = \mathrm{K_i}(\mathrm{e_{anterior}} + \mathrm{e_{actual}}) \frac{\mathrm{st}}{2} \qquad (5)$$

Componente derivada

La componente derivada actúa en función de la velocidad de cambio del error, otorgando al sistema una suerte de predicción sobre el error futuro. De forma discreta equivaldría (6) a restar el error anterior al error actual y dividirlo entre el periodo de tiempo transcurrido (st).

El principal inconveniente es que para cambios grandes en periodos de tiempo muy pequeños la respuesta solicitada por esta componente es tan elevada que puede desestabilizar el sistema. Además, es muy sensible a ruidos en la medición puesto que, aunque la variación sea de pequeña amplitud es muy rápida y la respuesta se verá amplificada por esta componente.

Para tiempos fijos de muestreo y cambios razonables del error esta componente para valores moderados de su constante mejorará la respuesta general del sistema [8].

$$\mathrm{K_d} * \frac{de(t)}{dt} = \mathrm{K_d} \frac{e_{actual} - e_{anterior}}{st} \qquad (6)$$

2.1.2 AJUSTE DE PARÁMETROS

El correcto funcionamiento del regulador y por tanto una buena respuesta del sistema dependerá de la elección de los parámetros $\mathrm{K_p}$, $\mathrm{K_i}$ y $\mathrm{K_d}$. Resumiendo lo visto en el apartado anterior:

Componente proporcional

- $\mathrm{K_p}$ pequeña → respuesta lenta
- $\mathrm{K_p}$ muy grande → respuesta inestable
- No elimina el error estacionario

Componente integral

- Elimina el error estacionario
- $\mathrm{K_i}$ elevada → respuesta inestable

Componente derivada

- Mejora el comportamiento
- K_d muy grande → respuesta inestable
- Sensible al ruido
- Sensible a cambios bruscos de error

Teniendo esto en cuenta y gracias a la maqueta podremos jugar con los parámetros para conseguir la mejor respuesta posible del sistema.

Se suele optar por ajustar K_p primero hasta que el sistema sobrepase o empiece a sobreoscilar, a continuación, ajustar K_i hasta eliminar el error estacionario y finalmente incrementar K_d hasta justo antes de que el sistema empiece a desequilibrarse [9].

3 IMPLEMENTACIÓN ARDUINO

Ahora sí, después de la introducción necesaria ya podemos ponernos manos a la obra con la parte práctica del proyecto.

3.1 ARDUINO

Arduino es una plataforma de código abierto que crea prototipos electrónicos para que cualquier persona interesada en la electrónica sea capaz de crear objetos. Se basa en un hardware y software flexibles y fáciles de usar.

3.1.1 Hardware o placa Arduino

El hardware de Arduino se trata de un PCB que es una placa de circuito impreso que contiene un microprocesador y los componentes necesarios para interactuar con el entorno. Existen diferentes modelos de estas placas, cada una con diferentes características y capacidades (Arduino UNO, Nano, Mega, Due, etc.) pero que comparten su simplicidad y accesibilidad. Estas placas llevan pines de entrada y salida (digitales y analógicos) para permitir la conexión de diferentes componentes como sensores, motores y diferentes dispositivos electrónicos.

3.1.2 Elección de la placa

Para este proyecto utilizaremos un Arduino Mega 2560. Hemos optado por este ya que se requieren más puertos capaces de usarse como salidas PWM para controlar el motor y puesto que al conectar la pantalla con un Arduino UNO no tendríamos ni suficientes salidas PWM ni suficientes pines disponibles para conectar los diferentes componentes.

3.1.3 Entorno de desarrollo

El software utilizado para el correcto funcionamiento va a ser el IDE de Arduino que es gratuito y multiplataforma. Lo utilizamos para escribir, compilar y cargar el código (llamado “sketches”) en el Arduino Mega. En la Figura 3.1 podemos ver como es el IDE de Arduino

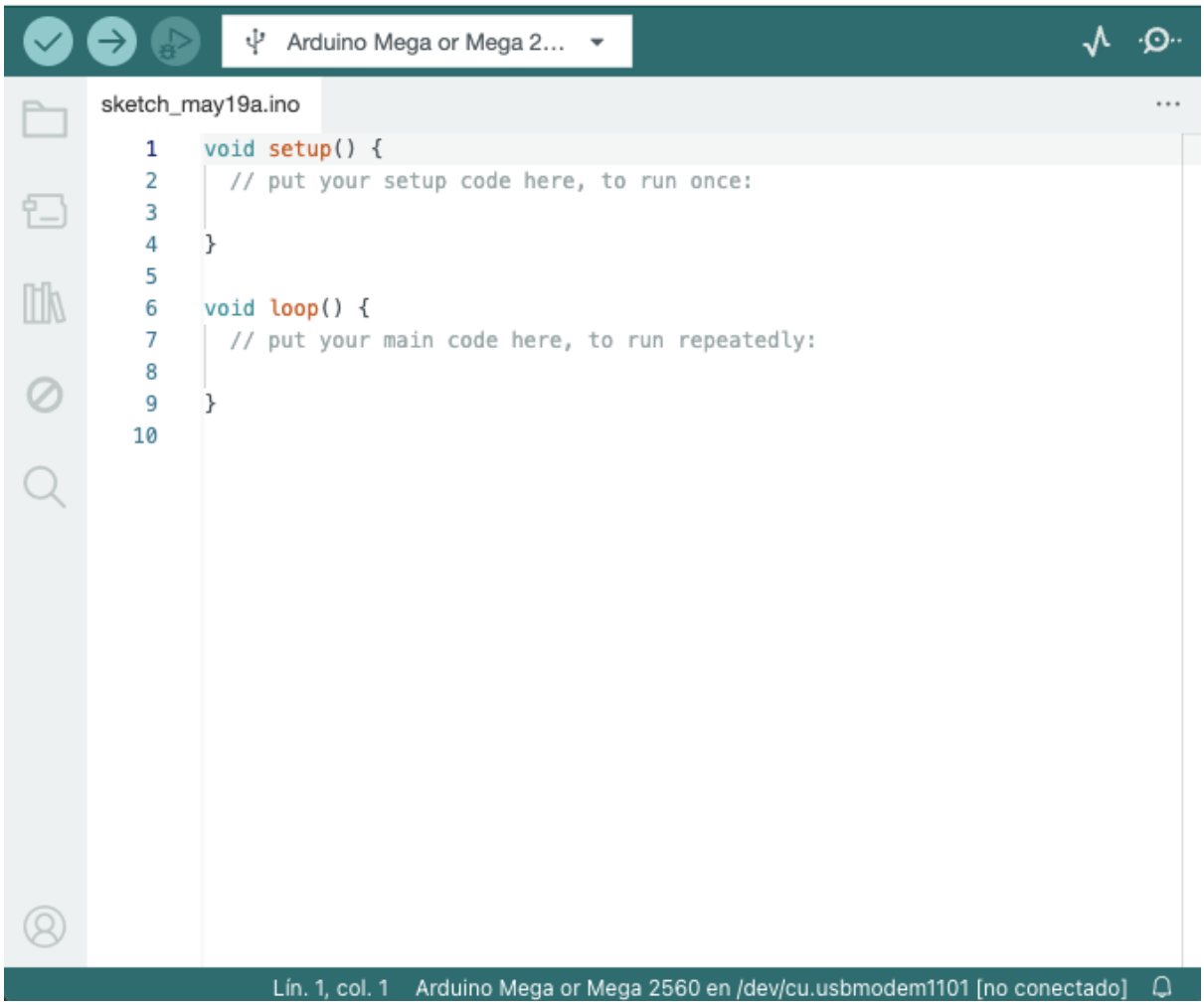


Figura 3.1 IDE Arduino

El lenguaje de programación que utiliza Arduino, basado en C++, no tiene nombre propio y se le conoce directamente como lenguaje Arduino. Dicho lenguaje consta de una serie de comandos que dicen al microcontrolador cómo tiene que estar configurado al iniciarse y lo que tiene que hacer mientras funciona. Para llevar a cabo estos comandos Arduino tiene un entorno de programación (Arduino IDE) que es el utilizado en este caso, aunque se pueden usar otros entornos como por ejemplo Visual Studio Code con una extensión de Arduino.

El código siempre debe contener dos funciones:

Función setup():

En esta función se declara la configuración del microcontrolador, como por ejemplo la configuración de los pines a utilizar, si son entradas o salidas. Estos comandos solo se ejecutan al inicio.

Función loop():

Debe ir siempre después de la función setup(), y alberga los comandos, llamadas a otras funciones para que el programa ejecute las acciones deseadas [1].

Aparte de estas dos funciones, también se pueden declarar las librerías necesarias (previa instalación en el propio IDE), crear librerías propias y crear funciones para ordenar el código. Para instalar librerías se puede hacer de dos maneras, una en la barra lateral en “GESTOR DE BIBLIOTECAS” y buscar ahí la que nos interesa y pulsar en instalar o si no aparece y tenemos el .zip con la librería descargada podemos, como aparece en la Figura 3.2, seleccionar el archivo pulsando en “Añadir biblioteca ZIP” [1].

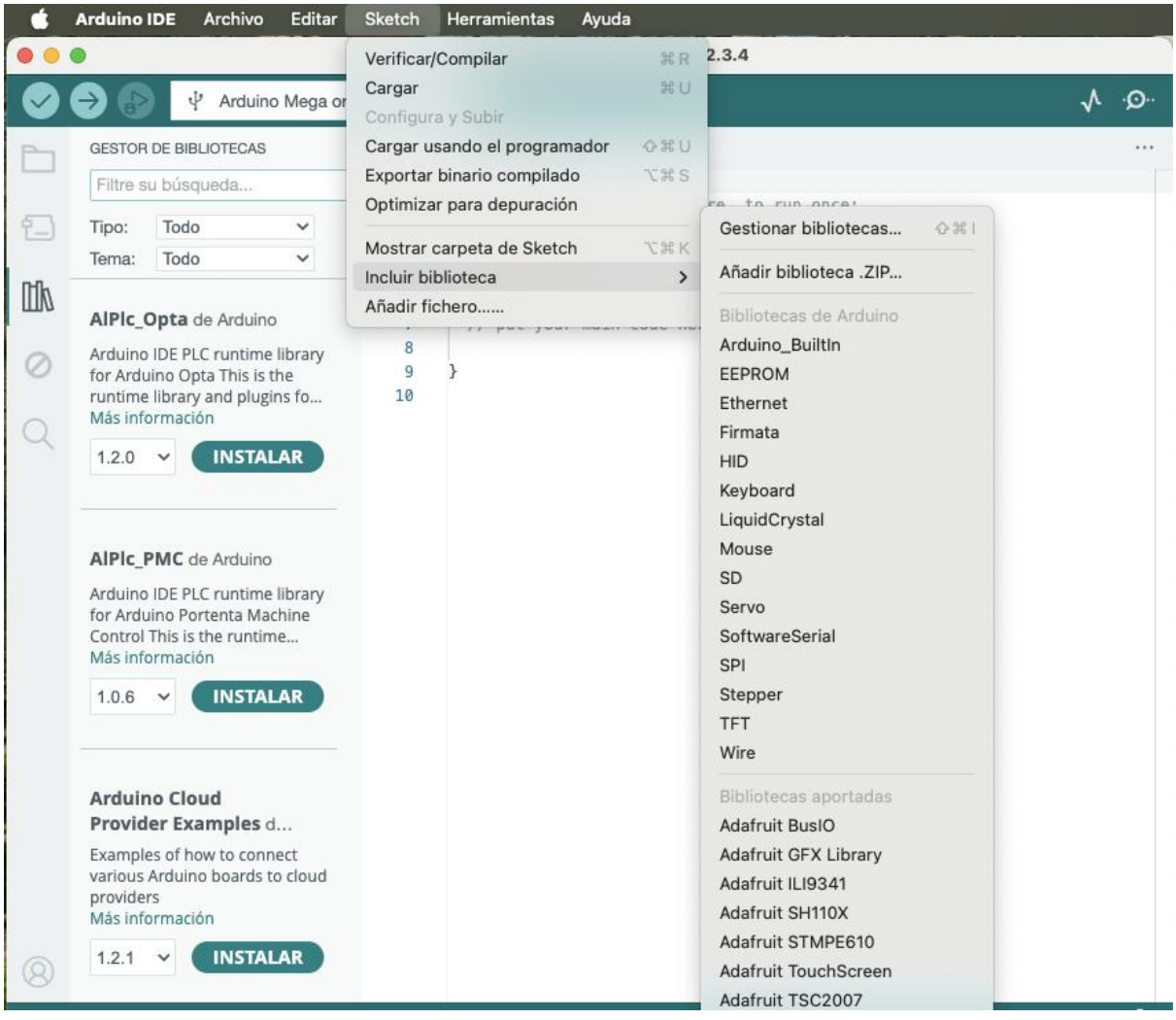


Figura 3.2 Instalar librerías

3.2 ELEMENTOS

La maqueta se compone de los siguientes elementos:

3.2.1 Motor JGA25-371

Se trata de un motor de corriente continua de 12 Voltios (Figura 3.3) con soporte de montaje 65mm, encoder, caja de cambios magnética, Kit de ruedas, micro reductor de velocidad y carcasa metálica con una velocidad máxima de 1000 RPM.

Figura 3.3 Motor [10]

En la Figura 3.4 se pueden ver las dimensiones.

Figura 3.4 Esquema motor [10]

Las especificaciones del fabricante para la conectividad del motor y el encoder se pueden ver en la Tabla 3.1.

Tabla 3.1 Conectividad Motor [10]

COLOR	ESPECIFICACIÓN
Red	+ del Motor (el intercambio entre positivo y negativo controlan CW/CCW o rotación horaria/antihoraria)
Black	- del Encoder (no pueden conectarse mal, rango de voltaje 3.3-5V)
Yellow	Fase A del Encoder (11 pasos)
Green	Fase B del Encoder (11 pasos)
Blue	+ del Encoder (no pueden conectarse mal, rango de voltaje 3.3-5V)
White	- del Motor (el intercambio entre positivo y negativo controlan CW/CCW o rotación horaria/antihoraria)

Un encoder es un sensor que genera señales digitales en respuesta al movimiento. En nuestra maqueta es el encargado de informar de la posición del motor y, por tanto, da cuenta de la salida del sistema.

En este caso se trata de un encoder rotacional incremental de dos fases que genera pulsos en respuesta al movimiento (Figura 3.5). Rotacional en cuanto a que responde a un movimiento de esta tipología, incremental porque posee un número específico de pulsos espaciados equitativamente por revolución (PPR) y con dos fases para poder determinar el sentido de giro del motor (a esto se le denomina salida de cuadratura o bidireccional) [11].

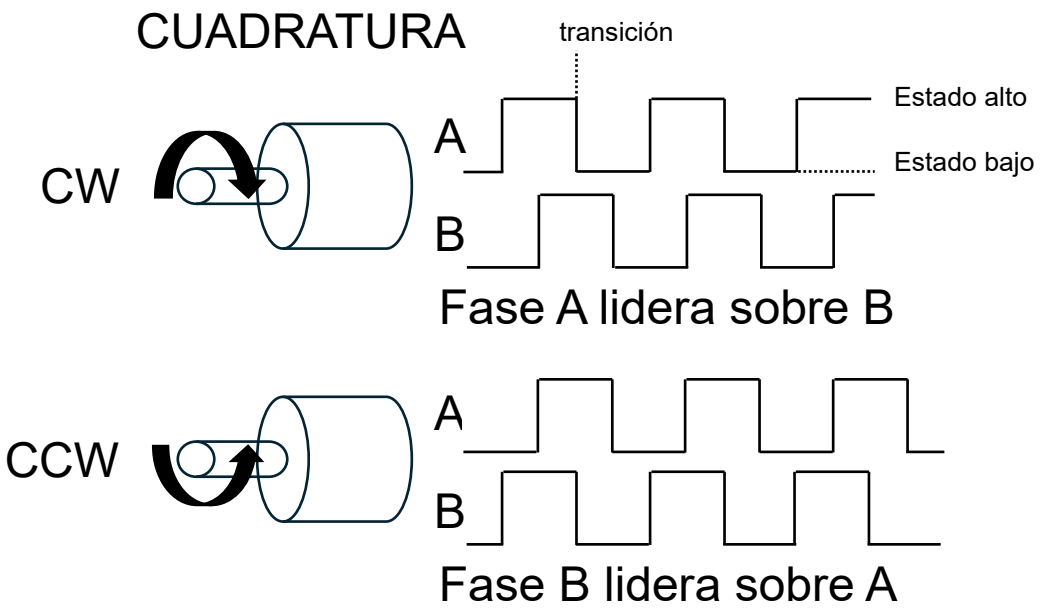


Figura 3.5 Encoder

3.2.2 Placa Arduino mega 2560 REV3

El Arduino Mega 2560 es una placa microcontroladora basada en el ATmega2560. Tiene 54 pines de entradas/salidas digitales (de los cuales 15 se pueden usar como salidas PWM), 16 entradas analógicas, 4 UART (puertos serie de hardware), un oscilador de cristal de 16 MHz, una conexión USB, un conector de alimentación, un encabezado ICSP, y un botón de reinicio. Contiene todo lo necesario para soportar el microcontrolador; podemos conectarlo a una computadora con un cable USB o encenderlo con un adaptador de CA a CC o una batería para comenzar a trabajar con él [13].

Figura 3.6 Arduino mega [12]

A continuación, se muestra la Tabla 3.2 con un resumen de sus especificaciones técnicas.

Tabla 3.2 Arduino mega especificaciones (Fabricante) [13]

Microcontrolador	ATmega2560
Tensión de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límite)	6-20V
Pines de E/S digitales	54 (de los cuales 15 proporcionan salida PWM)
Pines de entrada analógica	dieciséis
Corriente CC por pin de E/S	20 mA
Corriente CC para pin de 3,3 V	50 mA
Memoria flash	256 KB de los cuales 8 KB utilizados por el gestor de arranque
SRAM	8 KB
EEPROM	4 KB
Velocidad de reloj	16MHz
LED_BUILTIN	13
Longitud	101,52 milímetros
Ancho	53,3 milímetros
Peso	37 gramos

3.2.3 Arduino Motor Shield REV3

Arduino Motor Shield (Figura 3.8) se basa en el L298, que es un controlador de puente completo dual diseñado para controlar cargas inductivas como relés, solenoides, CC y motores paso a paso. Te permite controlar dos motores DC con tu placa Arduino, controlando la velocidad y dirección de cada uno de forma independiente. También se puede medir la absorción de corriente de cada motor, entre otras características [15].

Inicialmente necesitábamos el chip L293D (Figura 3.7) para interconectar el motor con Arduino y así poder controlar su giro y velocidad. El Arduino motor Shield nos permite realizar dicho control simplificando el cableado [15]. Se pueden ver las especificaciones en la Tabla 3.3

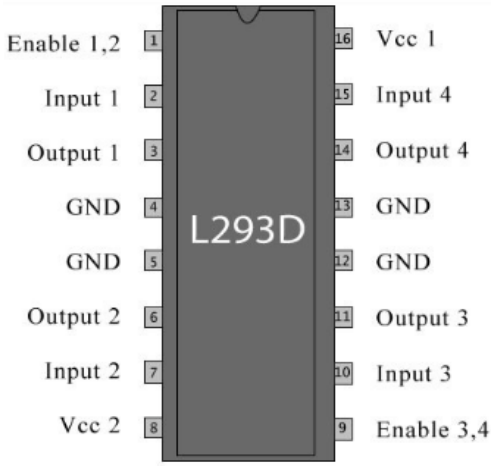


Figura 3.7 Chip L293D [24]

Tabla 3.3 Motor Shield especificaciones [13]

Tensión de funcionamiento	5V a 12V
Controlador del motor	L298P, Acciona 2 motores CC o 1 motor paso a paso
Corriente máxima	2A por canal o 4A máx. (con fuente de alimentación externa)
Detección de corriente	1,65 V/A
Función de parada y freno de funcionamiento libre	

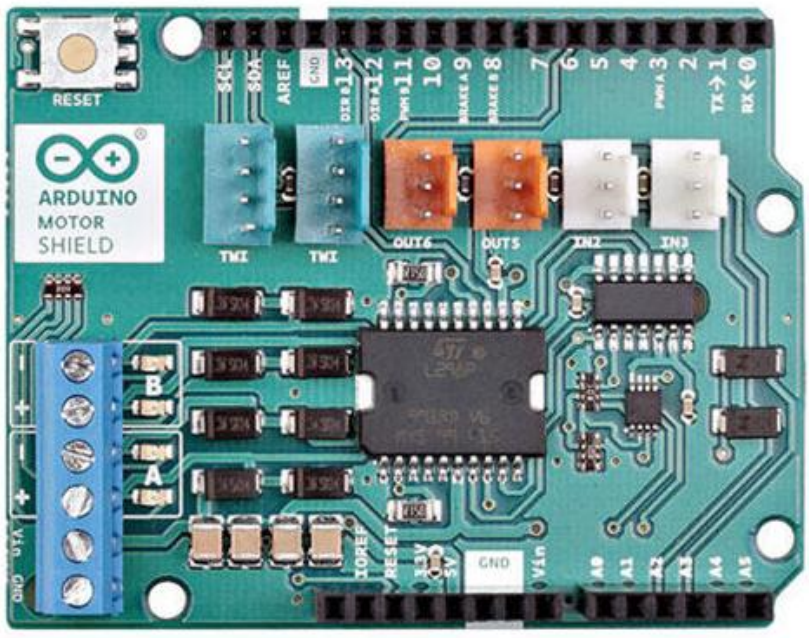


Figura 3.8 Arduino motor Shield rev3 [12]

3.2.4 Pantalla táctil TFT de 2,8 pulgadas

Se trata de una pantalla táctil TFT (Figura 3.9) con ranura para tarjeta SD y compatible con Mega 2560. Con la ayuda de la Tabla 3.4 se puede ver a qué corresponde cada patilla y a donde va conectado en Arduino conectándose directamente insertando el pin en la interfaz sin necesidad de cables como se puede ver en la Figura 3.24. Es también compatible con todo tipo de MCU (Micro Controller Units) de 5V o 3V con circuito de cambio de 5V a 3.3V.

Tiene una resolución HD de 320x240, adopta un bus paralelo de 8 bits consiguiendo una actualización rápida y suave y ofrece soporte con las bibliotecas de Arduino simplificando el desarrollo de programas [17].

Esta pantalla se ha usado en sustitución de los potenciómetros que inicialmente controlaban las constantes del controlador de la maqueta, consiguiendo una interacción más intuitiva y atrayente.

Tabla 3.4 Pantalla TFT conexiones [17]

PINS LCD	PINS MEGA	INSTRUCCIÓN
LCD_RST	A4	Reset Signal
LCD_CS	A3	Chip Sellect
LCD_RS	A2	Comand/Data Sellect
LCD_WR	A1	Write Signal
LCD_RD	A0	Read Signal
GND	GND	Power GND
5V	5V	Power VCC
3V3	3.3V/NC	No Connected
LCD_D0	8	LCD Data Bit0
LCD_D1	9	LCD Data Bit1
LCD_D2	2	LCD Data Bit2
LCD_D3	3	LCD Data Bit3
LCD_D4	4	LCD Data Bit4
LCD_D5	5	LCD Data Bit5
LCD_D6	6	LCD Data Bit6
LCD_D7	7	LCD Data Bit7
SD_SS	10	SD-card Chip Sellect signal
SD_DI	11	SD-card SPI Bus MOSI Signal
SD_DO	12	SD-card SPI Bus MISO Signal
SD_SCK	13	SD-card SPI Bus SCLK Signal

Figura 3.9 Pantalla TFT [16]

3.2.5 Pila 9V 1200mAh

Para la alimentación se ha optado por una pila de 9V (Figura 3.10), recargable de 1200mAh, para que la maqueta sea portátil y no depender de que esté conectada a la corriente cada vez que se quiera usar.

Figura 3.10 Pila 9 [20]

3.2.6 Interruptor de palanca ON-OFF

Se trata de un interruptor ON-OFF de 2 pines (Figura 3.11), para poder encender y apagar la maqueta cuando sea necesario. El interruptor escogido funciona con un voltaje y corriente nominal de 125V y 3A respectivamente.

Figura 3.11 Interruptor [19]

3.2.7 Caja contenedora

Para albergar los componentes se va a utilizar una caja (Figura 3.12) fabricada en plástico ABS de 200×120×75mm (alto×ancho×profundo).

Figura 3.12 Caja contenedora [25]

Se ha escogido esta caja por su tamaño, ya que se pueden acoplar todos los componentes.

3.2.8 Adaptador USB-B a USB-C

Para la conexión con el PC utilizamos un adaptador (Figura 3.13) para convertir la entrada de Arduino (USB-B) en una entrada USB del tipo C. Este adaptador tendrá conectado un cable que lo comunica con la que va a ser la entrada (Figura 3.14) de conexión con el PC.

Figura 3.13 Adaptador USB-B a USB-C [22]

3.2.9 Acoplador de montaje en Panel USB-C

Este conector de montaje (Figura 3.14) sirve para tener acceso al puerto de entrada de datos de Arduino, en otras palabras, para conectar con el PC.

Figura 3.14 Acoplador de Panel USB-C [23]

3.3 MONTAJE

3.3.1 Preparación

Inicialmente, en el Motor Shield hay que deshabilitar los frenos (D8 y D9) y la detección de corriente (A0 y A1) puesto que se necesitan estos pines para la conexión de la pantalla y no son necesarios para nuestra maqueta. Existe dos canales (A y B), y el Motor Shield viene preparado para poder controlar hasta dos motores y solo se necesita un canal. El proceso de deshabilitar estos pines se realiza cortando los puentes en la parte posterior del Shield con un cúter (Figura 3.15).

En la Tabla 3.5 se pueden ver las correspondencias de los pines de cada canal una vez esté el escudo (Motor Shield) colocado en el Arduino. Como el pin D3 es necesario para la interconexión con la pantalla (imposibilitando el uso del canal A) se usa el canal B para controlar el motor. Se utilizan los pines D11 y D13, puesto que los equivalentes en la pantalla están reservados para una memoria SD que no se requiere. Se dejarán conectados dos cables macho/hembra al canal B ya que tras el acople de la pantalla se perderá el acceso a los terminales de tornillo.

En la Figura 3.16 se pueden ver los pines a utilizar y el canal B donde se va a conectar la alimentación.

Tabla 3.5 Motor Shield conexiones [15]

FUNCIÓN	CANAL A	CANAL B
Dirección	D12	**D13**
PWM	**D3**	**D11**
Freno	D9	D8
Detección actual	A0	A1

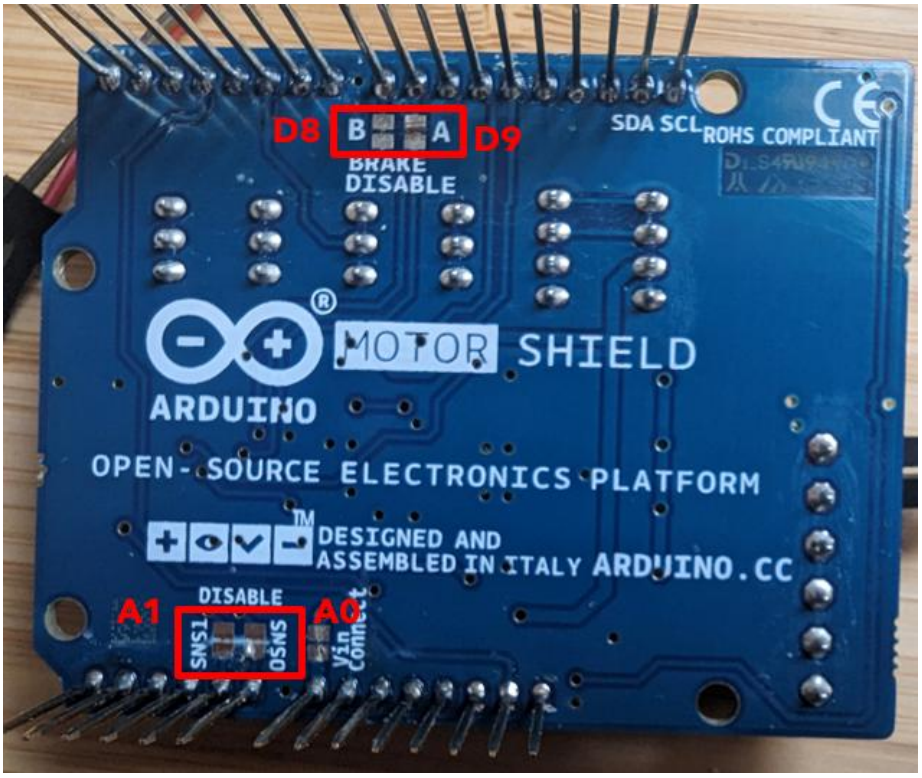


Figura 3.15 Arduino Motor Shield pines deshabilitados

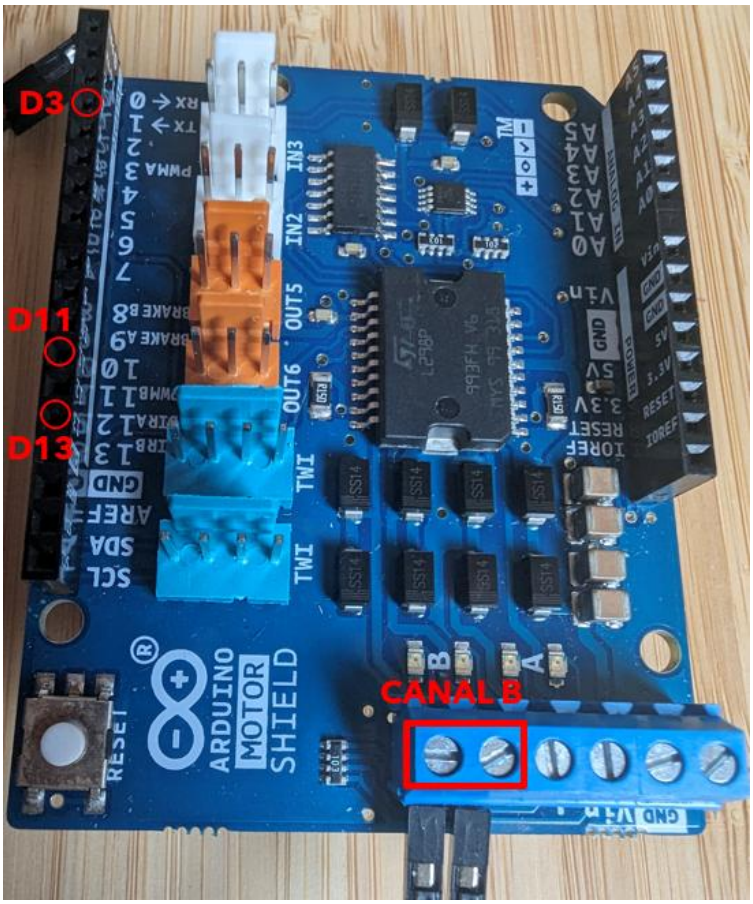


Figura 3.16 Arduino Motor Shield pines importantes

Para facilitar las conexiones se dejan puestos los cables macho-hembra en los pines de Arduino según la Tabla 3.6 y en la Figura 3.16 se puede ver cómo están conectados al canal B.

