Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TRABAJO FIN DE GRADO

INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

# Control electrónico 2D para una plataforma móvil portaobjetos mediante micro-controlador de 32 bits

**AUTOR: Miguel Penadés Moltó**

**TUTOR: Rafael Masot**

**CO-TUTOR: Javier Ibáñez**

# Contenido

**Introducción** ..... 3

**Introducción al concepto de la Hipertermia** ..... 4

**1. Objetivo** ..... 6

**2. Diseño de Hardware.** ..... 7

**2.1. Sistema electrónico** ..... 10

**2.1.1. Tarjeta Discovery** ..... 10

**2.1.2. Finales de carrera (fotosensor)** ..... 11

**2.1.3. Driver de los motores** ..... 12

**2.1.4. Fuente de Alimentación.** ..... 13

**2.1.5. Motores bipolares paso a paso** ..... 14

**2.2. Resto del hardware** ..... 15

**2.2.1 Plataforma Móvil** ..... 16

**2.2.2 Cabezal láser.** ..... 17

**2.2.3. Placa Elisa** ..... 18

**3. Diseño del software** ..... 19

**3.1. Firmware** ..... 22

**3.1.1. Inicialización de las líneas de E/S del microcontrolador** ..... 23

**3.1.2 Lectura de los fotosensores** ..... 25

**3.1.3 Actuación sobre los motores** ..... 26

**3.1.4 Bucle principal** ..... 29

**3.1.5 Interrupción y recepción de datos** ..... 30

**3.1.6 Calibración de la plataforma** ..... 31

**3.2 Interfaz gráfica** ..... 32

**4. Resultados.** ..... 35

**5. Mejoras y sugerencias** ..... 39

**6. Conclusiones** ..... 40

**7. Presupuesto** ..... 41

**Anexo** ..... 44

**Anexo 1 Hojas de datos** ..... 44

**Anexo 2 Código** ..... 54

# Introducción

En el Instituto Interuniversitario de Investigación de Reconocimiento Molecular y Desarrollo Tecnológico (IDM) de la Universitat Politécnica de Valéncia (UPV) se está llevando a cabo un proyecto de investigación sobre como mediante técnicas de hipertermia óptica, es posible interactuar con células cancerosas.

El proyecto se basa en conseguir irradiar con un láser unas nanoparticulas que pueden ser insertadas de manera muy precisa y que al calentarse las vibraciones de estas consigan alterar el estado de dichas células.

Es aquí donde nace la necesidad de poder controlar el movimiento de una Placa Elisa de una manera totalmente automática y con un alto nivel de precisión, ya que el sistema se montará dentro de una caja cerrada donde es muy importante no causar alteraciones en la temperatura.

Dado que los profesores de la Universitat Politècnica de València (UPV) Rafael Masot y Javier Ibañez no consiguieron encontrar un sistema móvil que se adaptara a las necesidades que requería el proyecto, ya que la gran mayoría de sistemas de plataformas móviles automatizadas son empleadas en un ambiente industrial y no están pensadas para la investigación, decidieron que se debería implementar un sistema diseñado desde cero que consiguiera solventar este problema.

El Trabajo Fin de Grado (TFG) se basa en el control de una plataforma móvil que se mueve mediante dos motores bipolares paso a paso controlados por una tarjeta de microcontroladores de 32 bits y el diseño e implementación de una interfaz gráfica que permita a un usuario comunicarse y controlar el sistema de una manera que resulte simple e intuitivo.

Durante la realización de este trabajo he tenido que amoldarme a ciertas restricciones y trabajar ajustándome a ciertos parámetros impuestos por otras personas afines al proyecto de investigación sobre la hipertermia óptica. La tarjeta seleccionada para mover los motores y comunicarse con la interfaz debía ser la STM32f4 fabricada por la empresa STMicroeletronics ya que no solo iba a encargarse de estas dos tareas sino también del control del láser y la temperatura del sistema, que no forman parte de este trabajo, pero debían llevarse a cabo utilizando la misma tarjeta, ya que luego todo sería englobado en el proyecto de investigación mencionado anteriormente. También debía de trabajar teniendo en cuenta el diseño de la plataforma móvil, que ha sido diseñada por un alumno de ingeniería mecánica como parte de su trabajo final de grado. Estos factores han influenciado mucho en la realización del trabajo y han hecho que la comunicación entre todos fuera un aspecto fundamental de este.

A lo largo de esta memoria se mostrará el papel de cada uno de los elementos que forman el proyecto y se explicara de manera detallada el proceso de realización de un sistema de movimiento en un plano bidimensional totalmente automatizado y controlado por un usuario desde un ordenador a través de una interfaz gráfica de uso sencillo.

## Introducción al concepto de la Hipertermia

La hipertermia consiste en un calentamiento de los tejidos vivos por encima de su temperatura normal, bien por causas naturales o artificiales. El rango de temperatura de la hipertermia está definido entre 40 ºC a 48 ºC (Cherukuri, Glazer and Curley, 2010) y debe ser mantenido durante un determinado periodo de tiempo, para debilitar y conducir a las células enfermas hacia una muerte natural. Por encima de esta temperatura se producen otros efectos; superados los 50 ºC la sangre comienza a coagularse. Si dicha temperatura supera los 60 ºC se produce termoablación (necrosis y carbonización), (Jordan et al., 1999), (Chicheł et al. 2007).

En este proyecto, la hipertermia se realiza mediante la irradiación con un láser infrarrojo de nanoestrellas de oro. Gracias al efecto de resonancia de plasmón superficial, una longitud de onda adecuada permite la vibración de las nanopartículas, calentando el entorno.

Las nanopartículas (NPs) tienden a acumularse en los lugares donde existe una alteración en la vascularización del tejido. Esto ocurre en ciertas patologías concretas, como los tumores. Esta acumulación de nanopartículas se utiliza para realizar tareas de diagnóstico, al facilitarse la detección de masas tumorales mediante imagen de RM (Resonancia Magnética). También se usa con fines terapéuticos. Las NPs se pueden utilizar como vehículos para transportar fármacos o agentes citotóxicos a las zonas tumorales, así como para el tratamiento de tumores mediante técnicas de hipertermia. Las NPs suspendidas en una sustancia biocompatible, como por ejemplo el agua, se mantienen dispersas formando un coloide. Dicho coloide puede ser distribuido en los tejidos enfermos del paciente.

Conforme avanza la técnica, las terapias de hipertermia cada vez están más localizadas en la zona afectada. Una de las mejores contribuciones en esta evolución son las nanopartículas. Gracias a su reducido tamaño y a la capacidad de ser excitados con fuentes externas no invasivas, la hipertermia localizada alcanza su máxima precisión, pudiendo ser aplicada prácticamente célula a célula.

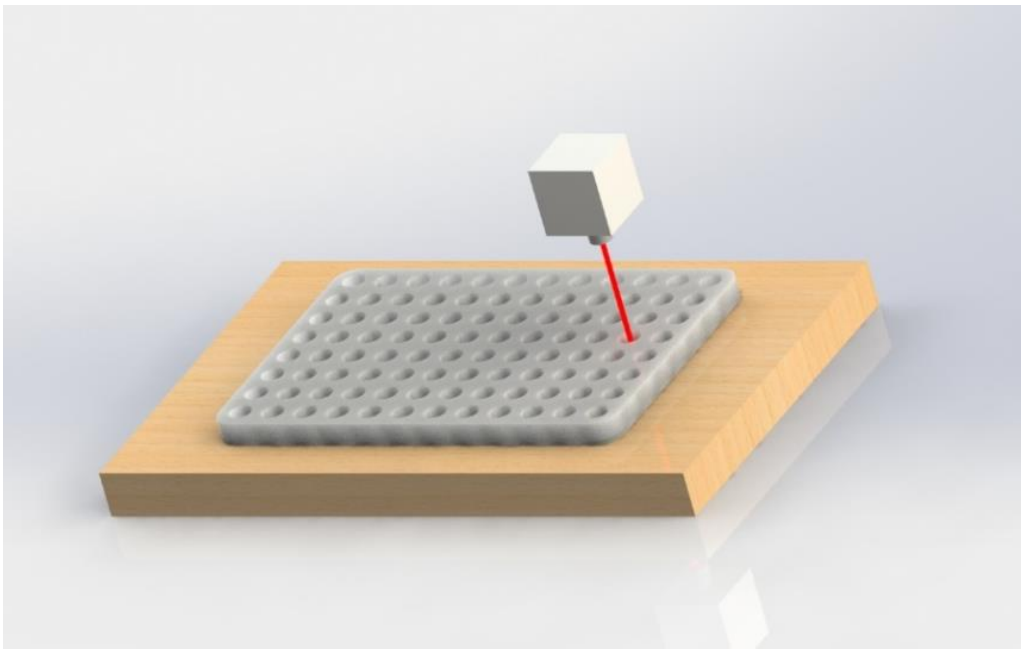


**Fig. Idea prototipo de cómo realizar pruebas sobre hipertermia óptica**

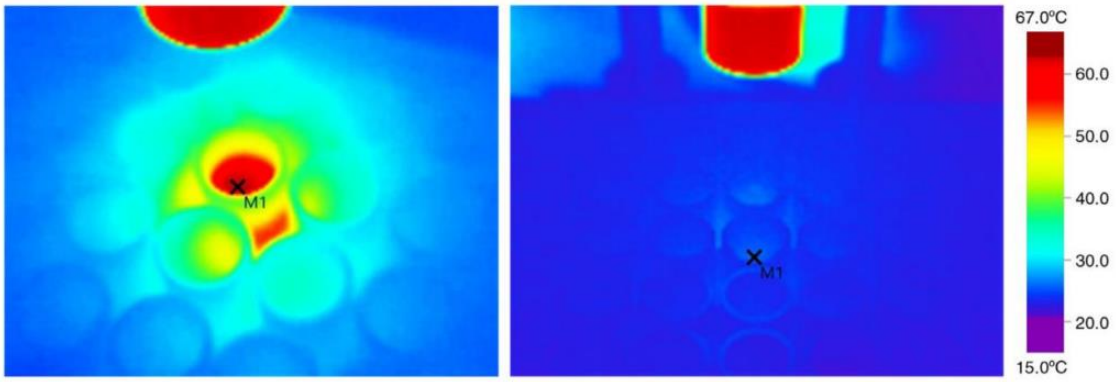



**Fig Imagen termográfica de la temperatura alcanzada tras 30 mins de exposición láser al coloide con nanoparticulas (izquierda) y al agua destilada (derecha)**

En la actualidad, en el equipo que dispone el IDM, el movimiento de la placa ELISA se hace de forma manual. Esto conlleva una serie de inconvenientes como por ejemplo poca precisión en el movimiento, así como variaciones térmicas en el interior de la cámara climática donde se aloja la placa.

# 1. Objetivo

Este trabajo está integrado en un proyecto de investigación sobre la lucha contra el cáncer de piel mediante hipertermia óptica basada en el comportamiento de nanopartículas tras ser irradiadas de una manera controlada que se está llevando a cabo en el Instituto Interuniversitario de Investigación de Reconocimiento Molecular y Desarrollo Tecnológico (IDM) de la Universitat Politècnica de València (UPV). El problema planteado fue, que era necesario poder controlar el movimiento en dos dimensiones de una plataforma móvil que soportara una placa ELISA de 96 pocillos, para que un láser situado en la parte superior, pudiera irradiar dentro del pocillo seleccionado a través de una interfaz gráfica en un ordenador.

La plataforma es capaz de moverse únicamente sobre dos railes, uno situado en el eje X y otro el eje Y, a través de dos tornillos sin fin que transmite el movimiento de dos motores bipolares paso a paso a un movimiento perpendicular a éstos.

Los motores han de ser controlados mediante el microcontrolador de 32 bits de la plataforma STM32F4 Discovery, que también se encargara de la comunicación con el P.C. a través de sus puertos UART.

El resultado final del proyecto será, por un lado:

- una interfaz gráfica que se comunicará con la STM32F4 donde el usuario pueda establecer la posición de la plataforma en la que el pocillo quede justo debajo del cabezal láser
- y por otro, el control de los motores paso a paso para obtener la resolución necesaria para poder acceder a los 96 diferentes pocillos de la placa ELISA. De esta manera un operario podrá seleccionar el pocillo que debe de ser irradiado y la plataforma se moverá de tal manera que dicho pocillo se sitúe justo debajo del cabezal del láser.

El software seleccionado para la programación en C de la STM32F4 ha sido el Atollic de la empresa STMicroelectronics y para el diseño de la interfaz gráfica y la comunicación con la tarjeta se ha utilizado el Matlab de MathWorks.

## 2. Diseño de Hardware.

Cuando empezó el proyecto de investigación mencionado anteriormente, la placa Elisa se movía manualmente y el pocillo que quería irradiarse se situaba debajo del cabezal laser con la precisión que un humano es capaz de alcanzar. Esto presentaba muchos inconvenientes ya que se conseguía muy poca precisión, la temperatura del coloide se veía afectada, ya que había que abrir la caja donde se había montado el sistema para poder mover la placa y se decidió que esto era demasiado ineficiente y que se debía implementar un sistema capaz de solventar este problema.

El problema más importante, que se pretende resolver, es que en el interior de los pocillos de la placa ELISA hay cultivos celulares vivos que requieren una temperatura estable entorno a los 36ºC. Por esta razón, la placa ELISA se encuentra dentro de una cámara climática. Cambiar de pocillo significa abrir la cámara climática y, por tanto, alterar las condiciones térmicas de su interior. Con el sistema automatizado, se puede cambiar de pocillo sin necesidad de manipular la cámara climática.

La solución fue diseñar y construir una plataforma capaz de mover la placa y automatizarla de tal manera que pudiera ser controlada desde un ordenador. La parte de diseño y ensamblaje de la plataforma se le fue encargada a un alumno de ingeniería mecánica de la Universitat Politècnica de València (David Cascales), y yo (Miguel Penadés) estudiante de ingeniería electrónica industrial y automática me encargué del control de los motores que moverán la plataforma y de las comunicaciones con el ordenador.

De esta manera ambas partes se unirían al final formando un único sistema, que consistirá en que el láser irradie el pocillo que seleccionemos en el ordenador si necesidad de tocar o mover ninguno de los elementos que lo forman una vez que haya sido instalado.

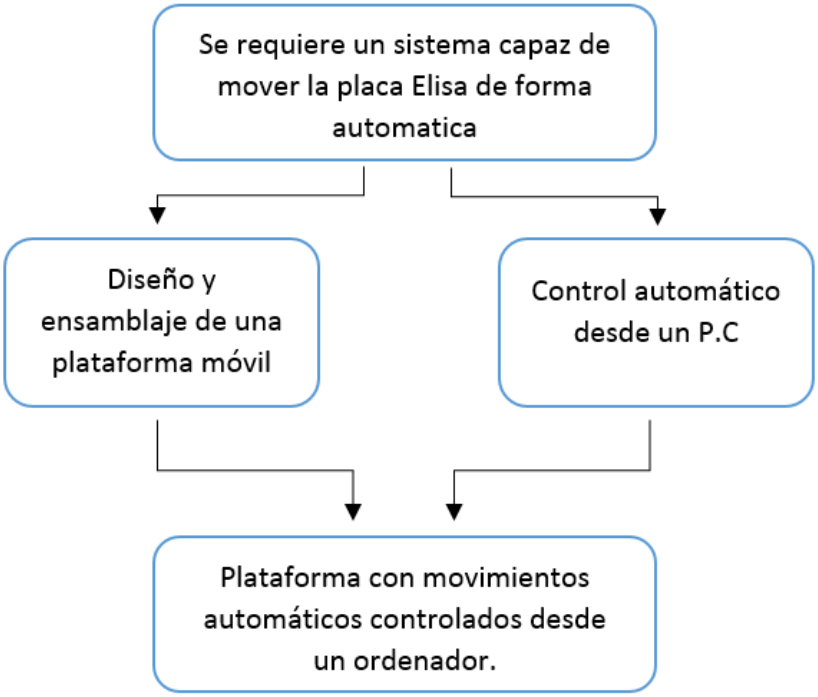



**Fig. Separación de la parte mecánica y electrónica del proyecto**

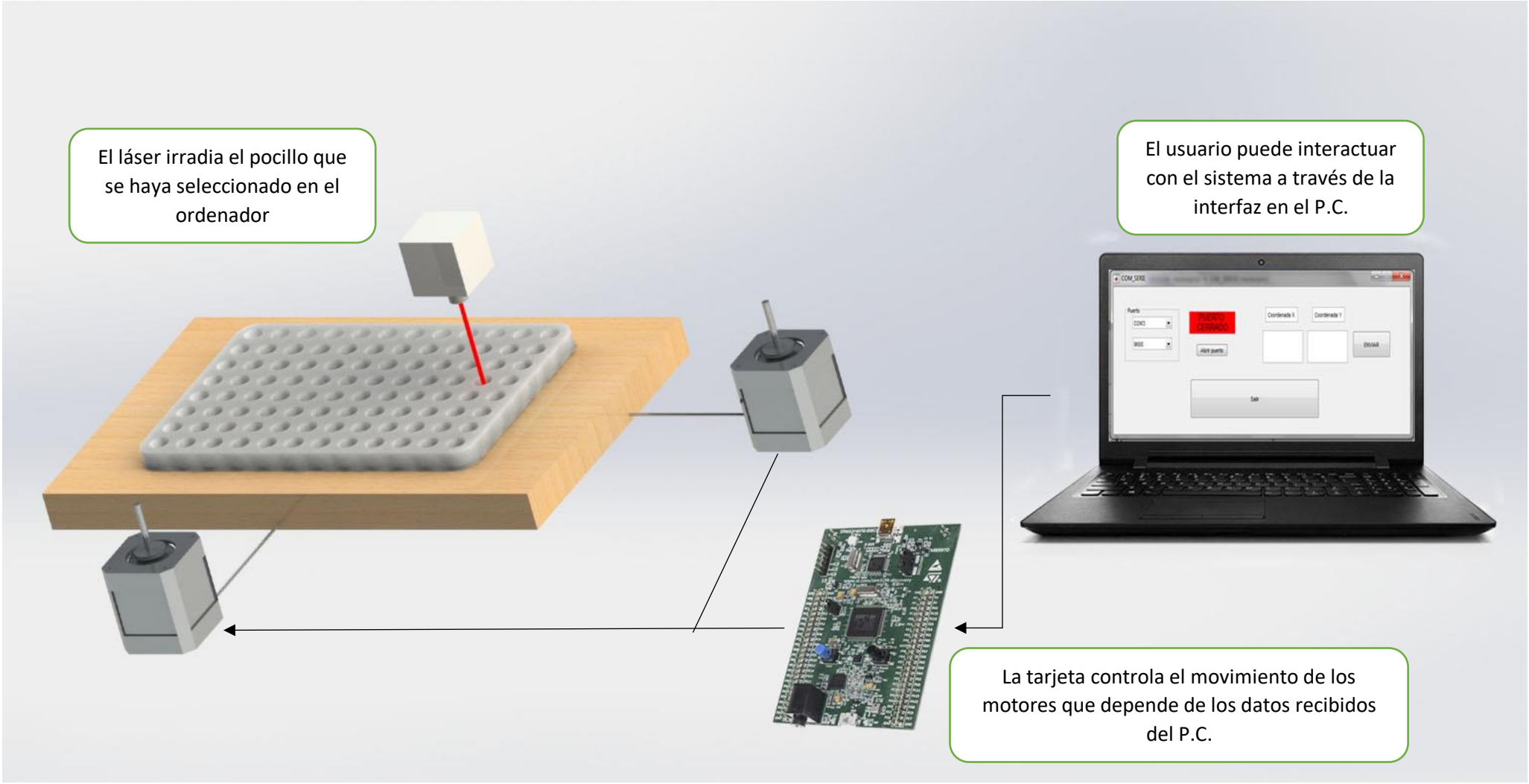
Los tres elementos más importantes a la hora de desarrollar este trabajo son: Los motores, la tarjeta de microcontroladores de 32 bits (STM32F4) y la interfaz gráfica que se desarrollará en el ordenador. El trabajo se basa en no solo comprender como funcionan cada uno de estos elementos, sino en establecer una comunicación de tal manera que simplemente escribiendo la coordenada sobre la que queremos situarnos, consigamos que la plataforma se mueva de forma totalmente automática.

