Capítulo 5

Software de control

5.1 Introducción

Recapitulando todo lo desarrollado hasta ahora, el robot tiene una estructura mecánica y una red electrónica de control, pero todavía necesita algo más para ser operativo. En concreto, necesita un programa de control que sea capaz de gestionar la electrónica para producir el movimiento correcto de los motores y con ello hacer que se mueva. Debido a la estructura modular y a la complejidad de realizar todo el programa en un sólo bloque ha sido necesario subdividirlo. Esto se ha realizado en concordancia con la red de control, es decir habrá un programa ejecutándose en los módulos esclavos, otro en el maestro y por último uno en el PC. El resultado final ha sido el siguiente:

1. Un programa servidor para los módulos esclavos. Será el mismo para todos excepto por una serie de parámetros que identifican al módulo en concreto. El servidor se encarga de recibir las tramas de datos procedentes del módulo maestro, las analiza y mueve los motores asociados a él de acuerdo con la orden que ha recibido. El código se ha escrito en ensamblador del 68hc11.

2. Un programa cliente para el módulo maestro. Este programa tiene dos funciones, la primera consiste en hacer de puente entre el PC y la red de control, es decir los módulos esclavos. La función de puente permite manejar la red de control desde el PC. Mediante la segunda, envía las tramas necesarias para mover al robot correctamente e independientemente del PC, en este caso el robot se comporta de forma autónoma. El código ha sido desarrollado en ensamblador del 68hc11.

3. Programas en el PC que van a servir de ayuda para buscar las secuencias de movimiento correctas. Estos programas están condicionados a la utilización del módulo maestro como puente. El código se ha escrito en un lenguaje de alto nivel, en concreto el C y además se han utilizado herramientas de programación visual.

Los programas anteriores no son independientes, como ya se ha dicho, están relacionados entre sí por medio de la red de control. Es necesario que entre ellos fluya una comunicación para que cada módulo sea consciente de cuándo y en qué forma debe intervenir. Dicha comunicación se tiene que hacer respetando unas normas, lo que se consigue utilizando un protocolo común. Aunque éste no es un programa, se puede considerar como

parte integrante de todos ellos y en tal caso se establece como nivel inferior del software. Es decir, primero se detallará el protocolo, ya que todos los demás programas tienen que implementarlo, luego se realizará el programa servidor de los módulos esclavos, para finalmente desarrollar el programa maestro y los del PC.

5.2 Protocolo de control

Los diferentes módulos del robot están conectados entre sí mediante una red sobre la cual se ha implementado un protocolo que mantiene el orden en las comunicaciones. Para desarrollarlo se ha partido de la constitución física de la red, que como ya se vio en el capítulo anterior, se trata de una serie síncrona (SPI) formada por las señales:

1. **MOSI Master Out Slave In**, es decir salida de datos del módulo maestro y entrada de datos del módulo esclavo. Ya viendo esto se puede decir que únicamente habrá comunicaciones unidireccionales: *del módulo maestro a los módulos esclavos*.

2. **#SS Slave Select**, es decir selección del módulo esclavo. Esta señal la genera el módulo maestro y sirve para avisar a un módulo esclavo. Como es la misma señal para todos quiere decir que cuando el maestro quiera mandar datos todos los esclavos recibirán la señal de aviso y todos ellos se pondrán a escuchar. Por lo tanto será la parte software del protocolo la que proporcione algún mecanismo que indique a qué módulo esclavo se dirige el resto de la información.

3. **SCK Slave Clock**, es una señal de reloj que genera el módulo maestro y que sirve para sincronizar la lectura de datos por parte de los módulos esclavos. Al ser la misma señal para todos, estarán sincronizados a la vez.

Con las señales anteriores se realiza una parte del protocolo, la otra se hará mediante software. Lo que se necesita al final es tener un mecanismo que permita mandar tramas desde el módulo maestro a cada uno de los módulos esclavos. En esas tramas se introducirán las órdenes del tipo: Pon el motor X en la posición Y. La señal MOSI establece el canal de datos que va desde el maestro hasta el esclavo, la señal SCK proporciona el sincronismo, pero ¿ quién indica a que módulo esclavo va dirigida la información?. La respuesta es que ninguna, ya que #SS es una señal que manda el maestro para indicar que está dispuesto a enviar datos, pero la dirige a todos los módulos esclavos, por lo que a partir de ese momento todos ellos deberán empezar a leer el mensaje. Pero entoces, ¿cómo se puede hacer para que cada uno de ellos reciba información única?. La respuesta es introduciendo en la primera trama de datos un identificador de módulo. Al recibirlo cada esclavo lo comparará con el suyo propio y en caso de que coincidan seguirá recibiendo los datos, en caso contrario los desestimará. Por eso el software juega un papel importante ya que una parte de él está dedicado a completar el protocolo.

Con todo lo anterior se obtiene un mecanismo de comunicación y de selección, ahora hay que completar el protocolo definiendo órdenes que aporten información útil para el robot. En concreto se establecen las siguientes:

Posicionar_motor: Esta orden está formada por dos bytes, el primero identifica el número del motor a mover. Los valores válidos para este byte son en decimal 1, 2, 3 y 4. El

otro byte contiene un valor entre 0 y 255 que indica la posición en la que se quiere dejar al motor.

Servicio_activa: Esta orden está formada por un byte del cual sus cuatro primeros bits forman la máscara de activación para cada uno de los cuatro motores. De esta manera para activar el motor 1 hay que poner el bit 0 a nivel alto. Para desactivarlo hay que poner el bit 0 a nivel bajo. Haciendo lo mismo con el bit 1,2 y 3 se consigue activar o desactivar los motores 2, 3 y 4 respectivamente.

Cambia_led: Las BT6811, es decir los módulos maestros, tienen un LED verde que se puede utilizar como indicador. Para poder tener acceso a él desde la red se ha considerado esta orden, la cual no tiene más bytes ya que cada vez que se recibe se cambia el estado del LED.

Salida_puertoC: Al igual que antes, las BT6811 tienen un puerto de salida que puede ser utilizado para poner barras de LEDs u otros dispositivos. Para poder tener acceso a dicho puerto se establece esta orden que necesita un byte de datos. Dicho byte será el que se envíe al puerto C de la BT6811.

Una vez vistas las órdenes, se tienen las tramas completamente definidas (ver figura 5.1). El primer byte indicará a qué módulo esclavo se dirige la información, el segundo indicará la orden transmitida y el resto contendrán los datos necesarios para ejecutar la orden correctamente. Todas las tramas tendrán la misma longitud, 4 bytes, en el caso de órdenes que no necesiten los cuatro habrá que introducir bytes de relleno. El valor de estos da igual pues serán ignorados. Un dato de interés es que, como para identificar a cada módulo esclavo se utiliza un solo byte (ID esclavo), el número máximo de módulos esclavos que podrá haber es de 256. En caso de querer introducir más, habrá que cambiar la longitud de las tramas y añadir un byte más en el identificador. Por ejemplo con dos bytes se podrían tener 65536 módulos esclavos, un número bastante superior al anterior.

5.3 Software de los módulos esclavos

Aunque en el robot se han conectado sólo tres módulos esclavos, la red de control ha sido diseñada para poder soportar hasta 256. En todos ellos hay un programa con dos funciones, la primera consiste en escuchar las tramas de la red y la segunda en mover los servomotores. Al realizar todos los módulos esclavos la misma función no es necesario desarrollar un software específico para cada uno de ellos, con realizar uno general es suficiente.

A medida que se programa, hay que ir haciendo una serie de cálculos relacionados con los tiempos de consumo de CPU. Para entender esto es necesario comprender la filosofía del programa. Como ya se ha dicho, hay una parte que se encarga de mover los servomotores y otra que se encarga de recibir las tramas. De éstas, la primera tiene que generar unas señales de PWM y la segunda estar continuamente atenta al puerto SPI. Se empiezan a ver problemas de ocupación, ¿primero se atiende al SPI o a producir las señales?. La solución adoptada es la siguiente: se utilizarán las interrupciones de los comparadores para generar el PWM de forma independiente al bucle principal del programa y por contra, éste estará dedicado exclusivamente a recibir e interpretar las tramas.

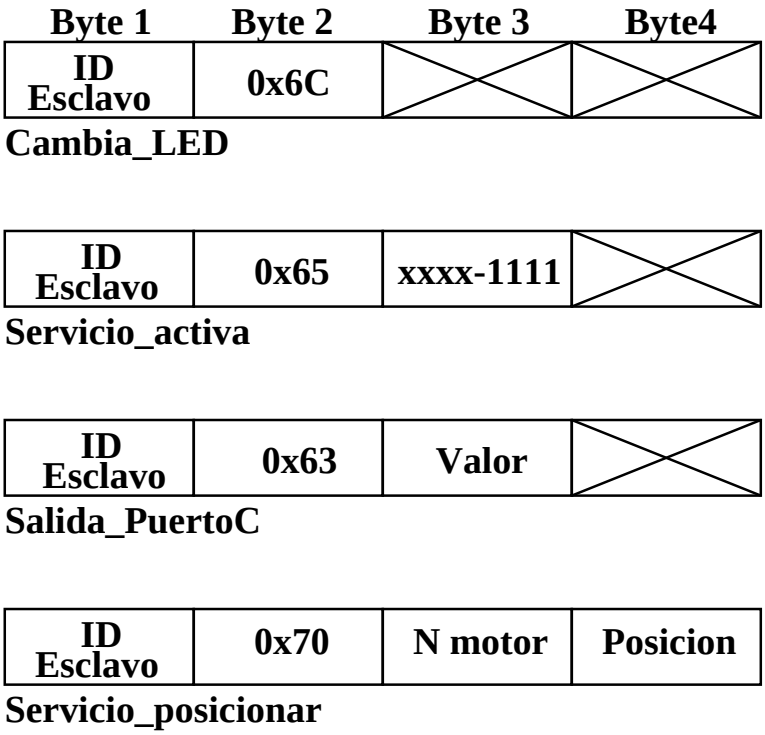

Figura 5.1: Tramas de control.

Ahora hay que plantearse la siguiente pregunta: ¿De cuánto tiempo dispondrá la CPU para ejecutar el bucle principal?, ¿estará totalmente ocupada atendiendo a las interrupciones?. La respuesta es que la CPU apenas estará ocupada por las interrupciones, es decir estará prácticamente todo el tiempo esperando a recibir los datos para interpretarlos. Esto se obtiene haciendo un análisis del código del programa, proceso que se explica en el apartado 5.3.4, tratando previamente la explicación del código en sí.

5.3.1 Configuración inicial del 68hc11

Como siempre antes de comenzar con los bucles principales de los programas es necesario configurar algunos recursos internos del microcontrolador y otros externos como las variables. A continuación se enumeran los puntos más importantes de la configuración del programa esclavo, cuyo código puede encontrarse en el apartado 5.3.5.

1. Se asigna a la **etiqueta M_ES** el valor 'a', 'b' o 'c', este valor identifica al módulo esclavo a la hora de recibir las tramas. Esta etiqueta se encuentra al principio del programa en la zona de declaraciones titulada: *Identificación del módulo esclavo*.

2. Se asigna el valor **$F800** a la **dirección de inicio** del programa. Es decir se deja grabado en los 2Kbytes de EEPROM interna disponible en un microcontrolador 68hc811E2. La razón de utilizar éste en lugar del 68hc11A1 es el poder utilizar el modo *single_chip* para arrancar el microcontrolador sin necesidad de cortocircuitar el puerto serie. Además, debido a la mayor capacidad de memoria, queda suficiente para ampliar el servidor con otras muchas funciones.

3. El **puntero de pila** se inicializa con **$FF** y el **registro de dirección X** a **$1000**. Lo primero es obligatorio para poder ejecutar rutinas de interrupción y subrutinas. Lo segundo es más una ayuda que permite utilizar instrucciones especiales como son BCLR/BRCLE, BSET/BRSET.

4. Se configura el **SPI** como **esclavo**. A partir de ahora la señal **SS** indica cuándo se pueden recibir las tramas, esa señal la controla el módulo maestro y es común para toda la red. Los datos que llegan por la red se reciben por la línea **MOSI**.

5. El **puerto C** se configura **como salida** ya que en algunos módulos se va a conectar barras de leds.

6. Se **configuran los comparadores** del microcontrolador. Estos son indispensables para controlar las señales de PWM de los servomotores. Se utilizan sin salida hardware y desde el principio sólo se activa el encargado de generar el periodo de la señal.

7. Por último se **configuran las posiciones de los motores y su estado de activación**. De forma predeterminada se establece que los motores estén desactivados y la posición asignada sea una cualquiera. Antes de activar los motores, el módulo maestro se tendrá que preocupar de actualizar las posiciones a unas correctas.

5.3.2 Primera parte: Recepción de tramas

El bucle principal del programa está continuamente esperando a recibir tramas por el SPI. En cuanto recibe el primer byte de una trama lo compara con el identificador **M_ES** y en caso de ser igual la sigue interpretando. En caso contrario, a medida que reciba el resto la irá ignorando. Se ha dicho que todos los módulos esclavos llevan el mismo código excepto por un byte. Éste es en concreto el depositado en **M_ES** y su valor es el siguiente:

- El *módulo esclavo A,* que controla los servomecanismos A1, A2, A3 y A4, tendrá **M_ES** igual a '**a**'.
- El *módulo esclavo B,* que controla los servomecanismos B1, B2, B3 y B4, tendrá **M_ES** igual a '**b**'.
- El *módulo esclavo C,* que controla los servomecanismos C1, C2, C3 y C4, tendrá **M_ES** igual a '**c**'.

