D ISEÑO, CONSTRUCCIÓN Y CONTROL DE UN ROBOT ARTICULADO MEDIANTE UNA RED DE MICROCONTROLADORES

10 de febrero de 2001

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

La evolución de la robótica

1.1 Marco general del proyecto

Desde que en 1917 el escritor Karel Capek[1] usara el término robot para referirse a unas máquinas en forma de humanoide, han tenido que aparecer numerosos avances tecnológicos y pasar más de medio siglo, para que el autor de este proyecto haya podido realizar un robot articulado en un espacio de tiempo relativamente corto y con un presupuesto mucho más reducido.

En el libro *Robótica Industrial* [1] se define la robótica como una ciencia aplicada que surge de la combinación de la tecnología de las máquinas-herramienta y de la informática. Una máquina-herramienta se define como una máquina que efectúa cualquier trabajo manual, y la informática como la ciencia del tratamiento automático y racional de la información. Uniendo ambos conceptos la robótica surge al automatizar de manera racional las máquinas-herramienta, es decir al permitir que un programa informático controle las operaciones que antes realizaba un operario. Ligado a la robótica aparece el robot, si lo primero es la ciencia lo segundo es el objeto. Se considera la aparición del primer robot, en su concepción moderna, como la unión del control numérico y de la telequérica. El control numérico fue una de las primeras formas de la informática y la telequérica es la ciencia que estudia los manipuladores controlados a distancia por un ser humano, o teleoperadores, que se pueden considerar como una máquina-herramienta avanzada. A partir de aquí la evolución de la robótica ha estado íntimamente relacionada con el desarrollo de la tecnología eléctrica e informática, todo en la aparición de nuevos y mejores sistemas de control. En 1959 se introdujo el primer robot comercial por Planet Corporation, estaba controlado por interruptores de fin de carrera y levas. En 1971 la Universidad de Stanford desarrolló el *Stanford Arm*, un pequeño brazo robot de accionamiento eléctrico. En 1981 la Universidad Carnegie-Mellon diseñó un robot que utilizaba motores eléctricos situados en las articulaciones del manipulador sin las transmisiones mecánicas habituales. Las primeras aplicaciones prácticas de los robots se encuentran en la industria y sobre todo en la del motor. En 1961 la Ford Motor Company utilizó uno para controlar una máquina de fundición en troquel y en 1974 Kawasaki instaló otro para soldar las estructuras de las motocicletas.

[1] Karel Capek utilizó en su obra 'Rossum's Universal Robots' la palabra 'robota', que en checo significa trabajador forzado. Cuando este libro se tradujo al inglés apareció el término ROBOT.

En paralelo, los lenguajes de programación de robots fueron mejorando, y en la exposición 'Robots 8' se mostraron los primeros lenguajes de programación, que permitían el desarrollo de programas de control utilizando gráficos interactivos en un ordenador personal que luego se podían cargar en el robot.

Al ser la industria la que adquirió la mayoría de ellos, una de las primeras definiciones del término robot la proporcionó el Robotics Industries Association (RIA) y vino a decir:

> Un robot industrial es un manipulador multifuncional reprogramable diseñado para desplazar materiales, piezas, herramientas o dispositivos especiales mediante movimientos programados variables para la ejecución de una diversidad de tareas.

Bajo el punto de vista anterior es fácil comprender la definición dada por la RIA, aunque actualmente se podría decir que se ha quedado pequeña, ya que en el 2000 existen muchas más aplicaciones que no responden a las características anteriores y no por ello dejan de ser robots. Además, se está produciendo un cambio en la actitud de la sociedad respecto a la robótica, todo por la sucesiva aparición de aplicaciones que, de no haber sido por ella, no podrían haberse llevado a cabo. Al principio existió un sentimiento generalizado de oposición al robot, se consideraban más enemigos que amigos, siendo la razón principal el temor a ser sustituido. Evidentemente esta opinión tiene su fundamento en un momento puntual de la historia y en todas aquellas personas que perdieron sus puestos de trabajo al introducir los robots en las fábricas. Pero ahora se puede decir que este hecho permitió la evolución de los trabajos, ya que se eliminaron las tareas repetitivas que no permitían un enriquecimiento *mental,* y se crearon nuevos puestos dentro de las empresas, que permitían una participación mayor del empleado, siendo su trabajo más humano que repetitivo. No obstante, no se quiere entrar a discutir en este proyecto si la robótica ha sido perjudicial o no desde un punto de vista social.

Otro concepto era la forma que tenía que tener un robot. Al principio la mayoría opinaba que un robot tendría un aspecto humanoide, con pies, manos, cabeza, más o menos inteligente y que estaba al servicio de su amo. Si hubiese que buscar un motivo de esto se podría decir que fueron las películas, libros, series de TV, etc., las que emplearon la figura del humano y le pusieron un cerebro electrónico. Por ejemplo, se podrían citar las numerosas obras del escritor Isaac Asimov que tenían como protagonistas a robots. Fue este escritor el que enunció las tres leyes de la robótica [2] , en ellas se puede ver la preocupación inicial del hombre en mantener el control de los engendros mecánicos, como si hubiese un temor a una revolución de una raza mecánica. Este argumento se ha usado en infinidad de películas y cuentos, teniendo un ejemplo clarísimo en el robot *TERMINATOR*[3]. Las tres leyes de la robótica, enumeradas más abajo, sirvieron en un principio para discutir el futuro amenazador de los robots, pero dado que todavía no se ha alcanzado dicho momento, se han usado más para escribir novelas que para construir robots, por lo menos hasta que estos no alcancen un grado de desarrollo, hablando de inteligencia, mucho mayor del actual.

1. Un robot no puede herir a un ser humano ni dejar de intervenir si éste está en peligro de sufrir algún daño.

[2] 'Runaround' historia escrita por Isaac Asimov y publicada por primera vez en el número de marzo de 1942 de la revista 'Astoundig Science Fiction' (línea 7, página 100). Editor John Campell.

[3] 'The Terminator'. Película dirigida por James Cameron en 1984. Orion Pictures International.

2. Un robot debe obedecer las órdenes de los seres humanos, salvo cuando éstas contradigan la primera ley.

3. Un robot debe proteger su propia existencia, siempre y cuando dicha existencia no contradiga las leyes primera y segunda.

Actualmente, la robótica sigue avanzando con paso firme, pero con un ritmo menor que el previsto por los expertos en los años 80. Además, la sociedad ha asumido el papel de estos, considerándolos como herramientas, sin una forma típica y con una inteligencia mínima en comparación con la del ser humano. La evolución de la tecnología ha permitido desarrollar nuevos materiales, sensores y chips que han facilitado y abaratado considerablemente la construcción de robots. Sin embargo todavía no se ha producido la explosión que ocurrió con los ordenadores personales, que pasaron de ser un privilegio de empresas multinacionales y universidades, a ser un electrodoméstico más dentro de una casa. Es más, es normal que al adquirir un ordenador se le permita al comprador seleccionar los diferentes componentes que lo van a integrar. Y todo se debe al acercamiento que sufrió la informática al público no especializado, todo cuando su uso se volvió más sencillo, barato y con aplicaciones más potentes y flexibles que las tradicionales. Por ejemplo, se puede destacar el paulatino desuso de las máquinas de escribir en favor de las aplicaciones de procesamiento de textos.

La reflexión anterior induce a pensar en una posible revolución de la robótica, pasando a ser un elemento más dentro del hogar, de la empresa y de la sociedad en general. Ésta opinión es una apuesta de futuro, avalada por la aparición a lo largo de estos últimos años, de robots y proyectos muy llamativos, que han facilitado los cambios de opinión de las personas hacia los robots. Ya no se ven como los grandes enemigos sino todo lo contrario: son tratados como herramientas amigas, que tratan de ayudar y de facilitar el trabajo. Esto último es un poco exagerado, seguro que más de una vez se ha desesperado por problemas con su ordenador, coche, etc. y ha pensado que cada día la sociedad lo tiene más complicado. Pero la prueba está ahí, delante de todos nosotros. Sin ir más lejos, en 1997 un pequeño robot móvil aterrizó en Marte permitiendo que millones de espectadores pudiesen apreciar fotografías captadas desde 191 millones de kilómetros de la Tierra. Este pequeño robot llamado Sojourner[4], más conocido como *Pathfinder* (nombre de la misión), recordó los primeros pasos de la llegada del hombre a la Luna y aumentó las esperanzas de que algún día, más cercano que lejano, el hombre llegará a Marte. Otro caso parecido fue el robot sumergible que bajó a 3500 metros de profundidad para introducirse en los restos del *Titanic* con objeto de fotografiarlo. En esta ocasión el robot se llamaba *Jason Junior* y se operaba por control remoto. Los ejemplos anteriores muestran aplicaciones fuera de la industria como tal, y no son las únicas: otros campos como la medicina, aeronáutica y agri-

[4]El todoterreno de seis ruedas y 10,5 kg llamado Sojourner dio su primer paseo por Marte el 5 de julio de 1997, un día después que su nave nodriza, la Mars Pathfinder, aterrizara en una llanura rocosa del hemisferio norte del planeta. Su tamaño era de 62 cm de longitud y 32 cm de altura, fue diseñado para obtener datos de la superficie marciana que luego serían interpretados por científicos en la Tierra. Se le ha calificado de geólogo interplanetario con visión estereoscópica, espectrómetro de partículas alfa, protones y rayos X, con inteligencia suficiente para llevar a cabo las órdenes dadas desde la Tierra, y con la sabia precaución de esperar nuevas órdenes cuando algún inconveniente imposibilitaba o ponía en peligro la ejecución de la misión. Esta información ha sido obtenida del reportaje de William R. Newcott en la revista National Geographic.[12]

Figura 1.1: Concurso de fútbol RoboCup.

cultura se beneficiaron con la llegada de robots telecontrolados para realizar operaciones, aviones espías autónomos y los sistemas de recolección y envasado directo.

Aún así, estos robots son excesivamente caros y su función está muy especializada, para lograr su integración total se necesitará hacerlos cada vez más baratos y generales, de manera que sean accesibles, económicamente, a las personas ajenas a toda esta evolución y que lo que quieren es que les resuelva alguna de sus tareas cotidianas. ¿ No sería fantástico un robot que limpiase el polvo, que cortara el césped y que además vigilase la casa ?. Más de uno invertiría en un artilugio de este tipo. Las dos características anteriores, económico y general, se encuentran en un tipo de robot que poco a poco se está abriendo camino. El lugar dónde hay que ir a buscarlo es el entorno universitario y, más recientemente, en algunas compañías jugueteras. ¿A qué es debido esto?. La razón fundamental se comentó antes: la informática, electrónica y mecánica han pasado de ser un privilegio a ser materia de estudio en muchos centros. La tecnología ha evolucionado tanto que los precios se han reducido mucho. Además construir un pequeño robot se ha convertido en un hobbie para muchos jóvenes, llenos de imaginación y en muchas ocasiones motivados por los concursos que se realizan por el mundo. Uno de los más famosos es ROBOCUP [4], que consiste en un partido de fútbol entre robots y que cuenta con la participación de un equipo español, RoGi Team[5]. En España se están empezando a celebrar concursos, sobre todo en el ámbito universitario de la ingeniería. Tal vez el más conocido sea el torneo que organiza la asociación AESS [5]de la UPC, SumoBot, cuyo objetivo es un combate de sumo entre dos robots[6]. También se producido un efecto curioso, la aparición de concursos ha hecho que la

[5]Más información en http://rogiteam.udg.es

[6]El autor tuvo la suerte de poder acudir al segundo certamen del concurso junto con un grupo de amigos, el Grupo J&J, y la experiencia fue inolvidable. Nuestro robot, llamado Tauro Viper por su aspecto de toro

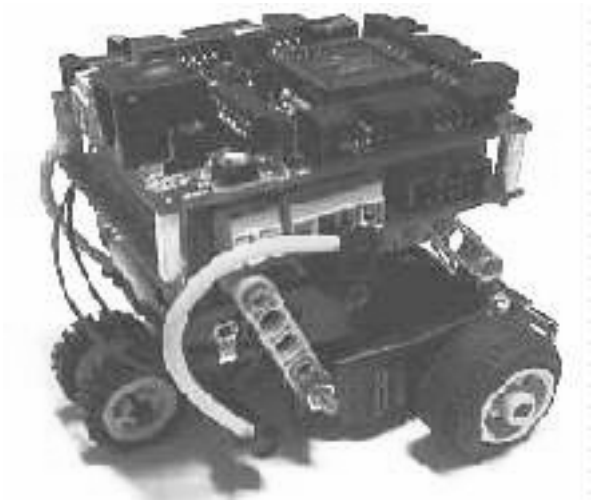


Figura 1.2: Microbot Tritt de la empresa Microbótica S.L.

aplicación práctica de investigaciones informáticas y electrónicas se lleven a cabo mediante pequeños robots que realizan actividades sorprendentes. Uno de esos robots es COG [3], desarrollado en el MIT, que es capaz de jugar al tenis de mesa. En él se ensayan estudios de visión artificial, mecánica, elasticidad en robótica y también actitudes de comportamiento, esto último más ligado a la inteligencia artificial que a la robótica, pero que sirve para darle un carácter más humano. Por último, han sido varias las compañías jugueteras que han sacado kits de aprendizaje o de iniciación a la robótica, la más avanzada ha sido LEGO con su kit LEGO MindStorms[7], pero también hay que destacar a Robotix[8] , Mattel o Bizak[9].

Después de una breve descripción del estado actual de la robótica, se describen las circunstancias que rodean al autor de éste proyecto, por considerarlas de interés para la comprensión del porqué de su realización. El autor cuenta en su haber con la realización de varias estructuras mecánicas, que junto con una electrónica diseñada por compañeros de trabajo, han proporcionado la base para la realización de pequeños robots llamados *Microbots*. Estos se han convertido en un instrumento de iniciación a la robótica, tanto para estudiantes como para cualquier otra persona interesada. Algunos de estos microbots se distribuyen a través de Microbótica S.L., empresa de ingeniería que desde el principio y debido al carácter de sus miembros siempre ha apostado y ha apoyado a este tipo de robótica. Bautizada como *Microbótica*, haciendo alusión a un tipo de robot que si bien es pequeño y sin un alto grado de control, consigue mediante la cooperación un grado de

y la alta velocidad que llegaba a adquirir, luchó hasta el final, y aunque no ganó dió un magnífico espectáculo. Pero lo que más nos llamó la atención fue el ambiente que rodeaba al concurso, los espectadores y el increíble interés que tenía la gente. Un año más tarde se organizó un curso de robótica en la Escuela Técnica Superior de Ingenieros de Telecomunicación y se culminó con la realización de un concurso. En él, volvimos a experimentar la sensación de Barcelona, y quedó demostrado que había bastante por el tema. A raíz de ese concurso se han organizado muchos otros por diferentes universidades españolas.

[7]LEGO es una marca registrada del Grupo LEGO. (http://www.LEGO.com)

[8]Robotix ha sacado en la línea de juguetes 'Learning Curve' el kit RoboDog.

[9]Bizak se diferencia del resto en que directamente ha sacado al mercado un perro robótico controlado mediante mando a distancia, se llama 'OPTIMUS'. La descripción que hacen de él dice 'Se enfada, refunfuña y ladra, mueve cabeza, cola y enciende los ojos'. No es un kit de construcción, sino un juguete para niños de pequeña edad que quieran algo diferente al clásico peluche.

NIVEL DE COOPERACIÓN
NIVEL DE COMUNIDAD
NIVEL DE INTELIGENCIA
NIVEL DE CONTROL
NIVEL DE REACCIÓN
NIVEL FÍSICO

Tabla 1.1: Niveles de la Torre Bot

complejidad bastante elevado.

Dentro de este marco se encuentra el autor, que ve la necesidad de evolucionar y de investigar en nuevas formas de control y de movimiento. De esa evolución surgirán nuevos productos para la empresa y una gran cantidad de información que será accesible vía Internet, permitiendo que el público pueda tener la documentación necesaria para desarrollar sus propios microbots. Los robots anteriormente desarrollados por el autor están dotados de ruedas u orugas, siendo el control de los motores muy sencillo, ya que únicamente se utilizan como motores de continua sin prestar atención a su uso como servo motores. Tan sólo hay un robot más complejo que utiliza los servo motores como tal, pero también aquí es necesario una evolución del software y del control. Por eso, en éste proyecto, hay tres focos de investigación y desarrollo: uno la mecánica, otro los motores y por último la electrónica. Tres pilares fundamentales para la realización de robots, aunque todavía quedarían dos más, el primero haría referencia a los sensores, necesarios para obtener información del medio; y el segundo, al software de alto nivel, que permite dotar al robot de comportamientos complejos. Estos dos últimos niveles no se tratarán en éste proyecto, dejándolos para unos posteriores trabajos. Por lo tanto, lo que se obtendrá al finalizar será un nuevo tipo de plataforma sobre la cual se podrán desarrollar nuevas aplicaciones.

1.2 Torre Bot: Etapas en la construcción de un robot

En el desarrollo de cualquier proyecto siempre hay que tener un orden de actuación determinado, es decir una buena planificación. En este caso, se ha recurrido a una herramienta útil que divide en diferentes niveles o etapas la fabricación de robots. Son los niveles proporcionados por la torre Bot [13]. El nivel inferior *Nivel Físico* representa el primer paso y el superior *Nivel de Cooperación* el último (Ver tabla 1.1). Realmente no es necesario llegar hasta el último nivel, dependerá de la aplicación en concreto. Lo que sí es importante es empezar por el primer nivel para evaluar las necesidades y tener un cierto orden de prioridades. También es evidente que, a medida que se avanza, se puede retroceder para ajustar detalles que hayan quedado pospuestos en niveles anteriores. La información se ha obtenido del documento titulado *Torre Bot*, en la sección 6 del *Manual Técnico de Microbótica* [6]. El significado de cada nivel es el siguiente :

- *Nivel Físico*: Comprende la estructura física, las unidades motoras y las etapas de potencia. Es posible encontrar desde sistemas muy sencillos basados en un único motor, hasta estructuras sumamente complejas que buscan emular las capacidades mecánicas de algunos insectos.

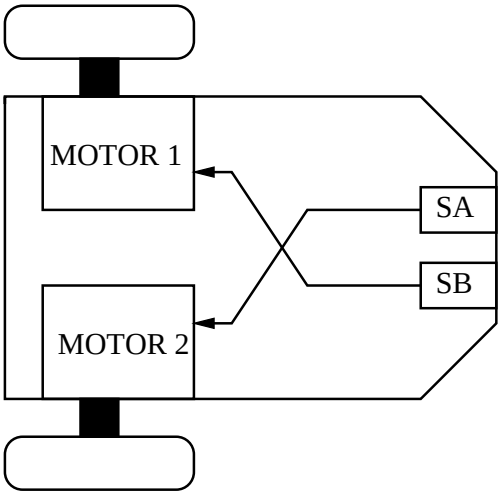


Figura 1.3: Esquema de microbot reactivo.

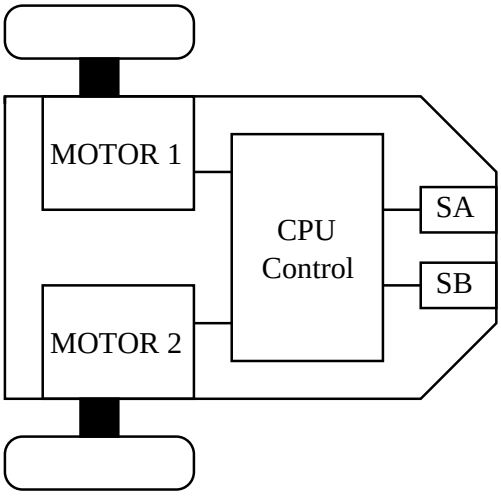


Figura 1.4: Esquema de un microbot con el nivel de control.

- *Nivel de Reacción:* Está formado por el conjunto de sensores y sus circuitos de polarización. Estos transductores cubren un amplio margen de posibilidades, tal que podemos encontrar desdes simples topes de fin de carrera, hasta micro cámaras digitales con sistemas de reconocimiento de imágenes. Un microbot que haya superado en cuanto a su construcción tanto el nivel físico como el de reacción, se denomina microbot reactivo. Este tipo de unidades trabajan cumpliendo la premisa, acción o reacción. En estos casos, los sensores son los propios controladores de las unidades motoras, sin ningún tipo de control intermedio.

- *Nivel de control*: Incluye los circuitos más básicos que relacionan las salidas de los sensores con las restantes unidades. Partiendo de una simple lógica digital hasta potentes microcontroladores, se busca dotar al microbot de la capacidad para procesar la información obtenida por los sensores, así como actuar de una manera controlada sobre las unidades motoras.

- *Nivel de Inteligencia*: Abarca el planificador a largo plazo; en este nivel, se introducen los objetivos del microbot que tienen relativa independencia de los sensores. Éste es el más alto nivel de inteligencia que puede alcanzar un microbot como una unidad individual.

- *Nivel de Comunidad*: Se trata de la puesta en funcionamiento de más de un microbot, dentro de un mismo entorno, de forma simultánea y sin que ninguno de ellos tenga conocimientos explícitos de la existencia de otros en su mismo entorno. A estos recintos se les denomina granjas.

- *Nivel de Cooperación*: Comprende los sistemas donde a partir de un nivel de comunidad, se planifican o programan los microbots para que tengan conocimiento de la existencia de otros, tal que posean la capacidad de cooperar para el buen desarrollo de una tarea.

Para aclarar la torre Bot se expondrá mediante un ejemplo sencillo la diferencia entre los distintos niveles. Antes de nada hay que tener un objetivo fijado: en este ejemplo será la implementación de un microbot que sea capaz de seguir una línea negra trazada en el suelo. Luego se irán aplicando los distintos niveles de la torre Bot para obtener el resultado final. En este caso, el nivel físico comprende una estructura mecánica, dos motores de corriente continua y un driver de potencia que implemente dos puentes en H para controlarlos. El nivel de reacción comprende un par de sensores de infrarrojos mediante los cuales se distingue entre blanco y negro.

Una vez que se ha llegado aquí, ya se puede lograr el objetivo final, tan sólo hay que unir los sensores a los motores como indica la figura 1.3. La explicación del movimiento es fácil; cuando los dos sensores están la línea su salida es la misma (nivel alto), que hace que los motores avancen. Cuando el sensor SA se sale de la línea, su salida se pone a nivel bajo, con lo que se desactiva el motor 2. Al estar el motor 1 encendido hace que el microbot gire a la derecha, volviéndose a introducir el sensor SA. El mismo procedimiento ocurre si se sale el sensor SB, tan sólo que el microbot gira a la izquierda.

El control anterior, aunque cumple los objetivos iniciales, es muy rudimentario y poco seguro. Por ejemplo si se salen ambos sensores, el microbot se pararía. Por eso se añade el nivel de control, mediante el cual se añade un microprocesador, ver figura 1.4, que recibe las señales de los sensores, las procesa y genera las señales para los motores. Ahora se puede realizar un autómata mucho más complejo, siendo recomendable utilizar lógica reprogramable para pasar el problema de la electrónica al software. Por último, se podría añadir el nivel de inteligencia, que consistiría en programar ese microprocesador para tener un objetivo a largo plazo, por ejemplo seguir una línea negra para buscar la salida en un laberinto, para reconocer el trazado realizado, o para luego sin necesidad de cinta realizar el mismo recorrido.

Los siguientes niveles serían el de comunidad y cooperación. Para su puesta en marcha se necesita más de un robot por lo que no interesan en este ejemplo que tan sólo utiliza uno. Si el objetivo hubiese sido formar un equipo de robots para competir en el campeonato de fútbol, se tendría que utilizar el nivel comunidad para planificar el número de robots y las características de cada uno para poder realizar jugadas de peligro. Y luego el nivel de cooperación, para hacer que cada robot se pueda comunicar con sus compañeros para hacer tareas coordinadas. Si no se usase este nivel se podría dar el caso de que dos robots

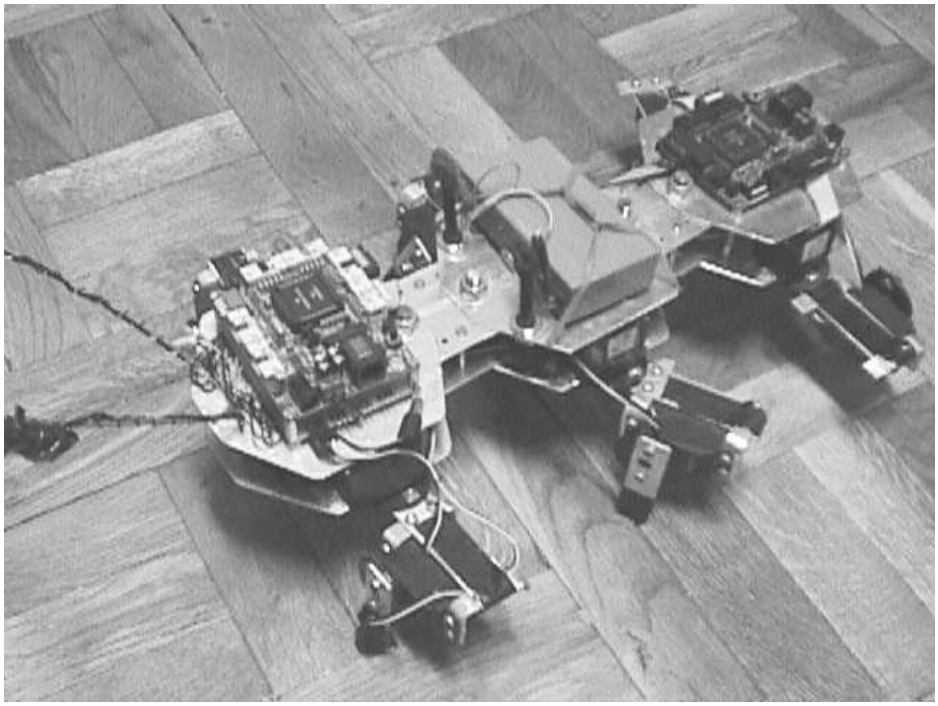


Figura 1.5: Hexápodo desarrollado con anterioridad por el autor de éste proyecto.

del mismo equipo luchasen por el balón, pues en realidad estos no sabrían de la existencia de sus compañeros.

1.3 Planificación del proyecto

La planificación de este proyecto se hizo en base a la utilización de los niveles de la torre Bot, explicados en el punto anterior. Por lo tanto y partiendo de un objetivo concreto, habrá que ir aplicando cada uno de los niveles para saber las necesidades que se tendrán que cubrir durante toda la fabricación. Además, éstas, estarán ya ordenadas permitiendo establecer una planificación muy eficaz para lograr el objetivo.

Como ya se ha dicho el objetivo es desarrollar una plataforma móvil articulada. En concreto se ha pensado en una estructura que se apoye cuatro extremidades, como lo hacen los perros, gatos, caballos, etc. El motivo principal de esta elección ha sido que el autor ya ha realizado con anterioridad estructuras basadas en seis extremidades (figura 1.5) y ahora se pretende una evolución de las mismas. Las estructuras de seis 'patas' tienen la ventaja de poder mantener más fácilmente el equilibrio, en relación a las de cuatro. Por ello, en este proyecto, además de investigar motores y electrónica se procederá a realizar una mecánica más avanzada que cualquiera de las desarrolladas por el autor anteriormente.

1. Con el primer nivel, o **nivel físico** se ve que es necesario una estructura ligera, fácilmente obtenible, lo más económico posible y articulada para permitir que el robot pueda realizar movimientos variados. Se pensó utilizar como material aluminio de un grosor menor a 1.5 mm (en el capítulo 4 se explicará detalladamente). Los motores a utilizar deben ser lo más pequeños y ligeros posibles, deben dar el sufi-

ciente par para que entre todos sean capaces de soportar el peso de la estructura y la puedan mover. La experiencia adquirida por el autor en el desarrollo de este tipos de robots indica que deben ofrecer un par entre tres y nueve kgxcm, siendo además fácilmente adaptable a las estructuras. Por último en este nivel se han de detallar etapas de potencia adecuadas a los motores seleccionados. En resumen, primero se realizar una búsqueda a través de Internet intentanto encontrar diferentes estructuras previamente utilizadas en perro robots, así como los motores empleados y sus mecanismos de control.

2. Segundo nivel o **nivel de reacción**: En este proyecto los únicos sensores a utilizar son los necesarios para conocer la posición de los motores para dejarlos fijos en un determinado ángulo. No se utilizan otro tipo de sensores como son los de seguimiento, luz, temperatura, etc, por no ser el objetivo del proyecto. Pero si en futuro se quisieran ampliar las funciones del robot se debería retornar a este nivel y añadir dichos sensores.

3. **Nivel de control** este es uno de los niveles más importantes del proyecto. Consiste en realizar un sistema de control flexible y modular que permita añadir motores sin necesidad de modificar lo realizado anteriormente. De esta manera primero se pondrán en movimiento las extremidades del perro y en un futuro cabeza, cola, etc. Además con este tipo de sistema se podrán hacer robots con diferente número de extremidades sin necesidad de rediseñar la electrónica de control. Para su realización, se utilizarán dos tipos de tarjetas electrónicas, la primera hará las funciones de cerebro y la segunda de receptora de órdenes. El cerebro sabrá qué motores y en qué orden se deben mover para conseguir que el movimiento sea el adecuado entre, otras cosas. Esta tarjeta, a través de un bus de comunicación, podrá, enviar órdenes a unas tarjetas periféricas esclavas. Éstas son las encargadas de mover un motor a una posición predeterminada, así el cerebro sólo informa y las tarjetas esclavas ejecutan. (Ver figura 1.6).

