Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño y construcción de un posicionador de coordenadas para la calibración de pinzas amperimétricas

Design and develop of a coordinate positioner for current clamp meter calibration purpouses

Autor

Julio Sebastián Sullca Trillo

Director

Miguel Samplón Chalmeta

Escuela de Ingeniería y Arquitectura

2019

Diseño y construcción de un posicionador de coordenadas para la calibración de pinzas amperimétricas

RESUMEN

Un estudio exterior a este proyecto desea conocer cómo afecta a la medida de corriente la posición relativa entre el conductor por el que circula la corriente y una pinza amperimétrica. Para ello se usará una espira, la cual conducirá una corriente conocida y la trasladaremos a varias posiciones dentro del instrumento. El objetivo del proyecto es la construcción de una máquina capaz de llevar a cabo esta labor. Es decir, se trasladará la espira a una posición dada en el interior de la pinza.

La primera parte del proyecto, y la más larga, consistirá en la construcción del prototipo de posicionador. Hasta obtenerlo pasaremos por diversos estadios. Empezando por la construcción de un solo eje que servirá como banco de pruebas; este nos dará una visión más amplia de lo que queremos conseguir y nos ayudará a concretar un primer diseño. El segundo estadio consiste en elaborar éste primer diseño a partir de las conclusiones extraídas del primer estadio. Este diseño se planteará desde un punto de vista teórico y será el que finalmente se ejecute. La última parte consiste en el montaje del prototipo. Durante el cual fueron surgiendo varios problemas e imprevistos que no se tuvieron en cuenta en el diseño y a los que se les dará solución. Informaremos de ellos para tenerlos en cuenta en el futuro.

La segunda parte del proyecto se lleva a cabo una vez que el prototipo es funcional. Se hará un estudio de calibración al posicionamiento del cual pretendemos obtener una estimación de la corrección o el sesgo e incertidumbre

El proyecto da solución a un problema de posicionamiento y sienta las bases para proyectos de posicionamiento posteriores.

Índice

1 Introducción

Hoy en día la calibración de pinzas amperimétricas es bastante común. En resumen, el proceso consiste en hacer pasar un corriente patrón por el interior de la pinza y calibrarla respecto a este [1] .Sin embargo, no se tiene en cuenta la distancia relativa entre la corriente y la pinza. A efectos de realizar un estudio de como afecta la posición de la corriente en la calibración de la pinza; por parte de una investigación externa, se nos planteará construir un prototipo de posicionador de coordenadas.

La motivación del proyecto es que se quiere realizar un estudio de posicionamiento en el interior de una pinza amperimétrica. Por lo que se requiere un posicionador de diseño específico para esta labor y que además resulte más económico que uno comercial.

El alcance del proyecto será construir un prototipo operativo y realizarle a éste un estudio de calibración para conocer la exactitud que tiene.

2 Definición

Para este proyecto se solicitó el desarrollo de un sistema de posicionamiento para ser usado en calibraciones de pinza amperimétrica. La calibración de estos instrumentos queda fuera del alcance de este proyecto.

Las especificaciones de partida son:

1. El posicionamiento en los ejes X e Y (movimiento plano).
2. El giro sobre su propio eje (al cual denominaremos eje Z).
3. La resolución debe estar por debajo de los milímetros. Con el fin de que el error de posicionamiento sea inferior al que somos capaces de determinar, y que además no acarree una mala medición en la pinza.

Además, como recomendación el prototipo debe evitar en la medida de lo posible causar distorsiones en el campo magnético que puedan llevar a un posible error de lectura en la pinza.

Al principio del proyecto teníamos mucha libertad para escoger un diseño y tomar un camino en el desarrollo. Decidimos como criterio principal el de abaratar costes, porque a pesar de que las máquinas de posicionamiento son bastante comunes y están presentes en nuestro día a día, no existe una para nuestra labor específica. El precio de una de estas máquinas (para usos más generales) puede ser del orden de miles de euros, en el caso de máquinas de gama alta.

3 Desarrollo previo

Este apartado consiste en definir las ideas y principios de funcionamiento que darán forma a nuestro proyecto. Con esas ideas definidas podremos pasar a planificar lo que será la construcción del prototipo. En este apartado de desarrollo nos centraremos en la construcción de un solo eje para poder tener una idea más amplia de como continuaremos el proyecto.

3.1 Materiales de partida

Con las prestaciones ya definidas, usaremos las piezas que nos pueda suministrar el Departamento de Ingeniería Eléctrica. Entre ellos destacaremos:

1. Un motor paso a paso(17hd40005-22b).

2. Drivers de potencia DRV8825.

3. Un kit de ejes y tornillos sin fin.

4. Finales de carrera inductivos(BES M12ML-PSC40F-BV00-002).

Con estos materiales debemos ser capaces de desarrollar la idea del posicionador, determinar si son compatibles con lo que queremos y determinar que otros elementos necesitaríamos para conseguir la precisión demandada. Por ejemplo, a priori podemos observar que con estos materiales no podemos medir la posición, la cual es una de las especificaciones principales.

Con estos componentes se puede modelar el movimiento; con el motor, finales de carrera y tornillo sin fin se hará el modelado mecánico del movimiento, mientras que con el driver se hará el modelado eléctrico del funcionamiento de los motores. Estos componentes ejecutan el movimiento, pero es el microprocesador el que lo controla a partir de la lectura de los sensores y actuando sobre la salida a los motores.

El microprocesador que se usará será el ATmega328P integrado en el dispositivo de Arduino, la interfaz de programación de Arduino es bastante interactiva, y el ATmega328P tiene las prestaciones suficientes para la consecución de este proyecto.

3.2 Idea inicial

El principio de funcionamiento a aplicar es convertir el giro del motor paso a paso en un desplazamiento lineal a través del tornillo sin fin. Conociendo el avance por giro del tornillo sin fin (8mm) y el giro por paso del motor (1.8º), si contásemos los pulsos que enviamos a los motores, podríamos conocer cuánto ha girado el motor y por tanto cuanto nos hemos desplazado. Sin embargo, este sistema en bucle abierto puede no ser preciso ya que el bucle abierto no nos permite conocer el valor del giro real producido. Este sistema es muy dependiente del correcto funcionamiento del motor, de que este gire siempre los 1.8º estipulados por el fabricante. Con este método estaremos ciegos ante atascos o fallos del sistema. Por lo que se decidió comprar un encoder (LPD3806-400BM-G5-24C)[1]. De esta manera tendremos un bucle cerrado con una realimentación del ángulo girado.

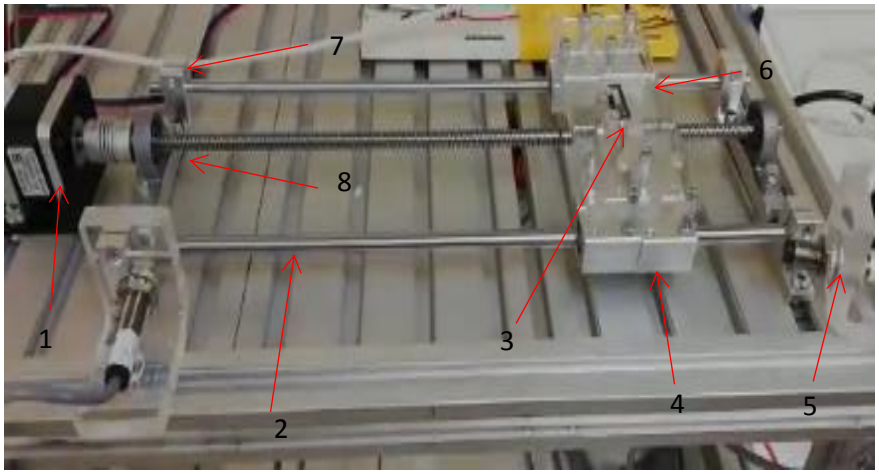
[1] Consultar Anexo II: Especificaciones de componentes

3.3 Primer eje

La primera etapa para el desarrollo del proyecto será la puesta en funcionamiento de un solo eje, que posteriormente se convertirá en nuestro eje X, para así poder tener una visión más amplia de lo que será la planificación y montaje, además de detectar problemas no previstos. En esta primera parte desarrollaremos la fase alfa del programa para el movimiento individual de un eje, y el control de los finales de carrera. Como el funcionamiento de los otros dos motores será parecido solo tendremos que replicar el trabajo, pero introduciendo una función en el software que permita elegir cuál de los tres motores va a funcionar. Dado que el control de la velocidad de posicionamiento o el control de trayectorias no son especificaciones de nuestro proyecto decidiremos que solo podrá funcionar un motor a la vez. Esta decisión nos permite simplificar el montaje eléctrico, el software y reduce la corriente que nos tendrá que suministrar la alimentación. Se diseñará un montaje mecánico y un montaje eléctrico.

3.4 Montaje mecánico

Para el montaje mecánico usaremos el motor paso a paso, finales de carrera, y todos los componentes del kit de ejes y tornillo sin fin; posteriormente se podrá decidir qué elementos eliminar, sin afectar a la funcionalidad del sistema. A continuación, se muestra la disposición de los componentes mecánicos en nuestro primer eje.



Configuración del primer eje, construido en bucle abierto. 1. Motor paso a paso 2. Eje guía 3. Tuerca de avance 4. Apoyos deslizantes 5 . final de carrera 6. Plataforma 7. apoyo fijo 8. Rodamiento

Ilustración 1 Montaje mecánico, primer eje

Destacamos de este montaje que la plataforma y los soportes de los finales de carrera se hicieron con partes de metacrilato recicladas en el Departamento de Ingeniería Eléctrica. Para soportar toda la estructura se nos dio una base de perfiles de aluminio, cuyas ranuras internas eran ideales para atornillar nuestros componentes. Para alinear los rodamientos con el eje del motor, se usaron también materiales excedentes del departamento (aluminio, metacrilato…) estos apoyos, aunque imprecisos, dieron buenos resultados iniciales, ya que la plataforma avanzaba (manualmente) sin dificultad a lo largo de todo el eje.

Por retrasos en pedidos y envíos no se pudo disponer del encoder en el desarrollo de este primer eje, se consideró que su ausencia no afectaría drásticamente a la funcionalidad en este punto. Para esta primera parte nos pondremos como objetivos; conseguir el movimiento lineal de la plataforma respetando los finales de carrera.

3.5 Montaje eléctrico

En lo que respecta al montaje eléctrico lo dividiremos en tres partes; motor, driver de potencia 8825 y el circuito lógico para implementar el final de carrera.

3.5.1 Motor

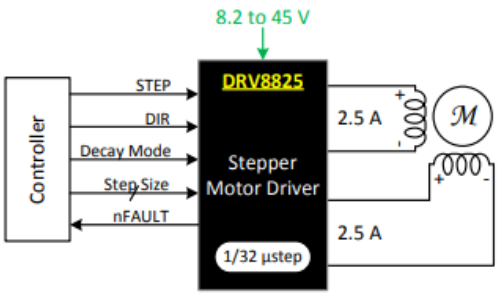
Tenemos un motor paso a paso bipolar, del tipo Nema 17, la diferencia de estos motores con el resto es que nos permiten discretizar el movimiento ya que este se mide en pasos de 1,8º.

El principio de funcionamiento se simplifica si comparamos el rotor con un imán que tiende a alinearse con los polos de las bobinas, esto es lo que visualmente se corresponde con un paso, al tener un comportamiento discreto solo nos fijamos cuando el rotor se ha alineado y no en como lo ha hecho. Si conseguimos que los polos de las bobinas cambien en la secuencia correcta conseguimos un giro por pasos, además cada paso se puede reducir en pasos más pequeños. Esto se conoce como micro stepping, la división de los pasos es siempre por un múltiplo de 2 y dependerá del driver. Este método de control consiste en actuar sobre la corriente que pasa por la bobina, dejando pasar una fracción de la corriente nominal. Resumiendo, si reducimos o aumentamos la corriente en las bobinas se tendrá un campo magnético más o menos intenso por lo que hará girar el motor la fracción de paso correspondiente.

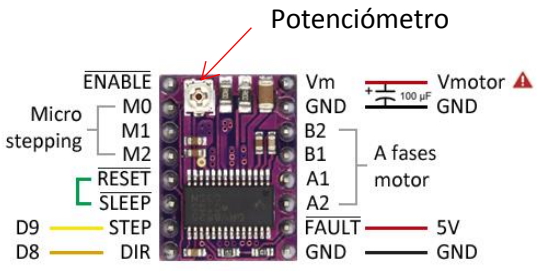
3.5.2 Driver de potencia

Para la documentación previa al montaje se usó el datasheet del fabricante [2] y para conocer procedimientos de montaje y ejemplos de aplicación recurriremos al artículo del driver8825 [3] del blog de Luis Llamas (Ingeniero Industrial por la Universidad de Zaragoza). En puntos posteriores del proyecto usaremos varios artículos de este blog en los que se explican detalladamente ejemplos de aplicación de Arduino.

Los esquemas del driver son los siguientes:



Esquema simplificado [2]



Módulo de conexiones (driver comercial) [3]

Ilustración 2 DRV8825

- **Alimentación:** Las especificaciones del motor por fase indican una corriente nominal de 1.3A, 1.6Ω resistencia y 2V de tensión nominal[2].El pin Vm se conectará a estos 12V y a un condensador de desacoplo de 100 uF. La alimentación debe ser superior a la tensión nominal para que el motor funcione correctamente, un valor de 12V debería ser suficiente [3]. Esto se debe a que el valor más importante para nuestro motor es la corriente ya que es el parámetro que le hará recorrer un paso, la tensión es un parámetro debido a la corriente nominal que circula por la bobina. La alimentación de 12V se hace para que la corriente alcance el valor nominal lo más rápido posible, ya que internamente el driver tiene un control PWM que cortará la alimentación cuando el valor programado de corriente se alcance. De esta forma el voltaje de alimentación promedio es mucho menor.

- **Control de corriente:** Si alimentamos a 12V y tenemos una resistencia de 1.6Ω, el valor de la corriente se dispararía, por eso el driver tiene un limitador de corriente, este es regulable por un potenciómetro, que podemos observar en la imagen derecha de la ilustración 2 (página anterior) junto al pin ENABLE. La forma de limitar la corriente pasa por la siguiente expresión:

$$I_{LIMITE} = \frac{V_{REF}}{5 \times R_{SENSE}}$$

extraída de la hoja de datos del fabricante. La R_{sense} depende de la pcb comprada, en nuestro caso es de 0.1Ω .Además debemos saber que la I_{LIMITE} se reduce a un 71% porque no usamos los pines de micro stepping. Con estos valores calculamos el valor de V_{REF} usando el valor nominal de la corriente. El valor de V_{REF} en todos los drivers será de 0.91V. El proceso para calibrar este valor pasa por medir con un voltímetro el valor de la tensión en el potenciómetro integrado en el módulo de conexiones

- **Pines M0, M1, M2:** Estos pines servirían para usar la función de microstepping en la que a partir de fraccionar el límite de corriente se consigue que el motor gire una determinada fracción de un paso cada pulso. No usaremos esta función. La relación de movimiento entre el motor y el driver es de un paso por pulso. A continuación, mostramos una imagen de como evolucionarían las corrientes por las bobinas con el micro stepping, la amarilla es para la salida A, la verde para la B y ambas son la corriente sin micro stepping.

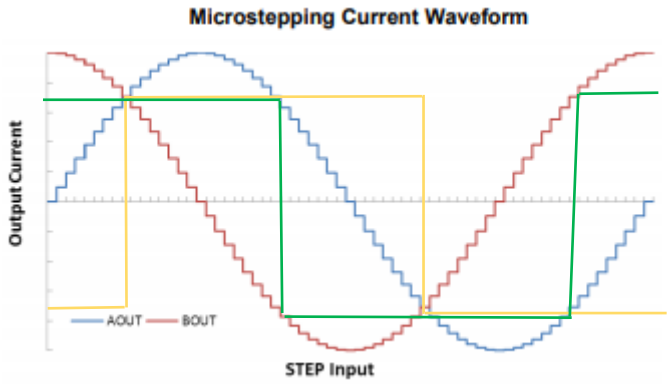


Ilustración 3 Corriente por el motor

[2] Consultar Anexo II: Especificaciones de los componentes

- **Pines RESET, SLEEP, FAULT:** Los dos primeros estarán cortocircuitados y en nivel alto (5V) para que el driver pueda funcionar, el pin RESET a nivel bajo reinicia el driver y el pin SLEEP a nivel bajo activa el modo de bajo consumo que es lo mismo que estar apagado. El pin FAULT es un pin de lectura, si está en nivel alto significa que el driver funciona correctamente, si se pone a nivel bajo es que ha ocurrido algún fallo interno como por sobrecorriente o por un calentamiento excesivo. Cuando estos fallos ocurren los pines RESET Y SLEEP bloquean el dispositivo y solo lo desbloquearan si son reiniciados. El pin FAULT también puede conectarse directamente a 5V como en la ilustración 2 y funcionaría igualmente solo que el sistema de protección interno quedaría desactivado, finalmente se opta por dejar el pin FAULT desconectado y conectar SLEEP, RESET a 5V.

- **Pines ENABLE, STEP, DIR:** Son los tres pines que controlan el movimiento del motor; STEP recibe pulsos del Arduino y por cada flanco de subida hace girar al motor un paso; DIR controlara el sentido de giro del motor según este en nivel alto o bajo; también dependerá del sentido en el que se conecta el motor al driver pero esto es menos relevante; ENABLE se encarga de habilitar o no el movimiento, si ENABLE está en nivel alto se desactivara la corriente en el motor y por tanto el movimiento, destacar que desactivar la corriente es equivalente a apagar el motor.

- **Pines B1, B2, A1, A2:** Son los pines de alimentación al motor según como conectemos los terminales el motor girara en un sentido u otro. Hablaremos del conexionado del motor en el punto de planificación.

3.5.3 Circuito final de carrera

Se procede a la implantación de los finales de carrera con el propósito de controlar y acotar el movimiento de nuestro primer eje. En este punto se decide que la señal del final de carrera y del encoder tienen que ser interpretadas de inmediato, por lo que se conectaran a los pines de interrupción del Arduino; el problema surge que en el montaje final tendremos 4 finales de carrera, 2 encoders y solo 2 pines de interrupción para todos estos dispositivos.

Se tendrá que plantear un circuito que a partir de órdenes del Arduino nos permita “oír” los sensores adecuados. Aunque esta idea se desarrollará más adelante, en esta parte se sentarán las bases.

Según los datos del fabricante el sensor funciona conectado a valores desde 10 a 30 V, y su funcionamiento consiste en detectar metales cercanos a aproximadamente 3.2mm, los valores del sensor son o saturación o corte, cuando detecta el metal entra en saturación, encendiendo un led de aviso, internamente conecta su pin de salida al voltaje de alimentación (12V). Los pasos que seguiremos en el montaje son los siguientes:

- Conectamos los finales de carrera a 12 V

- La conexión con la información puede estar en nivel alto a 12V si el sensor se ha saturado, y en nivel bajo si el sensor no detecta nada. Estos valores de tensión habrá que reducirlos a 5 y 0 V para que el microprocesador pueda leerlos. Se usó un zener de 5V y una resistencia de 1KΩ.

- Los valores ahora ya adaptados de ambos finales de carrera se llevarán a un circuito de transistores que funcionarán como interruptores (saturación y corte), estos “interruptores”, dejaran pasar o no la señal del sensor, la señal que pone el corte y saturación a los transistores será la procedente del pin DIR. La señal escogida se conectará a un pin de interrupción y al pin ENABLE para así desactivar el movimiento.

El pin DIR controlará que final de carrera será escuchado en cada momento, ya que solo debemos escuchar al final de carrera al que nos estamos acercando.

En esta parte se decide que se usará un circuito de transistores, actuando como interruptores, para el control de los sensores cuando hagamos la planificación.

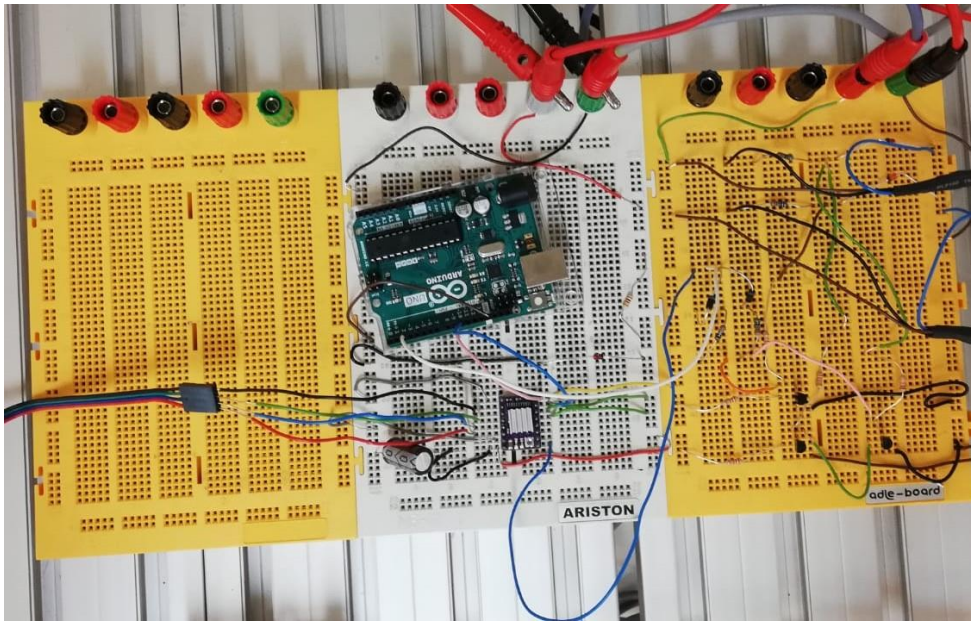


Ilustración 4 Montaje eléctrico, primer eje

3.6 Prueba de funcionamiento

Con el montaje del primer eje terminado se pasó a comprobar si el sistema responde bien al movimiento, se diseñaron dos programas en Arduino para comprobar el correcto funcionamiento, ambos programas se encuentran en el anexo Programas.

Se quiere comprobar si el desplazamiento de la plataforma ocurre sin problema alguno tanto en un sentido como el otro además se quiere medir el avance por giro del tornillo sin fin para comprobar si coincide con los datos del fabricante, de esto se encargará el programa `paso_del_tornillo_sin_fin`.

Lo siguiente que se quiere comprobar es el correcto funcionamiento del final de carrera a la hora de bloquear el circuito, además de si sería viable calibrar el cero absoluto con uno de los finales de carrera, esto se verá en el programa `puesta_a_cero`.

3.6.1 Programa `paso_del_tornillo_sin_fin`

Este programa es bastante sencillo. Se desea comprobar que el montaje funciona correctamente, sin tener en cuenta el final de carrera ya que este estará desconectado. El programa se extrae del blog de Luis Llamas [3] con unas ligeras modificaciones. Este programa se encarga de hacer girar al motor una secuencia de vueltas en diferentes sentidos. La secuencia está integrada en el código del programa. Primero se comprueba que el movimiento es correcto y ocurre sin problemas aparentes así que se pasa al segundo objetivo, medir el paso del tornillo sin fin. La descripción del proceso es la siguiente:

Colocando la plataforma en el extremo del motor para tomar este como referencia, haremos inicialmente 10 giros para medir el avance y calcular el paso, seguido a esto el motor se desplazara de 3 en 3 giros hasta llegar a 28 cada 3 giros recalcularemos el paso midiendo la distancia entre la plataforma y motor, para ordenar al motor que se mueva tres giros deberemos

cambiar el estado del pin 7 del Arduino manualmente, esto nos dará tiempo para tomar las medidas de longitud correspondientes con un pie de rey. Cuando se llegue a 28 giros el motor retrocederá 15 giros, para así poder comprobar el funcionamiento en sentido contrario y comprobar visualmente la coherencia de los desplazamientos. Se harán 5 ciclos de 28 giros cada uno, finalmente se hará una media de todos los avances medidos para calcular así el avance medio por giro, valor que se usaremos en el futuro desarrollo del proyecto. Destacamos que hemos considerado un giro igual a 200 pasos del motor, según el fabricante el avance del tornillo sin fin debe ser de 8mm, los resultados empíricos son los siguientes:

Ciclos	10 giros	13 giros	16 giros	19 giros	22 giros	25 giros	28 giros	Paso medio
1º	81	106	130	155	179	204	228	8,139
2º	82	106	131	155	179	203	227	8,151
3	82	106	130	155	179	203	228	8,147
4º	81	106	130	155	179	203	227	8,128
5º	82	106	130	154	179	203	227	8,135

**Las unidades de la tabla están en mm.*

Destacamos que la repetibilidad del proceso es alta ya que casi todos los resultados coinciden de un ciclo a otro, el paso medio del ciclo se calcula como la media de los resultados obtenidos entre los giros que se han dado para ese valor, la media total se calcula como 8.14mm por giro, a partir de ahora este es el valor del avance, difiere en 0.14 mm del valor del fabricante.

3.6.2 Programa `puesta_a_cero`

Este programa se encarga de desplazar la plataforma del tornillo sin fin hasta uno de los finales de carrera, el cual está fijado en el código del programa. Al ejecutar el programa se comprueba que al llegar al final de carrera la plataforma frena de forma instantánea. Esto en gran parte es debido a la irreversibilidad de la transmisión por tornillo sin fin. El punto alcanzado, en el que la plataforma se detiene, debido al final de carrera, es siempre el mismo. Por esto se decide asociar el origen de coordenadas con un final de carrera, concretamente el más próximo al motor. En cuanto a software este programa nos ayudará a sentar las bases de lo que en un futuro será la función `movereje` la cual veremos más adelante. Esta es la encargada de posicionar la plataforma a la posición deseada en la versión final del programa de posicionamiento. Ya en el programa anterior advertimos un problema, Arduino no permite programar tareas en paralelo (threads). Esto nos impide separar las tareas de movimiento y sensado. La lógica secuencial del Arduino nos obliga a que una tarea vaya después de la otra, lo que nos quita eficiencia. Por ello, se optó por enviar la orden de movimiento a través de una señal cuadrada creada en el timer1 del Arduino. De esta manera mientras el timer manda un tren de pulsos el programa principal solo se centrará en interpretar las señales de los sensores. La frecuencia de la señal pulsada se escoge a 1000Hz, el motor tiene un límite[3] de funcionamiento superior a 1300 Hz y uno inferior a 300Hz.

3.6.3 Conclusiones

- El diseño mecánico del tornillo sin fin cumple con la funcionalidad deseada, posteriormente se lubricará, para reducir más su fricción.
- El motor funcionara a 1000Hz o a lo que es lo mismo, 5 revoluciones por segundo o 4 cm/s aproximadamente.

[3] Revisar Anexo II: Especificaciones de los componentes

- Los pulsos se realizarán con el timer1 del Arduino, de esta manera el software y los pulsos (ordenes de movimiento) serán independientes.
- Necesitamos un componente integrado que sustituya a los transistores que funcionan como interruptores, en el departamento se dispone del ULN2003 (transistores Darlington en serie que funcionan como puerta not pero que su estructura de transistores nos permite usarlos como interruptores)

Los diseños del eje X se replicarán en el eje Y ya que ambos son iguales en cuanto a funcionamiento. El control del eje Z será diferente porque esta vez no estamos convirtiendo el giro en desplazamiento, por eso este eje se controlará por separado de ahora en adelante.

4 Planificación del prototipo

Con las ideas asentadas y decisiones de funcionamiento tomadas, procederemos a las siguientes tareas previas al montaje:

- Elaborar lista de materiales y presupuesto total
- Diseño de piezas para impresión 3D
- Diseño de pcb, en formato de shield para Arduino
- Programa para el control de movimiento e interfaz de comunicaciones

4.1 Lista de materiales

En la elaboración de esta lista se nombrarán los componentes que usaremos a priori en la construcción del prototipo. Con este presupuesto podemos comparar el precio del proyecto con el valor de un posicionador comercial. Hay que recalcar que los componentes electrónicos obtenidos en el departamento tendrán un valor añadido nulo, ya que son componentes que estamos reutilizando y no suponen un gasto real para nuestro proyecto. Añadimos también que algunos materiales estaban disponibles previos al proyecto debido a un intento anterior de realización de este, estos componentes sí que representaron un gasto por lo que su precio sí que aparecerá.

Componente	Referencia	Cantidad	Precio	Observaciones
Encoder	lpd3806 400bm g5 24c	2	18,8€	
Motor paso a paso	17hd40005-22b	3	13,28€	
Soporte nema 17	Marca: WINOMO Referencia: KGU163401AE18Q5253	1	14,99€	Estos soportes estaban disponibles previos al proyecto, vienen en paquetes de dos.
Soporte nema 17	Marca: Makeblock	3	8,05€	Compramos diferentes, ya que estos los escogemos durante el proyecto
Kit de ejes y soportes	Marca: BQLZR Modelo: N23945	2	29,07€	Incluye: 1 tornillo sin fin, 1 tuerca de avance, 2 ejes lineales, 4 soportes deslizantes, apoyos fijos y 2 rodamientos
Tuerca de avance	Marca: RoadRomao Modelo: QJ790	2	1€	Se compran 2 más porque usaremos 4
Acoplador 6-8 mm	Marca: Winwill Modelo: 3D190	2	2,49€	Acopla el eje del encoder al tornillo sin fin
Acoplador 5-8mm	Marca: Winwill Modelo: 3D190	2	2,49€	Acopla el eje del motor al tornillo sin fin
PCB driver de potencia	DRV8825	2	13,99€	Vienen 5 en cada paquete, solo hacen falta 3 pero son elementos críticos por eso compramos de mas

En total son 214,66€ pero asumimos un presupuesto de 250€ por gastos de envió y posibles gastos adicionales.

Los materiales que se nombraran ahora son piezas que se encargaron a imprimir en 3D y que además no representaron un gasto durante la ejecución del proyecto:

Piezas para la impresión 3D[4]			
Componente	Tiempo de fabricación	Cantidad	Observaciones
Plataforma para el eje X motriz	16h20’	1	Esta plataforma servirá de apoyo para el motor Y está situada en el eje del motor X
Plataforma para el eje X de arrastre	14h30’	1	Esta plataforma servirá de apoyo para el encoder Y está situada en el eje arrastrado por el motor X
Plataforma para el motor Z	8h40’	1	Esta plataforma soporta el motor que da el giro en Z
Anillo 20-22mm	4’	2	Este anillo sirve para que los encoders de anillo exterior 20mm puedan ser encajados en los soportes nema 17 de anillo interior 22mm

La construcción de las piezas no supuso un gasto adicional de dinero, pero sí de tiempo, requiriendo hasta 40 horas de trabajo. El modelo de impresora usada fue una Creality Ender-3.

Las tuercas, tornillos, arandelas no se han tenido en cuenta en la lista de materiales, ya que su número es variable y dependerá del montaje, además disponemos de gran variedad y disponibilidad. Faltaría la lista de componentes electrónicos, que encontraremos en el anexo Planos.

4.2 Diseño de piezas para impresión 3D

El par motriz del motor se transforma en fuerza lineal a través del tornillo sin fin, desplazando una plataforma a lo largo del eje consiguiendo así el posicionamiento, además el giro real se realimentará por el encoder para así conocer la posición en cada momento.

Las plataformas son necesarias ya que estas soportarán el objeto que se quiera posicionar. Tendremos 3 plataformas cada una para un trabajo específico. Distinguimos entre plataforma motriz para el eje X, plataforma de arrastre para el eje X y plataforma para el motor Z.

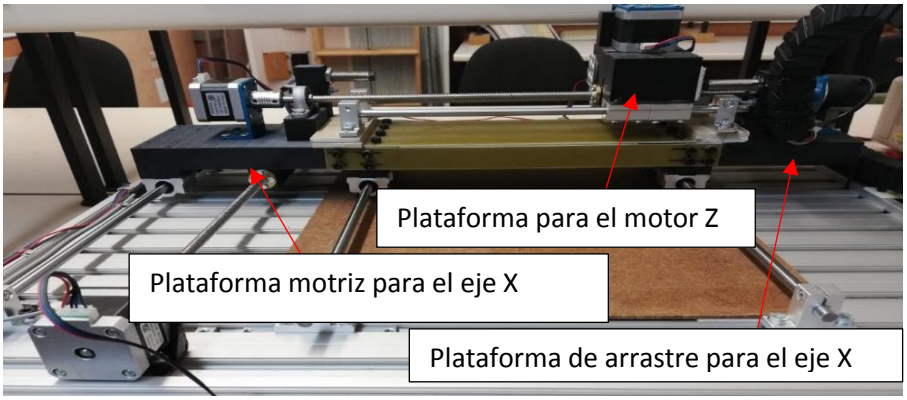


Ilustración 5 Disposición final de las plataformas

Antes de dimensionar las plataformas vamos a hacer un cálculo aproximado del peso que será capaz de mover el tornillo sin fin.