Tabla 3.6 Conexiones a Arduino

Color cable	Pin Arduino
Verde	20
Amarillo	21
Naranja	5V
Negro	GND

Hay que preparar la caja contenedora para poder colocar cada componente en su lugar, y para ello se ha creado un boceto de cómo va a quedar la maqueta (Figura 3.17).

Este boceto nos sirve de guía para hacer los agujeros donde van los distintos componentes. Se separa la tapa del resto de la caja (los componentes van anclados a la tapa), se pegan los planos que sirven de guía para el corte, y con un dremel se realizan las diferentes perforaciones y cortes que lijaremos para mejorar el acabado.

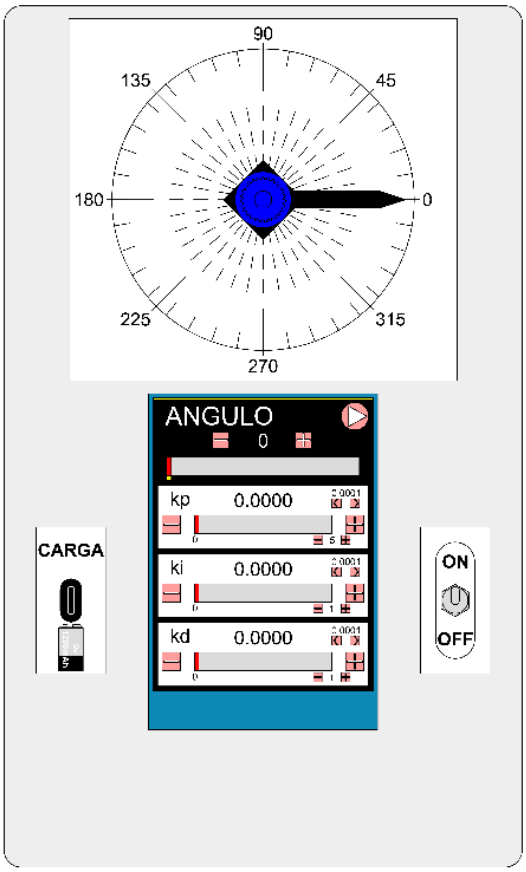


Figura 3.17 Resultado esperado

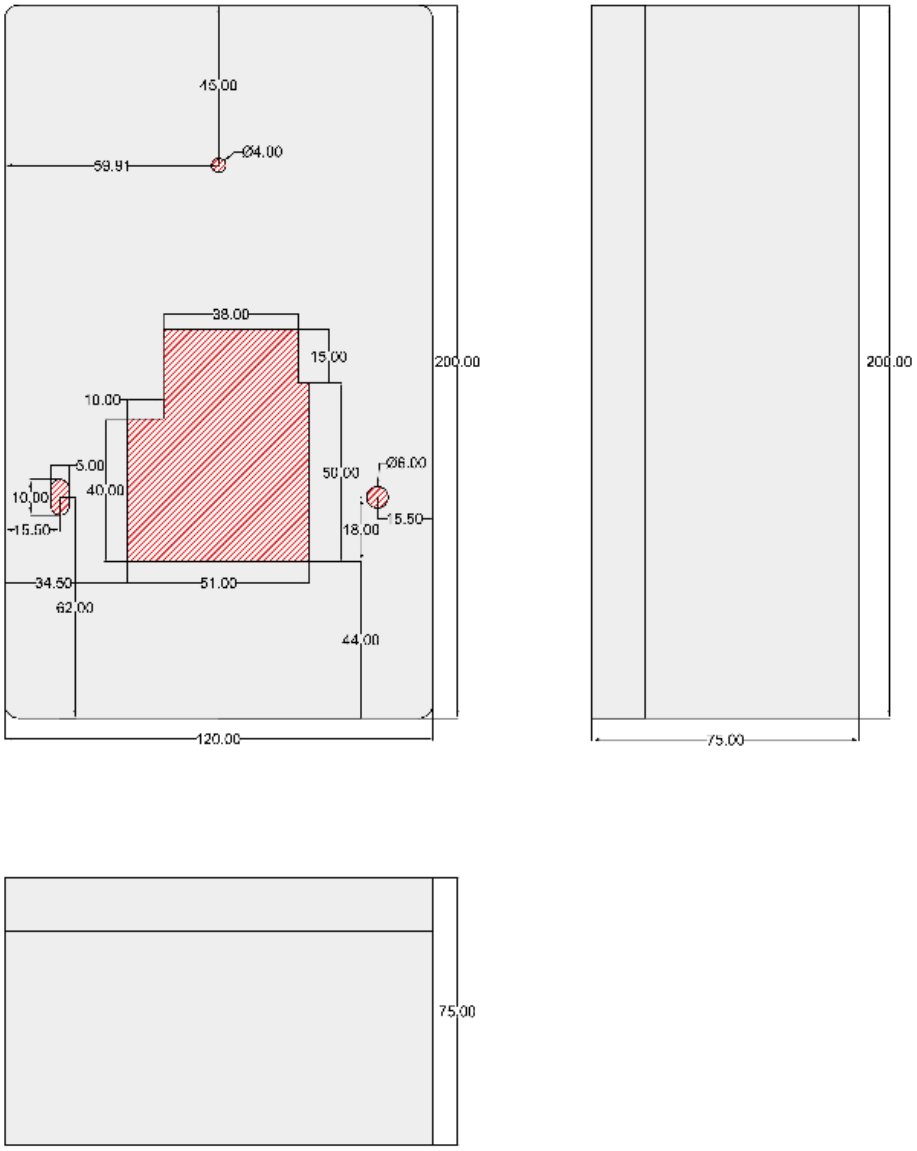


Figura 3.18 Alzado, planta y perfil de la caja contenedora con los agujeros necesarios (todas las medidas están en milímetros)

3.3.2 Conexión

Una vez realizadas las preparaciones del apartado anterior la interconexión es sencilla. Lo primero es acoplar la pantalla encajándola en el hueco como se puede ver en la Figura 3.19 y Figura 3.20.

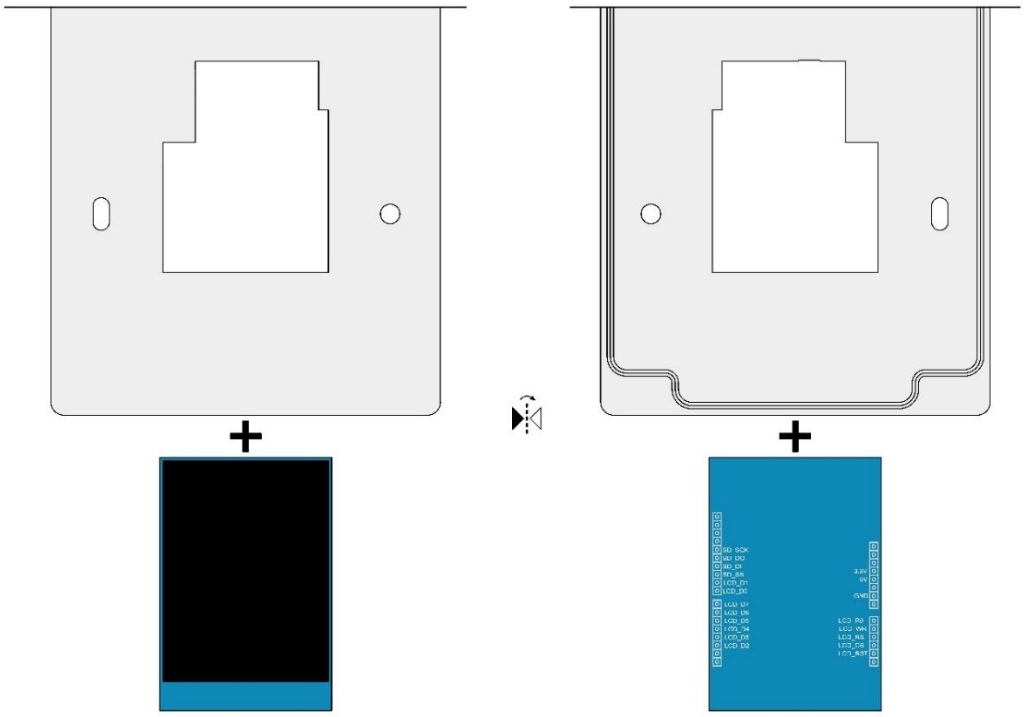


Figura 3.19 Montaje de la pantalla 1

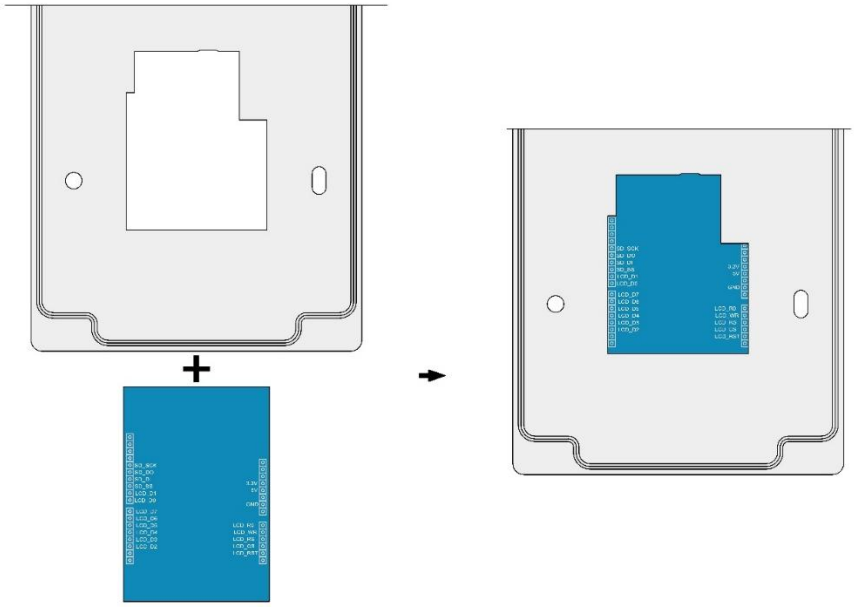


Figura 3.20 Montaje de la pantalla 2

Después se conecta el Arduino con el escudo (Motor Shield) haciendo coincidir los pines con la misma nomenclatura (Figura 3.21 y Figura 3.22).

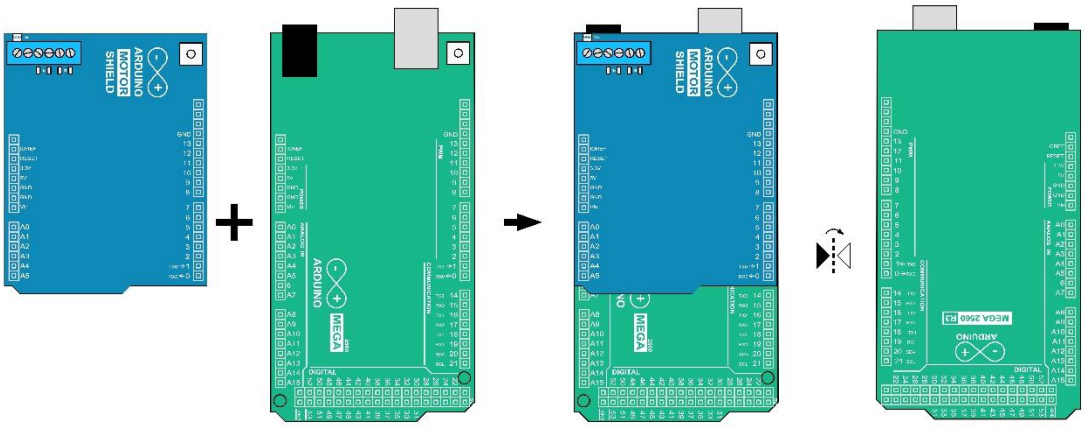


Figura 3.21 Montaje Arduino mega y Arduino Motor Shield

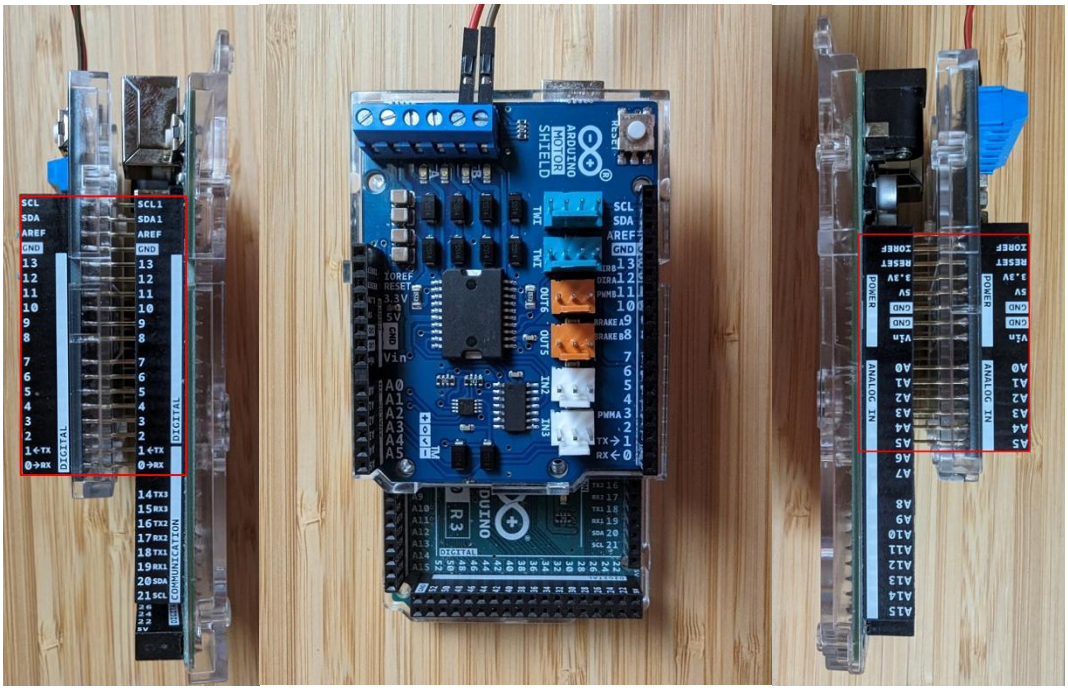


Figura 3.22 Vista previa del montaje Arduino mega y Arduino Motor Shield

Una vez tenemos colocada la pantalla y por otro lado el escudo montado en el Arduino toca conectar ambos (Figura 3.23) lo cual se realiza también de forma rápida e intuitiva haciendo coincidir los pines GND, 5V y 3.3V. Esto servirá además para que la pantalla no se salga, haciendo el bloque interior de tope contra la tapa.

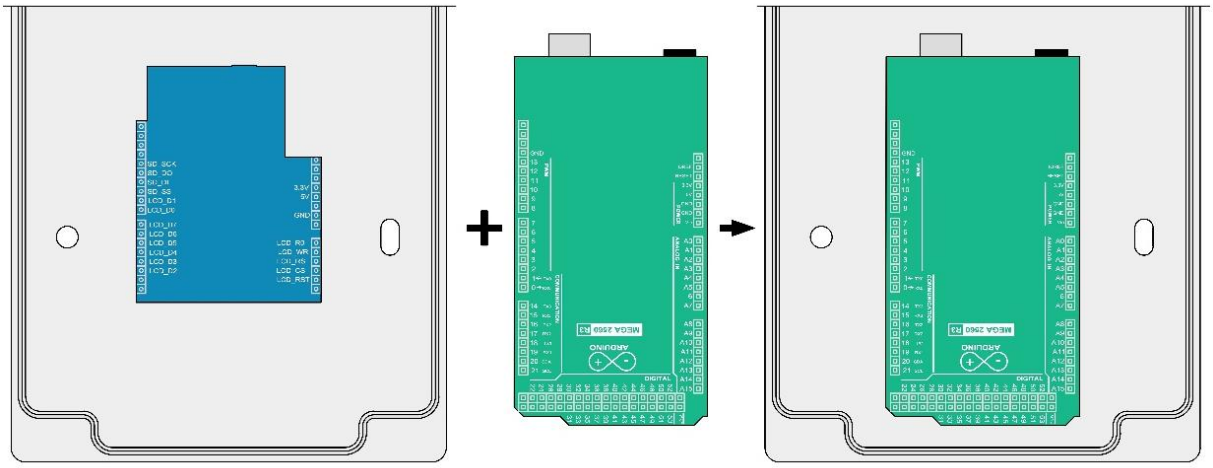


Figura 3.23 Montaje de la Pantalla con la caja y Arduino mega con Arduino Motor Shield

En la Figura 3.24 se puede ver el montaje sin la tapa de por medio, con la interconexión ya comentada de los pines.

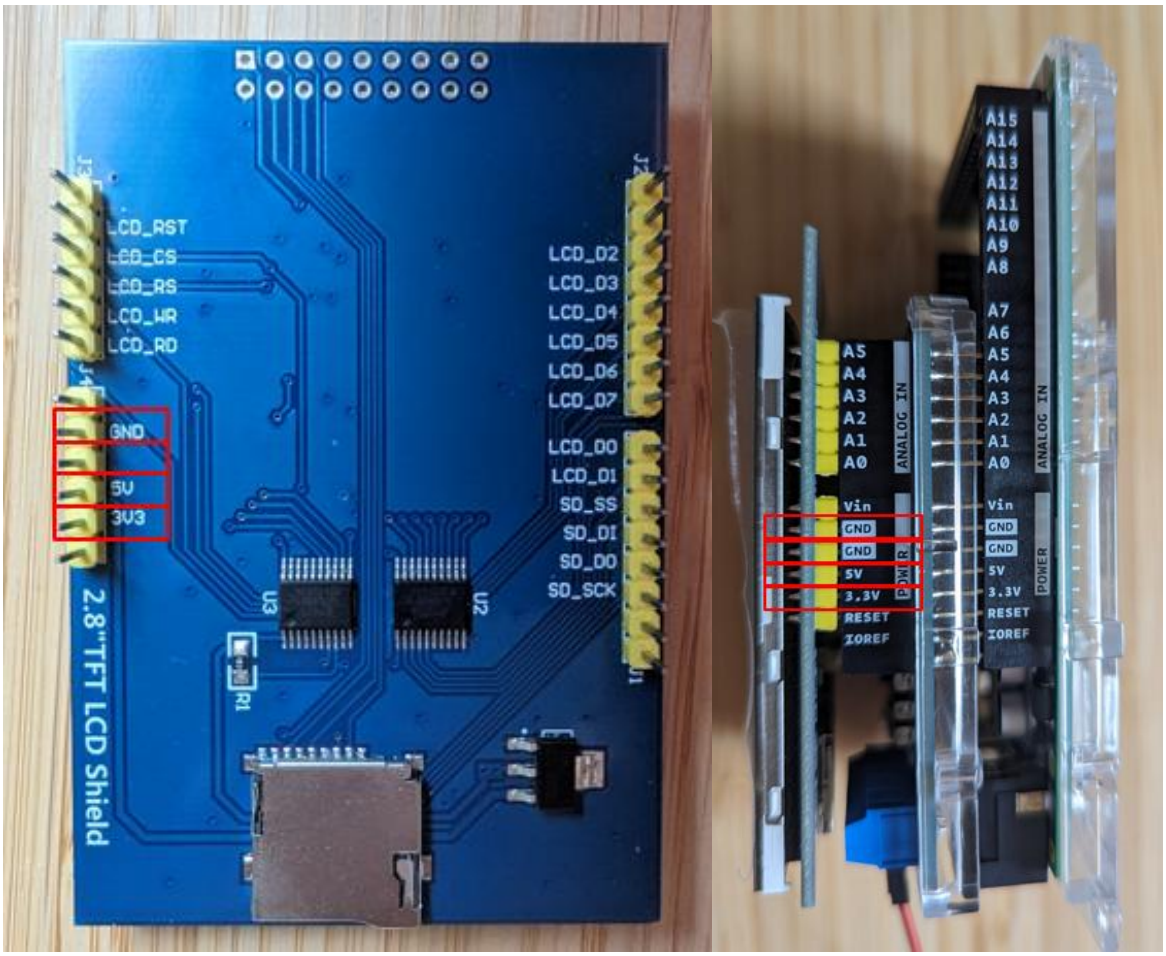


Figura 3.24 Vista previa del montaje de la pantalla con Arduino mega y Motor Shield

Para acoplar el motor en la caja necesitamos una placa de madera fina donde va a ir atornillado el motor (Figura 3.25) y dos pinzas que pegadas a la caja servirán para fijar el motor.

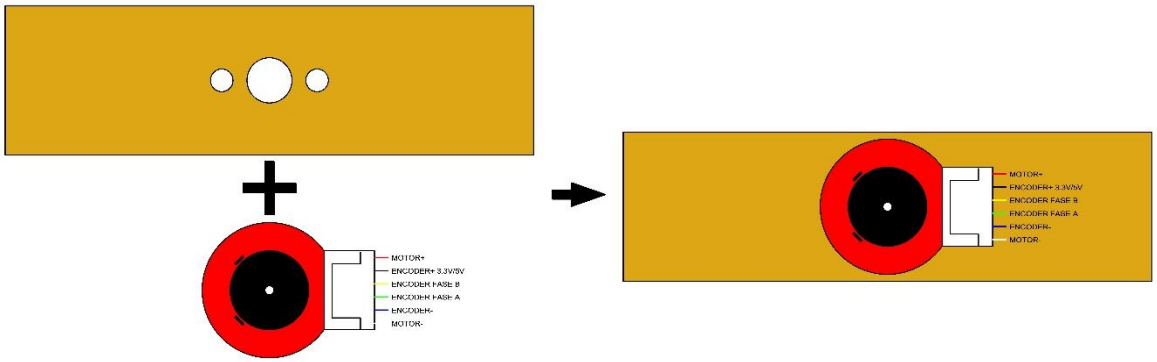


Figura 3.25 Preparación del montaje del motor

Las pinzas se dispondrán en diagonal permitiéndonos insertar el motor (Figura 3.26) girando la placa de madera en sentido antihorario (Figura 3.27).

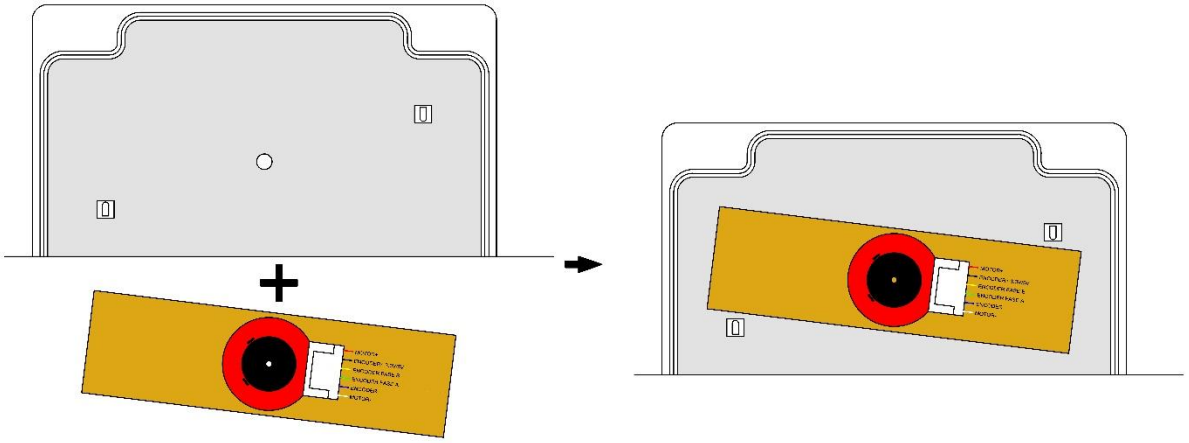


Figura 3.26 Montaje del motor 1

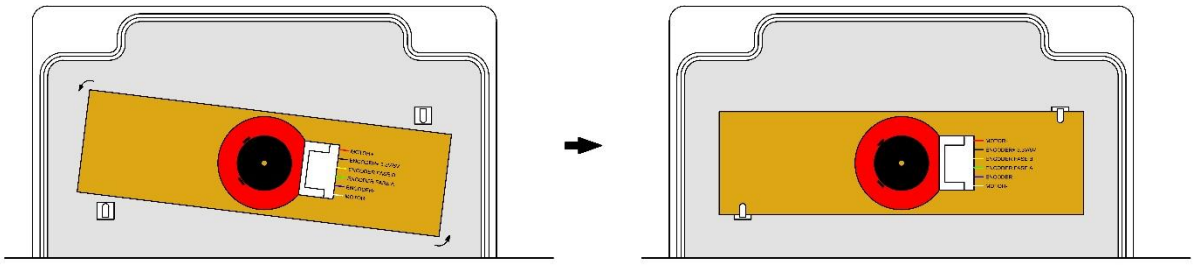


Figura 3.27 Montaje del motor 2

A continuación, para añadir el interruptor basta con introducirlo en el agujero realizado previamente y fijarlo a la caja mediante una tuerca (Figura 3.28).

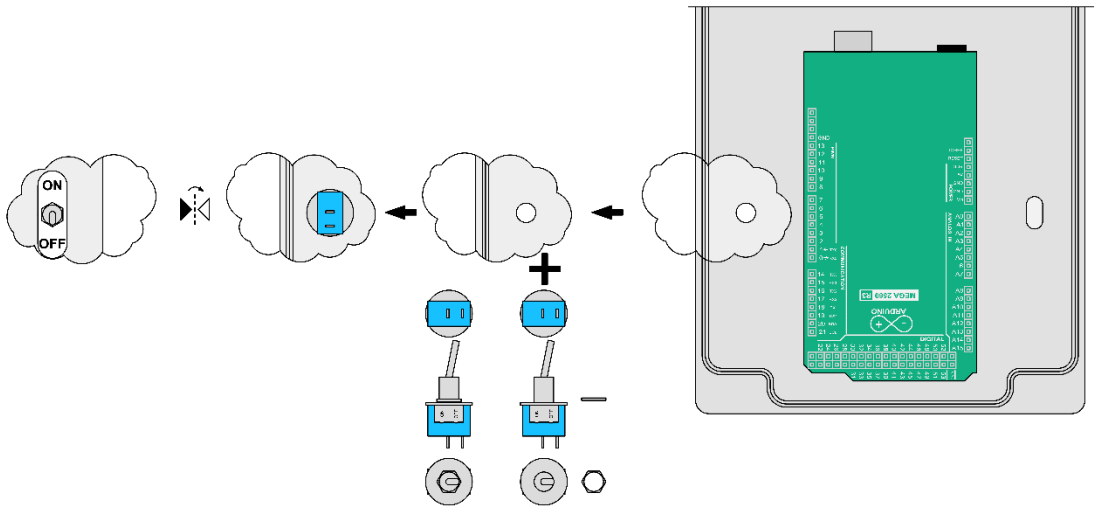


Figura 3.28 Montaje del interruptor en la caja

La alimentación de la maqueta se realiza a través de una pila de 9V que recargaremos utilizando un adaptador. Dicho adaptador nos permitirá emplear un cable USB-C (estándar actual) frente al de tipo micro USB (que trae la pila) y también nos será útil si en algún momento la pila se estropea puesto que podremos intercambiarla fácilmente. Para anclar el conjunto le pegamos una plaquita de madera al adaptador (Figura 3.29) que servirá de tope para poner y quitar el cable (Figura 3.30).

