

**Diseño y construcción de un sistema de automatización para el control de la dirección del campo magnético en experimentos criogénicos, y su aplicación en el estudio de señales de fondo.**

Ingrid Heuer, Marco Crivaro Nicolini

Laboratorio 7  
Dept. de Física – FCEyN - UBA

Octubre 2020

**ALUMNOS:**

Ingrid Heuer - LU N° : 266/16 - ingridheuer94@gmail.com

Marco Crivaro Nicolini - LU N° : 724/16 - crivaronicolini@gmail.com

**LUGAR DE TRABAJO:**

Laboratorio de Bajas Temperaturas, DF FCEN, UBA

**DIRECTORA DEL TRABAJO:**

Dra. Gabriela Pasquini – pasquini@df.uba.ar

Dra. Gabriela Pasquini

Ingrid Heuer

Marco Crivaro Nicolini

# Índice

<b>1. Introducción</b>	<b>1</b>
1.1. Experimentos con Electroimán rotante . . . . .	1
1.2. Señales de fondo en mediciones de transporte alterno . . . . .	2
<b>2. Automatización del giro del electroimán</b>	<b>2</b>
2.1. Diseño y fabricación del equipo . . . . .	2
2.2. Electrónica . . . . .	4
2.3. Control y automatización . . . . .	6
2.4. Protocolo de comunicación . . . . .	9
<b>3. Modelado: Ruidos microfónicos causados por un campo magnético rotante</b>	<b>10</b>
3.1. Antecedentes y motivación . . . . .	10
3.2. Inducción en una espira bajo un campo magnético rotante . . . . .	12
3.3. Vibración de la espira debido al campo magnético . . . . .	13
<b>4. Simulación y medición del fondo angular</b>	<b>14</b>
4.1. Discusión . . . . .	17
<b>5. Conclusiones</b>	<b>18</b>
<b>Referencias</b>	<b>18</b>
<b>Apéndices</b>	<b>19</b>

## Resumen

En este trabajo se detalla el diseño y ensamblado de un dispositivo para rotar un electroimán instalado en el Laboratorio de Bajas Temperaturas, con foco en la facilidad del armado y el costo de los componentes. También se detalla la implementación del software de control y automatización realizado en Arduino C++, junto a una interfaz gráfica de LabVIEW que vincula el equipo al software de medición experimental.

Por último se aplica el nuevo dispositivo al estudio de una señal de fondo preexistente en las mediciones, y se comparan los resultados con un modelo matemático basado en ruidos microfónicos.

# 1. Introducción

## 1.1. Experimentos con Electroimán rotante

Una de las líneas de investigación del laboratorio de Bajas Temperaturas en los últimos años se centra en investigar el acople entre la fase superconductora y las llamadas fases nemáticas de ciertos superconductores no convencionales. Estas fases se estudian modificando las simetrías del material, comprimiendo y estirando la muestra en direcciones relevantes de la estructura cristalina [1]. En particular, es relevante el estudio de la dependencia de las propiedades superconductoras con la dirección del campo magnético. A lo largo de estas investigaciones [2] se ha desarrollado un montaje experimental (figura 1) que consiste de un electroimán modelo GMW 3472-70[3] que en general se opera en campos del orden de 5000G, y está apoyado sobre una base giratoria marcada con un goniómetro. Entre las piezas polares se coloca un crióstato que aloja la muestra en la región donde el campo es homogéneo, junto a los circuitos de control y medición a cuatro puntas, éste último es el más relevante para este trabajo.

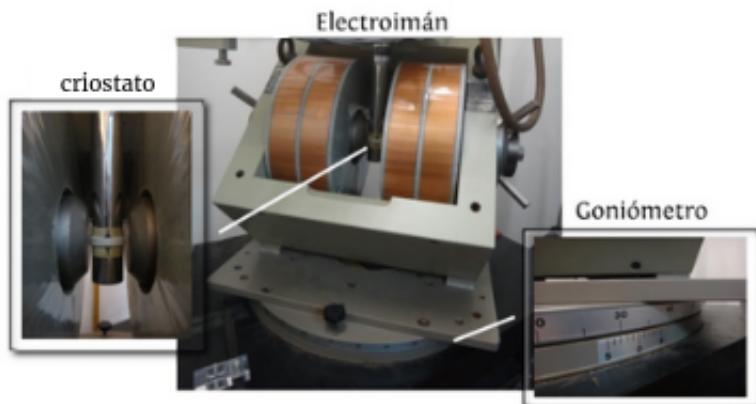


Figura 1: Montaje experimental, se señalan las partes principales del equipo.

Para medir la resistencia de la muestra en función de la dirección del campo magnético, el procedimiento requería girar manualmente el electroimán, que pesa alrededor de 400 kg, mientras con una cámara se grababa el avance del goniómetro. Esta grabación luego se sincronizaba con los valores tomados en el lock-in en una etapa posterior. Este proceso además de ser poco práctico puede llevar a incertezas adicionales en los valores de ángulo. El principal objetivo de este trabajo es mejorar este proceso de medición. Para ello se implementó un sistema motorizado para rotar el electroimán y un software de control para automatizar las mediciones.

El sistema mecánico que se diseñó consta de una plataforma giratoria, impulsada por un motor paso a paso que se acopla a la base del electroimán. Se realizaron los planos en CAD y se mandaron a fabricar las piezas necesarias en el taller de la facultad. El dispositivo se controla con un Arduino UNO y se implementó una interfaz en LabVIEW para su uso desde la computadora. Con el software se logró automatizar el proceso de medición, incorporar rutinas de calibración y medidas de seguridad. También se armó un gabinete para almacenar los componentes electrónicos.

## 1.2. Señales de fondo en mediciones de transporte alterno

Por otro lado, en las mediciones de resistividad se ha observado una señal de fondo que depende del campo magnético en temperaturas donde las muestras no interactúan con él.

La señal de fondo tiene una amplitud significativa en el intervalo de frecuencias de 800 Hz a 1200 Hz aproximadamente. Estas señales tienen la particularidad de que no pueden ser *a priori* explicadas por física de la muestra, tanto en su estado superconductor como a temperatura ambiente y por lo tanto deben ser analizadas como señales espurias. Esto implica que el circuito no puede modelizarse idealmente como en está planteado la figura sino que se debe idear un modelo que explique estas señales.

Se propuso como objetivo construir un modelo que logre explicar las características de la señal detectada, en base a los datos recolectados previamente en el laboratorio. En la sección 3 se dan los detalles del circuito de medición, y se propone un modelo para explicar este fondo, basado en ruidos microfónicos causados por la interacción del campo magnético con el circuito de medición. Además, se buscó que el modelo sea de utilidad para estimar la magnitud de señales espurias de ésta naturaleza en futuros experimentos.

## 2. Automatización del giro del electroimán

En esta sección se detallan el diseño e implementación del equipo de automatización del equipo. En las subsección 2.1 se comentan las partes que se diseñaron y mandaron a fabricar para armar el equipo, mientras que en la subsección 2.2 se detalla el hardware de control. En la subsección 2.3 se detallan todos los protocolos y rutinas de software implementadas para la correcta automatización del equipo.

### 2.1. Diseño y fabricación del equipo

En esta sección se resume el proceso de diseño y fabricación de los componentes mecánicos del equipo. Se estudiaron varias soluciones para la transmisión de potencia desde el motor a la plataforma giratoria del electroimán. En la figura 2 se muestra el diseño final, en el que se atornilla a la plataforma una placa circular que a su vez se atornilla a un eje macizo. Este atraviesa la mesa y llega a un reductor a sinfin y corona, que es alimentado por un motor paso a paso. El conjunto del reductor y el motor se monta en una placa sostén sujetada a media pulgada de la parte inferior de la mesa, aprovechando los bulones existentes que sujetan el aro de la plataforma a la mesa.

Se midió el torque necesario para vencer el rozamiento estático de la plataforma. Para ello se armó un torquímetro atando una canasta a un extremo de la plataforma, dejándola colgar al costado de la mesa con una polea. En la canasta se colocaron pesas y se midió el peso que hacía girar el electroimán, cuidando que la cuerda siga el sentido tangencial a la circunferencia de la plataforma. Se multiplicó el máximo de las fuerzas por la distancia al eje de rotación y se obtuvo un torque mínimo necesario para el equipo de  $38 \pm 4 \text{ Nm}$ . Además se resolvió agregar un margen de seguridad del 10 %, requiriendo una reducción que otorgue al menos  $42 \text{ Nm}$ .

Conociendo el valor de torque estático se pudieron optimizar los parámetros que definen la transmisión de potencia: el torque del motor paso a paso y la reducción mecánica. Estos valores siguen una relación inversamente proporcional según  $T_{out} = RT_{in}$ , siendo  $R = N_{out}/N_{in}$  la relación de transmisión definida como el cociente de la cantidad de dientes entre el engranaje de salida y el de entrada. Manteniendo el torque del motor fijo, el torque a la salida del reductor se puede multiplicar usando una relación de transmisión mayor, a costas de una reducción en la velocidad final. Como en la operación normal del instrumento no se

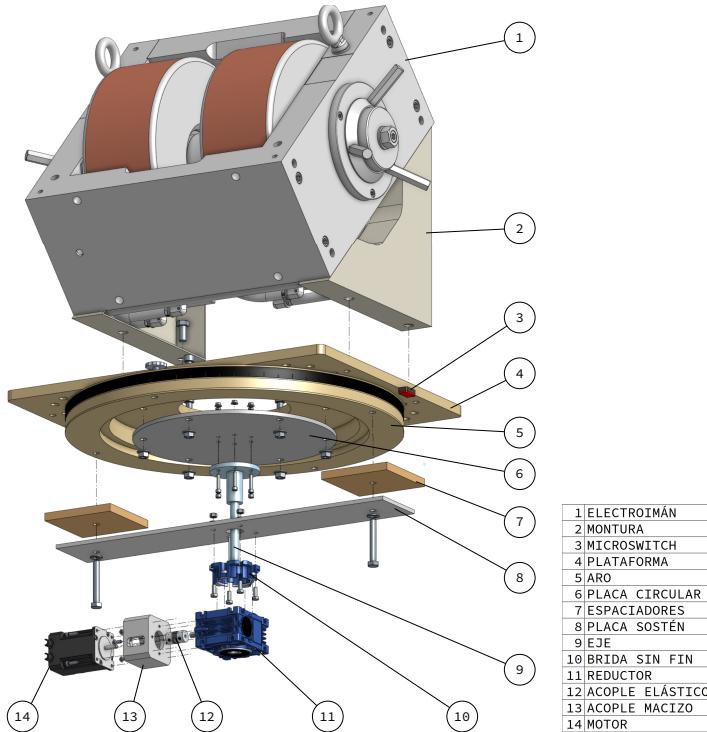
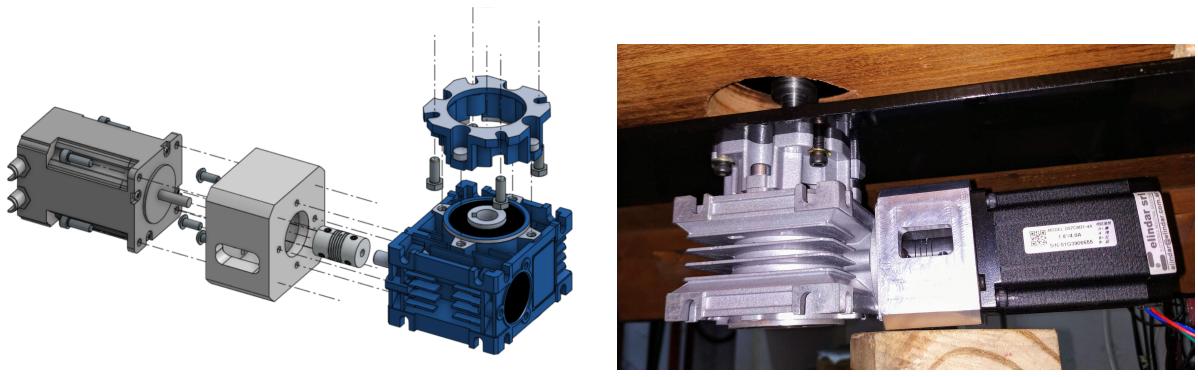


Figura 2: Dibujo de despiece de todo el dispositivo.

requieren velocidades altas, se optó por la reducción mas grande accesible. De las reducciones disponibles en el mercado se eligió el modelo STM WMI-30 con una relación  $R = 1 : 60$  ya que esto permite usar un motor económico de 3, 1Nm modelo LEADSHINE 57CM31A-4A. Teniendo en cuenta la eficiencia de la reducción, con estos componentes se puede alcanzar un torque máximo de 54Nm, que cumple los requisitos del dispositivo.

