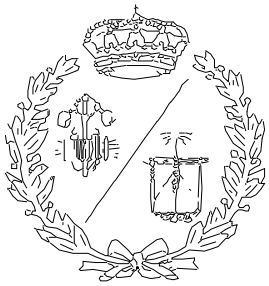
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS

INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Grado

SISTEMA MOTORIZADO PARA INSPECCIÓN DE MATERIALES USANDO TERMOGRAFÍA INFRARROJA Y EXCITACIÓN POR SPOT Y/O LÍNEA LASER

(Motorized system for material inspection using infrared thermography and excitation by laser spot and/or line)

Para acceder al Título de

GRADUADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Álvaro Sierra Díez

Director: Francisco Javier Madruga Saavedra

Marzo-2017

ÍNDICE GENERAL DEL PROYECTO:

****

Índice de figuras

Índice de tablas

DOCUMENTO I

MEMORIA

Índice de memoria

CAPÍTULO 1: INTRODUCCIÓN

1.1. ENUNCIADO

El proyecto consiste en el diseño y desarrollo de hardware y software para un sistema de control de una plataforma con movimiento lineal que se utilizará en la inspección no-destructiva mediante termografía infrarroja activa. La plataforma que portará una pieza a inspeccionar se moverá a una velocidad constante, consiguiendo así una perfecta homogenización del calor aplicado mediante un sistema láser al objeto. El comportamiento térmico predecible se medirá mediante una cámara térmica para su posterior análisis donde se detectará los comportamientos no predecibles para su identificación o no como defectos subsuperficiales de la pieza.

1.2. ANTECEDENTES

A lo largo de la historia se han ido desarrollando y perfeccionando técnicas de control automático con el fin de reducir costos de procesos, mejorar la precisión de productos y aumentar los niveles de seguridad, entre otros.

El control de velocidad para motores de corriente continua se lleva desarrollando a lo largo de las últimas décadas, arrojando resultados satisfactorios, aunque lentos y de baja precisión (debido fundamentalmente a las limitaciones de software y hardware).

Actualmente el control de velocidad continúa evolucionando en los procedimientos de control, así como en el software y en el hardware, lo que hace que la velocidad y la precisión sean cada vez mejores.

1.3. ALCANCE DEL PROYECTO

El proyecto de título tiene como objetivo la implantación física de un sistema de control automático de velocidad constante de una mesa lineal. El control de la mesa lineal se realizará por medio de una interfaz de usuario. Se desarrollará experimentalmente en el laboratorio utilizando alternativas comerciales contrastadas para realizar la aplicación e instalación del sistema.

1.4. JUSTIFICACIÓN

En el ámbito industrial siempre es necesario controlar los procesos o sistemas de producción en variables como posición, velocidad, aceleración, peso, etc. De tal manera que se produzcan el menor número de errores posibles y a la vez aumente la exactitud y la eficiencia.

El control de velocidad ofrece un campo amplio de aplicaciones del control automático, por ejemplo, en este proyecto el control de velocidad dependerá del tiempo deseado para realizar el movimiento completo de la mesa lineal.

1.5. OBJETIVOS

1.5.1. Objetivo general

Diseñar y construir un control automático de velocidad constante para su uso en sistemas de ensayos no destructivos de piezas grandes basado en termografía infrarroja.

1.5.2. Objetivos particulares

- **Control de inicio y final de carrera del sistema.**
- **Control preciso de la velocidad y la posición de inicio y fin de cada movimiento del sistema.**
- **Diseñar e implementar una interfaz gráfica de control para el movimiento de la plataforma en Matlab, lo que implica dos objetivos más:**
    - Comunicación mediante puerto serie entre Matlab y la tarjeta microcontroladora Arduino Uno que ejercerá el control del motor.
    - Comunicación entre la tarjeta microcontroladora Arduino Uno y el driver motor DM556 Fully Digital Stepping Driver.

CAPÍTULO 2: CARACTERÍSTICAS TÉCNICAS

El sistema estará compuesto por un driver motor que controlará un motor paso a paso encargado de mover una mesa lineal. En éste capítulo se explicarán las características técnicas de los componentes del proyecto.

2.1. MOTORES PASO A PASO

Los motores paso a paso (PAP), son un tipo de motores que permiten el movimiento del eje por pasos en ambos sentidos de giro del eje, debido a esta característica los motores paso a paso son considerados como motores con una gran precisión, ya que permiten el control de la posición, la velocidad y el sentido de giro.

Cada paso tiene un ángulo muy preciso que viene determinado por la construcción del motor y vendrá dado por el fabricante en la hoja de características, esto permite realizar movimientos exactos sin necesidad de un sistema de control de lazo cerrado.

Los motores paso a paso tienen un comportamiento totalmente diferente a los motores de corriente continua. En primer lugar, no giran libremente por sí mismos. Los motores paso a paso, como su nombre indica, avanzan girando por pequeños pasos, otra diferencia es en la relación velocidad y par motor, en los motores PAP el mayor par motor se produce a baja velocidad mientras que en los motores de CC el par motor a velocidades bajas es muy reducido necesitando el uso de un mecanismo reductor para mejorar sus prestaciones.

Este tipo de motores son ampliamente utilizados para la implementación de automatismos y en aplicaciones robóticas debido a su precisión.

Además, presentan otras grandes ventajas respecto a otro tipo de motores ya que se pueden manejar digitalmente sin realimentación, se puede controlar fácilmente su velocidad, tienen una larga vida, el coste es bajo y el mantenimiento es mínimo debido a que carecen de escobillas. [8]

2.1.1. Principio de funcionamiento

Los motores eléctricos basan su funcionamiento en las fuerzas ejercidas por un campo electromagnético creadas al hacer circular una corriente eléctrica a través de una o varias bobinas.

Están constituidos por dos partes, el estator (parte fija) y el rotor (parte móvil).

Al excitar el estator, se crean los polos norte-sur lo que provoca la variación del campo magnético. La respuesta inmediata del rotor es seguir el movimiento de dicho campo ya que tenderá a buscar la posición de equilibrio magnético, es decir, orientará sus polos norte-sur

hacia los polos sur-norte del estator. En el momento que el rotor alcanza la posición de equilibrio, el estator cambia la orientación de sus polos lo que provoca que el rotor vuelva a buscar la posición de equilibrio. Manteniendo esta situación de manera continuada se conseguirá un movimiento giratorio y continuo del rotor lo que provoca que gire el eje del motor transformando así una energía eléctrica en energía mecánica en forman de movimiento circular.

Al número de grados que gira el rotor, cuando se efectúa un cambio de polaridad en las bobinas del estator, se le denomina ángulo de paso.

2.1.2. Tipos de motores paso a paso

Los motores paso a paso según sus características físicas se dividen en dos categorías principales: de imanes permanentes y de reluctancia variable. También existe una combinación de ambos llamados híbridos.

Otra clasificación que se puede hacer de este tipo de motores en función de la forma de conexión y excitación de las bobinas como veremos posteriormente.

2.1.2.1. Motor paso a paso de imanes permanentes

Los motores paso a paso de imanes permanentes [9] se pueden dividir en dos subgrupos principales en función del tipo de bobinado utilizado:

Motores paso a paso unipolares

Los motores unipolares tienen dos bobinas en cada eje del estator y están unidas por extremos opuestos, de manera que al ser alimentadas (solo una de ellas), generan un campo magnético inverso al de la otra bobina.

Motores paso a paso bipolares

Los motores bipolares requieren circuitos de control y de potencia más complejos.

2.1.2.2. Motor paso a paso de reluctancia variable

Los motores de reluctancia variable están constituidos por un rotor de láminas ferromagnéticas no imantadas, formando un cilindro alrededor del eje, las láminas se encuentran ranuradas de forma longitudinal formando dientes (polos del rotor) lo que conlleva una variación de la reluctancia en función de la posición angular. [1]
Al igual que el rotor, el estator también está formado por láminas de material ferromagnético no imantado con una serie de ranuras longitudinales que albergan los bobinados de las fases y forman los polos del estator.

El número de dientes del rotor es menor que el número de dientes del estator.

2.1.2.3. Motor paso a paso híbrido

Los motores paso a paso híbridos funcionan combinando los principios de los motores de imanes permanentes y los de reluctancia variable. Este tipo de motores presentan unos ángulos de paso pequeños además de un par elevado.
Las características del estator son prácticamente iguales que las de los otros tipos de motores paso a paso de imanes permanentes y de reluctancia variable. Las diferencias se encuentran en el rotor, que está formado por un disco cilíndrico imantado en posición longitudinal al eje. El rotor está altamente magnetizado lo que produce un flujo unipolar y las líneas magnéticas generadas por el imán son guiadas por dos cilindros acoplados en ambos extremos de cada uno de los polos (norte y sur).
La excitación de este tipo de motores se produce conjuntamente entre el bobinado y el imán.

2.2. DRIVER MOTOR

La mayoría de dispositivos eléctricos y electrónicos requieren tensiones y corrientes que destruirán los circuitos digitales, por tanto, en términos generales, debemos confiar dicha labor a los llamados circuitos controladores o drivers.

Un driver motor es básicamente un amplificador de corriente. La función de los controladores de motor es tomar una señal de control de baja corriente y convertirla en una señal de corriente más alta que puede conducir un motor.

2.2.1. Aplicaciones del driver motor

- Conmutación de relés
- Motores paso a paso
- Pantallas LED
- Aplicaciones de automoción
- Equipo audiovisual
- Sistemas de navegación

2.3. MESA DE DESPLAZAMIENTO LINEAL

Las mesas lineales son combinaciones compactas y económicas de sistemas de guiado y accionamiento para una gran variedad de aplicaciones.

Dependiendo de la aplicación requerida, existen multitud de mesas de desplazamiento para elegir, por lo que es muy importante conocer el trabajo que va a realizar la mesa para poder seleccionar la más adecuada. Las características más importantes a tener en cuenta son:

- Condiciones de uso.
- Tipo de precisión
- Forma de aplicación.
- Selección del motor.
- Calculo de la carga del motor.
- Análisis de la operación.
- Accesorios.

Para el caso de este proyecto, se ha seleccionado una mesa de desplazamiento accionada mediante un motor paso a paso.

El desplazamiento de la plataforma es producido mediante el mecanismo interior de la mesa lineal (ver Figura 1), llamado tornillo sin fin (husillo). La tuerca husillo es un tipo de mecanismo que está constituido por un tornillo (husillo) que al girar produce el desplazamiento longitudinal de la tuerca en la que va enroscado (movimiento rectilíneo).

Este tornillo irá conectado a un motor paso a paso que facilitará el control del movimiento de la plataforma, de la posición y de la velocidad de desplazamiento.

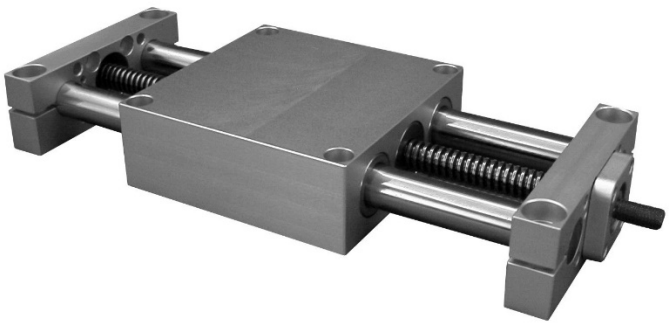


Figura 1. Mesa lineal de desplazamiento

CAPÍTULO 3: ENTORNO DE PROGRAMACIÓN Y SOFTWARE UTILIZADO

Para la realización de este proyecto ha sido necesario el uso de diversos programas y herramientas para cada una de las fases y tareas.

3.1. MATLAB

La programación de la interfaz gráfica se ha realizado mediante el software MATLAB 2016b proporcionado por la Universidad de Cantabria.

MATLAB, de la abreviatura MATrix LABoratory, es un software matemático con un lenguaje de programación propio (lenguaje M). Las funciones principales de este software son las de manipulación de matrices, representación de datos y funciones, la comunicación con otros programas y otros dispositivos hardware. Otra prestación básica que tiene MATLAB y que será lo que se desarrollará en este proyecto son las interfaces gráficas de usuario.

3.2. IDE Arduino

La programación de Arduino se ha realizado mediante software gratuito que puede descargarse en su página oficial de Arduino, aunque es posible utilizar otras herramientas como un editor de texto y un programador de microcontroladores.
Para el funcionamiento del controlador Arduino, primero se crea un programa, conocido como “sketch” en el editor de texto del IDE (ver Figura 2). Posteriormente, se compilará y volcará en la memoria del microcontrolador.

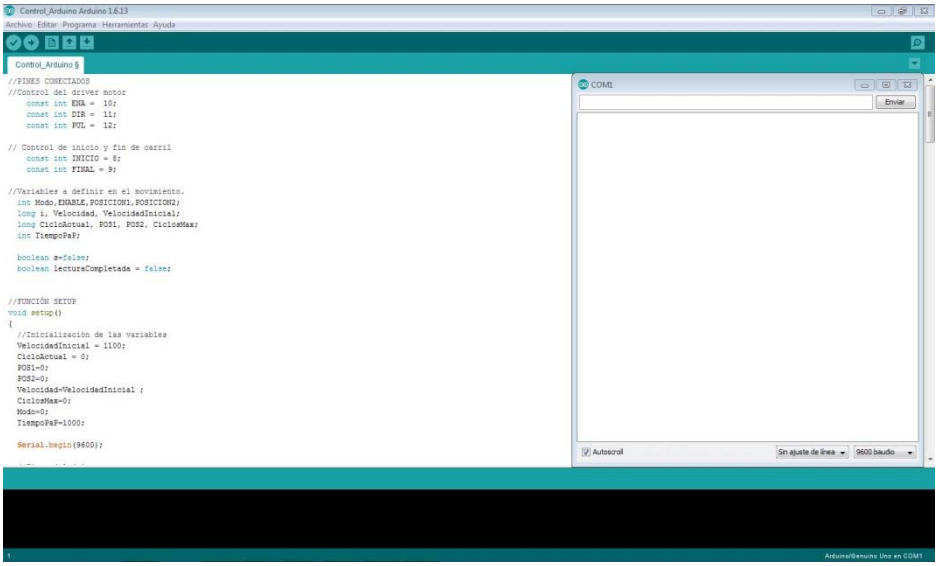


Figura 2. Vista IDE Arduino y Monitor Serie.

CAPÍTULO 4: INTERFAZ GRÁFICA DE USUARIO

La interfaz gráfica de usuario o GUI (del inglés *graphical user interface*) [4], es un programa informático que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador. [7]

4.1. FUNCIONAMIENTO DE LA INTERFAZ

La interfaz de usuario (mostrada en la Figura 3) creada para esta aplicación se encarga de establecer una comunicación por el puerto serial con el microcontrolador de Arduino y pasarle una serie de parámetros que determinarán el movimiento de la plataforma.

