UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Final de Grado

Diseño y realización de un controlador de motores paso a paso aplicado a un microposicionador de tres ejes de alta precisión

Ingeniería Electrónica Industrial y Automática
Curso Académico: 2020/2021

AUTOR: SANTIAGO JUNQUERA CARRERO
TUTOR: SALVADOR PONCE ALCÁNTARA
COTUTOR: JAIME GARCÍA RUPÉREZ

RESUMEN:

Los laboratorios del Centro de Tecnología Nanofotónica de Valencia disponen de múltiples herramientas para el posicionamiento de elementos, como por ejemplo microposicionadores o goniómetros. Estas herramientas son de gran utilidad en experimentos que requieren modificar la posición de una muestra o de un sensor para monitorizar cambios en las mediciones realizadas. La mayor parte de estos elementos cuentan con un sistema de posicionamiento manual, por lo que se dificulta la tarea de obtener movimientos precisos y repetibles. Para el caso específico de este proyecto, se propone implementar una solución de bajo coste para motorizar un microposionador manual de tres ejes.

El presente proyecto aborda la creación de un controlador de motores paso a paso de tres canales, así como también la elección de los motores y el sistema que accionará los ejes del microposionador. El controlador de motores se ha diseñado específicamente para este proyecto, integrando todos los componentes necesarios en un circuito impreso para proporcionar una solución sencilla de utilizar. El diseño hardware incorpora las protecciones necesarias para dotar de robustez al circuito, por ejemplo: protección contra sobrevoltajes, polaridad inversa, señales de control aisladas por optoacoplador y referencias digital y analógica separadas.

También se ha desarrollado el firmware que se carga en el controlador, así como una interfaz gráfica de usuario que permite el control de los motores desde el ordenador. Se trata, por lo tanto, de un diseño personalizado en todos sus niveles (software, firmware, hardware) que proporciona una solución compacta para el control de motores aplicado al posicionamiento de elementos en un laboratorio.

PALABRAS CLAVE:

Microposicionador, Motor paso a paso, Controlador de motores, QT Creator, Circuito impreso

ABSTRACT:

Valencia Nanophotonics Technology Center's labs have multiple tools for positioning elements, such as micro-positioners or goniometers. These tools are very useful in experiments that require changing the position of a sample or a sensor to check variations in the measurements. Most have a manual positioning system, making it difficult to obtain precise and repeatable movements. For the specific case of this project, it is proposed to implement a low-cost solution to motorize a manual three-axis micropositioner.

This project addresses the creation of a three-channel stepper motor controller, as well as the possibility to choose the motors and the system that will drive the axes of the micropositioner. The motor controller has been specifically designed for this project, integrating all the necessary components on a printed circuit to provide an easy-to-use solution. The hardware design incorporates the necessary protections to provide robustness to the circuit, for example, protection against overvoltages, reverse polarity, control signals isolated by optocoupler and separate digital and analog references.

It has been also developed the firmware that is loaded into the controller, as well as a graphical user interface that allows the control of the motors from the computer. It is, therefore, a personalized design at all levels (software, firmware, hardware) that provides a compact solution for motors control applied to the positioning of elements in a laboratory.

KEYWORDS:

Micropositioner, Stepper Motor, Motor Controller, QT Creator, Printed Circuit Board

RESUM:

En els laboratoris del Centre de Tecnologia Nanofotònica de València hi ha múltiples eines per al posicionament d'elements, com ara microposicionadores o goniòmetres. Aquestes eines són de gran utilitat en experiments que requereixen canviar la posició d'una mostra o un sensor per monitorar canvis en els mesuraments realitzats. La major part compten amb un sistema de posicionament manual, de manera que es dificulta la tasca d'obtenir moviments precisos i repetibles. Per al cas específic d'aquest projecte, es proposa motoritzar un microposionador manual de tres eixos.

El present projecte aborda la creació d'un controlador de motors pas a pas de tres canals, així com l'elecció dels motors i el sistema que accionaran els eixos de l'microposionador. El controlador de motors s'ha dissenyat específicament per a aquest projecte, integrant tots els components necessaris en un circuit imprés per a proporcionar una solució senzilla d'utilitzar. El disseny maquinari incorpora les proteccions necessàries per a dotar de robustesa al circuit, per exemple: protecció contra sobrevoltatges, polaritat inversa, senyals de control aïllats per optoacoblador i referències digital i analògica separades.

També s'ha desenvolupat el firmware que es carrega al controlador, així com una interfície gràfica d'usuari que permet el control dels motors des de l'ordinador. Es tracta, per tant, d'un disseny personalitzat en tots els seus nivells (programari, firmware, maquinari) que proporciona una solució compacta per al control de motors aplicat al posicionament d'elements en un laboratori.

PARAULES CLAU:

Microposicionador, Motor pas a pas, Controlador de motors, QT Creator, Circuit imprès

Agradecimientos

A mis padres, Raúl y Nora, por su esfuerzo a lo largo de toda esta etapa y por el enorme apoyo que recibo en todo momento, sin el cual no hubiera podido llegar hasta aquí.

A Salvador Ponce y Jaime García, por el tiempo y el interés dedicado a la tutorización del proyecto y a su continuo seguimiento.

ÍNDICE

Índice de ilustraciones

Índice de tablas

1 OBJETO

El presente Trabajo de Fin de Grado (TFG) tiene como objeto el desarrollo del hardware y software de un controlador de motores paso a paso para cumplir la función de posicionador preciso, repetitivo, versátil, de fácil uso y de bajo coste. La aplicación particular para la que fue diseñado es la motorización de un microposicionador de tres ejes, por lo que además se han seleccionado los motores y el sistema de acoplamiento mecánico al microposicionador. No obstante, se trata de un sistema adaptable a cualquier aplicación en la que se requiera motorizar partes mecánicas.

El controlador de motores propuesto en este TFG proporciona una solución compacta y robusta, reuniendo todos los elementos en un único circuito impreso. Se evita de este modo tener un exceso de cableado, y permite un sencillo uso del dispositivo. El controlador requerirá de una fuente de alimentación con voltaje y conectores comunes para facilitar su integración a cualquier sistema. Un microcontrolador embebido gestionará los comandos recibidos desde el ordenador, y los traducirá en el número de pasos que cada motor debe dar. El circuito en su conjunto ha de ser robusto y resistir condiciones de uso desfavorables. Entre ellas cabe destacar la polaridad inversa y los sobrevoltajes. También ha de estar protegido frente a ruidos eléctricos provocados por la conmutación de las bobinas de los motores. Por ello se precisa del uso de circuitos de protección y aislamiento, que han sido diseñados para tal función.

Por otra parte, se ha diseñado una interfaz gráfica específicamente para el control de los tres motores. Desde ella se pueden modificar los parámetros básicos como velocidad y posición, permitiendo también el control a través del teclado y la introducción de coordenadas programadas para acceder a ellas cuando se desee.

Además, el proyecto y las herramientas utilizadas para su desarrollo son de software y hardware libre, siendo éste accesible a cualquier usuario para su utilización o modificación/adaptación del mismo.

2 SITUACIÓN DE PARTIDA, ESTUDIO DE NECESIDADES Y ALTERNATIVAS

2.1 Situación de partida

El posicionamiento de muestras es una actividad cotidiana en muchos campos de investigación, principalmente en aquellos en los que se requiere de sistemas de medida ópticos. Si bien es común contar con herramientas o instrumentos de laboratorio que incorporen la mecánica necesaria para posicionar los elementos deseados, éstos suelen ser de accionamiento manual o tener un coste elevado. Por un lado, se trata de sistemas de pequeño tamaño que necesitan del accionamiento sobre un tornillo cuyo movimiento se limita a solo pocos centenares de micrómetros por vuelta. El control manual tiene consigo la falta de repetitividad a la hora de realizar posicionamientos continuos de forma precisa.

En el Instituto Universitario de Tecnología Nanofotónica de Valencia se están llevando a cabo medidas de caracterización y sensado de sensores fotónicos. Según la tecnología utilizada en su fabricación, se puede disponer de varios sensores basados en guías fotónicas situadas en paralelo en un mismo chip. El alineamiento del haz de luz con las comentadas guías se realiza en la actualidad de forma manual. Por ello, este TFG tiene como finalidad la creación de un sistema de posicionamiento motorizado de alta precisión y repetitividad, que permita automatizar las medidas llevadas a cabo con los sensores fotónicos.

Para esta tarea el proyecto cuenta inicialmente con el microposicionador manual de tres ejes modelo M-DS40-XYZ de la casa Newport (Ilustración 1). En este contexto se plantea la creación de un sistema que permita motorizar los tres ejes del indicado microposicionador. Atendiendo a su versatilidad y modificando únicamente la parte mecánica, el presente TFG puede tener múltiples finalidades. Por ejemplo, en sistemas de posicionamiento presentes en mesas ópticas, o en sistemas de control que presenten una necesidad similar, como puede ser el caso de un goniómetro (equipo de medida angular). Con modificaciones mínimas, el sistema propuesto en este TFG puede ser empleado en los posicionadores presentes en líneas de fabricación industrial.

Ilustración 1. Microposicionador M-DS40-XYZ

2.2 Requerimientos

Para la correcta realización del proyecto, se parte de unos requerimientos iniciales que se deberán cumplir con el fin de proporcionar una herramienta robusta y sencilla para el usuario. Se propone evitar, por tanto, soluciones que requieran una excesiva manipulación por parte del operario (cableado, configuración inicial, etc.) y/o aquellas soluciones que no sean lo suficientemente robustas para trabajar en el entorno de un laboratorio. Es importante, además, que tanto la interfaz gráfica como el sistema permita un uso "plug-and-play", es decir, que esté libre de complejas configuraciones iniciales. Con ello, se pretende que pueda ser utilizado mediante sencillos pasos.

A continuación, se indican los requerimientos deseados:

- Capacidad de replicar y modificar el sistema de posicionamiento por personal con conocimientos básicos en electrónica. El hardware y el software han de utilizar herramientas libres de licencia.
- El dispositivo debe permitir un rango de voltaje de entrada amplio con el fin de proporcionar flexibilidad en la selección de la fuente de alimentación. Por tanto, deberá admitir, al menos, los valores de voltaje 12V y 24V. Además, debe poseer un conector de entrada estándar y con funcionalidad poka-yoke para eliminar el riesgo de conexión inversa, es decir, debe prevenir errores involuntarios que resulten en polaridad inversa a la hora de introducir el conector.
- El dispositivo deberá estar protegido contra sobretensiones para evitar daños en el circuito producidos por una mala configuración/elección de la fuente de alimentación. Además, contará con una protección contra polaridad inversa en caso de producirse.
- El dispositivo deberá ser capaz de controlar, al menos, tres motores simultáneamente.
- El dispositivo contará con un conector de entrada cuya función sea identificar eventos de final de carrera. Como medida de protección, dicha entrada estará aislada del circuito de control, ya sea galvánicamente o mediante optoacoplamiento.
- Las señales de control estarán aisladas de la electrónica de potencia, así como también la alimentación del circuito de control y el de potencia. Es condición, por tanto, disponer de referencias aisladas entre el bloque de control y el de potencia.
- El sistema contará con una interfaz gráfica de usuario con la que se podrá controlar el movimiento de los motores en tiempo real. Se deberá habilitar el movimiento de los tres motores en ambos sentidos mediante la pulsación de teclas del teclado.
- La interfaz gráfica permitirá, adicionalmente, conocer la posición relativa respecto a un origen. Dicho origen podrá ser establecido por el usuario.
- La comunicación del dispositivo con la interfaz gráfica del PC será compatible con el protocolo USB 2.0.
- El sistema deberá permitir una resolución de movimiento igual o inferior a 1µm.

2.3 Descripción del sistema de posicionamiento

El sistema de posicionamiento consta de cuatro partes principales: el microposicionador de tres ejes, los motores que accionan dichos ejes, el controlador de motores de tres canales y la interfaz gráfica de usuario que permite controlar el sistema.

2.3.1 Motor

Una de las primeras cuestiones que surgen cuando se plantea la necesidad de motorizar un eje mecánico es el tipo de motor a utilizar. Existen multitud de tipos y difieren entre sí principalmente por su modo de funcionamiento, pero también por su tamaño, eficiencia, par motor y control que puedan ofrecer.

En términos generales, el motor ideal para esta aplicación es aquel que ofrezca una dimensión reducida, así como también capacidad de control de posición de una forma precisa y repetitiva. El par motor necesario para rotar el eje mecánico es un parámetro a tener en cuenta, pero no se requieren altas prestaciones en este sentido puesto que las cargas que tendrá que movilizar no son excesivamente altas, eliminando la necesidad de optar por motores voluminosos de alta potencia.

Con estas condiciones iniciales los motores candidatos a utilizarse en esta aplicación pueden resumirse en dos grupos:

Motores con control tipo Servo

Estos motores tienen la principal característica de incorporar un circuito de control interno que permite conocer con exactitud el ángulo en el que se encuentra el eje del motor. Su estructura interna se fundamenta en un motor de corriente continua que, mediante un acoplamiento mecánico, ofrece realimentación sobre su posición angular a un circuito de control y éste corrige el error de posición hasta alcanzar el ángulo deseado [1]. La señal de control suele presentarse en forma de pulsos de frecuencia constante y con ciclo de trabajo variable. Suelen ofrecer gran par motor debido a la utilización de engranajes reductores en su eje.

Los servomotores suelen dividirse en dos tipos: con movimiento angular limitado a 180º y con movimiento libre. Los que limitan su giro a 180º quedan descartados para esta aplicación debido a que el microposicionador requiere un giro multivuelta.

Aunque su capacidad de control de posición los hace interesantes para esta aplicación, hay que tener en cuenta que este tipo de motores suelen venir preparados para controlar grandes cargas mecánicas, acompañado en muchos casos de juegos de engranajes. Sus aplicaciones más comunes suelen encontrarse en la robótica, aunque también tienen cabida en muchas aplicaciones industriales. Aunque es común encontrarse con servomotores de pequeño tamaño y poca potencia, estos suelen venir limitados a un giro de 180º y su uso está más bien reservado a aplicaciones que no requieran un posicionamiento preciso. Por último, hay que tener en cuenta que, por

norma general, su precio es superior al de otras alternativas, por lo que pueden añadir un sobrecoste al proyecto.

Motor paso a paso

Los motores paso a paso se caracterizan por realizar un giro de incremento angular discreto. Cada incremento de ángulo es conocido como “paso”, y su valor estándar es de 1.8º, necesitando 200 pasos para dar una vuelta completa. Por otro lado, también es posible encontrar en el mercado motores con un ángulo de paso inferior.

Su construcción interna difiere notablemente de la de los motores tipo servo, puesto que realizan su giro secuenciando la polarización de las bobinas que lo componen. Es por ello que los motores paso a paso requieren de un controlador especial para su operación [2]. En la Ilustración 2 se observa el detalle del bobinado interno de un motor paso a paso.

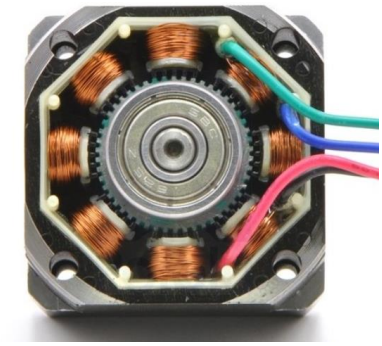


Ilustración 2. Interior de un motor paso a paso

Los motores paso a paso ofrecen una forma sencilla de controlar el ángulo de su eje. Este control es, a diferencia de los servomotores, en lazo abierto. Cuando el controlador recibe la señal para dar un paso, modifica su salida para actuar sobre las bobinas del motor, pero éste no realimenta al controlador sobre su posición real. No obstante, para la mayoría de las aplicaciones, un motor paso a paso ofrece un posicionamiento preciso, utilizándose en aplicaciones como impresoras 3D o máquinas de control numérico por computadora (CNC).

Cabe destacar que el incremento de ángulo por cada paso varía en función del modelo del motor. Si bien 1.8º es el estándar, se pueden encontrar motores con valores superiores o inferiores, como de 0.9º o incluso inferior. Existen también, motores a los que se le añade un engranaje planetario en su eje con el fin de obtener una reducción de paso aún mayor, ganando par motor.

Los motores paso a paso híbridos se dividen fundamentalmente en dos tipos: bipolar y unipolar. El modo de control es distinto según el tipo de motor utilizado. A grandes rasgos se puede diferenciar el tipo del motor según la cantidad de cables de los que disponga. Los unipolares tienen 5 o 6 cables, dependiendo de cómo estén realizadas las

conexiones internas de las bobinas, mientras que los bipolares disponen de 4 cables. En la Ilustración 3 se comparan los esquemas de los motores paso a paso bipolares y unipolares.

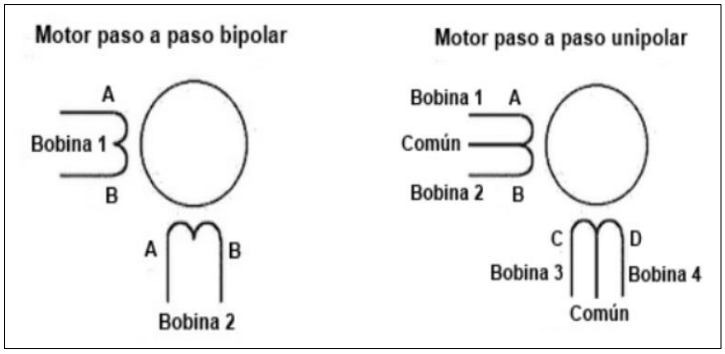


Ilustración 3. Motor paso a paso bipolar vs unipolar

Desde el punto de vista del control, los motores unipolares son más sencillos de controlar. Esto se debe a que su principio de operación se basa en energizar sus bobinas mediante un nodo común, y cambiar la polaridad de éstas tan solo alimentando un borne u otro de la bobina.

En el caso de los motores bipolares la estructura interna es más sencilla, puesto que se puede modelar como dos bobinas a las cuales se les invierte la polaridad para conseguir el giro del motor. De aquí se puede extraer la ventaja más inmediata de este tipo de motores: en los motores unipolares se energiza la mitad de la bobina (desde el punto común hasta uno de los dos bornes, quedando la otra mitad sin utilizar), mientras que en los bipolares se energiza toda la bobina al no existir el nodo común. Sin embargo, esta ventaja trae consigo el inconveniente de un control más complejo. Esto se debe a que, desde el punto de vista electrónico, es más complicado invertir la polaridad de un circuito que energizar dos puntos distintos bajo una referencia común.

A pesar de que los motores bipolares requieren un control más complejo, su uso compensa debido a su mayor par motor y a la existencia de múltiples soluciones integradas que gestionan el control adecuado del motor.

Solución adoptada

Por lo expuesto en los anteriores apartados, el motor que mejor se ajusta a las necesidades del sistema es un motor paso a paso. En el mercado existen motores paso a paso con gran variedad de dimensiones, por lo que es fácil elegir uno de un tamaño apropiado para nuestros requerimientos. Además, su principal ventaja frente al resto radica en la facilidad que ofrecen para controlar su posición angular de una forma precisa y con un control sencillo. Como principal inconveniente se tiene su modo de giro discreto. Cuanto más pequeño sea el incremento de ángulo por paso más resolución tendrá el sistema, además de obtener un movimiento más fluido.

El microposicionador utilizado en el proyecto desplaza su base 318 µm por cada vuelta del tornillo sin fin. Puesto que el sistema debe ser capaz de ofrecer 1 µm de resolución de desplazamiento, el ángulo de paso del motor debe ser inferior a 1.13°. Como se ha comentado previamente, el ángulo de paso estándar es 1.8°, aunque es posible encontrar en el mercado motores con ángulos de paso de 0.9° o incluso 0.36°. No obstante, se ha decidido utilizar un motor con engranajes planetarios acoplados a su eje para obtener un ángulo de paso notablemente inferior, siendo éste de 0.02°. Se pretende conseguir con esto un movimiento más fluido y evitar así perturbaciones mecánicas.

2.3.2 Controlador del motor

El controlador es un elemento imprescindible para el manejo de un motor. Ya sean de corriente alterna o continua, motor paso a paso o servomotor, la mayor parte de las aplicaciones tienen en común la necesidad de un bloque intermedio -entre la fuente de alimentación y el motor- que gestione las señales de control para accionarlo correctamente [3].

Tal y como se ha planteado en el apartado anterior, el motor seleccionado para esta aplicación ha sido un motor paso a paso, por lo que el controlador será el apropiado para este tipo de motores. Aunque existen ciertas combinaciones de equipos comerciales que hubieran sido válidos para el proyecto, se decidió abordar el diseño de dicho controlador para poder ofrecer una solución todo en uno.

Algunos controladores comerciales no cuentan con comunicación USB y requieren de un sistema embebido externo para enviar las señales de control. Estos controladores fueron descartados desde un primer momento porque no ofrecen una solución compacta. Otros sí cuentan con comunicación USB pero no disponen los tres canales integrados, siendo necesario replicar el sistema en tres bloques. Además, gran parte de ellos vienen preparados para trabajar con una interfaz cerrada, haciendo imposible la creación de un software libre para operar el motor. Con la realización de un controlador de diseño propio se asegura el cumplimiento de los requerimientos y se proporciona una solución flexible y compacta para su uso en el laboratorio. La Ilustración 4 y la Ilustración 5 muestran ejemplos de algunos controladores de motores típicos.

Ilustración 4. Controlador JOVA Tims-0201

Ilustración 5. Controlador DM320T

2.3.3 Interfaz gráfica de usuario

La interfaz gráfica es la herramienta que permite al usuario controlar los motores y todos los parámetros relevantes del sistema de posicionamiento. Se despliega un panel de control desde el cual se puede visualizar todas las opciones disponibles para el manejo de los motores. Una interfaz gráfica se desarrolla mediante herramientas software que las integran en sus entornos de desarrollo.

Una de las herramientas ampliamente utilizadas para esta tarea es LabVIEW. Se trata de una plataforma de programación visual que incorpora todas las herramientas necesarias para realizar interfaces gráficas de forma sencilla. Surge con la finalidad de ofrecer una herramienta que permita la automatización de pruebas y sistemas de test. Su uso es adecuado en la mayor parte de los procesos industriales o incluso en equipamiento de laboratorio cuando se requiere controlar o automatizar una tarea. No obstante, se trata de una herramienta que se encuentra bajo licencia, por lo que se descartó su uso para este proyecto [4].

En su lugar, se optó por una plataforma de software libre que permitiera la creación de interfaces gráficas eficazmente, y que además de lugar a ficheros ejecutables que puedan operar en cualquier ordenador sin la necesidad de disponer del código fuente ni licencia de uso. La plataforma QT Creator, ampliamente utilizada en el mundo de la programación con tales propósitos, permite la creación de interfaces gráficas mediante la programación orientada a objetos en C++, y de ficheros ejecutables (.exe) [5]. La Ilustración 6 muestra una pantalla asociada al entorno de desarrollo QT.