La unión entre el motor paso a paso y el reductor se realizó mediante dos acoples, uno elástico que une los ejes y otro macizo que une las carcasas. El primero se compró junto al motor, y está fabricado en aluminio, con tornillos que ajustan los ejes en los extremos. Tiene un corte en espiral en el medio que hace que actúe como resorte, y esto permite que haya una ligera desalineación entre los ejes. El segundo se envió a fabricar bajo plano (página 25) en el taller de la facultad. Está hecho también en aluminio, y se diseñó para soportar el peso del motor y para garantizar que los dos ejes estén lo más alineados posible.

La instalación del conjunto motor-reductor se realiza atornillando el acople a la carcasa del sin fin, con el acople elástico ya instalado en su eje. Luego se inserta el motor a la otra cara y se lo sujetta con tornillos pasantes hacia el acople de aluminio. Por último se ajusta el acople elástico al eje del motor a través de la ventana hecha en el acople. En la figura 3 se detalla la unión del reductor al motor junto a una captura del armado terminado.



(a)

(b)

Figura 3: a) Modelo del ensamble reductor, acoples y motor. b) Captura del montaje desde abajo de la mesa.

Las placas del montaje se diseñaron sobre el modelo 3D del electroimán que provee el fabricante para tener la mejor exactitud en la localización de los agujeros que se realizaron por corte de plasma CNC. Este método provee una excelente precisión en los detalles de las piezas a un costo muy inferior a otros métodos de corte como corte por agua o fresado. Previo a esto se hizo un análisis de fuerzas y deformaciones en las placas mediante elementos finitos con el programa SimScale para determinar el tipo de acero y espesor a utilizar (figura 4). Para tener un margen de seguridad en el modelo se modelaron fuerzas tres veces mas grandes que las esperadas por la medición. Se observa que para un espesor de un cuarto de pulgada y acero SAE 1010 (un acero de fabricación usual [4]) la deformación máxima es mucho menor a 1mm, y el stress máximo está cómodamente dentro del margen de seguridad para este tipo de acero [4]. Por lo tanto el espesor y tipo de acero son adecuados para esta aplicación.

El eje de transmisión se envió a fabricar bajo plano (página 22) al taller de la facultad en acero SAE 1060, un acero recomendado para transmisiones. En el diseño se calculó el diámetro requerido para transmitir el torque de salida de la reducción. En base a los métodos especificados en [4] se resolvió que un diámetro de una pulgada es suficiente para transmitir el torque de salida del reductor. Ese diámetro tiene un rebate a 14mm para encastrar con la salida de la transmisión con una chaveta y en el extremo superior se atornilla a la placa redonda con seis tornillos.

## 2.2. Electrónica

Habiendo elegido el modelo de motor paso a paso a utilizar se compraron el resto de los componentes electrónicos. Para controlar el dispositivo se utilizó un ARDUINO UNO que le envía señales a un driver modelo LEADSHINE DM542E de 4,2A que combina estas señales con la corriente provista por una fuente de tipo switching modelo POWERSWITCH OF24-5A de 120W. Este modelo de driver en particular permite un alto nivel de *microstepping*: una forma de interpolar las corrientes en las dos bobinas del motor de manera que el movimiento resulte más fluido y se minimice el ruido y las vibraciones. Además se incorporó un convertidor DC-DC *step-down* regulable, que usa los 24V de la fuente para alimentar el arduino con 12 a 7V. De esta manera el equipo es independiente de la computadora y se puede usar con botones.

Para contener la electrónica del proyecto se recicló el gabinete de un amplificador en desuso del labo-

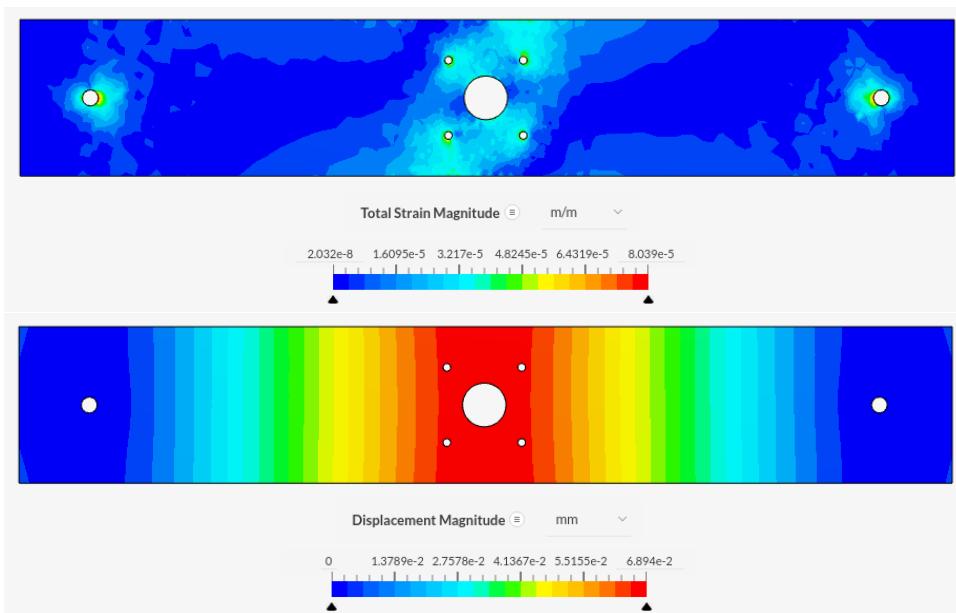


Figura 4: Modelado de estrés y desplazamiento ejercido sobre la placa sostén.

ratorio. En la figura 5 se muestran varias capturas del armado. Consiste de un cajón metálico al cual se le hicieron perforaciones en la parte trasera para un interruptor de prendido y apagado, un echufe de interlock, y conectores para los microswitches y el motor. En el interior se atornillaron el driver, fuente, arduino y step-down con espaciadores en una chapa de aluminio cortada y doblada a mano con la guillotina y dobladora del taller de la facultad. Al frente del gabinete se le colocaron botones para las funciones 'izquierda', 'derecha' y también botones que realizan el homing y el frenado, junto con la ficha USB del arduino.



Figura 5: Distintas capturas del interior y exterior del gabinete armado.

Los componentes se cablearon siguiendo el esquema del circuito de la figura 11, el mismo se realizó en base a las especificaciones de la fuente y del driver. Las conexiones al arduino se hicieron teniendo en cuenta las distintas funciones de los pines en el arduino UNO, por ejemplo, los microswitches solo pueden ir a los pines 2 y 3, ya que solo estos tienen la función de interrupción.

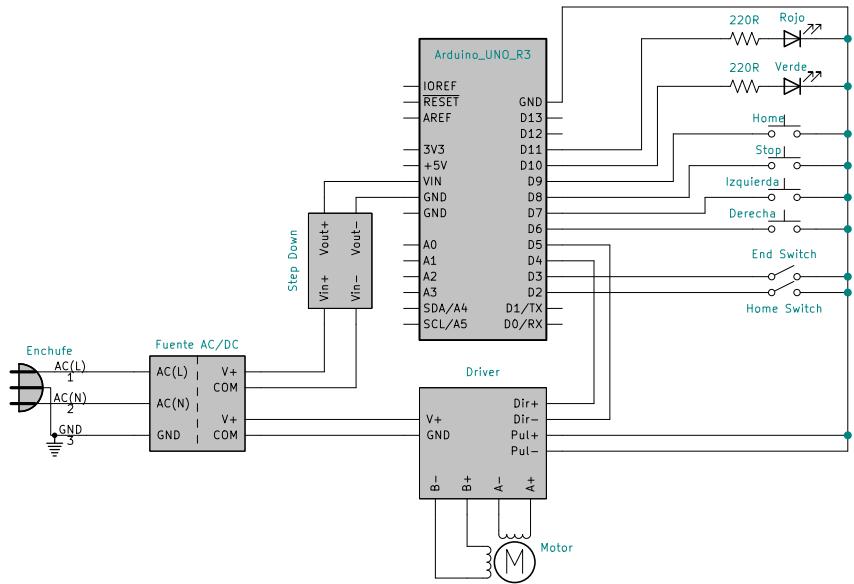


Figura 6: Esquema del circuito del dispositivo.

### 2.3. Control y automatización

Se implementó un software de control para el dispositivo que permite automatizar el proceso de medición con un Arduino UNO. En su desarrollo se tuvo en cuenta el tiempo de ejecución de cada ciclo, tratando de obtener un programa rápido y eficiente y economizar el uso de memoria del controlador. Además se buscó que el código sea fácil de interpretar y modificar, para que se pueda adaptar en un futuro a otras aplicaciones del laboratorio. El código de C++ del programa se adjunta en el apéndice B y en la figura 7 se muestra un diagrama de flujo simplificado del mismo.

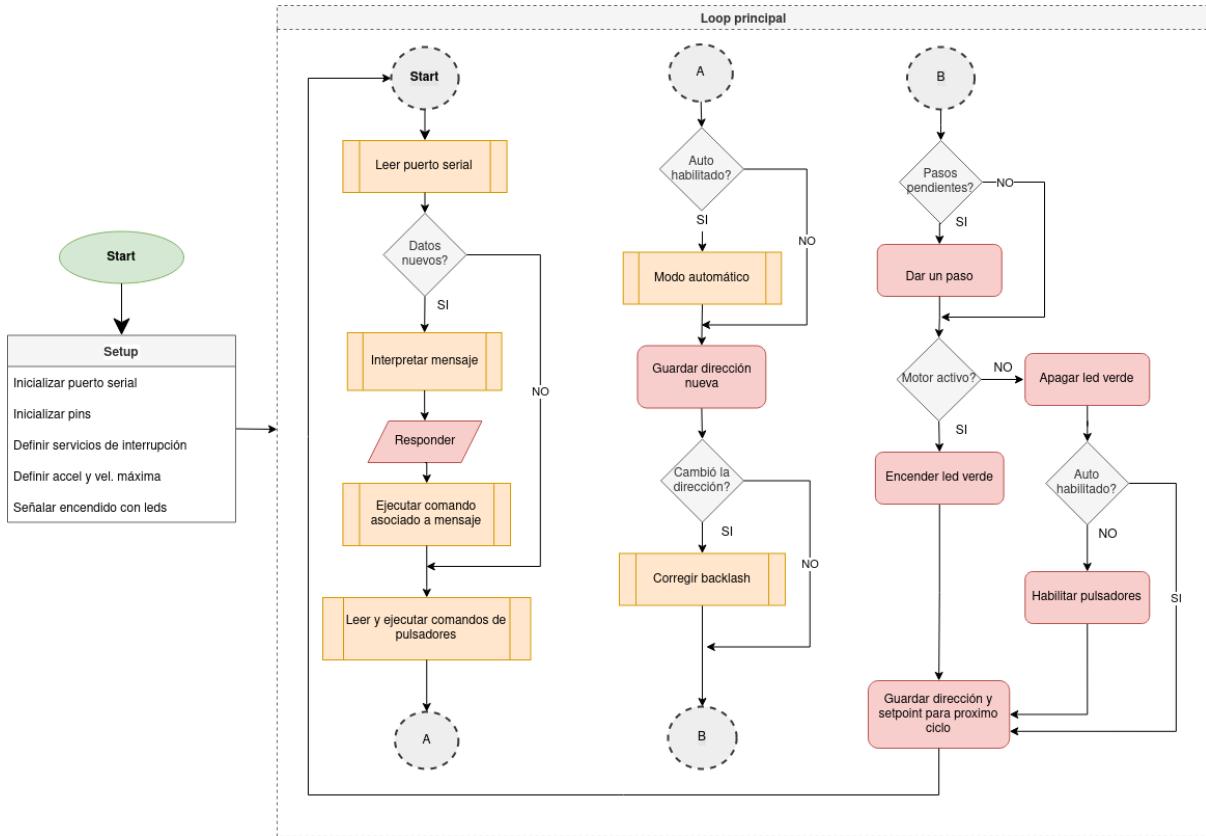


Figura 7: Diagrama de flujo del programa. Se muestra el funcionamiento general del mismo, los procesos más complejos se representan recuadrados en naranja con el nombre de la función correspondiente.