El control de la plataforma móvil es la capacidad dirigir dicha plataforma de forma automática y se compone de estas tres partes.

- Mediante la interfaz enviaremos información a la tarjeta de donde queremos situarnos.
- La tarjeta recibirá esta información e implementara las funciones que controlan los motores y llevara el registro de sobre que coordenada está situado el laser
- Los motores moverán la plataforma situando la placa Elisa sobre la posición que se haya introducido en la interfaz.

Durante las primeras etapas, el diseño de la plataforma móvil no afecta al desarrollo del trabajo. Es durante la instalación de los finales de carrera y la calibración del sistema donde se ha de trabajar en conjunto.

En la siguiente página se muestra un esquema de cómo se ha planteado el trabajo y de qué manera interactúan los 3 elementos más importantes entre si.


El láser irradia el pocillo que se haya seleccionado en el ordenador
El usuario puede interactuar con el sistema a través de la interfaz en el P.C.
La tarjeta controla el movimiento de los motores que depende de los datos recibidos del P.C.

## 2.1. Sistema electrónico

### 2.1.1. Tarjeta Discovery

Uno de los requisitos impuestos a la hora de formar parte de este proyecto fue que el control de los motores debía hacerse con la STM32F4, ya que el cabezal laser y algunos elementos que forman parte del sistema electrónico, estaban ya siendo controlados con esta tarjeta. Dado su alto número de líneas de E/S, su bajo precio y alta accesibilidad para los alumnos de la Universitat Politècnica de València (UPV), ya había sido seleccionada con anterioridad a que se me propusiera el trabajo, así que para formar parte del proyecto era necesario que ya estuviese familiarizado con la tarjeta y su entorno de programación.

La STM32f4 Discovery es una tarjeta de microcontroladores de 32 bits de STMicroelectronics, con 1-Mbyte memoria flash y 192-kbyte RAM. También cuenta con una fuente de alimentación de 3V y 5V.

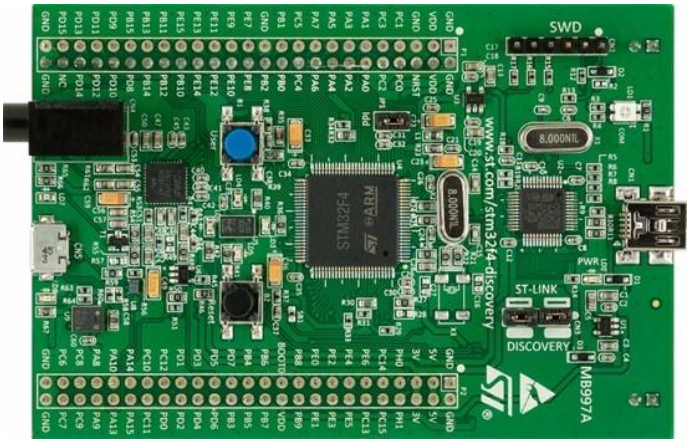



**Fig STM32f4 Discovery**

La placa STM32F4DISCOVERY ofrece las siguientes características:

- Microcontrolador STM32F407VGT6 con 1 MB de memoria flash, 192 KB de RAM, encapsulado LQFP100.
- ST-LINK/V2 incorporado con selector usar el kit como un ST-LINK/V2 independiente (con conector SWD para programación y depuración).
- Fuente de alimentación: a través del bus USB o desde una fuente de alimentación externa de 5V.
- Sensor de movimiento ST MEMS LIS302DL, acelerómetro con salida digital de 3 ejes
- Sensor de audio ST MEMS MP45DT02, micrófono digital omnidireccional
- Audio DAC CS43L22 con controlador integrado de altavoz clase D
- Ocho LEDs:
    - LD1 (rojo / verde) para la comunicación USB
    - LD2 (rojo) alimentación 3,3 V
    - Cuatro LEDs de usuario, LD3 (naranja), LD4 (verde), LD5 (rojo) y LD6 (azul)
    - 2 LEDs USB OTG LD7 (verde), VBus y LD8 (rojo)
- Dos pulsadores (usuario y reset)
- USB OTG con conector micro-AB

### 2.1.2. Finales de carrera (fotosensor)

Al no emplear sensores que detecten la posición de la plataforma móvil, es necesario que, tras la puesta en marcha, la plataforma siempre comience desde un mismo punto, es decir se debía establecer un origen de coordenadas. La solución fue crear un estado de inicialización en el que los motores girarían en sentido negativo hasta que la plataforma entrara en contacto con unos finales de carrera situados donde quisiéramos establecer el punto (0,0).

Primero se pensó en utilizar finales de carrera mecánicos convencionales, pero finalmente se optó por utilizar un fotosensor que detectara la llegada de la plataforma, que a pesar de que requería un montaje un poco más complejo, reducía errores al no haber rozamiento entre la plataforma y el propio final de carrera.

El fotosensor está formado por un diodo infrarrojo y un fototransistor que forman una U con una separación de 8mm, de esta manera cuando se introduce un objeto dentro de esta ranura cortara el haz de luz y el detector dejara de captar el haz de luz y se llevara a cabo la detección.


**Fig. Photomicro sensor**

Para conseguir un correcto funcionamiento del sensor aprovecharemos el pin que nos aporta 5V en la STM32f4 para la alimentación y le añadiremos dos resistencias formando el circuito mostrado en la siguiente figura. De esta manera conseguimos detectar cambios en el momento cerremos el paso al rayo de luz que cruza la ranura y la tarjeta lo interpretara como que la tabla ha llegado a su posición inicial.

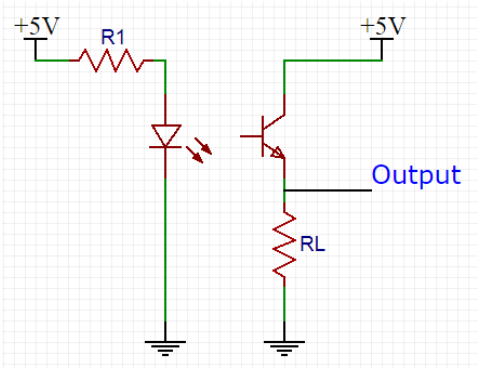



**Fig. Circuito empleado**

### 2.1.3. Driver de los motores

Para controlar el motor paso a paso es necesario utilizar un driver que consiga alimentar las bobinas con la corriente adecuada además de que los diodos actúan como controladores de la polaridad que se le aplica al motor.

Por ello fue seleccionado el LN298 un driver que es capaz de suministrar los 0.33A de corriente que necesitan los motores y es sencillo de utilizar.

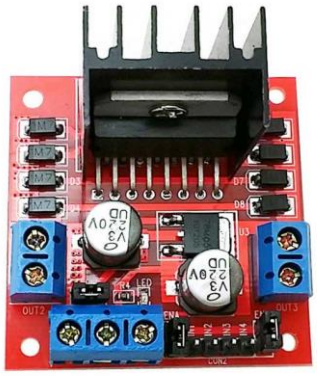


**Fig. Driver LN298**

A la hora de conectar el driver al motor es necesario conectar las bobinas del motor con los pines correspondientes de entrada al driver. Alimentamos con 12V ya que dado que los motores que hemos seleccionado tienen un alto par no es suficiente con la alimentación de 5V que nos aporta la STM32F4 y por lo tanto hemos empleado una fuente de alimentación externa.

Por último, los cables de salida que se conectaran a los 4 pines de la tarjeta de microcontroladores se cruzan entre sí para que el orden en el que se activen las bobinas sigan el orden de la tabla de verdad de la manera más lógica, más adelante en el apartado de la programación de los motores se entrara en detalle de su funcionamiento.

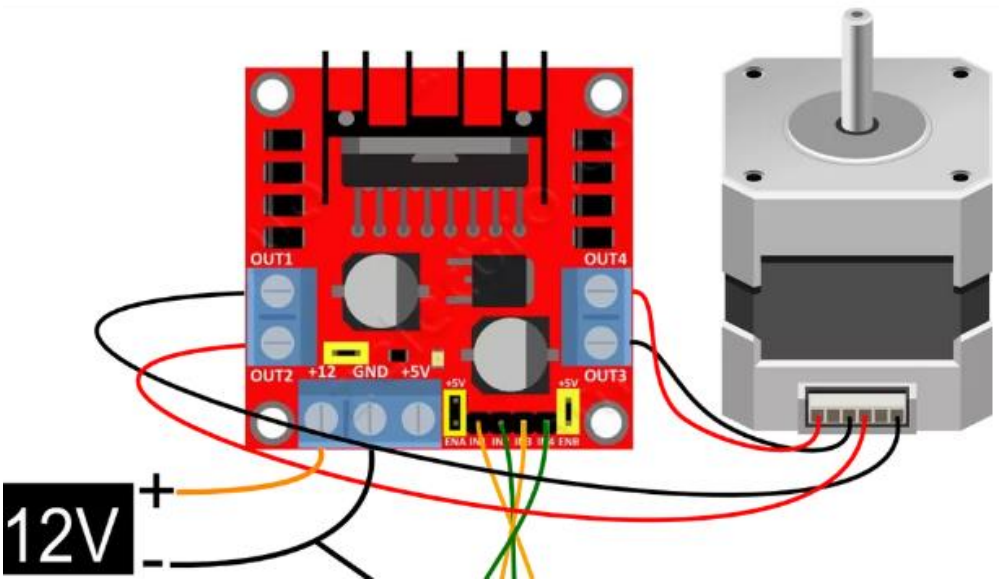



**Fig. Conexión entre el driver y el motor**

### 2.1.4. Fuente de Alimentación.

Al necesitar alimentar los motores con 12V nos vemos obligados a incluir en el proyecto una fuente de alimentación de 12V y 2A. Para poder contar con unos motores que tengan un par necesario para mover la plataforma sin ningún problema es un requisito que debemos incluir.

La fuente seleccionada es una fuente de tensión constante de 12Vcc con una carga máxima de 24W. La fuente de alimentación, está protegida contra cortocircuitos y sobrecargas, con una corriente de trabajo máxima de 2A.

Con estas especificaciones cumplimos los requisitos mínimos necesarios, ya que los motores tienen una corriente nominal de 0.33A así que nos es necesario una fuente de alimentación que nos proporcione una corriente mayor.


**Fig. Fuente de alimentación Serie Basic 12V 2A**

### 2.1.5. Motores bipolares paso a paso

A la hora de seleccionar los motores para el proyecto hubo que plantearse cuales eran los requisitos mínimos que tenían que cumplirse.

Tenían que tener suficiente par para poder mover la plataforma sin ningún tipo de dificultad, y la resolución tenía que ser lo suficiente para que pudiera moverse una distancia mínima del centro un pocillo al centro del siguiente en su misma fila o columna sin que haya perdida de pasos, para que se le pudiera ordenar hacer otro movimiento a continuación.

Con todo esto en mente se seleccionaron unos motores bipolares paso a paso de 5 mm de eje, con el único inconveniente que tenían que ser alimentados a 12V, por lo tanto, la alimentación de la STM32F4 no sería suficiente para moverlos y tendríamos que hacer uso de una fuente de alimentación externa que nos proporcionara esos 12V.


**Fig. Motor paso a paso**

Características:

- Angulo por cada paso (grados) :1.8º (200 pasos por revolución)
- 2 Fases
- Voltaje Nominal: 12V
- Corriente Nominal: 0.33A
- Par: 2.3kg*cm
- Diámetro eje: 5 mm

## 2.2. Resto del hardware

El resto del hardware se  ha diseñado por parte de otros alumnos que han trabajado en el grupo de investigación. Estas partes son:

Plataforma móvil, diseñada por: David Cascales

Cabezal láser, diseñado por: JoseManuel Terres

### 2.2.1 Plataforma Móvil

Dada la complejidad del cabezal laser y todos los componentes que lo forman se decidió que era la Placa Elisa la que debía moverse, en vez del propio cabezal. El alumno de ingeniería mecánica, (David Cascales), se encargó de diseñar y construir lo que denominamos, plataforma móvil como parte de su trabajo de final de grado (TFG).

La idea con la que se decidió trabajar finalmente se basa en dos sistemas de railes, uno encima del otro, para conseguir que la tablilla se pudiera mover libremente en el eje X e Y del plano horizontal. Para establecer el movimiento se empleó un mecanismo compuesto por dos tornillos sin fin, uno para cada eje, enlazado en sus extremos por los motores paso a paso y un retenedor que se encargara de fijarlos bien. El movimiento será estable ya que la plataforma solo podrá moverse por los dos railes que hay en cada eje.

La plataforma también cuenta con un espacio en la parte inferior donde ira colocada la tarjeta de microcontroladores con la placa con el circuito integrado con todos los componentes necesarios para el correcto funcionamiento del cabezal laser, las cámaras, los fotosensores que actúan de final de carrera y drivers empleados para los motores.

En resumen, la plataforma móvil está compuesta por tres pisos, el más elevado sostiene la tablilla con 96 pocillos, el de en medio asegura el movimiento en el eje horizontal y el piso inferior que asegura el movimiento en el eje vertical.

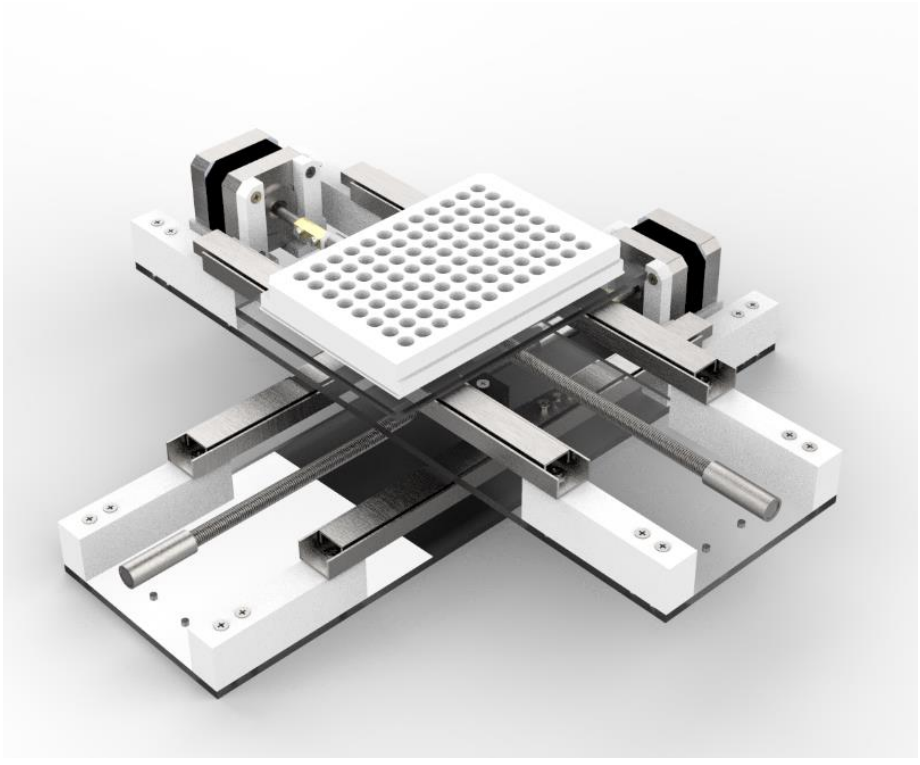


**Fig. Plataforma Móvil**

### 2.2.2 Cabezal láser.

El cabezal está compuesto por un láser, un refrigerador y dos cámaras; una térmica y otra común.

El láser es un láser de roithner, de 808 nm, 500 mW. Se seleccionó este porque las nanoparticulas que tienen que irradiarse corresponden a esa misma longitud de onda.

Fue necesario incluir una célula de peltier como refrigerador, ya que el láser se calentaba demasiado y alcanzaba temperaturas muy altas que podría haber afectado su funcionamiento.

Se decidió incluir también dos cámaras, una térmica para poder controlar la temperatura del contenido de los pocillos de una manera precisa y así saber cuándo encender o apagar el láser y otra común para determinar que la posición era la correcta en caso de que queramos comprobar si el sistema se ha descalibrado o si ha habido algún error de posición.

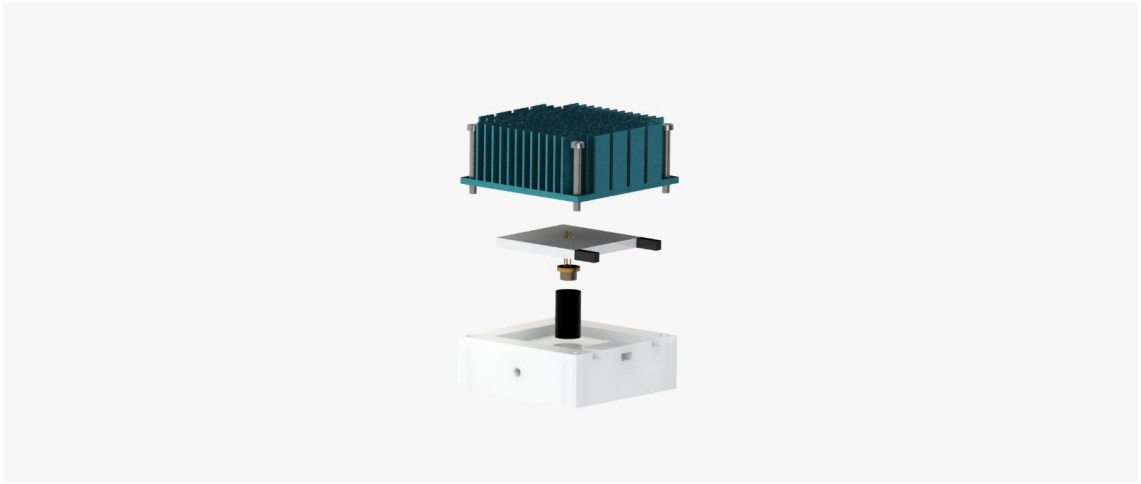


**Fig. Modelo del cabezal con el refrigerador**

### 2.2.3. Placa Elisa

Es una microplaca de 96 pocillos de plástico tratado, para aumentar su capacidad de adsorción (en su superficie) de moléculas, y con fondos de pocillo ópticamente claros.