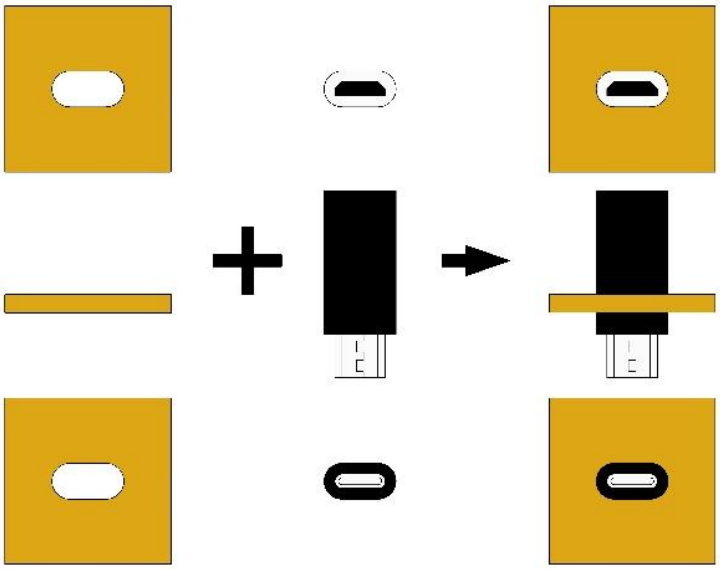


Figura 3.29 Preparación de la pila 1. Vista inferior, planta y alzado

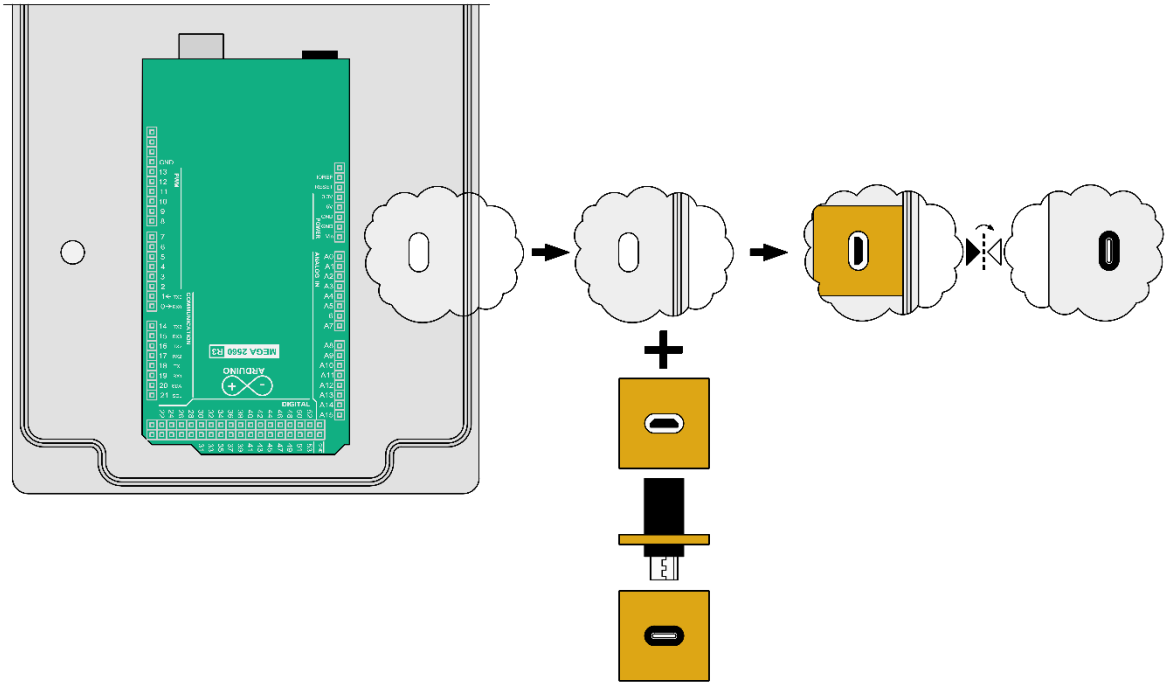


Figura 3.30 Preparación de la pila 2

La Figura 3.31 muestra el ensamblaje final de la pila contra el adaptador.

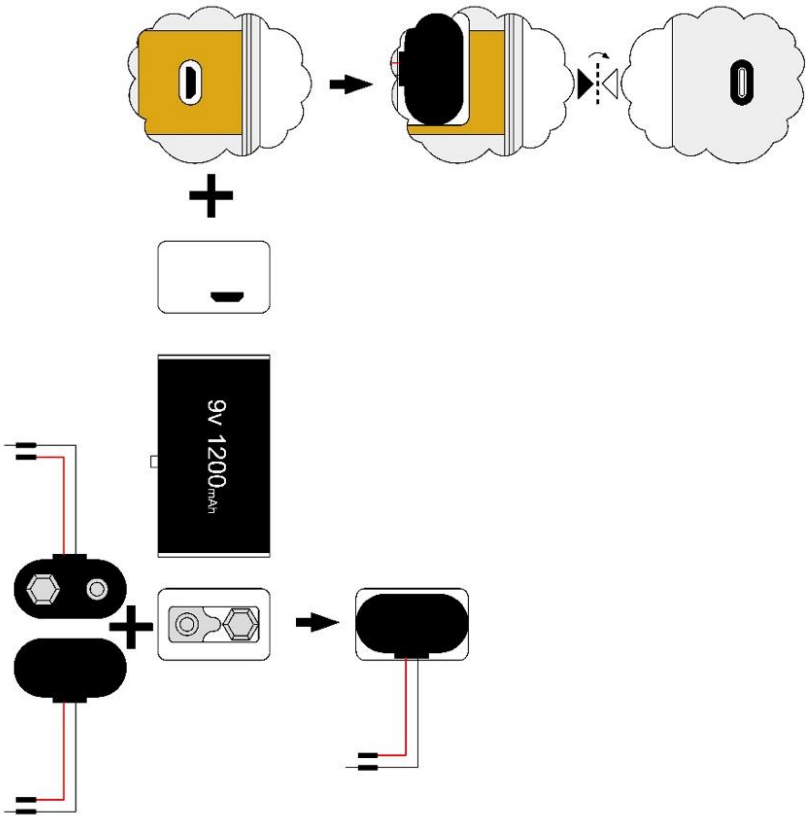


Figura 3.31 Montaje de la pila 3

Una vez montados todos los componentes solo queda conectarlos siguiendo el esquema de conexiones mostrado en la Figura 3.32.

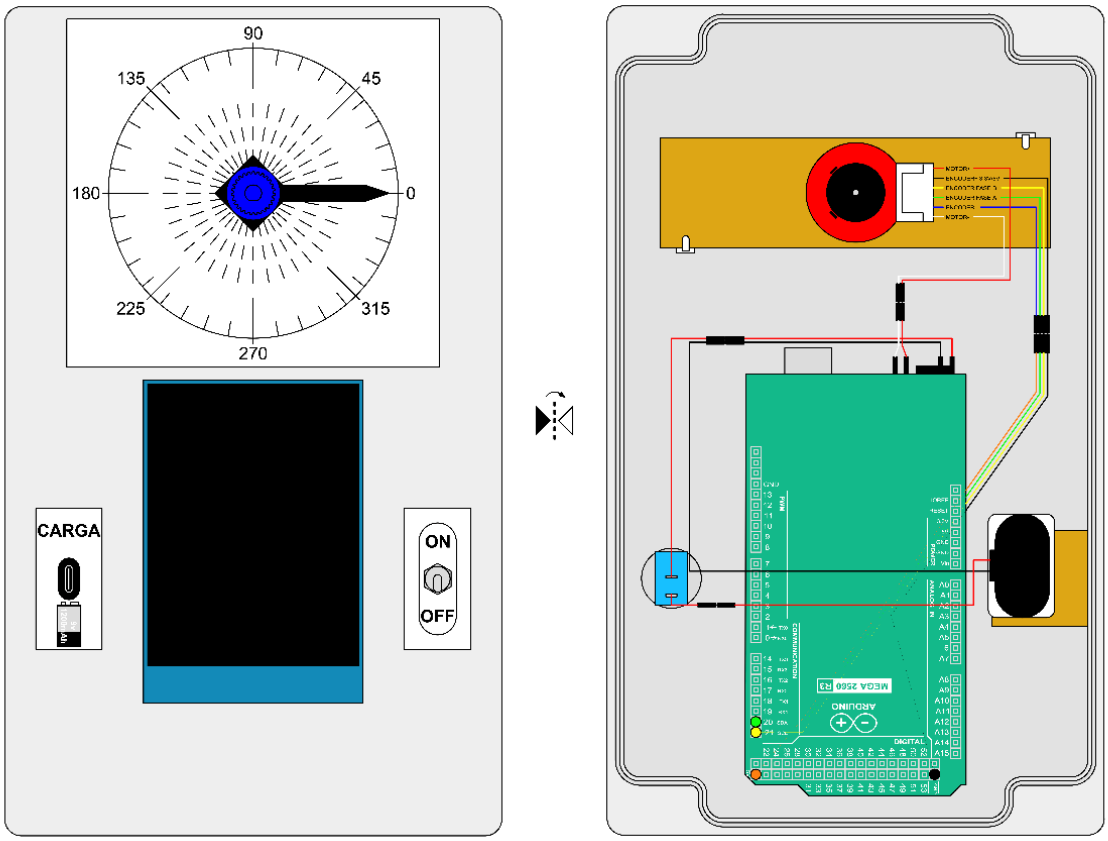


Figura 3.32 Montaje final con esquema de conexiones

Para poder conectar la maqueta al ordenador teniendo acceso desde fuera y no tener que desmontar la tapa se ha optado por añadir dos adaptadores diferentes. Un adaptador de montaje en panel tipo USB-C (Figura 3.33) que nos servirá para conectar la maqueta mediante un cable al ordenador y un segundo adaptador que va en el interior y que convierte la entrada de Arduino (USB-B) a USB-C (Figura 3.34). Se ha optado por esta solución para que no quedase un cable colgando de la maqueta que pudiese arrancarse o estropearse fácilmente.

Figura 3.33 Adaptador montaje en panel

Figura 3.34 Adaptador USB-b a USB-C

En la Figura 3.35 podemos ver el esquema con los dos adaptadores colocados.

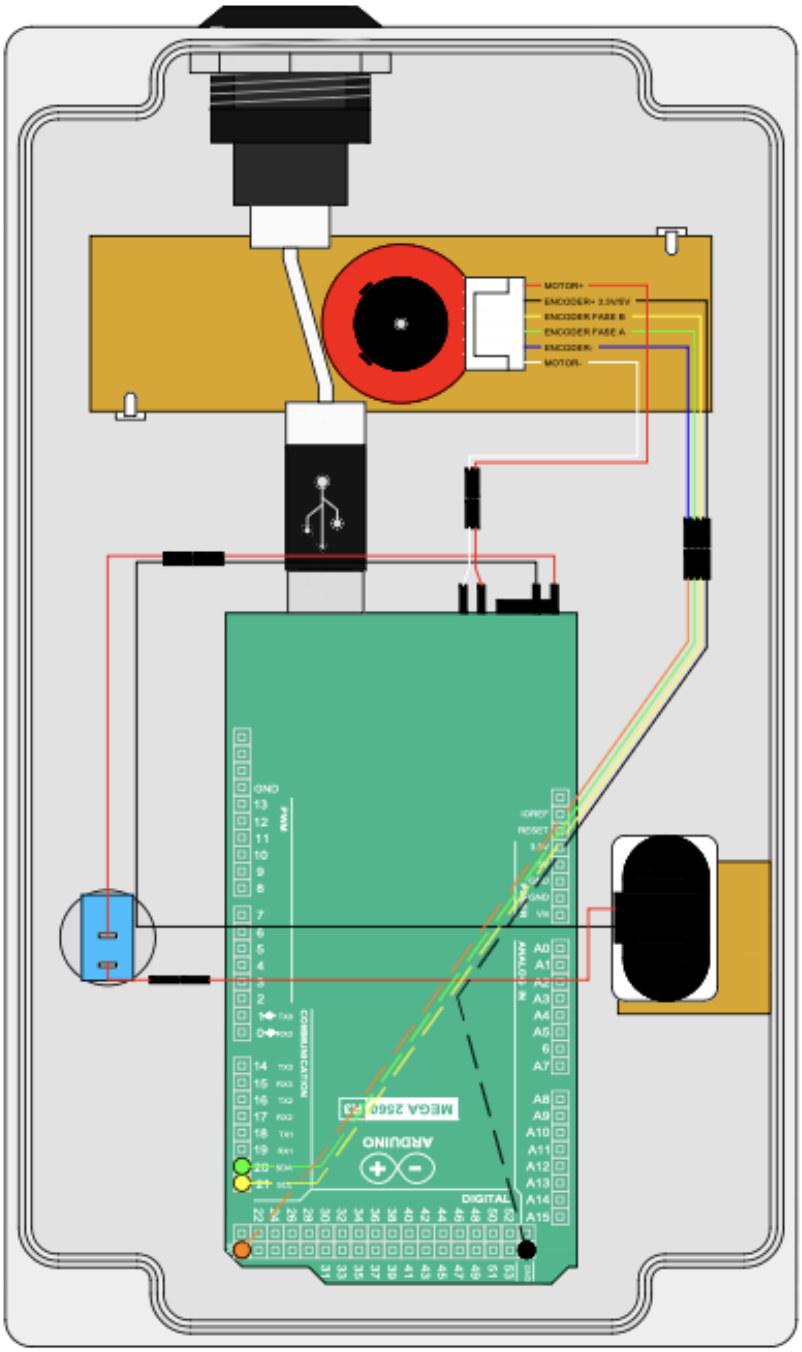


Figura 3.35Montaje final

3.4 RESULTADO

En la Figura 3.36 se puede ver el resultado final de la maqueta tanto apagada como encendida.

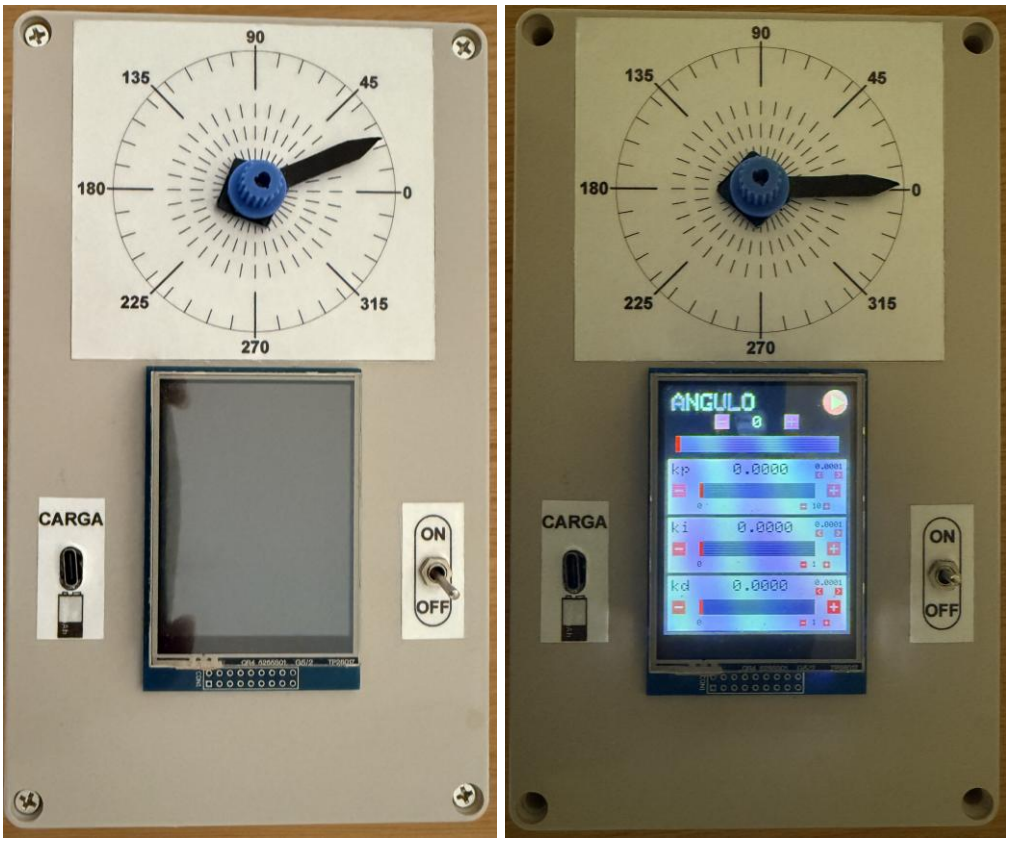


Figura 3.36 Resultado final

Para ver y analizar los datos resultantes de la maqueta conectaremos el Arduino al ordenado mediante un cable USB. El código está preparado para que en el monitor serie se muestre el valor del ángulo en el que se encuentra. Los datos se pasan a un Excel y se generan las gráficas.

En la Figura 3.37 se puede observar cómo se ve afectado el sistema cambiando el valor de la constante proporcional K_p y cómo al aumentarlo en exceso el sistema se vuelve inestable.

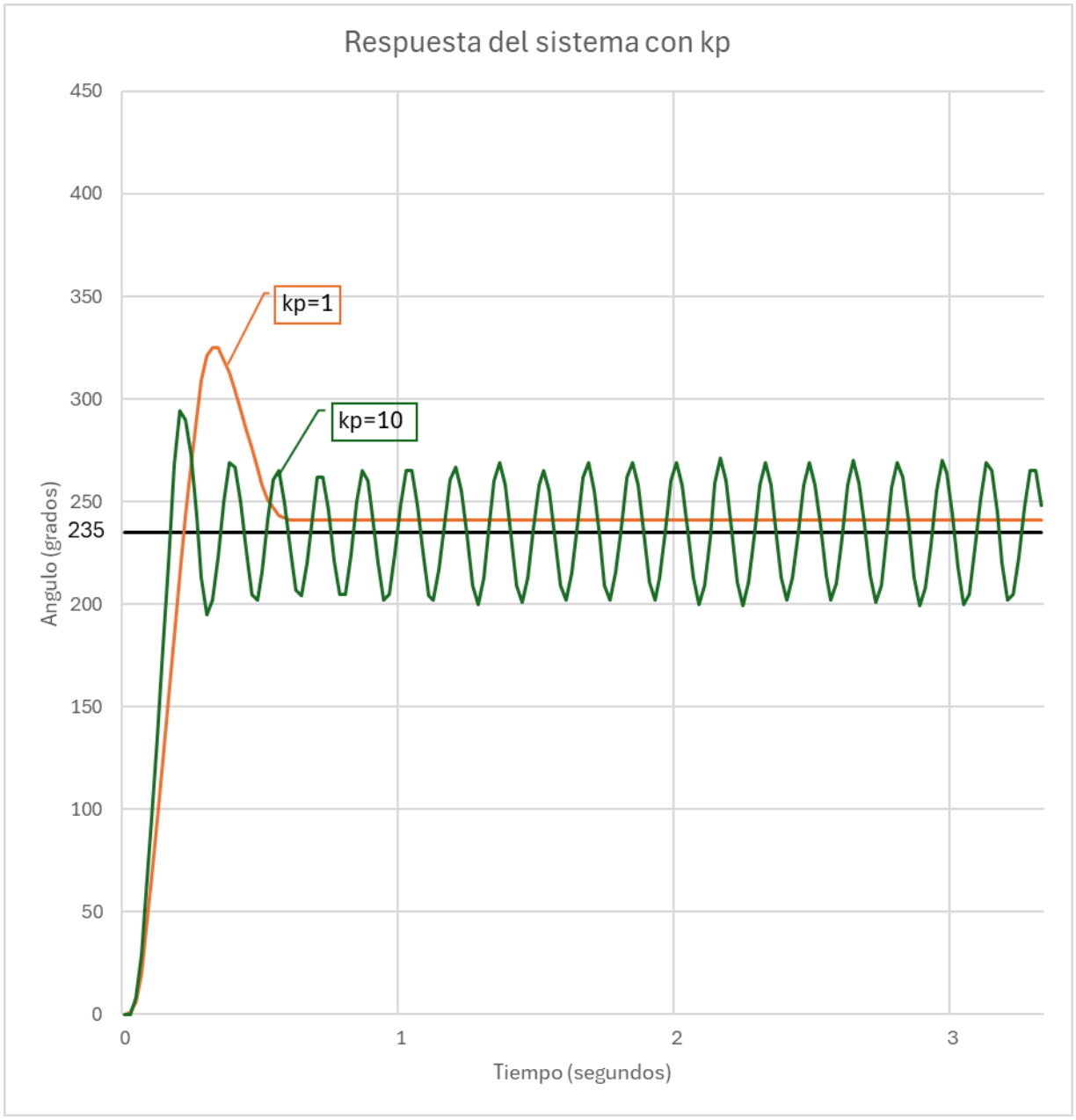


Figura 3.37 Respuesta del sistema con K_p

La parte integral (K_i) tiende a eliminar el error por lo que si K_i es muy pequeño tardará mucho en eliminarlo y si es muy grande el sistema se volverá inestable. En la Figura 3.38 se puede ver dicha respuesta variando la componente integral.

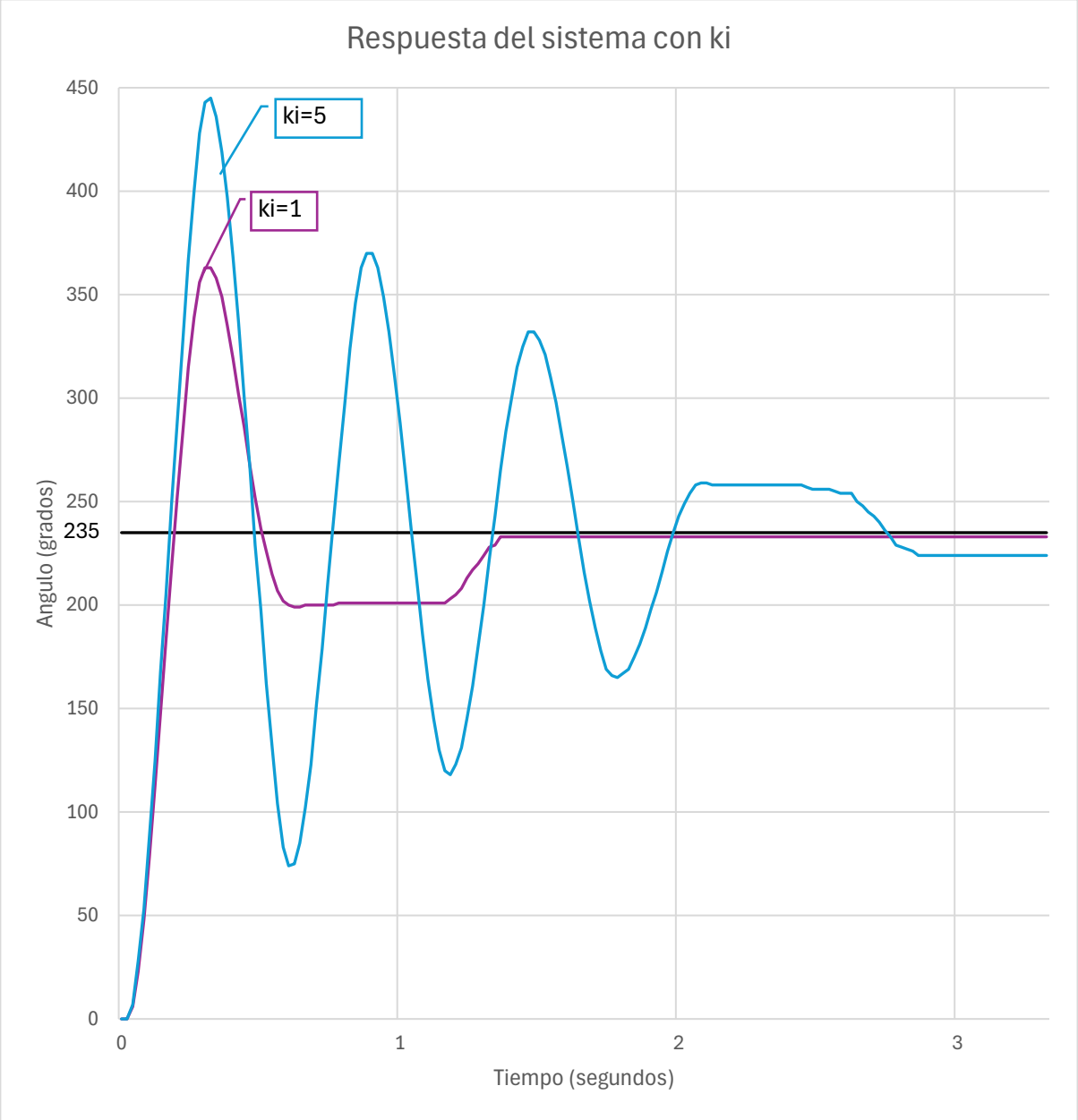


Figura 3.38 Respuesta del sistema con K_i

La parte derivativa (K_d) hace que el sistema sea más suave eliminando los rebotes, pero si el valor es muy grande aparecerán cambios bruscos en la respuesta. En la Figura 3.39 se puede apreciar el comportamiento del sistema frente al cambio de la parte derivativa.

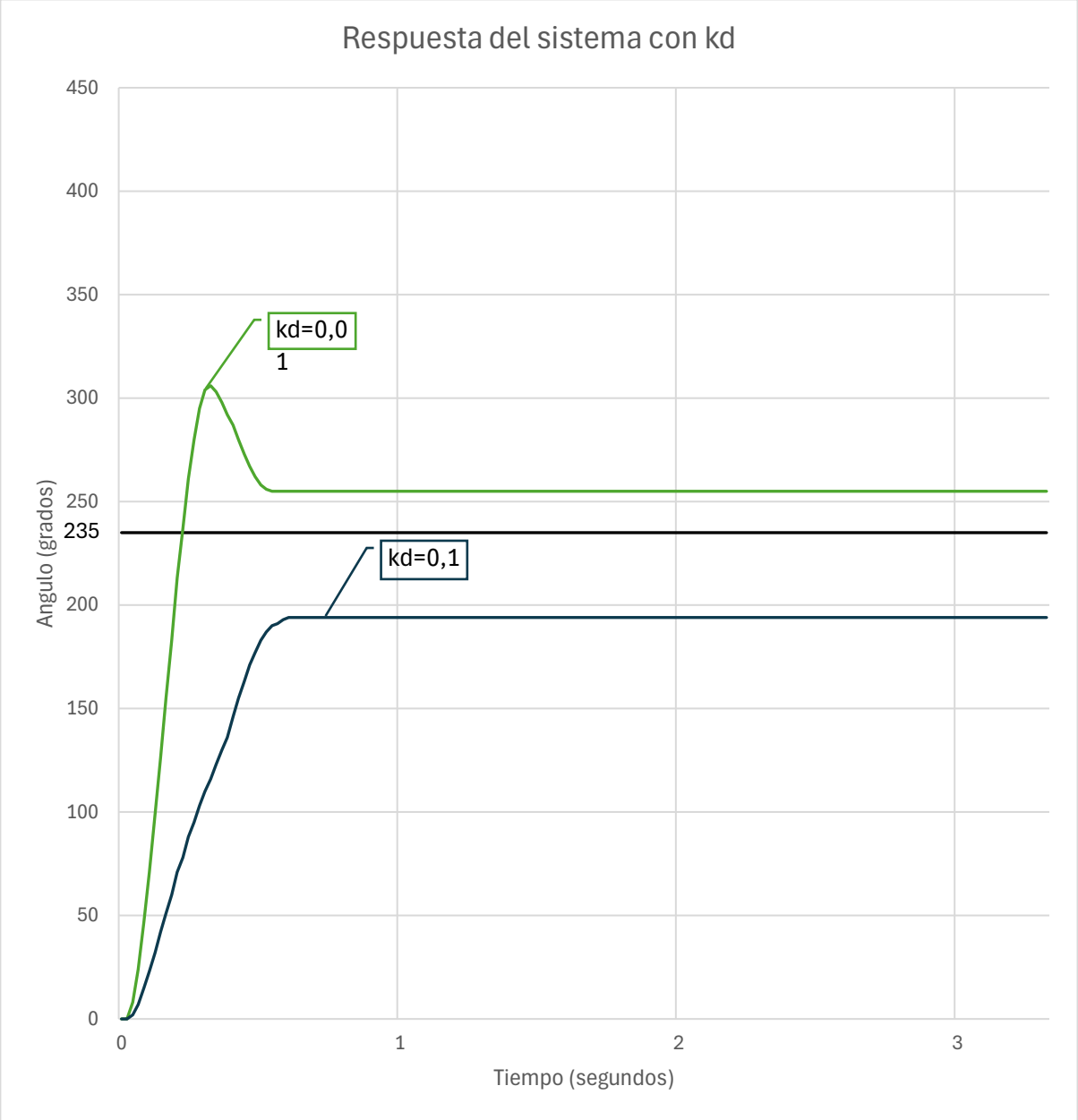


Figura 3.39 Respuesta del sistema con K_d

Al combinar las tres constantes se obtiene un controlador PID induciendo un comportamiento en el sistema como el mostrado en la Figura 3.40. Para obtener esta imagen en concreto se han ajustado los parámetros con los valores $\mathrm{K_p} = 1$, $\mathrm{K}_i = 1$ y $\mathrm{K_d} = 0{,}1$. Podemos observar que el sistema no llega a tener sobre impulso y se estabiliza manteniendo un error de 11 grados.