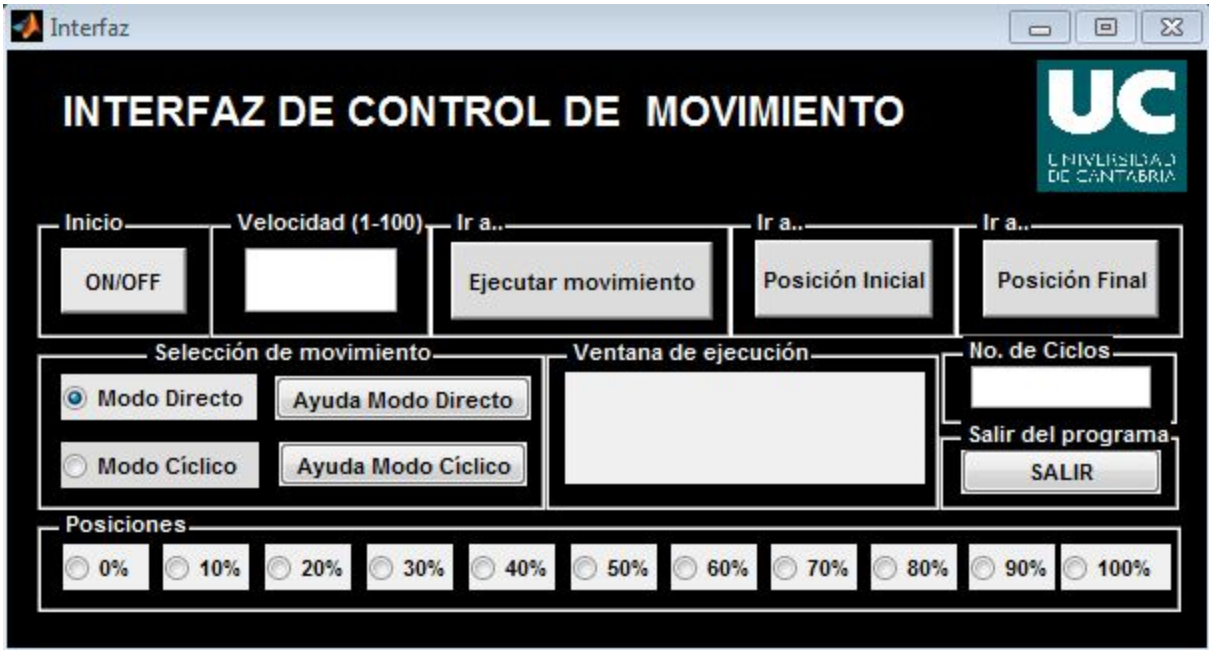


Figura 3. Interfaz de control de movimiento

4.1.1. Interruptor de inicio

El interruptor de inicio es el encargado de activar el funcionamiento del resto de la interfaz, es decir, si este interruptor no está activado no se podrá establecer comunicación con el microcontrolador de Arduino.

4.1.2. Selección de movimiento

El cuadro de selección de movimiento se encarga de seleccionar uno de los dos tipos de movimientos que se pueden realizar.

- El modo de funcionamiento directo permite mover la plataforma hasta la posición seleccionada (si se selecciona más de una posición el programa tendrá en cuenta solamente la posición de menor valor) por el usuario.
- El modo de funcionamiento cíclico permite mover la plataforma entre las dos posiciones seleccionadas (si se seleccionan más de dos posiciones el programa tendrá en cuenta solamente las dos posiciones de menor valor) por el usuario el número de ciclos que desee el usuario.
- Además de la selección de movimiento, en el cuadro de selección aparecen dos botones de ayuda, una para el modo directo y otra para el modo cíclico, estos botones al ser pulsados muestran una pequeña explicación al usuario sobre el tipo de movimiento que realiza la plataforma en cada modo de funcionamiento y a continuación los pasos a seguir para realizar dichos movimientos.

4.1.3. Selección de posiciones

El cuadro de selección de posiciones contiene once posiciones predeterminadas (del 0 al 100%) que podrán ser seleccionadas por el usuario. Dependiendo del tipo de movimiento seleccionado se deberán seleccionar una o dos posiciones.

En caso que se seleccionen más posiciones de las requeridas, el programa solamente tendrá en cuenta una posición para el modo directo y dos para el modo cíclico, que serán siempre las de menor valor, es decir, si para el modo directo están seleccionadas las posiciones 10 y 20, el programa solo tendrá en cuenta la menor de ellas, es decir, la posición 10.

4.1.4. Cuadros de texto

Existen en la interfaz tres tipos de cuadros de texto que pasaremos a explicar a continuación.

- El cuadro de velocidad es una entrada de datos por parte del usuario, donde se le pide escoger un valor de velocidad dentro del rango de valores 1-100%. Siempre que se quiera realizar un movimiento habrá que introducir un valor para la velocidad.
- El cuadro de número de ciclos es otra entrada de datos por parte del usuario, pero solamente será necesario introducir un valor cuando se desee realizar un movimiento cíclico, el dato que se introduzca en este cuadro de texto será el número de ciclos que realice la plataforma entre las dos posiciones seleccionadas.

- El cuadro de la ventana de ejecución es un cuadro que mostrará al usuario el movimiento que ha seleccionado, así como la cadena de caracteres una vez enviada al microcontrolador de Arduino.

4.1.5. Pulsadores

Existen en la interfaz cuatro pulsadores diferentes que pasaremos a explicar a continuación.

- El pulsador Ejecutar movimiento se encarga de crear una cadena de caracteres que contendrá los diferentes parámetros que caracterizan a un movimiento determinado en función de la selección realizada por el usuario. Una vez creada la cadena de caracteres se envía al microcontrolador de Arduino.
- Los pulsadores Posición Inicial y Posición Final se encargan de llevar la plataforma a la posición inicial o a la posición final directamente sin la necesidad de seleccionar el tipo de movimiento o la posición.
- El pulsador Salir activa una ventana de comandos que pregunta al usuario si desea salir realmente de la interfaz, en caso de respuesta positiva se cierra la interfaz, en caso contrario se vuelve a la interfaz.

4.2. DIAGRAMAS DE FLUJO

Como se acaba de mostrar en el apartado anterior, la interfaz de usuario tiene tres pulsadores que ejecutan las acciones seleccionadas por el usuario (pulsador 'Ejecutar movimiento', 'Posición Inicial' y 'Posición Final'), por lo que la parte más importante de la programación se realizará en dichos pulsadores. Para facilitar la comprensión de la programación de la interfaz se mostrarán a continuación un diagrama de flujo para cada uno de los tres pulsadores mencionados.

4.2.1. Pulsadores 'Posición Inicial' y 'Posición Final'

Para el caso de los pulsadores 'Posición Inicial' y 'Posición Final', la programación es idéntica, la única diferencia es la posición de destino que se envía a Arduino que en un caso será la posición inicial y en el otro la posición final.

El funcionamiento de estos dos pulsadores consiste en comprobar el valor de la velocidad introducido por el usuario y, si éste es correcto, establecer el valor necesario de las variables a enviar a Arduino para que la plataforma realice el movimiento deseado.

En la Figura 4 se muestra el diagrama de flujo de los pulsadores mencionados.

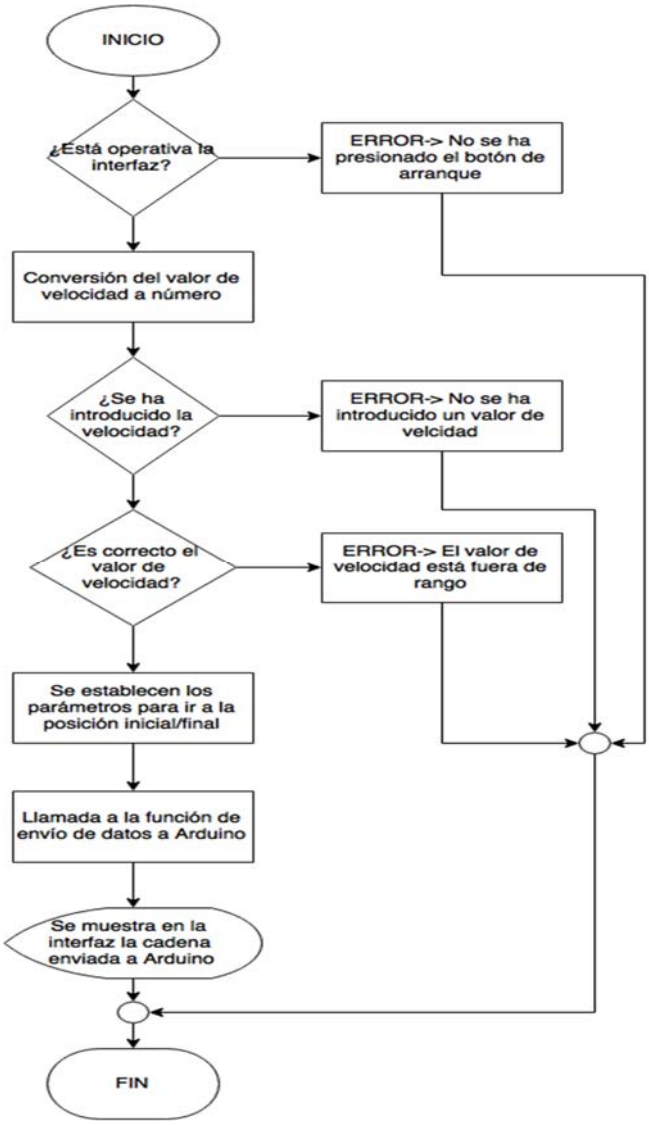


Figura 4. Diagrama de flujo referente a los pulsadores 'Posición Inicial' y 'Posición Final'

4.2.2. Pulsador 'Ejecutar movimiento'

El pulsador 'Ejecutar movimiento' es el que se utilizará en la mayoría de los casos, puesto que es el pulsador que ejecuta los diferentes movimientos permitidos.

El funcionamiento de dicho pulsador consiste en realizar una comprobación de cada uno de los datos necesarios para realizar una acción. Estos datos vendrán dados por el usuario mediante la interfaz gráfica. Se comienza comprobando si la interfaz está operativa o no, a continuación, se comprueba que el valor de velocidad introducido sea correcto, después se comprueba el modo de funcionamiento seleccionado y las posiciones seleccionadas. Una vez comprobado que todos los datos son correctos se enviarán al controlador Arduino.

En la Figura 5 se muestra el diagrama de flujo del pulsador mencionado.

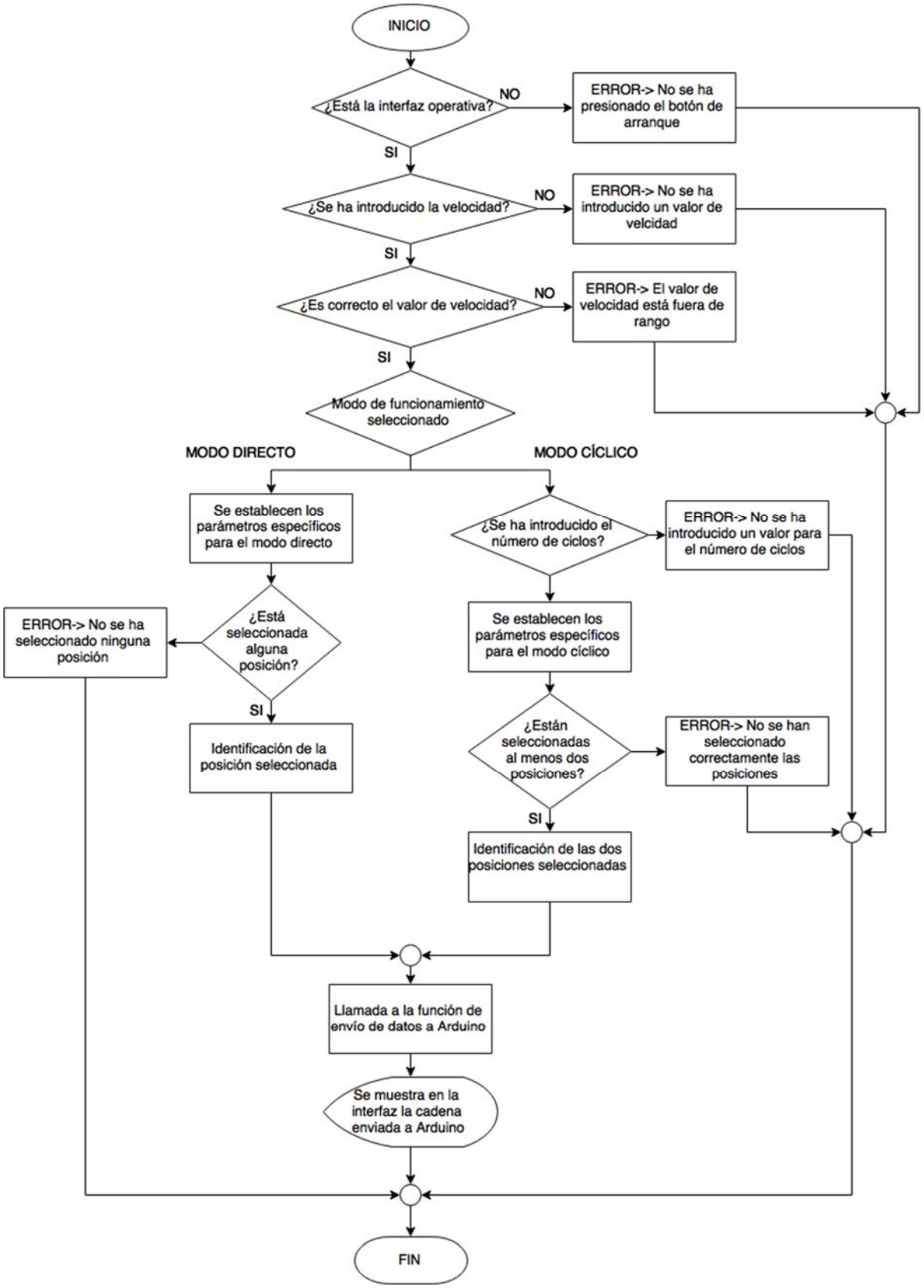


Figura 5. Diagrama de flujo referente al pulsador 'Ejecutar movimiento'

4.3. GESTIÓN DE ERRORES

Como el programa de Arduino, en principio, no se debería poder modificar una vez instalado hay que tener en cuenta que los parámetros que le enviemos deben estar dentro de un rango de valores, por lo que se debe comprobar cada parámetro individualmente con objeto de no enviar nada incorrecto al microcontrolador que pueda producir daños.

Antes de enviar los datos recogidos en la interfaz al controlador Arduino se tiene que realizar un control de que todos los datos son correctos, de lo contrario se podría producir un error en el programa del controlador que no se podría solucionar desde la interfaz de usuario.

4.3.1. Error de arranque

Siempre que se quiera realizar cualquier operación en la interfaz de usuario, el interruptor de arranque tiene que estar activado, si se intenta realizar cualquier operación sin estar el botón de arranque activado aparecerá una ventana de error en la pantalla.