En el eje X las columnas van numeradas desde la 1 a la 12 y en el eje Y las filas con letras desde la A a la H. De esta manera conseguimos denominar el pocillo que queramos de manera sencilla utilizando el sistema de coordenadas empleado en los gráficos con dos ejes.

Un ejemplo de esto es que si nos queremos referir al pocillo que se encuentra en la intersección de la quinta columna con la segunda fila lo denominaremos el pocillo (5,B).

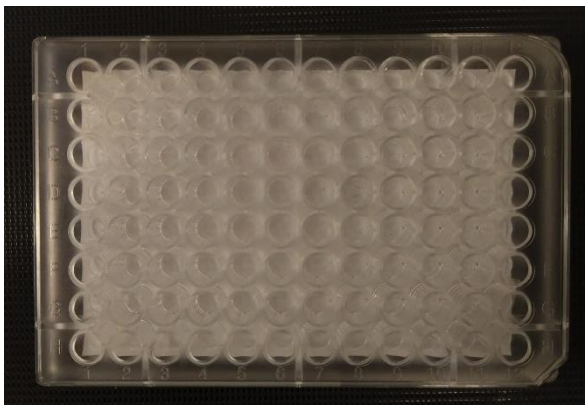


**Fig. Placa Elisa**

## 3. Diseño del software

El software no podía ser diseñado con total libertad ya que el tutor del trabajo estableció ciertos requisitos que debían cumplirse. Por ello antes de empezar a programar se diseñó la estructura que debía seguir el programa principal.

Al ponerse en marcha el sistema, se calibra y una vez terminado este proceso, debe de ser capaz de ejecutar ordenes una detrás de la otra sin necesidad de volver a pasar por esta etapa de calibración. Además de esto, no se hará uso de sensores de posición, así que el programa debía calcular distancias entre pocillos y almacenar las posiciones que iba alcanzando.

Si simplificamos el sistema podemos ver que se divide en tres etapas, y dos de ellas forman un bucle que termina cuando el usuario quiera dar por finalizada la sesión.

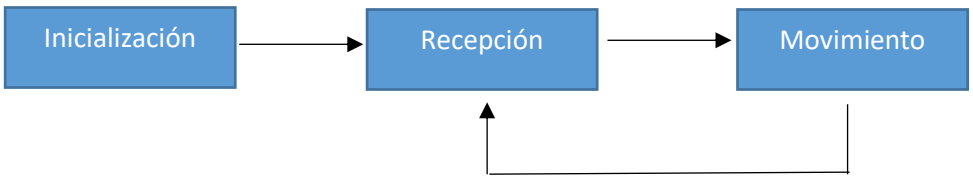



**Fig. Tres etapas principales**

Una vez establecidas estas tres etapas y su orden, podemos elaborar en que es lo que debe de hacer el sistema y que orden debe seguir:

- El sistema arranca y se posiciona en la coordenada 0,0. Para ello, activa los motores y queda a la espera de detectar una señal por la entrada correspondiente a los fotosensores.
- Queda a la espera de recibir la coordenada enviada desde la interfaz gráfica de usuario. Mientras no reciba datos a través del puerto UART el programa se mantendrá a la espera
- Una vez se haya recibido la coordenada deseada el programa calculara el número de pasos que el motor encargado de mover la plataforma en el eje X ha de dar para alcanzar esta posición destino.
- Si el número de pasos es positivo se aplicará la función correspondiente para que gire en el sentido deseado. Si es negativo, se ejecutará la otra función que hace que gire en sentido contrario.
- Se repite este mismo proceso, pero para el motor encargado de mover la plataforma en el eje Y.
- Se sobrescriben dos variables que determinan las coordenadas actuales para ser posteriormente utilizadas en el cálculo de los pasos de los motores.
- El programa vuelve al estado donde se mantiene a la espera de recibir una nueva coordenada destino y se repiten todos los pasos desde ahí indefinidamente hasta que se apague el sistema.

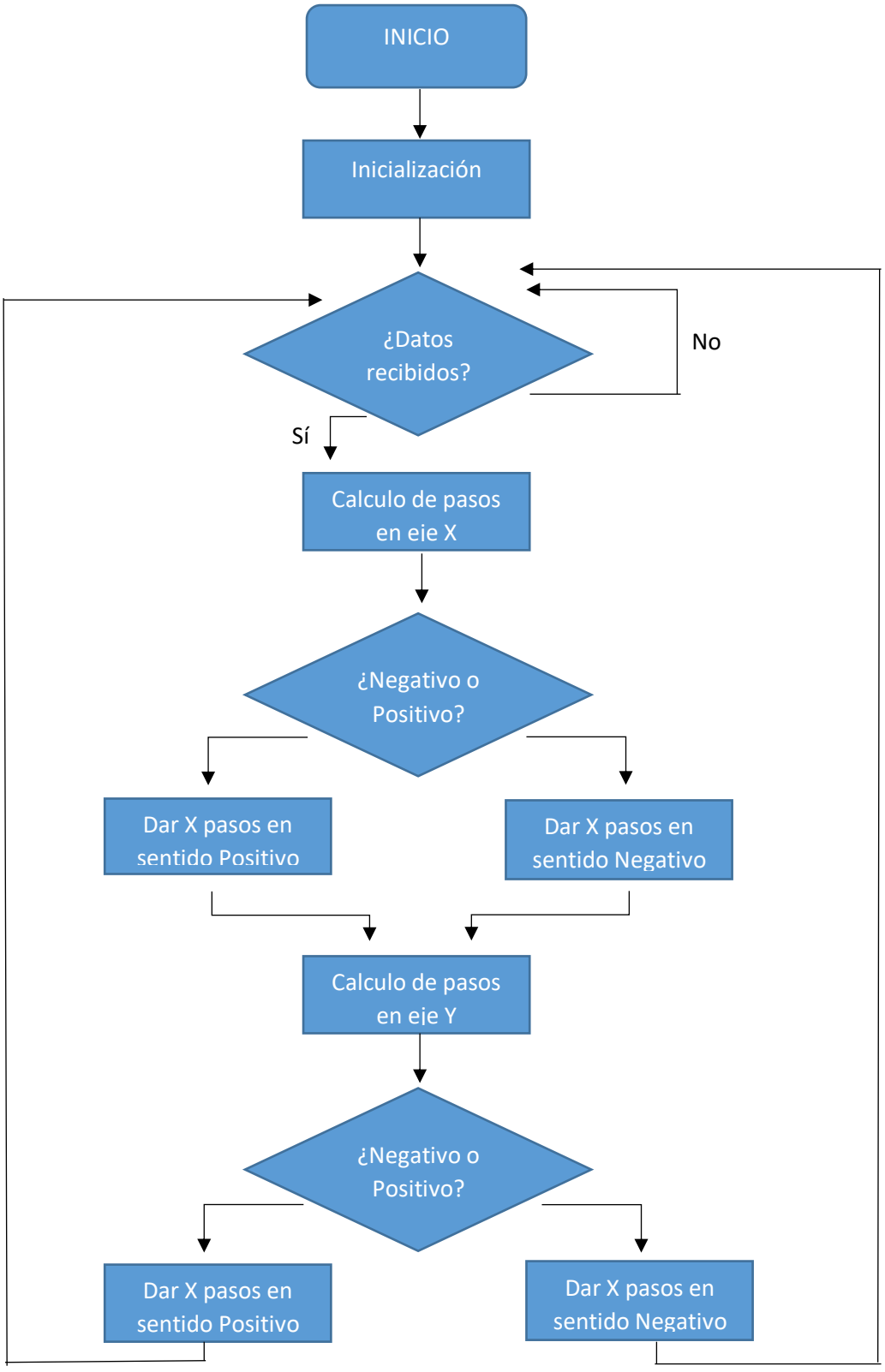



**Fig. Flujograma del programa principal**

Al poner en marcha el sistema los motores se pondrán en marcha moviendo la plataforma en sus dos ejes X,Y hasta que se activen dos fotocélulas que harán la función de finales de carrera. Cuando la plataforma alcance su posición inicial, en el ordenador se nos dará la opción de elegir qué pocillo queremos situar debajo del cabezal del láser. Las coordenadas X se darán en números y las coordenadas Y con letras ej. (4,B). A continuación, el motor encargado de mover la plataforma a lo largo del eje X se pondrá en marcha hasta llegar a su destino, una vez el movimiento haya terminado se repetirá el proceso, pero en el eje Y. Cuando se haya llegado a la posición deseada se almacenará como la posición actual y se nos volverá a dar la opción de elegir un nuevo destino sin necesidad de volver a la posición inicial.

Ha sido necesario utilizar 2 softwares diferentes, el Atollic studio para la programación de la STM32F4 y el Matlab GUIDE para el diseño de la interfaz y la comunicación con la tarjeta.

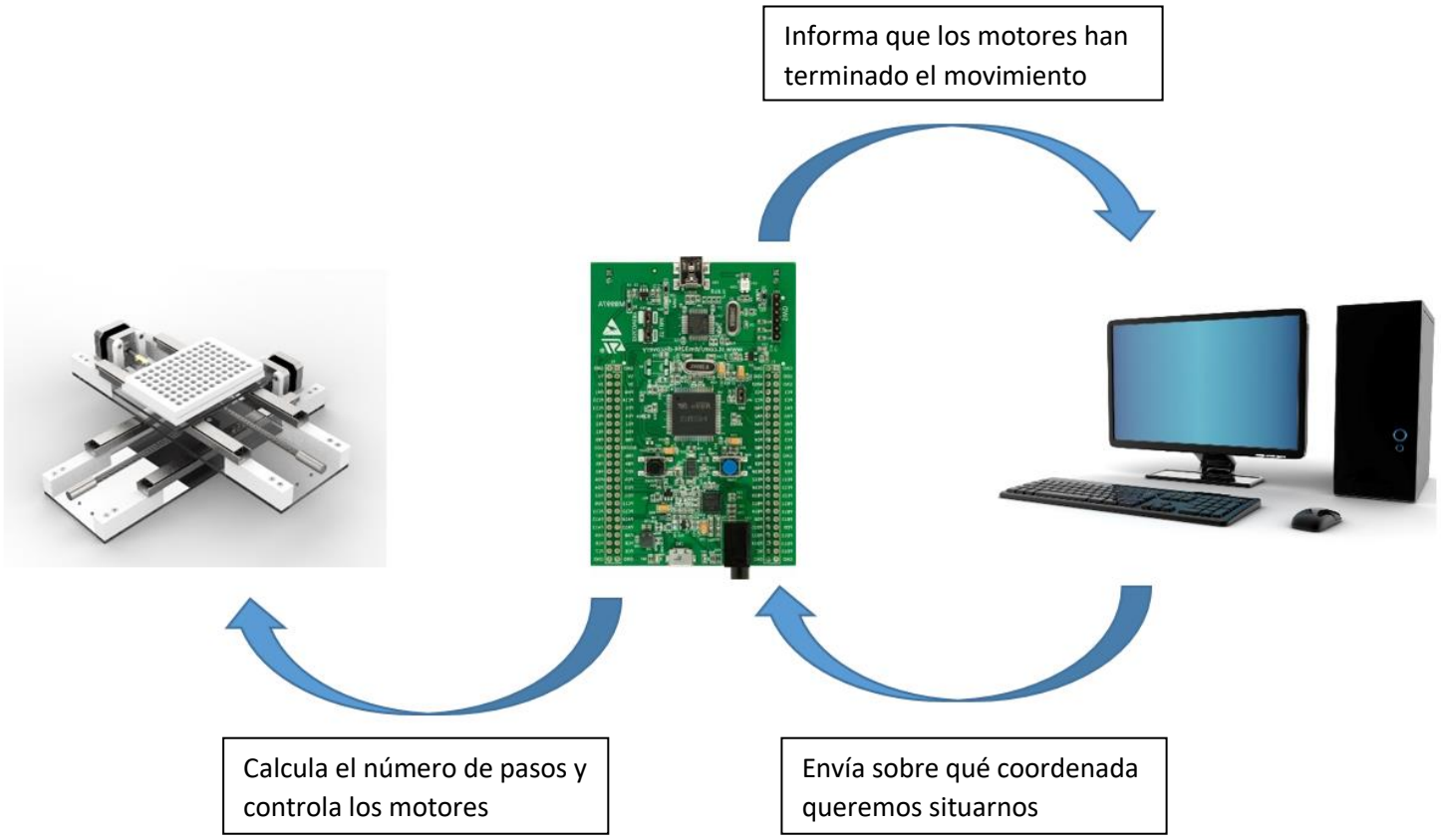



**Fig. Diagrama del flujo de información**

## 3.1. Firmware

El firmware del sistema se ha realizado mediante las siguientes etapas:

1. **Inicialización de las líneas de E/S del microcontrolador**
    - Las líneas que corresponden a cada uno de los elementos no han sido elegidas al azar y han de ser inicializadas con las características adecuadas para cada uno de ellos.

2. **Lectura de los fotosensores**
    - En esta etapa se explicará como el programa es capaz interpretar la detección de plataforma por los fotosensores e interpretarlo como la activación de un final de carrera.

3. **Actuación sobre los motores**
    - Para que los motores se muevan a la velocidad y en el sentido que queremos debemos de crear varias funciones que activen y desactiven las líneas asignadas a las bobinas en cierto orden y de esta manera conseguir un movimiento uniforme y controlado.

4. **Bucle principal**
    - Es un bucle sin fin donde se ejecutan de forma ordenada las distintas las funciones creadas en otros apartados del programa.

5. **Interrupción y recepción de datos**
    - Cada vez que la tarjeta detecte una entrada de datos hará uso de una interrupción para detener el programa principal y así poder interpretarlos como coordenadas destino y almacenarlos en variables para su posterior uso en el cálculo de pasos de motor.

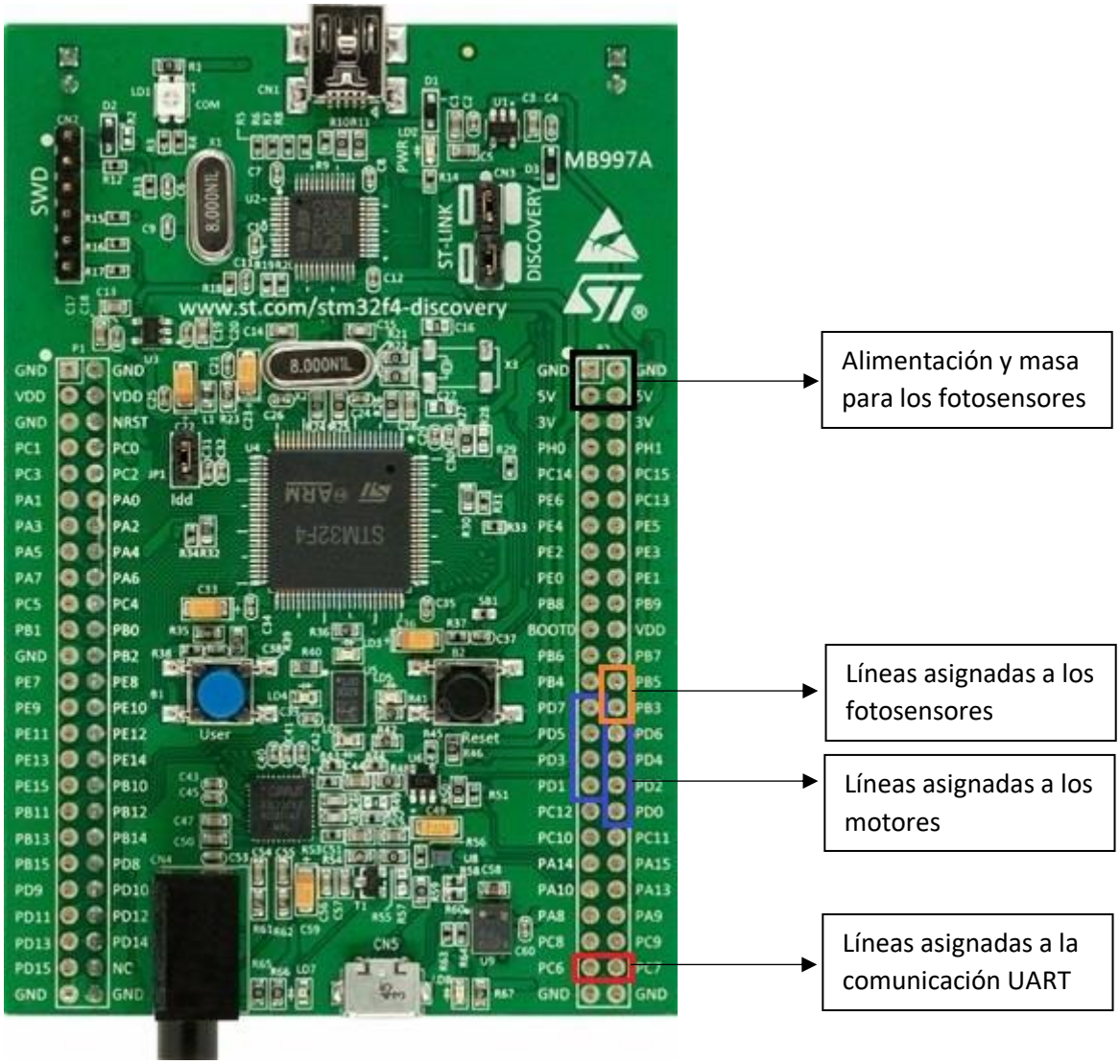
6. **Calibración**
    - Una vez el sistema electrónico se haya instalado en la plataforma móvil, hay que ajustar algunos parámetros como el punto inicial y la distancia entre pocillos para que la plataforma se mueva con la exactitud y precisión que queremos.

### 3.1.1. Inicialización de las líneas de E/S del microcontrolador

A la hora de utilizar la STM32f4 es importante que tengamos conocimiento de que las líneas de E/S que se vayan a utilizar van a tener que estar correctamente inicializadas. Esto conlleva seleccionar las líneas de tal manera que sean compatibles con la función que le queremos dar, ya que algunos solo son capaces de trabajar a ciertas velocidades o no servirán para ciertas funciones, como por ejemplo comunicación UART o simplemente como puertos de entrada salida.

Como la STM32F4 también se encarga de controlar el láser y otros componentes que han sido implementados por otros compañeros del grupo, las líneas fueron organizadas y no podían ser elegidas al azar.

En total era necesario utilizar: 8 líneas en modo OUT (4 para cada motor), 2 en modo IN para los fotosensores que actuarían de final de carrera, y por ultimo 2 más, una como IN, y otra como OUT para la comunicación con el PC.



Una vez seleccionados se llevó a cabo la programación de las líneas.