[4] Consultar Anexo III: Planos

Según el fabricante el torque máximo del motor será de 360 mN*m→360 N*mm; con un radio de 4 mm en el tornillo obtenemos 90N de fuerza tangencial, para convertir la fuerza tangencial en axial lo haremos con las siguientes expresiones [4]:

$$A_T = \frac{T_T}{tg(\mu + \varphi')} \qquad Dp_1 = \frac{P_{h1}}{\pi * tg(\mu)} \qquad f' = tg(\varphi') \quad f' = \frac{0{,}122}{V_d{}^{0,2}} \quad V_d = \frac{V_t}{\cos(\mu)}$$

La primera ecuación nos da la fuerza de avance en [N], la segunda sirve para obtener μ (ángulo de inclinación), las otras tres restantes tienen que ver con el cálculo del coeficiente de fricción, siendo V_t la velocidad tangencial de giro en m/s y V_D la velocidad de deslizamiento.

Los datos de partida son T_T=90[N], n=5 [rev/s], no conocemos el diámetro primitivo pero si el exterior e interior, D_E=8mm y D_I=6mm, consideramos que $D_{P1}=D_E-\frac{1.25}{2,25}(D_E - D_I)*0.5$ [5] el P_{H1} es 8 mm (avance por giro).

Resultados:

D. primitivo= 7,44 mm

μ=18,89º

V. deslizamiento = 0,247 m/s

$\varphi' = 9{,}166º$

A_T=168,86 N

Por lo que si consideramos que la única fuerza que se opone en al movimiento está en la guía de acero inoxidable y tomando un coeficiente de rozamiento aproximado de 0,3 tendremos que la plataforma puede llevar como máximo 57kg para que el mecanismo siga funcionando.

Los elementos motrices están sobre dimensionados, nunca llegaremos a ese peso ya que antes se romperían las plataformas y sus apoyos, el conjunto de las plataformas y motores movidos por el eje X no llega a 3 Kg por lo que podemos decir que la fuerza motriz será suficiente por exceso.

4.2.1 Plataforma motriz para el eje X

Esta plataforma deberá apoyarse sobre los carriles deslizantes que se moverán por la guía lineal la cual asegura la rectitud del movimiento, además le atravesara por en medio el tornillo sin fin el cual le proporciona el avance. Resumiendo debemos diseñar la plataforma para que por abajo albergue los carriles deslizantes (aportan rectitud y fijan la dirección) y el tornillo sin fin (elemento motriz), la relación con este último debe ser a través de dos tuercas de avance (una por delante y otra por detrás), no podemos roscar el interior de la plataforma ya que la fuerza del tornillo acabaría desgastando el plástico, por ello el agujero pasante que se hizo en la base de la plataforma es de diámetro igual al de la tuerca de avance, así esta queda fijada a la plataforma y transmite el empuje del tornillo a esta. Por la parte de arriba la plataforma deberá dar soporte al motor del eje Y, además esta plataforma impulsará todo el eje Y a lo largo de X. También se deberá dar un soporte para apoyar el final de carrera, un soporte para los rodamientos y unos agujeros para fijar los apoyos fijos del eje Y. La estructura del eje Y es idéntica al eje X.

Se escoge la impresión 3D ya que esta nos dará mucha precisión a la hora de colocar los componentes. Estamos usando ejes largos y paralelos que deben moverse por caminos rectos, los más ligeros desajustes de alineamiento pueden dar lugar a un mal funcionamiento debido a

montaje. Los elementos más críticos son la posición del rodamiento y de los apoyos fijos, ya que deben estar lo más alineados posibles con los agujeros sucesivos y ser paralelos con los ejes contiguos. La imagen siguiente nos ilustra una simulación de la pieza en el software de Inventor.

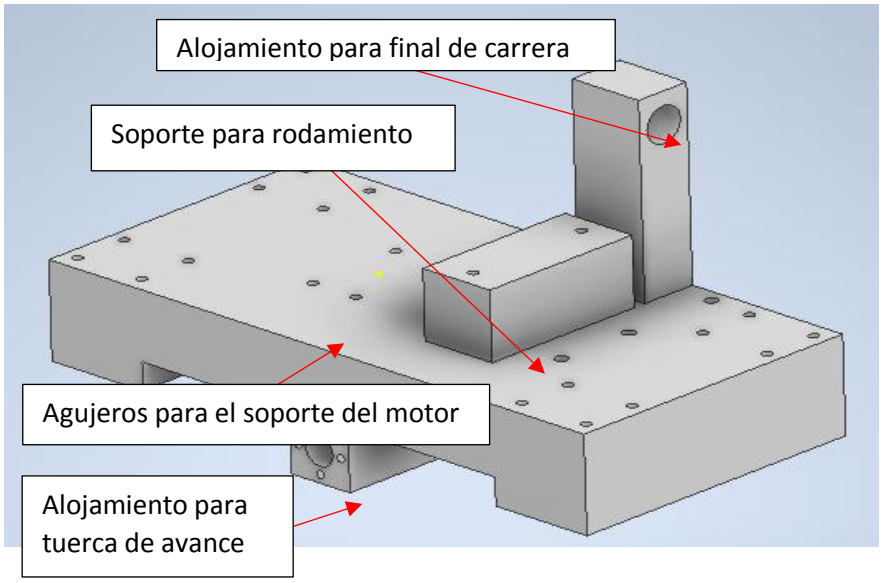


Ilustración 6 Plataforma motriz para el eje X

4.2.2 Plataforma de arrastre para el eje X

Esta plataforma soportará el otro extremo del eje Y donde estará situado el encoder, esta plataforma es arrastrada por el motor del eje X, el arrastre se hace a través de las guías lineales que unen las dos plataformas que hemos nombrado. Es importante que esta plataforma este alineada con la otra para evitar que el avance retuerza al eje Y (que esta encima del eje X) , esto provocaría que el avance quedase atascado, debido a que el propio material de las guías impediría el movimiento. El mismo problema ocurriría si la fuerza de empuje no consigue que nos desplacemos a la misma velocidad por todas las guías debido a una falta de rigidez en la conexión de las dos plataformas del eje X.

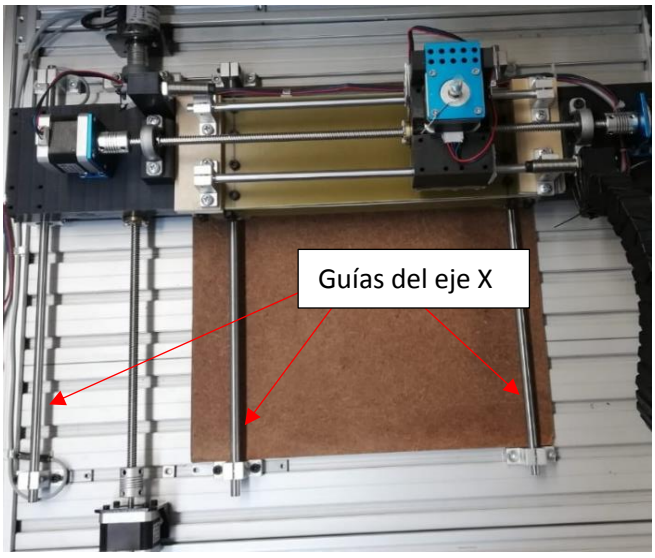


Ilustración 7 Disposición final de las guías deslizantes

A diferencia de la imagen anterior nosotros diseñaremos la plataforma para que vaya sobre dos guías deslizantes. La ilustración se corresponde a la situación después del montaje. El resto de las características son iguales a la plataforma motriz salvo por el hecho de que ésta no alberga tornillo sin fin. Igual que antes mostraremos en la siguiente imagen una simulación en Inventor.

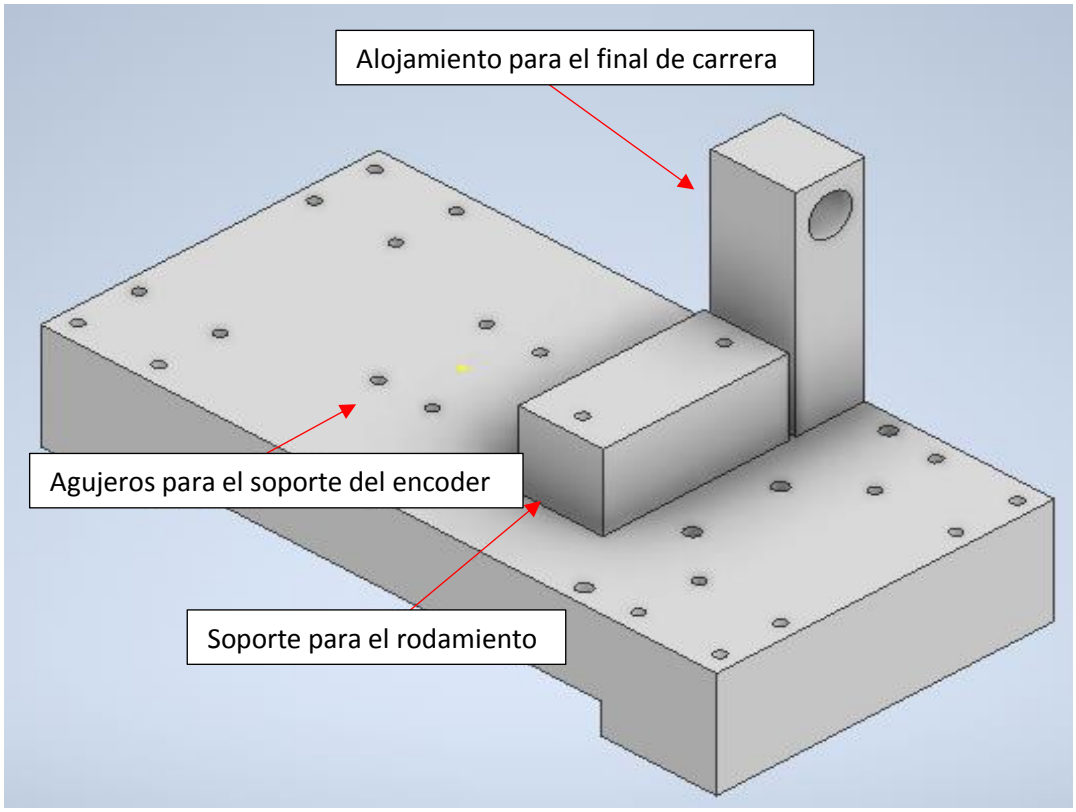


Ilustración 8 Plataforma de arrastre para el eje X

Para el dimensionado de todas las piezas se tendrán en cuenta las medidas que aparecen en el anexo Especificaciones de los componentes, la longitud y la anchura se pensó para que puedan caber todos los componentes en el menor espacio posible.

4.2.3 Plataforma para el motor Z

Esta es la plataforma más pequeña, ya que solo soportará al motor que da el giro en Z. El eje de este motor es el punto por posicionar. Entre sus especificaciones debe tener un agujero para albergar las tuercas de avance del eje Y además de dos carriles deslizantes, especificaciones similares a la plataforma motriz del eje X. La diferencia es que ahora queremos ocupar aún menos espacio. las dimensiones deben ser coherentes con las posiciones relativas de los apoyos fijos y rodamientos del eje Y, para que la plataforma Z pueda ser montada sobre el eje Y. Por último, esta plataforma deberá tener un apoyo vertical en su cara superior. Este apoyo serviría para poder atornillar el soporte del motor y hacer que este último apunte hacia arriba.

La imagen de la siguiente página será el correspondiente modelo de la pieza en Inventor

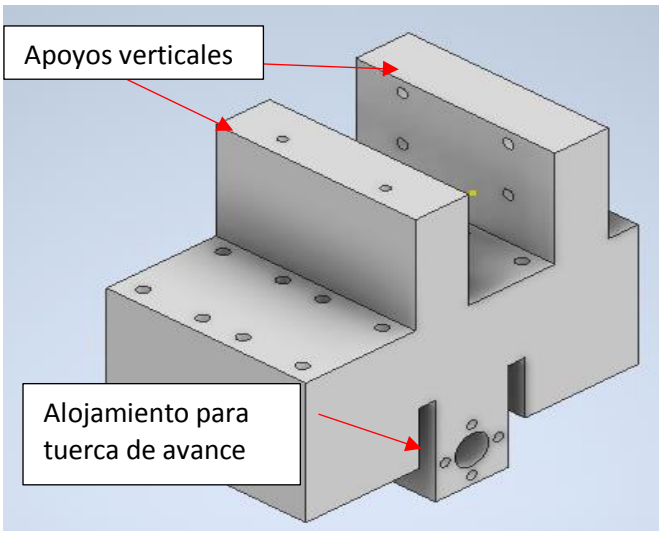


Ilustración 9 Plataforma para el motor Z

Con esto damos por terminado el diseño de las plataformas, las ilustraciones 6, 8 y 9 es lo que se imprimirá en 3D sin embargo, como ya hablaremos más adelante, durante el montaje se les harán modificaciones e incluso como en el caso de la plataforma Z se hará un cambio en los planos, por ello la imagen superior de la plataforma Z no se corresponde a la que aparece en el anexo Planos.

4.2.4 Anillo 20-22mm

Esta será la última pieza que se imprimirá en 3D, además es la más simple se trata de un anillo de plástico de diámetro interior 20 mm y exterior 22 mm y con una altura de 5 mm, la necesidad de esta pieza surge debido a que los encoders no tienen un apoyo normalizado de dimensiones coherentes con los apoyos del motor. Recordemos que es importante que encoder y motor queden perfectamente alineados a la misma altura. Por ello usaremos los apoyos del motor como apoyos para el encoder. El único problema es que estos soportes están diseñados con un agujero de 22 mm de diámetro mientras que el encoder encajaría en un agujero de 20 mm, por eso se fabrica este anillo, para poder adaptar el encoder a los soportes del motor.

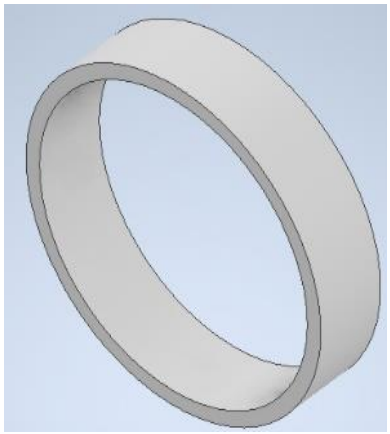


Ilustración 10 Anillo 20-22 mm

4.3 Diseño de pcb

El plano del esquema eléctrico se encuentra en el anexo Planos. La pcb debe recibir las entradas analógicas de los sensores y llevarlas al Arduino en un nivel que este pueda interpretar como lógica digital. La otra función es la de enviar las salidas analógicas para el control de los motores. Se debe convertir la lógica digital del Arduino en una orden analógica que mueva el motor. El diseño de la pcb se puede dividir en control de los motores y acondicionamiento de las entradas.

4.3.1 Control de los motores

Como bien sabemos la lógica digital es un espacio discreto, mientras que el movimiento de un motor es de tipo continuo. La ventaja de los motores paso a paso es que discretizan ese movimiento. Pasamos de tener un movimiento continuo a uno discretizado en pasos. Esto facilita mucho el control de movimiento ya que se tratará todo en un espacio discreto; un pulso eléctrico para avanzar un paso. El driver de potencia se encarga de controlar el movimiento durante la realización de un paso. De esta manera el driver nos permite convertir el giro del motor de un espacio continuo a uno discreto. En resumen, cada vez que enviamos un pulso eléctrico al driver del motor este producirá una salida analógica, la cual es convertida en un paso. Cada pulso eléctrico se corresponde con girar 1.8º. Solo podemos movernos en múltiplos de este y no en valores intermedios (salvo micro stepping, función que no vamos a usar).

Estos drivers son elementos críticos. Por un lado, son más propensos a fallar. En lugar de ser soldados a la placa, se conectarán a la pcb a través de pines para que sean fácilmente intercambiables en caso de fallo. Por otro lado, no dejan de ser transistores funcionando a una frecuencia considerable y que conducen corrientes elevadas (1.3 A). Estos factores pueden producir un ruido que alteraría el funcionamiento del resto de componentes del circuito. Por ello para compensar estos efectos inductivos se conectarán a la alimentación de cada driver un condensador de desacoplo de 100 uF como recomienda el fabricante. Además, en paralelo a este se pondrá otro condensador de 1 nF para así garantizar el correcto funcionamiento del condensador anterior. Estos condensadores se colocan lo más cerca posible de la alimentación de 12 voltios de los drivers. Adicionalmente para protegernos de la diafonía los drivers estarán situados a un lado de la pcb y los componentes de lectura de sensores estarán en el otro extremo con cierta separación hacia los drivers.

El conexionado de los drivers será el siguiente:

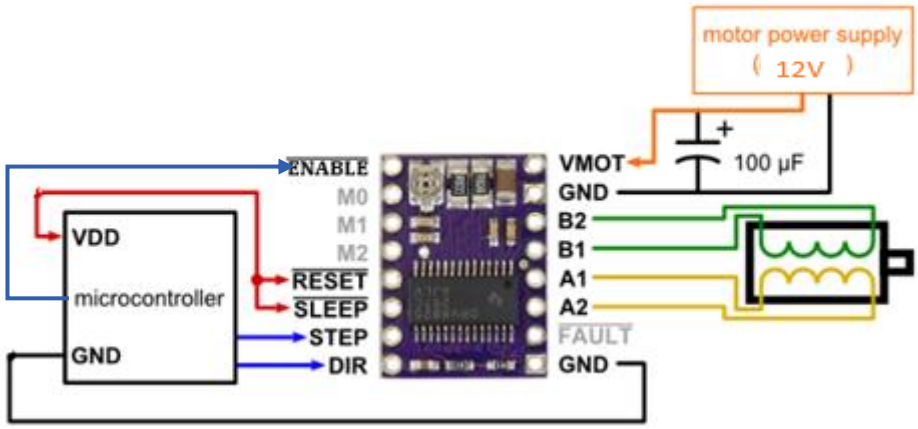
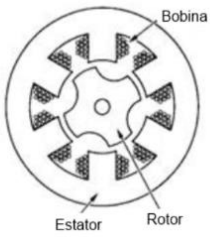


Ilustración 11 Conexionado de los drivers

Vamos a usar la imagen anterior para analizar los drivers individualmente. Sabemos que en total hay 3, uno para cada motor. Los pines de entrada que usamos son STEP, DIR, EN. Son 3 entradas digitales. Cada flanco de subida en STEP girará al motor un paso. ENABLE en nivel alto corta la corriente al motor es decir apaga el driver, en nivel bajo permite el funcionamiento. Por último, DIR fija el sentido de giro del motor para cada paso. Con el conexionado que tenemos DIR en nivel bajo avanzará hacia el encoder (hacia adelante) o lo que es lo mismo girará contrario a las agujas del reloj; mientras que en nivel alto avanzará hacia el motor (hacia atrás), gira en el sentido de las agujas del reloj. Por otro lado, los pines RESET y SLEEP deben estar a nivel alto para que el driver funcione. El pin SLEEP enciende el dispositivo y el pin RESET resetea el driver si está en nivel bajo, por eso si hay algún fallo en el driver el problema podría solucionarse conectando y desconectándolo de la shield. Los pines V_{MOT} y GND corresponden a la fuente de potencia.

Las salidas analógicas son B2,B1 Y A2,A1 estas son la alimentación a las bobinas del motor. La activación de sus terminales sigue una secuencia para poder realizar un giro continuo. Esta secuencia se repite cada 4 pasos, sin esta secuencia el movimiento no se haría como esta especificado, el rotor no giraría de la forma correcta. Explicamos la secuencia de activación de las bobinas con la siguiente imagen:



Paso	Bobina 1A	Bobina 1B	Bobina 2A	Bobina 2B
Paso 1	1	0	1	0
Paso 2	1	0	0	1
Paso 3	0	1	0	1
Paso 4	0	1	1	0

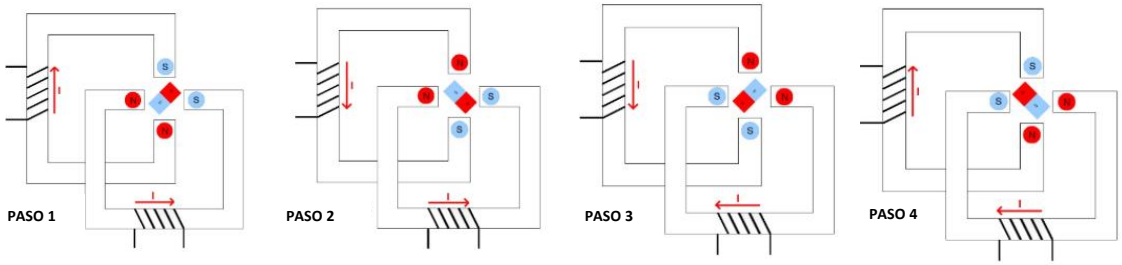


Ilustración 12 Secuencia de activación de las bobinas [11]

Tenemos un motor paso a paso bipolar, con una estructura muy semejante a la imagen superior izquierda. El resto de las imágenes son una simplificación del funcionamiento. Como no usamos micro stepping, siempre circula corriente por las bobinas, ya sea en una dirección u otra; por eso tenemos 4 posiciones en los polos del estator. Cada vez que cambiamos el sentido de las bobinas (en la secuencia correcta) avanzamos 90 grados eléctricos (360º/4), estos 90º son lo que conocemos como un paso. Los 90º se transforman en 1,8º por la configuración de rueda dentada del estator, lo que importa es aplicar la corriente en la bobina siguiendo la secuencia de la tabla de arriba. El 1 es el terminal donde se aplica el voltaje y el 0 donde se aplica GND, el driver ya tiene esta secuencia integrada.

Hasta ahora hemos analizado el funcionamiento de los drivers de forma individual, ahora analizaremos las conexiones globales.

Previamente habíamos decidido que solo funcionaría un motor a la vez, porque nuestra fuente de alimentación no es capaz de dar suficiente corriente y por el modelo que estamos desarrollando. Nuestro principio de funcionamiento se puede resumir como al Arduino mandando girar a un motor a velocidad constante hasta que un sensor le avise de cuando parar. El control del motor desconoce la posición en la que se encuentra, esta solo la saben los sensores.

Con esta idea decidimos cortocircuitar las líneas STEP y DIR de los motores X e Y, conectando DIR al pin 10 del Arduino y STEP al 9. Estos dos motores tienen un funcionamiento idéntico, el pin 9 está conectado al timer 1 interno del Arduino el cual hemos programado a 1000Hz. El timer es programado usando los registros del microprocesador [6]. Escogemos este timer porque no afecta a las librerías de Arduino que vamos a usar. Los pines ENABLE están separados solo puede estar uno en nivel bajo al mismo tiempo (nivel bajo significa que puede funcionar), de esta manera, aunque se mande dirección y pulsos de movimiento, solo el driver que el Arduino elija funcionara o dejara de hacerlo. El pin ENABLE del eje X es el 7 mientras que para él Y es el 6.

El driver del eje Z tiene un conexionado diferente, su desplazamiento no está realimentado por tanto para conocer su movimiento solo podemos contar los pulsos del motor que hemos enviado. No podemos usar un tren de pulsos constante como en los otros dos ejes, el funcionamiento de este motor consiste en sumar de pulso en pulso hasta llegar al ángulo que deseamos[5]. Lo haremos usando la función delay() del Arduino, el pin DIR es 11, el STEP es 13 y el ENABLE 12.

4.3.2 Acondicionamiento de las entradas

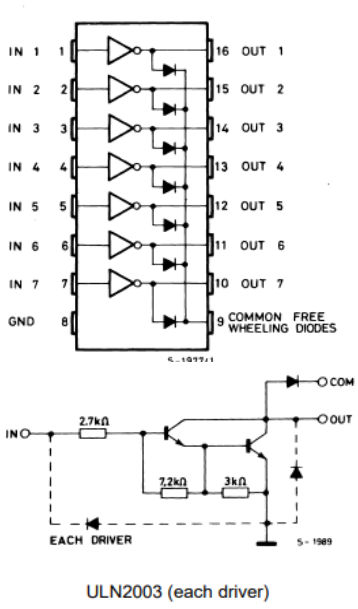
Las entradas se deberán leer por interrupciones, porque debemos actuar de inmediato cuando nos llegue una. Usaremos el pin 3 para leer los encoders y el 2 para los finales de carrera. Primero hablaremos de la alimentación de estos componentes, todos se alimentan a 12 voltios. Hemos optado por una conexión en forma de daisy chain (por necesidades de espacio), además porque estos componentes no generan mucho ruido eléctrico ya que sus conmutaciones son lentas y de bajas corrientes. Las salidas de estos sensores pasan por varias etapas para ser acondicionadas para su lectura. En el caso de los encoders (cada uno tiene dos canales); a un canal se le acopla una resistencia de pull-up de 10k conectada a los 5V del Arduino ya que el funcionamiento del encoder consiste en abrir y cerrar un interruptor, que no es más que un transistor, cuando su eje girá 0.9º. El otro canal del encoder se conecta directamente a la resistencia de pull-up del Arduino. Este canal se conecta al pin 4 para el encoder del eje X y al pin 5 para el encoder del eje Y. Comparando el nivel lógico de estos pines durante la interrupción del encoder podremos conocer en qué sentido está girando nuestro eje (el sentido de giro depende del estado lógico de un canal cuando el otro se encuentra en un flanco).

Los finales de carrera se activan por proximidad a un metal o a un campo magnético pero su funcionamiento es el mismo, cuando se saturan conecta su terminal de salida con el de su alimentación, es decir si un final de carrera se activa tendremos una salida de 12 V. el Arduino lee valores entre 0 y 5V por lo que debemos adaptar esta señal con un divisor de tensión primero con una resistencia de 15k y luego con una de 10k, así ya tendríamos la señal en 5V sin embargo entre estas dos resistencias añadiremos el componente que nos hará decidir a qué señal hacer caso.

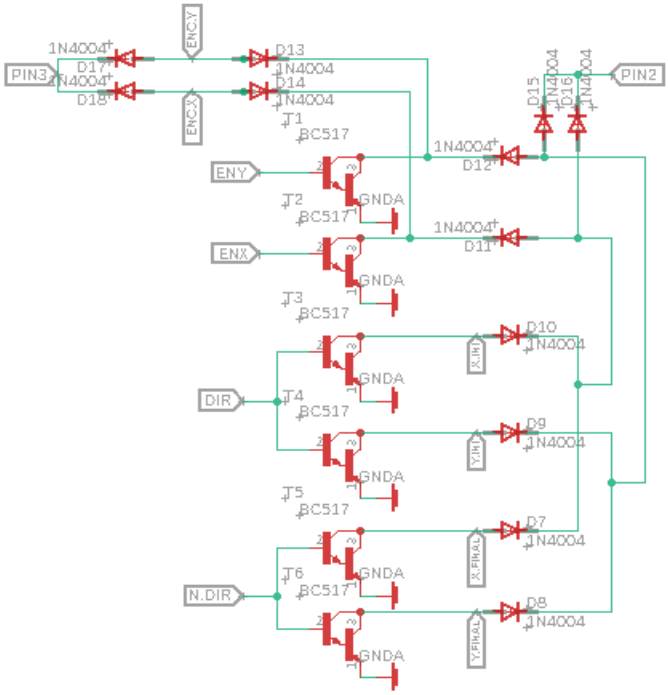
[5] El esquema eléctrico está en el Anexo III: Planos

Nuestro mayor problema en la lectura de las entradas es que tenemos dos pines de lectura y 6 sensores, por eso agrupamos los sensores de un tipo en cada pin. La lógica que se va a usar para interpretar las señales es que el Arduino va a elegir a que sensores escuchar (uno de cada tipo). El Arduino sabe cuál es el eje que se mueve (descartamos un encoder y dos finales de carrera) y en qué dirección lo hace (descartamos el otro final de carrera). Esto solo es aplicable al eje X e Y ya que el Z va por separado. La información del eje y sentido del movimiento se encuentra en los pines ENABLE(pin 7 para el X, 6 para el Y) y DIR(pin 10), además añadiremos en el pin 8 el valor del pin DIR pero negado (N.DIR como aparece en los planos). Por último, usaremos una lógica de interruptores para que solo la información de los sensores escogidos llegue al pin correspondiente. Mientras que las otras señales son conectadas a tierra. Usaremos el componente ULN2003.

Su principal uso es como puerta NOT, pero internamente está formado por 7 transistores Darlington en serie conectados a tierra(transistores de alta ganancia, básicamente pueden funcionar como un interruptor) [7]. Cambiaremos su función de puerta lógica a la de interruptor con conexión directa a tierra. Con este sistema las líneas con la información convergen todas en los colectores de los transistores, ya que es este punto donde ocurre la conexión o la no conexión a tierra, para que las líneas no se mezclen se separaran con diodos, donde el cátodo apunta al colector y el ánodo a la línea de tensión, de esta manera la corriente solo podrá circular desde el sensor a tierra o desde el sensor al pin, pero nunca del sensor a otras líneas. El conexionado se realiza de la siguiente manera.



ULN2003 (each driver)



La imagen superior izquierda muestra el esquema simplificado que nos da el fabricante (puerta NOT), a su derecha se muestra como es internamente esa puerta lógica. La imagen de la derecha es el ULN2003 implementado en nuestra pcb, este se encuentra representado mediante transistores donde la base seria la conexión IN y el colector la conexión OUT, destacar que el pin common wheling del esquema eléctrico se conecta a la alimentación de 12V, este pin sirve para evitar sobretensiones, por efectos inductivos, problemas que nuestro circuito no tendrá. Las etiquetas ENC son salidas del encoder, las EN son la señal del Arduino; X.ini, X.final, Y.ini, Y.final son las salidas de los finales de carrera. DIR y N.DIR vienen del Arduino también, N.DIR es la complementaria de DIR.

Ilustración 13 Esquema eléctrico ULN2003

4.4 Diseño del software de control

En esta parte describiremos la composición del programa de control final[6].

4.4.1 Control de movimiento

El control de movimiento lo hace la función `movereje`. Esta función sirve para desplazar las plataformas del eje X e Y un número de pasos previamente calculados por el programa. Previamente se debe de haber seleccionado el eje a mover. Por último, las funciones asociadas a interrupciones acabaran de completar el movimiento. Este se ejecuta en los siguientes pasos:

1. El programa fija el eje del próximo movimiento, además guarda dentro de la variable "`pasos`" que es del tipo volátil, entera y global el valor de los pasos que el motor debe recorrer para alcanzar la posición deseada. Se escoge volátil y global porque es usada por las interrupciones y se escoge entera porque este tipo de variables permite un rango de valores negativos y positivos mayor que 30000.

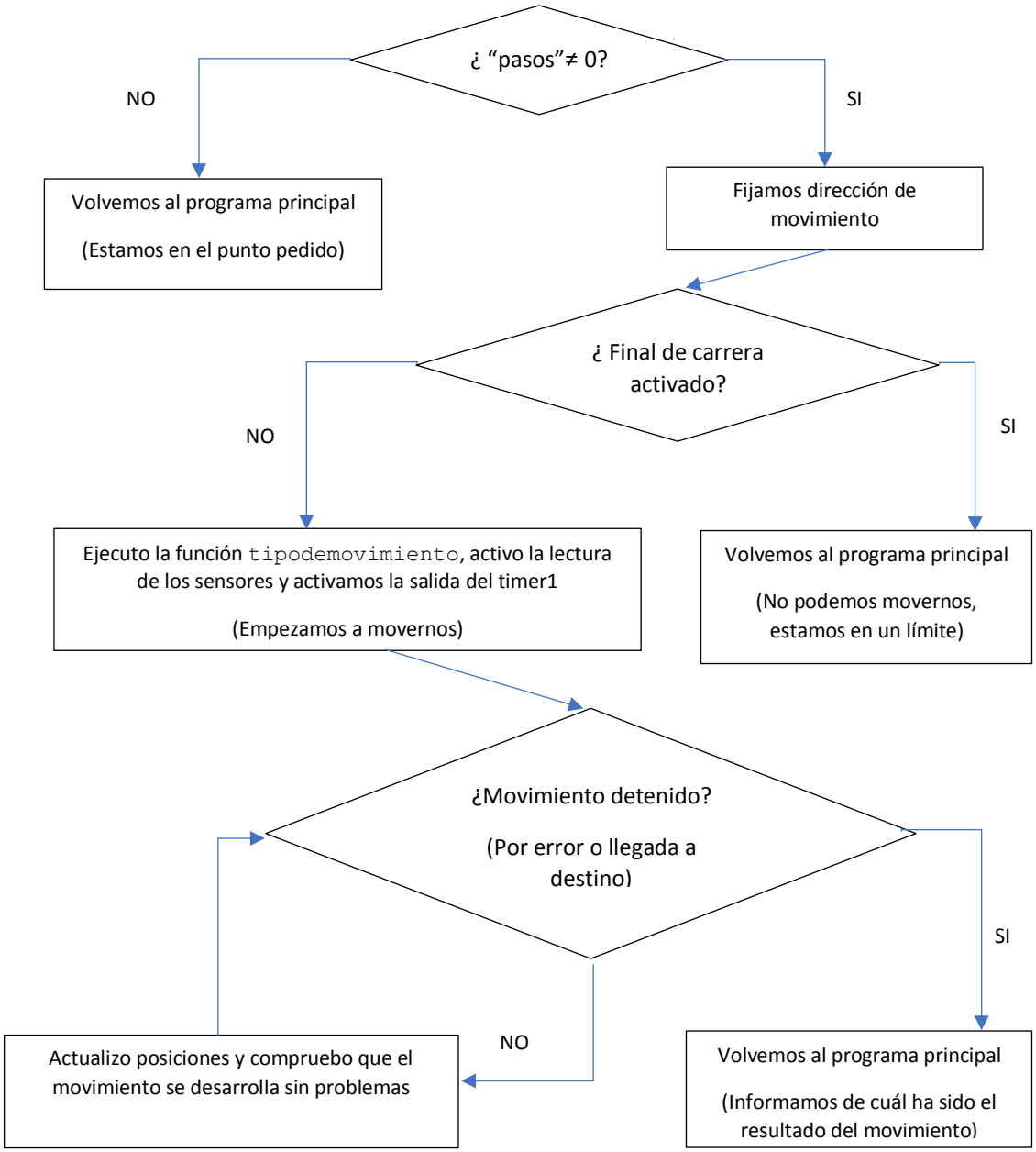
2. Se ejecuta `movereje`. Esta función primero interpreta la variable pasos(puede ser negativa o positiva) que codifica la dirección de giro. Llegados a este punto solo faltaría abrir la salida del timer1 para movernos.