El protocolo que se detalló en la sección anterior es interpretado por esta parte del programa directamente, es decir sin considerar posibles pérdidas de tramas e introducción de errores de transmisión. Esta suposición es muy arriesgada y no es recomendable, por lo menos habría que introducir algún sistema de seguridad para que, en caso de perder alguna trama, sea posible volver a sincronizarse. Sin embargo, debido a la necesidad de empezar a probar el robot, se optó por abordar la versión más simple y en caso de fallos ir mejorándola hasta obtener el rendimiento deseado. Por suerte, los resultados fueron bastante buenos, tanto es así que en ningún momento se apreció falta de sincronización o inclusión de errores, por lo que se dejó el protocolo en su estado original, es decir sin la detección de errores.

El diagrama de bloques del algoritmo de lectura se puede ver en la figura 5.2. Para su correcta interpretación hay que considerar que todas las tramas son de cuatro bytes y que la rutina *Recibir_SPI* se queda esperando hasta que se reciba un byte por el puerto SPI. Con esas dos consideraciones es fácil seguir el grafo:

1. Se espera a recibir un byte por el SPI, que será el primero de una trama.
2. Se compara ese byte con la etiqueta M_ES. Si no son iguales se leen los tres bytes siguientes de la trama y se ignoran, volviendo así al principio del bucle que vuelve a ser la espera del comienzo de una nueva trama. Si la comparación es igual seguimos adelante.
3. Si en la comparación anterior son iguales entonces se lee un nuevo byte y se compara con las distintas opciones 'l', 'e', 'c' y 'p'. Forzosamente alguna de ellas tendrá que ser buena y se ejecutará así la subrutina correspondiente. En caso contrario se devuelve el control al principio del programa, pero ya se habrá perdido el sincronismo y probablemente habrá que reiniciar el sistema. Las rutinas son las siguientes:
   (a) La subrutina *Cambia_LED* cambia el estado del LED de la BT6811 y desecha los dos siguientes bytes de la trama.
   (b) La subrutina *Servicio_activa* espera a recibir un nuevo byte que indica qué servomecanismos estarán activos y cuáles no. Luego desecha el último byte de la trama y vuelve al comienzo.
   (c) La subrutina *Salida_PuertoC* espera a recibir un nuevo byte qué indica que bits del puerto C se activarán y cuales no. Luego desecha el último byte de la trama y vuelve al comienzo.
   (d) Por último la subrutina *Servicio_posicionar* recibe un primer byte que indica el ńumero de motor que se quiere mover y luego recibe otro que indica la posición.

En el punto 3 se ha mencionado la posibilidad de perder el sincronismo por recibir el segundo byte erróneo, esto ratifica lo que se había mencionado antes sobre la vulnerabilidad del programa. Analizando la situación, una vez que el programa pierde una trama se descontrola pudiendo producirse una recuperación o no, aunque lo más normal es que no la recupere. Si es fácil que falle, también será fácil que se recupere y si es difícil que falle lo será también para recuperarse. Durante las pruebas realizas con la red no ocurrieron pérdidas de sincronismo, lo cual permite seguir adelante con este software sin necesidad de añadir mecanismos de seguridad. La única condición que hay que cumplir es que a la hora de poner en marcha el robot, el módulo maestro debe empezar a transmitir una vez que los módulos esclavos estén listos para recibir en el principio del programa. Situación que ocurre si primero se pulsa el reset de los módulos esclavos y luego el del módulo maestro.

5.3.3 Segunda parte: Control de los servomecanismos

Los servomecanismos se controlan mediante señales de PWM (figura 3.4, página 47) de periodo 20 mseg. En función del ancho del pulso el eje se situará en cualquier ángulo comprendido entre 0 y 180 grados aproximadamente. Cada módulo esclavo controla cuatro

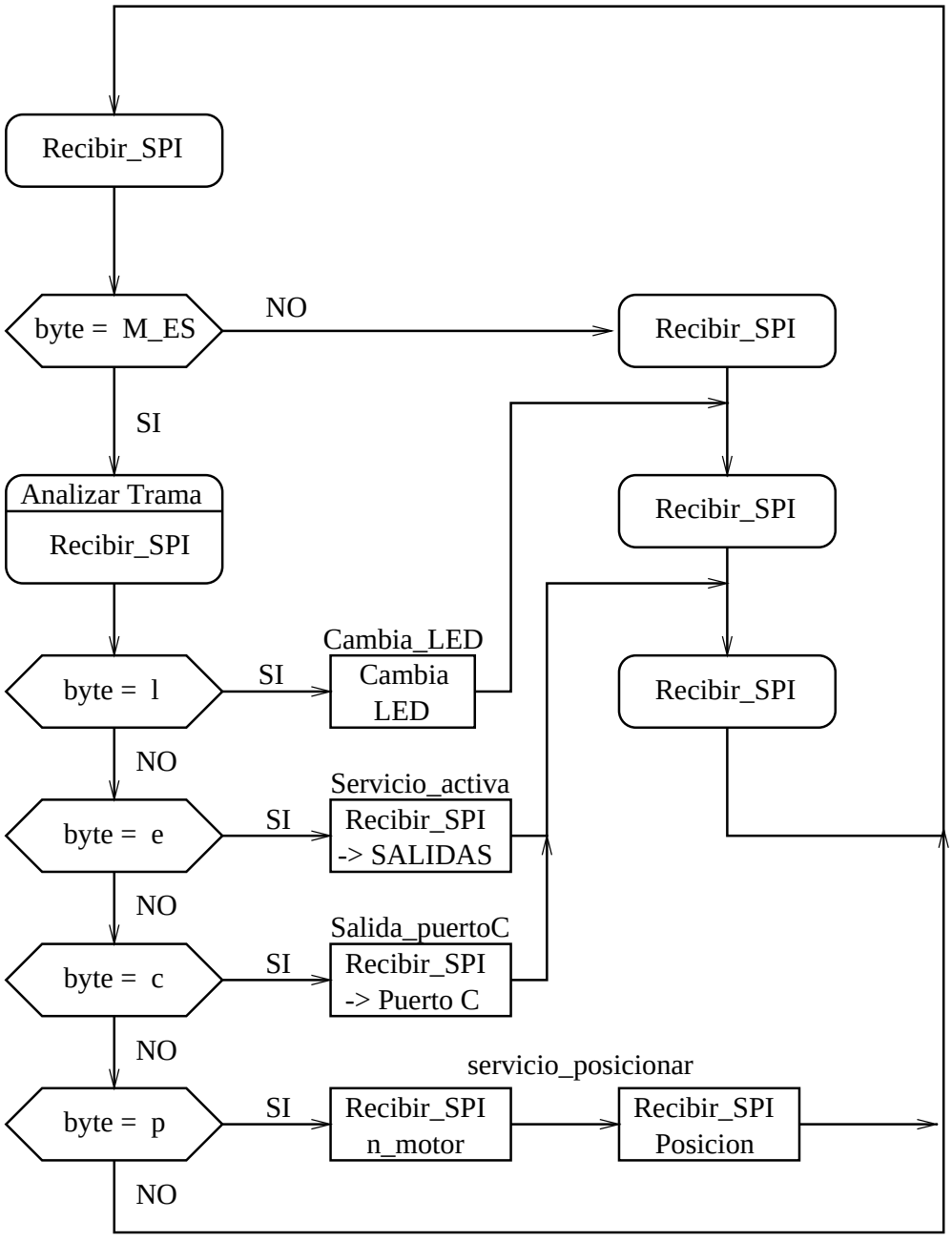

Figura 5.2: Diagrama de recepción de tramas en el módulo esclavo.

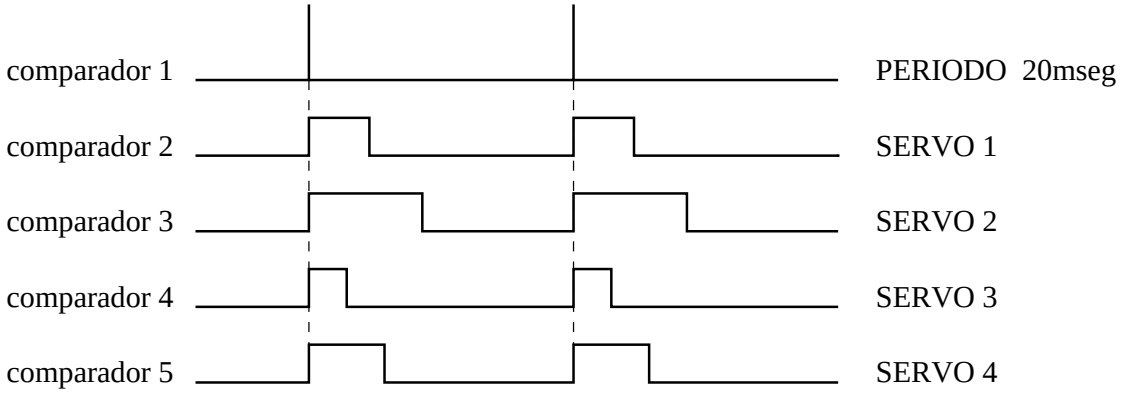

Figura 5.3: Señales de PWM de los cuatro servos.

servos por lo que serán necesarias cinco señales de temporización. Una común para todos, que controla el periodo de 20 mseg y otras cuatro que generan los cuatro anchos de las señales de PWM, (ver figura 5.3).

El 68hc11 tiene cinco comparadores que permiten generar intervalos de tiempo de manera automática. Para saber más de ellos se recomienda leer el capítulo 4.7 del libro *Microcontrolador 68hc11: Fundamentos, recursos y programación* [9]. Ahora sólo interesa conocer que el comparador 1 es el que se encarga de calcular los intervalos de 20 mseg y los comparadores del 2 al 5 se encargan de gestionar los anchos de los pulsos de las señales PWM de los servos del 1 al 4.

El funcionamiento del código es una ampliación del mostrado en el capítulo 3, pero ahora en lugar de generar una sola señal de PWM, se generan cuatro. Al iniciar cada periodo de 20 mseg se ponen a nivel alto las señales de PWM y se activan los comparadores 2, 3, 4 y 5. A medida que estos van terminando de contar sus intervalos, van desactivando sus salidas de PWM, de manera que, al final, todos estarán inactivos, salvo el número 1 que siempre está activo y que espera a iniciar otro periodo. No es posible que se inicie un periodo nuevo antes de que algún comparador haya terminado, porque la norma establece un ancho de pulso máximo de 2.3 mseg, que es inferior a los 20 mseg del periodo global.

El microcontrolador permite asociar a cada comparador una interrupción que avisa de cuándo su cuenta ha terminado. Al producirse la interrupción, se abandona la ejecución del bucle principal y se pasa a ejecutar un pequeño código asociado a cada comparador, en una *rutina de atención a interrupción*. Al terminar de ejecutar este código se devuelve el control al bucle principal, en el mismo punto donde se dejó. Como hay cinco comparadores, habrá cinco *rutinas de interrupción*, de las cuales la asociada al comparador 1 es la más grande y diferente, las demás son muy pequeñas e idénticas, salvo por los bits que modifican. Los diagramas de flujo asociados a las mismas se muestran en las figuras 5.4 y 5.5.

Todavía queda pendiente explicar cómo se comunican ambas partes del programa. La primera parte recibe los datos de la red pero, ¿cómo se los transmite a la segunda?. La respuesta es utilizando cinco variables situadas en una zona de memoria común (ver tabla 5.1). A éstas accede el bucle principal para actualizar sus valores, los cuales serán leídos después por la *rutina de interrupción* del comparador 1 para poder actualizar los anchos del resto de los comparadores, lo que produce una variación en la posición de los servomeca-