Al encender el sistema, se configuran los dispositivos periféricos y el puerto USB. También se definen las interrupciones y sus subrutinas asociadas. El ciclo principal (Sección B.4) comienza siempre leyendo el puerto USB y detectando si se recibieron datos. Si es así, se lleva a cabo el proceso de comunicación que se detalla en la sección 2.4. Luego, se lee el estado de los pulsadores y se efectúan las acciones que corresponden a cada uno. Si quedan pasos pendientes para llegar a la posición objetivo, se envía una señal al motor para que de un paso. Por último se indica si el motor está activo encendiéndo un led, y se decide si se habilitan los pulsadores, según el modo del programa.

Como medida de seguridad se instalaron microswitches al principio y final del recorrido del electroimán, que generan una interrupción de hardware al ser activados. Estas interrupciones son independientes del resto del programa, lo que permite desacoplarlas y darles mayor prioridad. Para ello se conectaron los microswitches a los puertos 2 y 3 del Arduino (que son los únicos con esta capacidad en el modelo UNO) y se les asoció una rutina de interrupción, que se puede ver en la sección B.2. Esta rutina detiene el motor, desactiva todos los pulsadores y el modo automático y enciende un led rojo para indicar la interrupción.

Para calibrar el dispositivo se incluyó la función de *homing* (sección B.5), que sirve para darle una posición de referencia al motor. Esta función consiste en desplazar lentamente el motor mientras se verifica, paso por paso, si se activó el microswitch de principio de recorrido. Una vez activado, se hace retroceder el motor hasta

que se suelta el interruptor y esa posición se define como el cero absoluto de pasos. En ese proceso se guarda la cantidad de pasos que se realizaron retrocediendo, y se utiliza para aproximar el *backlash* de la reducción, que es la cantidad de pasos que se pierden al cambiar de dirección.

Otra función importante que se implementó fue la corrección automática de *backlash* (sección B.10). El programa registra en cada ciclo la dirección del movimiento y detecta cambios en la misma. Si se cambia de dirección, antes de ir hacia la posición objetivo, da los pasos necesarios para compensar el *backlash* y redefine la posición del motor. Esta corrección se efectúa para cualquier movimiento ya sea manual, automático o con los pulsadores. Realizar la corrección es importante porque si no se realizara, el motor al cambiar de dirección aceleraría hasta impactar contra la reducción a una gran velocidad.

Para automatizar las mediciones se realizó una función de recorrido automático, implementada como una maquina de estados. Se especifican los puntos de partida y de llegada, las velocidades correspondientes a cada recorrido y la cantidad de repeticiones. Con estos datos la máquina modifica automáticamente la posición objetivo y la velocidad hasta completar la medición. También se implementó una versión discreta de esta función, donde el dispositivo rota de a pasos y espera en cada posición antes de continuar. Se muestra un diagrama simplificado de su funcionamiento en la figura 8 y su código fuente se adjunta en la sección B.6.

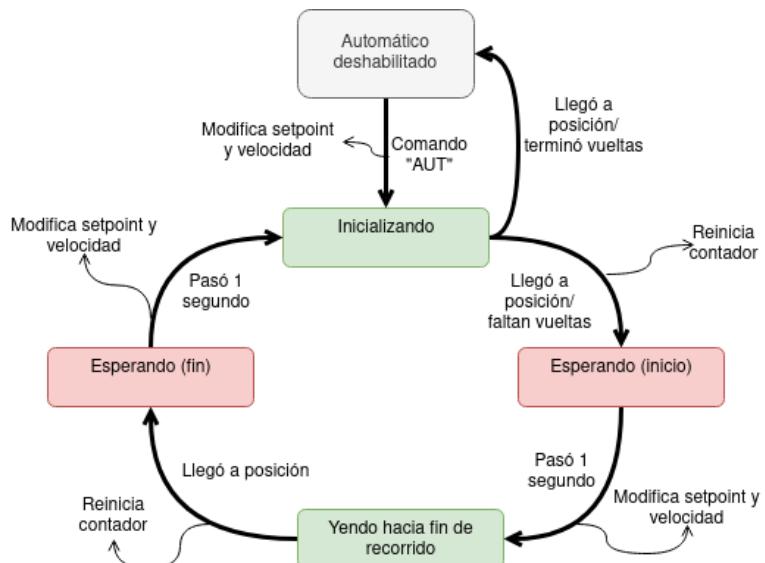


Figura 8: Diagrama de estados del modo automático de adquisición

En el diagrama se muestran recuadrados los distintos estados que puede tomar la máquina y las flechas representan las transiciones entre ellos. En cada transición se señala el evento que la provoca y la acción que realiza la máquina.

El funcionamiento de los pulsadores se implementó mediante una clase que permite manejar cada botón como un objeto lógico (sección B.11). La clase tiene la funcionalidad de *debounce* para filtrar ruidos típicos de las señales de interruptores, que pueden ser causados por imperfecciones mecánicas o de los cables.

Por último se diseñó un programa de LabVIEW para poder incorporar el nuevo dispositivo al sistema de adquisición que ya se utilizaba en el laboratorio, en la figura 9 se muestra una imagen del panel frontal.

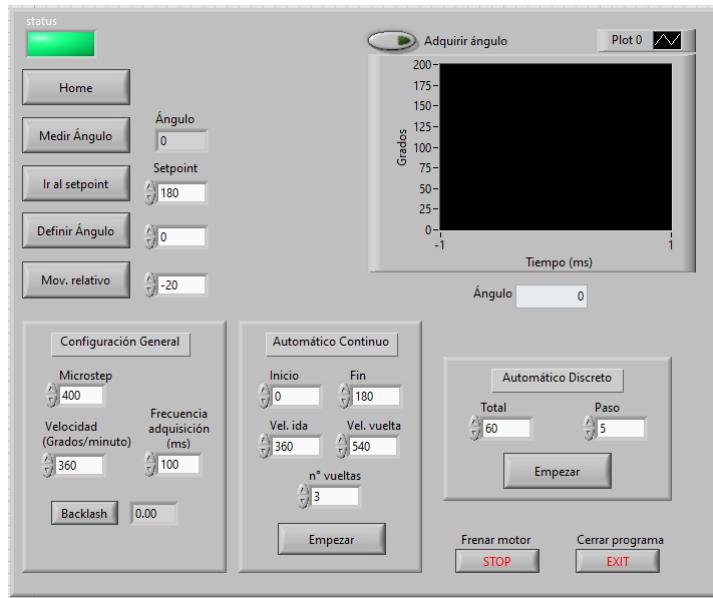


Figura 9: Panel frontal del programa de LabVIEW

Este programa funciona como una interfaz gráfica para el microcontrolador, pero además automatiza la adquisición de datos y calcula la conversión entre la rotación del electroimán y los pasos del motor.

#### 2.4. Protocolo de comunicación

Las comunicaciones entre equipos de laboratorio y la computadora de control siguen distintos paradigmas de comunicación que ordenan la forma de cada mensaje enviado y la información que se contiene. Para este trabajo implementó un protocolo de comunicación entre el Arduino y la interfaz de LabVIEW del tipo *request-response*, este protocolo además de ser fácil de usar, se puede extender y hacer que los mensajes se interpreten más rápido que otros protocolos. Los mensajes tienen el siguiente formato:

<CMD, int , long int>

Donde *CMD* es un código de 3 letras asociado a una función, *int* y *long int* son valores numéricos que luego serán guardados en variables de tipo *integer* y *long integer*. La diferencia entre estos tipos es su uso de memoria, el primero ocupa 16 bits mientras que el segundo ocupa 32 bits, esta distinción es importante para evitar errores causados por *overflows*, cuando se intenta guardar un número que ocupa más memoria de la disponible. En esta aplicación, un error de *overflow* podría causar que el dispositivo realice movimientos inesperados.

Se usan los caracteres '<,>' como principio y fin de linea, que el programa utiliza para diferenciar entre mensajes, lo que evita el uso de *delays* en la comunicación. Sin incluir los caracteres de principio y fin de linea ni las comas de separación, se pueden enviar mensajes de hasta 30 caracteres.

En la figura 10 se muestra un diagrama del proceso de comunicación entre los dispositivos.

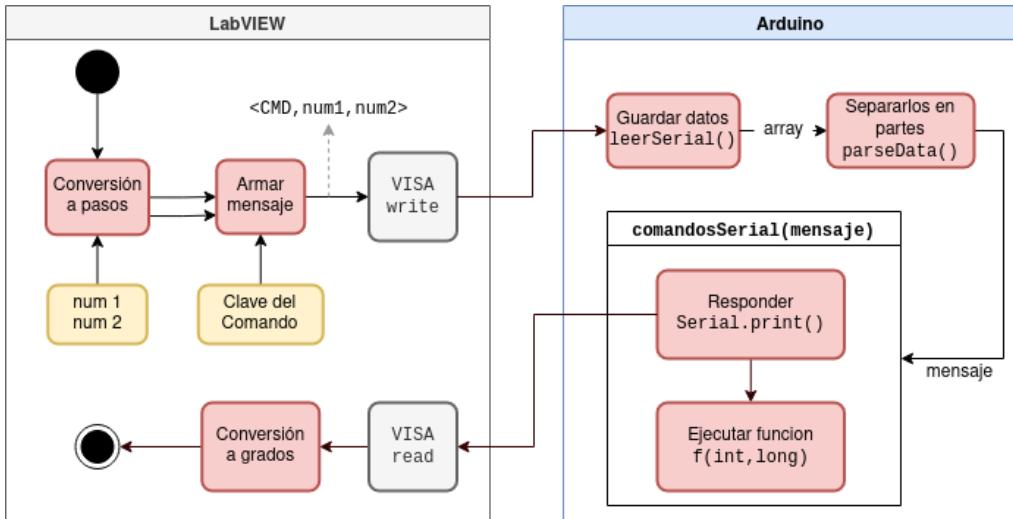


Figura 10: Diagrama de la comunicación entre la interfaz de LabVIEW y el programa de Arduino

El proceso comienza cuando se activa uno de los botones del panel frontal (figura 9), el programa toma los parámetros correspondientes y arma un mensaje con el formato que se mencionó anteriormente y lo envía al Arduino a través del puerto USB, usando un protocolo VISA. Cuando el Arduino recibe el mensaje, primero guarda los datos carácter por carácter en un array, luego los *parsea* o interpreta y los guarda en una variable *mensaje* que contiene toda la información recibida. Antes de ejecutar la función correspondiente, envía una respuesta a la PC, que lee los datos y calcula la conversión entre pasos del motor y la rotación del electroimán. El código fuente de este proceso se halla en la sección B.9.

La idea detrás de este sistema es que sea sencillo agregar nuevas funciones al programa, agregando botones al panel frontal de LabVIEW y con pequeñas modificaciones al código de Arduino.

Para la comunicación serial se utilizó una tasa de baudios de 115200 Bd, es decir 115200 bits por segundo. Como el tiempo de ejecución de cada ciclo es muy rápido (del orden de los microsegundos), la frecuencia de muestreo utilizada estará acotada principalmente por los tiempos de adquisición del resto de los equipos.

