UCIIIM

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA INDUSTRIAL:
ELECTRÓNICA INDUSTRIAL

PROYECTO FIN DE CARRERA

AUTOMATIZACIÓN DE UN SISTEMA DE POSICIONAMIENTO EN DOS DIMENSIONES

NOMBRE: OSCAR LUIS BARRANCO ASENSIO
TUTOR: JORGE PLEITE GUERRA

Febrero 2010

UCIIIM

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA INDUSTRIAL:
ELECTRÓNICA INDUSTRIAL

PROYECTO FIN DE CARRERA

AUTOMATIZACIÓN DE UN SISTEMA DE POSICIONAMIENTO EN DOS DIMENSIONES

NOMBRE: OSCAR LUIS BARRANCO ASENSIO
TUTOR: JORGE PLEITE GUERRA

Febrero 2010

Índice de contenidos

Índice de figuras

1. Introducción

El uso y desarrollo de materiales metálicos en sectores como la energía, ingeniería civil, industria, telecomunicaciones, alimentación, automoción y en aplicaciones biomédicas ha sufrido un desarrollo importante en las últimas décadas. Este desarrollo no solo se debe a la mejora de sus propiedades (mecánicas), sino también a un aumento de su durabilidad en condiciones de servicio. Este aumento de la durabilidad ha sido posible gracias a un mayor conocimiento de los procesos electroquímicos responsables de los fenómenos de corrosión que provocan su deterioro. En base a ese conocimiento se investiga para conseguir una mejora en los métodos de protección.

Uno de los métodos de protección más utilizados que no altera las propiedades específicas de los metales y aleaciones metálicas es el uso de recubrimientos tanto inorgánicos, cerámicos como orgánicos. Los recubrimientos orgánicos son los más utilizados para protección de materiales metálicos en contacto con la atmósfera o en ambientes marinos. En ambos casos el deterioro del recubrimiento y la consiguiente corrosión metálica no sólo afecta a la integridad de las estructuras metálicas con el correspondiente riesgo asociado, sino que también tiene alto impacto medioambiental si los constituyentes de los sistemas de protección son tóxicos. Históricamente, los fenómenos de corrosión en materiales metálicos se han estudiado mediante la utilización de técnicas electroquímicas, espectroscópicas y de análisis de superficie, de las cuales se obtiene información global sobre los procesos que ocurren en la interfase *material metálico / medio corrosivo*. En la actualidad, el desarrollo de las nuevas técnicas electroquímicas localizadas como (SVET (Scanning Vibrating Electrode Tecnique), LEIS (Local Electrochemical Impedance Spectroscopy) , SKP (Scanning Kelvin Probe)) con diferentes resoluciones laterales e información sobre los procesos localizados de corrosión, unido al amplio desarrollo y utilización de técnicas localizadas de microscopía como el microscopio de efecto túnel (STM), microscopio de fuerza atómica (AFM), microscopio electroquímico de barrido (SECM), aplicadas en el campo de la corrosión, hacen posible estudiar los procesos locales de corrosión (in-situ) a escala sub-microscópica. Combinando las técnicas electroquímicas convencionales y las técnicas de análisis de superficie con las técnicas locales se posibilita localizar el estudio de la corrosión y profundizar en los mecanismos y factores controlantes en los distintos procesos de corrosión.

Estos logros técnicos apuntan a una nueva dirección en la investigación de los fenómenos electroquímicos que ocurren en los procesos de corrosión, cuyo objetivo será la elaboración de nuevos modelos basados en el acercamiento a escala submicroscópica. El progreso en la ciencia de la corrosión con especial énfasis en los aspectos microscópicos y nanoscópicos, no hubiese sido posible sin el remarcado progreso en las técnicas de investigación.

2. Planteamiento general.

En este contesto se enmarca el objetivo principal de este proyecto que es el diseño, la construcción y programación de un sistema de posicionamiento en X, e Y , especialmente adaptado tanto a los requerimientos de precisión en el desplazamiento como a los condicionantes de diseño necesarios para poder adaptarlo a un sistema de medida localizado que consiste en una mini célula electroquímica capaz de realizar medidas de Espectroscopia de Impedancia Electroquímica (EIS) en áreas circulares en un diámetro de 0.6 mm en nuevos sistemas en desarrollo *metal / recubrimiento orgánico*.

El equipo de investigación en el que se va a realizar el presente proyecto cuenta con una Mini célula electroquímica de reciente adquisición, así como de varios potenciostatos y analizadores de frecuencia que hacen posible la realización de las medidas de EIS. En la figura 1 se muestra el sistema de medida, que se quiere automatizar y que consta de tres partes fundamentales: una mesa de desplazamiento, un soporte de la mini célula y mini célula electroquímica. [1,2]

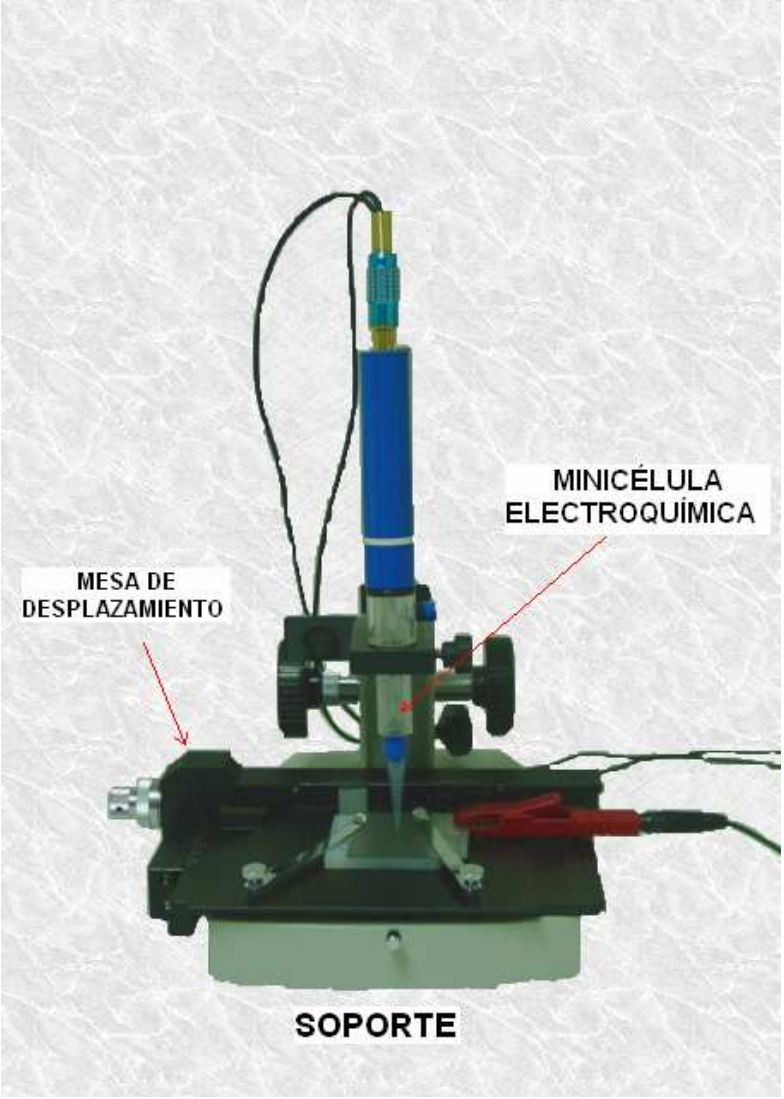


Figura 1: Aspecto del sistema completo: soporte más mesa de desplazamiento más mini célula.

En la figura 2 se puede ver en detalle la probeta con un recubrimiento bajo estudio.

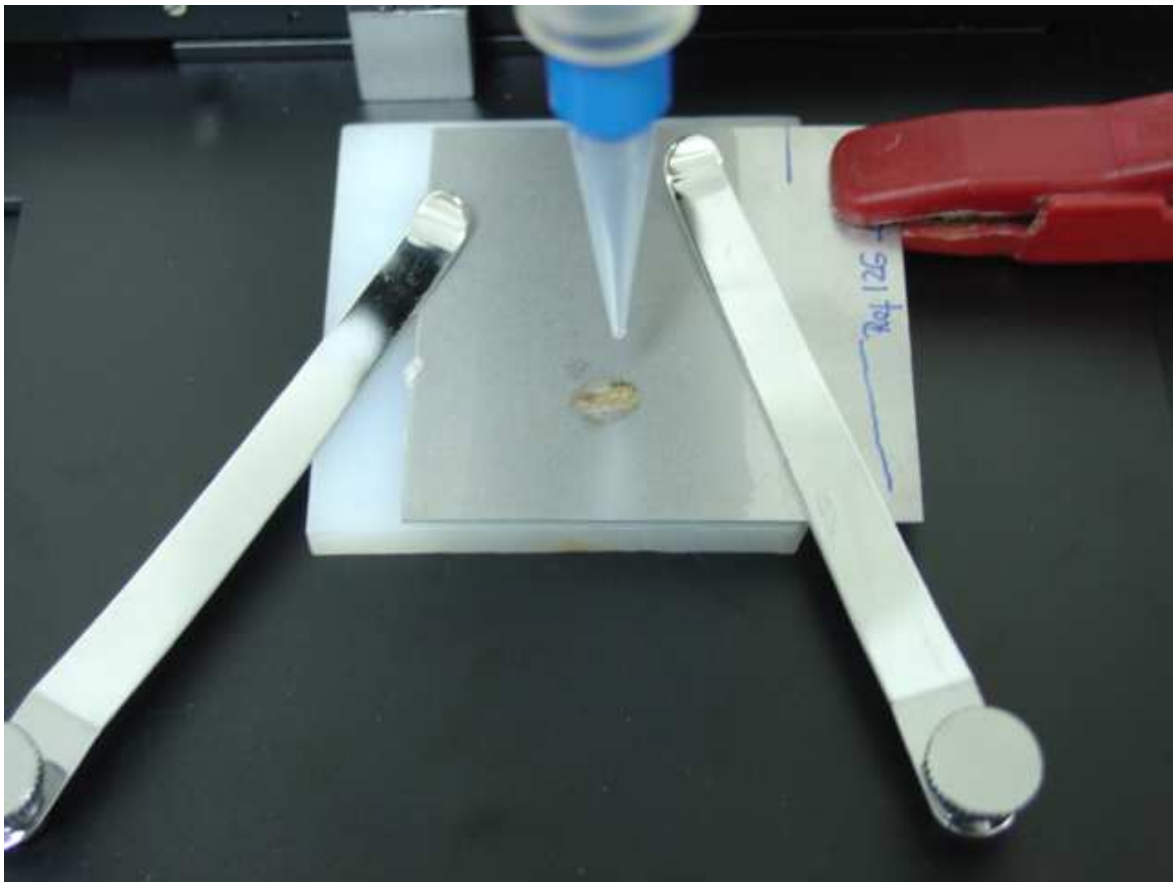


Figura 2: Detalle del área y el recubrimiento bajo estudio de una probeta metálica.

El principal objetivo de este proyecto es el diseño y realización de un sistema de posicionamiento bidimensional de alta precisión y resolución, para ser incorporado a la mini célula electroquímica.

Así mismo se diseñará y programará un software a medida que permita el control informatizado del sistema de posicionamiento.

La importancia de este proyecto y su consecución radica en el hecho de que este sistema de posicionamiento posibilita la obtención de mapas de potencial de corrosión, resistencia de polarización o transferencia de carga y capacidad del recubrimiento polimérico o de la doble capa electroquímica perfectamente asociados a la superficie en estudio. Por esta razón, el estudio de fenómenos de corrosión localizada en sistemas *metal / recubrimiento* sería posible en tiempo real sin tener que esperar al deterioro del recubrimiento, como en el caso de aplicación de las técnicas de resolución micro y nanométricas al estudio de los sistemas *metal / recubrimiento*.

El material necesario para la realización del sistema de posicionamiento robotizado ha sido financiado por el grupo de investigación para el que se realizó el proyecto, perteneciente al Centro Nacional de Investigaciones Metalúrgicas (CENIM) del Consejo Superior de investigaciones Científicas (CSIC).

Los investigadores responsables de la realización y codirección del presente proyecto en el CENIM, son el Dr. Juan Carlos Galván Sierra, Investigador de plantilla del CENIM y la Dra. Violeta Barranco Asensio, Científica contratada (I3P) en el CENIM al inicio del presente proyecto y Científica contratada Ramón y Cajal en el Instituto de Ciencia de Materiales de Madrid (ICMM) desde 2008, ambos centros de investigación pertenecientes al Consejo Superior de investigaciones Científicas (CSIC).

Su responsabilidad en la consecución del proyecto está relacionada con el asesoramiento y guía para el diseño cubriendo las necesidades requeridas para la realización de los ensayos electroquímicos, financiación del material necesario para su consecución y aplicación del sistema de posicionamiento adaptado a la mini célula para estudios reales en proyectos de I + D nacionales y europeos con sistemas metal / recubrimiento de última generación. [3,4]

En cuanto a la dirección y asesoramiento necesario para la consecución a nivel técnico y funcionamiento del sistema de posicionamiento así como de la programación y realización del software, la supervisión ha sido llevada a cabo por el Dr. Jorge Pleite Guerra. Profesor Titular de la Universidad Carlos III.

3. Planteamiento del sistema.

Se trata de crear un sistema *automático* o *semiautomático* de obtención de una medida dependiente de la posición. Para ello, se parte de una situación inicial en la que se dispone de una mesa de desplazamiento (mostrada en la Figura 3) en las dos dimensiones del plano X e Y, y una serie de accesorios, tales como una micro célula electroquímica y un soporte para la misma. (Ver ilustraciones 1 y 2)

Figura 3: Mesa de desplazamiento bidimensional.

Como solución, se propone adaptar mecánicamente dos *motores eléctricos*, uno para el desplazamiento en cada eje, crear un módulo hardware electrónico que accione dichos motores y una interfaz software que permita controlar el funcionamiento de todo el sistema.

Parece razonable descartar, desde un principio, comprar un sistema completo comercial de posicionamiento en dos dimensiones, puesto que se trata de sacar partido a la mesa presentada, que es un bien del que ya se dispone y en el que se ha invertido dinero y está diseñada expresamente para la actividad expuesta. Por otro lado, el coste de un sistema comercial que incluya mesa motorizada, tarjeta de control y aplicación software de gestión del conjunto es mayor si se tiene en cuenta que las calidades y acabados de los materiales empleados son muy superiores que los de la

solución propuesta a continuación, sin que esto afecte sin embargo a la efectividad del sistema en más que la vida útil en el peor de los casos. Y además la solución aquí propuesta está optimizada para dar solución al problema planteado en su totalidad, al contrario que la solución comercial a la que faltaría por realizar ciertos acondicionamientos para poder fijar la mini célula y adaptarla por completo al sistema de medida expuesto.

3.1 Elección de los motores.

Se ha optado por el uso de motores eléctricos de tipo paso a paso, monopolares y con reductora integrada. Para poder entender dicha decisión, a continuación se explica cuales son los motores alternativos a dicho tipo de motor eléctrico y una breve descripción de sus principios de funcionamiento:

Los motores eléctricos están constituidos por dos partes claramente diferenciadas, llamadas rotor y estator. El estator constituye la parte externa del motor, su aspecto suele ser de forma cilíndrica, se fabrica con materiales metálicos y conductores de la electricidad y de ella parten las bornas de conexión por donde recibirá el suministro eléctrico, por lo que ésta suele ser la parte inmóvil del motor y solidaria al sistema en el que se integra.

El rotor es la parte móvil del motor y está constituida por materiales metálicos y conductores de la electricidad y se alojan en el interior del estator dispuestos a lo largo del eje imaginario del cilindro estatórico, en un eje real de metal que descansa normalmente sobre dos cojinetes, situados en los extremos del estator con la finalidad de permitir el giro del rotor con el menor rozamiento mecánico posible.

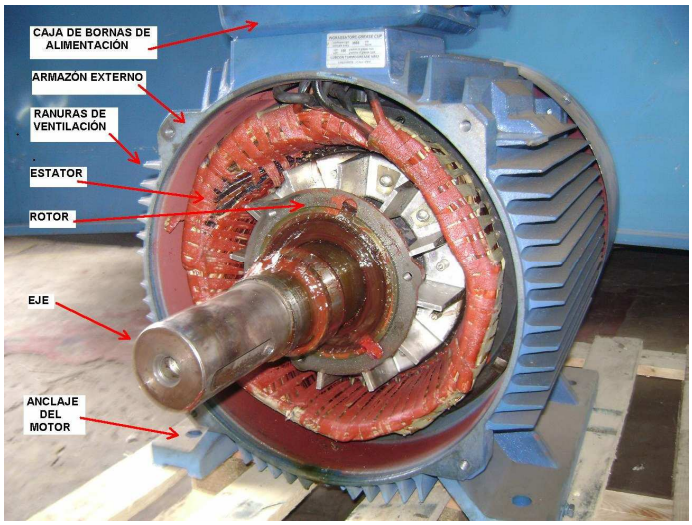


Figura 4: Partes de un motor eléctrico.

Motor eléctrico de corriente continúa tipo “imán permanente”:

En la figura 5 se muestra de forma esquemática su constitución. El estator, señalado como “armadura” en la figura 5, constituye un imán permanente dentro de la acción del cual se encuentra inmerso el rotor.

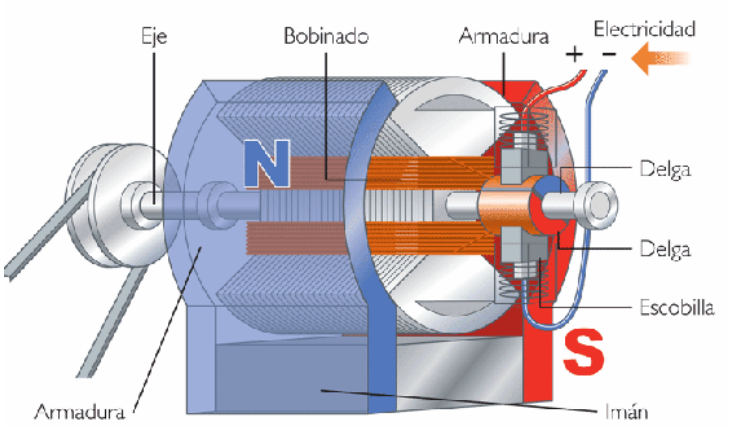


Figura 5 : Estator de un motor de imán permanente.

El rotor tiene una serie de devanados (*bobinado*) de conductor arrollados en un material ferromagnético, de modo que inyectando una corriente eléctrica continua por ellos se crea un campo magnético que interacciona con el creado por el estator como si de dos imanes se tratase provocando un giro del eje rotor hasta que los polos magnéticos de signo contrario de ambos “imanes” se vean enfrentados quedando finalmente en una posición de equilibrio o reposo.

El motor cuenta con un mecanismo, señalado en la figura 5 con el nombre de “escobillas y delgas”, que permite que cambien los polos magnéticos creados en el rotor a medida que este gira ya que invierte el sentido de la corriente que fluye por el rotor, gracias a lo cual se hace posible un giro continuo del eje motor ya que, mientras está presente la corriente eléctrica por los conductores del rotor, siempre están enfrentados polos de distinto signo provocando un par de giro constante.

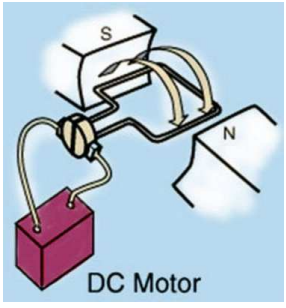


Figura 6: Esquema simplificado del principio de funcionamiento de las escobillas de un motor de c.c.

Dado que este proyecto precisa de gran precisión en la determinación de la posición del sistema, el inconveniente de este tipo de motor es que, una vez dejamos de aplicar la corriente por los conductores del rotor ya no tenemos el escenario análogo a dos imanes enfrentados sino a un eje que gira libre al dejar de actuar sobre él fuerza motriz alguna y quedando sólo las fuerzas ocasionadas por el rozamiento mecánico y la inercia, por lo cual éste continua moviéndose durante un cierto tiempo, que depende de la dinámica del sistema, llegando a la situación de reposo en una posición angular, por tanto, difícil de determinar.

Motor eléctrico de corriente continúa tipo “estator devanado”:

Su funcionamiento es idéntico al motor de CC tipo “imán permanente”. A diferencia, el campo magnético del estator se crea de la misma forma que el del rotor, a través de la inyección de una corriente continua por los devanados del estator.

Desde el punto de vista de las necesidades de este proyecto, dicho motor tampoco introduce, por tanto, ninguna mejora en la capacidad de determinación de la posición del eje respecto del motor de CC de imán permanente.

Motor eléctrico de corriente alterna.

Dicho motor genera los campos magnéticos de estator y de rotor mediante las corrientes eléctricas que se hace circular por sus devanados estatóricos. Los devanados se alimentan con corriente alterna, de tal forma que los polos magnéticos varían su posición espacial en el tiempo girando en el interior del estator y provocando el arrastre del eje del motor.

Al igual que en los motores de CC, en ausencia de corriente eléctrica, hay ausencia de fuerzas que provoquen el giro del eje por lo éste queda a merced de la inercia y rozamiento mecánico, habiendo gran incertidumbre en la determinación de la posición final del eje.

Motor eléctrico de pasos:

Este tipo de motor tiene arroyamientos de conductor en el estator, y en el rotor, puede tener un imán permanente o arroyamientos, de modo que, el principio de funcionamiento es igual que el de los motores ya citados, pero con la peculiaridad de tener acceso a distintos devanados del estator, como si de varios electroimanes independientes y contiguos se tratase. En la figura se puede ver a lo que nos referimos.

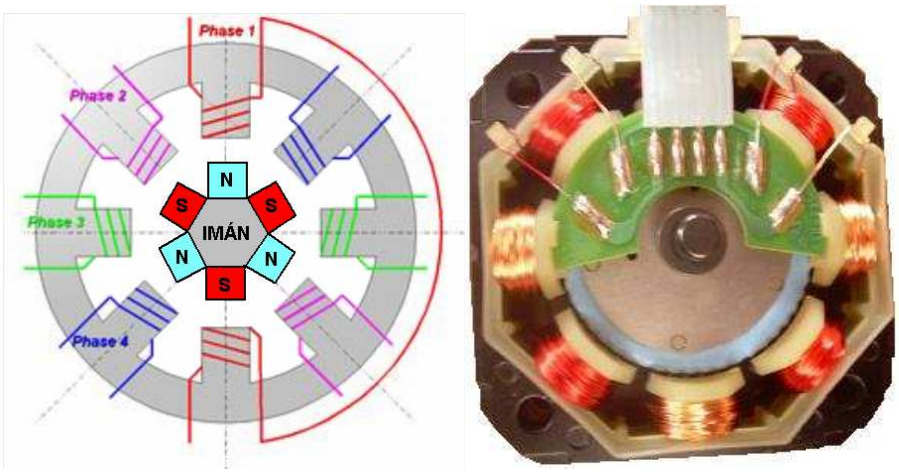


Figura 7: Esquema simplificado de las partes de un motor eléctrico tipo "paso a paso" y foto de un motor real.

De este modo, al aplicar una corriente continua a uno de los devanados el imán del rotor (bien conformado por un imán permanente o por devanados que habrá que alimentar) gira un determinado ángulo hasta enfrentar sus polos con los del campo creado. Este tipo de motores tiene un número elevado de devanados a lo largo de los 360 grados del cilindro de estator espaciados entre sí un ángulo concreto y sus conexiones están ligadas en serie por grupos de devanados de modo que con unos pocos cables de acceso se puede inyectar corriente de forma secuencial para provocar que funcione cada pequeño electroimán de forma que el campo magnético del estator gire y el eje del rotor siga dicho movimiento en forma de pequeños pasos.

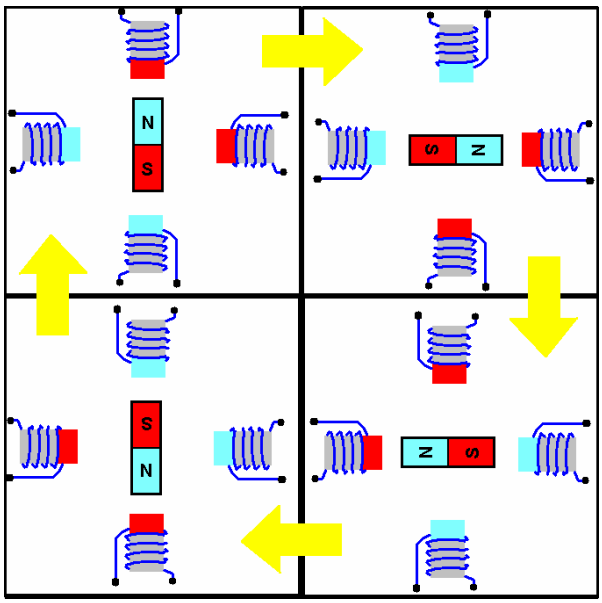


Figura 8: Esquema de funcionamiento de un motor eléctrico tipo "paso a paso"

Lo que hace atractivo a este tipo de motores para utilizarlo en este proyecto es que queda perfectamente determinado el ángulo de giro del eje entre cada paso. Las limitaciones de exactitud quedan impuestas por las tolerancias del fabricante.

Dado que se requiere una gran exactitud se ha elegido un motor con cuarenta y ocho pasos por cada 360 grados, además el motor está provisto de una reductora de engranajes metálicos cuya relación de reducción es de 125 a 1, lo que nos da como resultado un total de 48*125 = 6000 pasos por cada 360 grados de giro del eje rotor, lo que significa tener una exactitud de 0,06 grados por paso.

Normalmente, en el caso de utilización de motores de c.c. o c.a. se utiliza un sistema de realimentación para poder determinar con precisión magnitudes como la posición, la velocidad o la aceleración, ello implica la utilización de un sistema electrónico más o menos complejo y el uso de detectores de posición que encarecen el sistema. Tal y como se puede ver en la Figura 9, se ha de añadir un mecanismo (señalado como “encoder”) que permita medir la posición de giro del eje en cada instante de tiempo. Después hay que acondicionar las señales, a través del bloque *K* para que lleguen a la unidad de control (CPU) de forma adecuada para que ésta pueda realizar los cálculos oportunos y actuar en consecuencia sobre las señales de excitación del motor.

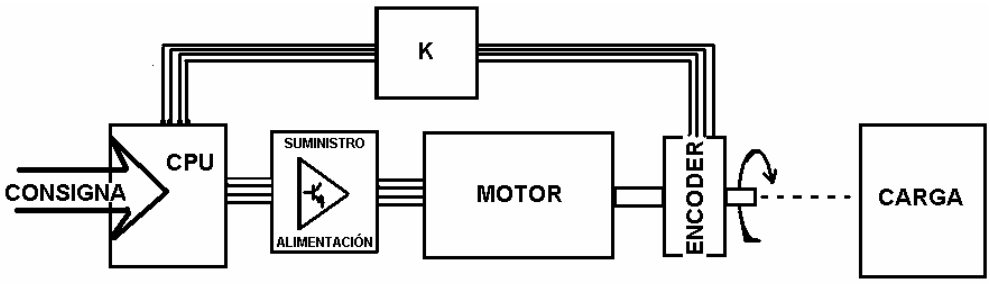


Figura 9: Esquema de un sistema realimentado.

Sin embargo, hacer uso de motores a pasos como el elegido para esta aplicación, permite que podamos prescindir de un sistema de realimentación de la posición. La única premisa a respetar es que se debe tener la certeza de que no se pierden pasos. Esto quiere decir, que el hecho de enviar un impulso eléctrico para dar un paso siempre se debe traducir en la generación de dicho movimiento, para lo cual las señales de control deben ser precisas y no debe haber pérdidas de energía en el camino.

3.2 Control del sistema.

En el plano de control del sistema, se ha optado por crear un módulo *hardware* con los mínimos componentes posibles, de modo que se pueda asegurar mayor robustez y sencillez en el funcionamiento. Para ello se ha diseñado un circuito electrónico con unos pocos transistores, redes resistivas, diodos, reguladores de alimentación, etc. que hacen posible la transferencia de la energía eléctrica a los motores de la forma que se precisa para su correcto funcionamiento.

Uno de los problemas planteados durante el diseño del sistema de control, ha sido determinar si las señales de control de los motores y la recepción de las señales de los sensores debía ser tarea a llevar a cabo por un ordenador personal u otro tipo de computadora, o por el contrario se debería dejar esa responsabilidad a un circuito integrado microcomputador, alojado en la propia placa de circuito impreso donde se alojan los demás componentes electrónicos.

Para ello se ha realizado un prototipo de circuito electrónico en el laboratorio, cuya funcionalidad es acondicionar las señales salientes del puerto paralelo de un ordenador personal (comúnmente utilizado como puerto para conectar la impresora) para que puedan provocar el movimiento de los motores.

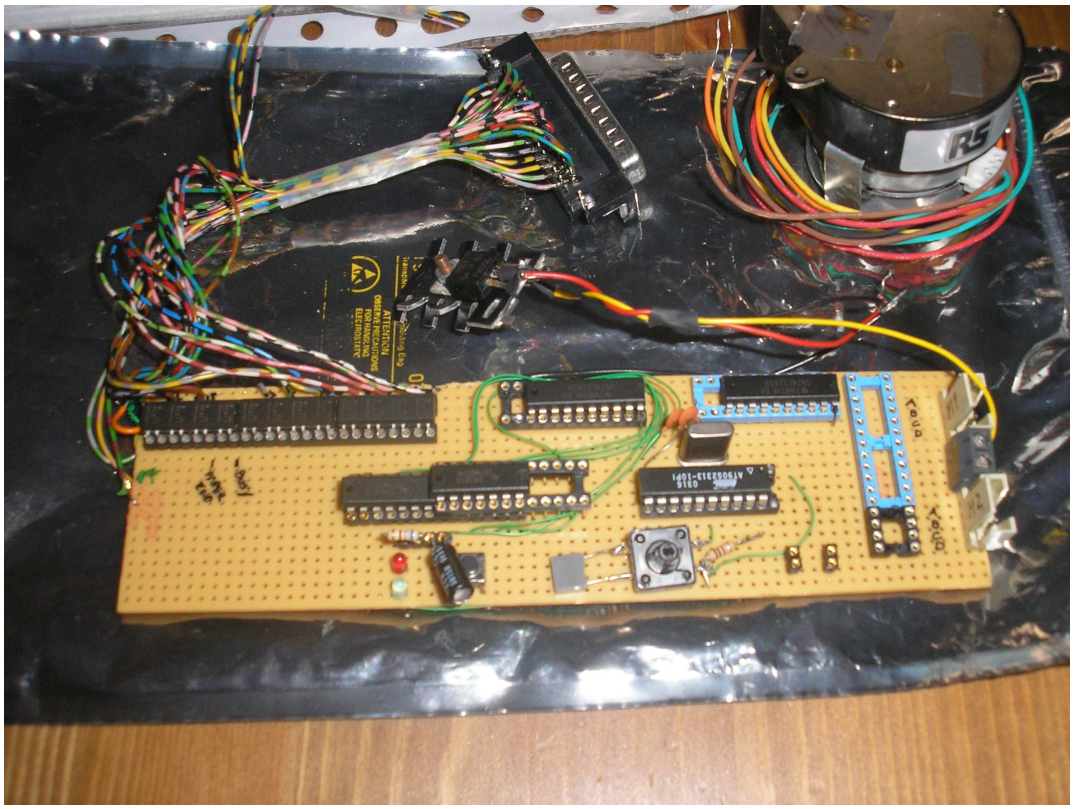


Figura 10: Prototipo de laboratorio.

Con dicho prototipo, además se ha realizado una aplicación software que funciona sobre el sistema operativo Windows de Microsoft, por ser el más extendido.

La aplicación software se encarga de sintetizar las señales de control de los motores basándose en una serie de temporizadores que se pueden programar. Su código fuente se puede ver en el anexo III

La experiencia se ha llevado a cabo en el laboratorio de la Universidad Carlos III conectando un solo motor y de ella se ha sacado la conclusión de que no es una opción válida confeccionar señales en tiempo real trabajando con Microsoft Windows porque no es un sistema operativo que permita trabajar de este modo con las herramientas con que se ha diseñado el software.

En el osciloscopio del laboratorio, saliendo del cable del PC, se observó una secuencia de señales cuadradas que coinciden con los momentos y el orden en que se deben activar los pequeños electroimanes del motor para producir un giro en un sentido determinado, pero se pudo apreciar que el tiempo registrado entre los impulsos no era constante. Dado que el ancho de los impulsos y latencia son directamente proporcionales a la fuerza y el tiempo que debe tardar el motor en dar cada paso, y sabiendo que, dadas las limitaciones físicas del motor, el ancho de los impulsos debe tener una duración mínima para garantizar que la energía eléctrica se convierte en cinética realmente y no se pierde, estamos ante una situación de gran incertidumbre en el movimiento exacto del eje del motor.

Durante la experiencia se pudo percibir como el eje del motor daba pequeños “tropiezos bruscos” en el giro del eje.

Dicho fenómeno se produce por las imprecisiones temporales al sintetizar la señal. Esto se puede entender como consecuencia de que Windows es un sistema operativo multiproceso, lo que significa realiza muchas tareas muy deprisa casi a la vez, pero no es un sistema operativo en tiempo real, por lo que no lo hace de forma totalmente simultanea de modo que atiende cada tarea o proceso de forma individual y va dedicando pequeños intervalos de tiempo de forma alternativa a cada uno según la prioridad que tenga establecida, por ello, mientras al sistema operativo le surge la necesidad de atender una tarea, puede dejar de lado la cuenta del temporizador que determina la duración de un impulso.[1]

Una solución a esto sería utilizar un sistema operativo que trabaje en tiempo real como puede ser Unix, RT Linux, Spectra, Soldaris o QNX pero ha sido rechazada puesto que no se encuentra en los ordenadores del CENIM en los que debe trabajar nuestro sistema.

Por tanto, la opción elegida ha sido utilizar un microcontrolador, que es un microcomputador integrado en un chip de silicio, el cual nos permite programarlo para que sintetice las señales de control de los motores y atienda a las señales recibidas de los interruptores como tarea exclusiva y además sincronizada con un circuito oscilador del propio chip de precisión suficiente para evitar el problema que se acaba de exponer.

3.3 Interfaz de usuario.

En tercer lugar, debemos hablar de la interfaz de control del sistema. Se ha optado por seguir teniendo en cuenta el PC, a pesar de que la labor más importante la realiza el chip microcontrolador, como herramienta de interfaz y no en unos sencillos pulsadores e indicadores luminosos insertados en alguna parte del sistema, en pro de una apariencia más amigable y atractiva para el usuario final. Por otra parte, también la interacción entre el sistema y el PC brinda un mayor abanico de posibilidades de cara a la programación de los movimientos que se quiera llevar a cabo con los motores y el procesado de los datos resultantes de las medidas realizadas por la mini-célula.

El PC se conecta a la tarjeta electrónica de control a través del puerto paralelo debido a su gran facilidad de acceso al manejo de las señales del mismo desde el punto de vista de la programación, en comparación con otros puertos de entrada – salida de la máquina. También esto posibilita gobernar las señales del puerto sin necesidad de seguir ningún protocolo estándar de comunicaciones en el uso del puerto paralelo de comunicaciones de un ordenador personal convencional. De este modo, se desarrollará una aplicación software que permitirá tener un fácil y total control de las señales en dicho terminal, sin grandes exigencias desde el punto de vista de la programación, para lograr enviarlas al puerto de entrada salida.

La aplicación software ha sido desarrollada bajo la plataforma .NET en Visual Basic.NET por su sencillez para proporcionar una interfaz visual amigable para el usuario final.

3.4 Mecánica del sistema.

En cuanto al acoplamiento mecánico de todo el sistema, se ha pensado en aprovechar los taladros que ya posee la mesa de desplazamiento para anclar una estructura metálica que actúe de soporte de los motores y de la electrónica de control. Las transmisiones del movimiento motor se efectúan a través de ruedas dentadas, con las que se puede tener acotadas, mediante la medida experimental, las holguras entre la parte motriz y la carga mecánica.

4. Desarrollo hardware.

4.1 Mecánica.

4.1.1 Mesa de desplazamiento.

Se parte de una mesa de desplazamiento en las dos dimensiones del plano, a las que nos referiremos por sus ejes imaginarios X e Y. La mesa ha sido aportada por el CNIM como parte de los elementos del sistema de los que ya disponía. Se trata de una mesa de desplazamiento fabricada en Alemania por encargo expreso del CENIM, en un material metálico robusto. Posee una par de ejes de engrane de gran exactitud mecánica.

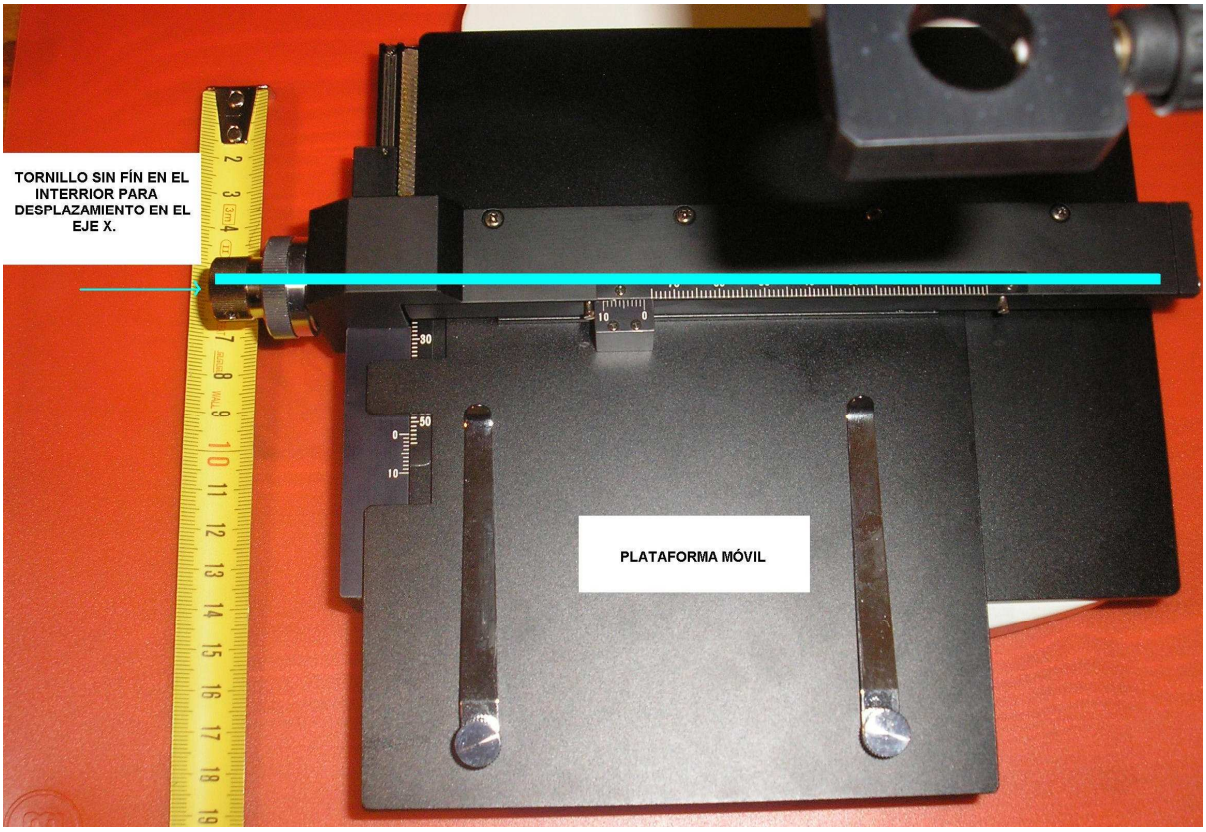


Figura 11: Ejes de la mesa de desplazamiento.

Uno de ellos, al que nos referiremos como eje X, (figura 11) está basado en un tornillo sin fin que arrastra una pieza roscada al mismo, solidaria a la plataforma móvil de la mesa. La mecánica del eje Y se basa en un engrane denominado “de cremallera” mostrado con más detalle en la figura 12.

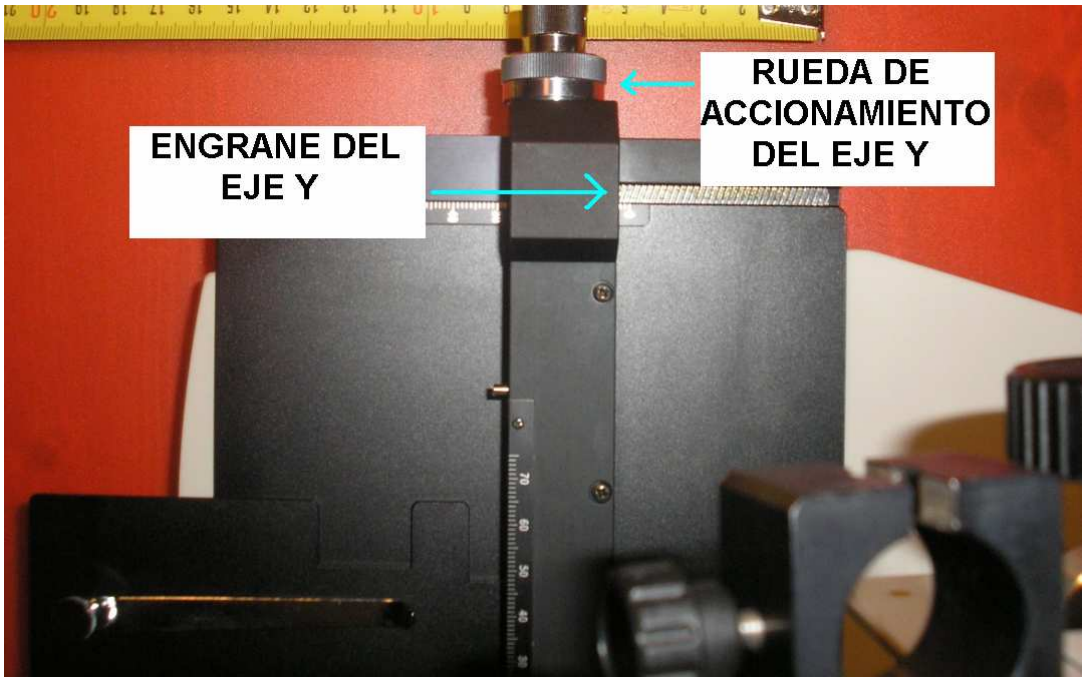


Figura 12: Detalle del eje Y de la mesa de desplazamiento.

Sobre la pieza longitudinal dentada descansa una rueda dentada que se encarga de transmitir el movimiento. Como accionamientos del movimiento, la mesa dispone de dos ruedas concéntricas conectadas mecánicamente a cada uno de los sistemas de desplazamiento que se acaban de describir, y permiten efectuar desplazamientos de la plataforma móvil de la mesa de forma manual. A lo largo de los ejes X e Y, en la parte inmóvil de la mesa, se dispone de marcas graduadas, a modo de regla, que junto con otras dos pequeñas piezas situadas en la parte móvil, que poseen también marcas graduadas, permiten conocer la distancia que se desplaza con una exactitud de 0,0001 metros ± 0,00005 metros sin tener en cuenta el posible error de paralaje cometido al medir a ojo (ver figura 14).

En los perfiles de las plataformas de la mesa, como se puede apreciar en la figura 14 se dispone de una serie de agujeros con rosca métrica, característica que se ha aprovechado para diseñar el acoplamiento mecánico de los motores y el resto de componentes.

4.1.2 Implementación mecánica.

Tal y como se explica en el apartado 3, se ha elegido el uso de motores de tipo paso a paso con imán permanente. En el plano mecánico, sus características, proporcionadas por el fabricante, son las mostradas en la figura 13.

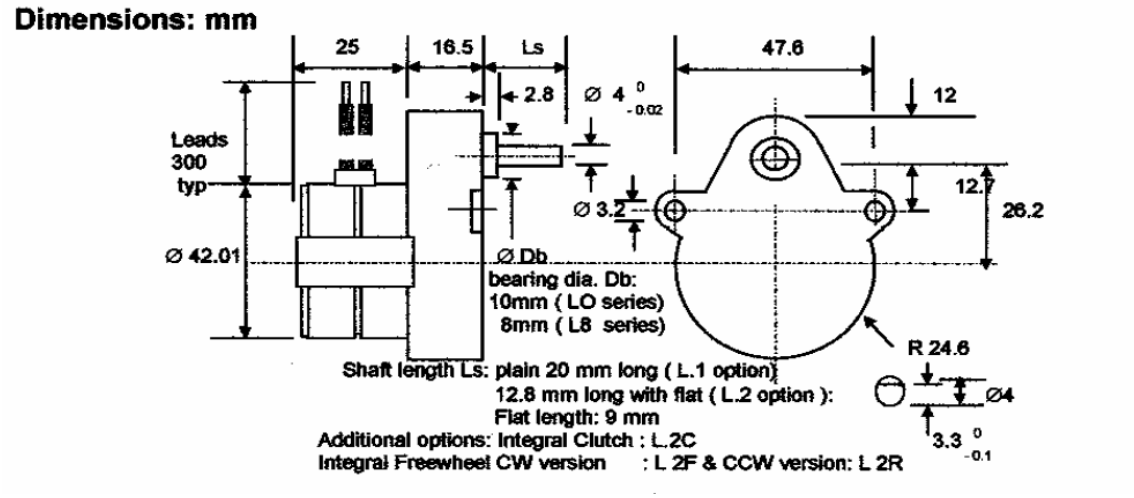


Figura 13: Información mecánica de los motores eléctricos elegidos.

Para realizar el acoplamiento mecánico se ha estudiado la disposición de los agujeros roscados mencionados anteriormente y los tornillos que posee la mesa.

Figura 14: Aspecto de los agujeros disponibles en la mesa de desplazamiento.

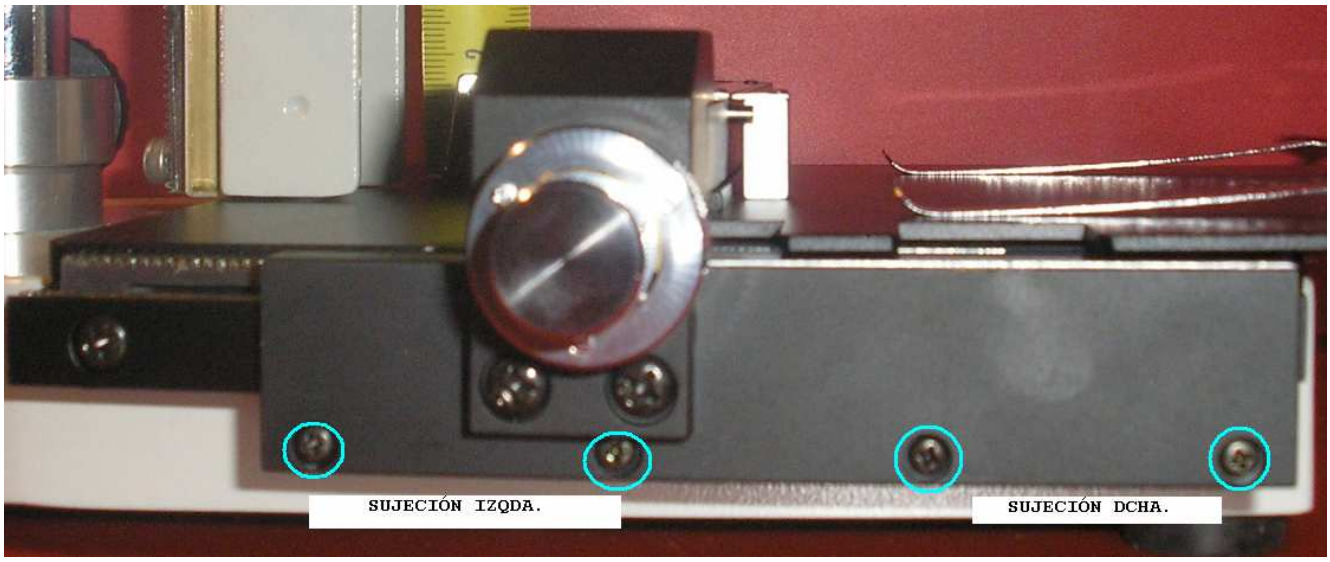


Figura 15: Aspecto del perfil donde se van a alojar los motores.