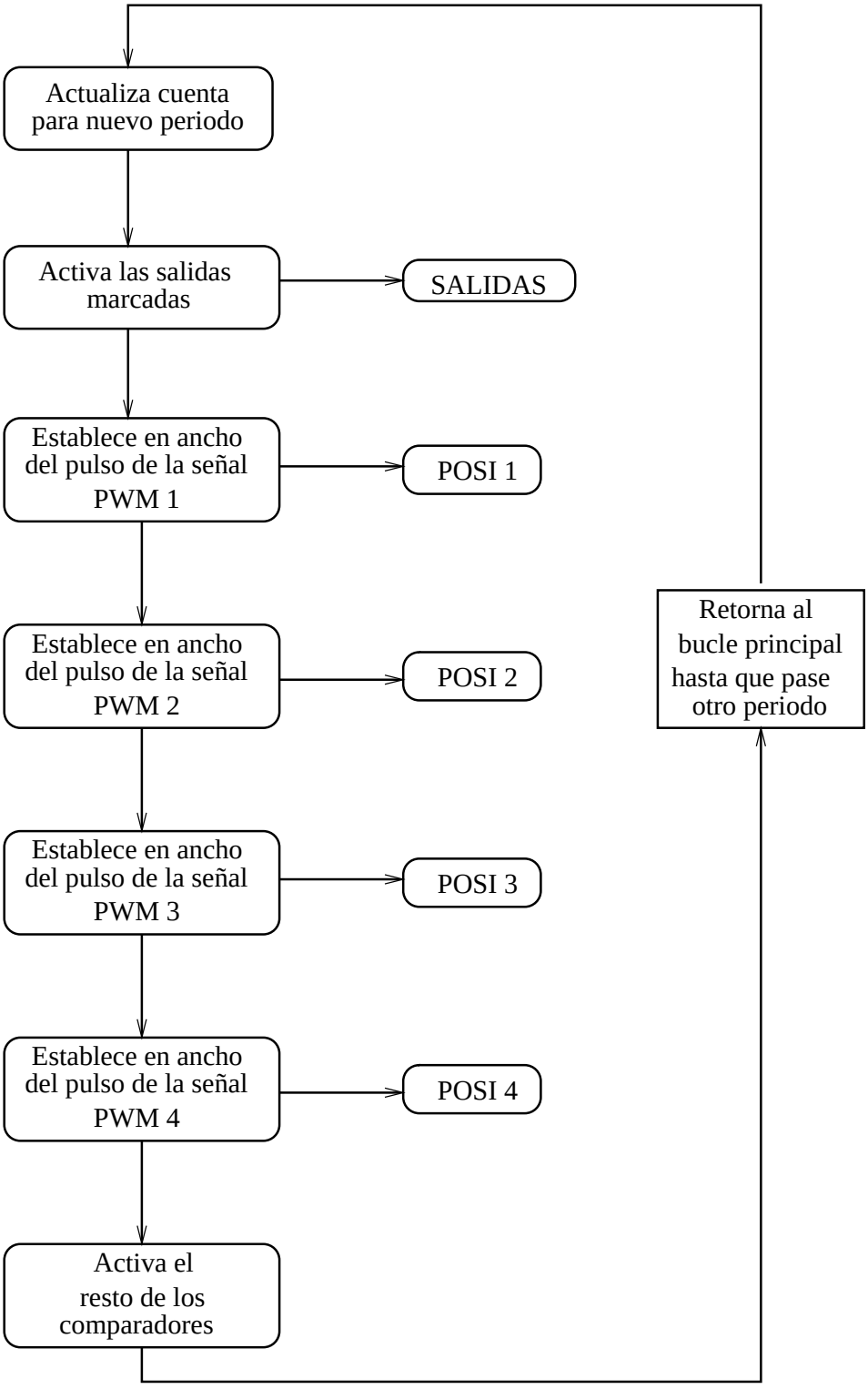

Figura 5.4: Rutina de interrupción del comparador 1.

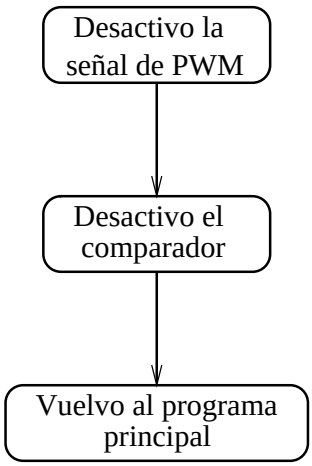

Figura 5.5: Rutina de interrupción de los comparadores 2, 3, 4 y 5.

Variable	Significado
posi1	Dos bytes que indican el ancho del PWM 1
posi2	Dos bytes que indican el ancho del PWM 2
posi3	Dos bytes que indican el ancho del PWM 3
posi4	Dos bytes que indican el ancho del PWM 4
salidas	Un byte que indica los motores activos

Tabla 5.1: Significado de las variables del programa.

nismos.

Examinando con detalle el código, se aprecia que los comparadores 2, 3, 4 y 5 no se configuran a la vez, sino consecutivamente. Esto produce un pequeño error en el ancho de los pulsos del PWM, sobre todo en el caso del comparador 5, ya que desde que se activa la señal de salida hasta que se configura el ancho pasa un pequeño tiempo que produce que dicho ancho se incremente (ver figura 5.6). El error cometido es de apenas 30 microsegundos que traducido a tics[1] se corresponde con:

$$Error = \frac{30\mu seg}{500nseg} = 60\, tics$$
$$\Delta Desplazamiento = \frac{60}{25} = 2.4$$

Un incremento de 25 tics produce un desplazamiento visible del eje, por lo que al tener un error de 60 tics se estará produciendo un error de 2 unidades mínimas de desplazamiento. Este error se puede considerar despreciable ya que es enmascarable por el error mecánico, explicado al final del apartado. A ese error hay que sumarle otro debido a la posibilidad de que dos comparadores se deban desactivar a la vez, con lo que siempre habrá alguno que tendrá que esperar un poco de tiempo. El caso peor es que los cuatro comparadores se tengan que desactivar a la vez, entonces y por diseño del microcontrolador será el comparador 5 el último que se desactive.

[1] En el capítulo 3.2 se comentó que un *tic* es la únidad mínima de cómputo de tiempo del microcontrolador, y que para apreciar un desplazamiento en el eje del servomotor por lo menos hay que variar el ancho de la

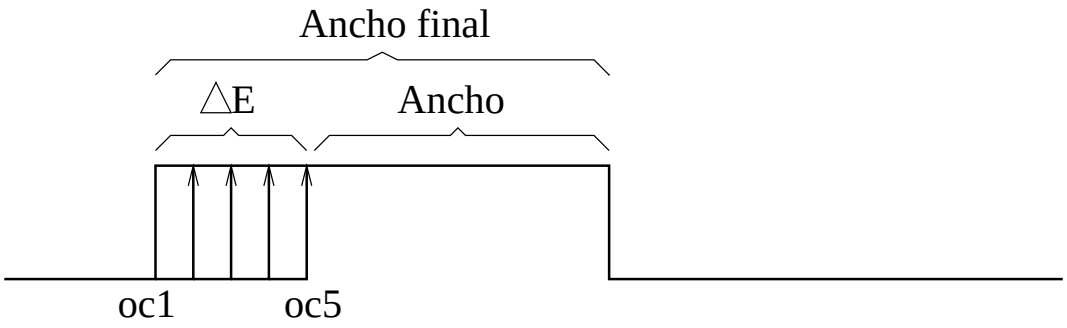

Figura 5.6: Primer error en el PWM debido a la activación anticipada de las salidas.

Haciendo los cálculos pendientes, resulta que el comparador 5 sufre los dos efectos, que además son acumulativos. Para evitar esto e igualar los errores de todos los comparadores, se puede hacer una pequeña modificación en el software consistente en cambiar el orden de configuración, es decir, empezar configurando el comparador 5 y terminar con el 2. Además, si se modifica el programa para que a la vez que se configura un comparador se active su salida, se consigue eliminar el primer error. El inconveniente de hacer esto es que la *rutina de interrupción* del comparador 1 aumenta de tamaño, incrementando el tiempo de CPU dedicado a ella.

Este estudio de errores es válido en la teoría pero no en la práctica, la razón es que el robot realiza movimientos mecánicos que no están libres de inercias, rozamientos y esfuerzos. Todo esto produce que los motores no lleguen a las posiciones correctas sino que se queden muy próximos a ellas, siendo el error cometido por la mecánica mayor que los anteriores y al ser independientes, los mecánicos enmascaran a todos los demás. A pesar de lo dicho en este párrafo, se ha optado por modificar el programa, pero como ya se preveía, al probar la nueva versión no se apreció ningún cambio de comportamiento.

Otro detalle de interés, es que debido al método usado para encontrar las secuencias de movimiento, los errores (excepto el mecánico) se estarán teniendo en cuenta automáticamente. En realidad lo que ocurre, es que cuando el usuario busca la secuencia de movimiento, utiliza unas barras deslizadoras que al final dan el ancho de la señal de PWM. Pero ese ancho no es el que se ha venido explicando en este capítulo, sino es el adecuado para que al sumarle el error de el ancho correcto al final. Al reproducir la secuencia no se utilizan las barras, sino los valores numéricos, pero da igual, estos son una representación del estado de la barra y por lo tanto también representan el ancho final. Como se dijo al principio sin darse uno cuenta se están evitando los errores aunque no eliminando.

5.3.4 Tiempo de uso de la CPU

Cuando en un programa hay dos partes independientes ejecutándose y una de ellas tiene prioridad sobre la otra, hay que verificar que las dos tengan suficiente tiempo de CPU, es decir que las dos se ejecuten concurrentemente. Aunque esto parezca algo trivial, en bastantes ocasiones ocurre que la parte gestionada con interrupciones está continuamente interrumpiendo a la otra, por lo que al final da la impresión de que sólo funciona una parte del software.

---

señal PWM en 25 *tics.*

Para estar seguros de que el programa realizado para el robot funciona correctamente, se ha hecho un estudio sobre el tiempo de ejecución de ambas partes. De él se han sacado las siguientes conclusiones:

1. La primera parte del programa se dedica a recibir tramas por el SPI y luego a analizarlas. La mayor parte del tiempo, la CPU se queda esperando a recibir los datos por lo que se puede considerar que no está ocupada.

2. La segunda parte del programa se encarga de generar las señales de PWM utilizando para ello los comparadores. Además, estos estarán siempre funcionando aunque no se activen los motores. Al estar activos cinco comparadores se sabe que habrá cinco interrupciones en cada periodo de 20mseg. Es decir, la primera parte del programa sufre cinco paradas cada 20 mseg. La pregunta es la siguiente, ¿cuánto tiempo tardan en ejecutarse las cinco interrupciones? El caso peor sería que fuese un tiempo de 20mseg, por lo que no quedaría más tiempo para realizar otras tareas, que es el caso en el que sólo se ejecuta uno de los dos programas. El caso más favorable sería que el tiempo fuese tan pequeño, que el transcurrido entre la recepción de dos caracteres por el SPI lo superara. De esta forma, aunque se interrumpiera el primer bucle en el momento crítico de recepción de caracteres daría tiempo a recibir el siguiente. En el apartado 4.1 se calculó ese intervalo y resultó ser de 8 μseg[2], pero además se vio que al mandar los datos desde el PC la velocidad la determinaría el puerto serie SCI y debido a la velocidad de éste, 9600 baudios, se obtiene un nuevo valor aproximado de 1mseg.

Una vez vistos los dos puntos anteriores, se calculará el tiempo que tardan las *rutinas de interrupción*. La más importante es la del comparador 1, que después de haber realizado los cambios anteriores ha crecido de tamaño y por lo tanto aumentado su tiempo de ejecución. Las otras rutinas son todas iguales, por lo que con estudiar el tiempo de una nos valdrá para todas. Recordando la teoría básica de sistemas digitales, para calcular el tiempo de ejecución hay que hacer lo siguiente:

1. Poner al lado de cada instrucción del algoritmo el número de ciclos de reloj que utiliza.

2. Multiplicar el número anterior por el tiempo que tarda en ejecutarse un ciclo, que en el caso del microcontrolador 68hc11 es de 500nseg.

De esta forma y empezando por la *rutina de interrupción* del comparador 1, que se recuerda que es la encargada de activar y configurar el resto de los comparadores, se tiene:

```
*.....................................
*. Rutina Interrupción Comparador 1 .
*.....................................
int_oc1
6       BSET TFLG1,X OC1
```

[2]Como la velocidad del SPI es de 1Mbit/seg y un carácter tiene 8 bits el tiempo en recibir uno es de 8 μseg.

```
5       LDD TCNT,X
4       ADDD #PERIODO
5       STD TOC1,X

4       LDAA PORTB,X
3       ORA  salidas
2       ANDA #$18
4       STAA PORTB,X
5       LDD  TCNT,X
5       ADDD posi4
5       STD  TOC5,X

4       LDAA PORTB,X
3       ORA  salidas
2       ANDA #$1C
4       STAA PORTB,X
5       LDD TCNT,X
5       ADDD posi3
5       STD TOC4,X

4       LDAA PORTB,X
3       ORA  salidas
2       ANDA #$1E
4       STAA PORTB,X
5       LDD TCNT,X
5       ADDD posi2
5       STD TOC3,X

4       LDAA PORTB,X
3       ORA  salidas
2       ANDA #$1F
4       STAA PORTB,X
5       LDD TCNT,X
5       ADDD posi1
5       STD TOC2,X

6       BSET TMSK1,X $78

12      RTI
```

Sumando todos los ciclos, indicados en los números situados delante de las instrucciones, se obtiene la cantidad de 145, que multiplicados por el valor de 500nseg da un tiempo de ejecución de de **72'5**μseg. Haciendo lo mismo para las rutinas de interrupción del resto de comparadores se obtiene:

```
*.....................................
```

```
*. COMPARADOR 2: POSICION DEL FUTABA 1  .
*.......................................

int_oc2
2       LDAA #OC2
4       STAA TFLG1,X

7       BCLR PORTB,X FUT1
7       BCLR TMSK1,X OC2
12      RTI
```

El número de ciclos total es de 32 que multiplicados por los 500nseg da un tiempo de ejecución de $16\mu seg$ que es bastante inferior al anterior, como ya se había adelantado. Al final, en cada periodo de 20mseg se dedican $72'5\mu + 4 * 16\mu seg = 136\mu seg$ a gestionar las interrupciones. Este tiempo es superior al establecido por el SPI ($8\mu seg$) pero bastante inferior al que realmente se tiene, por ser el SCI el cuello de botella (1mseg). Esto significa que cuando se utilice el PC no hará falta disminuir la velocidad de transmisión pero cuando no se utilice el PC y sea el módulo maestro el que mande las tramas habrá que aumentar el tiempo de espera entre la transmisión de dos tramas consecutivas, para garantizar un perfecto funcionamiento. Esto se hará en el programa maestro, por lo que el programa esclavo no se tendrá que modificar.