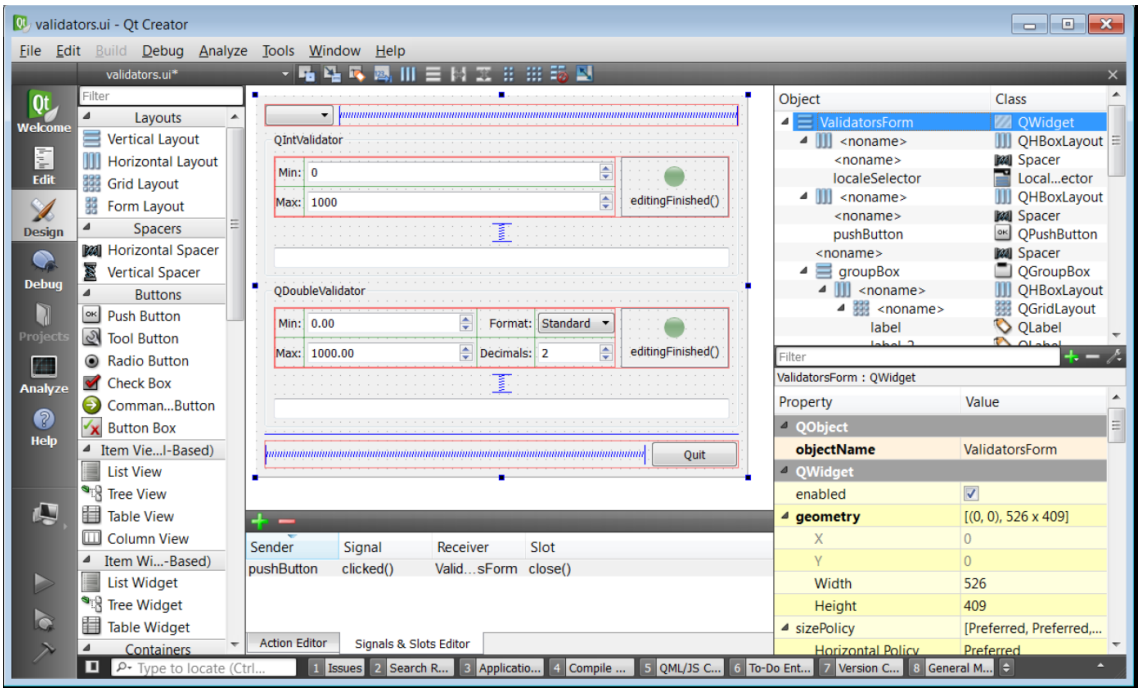


Ilustración 6. Ejemplo de entorno desarrollo QT

3 DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN ADOPTADA

3.1 Visión general del sistema

En los anteriores apartados se ha abordado la necesidad del proyecto, la situación de partida y sus requerimientos. Tras analizar las diferentes alternativas y contextualizarlas en la aplicación que se pretende construir, se han seleccionado los distintos bloques que la componen. La Ilustración 7 muestra el diagrama general del sistema. También se expone en los siguientes apartados la descripción de sus distintas partes. El esquema eléctrico completo se ha incluido en el apartado de Planos de este TFG.

El sistema está formado por:

- Motores paso a paso
- Controlador de motores paso a paso
- Firmware para el controlador de motores paso a paso
- Interfaz gráfica de usuario
- Acoplamiento mecánico entre motores y microposicionador

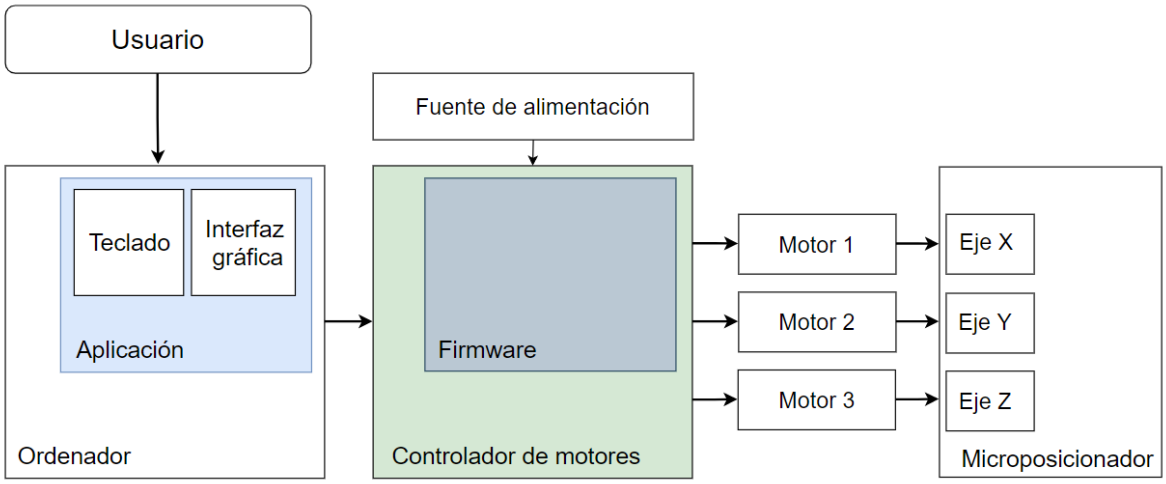


Ilustración 7. Esquema general del sistema

A grandes rasgos, el ordenador envía comandos al controlador y estos son interpretados por el firmware que incorpora, para posteriormente traducirlo a las señales de control que determinaran la velocidad y dirección de giro de los motores paso a paso. El controlador toma la alimentación de una fuente externa para proporcionar la suficiente potencia eléctrica a los motores. Finalmente, este movimiento se transmite al microposicionador mediante unos acopladores mecánicos flexibles. Cabe destacar la importancia de que los acopladores mecánicos dispongan de cierta flexibilidad, puesto que, de lo contrario, en una configuración en la que los motores están anclados a una base fija, se crearían esfuerzos mecánicos que impedirían el desplazamiento perpendicular a dichos ejes.

Cada bloque será explicado con más detalle en los siguientes apartados.

3.2 Desarrollo del controlador de motores

El controlador de motores paso a paso ha sido diseñado y ensamblado específicamente para este TFG. En los siguientes apartados se detallarán los elementos internos del controlador. En la Ilustración 8 aparece su diagrama de bloques., mientras que en la Ilustración 9 se muestra una vista general del dispositivo desarrollado.

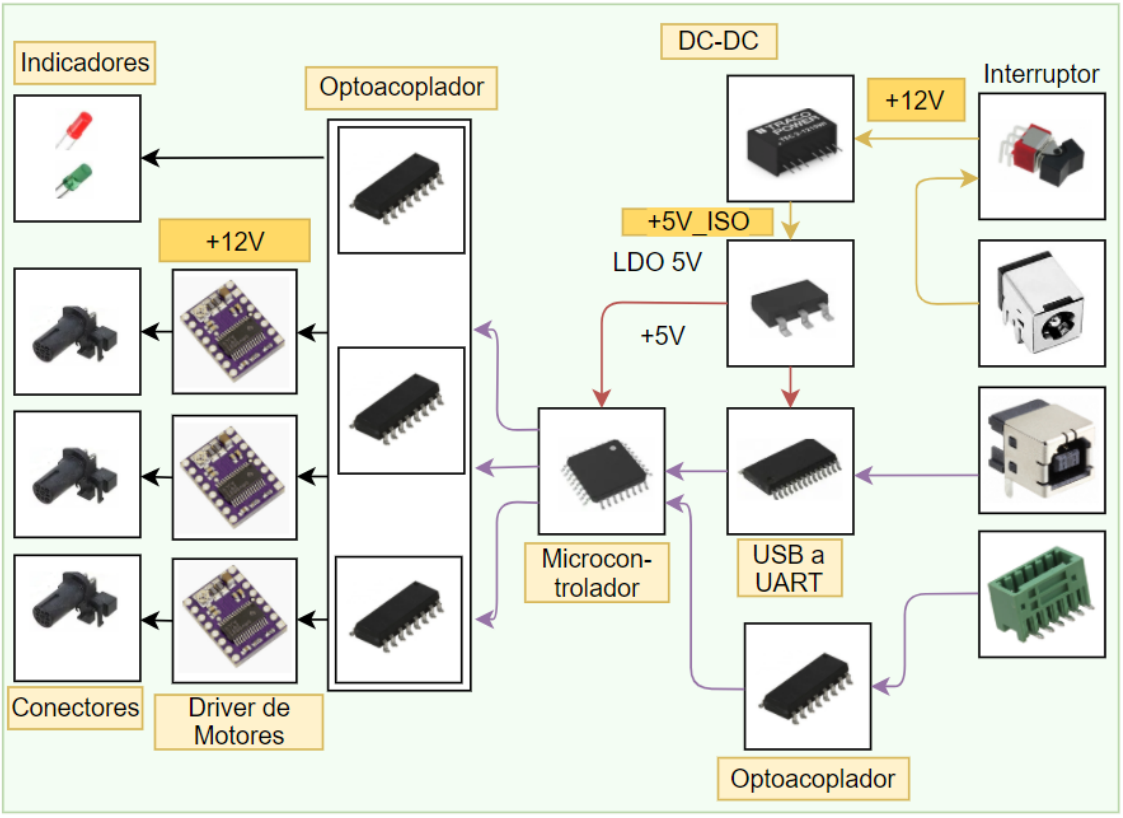


Ilustración 8. Diagrama de bloques del controlador de motores

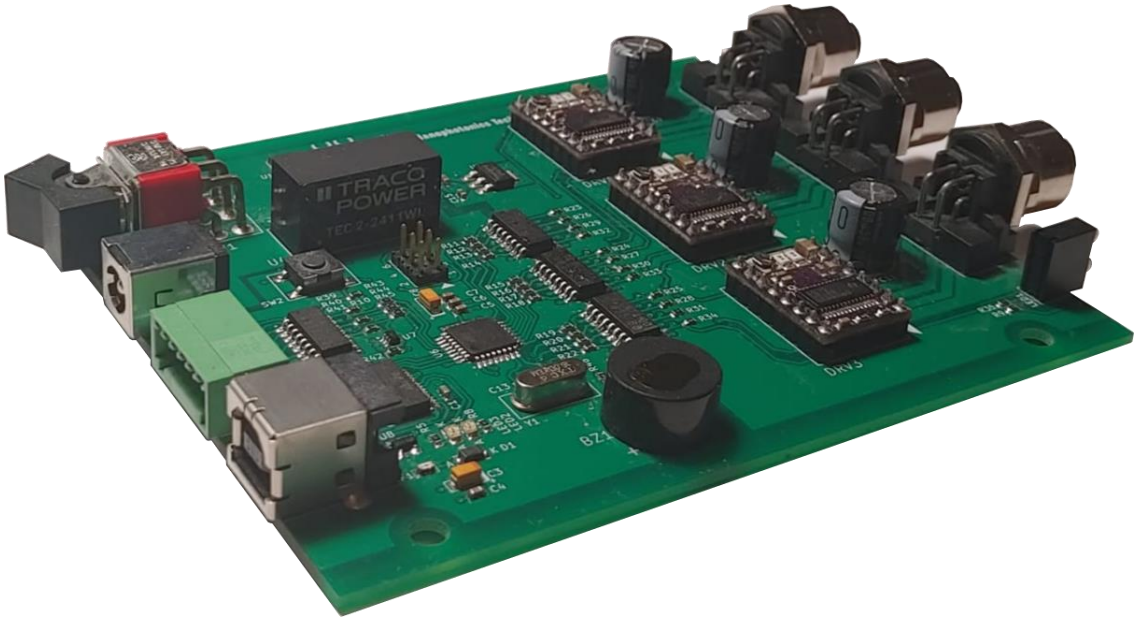


Ilustración 9. Vista general del controlador de motores paso a paso desarrollado.

El controlador se ha diseñado para ofrecer una solución compacta y robusta en el control de sus tres canales. La dimensión del circuito impreso realizado para este TFG es de 118x99.5 mm, por lo que es compatible con las cajas receptáculo de la familia 1455L12 y 1455N12 de Hammond Manufacturing (Ilustración 10) [6]. No obstante, también dispone de cuatro agujeros en cada esquina para facilitar su montaje en cualquier otra superficie en caso de no utilizar una carcasa estándar.

Ilustración 10. Caja Hammond Manufacturing

3.2.1 Alimentación y Protecciones

De acuerdo con el esquema eléctrico presente en la Ilustración 11, la alimentación es proporcionada a través del conector J1. El interruptor SW1 permite cortar la alimentación general del circuito del circuito. Aunque el nombre de las pistas que van conectadas a él indica un valor de voltaje de 24V (+24V_AUX, +24V_IN), esto tan solo se refiere al valor nominal para el que fue diseñado el circuito. No obstante, el circuito admite un rango de entrada amplio. Se debe diferenciar entre el máximo voltaje admisible en la entrada y el voltaje de operación del circuito (Tabla 1).

	Vmin (V)	Vmax (V)
Voltaje de entrada límite	-40	40
Voltaje de operación	9	30

Tabla 1. Valores voltaje de entrada

El circuito de entrada dispone del circuito integrado LTC4365 (U2) que en combinación con un transistor MOSFET dual tipo N (U1) otorgan una protección contra sobrevoltajes, infravoltajes y polaridad inversa. Mediante la red resistiva R1, R2 y R3 se consigue configurar los valores límites a partir de los cuales el MOSFET pasa a un estado de corte, interrumpiendo el flujo de corriente por el circuito.

El MOSFET sólo permitirá el paso de la corriente cuando el voltaje de entrada se encuentre entre 9V y 30V. Valores de voltajes superiores a 30V o inferiores a 9V cortarán el paso de la corriente y el dispositivo se mantendrá apagado. Por encima de 40V, se superará el voltaje de ruptura del MOSFET y el dispositivo quedará irremediablemente averiado.

La polaridad inversa será bloqueada debido a la acción de U2 sobre la puerta del MOSFET. Además, es importante la disposición de ánodos enfrentados en la que se encuentra el MOSFET dual U1. Si se tratara de un MOSFET simple, el diodo intrínseco permitiría la circulación de corriente en alguno de los dos casos de alimentación posible (polaridad inversa o directa). Al tratarse de un MOSFET dual con sus diodos internos en serie, tal y como se observa en la Ilustración 11, la corriente no circulará por los diodos ni en polaridad inversa ni en directa.

El conversor DC-DC aislado TEC-2-2411-WI (U4) se utiliza para alimentar exclusivamente la parte del control digital. Al tratarse de un conversor aislado, la referencia de la salida del conversor está aislada galvánicamente de la referencia de su entrada. Se tiene, por tanto, dos referencias (GND1 y GND2) aisladas entre sí. Esta característica combinada con el uso de optoacopladores permite separar el control digital de la etapa de potencia.

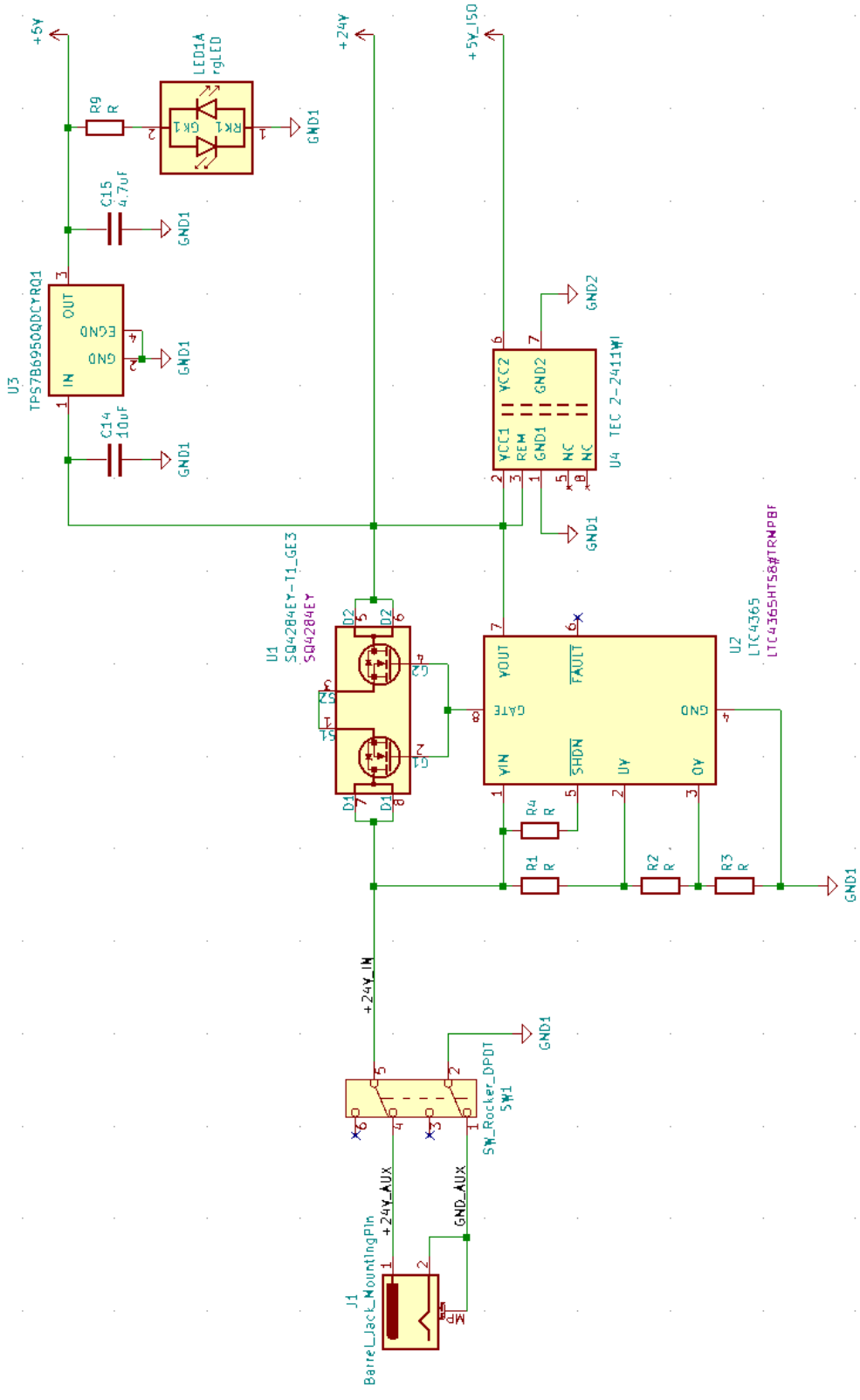


Ilustración 11. Esquema Alimentación y Protecciones

3.2.2 Microcontrolador

El microcontrolador es el componente central del dispositivo. Almacena el firmware que permite servir de interfaz entre el ordenador y los drivers de los motores, interpretando los comandos de entrada del puerto USB y traduciéndolo a los pulsos de control que se aplican a dichos drivers. Además, también permite disponer de entradas y salidas para interrumpir procesos por eventos externos o activar indicadores de tipo luminoso o de audio.

El microcontrolador empleado para esta aplicación es el modelo ATMEGA328P-AU [7] (Ilustración 12)

Ilustración 12. Microcontrolador ATMEGA328P

Se trata de un modelo ampliamente utilizado a lo largo de los años en múltiples productos electrónicos. Por ello, dispone de un extensa documentación y abundante soporte técnico en internet para su uso en proyectos. La principal característica por la que es ideal para este proyecto radica en la facilidad de implementar sistemas de software y hardware libre. La plataforma Arduino lo ha utilizado como microcontrolador principal en sus placas de desarrollo, por lo que esto ha contribuido a que se popularice en gran parte de los usuarios de microcontroladores. Con la elección de este microcontrolador se permite la compatibilidad con el entorno de programación de Arduino, lo que hace que el proyecto sea muy versátil y accesible por un gran número de posibles interesados. La Tabla 2 y la Tabla 3 muestran la distribución de los pines del microcontrolador utilizados para controlar el motor, así como las entradas de las interrupciones necesarias para su correcto funcionamiento.

Descripción	**Nombre net**	**Nombre puerto**	**Pin**
Modo 0	M0	PB5	17
Modo 1	M1	PB4	16
Modo 2	M2	PB3	15
Habilitar motor 1	EN1	PB2	14
Habilitar motor 2	EN2	PD7	11
Habilitar motor 3	EN3	PD4	2
Dirección motor 1	$\overline{\text{DIR1}}$	PB0	12
Dirección motor 2	$\overline{\text{DIR2}}$	PD5	9
Dirección motor 3	$\overline{\text{DIR3}}$	PC1	24
Paso motor 1	$\overline{\text{STEP1}}$	PB1	13
Paso motor 2	$\overline{\text{STEP2}}$	PD6	10
Paso motor 3	$\overline{\text{STEP3}}$	PD3	1

Tabla 2. Pines microcontrolador, control de motor

Descripción	Nombre net	Nombre puerto	Pin
Entrada externa 1	EX_SW1	PC3	26
Entrada externa 2	EX_SW2	PC4	27
Entrada externa 3	EX_SW3	PC5	28
Extrada interrupción	Interrupt	PD2	32
Salida piezoeléctrico	BZ1	PC0	23

Tabla 3. Pines microcontrolador, entrada interrupciones

3.2.2.1 Bootloader y comunicación

Los microcontroladores necesitan ser programados mediante herramientas hardware específicas. Esto se hace a través de los pines que el microcontrolador incorpora para su programación. Para evitar el uso de estas herramientas y habilitar la programación mediante USB, es necesario cargar un programa en su primer uso. Este programa es conocido como bootloader.

La programación por USB se hace mediante el circuito integrado FT232RL de la marca FTDI, que es un conversor USB-UART. Todos los comandos que el ordenador envía son procesados por el conversor y traducidos al protocolo UART, uno de los protocolos serie más comunes en la comunicación digital. Los pines PD0 y PD1 (RX y TX respectivamente) vienen preparados por hardware para interpretar el protocolo UART. Por lo tanto, la salida del conversor se conecta con estos pines.

Una vez cargado el bootloader, deja de ser necesario el uso de herramientas específicas y se habilita el puerto USB como periférico para comunicarse con el ordenador. El proceso de carga del bootloader puede consultarse con más detalles en la siguiente referencia [8].

3.2.2.2 Firmware

Una vez instalado el bootloader, el firmware puede cargarse a través del puerto USB. El controlador tiene que gestionar el movimiento de tres motores paso a paso en paralelo, a la vez que reacciona a eventos de forma instantánea tales como detectar pulsaciones de teclado que cambian el sentido de giro, o enviar periódicamente un comando a través del puerto serial. Todo esto tiene que hacerse de forma que una tarea no bloquee a la siguiente, es decir, no puede haber esperas activas. Por ello, se debe adoptar la filosofía de programación en tiempo real con el fin de atender a todas las tareas, garantizando que ningún evento se deja sin atender.