A partir de dicho punto surge la necesidad de pensar en el diseño de algunos elementos mecánicos que faciliten la implementación del anclaje. Como primera opción se ha pensado en la adquisición de piezas metálicas comerciales, pero no resulta fácil encontrar las piezas que se adapten adecuadamente a las medidas requeridas sin tener que reprocesarlas mecánicamente, por lo que se opta por el diseño y creación a medida de dichas piezas. Por este motivo, se han buscado materiales y proveedores adecuados eligiendo finalmente la utilización de planchas de aluminio de espesores entre 1 y 3 milímetros que resultan muy dúctiles y maleables, y por tanto, adecuadas realizar cortes, y agujeros sobre ellas a fin de obtener las piezas diseñadas.

Para llevar a cabo el acoplamiento de los motores, se opta por el mayor acercamiento posible a las ruedas de accionamiento de la mesa con el fin de simplificar al máximo la complejidad de la transmisión del movimiento. De dicha premisa resulta el diseño de la pieza mostrada a continuación en la figura 16.

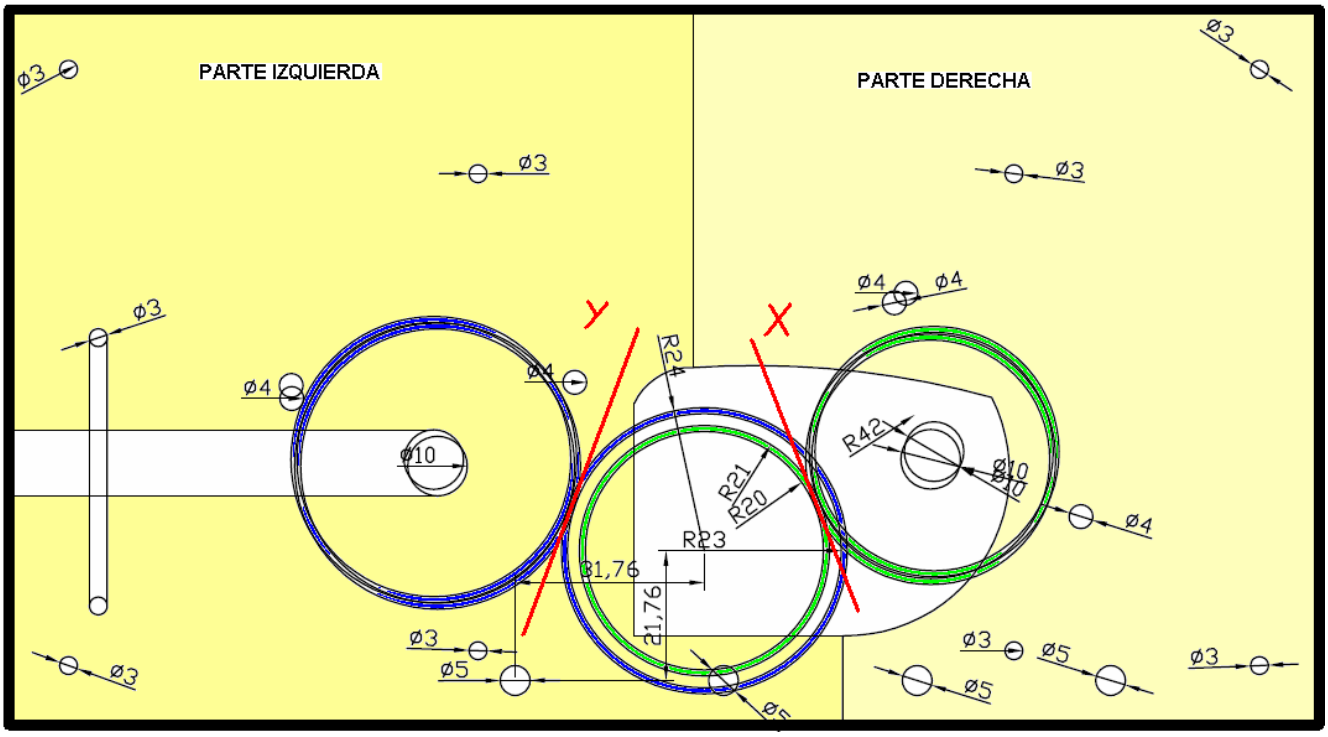


Figura 16: Sujeción principal de los motores. ***Pieza A****.*

Esta pieza constituye la sujeción principal de los motores a la mesa de desplazamiento; en ella se ha realizado un corte rectilíneo con el propósito de dividirla en dos mitades. La mitad izquierda de la figura, se atornilla directamente al perfil de la mesa en la que se encuentran las ruedas concéntricas de accionamiento de la mesa, *("sujeción izqda."* en figura 15 y ver plano general en figuras 27 y 28) de modo que la rueda dentada de ataque para el movimiento del eje Y, quedará en el mismo plano que la rueda de arrastre del mismo (resaltadas en color azul en al figura 16) obteniendo la línea de engrane marcada en rojo con la letra "Y". La mitad derecha se atornilla en el mismo perfil de la mesa *("sujeción dcha."* en figura 15), pero dejando una separación entre la mesa y la pieza de 1 cm para conseguir el alineamiento necesario de los planos en los que se encuentran los engranajes motrices con los engranajes centrales de arrastre del eje X (resaltados en color verde en la figura 16). Las circunferencias representan la posición de las ruedas dentadas respecto a la pieza diseñada.

La pieza mostrada en la figura 17 sirve de “tirante” de la plataforma de sujeción de los motores. Con ella se consigue dar rigidez al conjunto. (Ver plano general en figuras 27 y 28).

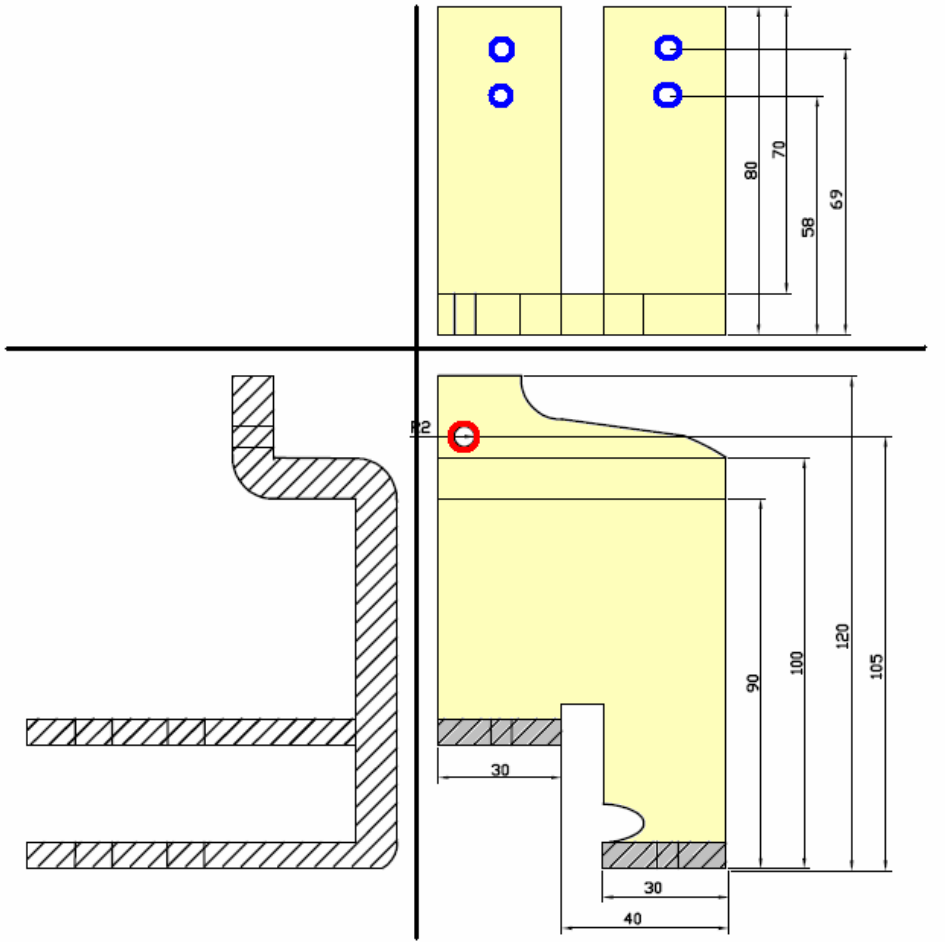


Figura 17: Tirante de fijación de la plataforma de los motores. ***Pieza B****.*

En planta se observa (resaltado en rojo) el agujero que sirve como pasante para atornillar la pieza a la mesa.

En el alzado se aprecian cuatro agujeros (resaltados en color azul) que sirven de fijación para cada una de las dos mitades de la plataforma de los motores vista con anterioridad (figura 16).

El plegado de la pieza es necesario para conseguir que se adapte a la mesa adecuadamente tal y como se muestra en la figura 18.

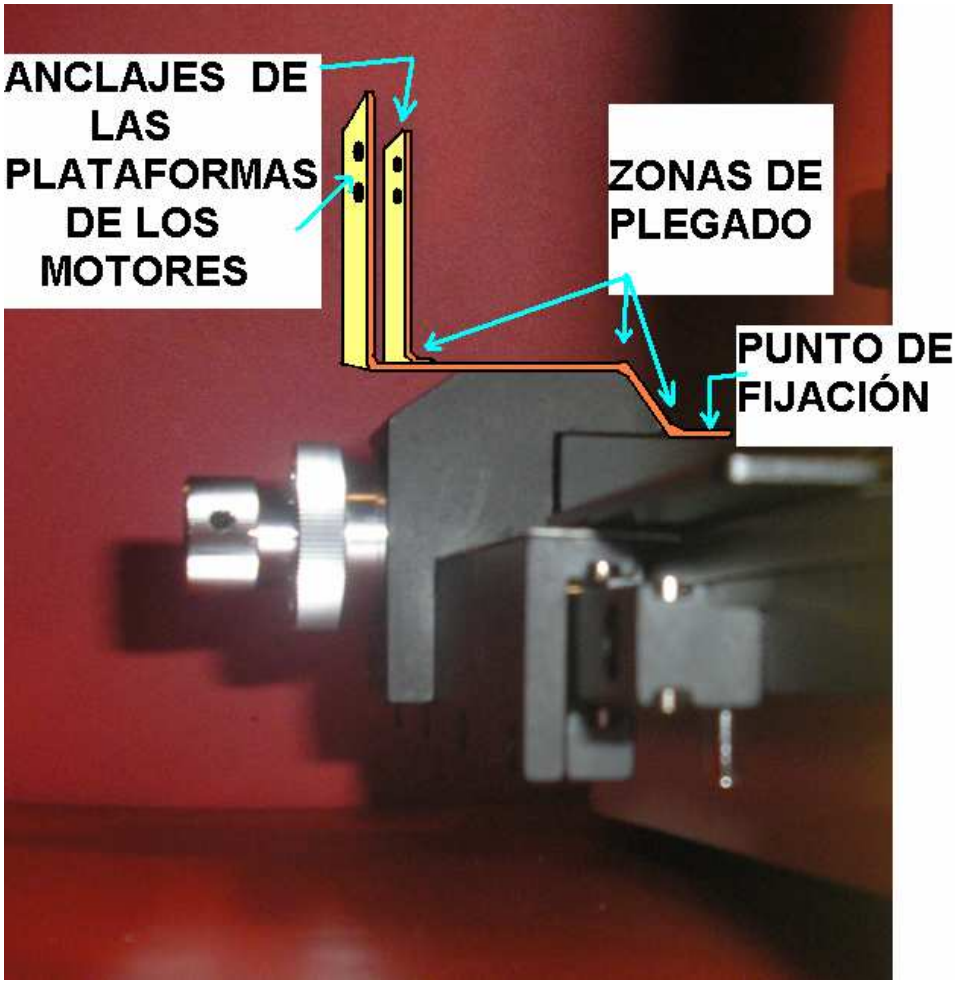


Figura 18: Detalle de la posición de la pieza B

Con las piezas anteriores queda perfectamente determinado el sistema de fijación de los motores y la tarjeta de control electrónica, la cual se atornilla directamente a cuatro postes que posicionan la tarjeta de circuito impreso en un plano paralelo al de los engranajes, lo cual se podrá ver más adelante con detalle.

De la necesidad de situar en diferentes puntos de la mesa sensores de posición, surge el diseño de las siguientes piezas.

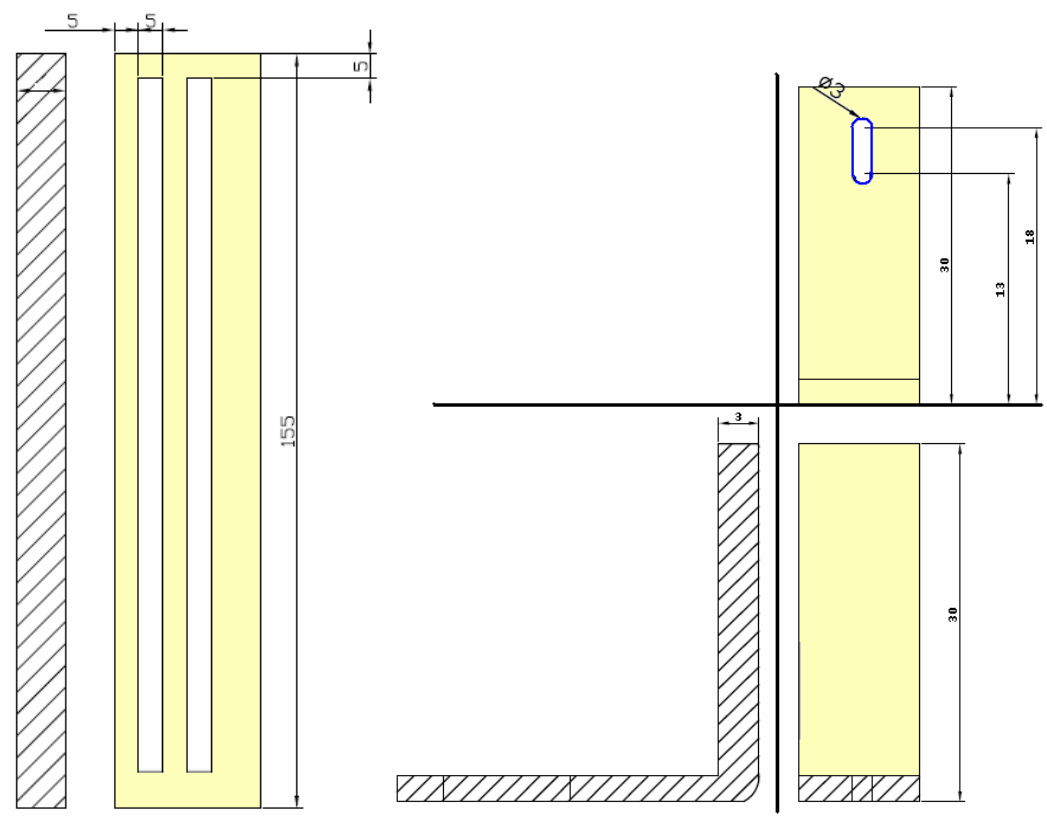


Figura 19: Piezas de sujeción de los interruptores de final de recorrido del eje X. ***Piezas C y D****.*

La pieza de la izquierda, pieza C, sirve como alojamiento de los interruptores de final de recorrido del eje X. (Ver plano general en figuras 27 y 28).Se trata de una pletina con dos carriles perforados que permiten sujetar con dos tornillos el interruptor en el punto deseado del recorrido. La pieza se dispone horizontalmente a lo largo del eje X tal y como se indica en la Figura 20 siendo sujeta a la mesa mediante dos escuadras como la mostrada en la Figura 19 a la derecha (pieza D). Las escuadras poseen un agujero rasgado (resaltado en color azul) para permitir un ajuste manual de la posición.

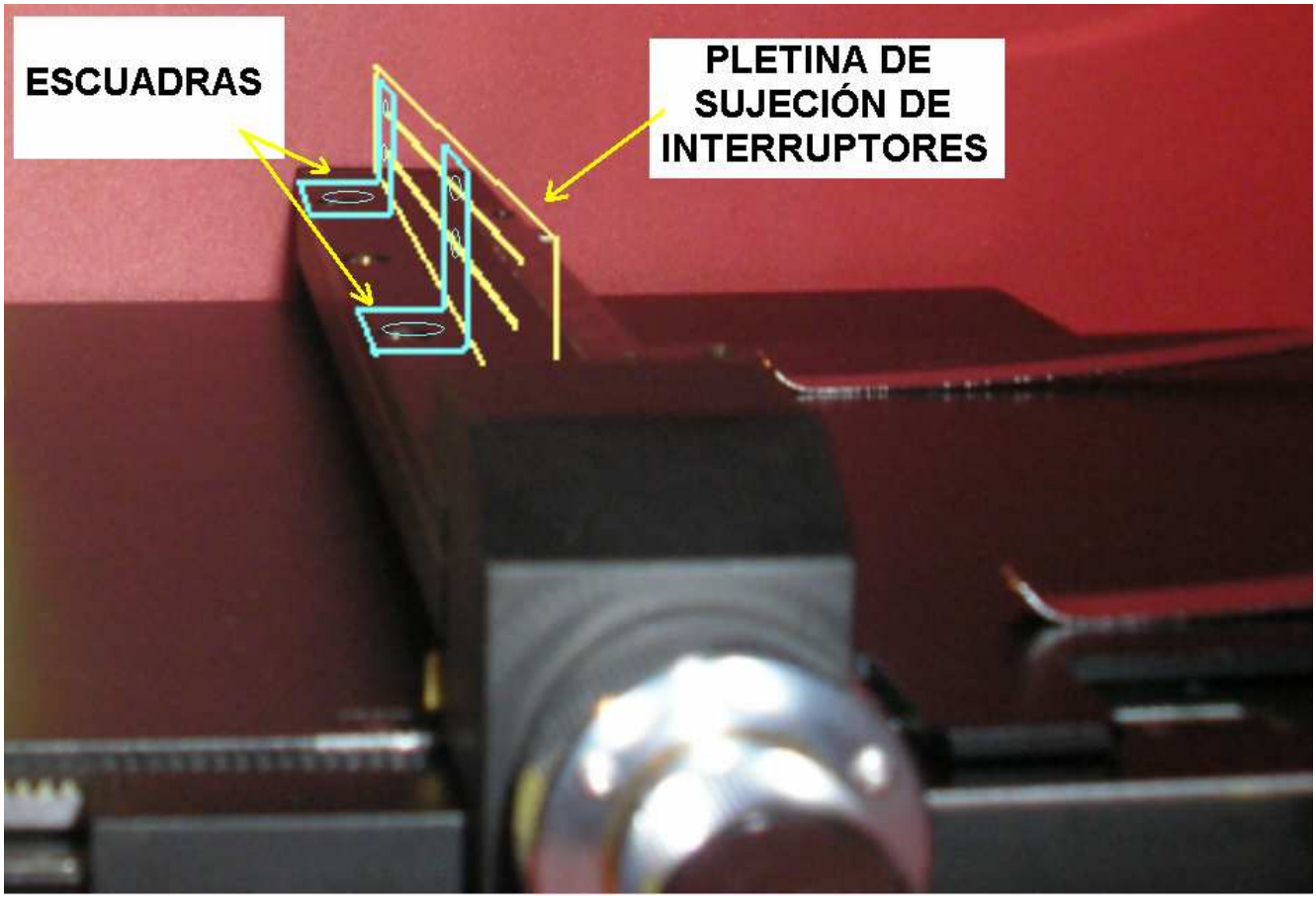


Figura 20: Detalle de situación de la pletina de sujeción del interruptor de final de recorrido del eje X.

De forma análoga a las piezas del eje X, se realizan las mostradas en la Figura 21 para el eje Y. (Ver plano general en figuras 27 y 28).La pletina alargada, a diferencia de la anterior tiene los carriles perforados hasta, aproximadamente, la mitad de su longitud y posee cuatro taladros para permitir la sujeción a ellos de dos escuadras como las mostradas en la Figura 21 a la derecha (pieza F).

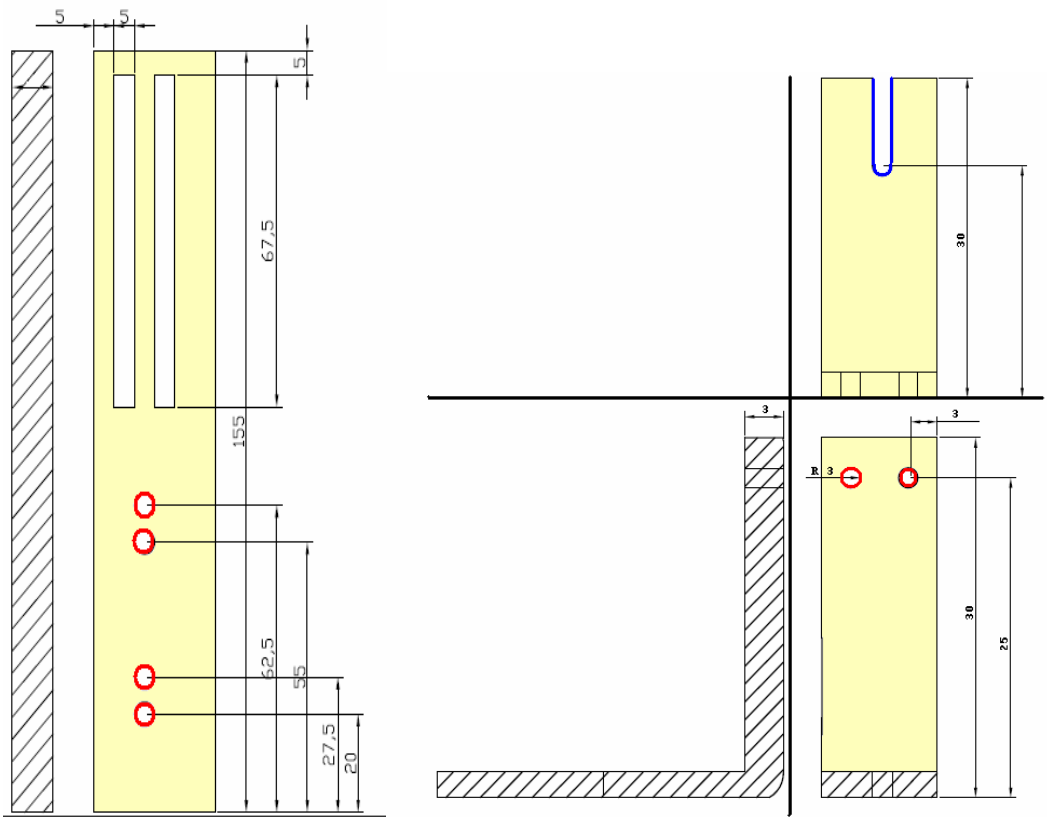


Figura 21: Piezas de sujeción de los interruptores de final de recorrido del eje Y. ***Piezas E y F.***

La ranura resaltada en la Figura 21 en color azul permite fijar las escuadras F de manera que queden a ras de la superficie bajo la plataforma móvil, de modo que ésta no tropiece cuando deba sobresalir, tal y como se muestra en la Figura 22.

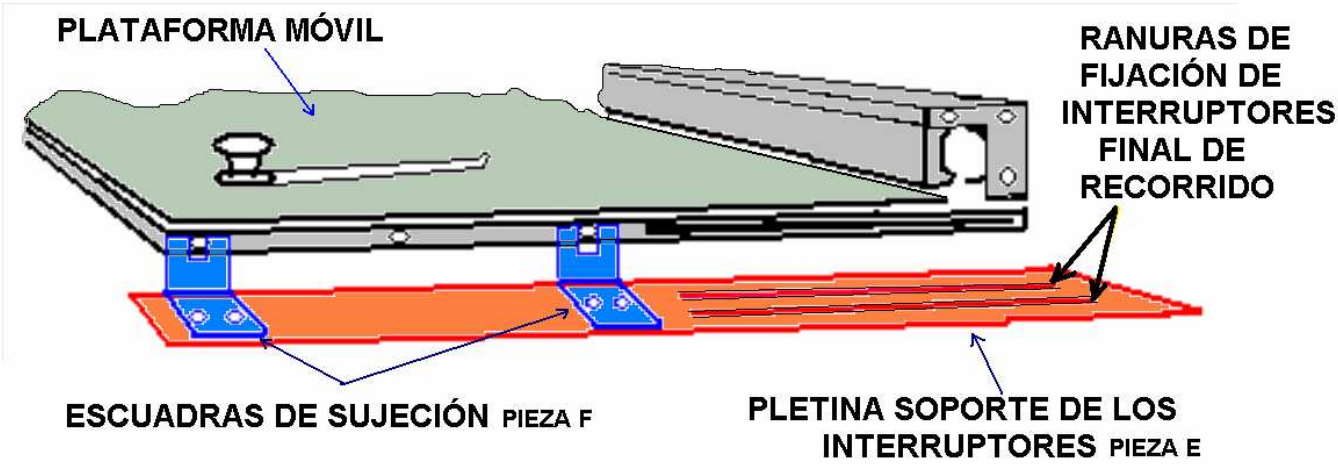


Figura 22: Detalle de montaje del conjunto pletina-escuadras para alojar el interruptor de final de recorrido del eje Y.

.

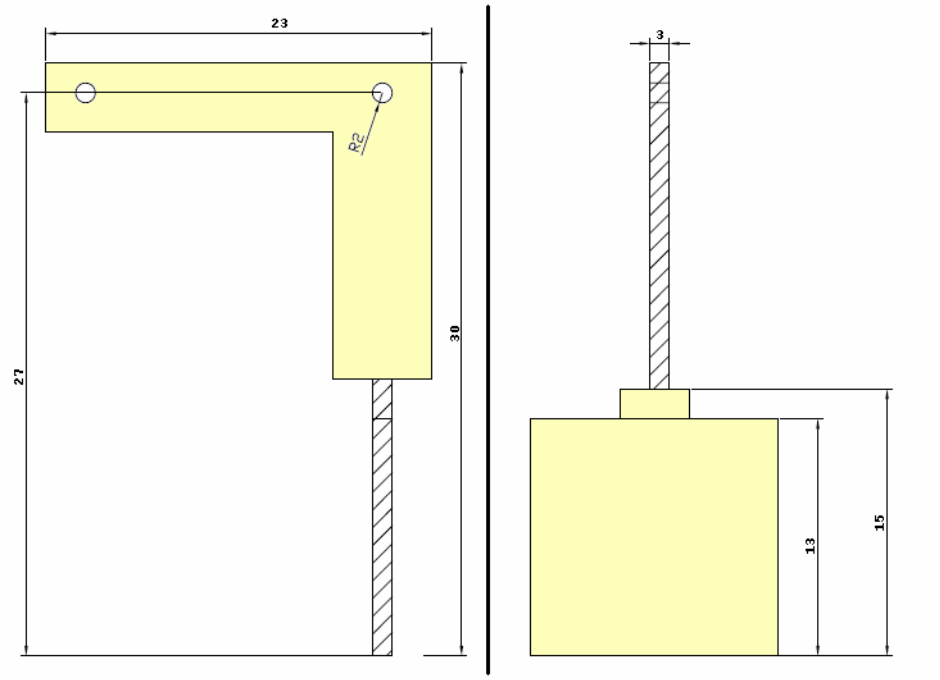


Figura 23: Pieza de accionamiento de los interruptores de final de recorrido del eje Y. ***Pieza G.***

En consonancia con las dos piezas anteriores es necesario confeccionar esta otra (Figura 23) la cual tiene una torsión de 90 grados en la parte central para conseguir la forma que se muestra en la figura 24. (Ver plano general en figuras 27 y 28).

En alzado (a la derecha) podemos ver con color la superficie que presionará directamente los interruptores, mientras que en el perfil de la pieza (parte izquierda) se observan los agujeros realizados para los tornillos de fijación.

Esta pieza se atornilla a la parte móvil de la mesa tal y como se muestra en la figura 24 para servir de accionamiento de los interruptores del eje Y.

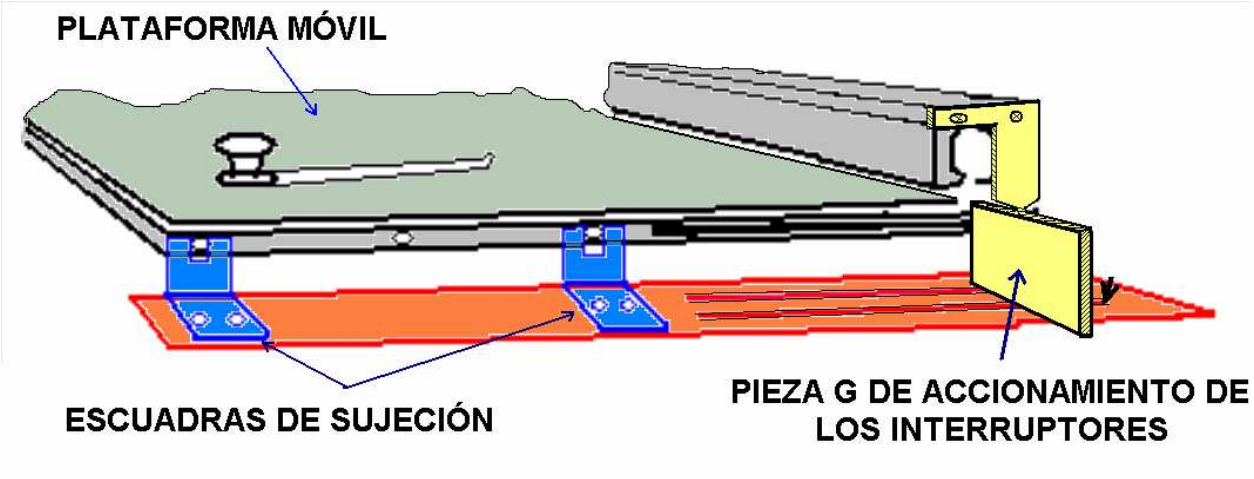


Figura 24: Detalle de situación de la pieza G.

Otra de las piezas diseñadas es la mostrada a continuación. Se muestra en planta, y constituye el accionamiento de los interruptores de final de recorrido del eje X. Dicha pieza presionará los interruptores con los brazos que sobresalen. (Ver plano general en figuras 27 y 28).

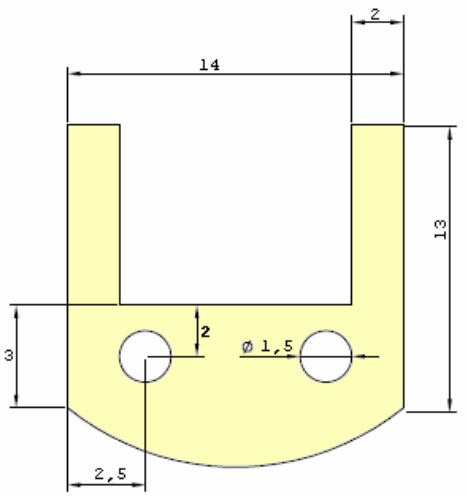


Figura 25: Pieza de accionamiento de los interruptores de final de recorrido del eje X. ***Pieza H.***

Como se puede ver en la figura 26 existe una pequeña pieza situada en la plataforma móvil de la mesa, con unas marcas graduadas que permiten realizar la medida de desplazamiento relativo al eje X. Dicha pieza se ha aprovechado para fijar a ella, mediante los propios tornillos que sujetan la graduación, la pieza diseñada de la figura 25.

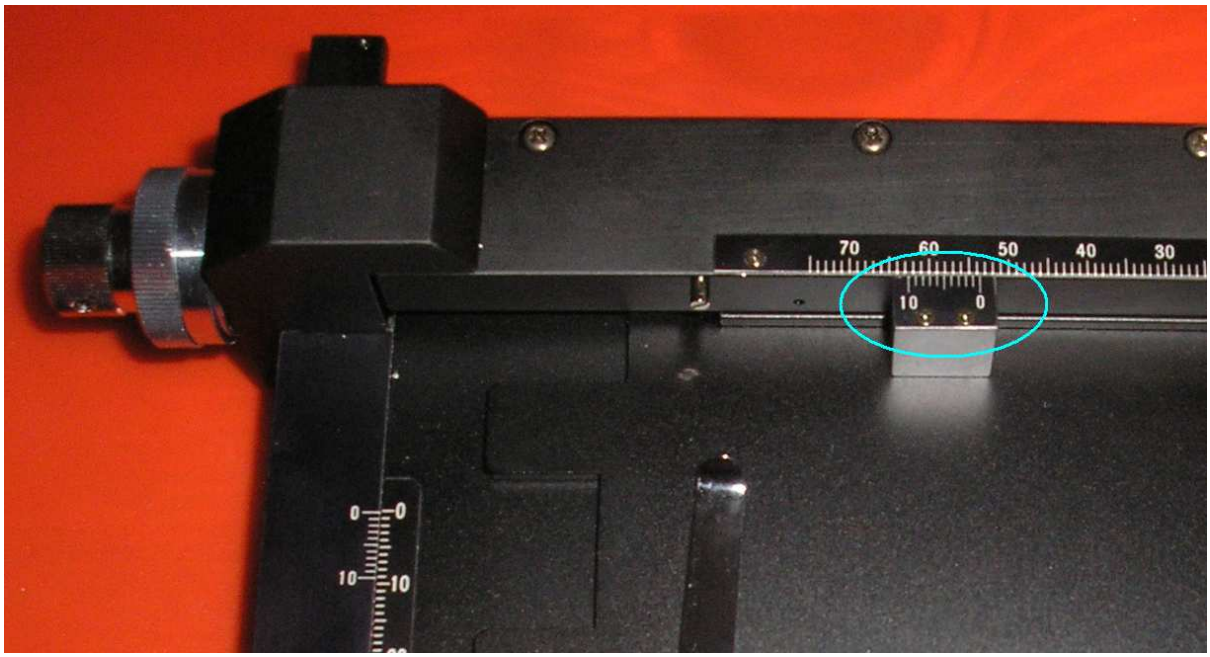


Figura 26: Pieza móvil graduada de medida del desplazamiento en el eje X.

4.2.3 Plano general del acoplamiento mecánico del conjunto.

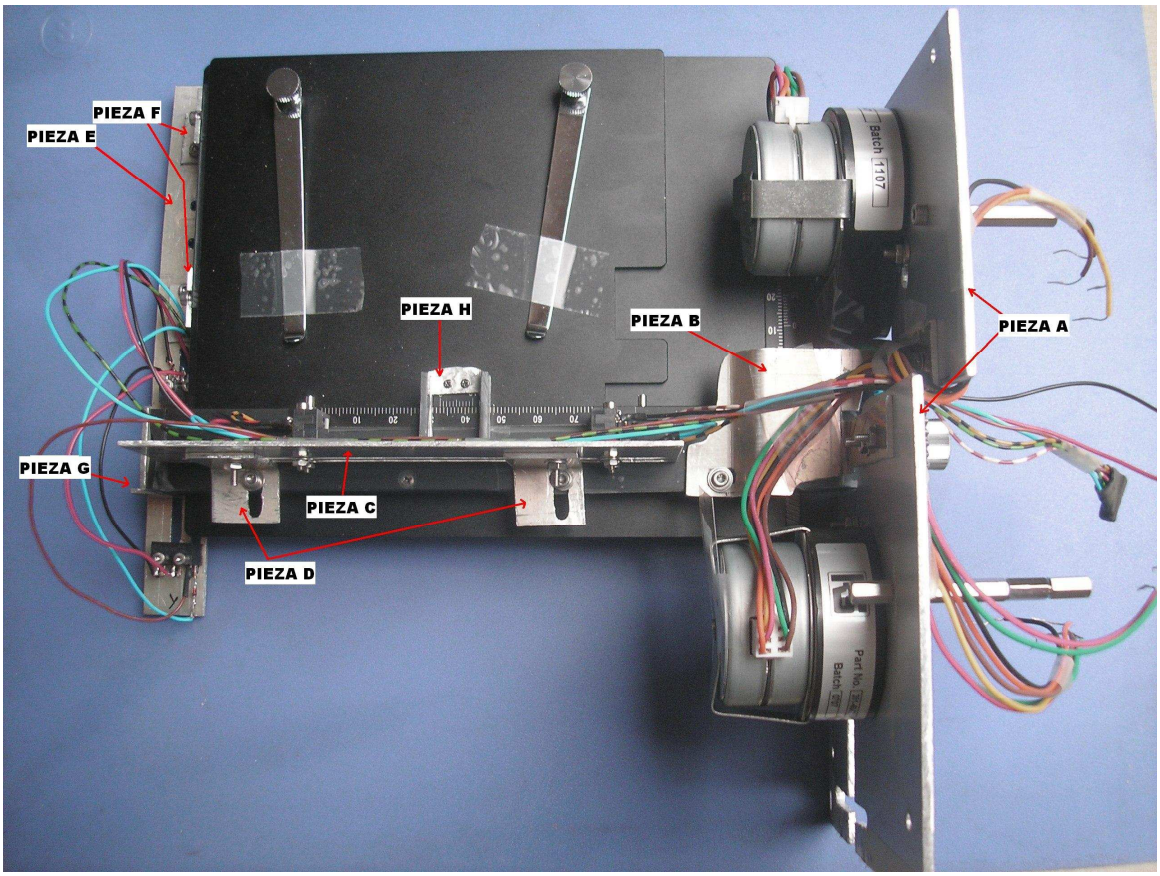


Figura 27: Vista del plano general del acoplamiento mecánico.

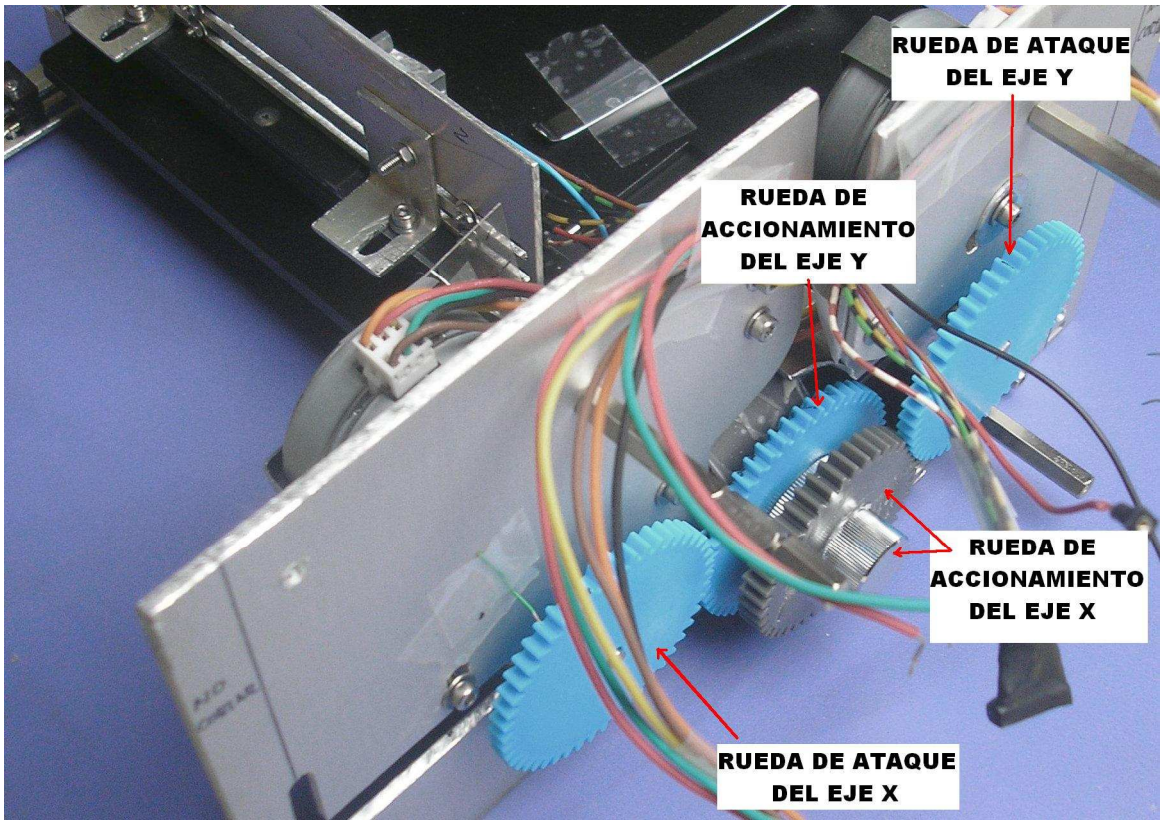


Figura 28: Vista en detalle del conjunto de transmisión mecánico.

4.2 Electrónica.

El circuito electrónico se compone de unos pocos bloques funcionales que básicamente son: (Ver esquema completo en anexo I)

4.2.1 Bloque de alimentación del sistema.

El principal objetivo del boque de alimentación es adaptar la energía eléctrica que proviene de la red eléctrica comercial a las necesidades de la tarjeta de control diseñada. Para ello, lo primero que debe hacerse es transformar los 220 V de a.c. en 5 V de c.c. que es la tensión a la que deben trabajar los motores según especificaciones del fabricante para que se tenga las prestaciones indicadas; para ello se ha elegido el uso de un adaptador de red comercial UNIROSS; la elección de dicho adaptador se ha hecho basándose en un borrador de diseño del sistema sobre el cuál se han calculado los consumos de los motores y el resto de circuitos, siendo 1.2 A corriente suficiente para asegurar el buen funcionamiento del sistema; dicho adaptador permite seleccionar la tensión de salida entre los siguientes valores: 1.5v ,3v, 4.5v, 6v, 7.5v, 9v, 10.5v y 12 v y es capaz de entregar una corriente en condiciones nominales de 1,2 A.

Figura 29: Imagen del adaptador de red utilizado.

Para garantizar que las condiciones en las que se realiza el suministro de energía al sistema de control y a los motores sean óptimas, se añade al diseño dos reguladores de tensión. Los reguladores de tensión reciben la alimentación del adaptador de red y proporcionan una tensión de salida de 5v. Es necesario el empleo de reguladores de tensión para reducir a 5v exactos la alimentación de los motores y de los circuitos de computación y control, y porque también ayudan a filtrar variaciones que suelen aparecer a la salida de cualquier adaptador comercial, debido a las perturbaciones en la línea de distribución comercial o las interferencias que se puedan producir por el entorno de trabajo. Por tanto, se utilizará la salida de 7,5v del adaptador de red para que el regulador después la baje a 5v y funcione adecuadamente. Otro motivo por el que son recomendables los reguladores, es porque ante picos de la demanda puntuales de corriente en el sistema, garantizan que no se producirá una caída notable de la tensión de alimentación, característica que está detallada en la hoja de características del fabricante. Por otro lado, se ha decidido utilizar dos reguladores para mejorar las prestaciones separando las vías de alimentación de la electrónica de bajo consumo (circuitos de computación y control) de los elementos de alto consumo (transistores MOSFET “drivers” y motores) porque, además los motores son gobernados por secuencias de señales cuadradas que demandan cambios bruscos de corriente y su alto contenido en armónicos podría afectar a la electrónica de bajo consumo.

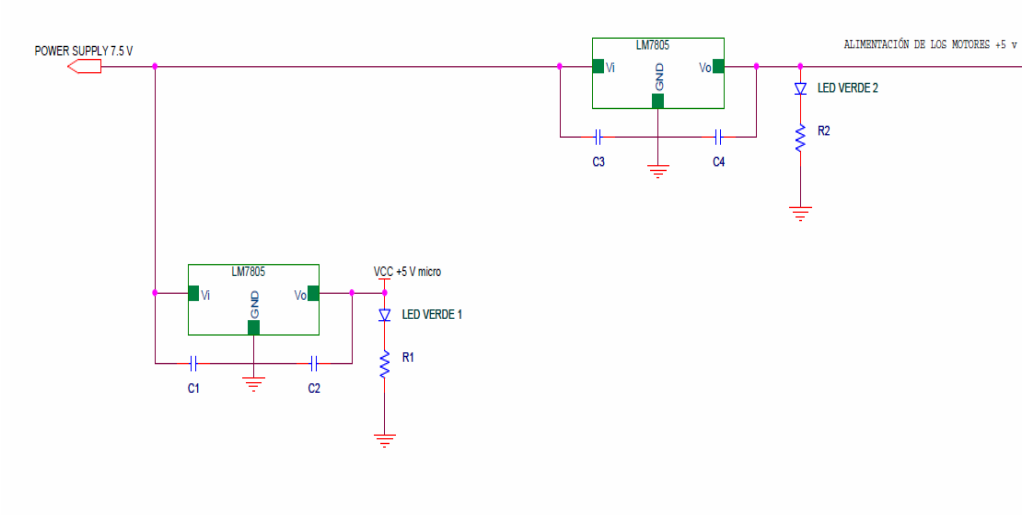


Figura 30: Detalle eléctrico de los reguladores de tensión.

A partir de los reguladores, el circuito eléctrico se divide en dos ramas, cada una de ellas competencia de un regulador; una es la rama eléctrica de los actuadores, de mayor amplitud de corriente y la otra, de bajo consumo, que constituye la parte computacional del sistema.

4.2.2 Unidad de procesamiento lógico.

Constituyendo lo que se puede considerar el corazón del sistema, se cuenta con un microcontrolador de arquitectura AVR RISC de 8 bits, de la casa ATMEL, modelo ATmega8535L. Este circuito integrado constituye un pequeño computador que está dotado de elementos de memoria RAM y ROM, donde se puede almacenar un programa en lenguaje máquina para que lo ejecute paso a paso. También dispone de cuatro puertos de entrada y salida de 8 bits, por los que se pueden enviar o recibir las señales eléctricas según se necesite en cada momento. Se puede ver en detalle características técnicas en el anexo I.

Todo diseño de hardware basado en microcontrolador lleva asociada la necesidad de un entorno de desarrollo que se compone normalmente de un entorno software que permite desarrollar el programa que se ejecutará en el microcontrolador, y por otro lado, de un dispositivo electrónico que permita grabar el programa en lenguaje máquina dentro de la memoria del microcontrolador. La facilidad con la que se puede encontrar versiones freeware de dicho entorno de desarrollo junto con la característica que posee este tipo de microcontrolador que hace que se pueda programar dentro del sistema final en el que va a trabajar sin más dificultad que la de hacer un cable determinado, ha sido en gran medida lo que ha hecho que se pensara en utilizarlo en nuestro proyecto. Además, por sus características técnicas, favorece al ahorro de algún componente externo, que para otros microcontroladores suele ser necesario añadir.

El microcontrolador desempeña las siguientes funciones:

- → Síntesis de las señales de control de los motores.
- → Comunicación con el PC.
- → Toma de decisiones ante las señales recibidas del entorno.

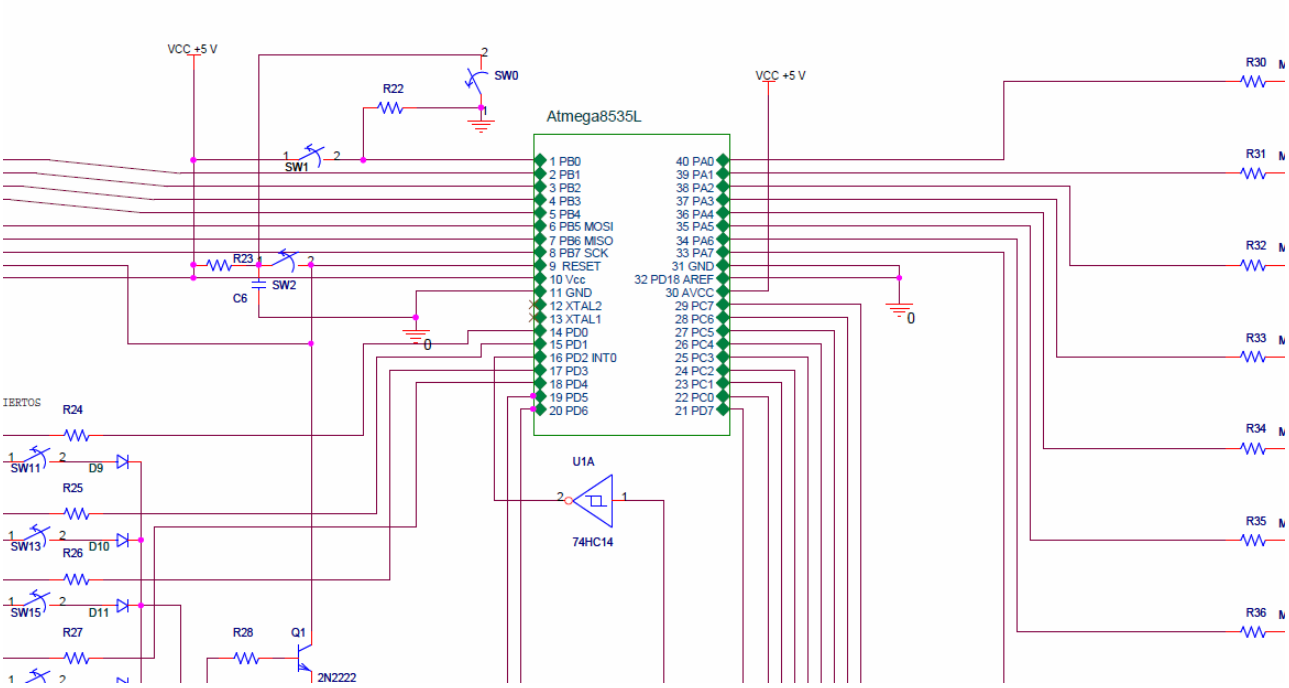


Figura 31: Detalle eléctrico del microcontrolador.