- El puerto GPIOB fue empleado para los fotosensores.

```
void GPIOB_Initialize(){
    GPIO_InitTypeDef GPIO_InitDef;

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
        GPIO_InitDef.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_5;
        GPIO_InitDef.GPIO_Mode = GPIO_Mode_IN;
        GPIO_InitDef.GPIO_OType = GPIO_OType_PP;
        GPIO_InitDef.GPIO_PuPd = GPIO_PuPd_DOWN;
        GPIO_InitDef.GPIO_Speed = GPIO_Speed_100MHz;
        GPIO_Init(GPIOB, &GPIO_InitDef);
}
```

**Fig Inicialización de las líneas del puerto GPIOB**

- El puerto GPIOD se empleó para los dos motores.

```
void GPIOD_Initialize(){
    GPIO_InitTypeDef GPIOD_Stepper;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    GPIOD_Stepper.GPIO_Mode = GPIO_Mode_OUT;
    GPIOD_Stepper.GPIO_OType = GPIO_OType_PP;
    GPIOD_Stepper.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7;
    GPIOD_Stepper.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIOD_Stepper.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIOD, &GPIOD_Stepper);
    }
```

**Fig Inicialización de las líneas del puerto GPIOD**

- Y finalmente el puerto C fue utilizado para la comunicación UART.

```
void GPIOC_Initialize(){
    GPIO_InitTypeDef    GPIO_InitStruct;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_USART6);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_USART6);
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOC, &GPIO_InitStruct);
    }
```

**Fig Inicialización de las líneas del puerto GPIOC**

### 3.1.2 Lectura de los fotosensores

El fotosensor en forma de U hará de función de final de carrera cuando se inicialice el sistema. Al conectar el sensor un haz de luz láser cruza la ranura que forman sus dos extremos. Cuando la plataforma llegue a la posición inicial un pequeño trozo de plástico cruzará por medio de la ranura cortando el haz de luz, en este momento el circuito que forma queda abierto y se lleva a cabo la detección.

Al iniciar el sistema, el motor encargado del movimiento en el eje X se pondrá en marcha, hasta que el fotosensor detecte que la plataforma está en la posición inicial, en ese momento se detendrá. Acto seguido se repetirá este proceso, pero con el motor en el eje Y.

Esto se consigue de manera sencilla simplemente utilizando una función while, que mientras no detecte que los pines asignados a los fotosensores se activen ejecutaran la función encargada de mover los motores en la dirección que digamos.

```
while(posiniX==0){
        if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_3)) {
            USART_SendData(USART6, 1);
            posiniX=1;
        }
        else{
            vuelta_motorX_neg();
        }
}
```

**Fig. Detección del sensor en el Pin B3**

### 3.1.3 Actuación sobre los motores

Una vez el sistema se haya inicializado y se haya alcanzado la posición inicial el operario podrá introducir que pocillo quiere que sea irradiado por el láser. Tras introducir la posición deseada esta se almacenará en dos variables distintas, una para la posición de X y otra para la posición Y, denominadas XDes y YDes respectivamente.

Cuando la tarjeta reciba datos del ordenador se activará una interrupción, el programa comparará las variables destino con la posición actual, que también se habrá almacenado en dos variables previamente y determinará la cantidad de pasos que debe dar cada motor para alcanzar la posición destino.

Se han implementado 4 funciones que determinan que motor y en qué dirección queremos que se muevan ej “vuelta_motorX_pos” para que el motor que se encarga de mover la plataforma a lo largo del eje X mueva la plataforma hacia la derecha. Estas cuatro funciones activan los pines que polarizan las bobinas con 10 milisegundos de diferencia. Cada vez que se activan 2 pines el motor da un paso, pero al no necesitar tanta resolución la función da 4 pasos seguidos, uno detrás de otra por lo tanto en el programa principal el motor solo podrá moverse de 4 pasos en 4 pasos.

Para implementar la función de forma correcta, debemos saber en qué orden debemos encender las bobinas, dependiendo de qué tipo de motor paso a paso estemos utilizando deberemos implementar una secuencia u otra para que el motor se mueva de forma fluida y constante, sin saltos ni tirones.

| Paso Simple | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

| Paso Doble | | | |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |

| Medio Paso | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |

| Bipolar | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |

**Fig Tablas de verdad correspondientes al tipo de motor.**

Como se ha mencionado con anterioridad, los pines 1,3,5,7 corresponden al motor encargado de mover la plataforma en el eje horizontal. Cuando queramos poner uno de estos pines a 1 sin necesidad alterar los otros pines deberemos conocer su correspondiente valor en hexadecimal.

| Pin | Bin | Hex |
|---|---|---|
| 0 | 0000 0001 | 1 |
| 1 | 0000 0010 | 2 |
| 2 | 0000 0100 | 4 |
| 3 | 0000 1000 | 8 |
| 4 | 0001 0000 | 10 |
| 5 | 0010 0000 | 20 |
| 6 | 0100 0000 | 40 |
| 7 | 1000 0000 | 80 |

**Fig Valores en Hex de los pines empleados para el motor del eje X**

El resultado de la función es que conseguimos que los pines estén en el estado que queremos que estén con una diferencia de tiempo que nosotros elijamos.

En el caso de la función “vuelta_motorX_pos” conseguimos que el motor rote en lo que hemos determinado como la dirección positiva, para ello por cada paso que de pondrá todos los pines a 0, acto seguido activara los pines que queramos activar, en este caso siguiendo el orden de la tabla de verdad. Una Vez haya dado los 4 pasos una variable que actúa de contador aumentara en 1 su valor, de esta manera podemos llevar cuenta de cuantos pasos ha dado el motor y detenerlo cuando se hayan dado los pasos que se hayan calculado que se tienen que dar hasta la siguiente posición.

```
void vuelta_motorX_pos(){
              GPIOD->ODR = GPIOD->ODR & 0xFF55;
              GPIOD->ODR = GPIOD->ODR | (0x0002 | 0x0020);
              Delay(3);
              GPIOD->ODR = GPIOD->ODR & 0xFF55;
              GPIOD->ODR = GPIOD->ODR | (0x0002 | 0x0080);
              Delay(3);
              GPIOD->ODR = GPIOD->ODR & 0xFF55;
              GPIOD->ODR = GPIOD->ODR | (0x0008 | 0x0080);
              Delay(3);
              GPIOD->ODR = GPIOD->ODR & 0xFF55;
              GPIOD->ODR = GPIOD->ODR | (0x0008 | 0x0020);
              Delay(3);
              i++;
                }
```

**Fig. Cuatro pasos del motor del eje X en sentido positivo**

Si queremos invertir el sentido del motor para que rote en la otra dirección es tan sencillo como invertir el orden en el que se activan las bobinas.

```
void vuelta_motorX_neg(){
              GPIOD->ODR = GPIOD->ODR & 0xFF55;
              GPIOD->ODR = GPIOD->ODR | (0x0008 | 0x0020);
              Delay(3);
              GPIOD->ODR = GPIOD->ODR & 0xFF55;
              GPIOD->ODR = GPIOD->ODR | (0x0008 | 0x0080);
              Delay(3);
              GPIOD->ODR = GPIOD->ODR & 0xFF55;
              GPIOD->ODR = GPIOD->ODR | (0x0002 | 0x0080);
              Delay(3);
              GPIOD->ODR = GPIOD->ODR & 0xFF55;
              GPIOD->ODR = GPIOD->ODR | (0x0002 | 0x0020);
              Delay(3);
              i++;
                }
```

**Fig Cuatro pasos del motor del eje X en sentido negativo**

Para la programación de las otras dos funciones restantes que controlar el segundo motor encargado en el eje vertical, “vuelta_motorY_pos” y “vuelta_motorY_neg”, se ha conseguido de la misma manera, pero utilizando los valores en hexadecimal correspondientes a los pines que se asignado a este segundo motor, siendo el 0,2,4,6 del puerto GPIOD.

| Pin | Bin | Hex |
|---|---|---|
| 0 | 0000 0001 | 1 |
| 1 | 0000 0010 | 2 |
| 2 | 0000 0100 | 4 |
| 3 | 0000 1000 | 8 |
| 4 | 0001 0000 | 10 |
| 5 | 0010 0000 | 20 |
| 6 | 0100 0000 | 40 |
| 7 | 1000 0000 | 80 |

**Fig Valores en Hex de los pines empleados para el motor del eje Y**

```
void vuelta_motorY_pos(){
              GPIOD->ODR = GPIOD->ODR & 0xFFAA;
              GPIOD->ODR = GPIOD->ODR | (0x0001 | 0x0010);
              Delay(3);
              GPIOD->ODR = GPIOD->ODR & 0xFFAA;
              GPIOD->ODR = GPIOD->ODR | (0x0001 | 0x0040);
              Delay(3);
              GPIOD->ODR = GPIOD->ODR & 0xFFAA;
              GPIOD->ODR = GPIOD->ODR | (0x0004 | 0x0040);
              Delay(3);
              GPIOD->ODR = GPIOD->ODR & 0xFFAA;
              GPIOD->ODR = GPIOD->ODR | (0x0004 | 0x0010);
              Delay(3);
              i++;
                }
```

**Fig. Cuatro pasos del motor Y en sentido positivo**

### 3.1.4 Bucle principal

Una vez la inicialización haya terminado y la plataforma este en la posición (0,0) el programa espera a que se active la interrupción. Una vez hayamos escrito la posición destino en la interfaz gráfica, y se la hayamos mandado a la tarjeta la variable “rec” estará a 1 y tendremos valores para “XDes” y “YDes”. A estos dos valores se les restara la posición actual y se multiplicara por el número de pasos que tiene que dar el motor para avanzar exactamente del centro de un pocillo al siguiente, de esta manera podremos saber el número de pasos exactos que hay entre el pocillo que actualmente se encuentra debajo del cabezal laser y el pocillo al que queremos que se situé debajo de dicho cabezal.

Estos valores se guardarán en las variables de “PasosX” y “PasosY”. Dichas variables primero se compararán a 0, ya que se puede dar el caso de que queramos mover la plataforma solo en eje horizontal o el vertical. Si se comprueba que efectivamente el valor es distinto de 0 se decidirá el sentido en el que el motor tiene que rotar viendo el signo de la variable, si es superior a 0 por lo tanto de signo positivo utilizaremos la función “vuelta_motorX_pos” para el eje x y “vuelta_motorY_pos” para el eje Y.

Una vez que la tarjeta haya ejecutado todo esto se repetirá la función seleccionada tantas veces como pasos tenga que dar y al terminar sobrescribirá las variables de “XAct” y “YAct” que corresponden a la posición actual de la tabla en el eje X e Y respectivamente con los valores que correspondían a la posición destino de ambos ejes. También se le enviara a través de la comunicación UART un 1 a la interfaz para indicarle que el movimiento de los motores ha terminado y se pondrán las bobinas a 0 para que no pase corriente a través de ellas y el motor no se caliente innecesariamente.

```
while (1) {
    if(rec==1){
        rec=0;
    PasosX= (XDes-XAct)*50;
    PasosY= (YDes-YAct)*50;

    if(PasosX!=0){
        USART_SendData(USART6, 0);
        if (PasosX > 0){
            while((i>=0) & (i<PasosX)){
                vuelta_motorX_pos();
                }
                i=0;
                XAct=XDes;
                PasosX=0;
                USART_SendData(USART6, 1);
                GPIOD->ODR = GPIOD->ODR & 0xFF55;
                GPIOD->ODR = GPIOD->ODR | 0x0000;
                Delay(1000);
```

**Fig. Código para cuando el valor de PasosX es positivo.**

### 3.1.5 Interrupción y recepción de datos

Cada vez que la tarjeta detecta que por su puerto de entrada recibe una transmisión de datos se detiene el programa principal y salta la interrupción. Dicha interrupción se ocupa de almacenar los valores introducidos en la interfaz que determinan la coordenada a la cual queremos dirigirnos y de darle un valor a una variable que le servirá al programa principal para saber que la interrupción se ha llevado a cabo y hay nuevos valores asignados a las variables de las coordenadas.

Desde la interfaz del PC nos llegan los datos de las coordenadas una detrás de otra y la coordenada Y que se introduce en forma de letra ya convertida a su valor numérico. Estos valores se almacenan en un array que acto seguido es separado en dos variables distintas. El primer valor del array es asignado a la variable "XDes" y el segundo a "YDes". Una vez ha terminado este proceso la variable "rec" se pondrá a uno y de esta manera el programa principal podrá saber que ha saltado la interrupción y que puede empezar con el cálculo que determina cuantos pasos han de dar los motores para alcanzar la posición deseada.

```
void USART6_IRQHandler(){
    if(USART_GetITStatus(USART6,USART_IT_RXNE)){
        //static uint8_t cnt=0;
        XDes = USART6->DR;
        arrayR[pos]=XDes;
        pos++;
        if(pos>1){
            pos=0;
            XDes=arrayR[0];
            YDes=arrayR[1];
            rec=1;
        }

    }
}
```

**Fig. Interrupción y almacenamiento de la posición destino.**

### 3.1.6 Calibración de la plataforma

Por cómo se ha programado el sistema la plataforma solo puede moverse un mínimo de cuatro pasos a la vez, ya que las funciones que se han implementado para el control de los motores ejecutan los cuatro pasos y luego se multiplican por el número que deseemos, moviéndose estrictamente en múltiplos de cuatro.

Como una vuelta del motor equivale a 200 pasos y el ejecutar la función equivale a cuatro, si ejecutamos la función en bucle cincuenta veces seguidas conseguimos que el motor gire exactamente 360º. Durante las primeras etapas del proyecto se utilizó cincuenta como el valor placeholder para hacer pruebas, para más adelante ser cambiado por la auténtica cantidad de pasos que ha de dar el motor para desplazar la plataforma desde el centro de un pocillo al centro del siguiente**.**

```
while (1) {
    if(rec==1){
        rec=0;
    PasosX= (XDes-XAct)*50;
    PasosY= (YDes-YAct)*50;
```

**Fig. Numero de pasos en una vuelta 50x4=200 pasos**

La distancia entre el centro de un pocillo al centro del siguiente es de 9mm, y es un valor estándar en las Placas Elisa. Como una vuelta de tuerca de la plataforma se traduce a 1mm de distancia recorrida esto quiere decir que el motor deberá dar nueve vueltas para que la plataforma recorra una distancia de 9mm.

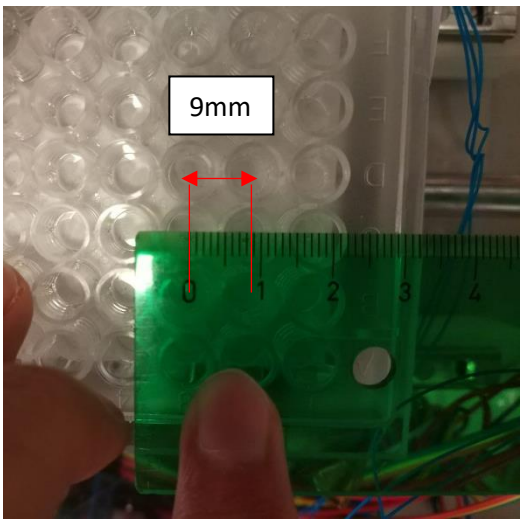



**Fig.**

La mínima cantidad de pasos que podrá entonces dar el motor es de 1800 pasos, es decir que solo podrá moverse de nueve vueltas en nueve vueltas. Una vez sabemos esto podemos multiplicar el valor placeholder que correspondía a una vuelta por nueve.

```
while (1) {
    if(rec==1){
        rec=0;
    PasosX= (XDes-XAct)*450;
    PasosY= (YDes-YAct)*450;
```

**Fig. Numero de pasos entre pocillo y pocillo 450x4=1800 pasos**

## 3.2 Interfaz gráfica

La interfaz se ha desarrollado mediante la herramienta GUIDE de Matlab por recomendación del tutor del trabajo, ya que es un software que los estudiantes del grado de ingeniaría electrónica industrial y automática de la UPV estamos familiarizados con, debido a que se trabaja con él en muchas de las asignaturas que impartimos. Es capaz de ejecutar todas las acciones que queremos de forma sencilla y es muy accesible ya que los estudiantes de esta universidad contamos con una licencia gratuita.

La interfaz se compone de tres apartados diferentes. Lo primero es seleccionar el puerto de comunicación y la velocidad a la que queremos transmitir la información. Esto se ha conseguido utilizando dos listas, que solo nos dejaran seleccionar las opciones que estén disponibles. Una vez hecho esto se pinchará en el botón de "abrir puerto" para establecer la comunicación y un letrero nos informará de el estado del mismo. Se mostrará rojo cuando no haya establecida una comunicación y verde cuando sí.

Acto seguido ya seremos capaces de enviar a la tarjeta las coordenadas que deseemos tantas veces como queramos.

Una vez hayamos terminado podremos pinchar en el botón de "Salir" donde una ventana emergente aparecerá en pantalla con la pregunta de si realmente estamos seguros de que queramos cortar la comunicación y cerrar el programa. Si seleccionamos que sí las ventanas se cerraran y se dará por finalizada la comunicación.

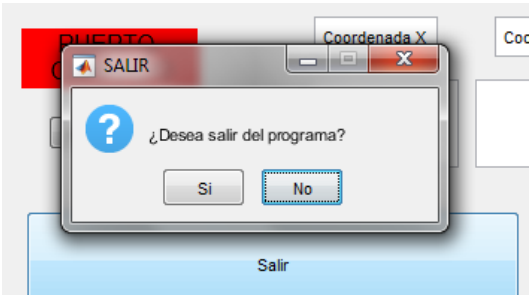



**Fig. Ventana emergente para cerrar el programa**

```
function cmdSalir_Callback(hObject, eventdata, handles)
global SerPIC

opc=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
if strcmp(opc,'No')
    return;
end
```

**Fig. Código para la ventana emergente "Salir"**

La Interfaz fue diseñada con el objetivo de que fuera lo más simple posible. Al principio se optó por un diseño más visual, donde se mostraría un diagrama de la tablilla con los 96 pocillos y el usuario pincharía con el ratón la posición donde quería que el cabezal laser se desplazara, pero finalmente se optó por dos cajas de texto donde se teclearían directamente las coordenadas X e Y. El motivo fue porque resultaba demasiado abrumador y confuso por la cantidad de pocillos que hay y también era posible equivocarse y pinchar al pocillo de al lado por la proximidad de estos. El resultado de la interfaz elegida es sencillo pero funcional.