Para implementar la programación en tiempo real, se inicializa un timer del microcontrolador que servirá de base de tiempos para el programa principal. Para esta aplicación específica, el timer es configurado para que cada 10 microsegundos genere una interrupción. Cuando la interrupción es generada, se suma una unidad a una variable que sirve como contador. Es esta variable la que se utiliza como referencia para conocer el tiempo transcurrido desde que se inició el programa. Cada evento propio del controlador que requiera ser temporizado, como por ejemplo el envío de datos hacia la interfaz gráfica a través de comunicación serial, o bien la generación de un tren de pulsos que controle los pasos de cada motor, se gestiona mediante la comparación del tiempo

transcurrido entre un instante y otro posterior. De esta forma se evita la utilización de funciones de espera activa.

3.2.3 Drivers de motores

Los drivers de los motores se encargan de convertir las señales provenientes del microcontrolador en la señal de potencia que se aplica a los motores. Para este proyecto se ha optado por el módulo Pololu – DRV8825 (Ilustración 13). Se trata de un módulo que incorpora todo el soporte hardware necesario para poner en funcionamiento el circuito integrado DRV8825 de Texas Instruments [9].

Algunas de las características del módulo son:

- Operación con motores paso a paso bipolares
- Límite de corriente ajustable
- Protección contra exceso de corriente y sobretemperatura
- Voltaje de operación: 8.2V a 45V
- Corriente por bobina: 1.5A máximo

Cabe destacar que el control de la corriente que pasa por la bobina del motor se realiza mediante el método llamado “chopper”. Tras el pulso de voltaje aplicado a la bobina el driver monitoriza los valores de corriente a través de ésta mediante una resistencia shunt. Cuando el valor de corriente alcanza el valor límite establecido, el driver corta el pulso de voltaje para provocar un descenso de la corriente que, una vez descienda por debajo del límite establecido, vuelve a aplicar el pulso de voltaje para incrementar la corriente nuevamente [10]. Esto permite controlar la corriente media que atraviesa la bobina para un amplio rango de voltajes de entrada.

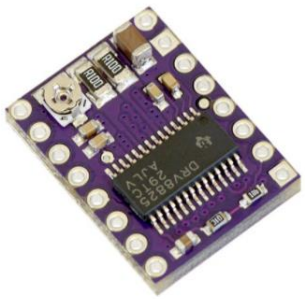


Ilustración 13. Driver Pololu DRV8825

3.2.4 Elementos y conectores parte trasera

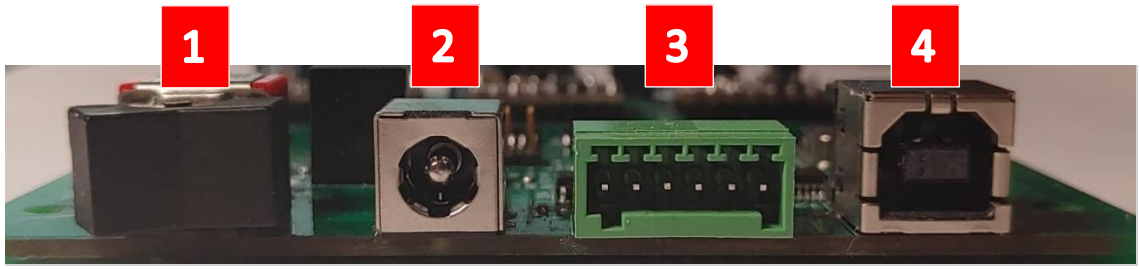


Ilustración 14. Controlador de motores, vista trasera

El interruptor [Elemento 1, Ilustración 14] permite apagar o encender el dispositivo. Enclavar el interruptor a la izquierda apagará el controlador, mientras que hacerlo hacia la derecha lo encenderá.

Cuenta con una entrada de alimentación tipo Jack 3.5 mm [Elemento 2, Ilustración 14], por lo que es compatible con la mayoría de los conectores de las fuentes de alimentación que se pueden encontrar en una oficina o un laboratorio. El rango de voltaje de entrada es de 9-30V

El bloque de terminales [Elemento 3, Ilustración 14] permite disponer de tres entradas digitales para reaccionar a diferentes eventos o interrupciones, como por ejemplo eventos de final de carrera o paros de emergencia. Este bloque está aislado del circuito interno mediante optoacopladores, por lo que su manipulación no conlleva riesgos. En la Ilustración 15 y en la Tabla 4 se observan la distribución de pines del bloque de terminales.

Ilustración 15. Pines bloque de terminales

Número de pin	Descripción
Pin 1	Positivo. +5V alimentación auxiliar
Pin 2	Negativo. Referencia alimentación auxiliar
Pin 3	Negativo. Referencia alimentación auxiliar
Pin 4	Entrada 1
Pin 5	Entrada 2
Pin 6	Entrada 3

Tabla 4. Pinout bloque de terminales

Por último, el conector USB tipo B permite la comunicación entre el PC y el dispositivo [Elemento 4, Ilustración 14].

3.2.5 Elementos y conectores parte delantera

En la parte frontal se encuentran los conectores a los que van conectados los motores. Tal y como se aprecia en la Ilustración 16, se considera, de derecha a izquierda, Motor 1, Motor 2 y Motor 3. Su distribución de pines se define como aparece en la Ilustración 17 y la Tabla 5. Para esta aplicación específica, se ha considerado el Motor 1, Motor 2 y Motor 3 conectados mecánicamente con los ejes del microposicionador X, Y y Z respectivamente.

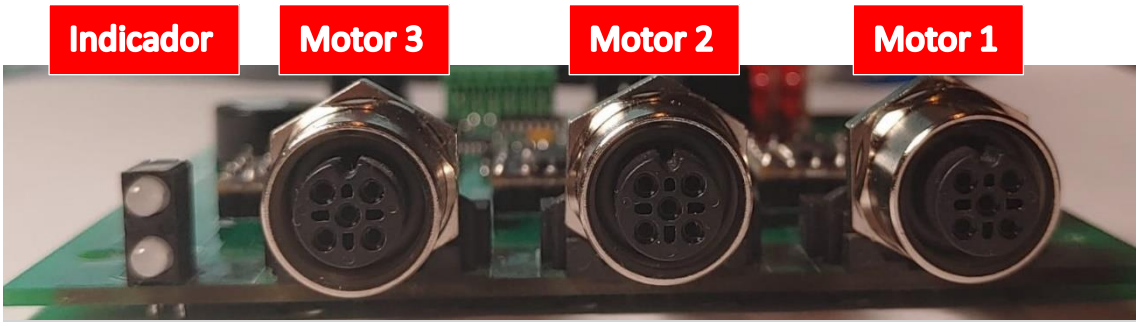


Ilustración 16. Controlador de motores, vista delantera

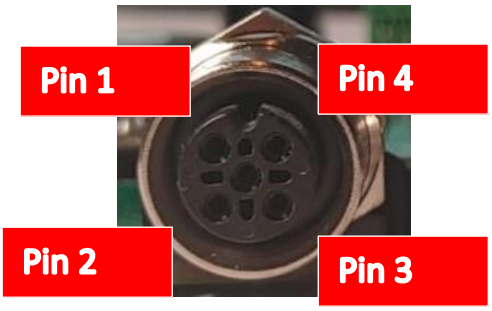


Ilustración 17. Pines conector de motor

Número de pin	**Descripción**
Pin 1	A+ (Ilustración 16)
Pin 2	B+ (Ilustración 16)
Pin 3	A- (Ilustración 16)
Pin 4	B- (Ilustración 16)

Tabla 5. Pinout conector de motores

Por otra parte, el dispositivo dispone de un indicador de alimentación y de fallo [Indicador, Ilustración 16] El led superior se ilumina en verde si la alimentación está en el rango predefinido y se queda apagado si tras conectar la alimentación, ésta no se encuentra en rango. El led inferior se ilumina en rojo si alguno de los circuitos integrados que controlan los motores detectan cortocircuito o exceso de temperatura. Bajo una operación normal el led superior se ilumina en verde y el led inferior se mantiene apagado.

3.2.6 Circuito impreso

Las dimensiones del circuito impreso son 118x99.5 mm. Consta de dos capas de cobre de 35 micrómetros de espesor. El espesor total del circuito es 1.6 mm, siendo la fibra de vidrio FR-4 el material base. Consta de máscara antisoldante color verde en ambas capas, lo cual protege al cobre de la oxidación. Los pads de cobre expuestos son tratados con acabado HASL, es decir, los pads quedan estañados protegiéndolos de la oxidación. Por último, el circuito cuenta con serigrafía en la capa superior para indicar la posición y orientación de los componentes. El fabricante de la PCB, que en el caso de este proyecto es AllPCB, aporta toda la información necesaria referente a los archivos de fabricación requeridos para su fabricación [11].

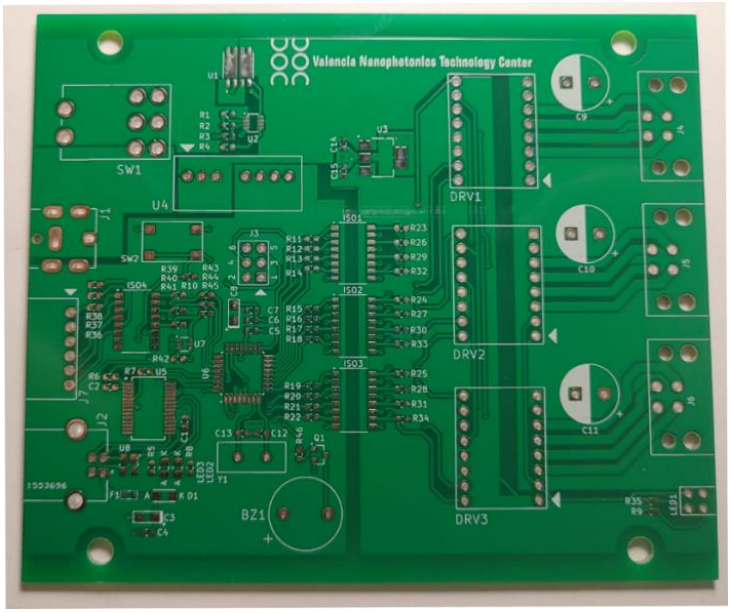


Ilustración 18. PCB capa superior

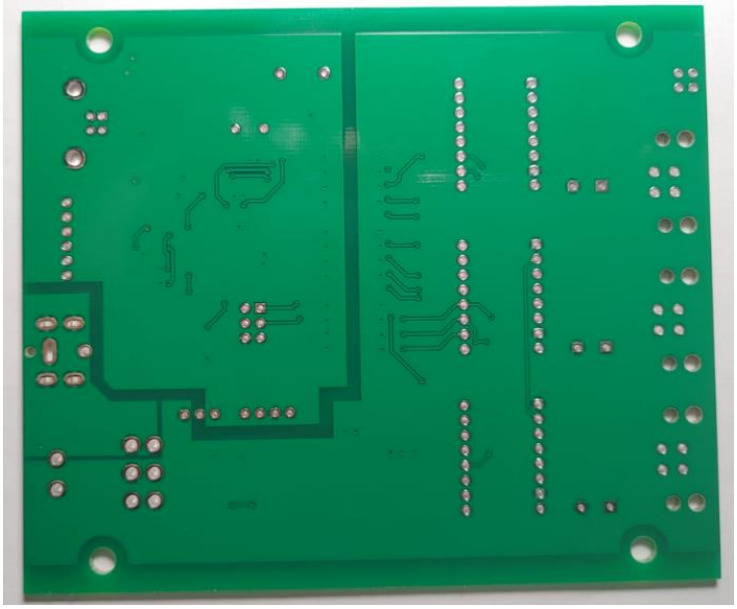


Ilustración 19. PCB capa inferior

3.3 Motor

Los motores que se utilizan en la aplicación son del modelo 8HS15-0604S-PG90 de la marca Stepperonline. Se trata de unos motores paso a paso de tipo bipolar, esto es, disponen de dos bobinas estándar, sin terminal central. Como se ha comentado previamente, la principal ventaja de estos motores respecto a los unipolares radica en su simpleza en su construcción y en su mayor par motor para una potencia constante [12]. Por contra, requieren un controlador de motores más sofisticado, puesto que se hace necesario poder invertir la polaridad de las bobinas, a diferencia de los motores unipolares que basta con tan solo conmutar la alimentación de la bobina entre los dos terminales externos y el central.

El motor posee 4 cables de conexión. La Ilustración 20 e Ilustración 21, extraídas del datasheet, indican el esquema de conexión interno del motor.

TYPE OF CONNECTION (EXTERN)		MOTOR	
PIN NO	BIPOLAR	LEADS	WINDING
1	A —	BLK	A
2	A\ —	GRN	A\
3	B —	RED	B
4	B\ —	BLU	B\

Ilustración 20. Datasheet del motor, pinout

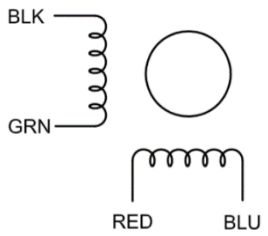


Ilustración 21. Bobinas del motor

Una de las características relevantes del motor es la incorporación de un engranaje planetario al eje del motor paso a paso. Estos engranajes actúan como una reductora de relación 90.25:1. Con ello, consigue un ángulo de paso de solo 0.02 grados. En la Ilustración 22 se muestra una fotografía del motor utilizado.

Ilustración 22. Motor 8HS15-0604S-PG90

3.4 Microposicionador

El microposicionador utilizado es el modelo M-DS40-XYZ de la casa Newport. Consta de tres ejes accionados manualmente. En la Ilustración 23 se muestra el microposicionador y el sistema de referencia adoptado para este proyecto. En azul se indican los ejes de coordenadas. En naranja, los tornillos sin fin que accionan cada eje. En verde, el sentido de giro para desplazar el microposicionador hacia valores positivos del eje, resultando el giro en sentido contrario en desplazamientos hacia valores negativos del eje. El controlador de motores dispone de tres canales (Motor 1, Motor 2 y Motor 3) cuya disposición puede verse en el capítulo 3.2.5 y que corresponden con los X, Y y Z respectivamente.

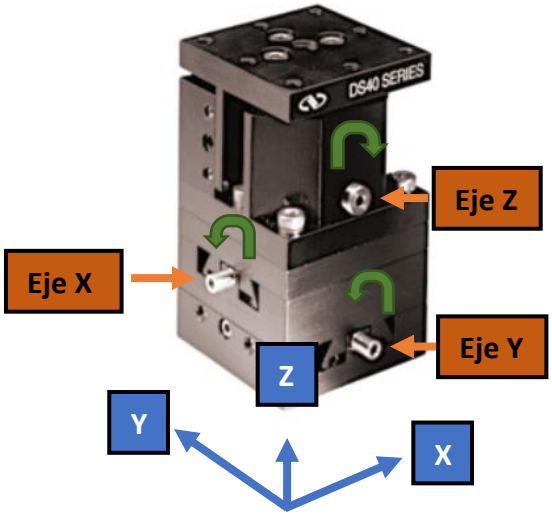


Ilustración 23. Microposicionador y sistema de referencia

El microposicionador permite realizar desplazamientos de hasta 14 mm en cada eje. Dispone de tornillos sin fin cuyo roscado es de 80 TPI (del inglés, Thread Per Inch), lo que se traduce en movimientos de 318 µm por cada vuelta de rosca. La sensibilidad del microposicionador, indicada por el fabricante, es de 1 µm. Por ello, se ha fijado este valor como límite inferior para el sistema de posicionamiento realizado en este TFG. Cabe destacar que el motor seleccionado realiza ángulos de paso de 0.02° debido a su reductor planetario acoplado en el eje del motor. El ángulo de giro necesario para que los tornillos sin fin desplacen el sistema 1 µm es de 1.13 °. Aunque la resolución de los pasos del motor es superior a la que el microposicionador puede gestionar, como ya se ha indicado, el controlador de motores realizará el control de tal forma que el mínimo desplazamiento sea de 1 µm.

3.4.1 Acoplamiento mecánico

El acoplamiento mecánico que transmite el movimiento desde el motor a los ejes del microposicionador es el modelo A 7C12M3320333 de SDP/SI. Se trata de un eje flexible de acero con unos encajes en los extremos para acoplarse tanto al eje del motor como a los tornillos sin fin del microposicionador mediante puntas allen. Tiene una longitud de 20.3 cm. En la Ilustración 24 se muestra el eje flexible utilizado.

Ilustración 24. Eje flexible

3.5 Desarrollo de Software

La interfaz gráfica de usuario (GUI) está realizada en la plataforma de desarrollo QT Creator. El lenguaje de programación utilizado para la GUI es C++. La funcionalidad principal de la interfaz gráfica es servir de intermediario entre el usuario y el controlador de motores. Por ello, la gestión de envío y recepción de datos a través del puerto USB es fundamental.

Al abrir por primera vez la interfaz gráfica, ésta aparece con la mayoría de sus opciones deshabilitadas. Para poder empezar a utilizarla, es necesario inicializar la comunicación serie con el controlador. Dicha comunicación es bidireccional:

- Por una parte, la interfaz gráfica envía comandos al controlador, que pueden ser órdenes de movimiento o bien parámetros como las coordenadas programadas.
- Por otra parte, el controlador envía comandos a la interfaz gráfica. El comando principal que el controlador envía es el que indica la posición actual del mismo. Se trata de un contador interno que es transmitido a la interfaz gráfica periódicamente con el fin de que sea el propio controlador quien reporte la posición actual, en lugar de ser calculada como un método indirecto desde la interfaz gráfica. Además, también envía otros comandos que indican el estado del controlador, como por ejemplo si ha iniciado la comunicación serieo si se ha desconectado de la misma.

3.5.1 Comandos enviados al controlador de motores

Los comandos que la interfaz gráfica envía al controlador de motores son cadenas de caracteres terminados con un salto de línea “\n”. Cada comando empieza con una palabra clave seguida de sus parámetros, utilizando la barra espaciadora como separador.

La sintaxis de los comandos enviados al controlador de motores aparece en la Tabla 6.

Comando		Parámetro 1		Parámetro 2		Parámetro n…	\n

Tabla 6. Sintaxis comandos enviados al controlador de motores

Las Tablas 7, 8, 9, 10, 11, 12, 13 y 14 presentan algunos de los principales comandos utilizados.

Comando	Parámetro 1	Descripción
MOV	MOT1	Acciona el motor 1 (Eje X)
	MOT2	Acciona el motor 2 (Eje Y)
	MOT3	Acciona el motor 3 (Eje Z)
	Parámetro 2	Descripción
	C	Sentido de giro del motor a favor de agujas del reloj (Clock)
	CC	Sentido de giro del motor en contra de las agujas del reloj (Counter Clock)

Tabla 7. Comando MOV

Ejemplo comando MOV: “MOV MOT1 CC\n”

Comando	Parámetro 1	Descripción
VEL	MOT1	Modifica la velocidad del motor 1 (Eje X)
	MOT2	Modifica la velocidad del motor 2 (Eje Y)
	MOT3	Modifica la velocidad del motor 3 (Eje Z)
	Parámetro 2	Descripción
	[1-1500]	Número entre 1 y 1500. Representa la velocidad del motor en pasos por segundo.

Tabla 8. Comando VEL

Ejemplo comando VEL: “VEL MOT1 1000”

Comando	Parámetro 1	Descripción
STOP	MOT1	Detiene el motor 1 (Eje X)
	MOT2	Detiene el motor 2 (Eje Y)
	MOT3	Detiene el motor 3 (Eje Z)

Tabla 9. Comando STOP

Ejemplo comando STOP: “STOP MOT2\n”

Comando	Parámetro 1	Descripción
P	[i]	Número del 1 al 7. Establece la coordenada a modificar
	Parámetro 2	Descripción
	[n]	Número entero con signo 32 bits. Representa la coordenada relativa del motor 1 (Eje x) expresada en pasos.
	Parámetro 3	Descripción
	[n]	Número entero con signo 32 bits. Representa la coordenada relativa del motor 2 (Eje y) expresada en pasos.
	Parámetro 4	Descripción
	[n]	Número entero con signo 32 bits. Representa la coordenada relativa del motor 3 (Eje z) expresada en pasos.

Tabla 10. Comando P

Ejemplo comando P: “P 3 100000 100000 200000\n”

Comando	Parámetro 1	Descripción
GOTO	[i]	Número del 1 al 7. Establece la coordenada a la que dirigirse

Tabla 11. Comando GOTO

Ejemplo comando STOP: “GOTO 3\n”

Comando	Descripción
CONNECT	Inicializa el controlador de motores

Tabla 12. Comando CONNECT

Ejemplo comando CONNECT: “CONNECT\n”

Comando	Descripción
DISCONNECT	Suspende el controlador de motores

Tabla 13. Comando DISCONNECT

Ejemplo comando DISCONNECT: “DISCONNECT\n”

Comando	Descripción
ORIGIN	Resetea el origen de coordenadas

Tabla 14. Comando ORIGIN

Ejemplo comando ORIGIN: “ORIGIN\n”

3.5.1 Comandos enviados por el controlador de motores

El controlador de motores envía periódicamente, cada 100 ms, la posición relativa en la que se encuentra. Esta posición es leída por el programa y tras realizar las conversiones pertinentes para trasladar el dato a micrómetros, es mostrado por pantalla. Con la finalidad de obtener una correcta lectura, es necesario delimitar cuando empieza el mensaje y cuando acaba. Por ello, el comando enviado cuenta con un carácter de inicio y otro de fin, que son “#” y “:” respectivamente.

La sintaxis de los comandos recibidos del controlador de motores es la siguiente:

#		Comando		Parámetro 1		Parámetro 2		Parámetro n…		:

Tabla 15. Sintaxis comando enviado por el controlador de motores

Comando	Parámetro 1	Descripción
POS	[n]	Número entero con signo 32 bits. Representa la coordenada relativa del motor 1 (Eje x) expresada en pasos.
	Parámetro 3	Descripción
	[n]	Número entero con signo 32 bits. Representa la coordenada relativa del motor 2 (Eje y) expresada en pasos.
	Parámetro 4	Descripción
	[n]	Número entero con signo 32 bits. Representa la coordenada relativa del motor 3 (Eje z) expresada en pasos.

Tabla 16. Comando POS

Ejemplo comando POS: “# POS 10000 10000 20000 :”

4 MANUAL DE USUARIO

La interfaz gráfica de usuario dispone de todos los controles necesarios para gestionar el control de los tres motores. En la Ilustración 25 se muestra el aspecto de dicha interfaz.

Ilustración 25. Interfaz gráfica de Usuario

A continuación, se explican las distintas posibilidades e información presentes en la interfaz.

4.1 Inicialización y Desconexión

En primer lugar, se debe inicializar la comunicación con el dispositivo mediante la opción Menú → Conectar. Tras iniciar la comunicación, se habilitan todas las funciones. Por ello, los motores pueden empezar a controlarse.