4.2.3 Sistema de protección.

El sistema de protección hace alusión a los componentes electro-mecánicos que se encargan de garantizar la actuación del sistema para protegerse ante eventualidades tales como el movimiento descontrolado y continuo de algún motor pudiendo derivar en sobreesfuerzos mecánicos y sobrecorrientes que pudieran dañar de forma permanente el sistema.

Los elementos participantes en dicho sistema son las puertas lógicas de tipo NOT (componente U1A, U1B y U1C del esquema eléctrico), resistencias (R24, R25, R26, R27, R38, R39, R40 y R41), interruptores (SW5 y de SW10 a SW17), diodos LED (led rojo), diodos D9 al D12 y transistor MOSFET “M0” que se pueden ver en la figura 32.

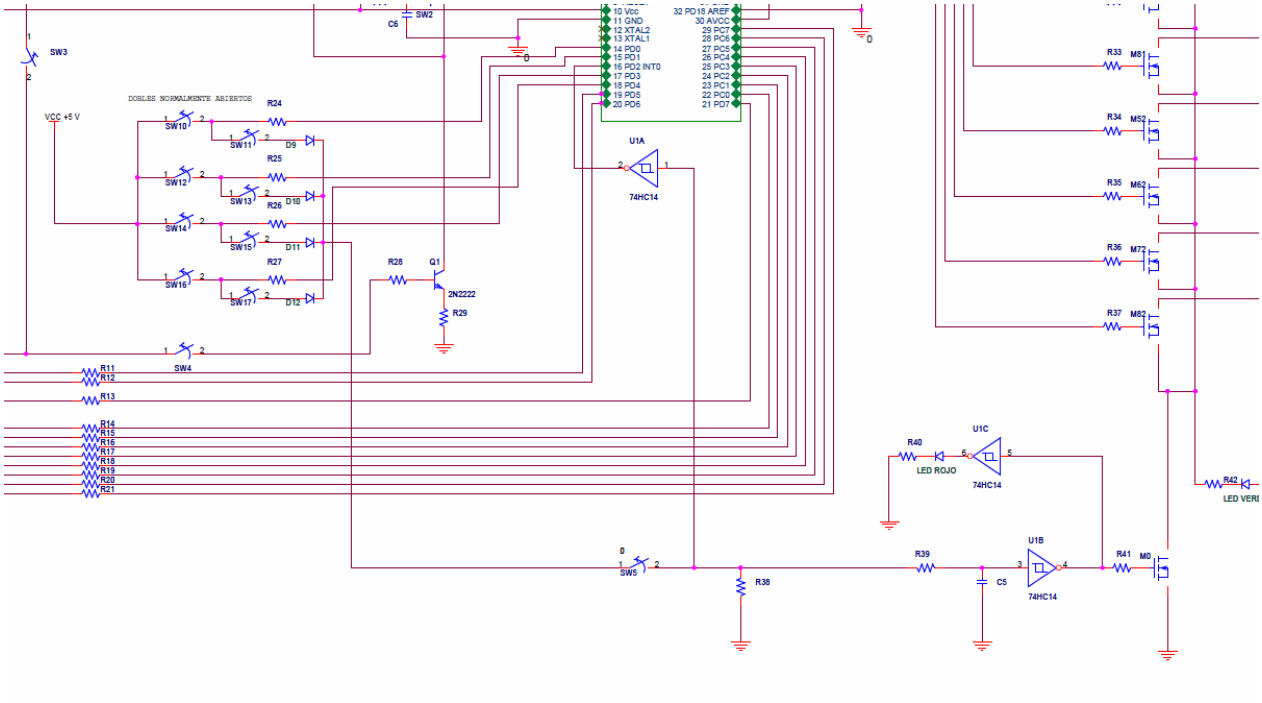


Figura 32: Detalle eléctrico del sistema de seguridad.

4.2.4 Modos de funcionamiento de la tarjeta de control.

El diseño eléctrico provee al sistema de dos modos de funcionamiento:

Modo operativo.

Modo de detección de códigos.

A continuación se muestra de forma gráfica la esencia de cada modo de funcionamiento desde el punto de vista del circuito eléctrico.

4.2.4.1 Modo operativo.

En este modo de funcionamiento, la tarjeta de control está a la espera, de forma continua, de las peticiones por parte del usuario a través del PC. Primero, la información del usuario se envía al microcontrolador, después el microcontrolador contesta al PC con la información que ha interpretado, esta es comparada por el PC con la que ha enviado y de este modo se valida la información llegada al microcontrolador. Una vez validada, el PC envía un código de comunicación exitosa o fallida, y acto seguido, si procede, el microcontrolador genera el patrón de forma de onda que excita los transistores de potencia de los motores para provocar el desplazamiento deseado por el usuario. En las ilustraciones 31 y 32 se muestra dicho funcionamiento de forma gráfica.

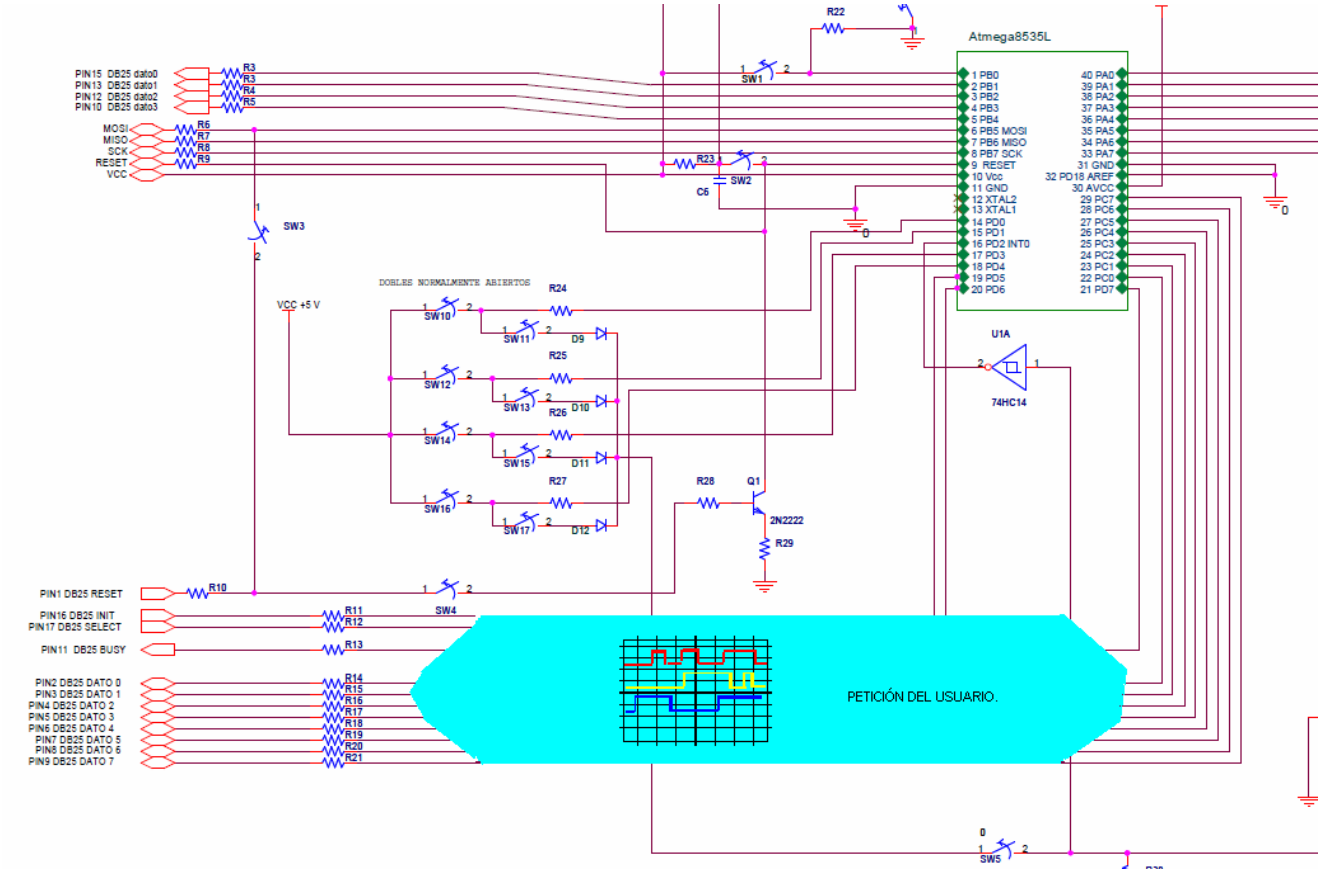


Figura 33: Modo operativo: propagación de las señales desde el PC hacia el microcontrolador.

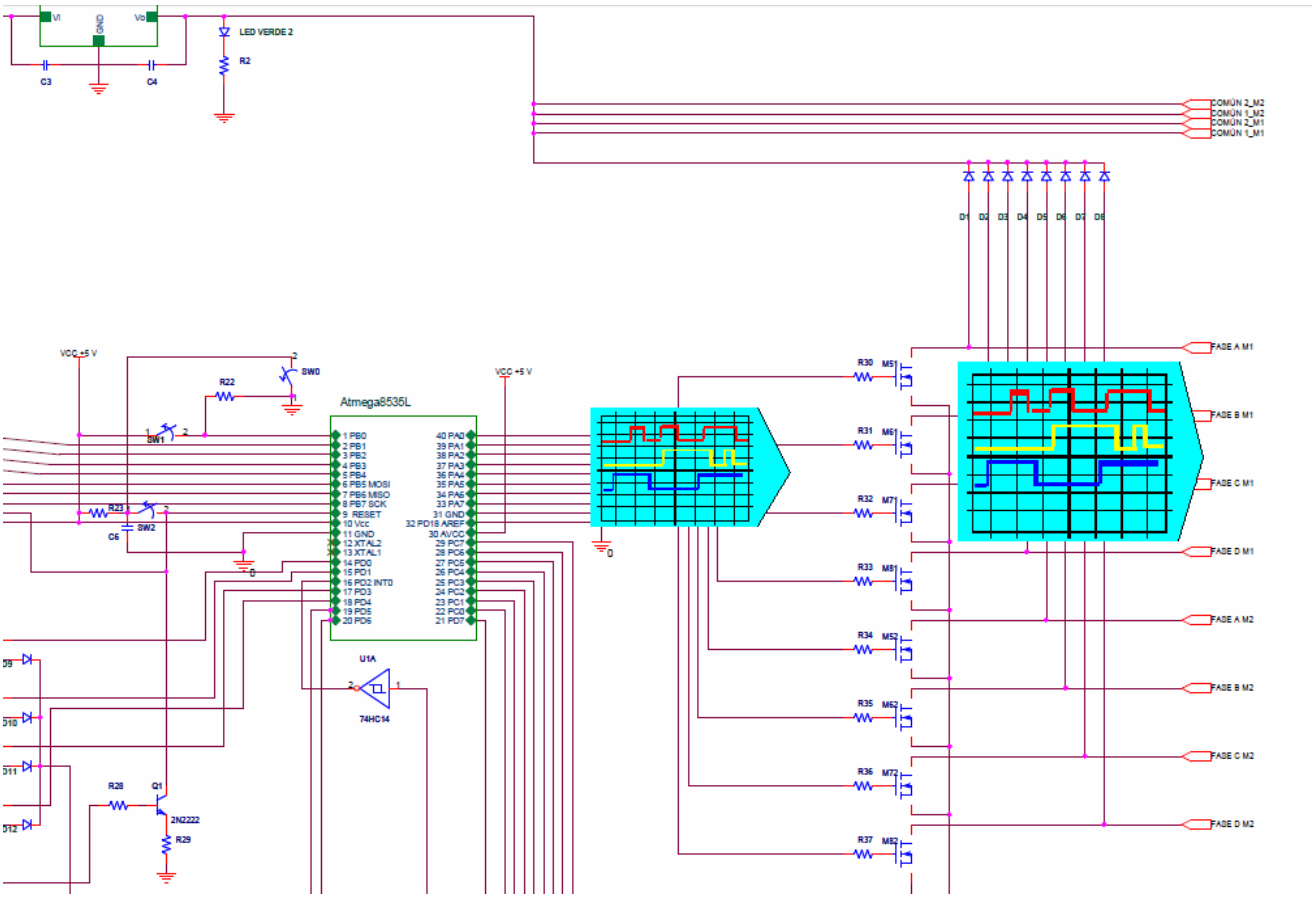


Figura 34: Modo operativo: propagación hacia los transistores de potencia de las señales sintetizadas por el microcontrolador.

Durante el tiempo que se están moviendo los motores, el sistema de seguridad está activo para detectar la posible llegada de la mesa de desplazamiento al final del recorrido en alguno de los dos ejes X o Y; si esto ocurre, se genera una petición de interrupción (gracias a los interruptores SW10 al SW17 que serán pulsados por parejas) hacia la unidad de control, la cual, de no ser atendida en un lapso de tiempo concreto, (determinado por el tiempo de carga del condensador C5) provoca que se desconecten los transistores de potencia de la alimentación, haciendo lucir un led rojo, parando inmediatamente los motores y quedando bloqueado el sistema hasta que el usuario actúe sobre el interruptor (SW5) para desbloquearlo.

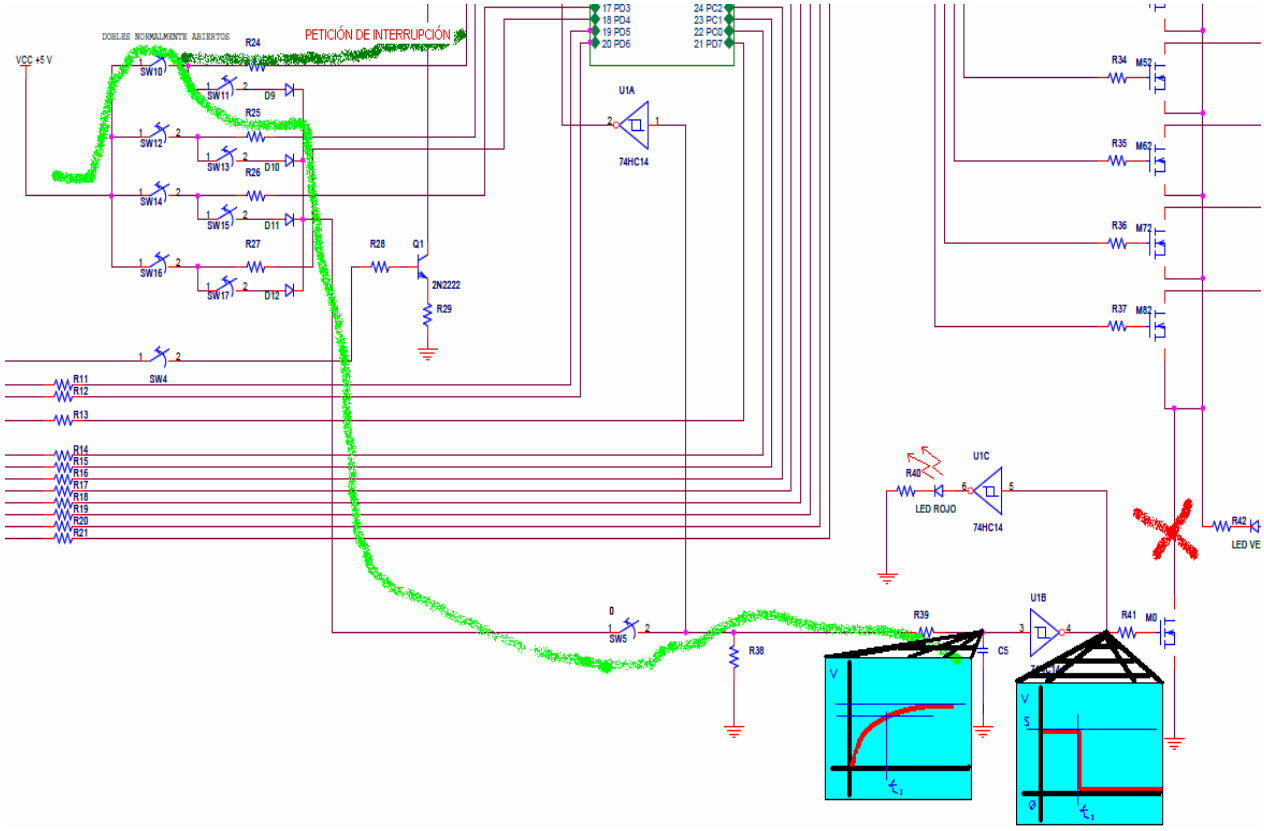


Figura 35: Detalle gráfico del funcionamiento del sistema de seguridad.

Como se puede observar en la figura 37, cuando se cierra una pareja cualquiera de interruptores, por ejemplo SW10 y SW11, los 5 voltios de la alimentación llegan a la red resistiva formada por R38 y R39 haciendo que el condensador C5, inicialmente descargado, se cargue a 5 voltios de forma exponencial como se ve en el primer gráfico azul. Cuando llegue el momento t1 la tensión en C5 será próxima a 5 v y estará dentro del umbral de tensión que la puerta lógica NOT (U1B) interpreta como un 1 lógico, y esto provocará que, a la salida de la puerta lógica, se produzca una transición (como se puede ver en el segundo grafico azul) de 5 v (1 lógico) a 0 v (0 lógico) que hace que el MOSFET “M0” actúe como un interruptor abierto al paso de la corriente que proviene de los transistores de potencia que excitan a los motores, provocando su parada inmediata, incluso aunque las señales de excitación de los MOSFET de potencia no cesen por algún motivo; dichos 0 v se dirigen a la entrada de una segunda puerta NOT (U1C) que hace lucir el led rojo para advertir al usuario de la situación.

4.2.4.2 Modo de detección de códigos.

Este modo de funcionamiento surge por el tipo de puerto de comunicaciones que se ha elegido, el puerto paralelo del PC.

Se trata de uno de los diferentes puertos de comunicaciones de que puede disponer una computadora personal, y aunque actualmente casi en desuso, muy apropiado cuando se quiere tener

un control absoluto sobre unas pocas señales con pocas exigencias desde el punto de vista de la programación.

El puerto paralelo del PC, se rige por el estándar creado por IBM para el manejo de impresoras y consiste a nivel de hardware de un conector de 25 pines cuyo aspecto y funcionalidad se muestra en la figura 34. [2]

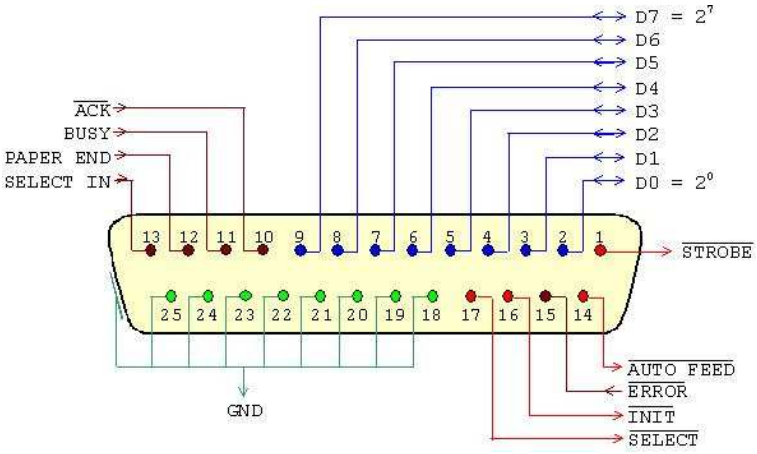


Figura 36: Las señales del puerto paralelo según la norma de IBM.

Como se puede apreciar, se dispone de 8 bits de datos en paralelo (D0 a D7, pin2 al pin 9 respectivamente) que pueden ser tanto de salida como de entrada hacia el conector. También se dispone de 9 señales que sirven para el control del puerto. Como se puede apreciar, los nombres de las señales hacen alusión a ciertos aspectos del manejo de una impresora, tales como “paper end” o “auto feed”, dado que esta fue la funcionalidad para la que fue concebido.

En el plano software del puerto, este se compone de tres registros de memoria, cada uno de los cuales está directamente relacionado con las señales que aparecen en el conector.

En la siguiente figura se puede ver cuál es dicha correspondencia.

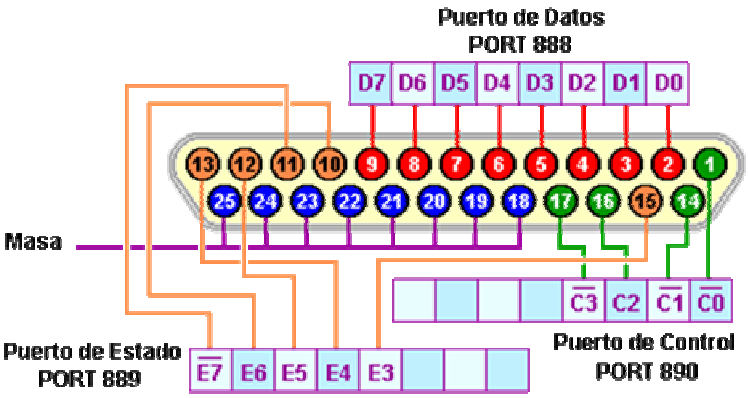


Figura 37: Registros asociados a las señales del puerto paralelo.

Con el paso del tiempo, el estándar del puerto paralelo ha ido evolucionando para adaptarse a las necesidades del mercado y de los usuarios, de modo que hoy día, podemos hablar de varias versiones del puerto paralelo que difieren, básicamente, en el tamaño y número de registros de memoria y en la lógica de sus señales, es decir, en si un nivel alto de tensión en una señal determinada debe considerarse como 1 lógico o como 0 lógico.

Normalmente, en todos los PC que poseen puerto paralelo, éste se puede configurar para que funcione según la versión más antigua, denominada SPP (Standar Parallel Port) u otras versiones más avanzadas que dependen del modelo de tarjeta instalada en el PC. Dicha configuración se puede realizar desde la BIOS, por ello, dado que no todos los usuarios del sistema de posicionamiento, objeto de este proyecto, tendrán los mismos privilegios sobre la máquina y no podrán entrar en el menú de la BIOS, se implementa el segundo modo de funcionamiento de la tarjeta de control, cuyo funcionamiento detallado se describe a continuación.

Mediante la configuración de los interruptores, SW0 al SW5 el microcontrolador ejecuta una rutina, según a cual, los niveles lógicos que detecta el microcontrolador en el puerto de entrada de datos (Puerto C, pines 21 al 29 del microcontrolador) desde el PC, los presenta en el puerto de salida (Puerto B, pines 1 al 8 del microcontrolador).

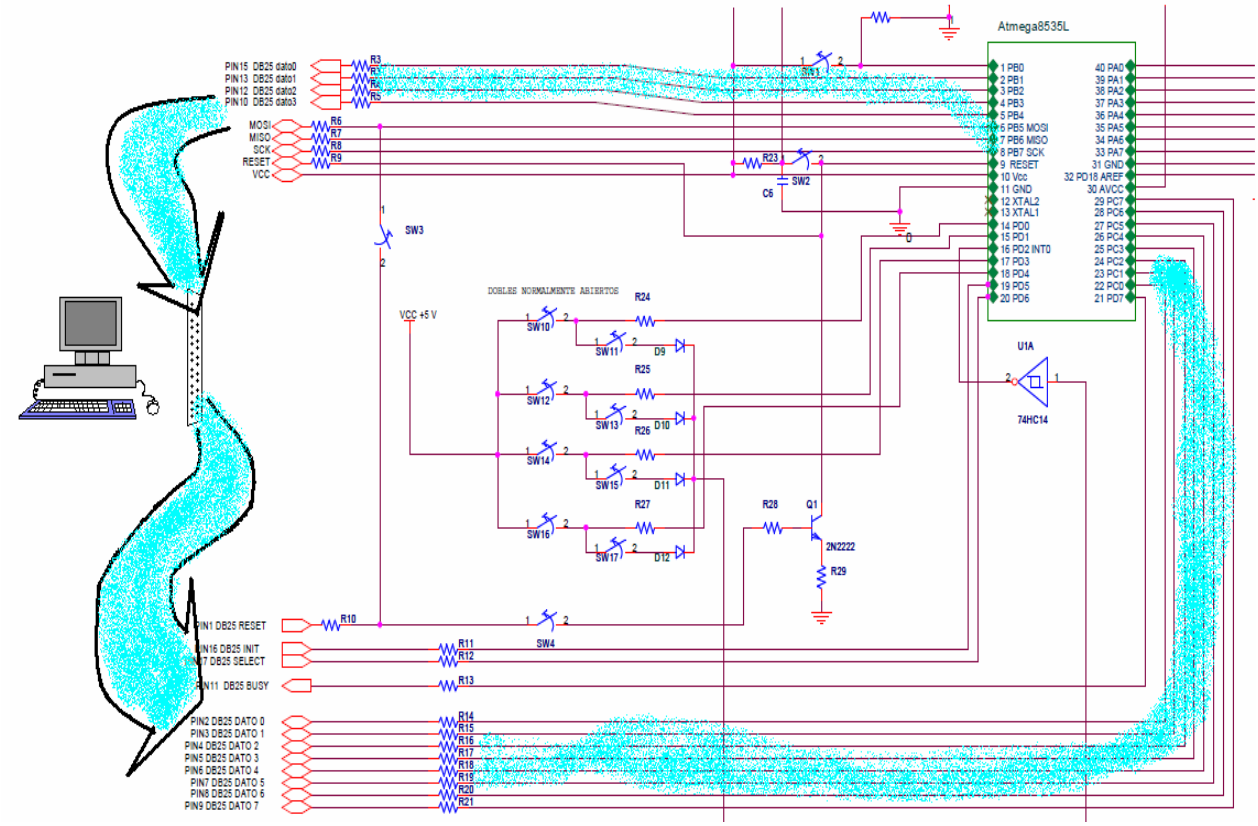


Figura 38: Modo detección de códigos: camino de la información binaria.

De este modo, el PC ejecutará una rutina de generación de códigos, que le servirán para conocer el código que está interpretando el microcontrolador en cada momento, y de ese modo saber qué códigos son los que deberá enviar al microcontrolador para activar las señales INIT o SELECT mencionadas anteriormente.

Resumiendo, dicho modo de funcionamiento está pensado para determinar los códigos que deberán usarse desde el PC independientemente del modelo de tarjeta de control del puerto paralelo que posea el PC utilizado, ya que según el modelo de puerto paralelo, la lógica puede variar, como ya se ha comentado.

5. Desarrollo software.

5.1 Programación del microcontrolador.

La programación del microcontrolador se ha realizado en lenguaje ensamblador.

El programa en ensamblador se ha estructurado de tal forma que queda dividido en diferentes secciones según su funcionalidad.

CUERPO PRINCIPAL: (desde el cual se hacen llamadas a las siguientes rutinas)

RUTINAS DE MOVIMIENTO DE MOTORES
RUTINAS DE COMUNICACIONES
RUTINAS DE ATENCIÓN DE INTERRUPCIONES
RUTINAS DE CONFIGURACIÓN DE PARÁMETROS
RUTINAS DE DETECCIÓN AUTOMÁTICA DE CÓDIGOS
RUTINAS DE LECTURA DE REGISTROS

5.1.1 Cuerpo principal.

El programa principal comenzará a correr desde el momento del encendido del microcontrolador, después de realizarse el correspondiente “reset” consecuencia de la carga del condensador C6 del circuito de reset implementado.

El programa principal se ejecutará de forma secuencial (arquitectura Von-Newman) conformando un bucle infinito dentro del cual se ejecutan las instrucciones que veremos más adelante. [3]

Los programas del PC y del microcontrolador hacen uso de una comunicación asíncrona. A diferencia de una comunicación síncrona, en la cual emisor y receptor están sincronizados por una señal de reloj común que se encarga de determinar en qué momento debe intervenir el receptor o el emisor, en el sistema de comunicación implementado, el control de la comunicación se hace de forma que mientras el PC no detecta un cierto nivel lógico en una determinada línea habilitada por el microcontrolador, no continuará con el proceso, de modo que dicho sistema nos brinda la posibilidad de intercomunicar elementos cuyas velocidades individuales de proceso son muy dispares de una forma más sencilla que mediante una comunicación síncrona. En nuestro caso, el PC (con una frecuencia de reloj del orden de GHz) es mucho más rápido que el microcontrolador, cuyo reloj interno funciona a 8 MHz, de modo que con el tipo de comunicación implementado garantizamos que la comunicación sea posible. [4]

Para entender adecuadamente el funcionamiento general del sistema, se detallan a continuación, en la figura 39, las operaciones que lleva a cabo la rutina de atención al PC citada anteriormente.

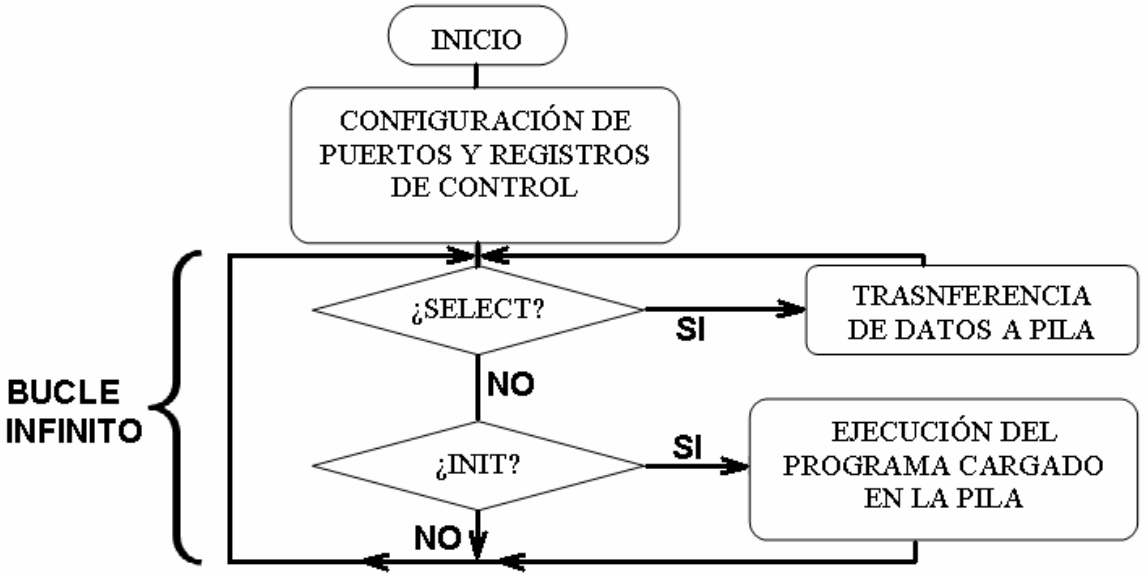


Figura 39: Organigrama de funcionamiento de la rutina de atención al PC.

El microcontrolador está continuamente esperando la activación de dos señales, INIT o SELECT. Cuando la señal activada por el PC es SELECT, el micro entiende que se pretende iniciar una transferencia de datos desde el PC al micro, que conforman el código del programa que el PC (o lo que es igual, el usuario) necesita que ejecute el microcontrolador. Si la señal que activa el PC es INIT, el microcontrolador comenzará con la ejecución de un programa cuyo código debe haber recibido previamente.

Cada uno de los programas que puede ejecutar el microcontrolador, puede necesitar información adicional en forma de parámetros, y deberán ser enviados por el PC en el orden adecuado para que el microcontrolador los almacene en la pila de datos.

Caso señal SELECT activa:

Primero se desencadena una transferencia de datos, 8 bits, desde el PC al micro que se explica con detalle más adelante.

Dichos 8 bits se almacenarán en la pila FILO (First In Last Out) de datos del microcontrolador. FILO es el nombre que recibe un determinado manejo de una estructura de datos, se muestra de forma gráfica en la figura 40. [5]

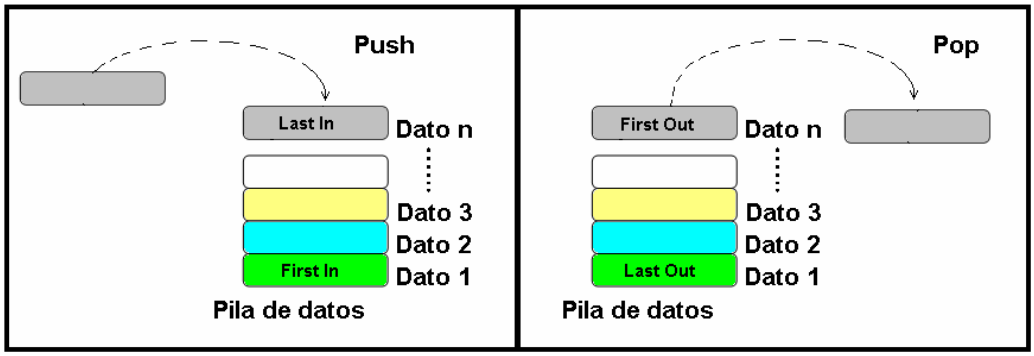


Figura 40: Esquema de funcionamiento de una pila de datos FILO.

Después se vuelve a quedar a la espera de la activación de una de las dos señales.

Caso señal INIT activa:

Se desencadena un proceso mediante el cual el microcontrolador va tomando los datos almacenados en la pila de forma ordenada y los interpretará de forma que sirvan para seleccionar un programa a ejecutar y una serie de parámetros que determinan cómo será ejecutado.

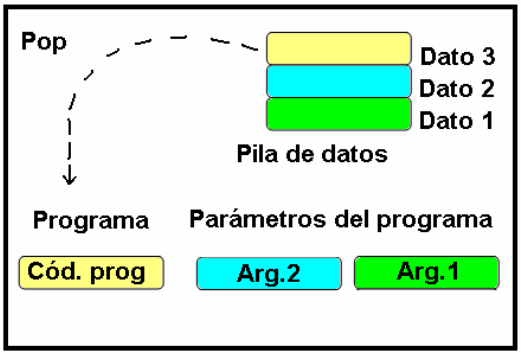


Figura 41: Proceso de descarga de la pila FILO para ejecutar un programa.

Una vez ejecutado el programa se vuelve a quedar a la espera de la activación de una de las dos señales.

También el programa principal del microcontrolador es el responsable de garantizar condiciones de seguridad, tales como resolver la situación de “llegada de un motor a final de

recorrido", dicha situación generará una interrupción, gracias a unos interruptores mecánicos, que será atendida para hacer retroceder al motor adecuado un cierto número de unidades hasta que dicho interruptor deje de activar la interrupción.

5.1.2 Rutinas de movimiento de motores.

Se han implementado 4 rutinas para generar las señales que provocarán el movimiento de los dos motores en los dos posibles sentidos.

Las cuatro rutinas tienen el mismo *esqueleto* funcional, pero cada una de ellas genera un patrón de señal diferente.

Su funcionamiento se basa en un registro de desplazamiento de 8 bits. Los cuatro bits más significativos afectan a un motor y los otros cuatro bits de menor peso, están asignados al otro motor.

Cada una de las rutinas interviene en el *nibble* (o conjunto de 4 bits consecutivos) adecuado para generar un código rotatorio como el que muestra la figura 42 según el motor y sentido de movimiento deseado.

Secuencia motor 1 derecha →
10000000
01000000
00100000
00010000
Secuencia motor 1 izquierda →
00010000
00100000
01000000
10000000
Secuencia motor 2 derecha →
00001000
00000100
00000010
00000001
Secuencia motor 2 izquierda →
00000001
00000010
00000100
00001000

Figura 42: Secuencias de activación de los motores.

Esto da lugar a la siguiente representación en el tiempo.

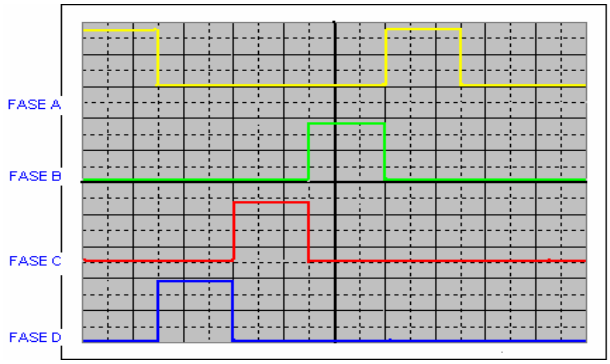


Figura 43: Representación temporal de la secuencia para el giro a izquierdas.

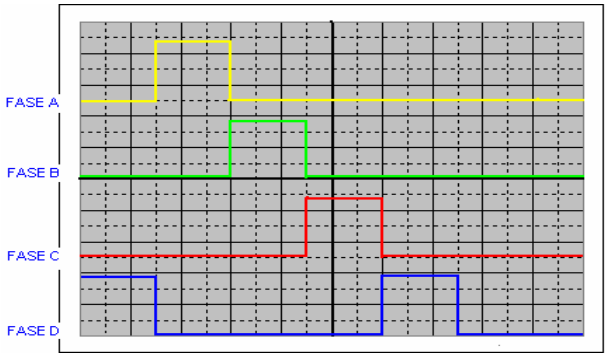


Figura 44: Representación temporal de la secuencia para el giro a derechas.

Una vez se tiene el código deseado en el registro, este se hace llegar al puerto de salida ***A*** del microcontrolador mediante la instrucción *out* con lo que se *ataca* a los transistores mosfet de potencia de los motores. Después de dicha operación se espera un tiempo determinado por un temporizador programable antes de dar paso la siguiente rotación del código.

Estas operaciones están incluidas en un bucle que será el responsable de iterar el proceso tantas veces como pasos se necesite que de el motor (para más detalle ver código ensamblador en el anexo II).

5.1.3 Rutinas de comunicaciones.

Todo el funcionamiento del micro está basado en la rutina de comunicaciones de atención al PC, es decir, el micro sólo puede recibir un código de programa que se desea ejecutar, y la señal de ejecución de programa.

Todos los programas que alberga la memoria Flash del micro provocan actuaciones bien hacia los drivers, bien hacia los registros internos.

Por tanto, podemos hablar de procesos de comunicación al hablar de la rutina de atención al PC y del programa de lectura del registro de pasos interno del micro.

Como ya se ha mencionado anteriormente, el microcontrolador, desde el encendido, entra en un bucle de escucha a las señales INIT y SELECT que llagan desde el PC.

En el momento que se activa alguna de ellas, se desencadena una rutina de comunicación.

Las rutinas de comunicaciones del programa ensamblador están formadas por bucles consecutivos que provocan la espera del microcontrolador hasta que las señales enviadas por el PC estén al nivel lógico adecuado, e instrucciones in y out que hacen cambiar las señales del puerto C de comunicaciones del microcontrolador haciendo posible una transferencia de datos desde o hacia el PC.

Entrando en algo más de detalle, podemos ver cómo trabaja la rutina de comunicaciones del bucle de atención al PC, para lo cual debemos tener presente la figura 45.

1º Desde el momento que el usuario presiona un botón hasta la primera línea vertical del gráfico (color rojo), se puede observar que todas las señales permanecen a nivel lógico bajo y el PC presenta 8 bits de datos (en color amarillo). Durante ese intervalo el micro espera a que la señal SELCT o INIT sea activada por el PC.

2º Coincidiendo con la primera línea vertical vemos que el PC activa la señal SELECT con lo que indica al micro que hay un dato en el puerto y lo puede leer. Desde ese mismo instante y hasta la segunda línea vertical (intervalo marcado como I) el microcontrolador realizará las operaciones oportunas para cargar el dato en la pila de datos FILO adecuadamente. Una vez hecho esto, el micro activa la señal BUSY para indicar al PC que ya ha procesado el dato.

3º A partir de dicho momento, es decir, desde la segunda hasta la tercera línea vertical, el PC retira el dato del puerto dejándolo en reposo para evitar posibles errores e indica, desactivando la señal SELECT, al micro que queda preparado para recibir los primeros 4 bits de los ocho que le ha mandado para realizar la comprobación del dato.

4º Entre las líneas verticales tres y cuatro, justo antes del intervalo marcado con II, el micro procesa el dato y presenta los 4 bits de mayor peso por el puerto B y pone a nivel bajo la señal BUSY para dar a entender al PC que puede leerlos.

5º Desde ese instante de tiempo, transcurre el intervalo marcado como II durante el cual el PC procesa los 4 bits de mayor peso y activa la señal SELECT para indicar al micro que puede mandar al puerto los 4 bits de menor peso.

6º Acto seguido, entre las líneas verticales quinta y séptima, el micro realiza las operaciones necesarias para presentar los 4 bits de menor peso del dato que recibió y activa la señal BUSY de nuevo, para indicar al PC que puede leerlos.

7º Desde dicho instante de tiempo, transcurrirá el intervalo marcado como III durante el cual el PC lee los 4 bits de menor peso y realiza una serie de operaciones para comparar el dato resultante

con el dato que envió originalmente. Si hay coincidencia en la comprobación, la comunicación habrá resultado exitosa y el PC activará la señal INIT como aviso hacia el micro de que el dato interpretado es el correcto.

8º Desde ese instante, hasta la penúltima línea vertical, el micro realiza la operación de cargar el dato en la pila FILO y espera a que el PC desactive la señal SELECT como muestra de finalización de las comunicaciones.

9º Una vez llegado dicho momento, el micro deja en reposo el bus de comunicaciones del puerto B y desactiva la señal BUSY, con lo que provocará que el PC se de por enterado y desactive finalmente la señal INIT quedando todas las señales en reposo y finalizando la sesión de comunicación hasta que se vuelva a desencadenar otro igual.

10º Una vez realizadas las operaciones anteriores el micro vuelve a atender a las señales del PC.

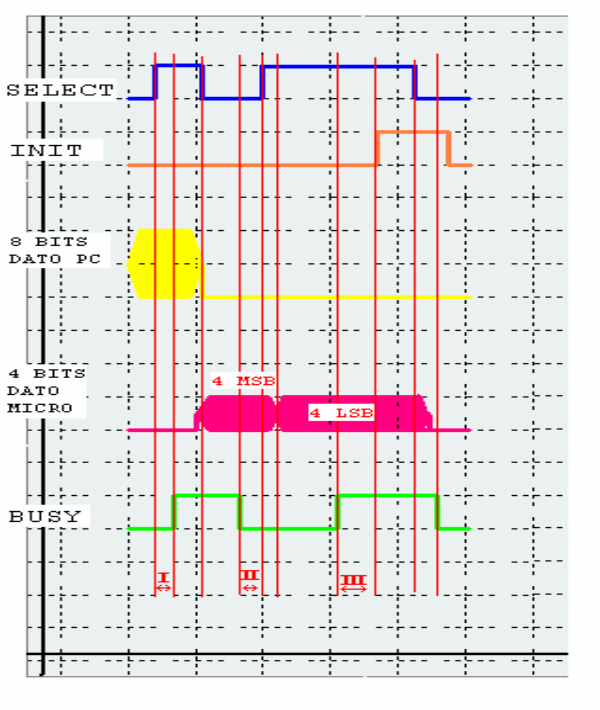


Figura 45: Cronograma de la rutina de comunicaciones.

De forma consecuente con lo que se acaba de explicar, el cronograma resultante de la acción de mandar ejecutar un programa desde el PC por el usuario sería el mostrado en la parte superior de la figura 46

Tal y como se aprecia, se repite tres veces el proceso ya explicado para mandar los dos parámetros necesarios en este ejemplo concreto, y después el código del programa que debe ejecutarse. Para finalizar se activa la señal INIT por parte del PC para indicar al micro que debe comenzar la ejecución.

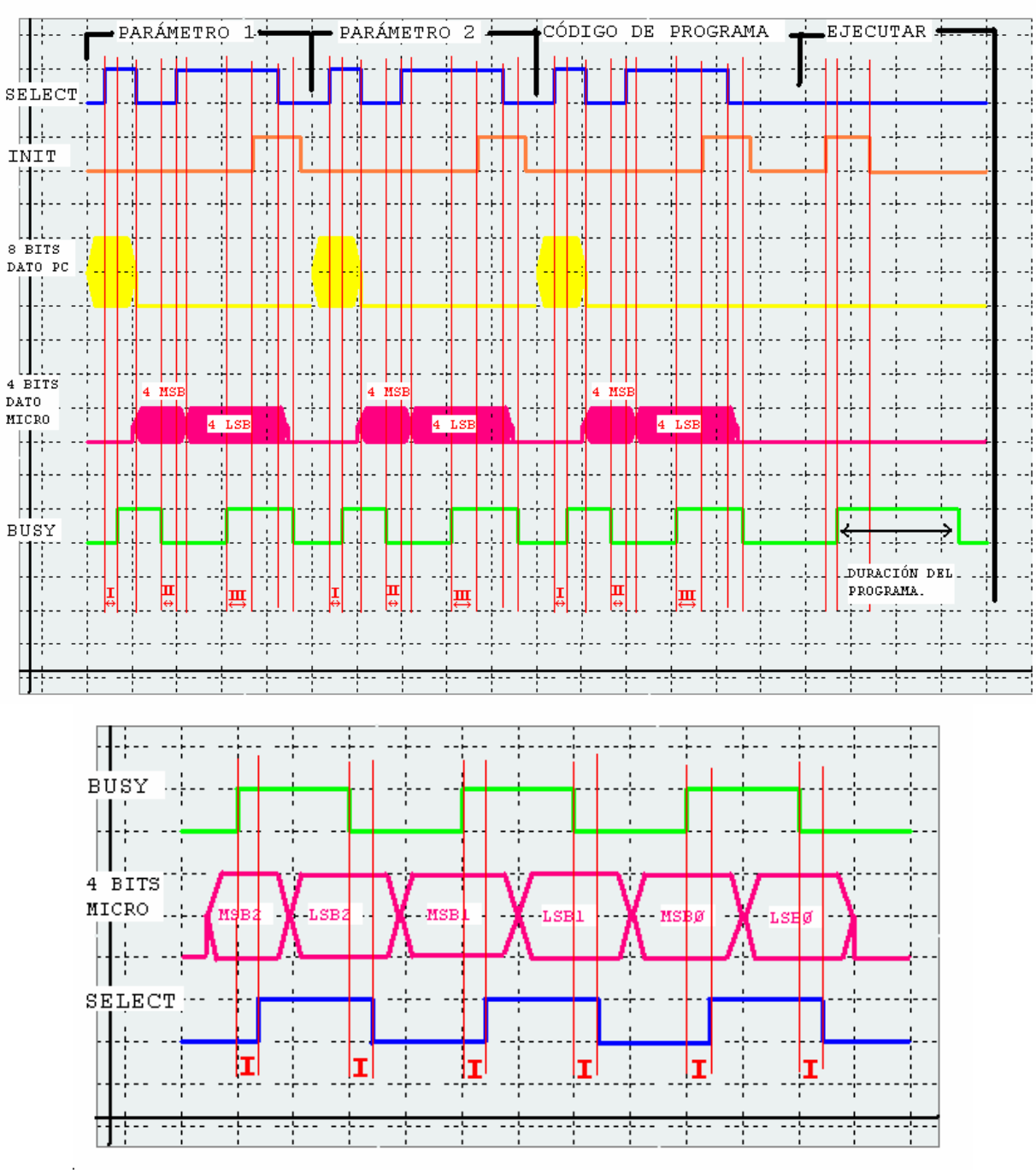


Figura 46: Cronograma de la lectura del registro de 24 bits interno de pasos del microcontrolador.

Para el caso del programa que lee el registro de pasos interno de 24 bits del micro, el patrón de señales que provocan la transferencia de los 24 bits sería el mostrado en la parte inferior de la figura 46, después de haberse realizado la petición de lectura de dicho registro mandando, tal y como se acaba de ver en la parte superior de la figura 46, ejecutar el programa correspondiente al micro. Como se puede apreciar, el proceso es similar al de lectura de cuatro en cuatro bits para realizar la comprobación del dato.

5.1.4 Rutinas de atención de interrupciones.

Tal y como se puede ver en el código del programa en ensamblador, se hace uso de la señal de interrupción externa INT0, cuyo vector de interrupción apunta a la rutina de atención de la interrupción.

La atención de interrupciones se habilita al comienzo del programa principal del microcontrolador, y dicha señal externa de interrupción llega al micro siempre que alguno de los 8 interruptores mecánicos que se alojan en diferentes puntos del recorrido de los motores, están siendo presionados por parejas. A continuación se muestra el funcionamiento del sistema.