Los siguientes niveles no forman parte de los objetivos de este proyecto, a lo sumo se incluirá un breve nivel de inteligencia que permita al robot hacer algún movimiento como respuesta a algún estímulo externo. Los niveles de comunidad y cooperación, no entrarán a formar parte del proyecto por necesitar disponer de más de una estructura.

1.4 Estado actual

Internet se ha convertido en una herramienta de búsqueda de información muy potente, debido sobre todo a la gran cantidad de sitios que se pueden visitar y a la velocidad de obtención de dichos datos, pues no hace falta ir a bibliotecas ni pedir informes a distancia. La desventaja que aporta es la falta de verificación de los datos obtenidos: hay que tener en cuenta que cualquiera puede poner una página web con la información que desee y nadie contrasta si es cierta o falsa. Por eso tiene que ser uno mismo el que se preocupe de analizar su fuente. Una forma de asegurarse de que la información presente es cierta, es buscar en páginas que estén avaladas por centros o individuos con un prestigio reconocido.

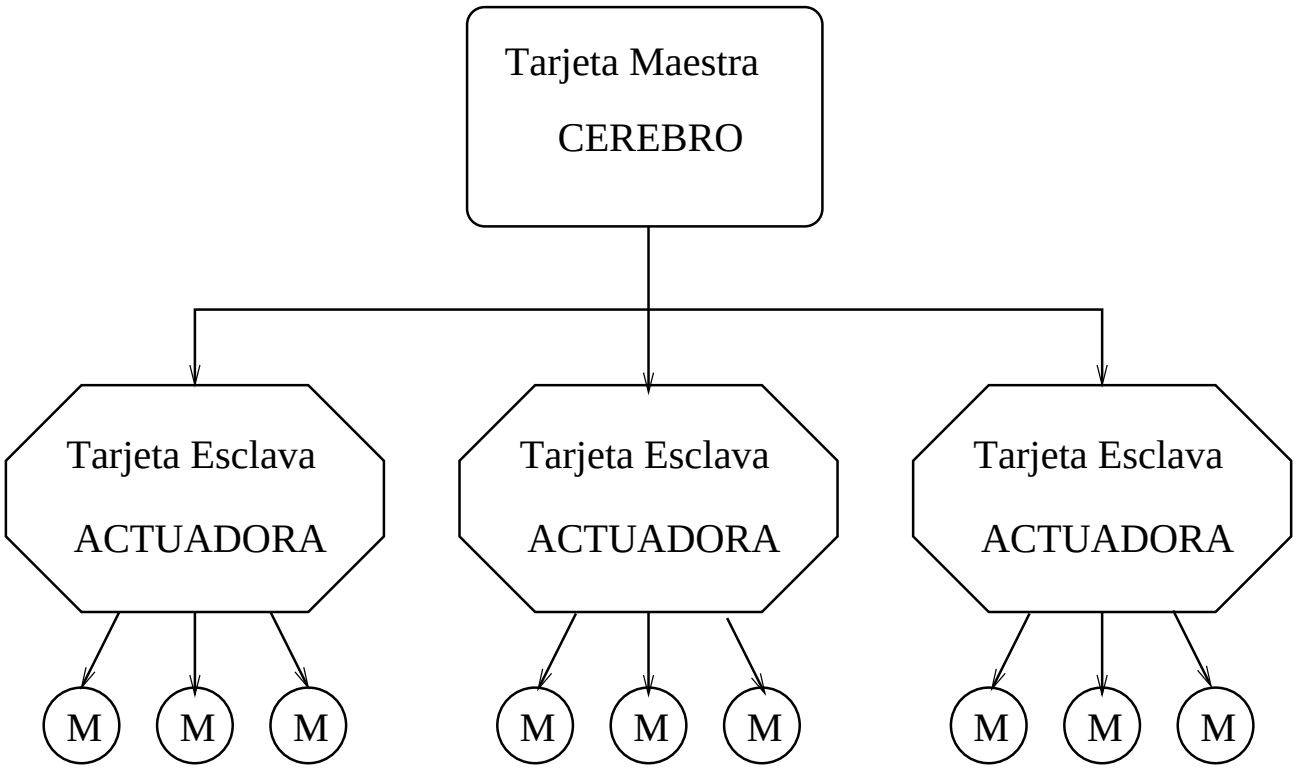


Figura 1.6: Diagrama preliminar de la red de control.

Es este caso se recurrirá a buscar información por los diferentes institutos y empresas de tecnología existentes, tanto en España como en el extranjero

En España encontramos una página ejemplar que corresponde al RogiTeam, mencionado antes, lo malo es que los robots mostrados son de ruedas, por lo que no se podrá sacar más información de la que ya se tiene. Pero es merecedora de una mención por tratarse del único equipo español que participa en la Robocup.

En el extranjero una parada obligatoria es la web del Instituto de Tecnología de Massachuset, o MIT, en la cual se puede iniciar una búsqueda con la palabra clave "robots", para acceder a una lista variada de artículos, robots y concursos. Otras webs visitadas han sido: la universidad Carnegie Mellon, la universidad de Stanford, la compañía Mitsubishi y Honda. En todas ellas hay enlaces y artículos relacionados con la robótica, pero en la mayoría de los casos son bastante teóricos y sin aplicación inmediata para el proyecto que aquí se trata. También se ha visitado la página de IS Robotics, empresa que puede considerarse una de las mejores en robótica, realizando trabajos excelentes para diferentes organizaciones, como por ejemplo la NASA. Esta empresa ofrece en su web bastantes productos y soluciones para dotar a los robots de complejos sistemas de guiado, reconocimiento, sensores, etc. Lo malo es que la información técnica sobre su realización no se encuentra, y además, los robots en su mayoría no son articulados. También es verdad que siendo un poco espabilados se puede encontrar dicha información en las páginas de la universidad del MIT, ya que la empresa y la universidad están muy ligadas.

Principalmente la búsqueda se orientó hacia los robots articulados y de todo lo encon-

Nombre lugar	Dirección
IS Robotics	www.isr.com
Arrick Robotics	www.robotics.com
Mondotronics	www.robotstore.com
Honda	www.honda.co.jp/english/technology/robot
AIBO	www.aibonet.com
MIT	web.mit.edu
RoGi Team	http://rogiteam.udg.es
Carnegie Mellon	http://www.cmu.edu

Tabla 1.2: Direcciones de Internet relacionadas con la robótica.

Figura 1.7: Robot AIBO de SONY.

trado lo más relevante es el robot AIBO de Sony. En un artículo [7] publicado en TechWeb[10] Sony anuncia la creación de una arquitectura abierta para la construcción de robots. En él se explica que la compañía electrónica va ha invertir en un mercado hasta ahora inexistente que consiste en los robots personales de entretenimiento. Y como muestra de ello fabricará una tirada limitada de pequeños perros robot.

En junio de 1999 Sony lanzó al mercado una cantidad limitada del perro robot que se vendieron en una semana con un precio aproximado de 400.000 pts y ya se ha centralizado la información del mismo en una página web: www.aibonet.com. La foto de la figura 1.7 se ha extraído de ella. Examinando el contenido se pueden sacar bastantes conclusiones; la primera es que la tecnología que lleva incorporada el robot está muy por delante de lo que incorporan el resto de robots estudiados. Es verdad que la compañía lleva muchos años de investigación en el tema de motores, chips electrónicos, estructuras mecánicas y sensores. Fruto de esa investigación ha permitido que SONY sea una de las principales multina-

[10]TechWeb es una página de Internet cuya información es una gaceta tecnológica. Su dirección es 'www.techweb.com'.

Característica	Descripción
Operating System	Aperios (Sony's Propietary RT OS)
OPEN-R bus	conector de 10 pines
Datos	2 pines
Velocidad Transmisión	12 Mbps
Reloj	1 pin
Velocidad reloj	12 MHz
Alimentación circuitos	3 pines (incluyendo GND)
Tensión / máxima corriente	3.3v, 1A
Alimentación motores	4 pines (incluyendo GND)
Tensión/ máxima corriente	5v, 2A

Tabla 1.3: Especificaciones de la arquitectura OPEN-R.

cionales de electrónica de consumo. Fabricando desde televisores, vídeos, ordenadores, walkmans, radios, todo tipo de consumibles y ahora mascotas electrónicas. Por esto, este proyecto no puede aspirar a lograr una reproducción exacta de AIBO, pero sí intentar una alternativa que esté al alcance de un público más general y con un coste mucho menor. De todo lo que incorpora el robot hay que resaltar la realización por parte de SONY de un estándar que describe una electrónica modular y un bus específico para robots. Se ha bautizado como OPEN-R, y a pesar de que todavía no hay mucha documentación disponible, en la tabla 1.3 se muestran las especificaciones. Aunque como ya se ha mencionado en la sección de planificación, la estructura que se va a desarrollar, también incorporará un sistema modular y un bus de control. Realmente la decisión de SONY de crear dicho estándar se puede considerar como una afirmación de la necesidad de ese tipo de estructuración electrónica a la hora de hacer robots.

Por otro lado el éxito de la estructura se debe a la utilización de minimotores y de un armazón específicamente diseñado para poder realizar más o menos todos los movimientos típicos de los perros y hay que decir que han conseguido un gran realismo. Pero todo esto se logra mediante la creación de moldes industriales que llevan el coste a unas cifras prohibitivas para nuestra estructura. Se esta hablando de tres millones por molde y echando cuentas de las diferentes piezas se alcanzan los veinte millones rápidamente. Por ello hay que seguir analizando otros robots para descubrir nuevos motores y estructuras. Del proyecto AIBO nos quedaremos con la necesidad de emplear un bus de control y una electrónica modular.

Después del análisis de AIBO hay que seguir buscando y uno de los lugares obligatorios donde mirar es Robot Store[11], que tiene una gran cantidad de kits, materiales y documentación robots. Examinando el catálogo sólo se encuentra un cuadrúpedo, que es en realidad una simplificación de su robot hexápodo y que se muestra en la figura 1.8. En sus modelos hay una simplificación de la estructura mecánica, no se han utilizado moldes sino piezas sueltas de plástico (probablemente PVC). Además en todos ellos se utiliza el mismo tipo de motor, en concreto servo motores de aeromodelismo. Por lo tanto ya se puede descartar la realización de moldes y la búsqueda de motores se puede centralizar en los diferentes tipos dentro de los servos de aeromodelismo.

[11]Robot Store pertence a la compañía Mondotronics. Web: www.RobotStore.com

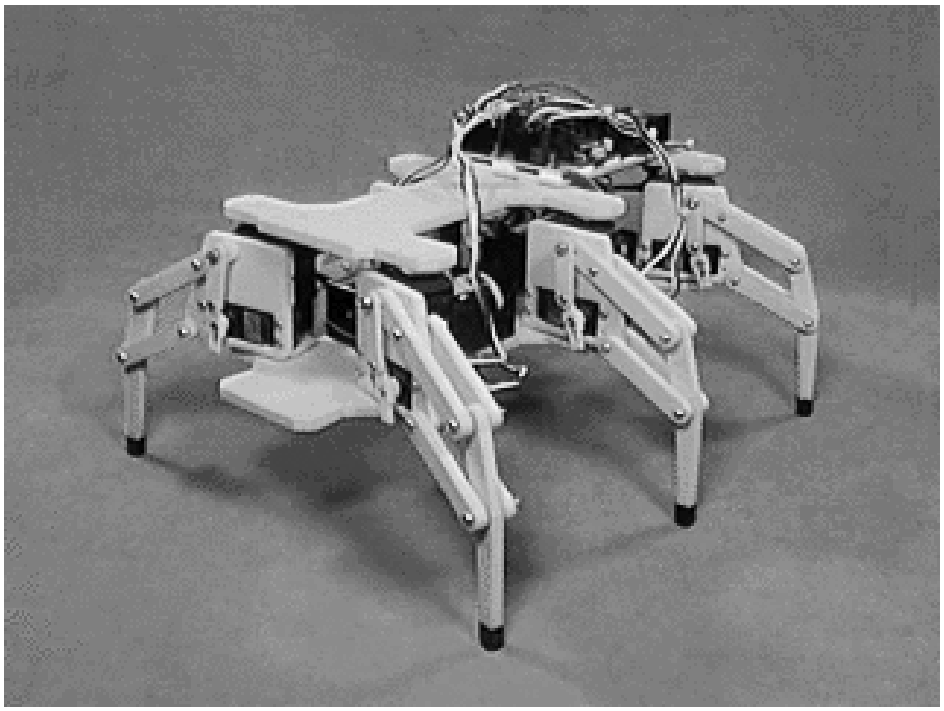


Figura 1.8: Uno de los kits articulados que distribuye Robot Store.

Una de las páginas más interesantes de las visitadas es la de ARRICK Robotics[12] , en ella hay una sección con enlaces a proyectos de robots realizados por particulares. Cada uno presenta una foto del robot, una breve descripción y una dirección donde encontrar más información. A continuación se muestran una serie de fotos las que se sacarán las conclusiones tomadas la estructura y los motores.

El primer robot examinado es *Gokiburi* (ver figura 1.9), un hexápodo que integra 12 motores. El propio autor dice que los motores son servomecanismos, que la tarjeta de control esta basada en el z80, que el material es plexiglass y que la estructura sin baterías pesa 600 gr. Además da razones de las elecciones anteriores. Así, por ejemplo, dice que utiliza servomecanismos porque cumplen una relación par-precio bastante buena. A esto le añadiremos otras dos características, son un estándar e incorporan facilidades para acoplarlos a las estructuras. También hace un análisis de materiales y llega a la conclusión que el aluminio a pesar de ser lo más indicado se descarta por la dificultad de tratarlo frente al plexiglass. Cada pata se levanta y gira lateralmente y la forma de andar es moviendo las patas de tres en tres, de manera que la estructura siempre se apoye tres de ellas. A pesar de no tener antenas tiene un aspecto muy parecido al de una hormiga, la clave de esto es utilizar unas patas largas y las articulaciones pegadas al cuerpo. Este detalle es la principal diferencia con relación al hexápodo fabricado por el autor, ver figura 1.5. Si se hace hincapié en una estructura de seis patas a pesar de no ser objeto de este proyecto es porque va a servir para comentar unos cuantos detalles de los robots siguientes.

En concreto se hace referencia a los robots *BERTA*, figura 1.11 y *Thing*, figura 1.10, que tienen una forma parecida y cuya disposición de motores, a efectos prácticos, es la misma. Ambos son muy parecidos a Gokiburi (figura 1.9) tan sólo que en lugar de tener seis patas

[12] Arrick Robotics: www.robotics.com

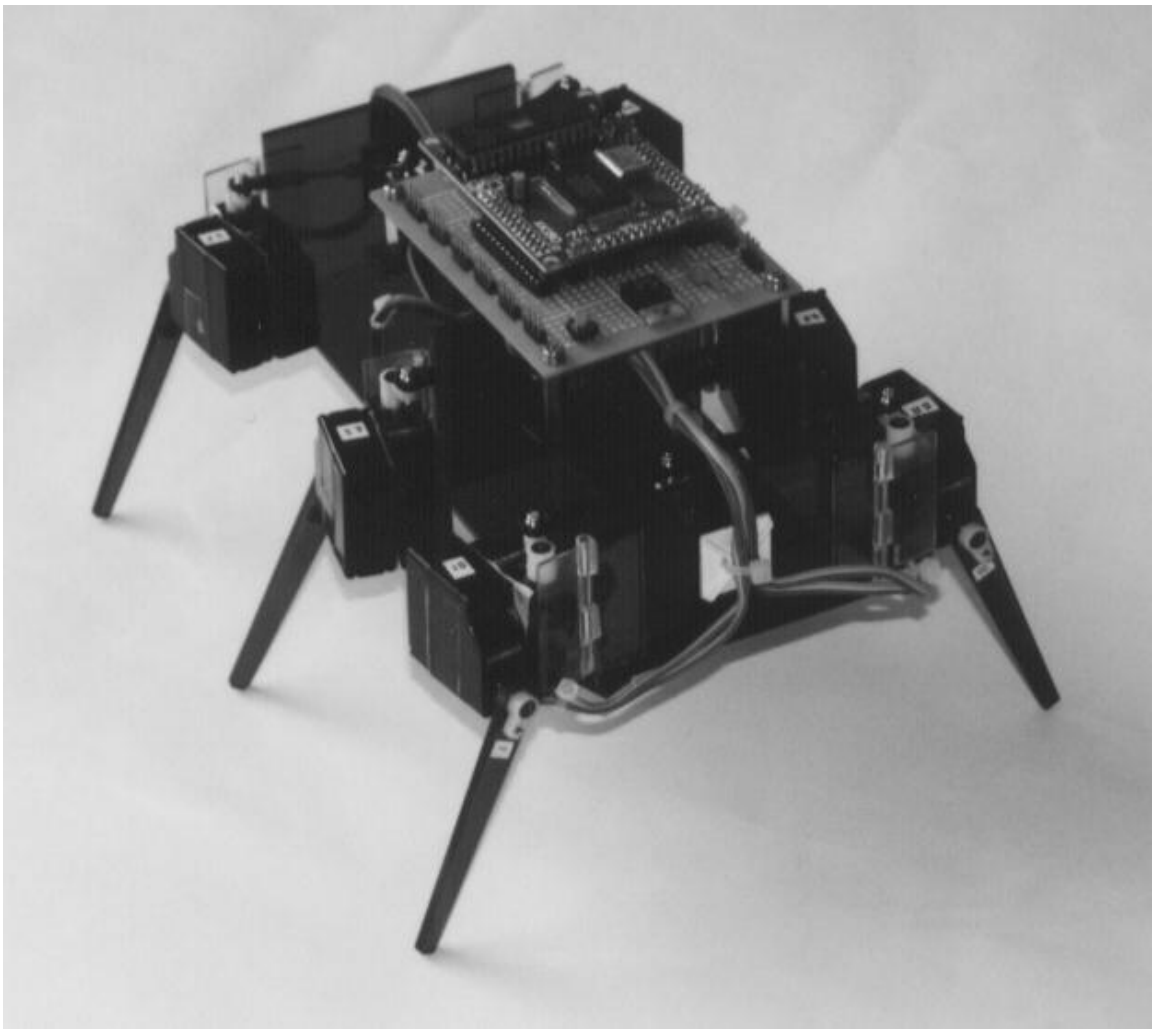


Figura 1.9: Robot “Gokiburi” diseñado por Ahmet ONAT.

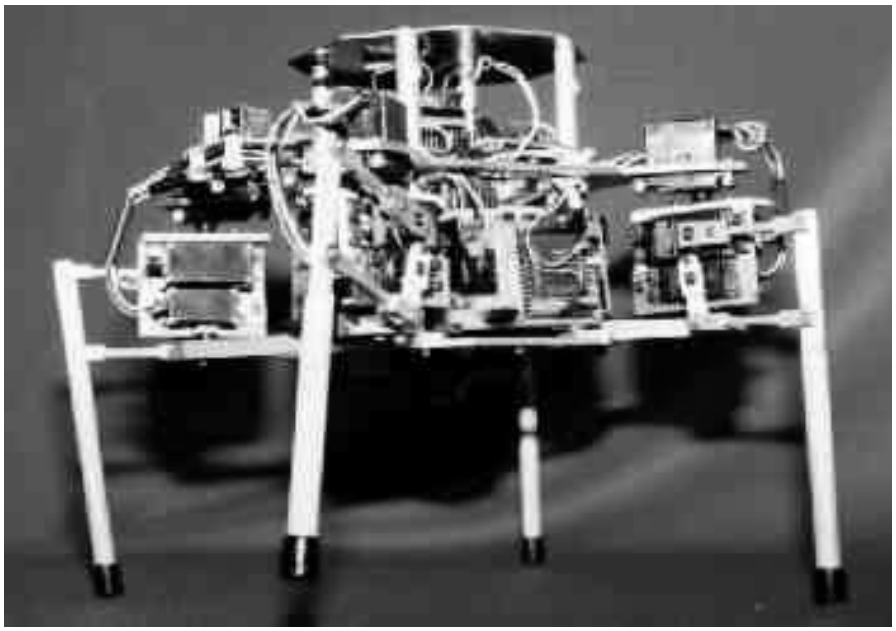


Figura 1.10: Robot "Thing" diseñado por Willad MacDonald.

tienen cuatro. Pero aún así, llevan doce motores y una vez más son servomecanismos. La razón de esto es que al quitar un par de patas se reducen las posibilidades de movimiento y para asegurar una amplia gama de ellos se añade un compañero al motor que levanta la pata. De esa forma cada una llevará tres motores en lugar de dos. Con todo ello se aprecia que cada pata se puede elevar, girar y desplazarse adelante y atrás. Con esa variedad de movimientos se ha creado una estructura que imita bastante bien a una araña, aunque el número de patas sea inferior. Como nota aparte queda mencionar que el control de Thing se realiza mediante una red de cinco microprocesadores, de los cuales hay uno que ejerce de control central y el resto controla cada una de las cuatro patas: vuelve a aparecer una electrónica modular.

Nuestra intención es crear un robot articulado que se parezca a un perro y para lograrlo las patas tienen que parecerse más a las de un perro que a las de un insecto, que es lo que ocurre en los robots anteriores. Se puede decir que partiendo del hexápodo y eliminando un par de patas han llegado a obtener un cuadrúpedo, pero que se identifica más con un insecto. La razón es que el modelo de partida era parecido a una hormiga y realmente la anatomía de ésta no tiene nada que ver con la de un perro. Por esta razón habrá que hacer un estudio más detallado del cuadrúpedo que se va a construir, pues realmente no se pueden utilizar las estructuras hasta ahora vistas, salvo la de AIBO, que sí tiene un aspecto de perro.

Después de esta búsqueda se sacan las siguientes conclusiones:

1. Se utilizará una red de microcontroladores, esta idea ya se tenía y se reafirma al haber sido utilizada en robots como AIBO.

2. Los motores serán del tipo servomecanismo, el modelo en concreto dependerá de la estructura final, sobre todo por los temas relacionados con la potencia necesaria para mover la estructura.

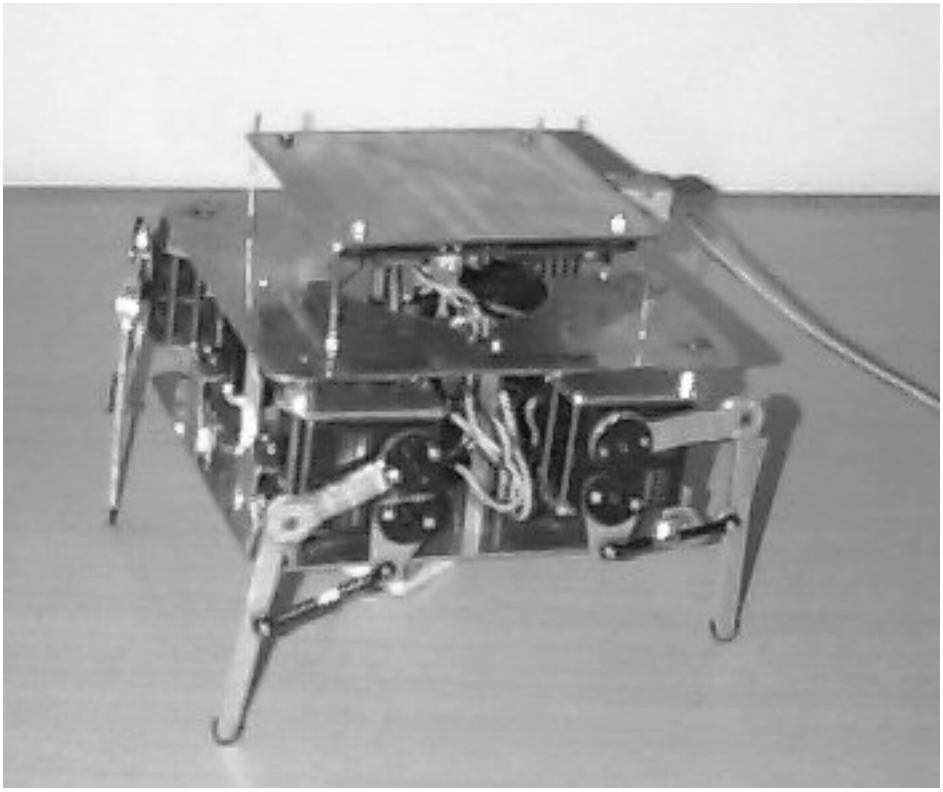


Figura 1.11: Robot *BERTA* desarrollado por Ivo Nijhuis & Ronnie Jansen.

3. Por último, el material será aluminio, aunque podría haber sido perfectamente plástico (PVC). La elección de éste se debe a que el autor ya tiene experiencia con él y además dispone de las herramientas para tratarlo.

4. Sobre la forma de la estructura se reafirma lo dicho en el párrafo anterior: habrá que diseñar una nueva que tenga aspecto de perro.

Capítulo 2

Estructura mecánica

2.1 Morfología de los insectos y mamíferos

Antes de nada se van a describir las principales características de los insectos y mamíferos desde un punto de vista morfológico, es decir, desde su estructura anatómica. Con ello se pretende descubrir cuál de ellas va a permitir que la estructura realizada se parezca más a un perro.

Los insectos, animales del tipo artrópodo, presentan las siguientes características comunes. El cuerpo está dividido siempre en tres partes: cabeza, tórax y abdomen. En la cabeza se encuentran los ojos, antenas y piezas bucales. El tórax está formado por tres segmentos y en cada uno de ellos hay un par de patas, que no deben confundirse con las alas. El abdomen no tiene apéndices, aunque en algunos insectos acaba en un aguijón. De todo lo dicho hasta ahora no se encuentra ninguna característica importante, salvo por el dato de que las patas surgen todas del tórax y que son seis. Cada una de ellas tiene cinco segmentos principales, de los cuales el fémur y la tibia forman la parte principal.

Los mamíferos, clase de los vertebrados, tienen cuatro miembros situados paralelamente al plano sagital: dos torácicos y dos abdominales o pélvicos, provistos de dedos, generalmente cinco. Se dice que las extremidades de los mamíferos y de los insectos son homólogas, lo que quiere decir que tienen distinto origen pero realizan la misma función. En los mamíferos da la impresión que las extremidades salen del cuerpo de forma perpendicular al suelo, mientras que en los insectos la situación es diferente, más bien parece que las extremidades salen de forma paralela. Otra diferencia es que en los primeros hay dos en el tórax y otras dos en el abdomen, mientras que en los segundos todas salen del tórax, pero esto no se va a utilizar como característica diferenciadora, ya que se pretende construir todo el cuerpo de una pieza. Ver figuras 2.1 y 2.2.

Ya se tiene la principal característica para dar al robot un aspecto de perro, hay que situar las extremidades perpendiculares al cuerpo. Ahora hay que ver cómo se puede dotar de movimiento a la estructura y cuántos motores se deben usar para que realice los movimientos básicos, andar y girar.

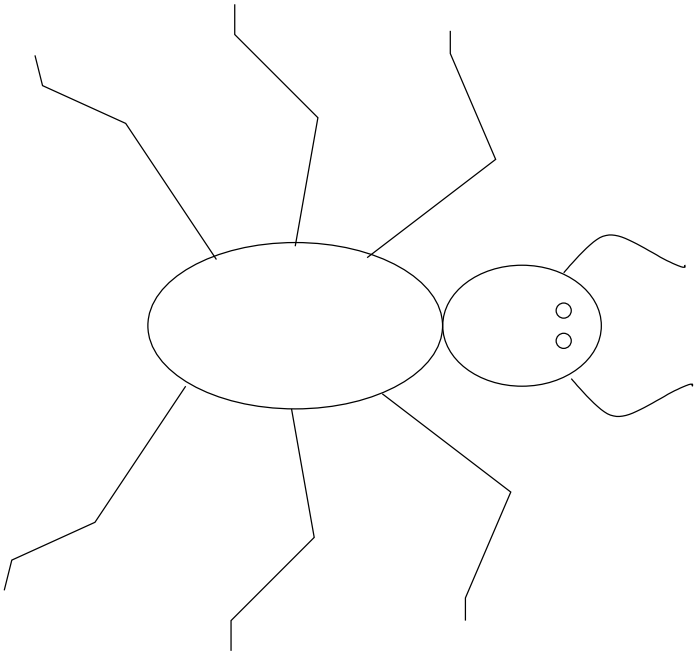


Figura 2.1: Esquema descriptivo de un insecto.

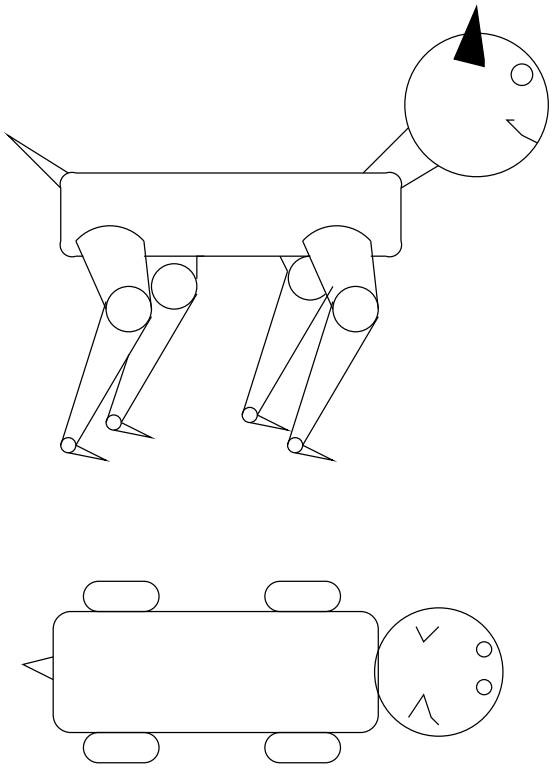


Figura 2.2: Morfología de un perro.