3. Antes del movimiento el programa comprueba si el final de carrera esta activado ya que en ese caso significaría que no podríamos avanzar. También ejecuta la función `tipodemovimiento`; esta función identifica el eje y sentido del movimiento y le asigna un número. De esta manera se podrá intervenir más rápido en caso de error. Esta función también sirve de comprobación previa al movimiento. Es capaz de identificar una configuración de movimiento errónea.

4. Si no hay problemas se activa la salida del timer1 y se inicia el movimiento, a partir de ahora el programa `movereje` solo se ocupa de actualizar la posición global y de testear que el movimiento se realiza de la forma correcta a partir de la información de los sensores; es decir, sin atascos, en el sentido correcto, sin llegar al extremo, etc. El movimiento es detenido por la función asociada al encoder cuando esta cuenta tantos pulsos como la variable pasos. El movimiento también lo puede detener la función del final de carrera si es que este se activa. Cualquiera de estos casos es interpretado por `movereje` y se les asocia un número, el cual puede ser interpretado como un error o como un movimiento completado según el control que se esté ejecutando.

Mientras el movimiento está siendo controlado, la posición global (respecto a la referencia absoluta) es constantemente actualizada por los encoders o por el final de carrera. Respecto a las funciones de interrupción; tendremos `encoder` y `finaldecarrera`. Ambas funciones permanecen desconectadas hasta que la función `movereje` las activa justo antes de activar el timer1. De esa manera se evitan falsas lecturas de los sensores. La función `encoder` se activa cada flanco de subida. Realiza la tarea de contar los pulsos que le llegan del encoder, si estos igualan a la variable "`pasos`" la función activara el pin ENABLE del driver correspondiente. Esta función también detecta el sentido de giro y lo almacena en una variable global para que `movereje` la pueda interpretar. La función `finaldecarrera` activa el pin ENABLE correspondiente y altera la posición global a uno de los límites acotados, al igual que encoder, esta se activa en los flancos de subida. El control de movimiento del eje Z es más simple, se mandan los pulsos necesarios para girar el ángulo pedido, no hay realimentación por sensores. El siguiente esquema detalla el árbol de decisiones asociado al desplazamiento en X e Y.

[6] Los programas se encuentran en el Anexo II: Programas



El árbol de decisiones se corresponde con la realización de un solo movimiento y es interpretado por `movereje`. Decir que volvamos al programa principal es relativo, ya que depende de que función de gestión de movimientos haya llamado a `movereje`. Por ejemplo, si es llamada para una calibración a cero o a un máximo cuando se termine el movimiento se regresará al menú principal (donde se elige que tipo de movimiento hacer). Por otro lado, si estamos ejecutando la función de movimientos sucesivos, ya que los posicionamientos absolutos y relativos podemos hacerlos en secuencias de 18 desplazamientos, volveremos a esta función al terminar un movimiento.

Cada vez que salimos del programa `movereje` almacenamos la información del EXITO/ERROR del movimiento en una variable que hemos llamado “`testigo`” codificada en forma de números del 1 al 5, con el siguiente significado:

1. No se ha detectado ningún movimiento en 200 ms. Se debe a un atasco.

2. El motor gira en sentido contrario. Ocurrirá si el driver o Arduino se estropean durante el movimiento, aunque empíricamente se ha demostrado que este error también se activa cuando hay un atasco por lo que el error 1 es redundante.

3. Ha saltado un final de carrera. Ocurre si se llega a un extremo en el sentido del movimiento, también puede ocurrir por activación externa

4. El movimiento se ha completado, es decir, el encoder ha contado tantos pulsos como el valor de la variable "`pasos`".

5. El movimiento es demasiado lento, han pasado 12 segundos y aun no se ha completado. Este error es muy improbable ya que se debería a que el motor gira a una velocidad mucho menor.

Los resultados 1, 2 y 5 son errores en el movimiento; mientras que el 4 y 5 dependerán de la situación. El 4 significara que se han contado tanto pasos del encoder como pasos que se habían pedido. Esto significa que el movimiento se ha completado satisfactoriamente salvo en el caso que queramos calibrar nuestra posición a un extremo, en ese caso lo que se busca es movernos hasta hacer saltar el final de carrera. Así que, la variable pasos se fija a un valor muy alto que nunca debería ser alcanzado por el encoder. Como ya hemos dicho el 3 es un resultado positivo si queríamos llegar a un extremo, pero en el caso del posicionamiento absoluto significaría que hemos interpretado que nuestra referencia está mal colocada, este problema se soluciona asignando la referencia al lugar en el que estamos. En el caso del movimiento relativo el final de carrera significaría que ya no podemos desplazarnos más y debemos cancelar los movimientos; el movimiento relativo consiste en desplazarse tomando como referencia el punto en el que nos encontramos en ese momento, sin tener en cuenta la posición global. Destacamos que cuando el Arduino se inicia este desconoce su posición absoluta, así que toma como cero absoluto el punto en el que está. Es recomendable hacer un ajuste a cero al iniciar el programa. Recordemos que el cero de referencia son los finales de carrera más cercanos al motor.

El programa `tipodemovimiento` antes nombrado, almacena la información del movimiento en forma de numero entero en la variable "`state`". La función `movereje` da información después del movimiento, mientras que `tipodemovimiento` verifica que los preparativos previos al movimiento son correctos. El significado de los números es el siguiente:

- El 0 y 1 significan que va a haber un doble movimiento. Se comunica por el puerto serie y se resetea el programa.
- El 2, 3 ,4 y 5 son movimientos admitidos. El 2 es moverse en X hacia adelante, el 3 en X, pero hacia atrás, el 4 en Y hacia adelante y el 5 en Y hacia atrás. Estos 4 números nos ayudan a identificar el movimiento y actuar sobre él lo más rápido posible, además también seleccionan si leer la entrada en el pin 4 o 5. Recordemos que estos dos pines están conectados al segundo canal de un encoder, el cual nos da la información del sentido de giro.
- El 6 y 7 significan que no va a haber ningún movimiento, a pesar de que ya lo hayamos programado. Se comunica el error y se reinicia el programa.

No se hace un reinicio del Arduino como tal, solo se vuelve a lanzar el programa. La asignación de los números es la transformación binaria del valor de los 3 pines que definen el movimiento:

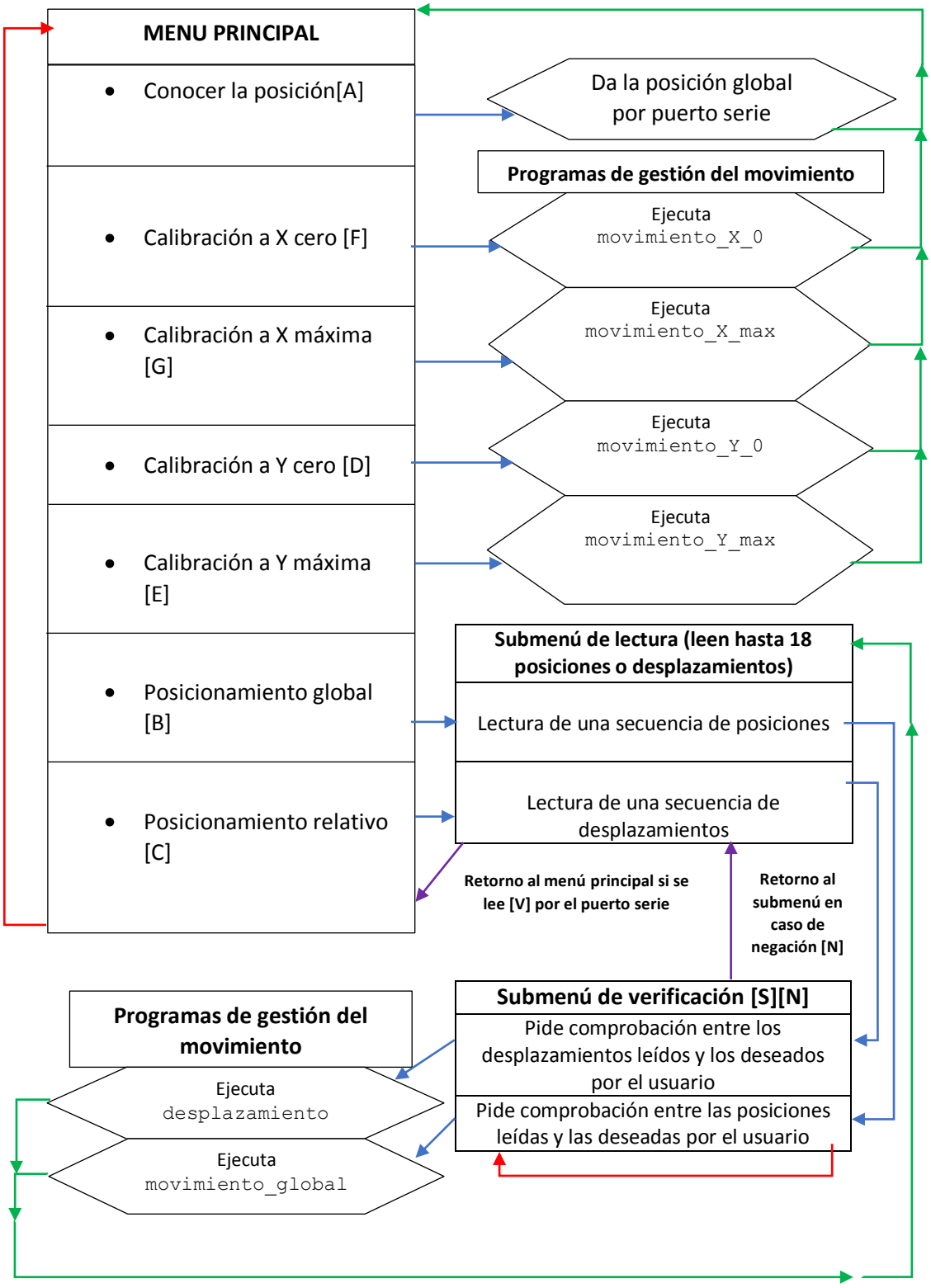
ENX →Pin 7	ENY→ Pin 6	DIR Pin→ 10

Número = 2^2 *ENX + 2*ENY +DIR

Los resultados de “testigo” y ”state” son notificados por el puerto serie con la información de lo que ha ocurrido.

4.4.2 Interfaz de comunicaciones

Ya hemos hablado del control de movimiento, ahora hablaremos del sistema de comunicaciones, el cual se comunica con el usuario a través del puerto serie. La interfaz solo recoge la información dada por el usuario y la almacena de la manera correspondiente para lanzar una función de gestión de movimientos. La interfaz da las siguientes opciones:



Respecto al esquema anterior:

- Las flechas azules significan continuación o siguiente paso.
- Las flechas verdes son retornos por finalización de movimiento.
- Las flechas rojas son retornos por error de lectura.
- Las flechas moradas son retornos al menú anterior.
- Los rectángulos son puntos donde se lee una orden, los hexágonos son donde se ejecuta esa orden.

La siguiente imagen se corresponde con el menú principal del programa:

```
Bienvenido a su posicionador de coordenadas de confianza
las letras y numeros deben escribirse todo junto.
Por ejemplo: X50Y20Z2X20; los movimientos se hacen segun
el orden en que se introducieron las letras.
Respetar los limites de los ejes
X_max=182.55 Y_max=180.27
Al iniciarse el posicionador toma su posicion actual
como 0, se recomienda una calibracion a 0 o a un maximo.


¿Que desea hacer?
[A]Posicion actual
[B]Movimiento, coordenadas globales
[C]Movimiento, coordenadas relativas
[D]Calibrar Y a 0
[E]Calibrar Y a Y_max
[F]Calibrar X a 0
[G]Calibrar X a X_max
```

Ilustración 14 Menú principal

En el menú principal podemos elegir que deseamos hacer introduciendo la letra clave que aparece al lado de cada orden. En el caso de introducir B o C iremos a un submenú donde tendremos que enviar una cadena de texto que codifique las posiciones sucesivas que deseamos hacer, posiciones en el caso de B y desplazamientos en el caso de C. Si en su lugar introducimos la letra V retrocederemos al menú principal. Una vez enviada la cadena de movimientos el Arduino la leerá y mostrara por puerto serie lo que ha leído para que el usuario pueda comprobar si coincide con lo introducido. En caso de querer confirmar el movimiento se introduce un S en caso contrario una N. Si en algún momento un programa de lectura lee un carácter que no corresponda con lo que el programa pide (salvo retorno de carro y salto de línea) este informara que no ha entendido la orden y pedirá que se repita, esto se corresponde con las flechas rojas del gráfico.

Las posiciones se leerán en milímetros y grados. Además, para que el Arduino pueda entenderlas se deben introducir una cadena de texto codificada, que ya hemos nombrado antes, la cadena es una secuencia letra y posición, por ejemplo: ”X50.56Y20.33Z2.3Y20.34X20.4”. Donde X, Y o Z son los ejes; los decimales se introducen con punto y no coma. La sucesión de posiciones es el orden en el que se realizaran. La comunicación entre Arduino y ordenador se ha elegido a 9600 baudios.

4.4.3 Gestión de movimientos

Los programas de gestión de movimiento son los encargados de interpretar las órdenes del usuario almacenadas por la interfaz de comunicaciones y convertirlas en movimiento ajustando los parámetros de la función `movereje`. Como hemos visto en el apartado anterior vamos a tener 6 funciones que definirá 6 tipos de movimiento. Tendremos 4 programas que sirven para ajustar la plataforma a un extremo que tomará como referencia; en este caso la exactitud es importante. Los 4 se parecerán mucho en cuanto a funcionamiento y programación. Por otro lado, tenemos 2 programas de posicionamiento. Uno será para posiciones globales tomando como referencia el cero absoluto situado en los finales de carrera de los motores (salvo que acabemos de iniciar el programa, en este caso el cero es el punto en el que iniciamos). El otro programa hace desplazamientos relativos tomando como referencia el punto del que parte.

4.4.3.1 Programas para ajustar a un extremo

El usuario ha pedido que ajustemos la plataforma a un extremo así que activamos las salidas digitales correspondientes al ENABLE del eje que corresponda, configuramos el valor del pin de dirección (DIR) y el de su complementario (N.DIR), según la dirección en la que nos vayamos a mover. Antes de ejecutar `movereje` comprobamos que el final de carrera no este activo; leyendo el valor del pin 2 (asociado a los finales de carrera). En caso de un nivel alto, invertiremos el nivel del pin de dirección y activaremos el timer1 durante 800ms; tiempo suficiente para alejar a la plataforma del final de carrera. Hecho esto configuramos el valor de la variable “`pasos`” a 30000 o -30000, según queramos avanzar o retroceder. Este valor es imposible de alcanzar, por nuestra limitación de recorrido, por último, ejecutamos `movereje`. Los programas correspondientes son; `movimiento_X_max`, `movimiento_Y_max`, `movimiento_X_0` y `movimiento_Y_0`.

Por último, el programa informará si la tarea se ha completado satisfactoriamente o se ha encontrado un error.

4.4.3.2 Programa para movimientos absolutos

El programa encargado de esta tarea es `movimiento_global`. Este programa recibe de la interfaz de comunicaciones el número de movimientos que se han pedido, ya que estos pueden ser hasta 18. En dos vectores globales, “`orden_mov[19]`” y “`posicion_deseada[19]`” se han guardado el orden de los movimientos. El primer vector está formado por caracteres, estos se corresponden con la secuencia de ejes a mover y el segundo vector complementa al primero como la posición a alcanzar en cada caso. El programa está formado por un bucle que lee estas posiciones desde 1 hasta el número de posiciones pedidas.

En cada iteración del bucle se transforma la posición deseada en pasos, realizando la conversión matemática correspondiente. Para realizar la conversión debemos restar la posición absoluta con la actual y realizar la conversión; ya que nosotros solo podemos transformar en pasos los desplazamientos. Previamente se debe haber comprobado que la posición no excede los límites. Con los pasos calculados se ejecuta `movereje` si el resultado es satisfactorio se pasa a la siguiente posición de la secuencia, si el resultado ha sido chocar con un final de carrera el

movimiento se repite. Si cualquier otro error es activado este se informa y se cancelan los movimientos.

4.4.3.3 Programa para movimientos relativos

El programa encargado de esta tarea es `desplazamiento`. El programa recibe de la interfaz de comunicaciones el número de desplazamientos que se han pedido. Estos pueden ser hasta 18. En los vectores globales, "`orden_desp[19]`" y "`desplazamiento_deseado[19]`" se han guardado el orden de los desplazamientos. El sistema es el mismo que el anterior. El programa está formado por un bucle que lee estas posiciones desde 1 hasta el número de desplazamientos pedidos. En cada iteración del bucle se transforma el desplazamiento en pasos, realizando la conversión matemática correspondiente. En este caso no se comprueba si el desplazamiento excederá los límites. Con los pasos calculados se ejecuta `movereje` si el resultado es satisfactorio se pasa Al siguiente desplazamiento de la secuencia, si el resultado ha sido chocar con un final de carrera o cualquier otro error, el programa informa y cancela los movimientos restantes.

En ambos programas la posición global es actualizada respecto a la referencia global (no local); la diferencia del programa de desplazamientos es que este no tiene en cuenta la posición en la que se encuentra. Respecto al posicionamiento en Z este es siempre relativo ya que no tenemos una realimentación que nos permita fijar una referencia. La referencia es el punto en el que se encuentra en ese momento. La ejecución del movimiento en Z consiste en convertir la información angular en pasos, y mandar tantos pulsos como pasos se han pedido por su pin STEP. Para conocer el funcionamiento de la comunicación serie y las interrupciones de Arduino se reunió información de los siguientes artículos del blog de Luis llamas [8] [9]. Por otro lado, el conexionado de los pines del Atmega328p en la plataforma de Arduino se dedujo de la siguiente referencia [10], fue necesario para conocer el conexionado de los timers y evitarnos recurrir a los esquemas eléctricos.

4.5 Finalización del apartado de planificación

Se pasarán los planos definidos a fabricación, tanto la pcb como los de impresión 3D, y también el programa final se implementará[7]. Pasamos a la parte de montaje donde aplicaremos nuestro diseño teórico a la realidad.

5 Montaje

Este será el último apartado en relación con la construcción del prototipo. En este punto disponemos de todos los materiales necesarios y sabemos cómo los vamos a usar. Consideraremos que el programa para el funcionamiento ya está terminado y solo falta implementarlo. Este se corresponde con el del punto 3.4. En este apartado no pretendemos detallar el proceso de construcción como tal, ni hacer un manual de instrucciones, queremos explicar los problemas que han surgido al intentar materializar el punto de vista teórico visto en el apartado de planificación, así como hablar de pequeños cambios de diseño. Dividiremos el montaje en tres partes:

- Montaje mecánico
- Fabricación y montaje de la shield de Arduino
- Implementación del software

[7] Consultar anexo Programas

5.1 Montaje mecánico

El montaje mecánico se refiere a la colocación de la estructura móvil sobre el soporte de aluminio. Nos referimos a la estructura que transmite el movimiento y soporta las piezas responsables de este. Estas piezas son las plataformas, ejes, motores, encoders, finales de carrera, etc. Este montaje se divide en tres partes:

- Montaje de estructura móvil. Esto se refiere a todos los componentes situados por encima del eje X. Estos son el eje Y junto con el Z. Los llamamos estructura móvil porque están sujetos al avance del eje X.
- Montaje eje X. Se refiere a colocar la estructura anterior sobre los cimientos fijos del eje X.
- Colocación de componentes eléctricos. Estos son los motores, encoders y finales de carrera.

5.1.1 Montaje estructura móvil

Se decide empezar por aquí ya que se considera que es la estructura más critica o la que más sufre con la falta de paralelismo en los ejes de su base (guías del eje X). Esto se corresponde con construir la estructura compuesta por las tres plataformas que hemos visto. Para poder fijar los componentes de apoyo sobre las plataformas se incluyeron agujeros (todos pasantes) en el diseño previo a la impresión 3D. Los diámetros se corresponden con la métrica de tornillo que cada apoyo necesite [8].

Los agujeros se roscarán manualmente sobre las plataformas ya impresas. La relación entre agujero y métrica será de un diámetro 3.3 para métrica 4, diámetro 2.90 para métrica 3.5 y diámetro 4.20 para métrica 5; estos datos han sido extraídos de una tabla para roscado. Las métricas que aparecen son 4, 3.5 y 5 con agujeros pasantes en todos los casos.

Una vez hechas todas las roscas se puede pasar a la colocación de las guías y ejes. Se uso un modelo descendente es decir primero se montó el eje Z después el Y para finalmente montar el eje X. Al empezar a montar el eje Z se ve que las guías deslizantes tienen un equilibrio muy delicado. Nos dimos cuenta de que el más ligero desajuste en los apoyos deslizantes provocaba que estos no deslizasen bien sobre las guías (falta de paralelismo y desviaciones). Por lo que encontrar el equilibrio era complicado, pero aún más complicado era mantenerlo. Para conseguir el equilibrio había que jugar con el apriete de cada tornillo, pero una vez conseguíamos el equilibrio este se perdía fácilmente, porque los tornillos no estaban correctamente sujetos. Es aquí donde hacemos la primera modificación. Primero ajustábamos manualmente los tornillos para que las guías deslicen correctamente, después para fijarlos rellenábamos el espacio entre apoyos deslizantes y plataformas con resina epoxi.

El siguiente punto crítico del diseño surgió cuando queríamos montar el eje Z sobre las plataformas que sujetaban el eje Y. Nos dimos cuenta de un error en la planificación, concretamente en el diseño de los planos. Este hizo que las posiciones de los apoyos del eje Z no fuesen coherentes con los del eje Y. Por eso cuando fuimos a montar el eje Z las guías de su base no estaban alineadas con sus apoyos en el eje Y. Recordemos que las guías deslizantes del eje Y tienen su apoyo fijo en este mismo eje, pero su apoyo deslizante está en la plataforma del eje Z . Esto error pudo llevar al proyecto a un retraso considerable ya que tendríamos que volver a encargar la fabricación de las plataformas. Así que se optó por una reparación in situ, como podemos observar en la imagen de la siguiente página:

[8] Consultar anexo Especificaciones de los componentes

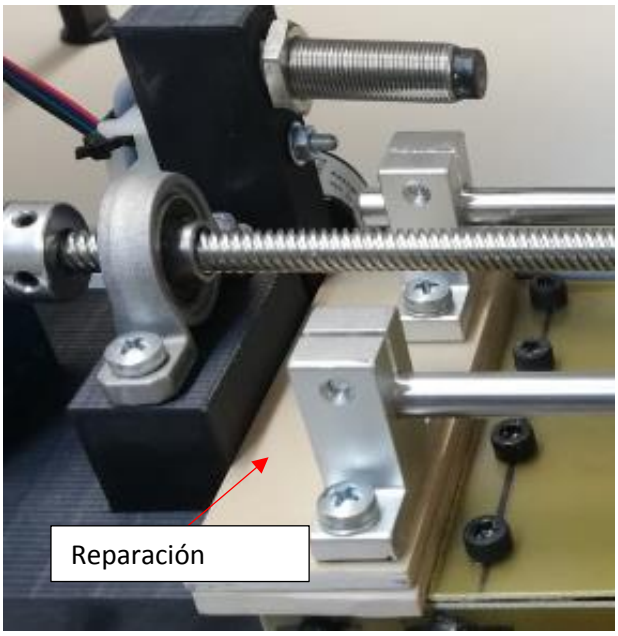


Ilustración 15 Reparación del eje Y

La reparación consistió en añadir dos láminas de madera en cada lado del eje Y . Estas servirían de soporte para los apoyos fijos. Les daban la altura que les faltaba y al no tener aún agujeros permitirán colocar éstos en el lugar correcto, corrigiendo así la posición de los apoyos fijos. En el anexo Planos aparece la plataforma Z ya corregida.

El siguiente problema que tuvimos trataba de que un juego de carriles no permitía a su eje deslizar correctamente. Debido a un exceso de fricción por un error de fábrica. No era algo muy notorio, pero como también nos encontramos ante una falta de espacio por la longitud del eje Y, se optó por cortar estos carriles. Es decir, se serró parte de la plataforma del encoder Y, concretamente la parte donde se encontraban sus carriles. Ganamos en espacio y reducimos la fricción, aunque ahora el eje Y solo deslizase sobre tres guías.

El ultimo problema en el montaje de las plataformas surge ante nuestra desconfianza de que los carriles deslicen a la misma velocidad. Si no lo hacen el eje Y se torcerá y el avance se frenará o no se hará correctamente. Las plataformas que constituyen el eje Y están unidas por las guías que soportan el eje Z, en este punto tenemos la duda de si estas guías serán capaces de transmitir la fuerza motriz de X correctamente de un extremo a otro del eje Y, dudamos de la rigidez de la estructura. Ante la duda decidimos rigidizar el eje Y. Unimos una plataforma con otra mediante láminas de fibra de vidrio. Podemos ver el resultado en la siguiente página.

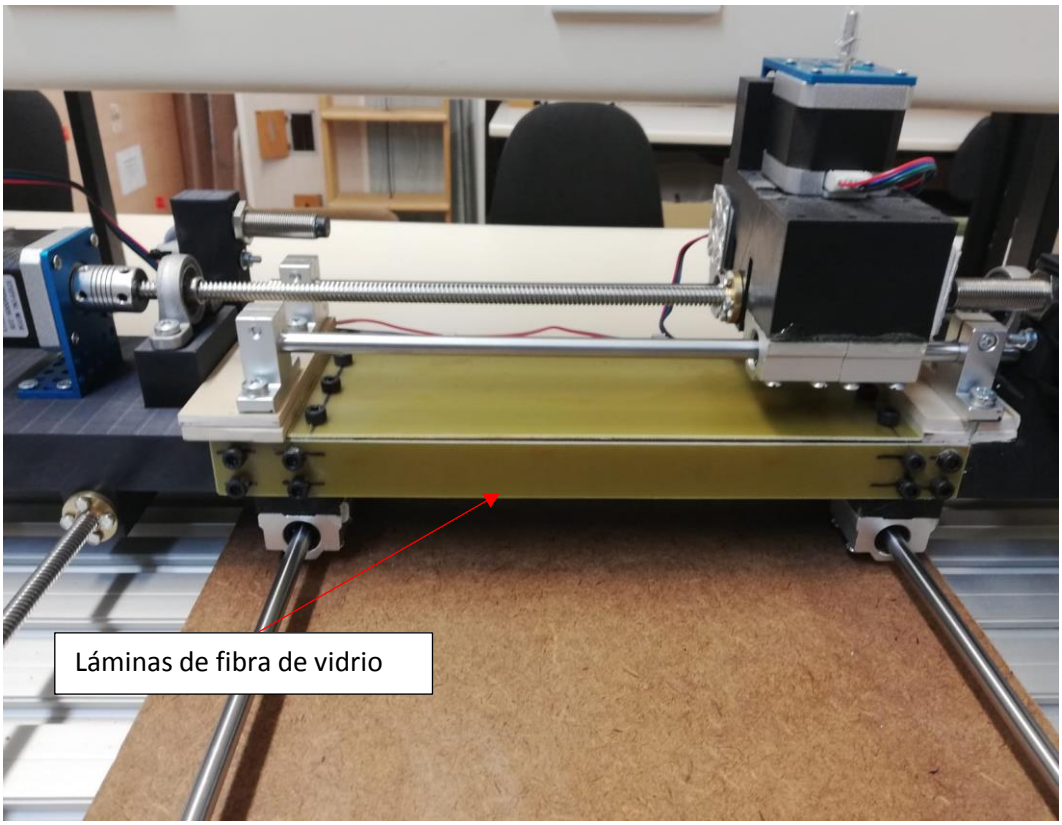


Ilustración 16 Unión de plataformas

El objetivo era convertir las dos plataformas en una a través de la unión atornillada de la imagen. Así aseguramos que las dos plataformas se muevan como si fuesen una sola.

5.1.2 Montaje del eje X

Con la estructura móvil terminada debemos pasar a colocarla sobre el eje X. Recordemos que los rodamientos del eje X estaban sujetos por apoyos ajustados a ojo. Es decir, su posición no era tan precisa como en el caso de la impresión 3D. Debido a esto apareció un problema de desajustes. Nos dimos cuenta del problema al colocar el encoder y motor del eje X, posterior a la colocación de la estructura móvil. El problema ocurría porque la altura de los rodamientos del eje X estaba ligeramente desviada con la del otro rodamiento, además de con el motor y encoder. Esto producirá que el eje este flexionado por lo que se requería bastante fuerza para generar avance. Simplificando sería algo similar a tener una viga biapoyada a flexión (como en la imagen de la página siguiente) e intentar hacer correr la tuerca de avance por ella.

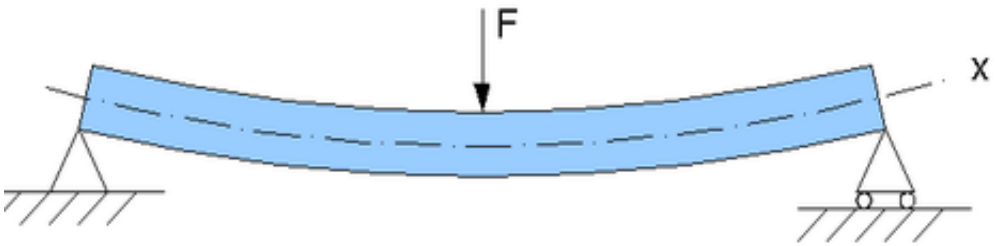


Ilustración 17 Viga biapoyada a flexión

Finalmente se decidió quitar un rodamiento, dejando solo el rodamiento del encoder. Esto solucionó el problema en gran medida. La tuerca avanzaba libremente salvo cuando nos acercábamos mucho al encoder, donde había que aplicar más fuerza. Sin embargo, este tramo era cortado por la presencia del final de carrera, asegurando así que la tuerca no tenga dificultades en su recorrido. Quitar un rodamiento no supuso ningún problema para la estructura. La función del rodamiento era para transmitir la carga del tornillo sin fin a la base de la estructura, pero a estas alturas la carga era distribuida por los carriles deslizantes. Se podría quitar los dos rodamientos y el mecanismo funcionaría igual.

5.1.3 Colocación de componentes eléctricos

Esta parte se refiere a la instalación de los componentes sobre la estructura final. Destacamos solo dos cosas.

Surgió un problema de cómo llevar el cableado hasta la shield del Arduino, además el problema era acentuado por el cableado de los componentes móviles. La solución paso por introducir los cables en una oruga de plástico, además del uso de bridas para sujetar los cables. Como se muestra en la imagen.

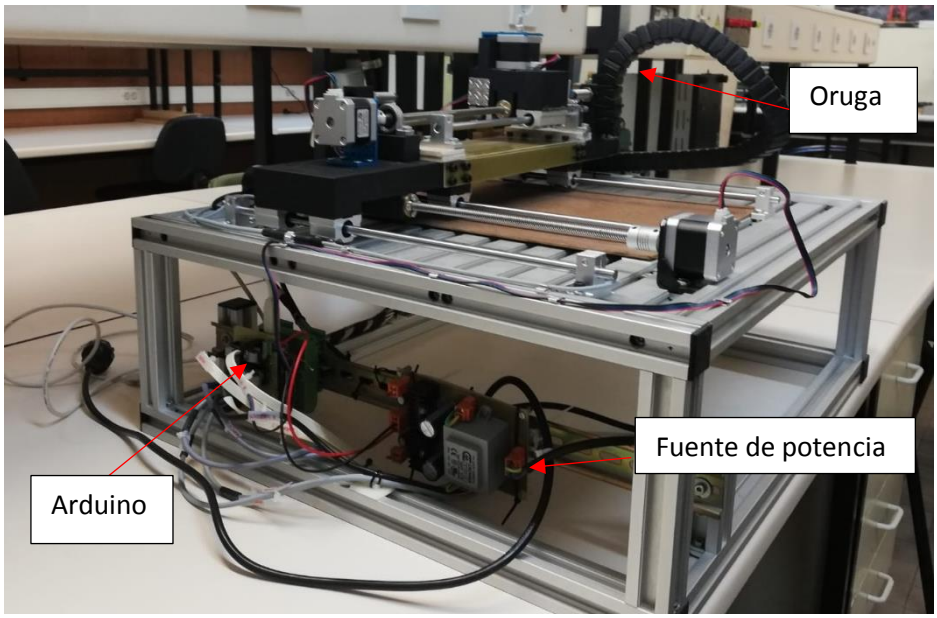


Ilustración 18 Cableado componentes eléctricos

En la imagen también se observa cómo se instaló la fuente de potencia y el Arduino sobre la base de aluminio.