En resumen, no se necesitará proteger la recepción de datos al poder ejecutarse todas las interrupciones entre la recepción de dos caracteres consecutivos. Además de ese intervalo de 20mseg quedarán 19'86mseg para ejecutar el código relativo al análisis y ejecución de las tramas. Es decir, prácticamente la CPU del microcontrolador estará sin trabajo, en concreto esperando a recibir tramas por el puerto serie SPI. Hay que hacer notar que esto no significa que el microcontrolador no esté haciendo nada, todo lo contrario, debido a todos los recursos internos el micro estará trabajando en paralelo, liberando de trabajo a la CPU.

5.3.5 Código interno de los módulos esclavos

```
*.........................................
*.  FUTABA.  Version 1.0            2000  .
*.........................................
*.  Programa de los módulos esclavos. El   .
*. valor de la etiqueta M_ES identifica al .
*. módulo esclavo donde debe grabarse el   .
*. programa.                               .
*.  El programa gestiona la recepcion de   .
*. tramas la red y la generacion del PWM   .
*. para mover 4 servomecanismos Futaba     .
*.........................................

*****************************************
* Identificador del módulo esclavo       *
```

```
*******************************************

M_ES  equ 'a'

*******************************************
* Registros y Puertos del 68hc11          *
*******************************************

* Puertos del 68hc11

PORTA equ $00
PORTB equ $04
PORTC equ $03
DDRC  equ $07

* Registros del SPI

PORTD   EQU $08
DDRD    EQU $09
SPCR    EQU $28
SPDR    EQU $2A
SPSR    EQU $29

* Registros de los comparadores

TCNT  equ $0E  * valor temporizador principal
TMSK1 equ $22
TFLG1 equ $23
TCTL1 equ $20
TOC1  equ $16  * este registro es de 16 bits
TOC2  equ $18  * este registro es de 16 bits
TOC3  equ $1A  * este registro es de 16 bits
TOC4  equ $1C  * este registro es de 16 bits
TOC5  equ $1E  * este registro es de 16 bits

*******************************************
* Constantes del programa                 *
*******************************************

PERIODO  EQU 42000 * señal cuadrada 21 mseg
EXTREMO1 EQU 600
CENTRO   EQU 2600
EXTREMO2 EQU 4300

*******************************************
* Máscaras de acceso                      *
```

```
*******************************************

OC1  equ $80   * Comparador 1
OC2  equ $40   * Comparador 2
OC3  equ $20   * Comparador 3
OC4  equ $10   * Comparador 4
OC5  equ $08   * Comparador 5

FUT1 equ $01   * Bit donde se encuentra el FUTABA 1
FUT2 equ $02   * Bit donde se encuentra el FUTABA 2
FUT3 equ $04   * Bit donde se encuentra el FUTABA 3
FUT4 equ $08   * Bit donde se encuentra el FUTABA 4
LED  equ $10   * Bit donde se encuentra el LED

*******************************************
* Variables del programa                  *
*******************************************

        ORG $0

posi1   RMB 2
posi2   RMB 2
posi3   RMB 2
posi4   RMB 2
salidas RMB 1  * indica las salidas hardware activas

*******************************************
* ------------------------------------- *
* |          PROGRAMA PRINCIPAL         | *
* ------------------------------------- *
*******************************************

        ORG $F800    ; Programa para el micro E2

*******************************************
* Inicialización del programa             *
*******************************************

inicializar
        LDS #$FF     * inicializo el puntero de pila
        LDX #$1000   * apunta a los registros internos

*  --- Configuración del SPI (como esclavo) ----

        LDAA  #$3C    * SS de entrada
        STAA  DDRD,X  * El SPI configura las salidas
```

```
        LDAA  #$40    * Colector cerrado
        STAA  SPCR,X  * Activa en modo esclavo el SPI

* --- Configuración del Puerto C como salida

        LDAA #$FF
        STAA DDRC,X

* --- Configuración de los comparadores ---

        CLRA          * Comparadores sin
        STAA  TCTL1,X *  salida hardware
        LDAA  #OC1    * Activo el comparador 1
        STAA  TMSK1,X

* --- Configuración de las salidas ---

        LDD   #2000
        STD   posi1
        STD   posi2
        STD   posi3
        STD   posi4

        CLRA          * indico que todos los
        STAA  salidas * motores estan inactivos

        CLI           * permito las interrupciones

************************************************
* BUCLE PRINCIPAL                              *
************************************************

inicio
        BSR  recibir_spi
        CMPA #M_ES       * verifica que la trama sea
        BEQ  analizar    * para este modulo
        BSR  recibir_spi
        BSR  recibir_spi
        BSR  recibir_spi
        BRA  inicio

analizar BSR  recibir_spi   * leo caracter por el SPI
         CMPA #'l'                * ¿ orden 'l'?
         BEQ  cambia_led          * Si -> cambia led
         CMPA #'e'                * ¿ orden 'e' ?
```

```
        BEQ  servicio_activa      * Si -> activa motor
        CMPA #'p'                 * ¿ orden 'p' ?
        BEQ  servicio_posicionar  * Si -> posicionar
        CMPA #'c'                 * ¿ orden 'c' ?
        BEQ  salida_puertoc       * Si -> Puerto C
        BRA  inicio               * Espera trama_nueva

*********************************************
* SUBRUTINAS DEL PROGRAMA                   *
*********************************************

* ..........................................
* . Rutina que cambia el estado del LED    .
* ..........................................

cambia_led
        BSR  recibir_spi
        BSR  recibir_spi
        LDAB PORTB,X
        EORB #$10
        STAB PORTB,X
        BRA  inicio

* ..............................................
* . Rutina que activa o desactiva los motores  .
* ..............................................

servicio_activa
        BSR  recibir_spi
        STAA salidas
        BSR  recibir_spi
        BRA  inicio

* ................................................
* . Rutina que envia un byte al puerto de salida C .
* ................................................

salida_puertoc
        BSR  recibir_spi
        STAA PORTC,X
        BSR  recibir_spi
        BRA  inicio

* ..........................................
* . Servicio posicionar motor              .
* ..........................................
```

```
servicio_posicionar
        BSR recibir_spi  * motor-> $1, $2, $3, $4

* Se ha solicitado servicio de posicionamiento
* Meter en Y elñumero (motor-1)*2
        DECA    * A=motor-1
        LSLA    * A=(motor-1)*2
        TAB
        CLRA
        XGDY    * Y=(motor-1)*2

        BSR  recibir_spi  * Leer posicion (0-255)
        CLRB
        LSRD
        LSRD
        LSRD
        LSRD
        ADDD #EXTREMO1    * D=(posicion*16+EXTREMO1)
        SEI
        STD  0,Y          * Almacenar posicion
        CLI
        BRA  inicio

* ..............................................
* . Rutina que recibe un dato por el SPI       .
* . La rutina espera hasta recibir el dato     .
* . Entradas: .Ninguna                         .
* . Salidas: El acumulador A contiene el dato  .
* ..............................................

recibir_spi
espera  BRCLR SPSR,X $80 espera  * esperar dato
        LDAA  SPDR,X
        RTS

***********************************************
* Rutinas de interrupcion de los  COMPARADORES *
***********************************************

* ..............................................
* . Comparador 1: Establece el principio de los .
* . pulsos                                      .
* ..............................................
* . Podemos poner BSET porque las interrupciones.
* . de los otros comparadores estan desactivadas.
```

```
* . En caso contrario habría que utilizar load y.
* . store.                                      .
* ..............................................

int_oc1
        BSET TFLG1,X OC1  * flag de interrupción a cero

        LDD TCNT,X      * Actualizar comparador 1
        ADDD #PERIODO   * próxima interrupción =
        STD TOC1,X      *  tiempo_actual(TCNT) + Periodo

        LDAA PORTB,X    * Futaba 4
        ORA  salidas
        ANDA #$18       * Si salida 4 habilitada
        STAA PORTB,X    * ponerla añivel alto
        LDD  TCNT,X     * Actualizar comparador 5
        ADDD posi4      * Establecer anchura del pulso
        STD  TOC5,X

        LDAA PORTB,X    * Futaba 3
        ORA  salidas
        ANDA #$1C       * Si salida 3 habilitada
        STAA PORTB,X    * ponerla añivel alto
        LDD TCNT,X      * Actualizar comparador 4
        ADDD posi3      * Establecer anchura del pulso
        STD TOC4,X

        LDAA PORTB,X    * Futaba 2
        ORA  salidas
        ANDA #$1E       * Si salida 2 habilitada
        STAA PORTB,X    * ponerla añivel alto
        LDD TCNT,X      * Actualizar comparador 3
        ADDD posi2      * Establecer anchura del pulso
        STD TOC3,X

        LDAA PORTB,X    * Futaba 1
        ORA  salidas
        ANDA #$1F       * Si salida 1 habilitada
        STAA PORTB,X    * ponerla añivel alto
        LDD TCNT,X      * Actualizar comparador 2
        ADDD posi1      * Establecer anchura del pulso
        STD TOC2,X

* Activar interrupciones de los comparadores 2,3,4 y 5

        BSET TMSK1,X $78
```

```
        RTI

* ..............................................
* . ¡OJO¡ Ño podemos poner BSET porque podria .
* . desactivar las otras interrupciones de los .
* . comparadores antes de ser atendidas        .
* ..............................................
* .    Rutina de servicio de interrupcion del  .
* . COMPARADOR 2 : POSICION DEL FUTABA 1       .
* ..............................................

int_oc2
        LDAA #OC2        * flag de interrupcion a cero
        STAA TFLG1,X
        BCLR PORTB,X FUT1  * fin ancho del pulso
        BCLR TMSK1,X OC2   * Deshabilitar interrupcion
        RTI

* ..............................................
* .    Rutina de servicio de interrupcion del  .
* . COMPARADOR 3 :  POSICION DEL FUTABA 2      .
* ..............................................

int_oc3
        LDAA #OC3        * flag de interrupcion a cero
        STAA TFLG1,X
        BCLR PORTB,X FUT2  * fin ancho del pulso
        BCLR TMSK1,X OC3   * Deshabilitar interrupcion
        RTI

* ..............................................
* .    Rutina de servicio de interrupcion del  .
* . COMPARADOR 4 : POSICION DEL FUTABA 3       .
* ..............................................

int_oc4
        LDAA #OC4        * flag de interrupcion a cero
        STAA TFLG1,X
        BCLR PORTB,X FUT3  * fin ancho del pulso
        BCLR TMSK1,X OC4   * Deshabilitar interrupcion
        RTI

* ..............................................
* .    Rutina de servicio de interrupcion del .
* . COMPARADOR 5 :  POSICION DEL FUTABA 4     .
* ..............................................
```

```
int_oc5
        LDAA #OC5        * flag de interrupcion a cero
        STAA TFLG1,X
        BCLR PORTB,X FUT4  * fin ancho del pulso
        BCLR TMSK1,X OC5   * Deshabilitar interrupcion
        RTI

***********************************************
*Inicializacion de los vectores de interrupcion*
***********************************************

         ORG $FFE0          * vector del comparador 5
v_comp5  FDB  #int_oc5      * FUTABA 4

         ORG $FFE2          * vector del comparador 4
v_comp4  FDB  #int_oc4      * FUTABA 3

         ORG $FFE4          * vector del comparador 3
v_comp3  FDB  #int_oc3      * FUTABA 2

         ORG $FFE6          * vector del comparador 2
v_comp2  FDB  #int_oc2      * FUTABA 1

         ORG $FFE8          * vector del comparador 1
v_comp1  FDB  #int_oc1

         ORG $FFFE          * vector del reset
v_reset  FDB  #inicializar

         END
```

5.4 Software del módulo maestro

El programa del módulo maestro tiene varias funciones siendo la más importante la de servir de *puente* entre el PC y la red de control. Se recuerda que dicho módulo es el que se encarga de enviar las tramas por la red y además tiene una conexión serie asíncrona (SCI) para poder conectar periféricos externos, como puede ser un PC, una calculadora HP, etc. La función de *puente* es muy sencilla, todo lo que se recibe por la conexión serie SCI, se manda por la red. En este proceso no se interpreta ni verifica nada y como la línea SCI es más lenta que la red, no será necesario introducir retardos. En este modo de funcionamiento el robot no es autónomo sino que depende del PC para moverse.

Otra de las funciones del módulo maestro es la ejecución de un programa que hace que el robot sea autónomo, es decir, que el robot no necesite del PC para moverse. En

este caso concreto se ha programado para ir recto, pero se podía haber realizado cualquier otro movimiento. Por ejemplo, otra función más compleja sería mover el robot de forma autónoma pero siguiendo un comportamiento, por ejemplo mover el robot recto, siempre y cuando no detecte luz, en caso contrario hacer que gire.

En estos momentos no es posible programar las tres funciones, hay que seguir un orden determinado. Primero se tiene que programar la función *puente*, de esa forma y utilizando el PC se podrán obtener las secuencias de movimiento del robot. Primero se buscará la de ir recto y luego se irán buscando otras más complejas: girar a un lado, al otro, levantarse, tumbarse, etc. Una vez encontradas éstas se procederá a introducirlas dentro del programa maestro para que sea éste el que las envíe por la red sin necesidad del PC. La primera prueba consiste en hacer que vaya recto, de manera que se puedan realizar demostraciones sencillas. Por último se podría añadir algo de inteligencia al programa para que pueda detectar un foco de luz y realizar diferentes tareas en función de la intensidad de luz recibida.