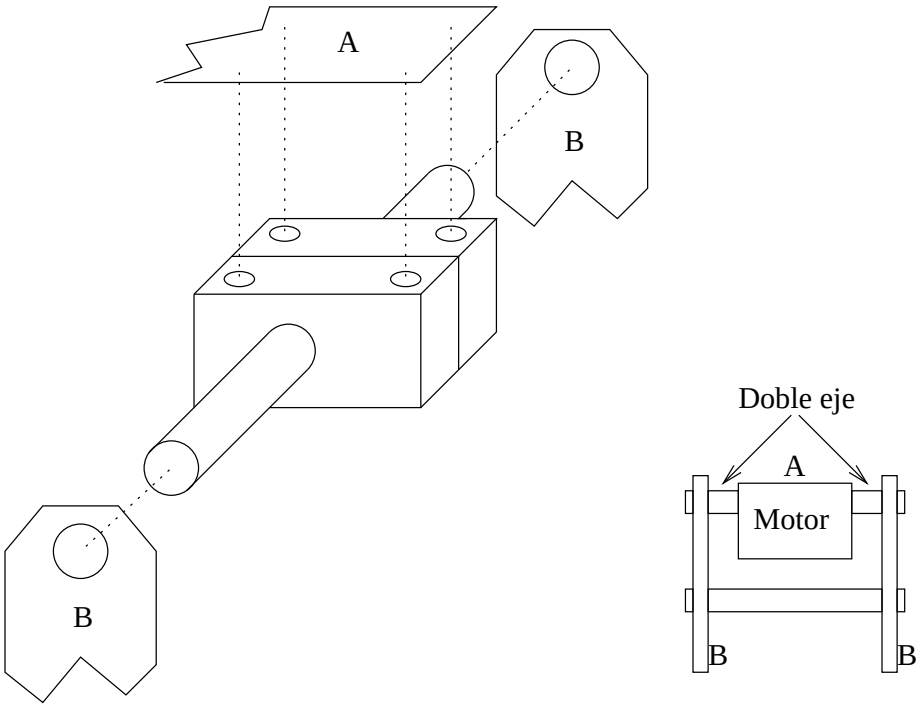


Figura 2.3: Estructura doble eje.

2.2 Esbozo del primer prototipo

Para diseñar el primer prototipo se han tenido en cuenta todas las ideas anteriores y alguna otra añadida por la propia experiencia del autor. A continuación se enumeran todas, sin ningún orden de importancia:

1. La estructura será de aluminio con un grosor entre 0'5mm y 1.5mm, de manera que será bastante ligero. El tratamiento mecánico de las piezas sólo permite cortarlas en polígonos, plegarlas y taladrarlas.

2. Los motores serán servomecanismos de aeromodelismo del tipo Futaba 3003, el modelo más básico de la familia y el más económico. En el capítulo siguiente se describe con detalle, pero ahora sólo interesa las dimensiones y el par de salida (3Kg x cm).

3. Los ejes de giro serán dobles, de forma que el robot adquiera una mayor rigidez y estabilidad. Esta característica la ha aprendido el autor después de construir el hexápodo (figura 1.5). En aquel robot debido a que los ejes eran simples había una pequeña holgura lateral que producía un tambaleo incómodo para el control final del movimiento. Por lo tanto ahora habrá que hacer las uniones de las diferentes partes móviles de la forma indicada en la figura 2.3. La estructura A se unirá al motor y por medio de dos ejes se sujetará la estructura B.

4. Las extremidades, que serán cuatro, estarán situadas perpendiculares al cuerpo. De esa forma la apariencia será la de un perro. Ver figura 2.2.

La primera estructura que se pensó estaba formada por ocho motores, dos por cada pata. Todas las patas eran iguales y estaban compuestas por cuatro piezas, iguales dos a dos.

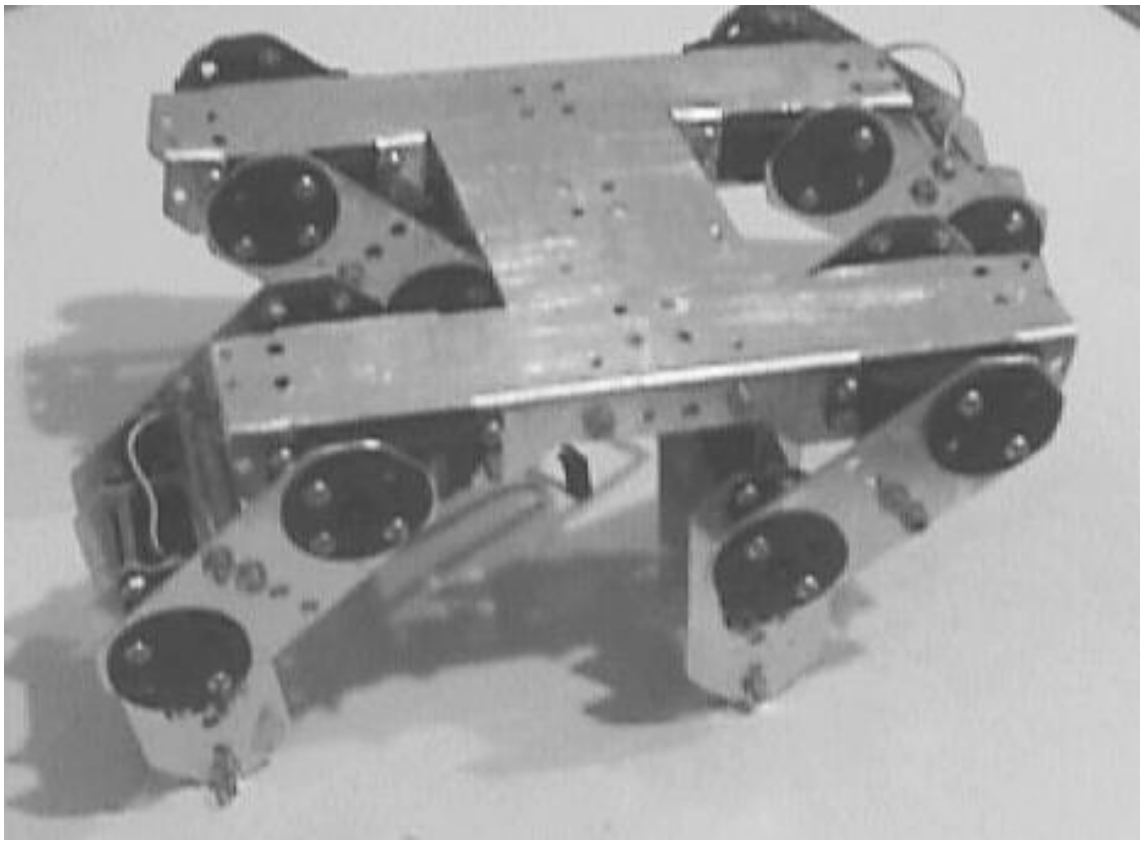


Figura 2.4: Vista del primer prototipo realizado.

Para construir el robot se hicieron unas plantillas a tamaño real, de forma que colocándolas sobre la chapa de aluminio y recortándolas no hacía falta tomar medidas. Este método ahorra tiempo de elaboración, ya que al haber bastantes piezas iguales, con diseñar una de ellas las otras se obtienen imprimiendo más.

En la figura 2.4 se muestra una foto del prototipo. En ella se aprecia que cada pata tiene dos articulaciones, formada cada una de ellas por un servo. Con estos motores se puede controlar la posición exacta de cada articulación en un recorrido teórico de unos 190^o, aunque el real es menor por limitaciones mecánicas de la estructura. Comparando el robot de esta figura con el de la 1.11 se observa que a pesar de ser ambos cuadrúpedos, éste se parece más a un mamífero y el otro a un insecto. La diferencia, una vez más, se debe a la posición totalmente distinta de los motores. En el diseño se intentó buscar una distribución y un tamaño más o menos proporcional al de un perro real, pero sin olvidar que tenía que ser operativo. Por eso en esta primera aproximación no se pusieron dedos, muñecas, tobillos, cabeza y cola. Se consideró que estos no son vitales para conseguir el movimiento básico. Tampoco se pusieron motores para poder extender las patas lateralmente, sobre todo por el incremento de peso, que haría inviable mover el robot con los servos básicos (Futaba 3003) ya que estos no tienen la fuerza necesaria para hacerlo. Además, se pensó que con los ocho motores sería suficiente para andar adelante, atrás y girar a izquierda y derecha.

En la figura 2.5 se muestra la estructura de una de las patas, apreciándose como el primer motor (M1) está unido al cuerpo del perro y mediante su eje mueve la primera

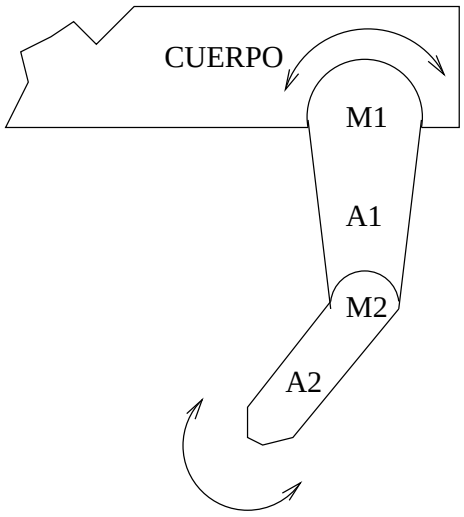


Figura 2.5: Diagrama de una de las patas del robot.

articulación (A1), que a su vez está unida al eje del segundo motor (M2). Siendo éste finalmente el que forma la articulación final (A2). En la figura 2.6 se ve una foto de la pata, en donde los rectángulos negros son los motores con sus dobles ejes. El truco para haber realizado este tipo de estructura ha sido conseguir las piezas sueltas de los motores, de forma que se ha adquirido la caja, junto con el engranaje que forma el eje.

Cuando se probó la estructura se tuvieron los resultados esperados salvo uno, no era posible girar, un fallo debido a la falsa estimación de creer que se podría hacer girar al robot mediante rozamientos y esfuerzos diagonales, cuando realmente no se podían producir por no existir la posibilidad de realizar movimientos laterales. Por lo demás, el robot avanzaba y retrocedía con un buen equilibrio, era capaz de mantenerse en pie sólo con dos patas (situadas en diagonal). También se observó que la fuerza estaba muy justa y que con el siguiente modelo de motor (servomecanismo Futaba S9404) se hubieran obtenido mayores movimientos. Por ejemplo la posibilidad de levantarlo a pesar de estar tumbado y también se hubiera mejorado la velocidad. Lo malo es que este motor costaba cuatro veces más que un servomecanismo normal y multiplicando eso por ocho el coste se hubiera disparado.

Con esta primera estructura el robot pudo andar de forma autónoma, es decir sin necesidad de hacer ninguna conexión exterior, pero su autonomía dejaba mucho que desear, tan solo se consiguió un cuarto de hora con la batería cargada. Duplicar este valor no era posible ya que se tendría que haber puesto una batería con el doble de peso y ya no podría andar por falta de fuerza. Tal vez con los motores potentes se hubiera conseguido, ya que al triplicar la fuerza de cada motor se hubiera contrarrestado el incremento al doble del peso de la batería. Pero todo esto es una suposición ya que no se llegó a probar. La razón principal era que dado que había que fabricar otra estructura ya se harían las pruebas en la nueva.

En las figuras 2.7, 2.8, 2.9 y, 2.12, están las plantillas que se utilizaron para construir la primera estructura. Están reducidas al 50%, aunque las cotas en ellas indicadas son reales. Una breve descripción de los pasos realizados para construir la estructura es la siguiente.

1. **Construcción del cuerpo**: Se recortó una chapa de aluminio de 1mm de grosor sigui-

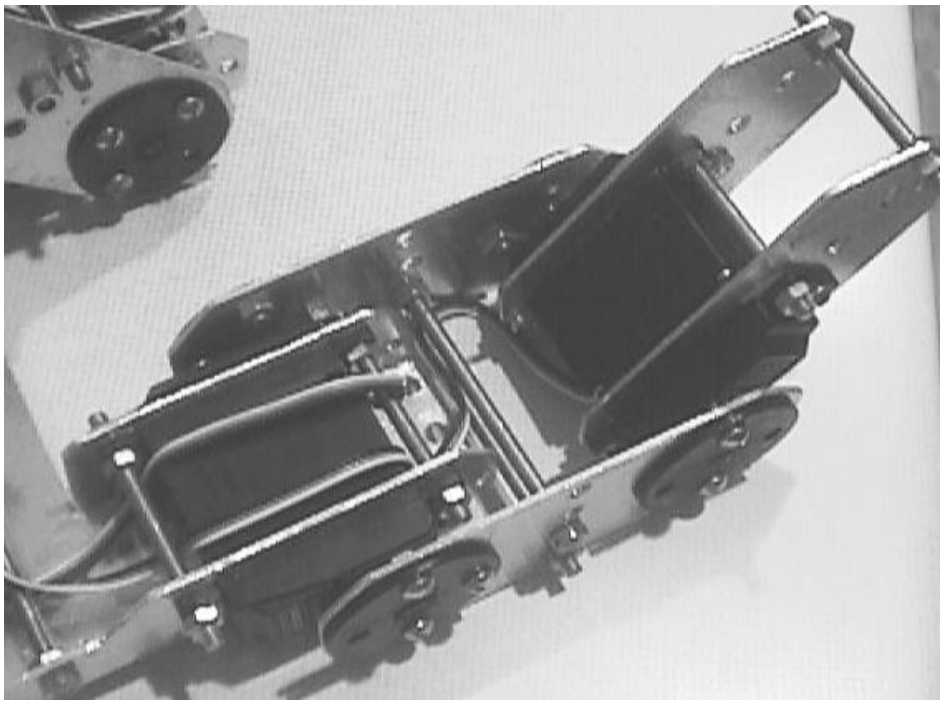


Figura 2.6: Foto de una pata real del perro robot.

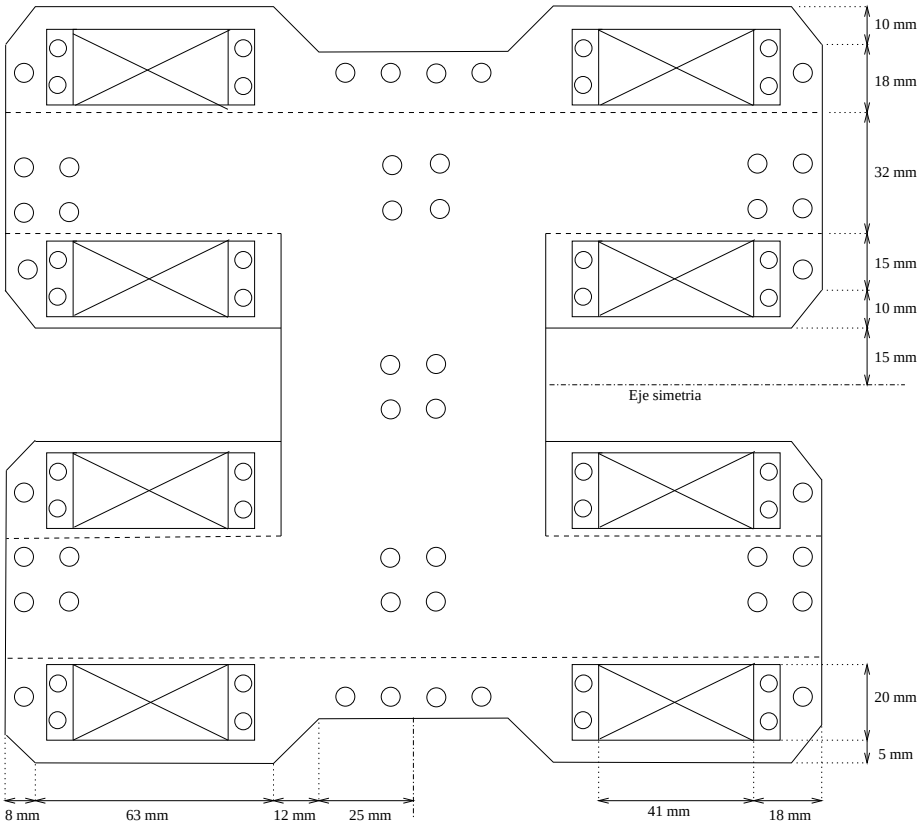


Figura 2.7: Cuerpo del primer prototipo.

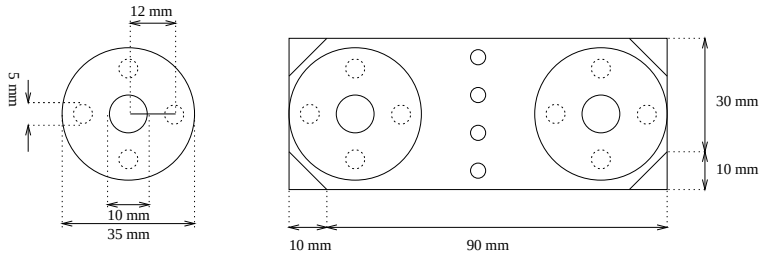


Figura 2.8: Plantilla de la pieza para formar la primera articulación de cada pata.

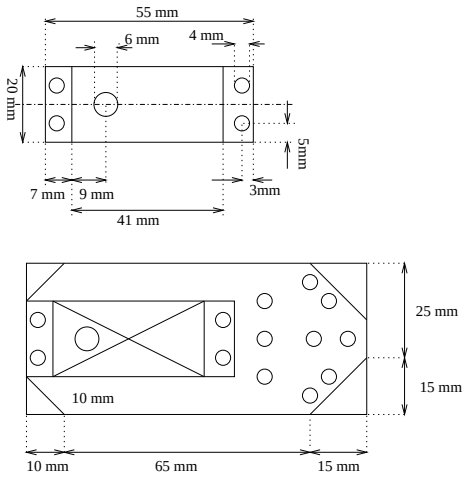


Figura 2.9: Plantilla de la pieza para formar la segunda articulación de cada pata.

endo el contorno de la plantilla del cuerpo (figura 2.7). Luego se hicieron los taladros indicados por los círculos y se recortaron los rectángulos interiores donde se sitúan los motores. Estos rectángulos están marcados por una cruz. Por último, se plegó la estructura para obtener un cuerpo como el de la figura 2.4.

2. Se colocaron los primeros cuatro motores que forman las articulaciones superiores de las patas. Junto con los motores se colocaron también las piezas para formar el doble eje[1]. Para unir los motores a la estructura se utilizaron los cuatro taladros, en dos de ellos (B) se colocaron dos tornillos con sus respectivas tuercas y en los otros dos (A) sendas varillas roscadas que fijaban el motor a la pieza que forma el doble eje, a partir de ahora llamada pieza de simetría. Ver figuras 2.6 y 2.10.

3. **Construcción de las patas**: Se recortaron el resto de las piezas que forman las patas utilizando las plantillas representadas en las figuras 2.8 y 2.9. Una vez hecho esto primero se montó la segunda articulación de cada pata, es decir, la que forma el pie e incorpora el segundo motor. Se colocó el motor junto a su pieza de simetría y se pusieron dos tuercas (B) y dos varillas roscadas (A) repitiendo la operación del paso segundo. Para dar mayor solidez al final de la articulación se colocó otra varilla roscada en el extremo de las chapas. Ver figura 2.11.

[1]Ests piezas son la caja de engranajes, el eje salida y la tapa inferior del servomecanismo. Se pueden comprar por separado en las tiendas de aeromodelismo y aunque el eje doble se pudo haber realizado de otra forma, fue una manera de asegurar la perfecta simetría.

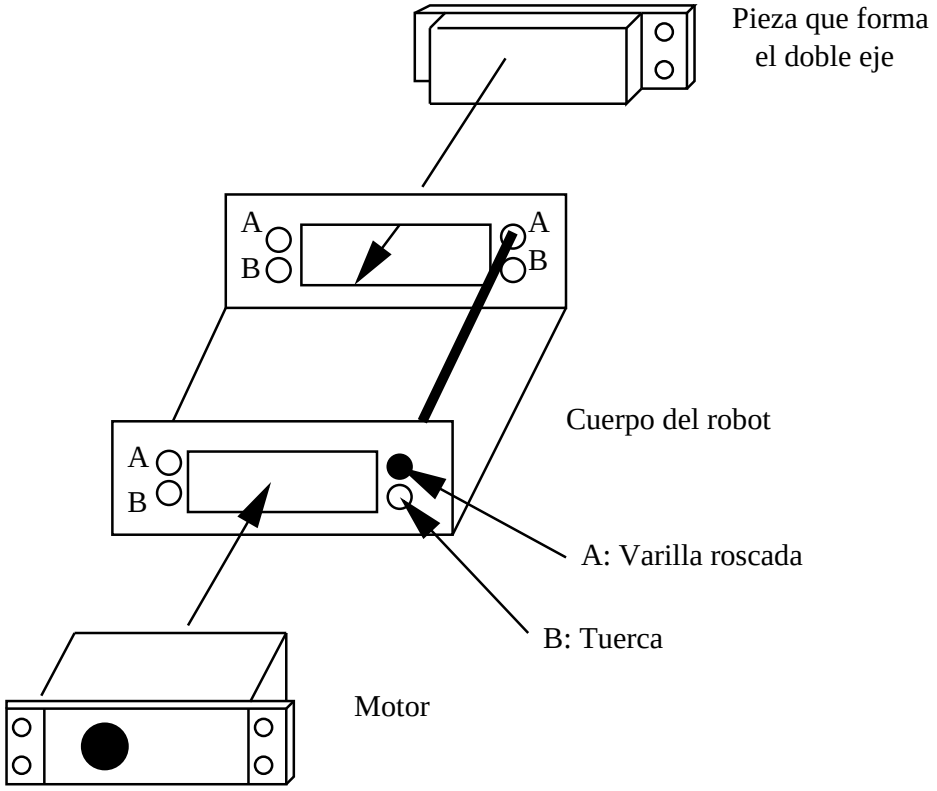


Figura 2.10: Esquema para ensamblar los motores en el cuerpo del robot.

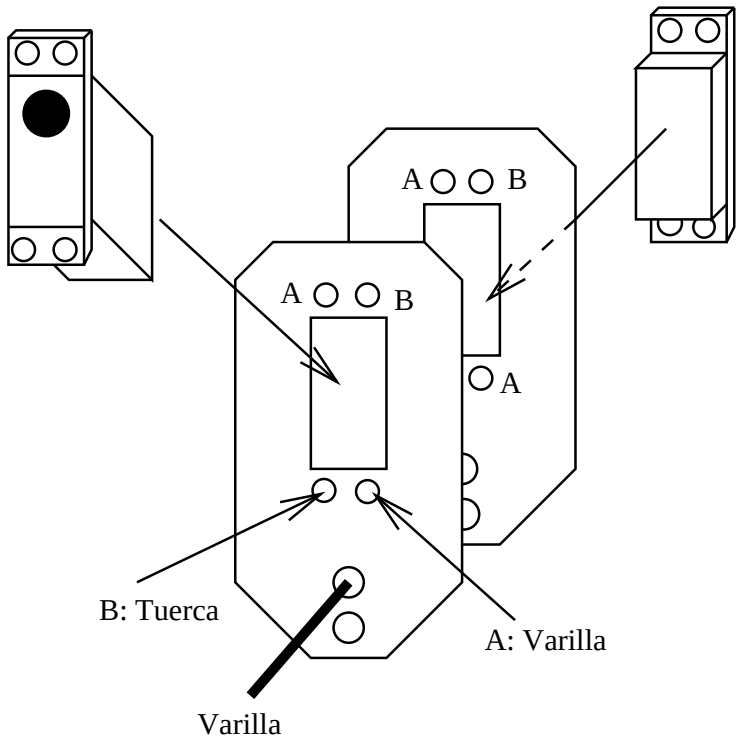


Figura 2.11: Esquema para montar la segunda articulación de cada pata.

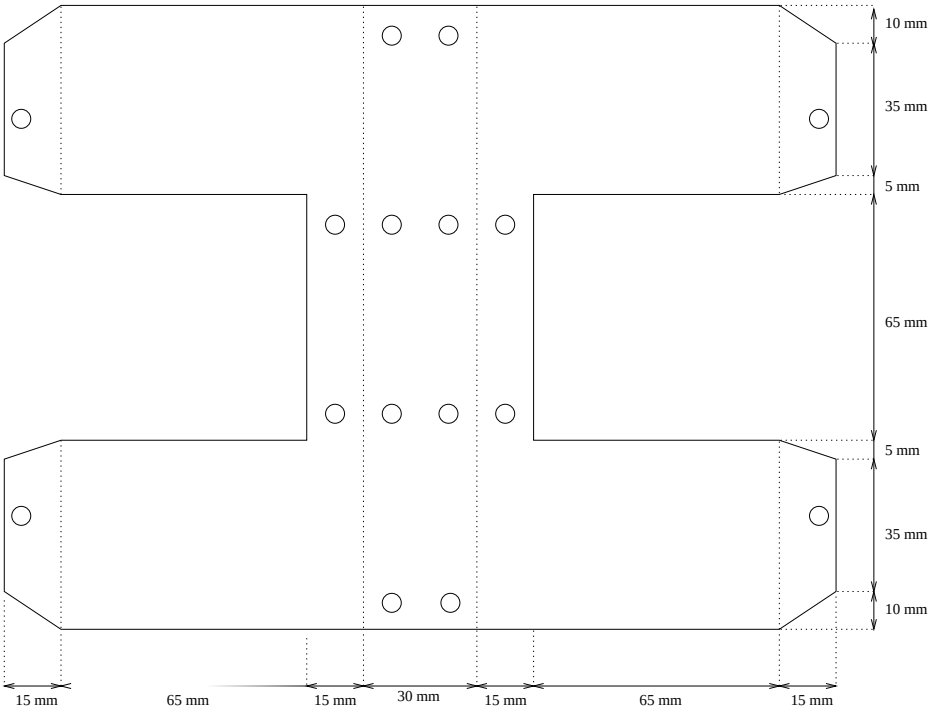


Figura 2.12: Plantilla para formar la cubierta del cuerpo.

4. **Unión de la pata con el cuerpo:** Finalmente se montó la primera articulación, la cual une el cuerpo del robot con la pieza anterior. Para ello se unieron entre sí los ejes de ambos motores y para proporcionar una mayor resistencia se colocaron dos varillas roscadas en el centro de la articulación. Todo esto se muestra, una vez más, en la figura 2.6.

5. **Cubierta del robot**: Para sujetar la electrónica y dejar sitio a las baterías se fabricó la última pieza estructural, que es la representada en la figura 2.12. Además proporciona volumen y resistencia al cuerpo del primer prototipo, cuyo aspecto final se muestra en la figura 2.13.

2.3 Prototipo final: Planos y montaje

Como ya se ha comentado en la sección anterior la primera estructura del robot no dio los resultados esperados, por ejemplo el robot no era capaz de moverse en todas las direcciones. Los principales inconvenientes encontrados fueron:

1. Movimientos limitados por la imposibilidad de girar debido a la falta de una articulación lateral que extendiera las extremidades.

2. La fuerza que proporcionaban los motores estaba muy justa, impidiendo realizar todos los movimientos que se habían previsto, como por ejemplo levantarse del suelo.

3. La autonomía era escasa por el pequeño tamaño de las baterías.

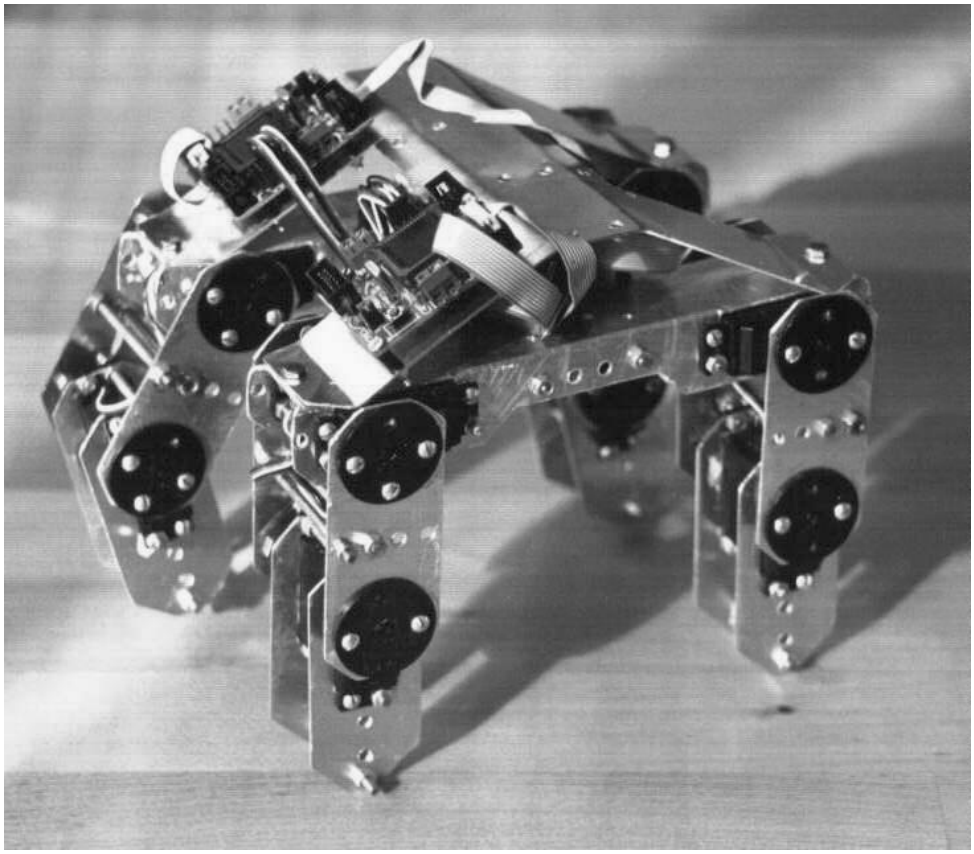


Figura 2.13: Foto del primer prototipo una vez construido.