4.3.2. Errores de velocidad

Siempre que se quiera realizar cualquier operación, se debe introducir un valor entre 1 y 100 en la casilla de velocidad. Si se intenta realizar una operación sin antes haber introducido un valor para la velocidad, aparecerá en la pantalla el siguiente error.

Otro posible error que puede aparecer a la hora de introducir la velocidad es que se introduzca un valor que se encuentra fuera del rango, si esto sucede se mostrará una ventana de error en la pantalla que lo indica.

4.3.3. Errores de selección de posiciones

Siempre que se quiera realizar la operación de ‘Ejecutar movimiento’ en modo directo es necesario seleccionar al menos una posición (si hay seleccionada más de una solo se tendrá en cuenta la menor de ellas), en caso contrario aparecerá una ventana de error en la pantalla.

Para el caso del modo cíclico no basta con seleccionar una única posición, es necesario seleccionar al menos dos (si hay seleccionadas más de dos solo se tendrán en cuenta las dos más pequeñas), en caso contrario se mostrará una ventana de error.

4.3.4. Error de selección de ciclos

Siempre que se seleccione el modo de funcionamiento cíclico es necesario introducir el número de ciclos que se desean realizar, si se intenta ejecutar un movimiento sin haber introducido un valor para el número de ciclos aparecerá una ventana de error en la pantalla.

CAPÍTULO 5: COMUNICACIÓN ENTRE MATLAB Y ARDUINO

Para establecer la comunicación entre MATLAB y Arduino se utiliza el puerto serie que se explicará a continuación.

5.1. PUERTO SERIE

Un puerto serie envía la información mediante una secuencia de bits. Para ello se necesitan al menos dos conectores para realizar la comunicación de datos, RX (recepción) y TX (transmisión). No obstante, pueden existir otros conductores para referencia de tensión, sincronismo de reloj, etc.

5.1.1. Antecedentes

Un ordenador convencional dispone de varios puertos de serie. Los más conocidos son el popular USB (Universal Serial Port) y el ya casi olvidado RS-232 (el de los antiguos ratones). Sin embargo, dentro del ámbito de la informática y automatización existen una gran cantidad adicional de tipos de puertos serie**,** como por ejemplo el RS-485, I2C, SPI, Serial Ata, Pcie Express, Ethernet o FireWire, entre otros.

En ocasiones veréis referirse a los puertos de serie como UART. La UART (universally asynchronous receiver/transmitter) es una unidad que incorporan ciertos procesadores, encargada de realiza la conversión de los datos a una secuencia de bits y transmitirlos o recibirlos a una velocidad determinada.

Por otro lado, también podéis oír el término TTL (transistor-transistor logic). Esto significa que la comunicación se realiza mediante variaciones en la señal entre 0V y Vcc (donde Vcc suele ser 3.3V o 5V). Por el contrario, otros sistemas de transmisión emplean variaciones de voltaje de -Vcc a +Vcc (por ejemplo, los puertos RS-232 típicamente varían entre -12V a 12V). [2]

5.1.2. Puerto serie en Arduino

Prácticamente todas las placas Arduino disponen al menos de una unidad UART. Las placas Arduino UNO utilizada en este proyecto dispone de una unidad UART que opera a nivel TTL 0V / 5V, por lo que es directamente compatible con la conexión USB.

Los puertos serie están físicamente unidos a distintos pines de la placa Arduino. Lógicamente, mientras usamos los puertos de serie no podemos usar como entradas o salidas digitales los pines asociados con el puerto de serie en uso. En nuestra placa Arduino UNO los pines empleados son 0 (RX) y 1 (TX). [6]

5.1.3. Envío de datos de MATLAB a Arduino

La comunicación entre MATLAB y Arduino es unidireccional, MATLAB es el encargado de enviar los datos por el puerto serie y Arduino es el encargado de recibirlos [10].

Los datos enviados por MATLAB son los siguientes:

- ENABLE (0/1): Este dato permite controlar la activación y la desactivación del driver motor, si el driver está desactivado no permitirá el movimiento de la plataforma.
- VELOCIDAD (1-100): Este dato establecerá la velocidad de movimiento de la plataforma (tiempo de espera entre pulsos).
- POSICION1 (0-10): Este dato contendrá el valor principal de la posición a la que se desea mover la plataforma en el modo directo. En caso de modo cíclico este dato será una de las posiciones entre las que se deberá mover la plataforma.
- POSICION2 (0-10): Este dato guardará la segunda posición de movimiento de la plataforma, solamente se tiene en cuenta en el modo de funcionamiento cíclico. En caso de modo directo su valor será 0.
- MODO (0/1): Este dato contendrá el modo de funcionamiento que se desea aplicar al movimiento de la plataforma.
- CICLOS (>0): Este dato contendrá para el caso del modo de funcionamiento cíclico el número de ciclos que se desea realizar. En caso de modo de funcionamiento directo este valor no será tenido en cuenta.

Cuando todos los datos han sido definidos correctamente por medio de la interfaz de usuario, para que el envío de estos datos a Arduino resulte más sencillo, se creará una cadena de caracteres donde se concatenarán todos los datos a enviar separados por las letras ‘a’, ‘b’…, por lo que la cadena tendrá la siguiente forma:

>ENABLEaVELOCIDADbPOSICION1cPOSICION2dMODOeCICLOS<

Una vez se tiene esta cadena se podrá proceder al envío de la misma a Arduino.

5.1.4. Recepción de datos en Arduino

Una vez enviada la cadena de caracteres desde MATLAB, se recepciona en Arduino, para ello se comprueba que existen datos disponibles en el puerto serie, si hay datos disponibles se procederá a leerlos, teniendo en cuenta el formato con el que llegan los datos, se realizará la lectura del puerto serie de manera individual, es decir, el primer dato que es el ENABLE viene sucedido por una ‘a’, el siguiente dato que es la VELOCIDAD le sucede una ‘b’ y así hasta el último dato.

Una vez que tenemos los datos separados, únicamente hay que transformarlos a un valor numérico para poder operar con ellos en el programa de Arduino.

5.1.5. Incidencias

Uno de los problemas encontrados a la hora de realizar la comunicación por el puerto serie entre MATLAB y Arduino, es una opción que viene activada por defecto en el controlador Arduino que cuando se realiza una comunicación por el puerto serial éste se resetea. Como se podrá ver más adelante en el código de Arduino, la posición de la plataforma viene dada por una variable que controla el número de pasos que ha dado el motor paso a paso desde una posición controlada, el problema de reseteo del microcontrolador es que se pierde el valor de dicha variable por lo que se pierde el control de la posición. [5]

Para solucionar el problema se han tenido en cuenta dos posibles soluciones, una manteniendo el reseteo y otra eliminándolo.

- Si se mantiene el reseteo del controlador habrá que modificar el código original del Arduino para realizar un guardado en la memoria EEPROM de la variable de control de la posición de la plataforma una vez que se haya alcanzado la posición seleccionada. La variable se guardaría al final del procedimiento 'Loop' y se recuperaría el valor en el procedimiento 'Setup'.
- La otra opción es la de eliminar el reseteo. Para eliminar la opción del reseteo automático basta con colocar un condensador entre la patilla RESET de Arduino y tierra, esto hará que no se produzca el cambio de flanco en la señal de control del reseteo, que es lo que activa el reseteo manteniendo dicha señal en un nivel fijo.

Se ha escogido la segunda opción que es la de eliminar el reseteo por ser la más cómoda y la más fácil de implementar ya que no requiere de modificaciones en el código, simplemente mediante la colocación de un condensador de 100μF. El único pequeño inconveniente de esta opción es que si se quiere reprogramar el controlador de Arduino habrá que quitar dicho condensador para poder cargar un código nuevo.

5.2. DIAGRAMA DE FLUJO

La función de envío de datos de Matlab a Arduino es una función que se encarga de crear una cadena de caracteres con un formato específico a partir de las variables de control generadas por el usuario mediante la interfaz para posteriormente enviar esa cadena por el puerto serie.

En la Figura 6 se muestra el diagrama de flujo de la comunicación.

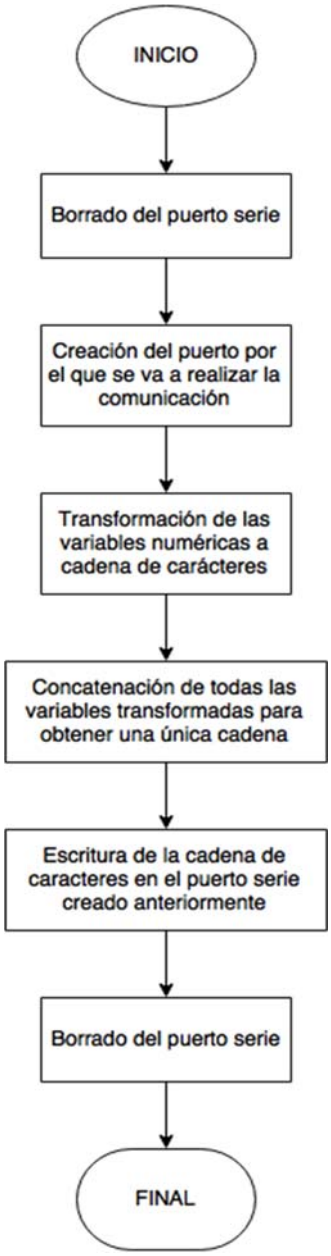


Figura 6. Diagrama de flujo referente a la comunicación entre Matlab y Arduino

CAPÍTULO 6: CONTROL DEL DRIVER MOTOR MEDIANTE ARDUINO

6.1. SOFTWARE

El software de control de Arduino se estructura en dos funciones diferenciadas:

La primera es la función 'Setup' que se caracteriza por que se ejecuta una única vez al comienzo de la ejecución del programa, se utiliza normalmente para realizar la configuración inicial deseada.

La segunda es la función 'Loop' que es la función principal (main) donde se desarrollará la parte más importante del código, esta función se ejecuta en bucle por lo que resulta muy útil a la hora de realizar un control.

6.1.1. Función 'Setup'

Este procedimiento se puede dividir en tres partes para facilitar la comprensión:

- *Configuración del hardware de Arduino*: Configuración de los pines de entrada y salida digitales.
- *Configuración del driver motor*: Se ejecuta una secuencia de inicialización del driver motor.
- *Inicialización de la plataforma*: Se ejecuta un movimiento de la plataforma hacia la posición inicial para poder comenzar el programa teniendo controlada la posición en la que se encuentra la plataforma.

6.1.2. Función 'Loop'

Esta función se puede dividir en cuatro partes:

- *Lectura del puerto serie*: Cuando se detecta que el puerto serie ha recibido una comunicación se lee la cadena recibida.
- *Procesamiento de la lectura del puerto serie*: Se divide la cadena de caracteres recibida por el puerto serie en las variables previamente definidas y se transforman a valor numérico.
- *Evaluación de las variables*: Se evalúan las variables obtenidas.
- *Ejecución de acciones*: Dependiendo de las variables evaluadas se ejecutarán diferentes acciones.

6.1.3. Listado de variables utilizadas

A continuación, se muestra la Tabla 1 con las variables fundamentales que se han utilizado en el código de programación de Arduino, así como una explicación detallada de cada una de ellas.

VARIABLE	TIPO	DEFINICIÓN
VelocidadInicial	INT	Variable utilizada en la función 'Setup' que define el tiempo entre pasos ejecutados por el motor. Esta variable tiene un valor fijo ya que únicamente se utiliza para llevar la plataforma a la posición inicial a la velocidad máxima en el periodo de 'Setup'.
j	INT	Variable que controla el número de ciclos ejecutados en el modo de funcionamiento cíclico.
i	LONG	Variable que controla el número de pasos realizados cuando se realiza un movimiento en la plataforma.
POS1	LONG	Variable que contiene los pasos que tiene que realizar el motor para ir desde la posición inicial hasta la posición seleccionada. Este valor se calcula a partir de la posición seleccionada por el usuario en la interfaz y el número total de pasos de un extremo al otro de la plataforma.
POS2	LONG	Variable que contiene los pasos que tiene que realizar el motor para ir desde la posición inicial hasta la posición seleccionada. Este valor se calcula a partir de la posición seleccionada por el usuario en la interfaz y el número total de pasos de un extremo al otro de la plataforma. Esta variable únicamente se tendrá en cuenta para el modo de funcionamiento cíclico donde es necesario seleccionar dos posiciones.
CiclosMax	LONG	Constante que contiene el número total de ciclos para mover la plataforma de la posición inicial a la posición final.
CicloActual	LONG	Variable que contiene el número de pasos actual en la que se encuentra la plataforma medidos desde la posición inicial.

Tabla 1. Variables utilizadas en el código de Arduino

6.2. HARDWARE

En este apartado de hardware se explicarán los pines que se han utilizado para las diferentes señales.

6.2.1. Finales de carrera

El final de carrera o sensor de contacto (también conocido como "interruptor de límite"), son dispositivos eléctricos, neumáticos o mecánicos situados al final del recorrido o de un elemento móvil, como por ejemplo una cinta transportadora, con el objetivo de enviar señales que puedan modificar el estado de un circuito.

En este proyecto se han utilizado dos finales de carrera para tener control sobre la posición en la que se encuentra la plataforma, para ello se han instalado dos pulsadores cuyo esquema eléctrico es el mostrado en la Figura 7, uno a cada extremo del recorrido de la plataforma de manera que si la plataforma alcanza uno de los extremos se active el pulsador correspondiente a ese extremo, mediante estos finales de carrera en el programa de Arduino se identificará si la plataforma se encuentra en las posiciones inicial o final y se detendrá el movimiento del motor (ya que al llegar a un extremo no puede seguir avanzando).

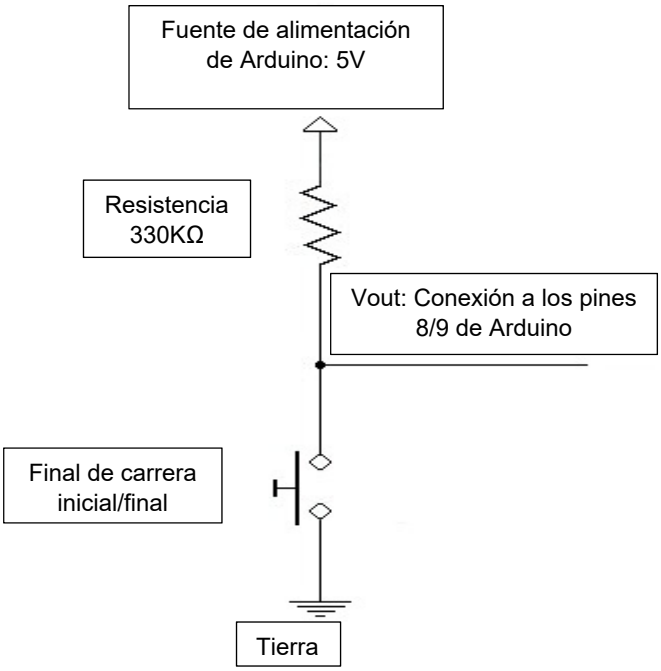


Figura 7. Esquema eléctrico de los finales de carrera

6.2.2. Otras conexiones con el exterior

A parte de los finales de carrera, existen otros tres pines de salida que serán los encargados de comunicar al driver motor las condiciones de movimiento establecidas.