El segundo punto para destacar fue que tuvimos que usar dos tipos distintos de final de carrera. El que habíamos usado hasta ahora era inductivo, pero no quedaban más en el inventario. Pero disponíamos de unos finales de carrera magnéticos, basados en un sensor hall. Su respuesta era idéntica que los inductivos. La única diferencia fue que tuvimos que colocar un imán en dos de los carriles deslizantes que iban sobre el eje X. De igual manera para activar el final de carrera inductivo tuvimos que colocar una chapa metálica sobre la plataforma del eje Z.

5.2 Fabricación y montaje de la shield de Arduino

La placa de circuito impreso se fabricó en la Universidad. A continuación, mostramos el módulo de conexiones en el software de Fritzing además del circuito finalmente montado.

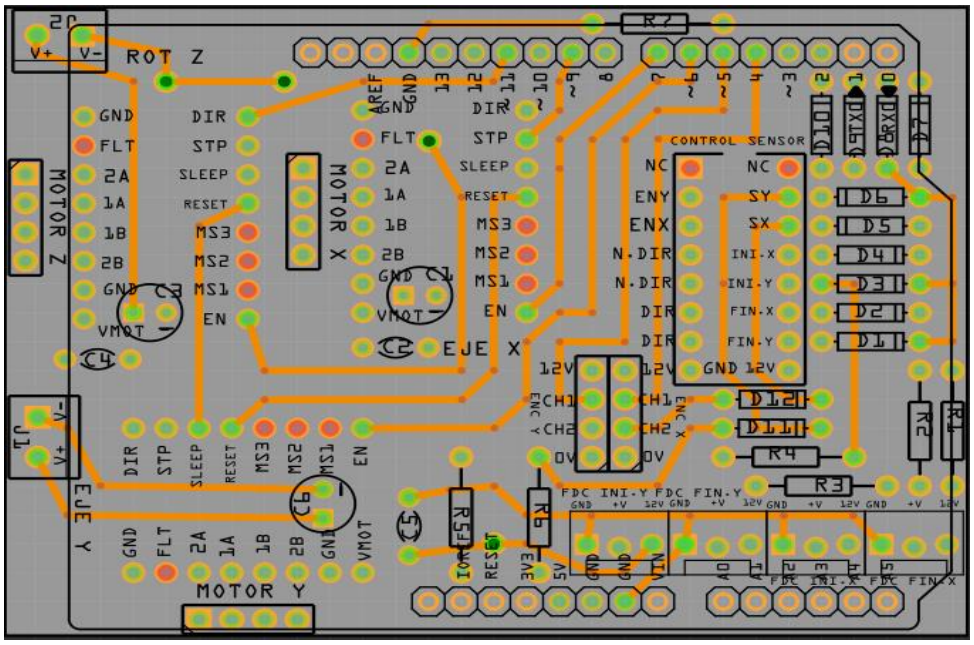


Ilustración 19 PCB capa inferior

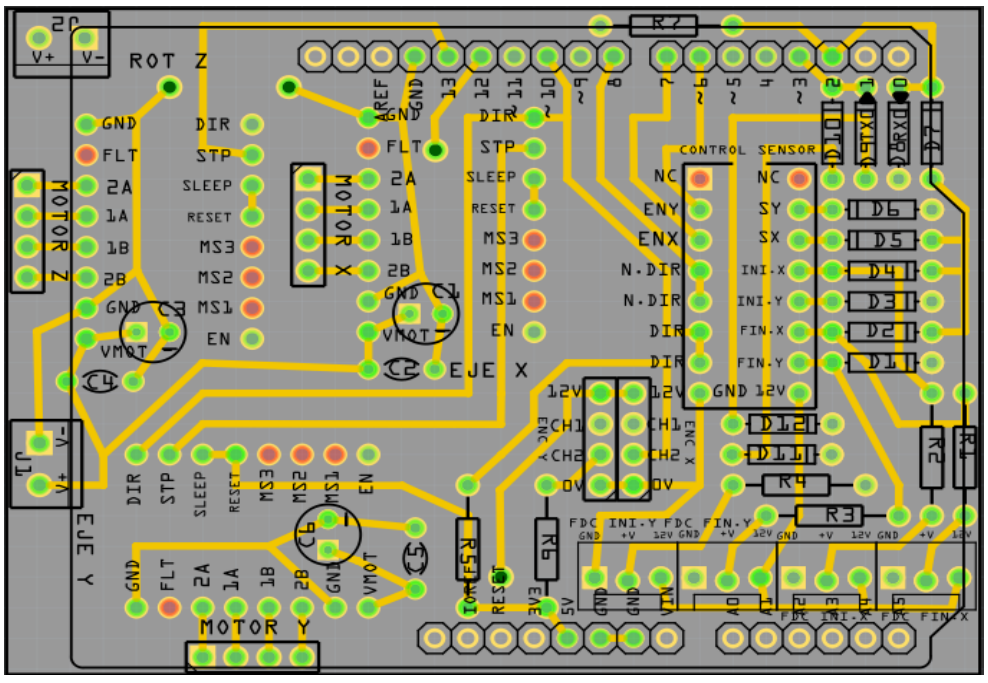
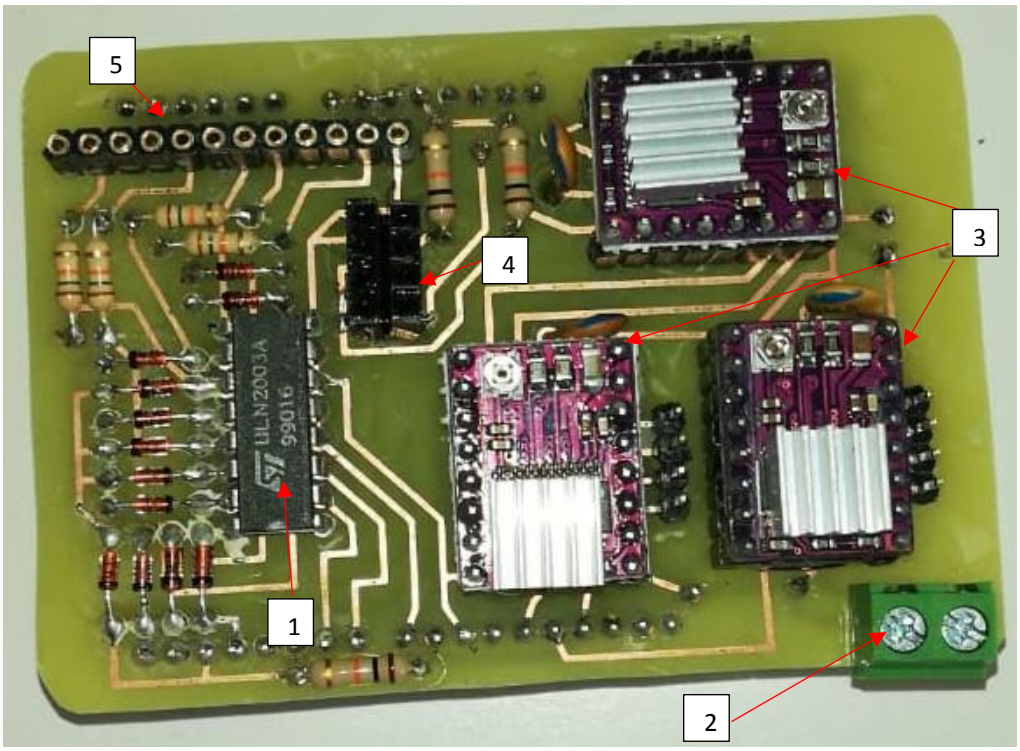


Ilustración 20 PCB capa superior



Montaje final. El punto 1 se corresponde con el driver ULN2003. El punto 2 es la alimentación de 12 voltios. El 3 son los drivers de potencia junto a ellos se encuentra la conexión del motor y debajo los condensadores de 100uF. El 4 son las conexiones hembra del encoder. El punto 5 son las conexiones hembra para los finales de carrera.

Ilustración 21 PCB montaje final

Del montaje hay que destacar que tanto los taladros y soldadura se realizaron de forma manual. La agresividad de la fabricación por acido y la inexactitud del taladro manual obliga a que aumentemos el tamaño de los pads (anillos de conexión), y también a aumentar la anchura de los tracks (caminos de corriente). Esto se hace por que en la fabricación las dimensiones se reducen. Lo más importante era hacer los pads de al menos 2.3 mm de diámetro exterior.

Para evitar quemar componentes durante la soldadura manual, será conveniente en futuros diseños hacer las conexiones en la capa inferior. Es más cómodo soldar en la capa inferior que en la superior, ya que si se suelda en la capa superior el propio componente estorba.

Durante el montaje de los componentes surgieron los siguientes imprevistos:

- El conector de alimentación de la librería de Fritzing no coincidía con ninguno de los que había en inventario, por lo que se tuvo que hacer una modificación manual en la placa para que el nuevo componente pudiese ser soldado. Además, inicialmente iba a haber dos conectores, pero debido a este imprevisto uno tuvo que ser quitado. Los terminales de este estaban eléctricamente conectados con el primero (equipotenciales). Su uso era para llevar los 12 voltios a una segunda shield en caso de que fuese necesario.
- La escasez de pines hembra nos lleva a usarlos de dos tipos. Como vemos en la imagen, los del final de carrera son diferentes. Estos son pines torneados y son más fáciles de soldar que los otros.
- En el diseño inicial se pretendía que la shield tuviese una conexión macho por debajo y hembra por arriba. Para poder añadir futuras placas. No disponíamos del material así

que se optó solo por la conexión macho (necesaria para conectarnos al Arduino). Incluso aquí sufrimos la escasez de material ya que nos disponíamos de suficientes pines para todas las conexiones. Había pines macho de sobra en el inventario, pero no de la longitud que necesitábamos. Se opto por conectar pines macho más cortos en aquellas conexiones entre la shield y Arduino que no se usasen. Como las salidas analógicas y se reservó los pines más largos para las demás conexiones. Recalcamos que los pines cortos no llegan a conectar con el Arduino, solo sirven como guía y apoyo. Este diseño hace que el montaje de la shield sobre el Arduino sea delicado ya que los pines se tuercen con mucha facilidad.

Finalmente se hará una última corrección en el esquema eléctrico, pero hablaremos de ello en el apartado siguiente.

5.3 Implementación de software

El programa que se expuso en el apartado de planificación no sufrirá ninguna modificación y es el que se usará finalmente.

Antes de subir nuestro programa principal a Arduino hacemos una comprobación de que el mecanismo funciona correctamente, como hicimos en el apartado de diseño de un eje. Haremos un programa que nos permita conocer cuántos pulsos se necesitan en el motor (bucle abierto) para ir de un extremo a otro en cada eje, para así conocer el recorrido en milímetros que tenemos. También haremos un programa similar, este recorre los ejes de un extremo a otro, pero que en lugar de contar los pulsos del motor los cuente en el encoder y así conocer la relación de pasos entre motor y encoder. Necesitamos ambos valores para acotar el movimiento y para calcular la conversión de desplazamiento a pasos de encoder. Parámetros necesarios para el programa principal. Hablaremos primero de los resultados y luego del problema encontrado. Los programas usados son `limite_por_motor` y `limite` ambos están en el anexo Programas.

EJE	Pasos por motor	Pasos por encoder
X	4485	8912
	4485	8912
	4485	8913
	4485	8911
Media	4485	8912
Y	4429	8890
	4429	8890
	4429	8890
	4429	8889
Media	4429	8889.75

Con el promedio de pasos del motor y sabiendo que cada 200 pulsos avanzamos 8.14 mm, el eje X tiene un recorrido de 182.554 mm mientras que el Y de 180.275 mm. La relación entre los pasos del encoder y del motor es de 2 aproximadamente como indico el fabricante.

Durante el conteo de pasos del encoder surgió un problema. El Arduino no contaba los pasos, esto no era debido a un fallo del software, sino a un fallo electrónico. La siguiente imagen muestra la salida que recibía el Arduino en la lectura del encoder.

Ilustración 22 Salida del encoder

Lo que se intenta ilustrar en la imagen es que la salida no es cuadrada y además no llega hasta 0 voltios. El origen del problema estaba en el circuito del ULN2003. Recordemos que este circuito nos permitía escoger que salidas leer. Durante el modelado de este circuito se consideró a los transistores como interruptores y a los diodos como cortocircuitos, sin tener en cuenta las especificaciones referidas a su funcionamiento. El problema aparece en los diodos que conectan la salida de los encoders con el pin 3 del Arduino (la salida de los encoders hacia el Arduino es a través de un diodo individual).La conexión es parecida a la imagen inferior.

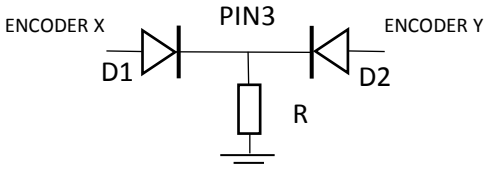


Ilustración 23 Salida de los encoders

El nudo donde ambos cátodos coinciden es la entrada al microcontrolador. Este esquema es un selector de máximos. Este siempre escoge la línea por la que pasan los pulsos eléctricos, ya que la otra se considera que permanece a 0 voltios porque solo un encoder puede funcionar al mismo tiempo, debido al diseño. En la imagen, la resistencia que vemos se corresponde con la alta impedancia de entrada al Arduino, esta resistencia es de 100MΩ. Por lo que el circuito es como si tuviese un terminal al aire o sin conectar. Esto provoca que D1 y D2 no entren en estado de conducción. Para que el valor del PIN3 sea igual al de la línea que corresponda en ese momento, será necesario que el diodo correspondiente esté en conducción. De lo contrario es posible que el sistema no funcione de la manera que queremos. Para forzar a que el diodo se polarice colocaremos una resistencia de pull-down en paralelo a la resistencia interna del Arduino. La resistencia se escoge probando hasta que la salida del encoder sea cuadrada, finalmente se escogió una resistencia de 50kΩ. Esta resistencia provoca que la salida sea cuadrada y que el valor de la tensión baje casi hasta cero (siempre queda una tensión remanente debido a los transistores). La resistencia es soldada en los pines de la shield del Arduino.

Con esto se podría dar por terminada la construcción del prototipo y solo quedaría subir el programa de control al Arduino, el programa se llama `posicionador_usuario` a este programa le caracteriza que es muy interactivo y fácil de usar para un usuario que use el posicionador por primera vez. Llegados a este punto se nos pidió una modificación del programa, se consideraba que el primer programa estaba hecho para un uso personal o comercial y se nos

pidió un programa que sea fácil de automatizar. Es decir, un programa más simple que reduzca la comunicación lo máximo posible, que estuviese pensado para ser un usado por otro programa, para un uso más automático.

Este nuevo programa se llama `posicionador_automatizado`[9]. Básicamente la esencia es la misma que el primer programa, pero borrando todos los mensajes de interacción. El funcionamiento es el mismo, solo cambia la comunicación para mandar órdenes y recibir informes. Empezaremos hablando de los informes, mensajes que manda el Arduino por puerto serie.

- "I". Significa que podemos mandar posiciones
- "OK". Significa que el posicionamiento o la secuencia de estos ha sido completada con éxito.
- "ER0". Significa error de lectura.
- "ER1". Significa atasco superior a 200 ms.
- "ER2". Significa que ha habido un giro en sentido contrario, también puede significar atasco.
- "ER3". Significa que se ha activado el de final de carrera.
- "ER4". Este error solo aparece si buscando un final de carrera (posicionamiento a un extremo) contamos 30000 pasos por el encoder, a priori imposible salvo intervención externa.
- "ER5". El movimiento es demasiado lento, ha tardado más de 12 segundos.
- "ER6" y "ER7" . Significa que hay doble movimiento, es un error de preparación, se reiniciará el programa.
- "ER8" y "ER9". Significa que no va a haber movimiento, es un error de preparación, se reiniciará el programa.

Los siguientes mensajes son las ordenes que interpreta el Arduino:

- "A". Si se lee este mensaje el posicionador se mueve a 0 en X
- "B". Si se lee este mensaje el posicionador se mueve a la posición máxima en X
- "C". Si se lee este mensaje el posicionador se mueve a 0 en Y
- "D". Si se lee este mensaje el posicionador se mueve a la posición máxima en Y
- "P". Si se lee este mensaje se muestra la posición absoluta por puerto serie, POR ejemplo: X6.5Y7.2
- "G". Esta letra indica que se quiere hacer un posicionamiento absoluto. Debe de ir seguida de la posición, por ejemplo: X3.2Y76Z2
- "R". Esta letra indica que se quiere hacer un posicionamiento relativo. Debe de ir seguida del desplazamiento, por ejemplo: X60.7Y40.9Z10.6

[9] Ambos programas están en el Anexo programas

La diferencia del programa `posicionador_automatizado` radica en que la comunicación es más simplificada. Por lo demás sigue siendo igual a `posicionador_usuario`. Como podemos observar los mensajes de error son los mismos en ambos programas, incluso los códigos de error en muchos casos son iguales.

Con el prototipo ya en estado funcional se nos pidió una última especificación. El software no daba ninguna especificación sobre la posición del eje Z, debido a que este se encuentra en bucle abierto. Se nos pidió dar de alguna manera más información sobre la posición del eje Z. A estas alturas era complicado hacer una implementación electrónica o de software que realimente el ángulo girado de Z, además la discretización del giro en Z era mucho más imprecisa que la discretización del desplazamiento de los ejes anteriores. En Z cada paso era 1.8º mientras que en los ejes X e Y el tornillo sin fin hacia corresponder cada paso con 0.04 mm. La solución por la que se opto fue construir una última pieza. Usando los mismos materiales de una placa de circuito impreso y el mismo método de fabricación de este se hará un disco con marcas de los ángulos, algo muy parecido a un transportador. El objetivo es colocar este disco encima del eje Z para poder tener una referencia angular para este eje.

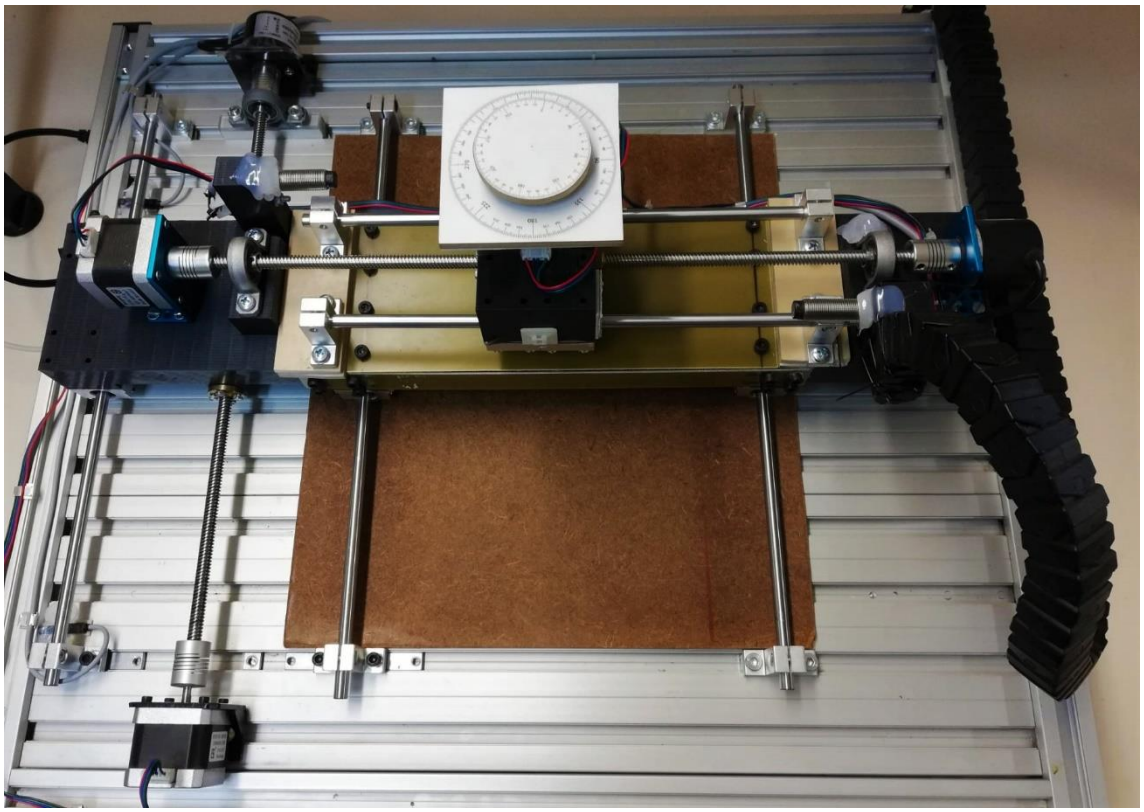


Ilustración 24 Prototipo de posicionador de coordenadas

Con esto se puede dar por terminado el montaje del prototipo, este se encuentra ya en estado funcional. Solo quedaría por ver cuál es el error y la desviación que este tiene cuando quiere alcanzar una posición.

6 Calibración

En este último apartado queremos estimar el valor de la desviación de la medida y el valor del error respecto a la medida real. El proceso por seguir será desplazar el posicionador una distancia fija y repetir este proceso para crear una nube de puntos. La dispersión de estos puntos será medida con una regla. La exactitud de la regla se considerará que es mucho mayor que la de nuestro posicionador.

Antes de empezar tendremos que hacer una serie de suposiciones:

- Consideramos que el error e incertidumbre dependerán de la zona del tornillo en la que estemos. Dividiremos el tornillo en 3 zonas de 5 cm donde haremos nuestras pruebas. La longitud del eje es superior a 15cm así que el recorrido sobrante será una distancia de seguridad al final de carrera. Es decir, durante nuestro estudio queremos evitar la activación del final de carrera ya que este reajusta el valor del 0 absoluto eliminando cualquier error acumulado.

- El error y desviación de la posición son debidos a un error y desviación del avance por paso. Por lo que el error en la posición es proporcional al desplazamiento realizado.

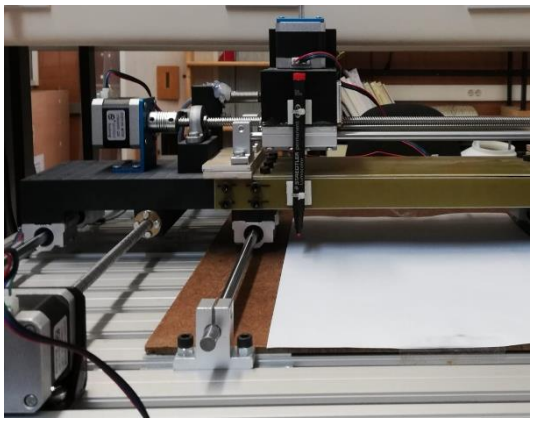
- El error en un eje es independiente a la posición del otro eje. Para ello habrá que suponer que la carga de un eje sobre el otro no afecta al avance por paso de manera considerable. Además, tendremos que comprobar que existe perpendicularidad para poder confirmar la independencia entre los ejes. La perpendicularidad se comprueba haciendo que los ejes hagan todo el recorrido. Primero el eje X y después el Y para que la trayectoria resultante sea representada con un rotulador acoplado al eje Z. El ángulo de la trayectoria fua comparado con la esquina de una escuadra, el resultado fue 90º.

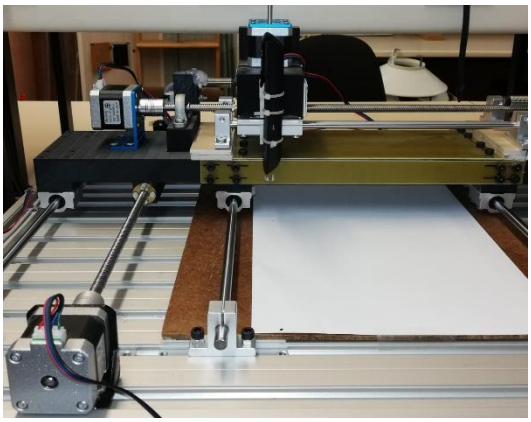
- Los instrumentos de medida utilizados serán los encoders y la regla. Consideraremos estos como ideales. Como son instrumentos de medida asumiremos que su error es muy pequeño en comparación al del posicionador.

Los errores en nuestras medidas se deben al error en el avance por paso. Pero este es afectado por diversos factores como pueden ser:

Error en el paso del tornillo sin fin, error en el paso del encoder, desajustes de los ejes, mala sujeción de los apoyos, deslizamiento sobre el tornillo sin fin, la inercia de giro, etc. Son varios los fenómenos adversos a nuestro posicionamiento. A pesar de esto, se puede intuir antes del estudio que la precisión que obtendremos será elevada. Debido a que por construcción el encoder y motor paso a paso son instrumentos muy precisos y porque los mecanismos de tornillo sin fin son muy estables y de bloqueo rápido.

A continuación, en la siguiente página mostramos los montajes usados para la calibración:



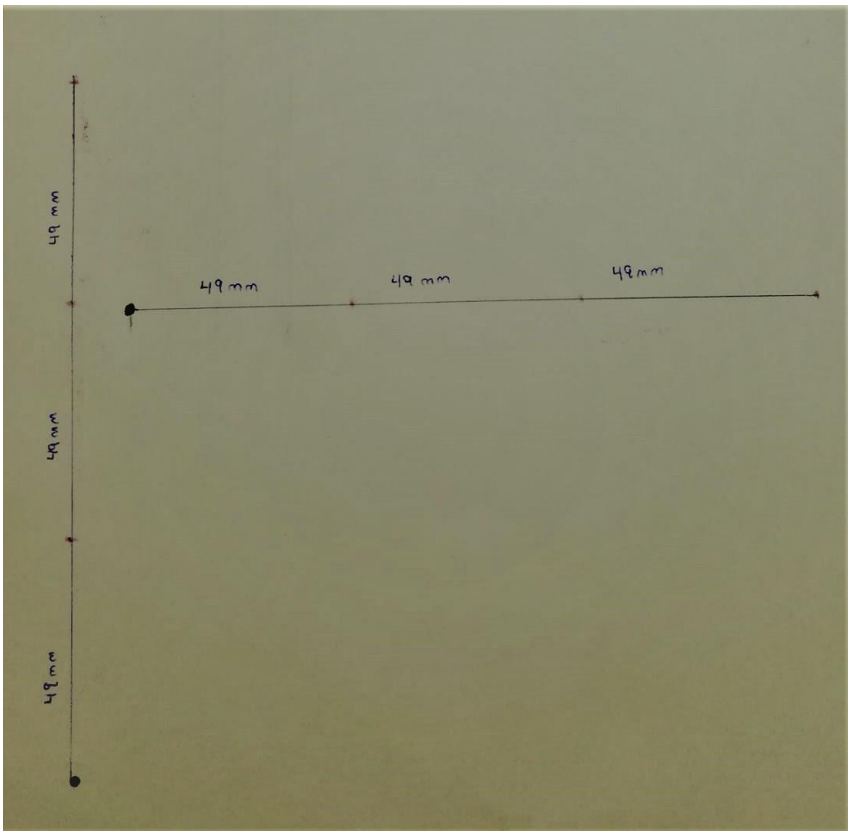


La imagen de la izquierda se corresponde con lo que fue el montaje que demostraría la perpendicularidad de los ejes, antes de empezar el deslazamiento en Y se desmonto la brida acoplada en la fibra de vidrio.

La imagen de la derecha muestra el montaje usado para crear la nube de puntos. El proceso consistía en activar el puntero laser cuando hiciésemos el recorrido programado y marcar con rotulador el punto señalado.

Ilustración 25 Montajes para la calibración

Para el proceso de calibración se empezó por dividir los tornillos sin fin en tres zonas. Para el eje X serian de 16-66, 66-116, 116-166; mientras que para el eje Y estas son 15-65, 65-115, 115-165. Las unidades en milímetros y la referencia de 0 en el final de carrera más cercano a los motores. Cada desplazamiento se ejecutó 10 veces, dando lugar al siguiente resultado (el punto negro es 16 para el eje X y 15 para el eje Y):

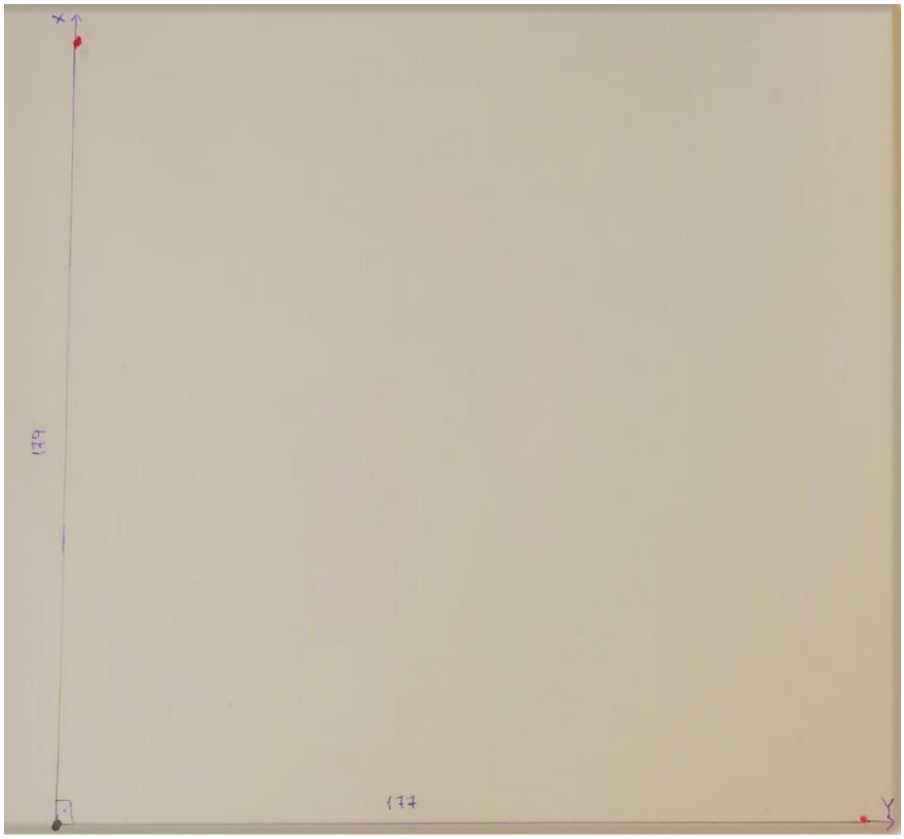


Eje X línea vertical, eje Y línea horizontal. Medidas en milímetros

Ilustración 26 Dispersión inicial

El resultado fue que los puntos de la nube se superponían y que además el desplazamiento de una región a otra era constante. Por un lado, esto nos permite demostrar que el error del avance por paso es constante en todo el recorrido. Ya no tenemos por qué distinguir 3 zonas podemos trabajar solo con una. En cuanto a la desviación, esta es más pequeña de la que podemos determinar con nuestro proceso. Nuestro inconveniente es debido a la inexactitud que tenemos a la hora de marcar los puntos. Hay muchos factores que nos influyen negativamente como el diámetro de la punta del rotulador, el diámetro de la luz del puntero laser o el propio error que introducimos al marcar el punto manualmente. En este punto, decidimos que la desviación o la incertidumbre no podrá ser estimada con los instrumentos que tenemos. La incertidumbre será solo la debida a la discretización del giro del encoder. La cual es igual a medio paso del motor o 0.9º. A pesar de que el encoder redondee los pasos que cuenta este no es capaz de hacer girar al motor menos de un paso. Lo que si calcularemos será el error respecto a la medida aceptada como real. Para ello moveremos el posicionador de un extremo a otro en cada eje. De esta manera el error, al ser proporcional, será más fácil de medir. Haremos 20 desplazamientos en cada uno tomaremos una medida de la posición, otra de los pasos girados por el encoder y otra de los pasos girados por el motor. Usaremos una modificación del programa `limite`. Este no elimina el error acumulado cuando llega al final de carrera. Las medidas son tomadas solo en el avance desde motor a encoder. Se considera que el error en el retroceso es igual que en el avance. Los resultados son los siguientes:

	Pruebas en X		Pruebas en Y	
	MOTOR	ENCODER	MOTOR	ENCODER
	4485	8878	4432	8863
	4484	8879	4432	8864
	4483	8877	4432	8863
	4483	8878	4431	8862
	4487	8879	4433	8864
	4485	8877	4432	8866
	4486	8879	4432	8863
	4486	8879	4431	8863
	4486	8879	4432	8864
	4486	8879	4432	8866
	4486	8877	4433	8865
	4486	8878	4432	8863
	4486	8878	4432	8863
	4485	8878	4433	8865
	4483	8878	4433	8866
	4482	8878	4432	8864
	4482	8877	4432	8863
	4482	8878	4432	8866
Media	4484.61	8878.11	4432.11	8864.05
Desviación típica	1.6852	0.7583	0,5829	1.3048



Eje X línea vertical, eje Y línea horizontal. Medidas en milímetros

Ilustración 27 Medidas de dispersión final

Analizando los resultados, por un lado, si nos fijamos en la tabla el resultado del encoder es siempre el mismo con una desviación muy pequeña. Este resultado es lógico y cabía de esperar debido a que cada paso del motor hace girar 2 al encoder, esos dos pasos de giro hacen que la medida del encoder se desvié la mitad es decir un paso. Por otro lado, los pasos que da el motor son casi siempre los mismos, salvo por una ligera desviación debida a errores puntuales que hacen que el giro del motor varíe ligeramente. Como podemos observar el valor 4486 se repite hasta 7 veces; casi la mitad de las muestras tomadas. Esto nos puede indicar que el ángulo girado por el motor va a estar siempre alrededor de un punto medio el cual tendrá la mayor frecuencia de repeticiones y que si tuviésemos mejores instrumentos de medida seguramente observaríamos que el ángulo por paso del motor sigue una distribución normal. Ya hemos dicho que este estudio no se va a centrar en calcular la incertidumbre, solo intentaremos estimar la corrección. Como hemos considerado que la precisión del giro del encoder es muy superior a la del motor podríamos ser capaces de calcular la corrección de los pasos del motor. De esta manera tendríamos por separado una corrección para el motor y otra para el tornillo sin fin. Sin embargo, para el posicionamiento del eje X e Y no nos afecta el error del motor porque el que fija la posición es el encoder. El encoder avisa al motor de cuando parar. El paso del motor solo nos afecta en la incertidumbre de la posición porque, aunque el encoder cuente 0.9º el motor parara cuando haya recorrido 1.8º. Para la incertidumbre de la posición consideramos que el giro por paso del motor es igual a:

$$\frac{8878.11[pasos\ de\ encoder] * 0.9º}{4484.61[pasos\ de\ motor]} = 1.78º$$

Con los datos de la tabla calculamos el paso del motor como 1.78º, según el fabricante este era 1.8º. Este valor está referido al eje X, para el eje Y el ángulo por paso es de 1.799º

Para el eje X e Y la corrección se calcula por el error asociado al tornillo sin fin o al avance por paso de encoder. Originalmente se había considerado el avance por paso de encoder como 0.02mm y con esta medida se había calculado el extremo del eje X como 182.55 y el extremo del eje Y como 180.27mm. Sin embargo, al hacer el recorrido completo se vio que el eje X media 179mm mientras que el Y media 177mm. Hay un error en el valor del avance por paso que se traducirá en un error en la posición proporcional al desplazamiento. Lo calcularemos con la siguiente formula:

$$\frac{(pulsos * avance\ por\ paso\ teorico) - (pulsos * avance\ por\ paso\ real)}{(pulsos * avance\ por\ paso\ teorico)} =$$

$$\frac{distancia\ teorica - distancia\ real}{distancia\ teorica} = \frac{extremo\ teorico\ del\ eje - extremo\ real}{extremo\ teorico\ del\ eje}$$

De esta manera para el eje X la corrección asociada al desplazamiento es del 1.94% mientras que para él Y es del 1.81%. Recalcamos que es asociada al desplazamiento, si queremos asociarla a la posición global necesitamos conocer primero la posición en la que estamos.