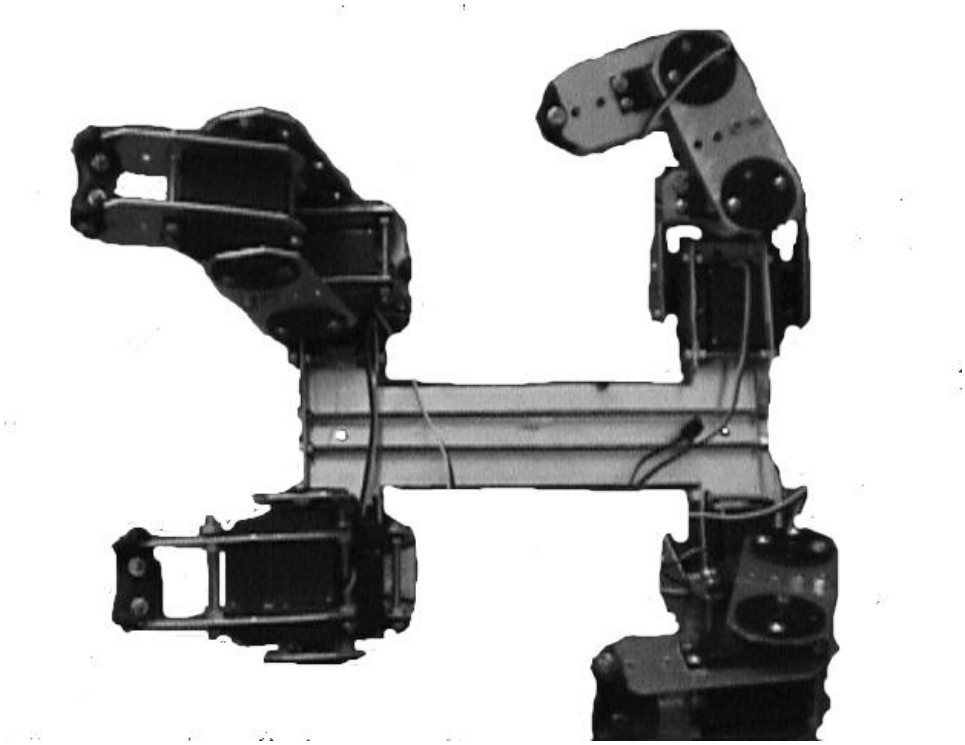


Figura 2.14: Foto del robot con las patas extendidas lateralmente. Esto es posible realizarlo por los cuatro servomecanismos que se han añadido.

Por todos estos motivos se desarrolló una nueva estructura, más evolucionada, que intentó solucionar los tres puntos anteriores. Y aunque ahora se describe a continuación, en realidad el proceso fue distinto. Una vez que se tuvo la primera estructura se realizó el software de control de los motores y el de programación de movimientos. Fue al integrar todo y realizar las pruebas cuando los errores aparecieron, siendo necesario volver al principio para modificar la estructura y adaptar el software. Pero esta vez el trabajo fue menor, ya que muchas de las piezas y del software se reutilizaron. Eso es lo bueno de haber planteado el proyecto como un conjunto de módulos.

La nueva estructura tiene las siguientes mejoras:

1. Incorpora cuatro motores más que permiten realizar movimientos laterales, ver figura 2.14. Gracias a estos se consiguen dos de los movimientos más buscados: girar en ambas direcciones y mejorar la estabilidad al andar. La consecuencia desfavorable es que la estructura ha aumentado y en consecuencia el peso de la misma. Ahora los servomecanismos normales no tienen la suficiente fuerza para desplazar el robot.

2. Por el motivo anterior se han cambiado cuatro de los servomecanismos normales (Futaba 3003) por unos de más fuerza (Futaba 9404). Lo ideal hubiera sido sustituir al menos ocho, pero por motivos de coste[2] no ha sido posible. Lo bueno es que al ser las dimensiones de los dos modelos prácticamente iguales, se pueden ir cambiando poco a poco. Con esta mejora el robot consigue tener la fuerza necesaria para desplazarse,

[2]Un servomecanismo Futaba S3003 cuesta unas 3000 pts, mientras que el modelo Futaba S9404 cuesta unas 13.000 pts.

al igual que para realizar movimientos del tipo levantarse, saludar, etc. Estos últimos movimientos no son fundamentales para andar pero sí lo son para dotar de simpatía al robot. Pensar por ejemplo en un perro que no levante la pata para saludar, o que no se siente cerca de su amo, en fin, que ahora sí se pueden programar este tipo de acciones. Pero como toda mejora, también añade una complicación extra. Ahora la potencia que consume el robot ha aumentado, hay más motores y encima consumen más, por lo que si antes la batería duraba poco ahora dura mucho menos.

3. Para solucionar el inconveniente de la batería lo que habría que haber hecho es lo siguiente: Sustituir todos los motores por los de alto par, y poner unas baterías de última generación, que son capaces de absorber picos elevados de corriente y tienen un peso reducido. En concreto son las baterías de NiCad utilizadas en los coches de radiocontrol. Pero lo anterior no se ha realizado, una vez más apareció el problema económico[3], y dado que los motores no se habían cambiado era inútil comprar esas baterías pues el robot no podría con ellas. La solución buscada ha sido utilizar una fuente externa para alimentar a los motores. De esta forma se puede mover, y siempre se tendrá la posibilidad de llegar a tener el robot autónomo comprando los motores que faltan y la batería más potente.

En resumen, con la nueva estructura, con los nuevos motores y con la fuente de alimentación externa se han conseguido unos buenos resultados. El robot puede andar en todas direcciones, saludar y levantarse. Además en caso de cambiar todos los motores por el modelo superior y de incorporar una batería, el robot podría ser todavía más autónomo.

Para acoplar los nuevos motores ha sido necesario modificar la estructura mecánica. En concreto se ha variado el cuerpo del robot para alojar los nuevos motores y se han introducido unas articulaciones nuevas, que de ahora en adelante se llamarán *hombros*. Las patas del robot se mantienen iguales a las del prototipo anterior por lo que no ha sido necesario modificarlas. La función de los *hombros* consiste en unir cada extremidad al cuerpo del robot, pero ofreciendo un grado más de movimiento. En la figura 2.15 se representan los grados de libertad de las extremidades. En la figura 2.16 se puede ver un esquema de la nueva morfología del robot.

El proceso de construcción de esta nueva estructura es muy parecido al anterior, apartado 2.2., por lo que se recomienda acudir a él. Las figuras 2.5, 2.10, 2.11, 2.3 son válidas ya que la forma de ensamblar los motores de las articulaciones primera y segunda no ha variado. Se puede decir que lo único que ha cambiado del robot es la orientación de los motores del cuerpo y el nuevo elemento añadido: el *hombro*. Que cambie la orientación de los motores significa que se ha tenido que hacer una nueva plantilla, para el cuerpo, que será la que se utilice a la hora de construirlo. El *hombro* es un elemento nuevo por lo que ahora se procederá a poner el plano y el esquema de construcción. Realmente es muy sencillo ya que sigue el método anterior: incorporar un motor con doble eje y unos enganches para unirse a los motores del cuerpo. (Ver figura 2.17). Los motores se unirán con dos tornillos y dos varillas, en la figura anterior se indica gráficamente por dónde se introducirán las varillas, los otros agujeros corresponden a los tornillos.

Al final los pasos para construir el robot son los siguientes:

[3]Una batería de radiocontrol de NiCad que ofrezca 3 A/h puede costar unas 7500pts y su cargador unas 15.000 pts.

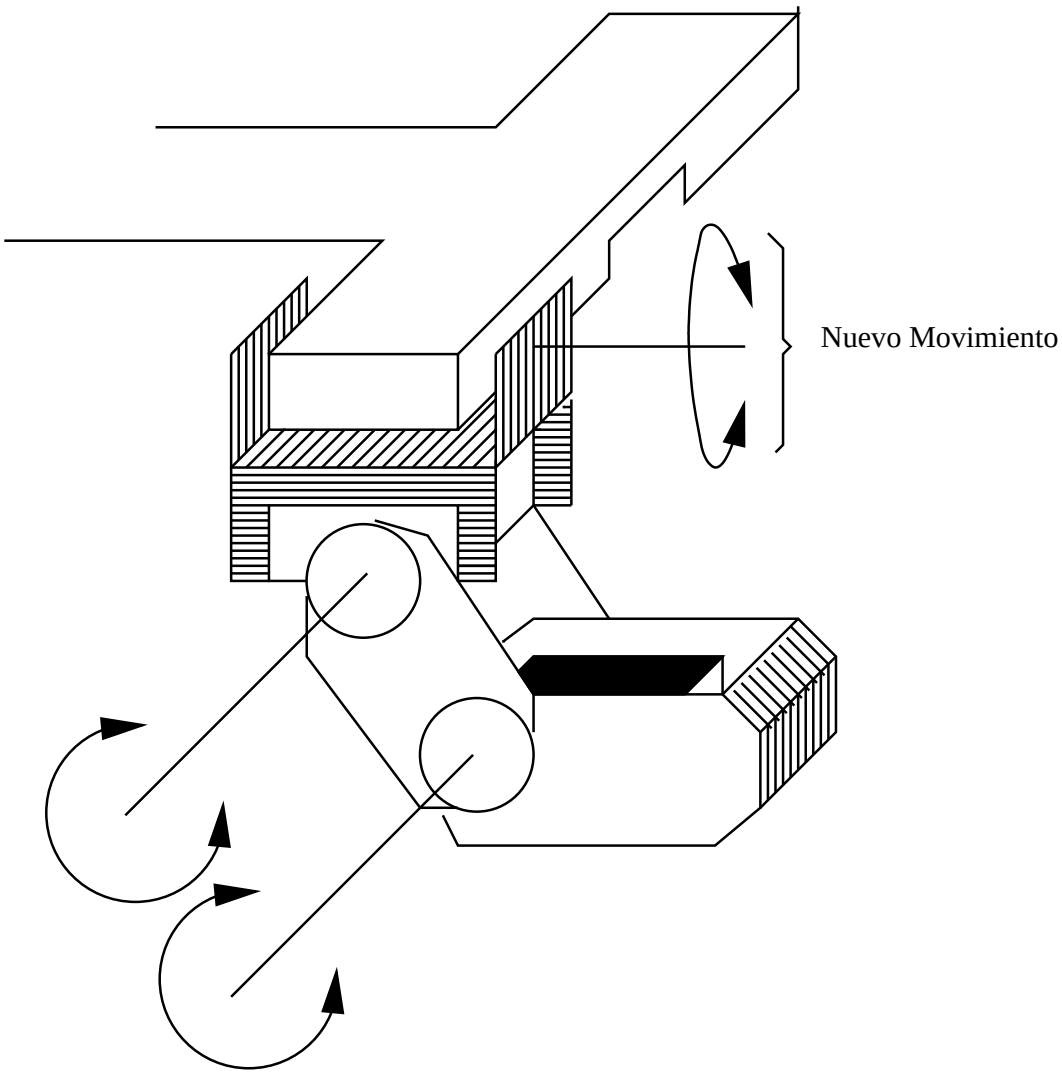


Figura 2.15: Grados de libertad de las patas del robot.

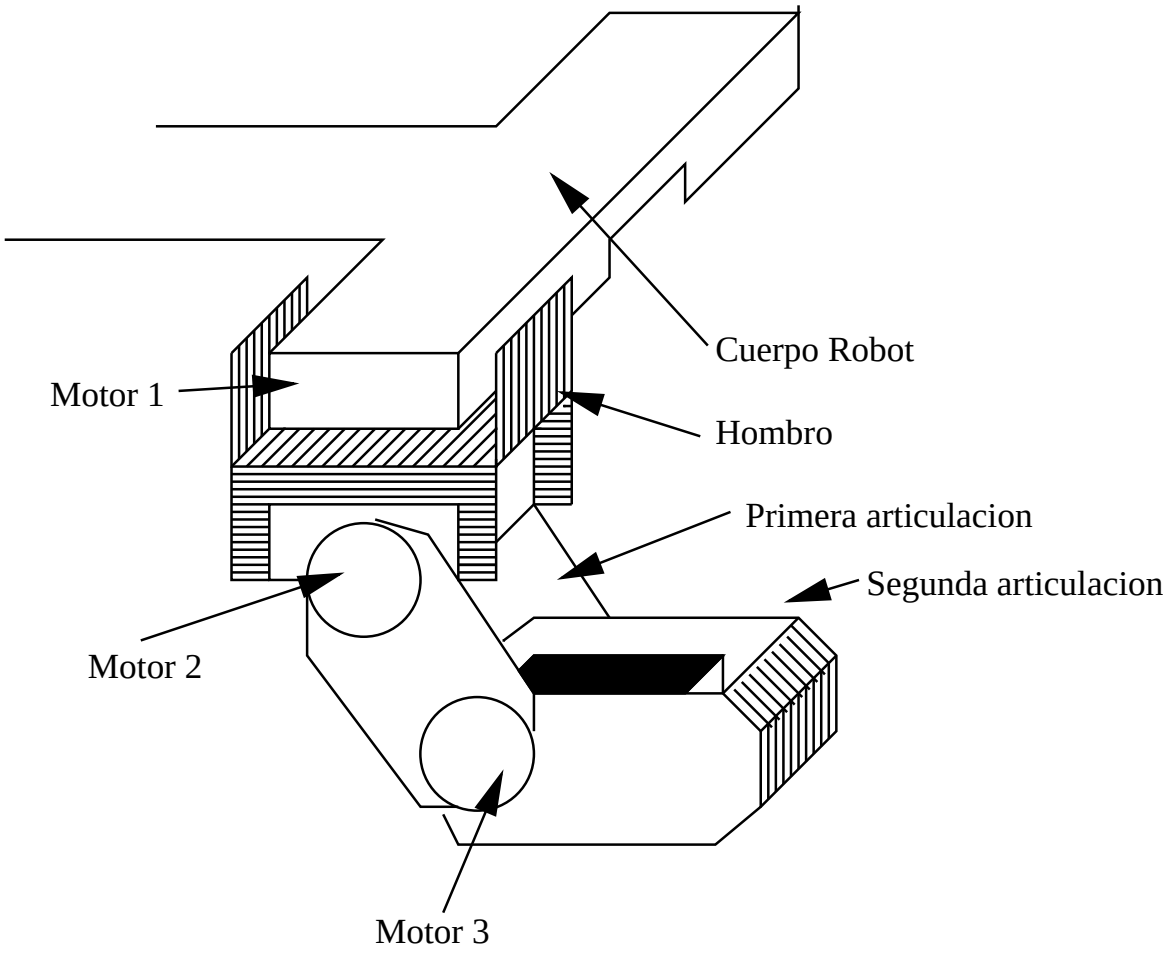


Figura 2.16: Morfología final de las patas del robot.

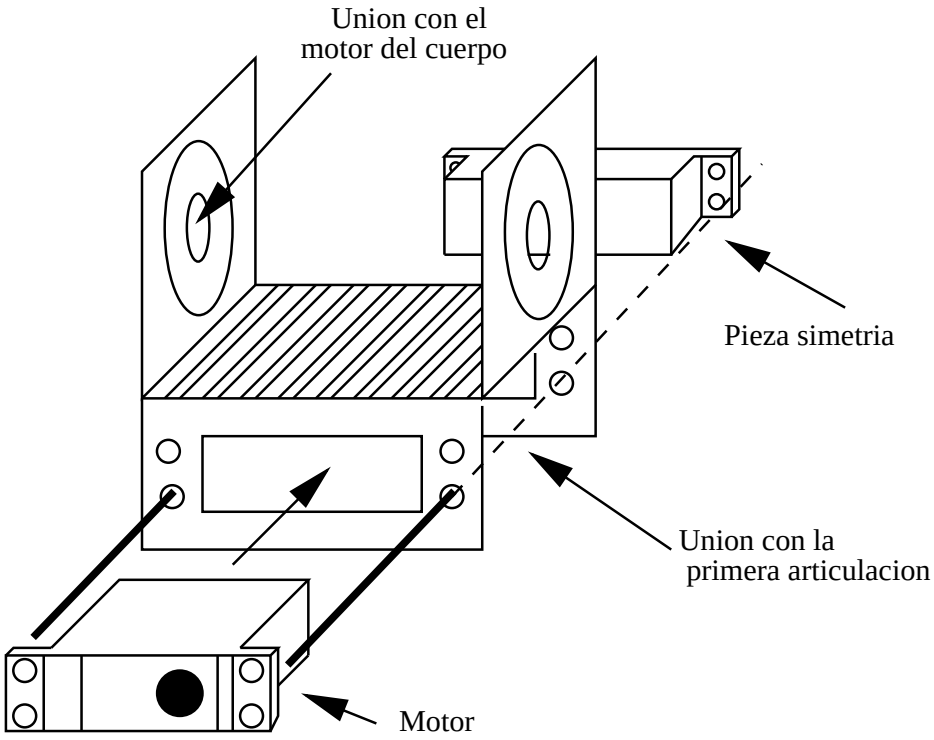


Figura 2.17: Ensamblado del hombro del robot.

1. Hay que recortar todas las plantillas. Se utilizará aluminio de 1 mm de grosor para las articulaciones y el cuerpo, y una lámina de 0.5mm de grosor para la cubierta del cuerpo.

2. Plegar la estructura que forma el cuerpo y colocar los cuatro servomotores Futaba S3003 junto con los dobles ejes. Ver figuras 2.19 y 2.18.

3. Armar cada pata del robot. Para ello se pueden seguir todos los puntos *'Construcción de las patas'* , *'Unión de las patas con el cuerpo'*, del apartado 2.2., y toda la explicación sobre cómo armar el *hombro* expuesta en este apartado. Para que la explicación sea válida cada vez que se mencione el motor del cuerpo se tendrá que considerar el motor del *hombro*. Ver figura 2.20.

Todos los planos de la nueva estructura se encuentran al final de la obra, aunque al igual que antes, en este capítulo se han incluido reducidos al 50%. En concreto, los planos del nuevo robot están formados por las figuras siguientes:

1. Cuerpo del robot: figura 2.22.

2. *Hombro* del robot: figura 2.21.

3. Primera articulación: figuras 2.8 y 2.23.

4. Segunda articulación: figuras2.9 y 2.23.

5. Cubierta del robot: figura 2.24 .

Figura 2.18: Foto del cuerpo del robot en el momento de la construcción.

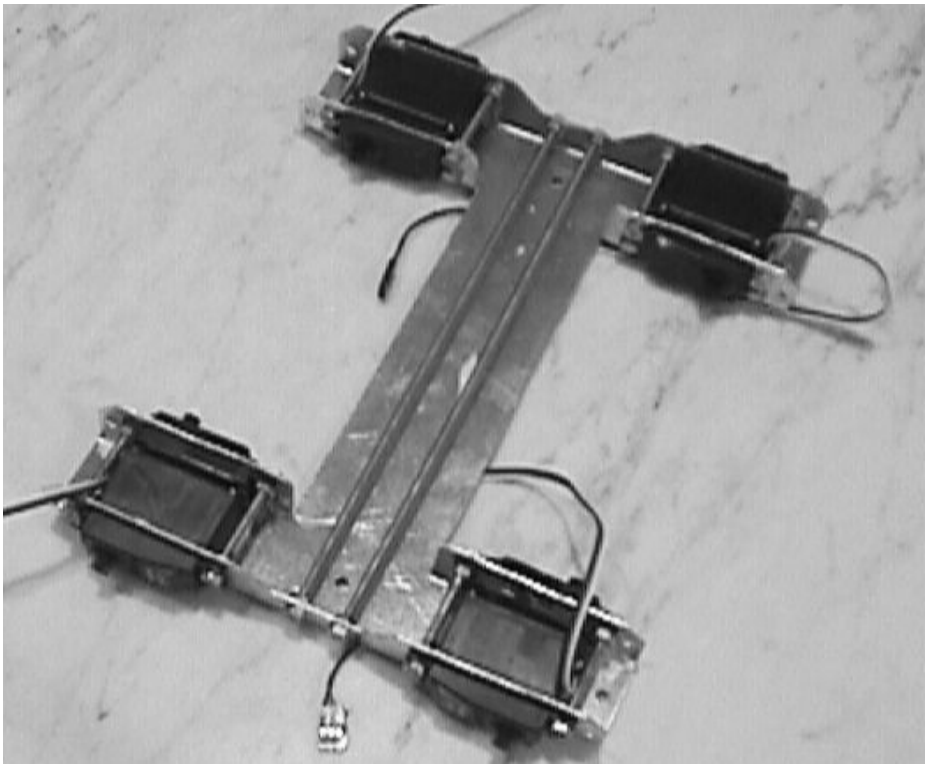


Figura 2.19: Foto del cuerpo del robot en el momento de la construcción.

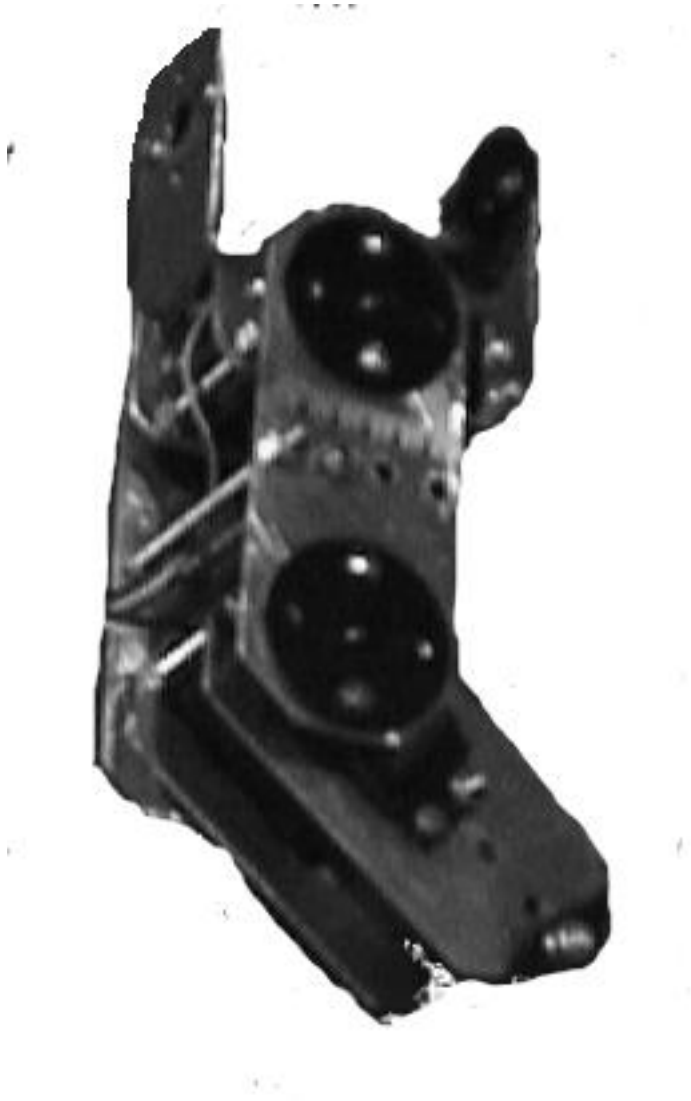


Figura 2.20: Foto de la pata del robot con las articulaciones y el hombro.

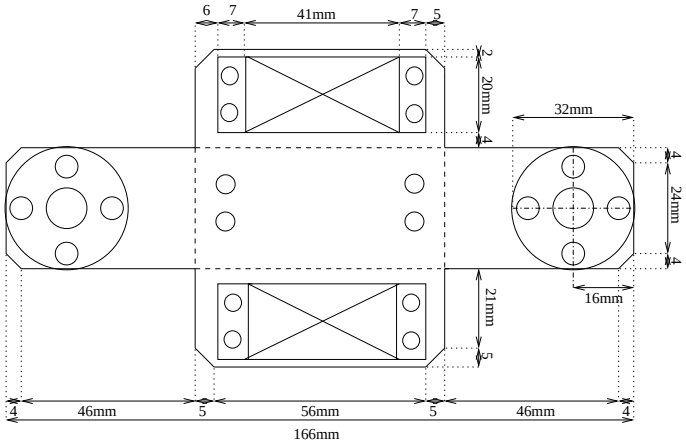


Figura 2.21: Plano de la nueva estructura (50%): Hombro.

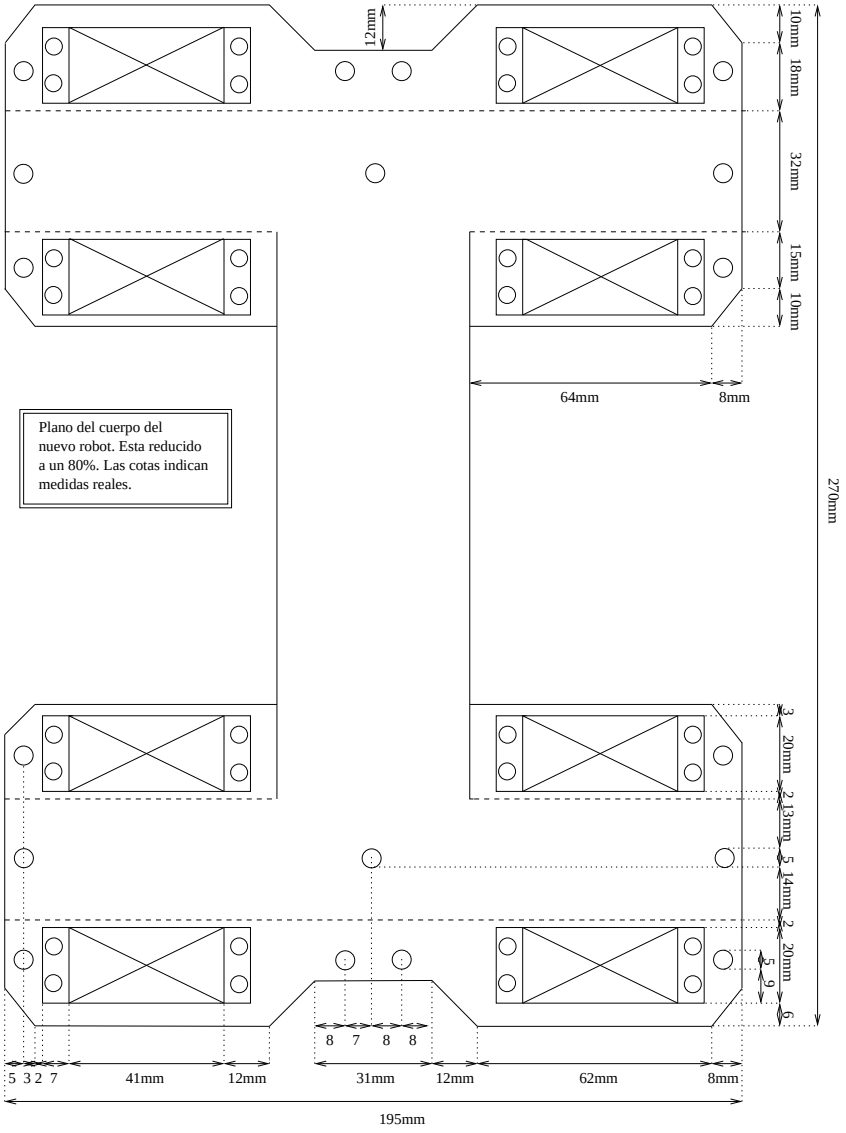


Figura 2.22: Plano de la nueva estructura (50%): Cuerpo

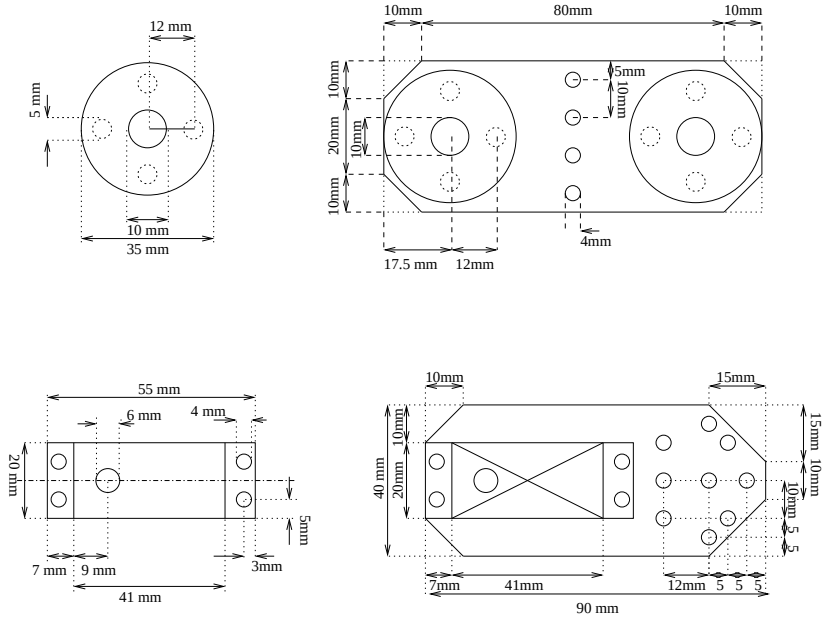


Figura 2.23: Plano de la nueva estructura (50%): Articulaciones.