Los tres pines corresponden a la señal ENABLE, que es la encargada de activar o desactivar el driver motor, la señal Dirección que se encarga de establecer el sentido de giro del motor y por último la señal Pulso que es la encargada de definir los pasos que irá ejecutando el motor.

6.2.3. Listado de pines utilizados en el controlador Arduino

A continuación, se muestra la Tabla 2 con los pines utilizados en el microcontrolador Arduino y una breve descripción de la utilización que se le da a dichos pines.

PIN	SEÑAL	NOMBRE	TIPO	DESCRIPCIÓN
10	ENABLE	ENA	Salida digital	Señal de des/activación del driver motor (0->Desactivado, 1->Activado).
11	DIRECCIÓN	DIR	Salida digital	Señal que define el sentido de giro del motor (0->Dirección posición inicial, 1->Dirección posición final).
12	PULSO	PUL	Salida digital	Señal que define el pulso del motor (0->Estado bajo, 1->Estado alto).
8	FINAL DE CARRERA INICIAL	INICIO	Entrada digital	Señal que indica si la plataforma ha alcanzado la posición inicial (0->Pulsador de inicio pulsado, 1->Pulsador de inicio sin pulsar).
9	FINAL DE CARRERA FINAL	FINAL	Entrada digital	Señal que indica si la plataforma ha alcanzado la posición final (0->Pulsador de final pulsado, 1->Pulsador de final sin pulsar).

Tabla 2. Pines utilizados del controlador Arduino

6.3. DIAGRAMAS DE FLUJO

A continuación, y para que quede más claro el funcionamiento del control en Arduino se mostrarán los diagramas de flujo referentes a la función ‘Setup’ y a la función ‘Loop’.

6.3.1. Función ‘Setup’

La función ‘Setup’ se encargará de inicializar el puerto serie, configurar las entradas y salidas de la tarjeta microcontroladora, inicializar el driver motor y mover la plataforma hasta una posición conocida para poder comenzar con el control de la misma por parte del usuario. A continuación en la Figura 8 se muestra el diagrama de flujo de esta función.

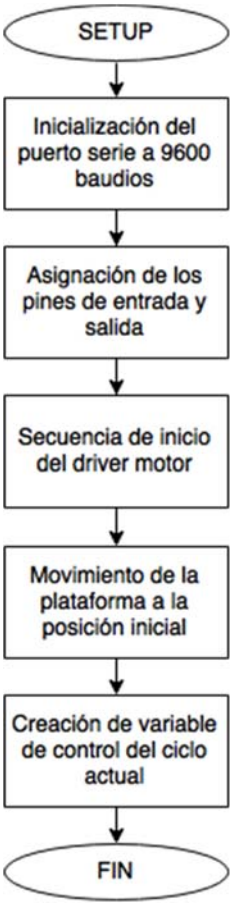


Figura 8. Diagrama de flujo referente al procedimiento de 'Setup' del programa de control de Arduino

6.3.2. Función 'Loop'

La función 'Loop' o función principal, se encargará del control de la plataforma. Lo primero se esperará a que lleguen datos por el puerto serie, como el formato en el que se van a recibir los datos es conocido (una cadena de caracteres), una vez leídos simplemente hay que procesarlos correctamente para obtener las variables deseadas por separado. A continuación, se evalúan el modo de funcionamiento seleccionado (modo directo, modo cíclico, movimiento directo a la posición inicial y movimiento directo a la posición final). Para los modos de funcionamiento directo y cíclico, se comparará la posición en la que se encuentra la plataforma y la posición seleccionada por el usuario en la interfaz para, dependiendo del resultado, establecer el sentido de giro del motor para alcanzar dicha posición. Para los otros dos modos de funcionamiento no será necesario hacer esa comparación puesto que el sentido de giro para alcanzar esas posiciones es siempre el mismo. En la página siguiente la Figura 9 muestra el diagrama de flujo de la función.

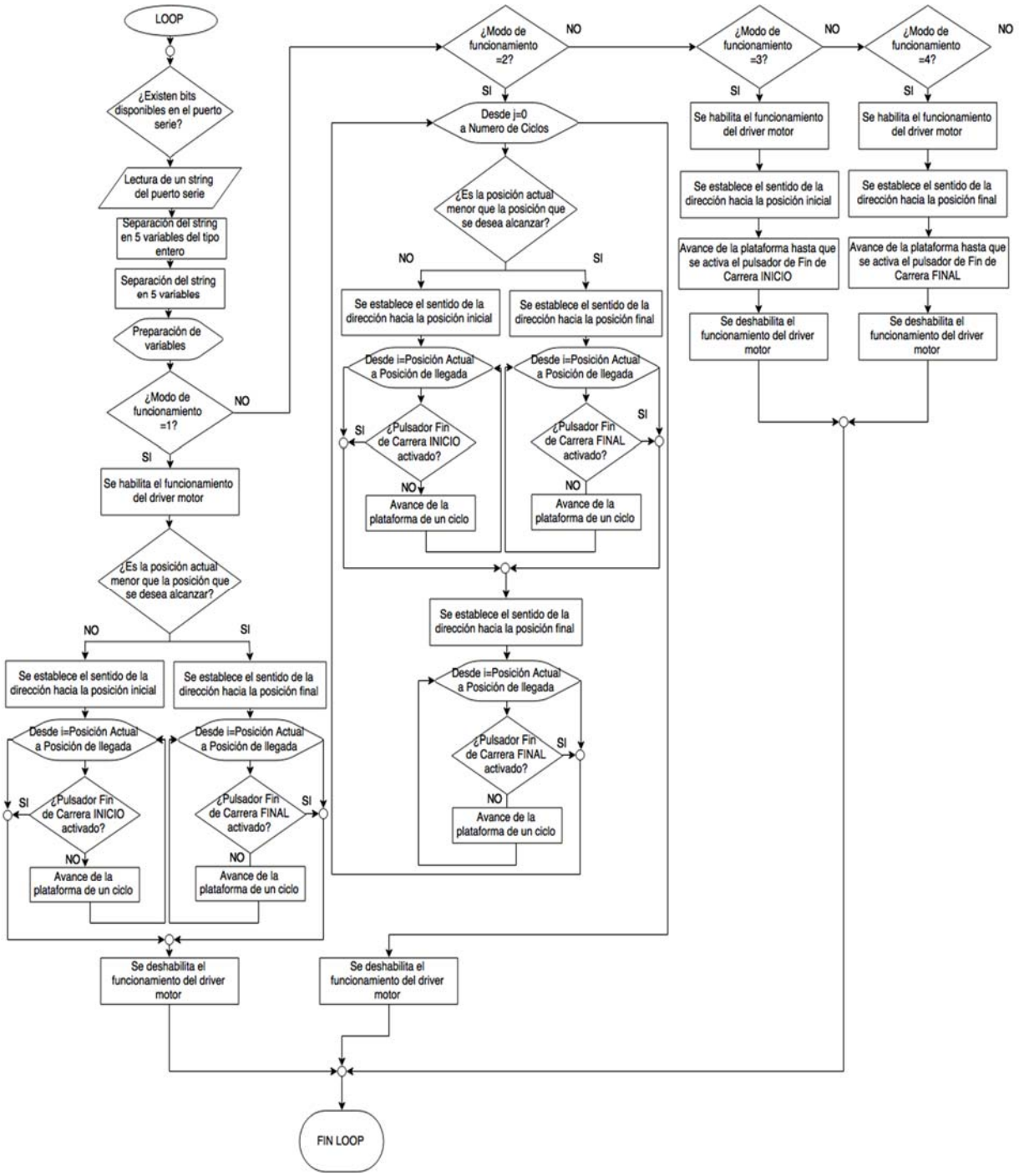


Figura 9. Diagrama de flujo referente al procedimiento 'Loop' del programa de control de Arduino

CAPÍTULO 7: IMPLEMENTACIÓN DEL PROYECTO

En este capítulo se va a hablar de los cálculos necesarios para facilitar el uso del programa por parte del usuario. En la Figura 10 mostrada a continuación se puede ver una imagen de la implementación física final del proyecto, con la fuente de alimentación, la mesa lineal, el microcontrolador Arduino y el driver motor.

Figura 10. Implementación del proyecto

Como se puede ver en el apartado 4.3.2 donde se habla de los errores de velocidad, es necesario introducir el valor de velocidad dentro de un rango comprendido entre 1 y 100. Se ha elegido este rango principalmente por dos motivos, uno para facilitar al usuario la elección del valor de la velocidad y dos debido a que existe un valor máximo para la velocidad (equivalente al valor 100) que tiene que ver con el tiempo de espera mínimo entre pasos del motor para que este funcione correctamente, lo que ocurre si se aumenta la velocidad a más del máximo es que el motor directamente no responde. Por tanto, será necesario crear dos funciones, una que permita convertir el valor introducido por el usuario en el valor real que se utilizará en el control en Arduino y la otra función transformará el valor introducido por el usuario en un valor de velocidad expresado en mm/s.

7.1. CÁLCULO DE LA VARIABLE DE CONTROL DE LA VELOCIDAD DE LA PLATAFORMA EN ARDUINO

El avance de la plataforma es controlado por el driver motor mediante una señal digital llamada PULSO, cada vez que se produce un flanco de bajada o de subida (configurado por

software) el motor avanza un paso. La velocidad por tanto depende directamente de la diferencia de tiempo que se produce en la señal PULSO entre dos flancos consecutivos.

Con objeto de facilitar al usuario el control de la velocidad que desea aplicarle a la plataforma, en la interfaz gráfica se ha establecido un rango de valores para la velocidad entre el 1 y el 100%, que posteriormente se transformará en el tiempo de espera entre dos flancos consecutivos, es decir, si se selecciona la velocidad del 1%, el tiempo de espera será el máximo posible que se ha establecido en 3000 milisegundos y seleccionando la velocidad del 100% el tiempo de espera será el mínimo posible que se ha establecido en 1100 ms. Para realizar este cambio lineal de variable bastará con aplicar la ecuación de una recta.

$$y = m \cdot x + b$$

Se tienen dos de los puntos por los que pasa la recta que son:

$$\begin{cases} A(x_A, y_A) \\ B(x_B, y_B) \end{cases} \Rightarrow \begin{cases} A(1,3000) \\ B(100,1100) \end{cases}$$

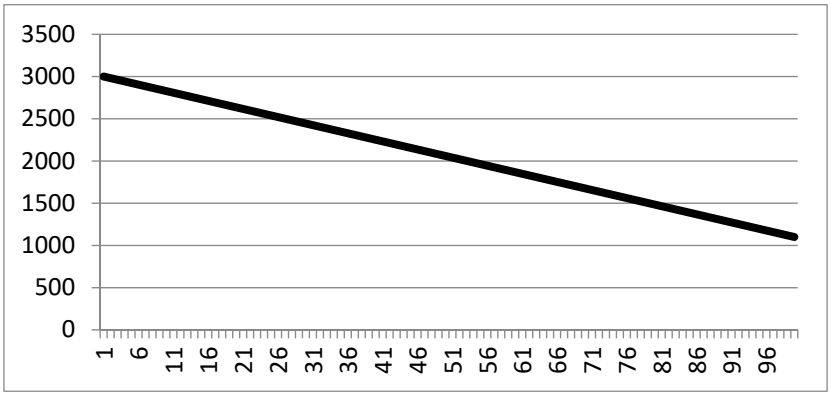


Figura 11. Función cambio de variable de velocidad a µs

Se sabe que la pendiente es: $m = \frac{y_A - y_B}{x_A - x_B} = \frac{3000-1100}{1-100} = -\frac{1900}{99}$

Una vez obtenida la pendiente y conociendo un punto por el que pasa la recta se procede a calcular 'b'.

$$y = \frac{-1900}{99} \cdot x + b$$

Sustituimos el punto A para obtener 'b'.

$$3000 = \frac{-1900}{99} \cdot 1 + b \Rightarrow b = 3000 + \frac{1900}{99} = \frac{(3000 \cdot 99)+1900}{99} = \frac{298900}{99}$$

La ecuación final de la recta que hemos obtenido tiene la siguiente forma:

$$\mathbf{y = \frac{-1900}{99} \cdot x + \frac{298900}{99}}$$

Mediante esta ecuación podremos transformar el dato de velocidad en el rango 1-100 introducido por el usuario en la interfaz (ver Figura 11), en el tiempo que deberá permanecer la señal de PULSO en valor alto o en valor bajo para que la velocidad coincida en el rango del 1 al 100%.

7.2. CÁLCULO DE LA VELOCIDAD DE LA PLATAFORMA

Para calcular la velocidad de la plataforma en mm/s en función de la velocidad introducida por el usuario en el rango de 1 a 100 será necesario como en el apartado anterior crear una función.

El valor de la velocidad se obtiene dividiendo la distancia que recorre la plataforma entre el tiempo que emplea en recorrer dicha distancia. Como se trata de una relación lineal, se han obtenido los valores de velocidad máximo y mínimo con los que se creará la función.

En la Tabla 3 se muestra cómo se han obtenido los datos de velocidad.

Velocidad (usuario)	Posición 1	Posición 2	Distancia (mm)	Tiempo (s)	Velocidad (mm/s)
1	P0->Inicio	P10	75	58	1.2931
100				16.5	4.5454

Tabla 3. Datos de velocidad

Se tienen dos de los puntos por los que pasa la recta que son:

$$\begin{cases} A(x_A, y_A) \\ B(x_B, y_B) \end{cases} \Rightarrow \begin{cases} A(1,1.2931) \\ B(100,4.5454) \end{cases} \qquad y = m \cdot x + b$$

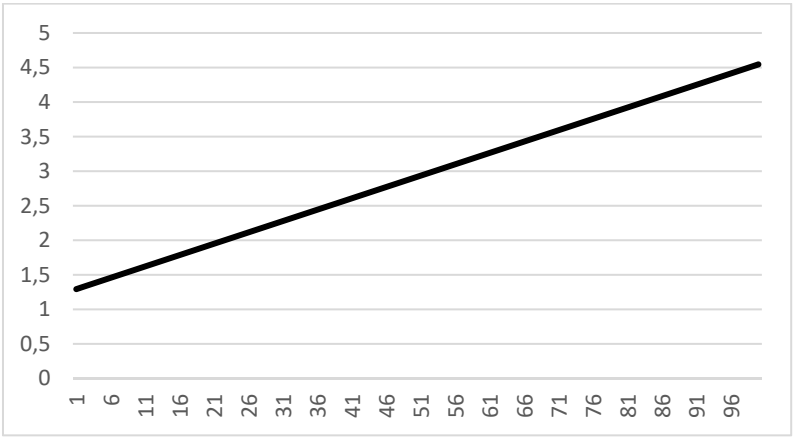


Figura 12. Función cambio de variable de velocidad a mm/s

Se sabe que la pendiente es: $m = \frac{y_A - y_B}{x_A - x_B} = \frac{1.2931 - 4.5454}{1 - 100} = \frac{3.2523}{99}$

Una vez obtenida la pendiente y conociendo un punto por el que pasa la recta se procede a calcular 'b'.

$$y = \frac{3.2523}{99} \cdot x + b$$

Sustituimos el punto B para obtener 'b'.

$$4.5454 = \frac{3.2523}{99} \cdot 100 + b \quad \Rightarrow \quad b = 4.5454 - \frac{325.23}{99} = \frac{(4.5454 \cdot 99) - 325.23}{99} = \frac{124.7646}{99}$$

La ecuación final de la recta que hemos obtenido tiene la siguiente forma:

$$\mathbf{y = \frac{3.2523}{99} \cdot x + \frac{124.7646}{99}}$$

Mediante esta ecuación podremos transformar el dato de velocidad en el rango 1-100 introducido por el usuario en la interfaz (ver Figura 12), en la velocidad real de la plataforma en mm/s.