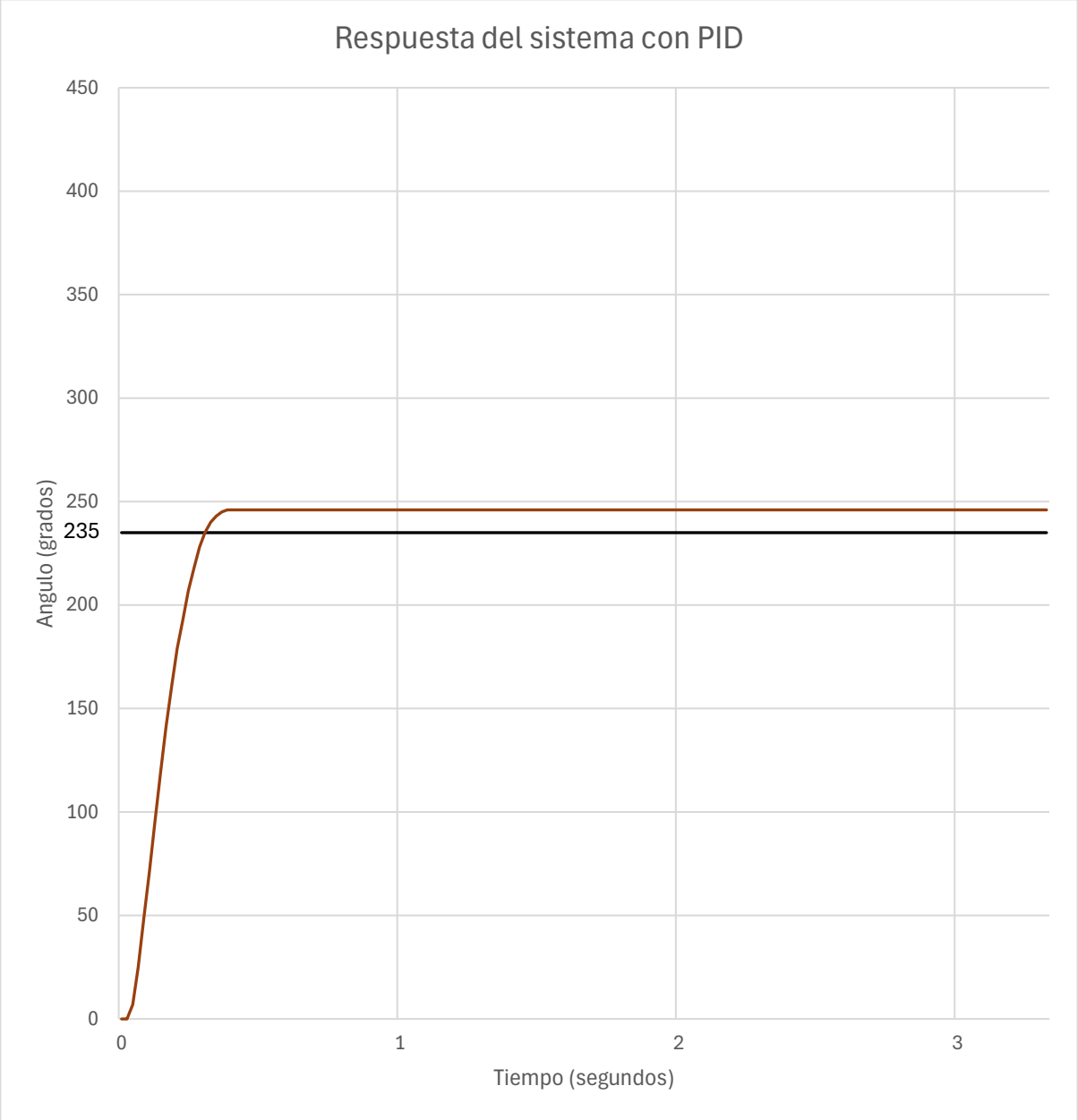


Figura 3.40 Respuesta del sistema con PID

4 CONTROL EN PC

Para la interacción con un PC se ha optado por la utilización de Python [3] que es un lenguaje de programación interpretado con una sintaxis clara y legible. Hemos escogido este frente a Matlab por su mayor flexibilidad y porque Matlab es un software propietario por lo que no es gratis.

4.1 ENTORNO DE DESARROLLO

Para crear el software que nos permita el control de la maqueta con el PC hemos escogido Anaconda [4][5] que es de código abierto y en su instalación incluye todos los programas para la creación de una interfaz gráfica, así como para escribir el código necesario.

Vamos a tener dos programas principales dentro del paquete de instalación de Anaconda que a continuación desarrollaremos, Spyder y Qt Designer.

4.1.1 Spyder

Este es el Entorno de Desarrollo Integrado (Figura 4.1) que está especialmente diseñado para la ingeniería y análisis de datos. Lleva una colección de herramientas y bibliotecas útiles para este fin y proporciona un espacio de trabajo ideal para escribir y depurar el código Python.

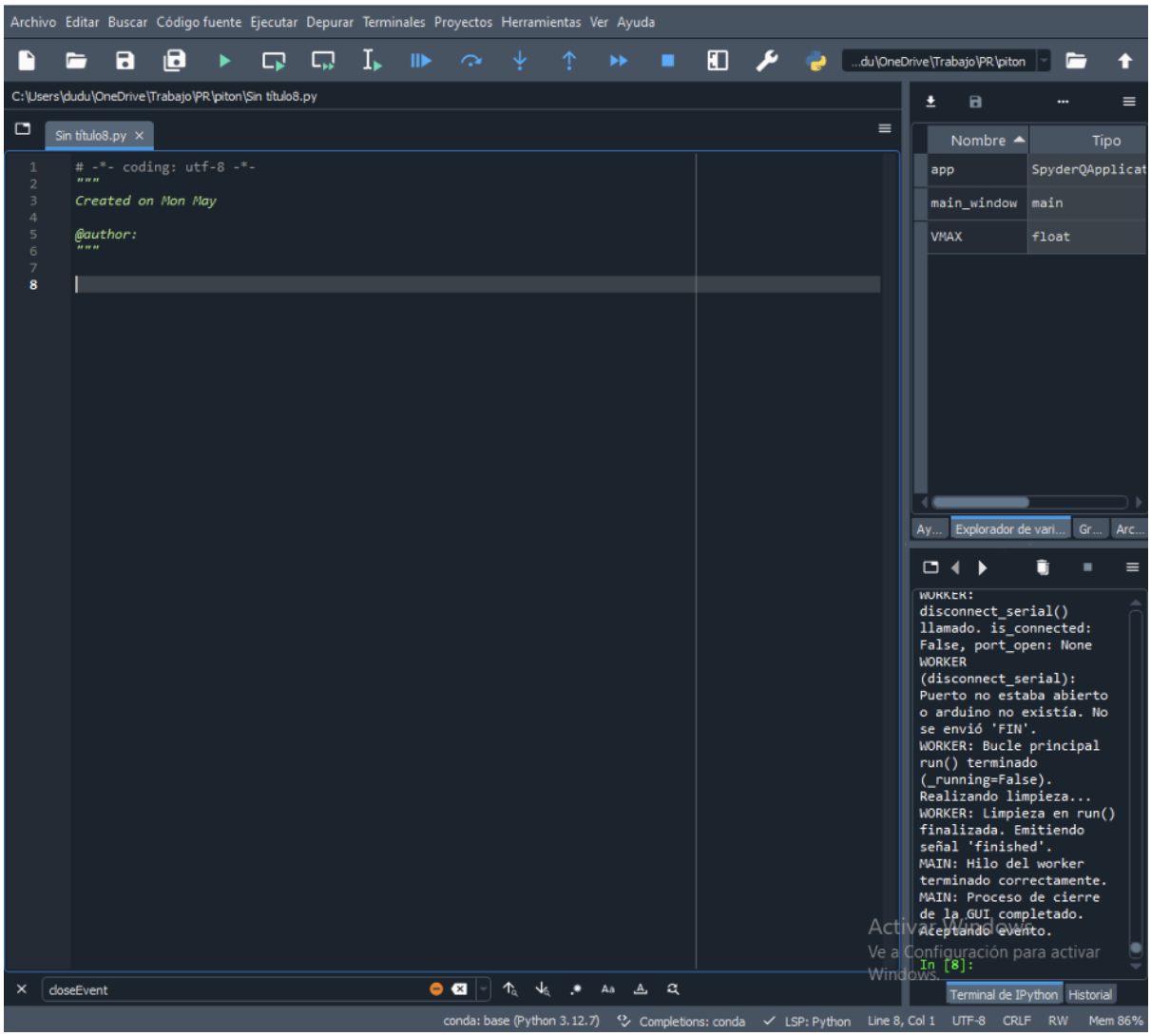


Figura 4.1 Spyder

4.1.2 Qt Designer

Para desarrollar la interfaz gráfica viene incluido el Qt Designer (Figura 4.2) que forma parte del framework Qt. Este programa nos permite diseñar y construir la interfaz de forma visual, arrastrando y soltando widgets (botones, deslizadores, textos, etc.) y configurar sus propiedades sin la necesidad de escribir el código manualmente. Este componente no aparece en el listado de aplicaciones cuando instalas Anaconda, por lo que hay que crear un acceso directo (C:\...\anaconda3\Library\bin).

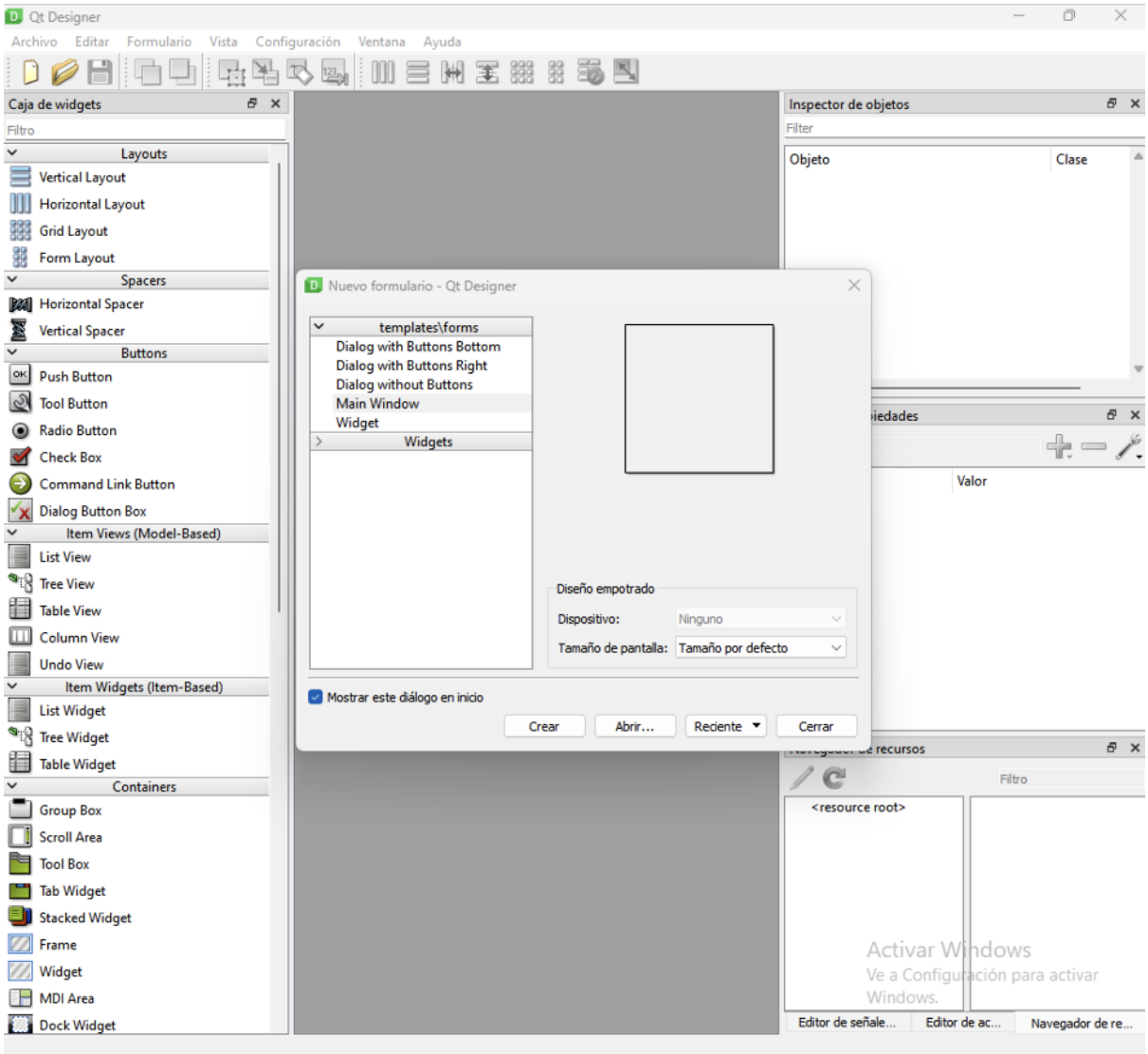


Figura 4.2 Qt Designer

4.1.3 Instalar librerías

Para instalar las librerías nuevas que sean necesarias se necesita abrir Anaconda Prompt (Figura 4.3).

Para instalar las librerías de Qt designer, si estas no viniesen, debemos escribir el comando correspondiente en el prompt, pulsar “intro” y esperar a que se instalen. El formato del comando es “pip install nombrelibreriainstalar”. Lo mismo sucede con cualquier librería que se desee instalar como por ejemplo la librería Serial que nos va a servir para la comunicación con Arduino. En este caso en concreto el comando es “pip install PySerial” [6][7].

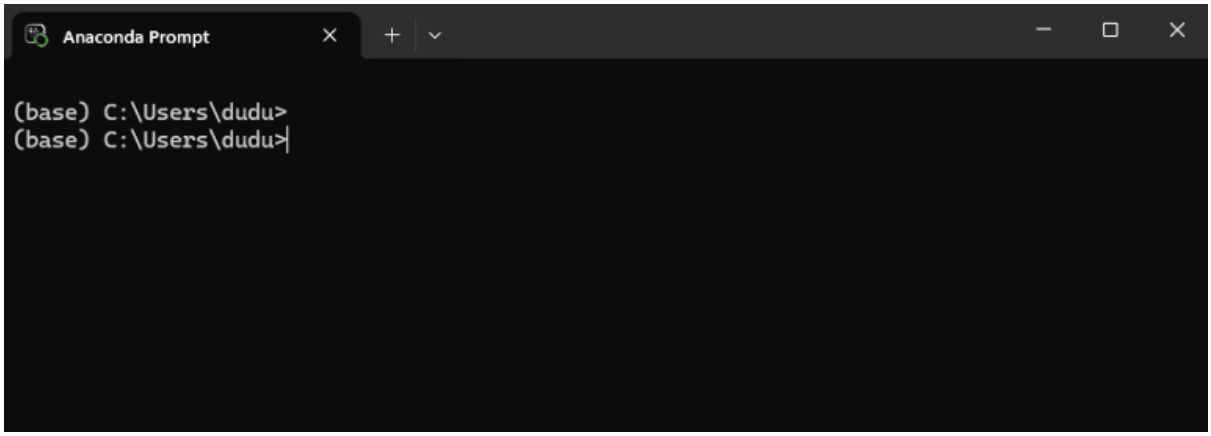


Figura 4.3 Anaconda Prompt

5 FUNCIONAMIENTO

5.1 ARDUINO

En este apartado nos centraremos en la interfaz gráfica, analizando su diseño y las posibles interacciones. El usuario podrá visualizar tres pantallas, la pantalla de calibración (Figura 5.1), la inicial (Figura 5.2) y una última que muestra el resultado (Figura 5.3).

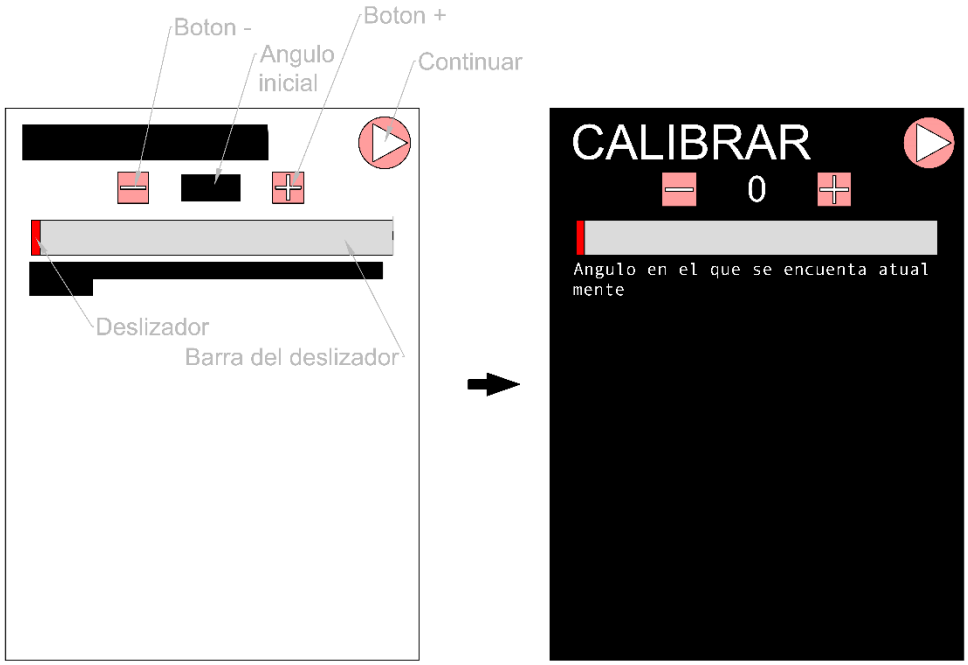


Figura 5.1 Pantalla calibración

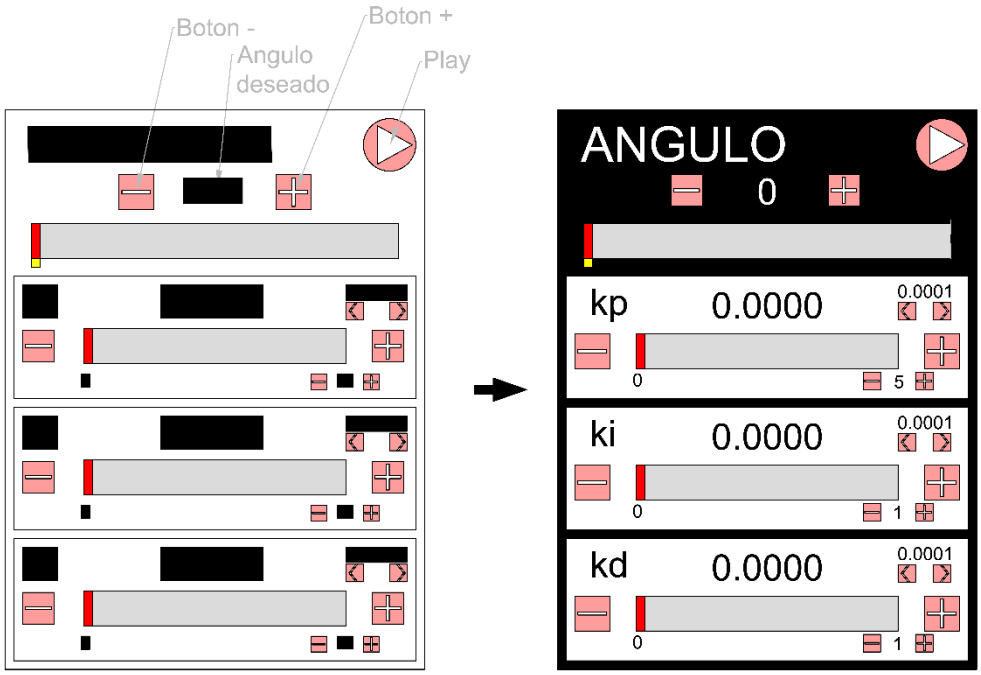


Figura 5.2 Pantalla inicial

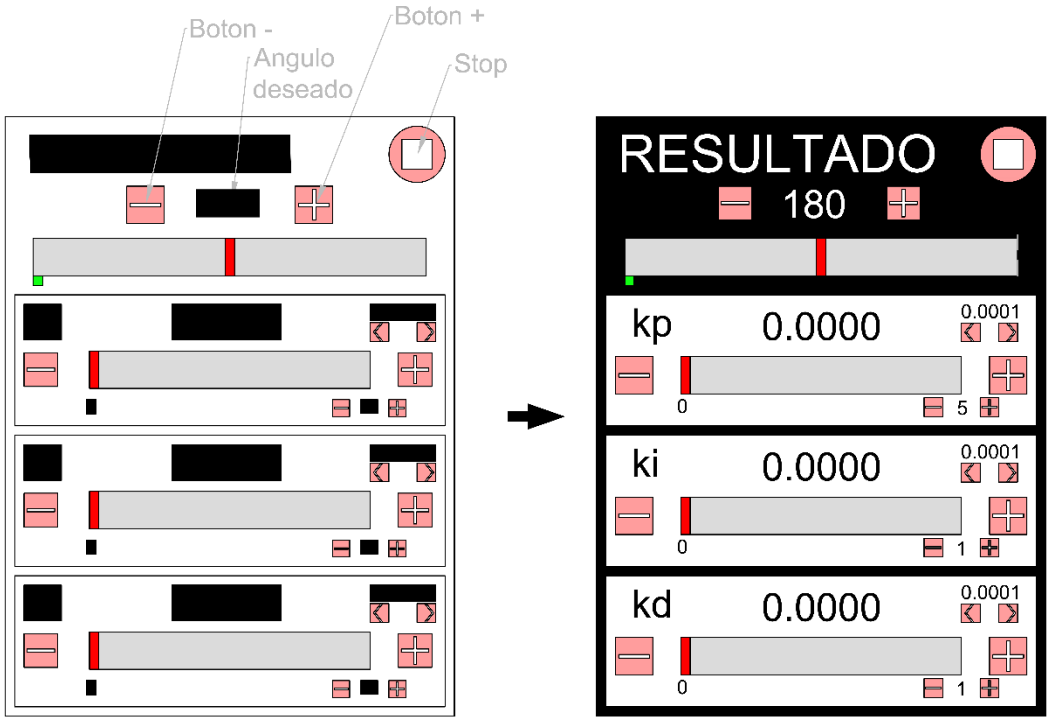


Figura 5.3 Pantalla resultado

Las tres pantallas tienen una sección en común (Figura 5.4) que muestra el título y nos permite distinguir el punto del proceso en el que nos encontramos. Junto al título podemos encontrar un botón para cambiar de pantalla, aunque hemos de tener en cuenta que la pantalla de calibración únicamente estará disponible al reiniciar el sistema (durante el proceso también podremos recalibrar de forma manual llevando la flecha al 0).

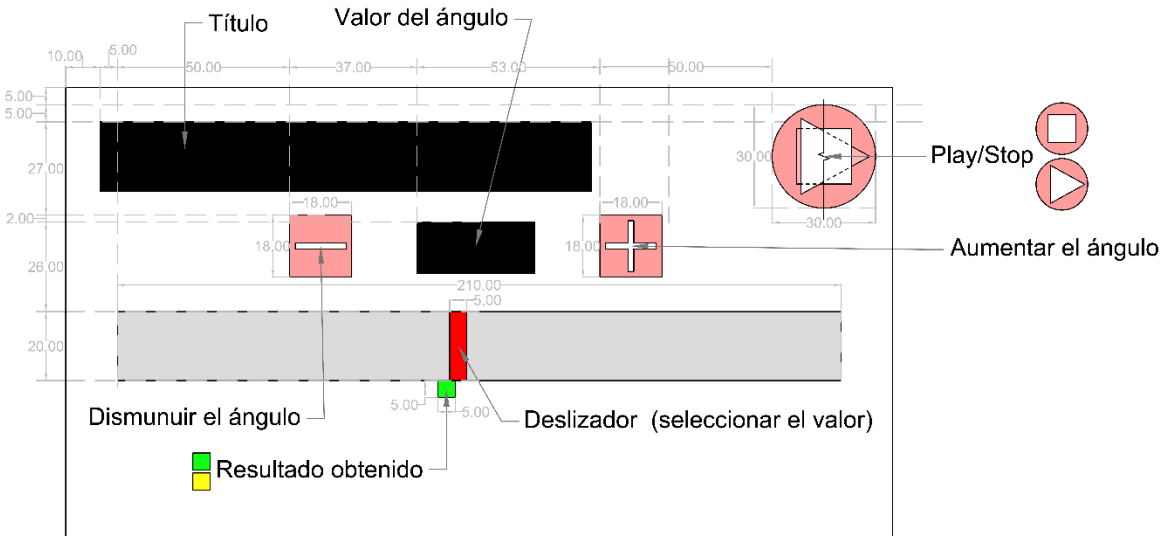


Figura 5.4 Zona común

La pantalla de calibración permite al sistema saber la posición de partida de la flecha. Podemos indicársela bien girando manualmente la flecha para empezar en el 0 (Figura 5.5) o bien variando el valor del ángulo en pantalla hasta que indique la situación visual de la flecha (Figura 5.6). Para variar el valor del ángulo en pantalla podemos mover directamente el deslizador y ajustar pulsando los botones “+” o “- “.

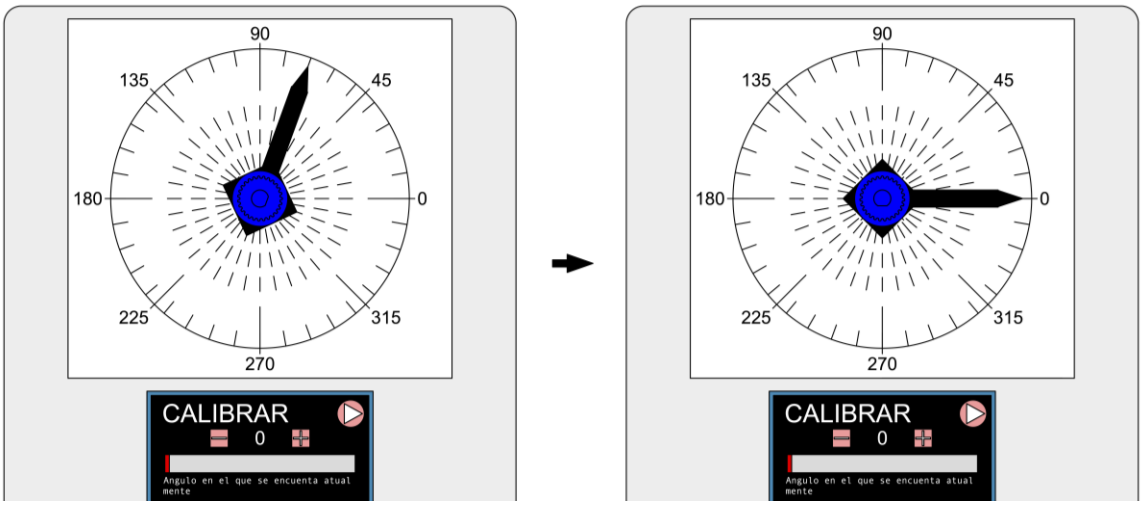


Figura 5.5 Calibración ejemplo 1

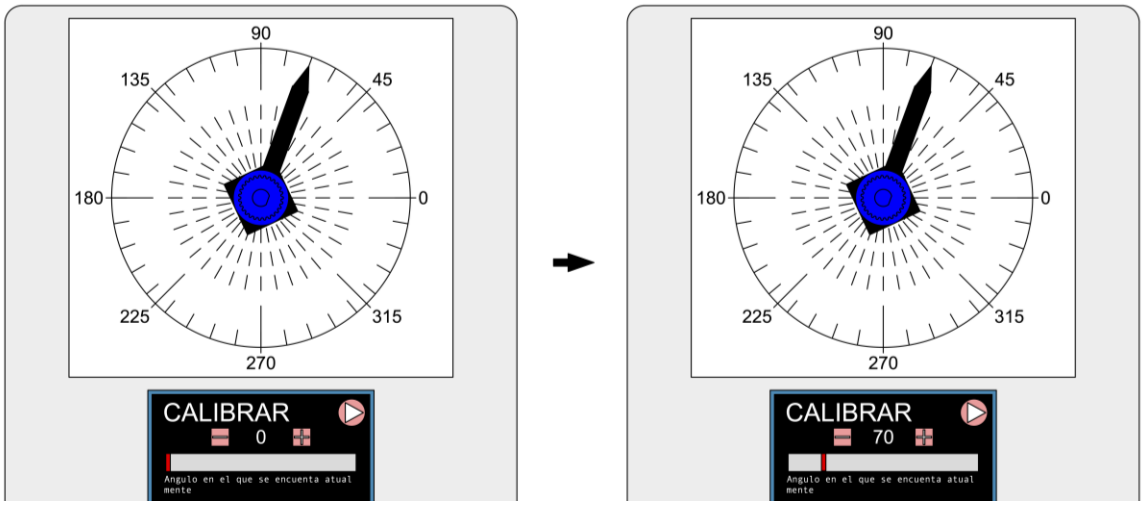


Figura 5.6 Calibración ejemplo 2

La pantalla inicial nos servirá para fijar el ángulo al que queremos llegar, así como los valores de cada una de las constantes. En la Figura 5.7 podemos ver en detalle las diferentes partes de las que se compone cada constante.