### 3. Modelado: Ruidos microfónicos causados por un campo magnético rotante

#### 3.1. Antecedentes y motivación

En los experimentos realizados en el laboratorio se detectaron señales que dependen del campo magnético a temperaturas donde las muestras no interactúan con él. A temperatura ambiente, éste fondo no puede ser atribuido a cuestiones propias del experimento, ya que a esta temperatura no hay fase superconductora y la muestra no presenta magnetoresistencia apreciable, pudiéndose idealizar como una pequeña resistencia fija. Otra característica particular de estas señales es la presencia de una componente en contra fase, que debería ser despreciable en un circuito que es puramente resistivo. La dependencia de la señal respecto al ángulo con el que incide un campo magnético sobre el circuito sugiere que su causa es de naturaleza inductiva. Se

planteó la posibilidad de que los cables de oro que se conectan a la muestra estén formando un lazo donde se induce una fem.

El circuito de medición utilizado consiste de un generador que alimenta una corriente que pasa por la muestra por los extremos, la diferencia de tensión que se genera en la muestra se capta por dos puntas intermedias que van a un lock-in sincronizado con el generador. Un esquema del circuito idealizado se ve en la figura 11.

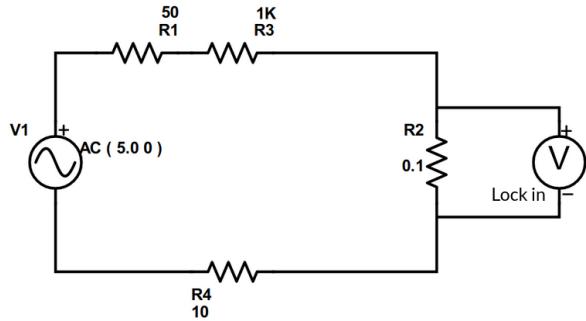


Figura 11: Esquema idealizado del circuito de medición de la resistencia de la muestra; R1 es la resistencia interna del generador, R3 la limitadora, R2 de la muestra y R4 es la resistencia patrón, usada para medir la corriente del circuito. Todas las resistencias tienen valores en Ohm

El circuito es alimentado a tensión AC de una amplitud constante de 0,5V pasando por una resistencia interna al generador de  $50\Omega$  (R1), y la corriente es limitada por una resistencia regulable de  $1\text{k}\Omega$  (R3) alojada en una caja externa donde además de pasar por una resistencia patrón de  $10\Omega$  (R4) se conectan las mallas de los cables BNC a tierra. Luego de la caja los cables se conectan a entradas especiales del crióstato. Dentro de este se usan cables de menor diámetro que se *twistean* para minimizar los lazos que puedan captar un acople magnético. Por ultimo llegan a la punta del crióstato donde se sueldan a unos *pads* desde los que se sueldan cables de oro de  $25\mu\text{m}$  que se pegan con pintura de plata a los extremos de la muestra según se ve en la figura 12. Este tramo de cables que llega y sale de la muestra tiene entre 1 y 3 cm de largo y es la única parte del circuito que no se encuentra *twisteadas*.

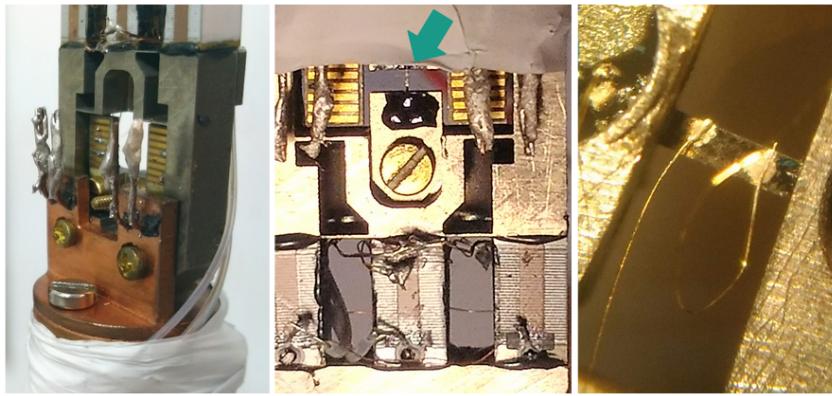


Figura 12: Izq: punta del crióstato, se ven los *pads* soldados. Medio: acercamiento de la anterior. La flecha señala la muestra en sí. Der: acercamiento a la muestra usando un microscopio en donde se observan los contactos de los cables de tensión.

### 3.2. Inducción en una espira bajo un campo magnético rotante

Como punto de partida para el modelado se consideró una espira idealizada, por la cual circula corriente alterna y está inmersa en un campo uniforme  $\vec{B}$  que rota alrededor de ella. Para simplificar, se tomó la espira centrada en el plano  $yz$  y  $\theta$  como el ángulo entre la normal de la espira ( $\hat{x}$ ) y la dirección de  $\vec{B}$ , como se ilustra en la figura 13.

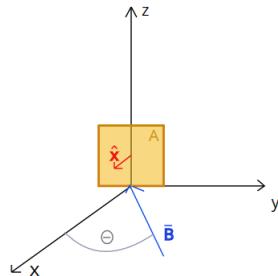


Figura 13: Esquema simplificado de la espira y la dirección del campo incidente

Entonces,  $\vec{B} = B \cos(\theta)\hat{x} + B \sin(\theta)\hat{y}$ . El campo  $\vec{B}$  es de magnitud constante pero cambia su orientación. Partiendo de la ley de Faraday, la fem que se induce en la espira es:

$$\varepsilon = -B \cos(\theta) \cdot \frac{dA}{dt} \quad (1)$$

Se puede ver que la fem varía con la derivada del área de la espira. Esto conduce a estudiar la idea de ruidos microfónicos, un fenómeno donde vibraciones mecánicas en un circuito se traducen en señales eléctricas. En éste caso la vibración mecánica es causada por el mismo campo magnético, que mueve los cables que llevan corriente.

### 3.3. Vibración de la espira debido al campo magnético

Se estudió la deformación de dos cables de longitud  $L$  fijos en sus puntas que vibran bajo la acción de un campo magnético en el plano paralelo al de la espira. Utilizando como base la ecuación de Euler Bernoulli [5] que describe la deflexión  $w$  de vigas frente a vibraciones:

$$\frac{\partial^2 w(x, t)}{\partial t^2} + 2\zeta\omega_n \frac{\partial w(x, t)}{\partial t} + \frac{EI}{\rho A} \frac{\partial^4 w(x, t)}{\partial x^4} = \frac{f(t)}{\rho A} \quad (2)$$

donde  $E = 79 \text{ GPa}$ ,  $I = \frac{\pi}{4}r^4$ ,  $\rho = 19,3 \text{ g/cm}^3$  son el módulo de Young del oro, la componente normal del momento de inercia de una sección de cable y densidad de los cables, respectivamente. El parámetro  $\zeta$  a determinar experimentalmente se agregó al modelo para tener en cuenta pérdidas en la vibración, que de lo contrario se presentan modos de resonancia en donde la amplitud diverge.

La fuerza aplicada a cada cable será una fuerza de Lorentz, considerando una corriente  $\vec{I} = I_0 e^{i(\omega t+\phi)} \hat{z}$  se tiene que

$$\vec{f} = \vec{I} \times \vec{B} = I_0 e^{i(\omega t+\phi)} B (-\sin(\theta)\hat{x} + \cos(\theta)\hat{y}) \quad (3)$$

de la que solo consideramos la componente en  $\hat{y}$  que es la única que contribuye a un cambio en el área de la espira.

Las condiciones de contorno para cada cable son entonces:

$$w(0, t) = w(\ell, t) = w_{xx}(0, t) = w_{xx}(\ell, t) = 0 \quad (4)$$

Resolviendo la ecuación 2 se obtiene la frecuencia de los modos de vibración y la deflexión de cada cable:

$$\omega_n = \sqrt{\frac{EI}{\rho A}} \left( \frac{n\pi}{\ell} \right)^2 \quad (5)$$

$$w_n = \sum_{n=1}^{\infty} \frac{4 \sin\left(\frac{n\pi}{\ell}x\right)}{\pi n} \frac{B \cos(\theta)}{\rho \pi r^2} \frac{I_0 e^{i\omega t+\phi}}{\sqrt{(2\omega_n \omega \zeta)^2 + (\omega_n^2 - \omega^2)^2}} \quad (6)$$

donde  $n$  es impar y la respuesta del sistema se desfase con  $\phi = \text{atan}\left(\frac{2\omega_n \omega \zeta}{\omega_n^2 - \omega^2}\right)$  gracias al término de amortiguación. Para obtener el área se integra sobre la longitud del cable y luego se deriva temporalmente para obtener la variación de la misma. Considerando la contribución de ambos cables al área del lazo se tiene:

$$\frac{dA}{dt} = \sum_{n=1}^{\infty} \frac{16}{n^2 \pi^2} \frac{B \ell \cos(\theta)}{\rho \pi r^2} \frac{i\omega I_0 e^{i\omega t+\phi}}{\sqrt{(2\omega_n \omega \zeta)^2 + (\omega_n^2 - \omega^2)^2}} \quad (7)$$

Reemplazando esta expresión en la ecuación 1, definiendo la raíz en el denominador como  $\Omega^2$  y agrupando las constantes en el término

$$F_n = \sum_{n=1}^{\infty} \frac{16}{n^2 \pi^2} \frac{B^2 \ell \cos^2(\theta)}{\rho \pi r^2} \quad (8)$$

se obtiene la fem inducida por la vibración del lazo:

$$\epsilon = -e^{i\phi} \frac{F_n}{\Omega^2} \frac{dI(t)}{dt} \quad (9)$$

donde el término  $\frac{F_n}{\Omega^2}$  tiene unidades de inductancia. Se puede observar que la forma de la ecuación 9 es similar a la de una fem causada por una inductancia, porque depende de la derivada de la corriente. De esta manera, se la puede pensar como una fem producida por una inductancia con dependencia frecuencial  $L(\omega)$ .

Con esta expresión para la fem se puede resolver la ecuación de la malla eléctrica para calcular su contribución al voltaje y la corriente. La ecuación de la malla está dada por:

$$Ve^{i\omega t} - i\omega e^{i\phi} \frac{F_n}{\Omega^2} I(t) - R_{total} I(t) = 0 \quad (10)$$

aquí  $V$  es la tensión dada por el generador,  $\omega$  es la frecuencia de la misma y  $R_{total}$  es la resistencia total del circuito sin contar la muestra. Resolviendo esta ecuación diferencial para  $I(t)$  y tomando parte real se obtiene la corriente del circuito secundario, la que podemos multiplicar por la resistencia de la muestra para tener el voltaje secundario a medir con un lock-in:

$$V(t) = R_{muestra} \frac{V\Omega^4}{(R\Omega^4 + \omega\omega_i^2 F_n)^2 + (\omega\omega_r^2 F_n)^2} [(R\Omega^4 + \omega\omega_i^2 F_n) \cos(\omega t) + \omega\omega_r^2 F_n \sin(\omega t)] \quad (11)$$

con  $\omega_r^2 = 2\omega_n\omega\zeta$  y  $\omega_i^2 = \omega_n^2 - \omega^2$  por como acompañan estos términos a las fases en la ecuación 7.

De esta expresión se puede ver el caso límite, cuando la intensidad del campo magnético tiende a cero se tiene el mismo efecto que un campo apuntando en la dirección perpendicular a la espira:  $B \rightarrow 0$  es igual a  $\cos(\theta = \frac{\pi}{2}) = 0$ . De esto se obtiene  $F_n = 0$ , y el modelo se reduce a la ley de Ohm como corresponde para un circuito puramente resistivo.

## 4. Simulación y medición del fondo angular

Una vez montado el nuevo dispositivo, se logró medir la dependencia angular a temperatura ambiente de la muestra, para distintas frecuencias y valores de campo magnético. Además se simuló el modelo obtenido en la sección 3 variando los parámetros más relevantes y se lo comparó con las mediciones. Las mediciones se realizaron con el modo automático discreto de adquisición, dando pasos de 5 grados y esperando 1 segundo en cada punto, barriendo 180 grados por medición. Se alimentó el circuito con tensión AC de 0,5V de amplitud y se midió la tensión resultante utilizando un lock-in con un tiempo de integración de 300ms.