La incertidumbre de posición pasaría por multiplicar medio paso del motor por la corrección del avance por paso, además se puede considerar que seguirá una distribución uniforme. Los resultados del estudio se recogen en la siguiente tabla.

	Corrección	Incertidumbre
Eje X	1.94%	0.017mm
Eje Y	1.84%	0.0176mm

Los resultados son más favorables para el eje Y, ya que se aproximan más al valor teórico. Esto es debido a que es un eje sometido a menos cargas, desviaciones y desajustes.

Con esto daría por terminado el estudio de los ejes X e Y. Para el eje Z nos surge el problema de que no hay realimentación del giro. Además, no podemos medir de manera fiable el giro de este. Por eso extrapolaremos los resultados obtenidos en el motor para el eje X e Y. El peor resultado es en el eje X. El valor del ángulo por paso es de 1.78º, difiere en 0.02º respecto el valor del fabricante. Consideraremos 1.78º el giro por paso del motor en el eje Z. Por tanto, la incertidumbre será la mitad 0.89º, aproximamos esta como una distribución uniforme.

7 Conclusiones

Nos habíamos propuesto como objetivo principal construir un posicionador de coordenadas. Debía permitir controlar el movimiento en el plano X-Y además del giro sobre su propio eje vertical. La resolución del movimiento debía ser inferior a los milímetros. Sobre el prototipo funcional debíamos conseguir una estimación de la corrección e incertidumbre que este tiene. Como factor adicional es deseable que la perturbación magnética sea mínima. El posicionador está destinado para un uso ajeno de este proyecto; él cual consiste en posicionar una espira en el interior de una pinza amperimétrica.

El control de la posición se ha conseguido convirtiendo el giro de un motor paso a paso en un movimiento lineal en cada eje. Mientras que el giro del eje vertical es directamente controlado por un tercer motor paso a paso. El microprocesador usado ha sido el ATmega328p integrado en una placa de Arduino. Por otro lado, el circuito analógico que conecta al Arduino con los motores y sensores se ha diseñado en un formato de shield. De este montaje destacamos el driver para el motor DRV8825 y el driver que selecciona los sensores ULN2003.

El resultado de la estimación del error de posición resultó ser proporcional con el desplazamiento. Alcanzó un valor máximo de 3.5 mm al final del recorrido del eje X. Sin embargo, este error es corregible por programación. Con respecto a la incertidumbre se ha estimado en medio paso del motor. Asociado a que el motor discretiza el giro. El resultado es una desviación inferior a una décima de milímetro.

Estos resultados se pueden considerar satisfactorios respecto a las especificaciones de entrada. Controlamos el movimiento plano con la exactitud adecuada. Esta nos permitirá posicionarnos dentro de la pinza amperimétrica sin problemas. Solo hay una especificación que puede crearnos conflicto y que se ha desestimado por su dificultad constructiva; esta es la referida a no interferir en las lecturas magnéticas de la pinza. Los materiales usados son metálicos, además los motores son inductivos, al igual que los finales de carrera del eje Y, además el final de carrera del eje X requiere de la presencia de un imán para funcionar. La aportación de todos estos aspectos a la distorsión magnética se espera que sea muy pequeña. En caso de constituir un problema se propone como solución aumentar la altura de separación entre la pinza y el posicionador.

8 Bibliografía

[1] C. e. d. metrologia, «Procedimiento EL-007».

[2] T. instruments, «http://www.ti.com/lit/ds/symlink/drv8825.pdf,» Julio 2014. [En línea].

[3] L. Llamas, «https://www.luisllamas.es/motores-paso-paso-arduino-driver-a4988-drv8825/,» 23 Agosto 2016. [En línea].

[4] H. Malón y P. Canalís, «Lección 5: Engranajes de tornillo sin fin y rueda helicoidal,» de *Cálculo y selección de elementos de máquinas*, 2018.

[5] H. Malón y P. Canalís, «Lección 1: Aspectos generales de la transmisión por engranajes,» de *Cálculo y selección de elementos de máquinas*, 2018.

[6] Microchip, «16-bit Timer/Counter1 with PWM,» de *ATmega328 Datasheet*, pp. 89-119.

[7] ST, «https://www.st.com/resource/en/datasheet/uln2001.pdf,» [En línea].

[8] L. Llamas, «https://www.luisllamas.es/cadenas-de-texto-puerto-serie-arduino/,» 27 junio 2017. [En línea].

[9] L. Llamas, «https://www.luisllamas.es/que-son-y-como-usar-interrupciones-en-arduino/,» 28 abril 2016. [En línea].

[10] «https://aprendiendoarduino.wordpress.com/2016/11/08/entradas-y-salidas-arduino/,» [En línea].

[11] diymakers, «http://diymakers.es/mover-motores-paso-paso-con-arduino/,» 28 12 2013. [En línea].

9 Índice de ilustraciones

ANEXOS

Anexo I: Programas

En este anexo vamos a mostrar el código de los programas que hemos creado y utilizado. Destacamos “posicionador_usuario” y ”posicionador_automatizado” los cuales son los programas de posicionamiento principal.

Programa paso_del_tornillo_sin_fin

El objetivo de este programa es calcular el paso del tornillo sin fin.

Es decir, saber cuánto se desplaza la plataforma cada giro del motor; según los fabricantes cada 200 pulsos avanzaremos 8 mm, el procedimiento es el siguiente:

Colocando la plataforma en el extremo del motor para tomar este como referencia, haremos inicialmente 10 giros y calcularemos el paso, seguido a esto el motor se desplazará de 3 en tres giros hasta llegar a 28. Cada 3 giros recalcularemos el paso midiendo la distancia entre la plataforma y motor. Para ordenar al motor que se mueva tres giros deberemos cambiar el estado del pin 7 del Arduino manualmente, esto nos dará tiempo para tomar las medidas de longitud correspondientes con un pie de rey. Cuando se llegue a 28 giros el motor retrocederá 15 giros, así comprobaremos el funcionamiento en sentido contrario. Comprobaremos visualmente la coherencia de los desplazamientos.

```
const int stepPin = 9;
const int dirPin = 4;
const int continuePin = 7;
volatile int i = 10;
volatile int x = 0;
volatile int val = 1;
volatile int v = 0;
void setup() {
  // set the two pins as outputs
  Serial.begin(9600);
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(continuePin, INPUT);
}
void loop() {
  digitalWrite(dirPin, HIGH); //Enables the motor to move in a per-
ticular direction
  // for one full rotation required 200 pulses
  while (i <= 28) {
    while (x < (200 * i)) {
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(500);
      digitalWrite(stepPin, LOW);
      delayMicroseconds(500);
      x = x + 1;
    }
    val = digitalRead(continuePin);
    Serial.println(i);
    if (val == 0) {
      i = i + 3;
    }
    else {
      i = i;
    }
  }
  while (v <= 3000) {
```

```
    digitalWrite(dirPin, LOW); // vuelta atras
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(500);
    v = v + 1;
  }
  val = digitalRead(continuePin);
  if (val == 0) {
    i = 10;
    x = 0;
    val = 1;
    v = 0;
  }
}
```

Programa puesta a cero

En este programa se alternará el movimiento hacia un final de carrera y hacia el otro. Para comprobar si la respuesta deseada coincide con la real.

```
const int dirpin = 9; //variables de direccion junto al pin 10(ti-
mer1) controlan el movimiento del motor
const int fdcpin = 2; //pin del final de carrera
volatile unsigned int Flagfdc, c; //control del final de carrera
void setup() {
  Serial.begin(9600);//Inicializo el puerto serie a 9600 baudios
  Timer1();//Iniciamos el timer del arduino
  pinMode(dirpin, OUTPUT); //Configuramos el pin de dirrección
  noInterrupts(); //Apagamos y encendemos las interrupciones, es
redundante, tiene una final didactica
  interrupts();
  attachInterrupt(digitalPinToInterrupt(2), finaldecarrera, RI-
SING);
}
//Esta funcion fijara una señal PWM a 1000Hz, cada flanco de subida
mueve un paso del motor
void Timer1(void) {
  DDRB &= ~(1 << 2); // selecciono salida del timer y DESACTIVARLA
  TCCR1A = 0b00100001; // fijo la frecuencia 1000Hz
  TCCR1B = 0b00001011;
  OCR1B = 127;
}
//Configuramos la interrupción del final de carrera, establecemos
varios estados
//con switch case, para que en un futuro la señal del final de ca-
rrera se pueda interpretar
//de una forma u otra dependiendo de la situacion
void finaldecarrera(void) {
  switch (Flagfdc) {
    case 1://Función poneracero
      DDRB &= ~(1 << 2); //Puede ser que hayamos llegado a 0, des-
activo salida de pulsos
      c = 0;
      break;
  }
}
// el sistema del final de carrera funciona con una interrupcion
//(la cual alerta sobre la posibilidad de choque) y un muestreo
//(posterior a la interrupción) que decide si la interrupción es
//valida o ha sido un error, ademas de un muestreo para controlar
//que el sistema funciona correctamente
```

```
void poneracero(void) { //lleva al sitema hacia el motor
  int muestreo = 1; // muestreo de control activo
  int fdc; // lectura del pin del final de carrera
  fdc = digitalRead(fdcpin);
  if (fdc == 1) {
    muestreo = 0; // hemos llegado a 0 salimos de la función
  }
  else {}
  c = 1; //fijamos variable de control de la interrupcion
  digitalWrite(dirpin, HIGH); // Sentido del motor hacia atras
  Flagfdc = 1; //fijamos funcion de interrupción
  while (muestreo == 1) {
    DDRB |= (1 << 2); // Activamos la salida de pulsos en el pin 10
    Serial.println("abro la puerta");//nos avisa de que el movi-
miento ha empezado
    while (c != 0) {}; //es un bucle de control ademas una inte-
rrupcion  en el pin 2 desactivara la variable c poniendola a 0
    delay(2000);// espero dos segundos, para asegurarnos se que he-
mos llegado
    fdc = digitalRead(fdcpin);
    if (fdc == 1) {
      muestreo = 0; // hemos llegado a 0 salimos de la función
    }
    else {
      c = 1; //volvemos a fijar la variable de control de la inte-
rrupcion
    }
  }
  Flagfdc = 0; // Estado normal de la interrupcion del final de ca-
rrera(pin 2)
}
// el sistema del final de carrera funciona con una interrupcion
//(la cual alerta sobre la posibilidad de choque) y un muestreo
//(posterior a la interrupción) que decide si la interrupción es
//valida o ha sido un error, ademas de un muestreo para controlar
//que el sistema funciona correctamente
void poneracero1(void) { //lleva al sitema hacia el extremo contra-
rio del motor
  volatile int muestreo = 1; // muestreo de control activo
  int fdc; // pin del final de carrera
  fdc = digitalRead(fdcpin);
  if (fdc == 1) {
    digitalWrite(dirpin, LOW); // Sentido del motor hacia adelante
    DDRB |= (1 << 2); // Activamos la salida de pulsos en el pin 10
para desplazar el cero
    delay(800);// dejamos los pulsos abiertos 0.8 segundos que son
aproximadamente 800 pulsos, aprox 4 vueltas
    DDRB &= ~(1 << 2);
  }
  else {}
  c = 1; //fijamos variable de control de la interrupcion
  digitalWrite(dirpin, HIGH); // Sentido del motor hacia atras
  Flagfdc = 1; //fijamos funcion de interrupción
  Serial.println("Valor de muestreo"); Serial.println(muestreo);
  while (muestreo == 1) {
    DDRB |= (1 << 2); // Activamos la salida de pulsos en el pin 10
    Serial.println("abro la puerta");
    while (c != 0) {}; //es un bucle de control ademas una inte-
rrupcion  en el pin 2 desactivara la variable c poniendola a 0
```

```
    delay(2000);// espero dos segundos
    fdc = digitalRead(fdcpin);
    if (fdc == 1) {
      muestreo = 0; // hemos llegado a 0 salimos de la funcion
    }
    else {
      c = 1; //volvemos a fijar la variable de control de la inte-
rrupcion
    }
  }
  Flagfdc = 0; // Estado normal de la interrupcion del final de ca-
rrera(pin 2)
}
void loop() {//Alternamos la funcion para probar el movimiento ha-
cia un lado o hacia otro
  poneracero1();
}
```

Programa limite_por_motor

Este programa nos sirve para calcular cuantos pasos de motor hay de un extremo a otro en un eje. Nos sirve para obtener el valor del recorrido máximo para cada eje.

```
#define EN 6 // 7 para mover en X 6 para mover en Y
volatile byte mover;//permanece a 1 mientras el bucle de movimiento
este activo
volatile int conteo;//cuenta pulsos en el encoder
volatile int T = 0;//activaciones del final de carrera
void vaciar_buffer_entrada(void);//vacia el buffer de lectura
void esperar_dato(void);//espera dato por el puerto serie
void finaldecarrera(void);//interrupcion del final de carrera
void setup() {
  Serial.begin(9600);
  pinMode(6, OUTPUT); //ENY
  pinMode(7, OUTPUT); //ENX
  pinMode(10, OUTPUT); //DIR
  pinMode(9, OUTPUT); //STEP
  pinMode(8, OUTPUT); //DIR NEGADA
  pinMode(3, INPUT); //ENCODER ISR
  pinMode(4, INPUT_PULLUP); //ENCODER X CANAL 2
  pinMode(5, INPUT_PULLUP); //ENCODER Y CANAL 2
  pinMode(2, INPUT); //FINAL DE CARRERA
  pinMode(12, OUTPUT); //enable z
  digitalWrite(12, HIGH); //DESACTIVO Z
  digitalWrite(6, HIGH); //DESACTIVO X
  digitalWrite(7, HIGH); //DESACTIVO Y
  Serial.begin(9600);//inicio comunicación serie a 9600 baudios
  attachInterrupt(digitalPinToInterrupt(2), finaldecarrera, RI-
SING); //activo ISR final de carrera
}
void loop() {
  char a;//variable de lectura
  mover = 0;//bucle de movimiento
  esperar_dato();//esperamos ordenes por puerto serie
  a = Serial.read();//esta letra sirve para iniciar el movimiento,
como un interruptor
  delay(1000);
  digitalWrite(10, LOW); //direccion, HIGH HACIA ATRAS, LOW ADELANTE
  digitalWrite(8, HIGH);
  if (a == 'M') {//habilita pasos y los hace
    conteo = 0;  //inicia la variable que cuenta pasos del encoder
    mover = 1; //activa el control del bucle de movimiento
```

```
    delay(10);
    Serial.print(F("HAY ALGO "));//sirve para el debug
    Serial.println(T);//activaciones del final de carrera, sirve para
debug
    digitalWrite(EN, LOW);//activamos el movimiento
  }
  while (mover == 1) {//manda pulsos a 800  Hz aproximadamente
    digitalWrite(9, LOW);
    delayMicroseconds(600);
    digitalWrite(9, HIGH);
    delayMicroseconds(600);
    conteo++;
  }
  Serial.print(F("\r\nHa hecho: "));//informamos del resultado del mo-
vimiento
  Serial.print(conteo);
  Serial.println(F("pasos de motor\r\n"));
  Serial.print(F("entradas en final de carrera"));
  Serial.println(T);
  vaciar_buffer_entrada();
}
void esperar_dato(void) {//espera dato por puerto serie
  while (Serial.available() == 0) {}
  delay(70);
}
void vaciar_buffer_entrada(void) {//vacia el buffer de lectura
  while (Serial.available() != 0) {
    Serial.read();
  }
}
void finaldecarrera(void) { //funcion del final de carrera
  T++;
  digitalWrite(EN, HIGH); //para el movimiento
  mover = 0;//informa al arduino que el movimiento ha terminado
}
```

Programa limite

Este programa nos sirve para conocer la relación entre pasos de encoder y pasos de motor, para ello necesitamos saber los pasos de motor que se dieron en el programa anterior. Ambos programas son parecidos. El anterior cuenta los pasos de motor hasta un extremo y este contara, en lugar de pasos de motor, pasos de encoder.

```
#define EN  6// 7 para activar el eje X, 6 para el Y
volatile byte mover;//permanece a 1 si hay o va a haber movimiento
volatile int conteo;//cuenta pulsos del encoder
volatile int T = 0;//alamcena las activaciones del final de carrera
void Timer1(void);//funcion del timer1
void vaciar_buffer_entrada(void);//Vacia el buffer de lectura
void esperar_dato(void);//Permanece a la espera de un dato en
puerto serie
void encoder(void);// funcion de interrupción asociada al encoder
void finaldecarrera(void);// funcion de interrupción asociada al
final de carrera
void setup() {
  Serial.begin(9600);
  pinMode(6, OUTPUT); //ENY
  pinMode(7, OUTPUT); //ENX
  pinMode(10, OUTPUT); //DIR
  pinMode(9, OUTPUT); //STEP
  pinMode(8, OUTPUT); //DIR NEGADA
```

```
  pinMode(3, INPUT); //ENCODER ISR
  pinMode(4, INPUT_PULLUP); //ENCODER X canal 2
  pinMode(5, INPUT_PULLUP); //ENCODER Y canal 2
  pinMode(2, INPUT); //FINAL DE CARRERA
  pinMode(12, OUTPUT); //enable z
  digitalWrite(12, HIGH); //desactivo z
  digitalWrite(6, HIGH); //desactivo x
  digitalWrite(7, HIGH); //desactivo y
  Serial.begin(9600);//inicio la comunicación a 9600 baudios
  attachInterrupt(digitalPinToInterrupt(3), encoder, RISING); //ac-
tivo ISR encoder
  attachInterrupt(digitalPinToInterrupt(2), finaldecarrera, RI-
SING); //activo ISR final de carrera
  Timer1();// configuro el timer1
}
void loop() {
  char a;//variable de lectura
  esperar_dato();//esperamos un dato
  a = Serial.read(); //Esta letra solo sirve para iniciar el movi-
miento como un interruptor
  delay(1000);
  //MOVIMIENTO HACIA ENCODER
  digitalWrite(10, LOW); //direccion, HIGH HACIA ATRAS, LOW ADE-
LANTE
  digitalWrite(8, HIGH);
  //MOVIMIENTO HACIA MOTOR
  //digitalWrite(10,HIGH);//direccion, HIGH HACIA ATRAS, LOW ADE-
LANTE
  //digitalWrite(8,LOW);
  if (a == 'M') {//habilita pasos y los hace
    conteo = 0;//inicia la variable que cuenta pulsos del encoder
    DDRB |= (1 << 1);//activa la salida de pulsos
    mover = 1;//activa el bucle de control del movimiento
    delay(10);
    Serial.print(F("HAY ALGO "));//nos sirve para el debugg
    Serial.println(T);// entradas al final de carrera
    digitalWrite(EN, LOW);// se inicia el movimiento
  }
  while (mover == 1) {}//bucle vacio, activado mientras el sistema
se este moviendo
  Serial.print(F("\r\nHa hecho: ")); //informa de los resultados
del movimiento
  Serial.print(conteo); //n° pasos de encoder
  Serial.println(F("pasos de encoder\r\n"));
  Serial.print(F("entradas en final de carrera"));
  Serial.println(T);//n° de eentradas final de carrera, debe ser 1
  vaciar_buffer_entrada();
}
void esperar_dato(void) {//Permanece a la espera de un dato en
puerto serie
  while (Serial.available() == 0) {}
  delay(70);
}
void vaciar_buffer_entrada(void) {//Vacia el buffer de lectura
  while (Serial.available() != 0) {
    Serial.read();
  }
}
void encoder(void) { // la funcion del encoder debe actualizar con-
teo
  conteo = conteo + 1;
```

```
}
void finaldecarrera(void) { // bloquea el movimiento
  T++;
  digitalWrite(EN, HIGH); //para el movimiento
  DDRB &= ~(1 << 1); // selecciono salida del timer y DESACTIVARLA
  mover = 0;// comunica al arduino que el movimiento ha acabado
}
void Timer1(void) {// funcion que configura el timer1 elcual ac-
tuara como una salida de pulsos constante a 1000 HZ
  DDRB &= ~(1 << 1); // selecciono salida del timer y DESACTIVARLA
  TCCR1A = 0b10000001; // fijo la frecuencia 1000Hz
  TCCR1B = 0b00001011;
  OCR1A = 127;
}
```