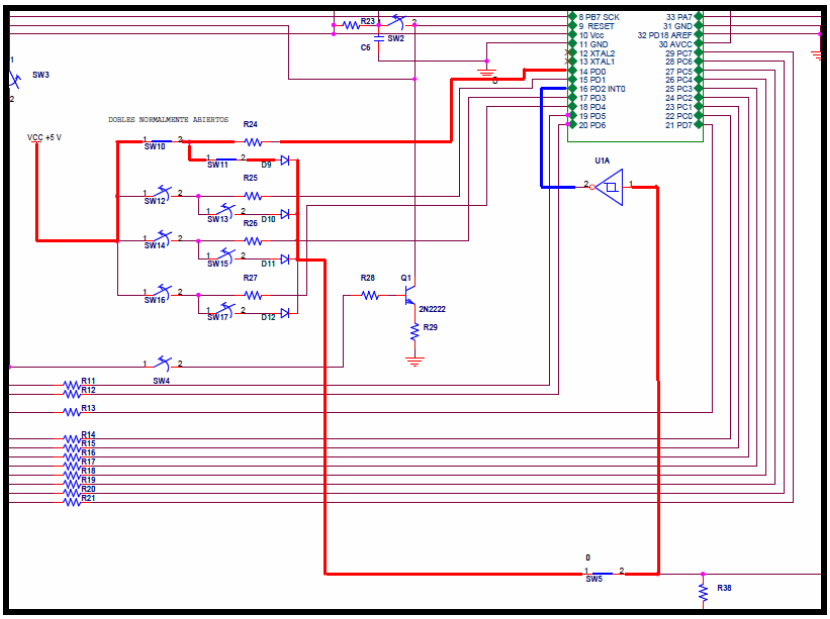


Figura 47: Detalle del estímulo eléctrico de los interruptores de final de recorrido a la dcha. en el eje X.

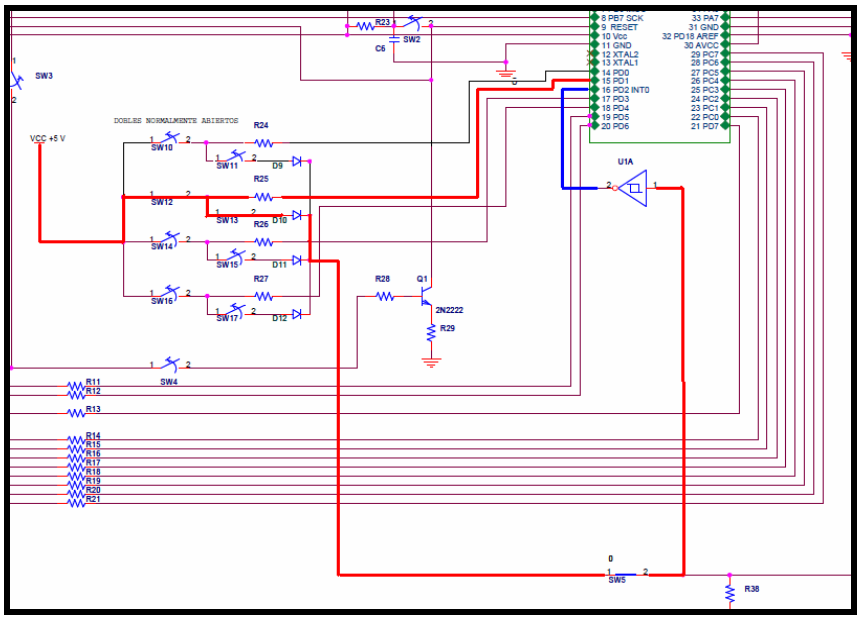


Figura 48: Detalle del estímulo eléctrico de los interruptores de final de recorrido a la izqda. en el eje X.

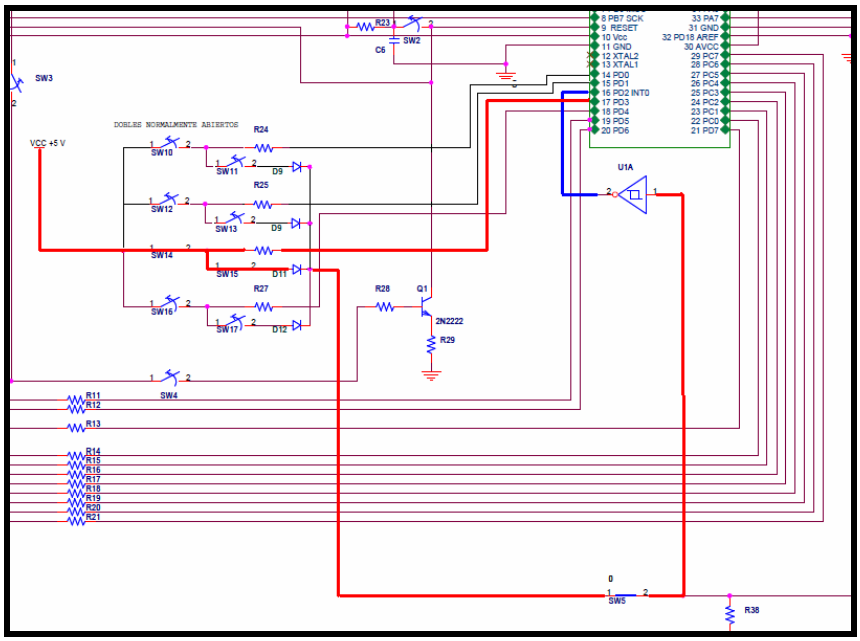


Figura 49: Detalle del estímulo eléctrico de los interruptores de final de recorrido a la dcha. en el eje Y.

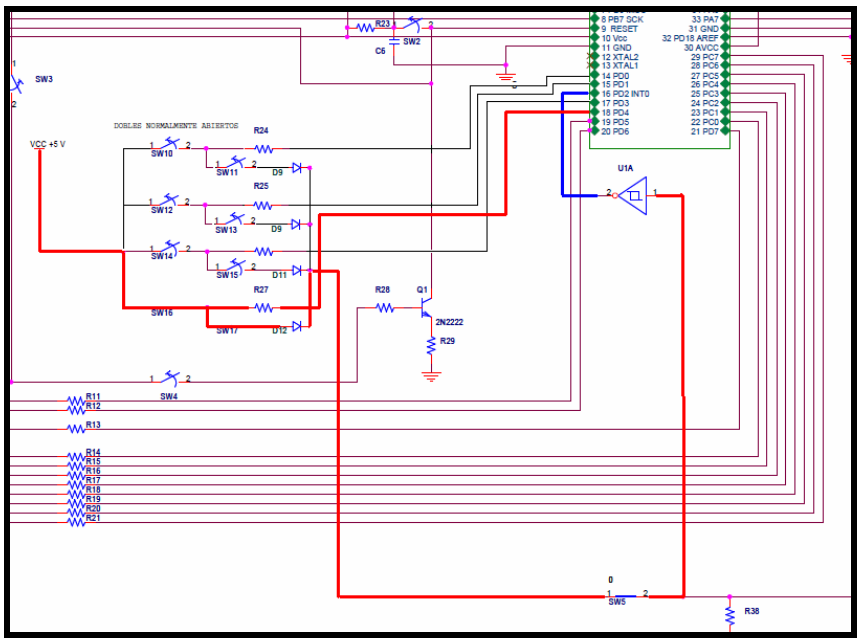


Figura 50: Detalle del estímulo eléctrico de los interruptores de final de recorrido a la izqda. en el eje Y.

Los interruptores se encuentran emplazados en los extremos recorrido por parejas, de modo que, al llegar el tablero de la mesa de posicionamiento a dichos puntos, pulsará una pareja de interruptores provocando que la señal INT0 se active desde uno de ellos y el segundo interruptor de la pareja activa otra señal de entrada al micro que sirve para identificar o codificar la pareja concreta de interruptores que ha sido presionada, y por tanto la posición del tablero móvil.

De este modo, la rutina de atención de la interrupción lo primero que hace es deshabilitar las interrupciones mientras la está atendiendo. Después lee el puerto de entrada para interpretar el dato y conocer de qué pareja de interruptores se trata. Una vez conocida esta información activa la rutina de movimiento del motor oportuno para desplazarse una cantidad fija de unidades en el sentido que provoca el alejamiento del tablero de la mesa de posicionamiento de los interruptores accionados.

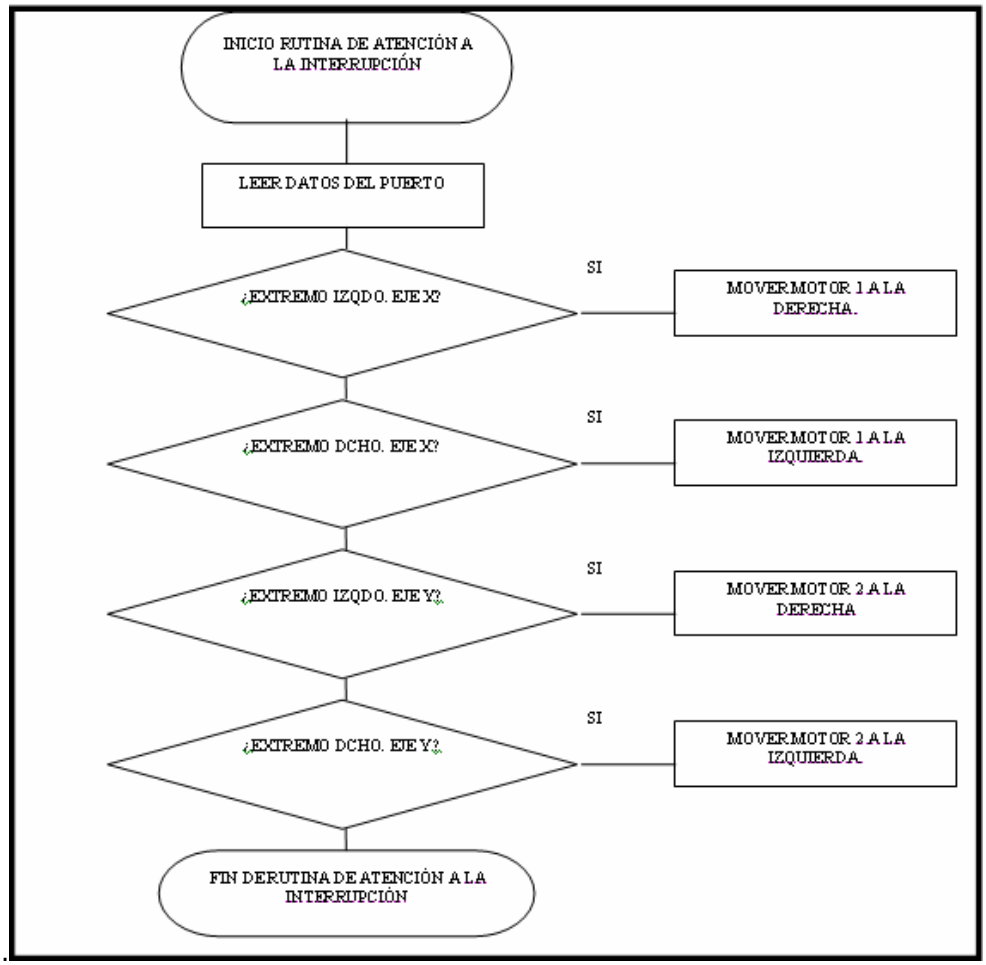


Figura 51: Organigrama de las rutinas de interrupción por final de recorrido.

Una vez finalizado el proceso se vuelve a habilitar la gestión de interrupciones y se regresa a la rutina de atención al PC.

5.1.5 Rutinas de configuración de parámetros.

Para la generación de las señales de control de los motores, se hace uso de un registro de desplazamiento de 8 bits tal y como se explica en el apartado 5.1.2. En dicho registro se va desplazando un uno lógico en el sentido adecuado y a una velocidad directamente proporcional a la velocidad de giro del rotor del motor. Por tanto, aparecen una serie de parámetros que deben ser

determinados para el correcto funcionamiento, tales como el tiempo entre desplazamientos, o la velocidad de variación de dicho parámetro.

Dada la arquitectura del contador implementada en el micro ATMEL 8535, la cual se ve en la figura 52, uno de los parámetros que determinan la velocidad de la cuenta es al valor del *prescaler,* con dicho parámetro se puede variar la frecuencia con la que se incrementa el contenido del registro de cuenta.

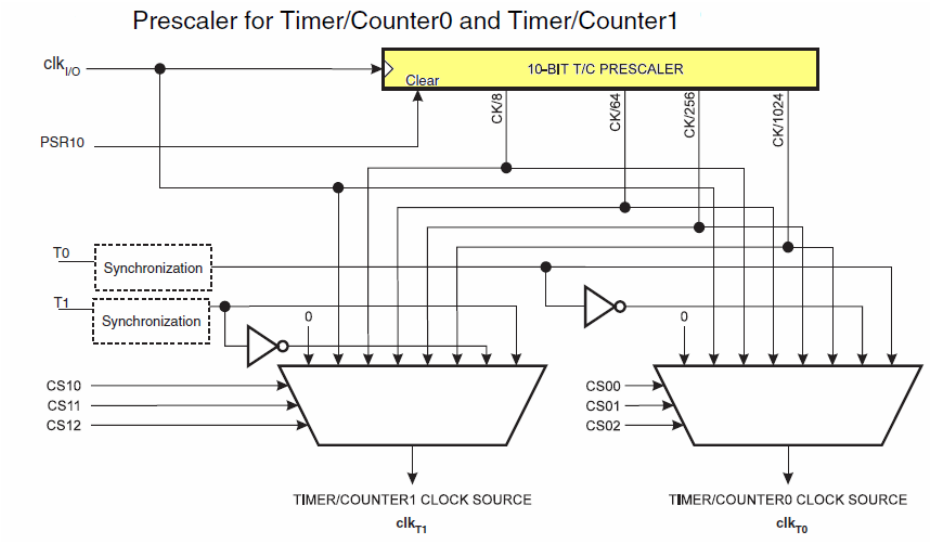


Figura 52: Detalle de arquitectura de los contadores temporizadores internos del microcontrolador.

Por otro lado, el valor inicial desde el que se comienza la cuenta o también llamado valor de precarga del contador, permite variar el tiempo que tardará el registro de cuenta en llegar al máximo (valor FF en hexadecimal en nuestro caso) y provocar una interrupción por desbordamiento.

El resultado de variar estos parámetros antes de arrancar el contador se puede expresar de forma gráfica como sigue:

Si nos fijamos en la figura 53, podemos observar que el cambio de estado de las señales de excitación de los motores se produce en el momento que se alcanza el valor de desbordamiento del contador, FF como ya hemos dicho, puesto que se trata de un contador de 8 bits.

Cada vez que se produce un desbordamiento, se genera una interrupción interna en el programa del micro y se ejecutará la rutina que cambia la señal de excitación actual por la que le toque según estemos girando en un sentido o en otro.

Dado que el tiempo que tarda en incrementarse una unidad el contador es un lapso de tiempo constante (posee una desviación despreciable para esta aplicación) determinado por la configuración del prescaler (ver figura 52), se puede jugar con dicha configuración y el valor desde el que se

comienza la cuenta y la frecuencia con la que variamos dicho valor inicial de cuenta para controlar la velocidad y la aceleración de los motores con gran precisión.

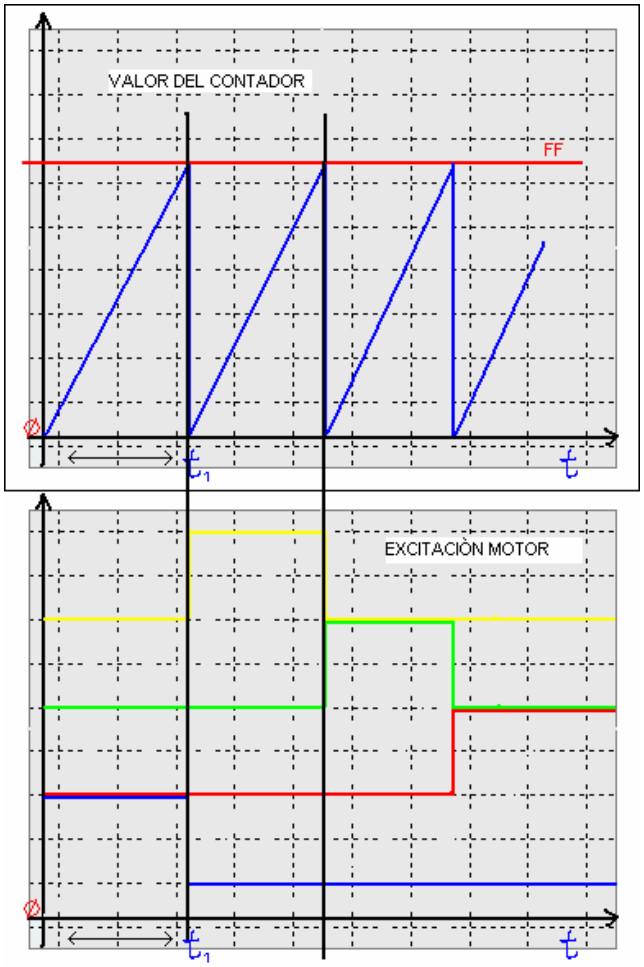


Figura 53: Representación gráfica de las señales de temporización y síntesis de las señales de control de los motores.

Como se acaba de decir, si, además tenemos en cuenta que cada vez que se activa una interrupción por desbordamiento del timer, la rutina de atención a dicha interrupción provoca un desplazamiento de los bits del registro de desplazamiento utilizado para sintetizar la señal de excitación de los motores, es decir, se provoca un paso del motor deseado, podemos pensar en el concepto de aceleración del motor si provocamos un incremento de x unidades del valor de precarga del temporizador durante la atención a la interrupción con lo que dicho valor x será directamente proporcional a la aceleración.

Para entenderlo mejor se puede expresar gráficamente como sigue:

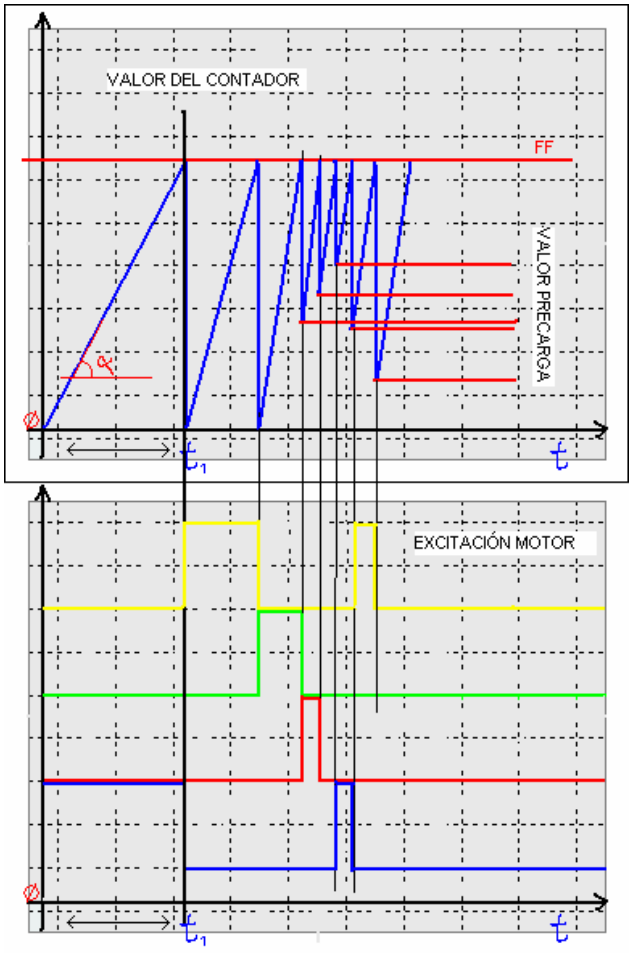


Figura 54: Representación gráfica de los conceptos velocidad y aceleración respecto a las señales de control de los motores.

Donde el parámetro α, se es directamente proporcional a la velocidad de giro del motor y el parámetro ∂α es directamente proporcional a la aceleración del mismo.

Resumiendo, estos parámetros: *valor de prescaler*, *valor de precarga* e *incremento del valor de precarga*, se traducen en la velocidad de giro y en la aceleración de los motores.

Con ello se consigue optimizar la respuesta de par motor y velocidad de los motores, ya que a bajas revoluciones los campos magnéticos conseguidos son elevados y vencen mejor la inercia mecánica del sistema de engranes y a mayor frecuencia aumenta la velocidad de giro y disminuye el par, siendo conceptualmente un efecto parecido al de una onda modulada en ancho del pulso (PWM).

Por tanto, para poder seleccionar y/o modificar dichos parámetros, se ha implementado una rutina que permite alojar en los registros oportunos los datos necesarios.

El funcionamiento del programa es el siguiente:

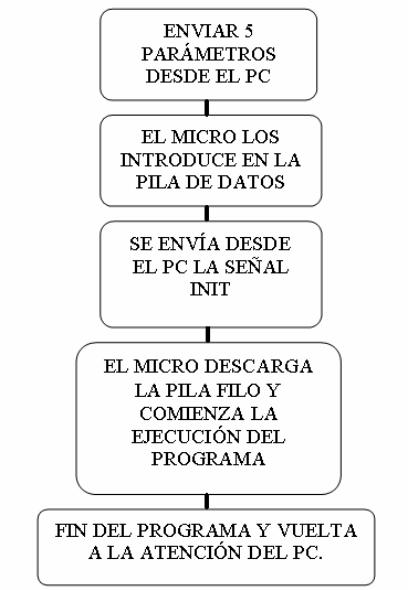


Figura 55: Organigrama de actuación para la configuración de los parámetros del microcontrolador.

1º se mandan desde el PC los datos a cargar en los cinco registros implicados en el orden adecuado hacia la pila de datos del microcontrolador.

2º el programa del microcontrolador toma uno a uno los datos introducidos en la pila siguiendo los pasos habituales de manejo de pilas tipo FILO y los va cargando en los registros utilizados a tal efecto.

3º devuelve el control a la rutina de atención al PC.

Dichos parámetros (velocidad y aceleración) se determinan de forma experimental. Una vez terminada la fase experimental y determinados estos parámetros, se modifica el cuerpo principal del programa y se incluyen instrucciones que almacenarán los valores determinados para los parámetros de forma automática cada vez que se alimente el microcontrolador, dejando que esta configuración sea transparente para el usuario final, ya que la alteración de dichos parámetros está directamente relacionada con el consumo de los motores y una selección errónea de valores podría dañar de forma irremediable los componentes electrónicos que conforman los drivers de los motores.

5.1.6 Rutinas de detección automática de códigos.

Dada la variedad de modos de funcionamiento del estándar que sigue el puerto paralelo de un PC, y aunque se puede seleccionar desde el menú de la BIOS del PC dicho modo de funcionamiento, se ha implementado un programa en el microcontrolador, que permitirá descodificar

los 256 posibles códigos que puede mandar el PC a través de los 8 bits de datos hacia el microcontrolador.

Dicho programa implementa una función de *eco* hacia el PC, de modo que el PC envía códigos desde 0 hasta 255 y el micro devuelve al PC el código que está interpretando (ver figura 56), el cual será consecuencia de haber seleccionado un modo de funcionamiento determinado del puerto paralelo del PC, y de este modo el PC puede "decidir" cuáles son los códigos que activarán correctamente las señales SELECT e INIT. Esta es una funcionalidad que proporciona compatibilidad para aquellos casos en que no sea posible acceder a la configuración del puerto paralelo del PC.

El organigrama de la rutina es el siguiente:

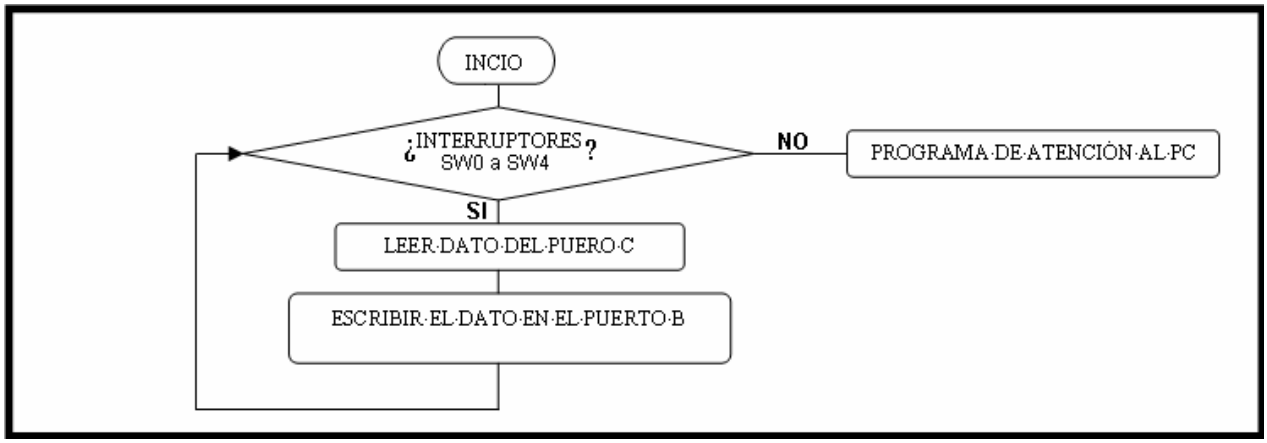


Figura 56: Organigrama de la rutina de atención al PC.

Dado que dicha funcionalidad es sólo necesaria a efectos de configuración del sistema en el momento de la primera instalación, por tanto, dicha rutina sólo entrará en funcionamiento al encender la tarjeta de control si se tiene la siguiente configuración de los micro-switch que hay alojados en la PCB:

Sw0 → abierto.
Sw1 → cerrado.
Sw2 → cerrado.
Sw3 → cerrado.
Sw4 → abierto.

Con dicha configuración de micro-switch, primero debemos encender la tarjeta de control y después lanzar el programa desde el PC para que funcione correctamente.

5.2 Programa interfaz de usuario.

5.2.1 Generalidades.

El programa residente en el PC que conforma la interfaz entre el usuario final y el sistema de posicionamiento se desarrolla en lenguaje VISUAL BASIC , para lo cual se utiliza la plataforma .NET que proporciona funcionalidades añadidas para un fácil desarrollo de aplicaciones que utilizan ventanas y que deben trabajar sobre el sistema operativo Windows.

El programa de interfaz con el usuario, presenta una ventana principal, cuyo enfoque de diseño pretende ser minimalista e intuitivo para que sea muy sencillo de manejar. La ventana principal muestra de forma esquemática el sistema de posicionamiento. Dicho esquema pretende ser interactivo, de modo que el usuario sólo deberá presionar un botón para lograr que se produzca el desplazamiento deseado de los motores y en pantalla se podrá ver las coordenadas a las que se ha desplazado.

5.2.2 Apariencia y funcionalidad.

Al lanzar la aplicación aparece la pantalla mostrada en la figura 57 en la cual se indica al usuario que debe presionar el botón para iniciar una prueba de detección del hardware.

Figura 57: Pantalla de inicio de la aplicación.

Al pulsar, la aplicación se comunica con la tarjeta de control, enviando unos códigos de prueba, de modo que verifica que las comunicaciones no fallan. Acto seguido muestra la siguiente pantalla.

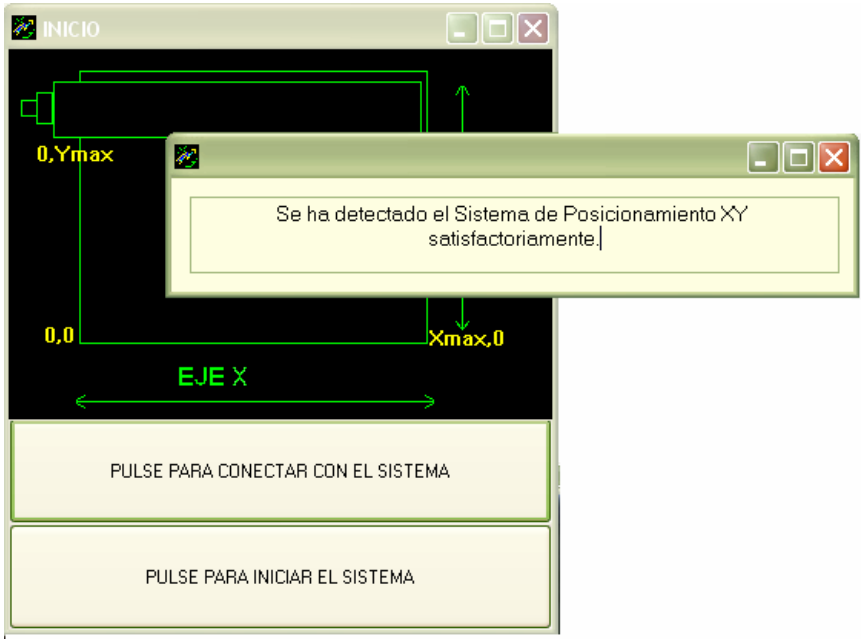


Figura 58: Pantalla de comunicación establecida con éxito.

Cuando el usuario pulsa el botón “PULSE PARA INICIAR EL SISTEMA” el sistema situará la plataforma móvil de la mesa en las coordenadas (0,0) haciendo uso de los interruptores de final de recorrido. Una vez llegado este punto, desaparece la pantalla de la figura 58 y aparece la pantalla principal de la aplicación que permitirá manejar la posición del tablero móvil.

Tal y como se puede ver en la figura 59, aparecen dos “pestañas” dentro de la ventana principal, cada una de ellas, correspondiente a un modo de funcionamiento.

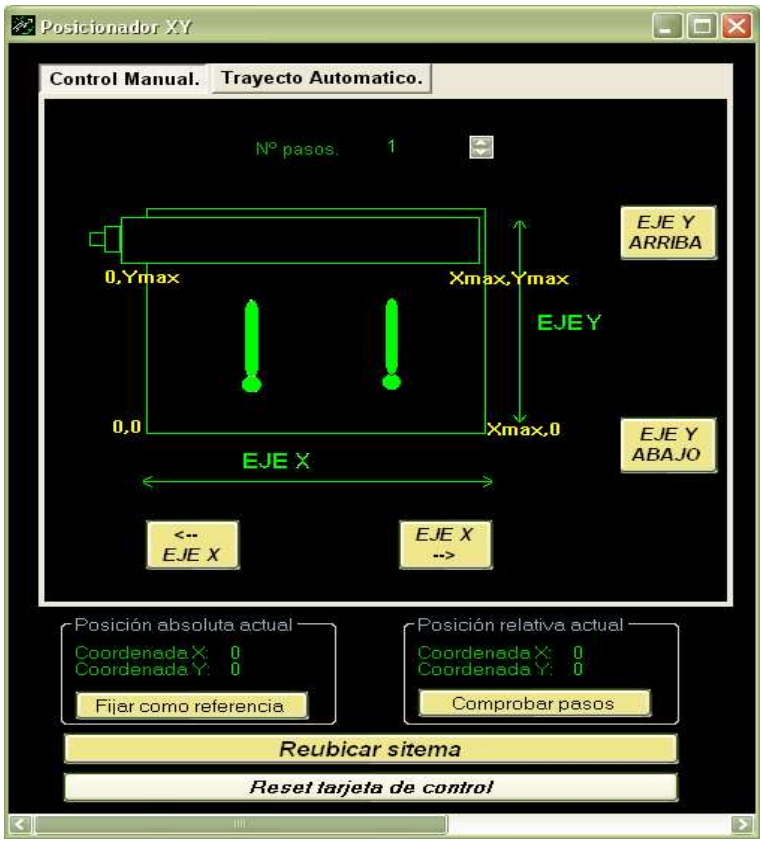


Figura 59: Pantalla principal de la aplicación: control manual.

El primero, “control manual”, incluye un campo donde se podrá escribir el número de unidades que se desea desplazar un motor. Dicho valor estará acotado en el intervalo [1 – 65564]

Dicho modo de funcionamiento presenta cuatro botones que se corresponden con cada uno de los dos ejes y de los dos sentidos posibles para cada eje de desplazamiento.

En dicho modo, por tanto, bastará con indicar el número de unidades que se quiere desplazar un motor y presionar el botón correspondiente con el eje y sentido de la marcha, para provocar dicho desplazamiento en el sistema de posicionamiento XY.

Fijándonos en la figura 60, vemos el segundo modo de funcionamiento, “trayecto automático”, el cual muestra dos campos, coordenada X y coordenada Y, dentro de los cuales se deberá especificar la posición final a la que se desea llevar el tablero de la mesa de posicionamiento.

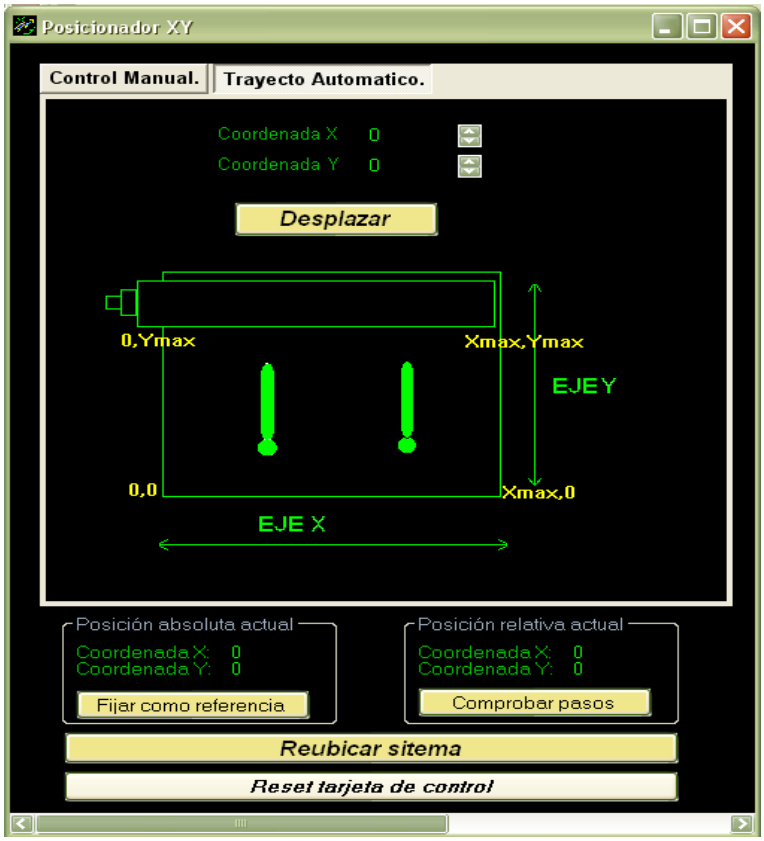


Figura 60: Pantalla principal de la aplicación: trayecto automático.

Con presionar el botón “desplazar” que también muestra dicho modo, se efectuará el desplazamiento desde la posición actual del tablero hasta la indicada, originando primero el desplazamiento correspondiente al eje X y después al eje Y.

Tanto en uno como en otro modo de funcionamiento se muestra un espacio donde se representan los valores de las coordenadas absolutas y relativas alcanzadas después de la última ejecución.

Además, se muestra un botón adicional “Reubicar sistema” presionando el cual, el tablero realiza los desplazamientos necesarios para ubicarlo en la posición inicial (0,0) de partida, y pondrá todos los marcadores de coordenadas a cero.

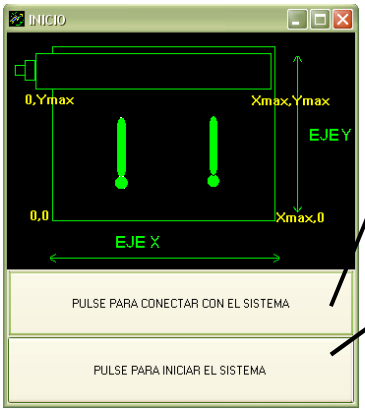
5.2.3 Desarrollo del programa.

Dado que se tarta de un lenguaje de programación orientado a objetos, el programa consta de una clase principal y desde ella se hacen llamadas a los diferentes miembros de la clase que se definen como funciones o subrutinas. En la figura se muestra el esquema general simplificado.

Public Class INICIO

5.2.3.1 Organigrama general del software del PC.

Private Sub INICIO_Load() 'Muestra la ventana de inicio.



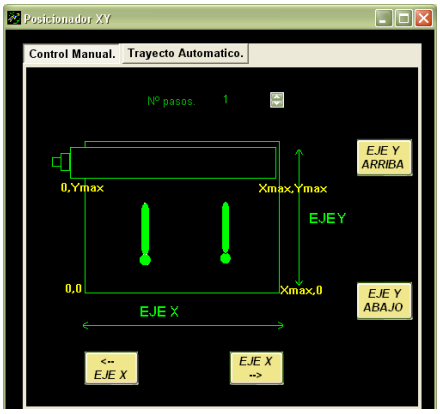
PosicionadorXY.Detecta_posicionador()

PosicionadorXY.iniciar()
- **Resetea_micro()**
- **Carga_pasos_reg_micro()**
- **Config_timer()**
- **Reubic_sist()**

PosicionadorXY.Show() 'Muestra la ventana de la clase PosicionadorXY.

Public Class PosicionadorXY

Private Sub PosicionadorXY_Load() → **Actualizar_coordenadas()**



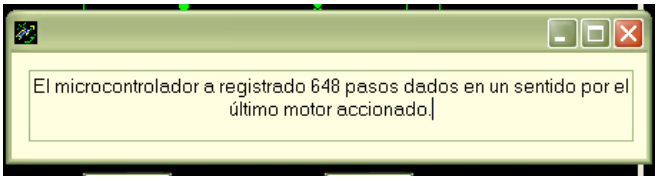
' Todos los botones que aparecen en estas dos figuras producen la misma secuencia de ejecución de subrutinas, variando sólo los parámetros de la ejecución.

- **Conversión_aBytes()**
- **Mueve_Motor _ ()** → **Enviar_al_micro()** → **Carga_dato()**
  - **Escribe_dato()** → **Inp()**, **Out()**
  - **Lee_dato()**
- **Ejecuta()**

Reubic_sist() → **Reubicar_eje_Y()**, **Reubicar_eje_X()**

Resetea_micro()
Fija_referencia()
Lee_reg_pasos()

Public Class INFO



' Ventana utilizada para mostrar mensajes durante la ejecución.

Como medio de entrada salida se hace uso del puerto paralelo del PC, de modo que se incluye una biblioteca de enlace dinámico (.dll) con la funcionalidad básica para manejar el puerto paralelo bajo Windows.

Al tratarse de un entorno visual interactivo, el cuerpo principal del programa se encargará de mantener actualizada la información de la ventana y quedará a la espera de forma continua de la interacción del usuario con los botones de la aplicación.

Podemos por tanto, hablar de grupos de subrutinas o funciones asociadas a cada una de las pantallas de modo de funcionamiento y otras generales que se usan de forma recurrente desde diferentes partes del programa principal.

Los grupos de funciones o subrutinas y las pantallas a las que están asociadas son las siguientes:

5.2.4 Grupo de rutinas asociadas a la pestaña "Control Manual."

5.2.4.1 Botones de desplazamiento.

La pulsación de cualquier botón realizará las mismas acciones pero referidas a un motor y sentido de movimiento diferente en cada caso, que son:

Mandar al microcontrolador el número de unidades que se quiere desplazar un motor.
Mandar el código del programa del microcontrolador que corresponde con el movimiento deseado.
Activar la señal INIT para que el microcontrolador inicie la ejecución.
Actualizar los campos de coordenadas.

5.2.5 Grupo de rutinas asociadas a la pestaña "Trayecto automático."

5.2.5.1 Botón "Desplazar."

La pulsación de dicho botón inicia unas cuentas para determinar la diferencia de coordenadas y, por tanto, el número de unidades que debe mandar avanzar a cada motor y en qué sentido, (en caso de que la posición actual sea diferente de la indicada).

Una vez realizado el cálculo, comenzará enviando el número de unidades a desplazar el motor del eje X.

Después envía el código del programa que activa dicho motor en el sentido adecuado, en función de si el resultado de las cuentas tiene signo negativo o positivo.

Por último activa la señal INIT para que el microcontrolador ejecute el programa y actualiza las coordenadas para presentarlas en pantalla.

5.2.6 Grupo de rutinas asociadas a los botones:

5.2.6.1 Botón "Reubicar sistema."

Este botón envía al microcontrolador un número de unidades mayor de las existentes en el recorrido total para cada eje, para garantizar que el motor llegará a final de recorrido y entrará en funcionamiento la rutina de atención de la interrupción de fin de recorrido del microcontrolador, que hará retroceder el motor un número de pasos conocido, quedando así el sistema en el punto (0,0) totalmente conocido.

Después manda los códigos de programa oportunos y los manda ejecutar, primero para el eje X y después el Y.

Pone a cero los campos de coordenadas.

5.2.6.2 Botón "Reset tarjeta de control."

Dicho botón ejecuta las subrutinas que permiten enviar la señal de reset al puerto paralelo durante un segundo, lo cual garantiza un reset de la tarjeta de control del sistema.

No se producirá ningún desplazamiento del sistema, y si se está efectuando alguno en dicho momento, será detenido.

5.2.6.3 Botón "Fijar como referencia."

Al presionar el botón, las coordenadas absolutas que haya en la pantalla pasarán a tomarse como referencia para presentar las coordenadas relativas.

5.2.6.4 Botón "Comprobar pasos."

Desencadena un proceso de comunicación, mostrado en la figura 46, a través del cual el PC obtiene la lectura de un registro de 24 bits con el número de pasos realmente dados por el último motor que se ha mandado mover.

6. Integración y resultados experimentales.

A continuación mostraremos con detalle los materiales, componentes, tolerancias y condiciones de contorno tenidas en cuenta a lo largo de la implementación del sistema de posicionamiento.

Como punto de partida se realiza el siguiente experimento para determinar cuál es la fuerza que debe ejercerse sobre cada rueda de accionamiento de la mesa de desplazamiento.

El experimento trata de colocar una masa M en cada rueda, colgando de ella, de modo que pueda caer libremente. Se irá aumentando la masa hasta que se comience a girar la rueda, de modo que podemos calcular el momento mecánico necesario.

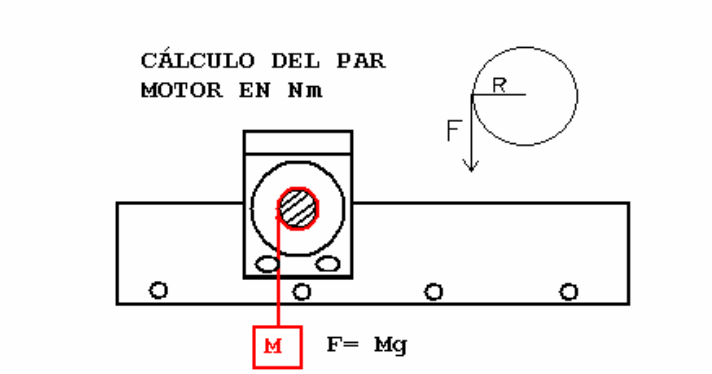


Figura 61: Experimento para el cálculo de la fuerza de rozamiento.

RUEDA EJE X		RUEDA EJE Y	
Masa (Kg)	Hay movimiento	Masa (Kg)	Hay movimiento
0,100	NO	0,480	NO
0,160	SI	0,660	SI
0,130	SI	0,520	SI

Figura 62: Resultados del experimento para el cálculo de la fuerza de rozamiento.

Observando los valores recogidos en la tabla, unas cuentas sencillas nos llevan a conocer que la fuerza en Newton que debe ser aplicada a 0,011 metros del eje central, que es el radio de la rueda mayor, es de (como mínimo)

(F=ma) F = 0,52 kg x 9.8 m/s = 5,096 N (Newton)

En el caso de la rueda pequeña sería de F = 0,13 Kg x 9,8 m/s = 1,274 N a una distancia del eje de 0,0095 metros.

Teniendo en cuenta que la fuerza que hay que aplicar a una distancia inferior al eje, para mantener el momento mecánico constante aumenta de forma directamente proporcional; a 1 cm del eje, es decir, a 0,01 metros la fuerza que habría que aplicar sería de:

Caso de rueda grande → F (0,01m) = (5,096 N x 0,011 m) / 0,01 = 5,6 N
o sea, Par = 5,6 Ncm

Caso de rueda pequeña → F (0,01m) = (1,274 N x 0,0095 m) / 0,01 = 1,21 N
o sea, Par = 1,21 Ncm

Con los resultados de dicha experiencia y dentro de las limitaciones comerciales se decide escoger un motor tipo paso a paso monopolar con reductora acoplada, sobredimensionado con un par motor de 100 Ncm para garantizar un buen funcionamiento a plena carga cuya ficha técnica se muestra a continuación:

BOBINADO	Monopolar
REDUCTORA	SI 125:1
PAR MÁX.	100 Ncm
I máx.	0,55 A
Nº FASES	4
ROTOR	Imán peranente
V alimentación	5 V
I fase.	0,55 A
R fase.	9,1 Ohm
L fase.	8,1 mH
Ángulo de paso	7,5º

Figura 63: Datos técnicos de los motores eléctricos empleados.

Una vez encontrado el motor se hace necesario diseñar un mecanismo de transmisión, de modo que se escoge implementar una transmisión por engranajes, dado que la transmisión por correa elástica implica diseñar un sistema de tensado y además puede haber deslizamientos entre la correa y la rueda de tracción, con la consecuente pérdida de precisión.

La problemática principal de dicho tipo de transmisión, para el sistema aquí expuesto, es que no resulta sencillo el acoplamiento de una rueda dentada comercial a cualquiera de las dos ruedas de accionamiento de la mesa.

Por dicho motivo se busca engranajes fabricados en un material plástico, que sea duro y a la vez fácilmente mecanizable a mano.

Finalmente las ruedas dentadas elegidas son las mostradas en la figura 64:

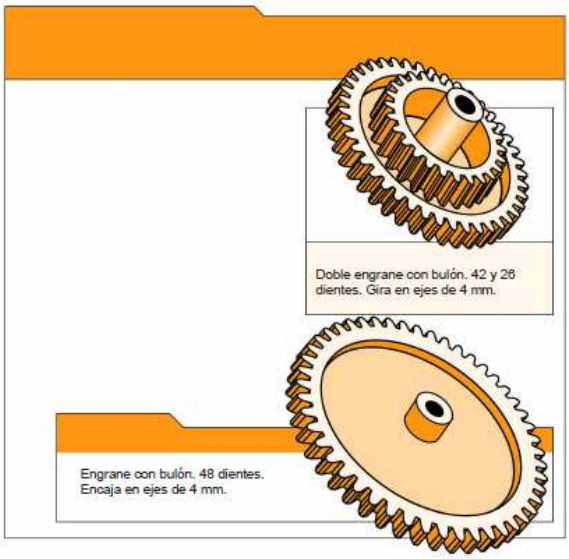


Figura 64: Datos de las ruedas dentadas utilizadas.

Fabricadas en un material plástico fácil de cortar con una segueta para madera.

Dos de ellas se acoplan directamente a los ejes de 4mm de salida de los motores, mientras que resulta necesario modificar las otras dos que deben girar solidarias a las ruedas de accionamiento de la mesa.

Para ello se toma una rueda de cada tipo y se realiza un vaciado del interior, de forma artesana con una segueta de madera, para que se acople a presión en cada rueda de la mesa, resultando como muestra la figura 65.

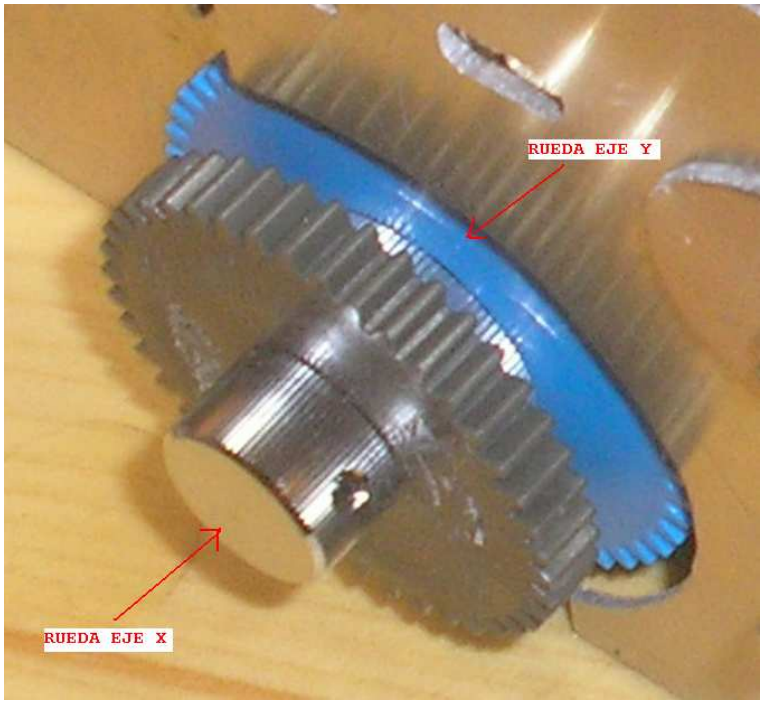


Figura 65: Vaciado de las ruedas dentadas.