De todas las funciones explicadas tan sólo se han programado las dos primeras. La tercera se ha pospuesto porque sólo tiene sentido una vez que la estructura está al cien por cien operativa y como se verá en el capítulo siguiente, todavía quedarán algunas mejoras pendientes. Para seleccionar una u otra función se utiliza el switch 3 del módulo maestro: puesto abajo ejecuta el programa *puente* y puesto arriba ejecuta el programa de ir recto. Más adelante se pensará si sustituir éste último o si por el contrario se añade el switch 4 para seleccionar el tercer programa.

5.4.1 Primera función: Programa *Puente*

Esta función es muy simple y no debe plantear ningún problema al lector a la hora de interpretarla. Para activarla hay que dejar el switch 3 de la CT6811 maestra abajo (posición ON). El detalle más importante que se debe mencionar es que se ha configurado el SCI a una velocidad de 9600 baudios. Lo bueno de utilizar este valor es que es un estándar y programas como el Visual Basic y Visual C tienen rutinas de transmisión a esta velocidad. Además la mayoría de dispositivos que pueden transmitir por un puerto serie la permiten.

Otro detalle muy importante, ya mencionado antes, es que no se han insertado pausas para la transmisión de datos por el puerto SPI, es decir por la red, porque la propia estructura del programa hace que no sea necesario. Se debe a que sólo se envía un dato por el SPI cuando ha sido recibido del PC y como la velocidad de recepción es lenta la de transmisión también lo será.

De esa forma el programa tiene un bucle principal muy pequeño '**prog_uno**' y dos subrutinas, una para recibir los datos '**leer_car**' y la otra para enviarlos '**enviar_spi**'. El bucle principal ejecuta la rutina de recepción y se mantiene en ella hasta recibir algo, luego pasaría a ejecutar la rutina de transmisión. Ver figura 5.7.

El programa *puente* no sabe nada de tramas ni de protocolos, su única misión es mandar los datos que le llegan por la red de control, por eso será el PC el que se encargue de montar las tramas con la estructura correcta. Se puede decir que se ha traspasado la inteligencia del módulo maestro al PC.

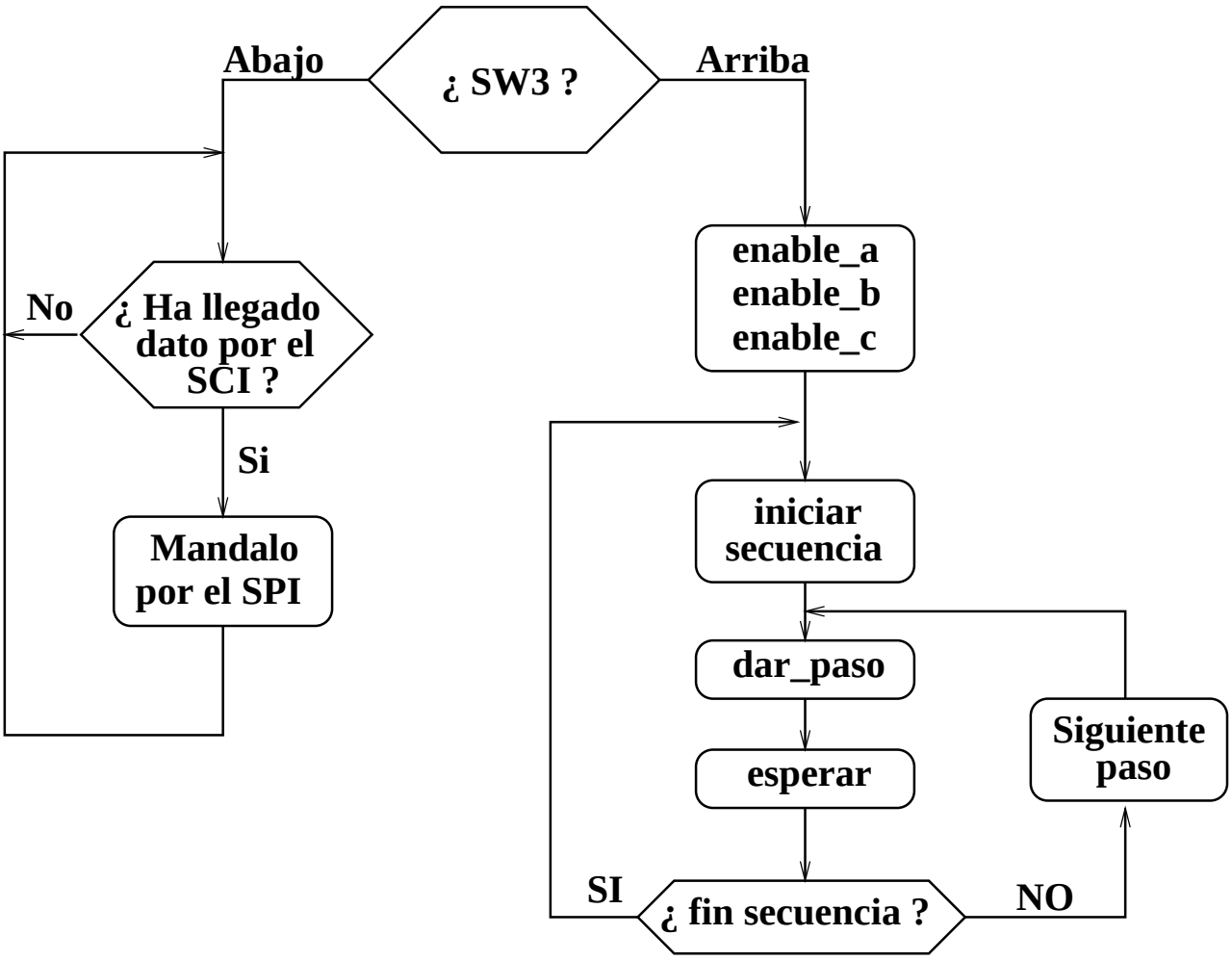

Figura 5.7: Diagrama del programa maestro.

5.4.2 Segunda función: Programa ir Recto

Colocando el switch 3 del módulo maestro arriba se entra en este nuevo modo de funcionamiento. Ahora el PC no realiza ninguna función y será el propio módulo maestro el que mueva el robot. Debido a que todavía no se le ha dotado de sensores no se puede interactuar con el entorno, por lo que el robot efectuará un solo tipo de movimiento, que en este primer caso se tratará de ir hacia adelante. Es importante que el lector entienda que aunque se está explicando y detallando el código justo después de la primera parte, para llegar a tener ésta, ha sido necesario desarrollar todo el software del PC, calcular las secuencias de movimiento, grabarlas y luego programar esta parte del código, que al integrarla con la primera, da la impresión de tratarse de un único programa.

Si además no se hubiesen detectado algunas anomalías, se hubieran puesto algunos sensores para poder haber terminado el programa añadiéndole la tercera parte, o lo que es lo mismo, la capacidad de reaccionar ante el entorno para realizar distintos tipos de movimientos.

Este programa monta la trama para transmitirla por la red, pero ahora sí es necesario introducir una pequeña pausa entre dos datos enviados consecutivamente. La razón, como ya se ha dicho, se debe a la alta velocidad del SPI que transmite tan rápido los datos que no da tiempo a que se ejecuten las rutinas de interrupción de los módulos esclavos antes de que llegue el siguiente dato, produciendo la ruptura de la trama y la pérdida de sincronismo. En este caso, al no haber mecanismo de seguridad todo dejaría de funcionar. La manera más fácil y directa de evitar esto es incluir un pequeño código que introduzca un retardo mayor de 150 μseg entre datos enviados. De esa forma dará tiempo a que se ejecuten a todas las rutinas de interrupción entre dos datos consecutivos y se tendrá un caso parecido a la transmisión desde el PC, con la salvedad que el cuello de botella se ha introducido a propósito.

La pausa de 150 μseg se genera mediante una espera activa, el código de la rutina se presenta a continuación:

```
4       LDY #$18  * Produce el retardo
4  sig  DEY
5       CPY #$0
3       BNE sig
```

Esta rutina consiste en ir decrementando el valor del *acumulador Y* hasta que sea igual a cero, momento en el que se sigue con el resto del programa. Cada vez que se repite el bucle se está consumiendo una pequeña porción de tiempo y según se varíe el valor del acumulador Y, se consumirá más o menos tiempo. Como hay que consumir aproximadamente 150 μseg la ecuación que hay que resolver es la siguiente:

$$[4 + (4 + 5 + 3) * Y] * 500nseg = 150\mu seg^{3}$$

Resolviendo para Y se obtiene el valor 24 en decimal que se corresponde con **$18 en hexadecimal**. El cálculo es aproximado, pues no se ha tenido en cuenta el resto del código, pero al cumplirse ya con este trozo está claro que con el resto se cumplirá la condición con más holgura.

[3] El valor de 500nseg es el tiempo que tarda el microcontrolador en ejecutar un ciclo de reloj.

El bucle principal de esta función se llama **prog_dos,** en él primero se activan los módulos esclavos por medio de las subrutinas **enable_a**, **enable_b** y **enable_c**. Luego se configura Y para que apunte a la zona de memoria donde se guardan las secuencias de movimiento[4] y se procede a formar las tramas de los movimientos para enviarlas por la red: **subrutina dar_paso**.

Una secuencia de movimiento es un conjunto de pasos que dados en el orden correcto producen que el robot avance, gire, retroceda, etc. En la ejecución de una secuencia se da lo siguiente: cuando se ha terminado con un paso se espera un poco, **subrutina espera** y se procede a ejecutar el siguiente hasta que Y llega al último paso que es el final de la secuencia (**fin_movi**), momento en el que se vuelve al principio del bucle (**secuencia**) para ejecutar el primer paso. Ver figura 5.7.

La *subrutina espera* es la que introduce el retardo para que los motores tengan tiempo de ir a su posición final. En el PC este tiempo es configurable, pero aquí se ha simplificado un poco ya que el tiempo de pausa es fijo y equivale a 0.3 seg, que es el valor que determina el programa del PC al iniciarse. En este tiempo, los motores pueden girar unos 90 grados, cantidad considerada como suficiente para realizarse dentro de un mismo paso. En caso de no haberla simplificado se hubiera tenido que leer el último byte (ret) de cada paso (moviX) y hacerlo corresponder con un número de repeticiones del bucle *espera*, de modo que el total del tiempo consumido coincidiese con el valor indicado. Mediante esta rutina de pausa se obliga al programa a no poder actualizar la posición de un servo hasta que no hayan transcurrido por lo menos 0.3 seg.

La subrutina *mover* es la encargada de enviar cada trama por la red. Para ello, va siguiendo el protocolo y utilizando la subrutina **enviar_spi_pausa** manda a la red cada dato. El lector puede darse cuenta de que ésta es la misma utilizada por el programa *puente* pero con la pequeña diferencia de la inserción del trozo de código que realiza la pausa entre datos, de manera que se cumpla lo dicho antes: entre datos debe haber un intervalo de unos 150 μseg para que dé tiempo a ejecutar las interrupciones. Al cumplir esta condición, se están variando algunas de las suposiciones que se discutieron al analizar la red (capítulo 4), en concreto aquella que establecía que en 0.384 mseg se actualizaban todos los motores. Ahora ese tiempo se incrementa hasta 7.2 mseg y con él el intervalo que podría recorrer el eje de un motor que ahora sería de 1.8º. De todas formas, esto ya no tiene la menor importancia, porque como se ha dicho en el párrafo anterior, se ha obligado a que esos 7.2mseg sean 0.3seg, descargando aún más la red de control y dando tiempo a la ejecución de las interrupciones.

5.4.3 Código del programa maestro

```
************************************************
* Programa para el modulo maestro              *
************************************************
* Tiene dos funciones principales seleccionables *
```