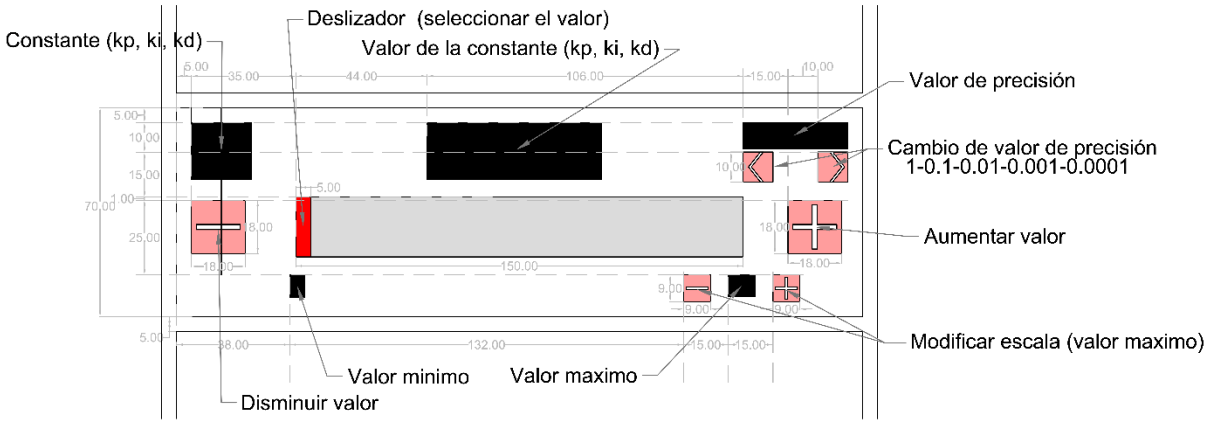


Figura 5.7 Zonas constantes

Mediante el deslizador o con los botones "+" y "-" ubicados en sus extremos podemos cambiar el valor de la constante que se nos irá mostrando en "Valor de la constante". Con las flechas "Cambio de valor de precisión" se cambia el número de decimales que va a tener la constante (podremos visualizarlo en "Valor de precisión") mientras que con los botones pequeños de "+" y "- " (Modificar escala) ajustamos el valor mínimo/máximo que podrá tener la constante.

Por último, al darle al Play en la pantalla inicial funcionará el motor para mover la flecha a la posición fijada y nos mostrará la "pantalla resultado" en la que también podremos modificar las constantes y el valor del ángulo deseado mientras el motor gira.

Para ver el error habría dos formas, mirando el ángulo al que apunta la flecha y comparándolo con el fijado (Figura 5.8) o viendo el desfase del cuadradito verde frente la posición del deslizador (Figura 5.9).

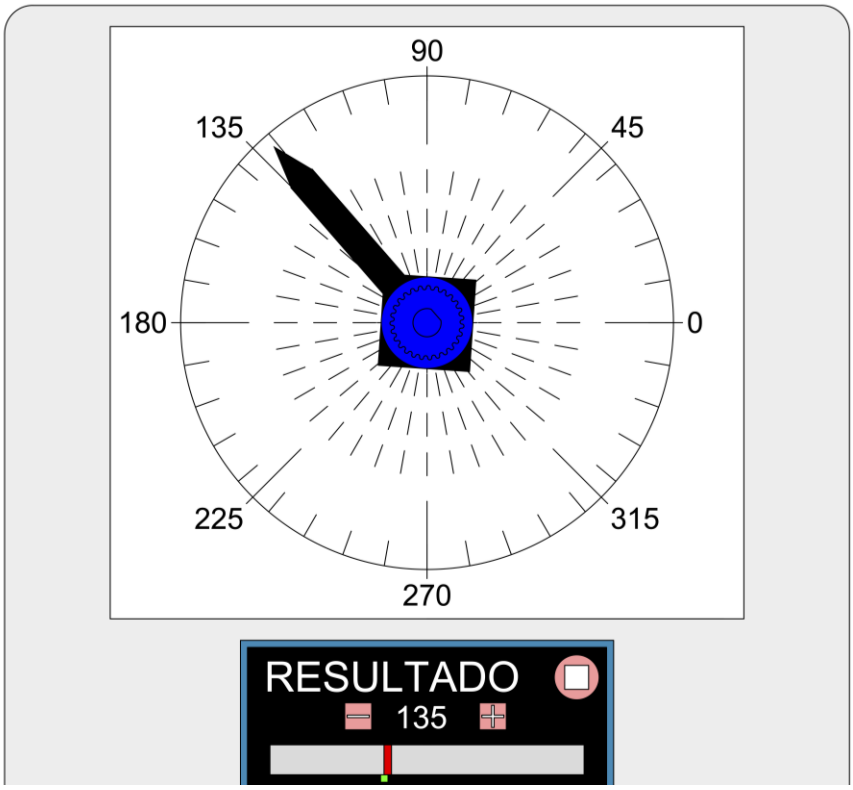


Figura 5.8 Resultado comparando flecha con ángulo fijado

Figura 5.9 Resultado comparando indicador verde con deslizador

5.2 PYTHON

Cuando arrancamos la aplicación, si no se detecta automáticamente la conexión con Arduino nos aparece sobre el programa un diálogo (Figura 5.10) que indica el puerto al que está intentando conectarse. Va rastreando los diferentes puertos disponibles en orden y si no consigue conectarse al primero pasa al siguiente. Cuando logra conectarse completamente debe recibir por el puerto serie “LISTO”.

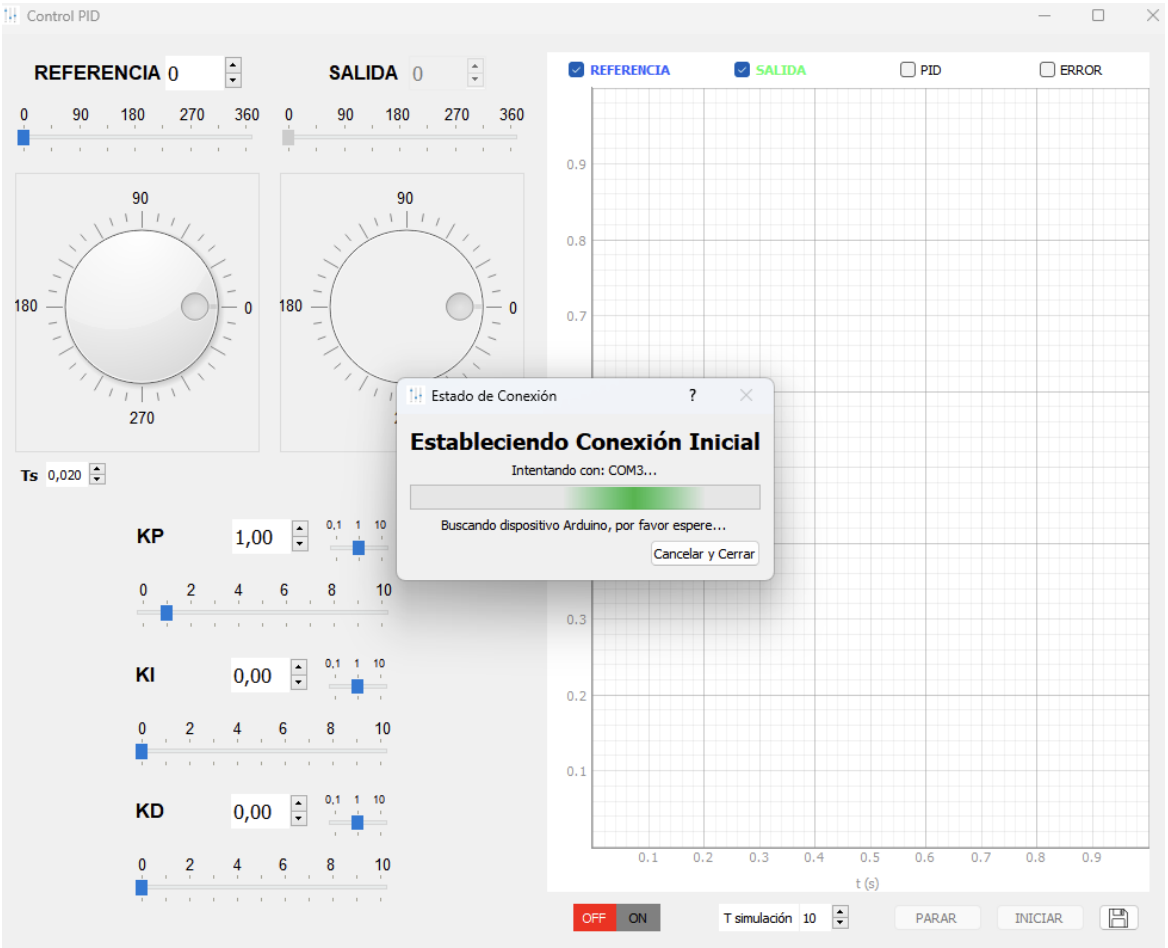


Figura 5.10 Diálogo conexión

En la Figura 5.11 podemos ver este diálogo con más detalle.

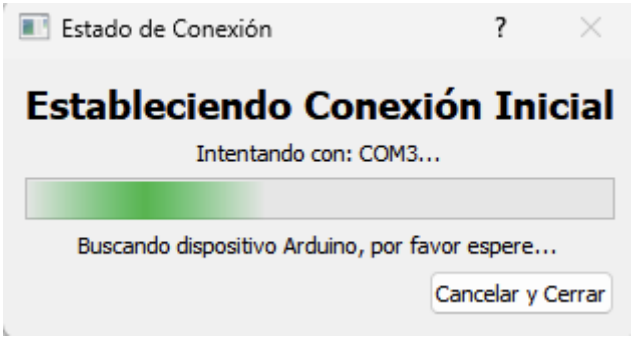


Figura 5.11 Dialogo de conexión ampliado

Si no se conecta aparecerá el diálogo de la Figura 5.12. Este nos servirá para dar instrucciones al usuario de lo que puede haber ocurrido para que falle la conexión y nos permitirá reintentar, cambiando el diálogo al de la Figura 5.13.

Figura 5.12 Diálogo de no conexión

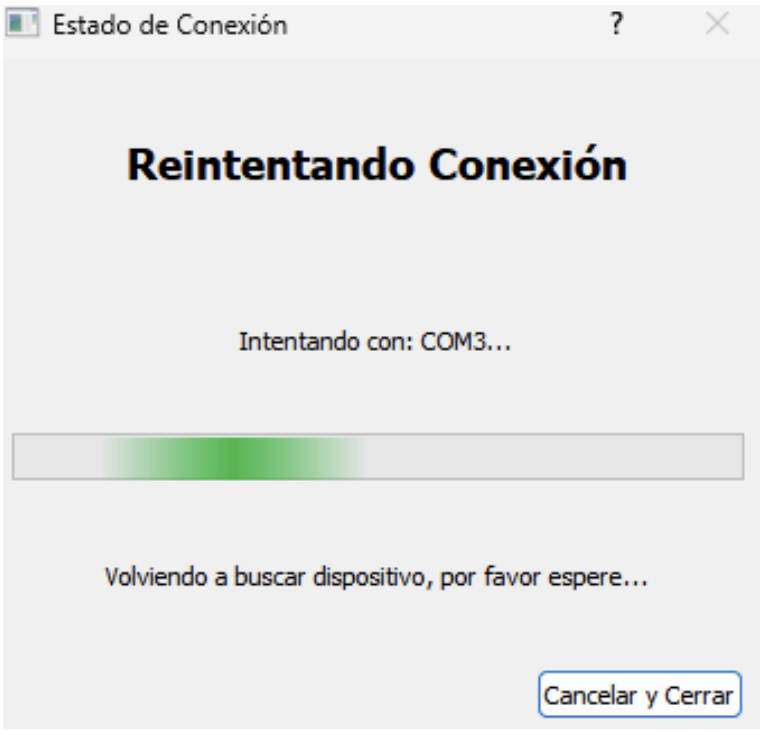


Figura 5.13 Diálogo de reintento de conexión

Cuando se ha establecido la conexión (Figura 5.14) nos aparece el ON en verde (Figura 5.15). Si se ha desconectado el cable o le hemos dado al OFF podemos seleccionar de nuevo el ON y que se reinicie el proceso de reconexión.

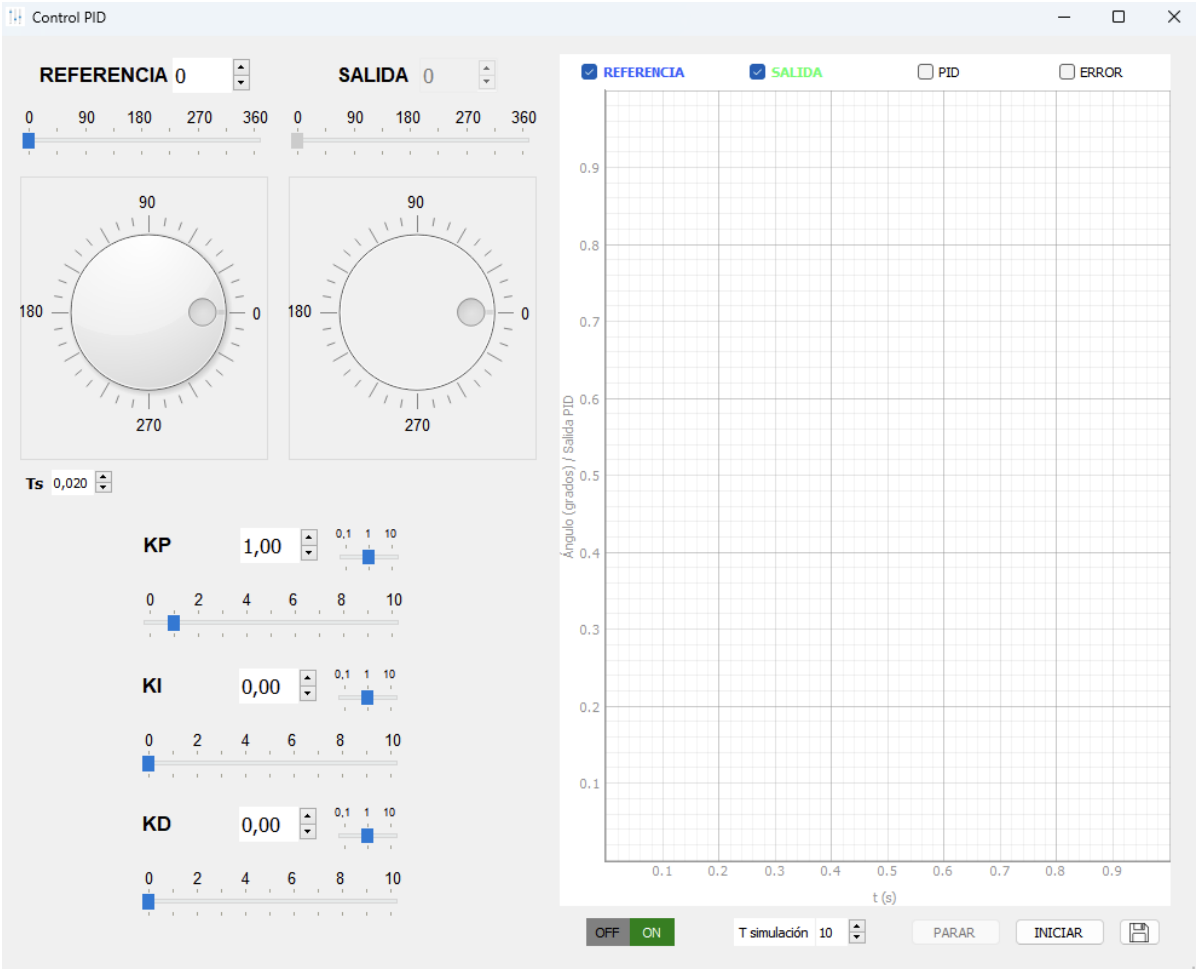


Figura 5.14 Interfaz gráfica

Figura 5.15 Conexión con Arduino

El funcionamiento del programa es intuitivo, pudiéndose controlar diferentes parámetros. Podemos escoger el tiempo de muestreo (Figura 5.16) que utiliza el sistema y esto nos servirá para ver cómo cambia el sistema al cambiar el tiempo que tarde en obtener los valores (posición del encoder dada por Arduino). Por defecto este valor es de 0,02 segundos y va a tener un mínimo de 0,015 segundos y un máximo de 1 segundo.

Figura 5.16 Tiempo de muestreo

Para escoger la referencia (ángulo deseado al que queremos llegar) lo podemos hacer de múltiples formas, bien escogiendo el ángulo en la rueda, bien mediante el deslizador que se encuentra justo encima o bien escribiendo el ángulo en el cuadro de texto (también se pueden pulsar la flechas que aparecen a la derecha). En la Figura 5.17 podemos ver estas tres opciones para la selección del ángulo. Al cambiar cualquiera de ellas se actualizan las otras.

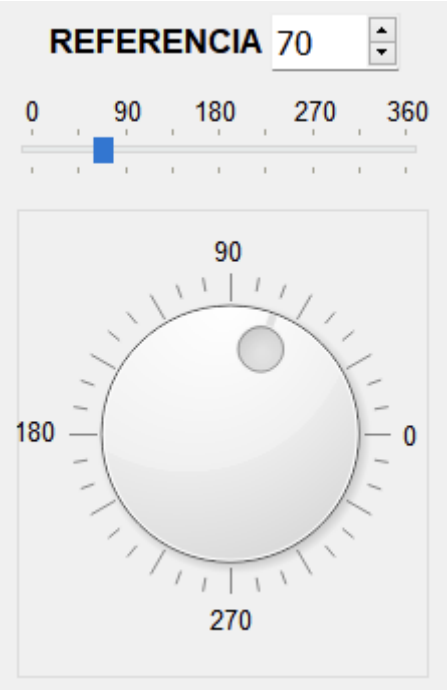


Figura 5.17 Panel de referencia

Para controlar las constantes de PID, como vemos en la Figura 5.18, tenemos tres partes. En este caso dos de ellas (el deslizador grande y el cuadro de texto) nos servirán para cambiar el valor de la constante, mientras que el deslizador a la derecha del cuadro de texto sirve para cambiar la escala. Por defecto solo K_p tiene valor = 1.

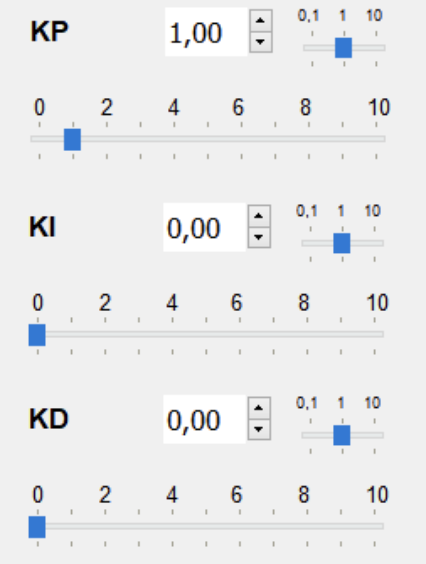


Figura 5.18 Panel de constantes PID

Podemos escoger el tiempo que queremos que esté el sistema en funcionamiento (Figura 5.19), por defecto 10 segundos. Una vez configurados estos valores solo debemos pulsar el botón “Iniciar” (Figura 5.20).

Figura 5.19 Tiempo de simulación

Figura 5.20 Botones Iniciar, ~~Parar~~

Al iniciar la simulación se activará el botón “Parar” y se desactivará el botón “Iniciar” (Figura 5.21). Esto sirve para detener la simulación si no deseamos esperar a que termine el tiempo de simulación previamente fijado.

Figura 5.21 Botones ~~Iniciar~~, Parar

Para la visualización de los datos en tiempo real tenemos una gráfica a la derecha en la que se pueden ver los resultados (Figura 5.22). Estos resultados se pueden activar o desactivar para verlos en la gráfica si se desea. Los valores que podemos ver son:

- Ángulo de referencia
- Ángulo resultado
- Valor PID
- Error

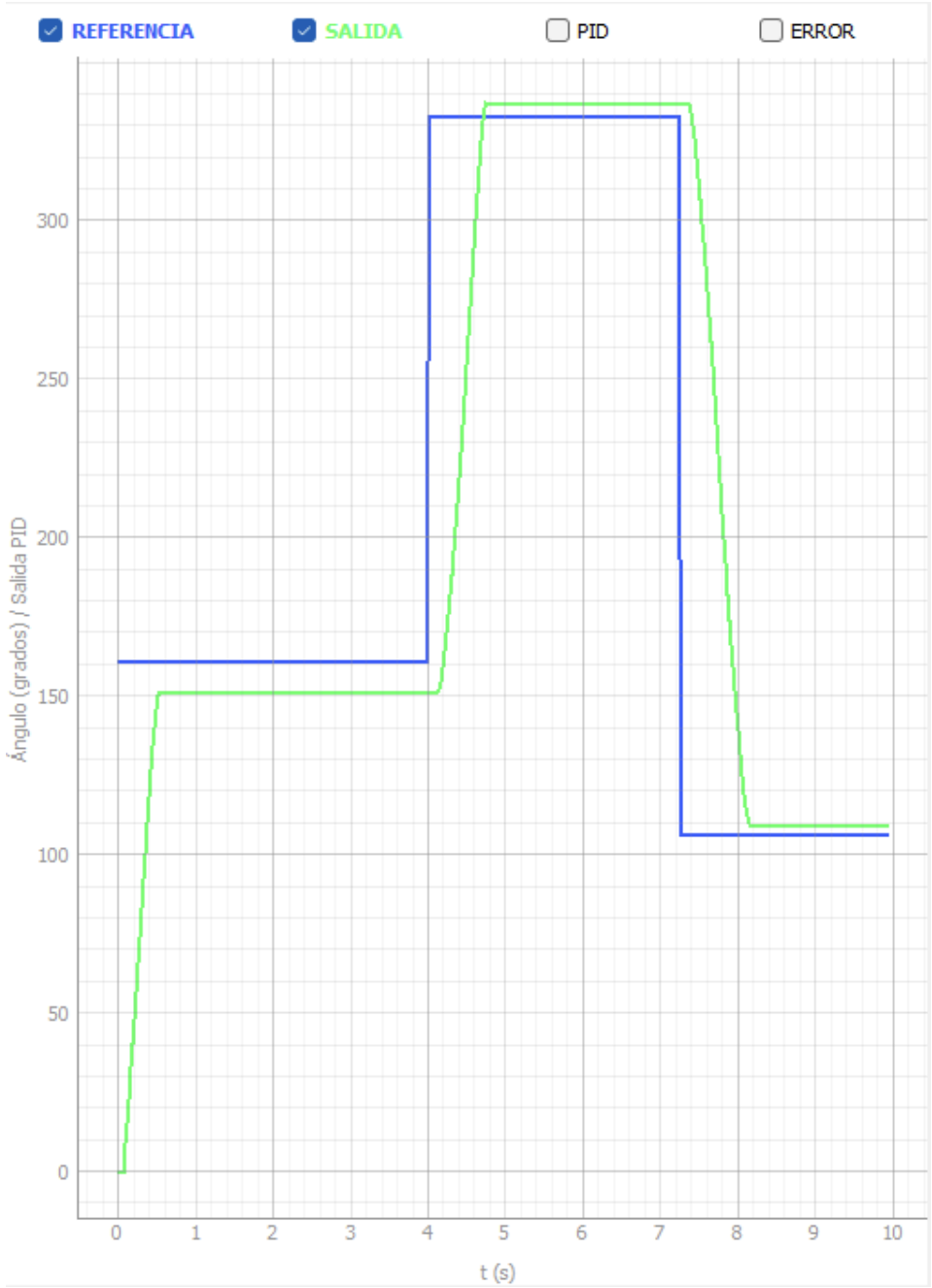


Figura 5.22 Grafica simulación

Los valores de estas gráficas se pueden exportar (en ".xlsx" y en ".mat") pulsando sobre el botón guardar (Figura 5.23). Al pulsar nos aparece una ventana en la cual tenemos que escoger la ubicación donde queremos que se guarde y el formato escogido (Figura 5.24).

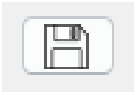


Figura 5.23 botón guardar

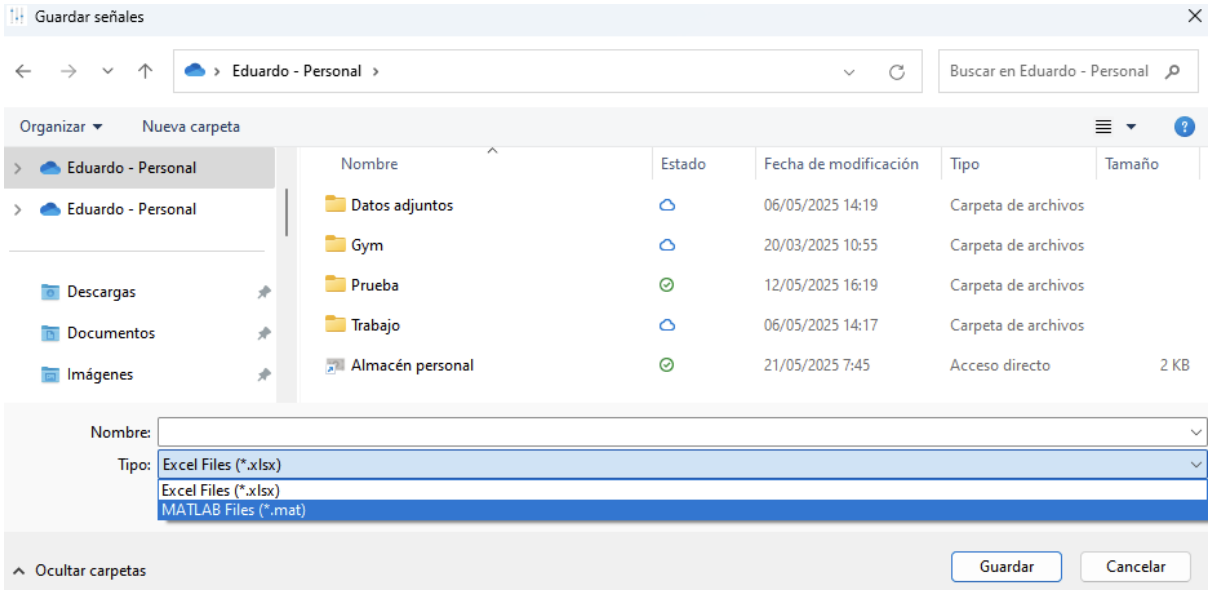


Figura 5.24 Diálogo guardar señales

6 PRESUPUESTO

A continuación, se muestra el desglose de presupuesto de ejecución de material.

UD.	DESCRIPCIÓN	PRECIO/UD (€)	CANTIDAD	TOTAL (€)
Ud.	MOTOR JGA25-371	26,99	1	26,99
Ud.	PLACA ARDUINO MEGA 2560 REV3	45,85	1	45,85
Ud.	ARDUINO MOTOR SHIELD REV3	29,99	1	29,99
Ud.	PANTALLA TACTIL TFT DE 2,8 PULGADAS	19,99	1	19,99
Ud.	PACK DE 40 CABLES MACHO/HEMBRA	5,5	1	5,5
Ud.	INTERRUPTOR DE PALANCA ON/OFF	4,2	1	4,2
Ud.	BATERÍA 9V DE 1200 mAh	9,95	1	9,95
Ud.	ADAPTADOR USB-C HEMBRA A MICRO USB MACHO	4,5	1	4,5
Ud.	CAJA CONTENEDORA	12,57	1	12,57
Ud.	ADAPTADOR PANEL USB-C	10,49	1	10,49
Ud.	ADAPTADOR USB-B A USB-C (3 Ud.)	7,99	0,33	2,64
				172,67

El presupuesto de ejecución de material asciende a la cantidad de **CIENTO SETENTA Y DOS EUROS CON SESENTA Y SIETE CÉNTIMOS.**

7 CONCLUSIONES

La realización de la maqueta didáctica presentada, así como de esta memoria me ha supuesto un reto a muchos niveles. Los dos primeros, compaginar su elaboración con mi jornada laboral y desempolvar conocimientos y apuntes de mi época de estudiante, me han llevado a ratificar que es mejor no dejar para mañana lo que puedas hacer hoy pero también a recordar una época no tan lejana en la que disfrutaba “enredando” en las prácticas.

El siguiente reto fue plasmar la idea que tenía en algo físico puesto que empecé implementando la idea con dos motores, de manera que al fijar la posición en uno el otro motor se alinease, pero no me pareció lo suficientemente visual ni atrayente. La idea de añadir una pantalla táctil surgió de esa necesidad de actualizar el concepto inicial de la maqueta y a mi parecer logra una mejor visualización e interacción con el resultado y los parámetros. A raíz de escoger esta opción surgieron otros problemas como por ejemplo utilizar pines que quedaban tapados al fusionar la pantalla con el Arduino lo cual solucioné añadiendo el motor Shield.

Otro de los retos superados era escoger una fuente de alimentación capaz de mover el sistema que fuese no solo funcional sino además portátil (peso y tamaño reducidos) lo que me llevó a escoger una pila recargable.

Conseguir una comunicación estable entre la maqueta y el PC a través de Python para que el tiempo de muestreo fuese el deseado en todo momento ha sido otro desafío salvado.

Finalmente dejar el código como yo quería, ordenado y reutilizable para ciertos procesos de la maqueta, me ha llevado más tiempo del que yo esperaba, pero he disfrutado del proceso y de todos los pequeños retos comentados que una vez superados han concluido en una maqueta final de la que me siento orgulloso.