4.2 Muestreo de posición relativa y reseteo

La posición relativa (actual) del sistema se muestra en la parte superior de la pantalla, en unidades de micrómetros. Cabe destacar que debido a la ausencia de retroalimentación esta posición no refleja el valor absoluto, sino que se obtiene en base

a un punto de referencia preestablecido por el usuario. Para cambiar este punto de referencia se pulsa en “RESETEAR ORIGEN”, poniendo a cero la posición actual.

El controlador guarda una variable interna que indica la posición actual, ésta es enviada desde el controlador al ordenador cada 100 ms para actualizar el valor mostrado por pantalla. La Ilustración 26 muestra la parte de la interfaz gráfica en la que se indica la comentada posición asociada a cada motor.

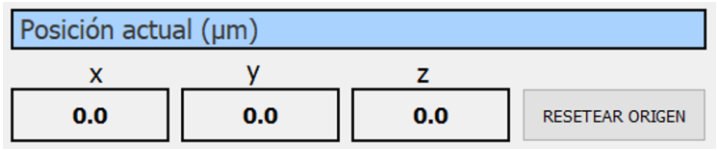


Ilustración 26. Display asociado a la posición del sistema

4.3 Control de posición manual

El cuadro de mando permite desplazar el microposicionador en todos los ejes, de forma manual y en tiempo real (Ilustración 27).

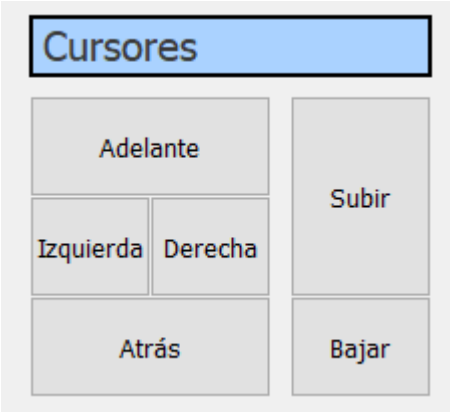


Ilustración 27. Cursores

A continuación, se describe la dirección y el sentido de giro de cada botón:

- Atrás: desplazamiento del eje Y negativo.

- Adelante: desplazamiento del eje Y positivo.

- Izquierda: desplazamiento del eje X negativo.

- Derecha: desplazamiento del eje X positivo.

- Bajar: desplazamiento del eje Z negativo.

- Subir: desplazamiento del eje Z positivo.

Además, se ha añadido la posibilidad de controlar el sistema mediante la pulsación de las teclas W, S, A, D, Q y R. La función de cada tecla de teclado viene descrita como:

- S: desplazamiento del eje Y negativo.

- W: desplazamiento del eje Y positivo.
- A: desplazamiento del eje X negativo.
- D: desplazamiento del eje X positivo.
- Q: desplazamiento del eje Z negativo.
- E: desplazamiento del eje Z positivo.

Además, el software permite pulsar dos teclas simultáneamente para conseguir desplazamientos diagonales.

4.4 Velocidad de los motores

La velocidad de los motores se puede variar mediante las barras situadas en la sección "Velocidad (µm/s). Cada eje es configurable de forma individual (Ilustración 28)

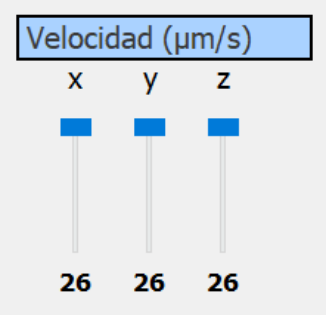


Ilustración 28. Control de velocidad

El software se ha configurado para permitir el control de velocidad entre un rango de 1 µm/s y 26 µm/s.

4.5 Coordenadas programadas

Aparte del control manual, la aplicación habilita la posibilidad de establecer siete coordenadas diferentes y acceder a cualquiera de ellas cuando sea necesario. Esta opción se añade con el objetivo de evitar el control manual en aquellos casos en los que un experimento necesita cambiar su posición a unas coordenadas conocidas.

En la Ilustración 29 se observa la parte de la interfaz que permite introducir las coordenadas y acceder a ellas.

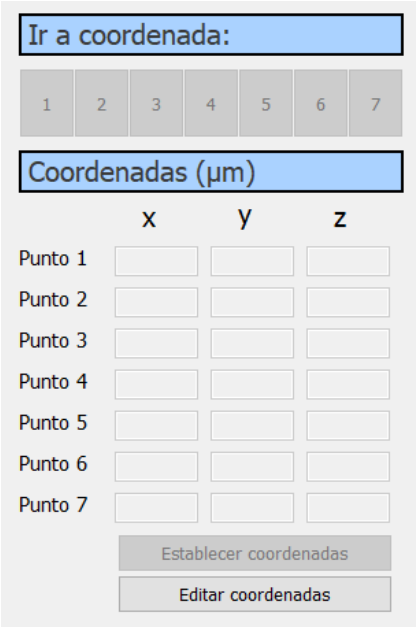


Ilustración 29. Coordenadas programadas

En primer lugar, se observa el campo “Ir a coordenada:”. Hay tantas coordenadas como puntos disponibles para programar en el apartado “Coordenadas (µm)” (siete en nuestro caso).

Para añadir una coordenada al sistema, primero se debe presionar en el botón “Editar coordenadas”. Tras esto, se habilitan todos los espacios en blanco para introducir dichas coordenadas en micrómetros. Tras finalizar, se presiona el botón “Establecer coordenadas”.

Aquellas coordenadas que dispongan de uno o más espacios vacíos, serán consideradas no válidas. Como consecuencia no se podrá acceder a ellas.

5 MONTAJE Y MEDIDAS EXPERIMENTALES

5.1 Montaje inicial

En primer lugar, se han verificado las funciones básicas del sistema. Para ello, se alimentó el controlador de motores con una fuente de voltaje de 12V y se conectó mediante un cable USB al ordenador. Tras la conexión del cable USB, se observa como parpadean los leds “LED2” y “LED3” (Ilustración 30) durante un breve periodo de tiempo. Esto indica que el ordenador ha reconocido el controlador de motores.

Ilustración 30. LED2 y LED3 FTDI

A continuación, se abre la aplicación y se inicializa la comunicación con el controlador (Menú → Conectar, Ilustración 31).

Ilustración 31. Menú de la aplicación

Una vez establecida la comunicación, el controlador de motores está listo para ser controlado por la aplicación. En este punto, se conectan los tres motores al controlador y se verifica que reacciona a los comandos enviados por la aplicación. Se comprueba, al menos, que los motores son capaces de girar en ambos sentidos, variar su velocidad de rotación y realizar movimientos programados con el menú “coordenadas programadas” de la aplicación.

Por último, se acopla mecánicamente el eje del motor con el eje del microposicionador para comprobar que éste desplaza su base móvil de acuerdo a las instrucciones dadas por el usuario a través de la aplicación.

5.2 Primera prueba

El sistema microposicionador se ha montado en el laboratorio con el fin de realizar pruebas de funcionamiento. Dichas pruebas tienen como objetivo comprobar la repetitividad de los movimientos. Así, se ha acoplado un motor al eje Y del microposicionador para poder controlar la posición de éste mediante la aplicación. Sobre la superficie del microposicionador se ha situado una muestra de silicio poroso. En la Ilustración 32 se muestra un diagrama simplificado de la prueba y en la Ilustración 33 la muestra de silicio poroso utilizada. Se trata de una pieza de silicio con perforaciones nanométricas, empleado a modo de sensor fotónico debido a su alta relación superficie/volumen. Las faltas de uniformidad presentes en la capa porosa permiten estimar la repetitividad del sistema de posicionamiento propuesto en este TFG, recorriendo una línea a lo largo de su superficie. La luz procedente de un diodo láser incide sobre esta muestra, y su reflejo es captado por un fotodiodo. En la Ilustración 34 y la Ilustración 35 se observa el montaje realizado.

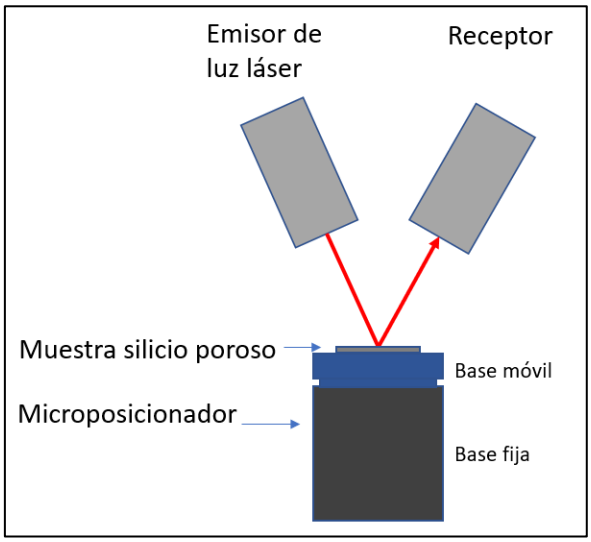


Ilustración 32. Experimento 1, diagrama simplificado

Ilustración 33. Muestra de silicio poroso

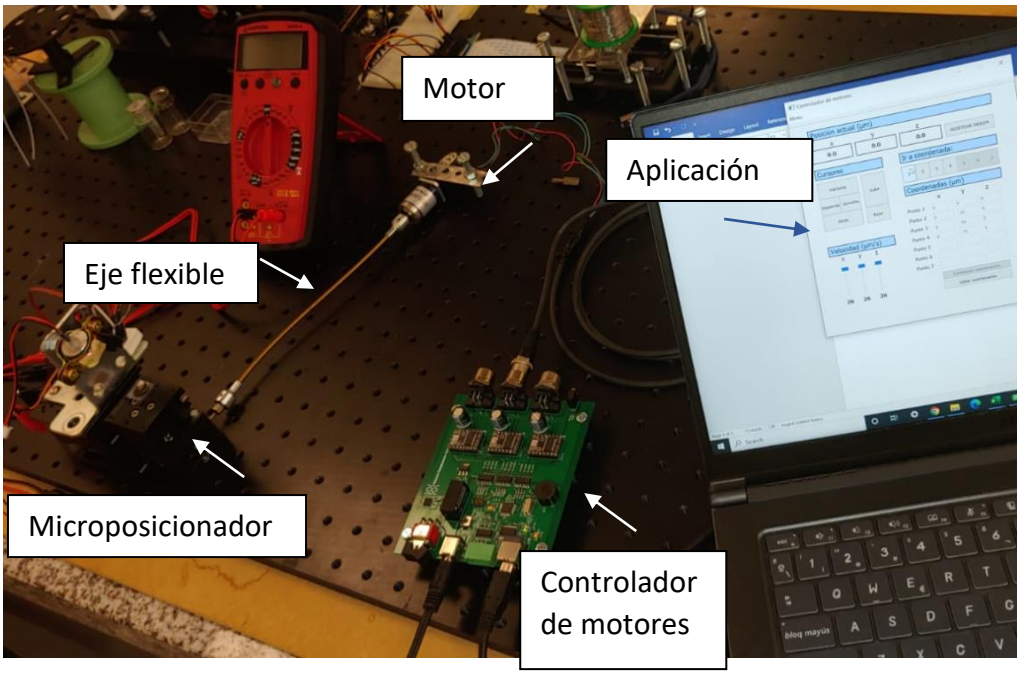


Ilustración 34. Experimento 1, vista general

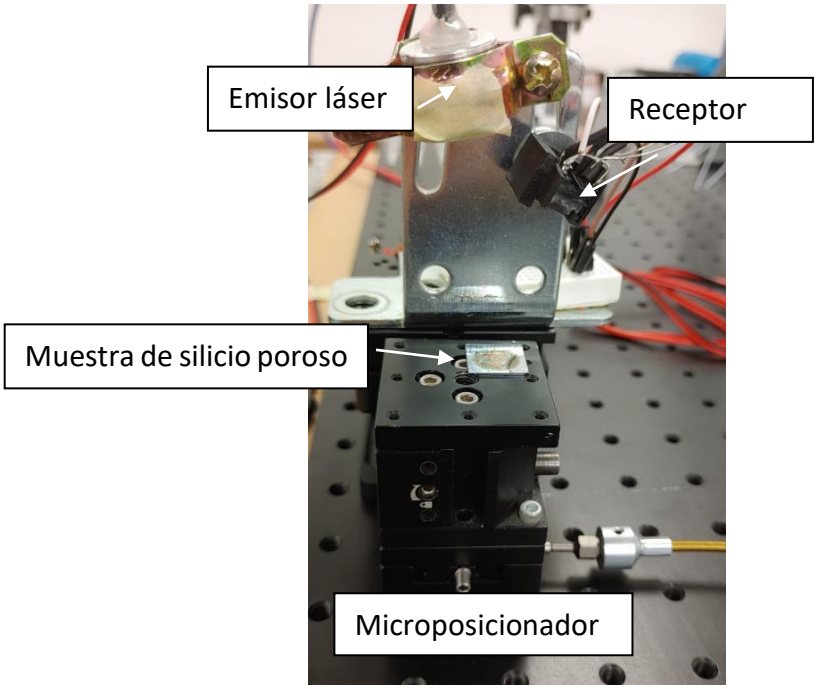


Ilustración 35. Muestra de silicio sobre microposicionador

El emisor de luz láser utilizado emite a 650 nm e incide sobre una zona específica de la muestra de silicio poroso (Ilustración 36). Previamente, se ha realizado una caracterización de la reflectividad en la zona que está siendo iluminada por el láser. Con la ayuda del espectrómetro Flame T de la marca Ocean Optics, se ha obtenido la reflectividad en el rango de 500 a 850 nm para la zona de estudio de la muestra de silicio poroso.

Tal y como se observa en la Ilustración 37, existe un mínimo de reflectividad cercano a la longitud de onda 650 nm. Por tanto, cualquier falta de uniformidad en la muestra se traduce en una notable variación en la reflectividad. La luz reflejada es captada por un fotodiodo, que ofrece a su salida un voltaje asociado al promediado de la reflectividad de la zona iluminada. Pequeños desplazamientos de la muestra se traducirán en variaciones de reflectividad que, a su vez, se traducirán en voltaje a la salida del fotodiodo.

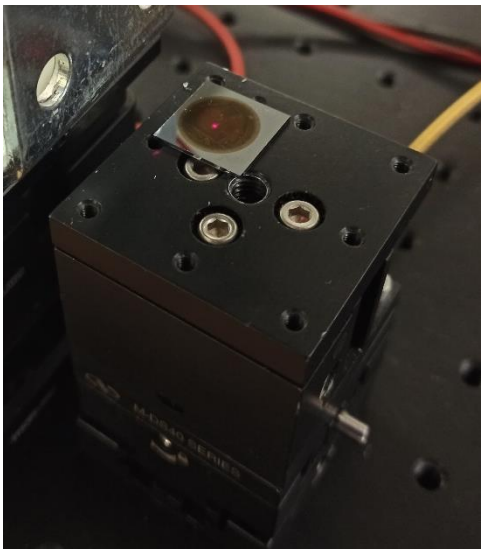


Ilustración 36. Luz láser sobre muestra de silicio poroso

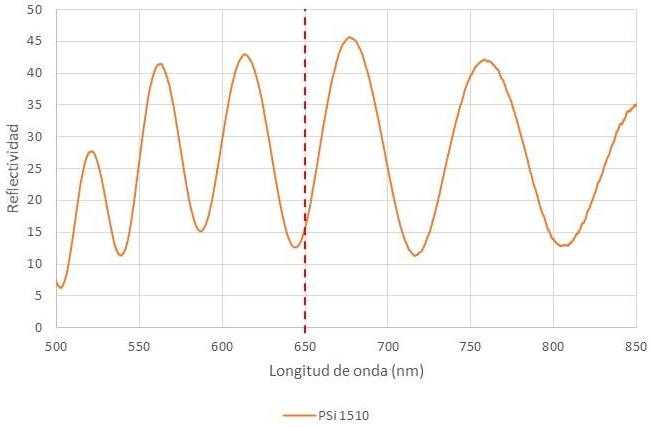


Ilustración 37. Reflectividad de la muestra de silicio poroso en la zona de estudio

La prueba consistió en llevar a cabo movimientos programados a lo largo del eje “Y”. En primer lugar, se partió de una posición inicial próxima al centro de la capa porosa, y se reseteó el origen de coordenadas. A partir de este punto, se desplazó la muestra 50 micrómetros, para luego volver a la posición de origen. Tal y como aparece en la Ilustración 38, al iluminar la muestra en el punto inicial, el voltaje inicial a la salida del fotodiodo era de 3.86V, llegando a ser de 2.63V al desplazarse los 50 micrómetros. Tras realizar el mismo movimiento en distintas ocasiones, se comprobó que el voltaje del fotodiodo tras el desplazamiento se mantenía en torno a los 3.85 ± 0.01V en el origen.

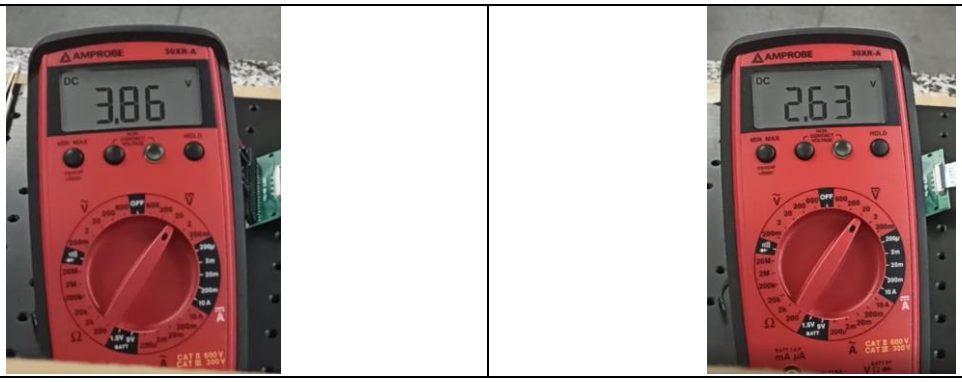


Ilustración 38. Tensiones medidas durante la prueba realizada

5.3 Segunda prueba

Esta prueba es similar al anterior: se toman medidas de reflectividad en diferentes posiciones y se comparan para comprobar la repetitividad del sistema posicionador. En esta ocasión, se ha utilizado una fuente de luz blanca, y la medida de reflectividad se ha realizado con espectrómetro de la casa Ocean Optics, modelo Flame T. El montaje puede observarse en la Ilustración 39.

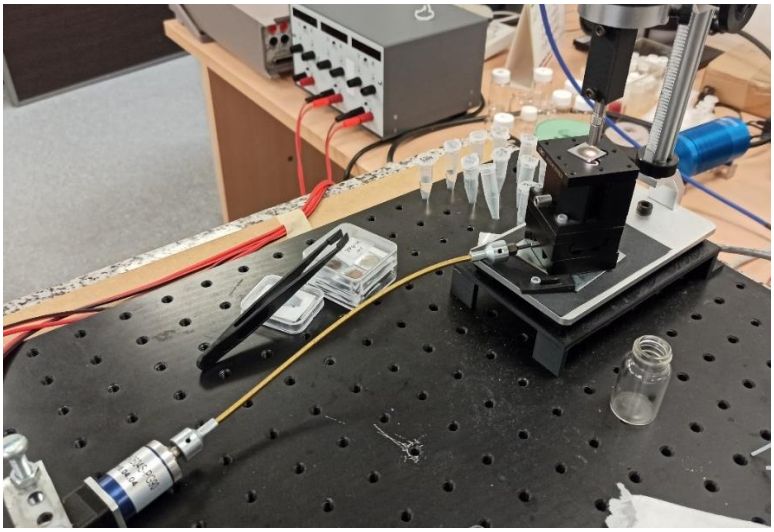


Ilustración 39. Experimento 2, vista general

El espectrómetro mide en tiempo real la reflectividad de la muestra en el rango comprendido entre 500 y 850 nm. De manera similar a lo comentado en el apartado anterior, al desplazar la muestra, la medida de reflectividad varía. En la Ilustración 40 puede observarse la reflectividad inicial de la capa de silicio poroso (en color naranja), y la obtenida tras un desplazamiento de 500 um (en color azul). La variación observada en el valor máximo de la reflectividad justifica la diferencia en las tensiones medidas en la prueba anterior, a la salida del fotodiodo.

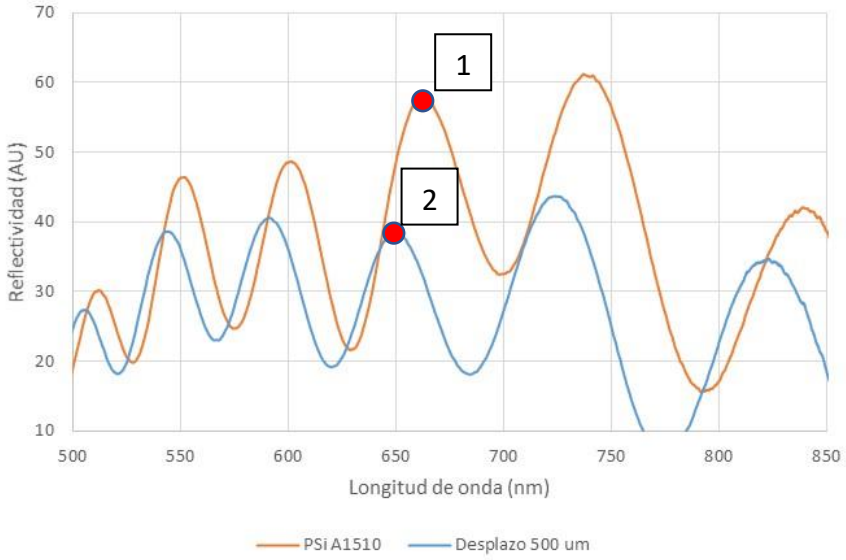


Ilustración 40. Reflectividad muestra de silicio poroso

La prueba ha consistido en realizar un seguimiento de la posición del máximo de reflectividad más próximo a 650 nm (Punto 1, Ilustración 40), mientras se produce el desplazamiento de la capa de silicio poroso. Tras realizar un desplazamiento de 500 µm, la posición del máximo de la reflectividad considerado se ha desplazado (Punto 2, Ilustración 40). Mediante un script de Matlab desarrollado en el Centro de Tecnología Nanofotónica, se monitoriza la posición de dicho máximo relativo a lo largo del recorrido realizado por el posicionador. De esta forma, se obtiene la gráfica mostrada en la Ilustración 41.

Ilustración 41. Longitud de onda del máximo relativo con respecto al tiempo

El primer tramo se corresponde con la posición inicial. Tras 80 segundos aproximadamente, se inicia el desplazamiento de 500 µm (segundo tramo). A continuación, se procedió a dejar el sistema detenido durante 120 segundos aproximadamente. Con ello se pretende comprobar si, estando detenido, hay alguna influencia del motor en la medida. Finalmente, la muestra vuelve a su posición inicial (tercer tramo).