CAPÍTULO 8: CONCLUSIONES

El presente trabajo fin de grado ha concluido exitosamente cumpliendo con todos los objetivos propuestos. Desde el punto de vista de los objetivos técnicos planteados en el capítulo 1.

- Se ha construido un sistema basado en Arduino y Matlab que permite controlar un motor paso a paso que acciona una mesa lineal a la que está conectado.
- El sistema ha sido totalmente caracterizado en su movimiento controlado para poder ser utilizado en ensayos no destructivos mediante termografía activa que exigen un preciso control de exposición de la pieza a medir a la fuente de calor para controlar la energía suministrada a la pieza y de cuya variación se extrae la defectología que puede existir en la misma.

Desde el punto de vista del aprendizaje:

1. La realización de este trabajo por su componente teórico-práctica me ha sido de mucha utilidad para poder aprender y consolidar el uso de diversas tecnologías.
2. Durante la investigación y las diversas pruebas, he podido comprobar otras funcionalidades relacionadas con el sistema Arduino como la comunicación por puerto serie a través del conector de micro USB.
3. He conocido la técnica de ensayos no destructivos basada en termografía infrarroja.

DOCUMENTO II

ANEXOS

Índice de anexos

ANEXO 1: HARDWARE UTILIZADO

A.1.1. Tarjeta microcontroladora Arduino Uno

Arduino / Genuino Uno es una placa microcontroladora basada en ATmega328P. Tiene 14 pines digitales de entrada / salida (de los cuales 6 se pueden utilizar como salidas PWM), 6 entradas analógicas, un cristal de cuarzo de 16 MHz, una conexión USB, un conector de alimentación, una cabecera ICSP y un botón de reinicio.

Technical specs

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Figura 13. Placa microcontroladora Arduino Uno

A.1.2. Fully Digital Stepping Driver DM556

El DM556 (ver Figura 14) es un controlador digital para motores paso a paso cuya función es la de amplificar la señal de control procedente del microcontrolador en una señal de corriente más elevada con la que pueda trabajar el motor.

Leadshine Technology Co., Ltd. DM556 Microstepping Driver Manual V1.0

1. Introduction, Features and Applications

Introduction

The DM556 is a versatility fully digital stepping Driver based on a DSP with advanced control algorithm. The DM556 is the next generation of digital stepping motor controls. It brings a unique level of system smoothness, providing optimum torque and nulls mid-range instability. Motor self-test and parameter auto-setup technology offers optimum responses with different motors and easy-to-use. The Driven motors can run with much smaller noise, lower heating, smoother movement than most of the Drivers in the markets. Its unique features make the DM556 an ideal solution for applications that require low-speed smoothness.
Compared to the DM432C, broader input voltage and output current ranges make the DM556 can Drive much more motors than the DM432C. What's more, owing to its higher performance DSP, Driven motors can achieve much higher speed (above 3000RPM) than that of the DM432C, offering servo-like performances.

Features

- Anti-Resonance, provides optimum torque and nulls mid-range instability
- Motor self-test and parameter auto-setup technology, offers optimum responses with different motors
- Multi-Stepping allows a low resolution step input to produce a higher microstep output for smooth system performance
- Microstep resolutions programmable, from full-step to 102,400 steps/rev
- Supply voltage up to +45 VDC
- Output current programmable, from 0.5A to 5.6A
- Pulse input frequency up to 200 KHz
- TTL compatible and optically isolated input
- Automatic idle-current reduction
- Suitable for 2-phase and 4-phase motors
- Support PUL/DIR and CW/CCW modes
- Over-voltage, over-current, phase-error protections

Applications

Suitable for a wide range of stepping motors, from NEMA frame size 17 to 34. It can be used in various kinds of machines, such as laser cutters, laser markers, high precision X-Y tables, labeling machines, and so on. Its unique features make the DM556 an ideal solution for applications that require both low-speed smoothness and high speed performances.

Figura 14. Driver motor DM556

A.1.3. NEMA size 23 1.8° 2-phase stepper motor

En este anexo se muestran las características técnicas del motor paso a paso que se ha utilizado en este proyecto.

Quick Reference **NEMA size 23 1.8° 2-phase stepper motor**

Schneider Electric

Notes and Warnings

Installation, configuration and maintenance must be carried out by qualified technicians only. You must have detailed information to be able to carry out this work.

- Unexpected dangers may be encountered when working with this product!
- Incorrect use may destroy this product and connected components!

For more information, go to www.imshome.com

Specifications

2.4 Amp motors		Single length	Double length	Triple length
Part number		M-2218-2.4S *(1)*	M-2222-2.4S *(1)*	M-2231-2.4S *(1)*
Holding torque	oz-in	90	144	239
	N-cm	64	102	169
Detent torque	oz-in	3.9	5.6	9.7
	N-cm	2.7	3.9	6.9
Rotor inertia	oz-in-sec^2	0.00255	0.00368	0.0065
	kg-cm^2	0.18	0.26	0.468
Weight	oz	16.9	21.2	35.3
	grams	480	600	1000
Phase current	amps	2.4	2.4	2.4
Phase resistance	ohms	0.95	1.2	1.5
Phase inductance	mH	2.4	4.0	5.4

(1) Only available with single shaft.

3.0 Amp motors		Single length	Double length	Triple length
Part number		M-2218-3.0 • *(1)*	M-2222-3.0 • *(1)*	M-2231-3.0 • *(1)*
Holding torque	oz-in	90	144	239
	N-cm	64	102	169
Detent torque	oz-in	3.9	5.6	9.7
	N-cm	2.7	3.9	6.9
Rotor inertia	oz-in-sec^2	0.00255	0.00368	0.0065
	kg-cm^2	0.18	0.26	0.468
Weight	oz	16.9	21.2	35.3
	grams	480	600	1000
Phase current	amps	3.0	3.0	3.0
Phase resistance	ohms	0.65	0.85	0.95
Phase inductance	mH	1.5	2.6	3.36

(1) Indicate S for single-shaft or D for double-shaft. Example M-2218-3.0S

6.0 Amp motors		Single length	Double length	Triple length
Part number		M-2218-6.0 • *(1)*	M-2222-6.0 • *(1)*	M-2231-6.0 • *(1)*
Holding torque	oz-in	100	150	257
	N-cm	71	106	181
Detent torque	oz-in	2.0	3.0	5.0
	N-cm	1.4	2.1	3.5
Rotor inertia	oz-in-sec^2	0.0017	0.00397	0.0068
	kg-cm^2	0.12	0.28	0.48
Weight	oz	16.6	24.7	35.3
	grams	470	700	1000
Phase current	amps	6.0	6.0	6.0
Phase resistance	ohms	0.16	0.19	0.23
Phase inductance	mH	0.47	0.73	1.04

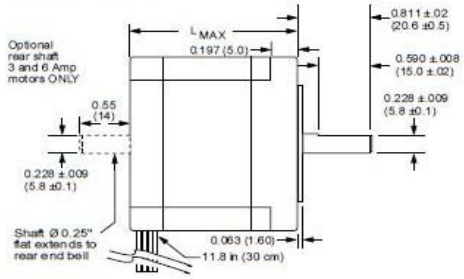
(1) Indicate S for single-shaft or D for double-shaft. Example M-2218-6.0S

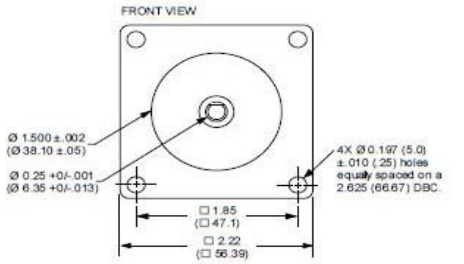
Wiring and Connections

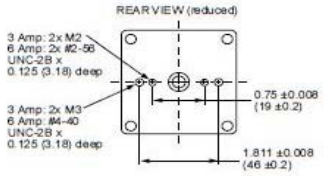
Signals and wire colors	2.4 Amp motors	3.0 Amp motors	6.0 Amp motors
Phase A	Red	Red	Black
Phase /A	White/red	White/red	Orange
Phase B	Green	Green	Red
Phase /B	White/green	White/green	Yellow

Mechanical Specifications

Dimensions in inches (mm)







Motor stack length inches (mm)	2.4 Amp motors	3.0 Amp motors	6.0 Amp motors
Single	1.77 (45)	1.77 (45)	1.75 (44.5)
Double	2.13 (54)	2.13 (54)	2.2 (56)
Triple	2.99 (76)	2.99 (76)	3.09 (78.5)

Part Numbers

Example:	M - 2 2 1 8 - 2 . 4 S
Stepper motor frame size **M -22** = NEMA 23 (2.3"/57 mm)	M - 2 2 1 8 - 2 . 4 S
Motor length **18 -** = single stack **22 -** = double stack **31 -** = triple stack	M - 2 2 1 **8** - 2 . 4 S
Phase current **2.4** = 2.4 Amps *(1)* **3.0** = 3.0 Amps **6.0** = 6.0 Amps	M - 2 2 1 8 - **2 . 4** S
Shaft **S** = single, front shaft only **D** = double, front and rear shafts	M - 2 2 1 8 - 2 . 4 **S**
Optional optical encoder *(2)* **ES** = Single-end **ED** = Differential	M - 2 2 1 8 - 2 . 4 **E S 1 0 0**
Line count 100, 200, 250, 400, 500 or 1000 *(3)*	

(1) Only available with single shaft.
(2) An encoder replaces the shaft designator in the part number.
(3) All encoders have an index mark, except the 1000 line count version.

NEMA 23 stepper motor Quick Reference R052710

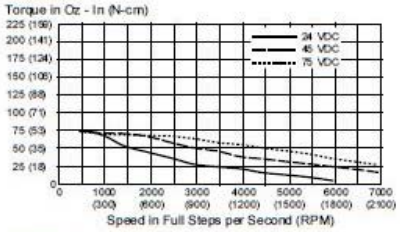
Torque-speed Performance

Measured at the rated phase current of the motor (RMS)

2.4 Amp motors

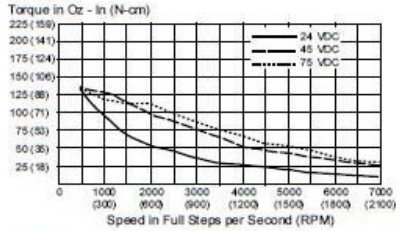
M-2218-2.4

Torque in Oz - In (N-cm)



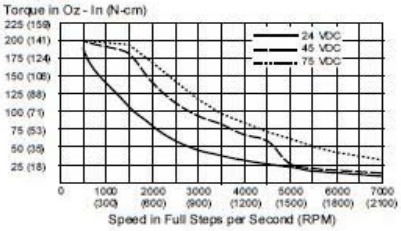
M-2222-2.4

Torque in Oz - In (N-cm)



M-2231-2.4

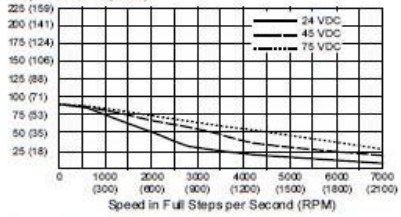
Torque in Oz - In (N-cm)



3.0 Amp motors

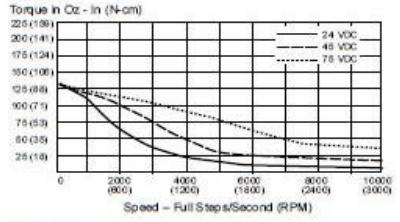
M-2218-3.0

Torque in Oz - In (N-cm)



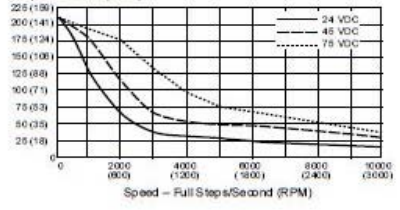
M-2222-3.0

Torque in Oz - In (N-cm)



M-2231-3.0

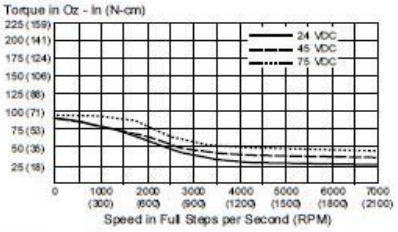
Torque in Oz - In (N-cm)



6.0 Amp motors

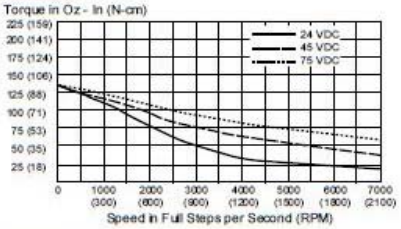
M-2218-6.0

Torque in Oz - In (N-cm)



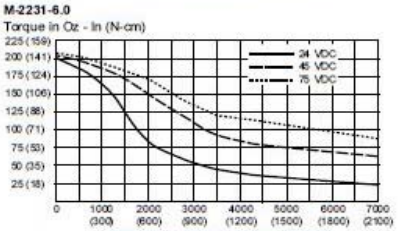
M-2222-6.0

Torque in Oz - In (N-cm)



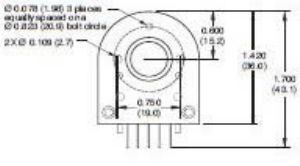
M-2231-6.0

Torque in Oz - In (N-cm)



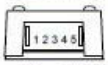
Optical Encoder Option

Dimensions in inches (mm)



Connectivity

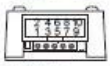
single-end encoder



wire	function
1 Brown	Ground
2 Violet	Index
3 Blue	Channel A
4 Orange	+5 VDC input
5 Yellow	Channel B

optional interface cable available: ES-CABLE-2

differential encoder

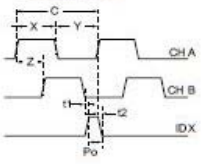


pin	function	pin	function
1	no connect	6	Channel A+
2	+5 VDC input	7	Channel B –
3	Ground	8	Channel B+
4	no connect	9	Index –
5	Channel A –	10	Index +

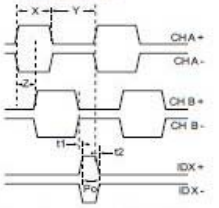
interface cable included

Timing

single-end encoder



differential encoder



Parameter	Symbol	Min	Typ	Max	Units
Cycle error			3	5.5	°e
Symmetry		130	180	230	°e
Quadrature		40	90	140	°e
Index pulse width	Po	60	90	120	°e
Index rise (after Ch A or B rise)	t1	-300	100	250	ns
Index fall (after Ch A or B fall)	t2	70	150	1000	ns

C One cycle: 360 electrical degrees (°e).
X/Y Symmetry: the measure of the relationship between X and Y, nominally 180°e.
Z Quadrature: the phase lead or lag between channels A and B, nominally 90°e.
Po Index pulse width, nominally 90 °e.
NOTE: Rotation is as viewed from the cover side of the encoder.