8 REFERENCIA BIBLIOGRÁFICA

Arduino ide:

[1] *(16-02-2022)* *https://docs.arduino.cc/software/ide-v1/tutorials/Environment/*

Python:

[2] *(06-05-2025)* *https://es.python.org/aprende-python/*

[3] *(06-05-2025)* *https://docs.python.org/es/3.13/reference/index.html*

Anaconda:

[4] *(06-05-2025)* *https://www.anaconda.com*

[5] *(06-05-2025)* *https://entrenamiento-data-scientist-python.*

Comunicación serial Arduino-Python:

[6] *(06-05-2025)* *https://omes-va.com/comunicacion-python-arduino-pyserial-mediapipe-opencv/*

[7] *(06-05-2025)* *https://pythonhosted.org/pyserial/*

Teoría controlador PID:

[8] *(11-02-2024)* *https://www.luisllamas.es/teoria-de-control-en-arduino-el-controlador-pid/*

[9] *(11-02-2024)* *https://www.luisllamas.es/como-ajustar-un-controlador-pid-en-arduino/*

Motor JGA25-371:

[10] *(14-02-2022)* *https://www.amazon.es/*

Encoder:

[11] *(20-02-2022)* *https://www.servomotorsadjust.com/encoders/*

Placa Arduino Mega 2560 REV3:

[12] *(14-09-2023)* *https://www.amazon.es/*

[13] *(16-09-2023)*https://store.arduino.cc/products/arduino-mega-2560-rev3

Arduino MOTOR SHIELD REV3:

[14] *(23-03-2024)* *https://www.amazon.es/*

[15] *(30-03-2024)* *https://store.arduino.cc/products/arduino-motor-Shield-rev3*

Pantalla táctil TFT DE 2,8 Pulgadas:

[16] *(14-03-2024)* *https://www.amazon.es/*

[17] *(16-03-2024)* *https://www.elegoo.com/pages/download*

Pack de cables:

[18] *(14-03-2024)* *https://www.amazon.es/dp/B09126W311?ref_=cm_sw_r_cso_wa_apan_dp*

Interruptor de palanca

[19] *(21-06-2024)* *https://www.amazon.es/dp/B08JVF6FJT?ref_=ppx_hzsearch_conn_dt_b_fed_asin_title_1*

Batería de 12V

[20] *(19-06-2024)* *https://www.amazon.es/dp/B09QG52XYK?ref_=ppx_hzsearch_conn_dt_b_fed_asin_title_7*

Adaptador USB-C hembra a micro USB macho

[21] *(20-07-2024)* *https://www.amazon.es/*

Adaptador USB-B a USB-C

[22] *(24-05-2025)* *https://www.amazon.es/*

Adaptador montaje en panel USB-C

[23] *(24-05-2025)* *https://www.amazon.es/*

Puente H L293D

[24] *(20-02-2022)* *laelectronica*

Caja contenedora

[25] *(10-06-2024)* *https://www.amazon.es/*

9 ANEXO I: CÓDIGO

9.1 ARDUINO

Para la programación de la maqueta se ha optado por la creación de diferentes librerías además de las necesarias para el control de la pantalla. Una primera librería (Código 1) donde se definen los pines de la pantalla y parámetros que se van a necesitar más adelante.

```
1   #ifndef _Untitled_H_
2   #define _Untitled_H_
3   //Configuracion de la pantalla
4   #define YP A3  // must be an analog pin, use "An" notation!
5   #define XM A2  // must be an analog pin, use "An" notation!
6   #define YM 9   // can be a digital pin
7   #define XP 8   // can be a digital pin
8
9   #define TS_MINX 110
10  #define TS_MAXX 930
11
12  #define TS_MINY 110
13  #define TS_MAXY 930
14
15  #define MINPRESSURE 10
16  #define MAXPRESSURE 1000
17
18  #define LCD_CS A3
19  #define LCD_CD A2
20  #define LCD_WR A1  //A9
21  #define LCD_RD A0  //A8
22  // Opcional
23  #define LCD_RESET A4
24
25  // Colores
26  #define negro 0x0000
27  #define azul 0x001F
28  #define rojo 0xF800
29  #define rojosuave 0xFF1C
30  #define gris 0xe73c
31  #define verde 0x07E0
32  #define cyan 0x07FF
33  #define magenta 0xF81F
34  #define amarillo 0xFFE0
35  #define blanco 0xFFFF
36  // Colores Boton
37  #define bColor gris
38  #define bColors rojosuave
39
40  #define espera 100
41  #define decimal 4
42  struct text {
43    int x;
44    int y;
45    String txt;
46    int size;
47    uint16_t color;
48  };
49  struct boton {
```

```
50    int x;
51    int y;
52    int size;
53    int alto;
54    int ancho;
55    int pulsado;
56    String simbolo;
57    uint16_t colortxt;
58    uint16_t color;
59    uint16_t colorpulsado;
60    int radio;
61  };
62  struct deslizar {
63    int x;
64    int y;
65    int altura;
66    int anchura;
67    uint16_t colorbarra;
68    uint16_t color;
69  };
70  struct valores {
71    double min;
72    double max;
73    double x;
74    double x0;
75    double valor;
76  };
77  struct escala {
78    double s[5];
79    int ns;
80    double sd[4];
81    int nsd;
82  };
83  struct fondo {
84    int x;
85    int y;
86    int altura;
87    int anchura;
88    uint16_t color;
89  };
90
91  struct seccion {
92    text titulo;
93    text valortxt;
94    valores valor;
95    text escalatxts;
96    escala escal;
97    boton flechai;
98    boton flechad;
99    boton mas;
100   boton menos;
101   boton masp;
102   boton menosp;
103   text escaladestxt1sd;
104   text escaladestxt2sd;
105   deslizar des;
106   fondo fond;
107   text comentario1;
108   text comentario2;
109   boton iniciar;
110 };
111 //parametros para caracteristicas del encoder
```

```
112  //struct encoder{unsigned long t;unsigned long t0;double reductor;double resolucion;double error;double error0;double
     derror;double ierror;int valor;int valor0;}
113  struct encoder {
114    unsigned long t = 0;
115    unsigned long t0 = 0;
116    double reductor = 1;
117    double resolucion = 1;
118    double error = 0;
119    double error0 = 0;
120    double derror = 0;
121    double ierror = 0;
122    double ierror0 = 0;
123    int valor = 0;
124    int valor0 = 0;
125    int posicion = 0;
126    int valorbuscado = 0;
127    double PID = 0;
128    double kp = 0;
129    double ki = 0;
130    double kd = 0;
131    double PID0 = 0;
132    int contador = 0;
133  };
134
135  #endif
```

Código 1 Parametros.h

Para crear el resto lo haremos mediante dos archivos:

- Un fichero de cabecera que tiene la extensión .h el cual contiene las definiciones de la librería.
- Un fichero con la extensión .cpp donde estará el código.

```
1   #ifndef _CONTROLADOR_H_
2   #define _CONTROLADOR_H_
3   #include<Arduino.h>
4   #include <Elegoo_TFTLCD.h>
5   #include"parametros.h"
6   #include"funciones.h"
7   extern double reductor;
8   extern double resolucion;
9   extern Elegoo_TFTLCD tft;
10  void pid(encoder*, int, int,int*,seccion);
11  void girarmotor(encoder*, int, int,int*);
12  void pararMotor(int);
13  void verresultado(encoder* ,seccion );
14  #endif
```

Código 2 Controlador.h

```
1   #include "controlador.h"
2   //struct encoder{unsigned long t;unsigned long t0;double reductor;double resolucion;double error;double error0;double
    derror;double ierror;int valor;int valor0;}
3   void pid(encoder *enc, int direccion, int enA, int *d, seccion p) {
4     //guardamos el tiempo actual
5     enc->t = millis();
6     double st = 0.02;  //tiempo de muestreo
7     if (enc->t - enc->t0 >= st * 1000) {
8       //convertimos la posicion del encoder en angulo
9       enc->valor = round(doublemap(enc->posicion, 0, enc->reductor * enc->resolucion, 0, 360));
10      //calculamos los errores (actual derivativo e integral)
11      enc->error = enc->valorbuscado - enc->valor;
```

```
    //calculamos la derivada del error (error actual-anterior)
    enc->derror = -(enc->valor - enc->valor0) / st;
    //calculamos la integral del error (error actual+anterior)*st/2
    enc->ierror += enc->ki * (enc->error + enc->error0) * st / 2;
    enc->ierror = constrain(enc->ierror, -255, 255);
    //calculamos la salida del regulador
    enc->PID = enc->kp * enc->error + enc->ierror + enc->kd * enc->derror;
    //pasamos el valor por un un filtro ya que el valor va de -255 a255
    enc->PID = constrain(enc->PID, -255, 255);
    //llamamos a la funcion que hace girar el motor
    girarmotor(enc, direccion, enA, d);
    verresultado(enc, p);
    //Guardamos el valor a valor anterior
    enc->valor0 = enc->valor;
    enc->ierror0 = enc->ierror;
    //Guardamos el error a error anterior
    enc->error0 = enc->error;
    enc->PID0 = enc->PID;
    //mostramos el resultado
    Serial.println(enc->valor);
    //Guardamos el tiempo a tiempo anterior
    enc->t0 = enc->t;
  }
}
void verresultado(encoder *enc, seccion p) {
  if (enc->valor0 != enc->valor) {
    int xant = doublemap(enc->valor0, p.valor.min, p.valor.max, p.des.x, p.des.x + p.des.anchura - 5);  //posicion anterior
    int xact = doublemap(enc->valor, p.valor.min, p.valor.max, p.des.x, p.des.x + p.des.anchura - 5);   //posicion actual
    tft.fillRect(xant, p.des.y, 5, 5, negro);                                    //deslizador
    tft.fillRect(xact, p.des.y, 5, 5, verde);                                    //deslizador
  }
}
void girarmotor(encoder *enc, int direccion, int enA, int *d) {
  if (enc->PID > 0 & *d != 1) {
    *d = 1;
    digitalWrite(direccion, HIGH);
  } else if (enc->PID < 0 & *d != 0) {
    *d = 0;
    digitalWrite(direccion, LOW);
  }
  analogWrite(enA, abs(enc->PID));
}
void pararMotor(int enA) {
  analogWrite(enA, 0);
}
```

Código 3 Controlador.cpp

```
#ifndef _TEXTOS_H_
#define _TEXTOS_H_

#include <Arduino.h>
#include <Elegoo_TFTLCD.h>
#include "parametros.h"

extern Elegoo_TFTLCD tft;
//extern  linea1;

void txt(text);
void txtvalor(double, int, text, int);

#endif
```

Código 4 Textos.h

```
#include "textos.h"

void txt(text titulo) {
  tft.setTextColor(titulo.color, negro);
  tft.setTextSize(titulo.size);
  tft.setCursor(titulo.x, titulo.y);
  tft.println(titulo.txt);
}
void txtvalor(double n, int decimales, text t, int modo) {
  tft.setTextColor(t.color, blanco);
  tft.setTextSize(t.size);
  tft.setCursor(t.x, t.y);
  if (modo == 0) {
    tft.setTextColor(t.color, negro);
    char aux[16];
    if (n < 10) sprintf(aux, " %d ", int(n));
    else if (n < 100) sprintf(aux, " %2d", int(n));
    else sprintf(aux, "%d", int(n));
    tft.println(aux);
  } else if (modo == 2) {
    tft.setTextColor(t.color, blanco);
    char aux[16];
    if (n < 10) sprintf(aux, "%d ", int(n));
    else sprintf(aux, "%d", int(n));
    tft.println(aux);
  } else if (modo == 3) {
    tft.setTextColor(t.color, blanco);
    char aux[16];
    if (n < 10) sprintf(aux, " %d ", int(n));
    else if (n < 100) sprintf(aux, " %2d", int(n));
    else sprintf(aux, "%d", int(n));
    tft.println(aux);
  } else tft.println(n, decimales);
}
```

Código 5 Textos.cpp

```
#ifndef _PANTALLAS_H_
#define _PANTALLAS_H_

#include <Arduino.h>
#include <Elegoo_TFTLCD.h>
#include "parametros.h"
#include "textos.h"
#include "barra_deslizador.h"
#include "botones.h"

extern seccion p11;
extern seccion p21;
extern seccion p31;
extern seccion kp;
extern seccion kd;
extern seccion ki;
extern int modo;

void verconstantes(seccion);
void pantallacalibracion(void);
void pantallaangulo(void);
void pantallaresultado(void);
void pantallaconstantes(void);
void fondoconstantes(fondo);

#endif
```

Código 6 Pantallas.h

```
#include "pantallas.h"

void pantallacalibracion() {
  //encabezado (linea1)
  txt(p11.titulo);
  txtvalor(double(p11.valor.valor), 0, p11.valortxt, 0);
  txt(p11.comentario1);
  txt(p11.comentario2);
  barradeslizadora(p11.des);
  deslizador(p11.valor, p11.des);
  botons(p11.mas);
  botons(p11.menos);
  boton_play(p11.iniciar);
}
void pantallaangulo() {
  txt(p21.titulo);
}
void pantallaresultado() {
  txt(p31.titulo);
}
// struct
constantes{titulo;valortxt;valor;escalatxt;escala;flechai;flechad;mas;menos;masp;menosp;escaladestxt1;escaladestxt2;des;};
void verconstantes(seccion p) {
  fondoconstantes(p.fond);

  txt(p.titulo);
  txtvalor(double(p.valor.valor), decimal, p.valortxt, 1);
  barradeslizadora(p.des);
  deslizador(p.valor, p.des);
  botons(p.mas);
  botons(p.menos);
  botons(p.flechad);
  botons(p.flechai);
  botons(p.masp);
  botons(p.menosp);
  txtvalor(p.escal.s[p.escal.ns], decimal, p.escalatxts, 1);
  txtvalor(0, 0, p.escaladestxt1sd, 1);
  txtvalor(p.escal.sd[p.escal.nsd], 0, p.escaladestxt2sd, 3);
}

void pantallaconstantes() {
  verconstantes(kp);
  verconstantes(ki);
  verconstantes(kd);
}

void fondoconstantes(fondo p) {
  tft.fillRect(p.x, p.y, p.anchura, p.altura, p.color);
}
```

Código 7 Pantallas.cpp

```
#ifndef _BARRA_DESLIZADOR_H_
#define _BARRA_DESLIZADOR_H_

#include <Arduino.h>
#include <Elegoo_TFTLCD.h>
#include "parametros.h"

extern Elegoo_TFTLCD tft;
```

```
9
10  void deslizador(valores, deslizar);
11  void barradeslizadora(deslizar);
12
13  #endif
```

Código 8 Barra_deslizador.h

```
1   #include "barra_deslizador.h"
2
3   void deslizador(valores valor, deslizar des)  //x posicion nueva; x0 posicion inicial; y posicion en eje y
4   {
5     if (valor.valor >= valor.min && valor.valor <= valor.max) {
6       tft.fillRect(valor.x0, des.y, 5, des.altura, des.colorbarra);  //Posicion anterior del deslizador
7       tft.fillRect(valor.x, des.y, 5, des.altura, des.color);        //Posicion nueva del deslizador
8     }
9   }
10  void barradeslizadora(deslizar des) {
11    tft.fillRect(des.x, des.y, des.anchura, des.altura, des.colorbarra);  //Posicion nueva del deslizador
12  }
```

Código 9 Barra_deslizador.cpp

```
1   #ifndef _BOTONES_H_
2   #define _BOTONES_H_
3
4   #include <Arduino.h>
5   #include <Elegoo_TFTLCD.h>
6   #include "parametros.h"
7
8   extern Elegoo_TFTLCD tft;
9
10  void botons(boton);
11  void boton_play(boton);
12  void boton_stop(boton);
13
14  #endif
```

Código 10 Botones.h

```
1   #include "botones.h"
2
3   //struct boton{int x;int y; int size;int alto;int ancho;int pulsado;String simbolo;uint16_t color;uint16_t
    colorpulsado;int radio;};
4
5   void botons(boton p) {
6     uint16_t color;
7     tft.setTextColor(p.colortxt);
8     tft.setTextSize(p.size);
9     if (p.pulsado == 1) {
10      color = p.colorpulsado;
11      tft.fillRect(p.x, p.y, p.ancho * p.size, p.alto * p.size, color);
12      tft.setCursor(p.x + 2 * p.size, p.y + 1 * p.size);  //5,3
13      tft.println(p.simbolo);
14      delay(espera);
15    }
16    color = p.color;
17    tft.fillRect(p.x, p.y, p.ancho * p.size, p.alto * p.size, color);
18    tft.setCursor(p.x + 2 * p.size, p.y + 1 * p.size);  //5,3
19    tft.println(p.simbolo);
20  }
21
22  void boton_play(boton p) {
23    uint16_t color;
```

```
24    if (p.pulsado == 1) {
25      color = p.colorpulsado;
26      tft.fillCircle(p.x, p.y, p.radio, color);
27      tft.fillTriangle(p.x - 5, p.y - 9, p.x - 5, p.y + 9, p.x + 11, p.y, p.colortxt);
28      delay(espera);
29    }
30    color = p.color;
31    tft.fillCircle(p.x, p.y, p.radio, color);
32    tft.fillTriangle(p.x - 5, p.y - 9, p.x - 5, p.y + 9, p.x + 11, p.y, p.colortxt);
33  }
34
35  void boton_stop(boton p) {
36    uint16_t color;
37    if (p.pulsado == 1) {
38      color = p.colorpulsado;
39      tft.fillCircle(p.x, p.y, p.radio, color);
40      tft.fillRect(p.x - 8, p.y - 8, p.ancho, p.alto, p.colortxt);
41      //pararMotor();
42      delay(espera);
43    }
44    color = p.color;
45    tft.fillCircle(p.x, p.y, p.radio, color);
46    tft.fillRect(p.x - 8, p.y - 8, p.ancho, p.alto, p.colortxt);
47  }
```

Código 11 Botones.cpp

```
1   #ifndef _TACTIL_H_
2   #define _TACTIL_H_
3
4   #include <Arduino.h>
5   #include <Elegoo_TFTLCD.h>
6   #include <TouchScreen.h>
7   #include "parametros.h"
8   #include "barra_deslizador.h"
9   #include "textos.h"
10  #include "botones.h"
11  #include "funciones.h"
12  #include "pantallas.h"
13
14  extern Elegoo_TFTLCD tft;
15  extern seccion p11;
16  extern seccion p21;
17  extern seccion p31;
18  extern seccion resultado;
19  extern int x;
20  extern int y;
21  extern int ykp;
22  extern int ykd;
23  extern int yki;
24
25  void tactil(TSPoint, seccion*, int*, encoder*);
26  void tactilmas(TSPoint, seccion*);
27  void tactilmenos(TSPoint, seccion*);
28  void tactildeslizador(TSPoint, seccion*);
29  void tactiliniciar(TSPoint, seccion*, int*);
30  void tactilflechai(TSPoint, seccion*);
31  void tactilflechad(TSPoint, seccion*);
32  void tactilmasp(TSPoint, seccion*);
33  void tactilmenosp(TSPoint, seccion*);
34
35  #endif
```

Código 12 Tactil.h

```
#include "tactil.h"

void tactil(TSPoint p, seccion *pp, int *m, encoder *enc) {
  if (*m == 0) {
    if (p.y < ykp) {
      tactilmas(p, pp);
      tactilmenos(p, pp);
      tactildeslizador(p, pp);
      tactiliniciar(p, pp, m);
      enc->posicion = round(doublemap(pp->valor.valor, pp->valor.min, pp->valor.max, 0, enc->reductor * enc->resolucion));
      enc->valorbuscado = pp->valor.valor;
    }
  } else {
    tactilflechai(p, pp);
    tactilflechad(p, pp);
    tactilmasp(p, pp);
    tactilmenosp(p, pp);
    tactilmas(p, pp);
    tactilmenos(p, pp);
    tactildeslizador(p, pp);
    tactiliniciar(p, pp, m);
    if (p.y < ykp) {
      enc->valorbuscado = pp->valor.valor;
      enc->ierror = 0;
    } else if (p.y < yki) enc->kp = pp->valor.valor;
    else if (p.y < ykd) enc->ki = pp->valor.valor;
    else enc->kd = pp->valor.valor;
  }
}
//deslizador(valores,deslizar);
void tactildeslizador(TSPoint p, seccion *pp) {
  if (p.x >= pp->des.x && p.x <= pp->des.x + pp->des.anchura - 5 && p.y >= pp->des.y && p.y <= pp->des.y + pp->des.altura) {
    pp->valor.x0 = pp->valor.x;
    pp->valor.x = p.x;
    pp->valor.valor = redondear(doublemap(pp->valor.x, pp->des.x, pp->des.x + pp->des.anchura - 5, pp->valor.min, pp->valor.max), decimal - pp->escal.ns);
    deslizador(pp->valor, pp->des);
    if (p.y < ykp) txtvalor(double(pp->valor.valor), 0, pp->valortxt, 0);
    else {
      if (pp->valor.valor < 10) txtvalor(double(pp->valor.valor), decimal, pp->valortxt, 1);
      else if (pp->valor.valor < 100) txtvalor(double(pp->valor.valor), decimal - 1, pp->valortxt, 1);
      else if (pp->valor.valor < 1000) txtvalor(double(pp->valor.valor), decimal - 2, pp->valortxt, 1);
      else if (pp->valor.valor < 10000) txtvalor(double(pp->valor.valor), decimal - 3, pp->valortxt, 1);
    }
  }
}
void tactiliniciar(TSPoint p, seccion *pp, int *m) {
  if (p.x >= pp->iniciar.x - pp->iniciar.radio && p.x <= pp->iniciar.x + pp->iniciar.radio && p.y >= pp->iniciar.y - pp->iniciar.radio && p.y <= pp->iniciar.y + pp->iniciar.radio) {
    if (*m == 1) {
      pp->iniciar.pulsado = 1;
      boton_stop(pp->iniciar);
      pantallaresultado();
      *m = 2;
    } else {
      pp->iniciar.pulsado = 1;
      boton_play(pp->iniciar);
      if (*m == 0) {
        pantallaangulo();
        pantallaconstantes();
        *m = 1;
      } else if (*m == 2) {
```

```
61          pantallaangulo();
62          *m = 1;
63        }
64      }
65    }
66  }
67  void tactilmas(TSPoint p, seccion *pp) {
68    if (p.x >= pp->mas.x && p.x <= pp->mas.x + pp->mas.ancho * pp->mas.size && p.y >= pp->mas.y && p.y <= pp->mas.y + pp->mas.alto * pp->mas.size) {
69      pp->mas.pulsado = 1;
70      botons(pp->mas);
71      if (pp->valor.valor < pp->valor.max) {
72        if (p.y < ykp) {
73          pp->valor.valor += 1;
74          txtvalor(double(pp->valor.valor), 0, pp->valortxt, 0);
75        } else {
76          if (pp->valor.valor + pp->escal.s[pp->escal.ns] <= pp->valor.max) {
77            pp->valor.valor += pp->escal.s[pp->escal.ns];
78            if (pp->valor.valor < 10) txtvalor(double(pp->valor.valor), decimal, pp->valortxt, 1);
79            else if (pp->valor.valor < 100) txtvalor(double(pp->valor.valor), decimal - 1, pp->valortxt, 1);
80            else if (pp->valor.valor < 1000) txtvalor(double(pp->valor.valor), decimal - 2, pp->valortxt, 1);
81            else if (pp->valor.valor < 10000) txtvalor(double(pp->valor.valor), decimal - 3, pp->valortxt, 1);
82          }
83        }
84        pp->valor.x0 = pp->valor.x;
85        pp->valor.x = doublemap(pp->valor.valor, pp->valor.min, pp->valor.max, pp->des.x, pp->des.x + pp->des.anchura - 5);
86        deslizador(pp->valor, pp->des);
87      }
88    }
89  }
90  void tactilmenos(TSPoint p, seccion *pp) {
91    if (p.x >= pp->menos.x && p.x <= pp->menos.x + pp->menos.ancho * pp->menos.size && p.y >= pp->menos.y && p.y <= pp->menos.y + pp->menos.alto * pp->menos.size) {
92      pp->menos.pulsado = 1;
93      botons(pp->menos);
94      if (pp->valor.valor > pp->valor.min) {
95        if (p.y < ykp) {
96          pp->valor.valor -= 1;
97          txtvalor(double(pp->valor.valor), 0, pp->valortxt, 0);
98        } else {
99          if (redondear(pp->valor.valor - pp->escal.s[pp->escal.ns], decimal) >= pp->valor.min) {
100           pp->valor.valor = redondear(pp->valor.valor - pp->escal.s[pp->escal.ns], decimal);
101           if (pp->valor.valor < 10) txtvalor(double(abs(pp->valor.valor)), decimal, pp->valortxt, 1);
102           else if (pp->valor.valor < 100) txtvalor(double(pp->valor.valor), decimal - 1, pp->valortxt, 1);
103           else if (pp->valor.valor < 1000) txtvalor(double(pp->valor.valor), decimal - 2, pp->valortxt, 1);
104           else if (pp->valor.valor < 10000) txtvalor(double(pp->valor.valor), decimal - 3, pp->valortxt, 1);
105         }
106       }
107       pp->valor.x0 = pp->valor.x;
108       pp->valor.x = doublemap(pp->valor.valor, pp->valor.min, pp->valor.max, pp->des.x, pp->des.x + pp->des.anchura - 5);
109       deslizador(pp->valor, pp->des);
110     }
111   }
112 }
113 void tactilflechai(TSPoint p, seccion *pp) {
114   if (p.x >= pp->flechai.x && p.x <= pp->flechai.x + pp->flechai.ancho * pp->flechai.size && p.y >= pp->flechai.y && p.y <= pp->flechai.y + pp->flechai.alto * pp->flechai.size) {
115     pp->flechai.pulsado = 1;
116     botons(pp->flechai);
117     if (pp->escal.ns < decimal) {
118       pp->escal.ns += 1;
119       pp->valor.valor = redondear(pp->valor.valor, decimal - pp->escal.ns);
```

```
      txtvalor(pp->escal.s[pp->escal.ns], decimal, pp->escalatxts, 1);
      if (pp->valor.valor < 10) txtvalor(double(pp->valor.valor), decimal, pp->valortxt, 1);
      else if (pp->valor.valor < 100) txtvalor(double(pp->valor.valor), decimal - 1, pp->valortxt, 1);
      else if (pp->valor.valor < 1000) txtvalor(double(pp->valor.valor), decimal - 2, pp->valortxt, 1);
      else if (pp->valor.valor < 10000) txtvalor(double(pp->valor.valor), decimal - 3, pp->valortxt, 1);
    }
  }
}
void tactilflechad(TSPoint p, seccion *pp) {
  if (p.x >= pp->flechad.x && p.x <= pp->flechad.x + pp->flechad.ancho * pp->flechad.size && p.y >= pp->flechad.y
&& p.y <= pp->flechad.y + pp->flechad.alto * pp->flechad.size) {
    pp->flechad.pulsado = 1;
    botons(pp->flechad);
    if (pp->escal.ns > 0) {
      pp->escal.ns -= 1;
      txtvalor(pp->escal.s[pp->escal.ns], decimal, pp->escalatxts, 1);
    }
  }
}
void tactilmasp(TSPoint p, seccion *pp) {
  if (p.x >= pp->masp.x && p.x <= pp->masp.x + pp->masp.ancho * pp->masp.size && p.y >= pp->masp.y && p.y <=
pp->masp.y + pp->masp.alto * pp->masp.size) {
    pp->masp.pulsado = 1;
    botons(pp->masp);
    if (pp->escal.nsd < 3) {
      pp->escal.nsd += 1;
      txtvalor(pp->escal.sd[pp->escal.nsd], 0, pp->escaladestxt2sd, 3);
    }
    pp->valor.max = pp->escal.sd[pp->escal.nsd];
    pp->valor.x0 = pp->valor.x;
    pp->valor.x = doublemap(pp->valor.valor, pp->valor.min, pp->valor.max, pp->des.x, pp->des.x + pp->des.anchura -
5);
    deslizador(pp->valor, pp->des);
  }
}
void tactilmenosp(TSPoint p, seccion *pp) {
  if (p.x >= pp->menosp.x && p.x <= pp->menosp.x + pp->menosp.ancho * pp->menosp.size && p.y >= pp->menosp.y
&& p.y <= pp->menosp.y + pp->menosp.alto * pp->menosp.size) {
    pp->menosp.pulsado = 1;
    botons(pp->menosp);
    if (pp->escal.nsd > 0) {
      pp->escal.nsd -= 1;
      txtvalor(pp->escal.sd[pp->escal.nsd], 0, pp->escaladestxt2sd, 3);
    }
    pp->valor.max = pp->escal.sd[pp->escal.nsd];
    if (pp->valor.valor > pp->valor.max) {
      pp->valor.valor = pp->valor.max;
      if (pp->valor.valor < 10) txtvalor(double(abs(pp->valor.valor)), 3, pp->valortxt, 1);
      else if (pp->valor.valor < 100) txtvalor(double(pp->valor.valor), 2, pp->valortxt, 1);
      else if (pp->valor.valor < 1000) txtvalor(double(pp->valor.valor), 1, pp->valortxt, 1);
    }
    pp->valor.x0 = pp->valor.x;
    pp->valor.x = doublemap(pp->valor.valor, pp->valor.min, pp->valor.max, pp->des.x, pp->des.x + pp->des.anchura -
5);
    deslizador(pp->valor, pp->des);
  }
}
```

Código 13 Tactil.cpp

```
#ifndef _FUNCIONES_H_
#define _FUNCIONES_H_

#include <Arduino.h>

```

```
6   double redondear(double, int);
7   double doublemap(double, double, double, double, double);
8
9   #endif
```

Código 14 Funciones.h

```
1   #include "funciones.h"
2
3   double redondear(double v, int decimales) {
4     if (decimales == 4) decimales = 10000;
5     else if (decimales == 3) decimales = 1000;
6     else if (decimales == 2) decimales = 100;
7     else if (decimales == 1) decimales = 10;
8     else if (decimales == 0) decimales = 1;
9     v = v * (decimales);
10    v = round(v);
11
12    return v / decimales;
13  }
14
15  double doublemap(double valor, double in_min, double in_max, double out_min, double out_max) {
16    return (valor - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
17  }
```

Código 15 Funciones.cpp

Al escribir el código con librerías, se consigue tenerlo más ordenado y poder reutilizarlo para otros proyectos.

Una vez se tienen creadas las librerías, hay que añadirlas y para ello hay dos formas:

- Añadiendo las librerías en la propia carpeta del programa.
- Instalando las librerías en el IDE de Arduino mediante un archivo .zip.