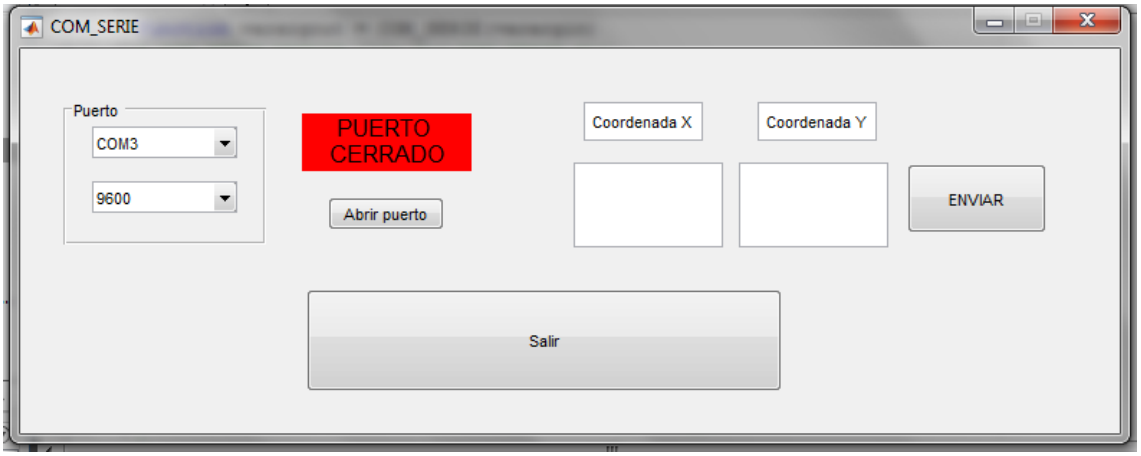



**Fig. Interfaz del programa**

Los dos botones que requieren de una programación un poco más compleja, son los botones de "Abrir puerto" y de "ENVIAR".

El botón "abrir puerto" dependerá de la velocidad de transmisión que se seleccione, y que se mantendrá a 9600 baudios/seg por defecto y el puerto que se seleccione, que podrá comprobarse una vez se haya conectado la tarjeta de microcontroladores a través del puerto USB en la ventana de "administrador de dispositivos" del P.C.

```
function cmdAbrir_Callback(hObject, eventdata, handles)
global SerPIC
puerto=handles.popCom.String;
puerto=puerto{handles.popCom.Value};
velocidad=handles.popVelocidad.String;
velocidad=str2double(velocidad{handles.popVelocidad.Value});
SerPIC = serial(puerto);
SerPIC.BaudRate=velocidad;
SerPIC.DataBits=8;
SerPIC.Parity='none';
SerPIC.StopBits=1;
SerPIC.FlowControl='none';
SerPIC.BytesAvailableFcnCount=1;
SerPIC.BytesAvailableFcnMode='byte';
SerPIC.BytesAvailableFcn={@Rx_Callback,handles};
fopen(SerPIC);

handles.lblEstadoPuerto.BackgroundColor=[0,1,0];
handles.lblEstadoPuerto.String='PUERTO ABIERTO';
```

**Fig. Código del botón "Abrir Puerto**

Como las coordenadas que insertemos han de ser un numero para el eje X y una letra en mayúscula en el eje Y, deberemos convertir esa letra a un valor numérico para que posteriormente cuando la STM35F4 reciba este valor, pueda aplicarlo a la fórmula que determina el número de pasos que debe de dar el motor. Esto se consigue convirtiendo este valor string en un double, y restándole el valor 64, que es el número que debemos restar al carácter A según la tabla de caracteres Ascii para que corresponda con el número 1, siguiendo esta norma la letra A corresponderá al número 1, la letra B al número 2 y así sucesivamente. De esta manera conseguimos que el usuario inserte la letra A en la interfaz, pero la tarjeta lo interpretara como el pocillo de la 1 columna.

```
function tglMuestrear_Callback(hObject, eventdata, handles)
global x y z contador Temporizador
global SerPIC dato
if (hObject.Value==1)
    dato=double(handles.coordY.String);

    fwrite(SerPIC,str2double(handles.coordX.String));
    fwrite(SerPIC,(dato-64));

    contador=0; x=0; y=0; z=0;
    handles.radiobutton1.BackgroundColor=[0,1,0];
    handles.radiobutton2.BackgroundColor=[0,1,0];
else
    handles.tglMuestrear.String='Enviar';
end
```

**Fig. Código botón "Enviar"**

Por último, para prevenir que el operario pudiera introducir los datos de otras coordenadas mientras que los motores estaban en funcionamiento se desarrollaron unas líneas de código que deshabilitarían el botón de enviar mientras aun hubiera movimiento.

Esto es posible ya que la STM32f4 está programada para enviar un 0 al iniciar el movimiento de los motores y un 1 al terminar. De esta manera cuando en el ordenador se recibe ese 1 a través de la comunicación UART deshabilita el botón de enviar, mostrándolo en la interfaz de un color más grisáceo y transparente.

De esta manera nos aseguramos que el sistema no se descalibre una vez iniciada la calibración durante la puesta en marcha.

```
function Rx_Callback(hObject, eventdata, handles)

global SerPIC contador x y z dato
%num=SerPIC.BytesAvailable;
%if num>0
    dato=fread(SerPIC,1);
    if dato == 0 handles.tglMuestrear.Enable='off';
        if dato == 1 handles.tglMuestrear.Enable='on';
```

**Fig. Código de habilitación y deshabilitación del botón enviar**

# 4. Resultados.

Cuando aún no se había terminado la fabricación de la plataforma móvil las pruebas se hacían simplemente con los motores sin estar acoplados a nada y los resultados eran los esperados. Los motores giraban de forma uniforme y la cantidad que se les exigía. Además, utilizando el modo debug del Atollic podíamos ver las variables en tiempo real, de esta manera podíamos ver si las vueltas que daba el motor correspondían con la hipotética posición que se suponía que se estaba colocando sobre.

El único contratiempo al que nos enfrentamos fue que al iniciar el sistema el motor daba un paso en la dirección contraria a la que debía girar, volviéndose hacia la otra dirección en una fracción de segundo y girando con normalidad. Esto parecía que iba a dar problemas ya que, al saltarse un paso, después de muchos movimientos el sistema se acabaría descalibrando. Después de todo, esto no resulto ser el caso ya que, aunque no se pudo detectar ni solucionar este error, solo se perdía este paso al iniciar el sistema del todo y acto seguido comenzaba la etapa de inicialización, donde el motor gira indefinidamente hasta situarse sobre el origen cuando la tarjeta detecta una señal proveniente del fotosensor. El que se diera un paso extra no afectaba en absoluto en este momento, ya que en este instante todavía no se había calibrado el sistema.

Una vez la plataforma móvil estuvo lista, se le acoplaron los motores y el láser, y se unió el software que controla a ambos en la misma tarjeta. Se instaló todo lo necesario y se hicieron las pruebas, ahora sí, con el sistema real.

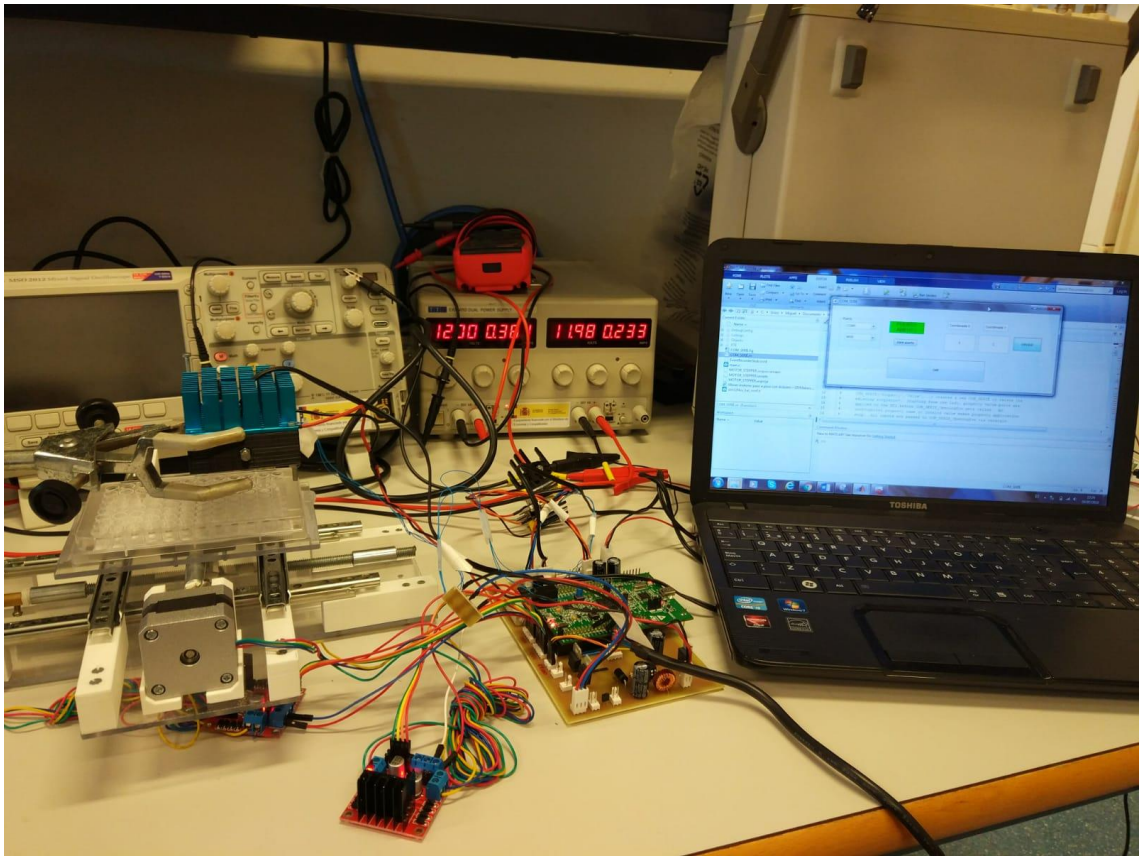



**Fig. Montaje para la realización de pruebas**

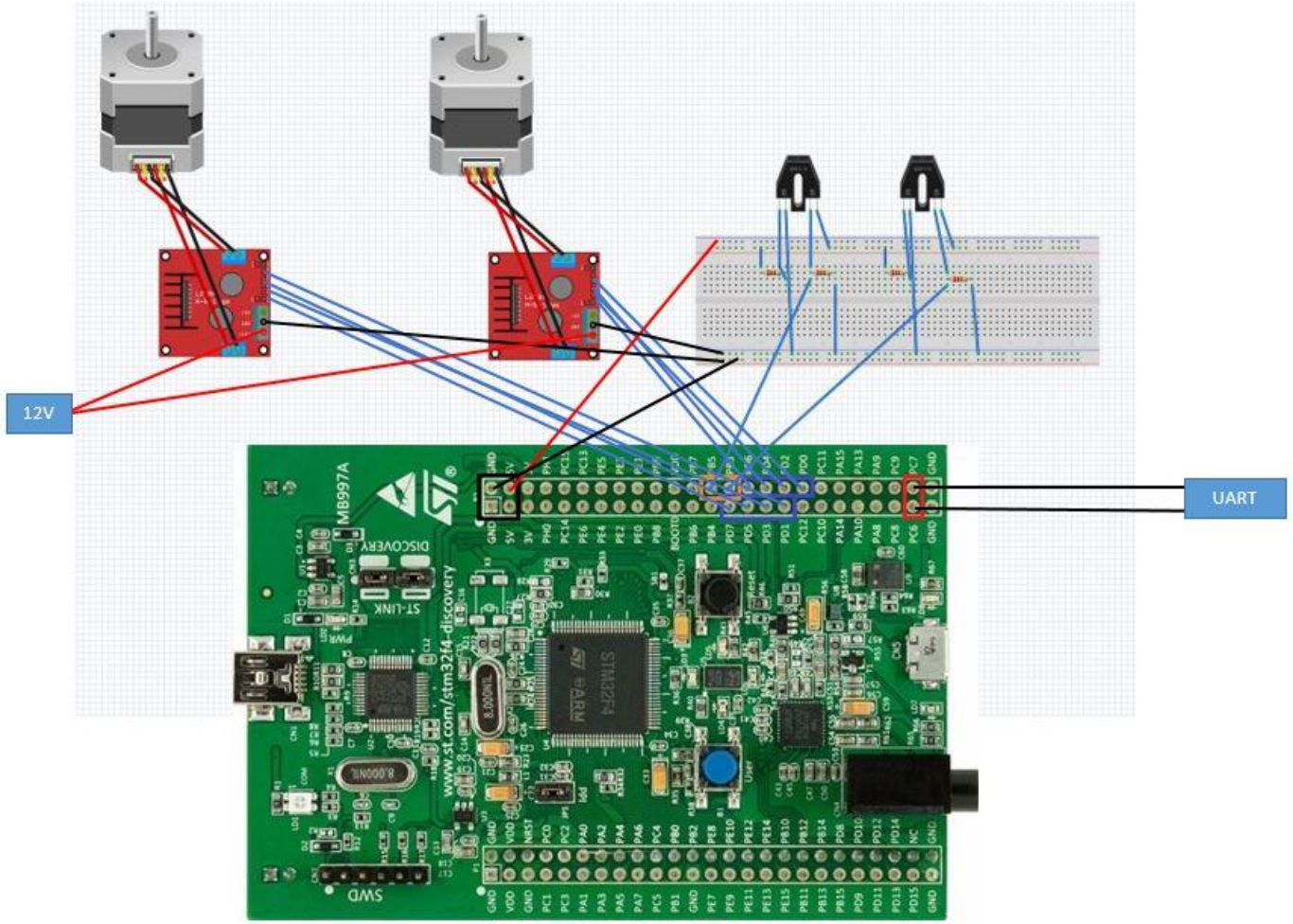



**Fig. Esquemático de las conexiones eléctricas**

Las primeras pruebas se basaron simplemente en mover la plataforma de un pocillo al siguiente simplemente en un eje, de esta manera podíamos medir con facilidad sobre el rail, si la plataforma se estaba moviendo 9mm, que es la distancia desde el centro de un pocillo al centro del siguiente.

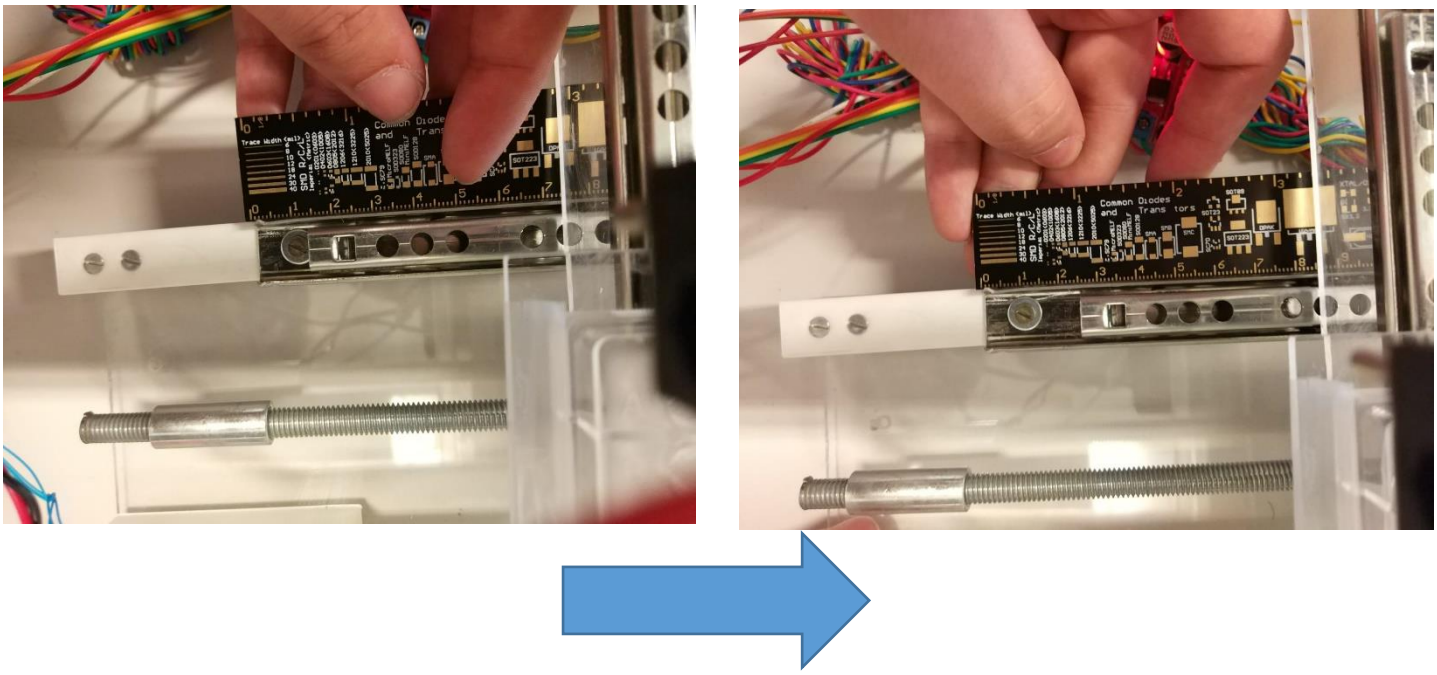
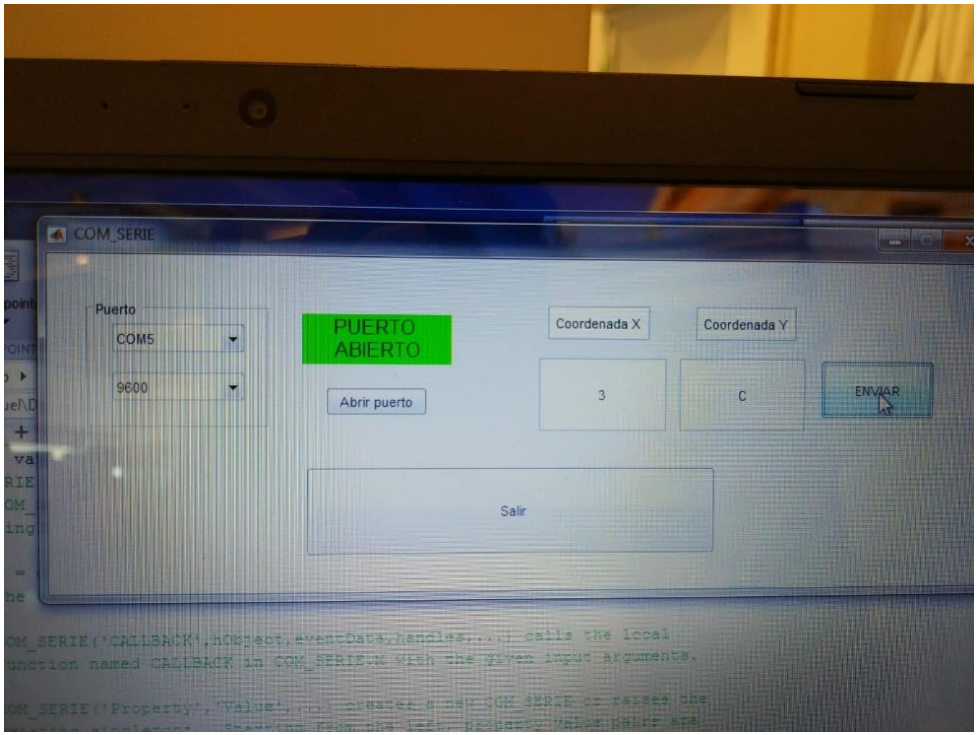


**Fig. 9mm de desplazamiento en el rail**

Se llevaron a cabo varias pruebas y repeticiones y se detectó que, en algunas zonas de la plataforma, el tornillo sin fin requería de más fuerza dado a que parecía que hubiera más rozamiento en alguna parte de la plataforma y esto causaba que ocasionalmente el motor se saltara algunos pasos. La solución a este problema podría haber sido aplicar algo que lubricara el tornillo entero, sobretodo la zona donde se conecta con el anclaje a los motores, o incluso los propios railes, pero el problema parecía ser estrictamente mecánico, ya que cuando sucedía esto podían verse vibraciones en la plataforma y se escuchaba un ruido chirriante.