A.1.4. Drylin® SHTC – Flexible

Para finalizar el apartado de hardware, se comentarán las características técnicas más significativas de la mesa lineal utilizada como son la longitud del tornillo y el número de pasos que realiza el motor para recorrer la longitud del tornillo, con lo que podemos obtener la longitud recorrida en función de los pasos de motor.

El tornillo sin fin utilizado tiene una longitud de 750 milímetros, para que la plataforma recorra esa distancia con el motor paso a paso utilizado tiene que realizar un total de 146584 pasos.

drylin® SHT linear drive technology

drylin® SHT | Delivery Program | Trapezoidal Thread

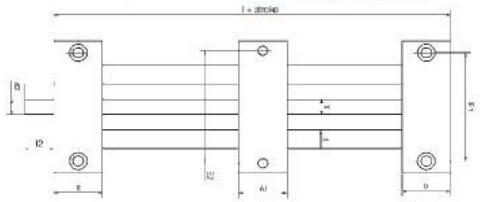
SHTC – Flexible

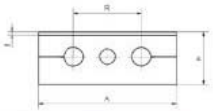
- High flexibility
- Ideal for 2 carriages
- Maintenance-free dry operation
- 5 types
- Adjustable bearing clearance
- Available accessories ▶ page 1203
- Lead screw nuts are available separately ▶ page 1096

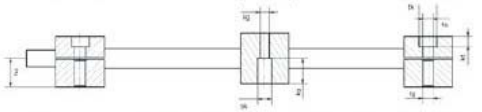
Order key
complete ▶ page 1172

SHTC-12-AWM
Shaft material
Type
Basic, compact

▶ page 1227







Technical Data

Part number	Max. length of stroke	Aluminum shaft		Steel shaft		Max. static load-bearing capacity	
		Weight	Additional (per 100 mm)	Weight	Additional (per 100 mm)	axial	radial
	[mm]	[kg]	[kg]	[kg]	[kg]	[N]	[N]
SHTC-12-AWM	750	0.7	0.1	0.8	0.2	700	2,800
SHTC-20-AWM	1,000	1.9	0.3	2.3	0.6	1,600	6,400
SHTC-30-AWM	1,250	4.6	0.6	5.8	1.4	2,500	10,000
SHTC-40-AWM	1,500	11.0	0.9	16.0	2.4	4,000	16,000
SHTC-50-AWM	1,500	17.0	1.2	26.3	3.5	6,250	25,000

Dimensions [mm]

Part number	A -0.3	Al -0.3	H	E1 ±0.15	E2 ±0.15	l	R	f	lt ±0.1	tk	ts	tg
SHTC-12-AWM	85	30	34	70	73	90	42	2	30	11	6.6	M8
SHTC-20-AWM	130	36	48	108	115	108	72	2	36	15	9.0	M10
SHTC-30-AWM	180	50	68	150	158	150	96	4	50	20	13.5	M16
SHTC-40-AWM	230	70	84	202	202	210	122	4	70	20	13.5	M16
SHTC-50-AWM	280	80	100	250	250	240	152	4	80	20	13.5	M16

Part number	kt ±0.1	sk	sg	kq	d	T	l2	d2 Standard	ha
SHTC-12-AWM	6.4	10	M6	6.0	12	TR10x2	17	TR10x2*	18
SHTC-20-AWM	8.6	11	M8	7.0	20	TR18x4	26	12h9	23
SHTC-30-AWM	12.6	18	M12	10.6	30	TR24x5	38	14h9	36
SHTC-40-AWM	12.6	20	M16	39	40	TR26x5	45	16h9	44
SHTC-50-AWM	12.6	20	M16	49	50	TR30x6	50	20h9	52

* TR10x2 lead screw end unmachined

1162 Lifetime calculation, CAD files and much more support ▶ www.igus.eu/eu/drylinSHT

ANEXO 2: CÓDIGO FUENTE DEL PROYECTO

A.2.1. CÓDIGO DE ARDUINO

```
//CONEXIÓN DE LOS PINES
//Control de motor
const int ENA = 10;
const int DIR = 11;
const int PUL = 12;

//Control de inicio y fin de carril
const int INICIO = 8;
const int FINAL = 9;

//Variables de control del movimiento
int Modo, ENABLE, POSICION1, POSICION2, NCiclos, j;
long i, Velocidad, VelocidadInicial;
long CicloActual, POS1, POS2, CiclosMax;
int TiempoPaP;

//Variable de control del paso del motor
boolean s = false;

//Variables de control de la comunicación
String Valores, trozo;


//FUNCIÓN SETUP
void setup()
{
 //Inicialización de variables
 VelocidadInicial=1100;
 CiclosMax = 146590;
 Valores = "0";

 //Inicialización del puerto serie
 Serial.begin(9600, SERIAL_8N1);

 //Definición de los pines de salida hacia el driver motor
 pinMode(ENA, OUTPUT);
 pinMode(DIR, OUTPUT);
 pinMode(PUL, OUTPUT);


 //Definición de los pines de los pulsadores de Fin de Carrera
 pinMode(INICIO, INPUT);
 pinMode(FINAL, INPUT);

 //Secuencia de inicio del driver
 digitalWrite(ENA, HIGH);
 digitalWrite(DIR, HIGH);
 delay(3000);
 digitalWrite(ENA, LOW);
 delayMicroseconds(10);



 //Movimiento de la plataforma a la posición inicial para comenzar el  control
mediante la interfaz
 //Se establece la dirección hacia la posición inicial
```

```
 digitalWrite(DIR, LOW);
 delayMicroseconds(10);

 //Movimiento de la plataforma a la posición inicial
 while ((digitalRead(INICIO) == HIGH))
 {
  digitalWrite(PUL, s);
  delayMicroseconds(VelocidadInicial);
  s = !(s);
 }
 //Inicialización de la variable de control de la posición de la plataforma.
 CicloActual = 0;

 //Desactivación del movimiento del motor
 digitalWrite(ENA, HIGH);
 delayMicroseconds(10);
}

//FUNCIÓN LOOP
void loop()
{
 if (Valores == "0")
 {
  Valores = Serial.readString();
  Serial.println(Valores);
 }

 if (Serial.available() > 0)
 {
  //Identificación de las variables enviadas desde Matlab y transformación
a valor numérico.
  ENABLE = Serial.readStringUntil('a').toInt();
  Velocidad = Serial.readStringUntil('b').toInt();
  POSICION1 = Serial.readStringUntil('c').toInt();
  POSICION2 = Serial.readStringUntil('d').toInt();
  Modo = Serial.readStringUntil('e').toInt();
  trozo = Serial.readString();
  NCiclos = trozo.toInt();

  //Transformación de variables a su valor verdadero
  Velocidad = ((398900/99)-((2900/99)*Velocidad));
  POS1 = POSICION1*(CiclosMax/10);
  POS2 = POSICION2*(CiclosMax/10);

  //MODO DE FUNCIONAMIENTO 1: Movimiento directo
  if (Modo == 1)
  {
   //Se activa el driver motor
   digitalWrite(ENA, LOW);
   delayMicroseconds(10);

   //Se comprueba si la posición a la que hay que ir es mayor que en la que
se está
   if (CicloActual < POS1)
   {


    //Se establece la dirección hacia la posición final
    digitalWrite(DIR, HIGH);
    delayMicroseconds(10);
```

```
    //Entrada en bucle for para ejecutar todos los pasos hasta la posición
de llegada
    for (i = CicloActual; i < (POS1 + 1); i = i + 1)
    {
     //Se comprueba que la plataforma no ha alcanzado el final
     if (digitalRead(FINAL) != LOW)
     {
      //Se envía un pulso al motor
      digitalWrite(PUL, s);
      delayMicroseconds(Velocidad);
      s = !(s);
      CicloActual = i;
     }
     //Si se ha alcanzado el final
     else
     {
      //Se establece la posición actual como la posición final
      CicloActual = CiclosMax;
      //Salida del bucle for
      break;
     }
    }
   }
   else
   {
    //Se establece la dirección hacia la posición final
    digitalWrite(DIR, LOW);
    delayMicroseconds(10);

    //Entrada en bucle for para ejecutar todos los pasos hasta la posición
de llegada
    for (i = CicloActual; i > (POS1 + 1); i = i - 1)
    {
     //Se comprueba que la plataforma no ha alcanzado el inicio
     if (digitalRead(INICIO) != LOW)
     {
      //Se envía un pulso al motor
      digitalWrite(PUL, s);
      delayMicroseconds(Velocidad);
      s = !(s);
      CicloActual = i;
     }
     //Si se ha alcanzado el inicio
     else
     {
      //Se establece la posición actual como la posición inicial
      CicloActual = 0;
      //Salida del bucle for
      break;
     }
    }
   }
   //Se desactiva el driver motor
   digitalWrite(ENA, HIGH);
   delayMicroseconds(10);
  }

  //MODO DE FUNCIONAMIENTO 2: Movimiento cíclico
  else if (Modo==2)
  {
   digitalWrite(ENA, LOW);
```

```
delayMicroseconds(10);

for (j = 1; j < NCiclos+1; j=j+1)
{
 if (CicloActual < POS1)
 {
  digitalWrite(DIR, HIGH);
  delayMicroseconds(10);
  for (i = CicloActual; i < POS1; i = i + 1)
  {
   if (digitalRead(FINAL) != LOW)
   {
    digitalWrite(PUL, s);
    delayMicroseconds(Velocidad);
    s = !(s);
    CicloActual = i;
   }
   else
   {
    CicloActual = CiclosMax;
    break;
   }
  }

  for (i = CicloActual; i < POS2; i = i + 1)
  {
   if (digitalRead(FINAL) != LOW)
   {
    digitalWrite(PUL, s);
    delayMicroseconds(Velocidad);
    s = !(s);
    CicloActual = i;
   }
   else
   {
    CicloActual = CiclosMax;
    break;
   }
  }

  digitalWrite(DIR, LOW);
  delayMicroseconds(10);

  for (i = CicloActual; i > POS1; i = i - 1)
  {
   if (digitalRead(INICIO) != LOW)
   {
    digitalWrite(PUL, s);
    delayMicroseconds(Velocidad);
    s = !(s);
    CicloActual = i;
   }
   else
   {
    CicloActual = 0;
    break;
   }
  }
 }

 else
```

```
 {
  digitalWrite(DIR, LOW);
  delayMicroseconds(10);

  for (i = CicloActual; i > POS1; i = i - 1)
  {
   if (digitalRead(INICIO) != LOW)
   {
    digitalWrite(PUL, s);
    delayMicroseconds(Velocidad);
    s = !(s);
    CicloActual = i;
   }
   else
   {
    CicloActual = 0;
    break;
   }
  }

  digitalWrite(DIR, HIGH);
  delayMicroseconds(10);

  for (i = CicloActual; i < POS2; i = i + 1)
  {
   if (digitalRead(FINAL) != LOW)
   {
    digitalWrite(PUL, s);
    delayMicroseconds(Velocidad);
    s = !(s);
    CicloActual = i;
   }
   else
   {
    CicloActual = CiclosMax;
    break;
   }
  }

  digitalWrite(DIR, LOW);
  delayMicroseconds(10);

  for (i = CicloActual; i > POS1; i = i - 1)
  {
   if (digitalRead(INICIO) != LOW)
   {
    digitalWrite(PUL, s);
    delayMicroseconds(Velocidad);
    s = !(s);
    CicloActual = i;
   }
   else
   {
    CicloActual = 0;
    break;
   }
  }
 }
}
j=1;
digitalWrite(ENA, HIGH);
```

```
    delayMicroseconds(10);
   }

   //MODO DE FUNCIONAMIENTO 3: Movimiento directo a la posición inicial
   else if (Modo==3)
   {
    digitalWrite(ENA, LOW);
    delayMicroseconds(10);

    digitalWrite(DIR, LOW);
    delayMicroseconds(10);

    while ((digitalRead(INICIO) == HIGH))
    {
     digitalWrite(PUL, s);
     delayMicroseconds(VelocidadInicial);
     s = !(s);
    }
       CicloActual = 0;
    digitalWrite(ENA, HIGH);
   }

   //MODO DE FUNCIONAMIENTO 4: Movimiento directo a la posición final
   else if (Modo==4)
   {
    digitalWrite(ENA, LOW);
    delayMicroseconds(10);

    digitalWrite(DIR, HIGH);
    delayMicroseconds(10);

    while ((digitalRead(FINAL) == HIGH))
    {
     digitalWrite(PUL, s);
     delayMicroseconds(VelocidadInicial);
     s = !(s);
    }

    CicloActual = CiclosMax;
    digitalWrite(ENA, HIGH);
   }
  }
}
```