[4]Estas secuencias son series de números que se han obtenido con el programa del PC. La secuencia tiene el siguiente significado: Primero se indica el número de paso 'moviX', luego se colocan los valores de las posiciones de los motores 'A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4' y finalmente el valor de los ojos junto con el tiempo a esperar para ejecutar el siguiente paso: 'ojo1, ojo2, ret'. El valor del retardo está en décimas de segundo.

```
* por el SW3 de la CT6811:                          *
*                                                   *
*   SW3 abajo: El programa hace que la tarjeta      *
*         maestra envie por el puerto SPI todo      *
*         lo que recibe por el puerto serie.        *
*                                                   *
*   SW3 arriba: El programa envia por la red las    *
*         tramasñecesarias para hacer que el        *
*         robot avance.                             *
*                                                   *
*****************************************************

* --- Puertos de entrada y salida

PORTA   EQU $0
PORTB   EQU $04
PORTC   EQU $03
DDRC    EQU $07

* --- Registros del SCI

BAUD    EQU $2B
SCCR1   EQU $2C
SCCR2   EQU $2D
SCSR    EQU $2E
SCDR    EQU $2F

* --- Registros del SPI

PORTD   EQU $08
DDRD    EQU $09
SPCR    EQU $28
SPDR    EQU $2A
SPSR    EQU $29


        ORG $F800  * Direccion de comienzo

inicio  LDS   #$FF
        LDX   #$1000

* ---- Configuración del SCI ----

wait    BRCLR SCSR,X $40 wait

        LDAA  #$30      * Velocidad de 9600 baudios
```

```
        STAA  BAUD,X
        LDAA  #$0
        STAA  SCCR1,X   * 8 bits de datos
        LDAA  #$0C      *.No usar interrupciones del SCI
        STAA  SCCR2,X   * Activar receptor y transmisor

* ---- Configuración del SPI (como maestro) ----

        LDAA  #$3C
        STAA  DDRD,X

* El maestro lo ponemos con colector cerrado.

        LDAA  #$50       * Colector cerrado
        STAA  SPCR,X     * Activa en modo maestro el SPI

* El puerto C tienen que ser de entrada
        CLRA
        STAA DDRC,X      * Puerto C de entrada

************************************
* Seleccion del programa a ejecutar *
************************************

seleccion
        LDAA PORTC,X     * Leo el puerto C
        ANDA #$01        * Mascara
        CMPA #$01        * ¿ Switch 3 Arriba ?
        BEQ  prog_dos    * SI -> salta al programa 2
        JMP  prog_uno    *.NO -> salta al programa 1

************************************
* ----------- PROGRAMA 1 ---------  *
************************************

prog_uno
        JSR  leer_car    * Leo dato del SCI
        JSR  enviar_spi  * mando dato por el SPI
        BRA  prog_uno

************************************
* ---------- PROGRAMA 2 ---------- *
************************************

prog_dos
        JSR  esperar
```

```
        JSR  enable_c
        JSR  enable_a
        JSR  enable_b
secuencia
        LDY  #movi0
bucle   JSR  dar_paso
        JSR  esperar
        CPY  #fin_movi
        BNE  bucle
        BRA  secuencia

************************************
*   Subrutinas utilizadas por los   *
* bucles principales uno y dos.     *
************************************

*..................................
*. Rutina que recibe un caracter por.
*. el puerto serie                 .
*..................................

leer_car
        BRCLR SCSR,X $20 leer_car
        LDAA  SCDR,X
        RTS

*..................................
*. Rutina que envia un caracter por .
*. el puerto serie                 .
*..................................

enviar_car
        BRCLR SCSR,X $80 enviar_car
        STAA SCDR,X
        RTS

*....................................
*. Rutina que envía un dato por el SPI .
*....................................
*. Si se quiere pausa entre datos      .
*. enviados llamar a: enviar_spi_pausa .
*.                                     .
*. Siño se quiere pausa entre datos   .
*. envados llamar a: enviar_spi        .
*....................................
```

```
enviar_spi_pausa
        PSHY          * pausa entre datos
        LDY   #$18    * valor de 150 microseg
pausa   DEY
        CPY #$0
        BNE pausa
        PULY
enviar_spi
        LDAB  PORTD,X  * Mandamos activarse
        ANDB  #$DF     * al esclavo
        STAB  PORTD,X
        STAA  SPDR,X   * introduzco el dato a mandar

espera  BRCLR SPSR,X $80 espera
        LDAB  PORTD,X
        ORB   #$20
        STAB  PORTD,X  * Desactivamos al esclavo

        RTS

*.........................
*. Rutina de pausa       .
*.........................

esperar
        PSHY
        LDY  #$C400
sigue   DEY
        CPY  #$0
        BNE  sigue
        PULY
        RTS

*.....................................
*. Rutinas para activar los módulos .
*.....................................
enable_c
        LDAA #'c'
        BRA  fin_enable
enable_b
        LDAA #'b'
        BRA  fin_enable
enable_a
        LDAA #'a'
fin_enable
        JSR  enviar_spi_pausa
```

```
        LDAA #'e'
        JSR  enviar_spi_pausa
        LDAA #$0F
        JSR  enviar_spi_pausa
        JSR  enviar_spi_pausa
        RTS

*................................
*.  Rutina que envía un paso de  .
*.  la secuencia                 .
*................................

dar_paso
        LDAA #'a'             * a1
        JSR  enviar_spi_pausa
        LDAA #'p'
        JSR  enviar_spi_pausa
        LDAA #$01
        JSR  enviar_spi_pausa
        LDAA $0,Y
        JSR  enviar_spi_pausa
        INY

        LDAA #'a'             * a2
        JSR  enviar_spi_pausa
        LDAA #'p'
        JSR  enviar_spi_pausa
        LDAA #$02
        JSR  enviar_spi_pausa
        LDAA $0,Y
        JSR  enviar_spi_pausa
        INY

        LDAA #'a'             * a3
        JSR  enviar_spi_pausa
        LDAA #'p'
        JSR  enviar_spi_pausa
        LDAA #$03
        JSR  enviar_spi_pausa
        LDAA $0,Y
        JSR  enviar_spi_pausa
        INY

        LDAA #'a'             * a4
        JSR  enviar_spi_pausa
        LDAA #'p'
```

```
JSR  enviar_spi_pausa
LDAA #$04
JSR  enviar_spi_pausa
LDAA $0,Y
JSR  enviar_spi_pausa
INY

LDAA #'b'             * b1
JSR  enviar_spi_pausa
LDAA #'p'
JSR  enviar_spi_pausa
LDAA #$01
JSR  enviar_spi_pausa
LDAA $0,Y
JSR  enviar_spi_pausa
INY

LDAA #'b'             * b2
JSR  enviar_spi_pausa
LDAA #'p'
JSR  enviar_spi_pausa
LDAA #$02
JSR  enviar_spi_pausa
LDAA $0,Y
JSR  enviar_spi_pausa
INY

LDAA #'b'             * b3
JSR  enviar_spi_pausa
LDAA #'p'
JSR  enviar_spi_pausa
LDAA #$03
JSR  enviar_spi_pausa
LDAA $0,Y
JSR  enviar_spi_pausa
INY

LDAA #'b'             * b4
JSR  enviar_spi_pausa
LDAA #'p'
JSR  enviar_spi_pausa
LDAA #$04
JSR  enviar_spi_pausa
LDAA $0,Y
JSR  enviar_spi_pausa
INY
```

```
        LDAA #'c'            * c1
        JSR  enviar_spi_pausa
        LDAA #'p'
        JSR  enviar_spi_pausa
        LDAA #$01
        JSR  enviar_spi_pausa
        LDAA $0,Y
        JSR  enviar_spi_pausa
        INY

        LDAA #'c'            * c2
        JSR  enviar_spi_pausa
        LDAA #'p'
        JSR  enviar_spi_pausa
        LDAA #$02
        JSR  enviar_spi_pausa
        LDAA $0,Y
        JSR  enviar_spi_pausa
        INY

        LDAA #'c'            * c3
        JSR  enviar_spi_pausa
        LDAA #'p'
        JSR  enviar_spi_pausa
        LDAA #$03
        JSR  enviar_spi_pausa
        LDAA $0,Y
        JSR  enviar_spi_pausa
        INY

        LDAA #'c'            * c4
        JSR  enviar_spi_pausa
        LDAA #'p'
        JSR  enviar_spi_pausa
        LDAA #$04
        JSR  enviar_spi_pausa
        LDAA $0,Y
        JSR  enviar_spi_pausa
        INY
        INY
        RTS


*****************************************************
*           SECUENCIAS DE MOVIMIENTOS               *
```

```
*****************************************************
* A1,A2,A3,A4,B1,B2,B3,B4,C1,C2,C3,C4,ojo1,ojo2,ret *
*****************************************************

*.....................
*. RECTO              .
*.....................

movi0   FCB 199,151,089,098,018,103,128
        FCB 066,215,108,085,023,$01,$01,3
movi1   FCB 197,151,011,082,016,103,126
        FCB 076,222,108,167,023,$02,$02,3
movi2   FCB 197, 96,011,082,016,103,126
        FCB 076,222,057,174,023,$04,$04,3
movi3   FCB 197,096,011,037,016,064,087
        FCB 117,222,057,174,023,$08,$08,3
movi4   FCB 197,096,011,037,016,021,000
        FCB 117,222,057,174,023,$10,$10,3
movi5   FCB 197,169,105,032,016,147,172
        FCB 117,222,108,094,023,$20,$20,3
movi6   FCB 197,169,112,126,016,147,172
        FCB 048,222,108,094,023,$40,$40,3
fin_movi FCB $0

*********************************
* Vectores de interrupción      *
*********************************

        ORG $FFFE
v_reset FDB #inicio

        END
```

5.5 Programas de ayuda en el PC

Contra todo pronóstico, el PC se ha convertido en una de las herramientas fundamentales de este proyecto. En el primer capítulo se comentó la experiencia que tenía el autor en la construcción de robots articulados. En aquel caso se trató de un hexápodo y la secuencia de movimiento se programó directamente en ensamblador, no hizo falta utilizar otros programas de ayuda. En este caso no es posible: la generación de la secuencia requiere un estudio y un control detallado de los movimientos de cada motor. Por eso, desde que se consiguió que la red funcionase, surgió la necesidad de disponer de una herramienta de trabajo que permitiese mover los motores de uno en uno, grabar sus posiciones y poder reproducirlas después.

El programa necesita disponer de una interfaz gráfica que sea amigable y fácil de usar.

Al estar en una plataforma Linux se ha optado por utilizar las *Xforms*. Para los que no estén familiarizados con ellas, se puede decir que son unas librerías en C que proporcionan las rutinas necesarias para crear interfaces gráficos, disponiendo además del '*fdesign*' : una herramienta que facilita la creación de paneles, barras deslizadoras[5], botones, etc.

Este primer programa de ayuda tiene que ser capaz de controlar la posición de cada uno de los doce motores del robot. Jugando con las posiciones se obtendrán los pasos que forman las secuencias de avanzar, retroceder, girar, etc. Una vez que se han obtenido, se podrá completar la función *puente* del programa maestro para añadir la función *autónoma*.

También se puede realizar otro programa de ayuda en el PC que permita mover el robot en todas direcciones sin necesidad de ir abriendo los ficheros que definen cada una de las diferentes secuencias que lo mueven en las distintas direcciones. En este caso se selecciona la dirección deseada y el programa directamente utilizada la secuencia adecuada que ya está definida en el código del mismo. Para distinguir ambos programas del PC se llamará **Programa Generador de Secuencias** al primero y **Programa de control** al segundo.

5.5.1 XPucho: Programa Generador de Secuencias

Como ya se ha dicho, básicamente este programa permite controlar el movimiento de cada motor y con ello el del robot entero. Antes de empezar a describir el funcionamiento hay que dejar claros una serie de conceptos. Cada motor se controla mediante una barra **deslizadora** en cuyo lado aparece un número que identifica la posición del eje del motor, de manera que dos números distintos identifican dos posiciones distintas. Se define **paso** como la posición instantánea del robot, es decir los valores de todas las barras en un momento dado. Se define **secuencia** como un conjunto de pasos, ordenados secuencialmente y con un retardo variable entre ellos. El paso que está siendo representado por las barras deslizadoras se define como **paso_actual** y su número se indica en la barra deslizadora llamada *pasos*. Al pulsar los botones de avanzar '>' o retroceder '<' por la secuencia el valor del paso_actual cambia al igual que el estado de las barras delizadoras. En esta versión del software las secuencias podrán tener un máximo de 1000 pasos cada una, no siendo necesario completarlos para poder reproducirlas.

El interfaz del programa es muy simple de manejar (ver figura 5.8): tan sólo hay que mover las barras deslizadoras hasta conseguir tener un paso estable, luego se graba éste y automáticamente se pasa al paso siguiente. Una vez en él, hay que ajustar los motores hasta encontrar la siguiente posición estable y volver a grabar para pasar al siguiente paso. Así se hará consecutivamente hasta que se llegue a la situación inicial o lo que es lo mismo, al paso uno. Esto siempre deberá ocurrir ya que no es lógico almacenar infinitos pasos para hacer que el robot avance, es más lógico tener un número determinado, de manera que al repetirse produzca el efecto de avanzar, girar o retroceder. Al ejecutar una secuencia, el programa mandará las órdenes a la red de control para situar los motores en los valores indicados por el primer paso; luego y tras esperar un pequeño tiempo se enviará la orden para situar los motores en los valores del segundo paso y así sucesivamente, hasta llegar al final. Según el modo de ejecución, que se haya seleccionado, en este momento se parará

[5]Una barra deslizadora es un instrumento de control consistente en un segmento rectilíneo con un pequeño cursor incorporado. De manera que al desplazar el cursor a lo largo del segmento un contador se va incrementando, si se desplaza en sentido contrario el contador se decrementa.