Dado lo artesanal de la práctica, se realiza un diseño en Autocad con las dimensiones reales exactas (ver figura 66), el cual servirá de plantilla para superponerlo a la rueda en el momento de recortarla y así tener la menor excentricidad posible. El centro del círculo queda bien determinado por Autocad, de modo que sólo con hacer cuidadosamente coincidir la circunferencia externa con los dientes de la rueda se tiene la garantía de que el centro del esquema es el de la rueda.

Figura 66: Diseño de la zona de recorte de las ruedas dentadas.

Una vez se tienen dichas partes se comienza a trabajar con las plataformas metálicas que sujetarán a los motores y a su vez quedarán fijadas a la estructura de la mesa. Para ello se parte de una plancha de aluminio de 3 mm de espesor la cual se trabaja de forma artesanal para realizar el corte y perforaciones oportunas para obtener las piezas mostradas en el apartado 4.1

En paralelo a este trabajo se ha diseñado la electrónica que permite mover los motores. Para ello se ha dado un enfoque minimalista en componentes electrónicos en aras de obtener sencillez, robustez y eficacia en el diseño.

Como requisitos marcados por las decisiones tomadas en pasos anteriores, tenemos que el circuito debe ser capaz de suministrar una corriente de fase de 0,55 A con una tensión de fase de 5 V de CC.

Esto nos lleva a elegir como sistema de alimentación, un adaptador de red comercial que transforme los 220 V de A.C. y los rectifique para proporcionar una salida de C.C. de 7.5 V con una capacidad de entrega de corriente de hasta 1.2 A, de modo que con un regulador comercial tal como el 7805 será suficiente para soportar máximos de hasta 1 A de continua con una salida regulada a 5 V, tal y como se explicó en detalle en el apartado 4.2

Se montan dos reguladores como ya se explicó en el apartado 4.2 para separar la parte de gran consumo de corriente de la de menor demanda de corriente y evitar problemas de caídas de tensión en la alimentación de la parte computacional por demanda de corriente puntual de los motores sobre el regulador.

Con el resultado del diseño llevado a cabo, se manda revelar la placa de circuito impreso y se realizan las perforaciones para insertar los componentes con encapsulado de tipo DIL (Dual In Line) con lo que resulta la PCB que muestra la foto siguiente.

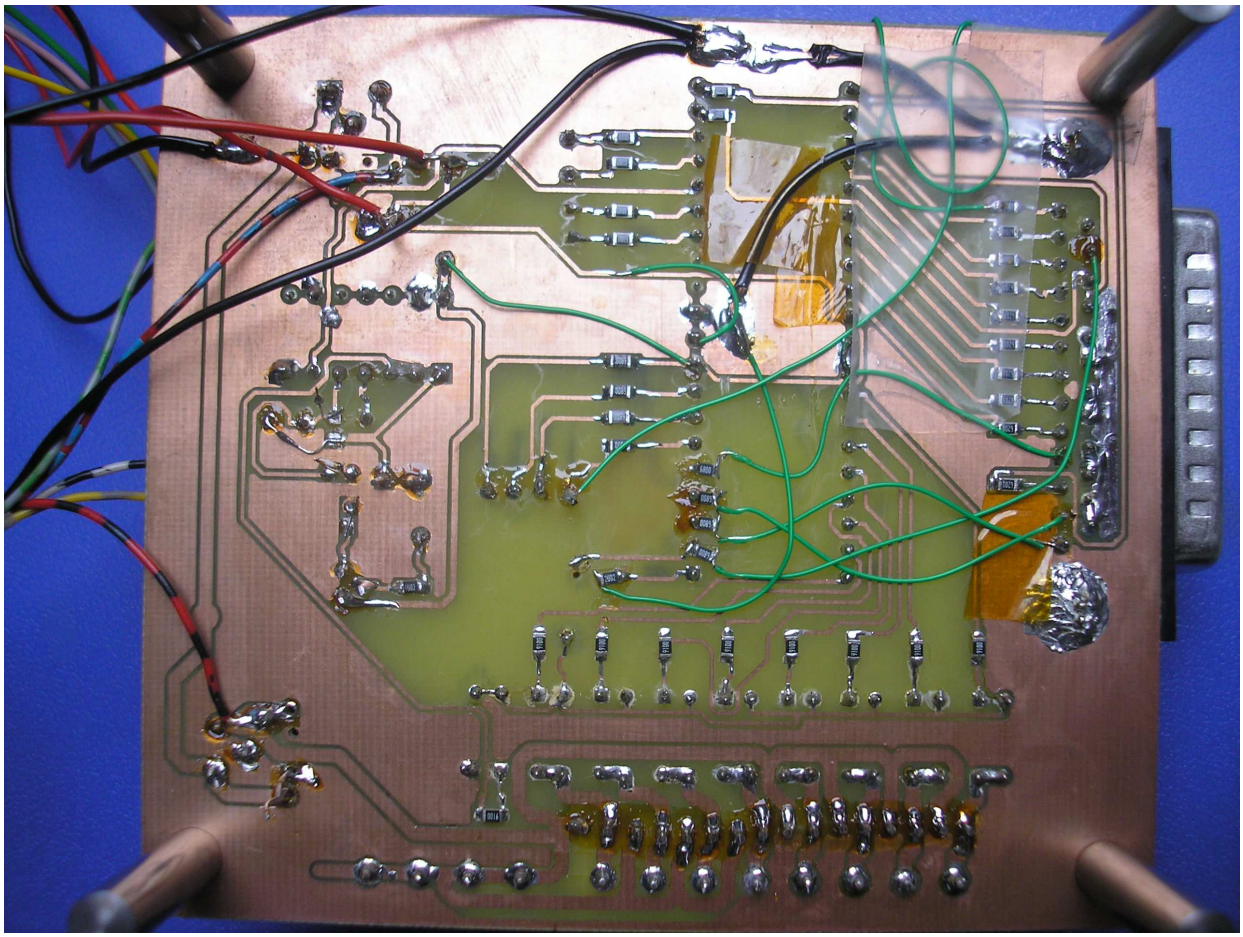


Figura 67: Placa de circuito impreso diseñada.

Se realiza el diseño en una PCB de una cara de cobre con abundancia de plano de masa para mejorar sus prestaciones, pero hay que cablear algunas conexiones que el diseño por ordenador no es capaz de rutar en una sola capa.

En las siguientes fotos se puede observar los componentes que se han utilizado y que también constan en los planos eléctricos.

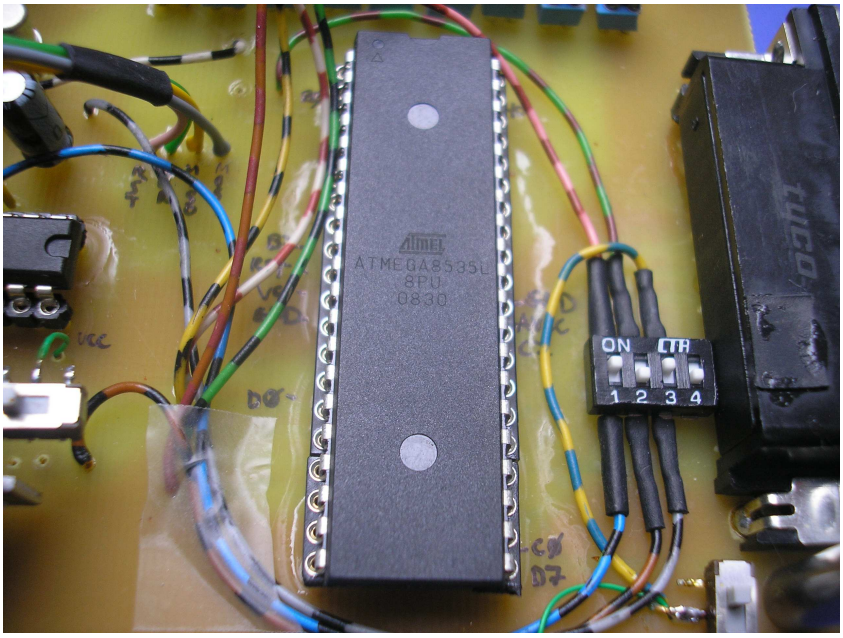


Figura 68:ATmega8535L, microcontrolador con arquitectura AVR RISC con bus de 8 bits.

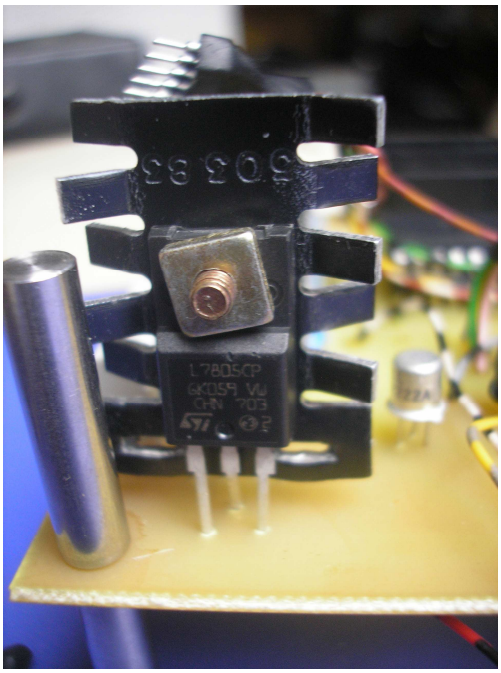


Figura 69: Detalle de los reguladores de tensión utilizados.

En la imagen se puede ver uno de los reguladores 7805 utilizados en el diseño, se ha montado un disipador de calor para alargar la vida del mismo (pieza atornillada de la foto).

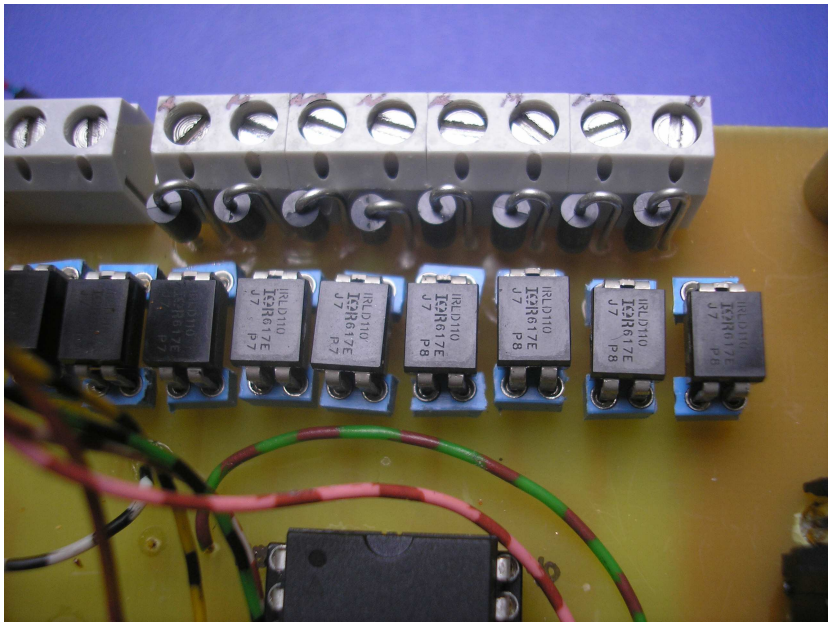


Figura 70: Transistores MOSFET utilizados como drivers de los motores.

Los transistores mosfet (M51 a M82, M52 al M82 y M0 del esquema eléctrico) de la foto son los encargados de conmutar las corrientes por los bobinados del motor.

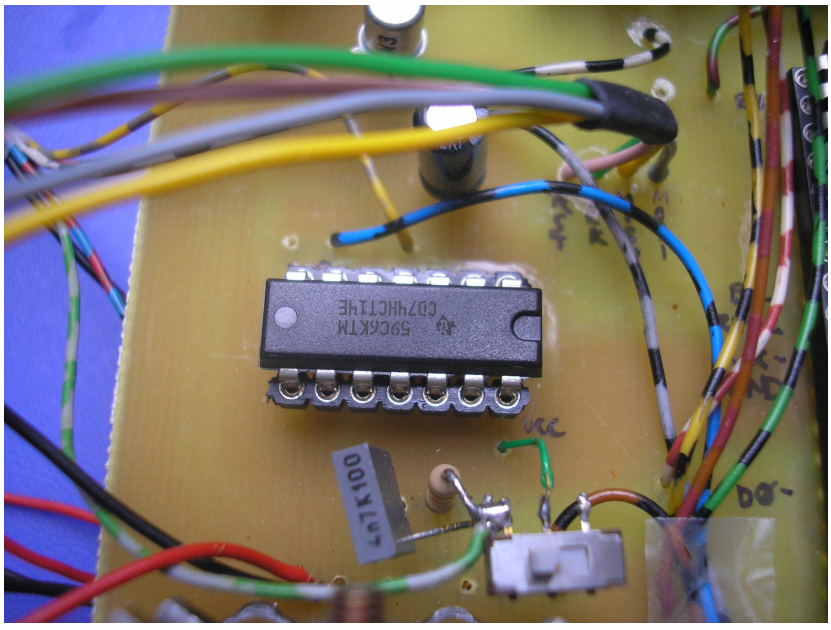


Figura 71: Detalle del circuito lógico utilizado en el sistema de seguridad.

En la imagen se ver el CD74HCT14, cuad de puertas lógicas inversoras, el cual posibilita la implementación de parte del circuito de seguridad (U1 en del esquema eléctrico).

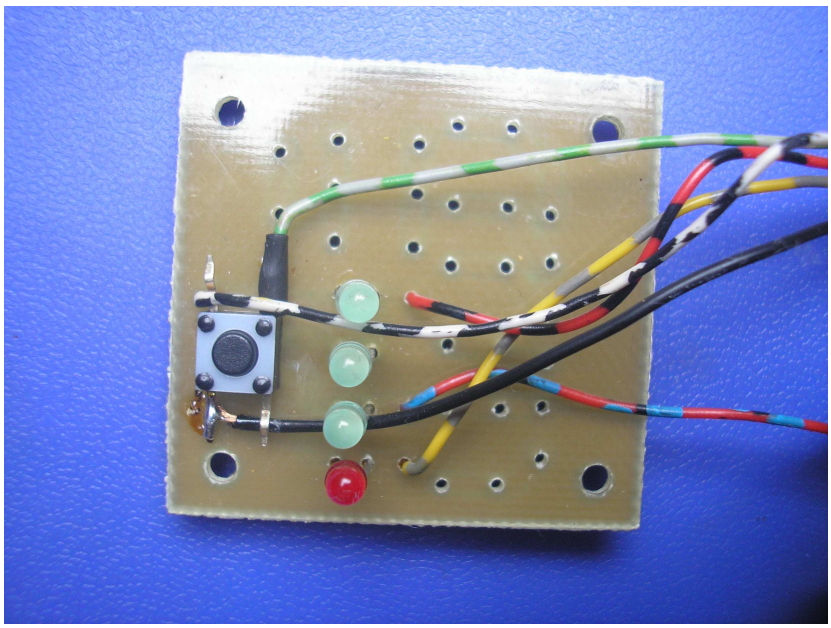


Figura 72: Detalle del circuito impreso de indicadores luminosos de la tapa.

Esta tarjeta es la encargada de informar al usuario del estado de funcionamiento del sistema mediante indicadores LED. Va alojada en la tapa de PVC de la tarjeta de control y además dispone de un interruptor pulsador para realizar reset manual de la tarjeta de control.

A continuación se muestran unas imágenes donde se puede observar cómo se ha realizado el acoplamiento mecánico.

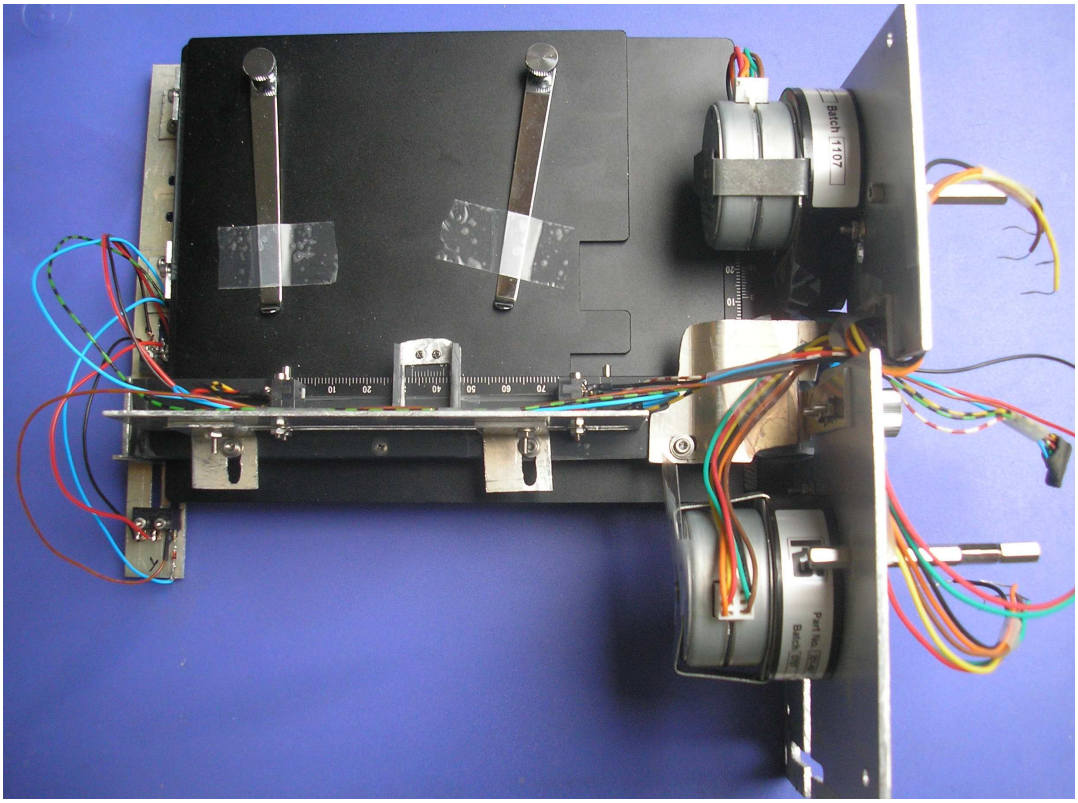


Figura 73: Foto de la mesa con todos los accesorios mecánicos montados.

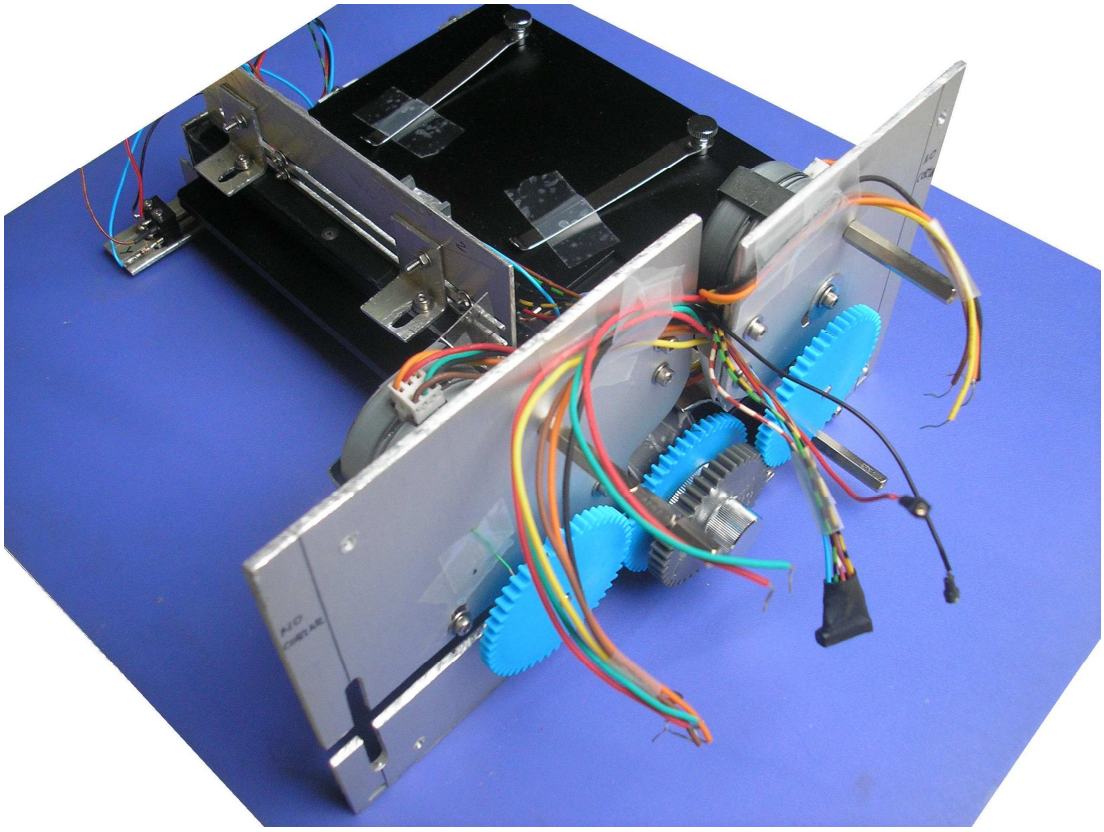


Figura 74: Foto de la mesa con todos los elementos mecánicos, detalle de engranajes.

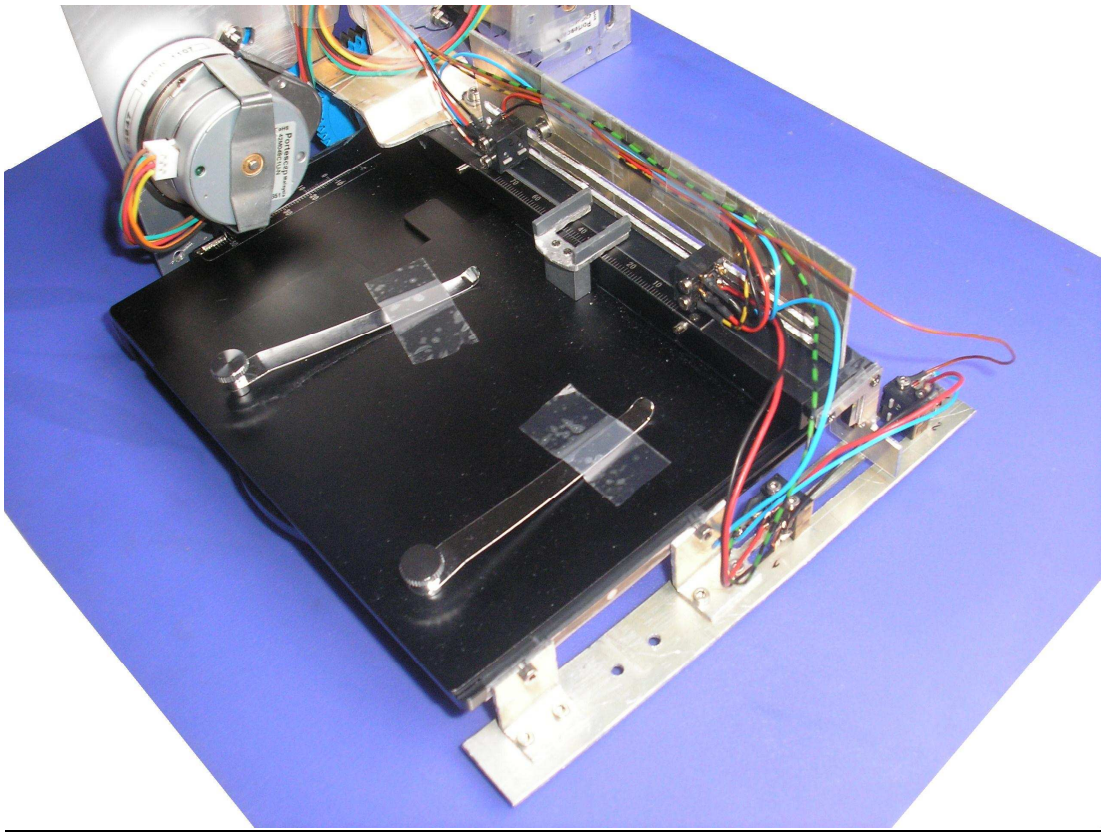


Figura 75: Detalle de las pletinas de sujeción de los interruptores final de recorrido.

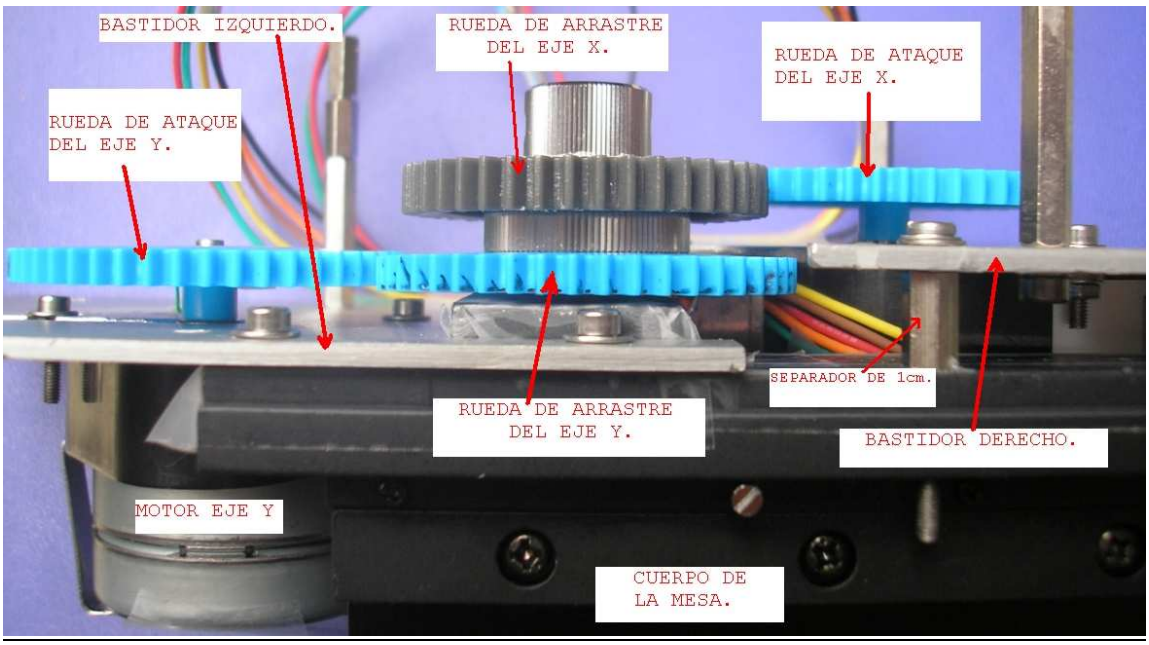


Figura 76: Vista inferior de la mesa de desplazamiento, detalle del acoplamiento mecánico.

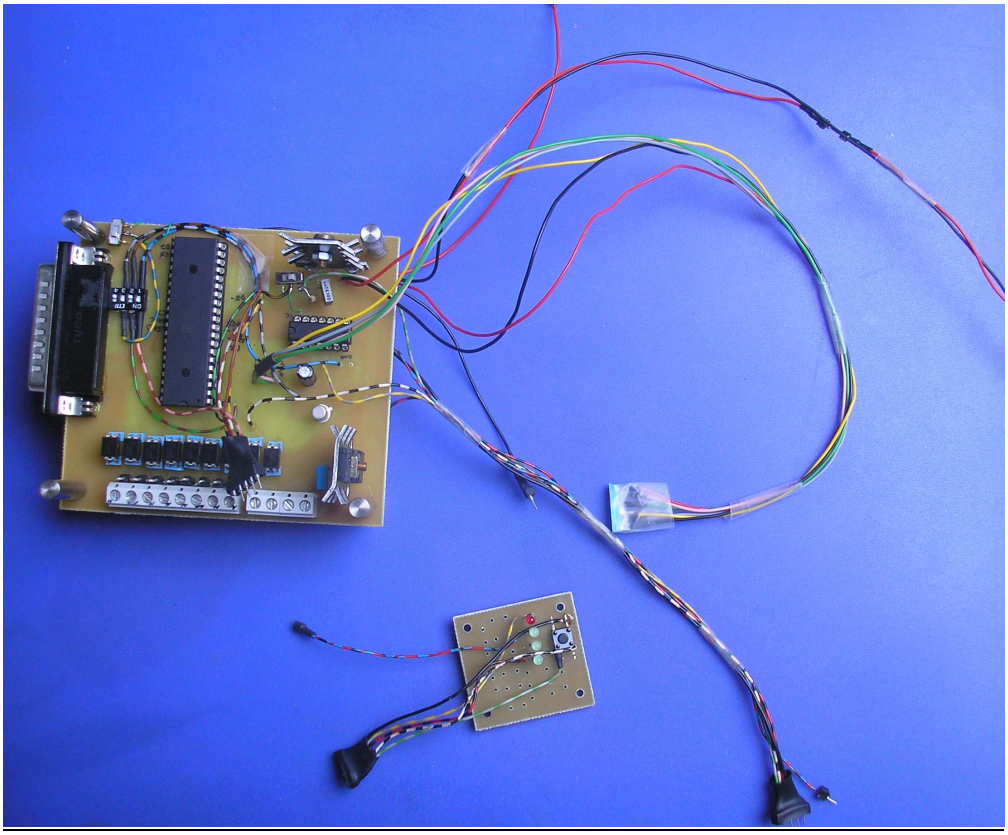


Figura 77: Placas de circuito impreso realizadas.

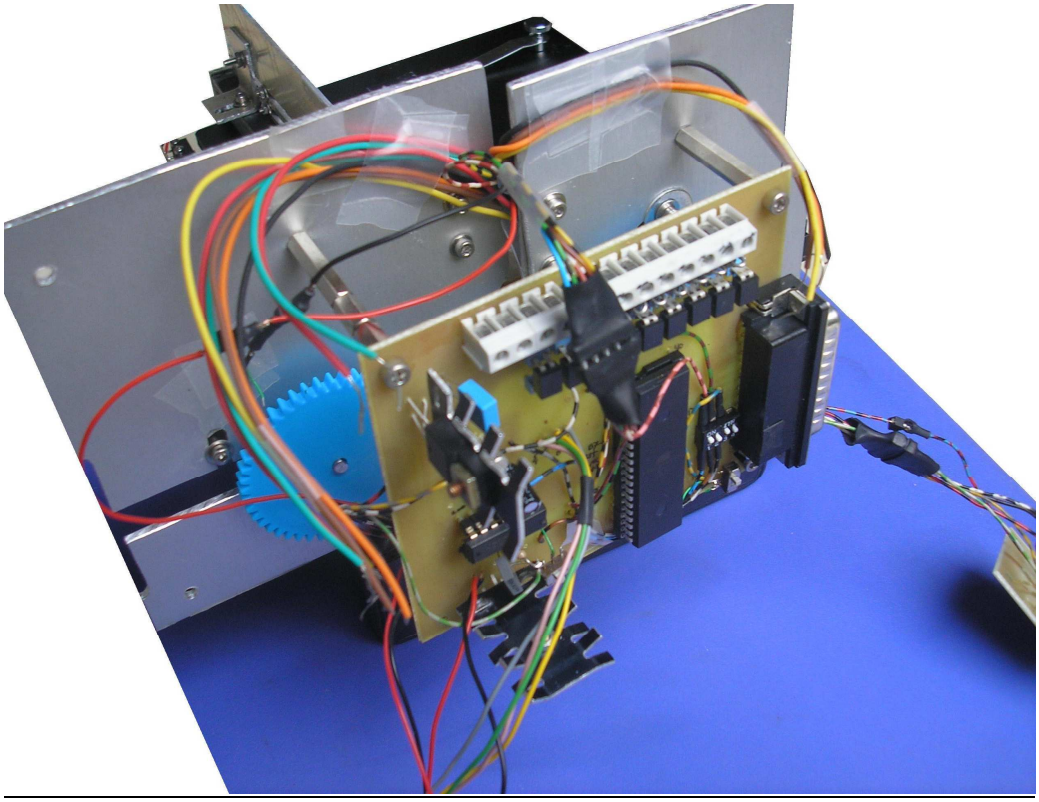


Figura 78: Ubicación de la tarjeta de control.

Para proteger la parte electromecánica del sistema, se ha diseñado una tapa que va atornillada a la estructura metálica, tal y como se puede apreciar en la figura 82. Además de proteger, sirve como alojamiento de la pequeña tarjeta de indicadores luminosos. Su diseño se ha realizado en un material plástico fácil de transformar mecánicamente, como es el PVC. En la figura 79 se muestra un esquema de los que se han realizado para su confección.

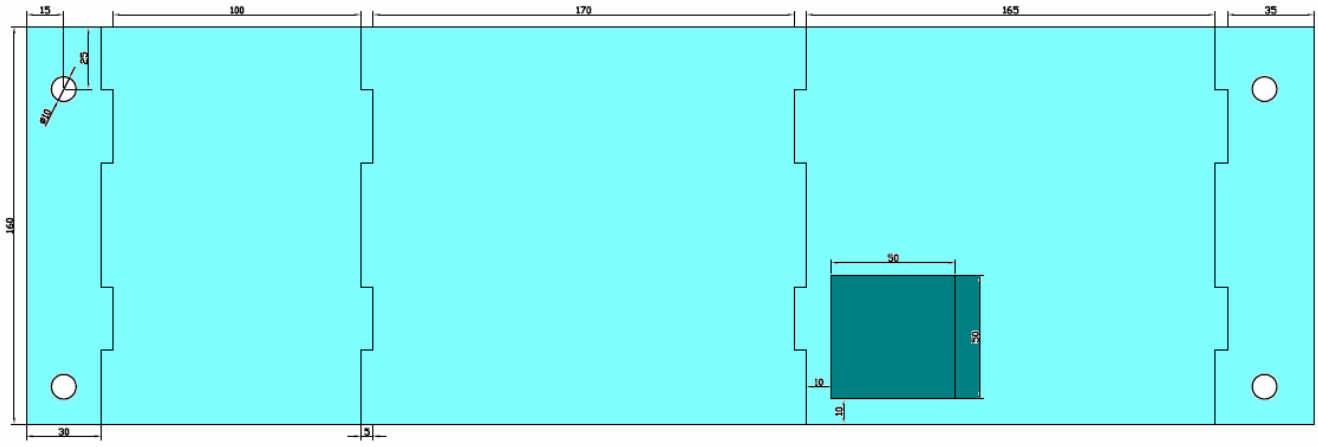


Figura 79: Esquema de la tapa y detalle de situación de la tarjeta de indicadores.

Figura 80: Realización de la tapa en PVC.

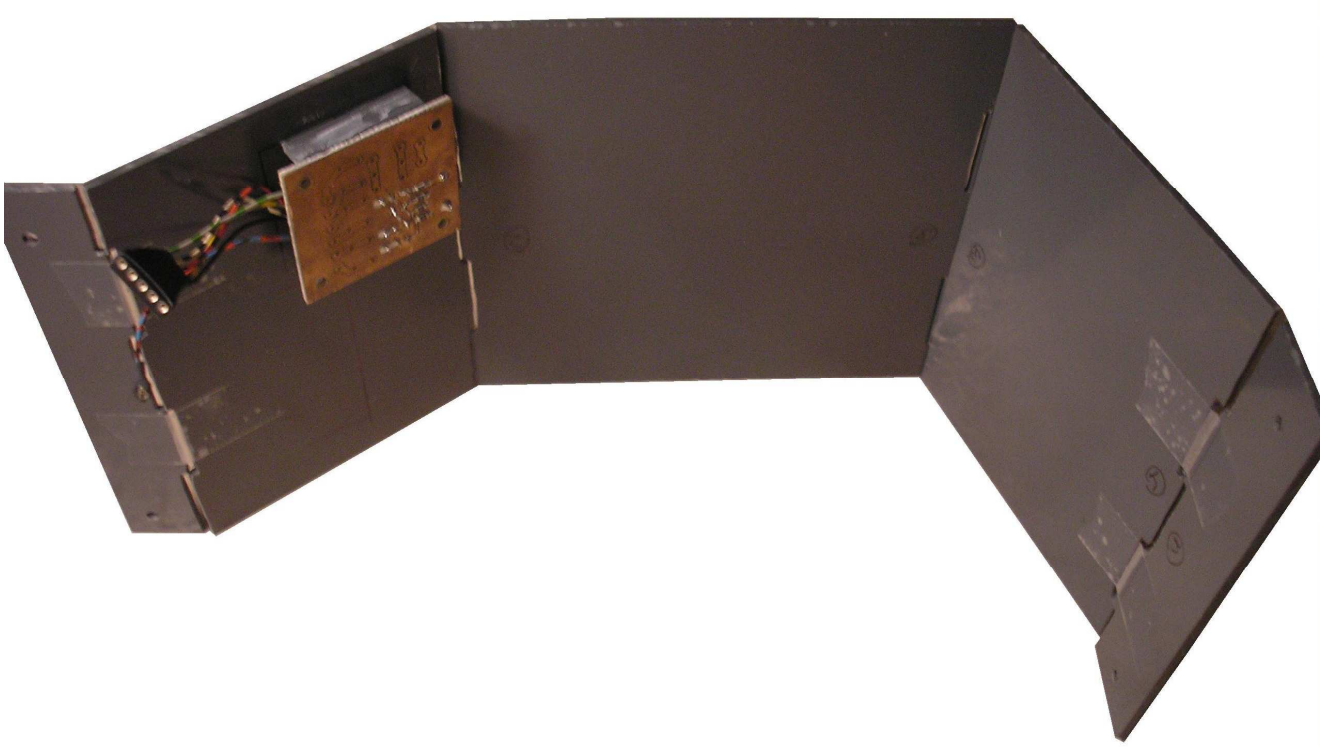


Figura 81: Detalle de las articulaciones de la tapa.

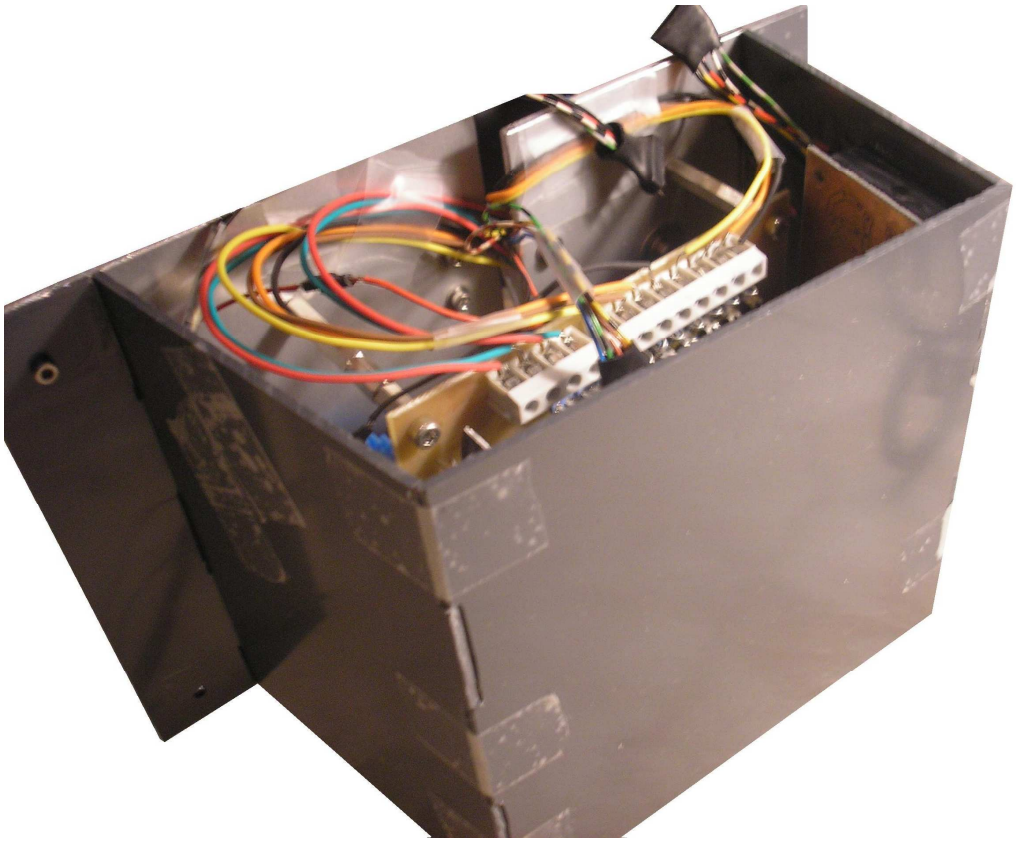


Figura 82: Colocación de la tapa, sin la cubierta superior.

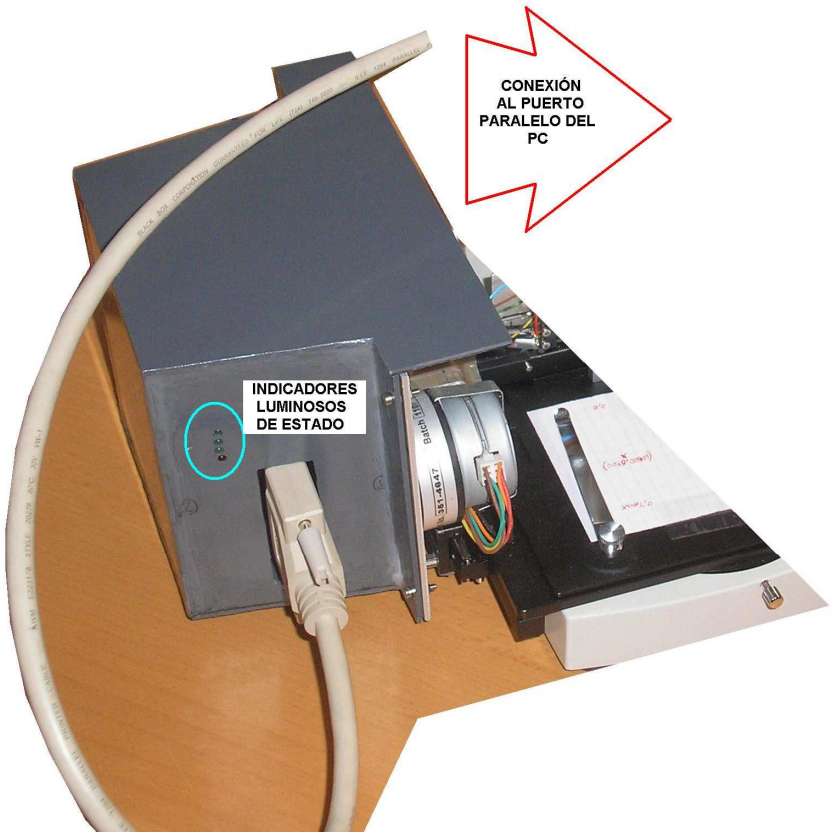


Figura 83: Detalle de la tapa montada y completamente terminada.

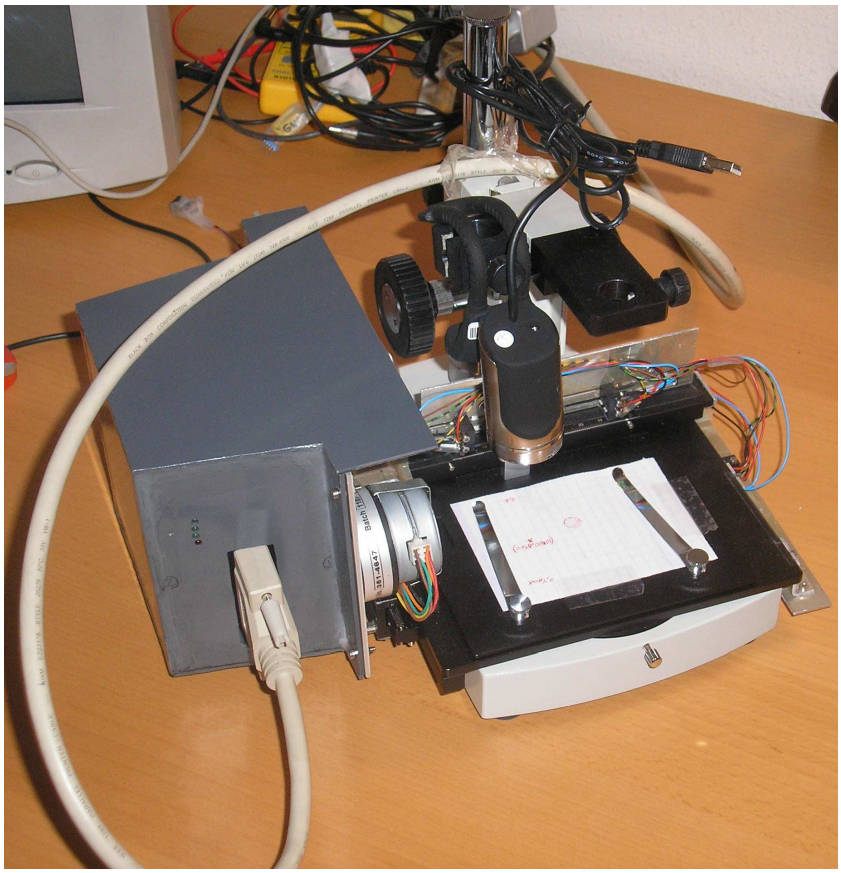


Figura 84: Sistema de medición completo sin la célula electroquímica montada.

6.1 Precisión y exactitud mecánica.

La experiencia que se ha tenido con el sistema completamente terminado nos lleva a los siguientes datos experimentales:

EJE X		
PASOS	61722	**metros x paso**
DISTANCIA (metros)	0,049	**7,93882E-07**
EJEY		
PASOS	8495	**metros x paso**
DISTANCIA (metros)	0,046	**5,41495E-06**

Figura 85: Tabla de resultados experimentales de la exactitud de desplazamiento conseguida.

Es decir, redondeando, unos 0,79 micrómetros de exactitud en el eje X y 5,41 micrómetros en el eje Y.

También se ha podido comprobar que existe una holgura mecánica en los engranajes, tal y como se esperaba. La rueda motriz comienza a moverse, pero no engrana con la otra hasta haber transcurrido 18 pasos para el eje X lo que se traduce en una pérdida de desplazamiento del tablero de la mesa de:

0,8 µm/paso **x** 18 pasos = 14,4 µm

Esto, cuando se encuentra en la posición más alejada de la situación de engrane con la otra rueda, lo que implica que el desplazamiento de la mesa será 14,4 µm menor de los esperados cada vez que se produzca un cambio de sentido de la marcha.

Para el caso del eje Y, los pasos medidos de holgura son 67, lo que da lugar a:

5,4 µm/paso **x** 67 pasos = 361,8 µm

Dado que el número de pasos que es necesario dar para que se produzca el engrane se ha medido de forma visual observando el sistema del mismo modo que se puede ver en la figura 86, sólo se puede garantizar que el error en el desplazamiento será igual al producido por 67 y 18 pasos respectivamente en el peor de los casos.

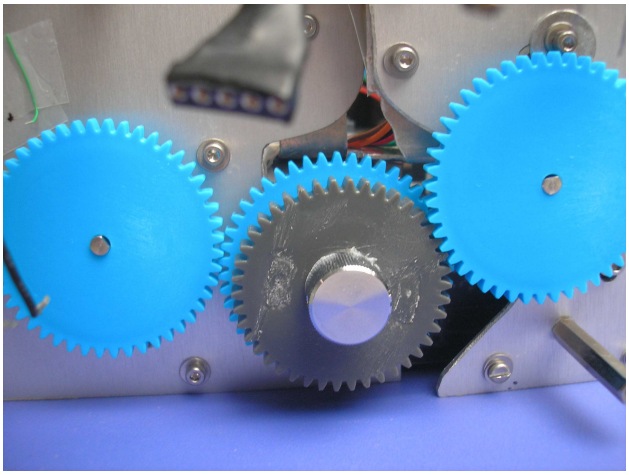


Figura 86: Detalle de la holgura mecánica de los engranajes.

Como sabemos que con la mini-célula se realizan medidas del potencial de corrosión en un área de 0,6 mm , si se suma 67 ó 18 pasos respectivamente a los movimientos que realice la mesa en cada eje cada vez que haya un cambio de sentido, estaremos incurriendo en una incertidumbre en la posición del orden de decenas de micrómetros, y no será percibida en el análisis de los datos obtenidos en el experimento. No obstante, aun queda cierta inseguridad en la posición, sin embargo existe una forma de proceder en el movimiento del tablero de la mesa de desplazamiento que nos garantiza la mayor precisión posible obtenida en este proyecto.