En este caso, las librerías para controlar la pantalla las proporciona el fabricante [31][32], se instalan con un archivo .zip y el resto se crean directamente en la carpeta trabajada. Una vez se tienen las librerías preparadas hay que declararlas al inicio del programa como se puede ver en el Código 16.

```
1   #include <Elegoo_GFX.h>     // Core graphics library
2   #include <Elegoo_TFTLCD.h>  // Hardware-specific library
3   #include <TouchScreen.h>
4   #include <digitalWriteFast.h>
5   #include "parametros.h"
6   #include "pantallas.h"
7   #include "tactil.h"
8   #include "controlador.h"
```

Código 16 Declaración de librerías

A continuación, y por último, declaramos las diferentes variables y funciones del sistema.

```
11  TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);
12  Elegoo_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
13
14  //struct linea1{titulo; valtxt; val; comentario1; comentario2; mas; menos; iniciar; des;};
15  int x = 5;
16  int y = 5;
17  int ykp = 95;
```

```
int yki = 170;
int ykd = 245;

int modo = 0;

seccion *toquepantalla;
```

Código 17 Declaración de variables globales

```
// struct seccion{titulo;valortxt;valor;escalatxt;escala;flechai;flechad;mas;menos;masp;menosp;escaladestxt1;escaladestxt2;des;};
seccion p11 = {
  { x + 5, y + 5, "CALIBRAR ", 3, blanco },                    //titulo
  { x + 97, y + 34, "n/a", 2, blanco },                        //valtxt
  { 0, 360, x + 10, x + 10, 0 },                               //val (struct valores{min; max; x; x0; valor;}; // Todo double )
  {},                                                          //escalatxt
  {},                                                          //escala (struct escala{int[4] s;int ns;double[4] sd;int nsd;};)
  {},                                                          //flechai (boton)
  {},                                                          //flechad (boton)
  { x + 150, y + 32, 2, 9, 9, 0, "+", blanco, rojosuave, rojo, 0 },    //mas (boton)
  { x + 60, y + 32, 2, 9, 9, 0, "-", blanco, rojosuave, rojo, 0 },     //menos (boton)
  {},                                                          //masp (boton)
  {},                                                          //menosp (boton)
  {},                                                          //escaladestxt1
  {},                                                          //escaladestxt2
  { x + 10, y + 60, 20, 210, gris, rojo },                     //deslizador (struct deslizar{x; y; altura; anchura; minimo; maximo;colorbarra,color};)
  {},                                                          //fondo (struct fondo{x; y; altura; anchura;color};)
  { x + 5, y + 90, "Angulo en el que se encuentra actual", 1, blanco },    //comentario1
  { x + 5, y + 100, "mente", 1, blanco },                      //comentario2
  { x + 215, y + 15, 2, 16, 16, 0, "stop", blanco, rojosuave, rojo, 15 },  //iniciar (boton)
};                                                             //fondo (struct fondo{x; y; altura; anchura;color};)
```

Código 18 Pantalla inicial

```
seccion p21 = {
  { x + 5, y + 5, "ANGULO   ", 3, blanco },                    //titulo
  { x + 97, y + 34, "n/a", 2, blanco },                        //valtxt
  { 0, 360, x + 10, x + 10, 0 },                               //val (struct valores{min; max; x; x0; valor;}; // Todo double )
  {},                                                          //escalatxt
  {},                                                          //escala (struct escala{int[4] s;int ns;double[4] sd;int nsd;};)
  {},                                                          //flechai (boton)
  {},                                                          //flechad (boton)
  { x + 150, y + 32, 2, 9, 9, 0, "+", blanco, rojosuave, rojo, 0 },    //mas (boton)
  { x + 60, y + 32, 2, 9, 9, 0, "-", blanco, rojosuave, rojo, 0 },     //menos (boton)
  {},                                                          //masp (boton)
  {},                                                          //menosp (boton)
  {},                                                          //escaladestxt1
  {},                                                          //escaladestxt2
  { x + 10, y + 60, 20, 210, gris, rojo },                     //deslizador (struct deslizar{x; y; altura; anchura; minimo; maximo;colorbarra,color};)
  {},                                                          //fondo (struct fondo{x; y; altura; anchura;color};)
  {},                                                          //comentario1
  {},                                                          //comentario2
  { x + 215, y + 15, 2, 16, 16, 0, "play", blanco, rojosuave, rojo, 15 },  //iniciar (boton)
};                                                             //fondo (struct fondo{x; y; altura; anchura;color};)
```

Código 19 Pantalla ángulo

```
seccion p31 = {
  { x + 5, y + 5, "RESULTADO", 3, blanco },                    //titulo
  { x + 97, y + 34, "n/a", 2, blanco },                        //valtxt
```

```
68   { 0, 360, x + 10, x + 10, 0 },                    //val (struct valores{min; max; x; x0; valor;}; // Todo double )
69   {},                                               //escalatxt
70   {},                                               //escala (struct escala{int[4] s;int ns;double[4] sd;int nsd;};)
71   {},                                               //flechai (boton)
72   {},                                               //flechad (boton)
73   { x + 150, y + 32, 2, 9, 9, 0, "+", blanco, rojosuave, rojo, 0 },       //mas (boton)
74   { x + 60, y + 32, 2, 9, 9, 0, "-", blanco, rojosuave, rojo, 0 },        //menos (boton)
75   {},                                               //masp (boton)
76   {},                                               //menosp (boton)
77   {},                                               //escaladestxt1
78   {},                                               //escaladestxt2
79   { x + 10, y + 60, 20, 210, gris, rojo },          //deslizador (struct deslizar{x; y; altura; anchura; minimo;
     maximo;colorbarra,color};)
80   {},                                               //fondo (struct fondo{x; y; altura; anchura;color};)
81   {},                                               //comentario1
82   {},                                               //comentario2
83   { x + 215, y + 15, 2, 16, 16, 0, "stop", blanco, rojosuave, rojo, 15 },  //iniciar (boton)
84 };                                                   //fondo (struct fondo{x; y; altura; anchura;color};)
```

Código 20 Pantalla resultado

```
86 seccion kp = {
87   { x + 5, ykp + 5, "kp", 2, negro },               //titulo
88   { x + 84, ykp + 5, "n/a", 2, negro },             //valtxt
89   { 0, 5, x + 40, x + 40, 0 },                      //val (struct valores{min; max; x; x0; valor;}; // Todo double )
90   { x + 190, ykp + 5, "n/a", 1, negro },            //escalatxt
91   { { 0.0001, 0.001, 0.01, 0.1, 1 }, 0, { 10, 25, 50, 100 }, 0 },   //escala (struct escala{int[5] s;int ns;double[4] sd;int
     nsd;};)
92   { x + 190, ykp + 15, 1, 10, 10, 0, "<", blanco, rojosuave, rojo, 0 },  //flechai (boton)
93   { x + 215, ykp + 15, 1, 10, 10, 0, ">", blanco, rojosuave, rojo, 0 },  //flechad (boton)
94   { x + 205, ykp + 31, 2, 9, 9, 0, "+", blanco, rojosuave, rojo, 0 },    //mas (boton)
95   { x + 5, ykp + 31, 2, 9, 9, 0, "-", blanco, rojosuave, rojo, 0 },      //menos (boton)
96   { x + 200, ykp + 56, 1, 9, 9, 0, "+", blanco, rojosuave, rojo, 0 },    //masp (boton)
97   { x + 170, ykp + 56, 1, 9, 9, 0, "-", blanco, rojosuave, rojo, 0 },    //menosp (boton)
98   { x + 38, ykp + 56, "n/a", 1, negro },            //escaladestxt1
99   { x + 180, ykp + 56, "n/a", 1, negro },           //escaladestxt2
100  { x + 40, ykp + 30, 20, 150, gris, rojo },        //deslizador (struct deslizar{x; y; altura; anchura; minimo;
     maximo;colorbarra,color};)
101  { x, ykp, 70, 230, blanco }
102 };  //fondo (struct fondo{x; y; altura; anchura;color};)
```

Código 21 Definición constante K_p

```
103 seccion kd = {
104   { x + 5, ykd + 5, "kd", 2, negro },               //titulo
105   { x + 84, ykd + 5, "n/a", 2, negro },             //valtxt
106   { 0, 1, x + 40, x + 40, 0 },                      //val (struct valores{min; max; x; x0; valor;}; // Todo double )
107   { x + 190, ykd + 5, "n/a", 1, negro },            //escalatxt
108   { { 0.0001, 0.001, 0.01, 0.1, 1 }, 0, { 1, 2, 5, 10 }, 0 },   //escala (struct escala{int[4] s;int ns;double[4] sd;int nsd;};)
109   { x + 190, ykd + 15, 1, 10, 10, 0, "<", blanco, rojosuave, rojo, 0 },  //flechai (boton)
110   { x + 215, ykd + 15, 1, 10, 10, 0, ">", blanco, rojosuave, rojo, 0 },  //flechad (boton)
111   { x + 205, ykd + 31, 2, 9, 9, 0, "+", blanco, rojosuave, rojo, 0 },    //mas (boton)
112   { x + 5, ykd + 31, 2, 9, 9, 0, "-", blanco, rojosuave, rojo, 0 },      //menos (boton)
113   { x + 200, ykd + 56, 1, 9, 9, 0, "+", blanco, rojosuave, rojo, 0 },    //masp (boton)
114   { x + 170, ykd + 56, 1, 9, 9, 0, "-", blanco, rojosuave, rojo, 0 },    //menosp (boton)
115   { x + 38, ykd + 56, "n/a", 1, negro },            //escaladestxt1
116   { x + 180, ykd + 56, "n/a", 1, negro },           //escaladestxt2
117   { x + 40, ykd + 30, 20, 150, gris, rojo },        //deslizador (struct deslizar{x; y; altura; anchura; minimo;
     maximo;colorbarra,color};)
118   { x, ykd, 70, 230, blanco }
119 };  //fondo (struct fondo{x; y; altura; anchura;color};)
```

Código 22 Definición constante K_d

```
120 seccion ki = {
121   { x + 5, yki + 5, "ki", 2, negro },               //titulo
```

```
  { x + 90, yki + 5, "n/a", 2, negro },                    //valtxt
  { 0, 1, x + 40, x + 40, 0 },                             //val (struct valores{min; max; x; x0; valor;}; // Todo double )
  { x + 190, yki + 5, "n/a", 1, negro },                   //escalatxt
  { { 0.0001, 0.001, 0.01, 0.1, 1 }, 0, { 1, 2, 5, 10 }, 0 },     //escala (struct escala{int[4] s;int ns;double[4] sd;int nsd;};)
  { x + 190, yki + 15, 1, 10, 10, 0, "<", blanco, rojosuave, rojo, 0 },  //flechai (boton)
  { x + 215, yki + 15, 1, 10, 10, 0, ">", blanco, rojosuave, rojo, 0 },  //flechad (boton)
  { x + 205, yki + 31, 2, 9, 9, 0, "+", blanco, rojosuave, rojo, 0 },    //mas (boton)
  { x + 5, yki + 31, 2, 9, 9, 0, "-", blanco, rojosuave, rojo, 0 },      //menos (boton)
  { x + 200, yki + 56, 1, 9, 9, 0, "+", blanco, rojosuave, rojo, 0 },    //masp (boton)
  { x + 170, yki + 56, 1, 9, 9, 0, "-", blanco, rojosuave, rojo, 0 },    //menosp (boton)
  { x + 38, yki + 56, "n/a", 1, negro },                   //escaladestxt1
  { x + 180, yki + 56, "n/a", 1, negro },                  //escaladestxt2
  { x + 40, yki + 30, 20, 150, gris, rojo },               //deslizador (struct deslizar{x; y; altura; anchura; minimo;
maximo;colorbarra,color};)
  { x, yki, 70, 230, blanco }
};  //fondo (struct fondo{x; y; altura; anchura;color};)
```

Código 23 Definición constante K_i

```
seccion resultado = {
  {},                          //titulo
  {},                          //valtxt
  { 0, 360, 0, 0, 0 },              //val (struct valores{min; max; x; x0; valor;}; // Todo double )
  {},                          //escalatxt
  {},                          //escala (struct escala{int[4] s;int ns;double[4] sd;int nsd;};)
  {},                          //flechai (boton)
  {},                          //flechad (boton)
  {},                          //mas (boton)
  {},                          //menos (boton)
  {},                          //masp (boton)
  {},                          //menosp (boton)
  {},                          //escaladestxt1
  {},                          //escaladestxt2
  { x + 10, y + 80, 5, 210, negro, rojo },  //deslizador (struct deslizar{x; y; altura; anchura;colorbarra,color};)
  {}
};  //fondo (struct fondo{x; y; altura; anchura;color};)
```

Código 24 Declaración de la variable resultado

```
    //----------------------------ENCODER---------------------------------------
int pinA = 21;
int pinB = 20;
//struct encoder{unsigned long t;unsigned long t0;double reductor;double resolucion;double error;double error0;double
derror;double ierror;int valor;int valor0;}
encoder enc;
int d = -1;  //variable para saber la direccion que tiene el motor
int posEncoder = 0;
int pinAnt;
int aVal;
```

Código 25 Declaración de variables del encoder

```
//-------------------------MOTOR-----------------------------------------
int direccion = 13;  //LOW = ANTIHORARIO || HIGH = HORARIO
int enA = 11;
int ppx = 5;
int ppy = 240;
```

Código 26 Declaración de variables del motor

```
//---------------------Variable nuevas---------------------------------
String caracterRecibido = "";
bool lineaCompletaRecibida = false;
double st = 0;
unsigned long t0;
int a = enc.valor;
```

```
bool pcModeActive = false;  // Puedes establecer el valor inicial que prefieras.
                            // false: Inicia en modo táctil.
                            // true: Inicia en modo PC.
```

Código 27 Nuevas variables

```
void doEncodeA() {
  (digitalReadFast(pinA) == digitalReadFast(pinB)) ? enc.posicion++ : enc.posicion--;
  //Serial.println(enc.posicion);
}
```

Código 28 Función encoderA

```
void doEncodeB() {
  (digitalReadFast(pinA) != digitalReadFast(pinB)) ? enc.posicion++ : enc.posicion--;
  //Serial.println(enc.posicion);
}
```

Código 29 Función encoderB

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println(F("LISTO"));
  tft.reset();
  uint16_t identifier = 0x9341;  //0x9341 LCD driver
  tft.begin(identifier);
  tft.setRotation(2);
  tft.fillScreen(negro);
  pantallacalibracion();
  //----------------------Encoder---------------------------------
  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
  pinAnt = digitalRead(pinA);
  volatile long timeCounter = 0;
  const int timeThreshold = 5;
  attachInterrupt(digitalPinToInterrupt(pinA), doEncodeA, CHANGE);
  attachInterrupt(digitalPinToInterrupt(pinB), doEncodeB, CHANGE);
  //-------------------------MOTOR-----------------------------------
  pinMode(direccion, OUTPUT);
  pinMode(enA, OUTPUT);
  enc.reductor = 9.277;
  enc.resolucion = 22 * 2;
  //---------------------------------------------------------------
}
```

Código 30 Función Setup

```
void loop() {

  // ------------------- Lectura de comandos por Serial (siempre activa) -------------------
  if (Serial.available() > 0) {
    char nuevoCaracter = Serial.read();
    if (nuevoCaracter == '\n' || nuevoCaracter == '\r') {
      if (caracterRecibido.length() > 0) {  // Solo procesar si hemos recibido algo
        lineaCompletaRecibida = true;
      }
    } else {
      if (nuevoCaracter != '\r') {
        caracterRecibido += nuevoCaracter;  // Acumular el carácter
      }
    }
  }
  // ------------------- Procesamiento de línea completa recibida por Serial -------------------
  if (lineaCompletaRecibida) {
```

```
222     caracterRecibido.trim();  // Eliminar espacios en blanco al inicio/final
223     if (caracterRecibido.equalsIgnoreCase("INICIO")) pcModeActive = true;
224     else if (caracterRecibido.equalsIgnoreCase("FIN")) pcModeActive = false;
225     else {
226       if (pcModeActive) {
227         double numero = caracterRecibido.toDouble();
228         enc.PID = numero;
229         //enc.valor = round(doublemap(enc.posicion, 0, (double)enc.reductor * enc.resolucion, 0, 360));
230         Serial.println(enc.posicion);
231       }
232     }
233     // Limpiar para el próximo comando, independientemente de si fue procesado o no
234     caracterRecibido = "";
235     lineaCompletaRecibida = false;
236   }
237   // ------------------- Ejecución según el modo activo -------------------
238   if (pcModeActive) {
239     girarmotor(&enc, direccion, enA, d);
240   } else {
241     TSPoint p = ts.getPoint();
242     pinMode(XM, OUTPUT);
243     pinMode(YP, OUTPUT);
244     if (modo == 2) {
245       if (enc.t0 == 0) enc.t0 = millis();
246       pid(&enc, direccion, enA, &d, resultado);
247     } else if (modo == 1) {
248       pararMotor(enA);
249       enc.t0 = 0;
250     }
251     if (p.z > MINPRESSURE && p.z < MAXPRESSURE) {
252       p.x = map(p.x, TS_MINX, TS_MAXX, tft.width(), 0);
253       p.y = (tft.height() - map(p.y, TS_MINY, TS_MAXY, tft.height(), 0));
254       if (p.y < ykp) toquepantalla = &p11;
255       else if (p.y < yki) toquepantalla = &kp;
256       else if (p.y < ykd) toquepantalla = &ki;
257       else toquepantalla = &kd;
258
259       tactil(p, toquepantalla, &modo, &enc);
260     }
261   }
262   // Fin de loop()
263   //-----------------------------------------------------------------
264 }
```

Código 31 Función loop

9.2 PYTHON

El código está estructurado en cinco partes, una en la que se declaran librerías (Código 32) y cuatro clases, la primera es donde se han declarado las variables que va a tener el controlador y los cálculos necesarios (Código 33), otra que es la encargada de establecer la comunicación con Arduino (Código 34) y el diálogo de conexión, otro “hilo” separado (QThread) encargado de manejar la comunicación serie con Arduino y la lógica que va a llamar y actualizar los diferentes parámetros del controlador (Código 35) y una última que es la que maneja la interfaz de usuario y los eventos que ocurren en ella (Código 36). Por último, irár la función encargada de hacer funcionar el programa (Código 37)

Se ha optado por la utilización de slots e hilos para mantener la interfaz responsiva en todo momento y que no haya bloqueos.

```python
import sys
from PyQt5 import  QtCore, uic
from PyQt5.QtWidgets import (QMainWindow, QApplication, QLabel, QVBoxLayout,
                             QFileDialog, QMessageBox, QDialog,
                             QDialogButtonBox, QProgressBar)
from PyQt5.QtCore import QThread, QObject, pyqtSignal, pyqtSlot, QTimer, QMetaObject, Qt
import serial, serial.tools.list_ports
import time
import pyqtgraph as pg
import scipy.io
import pandas as pd
```

Código 32 Importaciones (Python)

```python
VMAX = 255.0
reductor=9.277
resolucion=22*2

def saturado(x, xmin, xmax):
    if x < xmin:
        return xmin
    elif x > xmax:
        return xmax
    else:
        return x

class PID():
    def __init__(self):
        self.e = 0.0
        self.ie = 0.0
        self.de = 0.0
        self.pid = 0.0
        self.last_time = time.perf_counter()

    def update(self, u, y, kp, ki, kd):
        current_time = time.perf_counter()
        delta_time = current_time - self.last_time
        self.last_time = current_time
        # Calculo de errores
        e = u - y # Error actual
        if abs(self.pid) <= 255 : self.ie += (self.e + e) * 0.5 * delta_time # Error integral
        self.de = (e - self.e) / delta_time # Error derivativo
        # Calculo del PID
        self.pid = e * kp + ki * self.ie + kd * self.de
        self.pid = saturado(self.pid, -VMAX, VMAX)
        # Guardar el error anterior antes de actualizarlo
        self.e = e
        return delta_time
```

Código 33 Parámetros y Cálculo PID (Python)

```python
class RetryConnectionDialog(QDialog):
    retry_button_clicked = pyqtSignal() # Señal emitida cuando el usuario pulsa "Reintentar"
    cancel_attempt_and_close_app_clicked = pyqtSignal() # Para cerrar

    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Estado de Conexión")
        self.setModal(True)

```

```
57          self.setWindowFlags(self.windowFlags() & ~Qt.WindowCloseButtonHint | Qt.Dialog)
58
59          self.layout = QVBoxLayout(self)
60
61          self.title_label = QLabel("<b>Título del Estado</b>")
62          self.title_label.setAlignment(Qt.AlignCenter)
63          font = self.title_label.font()
64          font.setPointSize(14); self.title_label.setFont(font)
65          self.layout.addWidget(self.title_label)
66
67          self.message_label = QLabel("Mensaje general.")
68          self.layout.addWidget(self.message_label)
69
70          self.current_port_label = QLabel("Puerto actual: N/A")
71          self.current_port_label.setAlignment(Qt.AlignCenter)
72          self.layout.addWidget(self.current_port_label)
73
74          self.suggestions_label = QLabel("Sugerencias...")
75          self.suggestions_label.setWordWrap(True)
76          self.layout.addWidget(self.suggestions_label)
77
78          self.progress_bar = QProgressBar(self)
79          self.progress_bar.setRange(0, 0)
80          self.progress_bar.setTextVisible(False)
81          self.layout.addWidget(self.progress_bar)
82
83          self.status_label = QLabel("Estado detallado...")
84          self.status_label.setAlignment(Qt.AlignCenter)
85          self.layout.addWidget(self.status_label)
86
87          self.button_box = QDialogButtonBox(self)
88          self.action_button = self.button_box.addButton("Acción", QDialogButtonBox.AcceptRole)
89          self.close_app_button = self.button_box.addButton("Cerrar Aplicación", QDialogButtonBox.RejectRole)
90          self.layout.addWidget(self.button_box)
91
92          self.action_button.clicked.connect(self._on_action_button_clicked)
93          self.close_app_button.clicked.connect(self._on_close_app_button_clicked)
94
95          self._is_attempting_connection = False
96          self._current_mode = "" # "connecting_initial", "connecting_retry", "failed"
97
98      def _configure_ui(self, title, msg, suggestions, port_label_text, status_text,
99                        show_suggestions, show_port_label, show_progress,
100                       action_button_text, action_button_visible, action_button_enabled,
101                       close_button_text="Cerrar Aplicación", close_button_enabled=True):
102         self.title_label.setText(f"<b>{title}</b>")
103         self.message_label.setText(msg)
104         self.message_label.setVisible(bool(msg))
105         self.suggestions_label.setText(suggestions)
106         self.suggestions_label.setVisible(show_suggestions)
107         self.current_port_label.setText(port_label_text)
108         self.current_port_label.setVisible(show_port_label)
109         self.status_label.setText(status_text)
110         self.progress_bar.setVisible(show_progress)
111
112         self.action_button.setText(action_button_text)
113         self.action_button.setVisible(action_button_visible)
114         self.action_button.setEnabled(action_button_enabled)
115
116         self.close_app_button.setText(close_button_text)
117         self.close_app_button.setEnabled(close_button_enabled)
118
119         if not self.isVisible():self.show()
```

```
120
121     def show_connecting_initial(self):
122         self._current_mode = "connecting_initial"
123         self._is_attempting_connection = True
124         self._configure_ui(
125             title="Estableciendo Conexión Inicial", msg="", suggestions="",
126             port_label_text="Puerto: Buscando...", status_text="Buscando dispositivo Arduino, por favor espere...",
127             show_suggestions=False, show_port_label=True, show_progress=True,
128             action_button_text="", action_button_visible=False, action_button_enabled=False,
129             close_button_text="Cancelar y Cerrar"
130         )
131
132     def show_connecting_retry(self):
133         self._current_mode = "connecting_retry"
134         self._is_attempting_connection = True
135         self._configure_ui(
136             title="Reintentando Conexión", msg="", suggestions="",
137             port_label_text="Puerto: Buscando...", status_text="Volviendo a buscar dispositivo, por favor espere...",
138             show_suggestions=False, show_port_label=True, show_progress=True,
139             action_button_text="", action_button_visible=False, action_button_enabled=False,
140             close_button_text="Cancelar y Cerrar"
141         )
142
143     def show_connection_failed(self, is_after_retry=False):
144         self._current_mode = "failed"
145         self._is_attempting_connection = False
146         title = "Reintento Fallido" if is_after_retry else "No se pudo establecer la conexión"
147         msg = "El reintento de conexión también ha fallado." if is_after_retry else \
148               "No se ha podido establecer la conexión con el dispositivo Arduino."
149
150         suggestions_text = (
151             "<b>Por favor, compruebe lo siguiente:</b><br>"
152             "<ul>"
153             "<li>Que el Arduino esté conectado correctamente al PC mediante el cable USB.</li>"
154             "<li>Que el cable USB no esté dañado.</li>"
155             "<li>Que el Arduino esté recibiendo alimentación.</li>"
156             "<li>Que no haya otro programa utilizando el puerto serie del Arduino.</li>"
157             "<li>Que los drivers del puerto serie estén instalados si son necesarios.</li>"
158             "</ul>"
159         )
160         self._configure_ui(
161             title=title, msg=msg, suggestions=suggestions_text,
162             port_label_text="", status_text="Verifique las conexiones y configuraciones.",
163             show_suggestions=True, show_port_label=False, show_progress=False,
164             action_button_text="Reintentar Conexión", action_button_visible=True, action_button_enabled=True,
165             close_button_text="Cerrar Aplicación", close_button_enabled=True
166         )
167
168     @pyqtSlot(str)
169     def update_connecting_port(self, port_name):
170         if self._is_attempting_connection:
171             if port_name == "-":
172                 self.current_port_label.setText("Escaneo de puertos finalizado.")
173             elif port_name:
174                 self.current_port_label.setText(f"Intentando con: {port_name}...")
175             else:
176                 self.current_port_label.setText("Puerto: Buscando...")
177
178     def _on_action_button_clicked(self): # Botón "Reintentar Conexión"
179         if self._current_mode == "failed":
180             self.retry_button_clicked.emit()
181
182     def _on_close_app_button_clicked(self):
```

```
183         self.cancel_attempt_and_close_app_clicked.emit()
184
185     def connection_successful(self):
186         self.accept()
```

Código 34 Class Diálogo de conexión del puerto (Python)

```
188 class Worker(QObject):
189     finished = pyqtSignal()
190     data_received = pyqtSignal(int)
191     pid_value_sent = pyqtSignal(float)
192     serial_connected = pyqtSignal(bool)
193     control_cycle_time = pyqtSignal(float)
194     connection_attempt_failed = pyqtSignal()
195     trying_port_signal = pyqtSignal(str)
196
197     def __init__(self, baud_rate):
198         super().__init__()
199         self.current_serial_port = None
200         self.baud_rate = baud_rate
201         self._running = False
202         self.arduino = None
203         self.pid_controller = PID()
204         self.kp = 0.0
205         self.ki = 0.0
206         self.kd = 0.0
207         self.ts = 0.02
208         self.reference_angle = 0.0
209         self.actual_angle = 0.0
210         self.control_active = False
211         self.is_connected = False
212         self.current_pid_output = 0.0
213
214     def _close_current_port_silently(self):
215         """Para cerrar el puerto serie actual si esta abierto sin enviar FIN"""
216         if self.arduino and self.arduino.is_open:
217             self.arduino.close()
218         self.arduino = None
219
220     @pyqtSlot(str)
221     @pyqtSlot()
222     def connect_serial(self, port_to_try=None):
223         ports_to_attempt = []
224         is_specific_port_attempt = False
225
226         if port_to_try is not None:
227             ports_to_attempt.append(port_to_try)
228             is_specific_port_attempt = True
229         else:
230             available_system_ports = serial.tools.list_ports.comports()
231             if not available_system_ports:
232                 self._close_current_port_silently() # Asegurar que cualquier conexión se cierre
233                 self.is_connected = False
234                 self.serial_connected.emit(False)
235                 self.connection_attempt_failed.emit() # No se encontraron puertos
236                 return
237             for p in available_system_ports:
238                 ports_to_attempt.append(p.device)
239         # Cerrar cualquier conexión existente antes de probar nuevas
240         self._close_current_port_silently()
241         self.is_connected = False # Estado inicial para nuevos intentos de conexión
242
243         for current_port_to_try_iter in ports_to_attempt:
```

```
244             self.trying_port_signal.emit(current_port_to_try_iter) # Emitir el puerto actual
245             try:
246                 self.arduino = serial.Serial(
247                     current_port_to_try_iter,
248                     self.baud_rate,
249                     timeout=1.0,
250                     write_timeout=0.5
251                 )
252 
253                 arduino_ready = False
254                 if self.arduino.is_open:
255                     max_wait_time = 5
256                     start_wait_time = time.perf_counter()
257                     while time.perf_counter() - start_wait_time < max_wait_time:
258                         if self.arduino.in_waiting > 0:
259                             line = self.arduino.readline().decode('utf-8', errors='ignore').rstrip()
260                             if line == "LISTO":
261                                 arduino_ready = True
262                                 break
263                         time.sleep(0.01) # Pequeña pausa
264 
265                 inicio_enviado_exitosamente = False
266                 if arduino_ready:
267                     self.arduino.write("INICIO\n".encode('utf-8'))
268                     inicio_enviado_exitosamente = True
269 
270                 if inicio_enviado_exitosamente:
271                     self.current_serial_port = current_port_to_try_iter # Guardar el puerto exitoso
272                     self.is_connected = True
273                     self.serial_connected.emit(True)
274                     return # Conectado exitosamente, salir del método y del bucle
275                 else:
276                     if self.arduino and self.arduino.is_open:
277                         self.arduino.close()
278                     self.arduino = None # Reiniciar para el siguiente intento o si todos fallan
279 
280             except serial.SerialException as e:
281                 print(f"Worker: ERROR AL ABRIR o durante handshake en puerto serial {current_port_to_try_iter}: {e}")
282                 if self.arduino and self.arduino.is_open:
283                     try: self.arduino.close()
284                     except: pass # Mejor esfuerzo
285                 self.arduino = None
286             except Exception as ex:
287                 print(f"Worker: EXCEPCIÓN INESPERADA en connect_serial ({current_port_to_try_iter}): {ex}")
288                 if self.arduino and self.arduino.is_open:
289                     try: self.arduino.close()
290                     except: pass
291                 self.arduino = None
292 
293         if self._running:# Si el bucle termina (y _running sigue True) sin conexión:
294             self.is_connected = False
295             self.serial_connected.emit(False)
296             if not is_specific_port_attempt:
297                 self.connection_attempt_failed.emit()
298         self.trying_port_signal.emit("-") # Fin del escaneo
299 
300     @pyqtSlot()
301     def disconnect_serial(self):
302         port_abierto_antes = self.arduino and self.arduino.is_open
303 
304         if port_abierto_antes: # Enviar "FIN" ANTES de cerrar el puerto
305             try:
306                 self.arduino.write("FIN\n".encode('utf-8')) # Añadimos \n
```