Programa posicionador_usuario

Este es el programa final de posicionamiento. Destinado para un uso interactivo. Este programa y el del siguiente punto son bastante similares en cuanto a funcionamiento.

```
#define avance 0.02035160958//avance por pulso del encoder
#define max_Y 177.0// limite en Y
#define max_X 211.0// limite en X
#define dirpin 10 //  pin de dirección
#define Ndirpin 8//   pin de dirección negada
#define ENX 7//       pin de Enable del eje X
#define ENY 6//       pin de Enable del eje Y
volatile char orden_mov[19] {};//vector que guarda el orden del
movimiento en los valores del 1 al 18
volatile float posicion_deseada[19] {};// guarda las posiciones
correspondientes al vector anterior
volatile char orden_desp[19] {};//vector que guarda el orden del
desplazamiento en los valores del 1 al 18
volatile float desplazamiento_deseado[19] {};// guarda los
desplazamientos correspondientes al vector anterior
byte Flagfdc;//esta variable permitiria escoger una funcion de
trabajo dentro del final de carrera
byte state;//almacena el tipo de movimiento que se va a efectuar
antes de que este se produzca
volatile int pasos;// almacena los pasos de encoder que debemos
hacer
volatile byte control_bucle;//bucle de control asociado al muestreo
del movimiento
volatile int conteo;//cuenta los pulsos recibidos por el encoder
volatile byte DIR;//el encoder fija su valor en función del sentido
de giro
byte testigo;// variable asociada al resultado del movimiento
byte CH1;// puede ser 5 o 4 en función de que encoder se este
usando,nos permite leer
//el canal del encoder que no funciona con interrupciones
float Y, Y_ini, X, X_ini;//Y,X son los valores absolutos de la
posición,
//X_ini,Y_ini son los valores absolutos antes del movimiento
void(*resetfunc)(void) = 0;//sirve para reiniciar el sketch NO el
arduino
void Timer1(void);//configura el timer 1 en el pin 10
void esperar_dato(void);//mantiene el micro en pausa a la espera de
un dato en puerto serie
void vaciar_buffer_entrada(void);//vacia el buffer de lectura
void movimiento_Y_0(void);//Programa de gestión de movimiento, nos
mueve a Y 0
```

```
void movimiento_Y_max(void);//Programa de gestión de movimiento,
nos mueve a Y max
void movimiento_X_0(void);//Programa de gestión de movimiento, nos
mueve a X 0
void movimiento_X_max(void);//Programa de gestión de movimiento,
nos mueve a X max
void RUTINAB(void);//submenu para el movimiento global
void RUTINAC(void);//submenu para el movimiento realativo
void DESPLAZAMIENTO(byte i1);//Programa de gestión de movimiento,
para desplazamientos
void MOVIMIENTO_GLOBAL(byte i);//Programa de gestión de movimiento,
para posición global
void tipodemovimiento(void);//identifica en "state" el tipo de
movimiento
void movereje(void);//mueve el eje fijado tantos pulsos como pasos
void encoder(void);//asociado a las interrupciones del encoder
void finaldecarrera(void);//asociado a las interrupciones del final
de carrera
void setup() {
  X = 0.0;//Iniciamos en 0 absoluto
  Y = 0.0;
  //FIJAMOS ENTRADAS Y SALIDAS, ACTIVAMOS INTERRUPCIONES Y
CONFIGURAMOS EL TIMER
  pinMode(ENX, OUTPUT);
  pinMode(ENY, OUTPUT);
  pinMode(dirpin, OUTPUT);
  pinMode(Ndirpin, OUTPUT);
  pinMode(2, INPUT); //FDC
  pinMode(3, INPUT); //ENCODER
  pinMode(4, INPUT_PULLUP); //ch1 encoder x
  pinMode(5, INPUT_PULLUP); //ch1 encoder y
  pinMode(9, OUTPUT); //STEP
  pinMode(11, OUTPUT); //DIRZ
  pinMode(12, OUTPUT); //ENZ
  pinMode(13, OUTPUT); //STEPZ
  digitalWrite(12, HIGH); //desactivo en z
  digitalWrite(ENX, HIGH); //desaactivo en x
  digitalWrite(ENY, HIGH); //desactivo en y
  attachInterrupt(digitalPinToInterrupt(3), encoder,
RISING); //activo ISR encoder
  attachInterrupt(digitalPinToInterrupt(2), finaldecarrera,
RISING); //activo ISR final de carrera
  Timer1();
  //TEXTOS DE BIENVENIDA
  Serial.begin(9600);
  Serial.println(F("\r \nBienvenido a su posicionador de
coordenadas de confianza"));
  Serial.println(F("las letras y numeros deben escribirse todo
junto. "));
  Serial.println(F("Por ejemplo: X50Y20Z2X20; los movimientos se
hacen segun"));
  Serial.println(F("el orden en que se introducieron las
letras."));
  Serial.println(F("Respetar los limites de los ejes"));
  Serial.flush();
  Serial.print(F("X_max=")); Serial.print(max_X); Serial.print(F("
Y_max=")); Serial.println(max_Y);
  Serial.println(F("Al iniciarse el posicionador toma su posicion
actual "));
```

```
  Serial.println(F("como 0, se recomienda una calibracion a 0 o a
un maximo.\r \n"));
  Serial.flush();
}
void loop() {
  char funcion;//asociado a un parametro del menú
  bool submenu;
  Serial.println(F("\r \n¿Que desea hacer?"));
  Serial.println(F("[A]Posicion actual"));
  Serial.println(F("[B]Movimiento, coordenadas globales"));
  Serial.println(F("[C]Movimiento, coordenadas relativas"));
  Serial.flush();
  Serial.println(F("[D]Calibrar Y a 0"));
  Serial.println(F("[E]Calibrar Y a Y_max"));
  Serial.println(F("[F]Calibrar X a 0"));
  Serial.println(F("[G]Calibrar X a X_max\r \n"));
  Serial.flush();
  vaciar_buffer_entrada();
  esperar_dato();
  funcion = Serial.read();
  switch (funcion) {
    default:
      Serial.println(F("No le he entendido, vuelva a introducir un
comando valido A-G"));
      break;
    case 'D'://Hace un movimiento a 0 Y
      //Serial.println(F("He entrado en D"));
      movimiento_Y_0();
      break;
    case 'E':// Hace un movimiento a Y maxima
      // Serial.println(F("He entrado en E"));
      movimiento_Y_max();
      break;
    case 'F'://Hace un movimiento a 0 X
      // Serial.println(F("He entrado en F"));
      movimiento_X_0();
      break;
    case 'G':// Hace un movimiento a X maxima
      // Serial.println(F("He entrado en G"));
      movimiento_X_max();
      break;
    case 'A'://INDICA LA POSICION GLOBAL
      //Serial.println(F("He entrado en A"));
      Serial.print(F("Posicion x: ")); Serial.println(X);
      Serial.print(F("Posicion y: ")); Serial.println(Y);
      break;
    //MOVIMIENTOS ESPECIFICOS
    //Los movimientos especificos se han programado como un
procedimiento, originalmente se intento
    //escribir el codigo dentro del case, pero se vio empiricamente
que esto daba lugar a error,
    //concretamente el error consistia en que los "case" que
hubiesen debajo del codigo de  RUTINAB O RUTINAC
    //(si solo ponemos el codigo, al usar el procedimiento no hay
problema)eran ignorados.
    //Desconocemos el motivo, pero podemos creer que tiene que ver
con la memoria. Realizamos muchas pruebas,
    //sin encontrar un problema en el codigo, entre las pruebas
realizadas estaba sustituir RUTINAB() O RUINAC()
```

```
    //por su codigo entre corchetes("{}", los corchetes hacen al
arduino interpretar las ordenes como una funcion,
    //y se vio que con los corchetes,la comunicacion funcionaba,
pero sin corchetes dejaba de funcionar correctamente.
    case 'B'://INICIA LA COMUNICACIÓN PARA UN MOVIMIENTO GLOBAL
      RUTINAB();
      break;
    //Siguiente tipo de movimiento
    case 'C' ://INICIA LA COMUNICACIÓN PARA UN MOVIMIENTO RELATIVO
      RUTINAC();
      break;
  }
}
//
//
void Timer1(void) {// funcion que configura el timer1 el cual
actuara como una salida de pulsos constante a 1000 HZ
  DDRB &= ~(1 << 1); // selecciono salida del timer y DESACTIVARLA
  TCCR1A = 0b10000001; // fijo la frecuencia 1000Hz
  TCCR1B = 0b00001011;
  OCR1A = 127;
}
//FUNICONES BASICAS PARA LA COMUNICACION SERIE
//
//
void esperar_dato(void) {//espera dato por puerto serie
  while (Serial.available() == 0) {}
  delay(70);
}
void vaciar_buffer_entrada(void) {//vacia el buffer de lectura
  while (Serial.available() != 0) {
    Serial.read();
  }
}
//
//
//PROGRAMAS PARA LA COMUNICACION SERIE EN MOVIMIENTOS RELATIVOS Y
GLOBALES
//
//
void RUTINAB(void) { //Programa que realiza la comunicación para
realizar un movimiento global
  //Serial.println(F("He entrado en B"));
  bool submenuB = true;
  while (submenuB == true) {//nos obliga a permanecer en el
subemnu, en caso contrario el bucle
    //acaba y volvemos al menu principal
    byte p = 0;//almacenara el nº total de movimientos a efectuar,
hasta 18
    volatile char letra, data;
    Serial.println(F("\r \nIntroduzca coordenadas globales en
mayuscula, todo junto"));
    Serial.println(F("y en el orden que desea hacer, se admiten
hasta 15 posiciones"));
    Serial.println(F("unidades en milimetros y grados"));
    Serial.flush();
    Serial.println(F("si desea volver al menu principal introduzca
[V](volver)"));
```

```
    Serial.println(F("IMPORTANTE, EL DECIMAL ES CON PUNTO ¡NO
COMA!"));
    Serial.println(F("Por
ejemplo:X50.56Y20.33Z2.3Y20.34X20.4\r \n"));
    Serial.flush();
    //iniciamos los vectores con la información del movimiento
    orden_mov[19] = {};
    posicion_deseada[19] = {};
    vaciar_buffer_entrada();
    esperar_dato();
    while ((Serial.available()) && (p < 18)) { //LEE HASTA 18
POSICIONES INTERCALADAS DE X,Y,Z,
      //la letra V nos llevara de vuelta al menu principal,
reinicia el loop
      letra = Serial.read();
      switch (letra) {
        case 'X':
          p++;//Si p es igual a 0 es que ha saltado el default o la
letra V , p indica el orden de desplazamientos
          orden_mov[p] = 'X';
          posicion_deseada[p] = Serial.parseFloat();
          break;
        case 'Y':
          p++;
          orden_mov[p] = 'Y';
          posicion_deseada[p] = Serial.parseFloat();
          break;
        case 'Z':
          p++;
          orden_mov[p] = 'Z';
          posicion_deseada[p] = Serial.parseFloat();
          break;
        case 'V'://Reinicial el loop
          vaciar_buffer_entrada();//nos hara salir del while de
lectura
          submenuB = false;//salimos del submenu
          Serial.println(F("Volvemos al menu principal"));
          break;
        default:
          vaciar_buffer_entrada();//nos hara salir del while de
lectura
          if ((letra != '\n') && (letra != '\r')) {//condicion para
error de lectura
            p = 0;
            Serial.println(F("Letra desconocida no coincide con
X,Y,Z o V\r \n"));
          }
          break;
      }
    }
    vaciar_buffer_entrada();
    if ((p != 0) && (letra != 'V')) { // Si se han introducido
datos entramos aqui, donde
      //tendremos que confirmar las posiciones introducidas, es un
control de seguridad ante errores de lectura
      Serial.println(F("El movimiento sera en el orden: "));
      for (byte b = 1; b <= p ; b++) {//muestra por puerto serie el
orden de los movimientos y su valor
```

```
        Serial.print(orden_mov[b]); Serial.print(F(" =
")); Serial.print(posicion_deseada[b]);
        if ((orden_mov[b] == 'X') || (orden_mov[b] == 'Y')) {
          Serial.println(" mm");
        }
        else {
          Serial.println(" °");
        }
        Serial.flush();
      }
      Serial.flush();
      Serial.println(F("¿Desea continuar? [S]SI,[N]NO"));
      while (1) { // Esperamos confirmacion para hacer el
movimiento
        vaciar_buffer_entrada();
        esperar_dato();
        data = Serial.read();
        if (data == 'S') {
          MOVIMIENTO_GLOBAL(p);// se introduce el n° de movimientos
a efectuar
          break;
        }
        else if (data == 'N') {
          Serial.println(F("Movimiento cancelado, vuelva a
introducir datos"));
          break;
        }
        else {
          Serial.println(F("No he entendido, introduzca [S]SI,[N]NO
para realizar el movimiento ")); Serial.flush();
        }
      }
    }
    Serial.flush();
  }
}
//
//
void RUTINAC(void) { //INICIA UN MOVIMIENTO RELATIVO
  //Serial.println(F("\r \nHe entrado en C"));
  bool submenuC = true;
  while (submenuC == true) {
    byte p1 = 0;
    volatile char letra1, data1;
    Serial.println(F("\r \nIntroduzca coordenadas relativas en
mayuscula, todo junto"));
    Serial.println(F("y en el orden que desea hacer, se admiten
hasta 15 desplazamientos"));
    Serial.println(F("unidades en milimetros y grados"));
    Serial.flush();
    Serial.println(F("si se alcanza un final de carrera el
movimiento no se recalcula"));
    Serial.println(F("si desea volver al menu principal introduzca
[V](volver)"));
    Serial.println(F("IMPORTANTE, EL DECIMAL ES CON PUNTO ¡NO
COMA!"));
    Serial.println(F("Por
ejemplo:X50.56Y20.33Z2.3Y20.34X20.4\r \n"));
    Serial.flush();
```

```
    //iniciamos los vectores que almacenan las posiciones
    orden_desp[19] = {};
    desplazamiento_deseado[19] = {};
    vaciar_buffer_entrada();
    esperar_dato();
    while ((Serial.available()) && (p1 < 18)) { //LEE HASTA 15
POSICIONES INTERCALADAS DE X,Y,Z,
      //la letra V nos llevara de vuelta al menu principal,
reinicia el loop
      letra1 = Serial.read();
      switch (letra1) {
        case 'X':
          p1++;//Si p1 es igual a 0 es que ha saltado el default o
la letra V, p indica el orden de desplazamientos
          orden_desp[p1] = 'X';
          desplazamiento_deseado[p1] = Serial.parseFloat();
          break;
        case 'Y':
          p1++;
          orden_desp[p1] = 'Y';
          desplazamiento_deseado[p1] = Serial.parseFloat();
          break;
        case 'Z':
          p1++;
          orden_desp[p1] = 'Z';
          desplazamiento_deseado[p1] = Serial.parseFloat();
          break;
        case 'V'://Reinicial el loop
          vaciar_buffer_entrada();//Nos saca del bucle while
          submenuC = false;
          Serial.println(F("Volvemos al menu principal"));
          break;
        default:
          vaciar_buffer_entrada();//Nos saca del bucle while
          if ((letra1 != '\n') && (letra1 != '\r')) {
            p1 = 0;
            Serial.println(F("Letra desconocida no coincide con
X,Y,Z o V\r \n"));
          }
          break;
      }
    }
    if ((p1 != 0) && (letra1 != 'V')) { // Si se han introducido
datos entramos aqui, donde
      //tendremos que confirmar las posiciones introducidas, es un
control de seguridad ante errores de lectura
      Serial.println(F("El movimiento sera en el orden: "));
      for (byte b1 = 1; b1 <= p1 ; b1++) {//muestra por puerto
serie el orden de los movimientos y su valor
        Serial.print(orden_desp[b1]); Serial.print(F(" =
")); Serial.print(desplazamiento_deseado[b1]);
        if ((orden_desp[b1] == 'X') || (orden_desp[b1] == 'Y')) {
          Serial.println(" mm");
        }
        else {
          Serial.println(" °");
        }
        Serial.flush();
      }
```

```
      Serial.println(F("¿Desea continuar? [S]SI,[N]NO"));
      while (1) { // Esperamos confirmacion para hacer el
movimiento
        vaciar_buffer_entrada();
        esperar_dato();
        data1 = Serial.read();
        if (data1 == 'S') {
          DESPLAZAMIENTO(p1);//introducimos el total de movimientos
a efectuar
          break;
        }
        else if (data1 == 'N') {
          Serial.println(F("Movimiento cancelado, vuelva a
introducir datos"));
          break;
        }
        else {
          Serial.println(F("No he entendido, introduzca [S]SI,[N]NO
para realizar el movimiento ")); Serial.flush();
        }
      }
    }
    Serial.flush();
  }
}
//
//
//PROGRAMAS PARA EJECUTAR LOS MOVIMIENTOS DE MANERA CONTROLADA
//
//
void movimiento_Y_0(void) {
  //Configuramos el movimiento
  digitalWrite(ENY, LOW);
  digitalWrite(dirpin, HIGH);
  digitalWrite(Ndirpin, LOW);
  if (digitalRead(2)) {//Si ya estamos en el limite salimos de el
    digitalWrite(dirpin, LOW);
    DDRB |= (1 << 1);
    delay(800);
    DDRB &= ~(1 << 1);
    digitalWrite(dirpin, HIGH);
    delay(300);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
  }
  pasos = -30000;
  movereje();//nos movemos -30000 pasos, algo imposible de alcanzar
  if (testigo == 3) {
    Serial.println(F("\r\nHemos alcanzado Y 0, comprobar
visualmente si coincide\n\r"));
  }
  else {
    Serial.print(F("\r\nEl movimiento no se ha completado, codigo
de error: "));
    Serial.println(testigo);
  }
}
void movimiento_Y_max(void) {
  //Configuramos el movimiento
  digitalWrite(ENY, LOW);
```

```
  digitalWrite(dirpin, LOW);
  digitalWrite(Ndirpin, HIGH);
  if (digitalRead(2)) {//Si ya estamos en el limite salimos de el
    digitalWrite(dirpin, HIGH);
    DDRB |= (1 << 1);
    delay(800);
    DDRB &= ~(1 << 1);
    digitalWrite(dirpin, LOW);
    delay(300);//Evitar movimientos bruscos entre parada y
movimiento en direccion contraria
  }
  pasos = 30000;
  movereje();//Nos movemos 30000 pasos algo imposible de alcanzar
  if (testigo == 3) {
    Serial.println(F("\r\nHemos alcanzado Y maxima, comprobar
visualmente si coincide\n\r"));
  }
  else {
    Serial.print(F("\r\nEl movimiento no se ha completado, codigo
de error: "));
    Serial.println(testigo);
  }
}
void movimiento_X_0(void) {
  //Configuramos el movimiento
  digitalWrite(ENX, LOW);
  digitalWrite(dirpin, HIGH);
  digitalWrite(Ndirpin, LOW);
  if (digitalRead(2)) {//Si ya estamos en el limite nos alejamos de
el
    digitalWrite(dirpin, LOW);
    DDRB |= (1 << 1);
    delay(800);
    DDRB &= ~(1 << 1);
    digitalWrite(dirpin, HIGH);
    delay(300);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
  }
  pasos = -30000;//Nos movemos -30000 pasos algo imposible de
alcanzar
  movereje();
  if (testigo == 3) {
    Serial.println(F("\r\nHemos alcanzado X 0, comprobar
visualmente si coincide\n\r"));
  }
  else {
    Serial.print(F("\r\nEl movimiento no se ha completado, codigo
de error: "));
    Serial.println(testigo);
  }
}
void movimiento_X_max(void) {
  //Configuramos el movimiento
  digitalWrite(ENX, LOW);
  digitalWrite(dirpin, LOW);
  digitalWrite(Ndirpin, HIGH);
  if (digitalRead(2)) {//Si ya estamos en el limite nos alejamos de
el
    digitalWrite(dirpin, HIGH);
```

```
    DDRB |= (1 << 1);
    delay(800);
    DDRB &= ~(1 << 1);
    digitalWrite(dirpin, LOW);
    delay(300);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
  }
  pasos = 30000;//Nos movemos 30000 pasos algo imposible de
alcanzar
  movereje();
  if (testigo == 3) {
    Serial.println(F("\r\nHemos alcanzado X maxima, comprobar
visualmente si coincide\n\r"));
  }
  else {
    Serial.print(F("\r\nEl movimiento no se ha completado, codigo
de error: "));
    Serial.println(testigo);
  }
}
void DESPLAZAMIENTO(byte i1) {//Este programa ejecuta los
desplazamientos en orden
  //tambien verifica si estos son correctos
  for (byte r1 = 1; r1 <= i1; r1++) {//Hace los desplazamientos en
el orden pedido
    switch (orden_desp[r1]) {
      case 'X':
        pasos = round((desplazamiento_deseado[r1] * 49.13616274));/
/transforma los mm en pasos
        digitalWrite(ENX, LOW);
        movereje();
        //Solo nos sirve el "testigo 4"
        if (testigo == 3) {
          Serial.println(F("\r\nHA SALTADO UN FINAL DE CARRERA SE
ABORTA MOVIMIENTO\r\n"));
          i1 = 0;
        }
        else if (testigo != 4) {
          Serial.print(F("\r\nFALLO, SE ABORTA MOVIMIENTO, "));
          Serial.print(F("codigo de error: "));
          Serial.println(testigo);
          i1 = 0;
        }
        else {
          Serial.print(F("Movimiento: "));  //EL MOVIMIENTO ES
CORRECTO, SE PODRIA INTRODUCIR UNA ESPERA AQUI
          Serial.print(r1);
          Serial.println(F(" completado, paso al siguiente\r\n"));
        }
        break;
      case 'Y':
        pasos = round((desplazamiento_deseado[r1] * 49.13616274));/
/transforma los mm en pasos
        digitalWrite(ENY, LOW);
        movereje();
        //Solo nos sirve el "testigo 4"
        if (testigo == 3) {
          Serial.println(F("\r\nHA SALTADO UN FINAL DE CARRERA SE
ABORTA MOVIMIENTO\r\n"));
```

```
          i1 = 0;
        }
        else if (testigo != 4) {
          Serial.print(F("\r\nFALLO, SE ABORTA MOVIMIENTO, "));
          Serial.print(F("codigo de error: "));
          Serial.println(testigo);
          i1 = 0;
        }
        else {
          Serial.print(F("Movimiento: "));  //EL MOVIMIENTO ES
CORRECTO, SE PODRIA INTRODUCIR UNA ESPERA AQUI
          Serial.print(r1);
          Serial.println(F(" completado,paso al siguiente\r\n"));
        }
        break;
      case 'Z':
        pasos = round((desplazamiento_deseado[r1] * 0.555555555));/
/Transforma el angulo en pasos
        if (pasos > 0) {
          digitalWrite(11, LOW);
          digitalWrite(12, LOW);
        }
        else {
          digitalWrite(11, HIGH);
          digitalWrite(12, LOW);
        }
        for (int q1 = 1; q1 <= abs(pasos); q1++) {
          digitalWrite(13, LOW);
          delayMicroseconds(600);
          digitalWrite(13, HIGH);
          delayMicroseconds(600);
        }
        digitalWrite(12, HIGH);
        testigo = 4;//No hay realimentación, por lo que se
considera que el movimiento se realizo correctamente
        Serial.print(F("Movimiento: "));
        Serial.print(r1);
        Serial.println(F(" completado, paso al siguiente\r\n"));
        break;
    }
    Serial.flush();//Vaciamos el buffer de escritura
  }
  //Cuando acabamos toda la secuencia:
  if (testigo == 4) {
    Serial.println(F("\r\nHe acabado el movimiento relativo
programado \r\n"));
  }
}
void MOVIMIENTO_GLOBAL(byte i) {//Este programa ejecuta los
desplazamientos en orden
  //tambien verifica si estos son correctos
  for (byte r = 1; r <= i; r++) {
    //Hacemos la secuencia de movimimientos, verificamos que esten
dentro de los limites,
    //La activación de un final de carrera repite el movimiento,
los testigos diferentes a
    //3 y 4 abortan el programa. Igual que antes el movimiento en Z
se considera correcto y
    //este es relativo
```

```
    switch (orden_mov[r]) {
      case 'X':
        if ((posicion_deseada[r] >= 0.0) && (posicion_deseada[r] <=
 max_X)) {
          pasos = round(((posicion_deseada[r] - X) * 49.13616274));
          digitalWrite(ENX, LOW);
          movereje();
          //PARA HACER PRUEBAS//Serial.print(F())
          if (testigo == 3) {
            delay(150);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
            r--; //Hemos llegado a un final de carrera repetimos
movimiento
          }
          else if (testigo != 4) {
            Serial.print(F("\r\nFALLO, SE ABORTA MOVIMIENTO, "));
            Serial.print(F("codigo de error: "));
            Serial.println(testigo);
            i = 0;
          }
          else {
            Serial.print(F("Movimiento: "));  //EL MOVIMIENTO ES
CORRECTO, SE PODRIA INTRODUCIR UNA ESPERA AQUI
            Serial.print(r);
            Serial.println(F(" completado, paso al
siguiente\r\n"));
          }
        }
        else {
          Serial.print(F("Posicion: "));
          Serial.print(r);
          Serial.println(F(" fuera de rango pasamos a la
siguiente"));
        }
        break;
      case 'Y':
        if ((posicion_deseada[r] >= 0.0) && (posicion_deseada[r] <=
 max_Y)) {
          //PARA HACER PRUEBAS//Serial.print(F())
          digitalWrite(ENY, LOW);
          pasos = round(((posicion_deseada[r] - Y) * 49.13616274));
          movereje();
          if (testigo == 3) {
            delay(150);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
            r--; //Hemos llegado a un final de carrera repetimos
movimiento
          }
          else if (testigo != 4) {
            Serial.print(F("\r\nFALLO SE ABORTA MOVIMIENTO, "));
            Serial.print(F("codigo de error: "));
            Serial.println(testigo);
            i = 0;
          }
          else {
            Serial.print(F("Movimiento: "));  //EL MOVIMIENTO ES
CORRECTO, SE PODRIA INTRODUCIR UNA ESPERA AQUI
            Serial.print(r);
```

```
            Serial.println(F(" completado, paso al
siguiente\r\n"));
          }
        }
        else {
          Serial.print(F("Posicion: "));
          Serial.print(r);
          Serial.println(F(" fuera de rango pasamos a la
siguiente"));
        }
        break;
      case 'Z':
        pasos = round((posicion_deseada[r] * 0.555555555));
        if (pasos > 0) {
          digitalWrite(11, LOW);
          digitalWrite(12, LOW);
        }
        else {
          digitalWrite(11, HIGH);
          digitalWrite(12, LOW);
        }
        for (int q = 1; q <= abs(pasos); q++) {
          digitalWrite(13, LOW);
          delayMicroseconds(500);
          digitalWrite(13, HIGH);
          delayMicroseconds(500);
        }
        digitalWrite(12, HIGH);
        testigo = 4;
        Serial.print(F("Movimiento:
")); Serial.print(r); Serial.println(F(" completado, paso al
siguiente\r\n"));
        break;
    }
    Serial.flush();
  }
  if (testigo == 4) {
    Serial.println(F("\r\nHe acabado movimiento global
programado \r\n"));
  }
}
//
//
//
void tipodemovimiento(void) { //esta funcion nos dice en que eje y
sentido es el movimiento
  //que se esta a punto de realizar, la combinacion de
condicionales daria errror
  //si los enables son 0 pero damos por hecho que esta situación no
ocurre nunca,
  //debido a que no hay sincronización de movimientos, esto
ocurriria en caso de
  //que un driver falle o un pin de arduino deje de funcionar. No
hay sistema de proteccion
  //contra el movimiento doble, la unica solución seria un apagado
de emergencia.
  state = 0;
  if (digitalRead(dirpin) == HIGH) {
```

```
    state++; //La variable state se puede considerar un numero
binario compuesto por el valor de DIR en BIT0,ENY en bit1 y ENX en
bit2; la combinacion binaria lo convierte en un registro que
permitira el movimiento, siempre que no implique coordinacion o
haya un estado de no movimiento, requerira previamente haber
programado la dirección del movimiento y el eje.
  }
  if (digitalRead(ENY) == HIGH) {
    state = state + 2;
  }
  if (digitalRead(ENX) == HIGH) {
    state = state + 4;
  }
  switch (state) {
    case 0:
      Serial.println(F("CODIGO DE ERROR 6,HAY DOBLE
MOVIMIENTO,RESETEO PROGRAMA"));
      Serial.flush();
      resetfunc();
      break;
    case 1:
      Serial.println(F("CODIGO DE ERROR 7, HAY DOBLE
MOVIMIENTO,RESETEO PROGRAMA"));
      Serial.flush();
      resetfunc();
      break;
    case 6:
      Serial.println(F("CODIGO DE ERROR 8,NO HAY MOVIMIENTO,RESETEO
PROGRAMA"));
      Serial.flush();
      resetfunc();
      break;
    case 7:
      Serial.println(F("CODIGO DE ERROR 9,NO HAY MOVIMIENTO,RESETEO
PROGRAMA"));
      Serial.flush();
      resetfunc();
      break;
    default:
      if ((state == 2) || (state == 3)) {
        CH1 = 4;
      }
      else {
        CH1 = 5;
      }
      //Serial.print(F("ESTADO
"));Serial.print(state);Serial.println(F(" ADMITIDO,INICIO
MOVIMIENTO"));
      //Serial.flush();
      break;
  }
}
//
//
//
//
void movereje(void) { // mueve el eje un numero de pulsos
determinado;si se mueve el eje a mano, el arduino puede no
registrarlo,
```

```
  //cualquier correcion manual que lleve al posicionador a un
extremo puede causar un mal funcionamiento,
  //se recomienda siempre una puesta a cero al iniciar el programa
  Serial.flush();
  bool movimiento;// esta variable es local solo indica si estamos
ya en el destino antes de empezar
  int t_max = 0;
  control_bucle = 0;
  state = 0;
  testigo = 0;
  if (pasos == 0) {
    digitalWrite(ENX, HIGH);
    digitalWrite(ENY, HIGH);
    Serial.println(F("Ya estabamos en la posicion, no ha hecho
falta moverse"));
    movimiento = false;
  }// estamos en el sitio, no hay que hacer nada
  else {
    movimiento = true; // Tenemos que movernos
  }
  if (movimiento == true) { //si es necesario moverse entramos en
este if
    X_ini = X;//fijamos la posición de comienzo
    Y_ini = Y;
    if (pasos > 0) {//fijamos dirección en funcion de si pasos es
negativo o positivo
      digitalWrite(dirpin, LOW); digitalWrite(Ndirpin,
HIGH); DIR = 0;
    }// sentido positivo
    else {
      digitalWrite(dirpin, HIGH); digitalWrite(Ndirpin,
LOW); DIR = 1;
    }// sentido negativo
    tipodemovimiento();//Se identifica el movimiento que se llevara
a cabo
    Flagfdc = 2; //fijamos funcion de interrupción para el final de
carrera, alerta de mala referencia, activa el encoder
    conteo = 0; // inicializo la variable que cuenta pulsos del
encoder
    if (digitalRead(2)) {
      control_bucle = 11;  //Ha saltado un fin de carrera
      pasos = 0;
    }
    else {
      DDRB |= (1 << 1); // Activamos la salida de pulsos en el pin
10, si esque no salta un final de carrera
    }
    delay(10);
    //El movimiento ya esta en curso
    while (control_bucle <= 10) { // El movimiento se ejecuta en
paralelo a este while,
      //el cual solo sirve para actualizar el valor de la posicion,
      //ademas de limitar el tiempo si el motor se queda parado
hasta un maximo de 200ms.
      //El movimiento sera parado por las interrupciones.
      //El bucle es reiniciado constantemente por el encoder hasta
que el movimiento termine.
      delay(10);//Dividimos el tiempo en 2 delays de 10,
principalmente para que al principio del movimiento
```

```
      //la lectura de los encoders no de problema
      switch (state) {//En función del eje que se mueve hacemos el
muestreo y actualización
        case 5:
          if (DIR == 1) {
            Y = Y_ini - (conteo * avance); //actualiza posicion
          }
          else {
            digitalWrite(ENY, HIGH);  //testigo 2: el sentido no es
el correcto, el driver no funciona,
            //posteriormente se sumara 1,en caso de atasco puede
dar testigo 2 tambien
            testigo = 1;
          }
          break;
        case 3:
          if (DIR == 1) {
            X = X_ini - (conteo * avance); //actualiza posicion
          }
          else {
            digitalWrite(ENX, HIGH);  //testigo 2: el sentido no es
el correcto, el driver no funciona,
            //posteriormente se sumara 1,en caso de atasco puede
dar testigo 2 tambien
            testigo = 1;
          }
          break;
        case 4:
          if (DIR == 0) {
            Y = Y_ini + (conteo * avance); //actualiza posicion
          }
          else {
            digitalWrite(ENY, HIGH);  //testigo 2: el sentido no es
el correcto, el driver no funciona,
            //posteriormente se sumara 1,en caso de atasco puede
dar testigo 2 tambien
            testigo = 1;
          }
          break;
        case 2:
          if (DIR == 0) {
            X = X_ini + (conteo * avance); //actualiza posicion
          }
          else {
            digitalWrite(ENX, HIGH);  //testigo  el sentido no es
el correcto, el driver no funciona,
            //posteriormente se sumara 1,en caso de atasco puede
dar testigo 2 tambien
            testigo = 1;
          }
          break;
      }
      if (t_max >= 600) {
        digitalWrite(ENX, HIGH);  //el movimiento es muy
lento,tarda mas de 12 segundos, revisar,
        //posible atasco, el testigo seria el 5.
        digitalWrite(ENY, HIGH);
        testigo = 5;
      }
```

```
      //12 SEGUNDOS ES 600 CICLOS
      delay(10);
      control_bucle++;
      t_max++;
    }
    switch (state) { // aseguramos que ningun driver funcione
      case 5:
        digitalWrite(ENY, HIGH);
        break;
      case 3:
        digitalWrite(ENX, HIGH);
        break;
      case 4:
        digitalWrite(ENY, HIGH);
        break;
      case 2:
        digitalWrite(ENX, HIGH);
        break;
    }
    DDRB &= ~(1 << 1); // desactivo la salida de pulsos
    if ((pasos != 0) && (abs(pasos) > conteo) && (testigo != 5)) {
      testigo++; //Este testigo es 1 si el motor se ha atascado, es
dos si el sentido es erroneo.
    }
    else if (pasos == 0) { //Ha saltado el final de carrera
      //ACTUALIZAMOS POSICION ABSOLUTA
      switch (state) {
        case 5:
          Y = 0;
          break;
        case 3:
          X = 0;
          break;
        case 4:
          Y = max_Y;
          break;
        case 2:
          X = max_X;
          break;
        default:// No hay estado por defecto, siempre estaremos en
uno de estos 4 estados,
          //durante la ejecucion de esta funcion
          break;
      }
      testigo = 3; // testigo 3 hemos llegado a un final de
carrera, el movimiento se ha acabado,
      //se requiere un recalculo de la posición
    }
    else {
      testigo = 4; //Hemos llegado a la posicion deseada
    }
  }
  else {
    testigo = 4; //Hemos llegado a la posicion deseada, solo que
esta vez no ha hecho falta movernos
  }
  Flagfdc = 0; //el final de carrera vuelve a su funcion por
defecto
}
```

```
//
//
//
//
//
void encoder(void) {   // la funcion del encoder debe actualizar
conteo y bloquear los enables cuando
  //conteo y pasos sean iguales, ademas dara valor a DIR y reinicia
el bucle de control
  if (Flagfdc == 2) {//El encoder se activa cuando movereje() se
ejecuta
    //identificar DIR
    if (digitalRead(CH1) == LOW) {
      DIR = 1; //La interrupción se ha fijado en los flancos de
subida
    }
    else {
      DIR = 0;
    }
    conteo++;
    if (conteo >= abs(pasos)) {
      //cerramos ENABLE
      switch (state) { // aseguramos que ningun driver funcione
        case 5:
          digitalWrite(ENY, HIGH);
          break;
        case 3:
          digitalWrite(ENX, HIGH);
          break;
        case 4:
          digitalWrite(ENY, HIGH);
          break;
        case 2:
          digitalWrite(ENX, HIGH);
          break;
      }
    }
    else {
      control_bucle = 0;
    }//reiniciar bucle de control, ya que el movimiento no ha
terminado
  }
}
//
//
//
void finaldecarrera(void) { // funcion que controla los modos de
trabajo del final de carrera,
  // la funcion flagfdc 2 debe bloquear los enables y convertir la
variable pasos en 0
  switch (Flagfdc) {
    //USAMOS UN SWITCH POR SI EN EL FUTURO SE DESEAN AÑADIR MAS
FUNCIONES A LA INTERRUPCION
    case 2://Asociado a movereje()
      Flagfdc = 0; //Para que no haga dobles señales, empiricamente
demostrado
      //un final de carrera da dos pulsos en lugar de uno, cambiar
Flagfdc no altera el movimieno
```

```
      //Ya que una vez salta el final de carrera movereje()
termina.
      //Serial.println(F("salto el final de carrera"));
      pasos = 0; //identifica que hemos llegado a un final de
carrera
      switch (state) { // aseguramos que ningun driver funcione
        case 5:
          digitalWrite(ENY, HIGH);
          break;
        case 3:
          digitalWrite(ENX, HIGH);
          break;
        case 4:
          digitalWrite(ENY, HIGH);
          break;
        case 2:
          digitalWrite(ENX, HIGH);
          break;
      }
      break;
  }
}
```