Primeramente, debemos recordar que el tablero se sitúa de forma automática en la posición inicial a la cual llega desplazándose hasta los interruptores de final de recorrido, con lo que se provoca que retroceda (acción llevada a cabo por la rutina de interrupción del microcontrolador mencionada en el apartado 5.1.4) 400 o 200 pasos en caso del eje X e Y respectivamente y de ese modo se fija el punto (0,0). Una vez estamos situados en el punto (0,0) cualquier desplazamiento a cualquier punto de coordenadas positivas no llevará implícito el error introducido por la holgura de los engranajes puesto que ya se han dado más de 67 y 18 pasos respectivamente en el sentido de la marcha, y a partir de dicho punto, cualquier trayectoria trazada por el tablero de la mesa podrá ser reproducida con la seguridad de que el único error en el que incurrimos es en la incertidumbre del punto (0,0).

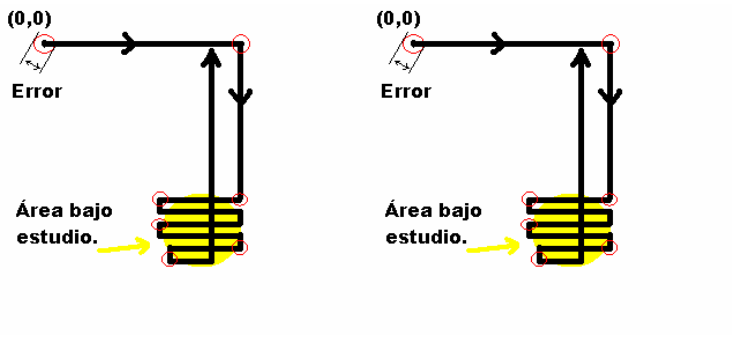


Figura 87: Ejemplo de trayectoria repetitiva de exploración de la superficie de la probeta metálica.

Para conocer cuál es dicha incertidumbre implícita en el posicionamiento del sistema en el punto (0,0) se ha llevado a cabo la siguiente experiencia:

Partiendo de la posición inicial del tablero (0,0) se ha mandado dar, en el caso del eje X, más de 400 pasos hacia el interruptor de final de recorrido, y en el caso del eje Y más de 200 pasos. Una vez alcanzado el interruptor, el microcontrolador da la orden de retroceder 200 o 400 pasos según el eje Y o X y queda de nuevo en el punto (0,0); en dicho momento se solicita al microcontrolador, a través del botón “Comprobar pasos” de la interfaz de usuario, la lectura del registro interno del microcontrolador donde se almacena el número de pasos reales que ha dado el último motor accionado y se toma nota.

Se han recogido 100 muestras (ver tabla de datos de figura 92) para cada eje y extremo de recorrido y se han obtenido los resultados mostrados en la figura 88 que muestran el número de sucesos que ha tenido cada número de pasos concreto.

distribuciones							
INTERRUPTOR EJE Y INICIO		INTERRUPTOR EJE X INICIO		INTERRUPTOR EJE Y FINAL		INTERRUPTOR EJE X FINAL	
nº pasos	nº sucesos	nº pasos	nº sucesos	nº pasos	nº sucesos	nº pasos	nº sucesos
189	0	389	0	189	0	389	0
190	0	390	0	190	0	390	0
191	0	391	0	191	0	391	0
192	0	392	0	192	0	392	0
193	1	393	0	193	1	393	0
194	0	394	0	194	0	394	0
195	0	395	0	195	0	395	0
196	0	396	0	196	0	396	0
197	0	397	0	197	1	397	0
198	12	398	15	198	2	398	3
199	73	399	70	199	94	399	86
200	**9**	**400**	**15**	**200**	**2**	**400**	**7**
201	4	401	0	201	0	401	2
202	0	402	0	202	0	402	0
203	1	403	0	203	0	403	0
204	0	404	0	204	0	404	1
205	0	405	0	205	0	405	0
206	0	406	0	206	0	406	0
207	0	407	0	207	0	407	0
208	0	408	0	208	0	408	0
209	0	409	0	209	0	409	1

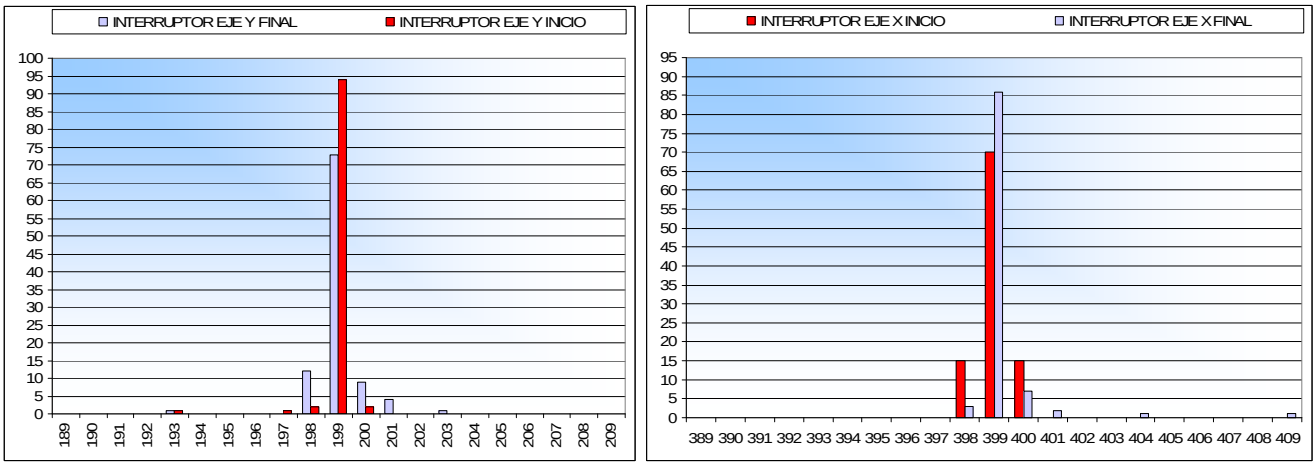


Figura 88: Tabla de datos e histograma del número de pasos medidos desde el punto (0,0) hasta los interruptores de ambos ejes.

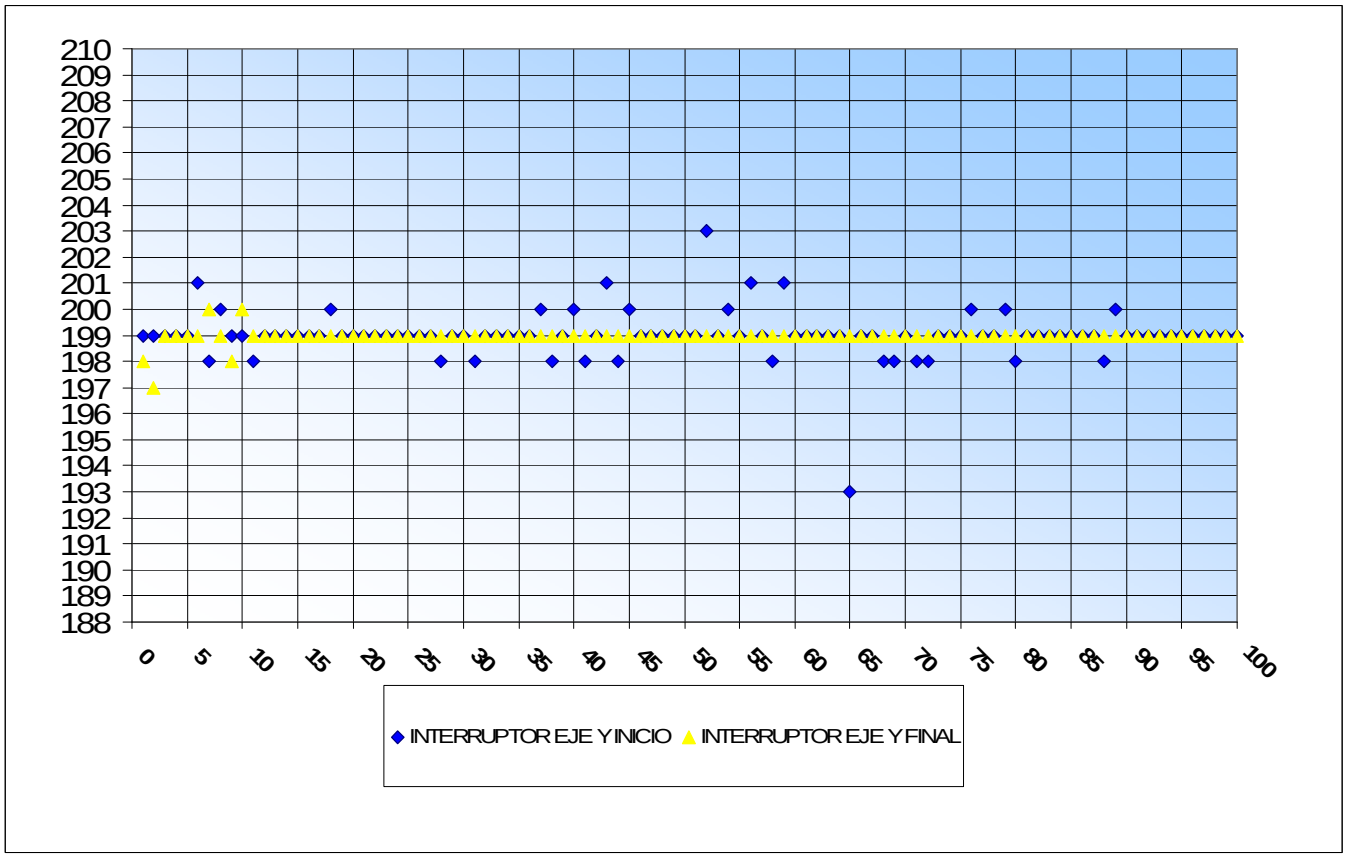


Figura 89: *Representación del espacio muestral obtenido para el eje X.*

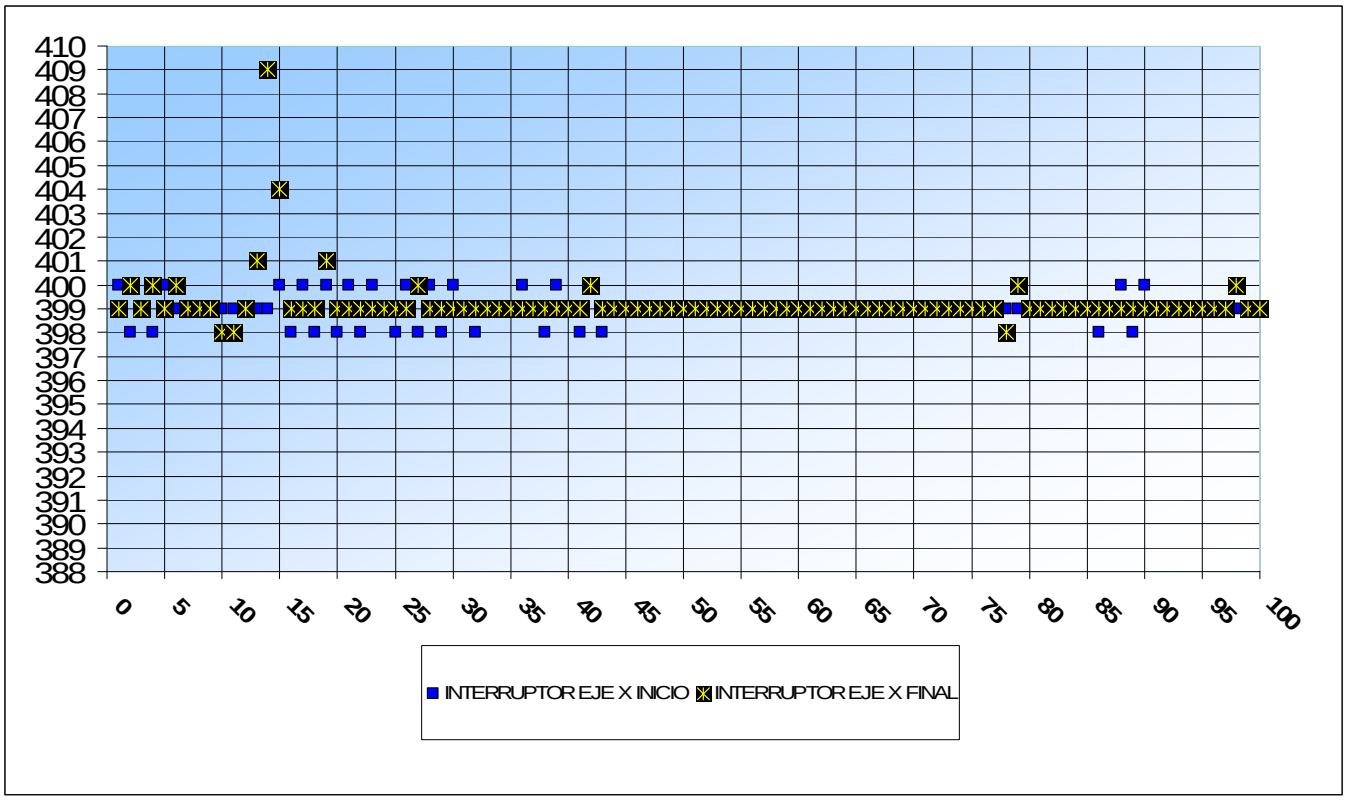


Figura 90: Representación del espacio muestral obtenido para el eje Y.

Las tablas con los datos de la experiencia son las mostradas en la figura 92.

A partir de estos resultados podemos realizar una estimación del error cometido al posicionar el sistema en el punto (0,0) mediante el uso de los interruptores de final de recorrido.

Para ello se ha utilizado las fórmulas de cálculo estadístico desviación típica y media, lo que se muestra en la figura 91.

VARIANZA	DESVIACIÓN TÍPICA
$\sigma^2 = \frac{\sum_{i=1}^{N} (X_i - \mu)^2}{N}$	$\sqrt{\sigma^2} = \sqrt{\frac{\sum_{i=1}^{N} (X_i - \mu)^2}{N}}$

	INTERRUPTOR EJE Y INICIO	INTERRUPTOR EJE X INICIO	INTERRUPTOR EJE Y FINAL	INTERRUPTOR EJE X FINAL
Nº PASOS MAS PROBABLE	199	399	199	399
N	100	100	100	100
VARIANZA	0,91	0,3	0,08	1,43
DESVIACIÓN TÍPICA	0,455	0,15	0,04	0,715

Figura 91: Cálculos estadísticos del error de posición en el punto (0,0).

Del análisis de los datos anteriores se puede inferir que se tiene un sistema con muy poca dispersión y muy poca distancia al número de pasos fijado como objetivo, por los valores tan bajos de desviación típica y varianza obtenidos. Igualmente podemos apreciar este hecho por análisis de la tabla de datos de la figura 88, donde se ha sombreado aquellos casos que se tendrán en cuenta por haber tenido al menos un suceso entre cien muestras, el resto de los casos, es decir con cero sucesos en cien iteraciones, se han despreciado. Analizando dicha tabla, vemos que el peor caso en cada eje, en lo que a error de medida del desplazamiento se refiere, que sería aquel en que hay mayor distancia al número de pasos fijado como objetivo, son los datos resaltados en rojo.

Para el eje Y, como sabemos que la distancia a la que corresponde un paso es de 5,4 µm, en el peor caso vemos que se incurre en un error de 5,4 µm/paso x 7 pasos = **37,8 µm** con un 1 % de probabilidad. El resto de casos son:

12% de probabilidad de incurrir en un error de 5,4 µm/paso x 2 pasos = 10,8 µm

73% de probabilidad de incurrir en un error de 5,4 µm/paso x 1 pasos = 5,4 µm

9% de probabilidad de incurrir en un error de 5,4 µm/paso x 0 pasos = 0 µm

12% de probabilidad de incurrir en un error de 5,4 µm/paso x 2 pasos = 10,8 µm

Para el eje X, como sabemos que la distancia a la que corresponde un paso es de 0,8 µm, en el peor caso vemos que se incurre en un error de 0,8 µm/paso x 9 pasos = **7,2 µm** con un 1 % de probabilidad. El resto de casos son:

3% de probabilidad de incurrir en un error de 0,8 µm/paso x 2 pasos = 1,6 µm

86% de probabilidad de incurrir en un error de 0,8 µm/paso x 1 pasos = 0,8 µm

7% de probabilidad de incurrir en un error de 0,8 µm/paso 0 pasos = 0 µm

Es importante darse cuenta de que en el experimento se han realizado cien iteraciones, lo que se correspondería a la realización de cien mapas de medida sobre una misma probeta en un caso de aplicación práctica, ya que el error que se está estimando y que es el único en el que se incurre cuando se trabaja con trayectorias con coordenadas idénticas como se ha explicado con la figura 87, es el de llevar el sistema a punto de referencia (0,0).

Esto quiere decir que las diferencias reales de posición entre los puntos de medida en dos mapas entre cien, serian del orden de **(**distancia deseada en eje X ± 7,2 µm **;** distancia deseada en eje Y ± 37,8 µm**)** con tan sólo un 1% de probabilidad, y este sería el mayor error cometido posible**.**

	fin ejeY 0	fin eje X 0	fin ejeY max	fin eje X max
iteración	nº pasos dados	nº pasos dados	nº pasos dados	nº pasos dados
1	199	400	198	399
2	199	398	197	400
3	199	399	199	399
4	199	398	199	400
5	199	400	199	399
6	201	399	199	400
7	198	399	200	399
8	200	399	199	399
9	199	399	198	399
10	199	399	200	398
11	198	399	199	398
12	199	399	199	399
13	199	399	199	401
14	199	399	199	409
15	199	400	199	404
16	199	398	199	399
17	199	400	199	399
18	200	398	199	399
19	199	400	199	401
20	199	398	199	399
21	199	400	199	399
22	199	398	199	399
23	199	400	199	399
24	199	399	199	399
25	199	398	199	399
26	199	400	199	399
27	199	398	199	400
28	198	400	199	399
29	199	398	199	399
30	199	400	199	399
31	198	399	199	399
32	199	398	199	399
33	199	399	199	399
34	199	399	199	399
35	199	399	199	399
36	199	400	199	399
37	200	399	199	399
38	198	398	199	399
39	199	400	199	399
40	200	399	199	399
41	198	398	199	399
42	199	400	199	400
43	201	398	199	399
44	198	399	199	399
45	200	399	199	399
46	199	399	199	399
47	199	399	199	399
48	199	399	199	399
49	199	399	199	399
50	199	399	199	399
51	199	399	199	399
52	203	399	199	399
53	199	399	199	399
54	200	399	199	399
55	199	399	199	399
56	201	399	199	399
57	199	399	199	399
58	198	399	199	399
59	201	399	199	399
60	199	399	199	399
61	199	399	199	399
62	199	399	199	399
63	199	399	199	399
64	199	399	199	399
65	193	399	199	399
66	199	399	199	399
67	199	399	199	399
68	198	399	199	399
69	198	399	199	399
70	199	399	199	399
71	198	399	199	399
72	198	399	199	399
73	199	399	199	399
74	199	399	199	399
75	199	399	199	399
76	200	399	199	399
77	199	399	199	399
78	199	399	199	398
79	200	399	199	400
80	198	399	199	399
81	199	399	199	399
82	199	399	199	399
83	199	399	199	399
84	199	399	199	399
85	199	399	199	399
86	199	398	199	399
87	199	399	199	399
88	198	400	199	399
89	200	398	199	399
90	199	400	199	399
91	199	399	199	399
92	199	399	199	399
93	199	399	199	399
94	199	399	199	399
95	199	399	199	399
96	199	399	199	399
97	199	399	199	399
98	199	399	199	400
99	199	399	199	399
100	199	399	199	399

Figura 92: Tabla de resultados para la estimación del error de posicionamiento del sistema en el punto (0,0).

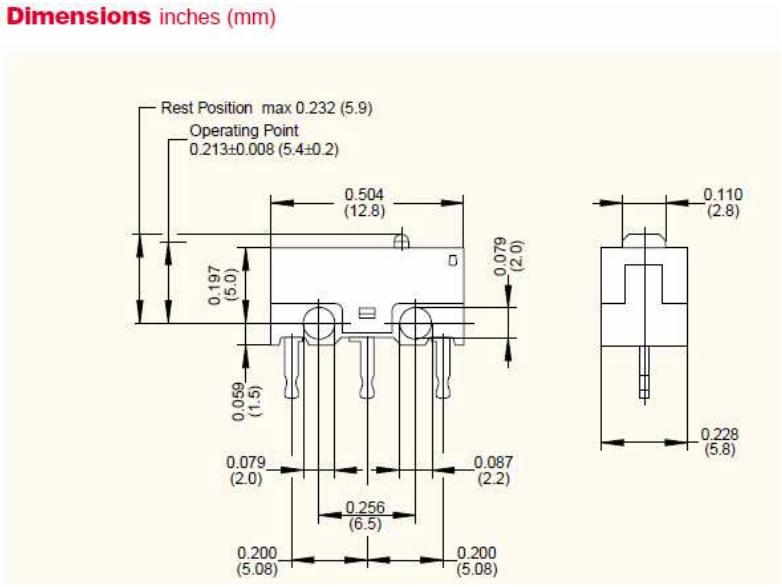


Figura 93: Detalle de dimensiones y tolerancias de los interruptores utilizados como final de recorrido.

7. Conclusión y propuestas de futuro.

El sistema de posicionamiento cumple perfectamente las expectativas del CENIM por lo que resultará de gran ayuda para los estudios de investigación de corrosión de metales con los recubrimientos objeto de estudio.

Se ha conseguido, por tanto, aumentar la exactitud del posicionamiento en prácticamente dos órdenes de magnitud de 100 µm a 5 µm haciendo posible además, reproducir dicho posicionamiento en otro momento del tiempo con un error de 37,8 µm ,en el peor caso con una probabilidad del 1%, a través de una interfaz de usuario sencilla y de fácil manejo.

Por el contrario, el tiempo que se debe invertir en llevar a cabo el posicionamiento ha aumentado dado el elevado número de pasos que deben dar los motores para cubrir 360 º de giro de las ruedas de accionamiento de la mesa.

No obstante, el sistema que se ha creado hace posible mejoras tales como:

Paquete software más completo que proporcione una herramienta de post-procesado de las medidas archivadas por los aparatos de medida del CENIM.

Las comunicaciones entre el PC y la tarjeta de control se pueden realizar utilizando el protocolo USB lo que permitiría mayor portabilidad del sistema.

Los sistemas de detección de la posición se pueden implementar con dispositivos ópticos, lo que aumentaría la exactitud, lo cual, por otro lado encarecería el producto final.

Como afinamiento del sistema de control se puede implementar un lazo de realimentación de la posición con un transductor de posición para tener una certeza mayor aun.

Como mejora del firmware del microcontrolador, se puede realizar modificaciones de las rutinas de movimiento de los motores para que fuera posible realizar desplazamientos de los dos motores al tiempo, permitiendo trazar trayectorias de todo tipo, no solo en ángulo recto.

El sistema hace posible pensar también en la implementación de un intérprete de comandos para poder programar secuencias de recorridos, en el caso de ser de utilidad tomar medidas de la corrosión en movimiento.

8. Presupuesto y estimación de horas invertidas.

Concepto		Coste (redondeado)	
Material mecánico		100	
Material electrónico		204	
			304
Tiempo invertido (horas)	528		
Coste hora	25 €		13200
			13504
		16 % IVA	2160,64
		TOTAL	**15664,64 €**

9. Bibliografía.

Referencias:

[1] Mueller WD, Ibendorf K (1994) Fresenius J Ana Chem 349:182

[2] Mueller WD, Manthey H, Lange KP, Gundlach HW, Plank T (1998) Fresenius J Anal Chem 361:662

[3] “Sensores químicos basados en recubrimientos híbridos para la protección de materiales metálicos y del patrimonio histórico”.MEC, Referencia: MAT- 2006-04486

[4] RENACO: “Reactive Nanoparticulate Coatings”/“Recubrimientos Reactivos Nanoparticulados”: Proyecto de Cooperación Transnacional del Programa MNT ERA-NET : MEC- Acción Estratégica de Nanociencia y Nanotecnología Referencia: NAN2006-27758-E/.

[5] http://support.microsoft.com/kb/22523/en-us/

[6] http://cfievalladolid2.net/tecno/cyr_01/control/puerto_paralelo.htm

[7] http://es.wikipedia.org/wiki/John_von_Neumann#Ciencia_Computacional

[8] http://www.ucontrol.com.ar/wiki/index.php/Fundamentos_de_la_Transmisi%C3%B3n_S%C3%ADncrona

[9] http://www.ldc.usb.ve/~adiserio/ci3815/clases/Laminas_EstructurasdeDatosObjetos2.pdf

Ilustraciones:

Figura 4: (Retocada)

http://www.electroservicioshama.com/imagenes/DSC00443.JPG.

Figura 5:

http://pr.kalipedia.com/kalipediamedia/ingenieria/media/200708/22/tecnologia/20070822klpingtcn_58.Ees.SCO.png

Figura 6:

http://2.bp.blogspot.com/_3Ld0ZNNa2Lc/SQJmQYnMplI/AAAAAAAAAXk/p2BeJBzEL7w/s400/principio+motor+cc.JPG

Figura 7: (Retocadas)

http://upload.wikimedia.org/wikipedia/commons/thumb/1/1a/Moteur_pas_%C3%A0_pas_MRV.png/250px-Moteur_pas_%C3%A0_pas_MRV.png

http://1.bp.blogspot.com/_TjhO5Ox-lqQ/SZuip2NUKpI/AAAAAAAAADk/ac-J0RTyfGw/s400/Motor+PAP.JPG

Provisión de materiales:

www.amidata.com
www.microlog.com

10. Anexos.

Anexo I.

Esquema eléctrico.

ANEXO II

Código fuente de la aplicación del PC para el control del prototipo.

```
Imports System.IO

Public Class DesplazaXY

    '*********************************************************
    '**                                                     **
    '**                 CÓDIGO RELATIVO A BIBLIOTECAS       **
    '**                                                     **
    '*********************************************************

    'Incluimos las funciones de lectura y escritura en puertos del PC para poder utilizarlas posteriormente.

    Public Declare Function Inp Lib "C:\inpout32.dll" Alias "Inp32" (ByVal PortAddress As Integer) As Integer
    Public Declare Sub Out Lib "C:\inpout32.dll" Alias "Out32" (ByVal PortAddress As Integer, ByVal Value As Integer)


    '*********************************************************
    '**                                                     **
    '**          FIN       CÓDIGO RELATIVO A BIBLIOTECAS    **
    '**                                                     **
    '*********************************************************
    '*********************************************************
    '**                                                     **
    '**                 CÓDIGO RELATIVO A VARIABLES         **
    '**                                                     **
    '*********************************************************

    ' DECLARACIÓN DE VARIABLES.
    'Arrays con las diferentes señales y secuencias posibles.
    Dim secuencia1_M1() As Integer = {1, 2, 4, 8} 'Secuencia para motor eje X.
    Dim secuencia1_M2() As Integer = {16, 32, 64, 128} 'Secuencia para motor eje Y.

    Dim indice_s1_M1 As Integer = 0 'Índices para recorrer los arrays anteriores.
    Dim indice_s1_M2 As Integer = 0

    Dim FIN_s1 As Integer = 3 'Determinan el último valor del índice permitido para recorrer cada secuencia.

    Dim buffer As Integer = 0 ' Buffer para el flujo de datos.
    Dim puerto_dinamico As Integer = 0 ' Donde se cargará la dirección del puerto al que se va a acceder
    '                           solo para el caso de estar en la parte de prueba de las comunicaciones.

    Dim puerto_fijo As Integer = 0 ' Aquí se debe introducir la dirección del puerto al que se va a acceder
    '                           para mover los motores, por defecto.

    Dim veces As Integer = 0 ' Un contador auxiliar para los bucles.
    Dim t As Integer = 0 ' Un contador auxiliar para los bucles.

    Dim CoorXini = 0 'Coordenadas de partida para el cálculo de trayectorias.
    Dim CoorYini = 0
    Dim CoorRefX = 0 'Coordenadas seleccionadas como pto de referencia, durante ejecución.
    Dim CoorRefY = 0
    Dim CoorXiniRel = 0 ' Coordenadas relativas calculadas a partir de las anteriores.
    Dim CoorYiniRel = 0

    Dim demora_t_ON As Integer 'Determinan el tiempo que permanecen las señales activas e inactivas.
    Dim demora_t_OFF As Integer
    Dim espera As Integer = 0
    '*********************************************************
    '**                                                     **
    '**        FIN            CÓDIGO RELATIVO A VARIABLES   **
    '**                                                     **
    '*********************************************************
```

```
'*****************************************************************************
'**                                                                         **
'**                CÓDIGO RELATIVO A LA PESTAÑA "COMUNICACIONES"            **
'**                                                                         **
'*****************************************************************************

' SUBRUTINA DE LECTURA DE DATOS DEL PUERTO.
Private Sub Lee_dato(ByVal port As Integer)
    buffer = Inp(port)
End Sub

' SUBRUTINA DE ESCRITURA DE DATOS EN EL PUERTO.
Private Sub Escribe_dato(ByVal port As Integer)
    Out (port, buffer)
End Sub

'LEE UN DATO DEL PUERTO ESPECIFICADO EN EL DIALOG Y LO PRESENTA
Private Sub Lectura_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Lectura.Click
    puerto_dinamico = EntradaPuerto.Text
    Lee_dato(puerto_dinamico)
    datos.Text = buffer
End Sub

'ESCRIBE EL DATO INTRODUCIDO EN EL TEXTBOX EN EL PUERTO ESPECIFICADO
Private Sub Envio_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Envio.Click
    puerto_dinamico = EntradaPuerto.Text
    buffer = datos.Text
    Escribe_dato(puerto_dinamico)
End Sub

' CADA VEZ QUE SE ESCRIBE UNA DIR DE PUERTO, SE ACTUALIZA EL VALOR DE "port"
Private Sub EntradaPuerto_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
EntradaPuerto.TextChanged
    puerto_dinamico = EntradaPuerto.Text
End Sub

' HACE UN RASTREO DESDE LA DIR INICIAL HASTA LA FINAL.
' PRIMERO INTENTA ESCRIBIR EL DATO INTRODUCIDO EN EL TEXTBOX
' Y LUEGO LEE DEL PUERTO PARA ESCRIBIRLO EN UN ARHCIVO DE TEXTO.

Private Sub Rastear_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Rastear.Click
    FileOpen(1, ofd.FileName, OpenMode.Output) 'Abre archivo de resultados.
    For veces = NumericUpDown1.Value To NumericUpDown2.Value 'comienza bucle.
        buffer = datos.Text        'carga el dato a escribir en el puerto.
        Escribe_dato(veces)            'escribe dato en el puerto.
        Lee_dato(veces)              'lee el dato del puerto.
        PrintLine(1, CStr(buffer) & " --> " & CStr(veces))       'lo escribe en el archivo.
    Next
    FileClose(1)                'cierro el archivo.
End Sub

'ABRE EL DIALOG QUE PERMITE SELECCIONAR EL ARCHIVO .TXT DE SALIDA PARA EL RASTREO DE PUERTOS.
Private Sub Label3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label3.Click
    ofd.ShowDialog()
End Sub
'*****************************************************************************
'**                                                                         **
'**        FIN        CÓDIGO RELATIVO A LA PESTAÑA "COMUNICACIONES"         **
'**                                                                         **
'*****************************************************************************
```

```
'*********************************************************************************************
'**                                                                                         **
'**                  CÓDIGO RELATIVO A LA PESTAÑA "MOVIMIENTO"                              **
'**                                                                                         **
'*********************************************************************************************
' LLAMA A LA SUB QUE MUEVE EL MOTOR DEL EJE X A DCHA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Mueve_MotorX_dcha(Pasos.Value)
End Sub

' LLAMA A LA SUB QUE MUEVE EL MOTOR DEL EJE X A IZQDA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button4.Click
    Mueve_MotorX_izqda(Pasos.Value)
End Sub

' LLAMA A LA SUB QUE MUEVE EL MOTOR DEL EJE Y A IZQDA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click
    Mueve_MotorY_izqda(Pasos.Value)
End Sub

' LLAMA A LA SUB QUE MUEVE EL MOTOR DEL EJE Y A DCHA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Mueve_MotorY_dcha(Pasos.Value)
End Sub

' SUB QUE MUEVE LOS MOTORES PARA DESPLAZAR LA MICROCÉLULA AL PUNTO DE COORDENADAS ESPECIFICADAS.
' PRIMERO SE CALCULA LA DISTANCIA EN CADA EJE Y LUEGO SE MUEVE, PRIMERO UNO Y DESPUES EL OTRO.
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button5.Click
    ' Variables donde se guarda el nº de pasos restantes para llegar al pto especificado.

    Dim pasosX = 0
    Dim pasosY = 0
    ' Actuamos de formas distintas en caso de que la coor sea mayor o menor que la actual.
    ' En caso de que sean iguales no se debe mover en ese eje y no se hace nada.
    If (COorX.Value > CoorXini) Then
        pasosX = COorX.Value - CoorXini 'Se calcula la diferencia.
        Mueve_MotorX_dcha(pasosX) ' Suponemos que hay que mover a dchas.
        CoorXini = COorX.Value ' Actualizamos el valor actual de la posición para el siguente
        '                  posible trayecto.
    End If
    If (COorX.Value < CoorXini) Then
        pasosX = CoorXini - COorX.Value
        Mueve_MotorX_izqda(pasosX) ' Suponemos que hay que mover a izqdas.
        CoorXini = COorX.Value ' Actualizamos el valor actual de la posición para el siguente
        '                  posible trayecto.
    End If
    If (COorY.Value > CoorYini) Then
        pasosY = COorY.Value - CoorYini 'Se calcula la diferencia.
        Mueve_MotorY_dcha(pasosY) ' Suponemos que hay que mover a dchas.
        CoorYini = COorY.Value ' Actualizamos el valor actual de la posición para el siguente
        '                  posible trayecto.
    End If
    If (COorY.Value < CoorYini) Then
        pasosY = CoorYini - COorY.Value
        Mueve_MotorY_izqda(pasosY) ' Suponemos que hay que mover a izqdas.
        CoorYini = COorY.Value ' Actualizamos el valor actual de la posición para el siguente
        '                  posible trayecto.
    End If
End Sub
'*********************************************************************************************
' **   FIN DEL                CÓDIGO RELATIVO A LA PESTAÑA "MOVIMIENTO"          **
'*********************************************************************************************
```

```
'*****************************************************************************
'**                                                                         **
'**                CÓDIGO RELATIVO A LA PESTAÑA "PRUEBAS"                   **
'**                                                                         **
'*****************************************************************************
'PERMITE GENERAR DE FORMA CONTINUA LAS SEÑALES DE MOVIMIENTO M X DCHA.
Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button7.Click
   While (CheckBox1.Checked) ' MIENTRAS SE ESPECIFIQUE...
      Mueve_MotorX_dcha(1) ' MUEVE UN PASO.
      CoorXini = CoorXini - 1 ' CONTRARESTA EL INCREMENTO DE POSICION PARA EVITAR
      '                  DESBORDAMIENTO.
   End While

End Sub
'PERMITE GENERAR DE FORMA CONTINUA LAS SEÑALES DE MOVIMIENTO M X IZQDA.
Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button8.Click
   While (CheckBox1.Checked) ' MIENTRAS SE ESPECIFIQUE...
      Mueve_MotorX_izqda(1) ' MUEVE UN PASO.
      CoorXini = CoorXini + 1 ' CONTRARESTA EL INCREMENTO DE POSICION PARA EVITAR
      '                  DESBORDAMIENTO.
   End While
End Sub

'PERMITE GENERAR DE FORMA CONTINUA LAS SEÑALES DE MOVIMIENTO M X DCHA.
Private Sub Button9_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button9.Click
   While (CheckBox1.Checked) ' MIENTRAS SE ESPECIFIQUE...
      Mueve_MotorY_dcha(1) ' MUEVE UN PASO.
      CoorYini = CoorYini - 1 ' CONTRARESTA EL INCREMENTO DE POSICION PARA EVITAR
      '                  DESBORDAMIENTO.
   End While

End Sub
'PERMITE GENERAR DE FORMA CONTINUA LAS SEÑALES DE MOVIMIENTO M X IZQDA.
Private Sub Button10_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button10.Click
   While (CheckBox1.Checked) ' MIENTRAS SE ESPECIFIQUE...
      Mueve_MotorY_izqda(1) ' MUEVE UN PASO.
      CoorXini = CoorXini + 1 ' CONTRARESTA EL INCREMENTO DE POSICION PARA EVITAR
      '                  DESBORDAMIENTO.
   End While
End Sub

' ESTA SUB ACTUALIZA LAS COORDENADAS DE POSICION ACTUAL EN LA PANTALLA.
Private Sub DesplazaXY_Paint(ByVal sender As Object, ByVal e As System.Windows.Forms.PaintEventArgs) Handles Me.Paint
   Label18.Text = CStr(CoorXini)
   Label17.Text = CStr(CoorYini)
   CoorXiniRel = CoorXini - CoorRefX
   CoorYiniRel = CoorYini - CoorRefY
   Label21.Text = CStr(CoorXiniRel)
   Label20.Text = CStr(CoorYiniRel)
End Sub

Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button6.Click
   CoorRefX = CoorXini
   CoorRefY = CoorYini
End Sub

'las cuatro siguentes SUB permiten que se hagan efectivos los cambios en los valores de las
' variables que controlan los tiempos Ton y Toff.
Private Sub TrackToff_Scroll(ByVal sender As Object, ByVal e As System.EventArgs) Handles TrackToff.Scroll
   demora_t_OFF = toffbase.Value * TrackToff.Value
```

```
    tablon.Text = CStr(demora_t_OFF)
End Sub

Private Sub TrackTon_Scroll(ByVal sender As Object, ByVal e As System.EventArgs) Handles TrackTon.Scroll
    demora_t_ON = tonbase.Value * TrackTon.Value
    tablon.Text = CStr(demora_t_ON)
End Sub

Private Sub tonbase_ValueChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles tonbase.ValueChanged
    demora_t_ON = tonbase.Value * TrackTon.Value
    tablon.Text = CStr(demora_t_ON)
End Sub

Private Sub toffbase_ValueChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles toffbase.ValueChanged
    demora_t_OFF = toffbase.Value * TrackToff.Value
    tablon.Text = CStr(demora_t_OFF)
End Sub

'*************************************************************************
'**                                                                     **
'**          FIN DEL        CÓDIGO RELATIVO A LA PESTAÑA "PRUEBAS"       **
'**                                                                     **
'*************************************************************************

'*************************************************************************************
'**                                                                                 **
'**                    SUBRUTINAS DE MOVIMIENTO DE LOS MOTORES.                     **
'**                                                                                 **
'*************************************************************************************

' SUB QUE MUEVE EL MOTOR DEL EJE X A DCHA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Mueve_MotorX_dcha(ByRef pasos As Integer)

    For t = 1 To pasos

        '*****************************************************************************************************
        If (indice_s1_M1 = 3) Then              'Permite que se mueva el índice de forma cíclica
            indice_s1_M1 = 0                    'para cada paso.
        Else                                    'Irá barriendo del array a derechas y al llegar al
            indice_s1_M1 = indice_s1_M1 + 1     'final comienza de nuevo en posición 0
        End If
        '*****************************************************************************************************
        '*****************************************************************************************************
        '
        buffer = secuencia1_M1(indice_s1_M1) 'Como el buffer siempre està a cero después de la
        '                                    anterior intervenciòn, cargamos en el buffer
        '                                    los 4 bits relativos al motor que nos interesa
        '                                    en cada momento.
        '*****************************************************************************************************

        'Escribe_dato(puerto_fijo) ' mandamos dicho dato al puerto.

        '*************************************************************
        'check point funcionamiento.
        tablon.Text = tablon.Text & buffer 'check point funcionamiento.
        'check point funcionamiento.
        '*************************************************************

        For espera = 0 To demora_t_ON 'Mantenemos el dato en el puerto el tiempo que deseemos.
        Next espera

        buffer = 0  'Cargamos todo a cero para desactivar las señales.
```

```
        '*************************************************************
        'check point funcionamiento.
        tablon.Text = tablon.Text & buffer 'check point funcionamiento.
        'check point funcionamiento.
        '*************************************************************

        'Escribe_dato()  'Mandamos de nuevo al puerto para poner a cero y volver a reposo.
        'TextBox1.Text = TextBox1.Text & "Nbajo" 'Escribimos aquí lo que mandaría al puerto para comprobar.
        'TextBox1.Text = TextBox1.Text & CStr(buffer) 'Escribimos aquí lo que mandaría al puerto para comprobar.

        For espera = 0 To demora_t_OFF 'Mantenemos el puerto en reposo el tiempo que deseemos.
        Next espera
        CoorXini = CoorXini + 1
    Next t
End Sub

' SUB QUE MUEVE EL MOTOR DEL EJE X A IZQDA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Mueve_MotorX_izqda(ByRef pasos As Integer)

    For t = 1 To pasos

        '*****************************************************************************************************
        If (indice_s1_M1 = 0) Then              'Permite que se mueva el índice de forma cíclica
            indice_s1_M1 = 3                    'para cada paso.
        Else                                'Irá barriendo del array a derechas y al llegar al
            indice_s1_M1 = indice_s1_M1 - 1     'final comienza de nuevo en posición 0
        End If
        '*****************************************************************************************************
        '*****************************************************************************************************
        '
        buffer = secuencia1_M1(indice_s1_M1) 'Como el buffer siempre está a cero después de la
        '                           anterior intervenciòn, cargamos en el buffer
        '                           los 4 bits relativos al motor que nos interesa
        '                           en cada momento.
        '*****************************************************************************************************

        'Escribe_dato(AQUÍ HAY QUE PONER EL PUERTO 0 <<puerto_fijo>>) ' mandamos dicho dato al puerto.
        'TextBox1.Text = TextBox1.Text & "Nalto" 'Escribimos aquí lo que mandaría al puerto para comprobar.
        'TextBox1.Text = TextBox1.Text & CStr(buffer) 'Escribimos aquí lo que mandaría al puerto para comprobar.

        For espera = 0 To demora_t_ON 'Mantenemos el dato en el puerto el tiempo que deseemos.
        Next espera

        buffer = 0  'Cargamos todo a cero para desactivar las señales.

        'Escribe_dato()  'Mandamos de nuevo al puerto para poner a cero y volver a reposo.
        'TextBox1.Text = TextBox1.Text & "Nbajo" 'Escribimos aquí lo que mandaría al puerto para comprobar.
        'TextBox1.Text = TextBox1.Text & CStr(buffer) 'Escribimos aquí lo que mandaría al puerto para comprobar.

        For espera = 0 To demora_t_OFF 'Mantenemos el puerto en reposo el tiempo que deseemos.
        Next espera
        CoorXini = CoorXini - 1
    Next t
End Sub

' SUB QUE MUEVE EL MOTOR DEL EJE Y A DCHA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Mueve_MotorY_dcha(ByRef pasos As Integer)

    For t = 1 To pasos

        '*****************************************************************************************************
```

```
        If (indice_s1_M1 = 3) Then              'Permite que se mueva el índice de forma cíclica
           indice_s1_M1 = 0                  'para cada paso.
        Else                              'Irá barriendo del array a derechas y al llegar al
           indice_s1_M1 = indice_s1_M1 + 1      'final comienza de nuevo en posición 0
        End If
        '*************************************************************************************************
        '*************************************************************************************************
        '
        buffer = secuencia1_M1(indice_s1_M1) 'Como el buffer siempre está a cero después de la
        '                           anterior intervenciòn, cargamos en el buffer
        '                           los 4 bits relativos al motor que nos interesa
        '                           en cada momento.
        '*************************************************************************************************

        'Escribe_dato(AQUÍ HAY QUE PONER EL PUERTO 0 <<puerto_fijo>>) ' mandamos dicho dato al puerto.
        'TextBox1.Text = TextBox1.Text & "Nalto" 'Escribimos aquí lo que mandaría al puerto para comprobar.
        'TextBox1.Text = TextBox1.Text & CStr(buffer) 'Escribimos aquí lo que mandaría al puerto para comprobar.

        For espera = 0 To demora_t_ON 'Mantenemos el dato en el puerto el tiempo que deseemos.
        Next espera

        buffer = 0  'Cargamos todo a cero para desactivar las señales.

        'Escribe_dato()  'Mandamos de nuevo al puerto para poner a cero y volver a reposo.
        'TextBox1.Text = TextBox1.Text & "Nbajo" 'Escribimos aquí lo que mandaría al puerto para comprobar.
        'TextBox1.Text = TextBox1.Text & CStr(buffer) 'Escribimos aquí lo que mandaría al puerto para comprobar.

        For espera = 0 To demora_t_OFF 'Mantenemos el puerto en reposo el tiempo que deseemos.
        Next espera
        CoorYini = CoorYini + 1
    Next t
End Sub

' SUB QUE MUEVE EL MOTOR DEL EJE Y A IZQDA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Mueve_MotorY_izqda(ByRef pasos As Integer)

    For t = 1 To pasos

        '*************************************************************************************************
        If (indice_s1_M1 = 0) Then              'Permite que se mueva el índice de forma cíclica
           indice_s1_M1 = 3                  'para cada paso.
        Else                              'Irá barriendo del array a derechas y al llegar al
           indice_s1_M1 = indice_s1_M1 - 1      'final comienza de nuevo en posición 0
        End If
        '*************************************************************************************************
        '*************************************************************************************************
        '
        buffer = secuencia1_M1(indice_s1_M1) 'Como el buffer siempre está a cero después de la
        '                           anterior intervenciòn, cargamos en el buffer
        '                           los 4 bits relativos al motor que nos interesa
        '                           en cada momento.
        '*************************************************************************************************

        'Escribe_dato(AQUÍ HAY QUE PONER EL PUERTO 0 <<puerto_fijo>>) ' mandamos dicho dato al puerto.
        'TextBox1.Text = TextBox1.Text & "Nalto" 'Escribimos aquí lo que mandaría al puerto para comprobar.
        'TextBox1.Text = TextBox1.Text & CStr(buffer) 'Escribimos aquí lo que mandaría al puerto para comprobar.

        For espera = 0 To demora_t_ON 'Mantenemos el dato en el puerto el tiempo que deseemos.
        Next espera

        buffer = 0  'Cargamos todo a cero para desactivar las señales.
```

```
        'Escribe_dato()  'Mandamos de nuevo al puerto para poner a cero y volver a reposo.
        'TextBox1.Text = TextBox1.Text & "Nbajo" 'Escribimos aquí lo que mandaría al puerto para comprobar.
        'TextBox1.Text = TextBox1.Text & CStr(buffer) 'Escribimos aquí lo que mandaría al puerto para comprobar.

        For espera = 0 To demora_t_OFF 'Mantenemos el puerto en reposo el tiempo que deseemos.
        Next espera
        CoorYini = CoorYini - 1
    Next t
End Sub

'*************************************************************************************************
'**                                                     **
'**          FIN     SUBRUTINAS DE MOVIMIENTO DE LOS MOTORES.                  **
'**                                                     **
'*************************************************************************************************


End Class
```