En conclusión, los resultados han sido satisfactorios, el sistema electrónico funciona como debería funcionar, los motores se controlan con la exactitud y precisión que exigimos, y la interfaz de usuario se conecta y mantiene las comunicaciones de forma estable siendo intuitiva y fácil de utilizar. A pesar de que hayamos encontrado algunos problemas parecen provenir del rozamiento entre algunos elementos de la propia plataforma móvil, y deberán ser solucionados variando un poco su diseño o aplicando métodos que ayuden a reducir estas fuerzas.

En la página siguiente podemos ver un ejemplo a través de dos fotografías, de cómo tras enviar desde la interfaz que la plataforma situara la posición (C,3) de la Placa Elisa bajo el cabezal del láser, se consiguió el resultado satisfactoriamente.



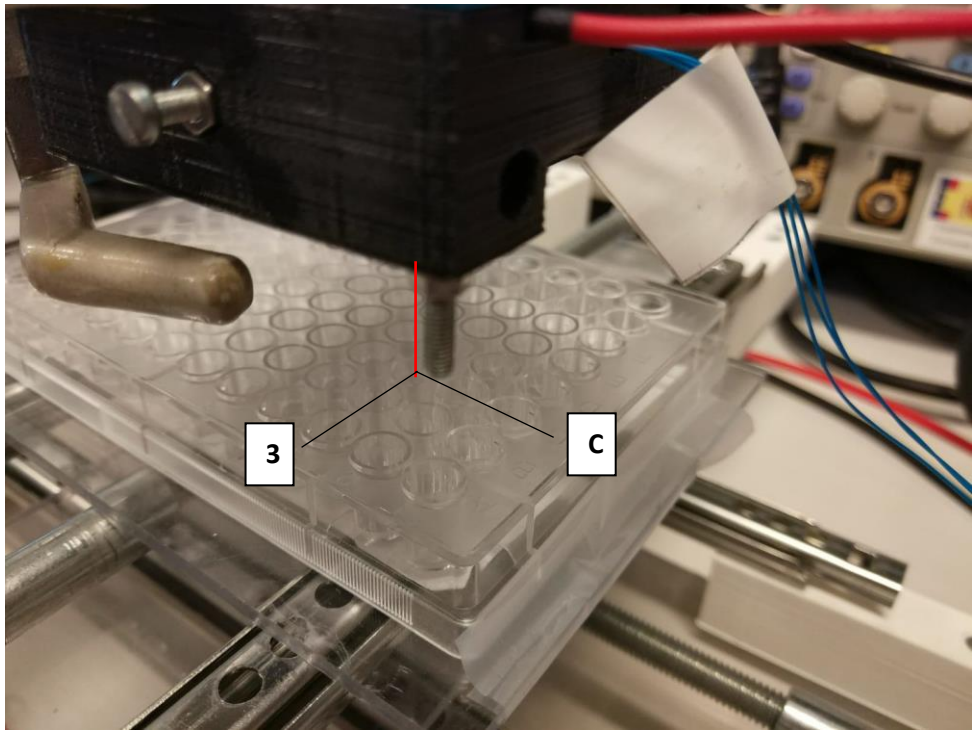




**Fig. Muestra de posicionamiento sobre la coordenada (3,C)**

# 5. Mejoras y sugerencias

Con el tiempo se podrían añadir nuevas funciones al sistema que mejoraran aún más la experiencia del usuario. Algunas de las ideas que el tutor recomendó que se podrían añadir con el tiempo, son las siguientes:

- Desarrollo y aplicación de un botón en la interfaz que moviera la plataforma muy sutilmente para que el láser irradie en diferentes zonas del mismo pocillo
- Desarrollo de un temporizador para que el sistema irradie automáticamente un tiempo determinado
- Implementación de un sistema para que se pueda insertar varias coordenadas a la vez y la plataforma se desplace a esas posiciones en el orden que le indiquemos.
- Incluir una función, en la interface gráfica, que permita mover manualmente la placa ELISA. 4 flechas para realizar desplazamientos.

# 6. Conclusiones

Mi trabajo ha consistido en realizar el control de una plataforma móvil, para que pudiera ser utilizada por investigadores en el Instituto Interuniversitario de Investigación de Reconocimiento Molecular y Desarrollo Tecnológico (IDM) de la Universitat Politécnica de Valéncia (UPV). He podido ver de primera mano, como campos tan distintos como pueden ser la electrónica, la mecánica, la química y la medicina, se han de unir para elaborar medios que puedan probar o demostrar, ideas o conjeturas.

En este caso se ha conseguido: Diseñar una interfaz gráfica que se comunica con una tarjeta de microcontroladores, el control de dos motores bipolares paso a paso que mueven una plataforma en dos ejes distintos, y por ultimo englobar todo esto creando un sistema con el que un usuario es capaz de controlar el movimiento de dicha plataforma de forma sencilla y a la vez precisa.

Los objetivos del trabajo se han cumplido y ha sido muy gratificante poder verlos de primera mano. También el pensar que esto es adaptable a muchas otras situaciones es algo que me ilusiona, porque el control de los motores, la comunicación entre un pc y una tarjeta de microcontroladores, son cosas que he aprendido a hacer durante la realización de este trabajo, y que como con multitud de cosas que abarca el campo de la electrónica, son aplicables a muchos otros sistemas.

La electrónica es algo que tenemos tan delante de nosotros que muchas veces ni si quiera nos damos cuenta de que está ahí, y a lo largo de este grado no solo he aprendido la teoría que nos enseñaban en las aulas, sino que me ha abierto los ojos a muchas cosas que antes ni me hubiera planteado, que llevan un desarrollo detrás, o que podemos mejorar haciendo uso de todo lo que me han enseñado, consiguiendo que ciertas tareas que resultan difíciles o tediosas de realizar, mucho más convenientes.

# 7. Presupuesto

| 1. Precios elementales | | | |
|---|---|---|---|
| **Ref.** | **Unidades** | **Descripción** | **Precio(€)** |
| m1 | U. | Tarjeta de desarrollo discovery STM32F4 con un núcleo ARM CORTEX-M4F, 1 MB de memoria Flash, 192KB de RAM. La alimentación de la tarjeta es a través del bus USB o desde una tensión externa de alimentación de 5V. Este modelo posee 8 LEDs. | 39,98 |
| m2 | U. | Cable UART | 0,89 |
| m3 | U. | Fotosensor OMROM modelo ee-sx1070 con una distancia de sensado de 5mm y un tiempo de respuesta de 4µs. | 2,91 |
| m4 | U. | Motor paso a paso bipolar 42x42 mm con un diámetro de 5mm, 2 fases, Par de 2,3Kg*cm, Voltaje nominal de 12V y una Corriente nominal de 0,33A | 19,60 |
| m5 | U. | Fuente de alimentación Serie Basic 12V, 2A | 14,00 |
| m6 | U. | Cables para las conexiones eléctricas entre los componentes electrónicos | 4,56 |
| m7 | U. | Driver modelo LN398, controlador motor paso a paso | 2,60 |
| m8 | U. | Licencia de Matlab anual para uso académico | 250,00 |
| h1 | h | Ingeniero/a electrónica industrial y automático programación del microcontrolador | 13,17 |
| h2 | h. | Técnico/a de laboratorio, montaje del proyecto | 9,54 |
| | % | Gastos generales | 7,00 |

| 2. Precios unitarios | | | | | |
|---|---|---|---|---|---|
| **Ref.** | **Unidades** | **Descripción** | **Precio(€)** | **Cantidad** | **Total** |
| d1 | U. | Diseño del software: FIRMWARE, Inicialización de las líneas de E/S del microcontrolador, Lectura de los fotosensensores, Actuación sobre los motores, Bucle principal, Interrupción y recepción de datos y finalmente, diseño de la interfaz gráfica. | | | |
| m8 | U. | Licencia de Matlab anual para uso académico | 250,00 | 1,00 | 250,00 |
| h1 | h | Ingeniero/a electrónica industrial y automático programación del microcontrolador | 13,17 | 30,00 | 395,10 |
| | % | Sobrecoste en el precio | 7,00 | 645,10 | 45,16 |
| | | | | **TOTAL** | **690,26** |

| 2. Precios unitarios | | | | | |
|---|---|---|---|---|---|
| **Ref.** | **Unidades** | **Descripción** | **Precio(€)** | **Cantidad** | **Total** |
| d2 | U. | Diseño y montaje del hardware en el laboratorio: Tarjeta discovery, finales de carrera (fotosensor), driver de los motores, fuente de alimentación, motores bipolares paso a paso. | | | |
| m1 | U. | Tarjeta de desarrollo discovery STM32F4 con un núcleo ARM CORTEX-M4F, 1 MB de memoria Flash, 192KB de RAM. La alimentación de la tarjeta es a través del bus USB o desde una tensión externa de alimentación de 5V. Este modelo posee 8 LEDs. | 39,98 | 1,00 | 39,98 |
| m2 | U. | Cable UART | 0,89 | 1,00 | 0,89 |
| m3 | U. | Fotosensor OMROM modelo ee-sx1070 con una distancia de sensado de 5mm y un tiempo de respuesta de 4µs. | 2,91 | 2,00 | 5,82 |
| m4 | U. | Motor paso a paso bipolar 42x42 mm con un diámetro de 5mm, 2 fases, Par de 2,3Kg*cm, Voltaje nominal de 12V y una Corriente nominal de 0,33A | 19,60 | 2,00 | 39,20 |
| m5 | U. | Fuente de alimentación Serie Basic 12V, 2A | 14,00 | 1,00 | 14,00 |
| m6 | U. | Cables para las conexiones eléctricas entre los componentes electrónicos | 4,56 | 1,00 | 4,56 |
| m7 | U. | Driver modelo LN398, controlador motor paso a paso | 2,60 | 2,00 | 5,20 |
| h2 | h. | Técnico/a de laboratorio, montaje del proyecto | 9,54 | 1,50 | 14,31 |
| | % | Sobrecoste en el precio | 7,00 | 123,96 | 8,68 |
| | | | | **TOTAL** | **132,64** |

| 3. Valoración | | | | | |
|---|---|---|---|---|---|
| **Ref.** | **Unidades** | **Descripción** | **Precio(€)** | **Cantidad** | **Total** |
| d1 | U. | Diseño del software: FIRMWARE, Inicialización de la lineas de E/S del microcontrolador, Lectura de los fotosensensores, Actuación sobre los motores, Bucle principal, Interrupción y recepción de datos y finalmente, diseño de la interfaz gráfica. | 690,26 | 1,00 | 690,26 |
| d2 | U. | Diseño y montaje del hardware en el laboratorio: Tarjeta discovery, finales de carrera (fotosensor), driver de los motores, fuente de alimentación, motores bipolares paso a paso. | 132,64 | 1,00 | 132,64 |
| | | | | **IMPORTE** | **822,89** |
| | | | | **IVA 21%** | **172,81** |
| | | | | **TOTAL** | **995,70** |

# Anexo

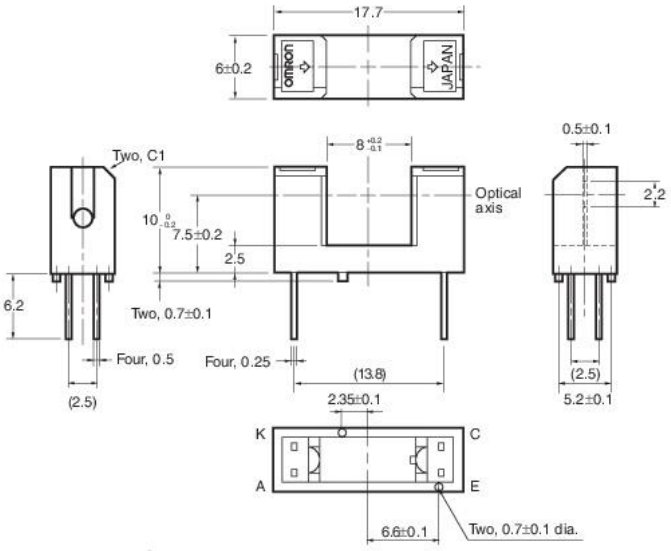
## Anexo 1 Hojas de datos

# Photomicrosensor (Transmissive)
# EE-SX1070

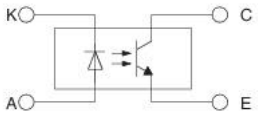
⚠ Be sure to read *Precautions* on page 25.

## ■ Dimensions

**Note:** All units are in millimeters unless otherwise indicated.




**Internal Circuit**




| Terminal No. | Name |
|---|---|
| A | Anode |
| K | Cathode |
| C | Collector |
| E | Emitter |

Unless otherwise specified, the tolerances are as shown below.

| Dimensions | Tolerance |
|---|---|
| 3 mm max. | ±0.3 |
| 3 < mm ≤ 6 | ±0.375 |
| 6 < mm ≤ 10 | ±0.45 |
| 10 < mm ≤ 18 | ±0.55 |
| 18 < mm ≤ 30 | ±0.65 |

## ■ Features

- Wide model with a 8-mm-wide slot.
- PCB mounting type.
- High resolution with a 0.5-mm-wide aperture.

## ■ Absolute Maximum Ratings (Ta = 25°C)

| Item | | Symbol | Rated value |
|---|---|---|---|
| Emitter | Forward current | $I_F$ | 50 mA (see note 1) |
| | Pulse forward current | $I_{FP}$ | 1 A (see note 2) |
| | Reverse voltage | $V_R$ | 4 V |
| Detector | Collector–Emitter voltage | $V_{CEO}$ | 30 V |
| | Emitter–Collector voltage | $V_{ECO}$ | --- |
| | Collector current | $I_C$ | 20 mA |
| | Collector dissipation | $P_C$ | 100 mW (see note 1) |
| Ambient temperature | Operating | Topr | –25°C to 95°C |
| | Storage | Tstg | –30°C to 100°C |
| Soldering temperature | | Tsol | 260°C (see note 3) |

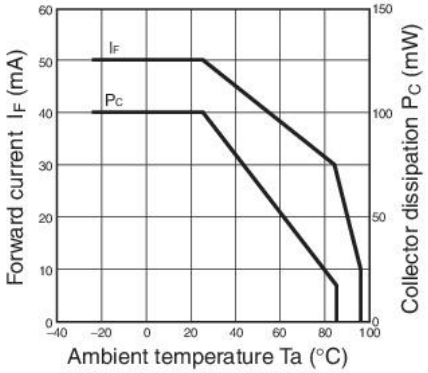
**Note: 1.** Refer to the temperature rating chart if the ambient temperature exceeds 25°C.
**2.** The pulse width is 10 μs maximum with a frequency of 100 Hz.
**3.** Complete soldering within 10 seconds.

## ■ Electrical and Optical Characteristics (Ta = 25°C)

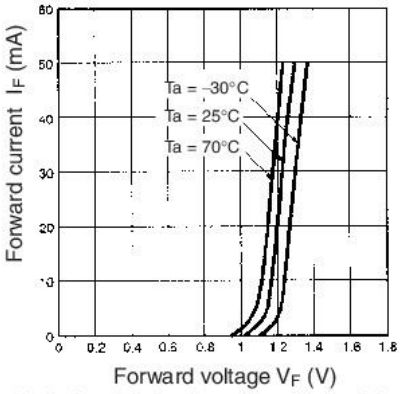
| Item | | Symbol | Value | Condition |
|---|---|---|---|---|
| Emitter | Forward voltage | $V_F$ | 1.2 V typ., 1.5 V max. | $I_F$ = 30 mA |
| | Reverse current | $I_R$ | 0.01 μA typ., 10 μA max. | $V_R$ = 4 V |
| | Peak emission wavelength | $\lambda_P$ | 940 nm typ. | $I_F$ = 20 mA |
| Detector | Light current | $I_L$ | 0.5 mA min., 14 mA max. | $I_F$ = 20 mA, $V_{CE}$ = 10 V |
| | Dark current | $I_D$ | 2 nA typ., 200 nA max. | $V_{CE}$ = 10 V, 0 ℓx |
| | Leakage current | $I_{LEAK}$ | --- | --- |
| | Collector–Emitter saturated voltage | $V_{CE}$ (sat) | 0.1 V typ., 0.4 V max. | $I_F$ = 20 mA, $I_L$ = 0.1 mA |
| | Peak spectral sensitivity wavelength | $\lambda_P$ | 850 nm typ. | $V_{CE}$ = 10 V |
| Rising time | | tr | 4 μs typ. | $V_{CC}$ = 5 V, $R_L$ = 100 Ω, $I_L$ = 5 mA |
| Falling time | | tf | 4 μs typ. | $V_{CC}$ = 5 V, $R_L$ = 100 Ω, $I_L$ = 5 mA |

## ■ Engineering Data

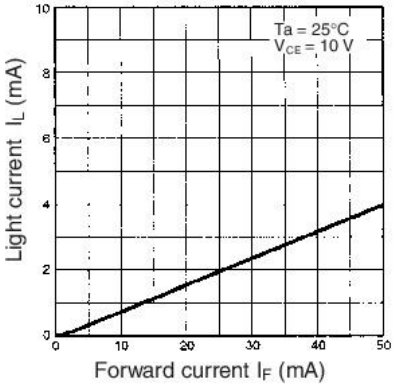
### Forward Current vs. Collector Dissipation Temperature Rating