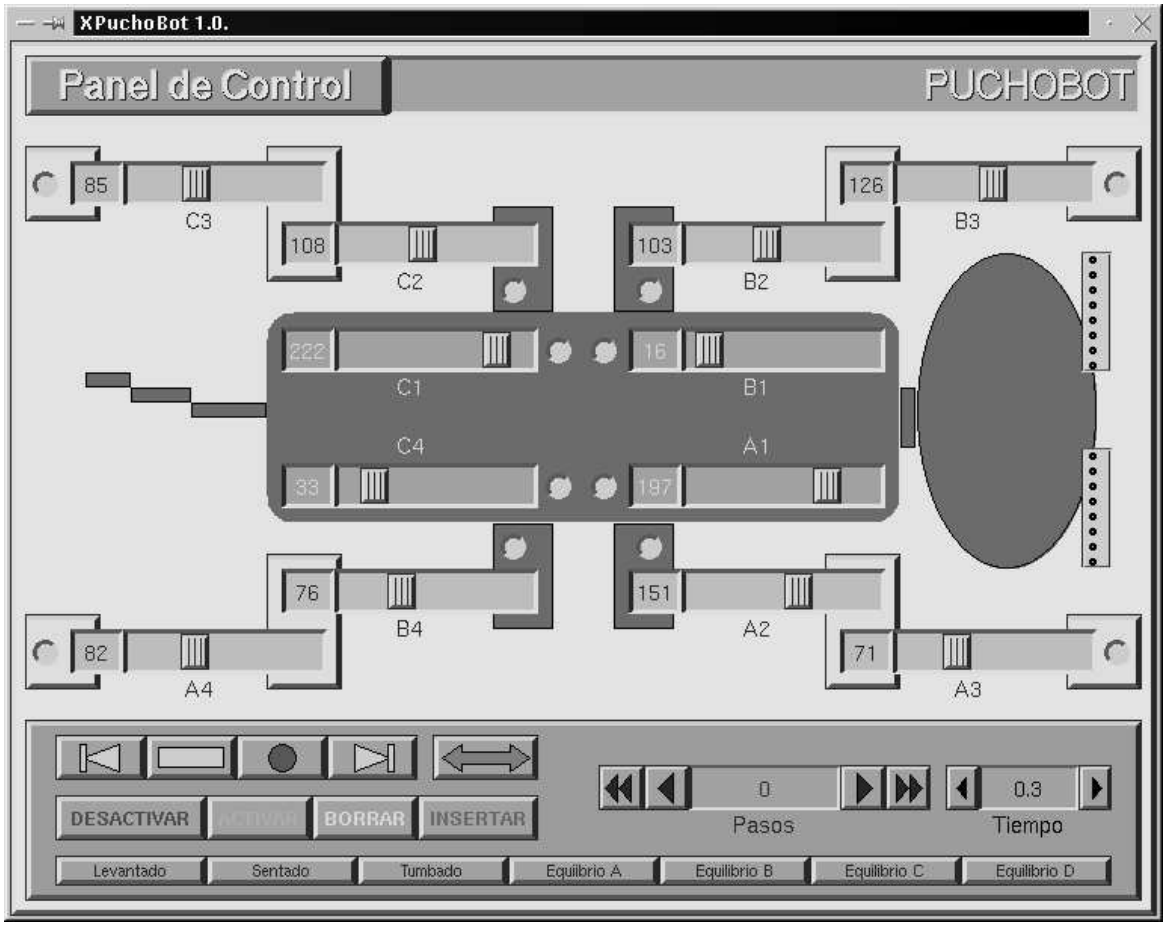

Figura 5.8: Interfaz gráfica del programa generador de secuencias: XPucho.

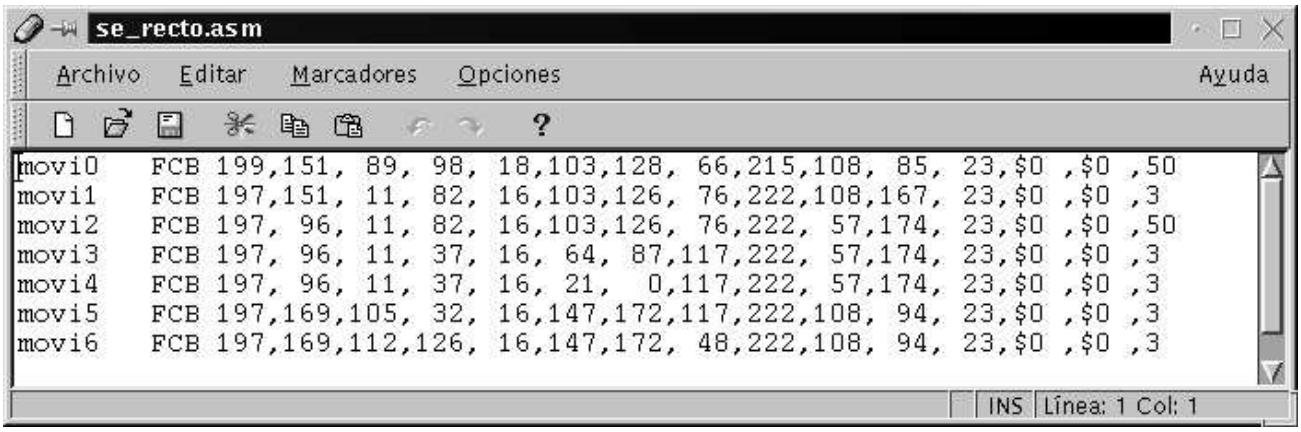

Figura 5.9: Fichero obtenido con la función *Convertir.*

el robot o se volverá al principio para repetir la secuencia. El tiempo entre pasos es fundamental: se podrá configurar para cada paso y significa el tiempo que se ha de esperar en un paso hasta poder pasar al siguiente. La razón es permitir que los motores puedan llegar a su posición final ya que, en caso contrario, el programa estaría mandando tramas continuamente y a alta velocidad, lo que produciría que los motores estuviesen cambiando de posición antes de llegar a sus posiciones finales, es decir posiciones intermedias que nada tienen que ver con las grabadas.

El programa tiene bastantes funciones distribuidas en los siguientes módulos:

1. **Botonera de control**: Compuesta por los controles situados en la parte inferior encuadrados por un rectángulo verde. Mediante estos, se grabarán los pasos para luego poder seleccionar distintos modos de reproducción. También hay funciones que permiten seleccionar el retardo entre pasos, ir a un paso determinado, borrar, etc.

2. **Menú desplegable**: Se encuentra oculto bajo el botón *'panel de control'* situado en la esquina superior izquierda. Para acceder a él hay que situar el ratón sobre el botón y apretar el botón izquierdo o pulsar directamente la tecla F1. Las funciones encontradas son '*abrir fichero*', '*guardar fichero*', '*convertir fichero*' y '*salir del programa*'.

3. **Panel de control**: Se trata de la zona central de la pantalla, en la que se representa la estructura del robot, junto con los operadores que permiten modificar su estado. En concreto hay doce barras deslizadoras que permiten mover los doce servomecanismos; unos botones que sirven para activar o desactivar los motores y dos barras de leds que sirven para configurar los ojos. Todos los operadores se controlan con el ratón.

De todas las funciones del programa hay una que merece una atención especial. Es la orden **Convertir** del menú desplegable. Mediante ésta se puede guardar la secuencia encontrada en un fichero ASCII y en formato ASM (ver figura 5.9), es decir lista para ser insertada en los programas realizados en ensamblador. Esta función es la que se ha utilizado para facilitar la programación de la segunda función del programa maestro. Por eso se dijo que antes de poder terminar dicho programa había que realizar el *generador de secuencias*.

Botonera de control

1. Los botones *levantado, sentado, tumbado, equilibrio A, equilibrio B, equilibrio C y equilibrio D* situados en la línea inferior colocan al robot en una determinada posición. Son útiles a la hora de programar secuencias ya que en ellas hay pasos que son siempre los mismos, como por ejemplo tener el robot levantado sobre sus cuatro patas. Por eso en lugar de ir ajustando una por una las barras deslizadoras se pulsa directamente el botón *levantado* y automáticamente se ajustan todas las barras en la posición correcta.

2. El botón *Desactivar* manda la orden de apagar los servomecanismos de todos los módulos esclavos. Esto significa que las señales de PWM no llegarán a los motores, pero el microcontrolador seguirá produciéndolas internamente. Por lo tanto cualquier modificación de las barras deslizadoras hará que el ancho del pulso del motor asociado cambie y por lo tanto cuando se vuelva a activar el motor cambiará la posición.

3. El botón *Activar* manda la orden inversa a la anterior, hace que todos los servomecanismos pasen al estado activo. Si durante el tiempo que han estado desactivados se ha variado alguna de las barras deslizadoras, al pasar al estado activo se actualizarán las posiciones de los motores.

4. El botón *Borrar* elimina el paso actual. La forma de hacerlo es adelantando una posición todos los pasos empezando por el paso siguiente al actual. Es decir, suponiendo que se elimina el paso 2, los valores que tenía el paso 3 pasan al 2, los del 4 al 3 y así sucesivamente hasta el final.

5. El botón *Insertar* hace lo contrario al anterior, inserta un paso en la secuencia. La forma de hacerlo es copiar los valores del paso actual en el siguiente, los que tenía el siguiente en su siguiente y así sucesivamente. Por ejemplo, si se inserta un paso en la posición 2, los valores de ésta se copian en la 3, los que tenía la 3 pasan a la 4 y así hasta el final.

6. En la línea superior se encuentran los botones asociados a la reproducción de secuencias: *reproducción hacia atrás, parada, grabar, reproducción hacia adelante.* Mediante el botón *Grabar,* representado por un círculo, se guardan todos los valores de las barras deslizadoras en el paso actual y automáticamente se pasa al siguiente paso. Se puede apreciar esto en la barra deslizadora de nombre *pasos* que indica el paso actual. El botón *Parar*, representado por un rectángulo, hace que se detenga la ejecución de una secuencia y se coloca como paso actual el que previamente estaba seleccionado antes de empezar la ejecución. El botón de *reproducción hacia atrás,* representado por '<', ejecuta la secuencia desde el paso actual hasta el primero y el botón de *reproducción hacia adelante*, representado por '>', la ejecuta desde el paso actual hasta el último.

7. El botón de *reproducción cíclica*, representado por una doble flecha <->, permite probar el movimiento del robot al ejecutar repetitivamente una secuencia. Para ello se definirá el paso inicial, el paso final y el número de subpasos intermedios, ver figura 5.10. Este último valor sirve para reproducir secuencias a cámara lenta. El procedimiento es el siguiente: Entre dos pasos consecutivos se insertarán tantos subpasos como se haya indicado, cada subpaso introduce una pausa de 0.3 seg por lo que la

Figura 5.10: Panel de reproducción cíclica.

impresión final es que el robot está moviéndose a cámara lenta. Cuando se llega al paso final se vuelve al primer paso para repetir la secuencia, siendo el botón de parada la única forma de detenerlo.

8. Por último quedan dos barras deslizadoras por explicar: ***pasos*** y ***tiempo***. La primera permite seleccionar el paso actual, es decir aquél cuyos valores se transmiten a las barras deslizadoras. Mediante las flechas de los lados se puede avanzar y retroceder por la secuencia, de manera que al ir cambiando de paso las barras deslizadoras se ajustan a los nuevos valores y si el robot está conectado, los motores cambian de posición. La segunda barra ajusta el tiempo que hay que esperar entre paso y paso una vez que se está reproduciendo la secuencia. El valor predeterminado es de 0.3 segundos.

Menú desplegable

Se encuentra oculto bajo el botón *'panel de control'* situado en la esquina superior izquierda. Para acceder a él hay que situar el ratón sobre el botón y apretar el botón izquierdo o pulsar directamente la tecla F1. El menú tiene varias funciones las cuales son accesibles mediante el ratón o pulsando la tecla equivalente a la primera letra de cada una. Las funciones encontradas son:

1. **Abrir fichero**. Permite cargar en memoria una secuencia previamente guardada en un fichero. Ver figura 5.11.

2. **Guardar fichero**. Guarda en un fichero la secuencia que está en memoria.

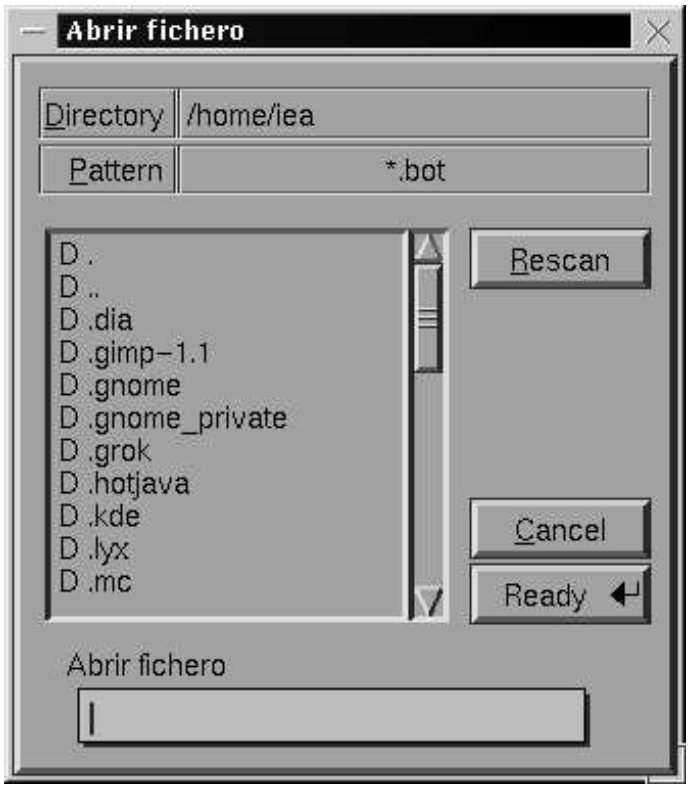

Figura 5.11: Pantalla de selección de ficheros para abrir, guardar o convertir.

3. **Convertir fichero**. Permite guardar en un fichero la secuencia que está en memoria pero en formato ASM, muy útil para insertarlo en programas realizados para el microcontrolador 68hc11.

4. **Salir del programa**. Mediante esta opción y previa confirmación se abandonará el programa. Ver figura 5.12.

Panel de control

El panel de control contiene los elementos necesarios para actuar sobre el perro robot. Para ofrecer un interfaz intuitivo se ha representado la estructura del perro mediante barras deslizadoras, ver figura 5.8. Cada una de ellas tiene un código que indica cual es el

Figura 5.12: Pantalla de confirmación de salida del programa.

motor que controla. Recordando la estructura final del robot (ver figura 4.12) se comprueba que dichos códigos coinciden con los de los motores y además hay una analogía en la representación. De tal forma que la articulación inferior de la pata delantera derecha que está formada por el motor A3, es decir el tercer motor conectado al módulo esclavo A, está representado en la figura 5.8 por la barra deslizadora de nombre A3. Además, en cada barra aparece un número comprendido entre 0 y 255. Su valor es el byte de posición que se ha mencionado en el protocolo de control (al principio de este capítulo) y es el que se envía a los módulos esclavos para, a partir, de él calcular el ancho de la señal de PWM y con ello posicionar el motor en el ángulo correcto.