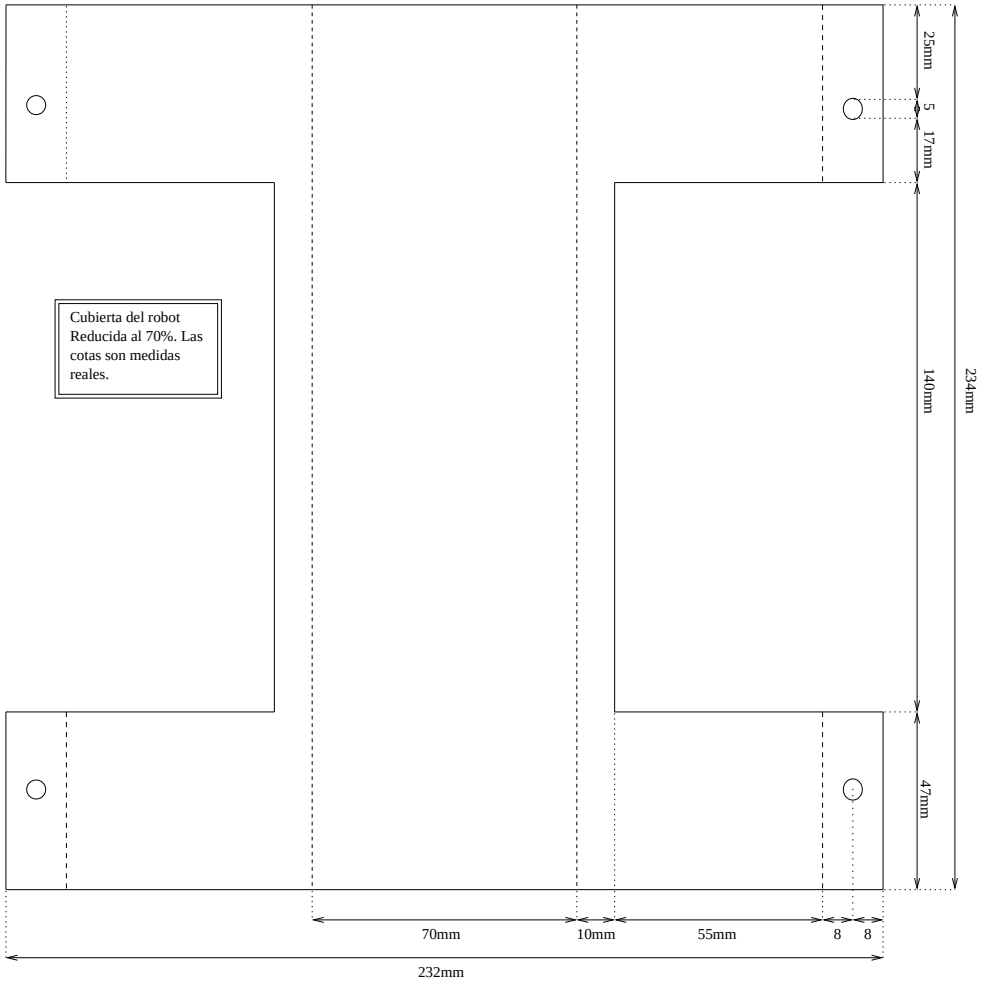


Figura 2.24: Plano de la nueva estructura (50%): Cubierta.

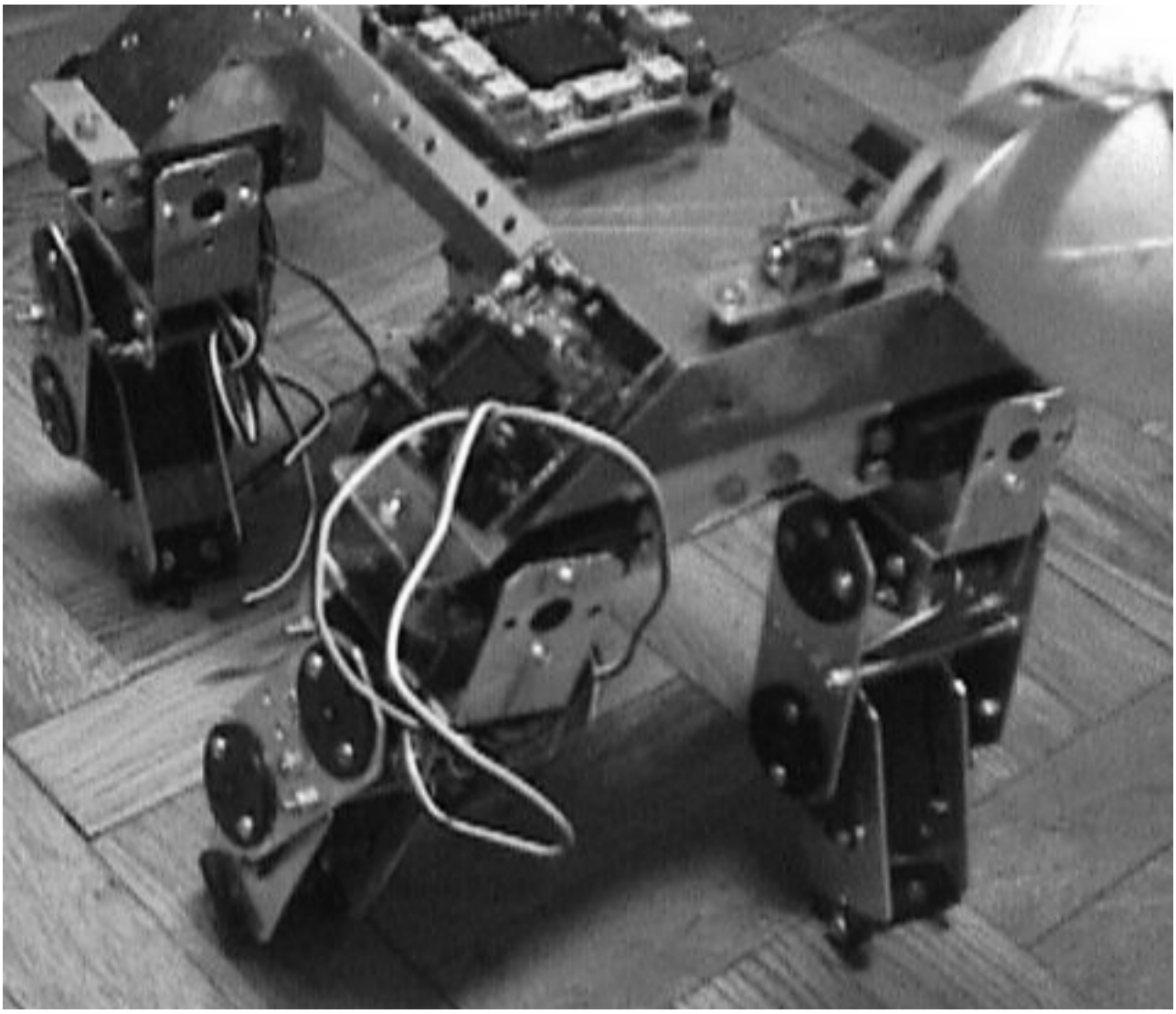


Figura 2.25: Robot andando.

Por último, para darle un aspecto de mamífero es necesario ponerle una cabeza y un rabo. Lo mejor sería poner unos pequeños motorcitos para poder moverlos, de esa forma se conseguirían acciones bastantes graciosas del perro, pero es preferible asegurar el correcto funcionamiento de toda la estructura antes de dotar de movimiento a tales elementos. Por eso se ha preferido esperar a tener todo controlado y dejar esto último para el final. Por lo tanto la estructura en este momento ha quedado como indican las figuras 2.25 y 2.26.

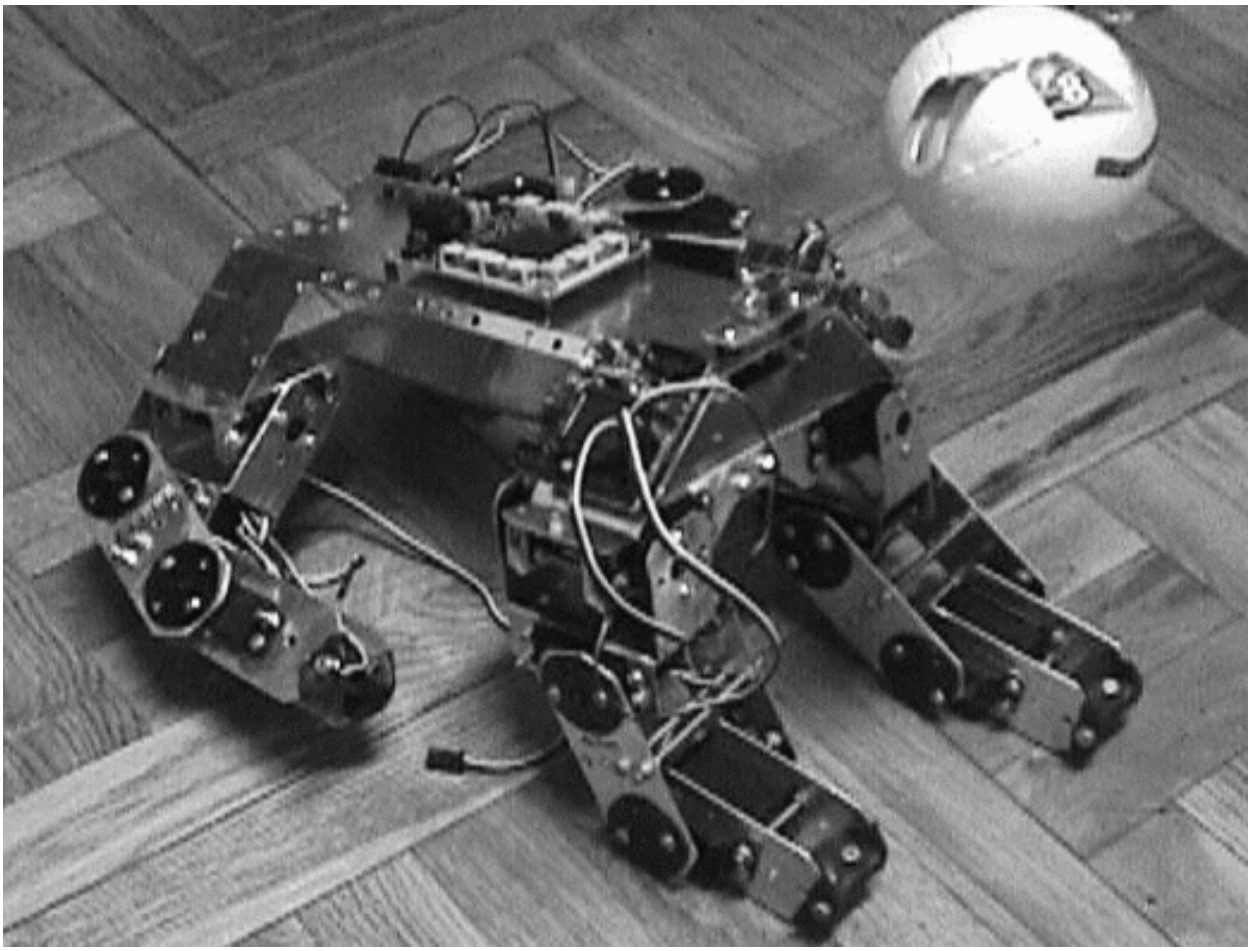


Figura 2.26: Robot descansando.

Capítulo 3

Servomecanismos

3.1 Descripción técnica

Para mover el perro se necesitan unos motores con control de posición, es decir que el eje se pueda situar en el ángulo que se desee. Además no es necesario que sean de revolución continua, con un giro de 180 grados es suficiente. Un tipo de motores que cumplen las características anteriores son los servomecanismos, que se utilizan bastante en aplicaciones de aeromodelismo para mover los alerones, subir y bajar trenes de aterrizaje, orientar hélices, acelerar o decelerar motores y un sinfín de aplicaciones más. Entre todos los modelos que existen, sobresale una marca por su calidad y prestigio, son los servomecanismos FUTABA. Dentro de la familia hay bastantes tipos con diferentes prestaciones, mayor o menor tamaño, velocidad o fuerza, pero todos ellos se controlan de la misma forma.

La conexión al exterior se realiza a través de tres cables, uno para la masa (cable negro), otro para la tensión de alimentación de 6v (cable rojo) y el último lleva la señal de control de movimiento (cable blanco). El servomotor internamente realiza un control de posición en bucle cerrado, para lo que utiliza un potenciómetro y un circuito de control. La señal que espera recibir dicho circuito es un tren de pulsos, estos pulsos se repetirán con un periodo de 20 mseg. La anchura de los pulsos indicará en qué posición se deberá quedar el eje. El centro se corresponde con una anchura de 1.3 mseg, los extremos con anchuras de 0,3 mseg y de 2,3 mseg.

Estos servos de posición son muy útiles para realizar accesorios de robots, como son los manipuladores, pinzas, brazos, en resumen todo aquello en donde el rango de movimiento no necesite revolución continua. En aplicaciones de movimiento continuo habrá que desmontar el servomotor para configurarlo como un simple motor de corriente continua con caja reductora incorporada. Al desmontarlo, se podrá optar por diferentes alternativas, cada una de ellas tendrá su aplicación más adecuada.

En la figura 3.1 se pueden ver las distintas partes del servomotor. Empezando por la parte superior se tiene: La rueda del eje de salida, la tapa de la caja reductora, los engranajes que forman la caja reductora, la caja del servomotor, la tarjeta de control (enumerados de izquierda a derecha: potenciómetro, circuito de control y motor) y por último la tapa del servomotor junto con los tornillos de sujeción.

El potenciómetro se encarga de cerrar el bucle de control, es el que examina la posición del motor. El circuito de control recibe la información del tren de pulsos y del potenciómetro y sitúa al eje del motor en su nueva posición. La caja reductora aumenta la

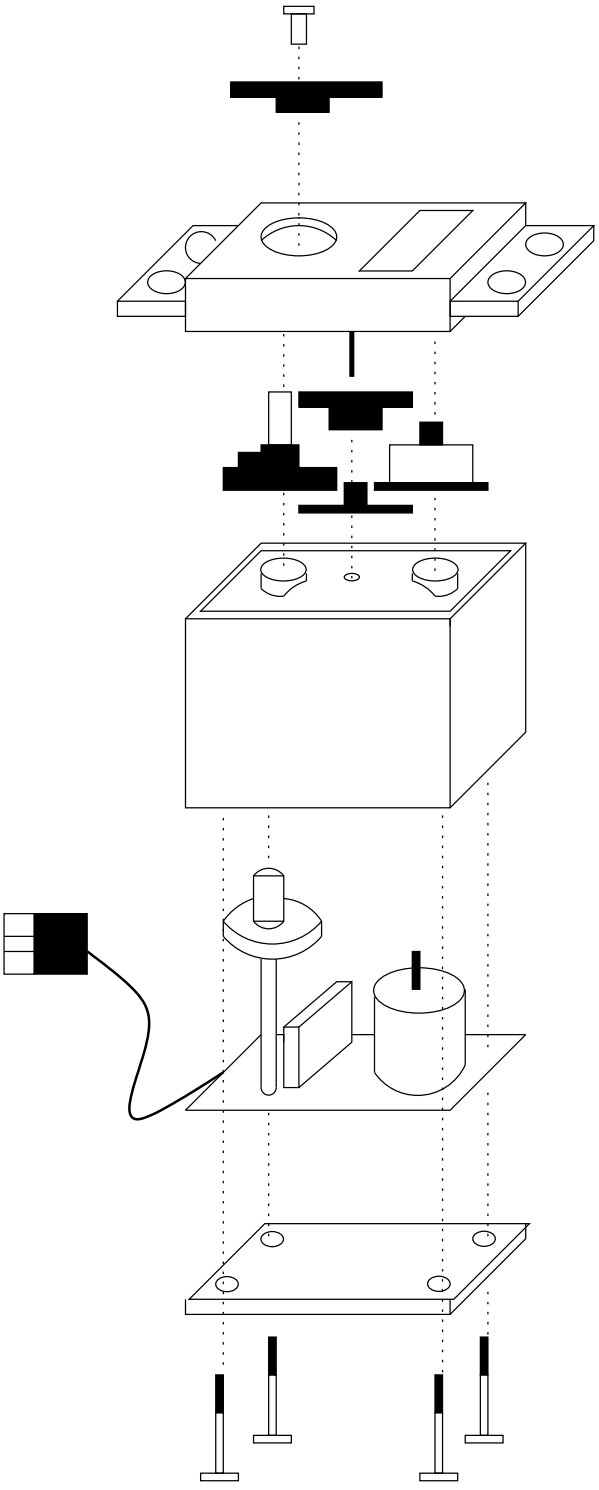


Figura 3.1: Estructura del Futaba S3003.

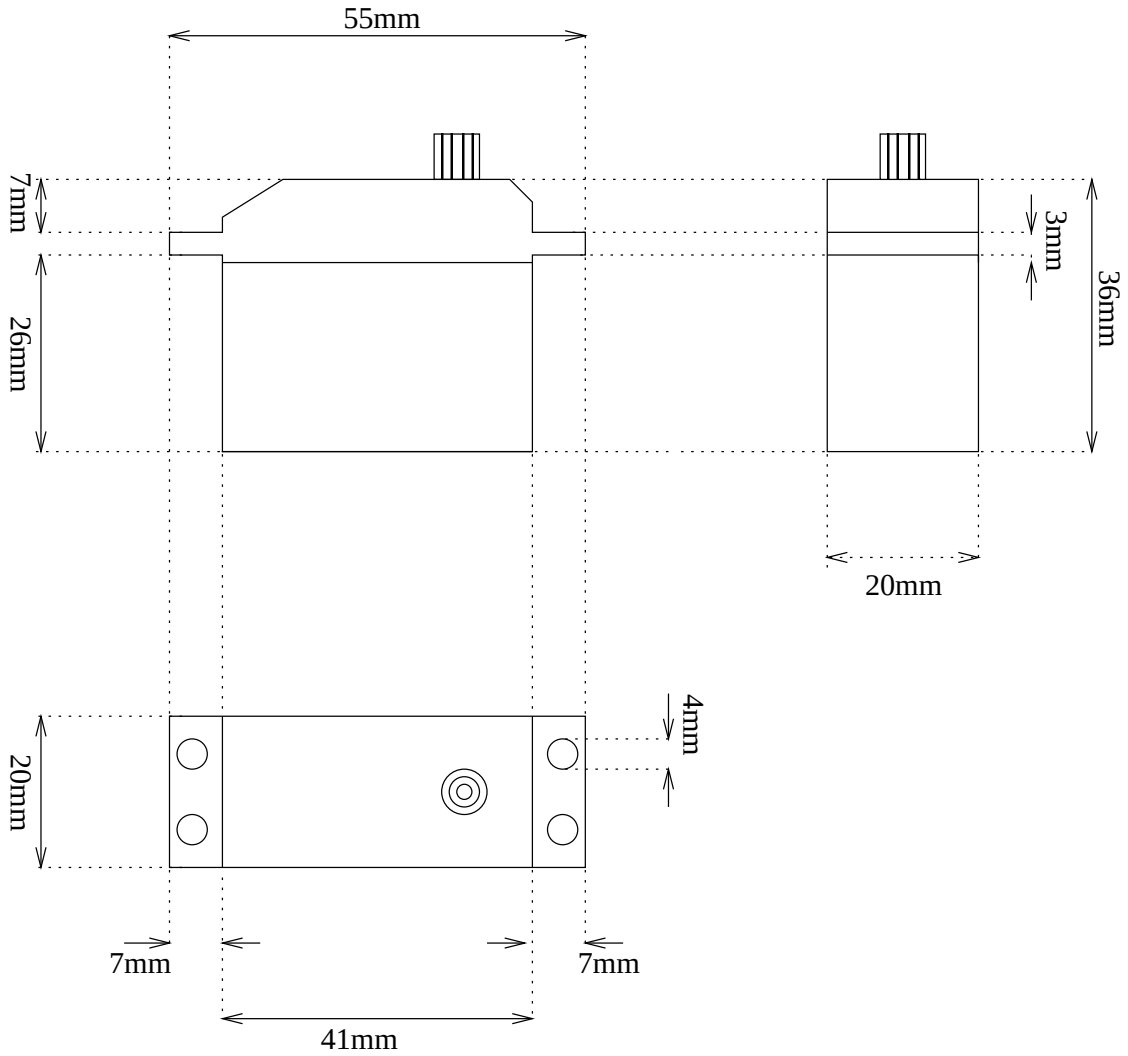


Figura 3.2: Dimensiones del Futaba S3003.

Velocidad	0.23 seg/60º	0.23 seg/60º
Par de salida	3.2 kg-cm	44.4 oz-in
Tamaño	40.4 x 19.8 x 36 mm	1.59 x 0.78 x 1.42 in
Peso	37.2 g	1.31 oz

Tabla 3.1: Especificaciones del Futaba S3003.

Velocidad	0.11 seg/60º	0.11 seg/60º
Par de salida	5.7 kg-cm	79.2 oz-in
Tamaño	39 x 20 x 37.4 mm	1.54 x 0.79 x 1.47 in
Peso	55 g	1.94 oz

Tabla 3.2: Especificaciones del Futaba S9404.

fuerza de salida del eje y reduce la velocidad del mismo. También existen un par de topes mecánicos, uno esta en el engranaje de salida del servomotor y el otro es el potenciómetro del circuito de control.

En la figura 3.3 se representa la señal de control que espera recibir un servomecanismo, se recuerda que la anchura de los pulsos determina la posición en que se deberá parar el eje.

En las tablas 3.1 y 3.2 se resumen las especificaciones dadas por el fabricante el servomecanismo Futaba S3003 y Futaba S9404 respectivamente. La tensión de alimentación es de 5 voltios y la amplitud de la señal de control es TTL (nominal 5 voltios).

Los servomecanismos tienen internamente una serie de componentes que en conjunto caracterizan su funcionamiento. Estos son: circuito de control, potenciómetro interno y el tope mecánico en el eje de salida. Según se modifiquen, se pueden obtener diferentes comportamientos. A continuación se enumeran dichos componentes y las características que se añaden o se eliminan:

1. **Con Control**: El circuito de control se encarga de recibir la modulación tipo pulsos y ordenar al motor situarse en una posición relacionada con la anchura del pulso recibido. Para ello es necesario que esté el potenciómetro. Si éste no se encuentra, el circuito de control sólo puede mover el eje del motor hacia la izquierda o hacia la derecha. Esta característica se puede emplear para evitar usar etapas de potencia para mover el motor, el inconveniente es que se manejan señales de control más complicadas.

2. **Sin control**: Al quitar el circuito de control se tendrá que usar un circuito de potencia externo, pero ahora la señal de control será más sencilla, no será obligatorio generar modulación. Otros inconvenientes se encuentran a la hora de cerrar el bucle. Para ello es necesario utilizar el potenciómetro pero el valor de éste hay que procesarlo con un circuito exterior.

3. **Con potenciómetro**: Establece un tipo de tope mecánico. Con él se pueden realizar bucles cerrados de control. Cuando se tiene el circuito de control el bucle se cerrará internamente. Esto es muy útil en aeromodelismo, ya que, por control remoto indicamos la posición que debe tomar el eje y el propio servomotor se encarga de

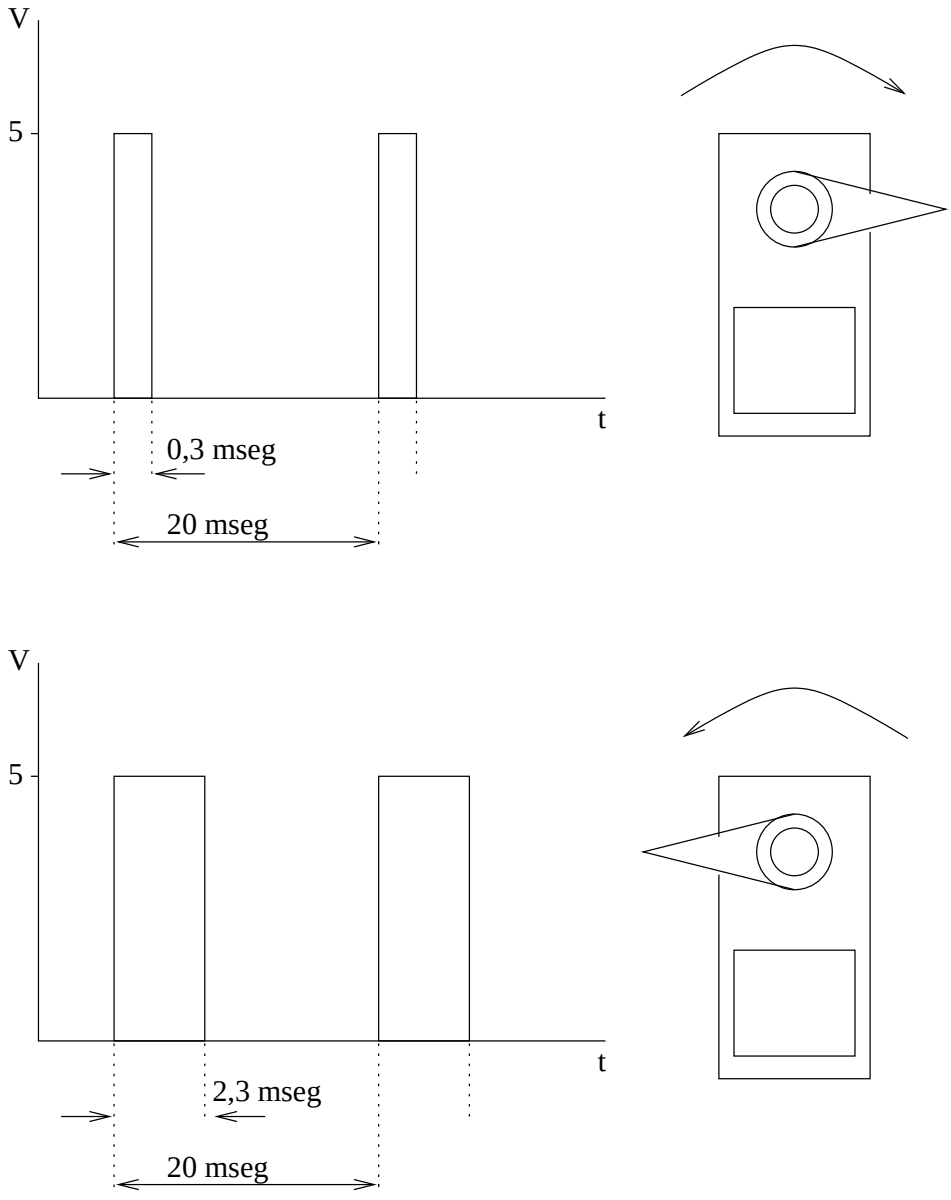


Figura 3.3: Señal de control del Futaba S3003.

Sin revolución continua	Sin potenciómetro	Con control	Ej: Sin etapa potencia
(Con Topes mecánicos)		Sin control	No se suele usar
	Con potenciómetro	Con control	Ej: Estado original
		Sin control	Ej: PIDs
Con revolución continua	Sin potenciómetro	Con control	Ej: Sin etapa potencia
(Sin topes mecánicos)		Sin control	Ej: Microbots

Tabla 3.3: Modos de funcionamiento de los servomecanismos.

buscarla y posicionar su eje en ella. De esa forma, no hay que transmitir datos desde el avión hasta el mando de control remoto. Si no hay circuito de control el bucle se tendrá que cerrar externamente.

4. **Sin potenciómetro**: Se elimina el primer tope mecánico y la posibilidad de cerrar el bucle. Si se mantiene el circuito de control se puede realizar un control izquierda-derecha en bucle abierto por medio de los pulsos, evitando poner un circuito de potencia externo.

5. **Con topes mecánicos**: Sólo se tienen giros limitados, su aplicación es muy útil en brazos robots, pinzas, manipuladores, mecanismos ON/OFF, aeromodelismo, etc.

6. **Sin topes mecánicos**: Se eliminará el tope del rodamiento y el potenciómetro, por lo tanto se pierde la posibilidad de cerrar el bucle internamente.

Combinando los seis puntos anteriores se obtiene la tabla 3.3 que describe todas las formas de funcionamiento y su aplicación más común. De todas ellas las más frecuentes son utilizar el servomecanismo en su estado natural (con todos los elementos) o utilizarlo sin circuito de control y sin potenciómetro (sin los elementos). Alguna aplicación intermedia utiliza el servomecanismo en su estado original pero sin el circuito de control. Como ejemplo de cada caso se citan respectivamente: aplicaciones de aeromodelismo (alerones, timón,etc.), aplicaciones en microbots (ruedas motrices) y, por último, aplicaciones con etapa de potencia que cierran el bucle con el potenciómetro interno, por ejemplo para controles PID.