A.2.1. CÓDIGO DE MATLAB

```
function varargout = Interfaz(varargin)
%*****************************************************************
%*Autor: Álvaro Sierra Díez       Fecha: 05/01/2017      *
%*Estudios: Grado en Ingeniería Electrónica Industrial y Automatismos*
%*Programa: Interfaz de control de un motor paso a paso.       *
%*Función: Envío de una serie de datos al controlador Arduino    *
%*     mediante el uso de una interfaz gráfica.          *
%*****************************************************************

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
           'gui_Singleton', gui_Singleton, ...
           'gui_OpeningFcn', @Interfaz_OpeningFcn, ...
           'gui_OutputFcn', @Interfaz_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

% End initialization code - DO NOT EDIT

%Función que se ejecutará antes de que la ventana de la interfaz sea visible.
function Interfaz_OpeningFcn(hObject, eventdata, handles, varargin)

  axes(handles.ImgUC)
  handles.imagen=imread('UC.jpg');
  imagesc(handles.imagen)
  axis off
  handles.output = hObject;     %Se asigna el identificador para nuestra
Interfaz

  guidata(hObject, handles);    %Se realiza un salvado de los datos de la
aplicación

%Outputs from this function are returned to the command line.
function varargout = Interfaz_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

%Función que se ejecutará cuando se presione el boton de encendido.
function Encendido_Callback(hObject, eventdata, handles)

Inicio=get(hObject,'Value'); %Comprobación del botón de encendido

if Inicio==1

  set(handles.Encendido,'BackgroundColor','g'); %Color de fondo verde
  set(handles.Encendido,'String','ON'); %Botón-> ON
else
```

```
  set(handles.Encendido,'BackgroundColor','r'); %Color de fondo rojo
  set(handles.Encendido,'String','OFF'); %Botón-> OFF
  set(handles.Encendido,'Value',0); %Valor por defecto igual a cero
end

%Función referente a la posición 0
function P0_Callback(hObject, eventdata, handles)

%Función referente a la posición 10
function P10_Callback(hObject, eventdata, handles)

%Función referente a la posición 20
function P20_Callback(hObject, eventdata, handles)

%Función referente a la posición 30
function P30_Callback(hObject, eventdata, handles)

%Función referente a la posición 40
function P40_Callback(hObject, eventdata, handles)

%Función referente a la posición 50
function P50_Callback(hObject, eventdata, handles)

%Función referente a la posición 60
function P60_Callback(hObject, eventdata, handles)

%Función referente a la posición 70
function P70_Callback(hObject, eventdata, handles)

%Función referente a la posición 80
function P80_Callback(hObject, eventdata, handles)

%Función referente a la posición 90
function P90_Callback(hObject, eventdata, handles)

%Función referente a la posición 100
function P100_Callback(hObject, eventdata, handles)

%Función referente al botón Posición Final
function P_Final_Callback(hObject, eventdata, handles)
%Lectura del botón de arranque de la interfaz
if get(handles.Encendido,'Value')==1
  ENABLE=1;
  %Lectura y transformación a valor numérico de la velocidad
  Velocidad=str2double(get(handles.Frec,'String'));
  %Si el valor de la velocidad no está dentro del rango->ERROR
  if (Velocidad>100 || Velocidad<1)
    errordlg('El valor de la velocidad es incorrecto. Asegurese que se
encuentra dentro del rango disponible','ERROR');
  %Si la velocidad no ha sido especificada->ERROR
  elseif isnan(Velocidad)
    errordlg('Introduzca el valor de la velocidad.','ERROR')
  %Si el valor de velocidad es correcto
  else
    %Establecimiento de parámetros para realizar el movimiento
    POSICION1=10;POSICION2=0;Modo=4;Ciclos=0;
       %Llamada a la función de envío de datos a Arduino
       datos=Envio_Var(ENABLE,Velocidad,POSICION1,POSICION2,Modo,Ciclos);
       %Muestra en la interfaz la información enviada
```

```
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String',{'Movimiento hacia la posición
final.','Cadena enviada a Arduino:',datos});
  end
%Si el botón de arranque no está activado->ERROR
else
  errordlg('El botón de arranque debe estar presionado.','ERROR');
end

%Función referente al botón Posición Inicial
function P_Inicial_Callback(hObject, eventdata, handles)
%Lectura del botón de arranque de la interfaz
if get(handles.Encendido,'Value')==1
  ENABLE=1;
  %Lectura y transformación a valor numérico de la velocidad
  Velocidad=str2double(get(handles.Frec,'String'));
  %Si el valor de la velocidad no está dentro del rango->ERROR
  if (Velocidad>100 || Velocidad<1)
    errordlg('El valor de la velocidad es incorrecto. Asegurese que se
encuentra dentro del rango disponible','ERROR');
  %Si la velocidad no ha sido especificada->ERROR
  elseif isnan(Velocidad)
    errordlg('Introduzca el valor de la velocidad.','ERROR')
  %Si el valor de velocidad es correcto
  else
     %Establecimiento de parámetros para realizar el movimiento
    POSICION1=0;POSICION2=0;Modo=3;Ciclos=0;
       %Llamada a la función de envío de datos a Arduino
       datos=Envio_Var(ENABLE,Velocidad,POSICION1,POSICION2,Modo,Ciclos);
       %Muestra en la interfaz la información enviada
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String',{'Movimiento hacia la posición
inicial.','Cadena enviada a Arduino:',datos});
  end
%Si el botón de arranque no está activado->ERROR
else
  errordlg('El botón de arranque debe estar presionado.','ERROR');
end


% --- Executes on button press in Ejecutar.
function Ejecutar_Callback(hObject, eventdata, handles)
%Lectura del botón de arranque de la interfaz
if get(handles.Encendido,'Value')==1
  ENABLE=1;
  %Lectura y transformación a valor numérico de la velocidad
  Velocidad=str2double(get(handles.Frec,'String'));
  %Si el valor de la velocidad no está dentro del rango->ERROR
  if (Velocidad>100 || Velocidad<1)
    errordlg('El valor de la velocidad es incorrecto. Asegurese que se
encuentra dentro del rango disponible','ERROR');
  %Si la velocidad no ha sido especificada->ERROR
  elseif isnan(Velocidad)
    errordlg('Introduzca el valor de la velocidad.','ERROR')
  %Si el valor de velocidad es correcto
  else
  switch get(get(handles.Selec_Pos,'SelectedObject'),'Tag')
    case 'M_Directo'
         %Establecimiento de parámetros para realizar el movimiento
      Modo=1;POSICION2=0;Ciclos=0;
```

```
    %Lectura del pulsador de la posición 0
if get(handles.P0,'Value')==1
      %Se establece la posición de destino de la plataforma
 POSICION1=0;
      %Muestra en la interfaz la posición seleccionada
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento hacia P0.');

%Lectura del pulsador de la posición 10
elseif get(handles.P10,'Value')==1
 %Se establece la posición de destino de la plataforma
 POSICION1=1;
      %Muestra en la interfaz la posición seleccionada
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento hacia P10.');

%Lectura del pulsador de la posición 20
elseif get(handles.P20,'Value')==1
      %Se establece la posición de destino de la plataforma
 POSICION1=2;
      %Muestra en la interfaz la posición seleccionada
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento hacia P20.');

%Lectura del pulsador de la posición 30
elseif get(handles.P30,'Value')==1
      %Se establece la posición de destino de la plataforma
 POSICION1=3;
      %Muestra en la interfaz la posición seleccionada
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento hacia P30.');

%Lectura del pulsador de la posición 40
elseif get(handles.P40,'Value')==1
      %Se establece la posición de destino de la plataforma
 POSICION1=4;
      %Muestra en la interfaz la posición seleccionada
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento hacia P40.');

%Lectura del pulsador de la posición 50
elseif get(handles.P50,'Value')==1
      %Se establece la posición de destino de la plataforma
 POSICION1=5;
      %Muestra en la interfaz la posición seleccionada
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento hacia P50.');

%Lectura del pulsador de la posición 60
elseif get(handles.P60,'Value')==1
      %Se establece la posición de destino de la plataforma
 POSICION1=6;
      %Muestra en la interfaz la posición seleccionada
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento hacia P60.');

%Lectura del pulsador de la posición 70
elseif get(handles.P70,'Value')==1
    %Se establece la posición de destino de la plataforma
 POSICION1=7;
      Muestra en la interfaz la posición seleccionada
```

```
        set(handles.VentanaEjec,'String','');
        set(handles.VentanaEjec,'String','Movimiento hacia P70.');

      %Lectura del pulsador de la posición 80
      elseif get(handles.P80,'Value')==1
          %Se establece la posición de destino de la plataforma
       POSICION1=8;
          %Muestra en la interfaz la posición seleccionada
       set(handles.VentanaEjec,'String','');
       set(handles.VentanaEjec,'String','Movimiento hacia P80.');

      %Lectura del pulsador de la posición 90
      elseif get(handles.P90,'Value')==1
          %Se establece la posición de destino de la plataforma
       POSICION1=9;
             Muestra en la interfaz la posición seleccionada
       set(handles.VentanaEjec,'String','');
       set(handles.VentanaEjec,'String','Movimiento hacia P90.');

      %Lectura del pulsador de la posición 100
      elseif get(handles.P100,'Value')==1
          %Se establece la posición de destino de la plataforma
       POSICION1=10;
             Muestra en la interfaz la posición seleccionada
       set(handles.VentanaEjec,'String','');
       set(handles.VentanaEjec,'String','Movimiento hacia P100.');

         %Si no se ha seleccionado ninguna posición->ERROR
      else
       errordlg('No se ha seleccionado ninguna posición.','ERROR')
      end

      %Llamada a la función de envío de datos a Arduino
      datos=Envio_Var(ENABLE,Velocidad,POSICION1,POSICION2,Modo,Ciclos);
         %Muestra en la pantalla de la interfaz la cadena enviada a Arduino
      texto=get(handles.VentanaEjec,'String');
      set(handles.VentanaEjec,'String',{texto,'Cadena  enviada  a  Arduino:',
datos});

   case 'M_Ciclico'
      Modo=2;
      Ciclos=str2double(get(handles.NCiclos,'String'));
      if (Ciclos<0)
      errordlg('El número de ciclos debe ser un número positivo', 'ERROR');
      elseif isnan(Ciclos)
      errordlg('Introduzca el número de ciclos que desea realizar.', 'ERROR')
      else
      %Lectura de los pulsadores de las posiciones 0 y 20
      if (get(handles.P0,'Value')==1 && get(handles.P10,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
       POSICION1=0;POSICION2=1;
       set(handles.VentanaEjec,'String','');
       set(handles.VentanaEjec,'String','Movimiento entre P0 y P10.');

      %Lectura de los pulsadores de las posiciones 0 y 20
      elseif (get(handles.P0,'Value')==1 && get(handles.P20,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
       POSICION1=0;POSICION2=2;
       set(handles.VentanaEjec,'String','');
       set(handles.VentanaEjec,'String','Movimiento entre P0 y P20.');
```

```
    %Lectura de los pulsadores de las posiciones 0 y 30
    elseif (get(handles.P0,'Value')==1 && get(handles.P30,'Value')==1)
        %Se establece las posiciones de destino de la plataforma
     POSICION1=0;POSICION2=3;
     set(handles.VentanaEjec,'String','');
     set(handles.VentanaEjec,'String','Movimiento entre P0 y P30.');

    %Lectura de los pulsadores de las posiciones 0 y 40
    elseif (get(handles.P0,'Value')==1 && get(handles.P40,'Value')==1)
        %Se establece las posiciones de destino de la plataforma
     POSICION1=0;POSICION2=4;
     set(handles.VentanaEjec,'String','');
     set(handles.VentanaEjec,'String','Movimiento entre P0 y P40.');

    %Lectura de los pulsadores de las posiciones 0 y 50
    elseif (get(handles.P0,'Value')==1 && get(handles.P50,'Value')==1)
        %Se establece las posiciones de destino de la plataforma
     POSICION1=0;POSICION2=5;
     set(handles.VentanaEjec,'String','');
     set(handles.VentanaEjec,'String','Movimiento entre P0 y P50.');

    %Lectura de los pulsadores de las posiciones 0 y 60
    elseif (get(handles.P0,'Value')==1 && get(handles.P60,'Value')==1)
        %Se establece las posiciones de destino de la plataforma
     POSICION1=0;POSICION2=6;
     set(handles.VentanaEjec,'String','');
     set(handles.VentanaEjec,'String','Movimiento entre P0 y P60.');

    %Lectura de los pulsadores de las posiciones 0 y 70
    elseif (get(handles.P0,'Value')==1 && get(handles.P70,'Value')==1)
        %Se establece las posiciones de destino de la plataforma
     POSICION1=0;POSICION2=7;
     set(handles.VentanaEjec,'String','');
     set(handles.VentanaEjec,'String','Movimiento entre P0 y P70.');

    %Lectura de los pulsadores de las posiciones 0 y 80
    elseif (get(handles.P0,'Value')==1 && get(handles.P80,'Value')==1)
        %Se establece las posiciones de destino de la plataforma
     POSICION1=0;POSICION2=8;
     set(handles.VentanaEjec,'String','');
     set(handles.VentanaEjec,'String','Movimiento entre P0 y P80.');

    %Lectura de los pulsadores de las posiciones 0 y 90
    elseif (get(handles.P0,'Value')==1 && get(handles.P90,'Value')==1)
        %Se establece las posiciones de destino de la plataforma
     POSICION1=0;POSICION2=9;
     set(handles.VentanaEjec,'String','');
     set(handles.VentanaEjec,'String','Movimiento entre P0 y P90.');

   %Lectura de los pulsadores de las posiciones 0 y 100
   elseif (get(handles.P0,'Value')==1 && get(handles.P100,'Value')==1)
        %Se establece las posiciones de destino de la plataforma
     POSICION1=0;POSICION2=10;
     set(handles.VentanaEjec,'String','');
     set(handles.VentanaEjec,'String','Movimiento entre P0 y P100.');

    %Lectura de los pulsadores de las posiciones 10 y 20
   elseif (get(handles.P10,'Value')==1 && get(handles.P20,'Value')==1)
        %Se establece las posiciones de destino de la plataforma
     POSICION1=1;POSICION2=2;
     set(handles.VentanaEjec,'String','');
```

```
    set(handles.VentanaEjec,'String','Movimiento entre P10 y P20.');

  %Lectura de los pulsadores de las posiciones 10 y 30
  elseif (get(handles.P10,'Value')==1 && get(handles.P30,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=1;POSICION2=3;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P10 y P30.');

  %Lectura de los pulsadores de las posiciones 10 y 40
  elseif (get(handles.P10,'Value')==1 && get(handles.P40,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=1;POSICION2=4;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P10 y P40.');

  %Lectura de los pulsadores de las posiciones 10 y 50
  elseif (get(handles.P10,'Value')==1 && get(handles.P50,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=1;POSICION2=5;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P10 y P50.');

  %Lectura de los pulsadores de las posiciones 10 y 60
  elseif (get(handles.P10,'Value')==1 && get(handles.P60,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=1;POSICION2=6;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P10 y P60.');

  %Lectura de los pulsadores de las posiciones 10 y 70
  elseif (get(handles.P10,'Value')==1 && get(handles.P70,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=1;POSICION2=7;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P10 y P70.');

  %Lectura de los pulsadores de las posiciones 10 y 80
  elseif (get(handles.P10,'Value')==1 && get(handles.P80,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=1;POSICION2=8;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P10 y P80.');

  %Lectura de los pulsadores de las posiciones 10 y 90
  elseif (get(handles.P10,'Value')==1 && get(handles.P90,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=1;POSICION2=9;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P10 y P90.');

  %Lectura de los pulsadores de las posiciones 10 y 100
  elseif (get(handles.P10,'Value')==1 && get(handles.P100,'Value')==1)
      %Se establece las posiciones de destino de la plataforma
   POSICION1=1;POSICION2=10;
   set(handles.VentanaEjec,'String','');
   set(handles.VentanaEjec,'String','Movimiento entre P10 y P100.');

  %Lectura de los pulsadores de las posiciones 20 y 30
  elseif (get(handles.P20,'Value')==1 && get(handles.P30,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
```

```
    POSICION1=2;POSICION2=3;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P20 y P30.');

  %Lectura de los pulsadores de las posiciones 20 y 40
  elseif (get(handles.P20,'Value')==1 && get(handles.P40,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=2;POSICION2=4;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P20 y P40.');

  %Lectura de los pulsadores de las posiciones 20 y 50
  elseif (get(handles.P20,'Value')==1 && get(handles.P50,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=2;POSICION2=5;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P20 y P50.');

  %Lectura de los pulsadores de las posiciones 20 y 60
  elseif (get(handles.P20,'Value')==1 && get(handles.P60,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=2;POSICION2=6;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P20 y P60.');

  %Lectura de los pulsadores de las posiciones 20 y 70
  elseif (get(handles.P20,'Value')==1 && get(handles.P70,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=2;POSICION2=7;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P20 y P70.');

  %Lectura de los pulsadores de las posiciones 20 y 80
  elseif (get(handles.P20,'Value')==1 && get(handles.P80,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=2;POSICION2=8;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P20 y P80.');

  %Lectura de los pulsadores de las posiciones 20 y 90
  elseif (get(handles.P20,'Value')==1 && get(handles.P90,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=2;POSICION2=9;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P20 y P90.');

  %Lectura de los pulsadores de las posiciones 20 y 100
  elseif (get(handles.P20,'Value')==1 && get(handles.P100,'Value')==1)
      %Se establece las posiciones de destino de la plataforma
   POSICION1=2;POSICION2=10;
   set(handles.VentanaEjec,'String','');
   set(handles.VentanaEjec,'String','Movimiento entre P20 y P100.');

  %Lectura de los pulsadores de las posiciones 30 y 40
  elseif (get(handles.P30,'Value')==1 && get(handles.P40,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=3;POSICION2=4;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P30 y P40.');

  %Lectura de los pulsadores de las posiciones 30 y 50
```

```
elseif (get(handles.P30,'Value')==1 && get(handles.P50,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=3;POSICION2=5;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P30 y P50.');

%Lectura de los pulsadores de las posiciones 30 y 60
elseif (get(handles.P30,'Value')==1 && get(handles.P60,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=3;POSICION2=6;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P30 y P60.');

%Lectura de los pulsadores de las posiciones 30 y 70
elseif (get(handles.P30,'Value')==1 && get(handles.P70,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=3;POSICION2=7;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P30 y P70.');

%Lectura de los pulsadores de las posiciones 30 y 80
elseif (get(handles.P30,'Value') && get(handles.P80,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=3;POSICION2=8;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P30 y P80.');

%Lectura de los pulsadores de las posiciones 30 y 90
elseif (get(handles.P30,'Value')==1 && get(handles.P90,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=3;POSICION2=9;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P30 y P90.');

%Lectura de los pulsadores de las posiciones 30 y 100
elseif (get(handles.P30,'Value')==1 && get(handles.P100,'Value')==1)
    %Se establece las posiciones de destino de la plataforma
 POSICION1=3;POSICION2=10;
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento entre P30 y P100.');

%Lectura de los pulsadores de las posiciones 40 y 50
elseif (get(handles.P40,'Value')==1 && get(handles.P50,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=4;POSICION2=5;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P40 y P50.');

%Lectura de los pulsadores de las posiciones 40 y 60
elseif (get(handles.P40,'Value')==1 && get(handles.P60,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=4;POSICION2=6;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P40 y P60.');

%Lectura de los pulsadores de las posiciones 40 y 70
elseif (get(handles.P40,'Value')==1 && get(handles.P70,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=4;POSICION2=7;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P40 y P70.');
```

```
%Lectura de los pulsadores de las posiciones 40 y 80
elseif (get(handles.P40,'Value')==1 && get(handles.P80,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=4;POSICION2=8;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P40 y P80.');

%Lectura de los pulsadores de las posiciones 40 y 90
elseif (get(handles.P40,'Value')==1 && get(handles.P90,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=4;POSICION2=9;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P40 y P90.');

%Lectura de los pulsadores de las posiciones 40 y 100
elseif (get(handles.P40,'Value')==1 && get(handles.P100,'Value')==1)
    %Se establece las posiciones de destino de la plataforma
 POSICION1=4;POSICION2=10;
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento entre P40 y P100.');

%Lectura de los pulsadores de las posiciones 50 y 60
elseif (get(handles.P50,'Value')==1 && get(handles.P60,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=5;POSICION2=6;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P50 y P60.');

%Lectura de los pulsadores de las posiciones 50 y 70
elseif (get(handles.P50,'Value')==1 && get(handles.P70,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=5;POSICION2=7;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P50 y P70.');

%Lectura de los pulsadores de las posiciones 50 y 80
elseif (get(handles.P50,'Value')==1 && get(handles.P80,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=5;POSICION2=8;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P50 y P80.');

%Lectura de los pulsadores de las posiciones 50 y 90
elseif (get(handles.P50,'Value')==1 && get(handles.P90,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=5;POSICION2=9;
  set(handles.VentanaEjec,'String','');
  set(handles.VentanaEjec,'String','Movimiento entre P50 y P90.');

%Lectura de los pulsadores de las posiciones 50 y 100
elseif (get(handles.P50,'Value')==1 && get(handles.P100,'Value')==1)
    %Se establece las posiciones de destino de la plataforma
 POSICION1=5;POSICION2=10;
 set(handles.VentanaEjec,'String','');
 set(handles.VentanaEjec,'String','Movimiento entre P50 y P100.');

%Lectura de los pulsadores de las posiciones 60 y 70
elseif (get(handles.P60,'Value')==1 && get(handles.P70,'Value')==1)
     %Se establece las posiciones de destino de la plataforma
  POSICION1=6;POSICION2=7;
```

```
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P60 y P70.');

  %Lectura de los pulsadores de las posiciones 60 y 80
  elseif (get(handles.P60,'Value')==1 && get(handles.P80,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=6;POSICION2=8;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P60 y P80.');

  %Lectura de los pulsadores de las posiciones 60 y 90
  elseif (get(handles.P60,'Value')==1 && get(handles.P90,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=6;POSICION2=9;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P60 y P90.');

  %Lectura de los pulsadores de las posiciones 60 y 100
  elseif (get(handles.P60,'Value')==1 && get(handles.P100,'Value')==1)
      %Se establece las posiciones de destino de la plataforma
   POSICION1=6;POSICION2=10;
   set(handles.VentanaEjec,'String','');
   set(handles.VentanaEjec,'String','Movimiento entre P60 y P100.');

  %Lectura de los pulsadores de las posiciones 70 y 80
  elseif (get(handles.P70,'Value')==1 && get(handles.P80,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=7;POSICION2=8;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P70 y P80.');

  %Lectura de los pulsadores de las posiciones 70 y 90
  elseif (get(handles.P70,'Value')==1 && get(handles.P90,'Value')==1)
       %Se establece las posiciones de destino de la plataforma
    POSICION1=7;POSICION2=9;
    set(handles.VentanaEjec,'String','');
    set(handles.VentanaEjec,'String','Movimiento entre P70 y P90.');

  %Lectura de los pulsadores de las posiciones 70 y 100
  elseif (get(handles.P70,'Value')==1 && get(handles.P100,'Value')==1)
      %Se establece las posiciones de destino de la plataforma
   POSICION1=7;POSICION2=10;
   set(handles.VentanaEjec,'String','');
   set(handles.VentanaEjec,'String','Movimiento entre P70 y P100.');

  %Lectura de los pulsadores de las posiciones 80 y 90
  elseif (get(handles.P80,'Value')==1 && get(handles.P90,'Value')==1)
      %Se establece las posiciones de destino de la plataforma
   POSICION1=8;POSICION2=9;
   set(handles.VentanaEjec,'String','');
   set(handles.VentanaEjec,'String','Movimiento entre P80 y P90.');

  %Lectura de los pulsadores de las posiciones 80 y 100
  elseif (get(handles.P80,'Value')==1 && get(handles.P100,'Value')==1)
   %Se establece las posiciones de destino de la plataforma
   POSICION1=8;POSICION2=10;
   set(handles.VentanaEjec,'String','');
   set(handles.VentanaEjec,'String','Movimiento entre P80 y P100.');

 %Lectura de los pulsadores de las posiciones 90 y 100
 elseif (get(handles.P90,'Value')==1 && get(handles.P100,'Value')==1)
```

```
        %Se establece las posiciones de destino de la plataforma
     POSICION1=9;POSICION2=10;
     set(handles.VentanaEjec,'String','');
     set(handles.VentanaEjec,'String','Movimiento entre P90 y P100.');

   %Si no se ha seleccionado al menos dos posiciones->ERROR
   else
     errordlg('No se han seleccionado correctamente las posiciones.',
'ERROR')
   end
      %Llamada a la función de envío de datos a Arduino
      datos=Envio_Var(ENABLE,Velocidad,POSICION1,POSICION2,Modo,Ciclos);
      %Muestra en la pantalla de la interfaz la cadena enviada a Arduino
      texto=get(handles.VentanaEjec,'String');
      set(handles.VentanaEjec,'String',{texto,'Cadena enviada a Arduino:',
datos});

   end
  end
 end
 guidata(hObject,handles);

%Si el botón de arranque no está activado->ERROR
else
  errordlg('El botón de arranque debe estar presionado.','ERROR');
end

function P_Posiciones_CreateFcn(hObject, eventdata, handles)

function Selec_Pos_SelectionChangeFcn(hObject, eventdata, handles)

function Frec_Callback(hObject, eventdata, handles)

function Frec_CreateFcn(hObject, eventdata, handles)

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

function NCiclos_Callback(hObject, eventdata, handles)

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

function AyudaMD_Callback(hObject, eventdata, handles)
  helpdlg('Para activar el modo directo-> 1:Presionar el botón de arranque.
2:Seleccionar el modo de movimiento directo. 3:Seleccionar una posición.
4:Presionar el boton "Ejecutar Movimiento".','AYUDA');

function AyudaMC_Callback(hObject, eventdata, handles)
  helpdlg('Para activar el modo cíclico-> 1:Presionar el botón de arranque.
2:Seleccionar el modo de movimiento cíclico. 3:Seleccionar dos posiciones.
4:Presionar el boton "Ejecutar Movimiento".','AYUDA');

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

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

function       [datos]=Envio_Var(ENABLE,Velocidad,POSICION1,POSICION2,Modo,
NCiclos)
%Transformación de las variables numéricas a cadena de caracteres
ENABLE=num2str(ENABLE);
Velocidad=num2str(Velocidad);
POSICION1=num2str(POSICION1);
POSICION2=num2str(POSICION2);
Modo=num2str(Modo);
NCiclos=num2str(NCiclos);

%Concatenación de variables para crear una cadena de caracteres única
datos=strcat(ENABLE,'a',Velocidad,'b',POSICION1,'c',POSICION2,'d',Modo,'e',
NCiclos);
%Borrar conexiones previas
delete(instrfind('Port','COM3'));
%Se crea el objeto serie que nos permitirá comunicar Matlab con Arduino
arduino = serial('COM3','BaudRate',9600,'Terminator','CR/LF');
arduino.StopBits=1;
fopen(arduino);
fprintf(arduino,'%s',datos);
a=fscanf(arduino,'%s');
fprintf(arduino,'%s',datos);
fclose(arduino);
delete(arduino)
clear arduino
```