De acuerdo con la Ilustración 41, si el posicionador en estado de reposo ofreciese algún tipo de vibración o ruido eléctrico, se apreciarían modificaciones en la posición del máximo de reflectividad. Con el objetivo de analizar en detalle este efecto, a continuación, se incluyen ampliaciones de los tres tramos comentados en el párrafo anterior.

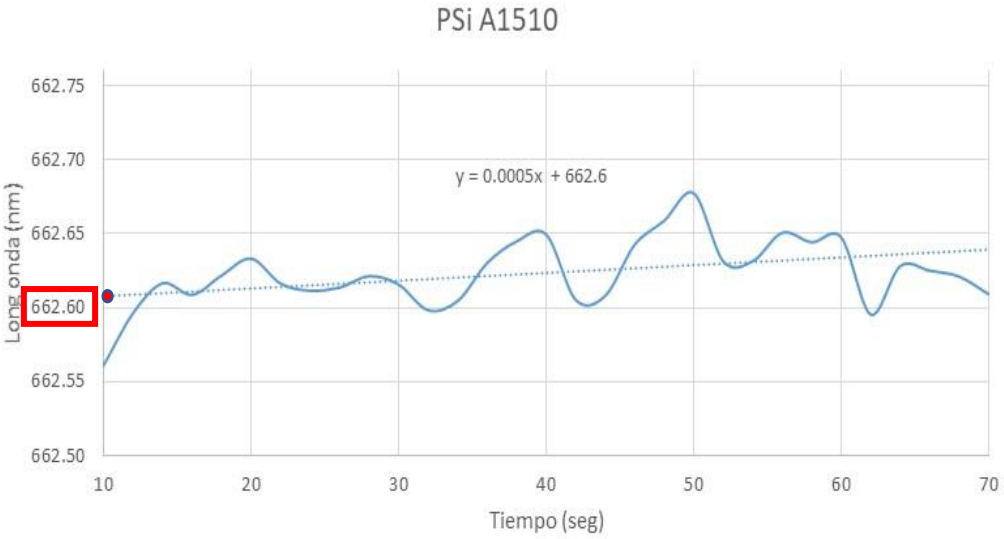


Ilustración 42. Longitud de onda con respecto al tiempo. Tramo 1: Posición inicial.

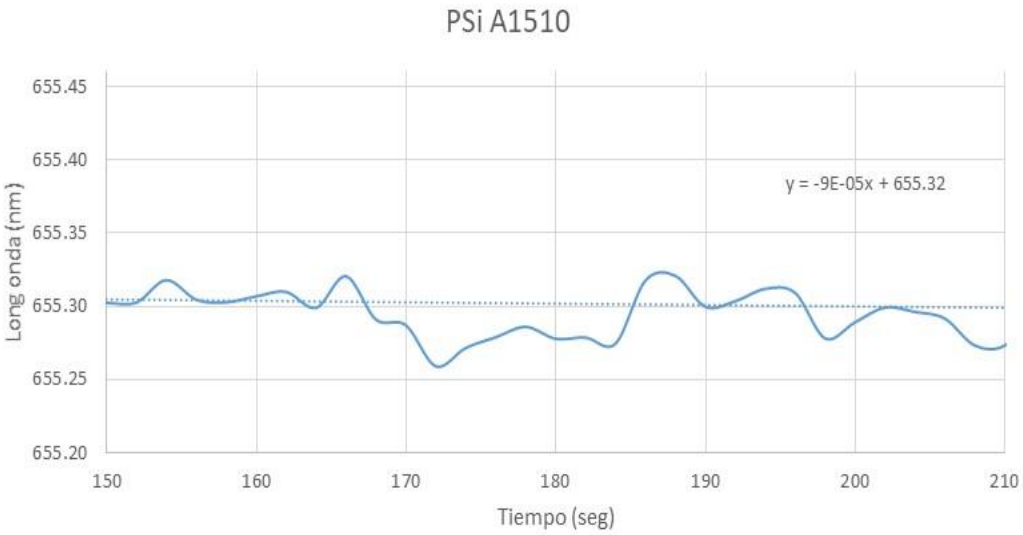


Ilustración 43. Longitud de onda con respecto al tiempo. Tramo 2: Tras desplazamiento.

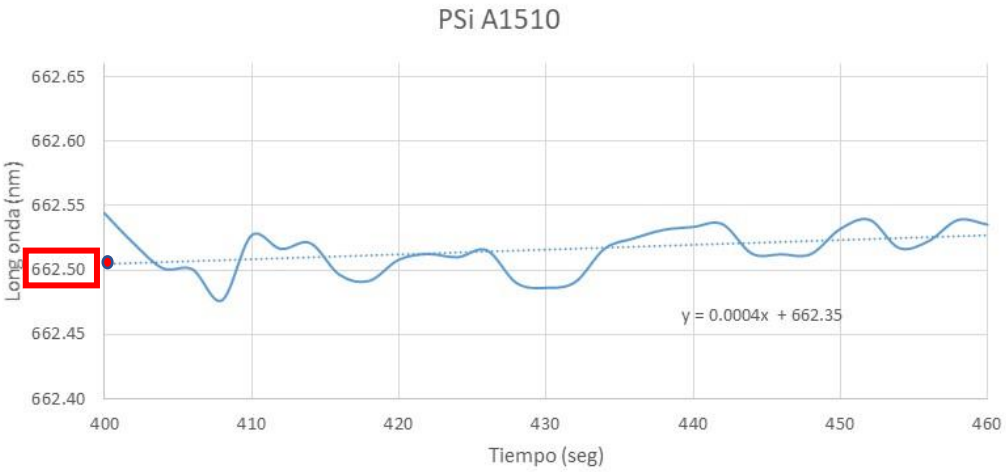


Ilustración 44. Longitud de onda con respecto al tiempo. Tramo 3: Tras la vuelta a la posición inicial.

La pendiente observada en todos los tramos es inferior a 0.5 pm/seg. Además, las desviaciones observadas durante la medida de cada tramo están por debajo de los 100 pm. Estos valores están dentro del error de medida del espectrómetro. Por lo tanto y en estado de reposo, la influencia del motor en la medida de caracterización es nula.

Por otro lado, se aprecian leves desviaciones en el momento de activar el motor. Éstas han tenido lugar aproximadamente en los segundos 85 y 355 de la Ilustración 41. Pueden deberse al "backslash" del engranaje planetario, término utilizado para referirse a la holgura entre los dientes del engranaje y que introduce un error de posición al efectuar un cambio de sentido de giro. El eje flexible que se utiliza como acoplamiento mecánico entre el motor y el microposicionador puede introducir un error del mismo tipo, por lo que se hace necesario un mayor estudio de este efecto, de cara a compensar por software sus posibles consecuencias en el desplazamiento total de la muestra.

6 CONCLUSIONES

El presente TFG ha abordado el diseño y realización de un sistema microposicionador electrónico de tres ejes. El proyecto abarca un amplio espectro de desarrollos y tareas: diseño de esquemas electrónicos y layout de PCBs, firmware, software para creación de interfaces gráficas, selección de motores, e integración mecánica de ellos en un sistema microposicionador.

El diseño hardware cuenta con características que le dotan de una alta robustez, como son las múltiples protecciones (sobrevoltaje, polaridad inversa, descarga electroestática), el aislamiento del control digital de la etapa de potencia y otros elementos como el uso de conectores industriales.

Tras comprobar el funcionamiento del circuito, se ha demostrado la capacidad de realizar un control sencillo, robusto y de bajo coste con herramientas de software libres de licencia, por lo que el proyecto es fácilmente replicable en cualquier otro laboratorio.

Las primeras pruebas realizadas en el laboratorio han sido satisfactorias. Inicialmente se comprobó que el sistema es capaz de accionar correctamente cada uno de los tres ejes del microposicionador. Tras esto, se continuó el resto de las pruebas con uno de ellos. Además, las coordenadas programadas han funcionado como se esperaba.

Sin embargo, el objetivo del TFG no radica exclusivamente en el diseño y montaje de un controlador de motores paso a paso, sino también en la elección de los componentes externos para motorizar un microposicionador manual, cuyos movimientos deben ser repetitivos. Por tanto, la realización y el estudio de movimientos programados ha sido relevante, ya que permite caracterizar la desviación propia de un desplazamiento de ida y vuelta.

El sistema puede realizar movimientos con una resolución micrométrica, sólo limitada por las características del propio microposicionador. Para ello, se ha escogido un motor paso a paso con engranajes planetarios acoplados a su eje cuya reducción permite obtener ángulos de paso de 0.02°. El microposionador utilizado se desplaza 318 µm por cada vuelta que realiza su tornillo sin fin, por lo que se tiene una resolución de desplazamiento notablemente inferior al micrómetro. Sin embargo, no se pretende con ello garantizar un correcto desempeño por debajo de la escala micrométrica debido a los errores que se introducen al utilizar mecanismos con engranajes o acoples mecánicos a un eje. Con un tamaño de paso tan reducido, se pretende, por tanto, mejorar notablemente el desempeño en la fluidez del movimiento, evitando así la introducción de perturbaciones mecánicas.

En las primeras pruebas, el sistema demostró movimientos repetitivos en el rango de 50-500 µm. No obstante, para trabajar en rangos inferiores, se debe tener en cuenta las dos principales fuentes de error a la hora de conseguir un movimiento repetitivo: el engranaje planetario y el acoplamiento mecánico entre el motor y el microposicionador. Ambos introducen un error de movimiento al cambiar el sentido de giro. Por ello, el mayor reto radica en mitigar este error.

Por una parte, el engranaje planetario introduce un error conocido como “backslash”, típico de los sistemas con engranajes. Por otra parte, el acoplamiento mecánico entre el motor y el microposicionador, necesita un ángulo de recuperación para ejercer momento en sentido contrario. Una posible solución pasa por caracterizar el error asociado al cambio de sentido y corregirlo por software mediante la introducción de un offset.

En este punto, se identifican dos vías para la continuación/mejora del TFG:

1. Diseño y realización de nuevos experimentos para caracterizar la repetitividad del sistema.
2. Actualización de la aplicación de usuario para compensar los errores asociados al cambio del sentido de giro mediante la introducción de un offset de movimiento.

7 PRESUPUESTO

Tabla 17. Presupuesto planificación, diseño y ejecución del proyecto por parte de un Ingeniero

Descripción	**Precio (eur/hora)**	**Horas**	**Precio (eur)**
Planificación del proyecto	40.00	18	720.00
Diseño de bloques y selección de componentes	40.00	40	1600.00
Desarrollo de software y firmware	40.00	32	1280.00
Montaje de sistema	20.00	16	320.00
Validación	40.00	8	320.00
		Subtotal	**4240.00**
21% IVA			890.40
		Total	**5130.40**

Tabla 18. Presupuesto: materiales

Descripción	**Precio ud (eur)**	**Unidades**	**Precio (eur)**
Componentes electrónicos*	77.97	1	77.97
Circuito impreso	19.75	1	19.75
Cables para motores	5.53	3	16.59
Motores	31.60	3	94.80
Acopladores mecánicos	27.65	3	82.95
Microposicionador	355.50	1	355.50
Fuente de alimentación 12V	23.70	1	23.70
Accesorios	25.00	1	25.00
		Subtotal	**696.26**
21% IVA			146.22
		Total	**842.48**

*Se ha desglosado la lista de componentes electrónicos en el Anexo I.

Tabla 19. Resumen del presupuesto

Descripción	**Precio (eur)**
Diseño y ejecución del proyecto	4240.00
Materiales	696.26
Subtotal	**4936.26**
21% IVA	1036.61
Total	**5972.87**

El presupuesto general asciende a la cantidad de CINCO MIL NOVECIENTOS SETENTA Y DOS EUROS CON OCHENTA Y SIETE CÉNTIMOS, IVA incluido.

8 PLIEGO DE CONDICIONES

8.1 Alimentación del sistema

En el siguiente apartado se recogen las condiciones relativas a la alimentación del sistema, en las cuales se incluye la fuente de alimentación externa, así como también los circuitos de protección y acondicionamiento de ésta para garantizar un correcto funcionamiento.

- La alimentación del sistema deberá proporcionar un voltaje comprendido en el rango de 12-24V y ser capaz de entregar 5A.
- El controlador de motores debe disponer de protecciones contra polaridad inversa.
- El controlador de motores debe disponer de protección contra sobretensiones de 36V. La alimentación debe ser cortada si se supera dicho umbral de tensión.
- El controlador de motores debe detener su funcionamiento en caso de ser alimentado por una tensión inferior a 8V.
- Las alimentaciones de los bloques digitales y analógicos estarán aisladas galvánicamente.
- Será necesario incorporar un interruptor para controlar el encendido/apagado del sistema.

8.2 Motor

El motor seleccionado ha de cumplir con una serie de características que le permitan accionar el eje del microposicionador. Además, ha de ser compatible con el controlador de motores empleado. Los requerimientos se detallan a continuación:

- El tipo de motor debe ser paso a paso bipolar.
- Debe ser capaz de ejercer un momento de 9.8 N/cm.
- Debe permitir ángulos de giros inferiores a 0.36° mediante reductora acoplada al eje.
- La corriente por bobina ha de ser inferior a 1A.

8.3 Controlador de motores

Respecto al microcontrolador empleado, se deben cumplir las siguientes especificaciones técnicas:

- Conexión USB para poder establecer el control desde el PC.
- Capacidad para gestionar el movimiento de tres motores paso a paso simultáneamente.
- El controlador de motores deberá ser capaz de entregar una corriente de 1A por bobina en cada canal.
- Permitir reprogramar el controlador mediante el puerto USB, sin necesidad de programadores específicos para tal función.

8.4 Software

El software empleado para controlar los motores debe cumplir con los siguientes requerimientos:

- Se debe proveer una interfaz gráfica de usuario que permita controlar la velocidad y la posición de los tres motores.
- El software permitirá definir y guardar coordenadas específicas, así como también acceder a ellas pulsando un botón.
- El control de posición se realizará tanto desde la interfaz gráfica como desde el teclado.
- Se debe mostrar la posición actual del sistema con un refresco de pantalla inferior a 500 ms.
- La comunicación con el controlador de motores se efectuará mediante el envío de cadenas de caracteres a través del puerto USB.
- Se debe garantizar que el software conserva todos los parámetros tras cerrar el programa.
- Se realizará con software libre de licencia de pago.

9 BIBLIOGRAFÍA

Todos los enlaces presentes a continuación han sido consultados entre Enero y Agosto del año 2021:

[1] https://harmonicdrive.de/es/glosario/servomotor

[2] https://www.tme.eu/es/news/library-articles/page/41861/Motor-paso-a-paso-tipos-y-ejemplos-del-uso-de-motores-paso-a-paso

[3] https://www.pololu.com/category/120/stepper-motor-drivers

[4] https://www.ni.com/es-es/shop/labview.html

[5] https://www.qt.io/product/development-tools

[6] https://www.hammfg.com/

[7] https://www.microchip.com/wwwproducts/en/ATmega328P

[8] https://www.luisllamas.es/usar-arduino-para-reprogramar-el-bootloader/

[9] https://www.pololu.com/product/2133

[10] https://www.allpcb.com/

[11] https://www.motioncontroltips.com/stepper-drives-whats-the-difference-between-an-l-r-drive-and-a-chopper-drive/

[12] https://www.prometec.net/unipolar-bipolar/

ANEXOS

Anexo I: Lista de materiales circuito impreso

#	Cantidad	Referencia	Descripción	Fabricante	Referencia fabricante	Distribuidor	Ref Distribuidor	Precio ud (eur)	Precio total (eur)
1	1	BZ1	Active buzzer	Kingstate	KPEG242	Farnell	1502726	2.57	2.57
2	4	C1, C2, C4, C5	CAP CER 0.1UF 16V X7R 0603	Samsung	CL10B104KO8NNNC	Digikey	1276-1005-1-ND	0.08	0.32
3	2	C3, C8	CAP TANT 4.7UF 20% 16V 1206	AVX Corporation	TAJA475M016RNJ	Digikey	478-3032-1-ND	0.31	0.62
4	2	C6, C7	CAP CER 1UF 25V X5R 0603	AVX Corporation	06033D105KAT2A	Digikey	478-5142-1-ND	0.17	0.34
5	3	C9, C10, C11	CAP ALUM 100UF 20% 50V RADIAL	Nichicon	URS1H101MPD	Digikey	493-15907-ND	0.43	1.29
6	2	C12, C13	Condensador 22 pF, 25 V, 0603	Wurth Elektronik	06033A220JAT2A	Farnell	2533830	0.05	0.1
7	1	C14	Capacitor 2.2 µF, 50 V, 0603	Murata	GRT188R61H225KE13D	Farnell	2672161	0.17	0.17
8	1	C15	Capacitor 4.7 µF, 16 V, 0603	TDK	C1608X5R1C475K080AC	Farnell	2211173	0.14	0.14
9	1	D1	Rectifier Diode DO-219AB	Vishay	RS07D-HM3-08	Farnell	3527854	0.46	0.46
10	3	DRV1, DRV2, DRV3	Stepper motor driver Pololu DRV8825	Pololu	Pololu-DRV8825	Pololu	Pololu-DRV8825	8.95	26.85
11	1	F1	SMD PTC 500 mA	Bourns	MF-FSMF050X-2	Digikey	MF-FSMF050X-2CT-ND	0.35	0.35
12	4	ISO1, ISO2, ISO3, ISO4	Quad Optocoupler SSOP16	Vishay	TCMT4100	Farnell	1779650	2.68	10.72
13	1	J1	PC MOUNTABLE JACK FOR 5.5X2.5MM	GlobTek	JACK-L-PC-10A-RA(R)	Digikey	1939-1087-ND	2.63	2.63
14	1	J2	CONN RCPT USB2.0 TYPEB 4POS R/A	TE Connectivity	292317-4	Digikey	A115152-ND	2.23	2.23
15	1	J3	Connector pitch 2.54 02x03	Samtec	TLW-103-05-G-D	Farnell	1668489	0.94	0.94
16	3	J4, J5, J6	CONN PLUG FMALE 4POS GOLD SOLDER	TE Connectivity	T4145015041-001	Digikey	A126523-ND	6.11	18.33
17	1	J7	Conector block pluggable pcb 6 pos	Wurth Elektronik	691382010006	RS	122-4925	0.01	0.01
18	1	LED1	LED 2HI 3MM RED/GRN BICLR PC MNT	Dialight	5530711F	Digikey	350-1834-ND	2.54	2.54

19	2	LED2, LED3	Green LED SMD 0805	Kinbright	KP-2012CGCK	Farnell	2290331	0.174	0.34
20	1	Q1	Transistor NPN SOT-23	Nexperia	PBSS4350T	RS	518-1671	0.23	0.23
21	1	R1	Resistor 4.42 Mohm, ± 1%, 100 mW, 0603	VISHAY	CRCW06034M42FKEA	Farnell	2138685	0.05	0.05
22	1	R2	Resistor 220 kohm, ± 1%, 100 mW, 0603	YAGEO	RC0603FR-07220KL	Farnell	9238760	0.02	0.02
23	1	R3	Resistor 80.6 kohm, ± 1%, 100 mW, 0603	VISHAY	CRCW060380K6FKEA	Farnell	2138489	0.04	0.04
24	1	R4	Resistor 100k ohm 0603 1% 0.1W	TE Connectivity	CRG0603F100K	RS	213-2531	0.01	0.01
25	16	R5, R6, R7, R8, R9, R24, R31, R33, R35, R36, R37, R38, R42, R43, R44, R45	Resistor 1k ohm 0603 5% 0.1W	Vishay	CRCW06031K00JNEAIF	RS	820-6953	0.07	1.12
26	13	R10, R23, R25, R26, R27, R28, R29, R30, R32, R34, R39, R40, R41, R46	Resistor 10k ohm 0603 1% 0.1W	Bourns	CR0603-FX-1002ELF	RS	740-8805	0.02	0.26
27	12	R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22	Resistor 680 ohm 0603 1% 0.25W	Vishay	CRCW0603680RFKEAHP	RS	820-6887	0.03	0.36
28	1	SW1	ROCKER SWITCH DPDT, THROUGH HOLE, R/A	TE Connectivity	1571987-2	Digikey	450-2384-ND	3.18	3.18
29	1	SW2	Tactile switch SPST	OMRON	B3FS-1000	Farnell	3121161	0.18	0.18
30	1	U1	Dual MOSFET N, 40 V, 40 A, 4.8 mohm, LFPAK56D	Nexperia	BUK7K6R2-40EX	Farnell	3439669	1.36	1.36
31	1	U2	IC Overvoltage, Undervoltage and Reverse Polarity Protection	Analog Devices	LTC4365HTS8#TRMPBF	Mouser	584-C4365HTS8TRMPBF	4.08	4.08
32	1	U3	IC REG LINEAR 5V 150MA SOT223-4	Texas Instruments	TPS7B6950QDCYRQ1	Digikey	296-40985-1-ND	0.82	0.82
33	1	U4	DCDC Converter Isolated 9-36V Input 5V Output	Traco Power	TEC 2-2411WI	Farnell	2854919	9.17	9.17
34	1	U5	IC USB FS SERIAL UART 28-SSOP	FTDI	FT232RL	Digikey	768-1007-1-ND	3.8	3.8
35	1	U6	IC MCU 8BIT 32KB FLASH 32TQFP	Microchip	Atmega328P-AU	Digikey	ATMEGA328P-AU-ND	1.92	1.92
36	1	U7	3 input AND logic gate	Nexperia	74LVC1G11GW-Q100H	Farnell	2445095	0.378	0.37
37	1	U8	USB 2.0 ESD protection	Nexperia	PRTR5V0U2X	Farnell	1524157	0.557	0.55
38	1	Y1	Crystal 16 Mhz 18 pF	TXC	9B-16.000MEEJ-B	Farnell	1842217	0.477	0.47