Para estudiar la dependencia del fondo angular con el valor campo magnético se fijó la frecuencia de la tensión en  $\omega = 1010\text{Hz}$ , que en un barrido preliminar se determinó como el máximo de la señal de contra fase, y se midió la respuesta para distintos valores de  $B$ . Los resultados se muestran en la figura 14 junto a una simulación del modelo para la misma frecuencia.

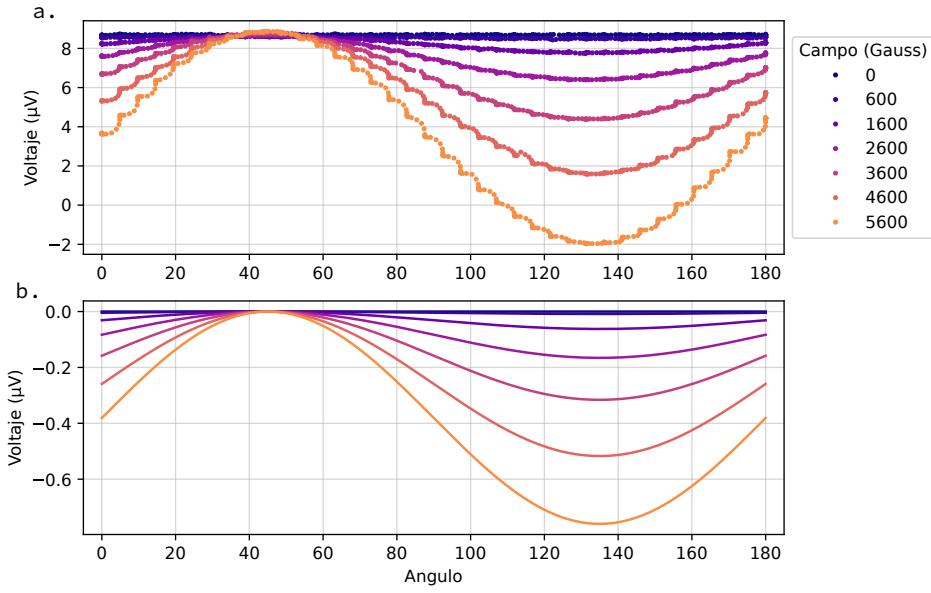


Figura 14: Dependencia angular de la señal de contra fase con tensión aplicada a frecuencia fija de  $\omega = 1010\text{Hz}$  y variando el campo magnético, obtenida de forma experimental (panel a.) y mediante simulaciones numéricas (panel b.). Se barrieron 180 grados, con pasos de 5 grados.

Se puede ver que mas allá de variaciones entre el offset y la amplitud de la señal, tanto las mediciones como el modelo tienen la misma dependencia angular. Luego se ajustaron estas curvas con funciones senoidales para obtener sus amplitudes, que se grafican en función del campo magnético  $B$  en la figura 15. Ajustando los datos se corroboró que la señal tiene dependencia cuadrática con el valor de  $B$ , como indica la ecuación 8 del modelo.

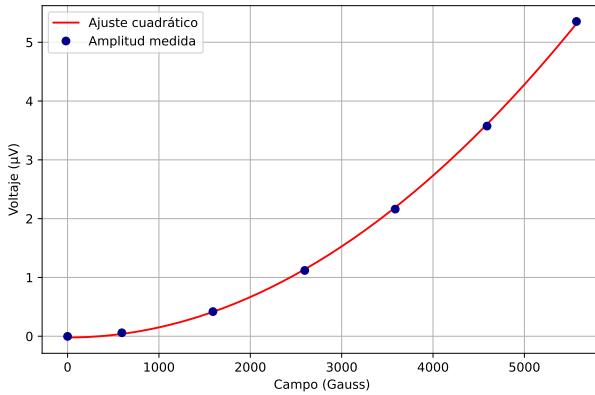


Figura 15: Comportamiento cuadrático de la amplitud del fondo angular en función de  $B$ , para frecuencia fija de  $\omega = 1010\text{Hz}$ .

Por otro lado se midió la dependencia angular de la muestra a campo fijo  $B = 5570G$  para distintas frecuencias y se tomó su amplitud como se mencionó anteriormente. En la figura 16 se muestran las amplitudes en función de la frecuencia junto a la simulación correspondiente.

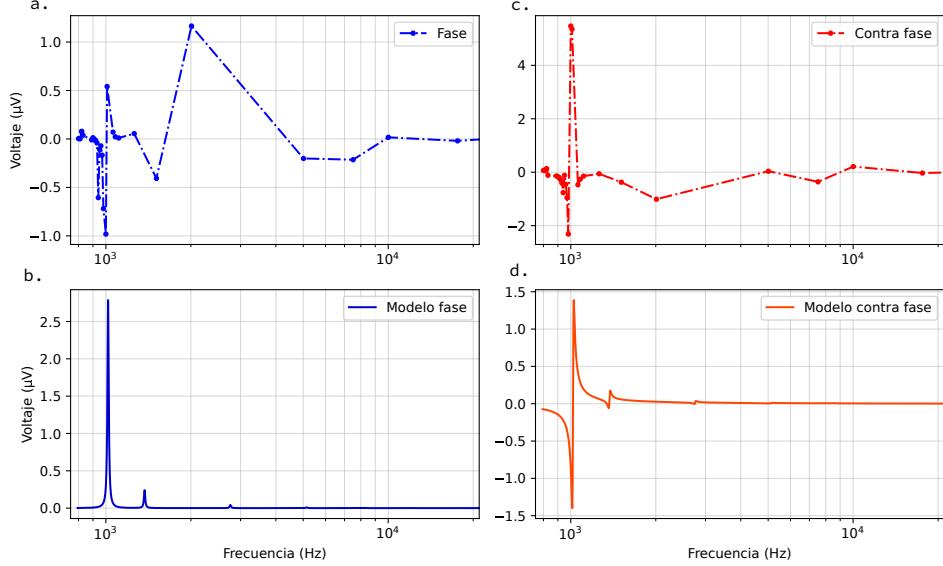


Figura 16: Barrido en frecuencias del fondo angular. En la fila superior se muestran las mediciones de la señal de fase (panel a.) , y la contra fase (panel c.) . En la fila inferior se ven las simulaciones correspondientes (b. d.). No se observó un fondo de dependencia angular en frecuencias mayores a 20kHz.

En la figura 16 se puede observar en las mediciones la presencia de distintos picos de resonancia, en particular, el pico principal se halla en  $\omega = 1010\text{Hz}$ . Se detectaron picos secundarios en el rango de frecuencias estudiado, este comportamiento es similar al que predice el modelo. Sin embargo, si se compara con la simulación de la figura 17, se puede ver que las frecuencias no coinciden para los modos superiores. Esto es esperable, ya que se trata un modelo idealizado donde se considera que los cables tienen la misma longitud, y como se ve en la ecuación 5, este parámetro es un factor determinante en la frecuencia de resonancia y los modos superiores.

Para realizar esta simulación se tuvieron que decidir valores  $\ell$  y  $\zeta$  apropiados. Para el primero se simularon las frecuencias naturales para 3 modos distintos de resonancia en función de la longitud de los cables (figura 17) utilizando la ecuación 5. Eligiendo la longitud que corresponde a una frecuencia de resonancia de  $\omega = 1010\text{Hz}$  se obtuvo  $\ell = 13,2\text{mm}$ . De una manera similar se eligió el  $\zeta$  que mejor reproduce el ancho de la campana principal, con un valor de  $\zeta = 0,006$ .

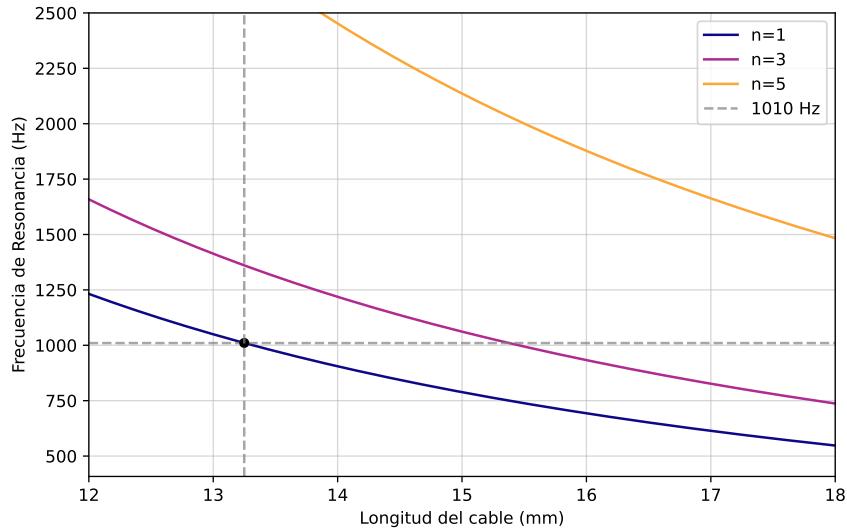


Figura 17: Frecuencia de resonancia para modos impares en función de la longitud de los cables. Se indica el corte entre la frecuencia pico medida y el primer modo, para una longitud de  $\ell = 13,2\text{mm}$ .

#### 4.1. Discusión

El modelo que se propone es consistente con algunas características del fondo angular observado. En primer lugar, logra explicar la dependencia angular de la tensión y predice una dependencia cuadrática con el campo magnético, resultados que se pueden ver en las figuras 14 Y 15. Esta dependencia respalda la idea de un fenómeno inductivo, donde el campo contribuye tanto a la fem como a la vibración de los cables.

El modelo también da una posible explicación para la dependencia en frecuencias de este fondo, aunque no logra predecir con exactitud las frecuencias de los picos secundarios. Esto es esperable si se tiene en cuenta que el modelo es una idealización, donde se consideró que los cables tienen el mismo largo y son paralelos, dos factores que pueden variar estas frecuencias, como indica la ecuación 5. Por otro lado, las longitudes y dimensiones características necesarias para que la frecuencia de resonancia del modelo coincida con la experimental son muy razonables.

También explica la presencia de una contra fase en un circuito que debería ser resistivo, aunque en las mediciones se observó que la tensión en contra fase siempre es mayor a la fase, lo cual contradice los resultados de la simulación como se ve en la figura 16. Puede contribuir a este comportamiento un offset de fase en el lock-in que esté mezclando las señales, de manera que las componentes medidas no están en fase con las componentes simuladas.

## 5. Conclusiones

En este trabajo se diseñó y ensambló un dispositivo mecánico para rotar el electroimán del Laboratorio de Bajas Temperaturas utilizando componentes económicos y piezas fabricadas por el equipo y en el taller de la facultad. Se implementó además un software para controlar y automatizar las mediciones, que incluye rutinas de calibración y mecanismos de seguridad, con una interfaz gráfica para su uso desde la computadora, y que se integra con el software de LabVIEW ya usado en el equipo. También se fabricó un gabinete para contener los componentes electrónicos del equipo, se realizaron todas las conexiones y se montaron en el frente pulsadores y leds para controlar el dispositivo sin necesidad de conectarlo a la PC.

Por otro lado, se interpretaron mediciones existentes de una problemática que se presentaba en el laboratorio y se propuso un modelo físico que explique éstas señales, tomando como punto de partida la idea de ruidos microfónicos causados por un lazo que formar los cables de oro conectados a la muestra. Se realizaron simulaciones de este modelo y se contrastaron con mediciones que se pudieron hacer con el nuevo dispositivo.

El modelo logró explicar características del fondo como su dependencia cuadrática con el valor del campo magnético, su dependencia angular y la existencia de una señal de contra fase. Además dió una buena aproximación de la frecuencia donde se espera la mayor amplitud de este fondo teniendo en cuenta la longitud de los cables instalados en el dispositivo. Sin embargo, no logra predecir con exactitud los picos secundarios de resonancia ni la amplitud de la señal.