En las aplicaciones de aeromodelismo se mantiene el estado original para no tener que enviar información desde el avión, barco, coche, etc. hasta el mando de radio control. Se envía la posición codificada con pulsos y el servomotor se encarga del posicionamiento y control. Pero en algunos casos puede ser útil tener confirmación de cuándo se ha llegado a la posición. Para ello se puede sacar el cable del cursor del potenciómetro al exterior y analizarlo con algún tipo de electrónica. Es evidente que esto se aplica sólo en sistemas que tengan transmisión de información bidireccional, es decir, del control a los actuadores y de los sensores al control.

En los microbots con ruedas se necesita rotación continua, para ello hay que eliminar todos los topes mecánicos, incluyendo el potenciómetro. Para controlar el movimiento (giro a izquierdas o a derechas) se puede utilizar el circuito de control o directamente eliminarlo. Si se mantiene el circuito, la señal de control será la modulada, para indicar hacia dónde debe girar el motor se empleará el ancho máximo (2.3 ms) y el mínimo (0.3ms). Lo bueno de esta técnica es que no hará falta usar una etapa de potencia previa. Si por el

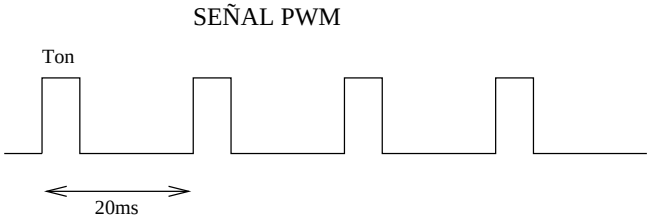


Figura 3.4: Señal PWM para el control de un Futaba

contrario se elimina también el circuito de control, se tendrá que usar una etapa de potencia externa aunque las señales de control serán más sencillas (directamente tensión en bornas del motor).

Por último, en algunas aplicaciones de robótica se puede utilizar una alternativa a la utilización del servomecanismo en su estado original. Consiste en eliminar el circuito de control pero mantener el potenciómetro. Con un hardware externo se puede leer el valor de éste y ver si se ha llegado o no a la posición deseada. Al quitar el circuito de control se usará una etapa de potencia externa.

Como resumen, hay que tener presente que el circuito de control ahorra la etapa de potencia pero se complica la señal de control, al utilizar el potenciómetro se puede cerrar el bucle pero no será posible tener movimiento continuo.

3.2 Control de servomecanismos

La señal de control de un servomecanismo es digital y para generarla se deberá usar algún tipo de circuito digital, por ejemplo autómatas en lógica TTL o CMOS, autómatas en PALs, FPGAs, o microcontroladores. Aunque en el capítulo siguiente se estudiará la electrónica en detalle, adelantamos que todas las señales de control las genera una electrónica basada en el microcontrolador 68hc11 de Motorola. En concreto se utilizarán las tarjetas CT6811 y BT6811. Por eso todo el software descrito aquí hace referencia al microcontrolador 68hc11 y a la tarjeta CT6811, que es una entrenadora para dicho micro.

En este apartado se muestran programas ejemplo de generación de la señal PWM necesaria para que un motor Futaba se sitúe en una posición concreta. Las características de esta señal PWM son las que se han descrito anteriormente y que se resumen en la figura 3.4: señal cuadrada de 20ms de periodo. Variando el parámetro Ton se varía la anchura del pulso y con ello la posición del Futaba. El parámetro Ton está comprendido entre los valores 0.3ms y 2.3ms, correspondiente a las dos posiciones extremas del Futaba.

La generación de esta señal es muy sencilla cuando se utilizan los comparadores que incorpora el 68HC11. En total se necesitan dos comparadores, uno para generar una interrupción cada 20ms y otro para generar el pulso de Ton ms. Sin embargo, se pueden controlar hasta 4 motores Futaba simultáneamente utilizando los 5 comparadores disponibles en el microcontrolador. Con un comparador se genera una interrupción periódica de 20ms común para todos los Futaba y con los 4 comparadores restantes se generan las anchuras de los pulsos de cada Futaba.

A continuación se describen los programas en ensamblador del 68hc11 que permiten controlar los servomecanismos. Primero se explicará cómo mover uno, luego como hac-

Tiempo	Numero de Tics	Descripción
0.1ms	200	Utilizado para hacer cálculos
1ms	2.000	Utilizado para hacer cálculos
0.3ms	600	Posición de un extremo
1.3ms	2.600	Posición central
2.3ms	4.800	Posición del otro extremo
20ms	40.000	Valor del periodo

Tabla 3.4: Numero de tics y sus valores de tiempo asociados

erlo con interrupciones y finalmente un programa cliente-servidor para mover un Futaba desde el PC o cualquier otro dispositivo serie. Para los que no estén familiarizados con la programación del 68hc11 se recomienda el libro [9]'Microcontrolador MC68hc11: Fundamentos, recursos y programación'.

En los ejemplos posteriores se van a utilizar los comparadores 2 y 3. El comparador 2 se emplea para generar una señal del 20ms y el comparador 3 es el empleado para generar el pulso de anchura Ton.

De forma predeterminada el temporizador principal se incrementa cada 500ns, es decir, un tic del temporizador principal son 500ns. Para conseguir un periodo de 1μs se necesita que transcurran 2 tics, para 1ms 2000 tics y para 20ms 40000 tics. En la tabla 3.4 se resumen los tics necesarios para conseguir los valores empleados por el Futaba. Los valores del parámetro Ton, en *tics*, se corresponden con 600 para un extremo y 4.800 para el otro extremo. Variando Ton entre estos dos valores se consigue que el Futaba se posicione en cualquiera de las posiciones intermedias. En principio se podrían conseguir 4.800-600 = 4.200 posiciones diferentes del Futaba, sin embargo el valor mínimo apreciable por el ojo entre una posición y otra es de unos 25 tics. **Ejemplo**: Tenemos el Futaba en la posición 600. Si lo situamos en la posición 610 no se apreciará movimiento, sin embargo si se posiciona en la 625 se puede ver un ligerísimo movimiento.

La resolución aproximada es por ello de unos 25tics, lo que permite mover el Futaba a (4800-600)/25 = 168 posiciones diferentes.

Resumiendo: Cuando se trabaja con el 6811 se puede ver el Futaba como un motor que se puede situar en cualquier posición comprendida entre la 600 y la 4.800. Esto se consigue dando este valor de posición al parámetro Ton.

3.2.1 Generación de PWM sin interrupciones

En el ejemplo que se muestra a continuación se mueve el motor Futaba sin utilizar interrupciones. Este mecanismo no es el que se debe utilizar, ya que una vez que el Futaba ha alcanzado la posición, la señal PWM se deja de enviar y el Futaba deja de mantener su posición.

En este ejemplo se ha conectado un Futaba en el bit 3 del puerto B. Modificando la constante PORT se indica el puerto donde se encuentra el Futaba y con la constante BIF la máscara del bit donde se encuentra el mismo.

Lo importante de este programa es la función **moverf**, que es la que posiciona el Futaba. En el acumulador D se le pasa la posición del Futaba (valor entre 600-4800) y al llamar a

esta función el motor irá a la posición indicada. La señal enviada tiene en total 20 periodos de duración, pasado ese tiempo la función retorna. Modificando la constante NPULSOS se consigue variar la duración de la señal enviada. La onda cuadrada se genera de la siguiente manera. Se lanzan el comparador 2 y 3, como el tres contabiliza el tiempo Ton, que es menor que el periodo de la señal, se esperará a que éste termine. Desactivando la señal de salida inmediatamente después y pasando a esperar la finalización de comparador dos, es decir de los 20 mseg. Una vez finalizado se repetirá éste proceso durante NPULSOS.

El ejemplo que se presenta a continuación mueve el Futaba a un extremo, al centro, al otro extremo y finalmente al centro, donde se queda esperando a recibir un carácter por el SCI. Para probarlo se puede utilizar el MCBOOT, una calculadora HP o cualquier dispositivo que pueda enviar caracteres por un puerto serie estándar.

```
*******************************************************
* F3003.ASM                          FEBRERO   1999   *
*-----------------------------------------------------*
*  Programa ejemplo para ser ejecutado en la RAM      *
*  interna de la tarjeta CT6811.                      *
*  Ejemplo de movimiento de un Futaba. Se utiliza el  *
*  comparador 2 para esperar 20ms y el comparador 3   *
*  para generar la anchura del pulso.                 *
*                                                     *
*******************************************************

TMSK1   EQU $22
TFLG1   EQU $23
TCTL1   EQU $20
TOC2    EQU $18
TOC3    EQU $1A
TMSK2   EQU $24
TCNT    EQU $0E

* Registros del SCI

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

* Puertos del 6811

PORTA   EQU $00
PORTB   EQU $04
PORTC   EQU $03
DDRC    EQU $07
```

```
* PARAMETROS DE CONEXION DEL FUTABA

PORT    EQU PORTB   * Puerto donde esta el Futaba
BITF    EQU 0x08    * Bit donde esta el Futaba conectado
NPULSOS EQU 20      * .Numero de pulsos de la señal

* VALORES Ton PARA LAS DOS POSICIONES EXTREMAS

EXTREMO1 EQU 600    * Ancho del pulso: 0.3ms
CENTRO   EQU 2600   * Ancho del pulso: 1.3ms aprox.
EXTREMO2 EQU 4800   * Ancho del pulso: 2.3ms

* Valor del periodo del Futaba

T       EQU 40000

*************************************************
*  Equivalencias de valores de ancho del pulso:    *
*                                                  *
*  2000 ---> 1ms                                   *
*  200  ---> 0.1ms                                 *
*************************************************

        ORG $0000

        LDX #$1000
main
        LDD #EXTREMO1  * Futaba en un extremo
        BSR moverf
        LDD #CENTRO    * Futaba en el centro
        BSR moverf
        LDD #EXTREMO2  * Futaba en otro extremo
        BSR moverf
        LDD #CENTRO    * Futaba en el centro.
        BSR moverf

* Esperar a que llegue un caracter por el SCI

        BSR leer_car
        BRA main

************************************************
* Mover el Futaba a la posicion indicada por el  *
* registro D. Las posiciones posibles van desde  *
* la 600 hasta la 4800. La resolucion es aproxi- *
```

```
* madamente de 25 unidades, es decir que posi-   *
* ciones contiguas estan separas por 25 unidades.*
* EJ. la siguiente posicion a la 600 es la 625.  *
* En total se dispone de 168 posiciones:         *
*          (4800-600)/25                         *
************************************************

pos     RMB 2          * Sentido de giro
moverf
        LDY #NPULSOS
        STD pos

buclef  CPY #0
        BNE otro_pulso
        RTS
otro_pulso
        DEY
        LDD TCNT,X     * Activar comparador 2
        ADDD #T
        STD TOC2,X

        LDD TCNT,X     * Activar comparador 3
        ADDD pos       * Especificar sentido de giro
        STD TOC3,X

        LDAA #BITF     * Activar pulso
        STAA PORT,X

* Esperar que termine el comparador 3

oc3     BRCLR TFLG1,X $20 oc3
        BSET  TFLG1,X $20
        CLRA                  * Desactivar pulso
        STAA PORTB,X

* Esperar que termine el comparador 2

oc2     BRCLR TFLG1,X $40 oc2
        BSET TFLG1,X $40

        BRA buclef

************************************************
*  Rutina par leer un caracter del puerto serie *
*  (SCI). La rutina espera hasta que llegue     *
*  algun caracter                               *
```

```
* ENTRADAS: .Ninguna.                                *
* SALIDAS: El acumulador A contiene el caracter *
*          recibido                                  *
*****************************************************
leer_car BRCLR SCSR,X $10 leer_car
         LDAA SCDR,X
         RTS

         END
```

3.2.2 Generación del PWM mediante interrupciones

El control mediante interrupciones es mucho más interesante. Permite que mientras el 6811 está realizando cálculos, las señales de PWM de control del Futaba se generen en segundo plano o *background*. Aunque el 6811 se encuentre esperando a que llegue un carácter por el SCI, la señal PWM de control del Futaba se sigue produciendo, lo que hace que el Futaba se encuentre fijo en una posición y no se pueda mover por cargas externas.

El interfaz que se ofrece es el mismo que en el caso del programa f3003.asm. En el acumulador D se pasa la posición del Futaba y al llamar a la función se sitúa el Futaba en esa posición.

El ejemplo mueve el Futaba de un extremo a otro cada vez que se recibe un carácter por el SCI.

```
*****************************************
*  F3003INT.ASM              FEBRERO 1999  *
*---------------------------------------*
* Programa ejemplo para ser ejecutado en la *
* tarjeta CT6811.  Este  programa se debe   *
* cargar en la RAM interna del 6811.        *
*                                           *
* Movimiento de un servo Futaba mediante    *
* interrupciones                            *
*****************************************

* Registros del SCI

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

* Registros del temporizador

TMSK1   EQU $22
```

```
TFLG1   EQU $23
TCTL1   EQU $20
TOC2    EQU $18
TOC3    EQU $1A
TMSK2   EQU $24
TCNT    EQU $0E

* Registros de puertos

PORTA   EQU $00
PORTB   EQU $04
PORTC   EQU $03

* PARAMETROS DE CONEXION DEL FUTABA

PORT    EQU PORTB
BITF    EQU 0x08

* VALORES DEL ANCHO DEL PULSO DE PWM

EXTREMO1 EQU 600      * Ancho del pulso: 0.3ms
CENTRO   EQU 2600     * Ancho del pulso: 1.3ms aprox.
EXTREMO2 EQU 4800     * Ancho del pulso: 2.3ms

* Valor del periodo del Futaba
T       EQU 40000

*****************************************************
*  Equivalencias de valores de ancho del pulso:     *
*                                                   *
*  2000 ---> 1ms                                    *
*  200  ---> 0.1ms                                  *
*****************************************************

         ORG $0000

         BRA inicio

posicion RMB 2

inicio
        LDX #$1000
        LDAA #$60
        STAA TMSK1,X   * Permitir la interupcion
        CLI            * del comparador 2 y 3
main
```

```
        BSR leer_car
        LDD #EXTREMO1
        BSR posicion_futaba

        BSR leer_car
        LDD #EXTREMO2
        BSR posicion_futaba

        BRA main

************************************************
*  Mover el Futaba a la posicion indicada por el *
*  registro D. Las posiciones estan comprendidas *
*  en el rango 600-4800, aunque la resolucion es *
*  de 25 unidades.                               *
************************************************

posicion_futaba
        SEI
        STD posicion
        CLI
        RTS

************************************************
* Rutina par leer un caracter del puerto serie  *
* (SCI). La rutina  espera hasta  que llegue    *
* algun cara cter.                               *
* ENTRADAS: .Ninguna                            *
* SALIDAS: El acumulador A contiene el cara cter *
*          recibido                             *
************************************************

leer_car BRCLR SCSR,X $10 leer_car
         LDAA SCDR,X
         RTS

*******************************************
* Rutina de interrupcion del comparador 2   *
*******************************************

oc2
        BSET TFLG1,X $40  * flag del comparador 2 a cero

        LDD TCNT,X
        ADDD #T           * Actualizar comparador 2
        STD TOC2,X
```

```
        LDD TCNT,X        * Activar comparador 3
        ADDD posicion     * anchura pulso
        STD TOC3,X

        LDAA #BITF        * activar pulso
        STAA PORT,X
        RTI

************************************************
*  Rutina de interrupcion del comparador 3     *
************************************************
oc3
        BSET TFLG1,X $20  * flag del comparador 2 a cero
        CLRA              * desactivar pulso
        STAA PORT,X
        RTI

*******************************
*  Vectores de interrupcion   *
*******************************
        ORG $00D9
        JMP oc3

        ORG $00DC
        JMP oc2
```

3.3 Programación desde el PC

En este apartado se presentan ejemplos de cómo mover un Futaba desde el PC utilizando la CT6811. El software se ha desarrollado en Linux y se utiliza la librería CTS.1.2 de Microbótica, S.L.[1]. La CT6811 es una tarjeta microcontroladora de propósito general que se estudiará en el siguiente capítulo.

Para controlar un Futaba desde el PC se utiliza la filosofía de cliente-servidor. En la memoria RAM de la CT6811 se ejecuta un programa servidor que controla el Futaba. El programa cliente se ejecuta en el PC y envía al servidor la posición donde se quiere situar el motor. Ambos programas se comunicarán mediante un sencillo protocolo, consistente en una trama de bytes de longitud variable. Cada servicio tiene su propia trama especificada en la tabla 3.5. El primer byte de ésta se corresponde con el número de servicio, los dos siguientes (DIR) definen la dirección que se quiere leer o escribir del servidor. Los dos siguientes (TAM) definen el tamaño del bloque que se va a recibir o a enviar. Y por último, en el caso del servicio STORE, irán todos los bytes que se van a enviar al servi-

[1]La librería CTS.1.4. de Microbótica es libre y se puede encontrar en la dirección: www.microbotica.es, en la sección download y dentro de ella en la zona software.

Servicio	Dirección	Tamaño bloque	Contenido bloque	Descripción
65 (1 byte)	DIR (2 bytes)	TAM (2 bytes)	—	LOAD: del micro al PC
66 (1 byte)	DIR (2 bytes)	TAM (2 bytes)	TAM bytes	STORE: del PC al micro
68 (1 byte)	—	—	—	ALIVE: Supervivencia

Tabla 3.5: Protocolo entre el cliente y el servidor.

dor (la CT6811). La comunicación siempre la inicia el cliente, de tal manera que si pide el servicio ALIVE, tan sólo enviará el primer byte y se quedará esperando una respuesta del servidor, si todo va bien tiene que recibir el carácter ASCII J. El caso contrario significa que la conexión se ha perdido. Si el cliente desea recibir datos del servidor, es decir, se quieren obtener datos de la CT6811 en el PC, enviará el código 65, seguido de la dirección a partir de la cual se localizan los bytes deseados y seguido por el tamaño del bloque que se quiere leer. Una vez enviada la trama, el servidor procederá a enviar al PC, todo el bloque requerido. Por último está el servicio de transmisión del PC a la CT6811, que es el que se va a utilizar para mandar la posición de los motores. Este servicio comienza con el byte 66, continua con la dirección a partir de la cual se van a depositar los datos, con el tamaño del bloque que se va a enviar y al final todos los datos del bloque. El servidor leerá todos estos y los almacenará en la posición indicada por DIR.

3.3.1 El programa servidor: FSERVER

El **FSERVER** es un programa servidor que dispone de los servicios *check_conexion o alive, load* y *store*. El primero sirve para pedir datos a la CT6811 desde el PC; el segundo sirve para enviar datos desde el PC a la CT6811 y el tercero permite chequear la comunicación entre ambos dispositivos. Este servidor está constantemente generando una señal PWM mediante interrupciones. Existe una variable que contiene el valor del parámetro Ton de la señal PWM. Al modificar esta variable desde el programa cliente utilizando instrucciones *STORE* se consigue cambiar la posición del Futaba. El servidor se muestra a continuación:

```
************************************************
* FSERVER.ASM                FEBRERO 1999      *
************************************************
*  Programa servidor para mover un motor Futaba.*
*  Este servidor es compatible con el CTSERVER. *
*  Incluye los servicios:                       *
*        - check_conexion()                     *
*        - load()                               *
*        - store()                              *
*                                               *
*  Mediante interrupciones se genera todo el    *
*  rato una señal pwm para situar el Futaba en  *
*  una posicion. La posicion del Futaba esta    *
*  determinada por el parametro ton (anchura del*
*  pulso) que se puede cambiar mediante un      *
```

```
*  servicio de store en la direccion 2.         *
************************************************

*---- Definicion de los servicios
TMPC    EQU 65   * Transferencia datos del micro al PC
TPCM    EQU 66   * Transferencia datos del PC al micro
ALIVE   EQU 68   * Servicio de supervivencia

OKALIVE EQU 'J'

* --- Registros del SCI

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

* --- Registros del temporizador

TMSK1   EQU $22
TFLG1   EQU $23
TOC1    EQU $16
TOC2    EQU $18
TMSK2   EQU $24
TCNT    EQU $0E

* Registros de puertos
PORTA   EQU $00
PORTB   EQU $04
PORTC   EQU $03

* PARAMETROS DE CONEXION DEL FUTABA
PORT    EQU PORTB    * Puerto conectado el Futaba
BITF    EQU 0x08     * Bit el Futaba conectado

* --- Valores iniciales de los parametros
T       EQU 40000       * Parametro T (20ms)
TON     EQU EXTREMO1    * Parametro Ton


*****************************
*  Comienzo del servidor    *
*****************************

          ORG $0000
```

```
         BRA inicializar

* Los parametros del FSERVER se situan en las
* posiciones 2,4,6 y 8 de la RAM interna

parton   RMB 2   * Parametro ton
dirini   RMB 2   * Direccion inicio
longbloq RMB 2   * Tamano bloque
codigo   RMB 1   * Codigo de servicio solicitado

inicializar

       LDX   #$1000
waitt  BRCLR SCSR,X $40 waitt * estabilizar sci
       LDD   #TON
       STD   parton   * Inicializar parametro Ton
       LDAA  #$80
       STAA  TMSK1,X  * Permitir interrupcion T
       CLI            * Activar las interrupciones

*--- Bucle principal del servidor --

bgn_srv
        LDX #$1000
        BSR leer_car    * leer servicio solicitado
        STAA codigo     * Almacenar temporalmente
        CMPA #ALIVE     * ¿ Supervivencia?
        BEQ serv_alive
        CMPA #TMPC      * ¿LOAD (micro-PC)?
        BEQ serv_tmpc
        CMPA #TPCM      * ¿STORE (PC-micro)?
        BEQ serv_tpcm
        JMP bgn_srv

*--- Servicio de Supervivencia
serv_alive
         LDAA #OKALIVE
         BSR enviar
         JMP bgn_srv

*--- Servicio STORE
serv_tpcm
         INC codigo
         BRA comun
```

```
*--- Servicio LOAD
serv_tmpc
         CLR codigo

*--- Codigo comun a LOAD y STORE
comun   BSR leer_dir
        STY dirini
        BSR leer_dir      * Leer tamano del bloque
        STY longbloq      * Almacenar tamano bloque

bucle_tmpc
        CPY  #0           * ¿Bloque transferido?
        BEQ  fin_ser_tmpc * si -> fin
        LDY  dirini
        LDAA codigo
        TSTA
        BNE  tpcm
*                         * Transferencia micro->pc
        LDAA 0,Y          * Leer un byte
        BSR  enviar       * Enviar byte
        BRA  sigue
*                         * Transferencia pc->micro
tpcm    BSR  leer_car     * Leer un byte
        SEI               * Inhibir interrupciones
        STAA 0,Y          * Almacenar byte
        CLI               * Habilitar interrupciones

sigue   INY
        STY dirini        * Apuntar siguiente direccion
        LDY longbloq
        DEY
        STY longbloq      * Un byte menos por enviar
        BRA bucle_tmpc
fin_ser_tmpc
        JMP bgn_srv

*********************************************
*  Rutina para leer un caracter del puerto  *
*  serie (SCI). La rutina espera hasta que  *
*  llega algun caracter                     *
*  ENTRADAS:.Ninguna                        *
*  SALIDAS: caracter recibido en acumulador A *
*********************************************

leer_car
        BRCLR SCSR,X $20 leer_car
```

```
        LDAA  SCDR,X
        RTS

*********************************************
* Enviar un caracter por el puerto serie SCI  *
* ENTRADAS: caracter a enviar en acumulador A *
* SALIDAS: .Ninguna                           *
*********************************************

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

*********************************************
* LEER_DIR                                    *
*********************************************
* Leer una direccion por el SCI y devolverla  *
* en el registro Y                            *
* ENTRADAS: .Ninguna                          *
* SALIDAS: Y contiene la direccion leida      *
*********************************************

leer_dir
        PSHA
        PSHB

        BSR leer_car  * Leer byte bajo de la direccion
        TAB           * B contiene byte bajo direccion
        BSR leer_car  * En A byte alto de la direccion
        XGDY          * Y contiene la direccion leida

        PULB
        PULA
        RTS


*********************************************
*              INTERRUPCION  T                *
* Rutina servicio interrupcion del comparador 1 *
*********************************************
intT
        BSET TFLG1,X $80  * flag del comparador 1 a cero
        LDD TCNT,X
        ADDD #T           * Actualizar comparador 1
        STD TOC1,X
        LDAA #BITF
```

```
        STAA PORT,X       * Poner senal a ON
        LDD TCNT,X        * La interrupcion TON se
        ADDD parton       *   produce cuando transcurran
        STD TOC2,X        *   Ton tics de reloj
        BSET TMSK1,X $40  * Permitir interrupcion Ton
        RTI

***********************************************
*            INTERRUPCION Ton                 *
* Rutina servicio interrupcion del comparador 2 *
***********************************************
intTon

        BSET TFLG1,X $40  * flag del comparador 2 a cero
        CLRA
        STAA PORT,X       * Poner senal a OFF
        BCLR TMSK1,X $40  * Deshabilitar interrupcion Ton
        RTI


*************************************
*  VECTORES DE INTERRUPCION         *
*************************************

        ORG $00DC       * Interrupcion Ton. Comparador 2
        JMP intTon

        ORG $00DF       * Interrupcion T. Comparador 1
        JMP intT

        END
```

3.3.2 Un programa cliente: futaba.c

Este programa permite mover el Futaba a diez posiciones diferentes, cada una de ellas determinada al pulsar las teclas 0-9. El programa está realizado en C y utiliza la librería CTS para comunicarse por el puerto serie con la CT6811, tarjeta donde reside el servidor y a la cual están conectados los motores. La librería CTS implementa internamente las tramas del protocolo para comunicarse con el servidor, ofreciendo tres funciones : *store, load y check_conexion*. Tanto *load* como *store* trabajan con un byte, es decir con ellas sólo se puede enviar o recibir un byte. La primera tiene dos parámetros: primero el valor del byte a enviar y segundo la dirección donde se quiere depositar tal valor. La segunda devuelve el byte cuya dirección de memoria se ha pasado como parámetro a la función. La función *check_conexion()* devuelve uno si hay conexión o cero si no la hay. Su utilidad es permitir comprobar la comunicación con el microcontrolador antes de realizar cualquier intercambio de información con él. De esta manera, si no hay conexión, nuestra aplicación puede

sacar un mensaje indicándonos que se ha detenido toda comunicación con el microcontrolador.

La función más importante del programa es *set_pos()*, que es la que posiciona el Futaba. Hay que pasarle como parámetro la posición del Futaba, en el intervalo 600-4800.

```
/****************************************/
/* FUTABA.C                FEBRERO 1999  */
/****************************************/
/*  Programa para controlar la posicion  */
/*  de un Futaba a traves del FSERVER    */
/*                                       */
/*  Mediante las teclas 0-9 se situa el  */
/*  Futaba en diferentes posiciones      */
/****************************************/


/* Libreria CTS 1.2. */

#include "r6811pc.h"
#include "serie.h"
#include "bootstrp.h"
#include "s19.h"

/* Direcciones de acceso de
   los parametros en el FSERVER */

#define TON 0x02

/* Posiciones del Futaba */
#define EXTREMO1 600
#define EXTREMO2 4800

int i=0;
int m=0;

void accion_load()
/*****************************************/
/* Accion a realizar al cargar el servidor  */
/*****************************************/
{
  if (m==16 || i==255) {
    m=0;
    print ("*");
  }
  i++;
```

```
  m++;
}

int cargar_fserver()
/*************************************/
/* Cargar el servidor FSERVER en la RAM */
/* de la CT6811                      */
/*************************************/
{
  char *caderror;
  S19 fs19;

  if (abrir_s19("fserver.s19",&fs19,1)==0) {
    caderror=(char *)geterrors19();
    printf ("Error: %s\n",caderror);
    return 0;
  }
  printf ("\nCargando FSERVER:\n\n");
  printf("0%%     50%%   100%%\n");
  print ("...............\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b");

  if (cargars19_ramint(fs19,accion_load)==0) {
    printf ("\nError: %s\n",getloaderror());
    cerrar_s19(fs19);
    return 0;
  }
  printf (" OK!!\n\n");
  cerrar_s19(fs19);
  return 1;
}

void mover_palito()
/*********************************************/
/* Pequeño efecto para mostrar el estado de la */
/* conexión entre el servidor y el cliente     */
/*********************************************/
{
  static char fase[5]={'-','\\','|','/'};
  static int pf=0;

  print ("\b\b\b\b\b\b\b\b\b\bEstablecida");
  printcar(' ');
  printcar(fase[pf]);
  print("\b\b");;  /* Dibujar el palito que gira */
  pf=(pf+1) % 4;
}
```

```
void set_pos(int valor)
/*****************************************/
/* Establecer la posicion del Futaba     */
/* utiliza el servicio store para mandar la*/
/* informacion al servidor.              */
/*****************************************/
{
  byte th,tl;

  th=valor>>8;
  tl=valor&0xFF;
  store(th,TON);
  store(tl,TON+1);
}

void mover_futaba()
/*********************************************/
/* Esta funcion hace corresponder cada una de  */
/* las posiciones del Futaba con un digito.    */
/*********************************************/
{
  int pos=EXTREMO1;
  char c=0;
  int tabla_pos[]={600,4200,3800,3400,3000,2600,
                   2200,1800,1400,1000};

  printf ("Posicionamiento del Futaba\n");
  printf ("Teclas: 0-9 cambiar posicion\n\n");

  while (hay_conexion() && c!=27) {
    set_pos(pos);
    c=getch();
    if (c>='0' && c<='9') {
      pos=tabla_pos[c-'0'];
    }
  }
}

main()
/*******************************************/
/* Funcion principal, se encarga de abrir el */
/* puerto serie, definir la consola de texto,*/
/* cargar el servidor y llamar a la función */
/* mover Futaba.                            */
/*******************************************/
```

```
{
  char c;

  if (abrir_puerto_serie(COM2)==0) {
    printf ("Error al abrir puerto serie: %s\n",
             getserial_error());
    exit(1);
  }
  abrir_consola();
  baudios(7680);
  if (cargar_fserver()==0) {
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  usleep(200000);

  check_conexion();        /* Comprobar conexion */
  if (!hay_conexion()) {
    printf("No hay conexion");
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }

  mover_futaba();

  cerrar_puerto_serie();
  cerrar_consola();
  printf ("\n\n");
}
```

3.4 Adaptación a motores de corriente continua

Los servomecanismos Futaba (figura 3.5) tienen en su interior una serie de elementos que les impiden girar más de 180º, por eso, en caso de necesitar revolución continua, hay que modificarlos. Para ello, hay que desmontarlos, quitar el potenciómetro y lijar un pequeño saliente situado en uno de los rodamientos. El proceso en concreto se puede encontrar en [6] y recientemente en la revista electrónica práctica[2], aunque también se explicará ahora para tener toda la información del servomecanismo reunida en un documento.