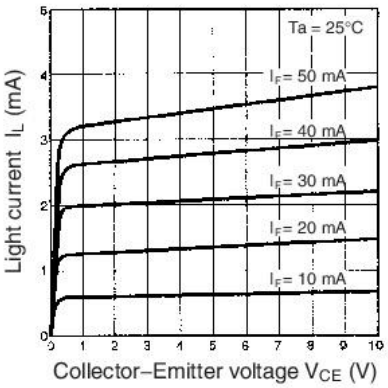

### Forward Current vs. Forward Voltage Characteristics (Typical)



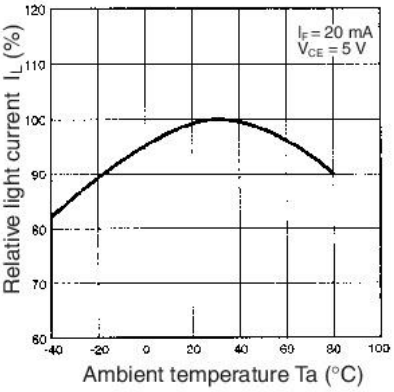

### Light Current vs. Forward Current Characteristics (Typical)



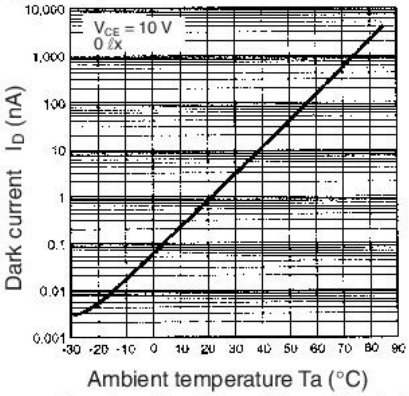

### Light Current vs. Collector–Emitter Voltage Characteristics (Typical)



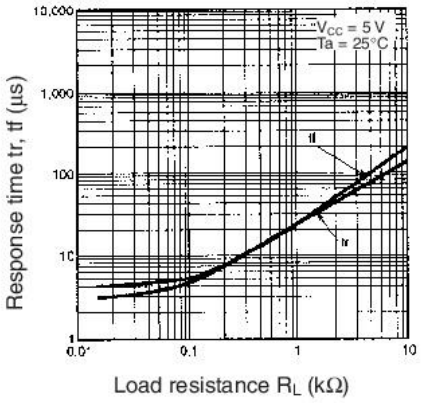

### Relative Light Current vs. Ambient Temperature Characteristics (Typical)



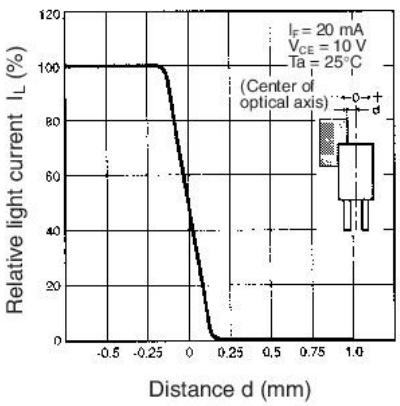

### Dark Current vs. Ambient Temperature Characteristics (Typical)



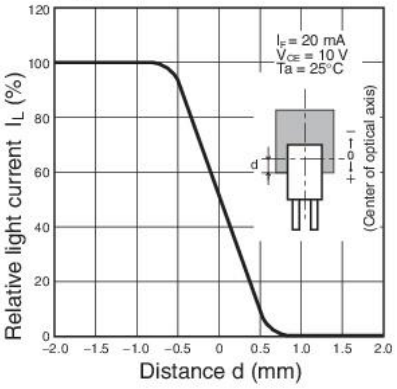

### Response Time vs. Load Resistance Characteristics (Typical)



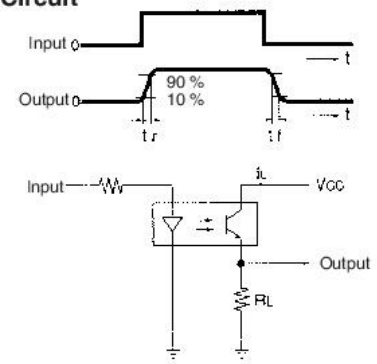

### Sensing Position Characteristics (Typical)

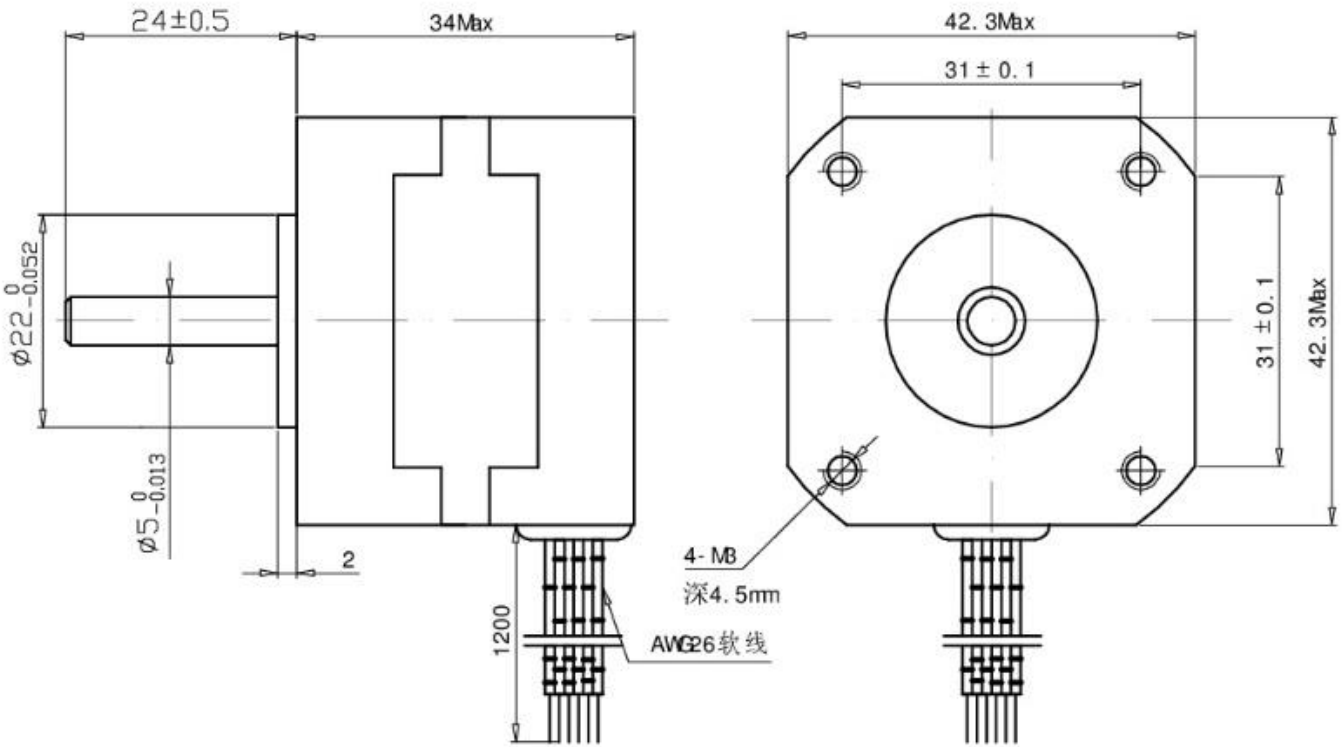



### Sensing Position Characteristics (Typical)

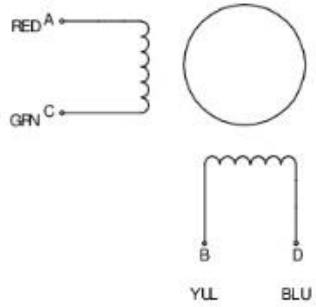



### Response Time Measurement Circuit






WIRING DIAGRAM




CONTROL SIGNATURE:



| COMMON RATEING | | SPECIVICATIONS | |
|---|---|---|---|
| STEP ANGLE | 1.8°±5% | VOLTAGE | 12V |
| PHASES | 2 | CURRENT | 0.33A |
| INSULATION RESISTANCE | 100Mohm(500V DC) | INDUCTANCE | 46±20% Mh |
| CLASS OF INSULATION | B | RESISTANCE | 34±10% |
| WEIGHT | 0.20Kg | HOLDING TORQUE | 0.23N.M |

| REV. | EPO# | DETAILS | DATE |
|---|---|---|---|
| A | | 1ST ISSUE | 090327 |

| | | |
|---|---|---|
| SCALE / UNIT mm / SIZE A4 | UNSPECIFIED TOLERANCE: | MERCURY MOTOR |
| DRAWN XUHEZHAC 20090307 | MATERIAL | TITLE: SM-42BYG011-25 |
| CHECKED | FINISH | SHEET 1 OF 1 REV A |
| APPROVED | | DWG.NO. SM-42BYG011-25-090327 |

FORM-ENG-07

# STM32F4DISCOVERY

## Discovery kit with STM32F407VG MCU

**Data brief**

## Features

- STM32F407VGT6 microcontroller featuring 32-bit ARM® Cortex®-M4 with FPU core, 1-Mbyte Flash memory, 192-Kbyte RAM in an LQFP100 package
- On-board ST-LINK/V2 on STM32F4DISCOVERY (old reference) or ST-LINK/V2-A on STM32F407G-DISC1 (new order code)
- USB ST-LINK with re-enumeration capability and three different interfaces:
  - Debug port
  - Virtual Com port (with new order code only)
  - Mass storage (with new order code only)
- Board power supply: through USB bus or from an external 5 V supply voltage
- External application power supply: 3 V and 5 V
- LIS302DL or LIS3DSH ST MEMS 3-axis accelerometer
- MP45DT02 ST-MEMS audio sensor omni-directional digital microphone
- CS43L22 audio DAC with integrated class D speaker driver
- Eight LEDs:
  - LD1 (red/green) for USB communication
  - LD2 (red) for 3.3 V power on
  - Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue)
  - 2 USB OTG LEDs LD7 (green) VBUS and LD8 (red) over-current
- Two push-buttons (user and reset)
- USB OTG FS with micro-AB connector
- Extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing
- Comprehensive free software including a variety of examples, part of STM32CubeF4 package or STSW-STM32068 to use legacy standard libraries.

1. Picture is not contractual.

## Description

The STM32F4DISCOVERY kit leverages the capabilities of the STM32F407 high performance microcontrollers, to allow users to easily develop applications featuring audio.

It includes an ST-LINK embedded debug tool, one ST-MEMS digital accelerometer, a digital microphone, one audio DAC with integrated class D speaker driver, LEDs, push-buttons and an USB OTG micro-AB connector.

To expand the functionality of the STM32F4DISCOVERY kit with ethernet connectivity, LCD display and more, visit the www.st.com/stm32f4dis-expansion webpage.

With the latest board enhancement, the new order code STM32F407G-DISC1 has replaced the old reference STM32F4DISCOVERY.

# System requirements

- Windows® OS (XP, 7, 8)
- USB type A to Mini-B cable

# Development toolchains

- IAR® EWARM (IAR Embedded Workbench®)
- Keil® MDK-ARM™
- GCC-based IDEs (free AC6: SW4STM32, Atollic® TrueSTUDIO®,...)

# Demonstration software

The demonstration software is preloaded in the board Flash memory. It uses the MEMS motion sensor to blink the four LEDs, according to the motion direction and speed. Connecting the board to a PC with a second USB 'type A to micro-B' cable, converts it into a standard mouse and the board motion controls the PC cursor.

The latest versions of the demonstration source code and associated documentation can be downloaded from the www.st.com/stm32f4-discovery webpage.

# Product marking

Tools marked as "ES" or "E" are not yet qualified and as such, they may be used only for evaluation purposes. ST shall not be liable for any consequences related with other ways of use of such non-qualified tools, for example, as reference design or for production.

Examples of location of "E" or "ES" marking:

- on target STM32 microcontroller part mounted on the board (for illustration, refer to section “Package information” of a STM32 datasheet at *www.st.com*)
- next to the evaluation tool ordering part number, as a label stuck or a silk-screen printed on the board

# Ordering information

To order the Discovery kit for the STM32F407 line of microcontrollers, refer to *Table 1*.

**Table 1. List of the order codes**

| Order code | ST-LINK version |
|---|---|
| STM32F4DISCOVERY | ST-LINK/V2 |
| STM32F407G-DISC1 | ST-LINK/V2-A |

# Revision history

**Table 2. Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 15-Sep-2011 | 1 | Initial version. |
| 28-Jan-2013 | 2 | Added URL for expanding functionality in *Description*. |
| 15-Jul-2013 | 3 | Modified to apply to STM32F407/417. Added LIS3DSH accelerometer. |
| 29-Sep-2014 | 4 | Updated *Section : Features* and *Section : Description* to introduce STM32CubeF4 and STSW-STM32078.<br>Updated *Section : System requirements* and *Section : Development toolchains*. |
| 25-Feb-2016 | 5 | Updated *Features*, *Description* and *System requirements* to introduce STM32F407G-DISC1. |
| 28-Oct-2016 | 6 | Updated *Features* and *Description* to remove reference to mbed™ and to add information on the new order code. |

ROITHNER LASERTECHNIK
Wiedner Hauptstraße 76, A-1040 Vienna, Austria
Tel.: ++43 1 586 52 43-0, Fax -44, office@roithner-laser.com

# LDM-0808-500m-92

## TECHNICAL DATA

## High Power Infrared Laser Diode

***Features***

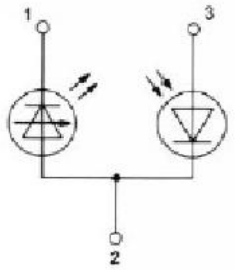
- CW Output Power: 500 mW
- Typical 808 nm Emission Wavelength
- High-efficiency Quantum Well Structure
- TO5 Package

***Applications***

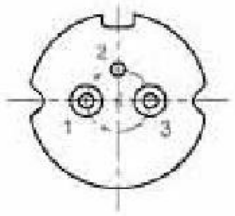
- Solid-state Laser Pumping
- Medical Usage
- Target Designator
- Free-space Optical Communication


**NOTE!**
**LASERDIODE**
**MUST BE COOLED!**

***PIN CONNECTION***




1. Laserdiode cathode
2. Laserdiode anode and photodiode cathode
3. Photodiode anode



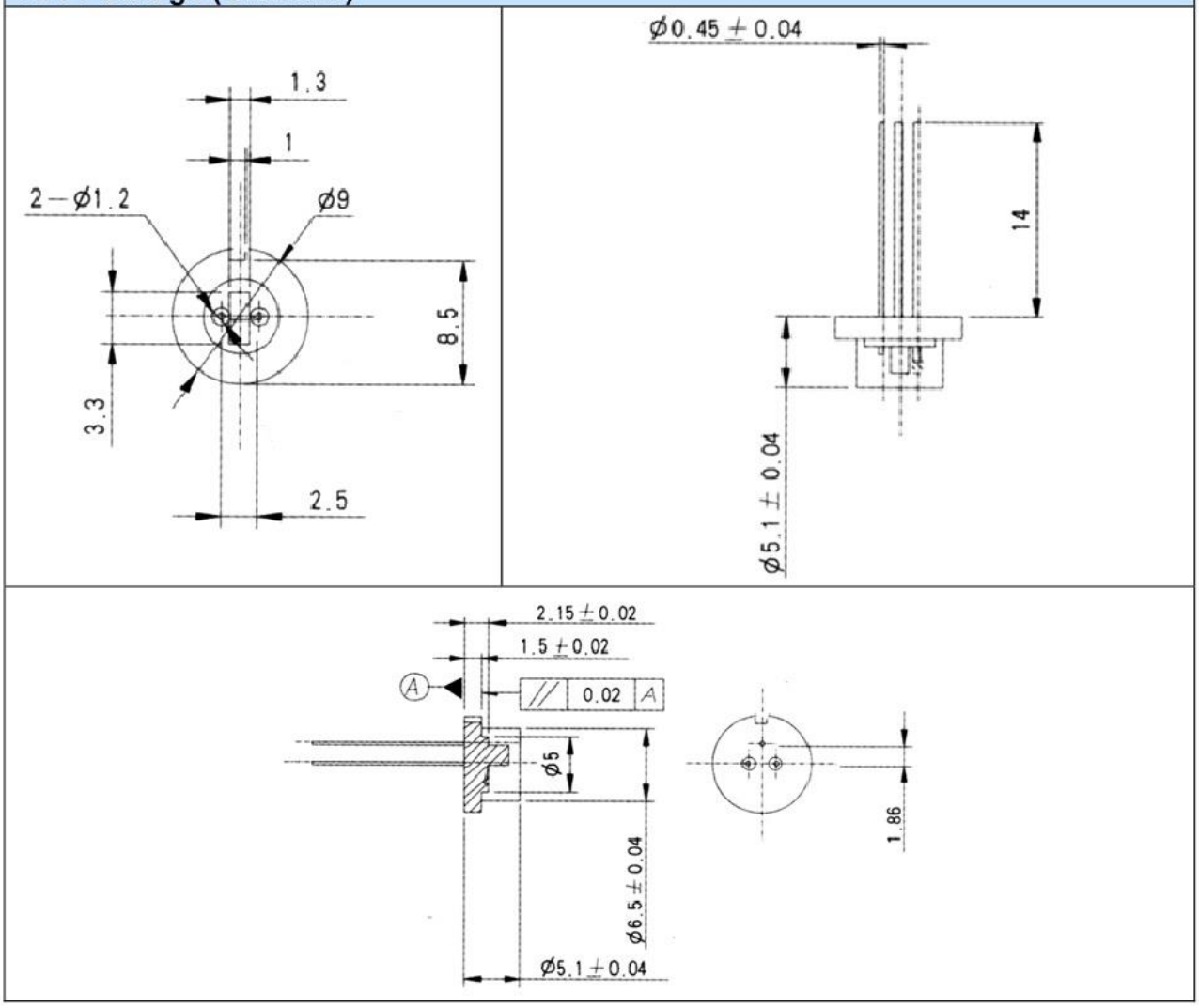

***Specifications (25°C)***

| Type | LDM-0808-500m-92 | Unit |
|---|---|---|
| **Optical Specification** | | |
| CW Output Power $P_O$ | 500 | mW |
| Peak Wavelength Δ | 808±10 | nm |
| Spectral Width Δλ | ≤ 3.0 | nm |
| Emitting Area | 50x1 | µm |
| Wavelength Temperature Coefficient | 0.3 | nm/°C |
| Beam Divergence $\theta_\perp \times \theta_\parallel$ | 40x10 | Deg |
| Polarization | TE | |
| **Electrical Specification** | | |
| Slope Efficiency $E_S$ | ≥ 1.0 | W/A |
| Threshold Current $I_{th}$ | ≤ 0.13 | A |
| Operation Current $I_O$ | ≤ 0.6 | A |
| Operation Voltage $V_f$ | ≤ 2 | V |
| Series Resistance $R_d$ | ≤ 0.6 | Ω |
| Package Style | TO5 | |
| **Absolute Maximum Ratings** | | |
| Reverse Voltage $V_r$ | 2.0 | V |
| Operating Temperature $T_O$ | 10 … 30 | °C |
| Storage Temperature $T_{stg}$ | -40 … 85 | °C |

ROITHNER LASERTECHNIK
Wiedner Hauptstraße 76, A-1040 Vienna, Austria
Tel.: ++43 1 586 52 43-0, Fax -44, office@roithner-laser.com

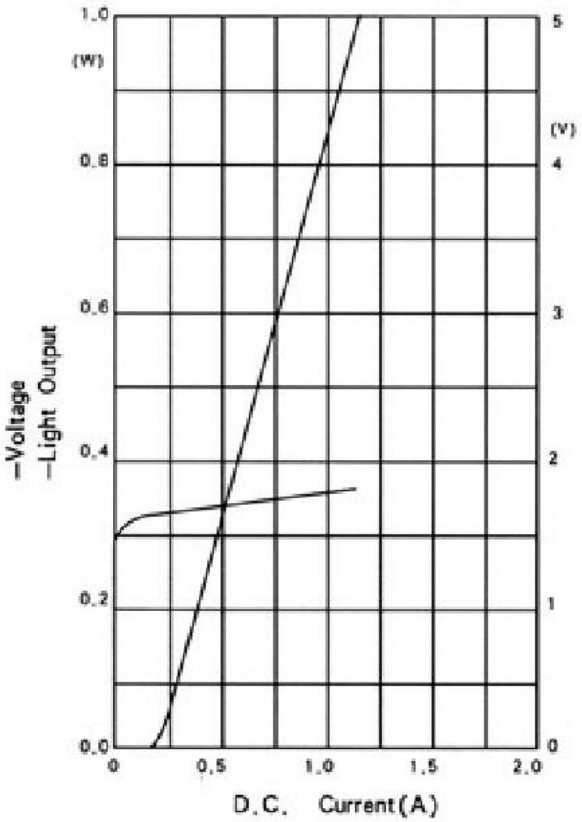
***Package Dimensons***

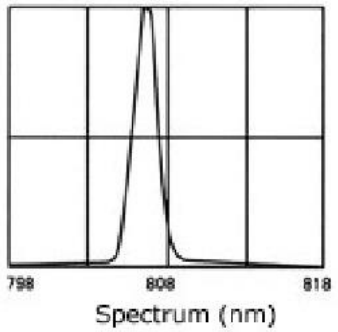
**TO5 Package (*Unit:mm*)**



ROITHNER LASERTECHNIK
Wiedner Hauptstraße 76, A-1040 Vienna, Austria
Tel.: ++43 1 586 52 43-0, Fax -44, office@roithner-laser.com

## *Typical Performance Curves*




## *Notes*

1. High power laser diodes are high energy laser devices. It is harmful to human body and health. Never look directly into the laser output port.
2. High power laser diodes could operate in forward voltage. The reverse current and voltage should not be higher than 25μA and 3V, respectively.
3. Heavy humidity can get dew on the LD then damage the LD.
4. The generated heat must be removed in time when the LD working.
5. The high temperature will effect the performance of the products. The lifetime can also be shortened by high temperature.
6. The operating current and optical power of laser must not be higher than the given rate current and power. The excessive current would accelerate aging and shorten lifetime, even damage the LD.
7. The semiconductor laser diode is a sensitive electronic device. Please observe precaution for handling electrostatitic sensitive devices.