Programa posicionador_automatizado

Este programa es la misma idea que el anterior solo que eliminando todos los mensajes que hacían su manejo más interactivo. Destinado para que lo use un programa externo o un autómata.

```
#define avance 0.02035160958//avance en mm por cada paso
#define max_Y 177.0//Limite en Y
#define max_X 211.0//Limite en X
#define dirpin 10//pin de dirección
#define Ndirpin 8// pin de direcció negada
#define ENX 7//Pin de enable del eje X
#define ENY 6//Pin de enable del eje Y
//Vectores de posicionamiento
volatile char orden_mov[19] {};
volatile float posicion_deseada[19] {};
volatile char orden_desp[19] {};
volatile float desplazamiento_deseado[19] {};
byte Flagfdc;//Almacena la funcion del encoder y final de carrera
byte state;//Almacena el tipo de movimiento que se va a realizar
volatile int pasos;//nº de pasos de encoder a realizar
volatile byte control_bucle;//Bucle de control de posicionamiento
volatile int conteo;//Pulsos del encoder
volatile byte DIR;//Dirección detectada en el encoder
byte testigo;//Resultado del movimiento
byte CH1;//Canal 1 del encoder qu hay que escuchar
float Y, Y_ini, X, X_ini;//X e Y son las posiciones absolutas;
//X_ini e Y_ini son las posiciones donde comenzamos a movernos
void(*resetfunc)(void) = 0;//Resetea el sketch, NO el arduino
void Timer1(void);//Configura el timer 1 al pin 9
void esperar_dato(void);//Espera dato por puerto serie
void vaciar_buffer_entrada(void);//Vacia el buffer de lectura
void movimiento_Y_0(void);//Programa de gestion de movimiento,
posiciona a Y 0
void movimiento_Y_max(void);//Programa de gestion de movimiento,
posiciona a Y max
```

```
void movimiento_X_0(void);//Programa de gestion de movimiento,
posiciona a X 0
void movimiento_X_max(void);//Programa de gestion de movimiento,
posiciona a X max
void RUTINAB(void);//Submenu de posicionamiento absoluto
void RUTINAC(void);//Submenu de posicionamiento relativo
void DESPLAZAMIENTO(byte i1);//Realiza la secuencia de
deplazamientos
void MOVIMIENTO_GLOBAL(byte i);//Realiza la secuencia de
posicionamiento
void tipodemovimiento(void);//Configura en "state" el movimiento
que esta a punto de realizarse
void movereje(void);//Mueve el eje configurado un numero de pasos
fijados
void encoder(void);//funcion de interrupcion del encoder
void finaldecarrera(void);//funcion de interrupcion del final de
carrera
void setup() {
  //Inciamos en el 0 absoluto
  X = 0.0;
  Y = 0.0;
  //Configuramos entradas y salidas, activamos interrupciones,
puerto serie y timer1
  pinMode(ENX, OUTPUT);
  pinMode(ENY, OUTPUT);
  pinMode(dirpin, OUTPUT);
  pinMode(Ndirpin, OUTPUT);
  pinMode(2, INPUT); //FDC
  pinMode(3, INPUT); //ENCODER
  pinMode(4, INPUT_PULLUP); //ch1 encoder x
  pinMode(5, INPUT_PULLUP); //ch1 encoder y
  pinMode(9, OUTPUT); //STEP
  pinMode(11, OUTPUT); //DIRZ
  pinMode(12, OUTPUT); //ENZ
  pinMode(13, OUTPUT); //STEPZ
  digitalWrite(12, HIGH); //desactivo en z
  digitalWrite(ENX, HIGH); //desaactivo en x
  digitalWrite(ENY, HIGH); //desactivo en y
  attachInterrupt(digitalPinToInterrupt(3), encoder,
RISING); //activo ISR encoder
  attachInterrupt(digitalPinToInterrupt(2), finaldecarrera,
RISING); //activo ISR final de carrera
  Timer1();
  Serial.begin(9600);
}
void loop() {
  char funcion;
  bool submenu;
  //La lectura incial de una letra de referencia efectuara una
acción sonre el posicionador
  //El significado de las salidas por puerto serie viene explicado
en la memoria
  vaciar_buffer_entrada();
  Serial.println(F("I"));//Se puede mandar ordenes
  esperar_dato();
  funcion = Serial.read();
  switch (funcion) {
    case 'G'://INICIA LA COMUNICACIÓN PARA UN MOVIMIENTO GLOBAL
      RUTINAB();
```

```
      break;
    case 'R' ://INICIA LA COMUNICACIÓN PARA UN MOVIMIENTO RELATIVO
      RUTINAC();
      Serial.print("\n\r");//Desconocemos el motivo pero sino no
funciona
      break;
    default:
      Serial.println(F("ER0"));//error de lectura
      Serial.flush();
      break;
    case 'C'://Hace un movimiento a 0 Y
      movimiento_Y_0();
      break;
    case 'D':// Hace un movimiento a Y maxima
      movimiento_Y_max();
      break;
    case 'A'://Hace un movimiento a 0 X
      movimiento_X_0();
      break;
    case 'B':// Hace un movimiento a X maxima
      movimiento_X_max();
      break;
    case 'P'://INDICA LA POSICION GLOBAL
      Serial.print(F("X")); Serial.println(X);
      Serial.print(F("Y")); Serial.println(Y);
      break;
      //MOVIMIENTOS ESPECIFICOS
      //Los movimientos especificos se han programado como un
procedimiento, originalmente se intento
      //escribir el codigo dentro del case, pero se vio
empiricamente que esto daba lugar a error,
      //concretamente el error consistia en que los "case" que
hubiesen debajo del codigo de
      //RUTINAB O RUTINAC (esto es si solo ponemos el codigo, al
usar el procedimiento no hay problema)eran ignorados.
      //Desconocemos el motivo, pero podemos creer que tiene que
ver con la memoria, ya que realizamos muchas pruebas,
      //sin encontrar un problema en el codigo, entre las pruebas
realizadas estaba sustituir RUTINAB() O RUINAC()
      //por su codigo entre corchetes("{}", los corchetes hacen al
arduino interpretar las ordenes como una funcion),
      //y se vio que con los corchetes,la comunicacion funcionaba,
pero sin corchetes dejaba de funcionar correctamente.
  }
}
//
//
void Timer1(void) {// funcion que configura el timer1 elcual
actuara como una salida de pulsos constante a 1000 HZ
  DDRB &= ~(1 << 1); // selecciono salida del timer y DESACTIVARLA
  TCCR1A = 0b10000001; // fijo la frecuencia 1000Hz
  TCCR1B = 0b00001011;
  OCR1A = 127;
}
//FUNICONES BASICAS PARA LA COMUNICACION SERIE
//
//
void esperar_dato(void) {//espera un dato por el puerto serie
  while (Serial.available() == 0) {}
```

```
  delay(70);
}
void vaciar_buffer_entrada(void) {//Vacia el buffer de lectura
  while (Serial.available() != 0) {
    Serial.read();
  }
}
//
//
//PROGRAMAS PARA LA COMUNICACION SERIE EN MOVIMIENTOS RELATIVOS Y
GLOBALES
//
//
void RUTINAB(void) { //Programa que realiza la comunicación para
realizar un movimiento global
  //Serial.println(F("He entrado en RUTINAB"));
  byte p = 0;
  volatile char letra, data;
  //Ponemos a 0 los vectores de informacion del posicionamiento
  orden_mov[19] = {};
  posicion_deseada[19] = {};
  while ((Serial.available()) && (p < 18)) { //LEE HASTA 18
POSICIONES INTERCALADAS DE X,Y,Z,
    letra = Serial.read();
    switch (letra) {
      case 'X':
        p++;//Si p es igual a 0 es que ha saltado el default
        orden_mov[p] = 'X';
        posicion_deseada[p] = Serial.parseFloat();
        break;
      case 'Y':
        p++;
        orden_mov[p] = 'Y';
        posicion_deseada[p] = Serial.parseFloat();
        break;
      case 'Z':
        p++;
        orden_mov[p] = 'Z';
        posicion_deseada[p] = Serial.parseFloat();
        break;
      default:
        vaciar_buffer_entrada();//nos hara salir del while de
lectura
        if (((letra != '\n') && (letra != '\r')) || (p == 0)) {
          p = 0;
          Serial.println(F("ER0"));//error de lectura,reinica el
bucle loop
          Serial.flush();
        }
        break;
    }
  }
  vaciar_buffer_entrada();
  if (p != 0)  { // Si se han introducido datos validos entramos
aqui
    MOVIMIENTO_GLOBAL(p);
  }
}
//
```

```
//
void RUTINAC(void) { //INICIA UN MOVIMIENTO RELATIVO
  byte p1 = 0;
  volatile char letra1, data1;
  //Ponemos a 0 los vectores de informacion del desplazamiento
  orden_desp[19] = {};
  desplazamiento_deseado[19] = {};
  while ((Serial.available()) && (p1 < 18)) { //LEE HASTA 18
POSICIONES INTERCALADAS DE X,Y,Z
    letra1 = Serial.read();
    switch (letra1) {
      case 'X':
        p1++;//Si p1 es igual a 0 es que ha saltado el default
        orden_desp[p1] = 'X';
        desplazamiento_deseado[p1] = Serial.parseFloat();
        break;
      case 'Y':
        p1++;
        orden_desp[p1] = 'Y';
        desplazamiento_deseado[p1] = Serial.parseFloat();
        break;
      case 'Z':
        p1++;
        orden_desp[p1] = 'Z';
        desplazamiento_deseado[p1] = Serial.parseFloat();
        break;
      default:
        vaciar_buffer_entrada();//Nos saca del bucle while
        if (((letra1 != '\n') && (letra1 != '\r')) || (p1 == 0)) {
          p1 = 0;
          Serial.print(F("ER0"));//error de lectura,reinica el
bucle loop
          Serial.flush();
        }
        break;
    }
  }
  vaciar_buffer_entrada();
  if (p1 != 0)  { // Si se han introducido datos validos entramos
aqui,
    DESPLAZAMIENTO(p1);
  }
}
//
//
//PROGRAMAS PARA EJECUTAR LOS MOVIMIENTOS DE MANERA CONTROLADA
//
//
void movimiento_Y_0(void) {
  //Configuramos el movimiento
  digitalWrite(ENY, LOW);
  digitalWrite(dirpin, HIGH);
  digitalWrite(Ndirpin, LOW);
  if (digitalRead(2)) {//Si ya estamos en el extremo nos alejamos
de el
    digitalWrite(dirpin, LOW);
    DDRB |= (1 << 1);
    delay(800);
    DDRB &= ~(1 << 1);
```

```
    digitalWrite(dirpin, HIGH);
    delay(300);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
  }
  pasos = -30000;
  movereje();//Nos movemos -30000 pasos algo imposible de alcanzar
  if (testigo != 3) {
    Serial.print(F("ER"));
    Serial.println(testigo);
  }
  else {
    Serial.println(F("OK"));
  }
}
void movimiento_Y_max(void) {
  //Configuramos el movimiento
  digitalWrite(ENY, LOW);
  digitalWrite(dirpin, LOW);
  digitalWrite(Ndirpin, HIGH);
  if (digitalRead(2)) {//Si ya estamos en el extremo nos alejamos
de el
    digitalWrite(dirpin, HIGH);
    DDRB |= (1 << 1);
    delay(800);
    DDRB &= ~(1 << 1);
    digitalWrite(dirpin, LOW);
    delay(300);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
  }
  pasos = 30000;//Nos movemos 30000 pasos algo imposible de
alcanzar
  movereje();
  if (testigo != 3) {
    Serial.print(F("ER"));
    Serial.println(testigo);
  }
  else {
    Serial.println(F("OK"));
  }
}
void movimiento_X_0(void) {
  //Configuramos el movimiento
  digitalWrite(ENX, LOW);
  digitalWrite(dirpin, HIGH);
  digitalWrite(Ndirpin, LOW);
  if (digitalRead(2)) {//Si ya estamos en el extremo nos alejamos
de el
    digitalWrite(dirpin, LOW);
    DDRB |= (1 << 1);
    delay(800);
    DDRB &= ~(1 << 1);
    digitalWrite(dirpin, HIGH);
    delay(300);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
  }
  pasos = -30000;//Nos movemos -30000 pasos algo imposible de
alcanzar
  movereje();
  if (testigo != 3) {
```

```
    Serial.print(F("ER"));
    Serial.println(testigo);
  }
  else {
    Serial.println(F("OK"));
  }
}
void movimiento_X_max(void) {
  //Configuramos el movimiento
  digitalWrite(ENX, LOW);
  digitalWrite(dirpin, LOW);
  digitalWrite(Ndirpin, HIGH);
  if (digitalRead(2)) {//Si ya estamos en el extremo nos alejamos
de el
    digitalWrite(dirpin, HIGH);
    DDRB |= (1 << 1);
    delay(800);
    DDRB &= ~(1 << 1);
    digitalWrite(dirpin, LOW);
    delay(300);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
  }
  pasos = 30000;//Nos movemos 30000 pasos algo imposible de
alcanzar
  movereje();
  if (testigo != 3) {
    Serial.print(F("ER"));
    Serial.println(testigo);
  }
  else {
    Serial.println(F("OK"));
  }
}
void DESPLAZAMIENTO(byte i1) {
  //Ejecuta la secuencia de desplazamientos
  for (byte r1 = 1; r1 <= i1; r1++) {
    switch (orden_desp[r1]) {
      case 'X':
        pasos = round((desplazamiento_deseado[r1] * 49.13616274));/
/transforma los mm en pasos
        digitalWrite(ENX, LOW);
        movereje();
        if (testigo != 4) {
          Serial.print(F("ER")); Serial.print(testigo); //SE ABORTA
PROGRAMA
          i1 = 0;
          //EL MOVIMIENTO ES CORRECTO, SE PODRIA INTRODUCIR UNA
ESPERA AQUI
        }
        break;
      case 'Y':
        pasos = round((desplazamiento_deseado[r1] * 49.13616274));
        digitalWrite(ENY, LOW);
        movereje();
        if (testigo != 4) {
          Serial.print(F("ER")); Serial.print(testigo);
          i1 = 0;
          //EL MOVIMIENTO ES CORRECTO, SE PODRIA INTRODUCIR UNA
ESPERA AQUI
```

```
        }
        break;
      case 'Z':
        //No hay realimentación, se ejecuta un movimiento relativo
que se considerara correcto, es decir sin errores
        pasos = round((desplazamiento_deseado[r1] * 0.555555555));
        if (pasos > 0) {
          digitalWrite(11, LOW);
          digitalWrite(12, LOW);
        }
        else {
          digitalWrite(11, HIGH);
          digitalWrite(12, LOW);
        }
        for (int q1 = 1; q1 <= abs(pasos); q1++) {
          digitalWrite(13, LOW);
          delayMicroseconds(600);
          digitalWrite(13, HIGH);
          delayMicroseconds(600);
        }
        digitalWrite(12, HIGH);
        testigo = 4;
        break;
    }
    Serial.flush();
  }
  if (testigo == 4) {
    Serial.print(F("OK"));
  }
}
void MOVIMIENTO_GLOBAL(byte i) {
  //Ejecuta la secuencia de posicionamiento
  for (byte r = 1; r <= i; r++) {
    switch (orden_mov[r]) {
      //Convierte el posicionamiento en pasos, asegura que esta
dentro de los limites, en caso
      //contrario pasa al siguiente. El testigo 3 repite el
movimiento y los demas salvo el 4 abortan
      //el posicionamiento
      case 'X':
        if ((posicion_deseada[r] > 0.0) && (posicion_deseada[r] < m
ax_X)) {
          pasos = round(((posicion_deseada[r] - X) * 49.13616274));
          digitalWrite(ENX, LOW);
          movereje();
          if (testigo == 3) {
            delay(150);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
            r--; //Hemos llegado a un final de carrera repetimos
movimiento
          }
          else if (testigo != 4) {
            Serial.print(F("ER")); Serial.println(testigo);
            i = 0;
          }
          else {} //EL MOVIMIENTO ES CORRECTO, SE PODRIA INTRODUCIR
UNA ESPERA AQUI
        }
        else {
```

```
          //Serial.print(F("Posicion: "));
          //Serial.print(r);
          //Serial.println(F(" fuera de rango pasamos a la
siguiente"));
        }
        break;
      case 'Y':
        if ((posicion_deseada[r] > 0.0) && (posicion_deseada[r] < m
ax_Y)) {
          //PARA HACER PRUEBAS//Serial.print(F())
          digitalWrite(ENY, LOW);
          pasos = round(((posicion_deseada[r] - Y) * 49.13616274));
          movereje();
          if (testigo == 3) {
            delay(150);//evitar movimientos bruscos entre parada y
movimiento en direccion contraria
            r--; //Hemos llegado a un final de carrera repetimos
movimiento
          }
          else if (testigo != 4) {
            Serial.print(F("ER")); Serial.println(testigo);
            i = 0;
          }
          else {} //EL MOVIMIENTO ES CORRECTO, SE PODRIA INTRODUCIR
UNA ESPERA AQUI
        }
        else {
          // Serial.print(F("Posicion: "));
          //Serial.print(r);
          //Serial.println(F(" fuera de rango pasamos a la
siguiente"));
        }
        break;
      case 'Z':
        //No hay realimentación, se ejecuta un movimiento relativo
que se considerara correcto, es decir sin errores
        pasos = round((posicion_deseada[r] * 0.555555555));
        if (pasos > 0) {
          digitalWrite(11, LOW);
          digitalWrite(12, LOW);
        }
        else {
          digitalWrite(11, HIGH);
          digitalWrite(12, LOW);
        }
        for (int q = 1; q <= abs(pasos); q++) {
          digitalWrite(13, LOW);
          delayMicroseconds(500);
          digitalWrite(13, HIGH);
          delayMicroseconds(500);
        }
        digitalWrite(12, HIGH);
        testigo = 4;
        break;
    }
    Serial.flush();
  }
  if (testigo == 4) {//Si el ultimo testigo ha sido 4 el movimiento
es correcto
```

```
    Serial.println(F("OK"));//Movimiento correcto
  }
}
//
//
//
void tipodemovimiento(void) { //esta funcion nos dice en que eje y
sentido es el movimiento
  //que se esta a punto de realizar, la combinacion de
condicionales daria errror si los enables son 0
  //pero damos por hecho que esta situación no ocurre nunca, debido
a que no hay sincronización de movimientos,
  //esto ocurriria en caso de que un driver falle o un pin de
arduino deje de funcionar.
  //No hay sistema de proteccion contra el movimiento doble, la
unica solución seria un apagado de emergencia.
  state = 0;
  if (digitalRead(dirpin) == HIGH) {
    state++; //La variable state se puede considerar un numero
binario compuesto por el valor de DIR en BIT0,ENY en bit1 y ENX en
bit2; la combinacion binaria lo convierte en un registro que
permitira el movimiento, siempre que no implique coordinacion o
haya un estado de no movimiento, requerira previamente haber
programado la dirección del movimiento y el eje.
  }
  if (digitalRead(ENY) == HIGH) {
    state = state + 2;
  }
  if (digitalRead(ENX) == HIGH) {
    state = state + 4;
  }
  switch (state) {
    case 0:
      Serial.println(F("ER6"));
      //Serial.println(F("CODIGO DE ERROR DE PREPARACION 0,HAY
DOBLE MOVIMIENTO,RESETEO PROGRAMA"));
      Serial.flush();
      resetfunc();
      break;
    case 1:
      Serial.println(F("ER7"));
      //Serial.println(F("CODIGO DE ERROR DE PREPARACION 1, HAY
DOBLE MOVIMIENTO,RESETEO PROGRAMA"));
      Serial.flush();
      resetfunc();
      break;
    case 6:
      Serial.println(F("ER8"));
      //Serial.println(F("CODIGO DE ERROR DE PREPARACION 6,NO HAY
MOVIMIENTO,RESETEO PROGRAMA"));
      Serial.flush();
      resetfunc();
      break;
    case 7:
      Serial.println(F("ER9"));
      //Serial.println(F("CODIGO DE ERROR DE PREPARACION 7,NO HAY
MOVIMIENTO,RESETEO PROGRAMA"));
      Serial.flush();
      resetfunc();
```

```
      break;
    default:
      if ((state == 2) || (state == 3)) {
        CH1 = 4;
      }
      else {
        CH1 = 5;
      }
      break;
  }
}
//
//
//
//
void movereje(void) { // Mueve el eje un numero de pulsos
determinado;si se mueve el eje a mano,
  //el arduino puede no registrarlo, cualquier correcion manual que
lleve al posicionador a un
  //extremo puede causar un mal funcionamiento, se recomienda
siempre una puesta a cero al iniciar el programa
  Serial.flush();
  bool movimiento;// esta variable es local solo indica si es
necesario realizar el movimiento
  int t_max = 0;
  control_bucle = 0;
  state = 0;
  testigo = 0;
  if (pasos == 0) {
    digitalWrite(ENX, HIGH);
    digitalWrite(ENY, HIGH);
    //Serial.println(F("Ya estabamos en la posicion, no ha hecho
falta moverse"));
    movimiento = false;
  }// estamos en el sitio, no hay que hacer nada
  else {
    movimiento = true; // Debemos movernos
  }
  if (movimiento == true) { //si es necesario moverse entramos en
este if
    X_ini = X;//Fijamos la posición de comienzo
    Y_ini = Y;
    if (pasos > 0) {
      digitalWrite(dirpin, LOW); digitalWrite(Ndirpin,
HIGH); DIR = 0;
    }// sentido positivo
    else {
      digitalWrite(dirpin, HIGH); digitalWrite(Ndirpin,
LOW); DIR = 1;
    }// sentido negativo
    tipodemovimiento();//Se identifica el movimiento que se llevara
a cabo
    Flagfdc = 2; //fijamos funcion de interrupción para el final de
carrera, activa el encoder
    conteo = 0; // inicializo la variable que cuenta pulsos del
encoder
    if (digitalRead(2)) {//Estamos en un extremo
      control_bucle = 11;  //Ha saltado un fin de carrera
      pasos = 0;
```

```
    }
    else {
      DDRB |= (1 << 1); // Activamos la salida de pulsos en el pin
10, si esque no salta un final de carrera
    }
    delay(10);
    while (control_bucle <= 10) { // El movimiento se ejecuta en
paralelo a este while, el cual solo sirve para
      //actualizar el valor de la posicion, ademas de limitar el
tiempo si el motor se queda parado hasta un maximo de 200ms.
      //El encoder reinicia constantemente "control_bucle" hasta
que el movimiento acaba
      delay(10);//Dividimos el tiempo en 2 delay de 10,
principalmente para que al principio del movimiento la lectura de
      //los encoders no de problema
      switch (state) {
        case 5:
          if (DIR == 1) {
            Y = Y_ini - (conteo * avance); //actualiza posicion
          }
          else {
            digitalWrite(ENY, HIGH);  //testigo 2: el sentido no es
el correcto, el driver no funciona,
            //posteriormente se sumara 1,en caso de atasco puede
dar testigo 2 tambien
            testigo = 1;
          }
          break;
        case 3:
          if (DIR == 1) {
            X = X_ini - (conteo * avance); //actualiza posicion
          }
          else {
            digitalWrite(ENX, HIGH);  //testigo 2: el sentido no es
el correcto, el driver no funciona,
            //posteriormente se sumara 1,en caso de atasco puede
dar testigo 2 tambien
            testigo = 1;
          }
          break;
        case 4:
          if (DIR == 0) {
            Y = Y_ini + (conteo * avance); //actualiza posicion
          }
          else {
            digitalWrite(ENY, HIGH);  //testigo 2: el sentido no es
el correcto, el driver no funciona,
            //posteriormente se sumara 1,en caso de atasco puede
dar testigo 2 tambien
            testigo = 1;
          }
          break;
        case 2:
          if (DIR == 0) {
            X = X_ini + (conteo * avance); //actualiza posicion
          }
          else {
            digitalWrite(ENX, HIGH);  //testigo  el sentido no es
el correcto, el driver no funciona,
```

```
            //posteriormente se sumara 1,en caso de atasco puede
dar testigo 2 tambien
            testigo = 1;
          }
          break;
      }
      if (t_max >= 600) {
        digitalWrite(ENX, HIGH);  //el movimiento es muy
lento,tarda mas de 12 segundos, revisar, posible atasco,
        //el testigo seria el 5
        digitalWrite(ENY, HIGH);
        testigo = 5;
      }
      //12 SEGUNDOS ES 600 CICLOS
      delay(10);
      control_bucle++;
      t_max++;
    }
    switch (state) { // aseguramos que ningun driver funcione
      case 5:
        digitalWrite(ENY, HIGH);
        break;
      case 3:
        digitalWrite(ENX, HIGH);
        break;
      case 4:
        digitalWrite(ENY, HIGH);
        break;
      case 2:
        digitalWrite(ENX, HIGH);
        break;
    }
    DDRB &= ~(1 << 1); // desactivo la salida de pulsos
    //Procedemos a dar un valor a la variable testigo
    if ((pasos != 0) && (abs(pasos) > conteo) && (testigo != 5)) {
      testigo++; //Este testigo es 1 si el motor se ha atascado. es
dos si el sentido es erroneo.
    }
    else if (pasos == 0) { //Ha saltado el final de carrera,
ACTUALIZAMOS POSICION ABSOLUTA
      switch (state) {
        case 5:
          Y = 0;
          break;
        case 3:
          X = 0;
          break;
        case 4:
          Y = max_Y;
          break;
        case 2:
          X = max_X;
          break;
        default:// No hay estado por defecto, siempre estaremos en
uno de estos 4 estados,
          //durante la ejecucion de esta funcion
          break;
      }
```

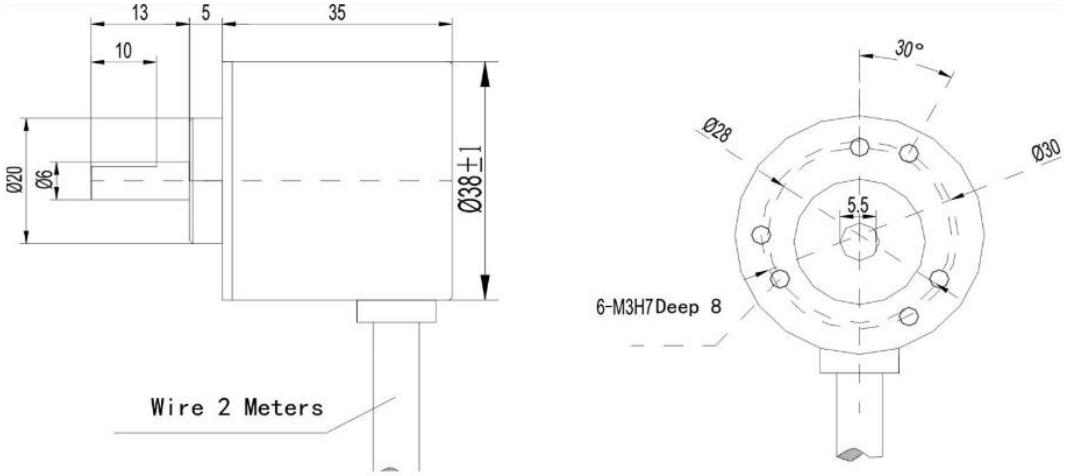
```
      testigo = 3; // testigo 3 hemos llegado a un final de
carrera, el movimiento se ha acabado,
      //se requiere un recalculo de la posición
    }
    else {
      testigo = 4; //Hemos llegado a la posicion deseada
    }
  }
  else {
    testigo = 4; //Hemos llegado a la posicion deseada,
    //solo que esta vez no ha hecho falta movernos
  }
  Flagfdc = 0; //el final de carrera vuelve a su funcion por
defecto
}
//
//
//
//
//
void encoder(void) {   // la funcion del encoder debe actualizar
conteo y bloquear los enables cuando
  //conteo y pasos sean iguales, ademas dara valor a DIR y reinicia
el bucle de control
  if (Flagfdc == 2) {//El encoder se activa cuando movereje() se
ejecuta
    //identificar DIR
    if (digitalRead(CH1) == LOW) {
      DIR = 1; ////La interrupción se ha fijado en los flancos de
subida
    }
    else {
      DIR = 0;
    }
    conteo++;
    if (conteo >= abs(pasos)) {
      //cerramos ENABLE
      switch (state) { // aseguramos que ningun driver funcione
        case 5:
          digitalWrite(ENY, HIGH);
          break;
        case 3:
          digitalWrite(ENX, HIGH);
          break;
        case 4:
          digitalWrite(ENY, HIGH);
          break;
        case 2:
          digitalWrite(ENX, HIGH);
          break;
      }
    }
    else {
      control_bucle = 0;
    }//reiniciar bucle de control
  }
}
///
//
```

```
//
//
//
//
void finaldecarrera(void) { // funcion que controla los modos de
trabajo del final de carrera,
  //la funcion flagfdc 2 debe bloquear los enables y convertir la
variable pasos en 0
  switch (Flagfdc) {
    //USAMOS UN SWITCH POR SI EN EL FUTURO SE DESEAN AÑADIR MAS
FUNCIONES A LA INTERRUPCION
    case 2://Asociado a movereje()
      Flagfdc = 0; //Para que no haga dobles señales
      pasos = 0; //Hemos llegado a un final de carrera
      switch (state) { // aseguramos que ningun driver funcione
        case 5:
          digitalWrite(ENY, HIGH);
          break;
        case 3:
          digitalWrite(ENX, HIGH);
          break;
        case 4:
          digitalWrite(ENY, HIGH);
          break;
        case 2:
          digitalWrite(ENX, HIGH);
          break;
      }
      break;
  }
}
```

Anexo II: Especificaciones de los componentes

En este apartado mostraremos planos y parámetros fundamentales de los componentes utilizados. No nombraremos componentes de la placa de circuito impreso salvo por el driver del motor, para hablar de pruebas hechas al motor y al encoder.

Encoder

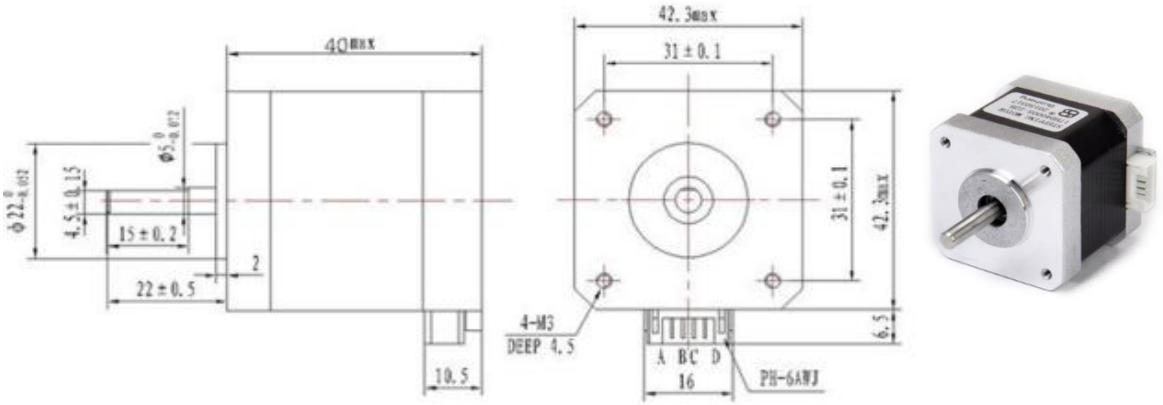


Feature:
AB two-phase incremental optical rotary encoder 400 pulses,including NPN open collector output and voltage output two kinds of output.
Standard with 1.5 m cable
Performance: 400 pulses/rev.
Operating voltage: DC5-24V
Maximum mechanical speed 5000 rev / min
The electrical response frequency 20K / sec
Integrated speed 2000 rev / min
Size: Encoder body size: φ38mm; shaft φ6 × 13mm; axis platform: High 5mm, φ20mm; fixing holes for: M3 screws
Three mounting holes on the circle 30, and the other three mounting holes on the 28 circle ; side qualify.

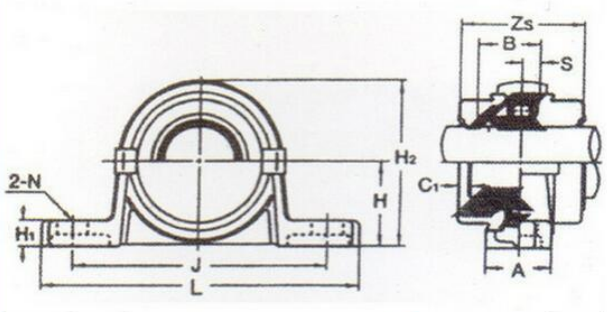
Motor

Corriente nominal (una fase): 1.3A DC
Tensión nominal: 2.4V
Ángulo de incremento: 1,8 °

De corriente continua resistencia del devanado (25 °C): 1,6 Ω ± 10%
Wnding inductancia: 3.2mH ± 20%
Par de detención: 18mN.m REF
Par de retención: ≥ 360mN.m (I = 1,3 A)

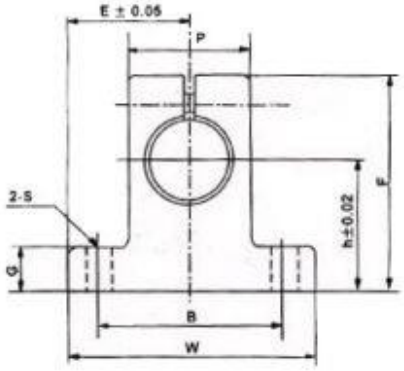


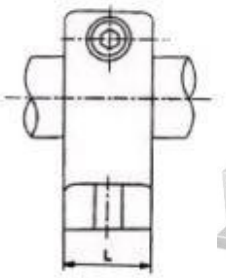
Rodamiento



带座轴承型号NO.	轴径	外形尺寸Dimensions(mm)										螺栓
	(mm)	H	L	J	A	N	H1	H2	BI	S	CI	ZS	(mm)
KP08	8	15	55	42	13	4.8	5	29	15	3.5			M4

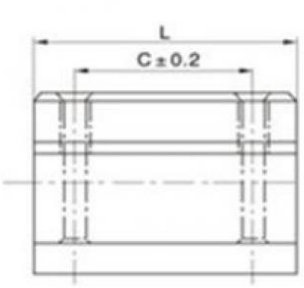
Apoyo fijo

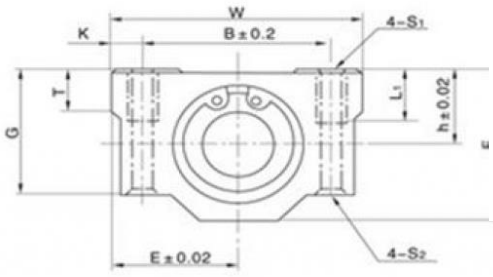




	Rod Dia.	h	E	W	L	F	G	P	B	S	Set Screw	Attach Screws	Weight (kg)
SK8	8	20	21	42	14	32.8	6	18	32	5.5	M4	M5	0.024

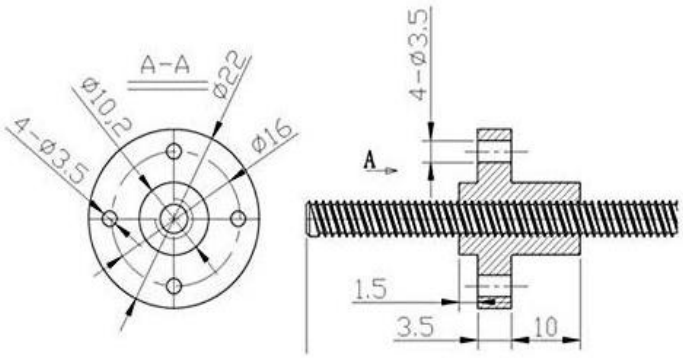
Carriles deslizantes



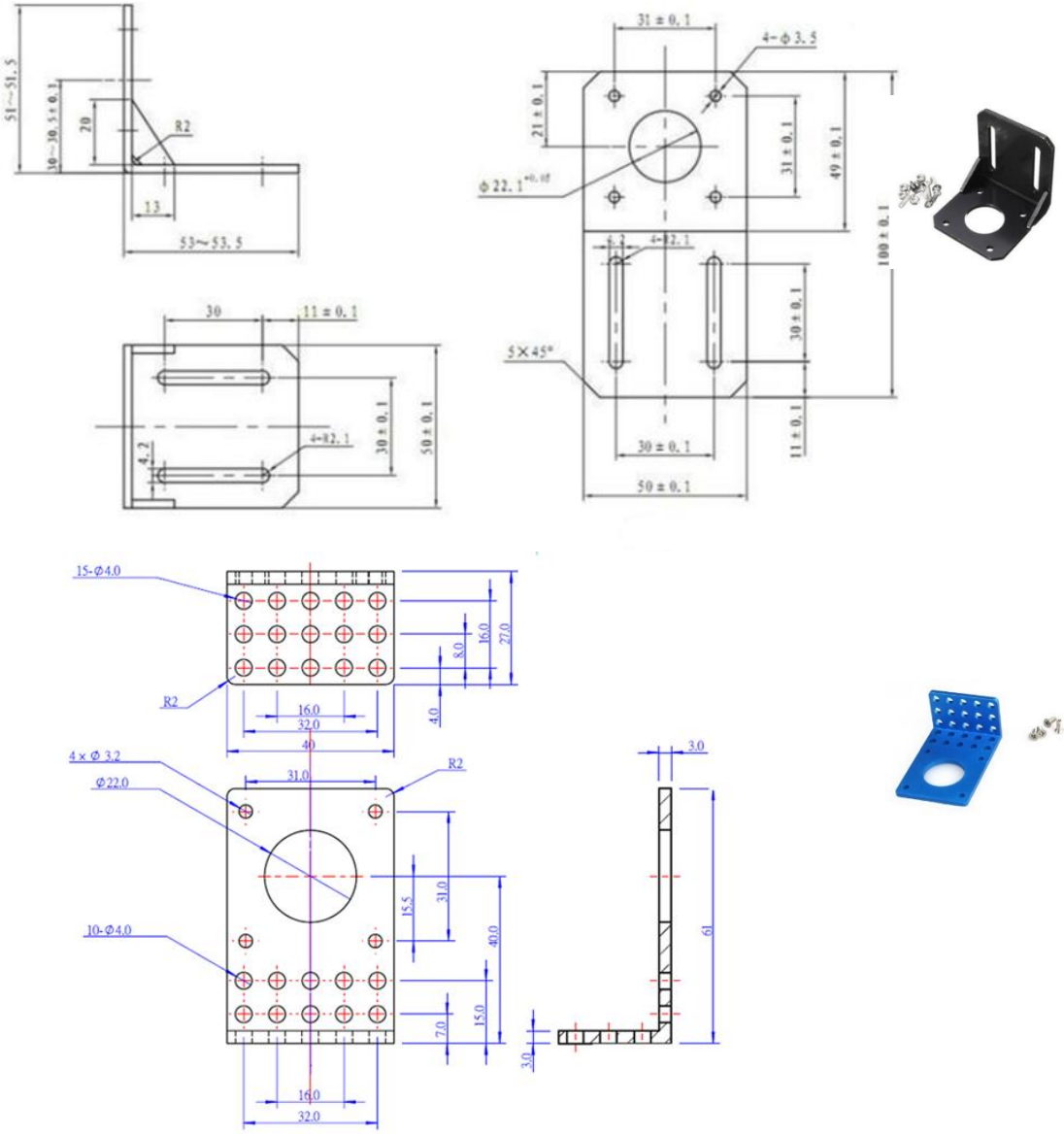


Bezeichnung Designation	Abmessungen / Dimensions [mm]												
	T	h	E	W	L	F	G	B	C	K	S1	S2	L1
SCS8UU	6	11	17	34	30	22	18	24	18	5	M4	3,4	8

Tuerca de avance

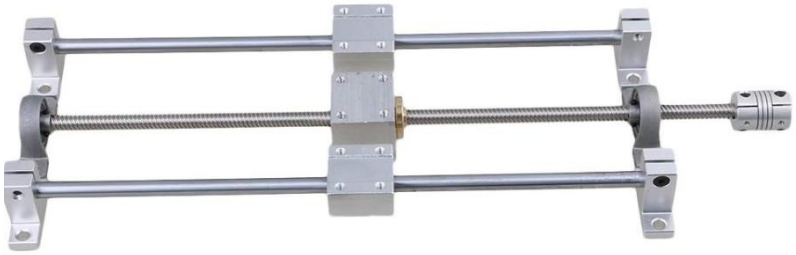


Soporte para motor



Acopladores y ejes

Acoplamiento del acoplador del eje del motor de 1PCS 5x8mm
Tuerca de asiento de tornillo 1PCS T8
1PCS 8 mm Tornillo de plomo 8 mm de diámetro 350 mm de longitud
2PCS 8mm Dia 300mm Longitud Cilindro Liner Linear Rail Eje Eje Óptico



Acopladores para eje del encoder 6-8mm

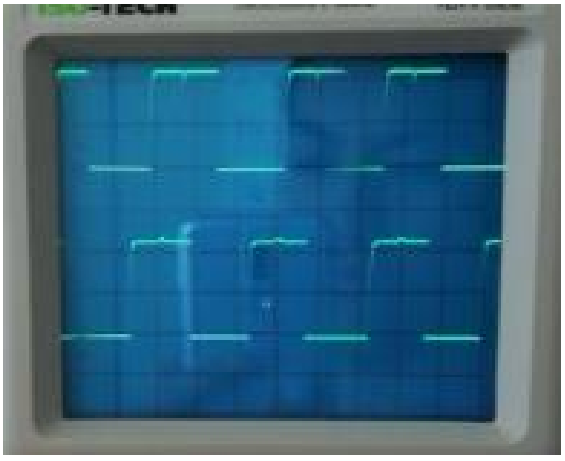
Material: Aluminio
Longitud: 25mm
Diámetro exterior: 19 mm

Pruebas de funcionamiento

Se hicieron pruebas para comprobar que componentes sensibles como el motor, encoder y el driver 8825 funcionaban correctamente o traían algún error de fábrica.