Para desmontar el servomecanismo es necesario quitar los cuatro tornillos posteriores y el tornillo que sujeta el eje de salida. Una vez hecho esto, se pueden quitar las cubiertas superior e inferior, dejando al descubierto unos engranajes blancos (ver figura 3.6). Estos forman la caja reductora del motor, siendo su misión la de proporcionar más par (fuerza)

[2]Electrónica Práctica Actual, número 20, sección Microbótica.

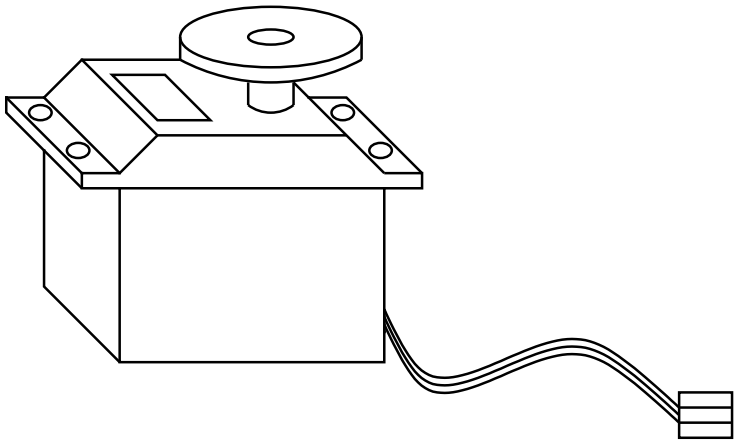


Figura 3.5: Servomecanismo Futaba.

de salida en el eje del motor y reducir la velocidad del mismo. Para quitar el circuito electrónico hay que desmontar los engranajes de la caja reductora. Con mucho cuidado para no perder las piezas se irán quitando las pequeñas ruedas dentadas blancas. Atención con el pequeño eje situado en las dos ruedas intermedias.

Una vez hecho lo anterior se puede presionar con un destornillador el saliente del potenciómetro, indicado en la figura 3.6 como RB. Se observará como el circuito electrónico sobresale por debajo. Ahora hay que hacer palanca para extraerlo entero. En la figura 3.7 se puede apreciar la apariencia de éste. Contiene tres elementos importantes, un potenciómetro, un motor y una serie de componentes electrónicos que forman el controlador del servomecanismo.

Ha llegado el momento de empezar a transformar el servomecanismo. Hasta ahora el proceso no ha sido destructivo, pero a partir de aquí sí lo será. Lo primero que hay que hacer es desoldar el motor, será la única pieza que se reutilice, el resto del circuito no se va a utilizar. El cable triple del circuito se puede cortar para utilizarlo en otras aplicaciones. Por ejemplo es muy útil para conectar sensores de tres pines a tarjetas electrónicas. En la figura 3.7 se señalan los puntos que hay que desoldar.

Antes de volver a colocar el motor en su sitio (dentro de la carcasa de plástico) se deben soldar dos cablecillos en sus bornas de alimentación. Se recomienda utilizar cablecillos de color rojo y negro, el primero se soldará a la borna positiva (la que tiene el punto rojo) y el segundo a la negativa.

Todavía hay que eliminar un tope mecánico para poder tener el servo totalmente modificado. Este tope es un pequeño saliente situado en el engranaje que forma el eje de salida del servomecanismo, ver figura 3.8. Para eliminar el tope habrá que cortarlo utilizando unas pinzas, lima, etc. Lo importante es no dañar las muescas de la rueda dentada, o peor aún, partir el eje. Una vez eliminado el saliente hay que limar la zona para que no queden restos, hay que evitar los rozamientos innecesarios en la reductora, de manera que el ruido se reduzca.

Una vez realizado lo anterior se procede a montar el servo. Lo primero es introducir el motor en el hueco cilíndrico del interior de la carcasa negra, es decir, el lugar de donde salió. Una vez introducido hay que montar la caja reductora, para ello mirar la figura 3.6 y sobre todo tener cuidado con la posición que deben tener los engranajes y nunca forzar su

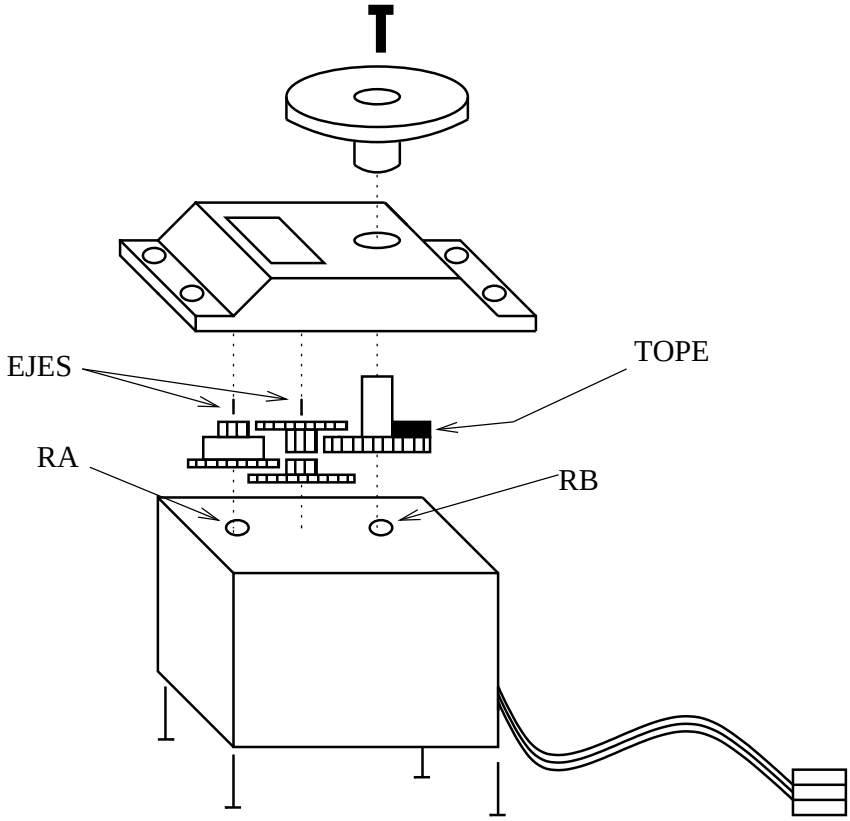


Figura 3.6: Descomposición de un servomecanismo.

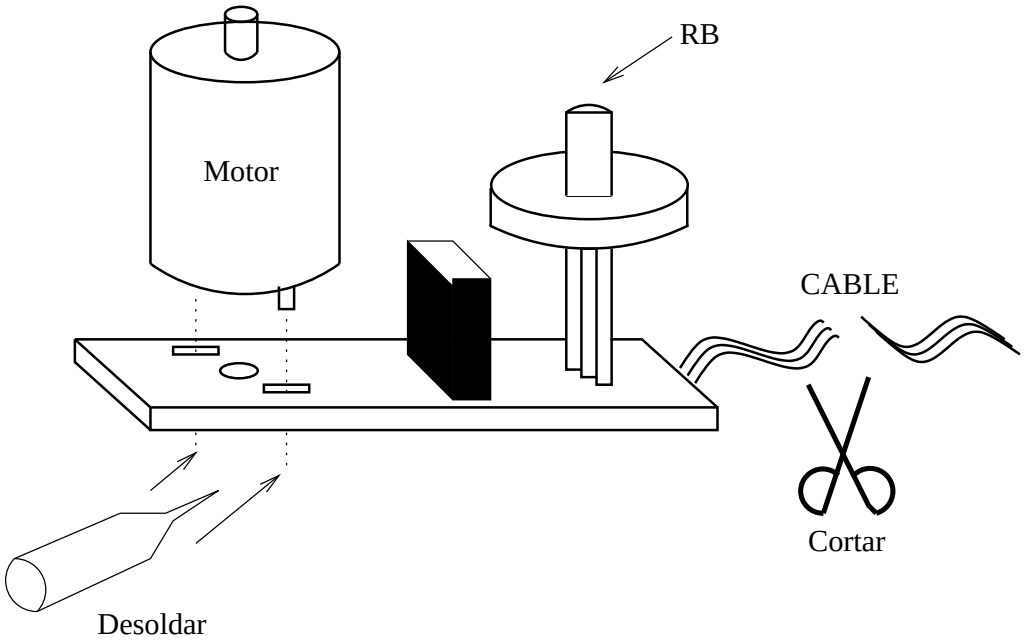


Figura 3.7: Circuito electrónico de un servomecanismo.

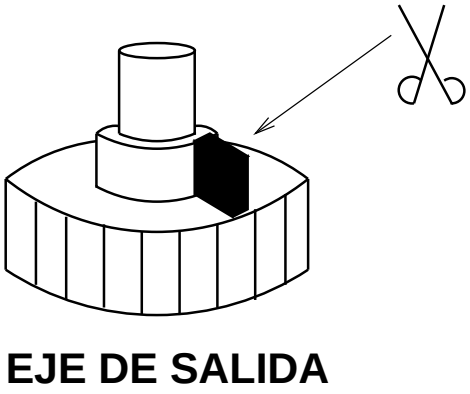


Figura 3.8: Tope mecánico del servomecanismo.

colocación. La tapa superior debe entrar sin ninguna resistencia, en caso contrario revisar los engranajes.

Por último se atornillará la tapa inferior, pero antes conviene hacer un pequeño nudo en los cablecillos del motor y dejarlo en el interior. Es una forma de proteger las soldaduras frente a tirones en los cables. Una vez realizado todo esto se tendrá el servomecanismo modificado, ahora está listo para aplicaciones que requieran giros de 360º.

Capítulo 4

Electrónica de control

4.1 Definición de la red de control

La red de control forma la base del tercer nivel de la torre Bot[1] y en relación a sus antecesores, es una de las principales innovaciones en este robot. Al analizar los robots localizados en Internet se vio que la mayoría de ellos y sobre todo cuando eran complicados, utilizaban una red de microcontroladores para mover los servos. Uno de los motivos principales de esto es por la complejidad que lleva realizar el control de los servos sin utilizar algún tipo de ayuda electrónica como son los temporizadores y las interrupciones. Otro motivo es, que aun disponiendo de algún recurso del tipo anterior, no siempre un microcontrolador los tiene en un número suficiente. Por último, aún disponiendo de la cantidad, siempre existiría la limitación de no poder ampliar el sistema y, por otro lado, en caso de necesitar menos se estarían desaprovechando recursos.

Una forma genérica de resolver todos los problemas anteriores es fabricar un módulo que pueda mover un número determinado de servos, exprimiendo lo más posible sus recursos internos y dotar a dicho módulo de la posibilidad de comunicarse con otros, es decir de unirlos entre sí, de manera que para controlar el doble de servos tan sólo haya que poner otro módulo al lado y así sucesivamente.

Para mover todos los motores del robot (al menos ocho) se necesita una electrónica capaz de generar todas estas señales de PWM. En el capítulo anterior se ha visto que para generar ésta utilizando la CT6811 se necesitan dos temporizadores: con uno de ellos se calcula el periodo de la señal (T) y con el otro el ancho del pulso (Ton), valor este último proporcional a la posición deseada del servo (ver figura 4.1). La CT6811 es una tarjeta

[1] En el apartado 1.1 se describe la torre Bot con detalle.

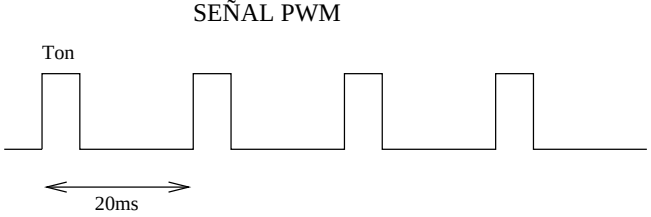


Figura 4.1: Señal de control de un servomecanismo.

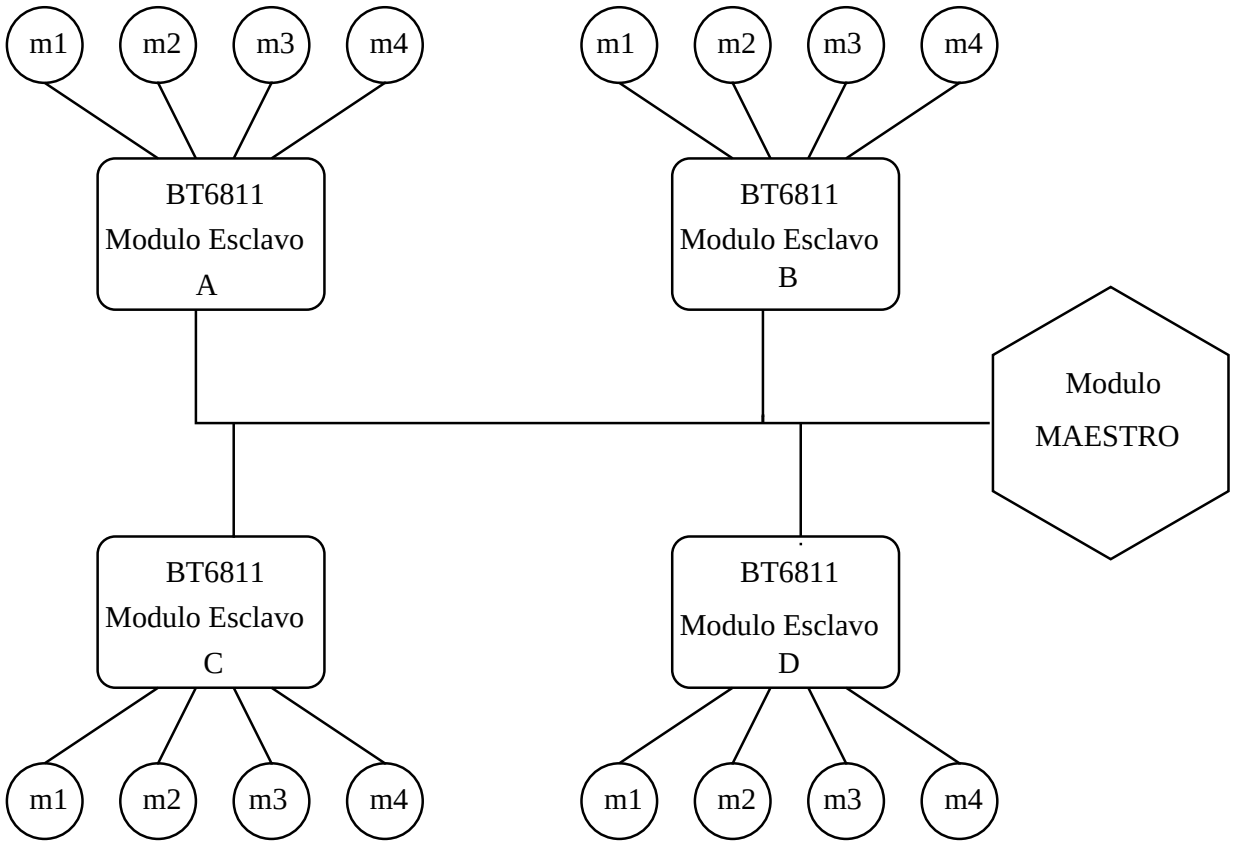


Figura 4.2: Esquema de la red de control.

electrónica que incorpora el microcontrolador 68hc11 de Motorola. Este dispone de cinco timers, por lo que teóricamente se podrá extender la aplicación al control de cuatro servos. Se utilizarán cuatro timers para modular la anchura de cada uno y otro para el periodo.

Al inicio del proyecto todavía no se tenía claro el número de motores necesarios para mover bien el robot. En un principio se pensó en utilizar ocho, pero siempre estaba la posibilidad de tener que utilizar doce. Esto, junto a la característica de que un micro 68hc11 puede mover cuatro servos hizo que se optara por diseñar un módulo electrónico que pudiera controlar cuatro servos y comunicarse con el exterior para conectarse a otros. De esta manera, la red de control seguiría siendo válida en caso de tener que acoplar cuatro motores más. Incluso en el caso futuro de que se incluyera una cabeza y rabo móviles, la red seguiría siendo válida una vez más.

Por otro lado, para lograr que todos los motores se muevan en sintonía permitiendo así que el robot avance, retroceda y gire, se necesita un programa de control unificado, que sepa en todo momento a qué posición mandar cada uno de los ejes de los motores. Este programa podría residir en uno de los módulos anteriores pero se perdería la homogeneidad entre ellos. Lo más interesante es añadir un módulo nuevo que sea el cerebro central y dejar los otros tal cual. A partir de ahora se denominará módulo maestro al que contiene el programa principal del robot y módulos esclavos a los encargados de recibir las posiciones enviadas por el módulo maestro y comunicárselas a los motores. Un diagrama de la red de control se representa en la figura 4.2.

Todavía quedan algunas cuestiones por plantear para tener una red operativa. La

primera es qué tipo de comunicación se va a establecer entre los módulos y qué volumen de información viajará a través de ella. Respondiendo a la primera pregunta se establecerá una comunicación serie síncrona (SPI) y el volumen de datos será proporcional al número de módulos que existan. Las tramas de control se explicarán en el capítulo siguiente, el del software, aunque ahora se comentarán algunas características. Por ejemplo, si se ha decido utilizar la comunicación serie en lugar de una paralelo es por que el 68hc11 está preparado para soportar las primeras ofreciendo mecanismos automáticos de gestión. Resumiendo, su utilización simplifica el software. Se ha utilizado la comunicación síncrona (SPI) en lugar de la asíncrona (SCI) porque se ha reservado ésta última para conectar el robot al PC y permitir depurar las secuencias de movimiento desde éste.

También hay que decidir si se quiere que la información viaje en ambos sentidos, del maestro a los esclavos y viceversa. La ventaja de esto es permitir que los módulos esclavos, los que manejan los servos, puedan también leer algún tipo de información del entorno mediante sensores y transmitirla al módulo maestro para que la procese. De esta manera se puede dotar al robot de una cierta inteligencia distribuida que hace que aunque uno de los módulos falle, existan otras tomas de adquisición de datos que suplanten a la anterior. Todo esto se detallará más concretamente en el capítulo del software, aunque ahora se adelanta la necesidad de añadir un cable para hacer esto. Es decir, además del cable serie se necesitará alguna línea más, que sirva para establecer un protocolo entre los diferentes módulos, de manera que si alguno de ellos quiere transmitir información pueda mandarle una señal al maestro, que será el que escuche.

Para responder a la segunda pregunta que se planteaba al principio, se debería analizar el tráfico de datos en la red y la viabilidad de la misma. Como ya se ha dicho, más adelante se expondrá con detalle el protocolo de la red, ahora se adelantan las siguientes características. El módulo maestro mandará tramas de cuatro bytes por la red, todos los módulos esclavos leerán las tramas y en función del primer byte sabrán si son para ellos o no. En caso de serlo el módulo interpretará el resto de la trama y en caso contrario dejará la interpretación y esperará a recibir la siguiente trama. En el primer prototipo realizado no se van a usar las comunicaciones en sentido opuesto, por lo que no hará falta utilizar más señales de protocolo.

Considerando lo anterior y teniendo en cuenta que la velocidad máxima del bus (SPI) es de 1 Mbit por segundo[2], que la velocidad de los servos es de 0.23seg/60º y que una trama tiene 4 bytes, se pueden sacar las siguientes conclusiones:

1. Suponiendo que el robot tiene 12 motores (caso peor) y que se envía una trama cada vez que se quiera indicar a un motor cualquiera la posición donde se debe situar, se gastarán 12 tramas, es decir 48 bytes, en reposicionar todos los servos del robot.

2. Esos 48 bytes son 384 bits y sabiendo que la red tiene una velocidad de 1 Mbit/seg, se obtiene que en 0.384mseg se habrá enviado toda la información para reposicionar todos los motores.

3. Los motores tardan 0.23seg en girar 60º por lo que en 0.384mseg girarán 0.1º.

[2]La velocidad del SPI cuando el 68hc11 está configurado como maestro es de 1 Mbit/seg, en cambio cuando está configurado como esclavo es de 2 Mbit/seg. En esta red como el flujo de información va del maestro a los esclavos, predominará la velocidad de éste, es decir 1 Mbit/seg.

Una de las características que se apreciaron al hacer las pruebas con los servos era que había un incremento mínimo en la anchura de la señal Ton que permitía apreciar con la vista un desplazamiento del eje. Esto hace que con un byte (255 valores distintos) se pueda codificar el ancho de la señal Ton. Si el rango de giro es de 180º y tenemos 255 valores distintos para indicar la posición la resolución del motor será de 0.7º. Lo que se quiere hacer ver con estos cálculos es que realmente la red no va a estar cargada ya que los motores tardarán más tiempo en llegar a su posición que lo que necesita la red para indicar una nueva posición a todos los motores del robot. Cada variación mínima de posición implica un desplazamiento de 0.7º lo que permite la llegada de al menos siete tramas. En el ejemplo siguiente se aclara la idea anterior: Supóngase que se quiere hacer un giro de 60º en el eje del motor, lo primero que se hará será indicar la nueva posición al motor, para ello se utilizará la red, pero en girar los 60º el servo tardará 0.23seg, intervalo de tiempo durante el cuál no se le deberá indicar a ese motor ningún otro valor de posición. En caso de hacerlo se moverá hasta alcanzar este último objetivo sin pasar por el primero. Vuelve a ocurrir un efecto que favorece a la red, el software tendrá que intercalar pausas para permitir que los motores lleguen a su posición final. Durante estas pausas todos los módulos podrán estar realizando otras funciones.

El lector puede considerar que codificar la posición de un motor con un byte no es suficiente y que para obtener más precisión es necesario hacerlo con dos bytes. Esto no debe plantear ningún problema, la trama se incrementará en una unidad, es decir estará formada por cinco bytes. Ahora en lugar de usar 48 bytes para reposicionar todo el robot se usarán 60, que son 480 bits a una velocidad de 1Mbit/seg, es decir un tiempo de reposicionamiento de 0.48mseg. En ese tiempo un motor puede girar 0.125º, lo que significa que si se quiere ir moviendo un motor paso a paso con incrementos de ángulo menores de 0.125 la red no lo soportará en tiempo real, es decir, el motor llegará antes a su posición que la trama que le indica la nueva posición. Ahora tendrá que quedarse parado mientras que antes se tenía que retardar la trama. Se tiene por tanto otra limitación, pero ésta no debe preocupar al lector, es todavía menos problemática que la anterior que obligaba a insertar pausas. La razón se encuentra en que la BT6811 indica posiciones finales y es el servomecanismo el que se encarga de girar el eje pasando por todas las intermedias. La única razón para que ocurra lo anterior, dar pasos con incrementos de ángulos mínimos, es que se quiera mover el robot a cámara lenta, y por lo tanto se obligue al servomecanismo a parar en cada incremento de ángulo. Pero para que se produzca el efecto visual de cámara lenta además hay que intercalar pausas y ya no habrá problemas con la red, porque entre movimiento y movimiento hay que esperar un tiempo mayor al necesario para transmitir las tramas por el SPI.

También queda por considerar otro efecto: cuando se entrene el robot con el PC, la comunicación entre ambos se hará mediante una conexión serie asíncrona (SCI) a una velocidad de 9600 baudios. Esta velocidad es menor que la del bus síncrono (SPI), por lo que, en este caso, el cuello de botella lo impondrá la conexión PC-Robot. Aunque se sature la comunicación por el SCI, la del SPI estará totalmente descargada.

Todas estas reflexiones inducen a dar por buena la viabilidad de realizar un control a través de una red distribuida, y por lo tanto se continuará con el diseño del módulo esclavo, llamado a partir de ahora BT6811. Por último, el lector puede estar preguntándose cuál es el porcentaje de utilización de la CPU del 68hc11, si le dará tiempo a implementar las cuatro señales de PWM, y si se podrán estar realizando otras tareas en paralelo. Todas

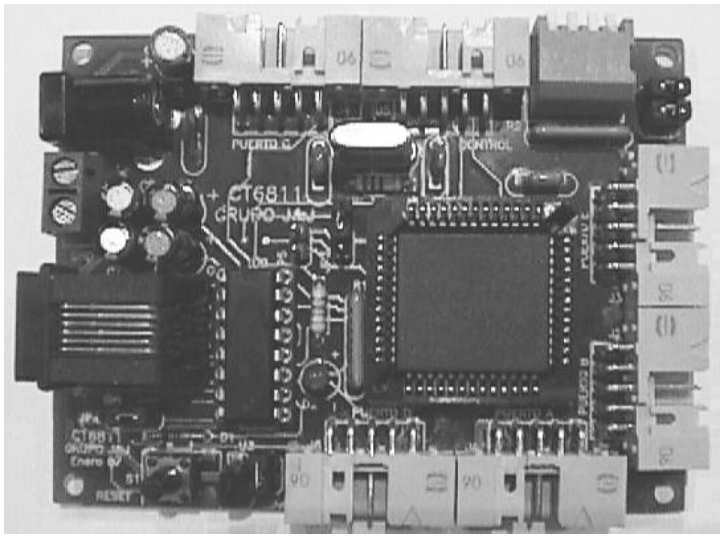


Figura 4.3: Foto de la tarjeta electrónica CT6811.

estas preguntas se responden en el capítulo del software, pero se adelanta que son todas afirmativas. El microcontrolador 68hc11 está la mayor parte del tiempo desocupado.

4.2 Módulo maestro: CT6811

El módulo maestro de la red será el que ejecute el programa principal del robot enviando las tramas de control a los módulos esclavos, que a su vez serán los que controlen los servomotores. En lugar de diseñar un nuevo hardware para tal función se ha optado por utilizar la tarjeta CT6811 de Microbótica (figura 4.3). Dicha tarjeta se utiliza para desarrollar aplicaciones hardware y software basadas en el microcontrolador MC68HC11 de Motorola. Pero, no sólo sirve para el desarrollo de *prototipos*, sino que también está pensada para funcionar en *sistemas profesionales* . Se trata de una tarjeta muy versátil que ha sido desarrollada por integrantes de Microbótica y cuyos esquemas están disponible en su web.

Entre las características principales de la CT6811 se destaca su posibilidad de utilización en distintos modos, la existencia de conectores para acceder a todos los pines del microcontrolador y su tamaño reducido. Respecto a la primera característica hay que mencionar que todos los modos se han utilizado en alguna de las fases del proyecto:

1. El modo **entrenador**: Al desarrollar el software se iban probando los algoritmos en la misma CT6811. De esa forma se verificaba el correcto funcionamiento.

2. El modo **autónomo**: Una vez terminado el programa principal se deja grabado en la tarjeta y a partir de ahí no es necesario mandar dicho programa desde el PC cada vez que se quisiera poner en marcha el robot.

3. El modo **periférico inteligente**: Este modo se ha utilizado para buscar las secuencias de movimiento del robot. Al ser bastante complicado calcularlas a ojo, se ha desarrollado un programa en el PC que permite generarlas, grabarlas y más tarde

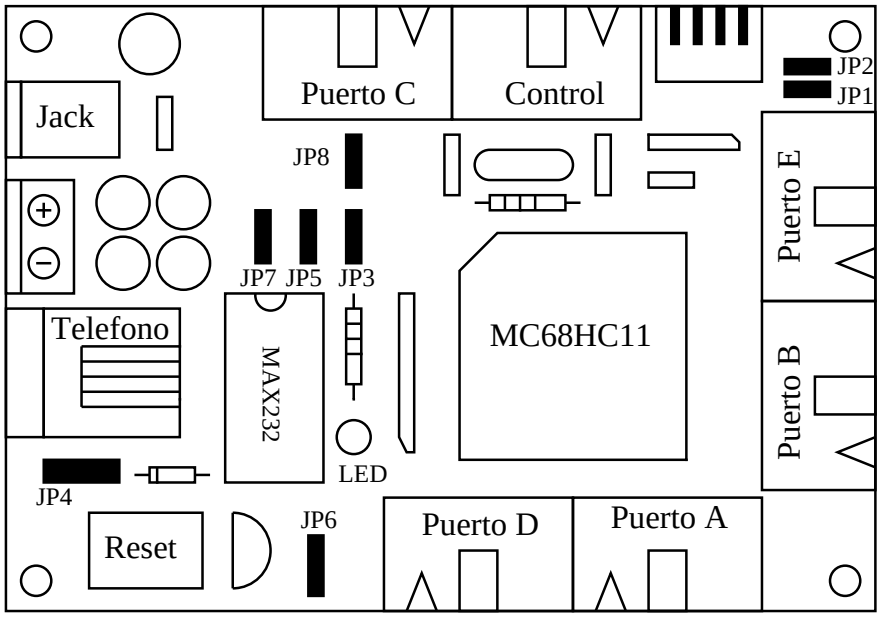


Figura 4.4: Componentes de la CT6811.

reproducirlas. Dicho programa se comunica permanentemente con la CT6811 para que ésta actúe de puente conectando el PC a la red de control, permitiendo así que las tramas las controle el PC en lugar del programa maestro de la CT6811.