Finalmente, las mediciones parecen respaldar la idea de un fenómeno inductivo, y con este modelo se puede aproximar un rango de frecuencias donde el fondo puede ser significativo, lo que resultará útil a la hora de elegir intervalos de frecuencias para experimentos.

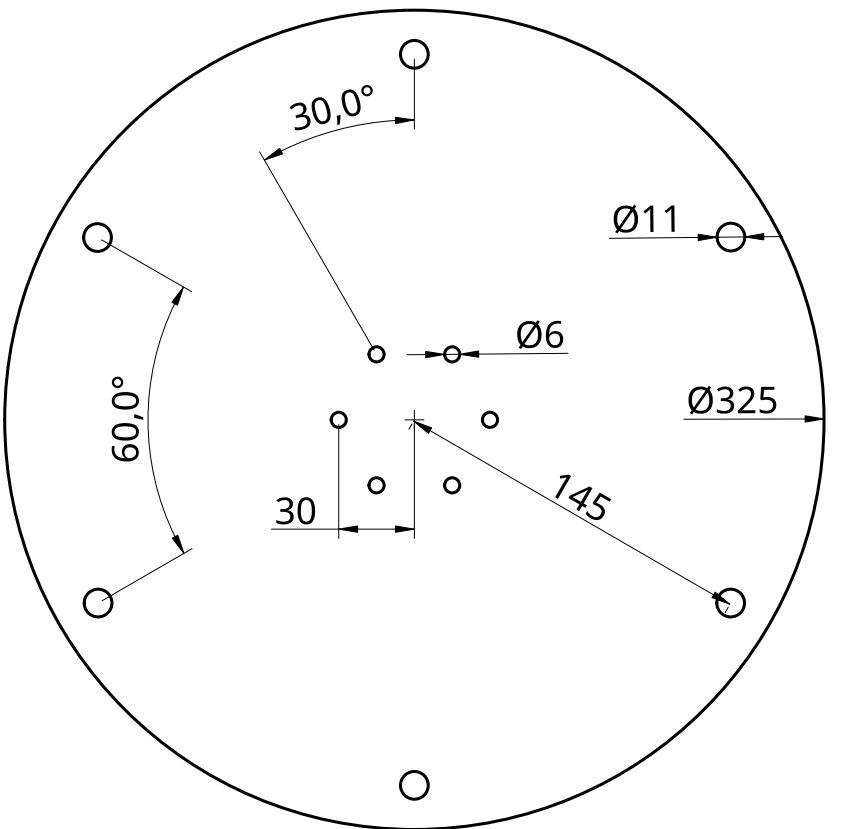
Se agradece enormemente por la ayuda en el proyecto a nuestra directora Dra Gabriela Pasquini, a Maxi por fabricar las piezas en el taller de la facultad, y a Diego y Dante por sus inmensos aportes en la parte técnica y del armado del equipo.

## Referencias

- [1] G. Pasquini *et al* J. Schmidt. Nematicity in the superconducting mixed state of strain detwinned under-doped  $Ba(Fe_{1-x}Co_x)_2As_2$ . *Physical Review B*, 99(6), feb 2019.
- [2] J. Schmidt. Fase nemática en  $Ba(Fe_{1-x}Co_x)_2As_2$  : alineación de dominios y elasto-resistividad, 2018.
- [3] *GMW Model users manual, 3472-50 76 mm electromagnet*, 1990.
- [4] Erik Oberg and Christopher J. McCauley. *Machinery's Handbook*. Industrial Press, Inc., 30 edition, 2016.
- [5] Daniel J. Inman. *Engineering vibration*, pages 533–563. Pearson, 4 edition, 2014.

# **Apéndices**

## **A. Planos del diseño mecánico**



UNLESS OTHERWISE SPECIFIED,  
DIMENSIONS ARE IN MILLIMETERS

ANGULAR =  $\pm$  °

SURFACE FINISH ✓

DO NOT SCALE DRAWING

BREAK ALL SHARP EDGES AND  
REMOVE BURRS

FIRST ANGLE PROJECTION



NAME

MARCO CRIVARO  
NICOLINI

SIGNATURE

DATE

2021-07-07

TITLE

PLACA ACOPLE

SIZE

A4

DWG NO.

REV.  
1

MATERIAL

SAE 1010, 1/4

FINISH

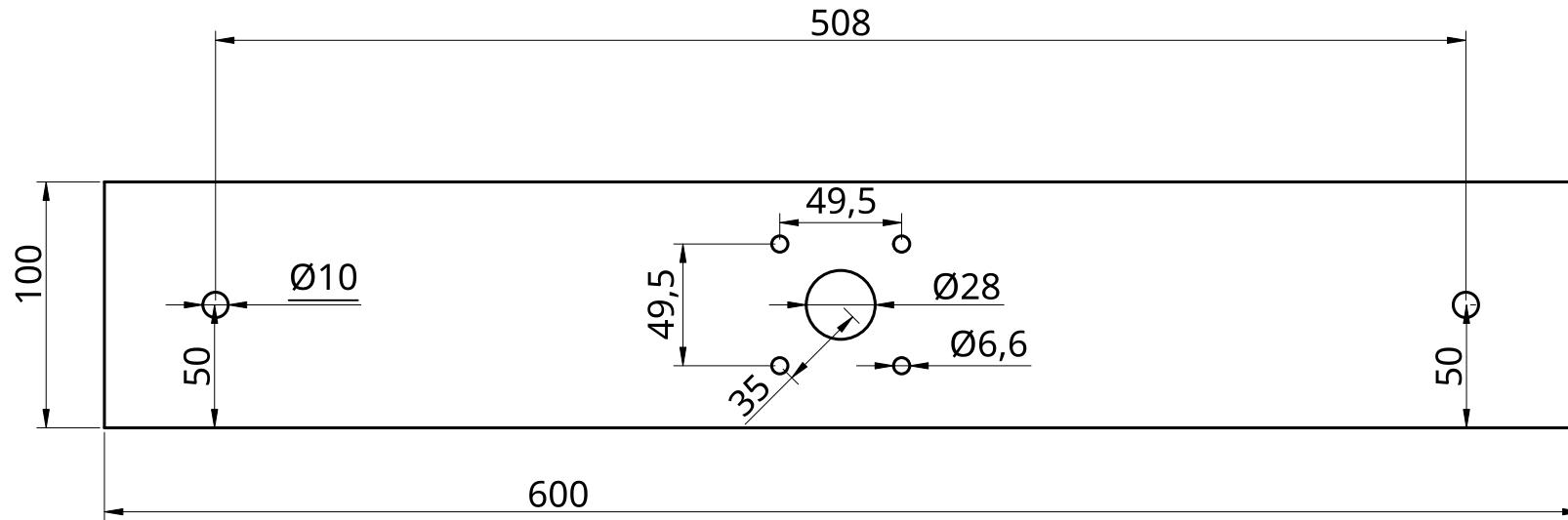
SCALE

1:3

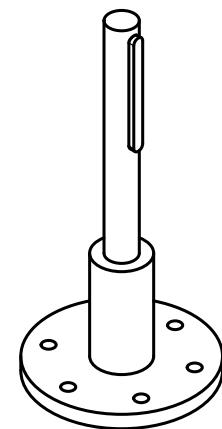
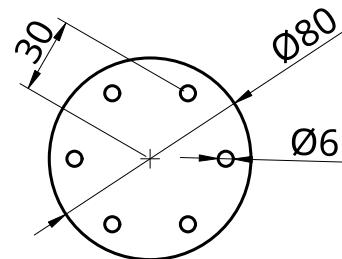
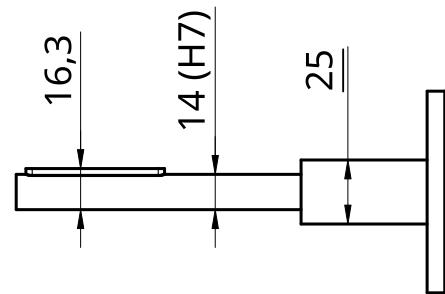
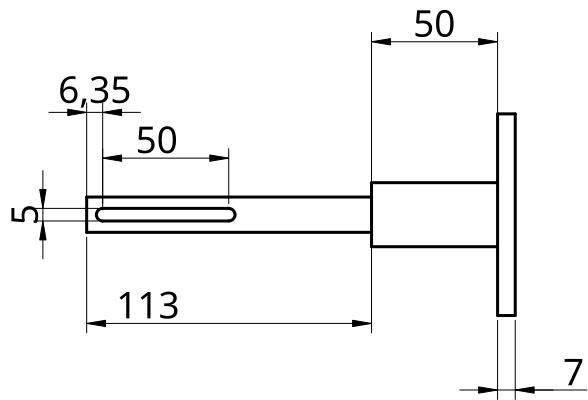
WEIGHT

SHEET  
1 of 2

21



UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS					NAME	SIGNATURE	DATE	
ANGULAR = $\pm$ °		DRAWN	MARCO CRIVARO NICOLINI				2021-07-07	
SURFACE FINISH $\checkmark$		CHECKED						
DO NOT SCALE DRAWING		APPROVED						
BREAK ALL SHARP EDGES AND REMOVE BURRS								
FIRST ANGLE PROJECTION		MATERIAL		FINISH				
		SIZE	A4	DWG NO.				REV. 1
		SCALE	1:3	WEIGHT				SHEET 2 of 2



UNLESS OTHERWISE SPECIFIED,  
DIMENSIONS ARE IN MILLIMETERS

ANGULAR =  $\pm$  °

SURFACE FINISH ✓

DO NOT SCALE DRAWING

BREAK ALL SHARP EDGES AND  
REMOVE BURRS

FIRST ANGLE PROJECTION



NAME

MARCO CRIVARO  
NICOLINI

SIGNATURE

DATE

2021-07-13

TITLE

PUNTA DE EJE

SIZE

A4

DWG NO.

1

REV.  
1

MATERIAL

SAE 1045

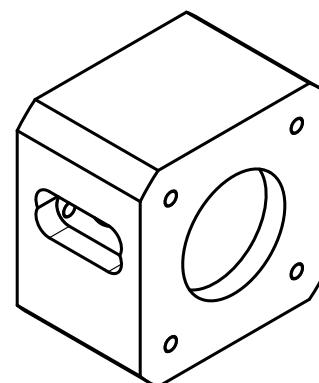
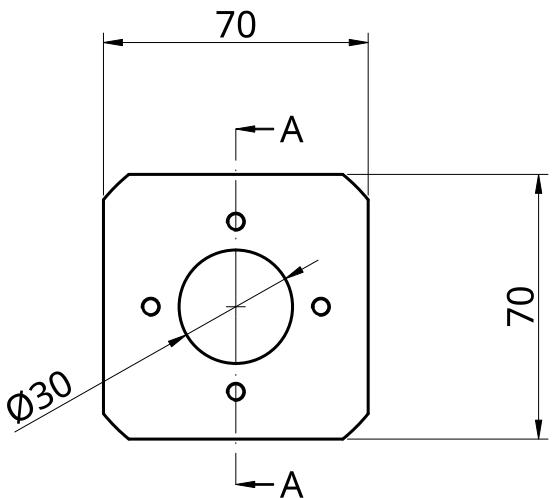
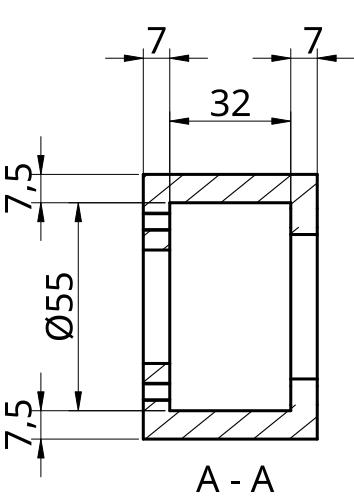
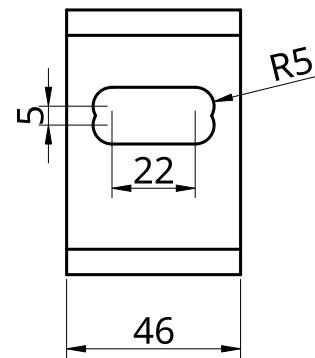
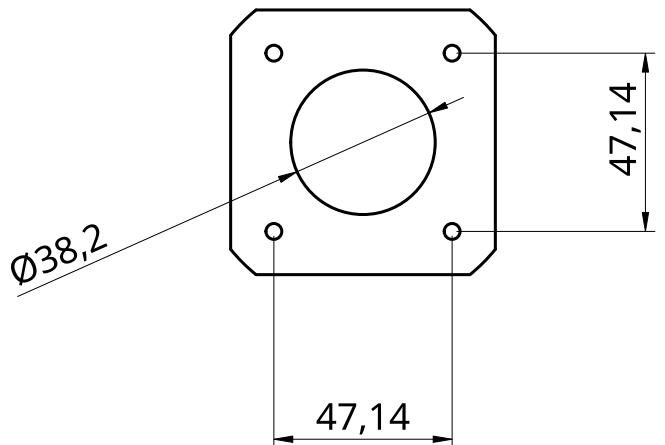
FINISH

SCALE

1:3

WEIGHT

SHEET  
1 of 1



Ahuecado completo es opcional, puede ser un agujero de 38.2mm hasta la otra pared mientras que la ventana del costado permita llegar a los tornillos del acople de los ejes.