Código fuente de la aplicación del PC definitiva.

```
Imports System.IO

Public Class PosicionadorXY


    '*************************************************************************************************
    '**                                                    **
    '**                CÓDIGO RELATIVO A BIBLIOTECAS                        **
    '**                                                    **
    '*************************************************************************************************

    'Incluimos las funciones de lectura y escritura en puertos del PC para poder utilizarlas posteriormente.
    Public Declare Function Inp Lib "C:\inpout32.dll" Alias "Inp32" (ByVal PortAddress As Integer) As Integer
    Public Declare Sub Out Lib "C:\inpout32.dll" Alias "Out32" (ByVal PortAddress As Integer, ByVal Value As Integer)


    '*************************************************************************************************
    '**                                                    **
    '**          FIN       CÓDIGO RELATIVO A BIBLIOTECAS                        **
    '**                                                    **
    '*************************************************************************************************
    '*************************************************************************************************
    '**                                                    **
    '**                CÓDIGO RELATIVO A VARIABLES                        **
    '**                                                    **
    '*************************************************************************************************

    ' DECLARACIÓN DE VARIABLES.
    Dim continua As Boolean = False

    Public buffer_datos As Integer = 0 ' buffer_datos para el flujo de datos.

    Public puerto_datos As Integer = 888 ' Dirección base del puerto al que se va a acceder por defecto.
    Dim puerto_estado As Integer = puerto_datos + 1
    Public puerto_control As Integer = puerto_datos + 2

    Dim veces As Integer = 0 ' Un contador auxiliar para los bucles.
    Dim veces1 As Integer = 0 'Otro contador auxiliar para los bucles.

    Dim CoorXabs = 0 'Coordenadas de partida para el cálculo de trayectorias.
    Dim CoorYabs = 0
    Dim CoorRefX = 0 'Coordenadas seleccionadas como pto de referencia, durante ejecución.
    Dim CoorRefY = 0
    Dim CoorX_Rel = 0 ' Coordenadas relativas calculadas a partir de las anteriores.
    Dim CoorY_Rel = 0

    Dim PROGRAMA_Mueve_MotorX_dcha As Integer = 1
    Dim PROGRAMA_Mueve_MotorX_izqda As Integer = 2
    Dim PROGRAMA_Mueve_MotorY_dcha As Integer = 3
    Dim PROGRAMA_Mueve_MotorY_izqda As Integer = 4
    Dim PROGRAMA_ConfigurarTimer As Integer = 5 '(velocidad y aceleración)
    Dim MSByte As Integer = 0
    Dim LSByte As Integer = 0

    'Tabla de códigos de actuación entre micro y pc.
    'CÓDIGOS PARA EL PUERTO DE ESTADO 889

    Dim busy_CODE As Integer = 7 'el micro indica al pc que está ocupado.


    'CÓDIGOS PARA EL PUERTO DE CONTROL 890  PARA UN PUERTO SPP
    Dim INIT_CODE As Integer = 207 ' el pc indica al micro que puede leer el dato presentado en el bus.
```

```
Dim SELECT_CODE As Integer = 195 ' el pc indica al micro que debe comenzar la ejecución.
Public pcReposo_CODE As Integer = 203 ' código de reposo para las señales del PC.
Dim reset_micro_CODE As Integer = 202 ' hace reset al micro.


'variables usadas en las sub y funciones de comunicación.

Dim COM_FAIL As Boolean 'bandera que indica si ha fallado la comunicación.
Dim t1_flag As Boolean = True 'bandera referida al timer.
Dim rep_com As Integer = 0 'contador de iteraciones de comunicación fallidas.
Dim repetir As Boolean = False ' bandera que indica si hay que repetir la transmisión de un dato.
Dim retardo As Integer = 0

Dim Msb, Lsb As Integer

'*************************************************************************************************
'**                                                    **
'**       FIN          CÓDIGO RELATIVO A VARIABLES                    **
'**                                                    **
'*************************************************************************************************
'*************************************************************************************************
'**                                                    **
'**              CÓDIGO RELATIVO A LA PESTAÑA "CONTROL MANUAL"              **
'**                                                    **
'*************************************************************************************************

' LLAMA A LA SUB QUE MUEVE EL MOTOR DEL EJE X A DCHA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button4.Click
    num_de_pasos.Value = num_de_pasos.Value + 1
    conversion_aBytes()
    Mueve_MotorX_izqda(Msb)
    'num_de_pasos.Value = num_de_pasos.Value - 1
    Actualizar_coordenadas()

End Sub

' LLAMA A LA SUB QUE MUEVE EL MOTOR DEL EJE X A IZQDA. TANTOS PASOS COMO SE DIGA EN <<Pasos>>.
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    num_de_pasos.Value = num_de_pasos.Value + 1
    conversion_aBytes()
    Mueve_MotorX_dcha(Msb)
    'num_de_pasos.Value = num_de_pasos.Value - 1
    Actualizar_coordenadas()

End Sub

' LLAMA A LA SUB QUE MUEVE EL MOTOR DEL EJE Y A IZQDA. TANTOS PASOS COMO SE DIGA EN <<pasos>>.
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click
    num_de_pasos.Value = num_de_pasos.Value + 1
    conversion_aBytes()
    Mueve_MotorY_dcha(Msb)
    'num_de_pasos.Value = num_de_pasos.Value - 1
    Actualizar_coordenadas()

End Sub

' LLAMA A LA SUB QUE MUEVE EL MOTOR DEL EJE Y A DCHA. TANTOS PASOS COMO SE DIGA EN <<pasos>>.
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    num_de_pasos.Value = num_de_pasos.Value + 1
    conversion_aBytes()
    Mueve_MotorY_izqda(Msb)
    'num_de_pasos.Value = num_de_pasos.Value - 1
```

```
    Actualizar_coordenadas()

  End Sub

  '*************************************************************************************************
  '**                                                                                             **
  '**        FIN DEL   CÓDIGO RELATIVO A LA PESTAÑA "CONTROL MANUAL"    **
  '**                                                                                           **
  '*************************************************************************************************
  '*************************************************************************************************
  '**                                                                                  **
  '**    CÓDIGO RELATIVO A LA PESTAÑA "TRAYECTO AUTOMÁTICO"            **
  '**                                                                              **
  '*************************************************************************************************

  ' SUB QUE MUEVE LOS MOTORES PARA DESPLAZAR LA MICROCÉLULA AL PUNTO DE COORDENADAS ESPECIFICADAS.
  ' PRIMERO SE CALCULA LA DISTANCIA EN CADA EJE Y LUEGO SE MUEVE, PRIMERO UNO Y DESPUES EL OTRO.
  Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button5.Click
    ' Variables donde se guarda el nº de pasos restantes para llegar al pto especificado.
    Dim pasosX = 0
    Dim pasosY = 0

    ' Actuamos de formas distintas en caso de que la coor sea mayor o menor que la actual.
    ' En caso de que sean iguales no se debe mover en ese eje y no se hace nada.
    If (COorX.Value > CoorX_Rel) Then
      pasosX = COorX.Value - CoorX_Rel 'Se calcula la diferencia.
      num_de_pasos.Value = pasosX + 1 ' corregimos el paso de mas que hay que mandar
      conversion_aBytes()
      Mueve_MotorX_izqda(Msb) ' Suponemos que hay que mover a dchas.

    End If
    If (COorX.Value < CoorX_Rel) Then
      pasosX = CoorX_Rel - COorX.Value
      num_de_pasos.Value = pasosX + 1 ' corregimos el paso de mas que hay que mandar
      conversion_aBytes()
      Mueve_MotorX_dcha(Msb) ' Suponemos que hay que mover a izqdas.

    End If
    If (COorY.Value > CoorY_Rel) Then
      pasosY = COorY.Value - CoorY_Rel 'Se calcula la diferencia.
      num_de_pasos.Value = pasosY + 1 ' corregimos el paso de mas que hay que mandar
      conversion_aBytes()
      Mueve_MotorY_izqda(Msb) ' Suponemos que hay que mover a dchas.

    End If
    If (COorY.Value < CoorY_Rel) Then
      pasosY = CoorY_Rel - COorY.Value
      num_de_pasos.Value = pasosY + 1 ' corregimos el paso de mas que hay que mandar
      conversion_aBytes()
      Mueve_MotorY_dcha(Msb) ' Suponemos que hay que mover a izqdas.

    End If
    Actualizar_coordenadas()
  End Sub

  '*************************************************************************************************
  '**                                                                                             **
  '**   FIN DEL   CÓDIGO RELATIVO A LA PESTAÑA "TRAYECTO AUTOMÁTICO"   **
  '**                                                                                           **
  '*************************************************************************************************
  '*************************************************************************************************
  '**                                                                                           **
```

```
'**                    SUBRUTINAS DE MOVIMIENTO DE LOS MOTORES.                **
'**                                                                            **
'********************************************************************************************

' SUB QUE MUEVE EL MOTOR DEL EJE X A DCHA. TANTOS PASOS COMO SE DIGA EN <<pasos>>.
Private Sub Mueve_MotorX_dcha(ByRef pasos As Integer)

   enviar_al_micro(pasos)
   enviar_al_micro(Lsb)
   enviar_al_micro(PROGRAMA_Mueve_MotorX_dcha)

   Ejecuta()

   num_de_pasos.Value = num_de_pasos.Value - 1

   CoorXabs = CoorXabs - num_de_pasos.Value
   CoorX_Rel = CoorX_Rel - num_de_pasos.Value

End Sub

' SUB QUE MUEVE EL MOTOR DEL EJE X A IZQDA. TANTOS PASOS COMO SE DIGA EN <<pasos>>.
Private Sub Mueve_MotorX_izqda(ByRef pasos As Integer)

   enviar_al_micro(pasos)
   enviar_al_micro(Lsb)
   enviar_al_micro(PROGRAMA_Mueve_MotorX_izqda)

   Ejecuta()

   num_de_pasos.Value = num_de_pasos.Value - 1

   CoorXabs = CoorXabs + num_de_pasos.Value
   CoorX_Rel = CoorX_Rel + num_de_pasos.Value

End Sub

' SUB QUE MUEVE EL MOTOR DEL EJE Y A DCHA. TANTOS PASOS COMO SE DIGA EN <<pasos>>.
Private Sub Mueve_MotorY_dcha(ByRef pasos As Integer)

   enviar_al_micro(pasos)
   enviar_al_micro(Lsb)
   enviar_al_micro(PROGRAMA_Mueve_MotorY_dcha)

   Ejecuta()

   num_de_pasos.Value = num_de_pasos.Value - 1

   CoorYabs = CoorYabs - num_de_pasos.Value
   CoorY_Rel = CoorY_Rel - num_de_pasos.Value

End Sub

' SUB QUE MUEVE EL MOTOR DEL EJE Y A IZQDA. TANTOS PASOS COMO SE DIGA EN <<pasos>>.
Private Sub Mueve_MotorY_izqda(ByRef pasos As Integer)

   enviar_al_micro(pasos)
   enviar_al_micro(Lsb)
   enviar_al_micro(PROGRAMA_Mueve_MotorY_izqda)

   Ejecuta()

   num_de_pasos.Value = num_de_pasos.Value - 1
```

```
    CoorYabs = CoorYabs + num_de_pasos.Value
    CoorY_Rel = CoorY_Rel + num_de_pasos.Value

End Sub

'*************************************************************************************************
'**                                                                                             **
'**          FIN     SUBRUTINAS DE MOVIMIENTO DE LOS MOTORES.                                   **
'**                                                                                             **
'*************************************************************************************************
'*************************************************************************
'**                                                                     **
'**                        SUBRUTINAS DE E/S PUERTO.                    **
'**                                                                     **
      ********************************************************************
' SUBRUTINA DE LECTURA DE DATOS DEL PUERTO.
Public Sub Lee_dato(ByVal port As Integer)

    If CheckBox1.Checked Then
        buffer_datos = Inp(port) 'si está marcada la opción IO.dll
    End If

    If CheckBox2.Checked Then
        buffer_datos = Inp(port) 'si está marcada la opción Inpout32.dll
    End If

End Sub

' SUBRUTINA DE ESCRITURA DE DATOS EN EL PUERTO.
Public Sub Escribe_dato(ByVal port As Integer)

    If CheckBox1.Checked Then
        PortWordOut(port, buffer_datos) 'si está marcada la opción IO.dll
    End If

    If CheckBox2.Checked Then
        Out(port, buffer_datos) 'si está marcada la opción Inpout32.dll
    End If

End Sub

'**                                                                     **
'**                                                                     **
'**             FIN        SUBRUTINAS DE E/S PUERTO.                    **
'**                                                                     **
'**                                                                     **
'*************************************************************************************************
'*************************************************************************************************
'**                                                                     **
'**               CÓDIGO RELATIVO A ACTUALIZACIÓN DE DATOS.             **
'**                                                                     **
'*************************************************************************************************
' ESTA SUB ACTUALIZA LAS COORDENADAS DE POSICION ACTUAL EN LA PANTALLA.
Private Sub Actualizar_coordenadas()

    Label18.Text = CStr(CoorXabs)
    Label17.Text = CStr(CoorYabs)

    'Label18.Text = CStr(CoorXabs)
    'Label17.Text = CStr(CoorYabs)
    'CoorX_Rel = CoorXabs - CoorRefX
```

```
        'CoorY_Rel = CoorYabs - CoorRefY
        Label21.Text = CStr(CoorX_Rel)
        Label20.Text = CStr(CoorY_Rel)
    End Sub
    ' ESTA SUB PERMITE SELECCIONAR LAS COORDENADAS ACTUALES COMO REFERENCIA.
    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button6.Click
        CoorX_Rel = 0
        CoorY_Rel = 0
        Actualizar_coordenadas()
    End Sub

    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        reubicar_eje_X()
    End Sub

    'ESTA SUB MANDA MOVERSE EL MOTOR DEL EJE X HASTA LLEGAR AL PUNTO X = 0
    Private Sub reubicar_eje_X()
        num_de_pasos.Value = 65534 'con esta sub se manda dar 65534 pasos en los ejes en el sentido convenido en el
        conversion_aBytes() 'esquema para tratar de que se alcancen los interruptores de final de recorrido y retomar
        Mueve_MotorX_dcha(Msb) 'la posición de partida.
        Actualizar_coordenadas()
        num_de_pasos.Value = 65534 'con esta sub se manda dar 65534 pasos en los ejes en el sentido convenido en el
        conversion_aBytes() 'esquema para tratar de que se alcancen los interruptores de final de recorrido y retomar
        Mueve_MotorX_dcha(Msb) 'la posición de partida.
        Actualizar_coordenadas()
    End Sub
    'ESTA SUB MANDA MOVERSE EL MOTOR DEL EJE Y HASTA LLEGAR AL PUNTO Y = 0
    Private Sub reubicar_eje_Y()
        num_de_pasos.Value = 65534 'con esta sub se manda dar 65534 pasos en los ejes en el sentido convenido en el
        conversion_aBytes() 'esquema para tratar de que se alcancen los interruptores de final de recorrido y retomar
        Mueve_MotorY_dcha(Msb) 'la posición de partida.
        Actualizar_coordenadas()
        num_de_pasos.Value = 65534 'con esta sub se manda dar 65534 pasos en los ejes en el sentido convenido en el
        conversion_aBytes() 'esquema para tratar de que se alcancen los interruptores de final de recorrido y retomar
        Mueve_MotorY_dcha(Msb) 'la posición de partida.
        Actualizar_coordenadas()
    End Sub
    '*************************************************************************************************
    '**                                                                                             **
    '**          FIN      CÓDIGO RELATIVO A ACTUALIZACIÓN DE DATOS.                    **
    '**                                                                                             **
    '*************************************************************************************************

    '*************************************************************************************************
    '**                                                                                             **
    '**          CÓDIGO RELATIVO A ACCIONES AL ARRANCAR Y AL SALIR DE LA APLICACIÓN.          **
    '**                                                                                             **
    '*************************************************************************************************
    'ESTA SUB DETERMINA LAS OPERACIONES QUE SE DEBEN REALIZAR AL FINALIZAR LA APLICACIÓN.
    Private Sub PosicionadorXY_FormClosing(ByVal sender As Object, ByVal e As System.Windows.Forms.FormClosingEventArgs)
Handles Me.FormClosing
        buffer_datos = 0    'ponemos todas las señales de salida del PC a cero.
        Escribe_dato(puerto_datos) 'ponemos todas las señales de salida del PC a cero.
        buffer_datos = pcReposo_CODE
        Escribe_dato(puerto_control) 'ponemos todas las señales de salida del PC a cero.
        INICIO.Close()
    End Sub
    'ESTA SUB DETERMINA LAS OPERACIONES QUE SE DEBEN REALIZAR AL INICIO DE LA APLICACIÓN.
    Private Sub PosicionadorXY_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim fallo As Boolean = True

        CoorXabs = 0 'Coordenadas de partida para el cálculo de trayectorias.
```

```
        CoorYabs = 0
        CoorRefX = 0 'Coordenadas seleccionadas como pto de referencia, durante ejecución.
        CoorRefY = 0
        CoorX_Rel = 0 ' Coordenadas relativas calculadas a partir de las anteriores.
        CoorY_Rel = 0

        Actualizar_coordenadas()

        num_de_pasos.Value = 1

        INICIO.Visible = False
    End Sub

    'ESTA SUB DETERMINA LAS OPERACIONES QUE SE DEBEN REALIZAR AL INICIO DE LA APLICACIÓN.
    Public Sub iniciar()

            Dim Resc_aux As Integer = 0

       Resetea_micro() 'Se vuelve a resetear.

        r13.Value = 0 ' se inicializa el contador de pasos del micro a cero.
        r12.Value = 0
        r11.Value = 0
        carga_pasos_reg_micro() ' se inicializa el contador de pasos del micro a cero.

        r5.Value = 180 'se configuran los parámetros de velocidad y aceleración de los motores.
        r6.Value = 1
        r7.Value = 205
        r8.Value = 4
        config_timer() 'se configuran los parámetros de velocidad y aceleración de los motores.

        reubic_sist()

    End Sub
    '*************************************************************************************************
    '**                                                                                             **
    '**    FIN DE   CÓDIGO RELATIVO A ACCIONES AL ARRANCAR Y AL SALIR DE LA APLICACIÓN.          **
    '**                                                                                             **
    '*************************************************************************************************

    Private Sub Panel1_MouseDoubleClick(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles
Panel1.MouseDoubleClick
        autor.Visible = True
    End Sub

    Private Sub autor_MouseClick(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles
autor.MouseClick
        autor.Visible = False
    End Sub

    '*************************************************************************************************
    '**                                                                                             **
    '**    CÓDIGO RELATIVO A LAS OPERACIONES PRINCIPALES DE FUNCIONAMIENTO.                       **
    '**                                                                                             **
    '*************************************************************************************************
    'ESTA FUNCIÓN ENVIA UN DATO AL MICRO PARA QUE LO ALMACENE EN LA PILA DE DATOS Y DEVUELVE 1 SI LA
COMUNICACIÓN
    'HA SIDO EXITOSA Y 0 SI OCURRE UN ERROR EN LA COMUNICACIÓN.

    Private Function Carga_dato(ByRef dato As Integer) As Boolean
        Dim a, b, datoRetorno, flag As Integer
        flag = 0
```

```
If (paso1.Checked) Then
   For retardo = 0 To Rescritura.Value
      retardo = retardo + 1
   Next
End If

'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA BUSY = 0 (DESOCUPADO)

Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.

While (buffer_datos = busy_CODE) 'si el micro está ocupado esparamos
   Lee_dato(puerto_estado)

End While

'una vez libre el micro...

If (paso2.Checked) Then
   For retardo = 0 To Rescritura.Value
      retardo = retardo + 1
   Next
End If

buffer_datos = dato 'cargamos los ocho bits de datos a transmitir.
Escribe_dato(puerto_datos)

If (paso3.Checked) Then
   For retardo = 0 To Rescritura.Value
      retardo = retardo + 1
   Next
End If

buffer_datos = SELECT_CODE  'Activa la señal SELECT, para indicar al micro que puede ocho bits.
Escribe_dato(puerto_control)

If (paso4.Checked) Then
   For retardo = 0 To Rescritura.Value
      retardo = retardo + 1
   Next
End If

'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA QUE BUSY = 1 (OCUPADO)

Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.

While (buffer_datos <> busy_CODE) 'si busy està a cero esperamos.
   Lee_dato(puerto_estado)
        End While

If (paso5.Checked) Then
   For retardo = 0 To Rescritura.Value
      retardo = retardo + 1
   Next
End If

buffer_datos = 0 'cargamos todo a cero.
Escribe_dato(puerto_datos)

If (paso6.Checked) Then
   For retardo = 0 To Rescritura.Value
```

```
        retardo = retardo + 1
    Next
End If

buffer_datos = pcReposo_CODE  'Desactiva la señal SELECT.
Escribe_dato(puerto_control)

If (paso7.Checked) Then
    For retardo = 0 To Rescritura.Value
        retardo = retardo + 1
    Next
End If

'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA BUSY = 0 (DESOCUPADO)

Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.

While (buffer_datos = busy_CODE) 'si el micro está ocupado esparamos
    Lee_dato(puerto_estado)
End While

'una vez libre el micro...

If (paso8.Checked) Then
    For retardo = 0 To Rescritura.Value
        retardo = retardo + 1
    Next
End If

Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo
'los cuatro bits más significativos devueltos por el micro.
a = Math.Abs(128 - buffer_datos)
a = a * 2
a = a - 14

buffer_datos = SELECT_CODE  'Activa la señal SELECT, a modo de acknowledge.
Escribe_dato(puerto_control)

If (paso9.Checked) Then
    For retardo = 0 To Rescritura.Value
        retardo = retardo + 1
    Next
End If

Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo
'los cuatro bits menos significativos devueltos por el micro.
b = buffer_datos
        b = b \ 8

datoRetorno = a + b

If (datoRetorno <> dato) Then 'si no son idénticos.

    buffer_datos = pcReposo_CODE  'Desactiva la señal SELECT .
    Escribe_dato(puerto_control)

    If (paso10.Checked) Then
        For retardo = 0 To Rescritura.Value
            retardo = retardo + 1
        Next
    End If
```

```
        COM_FAIL = True ' Activamos bandera de fallo en comunicaciones para que se repita la transmisión del byte.
    End If

    If (datoRetorno = dato) Then 'si son idénticos.

        buffer_datos = INIT_CODE  'Activa la señal INIT, para indicar al micro
        '                         que debe meter el dato en la pila.
        Escribe_dato(puerto_control)

        If (paso11.Checked) Then
            For retardo = 0 To Rescritura.Value
                retardo = retardo + 1
            Next
        End If

        'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA QUE BUSY = 1 (OCUPADO)

        Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.
        buffer_datos = buffer_datos - 128

        If buffer_datos > 0 Or buffer_datos = 0 Then
            flag = 1
        End If

        While (flag = 0) 'si busy està a cero esperamos.
            Lee_dato(puerto_estado)
            buffer_datos = buffer_datos - 128
            If buffer_datos > 0 Or buffer_datos = 0 Then
                flag = 1
            End If
        End While

        If (paso12.Checked) Then
            For retardo = 0 To Rescritura.Value
                retardo = retardo + 1
            Next
        End If

        buffer_datos = pcReposo_CODE  'Desactiva la señal SELECT e INIT.
        Escribe_dato(puerto_control)

        If (paso13.Checked) Then
            For retardo = 0 To Rescritura.Value
                retardo = retardo + 1
            Next
        End If

        COM_FAIL = False ' Desactivamos bandera de fallo en comunicaciones para que se repita la transmisión del byte.

    End If

    Return COM_FAIL

End Function

'ESTA SUB ACTIVA LA SEÑAL DE EJECUCIÓN DE PROGRAMA DEL MICRO INIT.

Private Sub Ejecuta()
    Dim no_fail As Boolean = False


    If (paso14.Checked) Then
```

```
        For retardo = 0 To Rescritura.Value
            retardo = retardo + 1
        Next
    End If

    'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA BUSY = 0 (DESOCUPADO)

    Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.

    While (buffer_datos = busy_CODE) 'si el micro está ocupado esparamos
        Lee_dato(puerto_estado)
            End While
    'una vez libre el micro...

    If (paso15.Checked) Then
        For retardo = 0 To Rescritura.Value
            retardo = retardo + 1
        Next
    End If

    buffer_datos = INIT_CODE  'Activa la señal INIT, para indicar al micro
    '                         que debe meter el dato en la pila.
    Escribe_dato(puerto_control)

    If (paso16.Checked) Then
        For retardo = 0 To Rescritura.Value
            retardo = retardo + 1
        Next
    End If

    'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA QUE BUSY = 1 (OCUPADO)

    Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.

    While (buffer_datos <> busy_CODE) 'si busy està a cero esperamos.
        Lee_dato(puerto_estado)
    End While

    If (paso17.Checked) Then
        For retardo = 0 To Rescritura.Value
            retardo = retardo + 1
        Next
    End If

    buffer_datos = pcReposo_CODE  'Desactiva la señal INIT.
    Escribe_dato(puerto_control)

    If (paso18.Checked) Then
        For retardo = 0 To Rescritura.Value
            retardo = retardo + 1
        Next
    End If

End Sub
'SUB QUE HACE RESET DEL MICRO Y UNA PRUEBA DE COMUNICACIONES PARA SABER SI AL OTRO LADO
'DEL CABLE SE ENCUENTRA EL POSICIONADOR O ALGO DESCONOCIDO.
Public Sub Detecta_posicionador()
    Dim detect As Boolean = False
    Dim dat As Integer = 0
    Dim flagg As Boolean = True
    For dat = 0 To 255
        detect = Carga_dato(dat) ' Probamos a cargar 103 pasos ,por ej.,  para probar si hay algo al otro lado del cable
```

```
        '                que responda como el sistema posicionador.

        If (detect = False) Then
            'Form2.TextBox3.Text = "Se ha detectado el Sistema de Posicionamiento XY satisfactoriamente."
            'Form2.Show()

        End If 'Si la comunicación no ha sido correcta...
        If (detect) Then
            flagg = False
            Form2.TextBox3.Text = "No se ha podido reconocer el dispositivo conectado."
            Form2.Show()
        End If
    Next

    If (flagg) Then
        Form2.TextBox3.Text = "Se ha detectado el Sistema de Posicionamiento XY satisfactoriamente."
        Form2.Show()
    End If

End Sub

'ESTA SUB DETERMINA EL TIEMPO DEL TEMPORIZADOR DE RESET DE LA TARJETA DE CONTROL.
Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles Timer1.Tick

    Timer1.Enabled = False 'deshabilitamos el temporizador de 1 segundo
    Timer1.Stop()

    buffer_datos = pcReposo_CODE
    Escribe_dato(puerto_control) 'desactivamos la señal de reset del micro.


End Sub

'Sub que hace un reset del micro con el tiempo especificado
Public Sub Resetea_micro()

    buffer_datos = reset_micro_CODE
    Escribe_dato(puerto_control) 'activamos señal de reset del micro.

    Timer1.Interval = tiempo.Value 'asignamos el valor especificado en la pantalla por el usuario.

    t1_flag = True

    Timer1.Start() 'habilitamos el temporizador de 1 segundo
    Timer1.Enabled = True

End Sub

'LA SIGUIENTE SUB MANDA AL MICRO LOS 4 PARÁMETROS QUE CONFIGURAN EL TIMER CERO RESPECTO DE SU
'INFLUENCIA EN LA VELOCIDAD Y ACELERACIÓN.
Private Sub config_timer()

    Resetea_micro()
    enviar_al_micro(r6.Value)
    enviar_al_micro(r5.Value)
    enviar_al_micro(r7.Value)
    enviar_al_micro(r8.Value)
    enviar_al_micro(PROGRAMA_ConfigurarTimer)
    Ejecuta()

End Sub
```

```
'ESTA SUB SE ENCARGA DE MANDAR UN DATO AL MICRO, HACIENDO HASTA TRES INTENTOS EN CASO DE FALLO
'Y SI NO FUNCIONA LA TRANSMISIÓN EN TRES ITERACIONES MUESTRA MENSAJE DE ERROR.
Private Sub enviar_al_micro(ByVal datoo As Integer)

    For rep_com = 0 To 3 ' se prueba a mandar el dato hasta tres veces en caso de fallo y si no damos msg de error.

        repetir = Carga_dato(datoo)
        If (repetir = False) Then
            rep_com = 3 'rompemos condición de permanencia en el bucle for.
        End If
    Next rep_com

    If (repetir) Then
        Form2.TextBox3.Text = "Se ha producido un error en las comunicaciones durante tres iteraciones."
        Form2.Show()
        Exit Sub
    End If

End Sub
'*************************************************************************************************
'**                                                                                             **
'**    FIN DEL CÓDIGO RELATIVO A LAS OPERACIONES PRINCIPALES DE FUNCIONAMIENTO.                 **
'**                                                                                             **
'*************************************************************************************************

'ESTA SUB HACE UNA LLAMADA A LA SUB <<Detecta_posicionador()>>
Private Sub Button11_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button11.Click
    Detecta_posicionador()
End Sub
'ESTA SUB HACE UNA LLAMADA A LA SUB <<Resetea_micro()>>
Private Sub Button10_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button10.Click
    Resetea_micro()
End Sub

'ESTA SUB HACE UNA LLAMADA A LA SUB <<config_timer()>>
Private Sub Button14_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button14.Click
    config_timer()
End Sub

'ESTA SUB HACE UNA LLAMADA A LA SUB <<detectar_codigos()>>
Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)

    detectar_codigos()

End Sub

Private Sub detectar_codigos()
    Dim codigo As Integer

    For codigo = 0 To 255 'escribimos uno a uno los 255 códigos posibles y capturamos los
        ' cuatro necesarios.
        buffer_datos = codigo
        Escribe_dato(puerto_control)

        For retardo = 0 To Rescritura.Value
            retardo = retardo + 1
        Next

        Lee_dato(puerto_estado)

        Select Case (buffer_datos)
```

```
            Case 7
               pcReposo_CODE = codigo
            Case 199
               reset_micro_CODE = codigo
            Case 167
               INIT_CODE = codigo
            Case 151
               SELECT_CODE = codigo

         End Select

      Next
      'Ahora leemos el código de BUSY
      Lee_dato(puerto_estado)
      busy_CODE = buffer_datos

   End Sub

Private Sub Button9_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button9.Click
      Detecta_posicionador()
   End Sub

   Private Sub Button12_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button12.Click
      Rescritura.Value = 1
      ajustar_retardo()
   End Sub

   'ESTA SUB HACE UNA PRUEBA DE COMUNICACIONES CON EL MICROCOTROLADOR ENVIANDO UN DATO DE FORMA
ITERADA
   ' Y AJUSTANDO UN PARÁMETRO DE RETARDO HASTA QUE EL DATO ENVIADO SE CORRESPONDE CON EL RECIBIDO
DESDE EL MICROCONTROLADOR.

   Public Sub ajustar_retardo()
      Dim detecta As Boolean = False

      detecta = Carga_dato(103) ' Probamos a cargar 103 pasos ,por ej.,  para probar si hay algo al otro lado del cable
      '                  que responda como el sistema posicionador.

      If (detecta = False) Then
         Form2.TextBox3.Text = "Se ha detectado el Sistema de Posicionamiento XY satisfactoriamente."
         Form2.Show()
      End If 'Si la comunicación no ha sido correcta...
      If (detecta) Then
         Rescritura.Value = Rescritura.Value * 10
         ajustar_retardo()
      End If
   End Sub

   'ESTA SUB DESCOMPONE EL VALOR DE LA VARIABLE TIPO INTEGER "NUMERO DE PASOS" A SUS BYTES
CORRESPONDIENTES PARA
   'PODER ENVIARLOS AL MICRO DADO QUE EL BUS DE DATOS DEL MICRO ES DE 8 BITS.

   Private Sub conversion_aBytes()

      Dim t, r, val As Integer
      Msb = 0
      Lsb = 0
      Dim cociente, resto As Integer
      resto = 0
      '********************** rellenamos un array de 16 bit de 0´s y 1´s para realizar la conversión a binario.
      cociente = num_de_pasos.Value ' cargamos el valor a convertir
      Dim bit(15) As Boolean
```

```
    For r = 0 To 15
       bit(r) = False
    Next r 'declaramos un array de 16 bits y inicializamos a "false"
    r = 0
    While (cociente > 2 Or cociente = 2)
       resto = cociente Mod 2

       cociente = cociente \ 2

       If (resto) Then
          bit(r) = True
       End If
       If (resto = 0) Then
          bit(r) = False
       End If 'se rellena el array con true o false según el caso
       r = r + 1
    End While

    If (cociente) Then
       bit(r) = True
    End If
    If (cociente = 0) Then
       bit(r) = False
    End If
    '**************hasta aquí rellenamos un array de 16 bit de 0´s y 1´s para realizar la conversión a binario.

    ' ahora formamos dos bytes Lsb y Msb para convertir a binario de nuevo pero en los dos bytes equivalentes
    ' para poder tranfesrirlos al micro.
    For t = 0 To 7 'continuamos con la conversión a dos bytes.
       If (bit(t)) Then
          val = 1
       End If
       If (bit(t) = False) Then
          val = 0
       End If
       Lsb = Lsb + (val * (2 ^ t))
    Next

    For t = 8 To 15
       If (bit(t)) Then
          val = 1
       End If
       If (bit(t) = False) Then
          val = 0
       End If
       Msb = Msb + (val * (2 ^ (t - 8)))
    Next

End Sub

'ESTA SUB HACE UNA LLAMADA A LA SUB <<lee_reg_pasos()>>
Private Sub Button15_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button15.Click
    lee_reg_pasos()
End Sub

'ESTA SUB OBTIENE EL VALOR DEVUELTO POR EL MICRO DEL REGISTRO DE PASOS INTERNO.
Private Sub lee_reg_pasos()
    Dim aa, bb, cc, byteresul, byteaux, leer As Integer
    byteresul = 0

    enviar_al_micro(6) 'MANDAMOS EL NUMERO DE PROGRAMA
    Ejecuta()   'LO MANDAMOS EJECUTAR
```

```
'*************************************************************************************************************
'repetimos los siguientes pasos tres veces para leer los 3 bytes
'*************************************************************************************************************

For leer = 0 To 2

   'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA QUE BUSY = 1
   If (paso19.Checked) Then
      For retardo = 0 To Rescritura.Value
         retardo = retardo + 1
      Next
   End If

   Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.
   buffer_datos = buffer_datos \ 128 'Se hace una máscara para ver si el bit de Busy está activo.

   While (buffer_datos <> 0) 'si busy está a cero esperamos.
      Lee_dato(puerto_estado)
      buffer_datos = buffer_datos \ 128 'Se hace una máscara para ver si el bit de Busy está activo.
   End While
   'UNA VEZ EL MICRO HA PUESTO BUSY A 1 CONTINUAMOS

   If (paso20.Checked) Then
      For retardo = 0 To Rescritura.Value
         retardo = retardo + 1
      Next
   End If

   Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo
   'los cuatro bits más significativos devueltos por el micro.
   cc = (128 - buffer_datos)
   If (cc < 0) Then
      aa = Math.Abs(128 - buffer_datos)
   End If
   If (cc > 0 Or cc = 0) Then
      aa = buffer_datos
   End If
            aa = aa * 2
   aa = aa - 14

   If (paso21.Checked) Then
      For retardo = 0 To Rescritura.Value
         retardo = retardo + 1
      Next
   End If

   buffer_datos = SELECT_CODE  'Activa la señal SELECT, para indicar al micro
   '                           que debe meter el dato en la pila.
   Escribe_dato(puerto_control)

   If (paso22.Checked) Then
      For retardo = 0 To Rescritura.Value
         retardo = retardo + 1
      Next
   End If

   'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA BUSY = 0 (DESOCUPADO)

   Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.
   buffer_datos = buffer_datos \ 128 'Se hace una máscara para ver si el bit de Busy está activo.
```

```
        While (buffer_datos = 0) 'si busy está a uno esperamos.
           Lee_dato(puerto_estado)
           buffer_datos = buffer_datos \ 128 'Se hace una máscara para ver si el bit de Busy está activo.
        End While
        'una vez libre el micro...

        Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo
        'los cuatro bits menos significativos devueltos por el micro.
                cc = (128 - buffer_datos)
        If (cc < 0) Then
           bb = Math.Abs(128 - buffer_datos)
        End If
        If (cc > 0 Or cc = 0) Then
           bb = buffer_datos
        End If
        bb = bb \ 8

        byteaux = aa + bb 'obtenemos el byte más significativo

        byteaux = (256 ^ (2 - leer)) * byteaux ' multiplicamos por su peso.

        byteresul = byteresul + byteaux 'traspasamos el valor del byte al registro auxiliar donde se irá sumando con los bytes
restantes.


        If (paso25.Checked) Then
           For retardo = 0 To Rescritura.Value
              retardo = retardo + 1
           Next
        End If

        buffer_datos = pcReposo_CODE  'Desactiva la señal SELECT O INIT.
        Escribe_dato(puerto_control)

        'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA QUE BUSY = 1
        If (paso19.Checked) Then
           For retardo = 0 To Rescritura.Value
              retardo = retardo + 1
           Next
        End If

        Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.
        buffer_datos = buffer_datos \ 128 'Se hace una máscara para ver si el bit de Busy está activo.

        While (buffer_datos <> 0) 'si busy está a cero esperamos.
           Lee_dato(puerto_estado)
           buffer_datos = buffer_datos \ 128 'Se hace una máscara para ver si el bit de Busy está activo.
        End While
        'UNA VEZ EL MICRO HA PUESTO BUSY A 1 CONTINUAMOS

    Next leer

    buffer_datos = SELECT_CODE  'Activa la señal SELECT, para indicar al micro
    '                           que debe meter el dato en la pila.
    Escribe_dato(puerto_control)

    'COMPROBAMOS EL ESTADO DE LA SEÑAL BUSY, ESPERANDO HASTA BUSY = 0 (DESOCUPADO)

    Lee_dato(puerto_estado) ' Se lee de la dirección del registro de estado del puerto paralelo.
    buffer_datos = buffer_datos \ 128 'Se hace una máscara para ver si el bit de Busy está activo.

    While (buffer_datos = 0) 'si busy está a uno esperamos.
```

```
        Lee_dato(puerto_estado)
        buffer_datos = buffer_datos \ 128 'Se hace una máscara para ver si el bit de Busy está activo.
      End While
      'una vez libre el micro...

      buffer_datos = pcReposo_CODE  'Desactiva la señal SELECT O INIT.
      Escribe_dato(puerto_control)

      TextBox3.Text = byteresul - 1 'presentamos en pantalla el dato resultante.
      'se le suma una unidad porque el micro devuelve los pasos que da reales el motor +1.

  End Sub

  'ESTA SUB PERMITE ESCRIBIR EN EL REGISTRO INTERNO DE PASOS DEL MICRO PARA HACER COMPROBACIONES DE COMUNICACIÓN.
  Private Sub carga_pasos_reg_micro()
      enviar_al_micro(r13.Value) 'MANDAMOS LOS VALORES A ALMACENAR EN LOS REGISTROS.
      enviar_al_micro(r12.Value)
      enviar_al_micro(r11.Value)
      enviar_al_micro(7) 'MANDAMOS EL CÓDIGO DEL PROGRAMA QUE PASARÁ LA INFO DE LA PILA A LOS REGISTROS r1,r2 Y r3.

      Ejecuta() 'MANDAMOS EJECUTAR DICHO PROGRAMA.
  End Sub

  'ESTA SUB HACE UNA LLAMADA A LA SUB <<carga_pasos_reg_micro()>>
  Private Sub Button13_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button13.Click
      carga_pasos_reg_micro()
  End Sub

  Private Sub Panel2_DoubleClick(ByVal sender As Object, ByVal e As System.EventArgs) Handles Panel2.DoubleClick
      TabControl2.Visible = True
  End Sub

  Private Sub Panel3_DoubleClick(ByVal sender As Object, ByVal e As System.EventArgs) Handles Panel3.DoubleClick
      TabControl2.Visible = False
  End Sub

  Private Sub Button8_Click_1(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button8.Click
      lee_reg_pasos()
      Form2.TextBox3.Text = "El microcontrolador a registrado " & TextBox3.Text & " pasos dados en un sentido por el último motor accionado."
      Form2.Show()
  End Sub

  'ESTA SUB REALIZA LLAMADAS A LAS FUNCIONES DE REUBICAIÓN DE LA POSICIÓN DEL TABLERO EN LOS DOS EJES E INICIALIZA LAS
  'COORDENADAS.

  Private Sub reubic_sist()

      reubicar_eje_Y()
      reubicar_eje_X()
      CoorXabs = 0 'Coordenadas de partida para el cálculo de trayectorias.
      CoorYabs = 0
      CoorRefX = 0 'Coordenadas seleccionadas como pto de referencia, durante ejecución.
      CoorRefY = 0
      CoorX_Rel = 0 ' Coordenadas relativas calculadas a partir de las anteriores.
      CoorY_Rel = 0

      Actualizar_coordenadas()
```

```
        num_de_pasos.Value = 1
    End Sub
    'ESTA SUB HACE UNA LLAMADA A LA SUB <<reubic_sist()>>
    Private Sub Button19_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button19.Click
        reubic_sist()
    End Sub
End Class
```

FORM3

```
Public Class INICIO

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

        PosicionadorXY.iniciar()
        PosicionadorXY.Show()

    End Sub

    Private Sub INICIO_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load

        ' PosicionadorXY.Rescritura.Value = 1
        PosicionadorXY.buffer_datos = 0    'ponemos todas las señales de salida del PC a cero.
        PosicionadorXY.Escribe_dato(PosicionadorXY.puerto_datos) 'ponemos todas las señales de salida del PC a cero.

        PosicionadorXY.buffer_datos = PosicionadorXY.pcReposo_CODE 'ponemos todas las señales de salida del PC a cero.
        PosicionadorXY.Escribe_dato(PosicionadorXY.puerto_control) 'ponemos todas las señales de salida del PC a cero.

        PosicionadorXY.Resetea_micro() 'reseteamos el micro para que evitar problemas en el encendido con el puerto del PC.

        Button1.Visible = False

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        Dim it As Integer = 0
        Dim Resc_aux As Integer = 0

        For it = 0 To 99 'el correcto funcionamiento de las comunicaciones, con el fin de quedarnos
            PosicionadorXY.Rescritura.Value = 1
            PosicionadorXY.ajustar_retardo() 'con el valor mayor para garantizar una buena comunicación.
            If (PosicionadorXY.Rescritura.Value > Resc_aux) Then
                Resc_aux = PosicionadorXY.Rescritura.Value
            End If

        Next
        PosicionadorXY.Rescritura.Value = Resc_aux ' asignamos el mejor valor para el retardo encontrdo
        'NumericUpDown1.Value = PosicionadorXY.Rescritura.Value

        PosicionadorXY.Detecta_posicionador() 'se realiza una comprobación exhaustiva con todos los posibles datos

        If (Form2.TextBox3.Text = "Se ha detectado el Sistema de Posicionamiento XY satisfactoriamente.") Then
            Button1.Visible = True
        End If
        If (Form2.TextBox3.Text = "No se ha podido reconocer el dispositivo conectado.") Then
            Form2.TextBox3.Text = "No se ha podido reconocer el dispositivo conectado. Pruebe a salir y lanzar de nuevo la aplicación."
        End If

    End Sub

End Class
```