```
307             time.sleep(0.05) # Pequeña pausa para asegurar el envío
308         except serial.SerialException as write_e:
309             print(f"WORKER: Error al enviar 'FIN' a Arduino: {write_e}")
310         except Exception as e_gen:
311             print(f"WORKER: Error general al enviar 'FIN': {e_gen}")
312         # Ahora cerrar el puerto
313         port_que_se_cierra = self.current_serial_port
314         try:
315             self.arduino.close()
316         except serial.SerialException as e:
317             print(f"WORKER: Error al cerrar puerto {port_que_se_cierra} en disconnect_serial(): {e}")
318     self.arduino = None
319 
320     if self.is_connected: # Solo emitir si el estado cambia de True a False aquí
321         self.is_connected = False
322         try:
323             self.serial_connected.emit(False)
324         except RuntimeError as e:
325             print(f"WORKER: RuntimeError durante el emit de serial_connected(False) en disconnect_serial(): {e}")
326     else:
327         self.is_connected = False # Asegurar que esté en False
328 
329 @pyqtSlot(float, float, float, int, float)
330 def update_pid_parameters(self, kp, ki, kd, reference_angle, new_ts):
331     self.kp = kp
332     self.ki = ki
333     self.kd = kd
334     self.reference_angle = reference_angle
335     self.ts = new_ts
336 
337 @pyqtSlot()
338 def start_pid_control(self):
339     if not self.is_connected:
340         return
341     self.control_active = True
342     self.pid_controller.last_time = time.perf_counter()
343     self.pid_controller.ie = 0.0
344     self.pid_controller.e = 0.0
345 
346 @pyqtSlot()
347 def run(self):
348     self._running = True
349     self.connect_serial()
350 
351     loop_count = 0
352 
353     while self._running:
354         process_events_interval = int(0.1 / self.ts) if self.ts > 0 else 5
355         loop_start_time = time.perf_counter()
356         if self.is_connected and self.arduino and self.arduino.is_open and self.control_active:
357             try:
358                 delta_time = self.pid_controller.update(self.reference_angle, self.actual_angle, self.kp, self.ki, self.kd)
359                 self.current_pid_output = self.pid_controller.pid
360                 pid_output_str = str(self.current_pid_output) + '\n'
361                 pid_output_bytes = pid_output_str.encode('utf-8')
362                 self.arduino.write(pid_output_bytes)
363                 self.pid_value_sent.emit(self.current_pid_output)
364                 self.control_cycle_time.emit(delta_time)
365                 if self.arduino.in_waiting > 0:
366                     v_str = self.arduino.readline().decode('utf-8', errors='ignore').rstrip()
367                     try:
368                         val = int(v_str)
369                         self.actual_angle =round((val - 0) * (360 -0) / (reductor*resolucion-0) + 0)
```

```
370                         self.data_received.emit(self.actual_angle)
371
372                     except ValueError:
373                         if v_str: print(f"Dato no válido o no entero del Arduino: '{v_str}'")
374                 except UnicodeDecodeError: print("Error de decodificación Unicode del Arduino.")
375                 except serial.SerialException as e:
376                     print(f"Error de lectura/escritura del puerto serial durante el control: {e}")
377                     self.disconnect_serial()
378                 elapsed_time_in_loop = time.perf_counter() - loop_start_time # Tiempo gastado en la lógica del if/else
379                 wait_time = self.ts - elapsed_time_in_loop
380
381                 if wait_time > 0:
382                     time.sleep(wait_time)
383             else: # Si no está conectado o control no activo, dormir un poco para no consumir CPU excesivamente
384                 time.sleep(0.01)
385             if loop_count % process_events_interval == 0:
386                 QtCore.QCoreApplication.processEvents()
387             loop_count += 1
388         self.disconnect_serial()
389         self.finished.emit()
390
391     @pyqtSlot()
392     def stop(self):
393         self.control_active = False
394         if self.is_connected and self.arduino and self.arduino.is_open:
395             pid_output_str = "0.0\n"
396             self.arduino.write(pid_output_str.encode('utf-8'))
```

Código 35 Class Worker (Python)

```
398 class main(QMainWindow):
399     COLOR_REFERENCIA_TUPLE = (51, 94, 255)
400     COLOR_RESULTADO_TUPLE = (51, 255, 94)
401     COLOR_PID_TUPLE = (255, 94, 255)
402     COLOR_ERROR_TUPLE = (255, 0, 0)
403
404     # Para hojas de estilo (CSS) - Formato "rgb(R, G, B)"
405     COLOR_REFERENCIA_CSS = "rgb(51, 94, 255)"
406     COLOR_RESULTADO_CSS = "rgb(51, 255, 94)"
407     COLOR_PID_CSS = "rgb(255, 94, 255)"
408     COLOR_ERROR_CSS = "rgb(255, 0, 0)"
409
410     STYLE_TEXT_DEFAULT = ""
411
412     gui_parameters_changed = pyqtSignal(float, float, float, int, float)
413
414     def __init__(self):
415         super().__init__()
416         uic.loadUi('UUII.ui', self)
417
418         self.retry_connection_dialog = None
419
420         self.control_timer = QTimer(self)
421         self.control_timer.timeout.connect(self.update_remaining_time_in_spinbox)
422
423         self.setup_ui_elements()
424         self.setup_worker_thread()
425         self.connect_signals_slots()
426
427         self.scrollBar_start.setEnabled(True)
428         self.duration_of_current_run = self.spinBox_tiempomuestreo.value()
429         self.button_guardar.setToolTip("Guardar señales")
430         self.button_Parar.setToolTip("Parar simulación")
```

```
        self.button_Iniciar.setToolTip("Iniciar simulación")
        self.scrollBar_start.setToolTip("Conectar/desconectar Arduino")

    def setup_ui_elements(self):
        self.spinBox_tiempomuestreo.setEnabled(True)
        self.button_Parar.setEnabled(False)
        self.button_Iniciar.setEnabled(False)
        self.scrollBar_start.setValue(0)
        self.ON.setStyleSheet("color: black;background: gray;")
        self.OFF.setStyleSheet("color: white;background: red;")
        self.control_active = False; self.serial_connected = False
        # Configuración Gráfica
        self.widget_senales.setBackground('w')
        self.widget_senales.setLabel('left', 'Ángulo (grados) / Salida PID')
        self.widget_senales.setLabel('bottom', 't (s)'); self.widget_senales.showGrid(x=True, y=True)
        self.plot_item = self.widget_senales.getPlotItem()
        self.widget_senales.enableAutoRange(axis='y', enable=True)

        self.pen_referencia = pg.mkPen(color=self.COLOR_REFERENCIA_TUPLE, width=2)
        self.pen_resultado = pg.mkPen(color=self.COLOR_RESULTADO_TUPLE, width=2)
        self.pen_pid = pg.mkPen(color=self.COLOR_PID_TUPLE, width=2)
        self.pen_error = pg.mkPen(color=self.COLOR_ERROR_TUPLE, width=2)

        self.reference_curve = self.widget_senales.plot(pen=self.pen_referencia, name="Ref")
        self.result_curve = self.widget_senales.plot(pen=self.pen_resultado, name="Res")
        self.pid_curve = self.widget_senales.plot(pen=self.pen_pid, name="PID")
        self.error_curve = self.widget_senales.plot(pen=self.pen_error, name="Error")

        self.time_data = []; self.reference_angle_data = []; self.result_angle_data = []
        self.pid_output_data = []; self.error_data = []; self.start_time = None
        self.update_plot_visibility()



    def connect_signals_slots(self):
        self.scrollBar_start.valueChanged.connect(self.toggle_serial_connection)
        self.button_Iniciar.clicked.connect(self.start_control_timed)
        self.button_Parar.clicked.connect(self.stop_control)

        self.slider_angulo.valueChanged.connect(self.slider_angulo_cambiado)
        self.spinBox_angulo.valueChanged.connect(self.spinBox_angulo_cambiado)
        self.dial_angulo.valueChanged.connect(self.dial_angulo_cambiado)

        self.slider_kp.valueChanged.connect(self.slider_kp_cambiado)
        self.doubleSpinBox_kp.valueChanged.connect(self.doubleSpinBox_kp_cambiado)
        self.escalakp.valueChanged.connect(self.escalakp_cambio)

        self.slider_ki.valueChanged.connect(self.slider_ki_cambiado)
        self.doubleSpinBox_ki.valueChanged.connect(self.doubleSpinBox_ki_cambiado)
        self.escalaki.valueChanged.connect(self.escalaki_cambio)

        self.slider_kd.valueChanged.connect(self.slider_kd_cambiado)
        self.doubleSpinBox_kd.valueChanged.connect(self.doubleSpinBox_kd_cambiado)
        self.escalakd.valueChanged.connect(self.escalakd_cambio)

        self.checkBox_referencia.stateChanged.connect(self.update_plot_visibility)
        self.checkBox_resultado.stateChanged.connect(self.update_plot_visibility)
        self.checkBox_pid.stateChanged.connect(self.update_plot_visibility)
        self.checkBox_error.stateChanged.connect(self.update_plot_visibility)
        self.button_guardar.clicked.connect(self.button_guardar_clicked)

        self.doubleSpinBox_Ts.valueChanged.connect(self.update_parameters)

```

```
494     def setup_worker_thread(self):
495         self.thread = QThread(self)
496 
497         QTimer.singleShot(0, self.initiate_first_connection_attempt)
498         self.worker = Worker(baud_rate=115200)
499         self.worker.moveToThread(self.thread)
500 
501         self.gui_parameters_changed.connect(self.worker.update_pid_parameters)
502         self.thread.started.connect(self.worker.run)
503 
504         self.worker.finished.connect(self.thread.quit)
505         self.worker.finished.connect(self.worker.deleteLater)
506         self.thread.finished.connect(self.thread.deleteLater)
507 
508         self.worker.data_received.connect(self.update_angle)
509         self.worker.serial_connected.connect(self.update_serial_status)
510         self.worker.connection_attempt_failed.connect(self.handle_connection_failure_from_worker)
511         self.worker.trying_port_signal.connect(self._update_dialog_port_attempt)
512 
513         self.thread.start()
514 
515     @pyqtSlot()
516     def initiate_first_connection_attempt(self):
517         if not self.retry_connection_dialog:
518             self.retry_connection_dialog = RetryConnectionDialog(self)
519             self.retry_connection_dialog.retry_button_clicked.connect(self._trigger_connection_retry_from_dialog)
520             self.retry_connection_dialog.cancel_attempt_and_close_app_clicked.connect(self.close)
521 
522         self.retry_connection_dialog.show_connecting_initial()
523         QMetaObject.invokeMethod(self.worker, "connect_serial", QtCore.Qt.QueuedConnection)
524 
525     @pyqtSlot(str)
526     def _update_dialog_port_attempt(self, port_name):
527         if self.retry_connection_dialog and self.retry_connection_dialog.isVisible():
528             self.retry_connection_dialog.update_connecting_port(port_name)
529 
530     @pyqtSlot()
531     def handle_connection_failure_from_worker(self):#Fallo en el intento de conexion
532         if not self.retry_connection_dialog:
533             self.initiate_first_connection_attempt() # Intenta crearlo y mostrarlo
534             return
535         is_retry = (self.retry_connection_dialog._current_mode == "connecting_retry")
536         self.retry_connection_dialog.show_connection_failed(is_after_retry=is_retry)
537 
538 
539     @pyqtSlot()
540     def _trigger_connection_retry_from_dialog(self): # Reintento de conexión solicitado
541         if self.retry_connection_dialog:
542             self.retry_connection_dialog.show_connecting_retry()
543         QMetaObject.invokeMethod(self.worker, "connect_serial", QtCore.Qt.QueuedConnection)
544 
545     def toggle_serial_connection(self, value): # Para conectar si se ha desconectado
546         if value == 1:
547             if not self.serial_connected: # Solo intentar conectar si no lo está ya
548                 if self.retry_connection_dialog and not self.retry_connection_dialog.isVisible():
549                     self.retry_connection_dialog.show_connecting_retry() # Mostrar como un reintento
550                 elif not self.retry_connection_dialog: # Si nunca se creó (improbable aquí)
551                     self.initiate_first_connection_attempt()
552                     return
553 
554                 QMetaObject.invokeMethod(self.worker, "connect_serial", QtCore.Qt.QueuedConnection)
555             else: # Ya conectado, el scrollbar no debería permitir moverse a 1 de nuevo si ya está en 1.
556                 # Forzar valor a 1 si ya está conectado.
```

```
                self.scrollBar_start.blockSignals(True); self.scrollBar_start.setValue(1); self.scrollBar_start.blockSignals(False)

        elif value == 0:
            if self.serial_connected: # Solo desconectar si está conectado
                QMetaObject.invokeMethod(self.worker, "disconnect_serial", QtCore.Qt.QueuedConnection)
            else: # Ya desconectado, forzar valor a 0.
                self.scrollBar_start.blockSignals(True)
                self.scrollBar_start.setValue(0)
                self.scrollBar_start.blockSignals(False)

    @pyqtSlot(bool)
    def update_serial_status(self, connected):
        self.dial_angulo.setEnabled(connected);self.spinBox_angulo.setEnabled(connected);self.slider_angulo.setEnabled(connected)
        self.slider_kp.setEnabled(connected);self.doubleSpinBox_kp.setEnabled(connected);self.escalakp.setEnabled(connected)
        self.slider_ki.setEnabled(connected);self.doubleSpinBox_ki.setEnabled(connected);self.escalaki.setEnabled(connected)
        self.slider_kd.setEnabled(connected);self.doubleSpinBox_kd.setEnabled(connected);self.escalakd.setEnabled(connected)
        self.spinBox_tiempomuestreo.setEnabled(connected)
        self.button_guardar.setEnabled(connected)
        self.checkBox_referencia.setEnabled(connected);self.checkBox_resultado.setEnabled(connected)
        self.checkBox_pid.setEnabled(connected);self.checkBox_error.setEnabled(connected)
        self.button_Iniciar.setEnabled(connected);self.button_Parar.setEnabled(False)
        self.doubleSpinBox_Ts.setEnabled(connected)
        if connected:
            if self.retry_connection_dialog and self.retry_connection_dialog.isVisible():
                self.retry_connection_dialog.connection_successful()
            self.serial_connected = True
            self.button_Iniciar.setEnabled(True)
            self.scrollBar_start.blockSignals(True)
            self.ON.setStyleSheet("color: white;background: green;")
            self.OFF.setStyleSheet("color: black;background: gray;")
            self.scrollBar_start.setValue(1)
            self.scrollBar_start.blockSignals(False)
        else:
            self.scrollBar_start.setEnabled(True)
            self.serial_connected = False
            if self.control_timer.isActive():
                self.control_timer.stop()
                if hasattr(self, 'duration_of_current_run'):
                    self.spinBox_tiempomuestreo.setValue(self.duration_of_current_run)
            self.scrollBar_start.blockSignals(True)
            self.ON.setStyleSheet("color: black;background: gray;")
            self.OFF.setStyleSheet("color: white;background: red;")
            self.scrollBar_start.setValue(0)
            self.scrollBar_start.blockSignals(False)

    def start_control_timed(self):
        kp = self.doubleSpinBox_kp.value()
        ki = self.doubleSpinBox_ki.value()
        kd = self.doubleSpinBox_kd.value()
        ts = self.doubleSpinBox_Ts.value()
        reference_angle = self.spinBox_angulo.value()

        self.gui_parameters_changed.emit(kp, ki, kd, reference_angle, ts) # Enviar parámetros al worker
        QMetaObject.invokeMethod(self.worker, "start_pid_control", QtCore.Qt.QueuedConnection) # Iniciar PID en worker

        self.duration_of_current_run = int(self.spinBox_tiempomuestreo.value())
        self.spinBox_tiempomuestreo.setEnabled(False)
        self.button_Iniciar.setEnabled(False) # Deshabilitar mientras funciona
        self.doubleSpinBox_Ts.setEnabled(False)
```

```
616         self.button_Parar.setEnabled(True)
617         self.control_active = True
618         self.control_timer.start(1000)
619 
620         self.time_data.clear(); self.reference_angle_data.clear(); self.result_angle_data.clear()
621         self.pid_output_data.clear(); self.error_data.clear()
622         self.start_time = None
623         for curve in [self.reference_curve, self.result_curve, self.pid_curve, self.error_curve]:
624             if hasattr(curve, 'clear'): curve.clear()
625         self.update_plot_visibility()
626 
627     def stop_control(self):# Detener PID en el worker
628         QMetaObject.invokeMethod(self.worker, "stop", QtCore.Qt.DirectConnection)
629 
630         self.spinBox_tiempomuestreo.setEnabled(True)
631         if hasattr(self, 'duration_of_current_run'):
632             self.spinBox_tiempomuestreo.setValue(self.duration_of_current_run)
633         self.doubleSpinBox_Ts.setEnabled(True)
634         self.button_Iniciar.setEnabled(self.serial_connected)
635         self.button_Parar.setEnabled(False)
636         self.control_active = False
637         if self.control_timer.isActive():
638             self.control_timer.stop()
639 
640     @pyqtSlot()
641     def stop_control_timed(self): # Cuando el tiempo de simulacion termina
642         self.stop_control()
643 
644     def update_remaining_time_in_spinbox(self):
645         if not (self.control_active and self.serial_connected):
646             if self.control_timer.isActive():
647                 self.control_timer.stop()
648             return
649         current_displayed_time = self.spinBox_tiempomuestreo.value()
650         if current_displayed_time > 0:
651             self.spinBox_tiempomuestreo.setValue(current_displayed_time - 1)
652             # Verificar si *después* de decrementar, ha llegado a cero
653             if self.spinBox_tiempomuestreo.value() == 0:
654                 self.stop_control_timed() # Esto también detendrá el temporizador
655         elif current_displayed_time == 0: # Debería haberse detenido en el tick anterior
656             if self.control_timer.isActive(): # Salvaguarda
657                 self.stop_control_timed()
658 
659     @pyqtSlot(int)
660     def update_angle(self, angle):
661         internal_angle = angle % 360 # Normalizar ángulo
662         if internal_angle < 0: internal_angle += 360
663 
664         self.slider_resultado.setValue(internal_angle)
665         self.spinBox_resultado.setValue(angle)
666 
667         expected_dial_val_at_zero = 270
668         new_dial_val = (expected_dial_val_at_zero - internal_angle) % 360
669         if new_dial_val < 0: new_dial_val += 360
670         self.dial_resultado.setValue(new_dial_val)
671 
672         if self.control_active:
673             current_time_plot = time.perf_counter()
674             if self.start_time is None:
675                 self.start_time = current_time_plot
676 
677             elapsed_time = current_time_plot - self.start_time
678             self.time_data.append(elapsed_time)
```

```
679             self.reference_angle_data.append(self.spinBox_angulo.value())
680             self.result_angle_data.append(angle)
681             self.pid_output_data.append(self.worker.current_pid_output) # Tomar del worker
682             self.error_data.append(self.spinBox_angulo.value() - angle)
683             # Actualizar datos solo si la curva correspondiente está visible
684             if self.checkBox_referencia.isChecked(): self.reference_curve.setData(self.time_data,
self.reference_angle_data)
685             if self.checkBox_resultado.isChecked(): self.result_curve.setData(self.time_data, self.result_angle_data)
686             if self.checkBox_pid.isChecked(): self.pid_curve.setData(self.time_data, self.pid_output_data)
687             if self.checkBox_error.isChecked(): self.error_curve.setData(self.time_data, self.error_data)
688 
689     @pyqtSlot()
690     def update_plot_visibility(self):
691         has_data = bool(self.time_data)
692         def set_curve_visibility(checkbox_widget, curve_item, y_data_list,css_color_str):
693             is_checked = checkbox_widget.isChecked()
694             curve_item.setVisible(is_checked) # Oculta o muestra el PlotDataItem
695             if is_checked:
696                 checkbox_widget.setStyleSheet(f"color: {css_color_str}; font-weight: bold;")
697                 if has_data:
698                     curve_item.setData(self.time_data, y_data_list)
699             else:
700                 curve_item.clear()
701                 checkbox_widget.setStyleSheet(self.STYLE_TEXT_DEFAULT)
702 
703         set_curve_visibility(self.checkBox_referencia, self.reference_curve, self.reference_angle_data,
self.COLOR_REFERENCIA_CSS)
704         set_curve_visibility(self.checkBox_resultado, self.result_curve, self.result_angle_data,
self.COLOR_RESULTADO_CSS)
705         set_curve_visibility(self.checkBox_pid, self.pid_curve, self.pid_output_data, self.COLOR_PID_CSS)
706         set_curve_visibility(self.checkBox_error, self.error_curve, self.error_data, self.COLOR_ERROR_CSS)
707 
708     # --- Angulo de referencia ---
709     def slider_angulo_cambiado(self): self.spinBox_angulo.setValue(self.slider_angulo.value())
710     def spinBox_angulo_cambiado(self):
711         value = self.spinBox_angulo.value()
712         self.slider_angulo.setValue(value)
713         expected_dial_val_at_zero = 270
714         new_dial_val = (expected_dial_val_at_zero - value) % 360
715         if new_dial_val < 0: new_dial_val += 360
716         self.dial_angulo.setValue(new_dial_val)
717         self.update_parameters()
718     def dial_angulo_cambiado(self):
719         dial_val = self.dial_angulo.value()
720         expected_dial_val_at_zero = 270
721         angle_val = (expected_dial_val_at_zero - dial_val + 360) % 360
722         self.spinBox_angulo.blockSignals(True); self.spinBox_angulo.setValue(angle_val);
self.spinBox_angulo.blockSignals(False)
723         self.slider_angulo.blockSignals(True); self.slider_angulo.setValue(angle_val);
self.slider_angulo.blockSignals(False)
724         self.update_parameters()
725 
726     # --- Métodos para KP, KI, KD ---
727     def _update_pid_spinbox_from_slider(self, slider, spinbox, escala_spinbox):
728         valor_base = slider.value() * (10.0 / slider.maximum())
729         valor_final = valor_base * (10**escala_spinbox.value())
730         spinbox.blockSignals(True); spinbox.setValue(valor_final); spinbox.blockSignals(False)
731     def _update_pid_slider_from_spinbox(self, slider, spinbox, escala_spinbox):
732         valor_spinbox = spinbox.value()
733         escala = escala_spinbox.value()
734         valor_base_calculado = valor_spinbox / (10**escala)
735         if valor_base_calculado > 10.0:
736             valor_base_calculado = 10.0
737             # Si se limita, ajustar también el spinbox para mantener la consistencia
```

```
            valor_spinbox_ajustado = valor_base_calculado * (10**escala)
            spinbox.blockSignals(True); spinbox.setValue(valor_spinbox_ajustado); spinbox.blockSignals(False)
        elif valor_base_calculado < 0.0: # Asumiendo que Kp, Ki, Kd no son negativos
            valor_base_calculado = 0.0
            valor_spinbox_ajustado = valor_base_calculado * (10**escala)
            spinbox.blockSignals(True); spinbox.setValue(valor_spinbox_ajustado); spinbox.blockSignals(False)
        # Convertir el valor base (0-10) a un valor de slider (0-max_slider)
        slider_val = int(round(valor_base_calculado * (slider.maximum() / 10.0)))
        slider.blockSignals(True); slider.setValue(slider_val); slider.blockSignals(False)


    def slider_kp_cambiado(self):
        self._update_pid_spinbox_from_slider(self.slider_kp, self.doubleSpinBox_kp, self.escalakp);
        self.update_parameters()
    def doubleSpinBox_kp_cambiado(self):
        self._update_pid_slider_from_spinbox(self.slider_kp, self.doubleSpinBox_kp, self.escalakp);
        self.update_parameters()
    def escalakp_cambio(self): self.doubleSpinBox_kp_cambiado()

    def slider_ki_cambiado(self):
        self._update_pid_spinbox_from_slider(self.slider_ki, self.doubleSpinBox_ki, self.escalaki);
        self.update_parameters()
    def doubleSpinBox_ki_cambiado(self):
        self._update_pid_slider_from_spinbox(self.slider_ki, self.doubleSpinBox_ki, self.escalaki);
        self.update_parameters()
    def escalaki_cambio(self): self.doubleSpinBox_ki_cambiado()

    def slider_kd_cambiado(self):
        self._update_pid_spinbox_from_slider(self.slider_kd, self.doubleSpinBox_kd, self.escalakd);
        self.update_parameters()
    def doubleSpinBox_kd_cambiado(self):
        self._update_pid_slider_from_spinbox(self.slider_kd, self.doubleSpinBox_kd, self.escalakd);
        self.update_parameters()
    def escalakd_cambio(self): self.doubleSpinBox_kd_cambiado()

    def button_guardar_clicked(self):
        options = QFileDialog.Options()
        fileName, selectedFilter = QFileDialog.getSaveFileName(
            self, "Guardar señales", "",
            "Excel Files (*.xlsx);;MATLAB Files (*.mat)",
            options=options, initialFilter="Excel Files (*.xlsx)")

        if fileName:
            data_dict = {
                'Tiempo (s)': self.time_data,
                'Angulo de Referencia (º)': self.reference_angle_data,
                'Ángulo Resultado (º)': self.result_angle_data,
                'Salida PID': self.pid_output_data,
                'Error': self.error_data }
            try:
                if selectedFilter == "Excel Files (*.xlsx)":
                    if not fileName.endswith(".xlsx"): fileName += ".xlsx"
                    pd.DataFrame(data_dict).to_excel(fileName, index=False)
                    QMessageBox.information(self, "Guardado", f"Datos guardados en Excel: {fileName}")
                elif selectedFilter == "MATLAB Files (*.mat)":
                    if not fileName.endswith(".mat"): fileName += ".mat"

                    mat_data = {'t': data_dict['Tiempo (s)'], 'u_ref': data_dict['Angulo de Referencia (º)'],
                                'y_out': data_dict['Ángulo Resultado (º)'], 'pid_val': data_dict['Salida PID'],
                                'error_val': data_dict['Error']}
                    scipy.io.savemat(fileName, mat_data)
                    QMessageBox.information(self, "Guardado", f"Datos guardados en MATLAB: {fileName}")
            except Exception as e:
```

```
801                 QMessageBox.critical(self, "Error al Guardar", f"No se pudo guardar el archivo {fileName}:\n{e}")
802
803     def closeEvent(self, event): # Cerrar ventana
804         if hasattr(self, 'control_active') and self.control_active:
805             self.stop_control() # Llama a worker.stop()
806
807         if hasattr(self, 'worker') and self.worker:
808             self.worker._running = False # Parar el bucle run() del worker se termine
809
810         if hasattr(self, 'thread') and self.thread and self.thread.isRunning():
811             self.thread.quit() # Pide al bucle de eventos del hilo que termine
812             if not self.thread.wait(3500): # Espera a que el hilo termine
813                 self.thread.terminate()
814                 self.thread.wait() # Esperar al cierre forzado
815
816         if self.retry_connection_dialog and self.retry_connection_dialog.isVisible():
817             self.retry_connection_dialog.reject()
818         event.accept()
819
820     def update_parameters(self):
821         reference_angle = self.spinBox_angulo.value()
822         kp = self.doubleSpinBox_kp.value()
823         ki = self.doubleSpinBox_ki.value()
824         kd = self.doubleSpinBox_kd.value()
825         ts= self.doubleSpinBox_Ts.value()
826         self.gui_parameters_changed.emit(kp, ki, kd, reference_angle, ts)
```

Código 36 Class Main (Python)

```
828 if __name__ == "__main__":
829     app = QApplication(sys.argv)
830     main_window = main()
831     main_window.show()
832     sys.exit(app.exec_())
```

Código 37 Funcion principal (Python)

9.3 BIBLIOGRAFÍA

Librerías Python

[26] *(06-05-2025)* *https://pypi.org/project/pyserial/*

[27] *(10-05-2025)* *https://pypi.org/project/PyQt5*

[28] *(10-05-2025)* *https://www.pyqtgraph.org*

[29] *(11-05-2025)* *https://pandas.pydata.org*

[30] *(11-05-2025)* *https://scipy.org*

Liberarías Arduino:

[31] *(16-03-2024)* *digitalWriteFast*

[32] *(16-03-2024)* *elegoo tft*

Recursos Python

[33] *(11-05-2025)* *https://doc.qt.io/qt-5/qtwidgets-index.html*

[34] *(11-05-2025)* *https://doc.qt.io/qt-5/stylesheet-examples.html*

[35] *(25-05-2025)* *https://icon-icons.com/es/download/168572/ICO/72/*