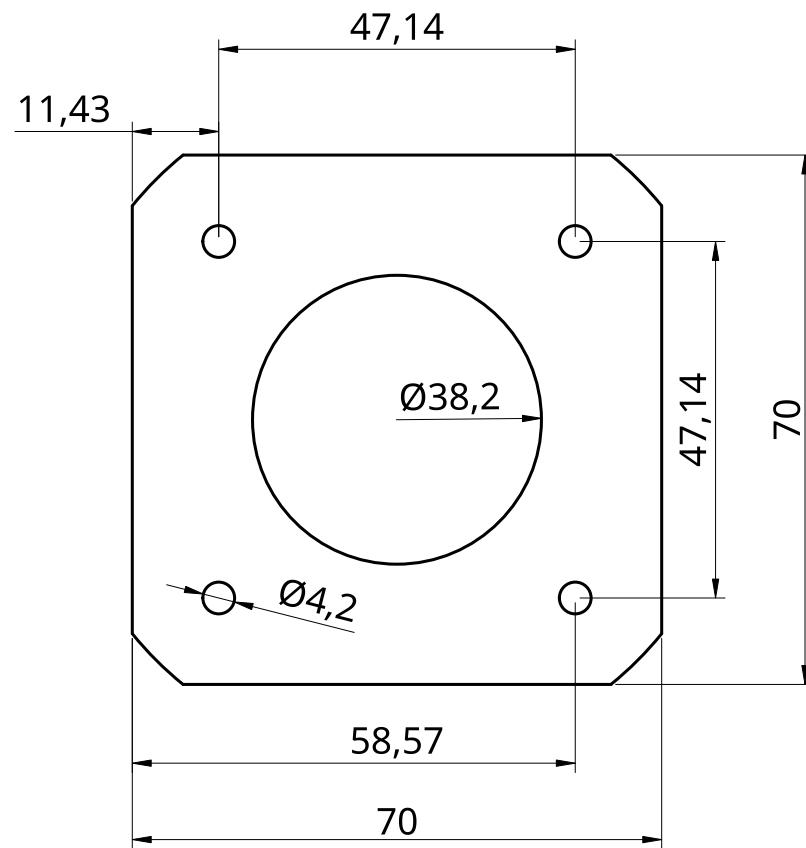
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS	
ANGULAR = $\pm 0^\circ$	
SURFACE FINISH $\checkmark$	
DO NOT SCALE DRAWING	
BREAK ALL SHARP EDGES AND REMOVE BURRS	
FIRST ANGLE PROJECTION	

MATERIAL	FINISH
ALUMINIO 6061	

SIZE	DWG NO.	ACOPLE SINFIN MOTOR		REV.
		1	1	
SCALE	1:2	WEIGHT	350g	SHEET
			1 of 3	

FRENTE (LADO MOTOR)

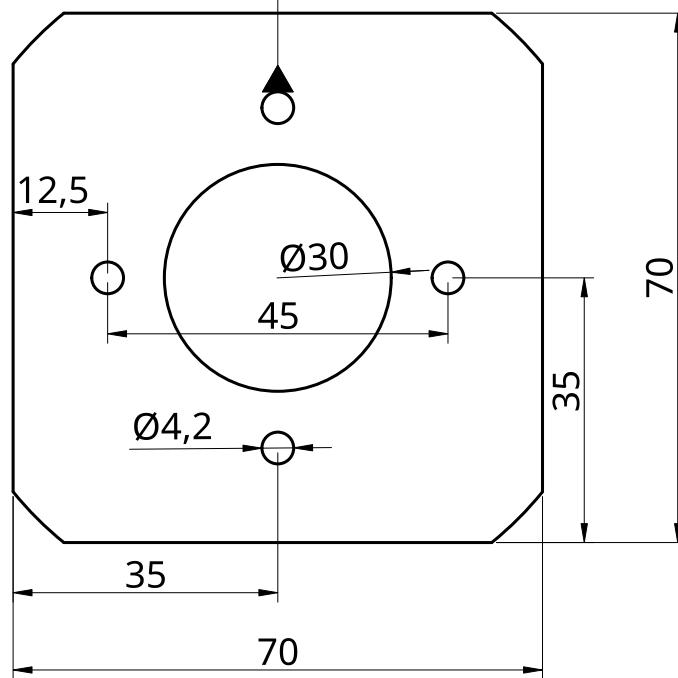
4 agujeros roscados  
para tornillo M5



## REVERSO (LADO SIN FIN)

A

Agujeros del tamaño del tornillo torx que está puesto en el sin fin.



## B. Código de Arduino

### B.1. Declaración de variables

```
//Incluyo librerías y archivos de clases
#include <AccelStepper.h> //librería para control de steppers
#include <digitalPinFast.h> //librería para mejorar la velocidad de pines
→ digitales
#include <PushButton.h> //clase que armamos controlar los botones

//Defino los pines como constantes que ocupan 1 byte de memoria
const byte homeSwitch = 2,
          endSwitch = 3,
          dirPin = 4,
          pulsePin = 5,
          rightPin = 6,
          leftPin = 7,
          stopPin = 8,
          homePin = 9,
          greenPin = 10,
          redPin = 11;

//Creo objetos: motor, botones y leds.
//El primer argumento indica que estamos usando un driver para controlar el
→ motor.
AccelStepper stepper(1, pulsePin, dirPin);

//Elegí un delay de 50ms para filtrar el ruido de los botones y microswitches
const long debounceDelay = 50;

//Inicializamos los objetos pulsadores (ver sección clase PushButton)
PushButton rightButton(rightPin, false, debounceDelay);
PushButton leftButton(leftPin, false, debounceDelay);
PushButton stopButton(stopPin, true, debounceDelay);
PushButton homeButton(homePin, true, debounceDelay);
PushButton homeMicroswitch(homeSwitch, false, debounceDelay);
PushButton endMicroswitch(endSwitch, false, debounceDelay);

//Inicializamos los objetos de pin rápido
digitalPinFast greenLed(greenPin);
digitalPinFast redLed(redPin);
```

```

//Variables lógicas para habilitar y deshabilitar pulsadores o indicar estados
// del programa
volatile bool rightButtonEnabled = true; //volatile porque los interrupts la
// pueden modificar
volatile bool leftButtonEnabled = true;
volatile bool interrupted = false;
bool homing = false;

//La tasa de comunicación por USB
long baudRate = 115200;

//Variables para movimiento automático

//Estados que puede tomar el programa cuando está en modo automático
enum states
{
    autoDisabled, goingToStart, waitingStart, goingToFinish, waitingFinish,
    // canceled
};

volatile states state = autoDisabled; //volatile porque los interrupts la
// pueden modificar

//Armamos un tipo de variable Recorrido para contener todos los parámetros del
// recorrido automático
typedef struct {
    long start;
    long finish;
    int total_loops;
    int speed_ida;
    int speed_vuelta;
} Recorrido;

Recorrido recorrido;
const long wait = 1000; //el tiempo de espera en cada punta

//=====
//Estados que puede tomar el programa cuando está en modo automático discreto
enum states2
{
    stepping, waiting, stepDisabled
};

```

```

volatile states2 state2 = stepDisabled;

long total_lenght = 0;
long step_lenght = 330;

//=====
//Variables para calibración
//=====
//Variable para guardar el juego angular
long backlash = 0;

//Variables para registrar la dirección
bool directionHasChanged = false;
bool skipNext = false;

enum dirección {IZQ, DER, DET};
dirección direc = DER;

//=====
// Variables para leer Serial
//=====

// Si entraron datos por serial o no
bool newData = false;

//Acá se van a ir guardando los datos a medida que los recibe y antes de
//interpretarlos
const byte numChars = 32;
char receivedChars[numChars];
char tempChars[numChars];

//Armamos un tipo de variable Message para contener todos los parámetros del
//mensaje una vez que lo interpreta
typedef struct {
    String command;
    int integer_input;
    long long_input;
} Message;

```

## B.2. Rutinas de interrupción

```

//Microswitch de home
void ISRhome() {

```

```

if (!digitalRead(homeSwitch)) {
    if (!homming && (state != waitingStart)) {
        stepper.stop();
        state2 = stepDisabled;
    }
    redLed.digitalWriteFast(HIGH);
    leftButtonEnabled = false;
    interrupted = true;
}
else {
    redLed.digitalWriteFast(LOW);
    leftButtonEnabled = true;
    interrupted = false;
}
}

//Microswitch de fin de recorrido
void ISRend() {
    if (!digitalRead(endSwitch)) {
        stepper.stop();
        state = canceled;
        state2 = stepDisabled;
        redLed.digitalWriteFast(HIGH);
        rightButtonEnabled = false;
        interrupted = true;
    }
    else {
        redLed.digitalWriteFast(LOW);
        rightButtonEnabled = true;
        interrupted = false;
    }
}

```

### B.3. Setup

```

void setup() {
    Serial.begin(baudRate); //abrimos el puerto USB

    stepper.setMaxSpeed(400); //configuramos velocidad y aceleración maximas
    stepper.setAcceleration(150);

    //configuramos todos los pines
    greenLed.pinModeFast(OUTPUT);

```

```

redLed.pinModeFast(OUTPUT);

homeButton.setPin();
stopButton.setPin();
leftButton.setPin();
rightButton.setPin();
homeMicroswitch.setPin();
endMicroswitch.setPin();

//agregamos rutinas de interrupción
attachInterrupt(digitalPinToInterruption(homeSwitch), ISRhome, CHANGE);
attachInterrupt(digitalPinToInterruption(endSwitch), ISREnd, CHANGE);

//indicamos encendido prendiendo los leds, 3 veces en un intervalo de 300 ms
blinkLeds(3, 300);
}

```

#### B.4. Ciclo principal

```

void loop() {

readserial(); //se fija si llegaron mensajes
if (newData == true) {
    strcpy(tempChars, receivedChars);
    Message newMessage = parseData(); //interpreta los datos y los guarda en un
    → mensaje
    newData = false;
    serialCommands(newMessage); //ejecuta la función que corresponde al mensaje
}

buttonCommands(); //lee estado de pulsadores y ejecuta funciones asociadas
automed(); //modos automáticos
autostep();

updateDirection(); //actualiza la dirección
adjustBacklash(); //corrige backlash si es necesario

stepper.run(); //da un paso

//esta parte del código maneja el led verde que se enciende si el motor está
→ activo
//y se encarga de desactivar o activar pulsadores si es necesario
static bool ledOn;
}