Anexo II: Código fuente firmware

```
#include <string.h>
#include <stdlib.h>
#define DISABLE 0x00
#define ENABLE 0x01
#define LOWSTAT 0x00
#define HIGHSTAT 0x01

#define pinDir_motor1 8
#define pinDir_motor2 5
#define pinDir_motor3 A1
#define pinStep_motor1 9
#define pinStep_motor2 6
#define pinStep_motor3 3
#define pinEN_motor1 10
#define pinEN_motor2 7
#define pinEN_motor3 4
#define pinMod0 13
#define pinMod1 12
#define pinMod2 11
#define C 0x00
#define CC 0x01
#define MODLOW 0x01
#define MODHIGH 0x00



// ---------------------------------------------------------------------------------------
// interrupt frequency
const float samplerate = 100000.0f;
uint32_t count = 0;
// ---------------------------------------------------------------------------------------
long waitCount_serialWrite,period_serialWrite;
long waitCount_motor1,waitCount_motor2,waitCount_motor3;
long previousCount_motor1,previousCount_motor2,previousCount_motor3,previousCount_serialWrite;
long actualCount=0;
//----------------------------------------------------------------------------------------
boolean outputState_motor1=LOW,outputState_motor2=LOW,outputState_motor3=LOW;

long ppsMot1=200,ppsMot2=200,ppsMot3=200;
//----------------------------------------------------------------------------------------
//DEBUG
long debug_before_serial,debug_after_serial;
long delta_motor1,delta_motor2,delta_motor3,point_motor1=0,point_motor2=0,point_motor3=0;
long steps_number_motor1=0,steps_number_motor2=0,steps_number_motor3=0;
char string_time[10];
long points[10][4];
//


byte MotorStop=0xFF;
byte Direction=0x00;
bool ModePoint=false;
bool Initialized=false;

const byte numChars = 32;
char receivedChars[numChars];   // an array to store the received data

boolean newData = false;

    int i = 0;
    int ArraySize = 0;
    int ArrayIndex = 0;
    char* ArrayString[7];


void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);

pinMode(pinStep_motor1,OUTPUT);
pinMode(pinStep_motor2,OUTPUT);
pinMode(pinStep_motor3,OUTPUT);
pinMode(pinDir_motor1,OUTPUT);
pinMode(pinDir_motor2,OUTPUT);
pinMode(pinDir_motor3,OUTPUT);
pinMode(pinEN_motor1,OUTPUT);
pinMode(pinEN_motor2,OUTPUT);
pinMode(pinEN_motor3,OUTPUT);
pinMode(pinMod0,OUTPUT);
pinMode(pinMod1,OUTPUT);
pinMode(pinMod2,OUTPUT);

digitalWrite(pinStep_motor1,outputState_motor1);
digitalWrite(pinStep_motor2,outputState_motor2);
```

```
digitalWrite(pinStep_motor3,outputState_motor3);
digitalWrite(pinDir_motor1,CC);
digitalWrite(pinDir_motor2,CC);
digitalWrite(pinDir_motor3,CC);
digitalWrite(pinEN_motor1,ENABLE);
digitalWrite(pinEN_motor2,ENABLE);
digitalWrite(pinEN_motor3,ENABLE);
digitalWrite(pinMod0,MODLOW);
digitalWrite(pinMod1,MODLOW);
digitalWrite(pinMod2,MODLOW);

period_serialWrite=100; //time in milliseconds

waitCount_motor1=50000/ppsMot1;
waitCount_motor2=50000/ppsMot2;
waitCount_motor3=50000/ppsMot3;
waitCount_serialWrite=(long)100*period_serialWrite;  //counts to wait to execution of serialWrite

// initialize timer1
noInterrupts(); // disable all interrupts
TCCR1A = 0;
TCCR1B = 0;
TCNT1 = 0;
OCR1A = 16000000.0f / samplerate; // compare match register for IRQ with selected samplerate
TCCR1B |= (1 << WGM12); // CTC mode
TCCR1B |= (1 << CS10); // no prescaler
TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
interrupts(); // enable all interrupts

}

//INTERRUPT ----------------------------------------
// timer 1 interrupt
ISR(TIMER1_COMPA_vect)
{
  // generate the next clock pulse on accumulator overflow
actualCount++;

}
// ----------------------------------------------------


void loop() {

  recvWithEndMarker();

if(Initialized==false){
digitalWrite(pinEN_motor1,DISABLE);
digitalWrite(pinEN_motor2,DISABLE);
digitalWrite(pinEN_motor3,DISABLE);
}


if(Initialized==true){
digitalWrite(pinEN_motor1,ENABLE);
digitalWrite(pinEN_motor2,ENABLE);
digitalWrite(pinEN_motor3,ENABLE);

if(ModePoint==true){
  if(steps_number_motor1>=point_motor1)Direction=Direction|(1<<0);//CC
else Direction=Direction&~(1<<0);//C

if(steps_number_motor2>=point_motor2)Direction=Direction|(1<<1);//CC
else Direction=Direction&~(1<<1);//C

if(steps_number_motor3>=point_motor3)Direction=Direction|(1<<2);//CC
else Direction=Direction&~(1<<2);//C

delta_motor1=point_motor1-steps_number_motor1;
delta_motor2=point_motor2-steps_number_motor2;
delta_motor3=point_motor3-steps_number_motor3;

if(abs(delta_motor1)<=0)MotorStop=MotorStop|(1<<0);
else MotorStop=MotorStop&~(1<<0);
if(abs(delta_motor2)<=0)MotorStop=MotorStop|(1<<1);
else MotorStop=MotorStop&~(1<<1);
if(abs(delta_motor3)<=0)MotorStop=MotorStop|(1<<2);
else MotorStop=MotorStop&~(1<<2);
}



if(((Direction&(1<<0))>>0)==LOWSTAT){
  digitalWrite(pinDir_motor1,C);
}
if(((Direction&(1<<0))>>0)==HIGHSTAT){
  digitalWrite(pinDir_motor1,CC);
```

```
}
if(((Direction&(1<<1))>>1)==LOWSTAT){
  digitalWrite(pinDir_motor2,C);
}
if(((Direction&(1<<1))>>1)==HIGHSTAT){
  digitalWrite(pinDir_motor2,CC);
}
if(((Direction&(1<<2))>>2)==LOWSTAT){
  digitalWrite(pinDir_motor3,C);
}
if(((Direction&(1<<2))>>2)==HIGHSTAT){
  digitalWrite(pinDir_motor3,CC);
}


if(((MotorStop&(1<<0))>>0)==HIGHSTAT)digitalWrite(pinStep_motor1,outputState_motor1);
else if(((MotorStop&(1<<0))>>0)==LOWSTAT){
  if((actualCount-previousCount_motor1)>=waitCount_motor1){
  previousCount_motor1=actualCount;
  outputState_motor1=!outputState_motor1;
  digitalWrite(pinStep_motor1,outputState_motor1);
  if((outputState_motor1==HIGHSTAT)&&(((Direction&(1<<0))>>0)==LOWSTAT))steps_number_motor1++;
  else if((outputState_motor1==HIGHSTAT)&&(((Direction&(1<<0))>>0)==HIGHSTAT))steps_number_motor1--;
}
}

if(((MotorStop&(1<<1))>>1)==HIGHSTAT)digitalWrite(pinStep_motor2,outputState_motor2);
else if(((MotorStop&(1<<1))>>1)==LOWSTAT){
  if((actualCount-previousCount_motor2)>=waitCount_motor2){
  previousCount_motor2=actualCount;
  outputState_motor2=!outputState_motor2;
  digitalWrite(pinStep_motor2,outputState_motor2); //toggle
  if((outputState_motor2==HIGHSTAT)&&(((Direction&(1<<1))>>1)==LOWSTAT))steps_number_motor2++;
  else if((outputState_motor2==HIGHSTAT)&&(((Direction&(1<<1))>>1)==HIGHSTAT))steps_number_motor2--;
  }
}

if(((MotorStop&(1<<2))>>2)==HIGHSTAT)digitalWrite(pinStep_motor3,outputState_motor3);
else if(((MotorStop&(1<<2))>>2)==LOWSTAT){
  if((actualCount-previousCount_motor3)>=waitCount_motor3){
  previousCount_motor3=actualCount;
  outputState_motor3=!outputState_motor3;
  digitalWrite(pinStep_motor3,outputState_motor3); //toggle
  if((outputState_motor3==HIGHSTAT)&&(((Direction&(1<<2))>>2)==LOWSTAT))steps_number_motor3++;
  else if((outputState_motor3==HIGHSTAT)&&(((Direction&(1<<2))>>2)==HIGHSTAT))steps_number_motor3--;
}
}

if((actualCount-previousCount_serialWrite)>=waitCount_serialWrite){
previousCount_serialWrite=actualCount;

Serial.print("#");
Serial.print(" ");
Serial.print("POS");
Serial.print(" ");
Serial.print(steps_number_motor1);
Serial.print(" ");
Serial.print(steps_number_motor2);
Serial.print(" ");
Serial.print(steps_number_motor3);
Serial.print(" ");
Serial.print(":");
}

}

}



void recvWithEndMarker() {
    static byte ndx = 0;
    char endMarker = '\n';
    char rc;

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

        if (rc != endMarker) {
            receivedChars[ndx] = rc;
            ndx++;
            if (ndx >= numChars) {
                ndx = numChars - 1;
            }
        }
        else {
```

```
            receivedChars[ndx] = '\0'; // terminate the string
            ndx = 0;
            newData = true;
        }
    }
     if (newData == true) {
  char *token = strtok(receivedChars," ");
    while (token != NULL)
    {
      ArrayString[ArrayIndex]=token;
      ArrayIndex++;
        //Serial.println(token);
        token = strtok(NULL," ");
    }
newData = false;
   ArrayIndex=0;
//************************
//********** MOV **********
if(strcmp(ArrayString[0],"MOV")==0){
  ModePoint=false;
  if (strcmp(ArrayString[1],"MOT1")==0){
      MotorStop=MotorStop&~(1);
        if(strcmp(ArrayString[2],"C")==0){
        Direction=Direction&~(1);
    }
        else if(strcmp(ArrayString[2],"CC")==0){
        Direction=Direction|(1);
    }
  }
  else if (strcmp(ArrayString[1],"MOT2")==0){
      MotorStop=MotorStop&~(1<<1);
        if(strcmp(ArrayString[2],"C")==0){
        Direction=Direction&~(1<<1);
    }
        else if(strcmp(ArrayString[2],"CC")==0){
        Direction=Direction|(1<<1);
    }
  }
  else if (strcmp(ArrayString[1],"MOT3")==0){
      MotorStop=MotorStop&~(1<<2);
        if(strcmp(ArrayString[2],"C")==0){
        Direction=Direction&~(1<<2);
    }
        else if(strcmp(ArrayString[2],"CC")==0){
        Direction=Direction|(1<<2);
    }
  }
}
//********** MOV **********
//************************
//********** VEL **********

if(strcmp(ArrayString[0],"VEL")==0){
  if (strcmp(ArrayString[1],"MOT1")==0){
    ppsMot1=atoi(ArrayString[2]);
  }
  else if (strcmp(ArrayString[1],"MOT2")==0){
    ppsMot2=atoi(ArrayString[2]);
  }
  else if (strcmp(ArrayString[1],"MOT3")==0){
    ppsMot3=atoi(ArrayString[2]);
  }

  waitCount_motor1=50000/ppsMot1;
  waitCount_motor2=50000/ppsMot2;
  waitCount_motor3=50000/ppsMot3;
}
//********** VEL **********
//************************
//********** STOP **********

if(strcmp(ArrayString[0],"STOP")==0){
  if (strcmp(ArrayString[1],"MOT1")==0){
  MotorStop=MotorStop|(1<<0);
  }
  else if (strcmp(ArrayString[1],"MOT2")==0){
  MotorStop=MotorStop|(1<<1);
  }
  else if (strcmp(ArrayString[1],"MOT3")==0){
  MotorStop=MotorStop|(1<<2);
  }
}
//*********** STOP *************
//******************************
//*********** POINTS ************

if(strcmp(ArrayString[0],"P")==0){
```

```
  for(int i=0;i<3;i++) points[atoi(ArrayString[1])-1][i]=atol(ArrayString[i+2]);//parsea el buffer y
guarda las coordenadas en el array.

}

if(strcmp(ArrayString[0],"GOTO")==0){
  ModePoint=true;

  point_motor1=points[atoi(ArrayString[1])-1][0];
  point_motor2=points[atoi(ArrayString[1])-1][1];
  point_motor3=points[atoi(ArrayString[1])-1][2];

}
//*********** POINTS ************
//*******************************
//*********** ORIGIN ************

if(strcmp(ArrayString[0],"ORIGIN")==0){
ModePoint=false;
steps_number_motor1=0;
steps_number_motor2=0;
steps_number_motor3=0;
}
//*********** ORIGIN ************
//*******************************
//*********** POS ACTUAL ************
if(strcmp(ArrayString[0],"POS")==0){
   steps_number_motor1=atoi(ArrayString[1]);
   steps_number_motor2=atoi(ArrayString[2]);
   steps_number_motor3=atoi(ArrayString[3]);

}

//*********** POS ACTUAL ************
//*******************************
//*********** CONNECT/DISCONNECT SIGNAL ************
if(strcmp(ArrayString[0],"CONNECT")==0){
   Initialized=true;
}
if(strcmp(ArrayString[0],"DISCONNECT")==0){
   Initialized=false;
}

//*********** CONNECT/DISCONNECT SIGNAL ************
//*******************************


}
}
```

Anexo III: Código fuente software

- **motores.pro**

```
QT       += core gui serialport

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = Motores
TEMPLATE = app

DEFINES += QT_DEPRECATED_WARNINGS

SOURCES += \
        main.cpp \
        mainwindow.cpp

HEADERS += \
        mainwindow.h

FORMS += \
        mainwindow.ui
```

- **mainwindow.h**

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QSerialPort>
#include <QKeyEvent>
#include <QShortcut>
#include <QByteArray>
#include <QString>

namespace Ui {
class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    void writeSettings(void);
    void readSettings(void);
    void closeEvent(QCloseEvent *event);
    QString result;
    QString command;
    qint8 sliderValue_1=0,sliderValue_2=0,sliderValue_3=0;
    qint32 x_pos_steps=0,y_pos_steps=0,z_pos_steps=0;
    quint16 um_per_revolution=318,step_per_revolution=200;
    double gear_ratio=90.25;
    qint8 valueSliderArray[11]={10,20,30,40,50,60,70,80,90,100};
    quint16 max_step_per_sec=1500;
    int i=0;
    int count=0;
    bool q_keyPressed=false;
    bool e_keyPressed=false;
    bool w_keyPressed=false;
    bool s_keyPressed=false;
    bool a_keyPressed=false;
    bool d_keyPressed=false;

protected:

private slots:
    void readSerial();
    void closeTask();
    void updateRGB(QString);
    void keyPressEvent(QKeyEvent *event);
    void keyReleaseEvent(QKeyEvent *event);
    void on_slider_vel_x_valueChanged(int value);
    void on_slider_vel_y_valueChanged(int value);
    void on_slider_vel_z_valueChanged(int value);
    void on_slider_vel_x_sliderReleased();
    void on_slider_vel_y_sliderReleased();
```

```
    void on_slider_vel_z_sliderReleased();
    void on_button_GOTO_1_clicked();
    void on_button_GOTO_2_clicked();
    void on_button_GOTO_3_clicked();
    void on_button_GOTO_4_clicked();
    void on_button_GOTO_5_clicked();
    void on_button_GOTO_6_clicked();
    void on_button_GOTO_7_clicked();
    void on_point_configure_clicked();
    void on_actionConectar_triggered();
    void on_pushButton_resetOrigin_clicked();
    void on_point_edit_clicked();
    void on_actionGuardar_triggered();
    void on_actionSalir_triggered();
    void on_actionDesconectar_triggered();
private:
    QByteArray serialData;
    QByteArray serialData_buffer;
    QString serialBuffer;
    Ui::MainWindow *ui;
    QSerialPort *arduino;
    static const quint16 arduino_uno_vendor_id=1027;
    static const quint16 arduino_uno_product_id=24577;
    QString arduino_port_name;
    bool arduino_is_available;
    QKeyEvent *key;
    QShortcut *enter_key;
    QShortcut *keyAltW;
    QShortcut *keyAltS;
};
#endif // MAINWINDOW_H
```

- **main.cpp**

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setFixedSize(650,700);
    w.setWindowTitle("Controlador de motores");
    w.show();
    return a.exec();
    }
```

- **mainwindow.cpp**

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QDebug>
#include <QKeyEvent>
#include <QGraphicsView>
#include <QShortcut>
#include <QObject>
#include <QtWidgets>
#include <QIODevice>
#include <QSettings>
#include <QThread>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    readSettings();

    foreach (const QSerialPortInfo &serialPortInfo, QSerialPortInfo::availablePorts()){
        if(serialPortInfo.hasVendorIdentifier() && serialPortInfo.hasProductIdentifier()){
           qDebug()<<serialPortInfo.vendorIdentifier();
            qDebug()<<serialPortInfo.productIdentifier();
            qDebug()<<serialPortInfo.portName();
            }}

    double m_1=(((max_step_per_sec*um_per_revolution)/(step_per_revolution*gear_ratio))-1)/(ui-
>slider_vel_x->maximum());
```

```
    quint16 vel_mot_x=m_1*ui->slider_vel_x->maximum();
    ui->display_vel_x->setText(QString("<span style=\" font-size:10pt; font-
weight:600;\">%1</span>").arg(vel_mot_x));

    double m_2=(((max_step_per_sec*um_per_revolution)/(step_per_revolution*gear_ratio))-1)/(ui-
>slider_vel_y->maximum());
    quint16 vel_mot_y=m_2*ui->slider_vel_y->maximum();
    ui->display_vel_y->setText(QString("<span style=\" font-size:10pt; font-
weight:600;\">%1</span>").arg(vel_mot_y));

    double m_3=(((max_step_per_sec*um_per_revolution)/(step_per_revolution*gear_ratio))-1)/(ui-
>slider_vel_z->maximum());
    quint16 vel_mot_z=m_3*ui->slider_vel_z->maximum();
    ui->display_vel_z->setText(QString("<span style=\" font-size:10pt; font-
weight:600;\">%1</span>").arg(vel_mot_z));

    ui->slider_vel_x->setValue(ui->slider_vel_x->maximum());
    ui->slider_vel_y->setValue(ui->slider_vel_y->maximum());
    ui->slider_vel_z->setValue(ui->slider_vel_z->maximum());

    ui->display_pos_x->setText(QString("<span style=\" font-size:10pt; font-weight:600;\">---
</span>"));
    ui->display_pos_y->setText(QString("<span style=\" font-size:10pt; font-weight:600;\">---
</span>"));
    ui->display_pos_z->setText(QString("<span style=\" font-size:10pt; font-weight:600;\">---
</span>"));

}

void MainWindow::writeSettings(void)
{
    QSettings settings("settings.ini", QSettings::IniFormat);

    settings.beginGroup("mainwidow");
    settings.setValue("p1_x", ui->p1_x->text());
    settings.setValue("p1_y", ui->p1_y->text());
    settings.setValue("p1_z", ui->p1_z->text());
    settings.setValue("p2_x", ui->p2_x->text());
    settings.setValue("p2_y", ui->p2_y->text());
    settings.setValue("p2_z", ui->p2_z->text());
    settings.setValue("p3_x", ui->p3_x->text());
    settings.setValue("p3_y", ui->p3_y->text());
    settings.setValue("p3_z", ui->p3_z->text());
    settings.setValue("p4_x", ui->p4_x->text());
    settings.setValue("p4_y", ui->p4_y->text());
    settings.setValue("p4_z", ui->p4_z->text());
    settings.setValue("p5_x", ui->p5_x->text());
    settings.setValue("p5_y", ui->p5_y->text());
    settings.setValue("p5_z", ui->p5_z->text());
    settings.setValue("p6_x", ui->p6_x->text());
    settings.setValue("p6_y", ui->p6_y->text());
    settings.setValue("p6_z", ui->p6_z->text());
    settings.setValue("p7_x", ui->p7_x->text());
    settings.setValue("p7_y", ui->p7_y->text());
    settings.setValue("p7_z", ui->p7_z->text());
    settings.setValue("x_pos_steps",x_pos_steps);
    settings.setValue("y_pos_steps",y_pos_steps);
    settings.setValue("z_pos_steps",z_pos_steps);
    settings.setValue("size", this->size());
    settings.endGroup();
}

void MainWindow::readSettings(void)
{
    //Int value_ = 0;
    QSettings settings("settings.ini", QSettings::IniFormat);

    ui->p1_x->setText(settings.value("mainwidow/p1_x","").toString());
    ui->p1_y->setText(settings.value("mainwidow/p1_y","").toString());
    ui->p1_z->setText(settings.value("mainwidow/p1_z","").toString());
    ui->p2_x->setText(settings.value("mainwidow/p2_x","").toString());
    ui->p2_y->setText(settings.value("mainwidow/p2_y","").toString());
    ui->p2_z->setText(settings.value("mainwidow/p2_z","").toString());
    ui->p3_x->setText(settings.value("mainwidow/p3_x","").toString());
    ui->p3_y->setText(settings.value("mainwidow/p3_y","").toString());
    ui->p3_z->setText(settings.value("mainwidow/p3_z","").toString());
    ui->p4_x->setText(settings.value("mainwidow/p4_x","").toString());
    ui->p4_y->setText(settings.value("mainwidow/p4_y","").toString());
    ui->p4_z->setText(settings.value("mainwidow/p4_z","").toString());
    ui->p5_x->setText(settings.value("mainwidow/p5_x","").toString());
    ui->p5_y->setText(settings.value("mainwidow/p5_y","").toString());
    ui->p5_z->setText(settings.value("mainwidow/p5_z","").toString());
    ui->p6_x->setText(settings.value("mainwidow/p6_x","").toString());
    ui->p6_y->setText(settings.value("mainwidow/p6_y","").toString());
    ui->p6_z->setText(settings.value("mainwidow/p6_z","").toString());
    ui->p7_x->setText(settings.value("mainwidow/p7_x","").toString());
    ui->p7_y->setText(settings.value("mainwidow/p7_y","").toString());
```

```
    ui->p7_z->setText(settings.value("mainwidow/p7_z","").toString());
    x_pos_steps=settings.value("mainwidow/x_pos_steps",0).toInt();
    y_pos_steps=settings.value("mainwidow/y_pos_steps",0).toInt();
    z_pos_steps=settings.value("mainwidow/z_pos_steps",0).toInt();
}

void MainWindow::closeEvent(QCloseEvent *event)
{
    writeSettings();
    MainWindow::updateRGB("DISCONNECT\n");
    QMainWindow::closeEvent(event);
}

MainWindow::~MainWindow()
{
    if(arduino->isOpen()){
    arduino->close();
    }
    delete ui;
}



void MainWindow::updateRGB(QString command){
    if(arduino->isWritable()){
        arduino->write(command.toStdString().c_str());
        qDebug() << command.toStdString().c_str();
    }
    else{
        qDebug() << "Couldn't write to serial!";
    }
}


void MainWindow::keyPressEvent(QKeyEvent *event)
{

    if(event->isAutoRepeat())
    {
        event->ignore();
    }

    else
    {

    if(event->key() == Qt::Key_W)
        {
        if(s_keyPressed==true){
            MainWindow::updateRGB("STOP MOT2\n");
        }
        else{
            MainWindow::updateRGB("MOV MOT2 C\n");
        }
        w_keyPressed=true;
        }

    if(event->key() == Qt::Key_S)
        {
        if(w_keyPressed==true){
            MainWindow::updateRGB("STOP MOT2\n");
        }
        else{
            MainWindow::updateRGB("MOV MOT2 CC\n");
        }
        s_keyPressed=true;
        }

    if(event->key() == Qt::Key_A)
        {
        if(d_keyPressed==true){
            MainWindow::updateRGB("STOP MOT1\n");
        }
        else{
            MainWindow::updateRGB("MOV MOT1 CC\n");
        }
        a_keyPressed=true;
        }

    if(event->key() == Qt::Key_D)
        {
        if(a_keyPressed==true){
           MainWindow::updateRGB("STOP MOT1\n");
        }
        else{
           MainWindow::updateRGB("MOV MOT1 C\n");
        }
        d_keyPressed=true;
```