ANEXO III

Código fuente del programa ensamblador del microcontrolador.

```
;*****************************************************************************************************************
;*****************************************************************************************************************
;*****************          --PROYECTO FIN DE CARRERA : POSICIONADOR X Y --                    ***********
;*****************************************************************************************************************
;********* Versión 2.0 ********************************************************************************************
;****************** PROGRAMA PRINCIPAL DE CONTROL DE LOS MOTORES PARA UN MICRO AVR ATmega8535 *********
;********* 2008 ***************************************************************************************************
;*****************************************************************************************************************
.cseg
.org 0x000
rjmp RESET ; Reset Handler
.org 0x001
rjmp EXT_INT0 ; IRQ0 Handler
                        ;rjmp EXT_INT1 ; IRQ1 Handler
                        ;rjmp TIM2_COMP ; Timer2 Compare Puntero
                        ;rjmp TIM2_OVF ; Timer2 Overflow Puntero
                        ;rjmp TIM1_CAPT ; Timer1 Capture Puntero
                        ;rjmp TIM1_COMPA ; Timer1 Compare A Puntero
                        ;rjmp TIM1_COMPB ; Timer1 Compare B Puntero
                        ;rjmp TIM1_OVF ; Timer1 Overflow Puntero
.org 0x009
                        rjmp TIM0_OVF ; Timer0 Overflow Puntero
                        ;rjmp SPI_STC ; SPI Transfer Complete Puntero
                        ;rjmp USART_RXC ; USART RX Complete Puntero
                        ;rjmp USART_UDRE ; UDR Empty Puntero
                        ;rjmp USART_TXC ; USART TX Complete Puntero
                        ;rjmp AnalogDC ; ADC Conversion Complete Puntero
                        ;rjmp EE_RDY ; EEPROM Ready Puntero
                        ;rjmp ANA_COMP ; Analog Comparator Puntero
                        ;rjmp TWSI ; Two-wire Serial Interface Puntero
                        ;rjmp EXT_INT2 ; IRQ2 Puntero
                        ;rjmp TIM0_COMP ; Timer0 Compare Puntero
                        ;rjmp SPM_RDY ; Store Program Memory Ready Puntero


.org 0x15
RESET:

;*****************************************************************************************************************
                ;CONFIGURAMOS LA POSICIÓN DE COMIENZO DE LA PILA DE DATOS, SE TRATA DE UNA PILA LIFO.

                        ldi r18,$02 ;
                        ldi r26,$5e ;prepara la direccion de memoria donde comienza la pila
                        st x,r18  ;para manejo de subrutinas.
                        ldi r18,$5c ;
                        ldi r26,$5d ;prepara la direccion de memoria donde comienza la pila
                        st x,r18  ;para manejo de subrutinas.

;*****************************************************************************************************************
                        ldi r18,$ff
                        ldi r26,$3A ;prepara el puerto A como salida
                        st x,r18  ;de datos
;*****************************************************************************************************************

;*****************************************************************************************************************
                        ldi r18,$1e
                        ldi r26,$37 ;prepara el puerto B como entrada y salida
                        st x,r18  ;de datos
;*****************************************************************************************************************

;*****************************************************************************************************************
                        ldi r18,$0
```

```
                                ldi r26,$34 ;prepara el puerto C como entrada
                                st x,r18  ;de datos
;*****************************************************************************************************************

;*****************************************************************************************************************
                                ldi r18,0b10000000
                                ldi r26,$31 ;prepara el puerto D como entrada menos el BUSY
                                st x,r18  ;de datos
;*****************************************************************************************************************

                                sbi $12,7  ;activamos el BUSY a nivel alto.

Atencion_PC:

                ;DESHABILITAMOS TODAS LAS INTERRUPCIONES.

                                ldi r18,0
                                ldi r26,$5b ;Prepara la direccion de memoria donde mandaremos el contenido de r18
                                st x,r18  ;ACTUAMOS SOBRE GMISK
                                ldi r26,$5A
                                st x,r18  ;ACTUAMOS SOBRE GIFR
                                cli       ;Actuamos sobre el bit I del SREG para deshabilitar todas las interrupciones.
                                ldi r26,$59 ;
                                st x,r18  ;ACTUAMOS SOBRE TIMSK
                                ldi r26,$58
                                st x,r18  ;ACTUAMOS SOBRE TIFR

                ;DESHABILITAMOS TODAS LAS INTERRUPCIONES.

                ;HABILITAMOS LA INTERRUPCION INT0
                                ldi r18,0b11100000 ; Primero borramos las interrupciones memorizadas
                                ldi r26,$5A ;hasta el momento.
                                st x,r18  ;ACTUAMOS SOBRE GIFR


                                ldi r18,0b01000000
                                ldi r26,$5b ;Prepara la direccion de memoria donde mandaremos el contenido de r18
                                st x,r18  ;ACTUAMOS SOBRE GICR
                                ldi r18,0b00000000
                                ldi r26,$55 ;Prepara la direccion de memoria donde mandaremos el contenido de r18
                                st x,r18  ;ACTUAMOS SOBRE MCUCR para determinar condiciones de disparo de INT0
                                sei; Actuamos sobre el bit I del SREG para habilitar todas las interrupciones configuradas.


monitoriza_activacion:

                                cbi $12,7  ;desactivamos el BUSY a nivel bajo.

                                ldi r18,0 ;desactivamos la bandera sobre la cuenta de pasos.
                                mov r1,r18

                                ldi r18,$0
                                ldi r26,$34 ;prepara el puerto C como entrada
                                st x,r18  ;de datos

                                ;MONITORIZACIÓN DE LA SEÑAL CAPTURA DE CÓDIGOS

                                sbic $16,0
                                rcall captura_codigos ;si está activado saltamos

                                ;MONITORIZACIÓN DE LA SEÑAL SELECT
```

```
                        sbic $10,6
                        rjmp leer_dato_apila ;si está activado lee dato a la pila desde el puerto B

                        ;MONITORIZACIÓN DE LA SEÑAL INIT

                        sbic $10,5
                        rjmp Ejecucion ;si está activado ejecuta el programa

                        rjmp monitoriza_activacion ;si no, sigue monitorizando.

;*****************************************************************************************************************
Ejecucion:
                        sbi $12,7  ;ponemos  el BUSY a nivel alto.

;ESPERAMOS QUE INIT VUELVA A CERO.
espera_pc4:             in r19,$10   ;leemos a r19 el dato del puerto D
                        ldi r16,0b00100000          ;cargamos una máscara.
                        and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                        cpi r16,0b00000000          ;comparamos para saber si está a uno dicho bit.
                        breq salto2
                        rjmp espera_pc4; si no seguimos monitorizando dicho bit.


salto2:
                        cbi $12,7  ;ponemos  el BUSY a nivel bajo.

;GESTIONAMOS LA ELECCIÓN DEL PROGRAMA.
                        pop r16   ;cargamos en r16 el programa y comparamos para saber de qué programa se trata.

                        cpi r16,1
                        breq Prg1 ;Motor 1 derecha
                        cpi r16,2
                        breq Prg2 ;Motor 1 izquierda
                        cpi r16,3
                        breq Prg3 ;Motor 2 derecha
                        cpi r16,4
                        breq Prg4 ;Motor 2 izquierda
                        cpi r16,5
                        breq Prg5 ;Modifica la temporización del timer0 y la aceleraciòn.
                        cpi r16,6
                        breq Prg6 ;Manda al PC el registro de pasos, reales.
                        cpi r16,7
                        breq Prg7 ;Carga el registro de pasos con un dato enviado por el PC
                                  ;conocido, para después probar el prog6.
                        rjmp Atencion_PC ; si no coincidiera con nigún código válido, vuelve a escuchar al PC.

Prg1:
                        pop r19 ;recuperamos de la pila LSByte de pasos
                        pop r24 ;recuperamos de la pila MSByte de pasos
                        mov r5,r4;restauramos valor precerga del timer.
                        rcall Prog1         ;apaño para que el salto relativo esté dentro del alcance
                        rjmp Atencion_PC ;devolvemos el control a la rutina de atención al PC.

Prg2:
                        pop r19 ;recuperamos de la pila LSByte de pasos
                        pop r24 ;recuperamos de la pila MSByte de pasos
                        mov r5,r4;restauramos valor precerga del timer.
                        rcall Prog2         ;permitido.
                        rjmp Atencion_PC ;devolvemos el control a la rutina de atención al PC.
Prg3:
                        pop r19 ;recuperamos de la pila LSByte de pasos
                        pop r24 ;recuperamos de la pila MSByte de pasos
```

```
                        mov r5,r4;restauramos valor precerga del timer.
                        rcall Prog3
                        rjmp Atencion_PC ;devolvemos el control a la rutina de atención al PC.

Prg4:                   pop r19 ; recuperamos de la pila LSByte de pasos
                        pop r24 ;recuperamos de la pila MSByte de pasos
                        mov r5,r4;restauramos valor precerga del timer.
                        rcall Prog4
                        rjmp Atencion_PC ;devolvemos el control a la rutina de atención al PC.
Prg5:
                        pop r8 ;cargamos el valor del prescaler.
                        pop r7 ;cargamos el valor del incremento o aceleración.
                        pop r5 ;cargamos el valor de precarga.
                        pop r6 ;cargamos el valor de variación.
                        mov r4,r5;hace una copia del valor de precarga en r4
                        rjmp Atencion_PC ;devolvemos el control a la rutina de atención al PC.

Prg7:                   pop r11
                        pop r12
                        pop r13
                        rjmp Atencion_PC ;devolvemos el control a la rutina de atención al PC.

Prg6:
; ENVIAMOS AL PUERTO LOS PRIMEROS 4 BITS MÁS SIGNIFICATIVOS.
                        ;mov r3,r13 ; cargamos el byte en el registro adecuado para hacer swapping
                        ;rcall SWAP_BYTE ;invocamos la función de swapping
                        ;mov r24,r9;volcamos el resultado del swapping en el registro auxiliar  r24
                        mov r24, r13

                        andi r24,0b11110000
                        lsr r24
                        lsr r24
                        lsr r24 ; colocamos para que coincidan los bits a enviar con los bits de salida
                                ; del puerto.

                        out $18,r24;enviamos el dato al puerto.

                        sbi $12,7  ;ponemos  el BUSY a nivel alto.

; ESPERAMOS HASTA SELECT A NIVEL ALTO.

espera_pc06:            in r19,$10   ;leemos a r19 el dato del puerto D
                        ldi r16,0b01000000          ;cargamos una máscara.
                        and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                        cpi r16,0b01000000          ;comparamos para saber si está a uno dicho bit.
                        breq CONT6 ;si está desactivado regresamos
                        rjmp espera_pc06; si no seguimos monitorizando dicho bit.

;ENVIAMOS AL PUERTO LOS PRIMEROS 4 BITS MÁS SIGNIFICATIVOS.
CONT6:

;ENVIAMOS AL PUERTO LOS SEGUNDOS 4 BITS MÁS SIGNIFICATIVOS.

                        ;mov r3,r13 ; cargamos el byte en el registro adecuado para hacer swapping
                        ;rcall SWAP_BYTE ;invocamos la función de swapping
                        ;mov r24,r9;volcamos el resultado del swapping en el registro auxiliar  r24
                        mov r24, r13

                        andi r24,0b00001111
                        lsl r24;colocamos para que coincidan los bits a enviar con los bits de salida
                                ;del puerto.
```

```
                        out $18,r24;enviamos el dato al puerto.

                        cbi $12,7  ;ponemos  el BUSY a nivel bajo.

;ESPERAMOS HASTA SELECT A NIVEL BAJO.
espera_pc16:            in r19,$10   ;leemos a r19 el dato del puerto D
                        ldi r16,0b01000000      ;cargamos una máscara.
                        and r16,r19     ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                        cpi r16,0b00000000      ;comparamos para saber si está a cero dicho bit.
                        breq CONT16 ;si está desactivado regresamos
                        rjmp espera_pc16; si no seguimos monitorizando dicho bit.

;ENVIAMOS AL PUERTO LOS SEGUNDOS 4 BITS MÁS SIGNIFICATIVOS.
CONT16:

;ENVIAMOS AL PUERTO LOS TERCEROS 4 BITS MÁS SIGNIFICATIVOS.
                        ;mov r3,r12 ; cargamos el byte en el registro adecuado para hacer swapping
                        ;rcall SWAP_BYTE ;invocamos la función de swapping
                        ;mov r24,r9;volcamos el resultado del swapping en el registro auxiliar  r24
                        mov r24,r12

                        andi r24,0b11110000
                        lsr r24
                        lsr r24
                        lsr r24;colocamos para que coincidan los bits a enviar con los bits de salida
                                ;del puerto.

                        out $18,r24;enviamos el dato al puerto.

                        sbi $12,7  ;ponemos  el BUSY a nivel alto.

;ESPERAMOS HASTA SELECT A NIVEL ALTO.
espera_pc26:            in r19,$10   ;leemos a r19 el dato del puerto D
                        ldi r16,0b01000000      ;cargamos una máscara.
                        and r16,r19     ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                        cpi r16,0b01000000      ;comparamos para saber si está a uno dicho bit.
                        breq CONT26 ;si está desactivado regresamos
                        rjmp espera_pc26; si no seguimos monitorizando dicho bit.
;ENVIAMOS AL PUERTO LOS TERCEROS 4 BITS MÁS SIGNIFICATIVOS.
CONT26:

;ENVIAMOS AL PUERTO LOS CUARTOS 4 BITS MÁS SIGNIFICATIVOS.

                        ;mov r3,r12 ; cargamos el byte en el registro adecuado para hacer swapping
                        ;rcall SWAP_BYTE ;invocamos la función de swapping
                        ;mov r24,r9;volcamos el resultado del swapping en el registro auxiliar  r24
                        mov r24,r12

                        andi r24,0b00001111
                        lsl r24;colocamos para que coincidan los bits a enviar con los bits de salida
                                ;del puerto.

                        out $18,r24;enviamos el dato al puerto.

                        cbi $12,7  ;ponemos  el BUSY a nivel bajo.

;ESPERAMOS HASTA SELECT A NIVEL ALTO.
espera_pc36:            in r19,$10   ;leemos a r19 el dato del puerto D
                        ldi r16,0b01000000      ;cargamos una máscara.
                        and r16,r19     ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                        cpi r16,0b00000000      ;comparamos para saber si está a cero dicho bit.
                        breq CONT36 ;si está desactivado regresamos
```

```
                                rjmp espera_pc36; si no seguimos monitorizando dicho bit.

;ENVIAMOS AL PUERTO LOS CUARTOS 4 BITS MÁS SIGNIFICATIVOS.
CONT36:

;ENVIAMOS AL PUERTO LOS QUINTOS 4 BITS MÁS SIGNIFICATIVOS.
                                ;mov r3,r11 ; cargamos el byte en el registro adecuado para hacer swapping
                                ;rcall SWAP_BYTE ;invocamos la función de swapping
                                ;mov r24,r9;volcamos el resultado del swapping en el registro auxiliar  r24
                                mov r24,r11

                                andi r24,0b11110000
                                lsr r24
                                lsr r24
                                lsr r24;colocamos para que coincidan los bits a enviar con los bits de salida
                                        ;del puerto.

                                out $18,r24;enviamos el dato al puerto.

                                sbi $12,7  ;ponemos  el BUSY a nivel alto.

;ESPERAMOS HASTA SELECT A NIVEL ALTO.
espera_pc46:                    in r19,$10   ;leemos a r19 el dato del puerto D
                                ldi r16,0b01000000          ;cargamos una máscara.
                                and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                                cpi r16,0b01000000          ;comparamos para saber si está a uno dicho bit.
                                breq CONT46 ;si está desactivado regresamos
                                rjmp espera_pc46; si no seguimos monitorizando dicho bit.
;ENVIAMOS AL PUERTO LOS QUINTOS 4 BITS MÁS SIGNIFICATIVOS.
CONT46:

;ENVIAMOS AL PUERTO LOS ÚLTIMOS 4 BITS MÁS SIGNIFICATIVOS.

                                ;mov r3,r11 ; cargamos el byte en el registro adecuado para hacer swapping
                                ;rcall SWAP_BYTE ;invocamos la función de swapping
                                ;mov r24,r9;volcamos el resultado del swapping en el registro auxiliar  r24
                                mov r24,r11

                                andi r24,0b00001111
                                lsl r24;colocamos para que coincidan los bits a enviar con los bits de salida
                                        ;del puerto.

                                out $18,r24;enviamos el dato al puerto.

                                cbi $12,7  ;ponemos  el BUSY a nivel bajo.

;ESPERAMOS HASTA SELECT A NIVEL ALTO.
espera_pc56:                    in r19,$10   ;leemos a r19 el dato del puerto D
                                ldi r16,0b01000000          ;cargamos una máscara.
                                and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                                cpi r16,0b00000000          ;comparamos para saber si está a cero dicho bit.
                                breq CONT56 ;si está desactivado regresamos
                                rjmp espera_pc56; si no seguimos monitorizando dicho bit.

;ENVIAMOS AL PUERTO LOS ÚLTIMOS 4 BITS MÁS SIGNIFICATIVOS.
CONT56:
                                ldi r18,0
                                out$18,r18;ponemos a cero las salidas del puerto.
                                sbi $12,7  ;ponemos  el BUSY a nivel alto.

;ESPERAMOS HASTA SELECT A NIVEL ALTO.
espera_pc46X:                   in r19,$10   ;leemos a r19 el dato del puerto D
```

```
                                ldi r16,0b01000000          ;cargamos una máscara.
                                and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                                cpi r16,0b01000000          ;comparamos para saber si está a uno dicho bit.
                                breq CONT46X ;si está desactivado regresamos
                                rjmp espera_pc46X; si no seguimos monitorizando dicho bit.
;ENVIAMOS AL PUERTO LOS QUINTOS 4 BITS MÁS SIGNIFICATIVOS.
CONT46X:

                                cbi $12,7  ;ponemos  el BUSY a nivel bajo.
                                rjmp Atencion_PC ;devolvemos el control a la rutina de atención al PC.

;*****************************************************************************************************************

                        ;CARGAMOS EL DATO LEIDO DEL PUERTO B A LA PRIMERA POSICIÓN DE LA PILA.
leer_dato_apila:
                                clz
                                in r23,$13 ; leemos dato del puerto C
                                sbi $12,7  ;ponemos  el BUSY a nivel alto.
;AHORA EL PC DEBE PONER TODOS LOS BITS A CERO Y LA SEÑAL DE SELECT TB
; EL MICRO ESPERA A QUE SELECT SE PONGA A CERO.

espera_pc0:                     in r19,$10   ;leemos a r19 el dato del puerto D
                                ldi r16,0b01000000          ;cargamos una máscara.
                                and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                                cpi r16,0b00000000          ;comparamos para saber si está a cero dicho bit.
                                breq CONT ;si está desactivado regresamos
                                rjmp espera_pc0; si no seguimos monitorizando dicho bit.

CONT:                           mov r24,r23 ;hacemos una copia para no perder la info.
                                andi r24,0b11110000 ;cribamos para dejar los 4 MSB
                                lsr r24
                                lsr r24
                                lsr r24    ;desplazamos los bits para ajustarlos a la máscare de salida por
                                        ;el puerto B


                                out $18,r24 ; devolvemos el dato al puerto B

                                cbi $12,7  ;ponemos  el BUSY a nivel bajo.

;AHORA EL PC DEBE LEER EL DATO COMPARALO CON EL ENVIADO Y SI COINCIDE PONE INIT A 1

;EL MICRO ESPERA A QUE SELECT SE PONGA A UNO PARA LUEGO LEER INIT.

CONTINUA:
                                in r19,$10   ;leemos a r19 el dato del puerto D
                                ldi r16,0b01000000          ;cargamos una máscara.
                                and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                                cpi r16,0b01000000          ;comparamos para saber si está a uno dicho bit.

                                breq continua_lectura_a_pila ;si está activado lee dato a la pila desde el puerto B
                                rjmp CONTINUA; si todavía no se activa la señal sigue monitorizando.

continua_lectura_a_pila:

                                mov r24,r23 ;hacemos una copia para no perder la info.
                                andi r24,0b00001111 ;cribamos para dejar los 4 MSB
                                lsl r24     ; desplazamos los bits para ajustarlos a la máscara de salida por
                                        ;el puerto B

                                out $18,r24 ; devolvemos el dato al puerto B
```

```
                                sbi $12,7  ;ponemos  el BUSY a nivel alto.

;ESPERAMOS SELECT PONGA A CERO DE NUEVO.
espera_pc1:                     in r19,$10   ;leemos a r19 el dato del puerto D
                                ldi r16,0b01000000          ;cargamos una máscara.
                                and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                                cpi r16,0b00000000          ;comparamos para saber si está a cero dicho bit.
                                breq CONTIN ;si está desactivado regresamos
                                rjmp espera_pc1; si no seguimos monitorizando dicho bit.
CONTIN:
                                in r19,$10   ;leemos a r19 el dato del puerto D
                                ldi r16,0b00100000          ;cargamos una máscara para ver INIT
                                and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                                cpi r16,0b00100000          ;comparamos para saber si está a uno dicho bit.

                                breq almacenar_dato ;si está activado cargamos el dato en la pila.

                                ldi r23,$0
                                out $18,r23;ponemos en reposo la palabra de datos.
                                rjmp monitoriza_activacion ;si no, sigue monitorizando init y select.

almacenar_dato:
                                push r23; cargamos el dato en la pila.
                                cbi $12,7  ;ponemos  el BUSY a nivel bajo.

;ESPERAMOS INIT PONGA A CERO DE NUEVO.
espera_pc2:                     in r19,$10   ;leemos a r19 el dato del puerto D
                                ldi r16,0b00100000          ;cargamos una máscara.
                                and r16,r19         ;efectuamos el enmascarado para dejar sólo el bit que interesa.
                                cpi r16,0b00000000          ;comparamos para saber si está a cero dicho bit.
                                breq next ;si está desactivado regresamos
                                rjmp espera_pc2; si no seguimos monitorizando dicho bit.
next:                           ldi r23,$0
                                out $18,r23;ponemos en reposo la palabra de datos.
                                rjmp monitoriza_activacion ;si no, sigue monitorizando init y select.

;*****************************************************************************************************************
;*****************************************************************************************************************
;****************** --PROYECTO FIN DE CARRERA : POSICIONADOR X Y --                                  ***********
;*****************************************************************************************************************
;********* ***************************************************************************************
;******************* CÓDIGO DE LOS PROGRAMAS DE FUNCIONAMIENTO PREVISTOS PARA EL PROYECTO.   **********
;********* ***************************************************************************************
;*****************************************************************************************************************
Prog1: ;Motor M2 izqda.

                clz ;pone a cero el flag para que no haya saltos inesperados
                mov r0,r24; hacemos duplicado de r24 para mandar dar ese num. de pasos
                ;al motor para compensar las perdidas producidas por el algoritmo de ejecución de
                ;los pasos.

salta1:
                ldi r18,1 ;habilitamos interrupción del timer0
                ldi r26,$59 ;habilitamos interrupción del timer0
                st x,r18 ;habilitamos interrupción del timer0

                mov r18,r1;comprobamos el estado de la bandera de la cuenta de pasos.
                cpi r18,1;
                breq no_reset_pasos0
                ldi r18,0;ponemos a cero la cuenta de pasos.
                mov r11,r18;ponemos a cero la cuenta de pasos.
                mov r12,r18;ponemos a cero la cuenta de pasos.
```

```
                mov r13,r18;ponemos a cero la cuenta de pasos.

no_reset_pasos0:

                rcall Mov_M2_izqda      ;llamamos a la rutina.

                mov r19,r0;cargamos los pasos adicionales
                cpi r19,0
                breq s2
                ldi r24,0;aseguramos que r24 queda a cero
                rol r28 ;movemos el 1 lógico un bit a la izqda.
                rcall Mov_M2_izqda      ;llamamos a la rutina.

s2:             mov r18,r8 ;arrancamos timer0 a frecuencia CK
                ldi r26,$53 ;arrancamos timer0 a frecuencia CK
                st x,r18 ;arrancamos timer0 a frecuencia CK
                sei
esp1:           cpi r18,33;el timer cargará el valor 33 en r18 para provocar la salida
                brne esp1;mientras r18 no sea 33 cierra el bucle
                ldi r16,0
                out $1b,r16 ;ponemos el puerto A en reposo.
                ldi r18,0 ;deshabilitamos interrupción del timer0
                ldi r26,$59 ;deshabilitamos interrupción del timer0
                st x,r18 ;deshabilitamos interrupción del timer0

                ret

Prog2: ;Motor M2 decha.

                clz ;pone a cero el flag para que no haya saltos inesperados
                mov r0,r24; hacemos duplicado de r24 para mandar dar ese num. de pasos
                ;al motor para compensar las perdidas producidas por el algoritmo de ejecución de
                ;los pasos.

salta2:
                ldi r18,1 ;habilitamos interrupción del timer0
                ldi r26,$59 ;habilitamos interrupción del timer0
                st x,r18 ;habilitamos interrupción del timer0

                mov r18,r1;comprobamos el estado de la bandera de la cuenta de pasos.
                cpi r18,1;
                breq no_reset_pasos1
                ldi r18,0;ponemos a cero la cuenta de pasos.
                mov r11,r18;ponemos a cero la cuenta de pasos.LSB
                mov r12,r18;ponemos a cero la cuenta de pasos.
                mov r13,r18;ponemos a cero la cuenta de pasos MSB.

no_reset_pasos1:

                rcall Mov_M2_dcha       ;llamamos a la rutina.

                mov r19,r0;cargamos los pasos adicionales
                cpi r19,0
                breq s1
                ldi r24,0;aseguramos que r24 queda a cero
                ror r28 ;movemos el 1 lógico un bit a la izqda.
                rcall Mov_M2_dcha       ;llamamos a la rutina.

s1:             mov r18,r8 ;arrancamos timer0 a frecuencia CK
                ldi r26,$53 ;arrancamos timer0 a frecuencia CK
                st x,r18 ;arrancamos timer0 a frecuencia CK
                sei
```

```
esp2:           cpi r18,33;el timer cargará el valor 33 en r18 para provocar la salida
                brne esp2;mientras r18 no sea 33 cierra el bucle.
                ldi r16,0
                out $1b,r16 ;ponemos el puerto A en reposo.
                ldi r18,0 ;deshabilitamos interrupción del timer0
                ldi r26,$59 ;deshabilitamos interrupción del timer0
                st x,r18 ;deshabilitamos interrupción del timer0

                ret

Prog3: ;Motor M1 izqda.

                clz ;pone a cero el flag para que no haya saltos inesperados
                mov r0,r24; hacemos duplicado de r24 para mandar dar ese num. de pasos
                ;al motor para compensar las perdidas producidas por el algoritmo de ejecución de
                ;los pasos.

salta3:
                ldi r18,1 ;habilitamos interrupción del timer0
                ldi r26,$59 ;habilitamos interrupción del timer0
                st x,r18 ;habilitamos interrupción del timer0

                mov r18,r1;comprobamos el estado de la bandera de la cuenta de pasos.
                cpi r18,1;
                breq no_reset_pasos2
                ldi r18,0;ponemos a cero la cuenta de pasos.
                mov r11,r18;ponemos a cero la cuenta de pasos.
                mov r12,r18;ponemos a cero la cuenta de pasos.
                mov r13,r18;ponemos a cero la cuenta de pasos.

no_reset_pasos2:

                rcall Mov_M1_izqda      ;llamamos a la rutina.

                mov r19,r0;cargamos los pasos adicionales
                cpi r19,0
                breq s3
                ldi r24,0;aseguramos que r24 queda a cero
                rol r25 ;movemos el 1 lógico un bit a la izqda.
                rcall Mov_M1_izqda      ;llamamos a la rutina.

s3:             mov r18,r8 ;arrancamos timer0 a frecuencia CK
                ldi r26,$53 ;arrancamos timer0 a frecuencia CK
                st x,r18 ;arrancamos timer0 a frecuencia CK
                sei
esp3:           cpi r18,33;el timer cargará el valor 33 en r18 para provocar la salida
                brne esp3;mientras r18 no sea 33 cierra el bucle.
                ldi r16,0
                out $1b,r16 ;ponemos el puerto A en reposo.
                ldi r18,0 ;deshabilitamos interrupción del timer0
                ldi r26,$59 ;deshabilitamos interrupción del timer0
                st x,r18 ;deshabilitamos interrupción del timer0

                ret

Prog4: ;Motor M1 decha.

                clz ;pone a cero el flag para que no haya saltos inesperados
                mov r0,r24; hacemos duplicado de r24 para mandar dar ese num. de pasos
                ;al motor para compensar las perdidas producidas por el algoritmo de ejecución de
                ;los pasos.
```

```
salta4:
                ldi r18,1 ;habilitamos interrupción del timer0
                ldi r26,$59 ;habilitamos interrupción del timer0
                st x,r18 ;habilitamos interrupción del timer0

                mov r18,r1;comprobamos el estado de la bandera de la cuenta de pasos.
                cpi r18,1;
                breq no_reset_pasos3
                ldi r18,0;ponemos a cero la cuenta de pasos.
                mov r11,r18;ponemos a cero la cuenta de pasos.
                mov r12,r18;ponemos a cero la cuenta de pasos.
                mov r13,r18;ponemos a cero la cuenta de pasos.

no_reset_pasos3:

                rcall Mov_M1_dcha       ;llamamos a la rutina.

                mov r19,r0;cargamos los pasos adicionales
                cpi r19,0
                breq s4
                ldi r24,0;aseguramos que r24 queda a cero
                ror r25 ;movemos el 1 lógico un bit a la izqda.
                rcall Mov_M1_dcha       ;llamamos a la rutina.

s4:             mov r18,r8 ;arrancamos timer0 a frecuencia CK
                ldi r26,$53 ;arrancamos timer0 a frecuencia CK
                st x,r18 ;arrancamos timer0 a frecuencia CK
                sei
esp4:   cpi r18,33;el timer cargará el valor 33 en r18 para provocar la salida
                brne esp4;mientras r18 no sea 33 cierra el bucle.
                ldi r16,0
                out $1b,r16 ;ponemos el puerto A en reposo.
                ldi r18,0 ;deshabilitamos interrupción del timer0
                ldi r26,$59 ;deshabilitamos interrupción del timer0
                st x,r18 ;deshabilitamos interrupción del timer0

                ret

TIM0_OVF:

                sei
                ldi r18,0 ;paramos el timer0
                ldi r26,$53 ;
                st x,r18

                ldi r18,33;cargamos un valor para que la comparación provoque la salida
                                        ;del bucle de espera.

                cp r5,r7 ;si r5 es >= 254 se mantiene, si no, se incrementa
                brsh noSuma; sumándole r6

                add r5,r6;se incrementa r5 disminuyendo el tiempo del T0

noSuma:
                ldi r26,$52 ;precargamos de nuevo el timer0
                st x,r5 ;

                reti;regresamos habilitando el bit I de SREG

EXT_INT0:

                ldi r18,0 ;paramos el timer0
```

```
		ldi r26,$53 ;
		st x,r18

		ldi r18,1 ;activamos una bandera para que la cuenta de pasos al producirse
		mov r1,r18;una interrrupción sea correcta. Esta será r1 = 1 para indicar que
		; no se debe poner a cero el registro de pasos al dar pasos de retroceso.
	;DESHABILITAMOS LA INTERRUPCION INT0
	;	ldi r18,0b11100000 ; Primero borramos las interrupciones memorizadas
	;	ldi r26,$5A ;hasta el momento.
	;	st x,r18  ;ACTUAMOS SOBRE GIFR


		ldi r18,0b00000000 ;
		ldi r26,$5b ;Prepara la direccion de memoria donde mandaremos el contenido de r18
		st x,r18  ;ACTUAMOS SOBRE GICR
		sei

		ldi r18,160;configuramos el timer con un valor por defecto
		mov r5,r18;para que puede funcionar.
		ldi r18,4;si se produce int0 sin haber dado tiempo a configurarlo.
		mov r8,r18

		sbic $10,0; si el bit es 1 ejecuta la siguiente línea, si no, se la salta.
		rjmp saltaaProg2; si se trata del fin de recorrido 1...
		sbic $10,1
		rjmp saltaaProg1; si se trata del fin de recorrido 2...
		sbic $10,3
		rjmp saltaaProg3; si se trata del fin de recorrido 3...
		sbic $10,4
		rjmp saltaaProg4; si se trata del fin de recorrido 4...

		reti ;si no se ha pulsado todavía el segundo interruptor
				;continua hasta que se pulse para saber qué prog
				;se debe ejecutar.


saltaaProg1:
			;rcall resta_pasosM2
		ldi r19,145 ;cargamos 200 pasos y mandamos M2 izqda.
		ldi r24,1
		push r24
		push r19
		rcall Prog1
		ldi r18,$02 ;;restauramos la pila
		ldi r26,$5e ;prepara la direccion de memoria donde comienza la pila
		st x,r18  ;para manejo de subrutinas.
		ldi r18,$5c ;
		ldi r26,$5d ;prepara la direccion de memoria donde comienza la pila
		st x,r18  ;para manejo de subrutinas.
			;HABILITAMOS LA INTERRUPCION INT0
		ldi r18,0b11100000 ; Primero borramos las interrupciones memorizadas
		ldi r26,$5A ;hasta el momento.
		st x,r18  ;ACTUAMOS SOBRE GIFR
		ldi r18,0b01000000 ;
		ldi r26,$5b ;Prepara la direccion de memoria donde mandaremos el contenido de r18
			st x,r18  ;ACTUAMOS SOBRE GICR
			ldi r18,0b00000000 ;
			ldi r26,$55 ;Prepara la direccion de memoria donde mandaremos el contenido de r18
			st x,r18  ;ACTUAMOS SOBRE MCUCR para determinar condiciones de disparo de INT0
			sei				;Actuamos sobre el bit I del SREG para habilitar todas las interrupciones
configuradas.
			rjmp monitoriza_activacion
```

```
saltaaProg2:
                        ;rcall resta_pasosM2
                        ldi r19,145 ;cargamos 400 pasos y mandamos M2 decha.
                        ldi r24,1
                        push r24
                        push r19
                        rcall Prog2

                        ldi r18,$02 ;;restauramos la pila
                        ldi r26,$5e ;prepara la direccion de memoria donde comienza la pila
                        st x,r18  ;para manejo de subrutinas.
                        ldi r18,$5c ;
                        ldi r26,$5d ;prepara la direccion de memoria donde comienza la pila
                        st x,r18  ;para manejo de subrutinas.

                        ;HABILITAMOS LA INTERRUPCION INT0
                        ldi r18,0b11100000 ; Primero borramos las interrupciones memorizadas
                        ldi r26,$5A ;hasta el momento.
                        st x,r18  ;ACTUAMOS SOBRE GIFR

                        ldi r18,0b01000000 ;
                        ldi r26,$5b ;Prepara la direccion de memoria donde mandaremos el contenido de r18
                        st x,r18  ;ACTUAMOS SOBRE GICR
                        ldi r18,0b00000000 ;
                        ldi r26,$55 ;Prepara la direccion de memoria donde mandaremos el contenido de r18
                        st x,r18  ;ACTUAMOS SOBRE MCUCR para determinar condiciones de disparo de INT0
                        sei                     ;Actuamos sobre el bit I del SREG para habilitar todas las interrupciones
configuradas.
                        rjmp monitoriza_activacion

saltaaProg3:

                        ;rcall resta_pasosM1
                        ldi r19,201 ;cargamos 200 pasos y mandamos M1 decha.
                        ldi r24,0
                        push r24
                        push r19
                        rcall Prog4

                        ldi r18,$02 ;;restauramos la pila
                        ldi r26,$5e ;prepara la direccion de memoria donde comienza la pila
                        st x,r18  ;para manejo de subrutinas.
                        ldi r18,$5c ;
                        ldi r26,$5d ;prepara la direccion de memoria donde comienza la pila
                        st x,r18  ;para manejo de subrutinas.
;HABILITAMOS LA INTERRUPCION INT0
                        ldi r18,0b11100000 ; Primero borramos las interrupciones memorizadas
                        ldi r26,$5A ;hasta el momento.
                        st x,r18  ;ACTUAMOS SOBRE GIFR
                        ldi r18,0b01000000 ;
                        ldi r26,$5b ;Prepara la direccion de memoria donde mandaremos el contenido de r18
                        st x,r18  ;ACTUAMOS SOBRE GICR
                        ldi r18,0b00000000 ;
                        ldi r26,$55 ;Prepara la direccion de memoria donde mandaremos el contenido de r18
                        st x,r18  ;ACTUAMOS SOBRE MCUCR para determinar condiciones de disparo de INT0
                        sei                     ;Actuamos sobre el bit I del SREG para habilitar todas las interrupciones
configuradas.
                        rjmp monitoriza_activacion

saltaaProg4:
```

```
                        ;rcall resta_pasosM1
                        ldi r19,201 ;cargamos 200 pasos y mandamos M1 izqda.
                        ldi r24,0
                        push r24
                        push r19
                        rcall Prog3

                        ldi r18,$02 ;restauramos la pila
                        ldi r26,$5e ;prepara la direccion de memoria donde comienza la pila
                        st x,r18  ;para manejo de subrutinas.
                        ldi r18,$5c ;
                        ldi r26,$5d ;prepara la direccion de memoria donde comienza la pila
                        st x,r18  ;para manejo de subrutinas.

                ;HABILITAMOS LA INTERRUPCION INT0
                        ldi r18,0b11100000 ; Primero borramos las interrupciones memorizadas
                        ldi r26,$5A ;hasta el momento.
                        st x,r18  ;ACTUAMOS SOBRE GIFR

                        ldi r18,0b01000000 ;
                        ldi r26,$5b ;Prepara la direccion de memoria donde mandaremos el contenido de r18
                        st x,r18  ;ACTUAMOS SOBRE GICR
                        ldi r18,0b00000000 ;
                        ldi r26,$55 ;Prepara la direccion de memoria donde mandaremos el contenido de r18
                        st x,r18  ;ACTUAMOS SOBRE MCUCR para determinar condiciones de disparo de INT0
                        sei                     ;Actuamos sobre el bit I del SREG para habilitar todas las interrupciones
configuradas.
                        rjmp monitoriza_activacion

captura_codigos:

                        cbi $10,7;  activamos BUSY para que detecte el código.

                        sbic $16,5 ;si se detecta 1 lógico en pin6 del micro tenemos señal RESET
                        rjmp pon_uno_PB1; se manda poner a cero PC0
                        rjmp pon_cero_PB1; se manda poner a uno PC0

pin19:
                        sbic $10,5 ;si se detecta 1 lógico en pin 19 del micro tenemos señal INIT
                        rjmp pon_uno_PB2; se manda poner a cero PC1
                        rjmp pon_cero_PB2; se manda poner a uno PC1

pin20:
                        sbic $10,6 ;si se detecta 1 lógico en pin 20 del micro tenemos señal SELECT
                        rjmp pon_uno_PB3; se manda poner a cero PC2
                        rjmp pon_cero_PB3; se manda poner a uno PC2

                        rjmp captura_codigos

pon_uno_PB1:
                        sbi $18,1 ;se manda poner a uno PC0
                        rjmp pin19

pon_uno_PB2:
                        sbi $18,2 ;se manda poner a uno PC1
                rjmp pin20

pon_uno_PB3:
                sbi $18,3 ;se manda poner a uno PC2
                rjmp captura_codigos

pon_cero_PB1:
```

```
                cbi $18,1 ;se manda poner a cero PC0
                rjmp pin19

pon_cero_PB2:
                cbi $18,2 ;se manda poner a cero PC1
                rjmp pin20

pon_cero_PB3:
                cbi $18,3 ;se manda poner a cero PC2
                rjmp captura_codigos

resta_pasosM1:

                dec r11
                brbs 1, byte14
                ret

byte14:                                 ;decrementa el byte 1 hasta que llegue al final y pase al byte2
                dec r12
                brbs 1, byte24
                ret
byte24:
                dec r13

                ret

resta_pasosM2:
                dec r11
                brbs 1, byte15
                ret
byte15:                                 ;decrementa el byte 1 hasta que llegue al final y pase al byte2
                dec r12
                brbs 1, byte25
                ret
byte25:
                dec r13

                ret

SWAP_BYTE: ;esta función invierte el orden de los 8 bits de r3 volcándolos en r9

                ldi r17,0 ; ponemos a cero el registro.
                mov r9,r17

                sbrs r3,0
                rjmp uno_r9_7
                inc r9 ;si está a 1 el bit 0 de r3 pone a 1 el bit 7 de r9
                lsl r9

                sbrs r3,0
uno_r9_7:
                        lsl r9 ; si está a 0 el bit 0 de r3 pone a 0 el bit 7 de r9

                        sbrs r3,1
                        rjmp uno_r9_6
                        inc r9 ;si está a 1 el bit 1 de r3 pone a 1 el bit 6 de r9
                        lsl r9

                        sbrs r3,1
uno_r9_6:
                        lsl r9 ; si está a 0 el bit 1 de r3 pone a 0 el bit 6 de r9
```

```
                        sbrs r3,2
                        rjmp uno_r9_5
                        inc r9 ;si está a 1 el bit 2 de r3 pone a 1 el bit 5 de r9
                        lsl r9

                        sbrs r3,2
uno_r9_5:
                        lsl r9 ; si está a 0 el bit 2 de r3 pone a 0 el bit 5 de r9

                        sbrs r3,3
                        rjmp uno_r9_4
                        inc r9 ;si está a 1 el bit 3 de r3 pone a 1 el bit 4 de r9
                        lsl r9

                        sbrs r3,3
uno_r9_4:
                        lsl r9 ; si está a 0 el bit 3 de r3 pone a 0 el bit 4 de r9

                        sbrs r3,4
                        rjmp uno_r9_3
                        inc r9 ;si está a 1 el bit 4 de r3 pone a 1 el bit 3 de r9
                        lsl r9

                        sbrs r3,4
uno_r9_3:
                        lsl r9 ; si está a 0 el bit 4 de r3 pone a 0 el bit 3 de r9

                        sbrs r3,5
                        rjmp uno_r9_2
                        inc r9 ;si está a 1 el bit 5 de r3 pone a 1 el bit 2 de r9
                        lsl r9

                        sbrs r3,5
uno_r9_2:
                        lsl r9 ; si está a 0 el bit 5 de r3 pone a 0 el bit 2 de r9

                        sbrs r3,6
                        rjmp uno_r9_1
                        inc r9 ;si está a 1 el bit 6 de r3 pone a 1 el bit 1 de r9
                        lsl r9

                        sbrs r3,6
uno_r9_1:
                        lsl r9 ; si está a 0 el bit 6 de r3 pone a 0 el bit 1 de r9

                        sbrs r3,7
                        rjmp uno_r9_0
                        inc r9 ;si está a 1 el bit 7 de r3 pone a 1 el bit 0 de r9

uno_r9_0:
                        ; si está a 0 el bit 7 de r3 pone a 0 el bit 0 de r9

                        ret

;*****************************************************************************************************************
;*****************************************************************************************************************
;****************** --PROYECTO FIN DE CARRERA : POSICIONADOR X Y --                    ***********
;*****************************************************************************************************************
;*****************************************************************************************
;************************************ LIBRERIA DE SEÑALES DE MANEJO DE LOS MOTORES   *****************
;********* ***************************************************************************************
;*****************************************************************************************************************
```

```
;
;          -LISTADO DE RUTINAS-
;
;
;Mov_M1_dcha:
;Mov_M1_izqda:    usamos r19 para determinar el número de pasos que queremos que de el motor.
;Mov_M2_dcha:
;Mov_M2_izqda:
;
;
;*****************************************************************************************************************
;*****************************************************************************************************************
;                    --- RUTINAS DE MOVIMIENTO DE LOS MOTORES --
;*****************************************************************************************************************
;*****************************************************************************************************************

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Mov_M1_dcha:

M1dcha_comienzo:

                                        cpi r19,0 ;comparamos LSByte para saber si ha llegado a cero.

                                        breq MSbyte_cont ;si el contador alcanza saltamos a rutina de comprobación
                                        ;                                                    del contador del byte mas significativo.

                                        cpi r25,0;tanto si es 10000000 como si es 0 recolocamos el bit

                                        breq M1dcha_retorno_carro;tanto si es 10000000 como si es 0 recolocamos el bit

                                                ;INTRODUCIMOS AQUÍ UNA ESPERA

M1dcha_regreso:                         mov r18,r8 ;arrancamos timer0 a frecuencia CK
                                        ldi r26,$53 ;arrancamos timer0 a frecuencia CK
                                        st x,r18 ;arrancamos timer0 a frecuencia CK

espera1:                                cpi r18,33;el timer cargará el valor 33 en r18 para provocar la salida
                                        brne espera1;mientras r18 no sea 33 cierra el bucle.

                                        out $1b,r25 ;mandamos al puerto.

                                        mov r18,r1;comprobamos el estado de la bandera de la cuenta de pasos.
                                        cpi r18,1;si venimos de una interrupción no contamos pasos.
                                        breq regresa

                                        inc r11;incrementamos la cuenta de los pasos.
                                        brbs 1,byte1;si termina la cuenta del byte 0 seguimos con el byte1

regresa:                                clc ; pone a 0 el bit de carry para que no entre en la rotación de la instrucción ror

                                        ror r25 ;movemos el 1 lógico un bit a la izqda.

                                        dec r19; decrementamos el contador LSByte.

                                        rjmp M1dcha_comienzo ;cerramos bucle.

M1dcha_salida:                  ;DEVUELVE EL CONTROL

                                        cpi r25,0;recuperamos la última rotación innecesaria.
                                        breq ponUno
```

```
                                rol r25;recuperamos la última rotación innecesaria.
                                ldi r18,0
                                ret

ponUno:
                                ldi r25,1;recuperamos la última rotación innecesaria.
                                ret

M1dcha_retorno_carro:   ;HACE QUE VUELVA A EMPEZAR LA ROTACIÓN DE BITS

                                ldi r25,8 ;cargamos la señal a enviar.

                                rjmp M1dcha_regreso

MSbyte_cont:

                                cpi r24,0 ;comparamos para saber si ha llegado a cero.

                                breq M1dcha_salida ;si el contador alcanza el límite sale de la rutina.

                                ldi r19,255 ;reseteamos contador pasos LSByte
                                dec r24;descontamos en el MSByte.

                                rjmp M1dcha_comienzo   ;volvemos a contar 255 pasos.



byte1:                                  ;incrementa el byte 2 hasta que llegue al final y pase al byte2
                                inc r12
                                brbs 1, byte2
                                rjmp regresa
byte2:
                                inc r13
                                rjmp regresa

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


        Mov_M1_izqda:

        M1izqda_comienzo:

                                cpi r19,0 ;comparamos LSByte para saber si ha llegado a cero.

                                breq MSbyte_cont1;si el contador alcanza saltamos a rutina de comprobación
                                ;                                               del contador del byte mas significativo.


                                cpi r25,16 ; si es 16 recolocamos el bit perdido

                                breq M1izqda_retorno_carro;tanto si es 0 o 1 recolocamos el bit perdido

                                cpi r25,0 ; si es 16 recolocamos el bit perdido

                                breq M1izqda_retorno_carro;tanto si es 0 o 1 recolocamos el bit perdido

M1izqda_regreso:
                                ;INTRODUCIMOS AQUÍ UNA ESPERA

                                mov r18,r8 ;arrancamos timer0 a frecuencia CK
                                ldi r26,$53 ;arrancamos timer0 a frecuencia CK
```

```
                                st x,r18 ;arrancamos timer0 a frecuencia CK

espera2:                        cpi r18,33;el timer cargará el valor 33 en r18 para provocar la salida
                                brne espera2;mientras r18 no sea 33 cierra el bucle.

                                out $1b,r25 ;mandamos al puerto.

                                mov r18,r1;comprobamos el estado de la bandera de la cuenta de pasos.
                                cpi r18,1;si venimos de una interrupción no contamos pasos.
                                breq regresa1

                                inc r11;incrementamos la cuenta de los pasos.
                                brbs 1,byte11;si termina la cuenta del byte 0 seguimos con el byte1

regresa1:                       clc ; pone a 0 el bit de carry para que no entre en la rotación de la instrucción rol

                                rol r25 ;movemos el 1 lógico un bit a la izqda.

                                dec r19; decrementamos el contador LSByte.

                                rjmp M1izqda_comienzo ;cerramos bucle.

M1izqda_salida:         ;DEVUELVE EL CONTROL

                                cpi r25,0;recuperamos la última rotación innecesaria.
                                breq ponOcho

                                ror r25;recuperamos la última rotación innecesaria.
                                ldi r18,0
                                ret

ponOcho:
                                ldi r25,8;recuperamos la última rotación innecesaria.
                                ret

M1izqda_retorno_carro:  ;HACE QUE VUELVA A EMPEZAR LA ROTACIÓN DE BITS

                                ldi r25,1 ;cargamos la señal a enviar.

                                rjmp M1izqda_regreso

MSbyte_cont1:
                                cpi r24,0 ;comparamos para saber si ha llegado a cero.

                                breq M1izqda_salida ;si el contador alcanza el límite sale de la rutina.

                                ldi r19,255 ;reseteamos contador pasos LSByte
                                dec r24

                                rjmp M1izqda_comienzo   ;volvemos a contar 255 pasos.

byte11:                                 ;incrementa el byte 2 hasta que llegue al final y pase al byte2
                                inc r12
                                brbs 1, byte21
                                rjmp regresa1
byte21:
                                inc r13
                                rjmp regresa1

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
Mov_M2_dcha:
dcha_comienzo:

                        cpi r19,0 ;comparamos LSByte para saber si ha llegado a cero.

                        breq MSbyte_cont2 ;si el contador alcanza saltamos a rutina de comprobación
                        ;                                                   del contador del byte mas significativo.

                        cpi r28,8 ;si el bit llega a la dcha de los 4 bits
                                                ;se hace un retorno de carro.
                        breq M2dcha_retorno_carro

                        cpi r28,0; si r28 es todo ceros comenzamos con el primer bit de la izqda.
                        breq M2dcha_retorno_carro

M2dcha_regreso:
                        ;INTRODUCIMOS AQUÍ UNA ESPERA

                        mov r18,r8 ;arrancamos timer0 a frecuencia CK
                        ldi r26,$53 ;arrancamos timer0 a frecuencia CK
                        st x,r18 ;arrancamos timer0 a frecuencia CK


espera3:                cpi r18,33;el timer cargará el valor 33 en r18 para provocar la salida
                        brne espera3;mientras r18 no sea 33 cierra el bucle.

                        out $1b,r28 ;mandamos al puerto.

                        mov r18,r1;comprobamos el estado de la bandera de la cuenta de pasos.
                        cpi r18,1;si venimos de una interrupción no contamos pasos.
                        breq regresa2

                        inc r11;incrementamos la cuenta de los pasos.
                        brbs 1,byte12;si termina la cuenta del byte 0 seguimos con el byte1

regresa2:               clc ; pone a 0 el bit de carry para que no entre en la rotación de la instrucción ror

                        ror r28 ;movemos el 1 lógico un bit a la izqda.

                        dec r19; decrementamos el contador LSByte.

                        rjmp M2dcha_comienzo ;cerramos bucle.

M2dcha_salida:  ;DEVUELVE EL CONTROL

                        rol r28;recuperamos la última rotación innecesaria.
                        ldi r18,0
                        ret

M2dcha_retorno_carro:   ;HACE QUE VUELVA A EMPEZAR LA ROTACIÓN DE BITS

                        ldi r28,$80 ;cargamos la señal a enviar.

                        rjmp M2dcha_regreso

MSbyte_cont2:
                        cpi r24,0 ;comparamos para saber si ha llegado a cero.

                        breq M2dcha_salida ;si el contador alcanza el límite sale de la rutina.

                        ldi r19,255 ;reseteamos contador pasos LSByte
```

```
                        dec r24

                        rjmp M2dcha_comienzo   ;volvemos a contar 255 pasos.

byte12:                 ;incrementa el byte 2 hasta que llegue al final y pase al byte2
                        inc r12
                        brbs 1, byte22
                        rjmp regresa2
byte22:
                        inc r13
                        rjmp regresa2

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


Mov_M2_izqda:

M2izqda_comienzo:

                        cpi r19,0 ;comparamos LSByte para saber si ha llegado a cero.

                        breq MSbyte_cont3 ;si el contador alcanza saltamos a rutina de comprobación
                        ;                                       del contador del byte mas significativo.

                        cpi r28,1 ;si es 1 recolocamos el bit

                        breq M2izqda_retorno_carro ;si el contador alcanza el límite sale de la rutina.
                                                                ;para reiniciar la rotación de bits.
                        cpi r28,0 ;si es 0 recolocamos el bit

                        breq M2izqda_retorno_carro ;si el contador alcanza el límite sale de la rutina.
                                                                ;para reiniciar la rotación de bits.
                        cpi r28,8 ;si es 8 recolocamos el bit

                        breq M2izqda_retorno_carro ;si el contador alcanza el límite sale de la rutina.
                                                                ;para reiniciar la rotación de bits.

M2izqda_regreso:

                        ;INTRODUCIMOS AQUÍ UNA ESPERA

                        mov r18,r8 ;arrancamos timer0 a frecuencia CK
                        ldi r26,$53 ;arrancamos timer0 a frecuencia CK
                        st x,r18 ;arrancamos timer0 a frecuencia CK


espera4:                cpi r18,33;el timer cargará el valor 33 en r18 para provocar la salida
                        brne espera4;mientras r18 no sea 33 cierra el bucle.

                        out $1b,r28 ;mandamos al puerto.

                        mov r18,r1;comprobamos el estado de la bandera de la cuenta de pasos.
                        cpi r18,1;si venimos de una interrupción no contamos pasos.
                        breq regresa3

                        inc r11;incrementamos la cuenta de los pasos.
                        brbs 1,byte13;si termina la cuenta del byte 0 seguimos con el byte1

regresa3:               clc ; pone a 0 el bit de carry para que no entre en la rotación de la instrucción rol

                        rol r28 ;movemos el 1 lógico un bit a la izqda.
```

```
                                    dec r19; decrementamos el contador LSByte.
                                    rjmp M2izqda_comienzo ;cerramos bucle.

M2izqda_salida:    ;DEVUELVE EL CONTROL

                                    cpi r28,0
                                    breq ponOcho1

                                    ror r28;recuperamos la última rotación innecesaria.
                                    ldi r18,0
                                    ret

ponOcho1:
                                    ldi r28,128
                                    ret

M2izqda_retorno_carro: ;HACE QUE VUELVA A EMPEZAR LA ROTACIÓN DE BITS

                                    ldi r28,16 ;cargamos la señal a enviar.

                                    rjmp M2izqda_regreso

MSbyte_cont3:
                                    cpi r24,0 ;comparamos para saber si ha llegado a cero.

                                    breq M2izqda_salida ;si el contador alcanza el límite sale de la rutina.

                                    ldi r19,255 ;reseteamos contador pasos LSByte
                                    dec r24

                                    rjmp M2izqda_comienzo   ;volvemos a contar 255 pasos.

byte13:                                     ;incrementa el byte 2 hasta que llegue al final y pase al byte2
                                    inc r12
                                    brbs 1, byte23
                                    rjmp regresa3
byte23:
                                    inc r13

                                    rjmp regresa3

;*****************************************************************************************************************
;*****************************************************************************************************************
;                    --- RUTINAS DE MOVIMIENTO DE LOS MOTORES --
;*****************************************************************************************************************
;*****************************************************************************************************
```

ANEXO IV

Hojas de características.