El motor y driver se testearon a la vez. Se fijó la corriente nominal en todos los drivers y se cablearon a una alimentación de 12 V y al motor en vacío. Se conectó a un generador de señales para comprobar el funcionamiento en frecuencia. Los motores respondieron bien a la prueba y además pudimos acotar las revoluciones máximas y mínimas a las que puede girar. Estas revoluciones se corresponden a un numero de pasos por segundo. Los motores funcionan entre 300 y 1300 pasos por segundo, fuera de esos valores empezamos a saltarnos pasos y el giro es incorrecto. Por otro lado los drivers demostraron ser elementos menos fiables ya que 2 de cada 10 probados no funcionaban. Llegamos a tener hasta 15 drivers.

Para el funcionamiento del encoder se hizo otra prueba. Se conecto el encoder al eje del motor y se alimento este a 1000 pulsos por segundo. Nuestro objetivo era observar con un osciloscopio si la respuesta del encoder era correcta pero sobre todo observar cual es la relación entre los dos canales para así determinar el sentido de giro a partir de la lectura del encoder. La respuesta fue la siguiente:



Lectura del encoder en sentido negativo, el canal 1 es el superior, el 2 el inferior

Esta lectura se corresponde con el encoder girando en sentido de las agujas del reloj o hacia atrás. El canal 2 es el que funciona con interrupciones mientras que el 1 es revisado cada vez

que hay un flanco de subida en el canal 2. Como observamos en la imagen, el nivel bajo en la lectura del canal 1 se corresponderá con un avance negativo (hacia el motor) y por tanto el nivel alto es un avance positivo. Solo se lee el canal 1 en los flancos de subida del canal 2.

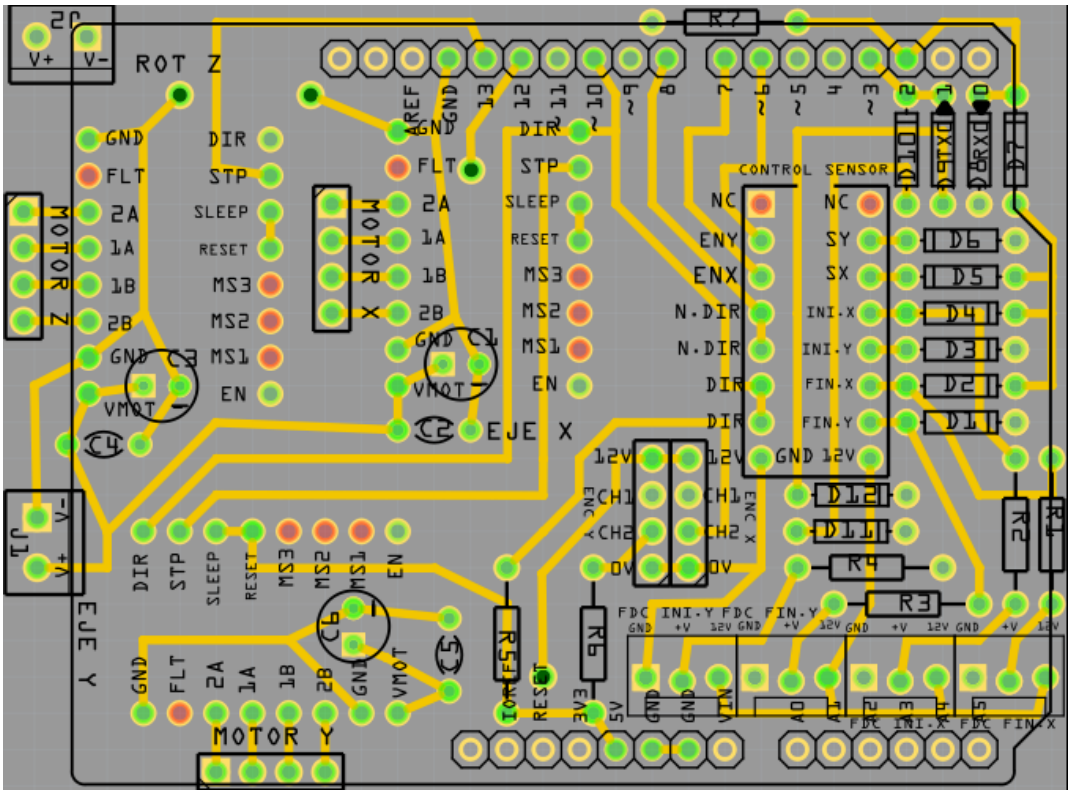
Anexo III: Planos

Listado de componentes			
Nº de plano	Designación	Cantidad	Material
1	Shield para Arduino	1	Fibra de vidrio
2	Anillo para encoder	2	Plástico
3	Plataforma para el eje Z	1	Plástico
4	Plataforma para tornillo sin fin del eje X	1	Plástico
5	Plataforma de arrastre del eje X	1	Plástico

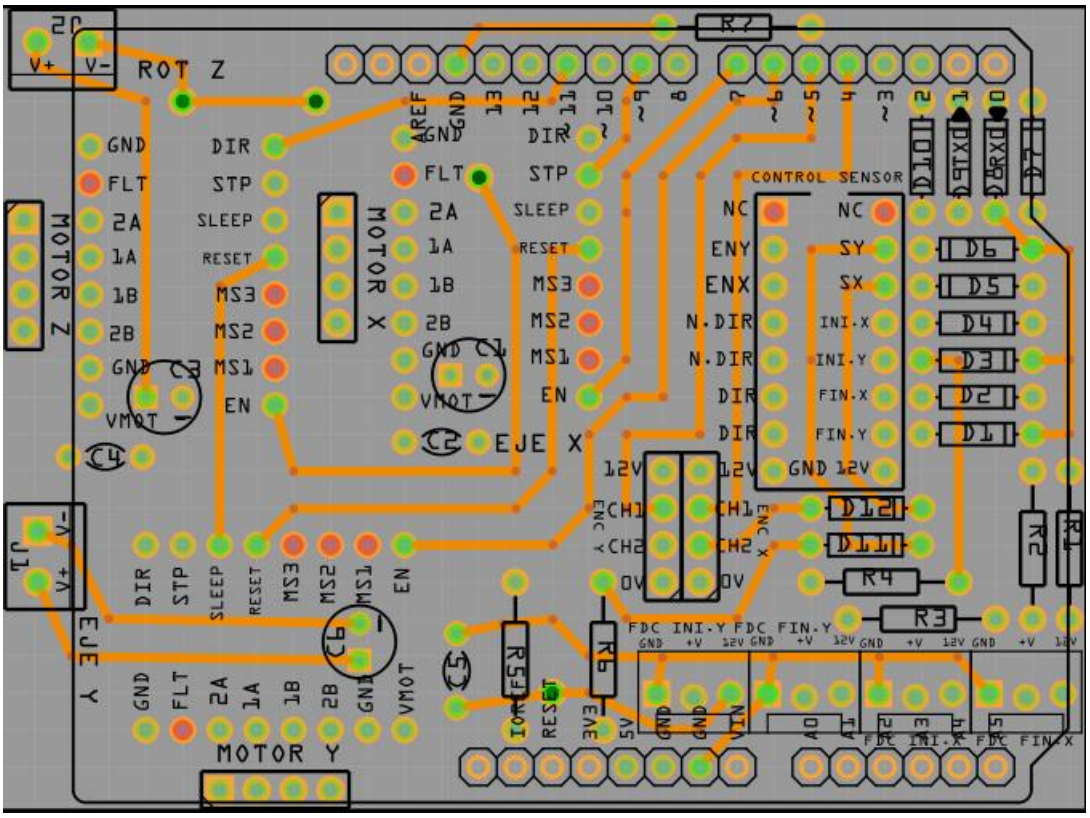
Lista de componentes electrónicos			
Nombre de componente	Designación en el plano	Cantidad	Observaciones
Arduino UNO	Arduino UNO	1	La shield se montará encima de este
Resistencia (10K)	R5, R6,R7	3	Servirán como pull-up y divisor de tensión para el final de carrera
Resistencia (15K)	R1, R2, R3, R4	4	Servirá como divisor de tensión para el final de carrera
Diodo 1N4148	D1,D2,D3,D4,D5,D6,D7,D8, D9,D10,D11,D12		Diodos para el circuito del ULN2003
ULN2003	IC1	1	Circuito selector de entrada
DRV8825	Motor_eje_X, Motor_eje_Y,Motor_eje_Z	3	Drivers de potencia del motor
Bloque de conexión	CNT 1-2	1	Conecta el circuito impreso a los 12V de la fuente
Tira de pines macho (Ref.diotronic: 40PW3 pines largos, 40PY en caso de pines cortos)	SV1, SV2,SV3	pines: 44	SV1,SV2,SV3 son 12 pines macho para el motor los otros 32 son para conectar la shield con el Arduino. No hubo suficientes así que se intercalaron cortos con largos en el montaje de las patas del circuito impreso.
Tira de pin torneado hembra (Ref.diotronic:SSQ36SSTG)	SSW1, SSW2, SSW3, SSW4, SSW5,SSW6	pines: 20	Escogimos pines hembra para la conexión de los sensores

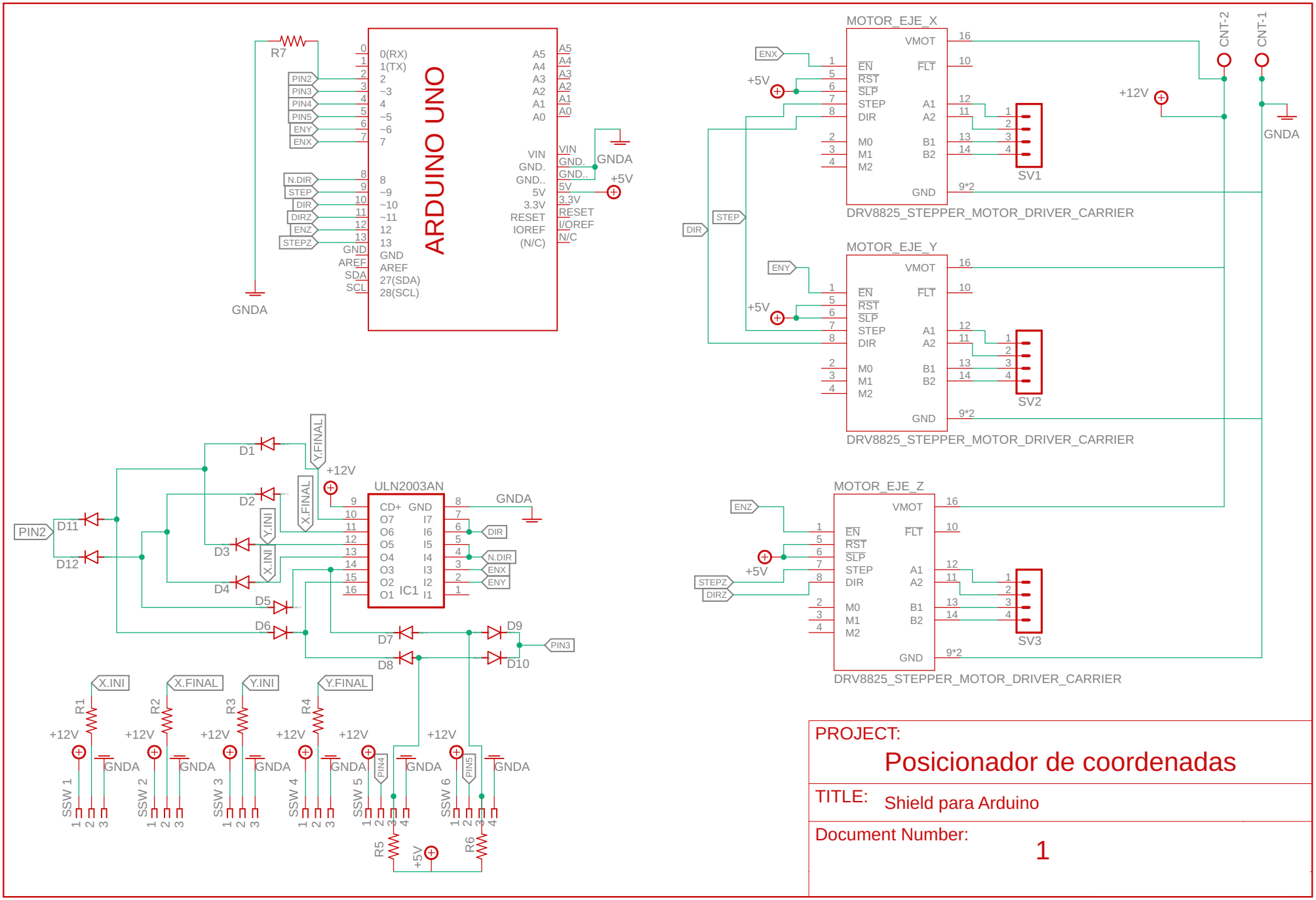
La disposición de los componentes en la placa es la siguiente:

CAPA SUPERIOR

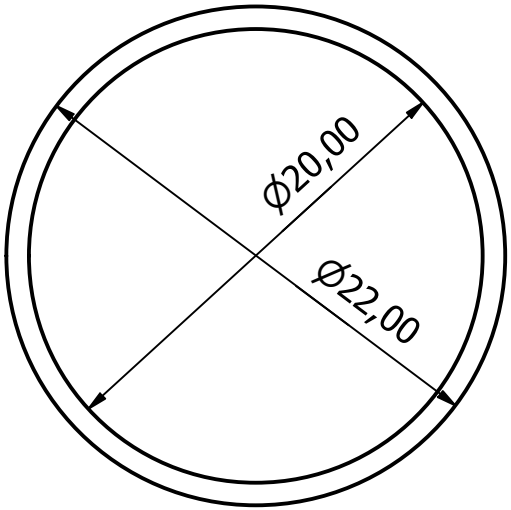


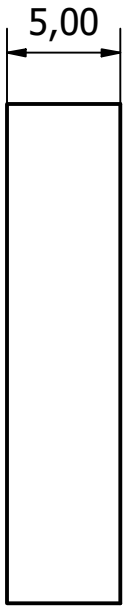
CAPA INFERIOR



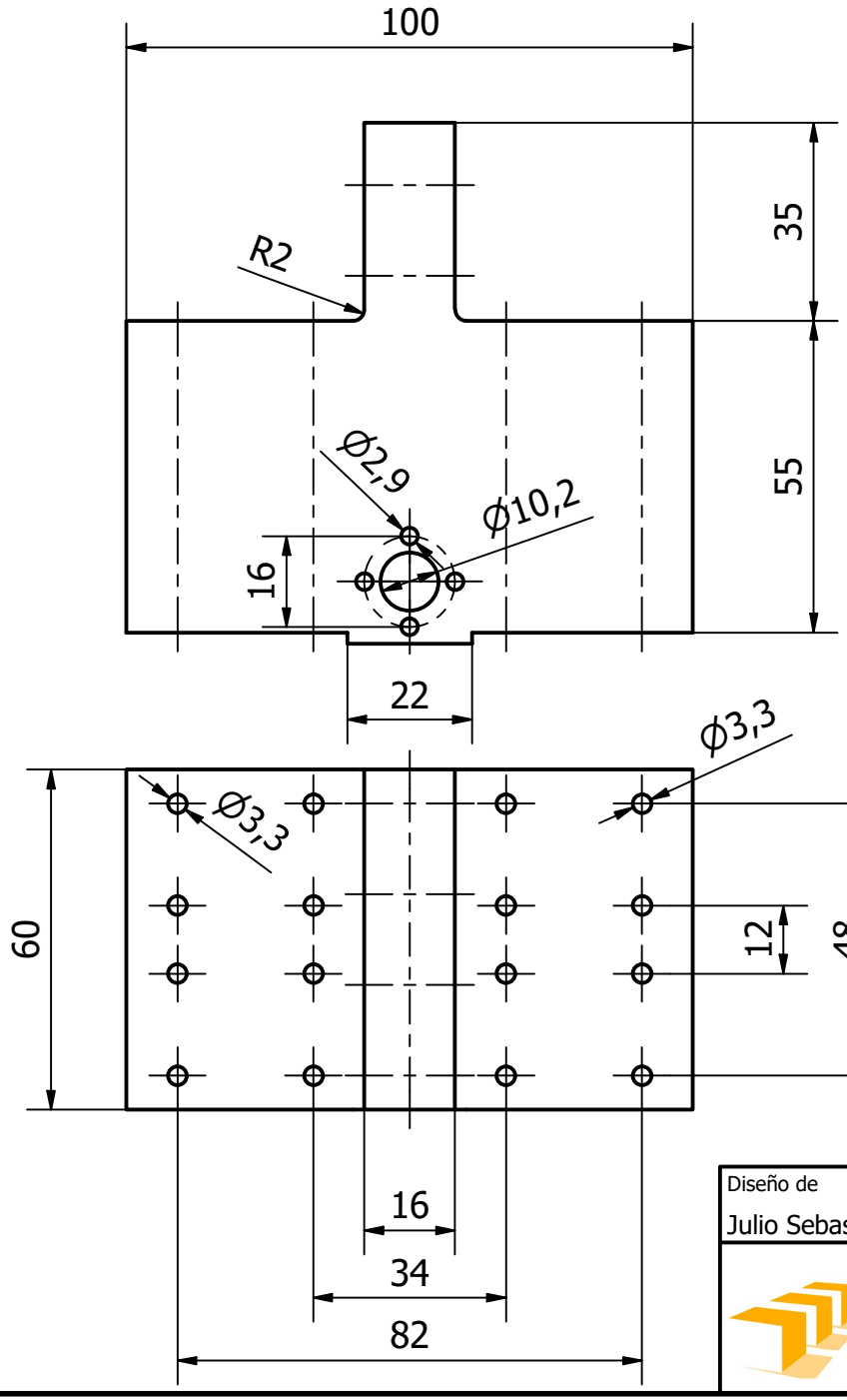


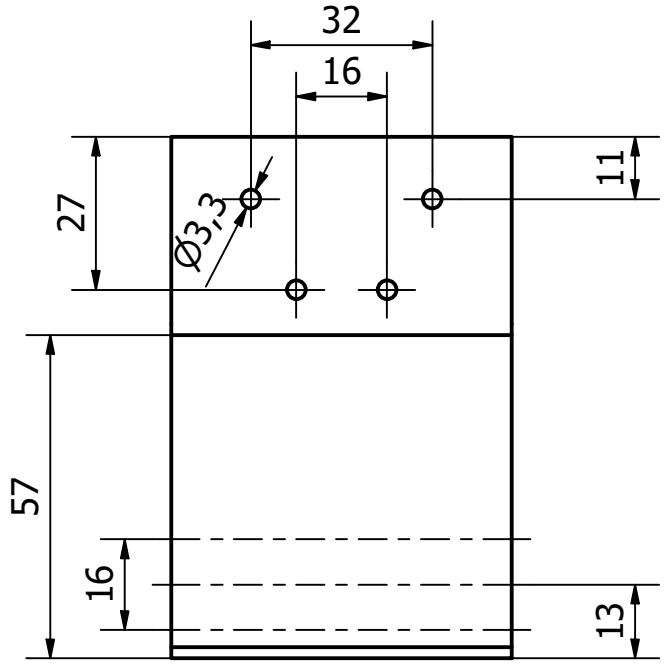
PROJECT:	Posicionador de coordenadas
TITLE:	Shield para Arduino
Document Number:	1



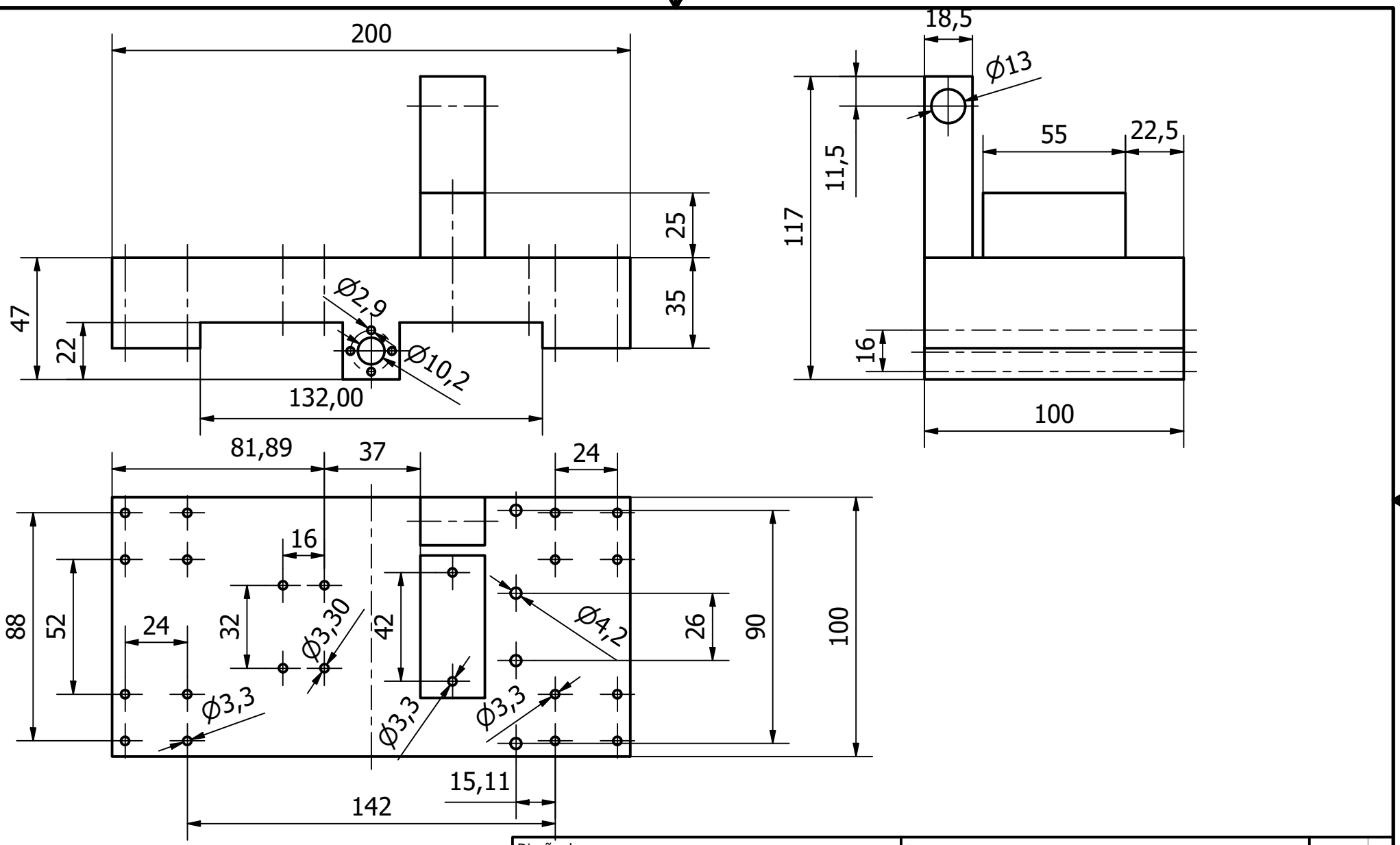


Diseño de Julio Sebastián Sullca Trillo	Proyecto: Posicionador de coordenadas
Escuela de Ingeniería y Arquitectura Universidad Zaragoza	Plano: Anillo para encoder
	Nº de plano: 2

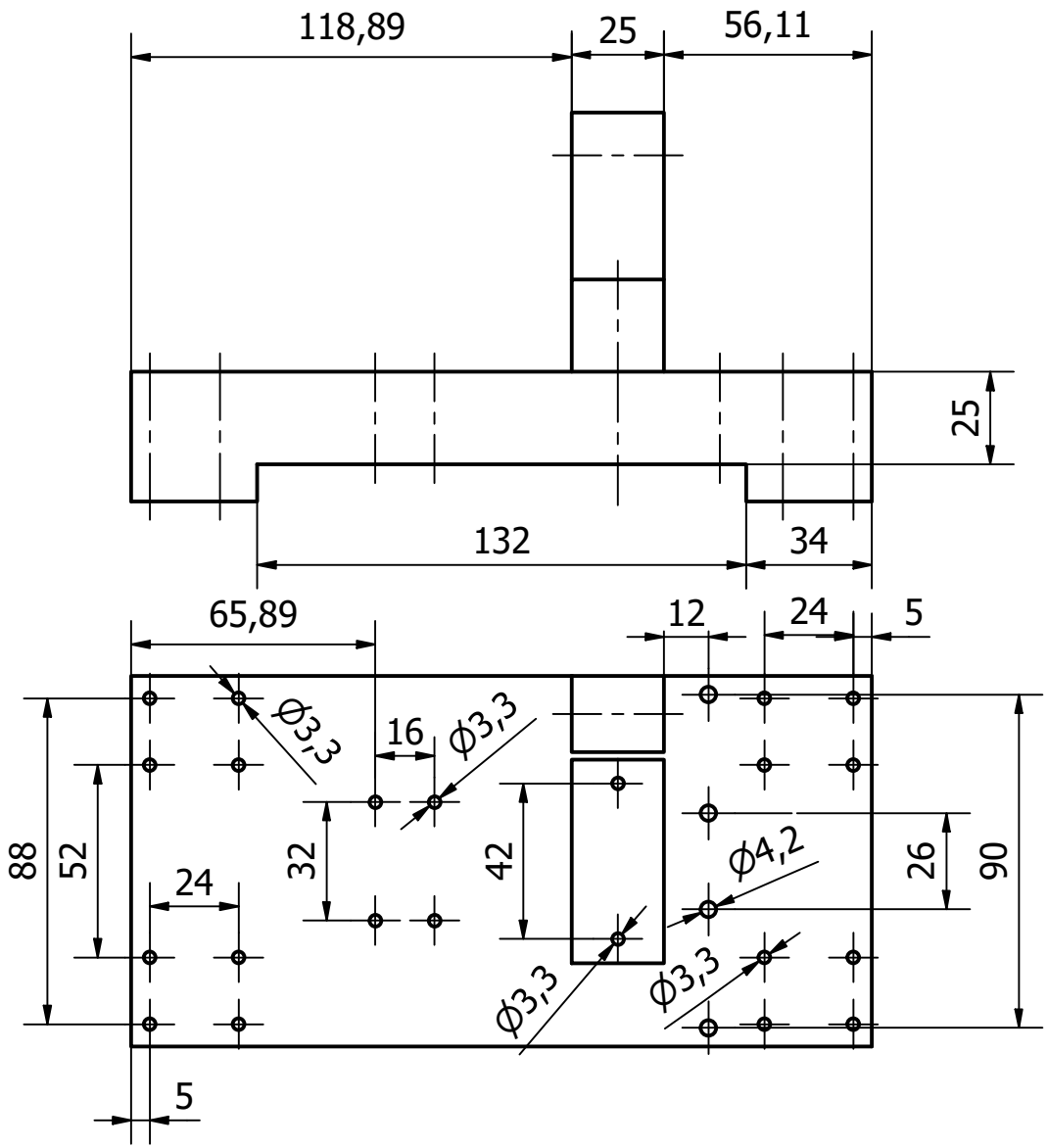


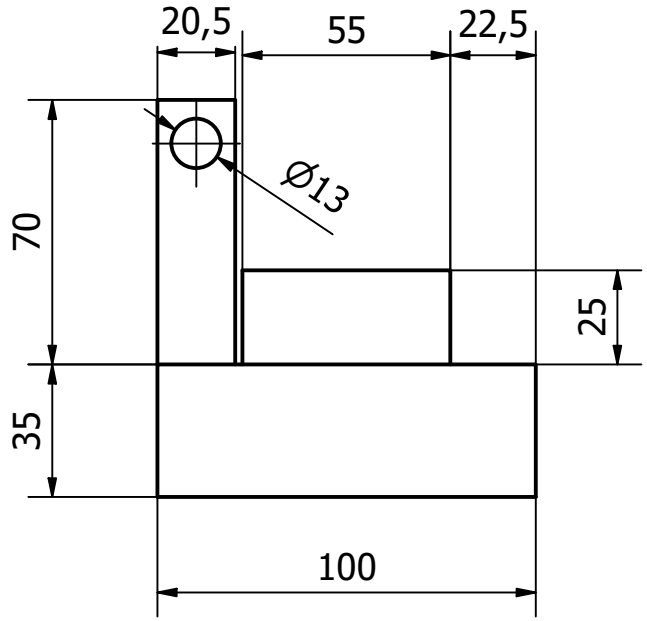


Diseño de Julio Sebastián Sullca Trillo	Proyecto: Posicionador de coordenadas
Escuela de Ingeniería y Arquitectura Universidad Zaragoza	Plano: Plataforma para el eje Z
	Nº de plano: 3



Diseño de Julio Sebastián Sullca Trillo	Proyecto: Posicionador de coordenadas
Escuela de Ingeniería y Arquitectura Universidad Zaragoza	Plano: Plataforma para tornillo sin fin del eje X
	Nº de plano: 4





Diseño de Julio Sebastián Sullca Trillo	Proyecto: Posicionador de coordenadas
Escuela de Ingeniería y Arquitectura Universidad Zaragoza	Plano: Plataforma de arrastre del eje X
	Nº de plano: 5