```
        }

    if(event->key() == Qt::Key_Q)
        {
        if(e_keyPressed==true)
        {
           MainWindow::updateRGB("STOP MOT3\n");
        }
        else{
           MainWindow::updateRGB("MOV MOT3 CC\n");
        }
        q_keyPressed=true;
        }

    if(event->key() == Qt::Key_E)
        {
        if(q_keyPressed==true){
           MainWindow::updateRGB("STOP MOT3\n");
        }
        else{
           MainWindow::updateRGB("MOV MOT3 C\n");
        }
        e_keyPressed=true;
        }
    }
}


void MainWindow::keyReleaseEvent(QKeyEvent *event)
{

    if(event->isAutoRepeat())
    {
        event->ignore();
    }
    else{

        if(event->key() == Qt::Key_W)
        {
            {
            if(s_keyPressed==true){
                MainWindow::updateRGB("MOV MOT2 CC\n");
            }
            else{
                MainWindow::updateRGB("STOP MOT2\n");
            }
            w_keyPressed=false;
            }
        }

        if(event->key() == Qt::Key_S)
        {
            if(w_keyPressed==true){
                MainWindow::updateRGB("MOV MOT2 C\n");
            }
            else{
                MainWindow::updateRGB("STOP MOT2\n");
            }
            s_keyPressed=false;
        }

        if(event->key() == Qt::Key_A)
        {
            if(d_keyPressed==true){
                MainWindow::updateRGB("MOV MOT1 C\n");
            }
            else{
                MainWindow::updateRGB("STOP MOT1\n");
            }
            a_keyPressed=false;
        }
        if(event->key() == Qt::Key_D)
        {
            if(a_keyPressed==true){
               MainWindow::updateRGB("MOV MOT1 CC\n");
            }
            else{
               MainWindow::updateRGB("STOP MOT1\n");
            }
            d_keyPressed=false;
        }
        if(event->key() == Qt::Key_Q)
        {
            if(e_keyPressed==true)
            {
               MainWindow::updateRGB("MOV MOT3 C\n");
            }
```

```
            else{
               MainWindow::updateRGB("STOP MOT3\n");
            }
            q_keyPressed=false;
        }
        if(event->key() == Qt::Key_E)
        {
            if(q_keyPressed==true){
               MainWindow::updateRGB("MOV MOT3 CC\n");
            }
            else{
               MainWindow::updateRGB("STOP MOT3\n");
            }
            e_keyPressed=false;
        }
    }
}



void MainWindow::on_slider_vel_x_valueChanged(int value)
{
    double m=(((max_step_per_sec*um_per_revolution)/(step_per_revolution*gear_ratio))-1)/(ui-
>slider_vel_x->maximum());
    quint16 vel_mot_x=m*value+1;
    ui->display_vel_x->setText(QString("<span style=\" font-size:10pt; font-
weight:600;\">%1</span>").arg(vel_mot_x));

}

void MainWindow::on_slider_vel_y_valueChanged(int value)
{
    double m=(((max_step_per_sec*um_per_revolution)/(step_per_revolution*gear_ratio))-1)/(ui-
>slider_vel_y->maximum());
    quint16 vel_mot_y=m*value+1;
    ui->display_vel_y->setText(QString("<span style=\" font-size:10pt; font-
weight:600;\">%1</span>").arg(vel_mot_y));

}

void MainWindow::on_slider_vel_z_valueChanged(int value)
{
    double m=(((max_step_per_sec*um_per_revolution)/(step_per_revolution*gear_ratio))-1)/(ui-
>slider_vel_z->maximum());
    quint16 vel_mot_z=m*value+1;
    ui->display_vel_z->setText(QString("<span style=\" font-size:10pt; font-
weight:600;\">%1</span>").arg(vel_mot_z));

}


void MainWindow::on_slider_vel_x_sliderReleased()
{
    double m=(max_step_per_sec-(1*step_per_revolution*gear_ratio/um_per_revolution))/(ui-
>slider_vel_x->maximum());
    quint16 vel_x=m*(ui->slider_vel_x->value())+(1*step_per_revolution*gear_ratio/um_per_revolution);
    MainWindow::updateRGB(QString("VEL MOT1 %1\n").arg(vel_x));
}

void MainWindow::on_slider_vel_y_sliderReleased()
{
    double m=(max_step_per_sec-(1*step_per_revolution*gear_ratio/um_per_revolution))/(ui-
>slider_vel_y->maximum());
    quint16 vel_y=m*(ui->slider_vel_y->value())+(1*step_per_revolution*gear_ratio/um_per_revolution);
    MainWindow::updateRGB(QString("VEL MOT2 %1\n").arg(vel_y));
}

void MainWindow::on_slider_vel_z_sliderReleased()
{
    double m=(max_step_per_sec-(1*step_per_revolution*gear_ratio/um_per_revolution))/(ui-
>slider_vel_z->maximum());
    quint16 vel_z=m*(ui->slider_vel_z->value())+(1*step_per_revolution*gear_ratio/um_per_revolution);
    MainWindow::updateRGB(QString("VEL MOT3 %1\n").arg(vel_z));
}

void MainWindow::readSerial()
{
    serialData=arduino->readAll();
    serialData_buffer.append(serialData);
    serialBuffer = QString::fromStdString(serialData_buffer.toStdString());
    QRegularExpression regex("^#(.+?):");
    QRegularExpressionMatch match = regex.match(serialBuffer);
    if(match.hasMatch()){

       qDebug()<<serialBuffer;

       QStringList bufferSplit = serialBuffer.split(" ");
```

```
        if(bufferSplit[1]=="POS"){
        x_pos_steps = bufferSplit[2].toLong();
        y_pos_steps = bufferSplit[3].toLong();
        z_pos_steps = bufferSplit[4].toLong();
        double x_pos=(x_pos_steps*um_per_revolution)/(step_per_revolution*gear_ratio);
        double y_pos=(y_pos_steps*um_per_revolution)/(step_per_revolution*gear_ratio);
        double z_pos=(z_pos_steps*um_per_revolution)/(step_per_revolution*gear_ratio);
        ui->display_pos_x->setText(QString("<span style=\" font-size:10pt; font-
weight:600;\">%1</span>").arg(QString::number(x_pos,'f',1)));
        ui->display_pos_y->setText(QString("<span style=\" font-size:10pt; font-
weight:600;\">%1</span>").arg(QString::number(y_pos,'f',1)));
        ui->display_pos_z->setText(QString("<span style=\" font-size:10pt; font-
weight:600;\">%1</span>").arg(QString::number(z_pos,'f',1)));
        }


        serialData_buffer="";
    }
}


void MainWindow::on_button_GOTO_1_clicked()
{
 MainWindow::updateRGB("GOTO 1\n");
}
void MainWindow::on_button_GOTO_2_clicked()
{
 MainWindow::updateRGB("GOTO 2\n");
}
void MainWindow::on_button_GOTO_3_clicked()
{
 MainWindow::updateRGB("GOTO 3\n");
}
void MainWindow::on_button_GOTO_4_clicked()
{
 MainWindow::updateRGB("GOTO 4\n");
}
void MainWindow::on_button_GOTO_5_clicked()
{
 MainWindow::updateRGB("GOTO 5\n");
}
void MainWindow::on_button_GOTO_6_clicked()
{
 MainWindow::updateRGB("GOTO 6\n");
}
void MainWindow::on_button_GOTO_7_clicked()
{
 MainWindow::updateRGB("GOTO 7\n");
}


void MainWindow::on_point_configure_clicked()
{

if(!(ui->p1_x->text().isEmpty()||ui->p1_y->text().isEmpty()||ui->p1_z->text().isEmpty())){
 qint32 x1_point=(step_per_revolution*gear_ratio*ui->p1_x->text().toDouble())/um_per_revolution;
 qint32 y1_point=(step_per_revolution*gear_ratio*ui->p1_y->text().toDouble())/um_per_revolution;
 qint32 z1_point=(step_per_revolution*gear_ratio*ui->p1_z->text().toDouble())/um_per_revolution;
 ui->button_GOTO_1->setEnabled(true);
MainWindow::updateRGB(QString("P %1 %2 %3 %4\n").arg(1).arg(x1_point).arg(y1_point).arg(z1_point));
}
if(!(ui->p2_x->text().isEmpty()||ui->p2_y->text().isEmpty()||ui->p2_z->text().isEmpty())){
 qint32 x2_point=(step_per_revolution*gear_ratio*ui->p2_x->text().toDouble())/um_per_revolution;
 qint32 y2_point=(step_per_revolution*gear_ratio*ui->p2_y->text().toDouble())/um_per_revolution;
 qint32 z2_point=(step_per_revolution*gear_ratio*ui->p2_z->text().toDouble())/um_per_revolution;
 ui->button_GOTO_2->setEnabled(true);
MainWindow::updateRGB(QString("P %1 %2 %3 %4\n").arg(2).arg(x2_point).arg(y2_point).arg(z2_point));
}
if(!(ui->p3_x->text().isEmpty()||ui->p3_y->text().isEmpty()||ui->p3_z->text().isEmpty())){
 qint32 x3_point=(step_per_revolution*gear_ratio*ui->p3_x->text().toDouble())/um_per_revolution;
 qint32 y3_point=(step_per_revolution*gear_ratio*ui->p3_y->text().toDouble())/um_per_revolution;
 qint32 z3_point=(step_per_revolution*gear_ratio*ui->p3_z->text().toDouble())/um_per_revolution;
 ui->button_GOTO_3->setEnabled(true);
MainWindow::updateRGB(QString("P %1 %2 %3 %4\n").arg(3).arg(x3_point).arg(y3_point).arg(z3_point));
}
if(!(ui->p4_x->text().isEmpty()||ui->p4_y->text().isEmpty()||ui->p4_z->text().isEmpty())){
 qint32 x4_point=(step_per_revolution*gear_ratio*ui->p4_x->text().toDouble())/um_per_revolution;
 qint32 y4_point=(step_per_revolution*gear_ratio*ui->p4_y->text().toDouble())/um_per_revolution;
 qint32 z4_point=(step_per_revolution*gear_ratio*ui->p4_z->text().toDouble())/um_per_revolution;
 ui->button_GOTO_4->setEnabled(true);
MainWindow::updateRGB(QString("P %1 %2 %3 %4\n").arg(4).arg(x4_point).arg(y4_point).arg(z4_point));
}
if(!(ui->p5_x->text().isEmpty()||ui->p5_y->text().isEmpty()||ui->p5_z->text().isEmpty())){
 qint32 x5_point=(step_per_revolution*gear_ratio*ui->p5_x->text().toDouble())/um_per_revolution;
 qint32 y5_point=(step_per_revolution*gear_ratio*ui->p5_y->text().toDouble())/um_per_revolution;
 qint32 z5_point=(step_per_revolution*gear_ratio*ui->p5_z->text().toDouble())/um_per_revolution;
```

```
 ui->button_GOTO_5->setEnabled(true);
MainWindow::updateRGB(QString("P %1 %2 %3 %4\n").arg(5).arg(x5_point).arg(y5_point).arg(z5_point));
}
if(!(ui->p6_x->text().isEmpty()||ui->p6_y->text().isEmpty()||ui->p6_z->text().isEmpty())){
 qint32 x6_point=(step_per_revolution*gear_ratio*ui->p6_x->text().toDouble())/um_per_revolution;
 qint32 y6_point=(step_per_revolution*gear_ratio*ui->p6_y->text().toDouble())/um_per_revolution;
 qint32 z6_point=(step_per_revolution*gear_ratio*ui->p6_z->text().toDouble())/um_per_revolution;
 ui->button_GOTO_6->setEnabled(true);
MainWindow::updateRGB(QString("P %1 %2 %3 %4\n").arg(6).arg(x6_point).arg(y6_point).arg(z6_point));
}
if(!(ui->p7_x->text().isEmpty()||ui->p7_y->text().isEmpty()||ui->p7_z->text().isEmpty())){
 qint32 x7_point=(step_per_revolution*gear_ratio*ui->p7_x->text().toDouble())/um_per_revolution;
 qint32 y7_point=(step_per_revolution*gear_ratio*ui->p7_y->text().toDouble())/um_per_revolution;
 qint32 z7_point=(step_per_revolution*gear_ratio*ui->p7_z->text().toDouble())/um_per_revolution;
 ui->button_GOTO_7->setEnabled(true);
MainWindow::updateRGB(QString("P %1 %2 %3 %4\n").arg(7).arg(x7_point).arg(y7_point).arg(z7_point));
}

ui->p1_x->setEnabled(false);
ui->p1_y->setEnabled(false);
ui->p1_z->setEnabled(false);
ui->p2_x->setEnabled(false);
ui->p2_y->setEnabled(false);
ui->p2_z->setEnabled(false);
ui->p3_x->setEnabled(false);
ui->p3_y->setEnabled(false);
ui->p3_z->setEnabled(false);
ui->p4_x->setEnabled(false);
ui->p4_y->setEnabled(false);
ui->p4_z->setEnabled(false);
ui->p5_x->setEnabled(false);
ui->p5_y->setEnabled(false);
ui->p5_z->setEnabled(false);
ui->p6_x->setEnabled(false);
ui->p6_y->setEnabled(false);
ui->p6_z->setEnabled(false);
ui->p7_x->setEnabled(false);
ui->p7_y->setEnabled(false);
ui->p7_z->setEnabled(false);

ui->point_configure->setEnabled(false);
ui->point_edit->setEnabled(true);

}


void MainWindow::on_actionConectar_triggered()
{
    arduino_is_available=false;
    arduino_port_name="";
    arduino = new QSerialPort;
    foreach (const QSerialPortInfo &serialPortInfo, QSerialPortInfo::availablePorts()){
        if(serialPortInfo.hasVendorIdentifier() && serialPortInfo.hasProductIdentifier()){
            if(serialPortInfo.vendorIdentifier()==arduino_uno_vendor_id){
                if(serialPortInfo.productIdentifier()==arduino_uno_product_id){
                   arduino_port_name=serialPortInfo.portName();
                   arduino_is_available=true;
                }}}}

    if(arduino_is_available){

        //open and configure the serialports
        arduino->setPortName(arduino_port_name);
        arduino->open(QSerialPort::ReadWrite);
        arduino->setBaudRate(QSerialPort::Baud9600);
        arduino->setDataBits(QSerialPort::Data8);
        arduino->setParity(QSerialPort::NoParity);
        arduino->setStopBits(QSerialPort::OneStop);
        arduino->setFlowControl(QSerialPort::NoFlowControl);
        ui->actionDesconectar->setEnabled(true);
        ui->actionConectar->setEnabled(false);
        QObject::connect(arduino,SIGNAL(readyRead()),this,SLOT(readSerial()));
        QObject::connect(arduino,SIGNAL(aboutToClose()),this,SLOT(closeTask()));

    ui->pushButton_resetOrigin->setEnabled(true);
    ui->pushButton_x_back->setEnabled(true);
    ui->pushButton_x_forward->setEnabled(true);
    ui->pushButton_y_back->setEnabled(true);
    ui->pushButton_y_forward->setEnabled(true);
    ui->pushButton_z_back->setEnabled(true);
    ui->pushButton_z_forward->setEnabled(true);
    ui->slider_vel_x->setEnabled(true);
    ui->slider_vel_y->setEnabled(true);
    ui->slider_vel_z->setEnabled(true);
    ui->point_edit->setEnabled(true);

    QThread::msleep(1600);
```

```
    MainWindow::updateRGB("CONNECT\n");
    QThread::msleep(200);

    if(!(ui->p1_x->text().isEmpty()||ui->p1_y->text().isEmpty()||ui->p1_z->text().isEmpty())){
     qint32 x1_point=(step_per_revolution*gear_ratio*ui->p1_x->text().toDouble())/um_per_revolution;
     qint32 y1_point=(step_per_revolution*gear_ratio*ui->p1_y->text().toDouble())/um_per_revolution;
     qint32 z1_point=(step_per_revolution*gear_ratio*ui->p1_z->text().toDouble())/um_per_revolution;
     ui->button_GOTO_1->setEnabled(true);
    MainWindow::updateRGB(QString("P %1 %2 %3
%4\n").arg(1).arg(x1_point).arg(y1_point).arg(z1_point));
    }
    if(!(ui->p2_x->text().isEmpty()||ui->p2_y->text().isEmpty()||ui->p2_z->text().isEmpty())){
     qint32 x2_point=(step_per_revolution*gear_ratio*ui->p2_x->text().toDouble())/um_per_revolution;
     qint32 y2_point=(step_per_revolution*gear_ratio*ui->p2_y->text().toDouble())/um_per_revolution;
     qint32 z2_point=(step_per_revolution*gear_ratio*ui->p2_z->text().toDouble())/um_per_revolution;
     ui->button_GOTO_2->setEnabled(true);
    MainWindow::updateRGB(QString("P %1 %2 %3
%4\n").arg(2).arg(x2_point).arg(y2_point).arg(z2_point));
    }
    if(!(ui->p3_x->text().isEmpty()||ui->p3_y->text().isEmpty()||ui->p3_z->text().isEmpty())){
     qint32 x3_point=(step_per_revolution*gear_ratio*ui->p3_x->text().toDouble())/um_per_revolution;
     qint32 y3_point=(step_per_revolution*gear_ratio*ui->p3_y->text().toDouble())/um_per_revolution;
     qint32 z3_point=(step_per_revolution*gear_ratio*ui->p3_z->text().toDouble())/um_per_revolution;
     ui->button_GOTO_3->setEnabled(true);
    MainWindow::updateRGB(QString("P %1 %2 %3
%4\n").arg(3).arg(x3_point).arg(y3_point).arg(z3_point));
    }
    if(!(ui->p4_x->text().isEmpty()||ui->p4_y->text().isEmpty()||ui->p4_z->text().isEmpty())){
     qint32 x4_point=(step_per_revolution*gear_ratio*ui->p4_x->text().toDouble())/um_per_revolution;
     qint32 y4_point=(step_per_revolution*gear_ratio*ui->p4_y->text().toDouble())/um_per_revolution;
     qint32 z4_point=(step_per_revolution*gear_ratio*ui->p4_z->text().toDouble())/um_per_revolution;
     ui->button_GOTO_4->setEnabled(true);
    MainWindow::updateRGB(QString("P %1 %2 %3
%4\n").arg(4).arg(x4_point).arg(y4_point).arg(z4_point));
    }
    if(!(ui->p5_x->text().isEmpty()||ui->p5_y->text().isEmpty()||ui->p5_z->text().isEmpty())){
     qint32 x5_point=(step_per_revolution*gear_ratio*ui->p5_x->text().toDouble())/um_per_revolution;
     qint32 y5_point=(step_per_revolution*gear_ratio*ui->p5_y->text().toDouble())/um_per_revolution;
     qint32 z5_point=(step_per_revolution*gear_ratio*ui->p5_z->text().toDouble())/um_per_revolution;
     ui->button_GOTO_5->setEnabled(true);
    MainWindow::updateRGB(QString("P %1 %2 %3
%4\n").arg(5).arg(x5_point).arg(y5_point).arg(z5_point));
    }
    if(!(ui->p6_x->text().isEmpty()||ui->p6_y->text().isEmpty()||ui->p6_z->text().isEmpty())){
     qint32 x6_point=(step_per_revolution*gear_ratio*ui->p6_x->text().toDouble())/um_per_revolution;
     qint32 y6_point=(step_per_revolution*gear_ratio*ui->p6_y->text().toDouble())/um_per_revolution;
     qint32 z6_point=(step_per_revolution*gear_ratio*ui->p6_z->text().toDouble())/um_per_revolution;
     ui->button_GOTO_6->setEnabled(true);
    MainWindow::updateRGB(QString("P %1 %2 %3
%4\n").arg(6).arg(x6_point).arg(y6_point).arg(z6_point));
    }
    if(!(ui->p7_x->text().isEmpty()||ui->p7_y->text().isEmpty()||ui->p7_z->text().isEmpty())){
     qint32 x7_point=(step_per_revolution*gear_ratio*ui->p7_x->text().toDouble())/um_per_revolution;
     qint32 y7_point=(step_per_revolution*gear_ratio*ui->p7_y->text().toDouble())/um_per_revolution;
     qint32 z7_point=(step_per_revolution*gear_ratio*ui->p7_z->text().toDouble())/um_per_revolution;
     ui->button_GOTO_7->setEnabled(true);
    MainWindow::updateRGB(QString("P %1 %2 %3
%4\n").arg(7).arg(x7_point).arg(y7_point).arg(z7_point));
    }

    double m_1=(max_step_per_sec-(1*step_per_revolution*gear_ratio/um_per_revolution))/(ui-
>slider_vel_x->maximum());
    quint16 vel_x=m_1*(ui->slider_vel_x-
>value())+(1*step_per_revolution*gear_ratio/um_per_revolution);
    MainWindow::updateRGB(QString("VEL MOT1 %1\n").arg(vel_x));

    double m_2=(max_step_per_sec-(1*step_per_revolution*gear_ratio/um_per_revolution))/(ui-
>slider_vel_y->maximum());
    quint16 vel_y=m_2*(ui->slider_vel_y-
>value())+(1*step_per_revolution*gear_ratio/um_per_revolution);
    MainWindow::updateRGB(QString("VEL MOT2 %1\n").arg(vel_y));

    double m_3=(max_step_per_sec-(1*step_per_revolution*gear_ratio/um_per_revolution))/(ui-
>slider_vel_z->maximum());
    quint16 vel_z=m_3*(ui->slider_vel_z-
>value())+(1*step_per_revolution*gear_ratio/um_per_revolution);
    MainWindow::updateRGB(QString("VEL MOT3 %1\n").arg(vel_z));

    MainWindow::updateRGB(QString("POS %1 %2
%3\n").arg(x_pos_steps).arg(y_pos_steps).arg(z_pos_steps));
    }

    else{
        //give an error message if not availabe
        QMessageBox::warning(this, "Port error","Couldn't find the Arduino");
    }
```