```

```

if (stepper.isRunning() && !ledOn) {
    greenLed.digitalWriteFast(HIGH);
    ledOn = true;
}
else {
    if (ledOn) {
        greenLed.digitalWriteFast(LOW);
        ledOn = false;
    }
    if (!interrupted) && (state == autoDisabled) {
        rightButtonEnabled = true;
        leftButtonEnabled = true;
    }
}
}

```

## B.5. Homing automático

```

void gohome() {
    homing = true;
    greenLed.digitalWriteFast(HIGH);

    //avanza de a 133 pasos (se traducen en aprox. 2 grados) hasta activar el
    // microswitch
    while (!homeMicroswitch.isOn()) {
        stepper.move(-133);
        stepper.run();
    }

    //se activó, define esa posición como 0
    stepper.setCurrentPosition(0);

    //retrocede de a 20 pasos hasta que desactiva el microswitch
    while (homeMicroswitch.isOn()) {
        stepper.move(20);
        stepper.run();
    }

    //guardamos los pasos que dió como backlash y redefinimos la posición de
    // referencia
    backlash = stepper.currentPosition();
    stepper.setCurrentPosition(0);
    greenLed.digitalWriteFast(LOW);
}

```

```

//Para que no interfiera con la corrección de backlash
if (direc == IZQ) {
    skipNext = true;
}
direc = DER;
homing = false;
}

```

## B.6. Automatización

```

//Función de recorrido de ida y vuelta automático
//Es una máquina de estados que siempre está corriendo en el código,
//pero según el estado del programa tiene comportamientos distintos
void automed() {

    //Variables para registrar el tiempo y la cantidad de vueltas
    static unsigned long startMillis;
    static unsigned long stopMillis;
    static int current_loop;

    switch (state) {

        case autoDisabled:
            //si está deshabilitado no hace nada
            break;

        case canceled:
            //si se cancela resetea contadores y pulsadores antes de pasar a disabled
            //es por una cuestión de seguridad
            current_loop = 0;
            rightButtonEnabled = true;
            leftButtonEnabled = true;
            state = autoDisabled; //cambia de estado
            break;

        case goingToStart:
            //Si está "yendo al principio" espera hasta llegar al objetivo
            //y pone un contador de tiempo, actualiza la cantidad de vueltas
            if (stepper.distanceToGo() == 0) {
                startMillis = millis();
                current_loop++;
                if (current_loop > recorrido.total_loops) {

```

```

        state = autoDisabled;
        current_loop = 0;
        rightButtonEnabled = true;
        leftButtonEnabled = true;
    }
    else {
        state = waitingStart; //cambia de estado
    }
}
break;

case waitingStart:
//Está esperando en el inicio un segundo
if (millis() - startMillis > wait) {
    stepper.moveTo(recorrido.finish);
    direc = getDirection();
    stepper.setMaxSpeed(recorrido.speed_ida);
    state = goingToFinish; //cuando pasó un segundo cambia de estado
}
break;

case goingToFinish:
//similar a "yendo al principio" pero un poco más simple
if (stepper.distanceToGo() == 0) {
    stopMillis = millis();
    state = waitingFinish;
}
break;

case waitingFinish:
//análogo a esperando en el principio
if (millis() - stopMillis > wait) {
    stepper.moveTo(recorrido.start);
    direc = getDirection(); //observar que siempre actualizamos dirección
    stepper.setMaxSpeed(recorrido.speed_vuelta);
    state = goingToStart;
}
break;
}

//Función de recorrido automático de a pasos

```

```

void autostep() {

    //variables para registrar el tiempo y los pasos dados
    static unsigned long startMillis;
    static long current_step;

    switch (state2) {

        case stepDisabled:
            break;

        case stepping:
            //si está dando pasos, espera hasta el objetivo
            if (stepper.distanceToGo() == 0) {
                startMillis = millis();
                current_step += step_lenght;
                if (current_step > total_lenght) {
                    state2 = stepDisabled; //si completó todos los pasos deshabilita
                    current_step = 0;
                }
                else {
                    state2 = waiting; //si faltan pasos pasa a esperar
                }
            }
            break;

        case waiting:
            //está esperando 1 segundo
            if (millis() - startMillis > wait) {
                stepper.move(step_lenght);
                direc = getDirection();
                state2 = stepping; //si pasó un segundo vuelve a dar pasos
            }
            break;
    }
}

```

## B.7. Comandos serial

```

//Esta función toma la variable mensaje y evalúa el comando contra todos los
//comandos definidos para ver que función tiene que correr
//Siempre envía una respuesta para cumplir el protocolo de command-response
void serialCommands(Message message) {

```

```

//Evalúa el comando contra todos los comandos definidos posibles
//y corre las funciones que correspondan
//incluimos solo algunos ejemplos porque el resto de la función
//es muy similar
if (message.command == "MES") {
    Serial.println(stepper.currentPosition());
}
else if (message.command == "STP") {
    Serial.println("STP");
    stepper.stop();
    state = canceled;
    state2 = stepDisabled;
}
else if (message.command == "SET") {
    Serial.println("SET");
    rightButtonEnabled = false;
    leftButtonEnabled = false;
    stepper.setMaxSpeed(message.integer_input);
    stepper.moveTo(message.long_input);
    direc = getDirection();
    ....
}
Se omitieron partes del código redundantes
.....
else {
    //si el comando no coincide con ninguno, se responde NAK (not acknowledged)
    Serial.println("NAK");
}
}

```

## B.8. Comandos de pulsadores

```

//Esta función lee el estado de los botones ayudandose con los métodos de la
//clase PushButton y define que acciones
//tienen que ejecutarse si se activan
void buttonCommands() {
    if (stopButton.isOn()) {
        stepper.stop();
        state = canceled;
        state2 = stepDisabled;
    }
    else if (rightButtonEnabled && rightButton.isOn()) {
        stepper.move(133);
        direc = getDirection();
    }
}

```

```

    }
    else if (leftButtonEnabled && leftButton.isOn()) {
        stepper.move(-133);
        direc = getDirection();
    }
    else if (homeButton.isOn()) {
        gohome();
    }
}

```

## B.9. Comunicación por serial

```

//Esta función lee el puerto USB y va guardando los datos en un array
void readserial() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<'; //definimos los caracteres de principio y fin
    char endMarker = '>';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (recvInProgress == true) {
            //vamos agregandolos a un array letra por letra
            if (rc != endMarker) {
                receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= numChars) {
                    ndx = numChars - 1;
                }
            }
            else {
                //si detectamos un fin de linea terminamos el array
                receivedChars[ndx] = '\0';
                recvInProgress = false;
                ndx = 0;
                newData = true;
            }
        }
        else if (rc == startMarker) {
            recvInProgress = true;
        }
    }
}

```

```

        }
    }
}

// Esta función detecta las comas que separan cada parámetro y guarda cada parte
// en una variable del tipo mensaje
// convierte los datos guardados, que son caracteres, a variables numéricas del
// tipo integer
Message parseData() {
    Message newMessage;
    char received[numChars] = {0};

    char * strtokIndx; //lo usa strtok() como índice

    strtokIndx = strtok(tempChars, ",");      // toma hasta la primer parte, el
    // string
    strcpy(received, strtokIndx); // y lo copia en received
    newMessage.command = String(received); //lo asignamos al miembro command del
    // mensaje

    strtokIndx = strtok(NULL, ",");
    newMessage.integer_input = atoi(strtokIndx);

    strtokIndx = strtok(NULL, ",");
    newMessage.long_input = atol(strtokIndx); //idem pero con el long

    return newMessage;
}

```

## B.10. Calibración y corrección de *backlash*

```

//Esta función calcula la dirección actual del motor
//Hay que correrla cada vez que se usa un comando que cambie la posición
//objetivo
dirección getDirection() {
    if (stepper.distanceToGo() > 0) {
        return DER;
    }
    else if (stepper.distanceToGo() < 0) {
        return IZQ;
    }
    else if (stepper.distanceToGo() == 0) {
        return DET;
    }
}

```

```

    }
}

//Esta función se fija si la dirección cambió y avisa al resto del código si hay
→ que corregirla o no
void updateDirection() {
    static dirección old_direc = DER; //la primer corrección no se va a hacer
    → porque no tiene calibrado el backlash así que no importa esto

    if ((direc != old_direc) && (direc != DET)) { //que no considere detenido como
        → un cambio de dirección
        //Serial.println("Direction has changed!");
        directionHasChanged = true;
        old_direc = direc;
    }
    else {
        directionHasChanged = false;
    }
}

//Si la dirección cambió, esta función hace que el motor de los pasos necesarios
→ para corregir el backlash
//antes de ir a la posición objetivo que se pidió
//una vez que corrigió, redefine la posición actual y la de objetivo
void adjustBacklash() {
    if (directionHasChanged) {
        if (skipNext) {
            skipNext = false;
        }
        else {
            long target = stepper.targetPosition();
            long current = stepper.currentPosition();
            if (direc == DER) {
                stepper.move(backlash); //muestra -backlash- pasos positivos/derecha
            }
            else if (direc == IZQ) {
                stepper.move(-backlash); //muestra -backlash- pasos negativos/izquierda
            }
            stepper.runToPosition();
            stepper.setCurrentPosition(current);
            stepper.moveTo(target);
        }
    }
}

```

```
    }  
}
```

### B.11. Clase para el manejo de pulsadores

```
//Esto es el header file, donde definimos los miembros y métodos de la clase  
//sin implementarlos  
//Distinguimos entre pulsadores "continuos" y de cambio de estado, la diferencia  
//es que los segundos solo queremos que se activen una vez, por ejemplo los de  
→ home y stop  
//los primeros queremos que se activen siempre que estan pulsados, como derecha  
→ e izquierda  
class PushButton  
{  
    //miembros publicos:  
    public:  
        PushButton(byte pin, bool stateChange, unsigned long debounceDelay);  
        //constructor  
        void setPin(); //método para inicializar el pin  
        void readAndDebounce(); //método para filtrar ruidos  
        void readAndDebounce_stateChange(); //versión para interruptor  
        bool isOn();  
  
    //miembros privados, no queremos que se modifiquen en el código principal  
    private:  
        byte _pin;  
        byte _state = HIGH;  
        byte _previousState;  
        unsigned long _lastDebounceTime = 0;  
        unsigned long _debounceDelay;  
        bool _stateChange;  
        bool _toggled = false;  
};  
  
//fin del header  
  
//constructor  
PushButton::PushButton(byte pin, bool stateChange, unsigned long debounceDelay)  
{  
    _pin = pin;  
    _debounceDelay = debounceDelay;  
    _stateChange = stateChange;  
    _state = HIGH;
```

```

}

//método para configurar pin
void PushButton::setPin()
{
    pinMode(_pin, INPUT_PULLUP);
}

//método para filtrar ruidos
void PushButton::readAndDebounce() {

    byte reading = digitalRead(_pin);

    if (reading != _previousState) { //si cambió el estado del pin (voltaje cambia
        ↵ entre HIGH y LOW)
        _lastDebounceTime = millis(); //guardamos el tiempo donde pasó
    }

    //solo lo tenemos en cuenta si la variación pasa en tiempos mayores al delay
    ↵ que definimos
    if ((millis() - _lastDebounceTime) >= _debounceDelay) {
        _state = reading;
    }

    _previousState = reading;
}

//Versión para cambio de estado
void PushButton::readAndDebounce_stateChange() {

    byte reading = digitalRead(_pin);

    if (reading != _previousState) {
        _lastDebounceTime = millis();
    }

    //Análogo al anterior, pero solo importa la primera vez que cambia el estado
    if (_state != reading) && ((millis() - _lastDebounceTime) >= _debounceDelay)
        ↵ {
            _state = reading;

    if (_state == LOW) {

```

```

        _toggled = true;
    }
}
else {
    _toggled = false;
}

_previousState = reading;
}

//Una vez que filtramos el ruido, redefinimos que significa que el pulsador
//esté activado
bool PushButton::isOn() {

    if (_stateChange) {
        readAndDebounce_stateChange();
        return _toggled;
    }
    else {
        readAndDebounce();
        return (_state == LOW);
    }
}

```