Asociada a cada barra existe un pequeño interruptor con forma de círculo que puede tener dos colores, verde o gris. En el primer caso indica que el motor asociado a la barra está activo y por lo tanto al moverla el motor girará. En el segundo caso indica que el motor no está activo y al mover la barra se quedará quieto. Las opciones de la botonera de control *activar* y *desactivar* actúan sobre todos los conmutadores a la vez.

Por último en la cabeza del robot hay colocadas dos barras de leds que representan los ojos del perro. Mediante el panel de control se pueden seleccionar los leds que deben encenderse en cada paso. La forma de hacerlo es muy sencilla, mediante el ratón se colocará el puntero en cualquiera de los mini círculos y apretando cualquier botón del ratón se encenderá o apagará el led correspondiente.

A continuación se propone un ejercicio práctico para ir familiarizándose con el software.

5.5.2 Ejemplo de utilización de XPucho

Antes de empezar a probar las secuencias hay que encender el robot. Para ello se pondrá el switch 2 del módulo maestro arriba y los demás abajo, luego se encenderá la alimentación de los motores y la de la electrónica. Una vez hecho esto se arrancará el programa Xpucho.

Nada más arrancar el programa los motores estarán desactivados y la posición del robot puede ser cualquiera. Por eso lo primero que hay que hacer es pulsar el botón *levantado* y luego el de *activar* los motores. Con este procedimiento el robot levantará sobre sus cuatro patas. Si no se ha llegado a este punto hay que comprobar que el robot esté enchufado al PC mediante el cable serie y que esté correctamente encendido. Se puede asegurar el encendido de la electrónica pulsando los *resets* de todos los módulos nada más activarlo. A continuación se creará una secuencia que haga que el robot salude moviendo una pata:

1. Estando el perro levantado sobre sus cuatro patas y en el *paso_actual 0* se pulsará el botón de grabar (círculo rojo). Con esto se asocia al paso 0 la postura de levantado sobre las cuatro patas. La posición de partida es muy importante ya que en este ejercicio todas las patas se quedarán quietas excepto una y el robot tendrá que mantener el equilibrio sobre esas tres. Un truco para facilitar esto es situar las patas oblicuas sobre el terreno, es decir abrir un poco las patas del robot.

2. Al grabar automáticamente el *paso_actual* se ha incrementado en una unidad. Ahora hay que mover la barra deslizadora B3 hasta que la articulación se sitúe paralela al suelo y se pulsará otra vez el botón de grabar. Con esto el paso 1 mantiene el robot

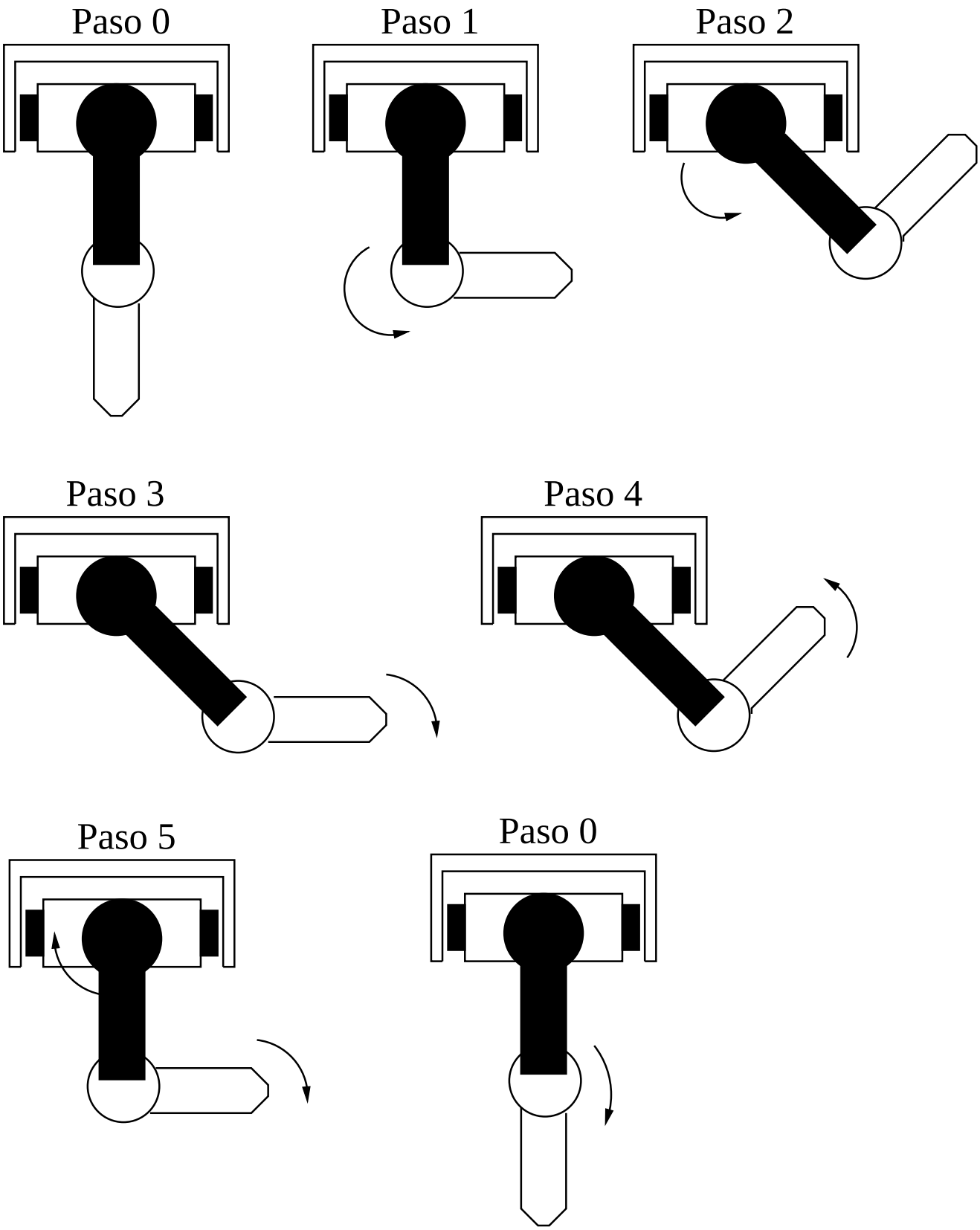

Figura 5.13: Diagrama de los pasos para configurar la secuencia *saludo*.

levantado sobre tres de sus patas y la cuarta tiene la última articulación elevada. Ver la viñeta 2 de la figura 5.13.

3. Los siguientes pasos se obtienen igual que el anterior pero copiando la posición de las viñetas. Una vez que se hayan grabado todos los pasos se procederá a la ejecución de la secuencia.

4. Ahora se ejecuta la secuencia de varias maneras. La primera es paso a paso, para ello se pondrá el contador de pasos a cero y se irá avanzando por los pasos de la secuencia incrementando el valor de la barra deslizadora. La segunda forma es realizar lo anterior de forma automática, para ello una vez situados en el paso 0 se pulsará el botón de reproducción hacia adelante. Por último está la reproducción cíclica que permite ver el efecto al repetir continuamente la secuencia. Para ello se pulsará el botón de reproducción cíclica, se ajustarán los parámetros siguientes: valor 0 en el paso inicial, el valor 5 en el paso final y número de subpasos a uno. Se aceptan los valores e inmediatamente después el robot empieza a saludar con la pata. Para parar hay que usar el botón de parada.

5. Cuando se prueben secuencias más complicadas es útil poder reproducirlas a cámara lenta mediante la opción de reproducción cíclica pero con un valor mayor en el número de subpasos, por ejemplo 5. Al aceptar se aprecia que el robot realiza la misma secuencia pero más lentamente, da la impresión de haber multiplicado el número de pasos.

6. Si nos gusta el movimiento que se ha conseguido se puede grabar la secuencia para reproducirla en otro momento. Para ello utilizar la tecla F1 y elegir la opción Guardar fichero. Tras indicar un nombre y un directorio, aceptar y ya se habrá grabado la secuencia en un fichero. Con la opción abrir se puede recuperar.

7. Para salir del programa pulsamos F1 y luego 's'.

Este simple ejemplo sirve para demostrar la utilidad del software y cómo sin él no se hubieran podido programar las secuencias de movimiento. Se aconseja al lector que practique intentando realizar movimientos simples, como saludar, sentarse, levantarse, etc. En el capítulo siguiente se expondrán una serie de conclusiones obtenidas al programar el robot y que afectan mucho a la cantidad de movimientos que el robot puede realizar.

5.5.3 Explicación del código fuente de XPucho

El programa tiene una estructura peculiar debido a la herramienta utilizada para su desarrollo. Antes de nada es necesario que el lector comprenda cuáles han sido los pasos para su elaboración. En concreto primero se utilizó el programa *fdesign* para definir el interfaz gráfico, es decir mediante las opciones que proporciona este programa se colocaron los botones, las barras deslizadoras, el texto y todo lo que conforma la apariencia externa de Xpucho. El *fdesign* a partir de ese dibujo y de una serie de parámetros crea el primer módulo del escrito en C y que ahorra toda la etapa de programación gráfica. El segundo paso fue asociar una función a cada barra, botón y opción de menú, de manera que al pulsar cualquiera de ellos el programa llame a la función asociada. Todas esas llamadas se

incluyen en el módulo anterior pero su implementación no, por lo que es necesario crear un segundo módulo del programa con esos contenidos. Por eso se puede considerar a éste como el principal, debido a que en él están definidas realmente las funciones del programa.

Otra parte del programa que se ha simplificado ha sido la gestión de las comunicaciones serie del PC. La forma de hacerlo ha sido utilizar la librería *lcts*[6] desarrollada específicamente para controlar la tarjeta CT6811 (módulo maestro) desde ordenadores PC con el sistema operativo Linux. Funciones como definir los baudios, hacer un reset software de la electrónica, mandar y recibir bytes están ya implementadas, siendo inmediata su utilización.

Para compilar todo este conjunto de librerías y módulos es recomendable crear un fichero *makefile* de forma que con una única instrucción se compile todo. En concreto: *make -f xpucho.mak.* Además se aprovecha este fichero para incluir las restantes librerías necesarias para la compilación, éstas son las propias de linux. Al final los archivos que forman el programa son:

panel.c: Módulo que implementa el interfaz gráfico de usuario.

panel.h: Interfaz de prototipos del módulo *panel.c*.

panel.fd: Descripción del panel en formato *fdesign*. No forma parte del código propiamente dicho pero es necesario para poder modificar el panel en versiones futuras.

xpucho.c: Módulo que implementa las funciones principales del programa, es decir aquellas que controlan el robot.

xpucho.h: Interfaz de prototipos del módulo *xpucho.c.*

xpucho.mak: Es el fichero *makefile* que contiene todas las declaraciones para compilar el programa. En el se especifican las dependencias, las librerías y la forma de borrar el programa.

Esta claro que para que funcione el programa es necesario tener instaladas las X, las Xforms y la librería CTS. Las dos primeras vienen en las distribuciones de Linux y la tercera como ya se ha dicho está disponible en la WEB de Microbótica [10]. Además en algunos sistemas es necesario incluir en el directorio de trabajo los ficheros **serie.h**[7] y **forms.h**[8] debido a que no se encuentran en el PATH de includes y al compilar aparece un error. Si esto no ocurre no hay que preocuparse por su situación y se pueden obviar. Si aparece un error entonces se recomienda copiarlos en el directorio donde estén el resto de programas y volver a compilar.

Todos los listados de los programas están en el CDROM que se adjunta con el proyecto.

[6] El autor ha participado en su creación y está disponible en www.microbotica.es en la sección 'download software'.

[7] Este fichero pertenece a la librería *cts*.

[8] Este fichero pertenece a la librería *xforms*.

Figura 5.14: Interfaz gráfica del programa Control.

5.5.4 Puchomovil: Programa de Control

Este es otro programa de ayuda para el PC, en esta ocasión se trata de un pequeño programa que permite mover el robot en todas direcciones sin necesidad de cargar previamente las secuencias de movimiento. La utilidad del mismo es telecontrolar el robot y probar su comportamiento ante adversidades, como pueden ser desniveles o cambios de material en el terreno.

Para realizar el programa se han usado todas las herramientas anteriores, incluido el programa Xpucho que ha permitido obtener las secuencias que mueven el robot. El funcionamiento es muy sencillo: en la derecha hay una serie de botones que al ser pulsados con el ratón hacen que se mueva el robot en la dirección indicada, por cada pulsación se ejecuta una maniobra. En la izquierda están situados otros botones que no ejecutan secuencias de movimiento sino que dejan colocado el robot en diferentes posiciones.

Al ejecutar el programa el robot estará desactivado por lo que será necesario pulsar el botón *Activar* para poder moverlo. Mientras que el botón se mantenga apretado el robot estará activo, y en cuanto el botón se desactive el robot se parará. Por último hay que decir que en este programa no hay menú de opciones y para salir del mismo hay que utilizar el botón 'Salir' o el icono del marco de la ventana.

El interfaz gráfico se muestra en la figura 5.14 y el código del programa está en el CDROM adjuntado con el proyecto.