```
}

void MainWindow::on_pushButton_resetOrigin_clicked()
{
    MainWindow::updateRGB("ORIGIN\n");
}

void MainWindow::on_point_edit_clicked()
{
    ui->point_configure->setEnabled(true);
    ui->p1_x->setEnabled(true);
    ui->p1_y->setEnabled(true);
    ui->p1_z->setEnabled(true);
    ui->p2_x->setEnabled(true);
    ui->p2_y->setEnabled(true);
    ui->p2_z->setEnabled(true);
    ui->p3_x->setEnabled(true);
    ui->p3_y->setEnabled(true);
    ui->p3_z->setEnabled(true);
    ui->p4_x->setEnabled(true);
    ui->p4_y->setEnabled(true);
    ui->p4_z->setEnabled(true);
    ui->p5_x->setEnabled(true);
    ui->p5_y->setEnabled(true);
    ui->p5_z->setEnabled(true);
    ui->p6_x->setEnabled(true);
    ui->p6_y->setEnabled(true);
    ui->p6_z->setEnabled(true);
    ui->p7_x->setEnabled(true);
    ui->p7_y->setEnabled(true);
    ui->p7_z->setEnabled(true);
    ui->button_GOTO_1->setEnabled(false);
    ui->button_GOTO_2->setEnabled(false);
    ui->button_GOTO_3->setEnabled(false);
    ui->button_GOTO_4->setEnabled(false);
    ui->button_GOTO_5->setEnabled(false);
    ui->button_GOTO_6->setEnabled(false);
    ui->button_GOTO_7->setEnabled(false);
    ui->point_edit->setEnabled(false);
    ui->point_configure->setEnabled(true);
}

void MainWindow::on_actionGuardar_triggered()
{
    writeSettings();
}

void MainWindow::on_actionSalir_triggered()
{
    writeSettings();
    QWidget::close();

}

void MainWindow::on_actionDesconectar_triggered()
{
    ui->pushButton_resetOrigin->setEnabled(false);
    ui->pushButton_x_back->setEnabled(false);
    ui->pushButton_x_forward->setEnabled(false);
    ui->pushButton_y_back->setEnabled(false);
    ui->pushButton_y_forward->setEnabled(false);
    ui->pushButton_z_back->setEnabled(false);
    ui->pushButton_z_forward->setEnabled(false);
    ui->slider_vel_x->setEnabled(false);
    ui->slider_vel_y->setEnabled(false);
    ui->slider_vel_z->setEnabled(false);
    ui->point_edit->setEnabled(false);
    ui->button_GOTO_1->setEnabled(false);
    ui->button_GOTO_2->setEnabled(false);
    ui->button_GOTO_3->setEnabled(false);
    ui->button_GOTO_4->setEnabled(false);
    ui->button_GOTO_5->setEnabled(false);
    ui->button_GOTO_6->setEnabled(false);
    ui->button_GOTO_7->setEnabled(false);
    ui->p1_x->setEnabled(false);
    ui->p1_y->setEnabled(false);
    ui->p1_z->setEnabled(false);
    ui->p2_x->setEnabled(false);
    ui->p2_y->setEnabled(false);
    ui->p2_z->setEnabled(false);
    ui->p3_x->setEnabled(false);
    ui->p3_y->setEnabled(false);
    ui->p3_z->setEnabled(false);
    ui->p4_x->setEnabled(false);
    ui->p4_y->setEnabled(false);
    ui->p4_z->setEnabled(false);
    ui->p5_x->setEnabled(false);
```

```
    ui->p5_y->setEnabled(false);
    ui->p5_z->setEnabled(false);
    ui->p6_x->setEnabled(false);
    ui->p6_y->setEnabled(false);
    ui->p6_z->setEnabled(false);
    ui->p7_x->setEnabled(false);
    ui->p7_y->setEnabled(false);
    ui->p7_z->setEnabled(false);
    arduino->close();
    ui->actionDesconectar->setEnabled(false);
    ui->actionConectar->setEnabled(true);
}

void MainWindow::closeTask(){
MainWindow::updateRGB("DISCONNECT\n");
    }
```

Anexo IV: Layout PCB Controlador de motores

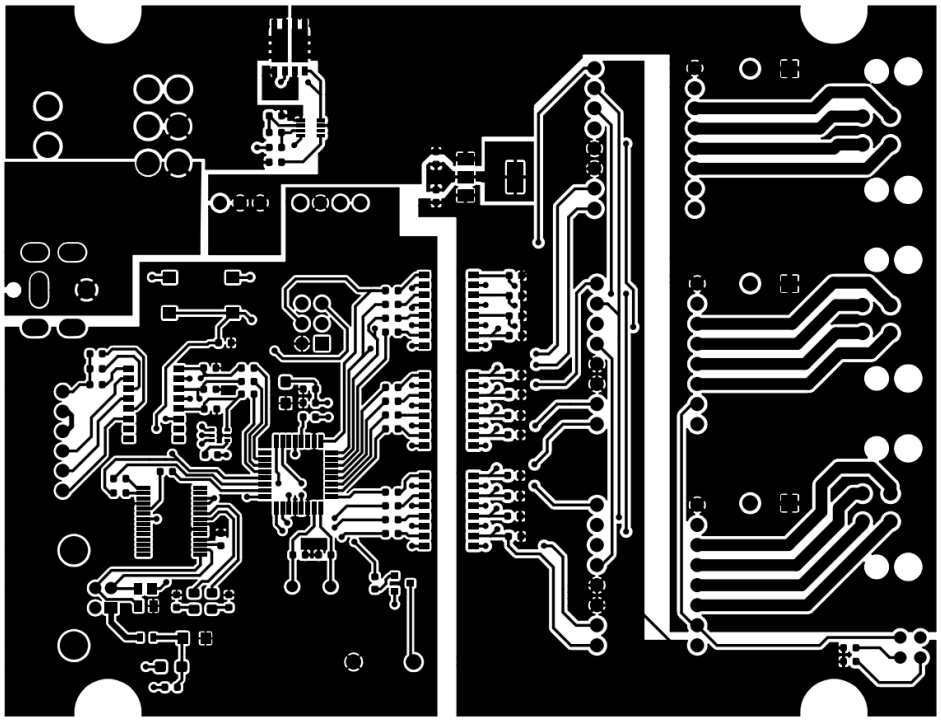


Ilustración 45. Layout capa superior

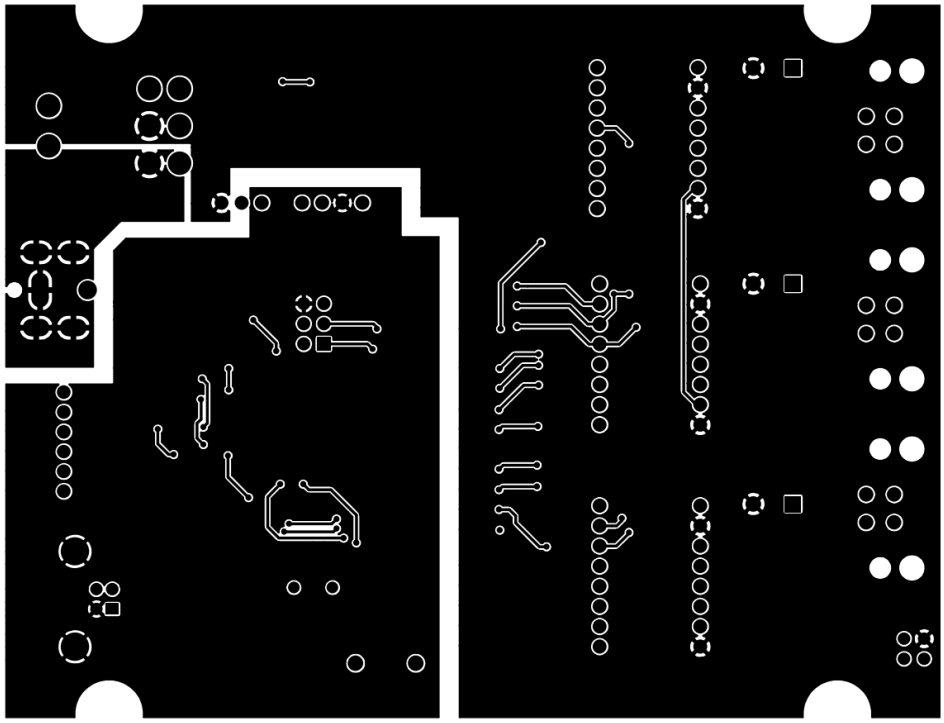
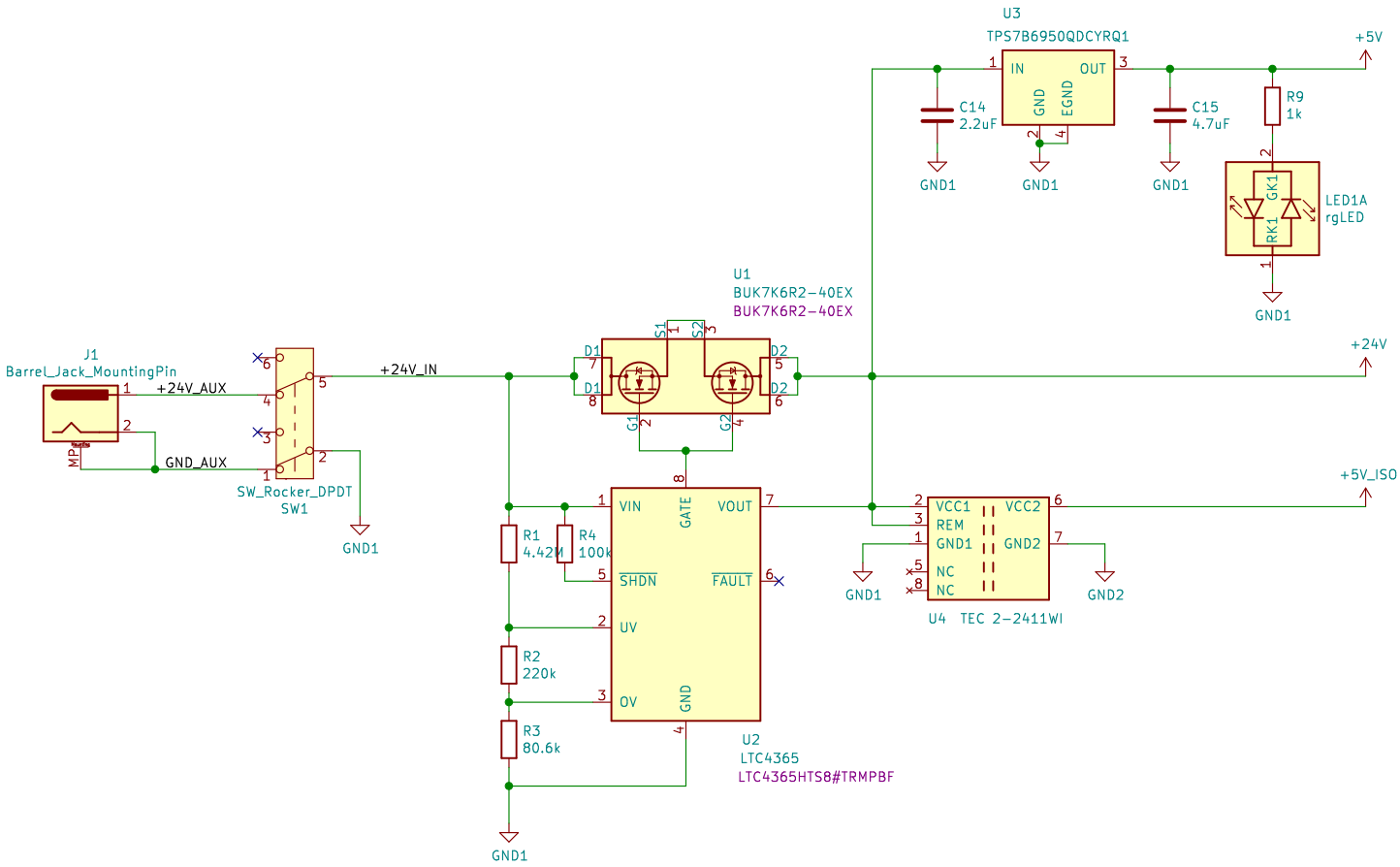
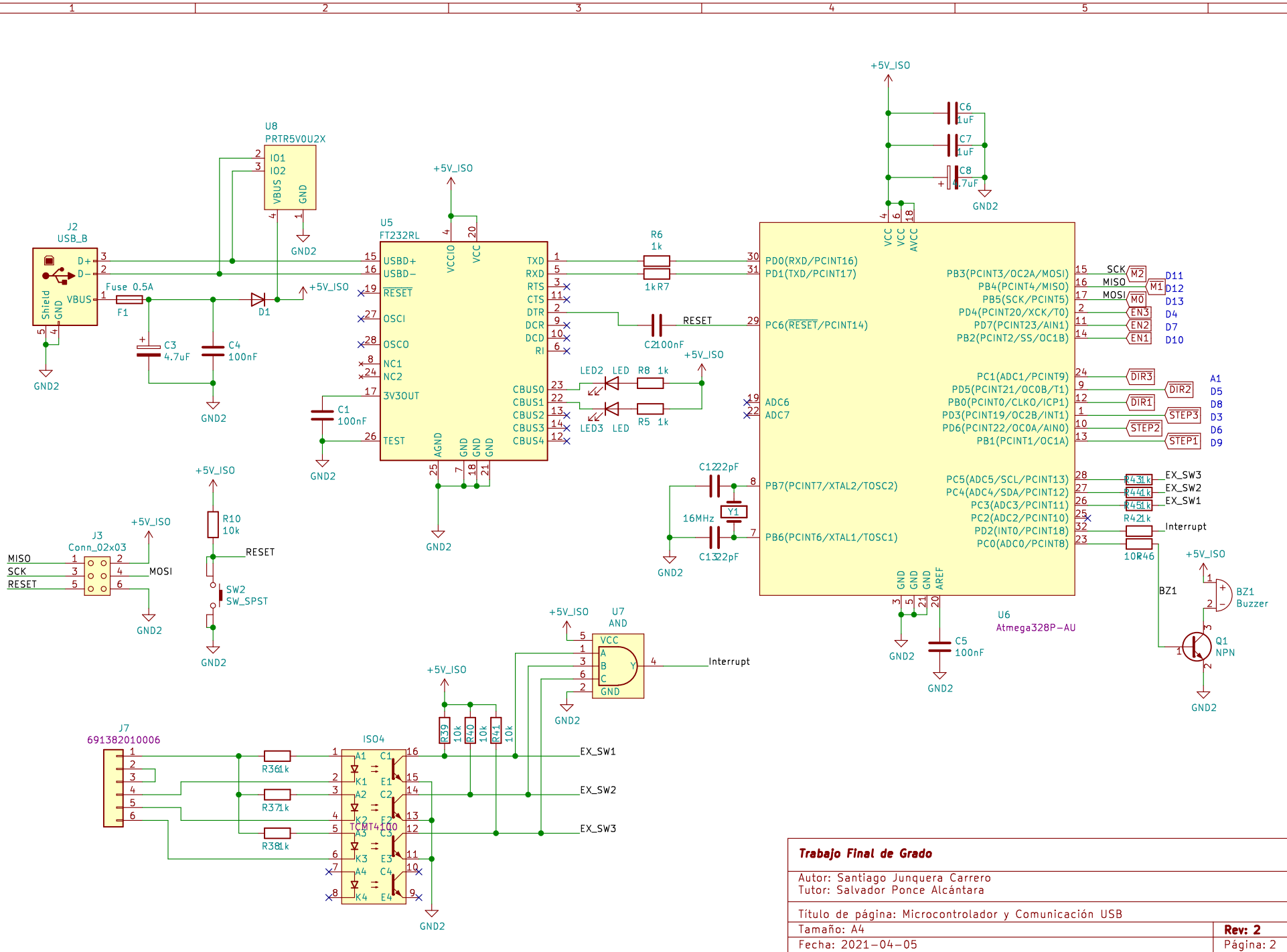


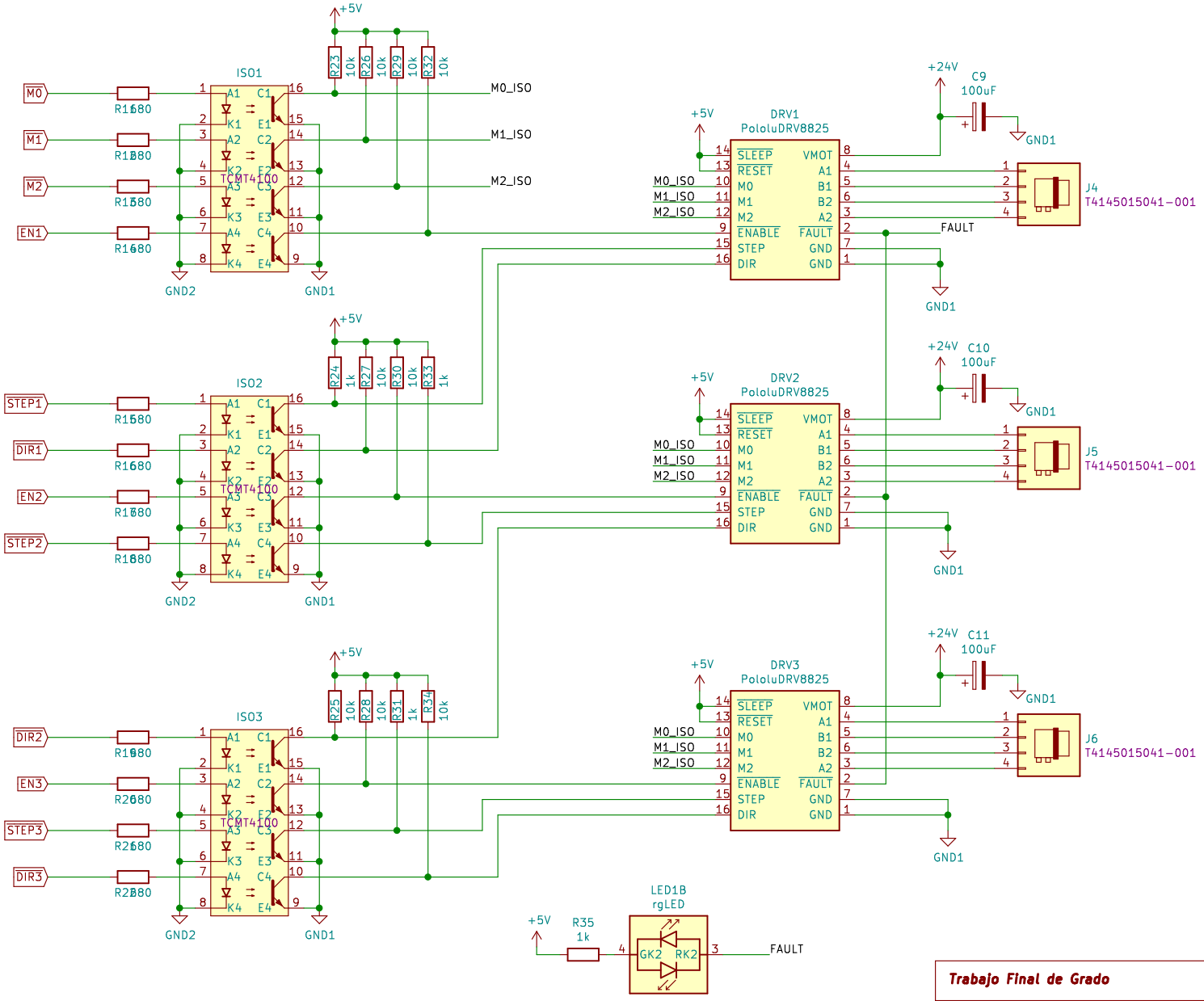
Ilustración 46. Layout capa inferior

Anexo V: Esquema Electrónico



Trabajo Final de Grado	
Autor: Santiago Junquera Carrero Tutor: Salvador Ponce Alcántara	
Título de página: Alimentación y Protecciones	
Tamaño: A4	**Rev: 2**
Fecha: 2021-04-05	Página: 1


U8
PRTR5V0U2X
IO1
IO2
VBUS
GND
J2
USB_B
D+
D-
VBUS
Shield
GND
Fuse 0.5A
F1
D1
+5V_ISO
C3
4.7uF
C4
100nF
GND2
U5
FT232RL
VCCIO
VCC
USBD+
USBD-
RESET
OSCI
OSCO
NC1
NC2
3V3OUT
TEST
AGND
TXD
RXD
RTS
CTS
DTR
DCR
DCD
RI
CBUS0
CBUS1
CBUS2
CBUS3
CBUS4
C1
100nF
R6
1k
1kR7
RESET
C2100nF
LED2
LED
R8 1k
LED3
R5 1k
C6
1uF
C7
1uF
C8
4.7uF
U6
Atmega328P-AU
PD0(RXD/PCINT16)
PD1(TXD/PCINT17)
PC6(RESET/PCINT14)
ADC6
ADC7
PB7(PCINT7/XTAL2/TOSC2)
PB6(PCINT6/XTAL1/TOSC1)
PB3(PCINT3/OC2A/MOSI)
PB4(PCINT4/MISO)
PB5(SCK/PCINT5)
PD4(PCINT20/XCK/T0)
PD7(PCINT23/AIN1)
PB2(PCINT2/SS/OC1B)
PC1(ADC1/PCINT9)
PD5(PCINT21/OC0B/T1)
PB0(PCINT0/CLKO/ICP1)
PD3(PCINT19/OC2B/INT1)
PD6(PCINT22/OC0A/AIN0)
PB1(PCINT1/OC1A)
PC5(ADC5/SCL/PCINT13)
PC4(ADC4/SDA/PCINT12)
PC3(ADC3/PCINT11)
PC2(ADC2/PCINT10)
PD2(INT0/PCINT18)
PC0(ADC0/PCINT8)
C1222pF
16MHz
Y1
C1322pF
C5
100nF
SCK
MISO
MOSI
M2
M1
M0
EN3
EN2
EN1
D11
D12
D13
D4
D7
D10
DIR3
DIR2
DIR1
STEP3
STEP2
STEP1
A1
D5
D8
D3
D6
D9
R43 1k
R44 1k
R45 1k
EX_SW3
EX_SW2
EX_SW1
R42 1k
Interrupt
R46
10k
BZ1
Buzzer
Q1
NPN
J3
Conn_02x03
MISO
SCK
RESET
MOSI
R10
10k
SW2
SW_SPST
U7
AND
VCC
A
B
C
Y
GND
J7
691382010006
ISO4
TCMT4100
R36 1k
R37 1k
R38 1k
R39 10k
R40 10k
R41 10k
EX_SW1
EX_SW2
EX_SW3
Trabajo Final de Grado
Autor: Santiago Junquera Carrero
Tutor: Salvador Ponce Alcántara
Título de página: Microcontrolador y Comunicación USB
Tamaño: A4
Rev: 2
Fecha: 2021-04-05
Página: 2



Trabajo Final de Grado

Autor: Santiago Junquera Carrero
Tutor: Salvador Ponce Alcántara

Título de página: Etapa de potencia

Tamaño: A4 | **Rev: 2**

Fecha: 2021-04-05 | Página: 3