# Anexo 2 Código

# Main.c

```
/*
******************************************************************************
*
File:     main.c
Info:     Generated by Atollic TrueSTUDIO(R) 9.0.0   2018-04-03

The MIT License (MIT)
Copyright (c) 2018 STMicroelectronics

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

******************************************************************************
*
*/

/* Includes */
#include "stm32f4xx.h"
#include "stm32f4_discovery.h"
#include <stdio.h>
#include <stdlib.h>
#include "inicializacion.h"
#include "stm32f4xx_usart.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"


/* Private macro */
/* Private variables */
/* Private function prototypes */
/* Private functions */

/**
**===========================================================================
**
**  Abstract: main program
**
**===========================================================================
*/
void vuelta_motorX_pos(void);
void vuelta_motorY_pos(void);
void vuelta_motorX_neg(void);
```

```
void vuelta_motorY_neg(void);
int start=1;
int XDes=0;
int YDes=0;
int XAct=0;
int YAct=0;
int PasosX=0;
int PasosY=0;
int i=0;
int rec=0;
int posiniX=0;
int posiniY=0;
volatile uint32_t Ticks = 0;
int button=0;

//Inicializacion Clock y Pines
void SysTick_Handler(void){
            Ticks++;
      }

void Delay(uint32_t miliseconds){
      uint32_t ticks_end;
      ticks_end = Ticks + miliseconds;
      while(Ticks < ticks_end){};
      }


//Funcion motor

void vuelta_motorX_pos(){
                    GPIOD->ODR = GPIOD->ODR & 0xFF55;
                    GPIOD->ODR = GPIOD->ODR | (0x0002 | 0x0020);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFF55;
                    GPIOD->ODR = GPIOD->ODR | (0x0002 | 0x0080);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFF55;
                    GPIOD->ODR = GPIOD->ODR | (0x0008 | 0x0080);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFF55;
                    GPIOD->ODR = GPIOD->ODR | (0x0008 | 0x0020);
                    Delay(3);
                    i++;
                        }

void vuelta_motorY_pos(){
                    GPIOD->ODR = GPIOD->ODR & 0xFFAA;
                    GPIOD->ODR = GPIOD->ODR | (0x0001 | 0x0010);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFFAA;
                    GPIOD->ODR = GPIOD->ODR | (0x0001 | 0x0040);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFFAA;
                    GPIOD->ODR = GPIOD->ODR | (0x0004 | 0x0040);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFFAA;
                    GPIOD->ODR = GPIOD->ODR | (0x0004 | 0x0010);
                    Delay(3);
                    i++;
```

```
                        }
void vuelta_motorX_neg(){
                    GPIOD->ODR = GPIOD->ODR & 0xFF55;
                    GPIOD->ODR = GPIOD->ODR | (0x0008 | 0x0020);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFF55;
                    GPIOD->ODR = GPIOD->ODR | (0x0008 | 0x0080);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFF55;
                    GPIOD->ODR = GPIOD->ODR | (0x0002 | 0x0080);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFF55;
                    GPIOD->ODR = GPIOD->ODR | (0x0002 | 0x0020);
                    Delay(3);
                    i++;
                        }
void vuelta_motorY_neg(){
                    GPIOD->ODR = GPIOD->ODR & 0xFFAA;
                    GPIOD->ODR = GPIOD->ODR | (0x0004 | 0x0010);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFFAA;
                    GPIOD->ODR = GPIOD->ODR | (0x0004 | 0x0040);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFFAA;
                    GPIOD->ODR = GPIOD->ODR | (0x0001 | 0x0040);
                    Delay(3);
                    GPIOD->ODR = GPIOD->ODR & 0xFFAA;
                    GPIOD->ODR = GPIOD->ODR | (0x0001 | 0x0010);
                    Delay(3);
                    i++;
                        }
void USART_puts(USART_TypeDef* USARTx, volatile char *s){
      while(*s){
            while(!(USARTx->SR &0x00000040));
            USART_SendData(USARTx, (uint16_t)*s);
            *s++;
      }
}

// Funcion Principal
int main(void)
{

      System_Init();
      SysTick_Config(SystemCoreClock/1000);
      /* USART_SendData(USART6, 0);
      while(posiniX==0){
            if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_3)) {
                  USART_SendData(USART6, 1);
                  posiniX=1;
            }
            else{
                  vuelta_motorX_neg();
            }
      }*/

      while (1) {
            if(rec==1){
                  rec=0;
```

```
PasosX= (XDes-XAct)*50;
PasosY= (YDes-YAct)*50;

if(PasosX!=0){
	USART_SendData(USART6, 0);
	if (PasosX > 0){
		while((i>=0) & (i<PasosX)){
			vuelta_motorX_pos();
			}
			i=0;
			XAct=XDes;
			PasosX=0;
			USART_SendData(USART6, 1);
			GPIOD->ODR = GPIOD->ODR & 0xFF55;
			GPIOD->ODR = GPIOD->ODR | 0x0000;
			Delay(1000);

			}
	if (PasosX < 0){
		USART_SendData(USART6, 0);
		PasosX= PasosX * (-1);
		while((i>=0) & (i<PasosX)){
			vuelta_motorX_neg();
			}
			i=0;
			XAct=XDes;
			PasosX=0;
			USART_SendData(USART6, 1);
			GPIOD->ODR = GPIOD->ODR & 0xFF55;
			GPIOD->ODR = GPIOD->ODR | 0x0000;
			Delay(1000);

			}
	}


if(PasosY!=0){
	USART_SendData(USART6, 0);
	if (PasosY > 0){
		while((i>=0) & (i<PasosY)){
			vuelta_motorY_pos();
			}
			i=0;
			YAct=YDes;
			PasosY=0;
			USART_SendData(USART6, 1);
			GPIOD->ODR = GPIOD->ODR & 0xFFAA;
			GPIOD->ODR = GPIOD->ODR | 0x0000;
			Delay(1000);

			}
	if (PasosY < 0){
		USART_SendData(USART6, 0);
		PasosY= PasosY * (-1);
		while((i>=0) & (i<PasosY)){
			vuelta_motorY_neg();
			}
			i=0;
			YAct=YDes;
```

```
                                PasosY=0;
                                USART_SendData(USART6, 1);
                                GPIOD->ODR = GPIOD->ODR & 0xFFAA;
                                GPIOD->ODR = GPIOD->ODR | 0x0000;
                                Delay(1000);

                                }
                                }
        }
        }

  /**
  *  IMPORTANT NOTE!
  *  The symbol VECT_TAB_SRAM needs to be defined when building the project
  *  if code has been located to RAM and interrupts are used.
  *  Otherwise the interrupt table located in flash will be used.
  *  See also the <system_*.c> file and how the SystemInit() function updates
  *  SCB->VTOR register.
  *  E.g.  SCB->VTOR = 0x20000000;
  */

  /* TODO - Add your application code here */

  /* Initialize LEDs */
  STM_EVAL_LEDInit(LED3);
  STM_EVAL_LEDInit(LED4);
  STM_EVAL_LEDInit(LED5);
  STM_EVAL_LEDInit(LED6);

  /* Turn on LEDs */
  STM_EVAL_LEDOn(LED3);
  STM_EVAL_LEDOn(LED4);
  STM_EVAL_LEDOn(LED5);
  STM_EVAL_LEDOn(LED6);

}


/*
 * Callback used by stm32f4_discovery_audio_codec.c.
 * Refer to stm32f4_discovery_audio_codec.h for more info.
 */
void EVAL_AUDIO_TransferComplete_CallBack(uint32_t pBuffer, uint32_t Size){
  /* TODO, implement your code here */
  return;
}

/*
 * Callback used by stm324xg_eval_audio_codec.c.
 * Refer to stm324xg_eval_audio_codec.h for more info.
 */
uint16_t EVAL_AUDIO_GetSampleCallBack(void){
  /* TODO, implement your code here */
  return -1;
}
```

# Inicialización.c

```
/*
 * inicializacion.c
 *
 *  Created on: 26 abr. 2018
 *      Author: Miguel
 */
/**
 *  Defines for your entire project at one place
 *
 *    @author      Tilen Majerle
 *    @email       tilen@majerle.eu
 *    @website     http://stm32f4-discovery.com
 *    @version     v1.0
 *    @ide         Keil uVision 5
 *    @license     GNU GPL v3
 *
 * |----------------------------------------------------------------------
 * | Copyright (C) Tilen Majerle, 2014
 * |
 * | This program is free software: you can redistribute it and/or modify
 * | it under the terms of the GNU General Public License as published by
 * | the Free Software Foundation, either version 3 of the License, or
 * | any later version.
 * |
 * | This program is distributed in the hope that it will be useful,
 * | but WITHOUT ANY WARRANTY; without even the implied warranty of
 * | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * | GNU General Public License for more details.
 * |
 * | You should have received a copy of the GNU General Public License
 * | along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * |----------------------------------------------------------------------
 */
#include "stm32f4xx.h"
#include "stm32f4xx_usart.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
extern int rec, XDes, YDes;
int pos=0;
int arrayR[2];

void GPIOB_Initialize(){
      GPIO_InitTypeDef GPIO_InitDef;

          RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

          GPIO_InitDef.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_5;
          GPIO_InitDef.GPIO_Mode = GPIO_Mode_IN;
          GPIO_InitDef.GPIO_OType = GPIO_OType_PP;
          GPIO_InitDef.GPIO_PuPd = GPIO_PuPd_DOWN;
          GPIO_InitDef.GPIO_Speed = GPIO_Speed_100MHz;
          GPIO_Init(GPIOB, &GPIO_InitDef);
}
void GPIOD_Initialize(){
      GPIO_InitTypeDef GPIOD_Stepper;
      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
      GPIOD_Stepper.GPIO_Mode = GPIO_Mode_OUT;
```

```
	GPIOD_Stepper.GPIO_OType = GPIO_OType_PP;
	GPIOD_Stepper.GPIO_Pin =
GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|
GPIO_Pin_7;
	GPIOD_Stepper.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIOD_Stepper.GPIO_Speed = GPIO_Speed_2MHz;
	GPIO_Init(GPIOD, &GPIOD_Stepper);
	}

void GPIOC_Initialize(){
	GPIO_InitTypeDef	GPIO_InitStruct;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
	GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_USART6);
	GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_USART6);
	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
	GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
	GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
	GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(GPIOC, &GPIO_InitStruct);
	}

void USART6_Initialize(){
	USART_InitTypeDef USART_InitStruct;
	NVIC_InitTypeDef NVIC_InitStruct;
	/**
	 * Enable clock for USART1 peripheral
	 */
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART6, ENABLE);
	/**
	 * Set Baudrate to value you pass to function
	 * Disable Hardware Flow control
	 * Set Mode To TX and RX, so USART will work in full-duplex mode
	 * Disable parity bit
	 * Set 1 stop bit
	 * Set Data bits to 8
	 *
	 * Initialize USART1
	 * Activate USART1
	 */
	USART_InitStruct.USART_BaudRate = 9600;
	USART_InitStruct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
	USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
	USART_InitStruct.USART_Parity = USART_Parity_No;
	USART_InitStruct.USART_StopBits = USART_StopBits_1;
	USART_InitStruct.USART_WordLength = USART_WordLength_8b;
	USART_Init(USART6, &USART_InitStruct);
	USART_Cmd(USART6, ENABLE);
	/**
	 * Enable RX interrupt
	 */
	USART_ITConfig(USART6, USART_IT_RXNE, ENABLE);

	/**
	 * Set Channel to USART1
	 * Set Channel Cmd to enable. That will enable USART1 channel in NVIC
	 * Set Both priorities to 0. This means high priority
	 *
```

```
	 * Initialize NVIC
	 */
	NVIC_InitStruct.NVIC_IRQChannel = USART6_IRQn;
	NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
	NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
	NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
	NVIC_Init(&NVIC_InitStruct);
	}

void USART6_IRQHandler(){
	if(USART_GetITStatus(USART6,USART_IT_RXNE)){
		//static uint8_t cnt=0;
		XDes = USART6->DR;
		arrayR[pos]=XDes;
		pos++;
		if(pos>1){
			pos=0;
			XDes=arrayR[0];
			YDes=arrayR[1];
			rec=1;
		}

	}
}


void System_Init(){
	GPIOD_Initialize();
	GPIOC_Initialize();
	GPIOB_Initialize();
	USART6_Initialize();
	}
```

# Inicialización.h

```
/*
 * inicializacion.h
 *
 *  Created on: 26 abr. 2018
 *      Author: Miguel
 */

#ifndef INICIALIZACION_H_
#define INICIALIZACION_H_

void GPIOD_Initialize(void);
void GPIOC_Initialize(void);
void GPIOB_Initialize(void);
void USART6_Initialize(void);
void System_Init(void);
void USART6_IRQHandler(void);

#endif /* INICIALIZACION_H_ */
```

# Com_serie.m

```
function varargout = COM_SERIE(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @COM_SERIE_OpeningFcn, ...
                   'gui_OutputFcn',  @COM_SERIE_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function varargout = COM_SERIE_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

%_____________________________________________________________________
____



% --- Executes just before COM_SERIE is made visible.
function COM_SERIE_OpeningFcn(hObject, eventdata, handles, varargin)
%Valores por defecto
global j

j=1;

handles.popCom.Value=3;  % Valor por defecto COM3
handles.popVelocidad.Value=3;   % Valor por defecto 9600
handles.lblEstadoPuerto.BackgroundColor=[1,0,0];
clc;
% Cierra los puertos que se hayan podido quedar abiertos
Puertos_Activos=instrfind; % Lee los puertos activos
if isempty(Puertos_Activos)==0 % Comprueba si hay puertos activos
    fclose(Puertos_Activos); % Cierra los puertos activos
    delete(Puertos_Activos) % Borra la variable Puertos_Activos
    clear Puertos_Activos % Destruye la variable Puertos_Activos
end
% Choose default command line output for COM_SERIE
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% --- Executes on button press in cmdAbrir.

function cmdAbrir_Callback(hObject, eventdata, handles)
global SerPIC
puerto=handles.popCom.String;
```

```
puerto=puerto{handles.popCom.Value};
velocidad=handles.popVelocidad.String;
velocidad=str2double(velocidad{handles.popVelocidad.Value});
SerPIC = serial(puerto);
SerPIC.BaudRate=velocidad;
SerPIC.DataBits=8;
SerPIC.Parity='none';
SerPIC.StopBits=1;
SerPIC.FlowControl='none';
SerPIC.BytesAvailableFcnCount=1;
SerPIC.BytesAvailableFcnMode='byte';
SerPIC.BytesAvailableFcn={@Rx_Callback,handles};
fopen(SerPIC);

handles.lblEstadoPuerto.BackgroundColor=[0,1,0];
handles.lblEstadoPuerto.String='PUERTO ABIERTO';



function Rx_Callback(hObject, eventdata, handles)

global SerPIC contador x y z dato j chksum
%num=SerPIC.BytesAvailable;
%if num>0
    dato=fread(SerPIC,1);
    if dato == 0 handles.tglMuestrear.Enable='off';
        if dato == 1 handles.tglMuestrear.Enable='on';



%handles.lblRes.String=strcat(char(dato(1)),char(dato(2)),char(dato(3)
),char(dato(4)));
            %end
        %end
    %end


function Timer(hObject, eventdata, handles)

global SerPIC

fwrite(SerPIC,'023e');


% --- Executes on button press in tglMuestrear.
function tglMuestrear_Callback(hObject, eventdata, handles)
global x y z contador Temporizador
global SerPIC dato
if (hObject.Value==1)
    dato=double(handles.coordY.String);

    fwrite(SerPIC,str2double(handles.coordX.String));
    fwrite(SerPIC,(dato-64));

    contador=0; x=0; y=0; z=0;
    handles.radiobutton1.BackgroundColor=[0,1,0];
    handles.radiobutton2.BackgroundColor=[0,1,0];
else
```

```
    handles.tglMuestrear.String='Enviar';
end

% --- Executes on button press in cmdSalir.

function cmdSalir_Callback(hObject, eventdata, handles)
global SerPIC

opc=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
if strcmp(opc,'No')
    return;
end

% Cierra los puertos abiertos
Puertos_Activos=instrfind; % Lee los puertos activos
if isempty(Puertos_Activos)==0 % Comprueba si hay puertos activos
    fclose(Puertos_Activos); % Cierra los puertos activos
    delete(Puertos_Activos) % Borra la variable Puertos_Activos
    clear Puertos_Activos % Destruye la variable Puertos_Activos
end

clear
clc
close all
%_____________________________________________________________________
_


function popCom_Callback(hObject, eventdata, handles)

function popCom_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function popVelocidad_Callback(hObject, eventdata, handles)

function popVelocidad_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function coordX_Callback(hObject, eventdata, handles)

function coordX_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function coordY_Callback(hObject, eventdata, handles)

function coordY_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function radiobutton1_Callback(hObject, eventdata, handles)

function radiobutton2_Callback(hObject, eventdata, handles)


function edit5_Callback(hObject, eventdata, handles)
```