DOCUMENTO III:
PRESUPUESTO

Índice del presupuesto

CAPÍTULO 1: PRESUPUESTO DE EJECUCIÓN DE MATERIAL

1.1. Hardware

Designación	Nº Unidades	Precio Unitario	Precio Total
Mesa lineal Drylin SHTC-Flexible + Motor paso a paso NEMA size 23 1.8° 2-phase	1	704,97 €	704,62 €
Fully Digital Stepping Driver DM556	1	56,41 €	56,41 €
Arduino Uno	1	20,00 €	20,00 €
Fuente de alimentación de 36V RS-320-36	1	81,93 €	81,93 €
		Subtotal	**781,03 €**

Tabla 4. Costes del hardware

1.2. Otros materiales

Designación	Nº Unidades	Precio Unitario	Precio Total
Pulsador	2	1,50 €	3,00 €
Resistencia de 330kΩ	2	0,20 €	0,40 €
Condensador de 100µF	1	0,50 €	0,50 €
		Subtotal	**3,90 €**

Tabla 5. Coste de materiales adicionales

1.3. Mano de obra

Designación	Nº Horas	Precio Unitario	Precio Total
Mano de obra	50	25,00 €	1.250,00 €
		Subtotal	**1.250,00 €**

Precio del Ingeniero: 25€/hora

1.4. Cálculo general de ejecución de material

Coste total de los materiales……..……784,93€

Coste total de la mano de obra……1250,00€

Total **2.034,93€**

El presupuesto de ejecución de material asciende a dos mil treinta y cuatro euros con noventa y tres céntimos.

Santander, 15 de marzo de 2017

El ingeniero:

Fdo.:

Álvaro Sierra Díez

BIBLIOGRAFÍA

[1] Alciro.org. (2017). *MOTORES PASO A PASO DE RELUCTANCIA VARIABLE*. [online] Disponible en: http://alciro.org/alciro/Plotter-Router-Fresadora-CNC_1/Motores-reluctancia-variable_42.htm

[2] Allaboutee.com. (2017). *HOW TO SEND DATA FROM ARDUINO TO MATLAB*. [online] Disponible en: http://allaboutee.com/2011/07/04/how-to-send-data-from-the-arduino-to-matlab/

[3] DIYMakers. (2017). *MOTORES PASO A PASO CON ARDUINO*. [online] Disponible en: http://diymakers.es/mover-motores-paso-paso-con-arduino/

[4] Dspace.espol.edu.ec. (2017). *MANUAL DE INTERFAZ GRÁFICA EN MATLAB*. [online] Disponible en: https://www.dspace.espol.edu.ec/bitstream/123456789/10740/11/MATLAB_GUIDE.pdf

[5] Forum.arduino.cc. (2017). *AUTORESET DE ARDUINO AL CONECTAR PUERTO SERIE*. [online] Disponible en: http://forum.arduino.cc/index.php/topic,158630.0.html

[6] Luis Llamas. (2017). *COMUNICACIÓN DE ADUINO CON PUERTO SERIE*. [online] Disponible en: https://www.luisllamas.es/arduino-puerto-serie/server-die.alc.upv.es

[7] Matpic.com. (2017). *INTERFAZ DE CONTROL DE UN MOTOR PAP EN MATLAB*. [online] Disponible en: http://www.matpic.com/esp/matlab/motor_paso_a_paso.html

[8] Server-die.alc.upv.es (2017). *CONTROL DE MOTORES PASO A PASO*. [online] Disponible en: http://server-die.alc.upv.es/asignaturas/LSED/2002-03/MotoresPasoaPaso/Motorespasoapaso.pdf

[9] Todoelectrodo.blogspot.fr. (2017). *MOTORES PASO A PASO DE IMANES PERMANENTES*. [online] Disponible en: http://todoelectrodo.blogspot.fr/2013/02/motor-paso-paso_17.html

[10] Toyscaos.tripod.com. (2017). *Puerto Serial con Matlab*. [online] Disponible en: http://toyscaos.tripod.com/serial.html