En relación a los buses de salida hay que recordar que la red de control se basa en el bus SPI del 68hc11. Por lo tanto para su realización se tendrán que sacar al exterior los pines MISO, MOSI, SCK y SS, que se encuentran en el conector llamado Puerto D. Además de los pines del Puerto D, hay muchos otros que permitirán ampliar la red, añadir sensores, o conectar un PC

Por último se ha mencionado el tamaño porque hay otras tarjetas electrónicas de mayor tamaño que hacen bastante difícil su integración en la estructura mecánica del robot. Las reducidas dimensiones se deben sobre todo a no utilizar un módulo extra con memoria externa, optando en lugar de esto por emplear un modelo de microcontrolador con mayor memoria interna del tipo EEPROM, como puede ser el MC68HC811E2 o el MC68HC711E9.

4.2.1 Configuración final de la CT6811

La CT6811 en este proyecto estará configurada como tarjeta autónoma, es decir al introducir la alimentación ejecutará un programa grabado en la EEPROM interna, que será el que controle al resto de módulos. Originalmente la tarjeta viene con el modelo de microcontrolador 68hc11A1 que tiene 512 bytes de EEPROM situados en la posición $B600. Esto implica que para hacer lo anterior hay que arrancar la placa en BootStrap y tener el jumper JP5 conectado (ver figura 4.4), con lo que las comunicaciones serie asíncronas se pierden, y con ello la posibilidad de conectar el PC. Por eso se ha optado por sustituir este micro por otro modelo, en concreto el 68hc811E2, que tiene 2 Kbytes de EEPROM situadas desde la dirección $F800 hasta la $FFFF. Ahora se podrá arrancar la tarjeta en SingleChip, es decir con el switch 2 arriba y los otros abajo, ya que se utilizará el vector de interrupción de reset

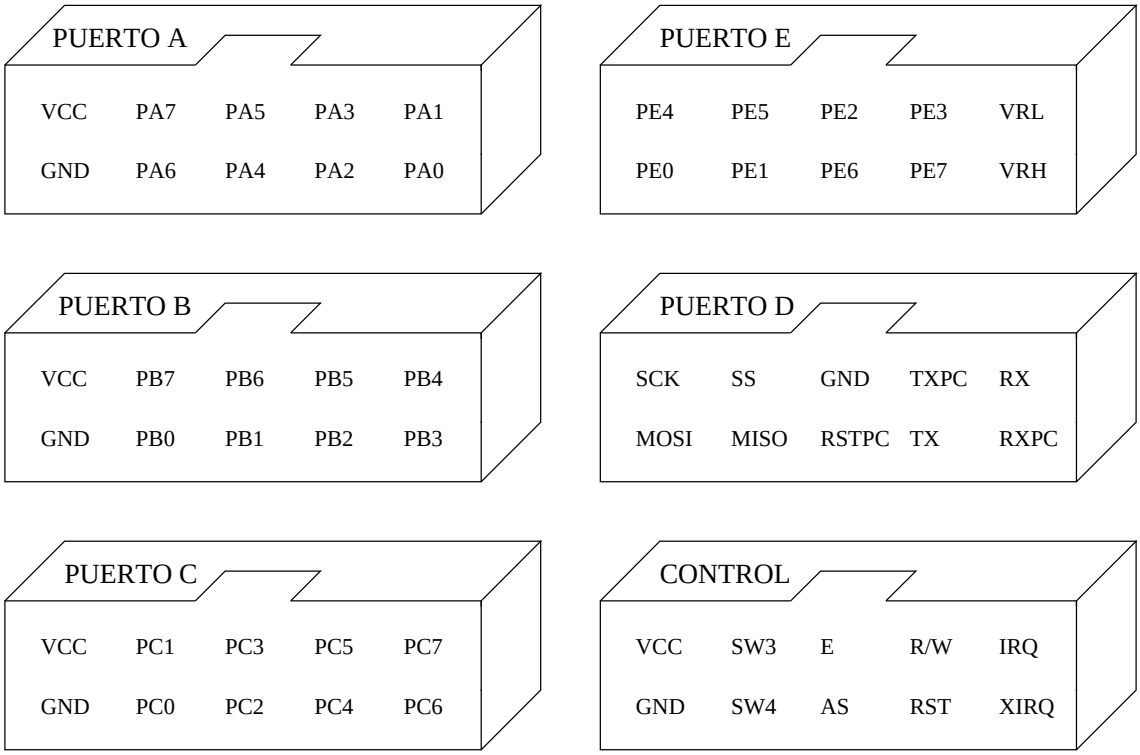


Figura 4.5: Buses de salida de la CT6811.

situado en la dirección $FFFE para indicar al microcontrolador el comienzo del programa a utilizar. Además el jumper JP5 estará quitado y las comunicaciones serie asíncronas en funcionamiento, por lo tanto el PC podrá comunicarse con la CT6811. En resumen, al cambiar el modelo de micro se consigue más memoria y conservar en modo autónomo las comunicaciones con el PC.

Después de esta pequeña aclaración se procede a enumerar la situación de cada jumper y switch, junto con una breve descripción del motivo. Más sobre información la tarjeta se puede encontrar en el manual de usuario de la CT6811 que se encuentra en el apéndice B del proyecto.

1. El conector de teléfono se usará para conectar el PC al robot. La comunicación que se establecerá entre ambos será del tipo serie asíncrona (SCI). El puerto serie del PC cumple la norma RS-232 que no es compatible con la TTL del microcontrolador, por lo que la CT6811 incluye un circuito integrado (MAX232) que actúa de interfaz entre ambas.

2. El Puerto D se usará para conectar la CT6811 a la red de control. En dicho puerto hay conexiones con los dispositivos: SCI del micro (TTL), SCI del PC (RS232) y con el SPI. Será este último el que se utilice para acceder a la red, por lo que de todos los pines del puerto, sólo se utilizarán MISO, MOSI, SS y SCK (ver figura 4.5). El significado de cada uno se explicará al final del capítulo.

3. El jumper JP6 estará quitado para inhabilitar el LVI, circuito que produce un *reset* en el microcontrolador cuando la tensión de alimentación cae por debajo de los 4.5

voltios. En este tipo de proyectos en los que hay varios motores es bastante frecuente que estos induzcan ruido en la línea de alimentación de la electrónica y produzcan con ello caídas de tensión por debajo del valor anterior. Por lo tanto, para evitar los *reset* esporádicos se quitará dicho jumper. Experiencia que se ha probado y ha dado buenos resultados.

4. El jumper JP5 estará también quitado. Como se comentó antes, para arrancar el programa no será necesario utilizar el modo *Bootstrap* junto con JP5. Bastará con configurar la CT6811 en *Single Chip* y disponer evidentemente de un microcontrolador modelo E2. Si no se hubiera optado por esta solución, al colocar dicho jumper, las comunicaciones SCI se perderían y con ellas la posibilidad de conectar el PC

5. Para configurar la tarjeta en *Single Chip* hay que poner el switch 2 arriba y el resto abajo.

6. La alimentación será de 6 voltios y se utilizará la clema de conexión situada entre el conector de teléfono y el jack de alimentación. La polaridad es la siguiente: Por Vcc se introducirá la tensión positiva y por GND la negativa.

7. EL jumper JP3 estará puesto, de manera que actuando el bit PA6 del puerto A del microcontrolador se encenderá o apagará el LED de la CT6811. Servirá de ayuda para establecer si el programa maestro está o no funcionando.

8. El jumper JP7 estará situado en la posición OFF para anular la función del reset software. Esto se hace ya que el robot en modo autónomo no estará conectado al PC, estando todas las líneas de conexión entre ambos al aire. De todas ellas, hay una que por motivos de ruido puede hacer un *reset* en el microcontrolador. Para evitar este efecto no deseado, se colocará el jumper JP7 en la posición OFF.

9. Los jumpers JP1 y JP2 estarán puestos para que automáticamente se establezcan los niveles de referencia del conversor analógico digital en VRH=Vcc y VRL=GND.

10. El jumper JP8 sólo se quitará cuando se quiera programar un microcontrolador de la familia E9. Dado que no es el caso en este proyecto siempre estará puesto.

11. El jumper JP4 estará situado en la posición RST, de forma que al pulsar el pulsador de la CT6811 se produzca un *reset* en el microcontrolador. Su otra posición no se suele utilizar salvo raras excepciones.

4.3 BT6811: Módulo esclavo

El módulo esclavo (ver figura 4.6) se va a encargar de escuchar la información que viaja por la red de control y de capturar las tramas que vayan dirigidas a él. Éstas las enviará el módulo maestro y contendrán información de la posición y estado de los servomecanismos. Además la BT6811 generará constantemente las señales de PWM necesarias para mover los cuatro servos asociados a ella.

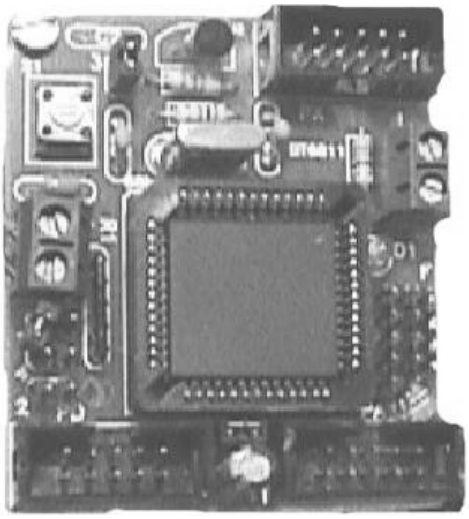


Figura 4.6: Foto del módulo esclavo o BT6811.

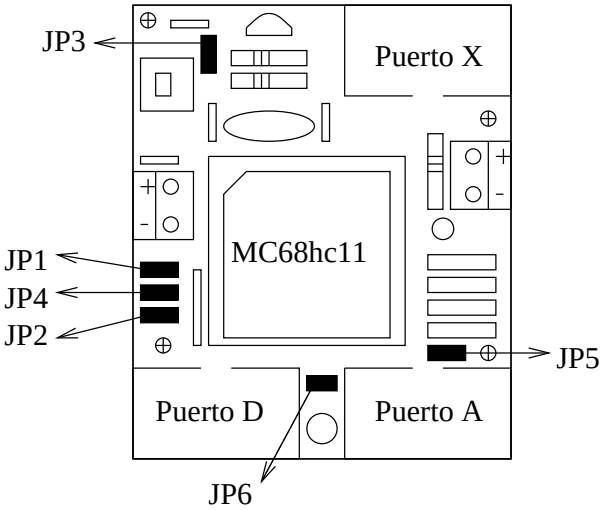


Figura 4.7: Componentes de la BT6811.

Para la realización del módulo se va a partir del conocimiento adquirido en el desarrollo de la CT6811[3], sobre todo para facilitar el diseño hardware y la programación posterior, factores determinantes en la puesta a punto del módulo. Realmente esta tarjeta es una simplificación de la CT6811, partiendo de su esquemático y eliminando todo aquellos elementos que no eran necesarios se ha llegado a una tarjeta con las siguientes características:

1. El módulo esclavo va a recibir toda la información de la red de control, por lo que no será necesario disponer del adaptador del PC. Se eliminan así dos componentes que ocupaban bastante espacio: el conector telefónico y el MAX232 junto con su electrónica asociada.

[3]La CT6811 es una tarjeta desarrollada por el autor y otros compañeros en el año 1996. Actualmente se encuentra disponible como hardware abierto en la WEB de Microbótica (www.microbotica.es). Esto significa que tanto su esquemático como manuales están publicados para todo el mundo.

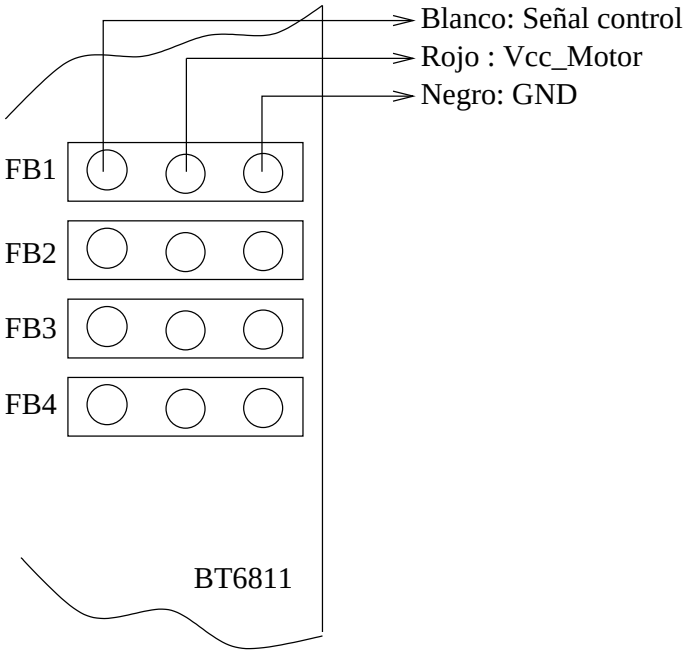


Figura 4.8: Colocación del cable de los servomecanismos en la BT6811.

2. En la BT6811 se han eliminado los switches de control y se han reconfigurado los jumpers. Ahora el *JP1* selecciona el modo de funcionamiento del microcontrolador, el *JP2* se colocará para obligar al modo *bootstrap* a ejecutar el programa de la EEPROM, el *JP3* activa o desactiva el LVI, el *JP4* permite sacar por el puerto D la alimentación de la tarjeta, el *JP5* permite que los motores se alimenten con la misma tensión de la electrónica y el *JP6* activa o desactiva la señal de control *SS*.

3. Se ha dejado el conector del Puerto A para poder seguir utilizando los capturadores y comparadores del microcontrolador, además viene muy bien para conectar otros módulos existentes en el mercado[4].

4. Se han incluido unos pines para poder conectar directamente los servomotores a la BT6811 sin necesidad de ninguna conexión adicional. La conexión se realiza mediante un cable triple que trae el servomecanismo de fabrica y cuyo sentido es el mostrado en la figura 4.8. La señal de control de los servos se conecta a través de los pines con el bus B del microcontrolador, y en concreto; el bit PB0 controla la salida FB1, el bit PB1 la salida FB2, el bit PB2 la salida FB3 y por último PB3 la salida FB4.

5. Para reducir el tamaño de la BT6811 se han eliminado los conectores de expansión que menos se usaban en la CT6811. Los puertos C y E del micro son muy útiles para conectar periféricos, debido sobre todo a que el primero es bidireccional y el segundo permite entradas analógicas. Por eso se ha decidido conservarlos pero utilizando un

[4]Microbótica S.L. dispone además de una tarjeta periférica que se conecta por medio del Puerto A a la CT6811 y a la BT6811. Esta tarjeta incorpora un driver de potencia para manejar dos motores de CC y unos circuitos de polarización para leer sensores. Se puede encontrar más información de la misma en la web de Microbótica [10].

PUERTO X	Puerto C (bidireccional D)	Puerto E (entada D/A)
PX0	PC0	PE3
PX1	PC1	PE7
PX2	PC2	PE2
PX3	PC3	PE6
PX4	PC4	PE5
PX5	PC5	PE1
PX6	PC6	PE4
PX7	PC7	PE0

Tabla 4.1: Correspondencia de bits en el Puerto X.

solo conector en lugar de dos. Es decir, ahora el puerto C y el puerto E del microcontrolador comparten el mismo conector de salida llamado: Puerto X, cuya disposición y equivalencia de pines se muestra en la tabla 4.1.

6. Se ha eliminado la posibilidad de usar el pulsador de reset como entrada de interrupción y el LED de pruebas ahora se ha conectado al bus B del microcontrolador, en concreto al bit PB4.

En el apéndice A se encuentra el manual de usuario de la BT6811, en él se describe todo lo anterior con mucho más detalle. También se explica que al no tener la BT6811 posibilidad de conectarse directamente al PC, para grabar el microcontrolador habrá que optar por una de las siguientes dos opciones:

- Sacar el microcontrolador del zócalo y grabarlo con un grabador o insertándolo en una CT6811. La ventaja es que es fácil para hacerlo un par de veces, pero para muchas ocasiones no es recomendable.
- Utilizar un módulo exterior que realiza la conversión de niveles. Esta opción parece más compleja pero se puede utilizar como adaptador una CT6811 sin el microcontrolador. Al final no hará falta sacar y meter el micro de la BT6811 y a la larga será más confortable. Esta última opción es la que se utilizado en este proyecto.

De los tres modos de funcionamiento que disponía la CT6811, la BT6811 sólo utiliza uno, en concreto el *modo autónomo*. Para disponer cualquiera de los otros dos, *modo entrenador* y *modo periférico inteligente*, se necesita añadir externamente el adaptador para el PC.

4.3.1 Configuración final de la BT6811.

La BT6811 sólo funciona en modo autónomo, lo que implica que, o bien se utiliza el truco del *bootstrap*, o se utiliza un micro 68hc11E2 en modo *singlechip* [5]. En el robot se ha optado por la segunda opción ya que permite almacenar programas más grandes y además mantiene las comunicaciones serie, aunque éstas no serán utilizadas.

Una vez fijado el modelo de microcontrolador se procede a enumerar la configuración de cada uno de los jumpers:

[5] En el apéndice A se explican con detalle ambos métodos.

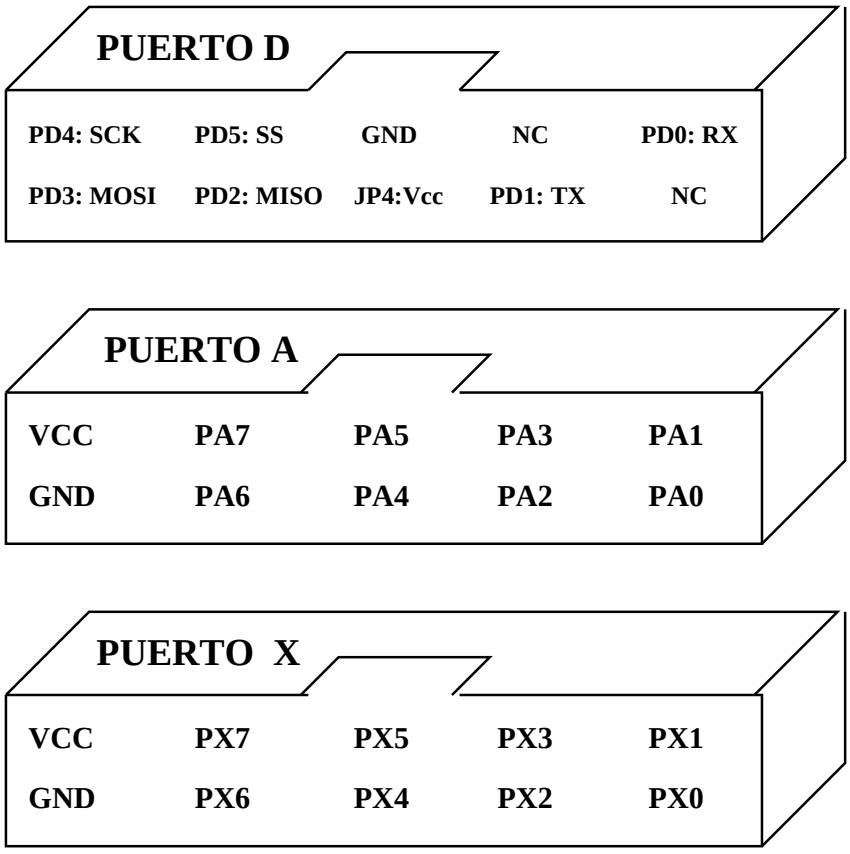


Figura 4.9: Buses de salida de la BT6811.

1. El Puerto D se usará para conectar la BT6811 a la red de control. En dicho puerto hay conexiones con los dispositivos: SCI del micro (TTL), SCI del PC (RS232) y con el SPI. Será este último el que se utilice para acceder a la red, por lo que de todos los pines del puerto, sólo se utilizarán MISO, MOSI, SS y SCK (ver figura 4.9). El significado de cada uno se explicará al final del capítulo

2. El jumper JP1 (#B/S) estará quitado para seleccionar el modo de funcionamiento *singlechip*.

3. El jumper JP2 (Go EEPROM) estará quitado. Este jumper tiene significado cuando el JP1 está conectado, es decir cuando la BT6811 esta configurada en modo *bootstrap*, como no es el caso no se utilizará.

4. El jumper JP3 estará quitado para inhabilitar el LVI, circuito que produce un *reset* en el microcontrolador cuando la tensión de alimentación cae por debajo de los 4.5 voltios. En este tipo de proyectos en los que hay varios motores es bastante frecuente que estos induzcan ruido en la línea de alimentación de la electrónica y produzcan con ello caídas de tensión por debajo del valor anterior. Por lo tanto, para evitar los *reset* aleatorios se quitará dicho jumper.

5. El jumper JP4 (EXP) estará quitado. Su función es permitir sacar por el puerto D la alimentación TTL, pero debido a la existencia de una CT6811 en la red de control tendrá que estar quitado, ya que la CT6811 no puede recibir ninguna señal de alimentación por dicho puerto.

6. El jumper JP5 (Vcc ON) estará quitado. Su utilidad es permitir llevar la alimentación TTL a los motores, de manera que no sea necesario conectar una fuente externa (batería, pila, etc) para poder moverlos. Debido a la cantidad de motores que se van a conectar y a la potencia eléctrica que estos van a consumir es necesario añadir la fuente externa, y por lo tanto no será necesario utilizar la propia de la electrónica (TTL).

7. El jumper JP6 (SS) estará puesto. La naturaleza del protocolo que se va a implementar sobre la red de control hace que sea necesario que este jumper este puesto, de manera que la señal SS (Slave Select) del maestro (CT6811) estará conectada a todas los esclavos (BT6811).

8. La alimentación TTL, es decir la de la electrónica, será de 6 voltios y se utilizará la clema de conexión situada entre el pulsador de reset y el jumper JP1. Por '+' se introducirá la tensión positiva.

9. La alimentación de los motores será de 6 voltios y se utilizará la clema de conexión situada entre el conector PX y los conectores de los motores. Por '+' se introducirá la tensión positiva y por '-' la negativa.

4.4 Descripción física de la red

Ya se han visto los diferentes módulos que forman la red y alguna de las características de la misma. Pero todavía queda por explicar su realización física. Partiendo de la figura

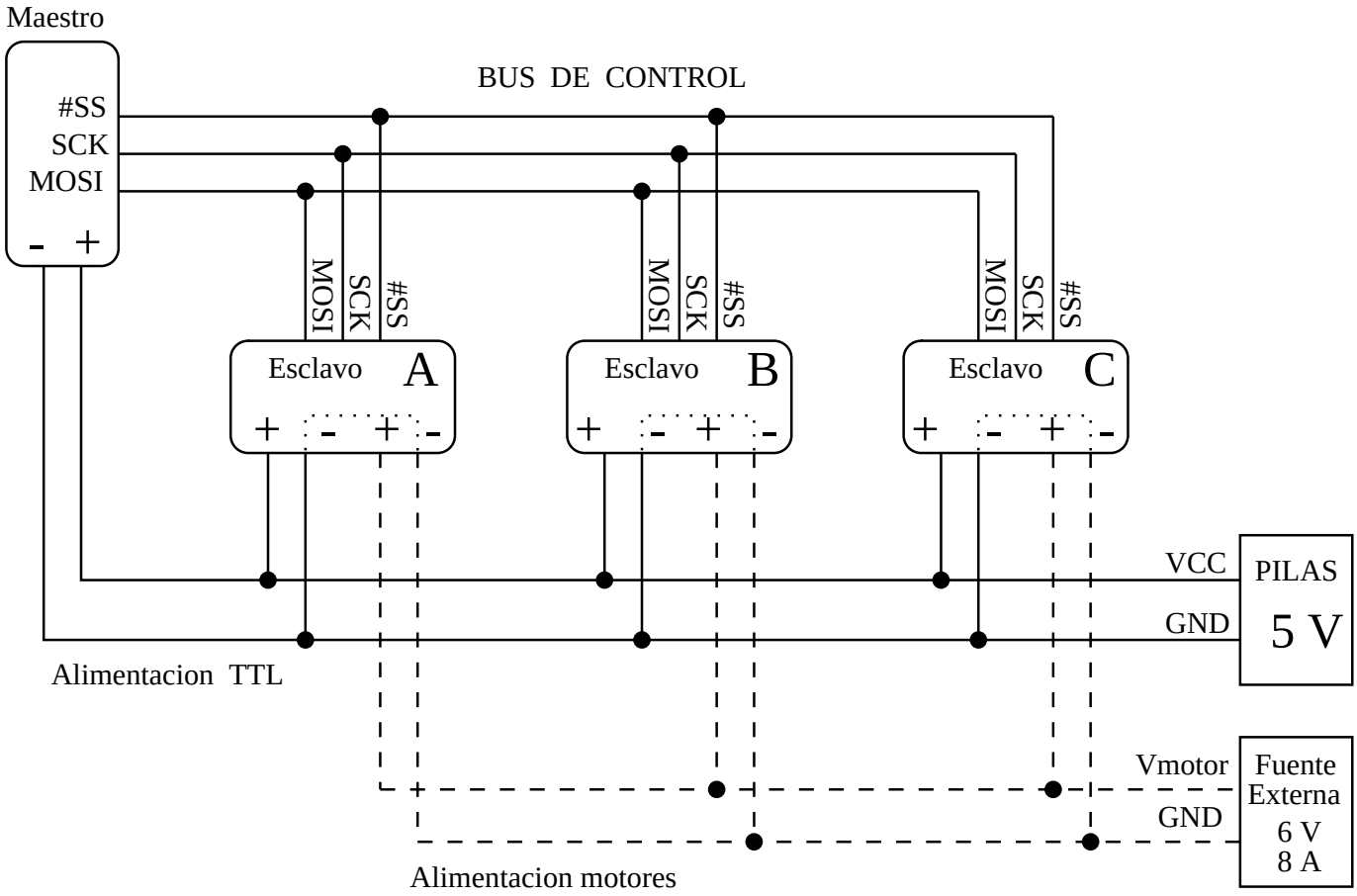


Figura 4.10: Diagrama de conexión de la red de control.

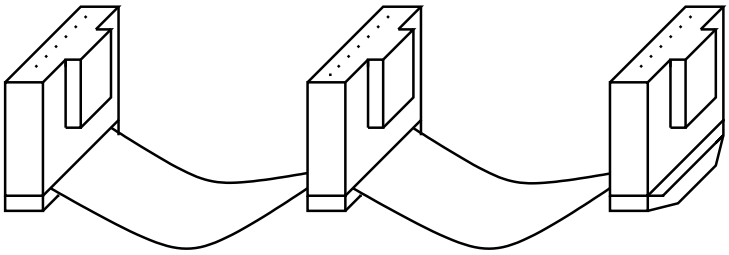


Figura 4.11: Cable tipo bus utilizado para conectar los distintos módulos.

4.10 que representa el diagrama de conexión se estudiarán para empezar los tres grupos de señales de control:

1. El primero, *BUS DE CONTROL*, conecta mediante tres líneas el módulo maestro (CT6811) con los tres módulos esclavos (BT6811). La línea *#SS*, es la encargada de avisar a los esclavos del inicio de una comunicación por parte del maestro. La línea *MOSI* transporta los datos que salen del maestro y llegan a todos los esclavos. Por último la línea *SCK* transporta la señal de reloj o sincronismo.

2. El segundo, *Alimentación TTL*, es el encargado de conectar la fuente de alimentación de 5 voltios con las entradas de tensión de los diferentes módulos. En la CT6811 ésta se introduce por la única clema que hay y en la BT6811 por la clema V_TTL (ver figura A.4).

3. El tercero, *Alimentación motores*, se encarga de conectar la fuente de potencia (5 voltios, 8 amperios) con las entradas de tensión para los servomecanismos, únicamente presentes en los módulos esclavos. La conexión se realizará por la clema indicada como V_Motor (ver figura A.4).

Para conectar las diferentes líneas del *bus de control* se utilizará un cable tipo bus, representado en la figura 4.11 y que une pin a pin los conectores Puerto D de todos los módulos. Al hacer esto, no sólo se estarán conectando las tres señales que interesan (MOSI, #SS y SCK), sino también todas las demás. Esto no plantea ningún problema, al contrario permitirá que la red siga siendo válida en futuras mejoras. Por ejemplo, en esta red la información sólo viaja por la línea MOSI del maestro a los esclavos, pero en el Puerto D existe otra que permite el retorno de la información, es decir de los esclavos al maestro. Aunque en la red preliminar no se ha contemplado ésta por no ser utilizada, realmente está presente, ya que al unir los módulos con el cable de bus se habrá realizado la conexión física.

Los siguientes buses se implementan con un par de cables cada uno, que se irán pasando de módulo en módulo conectando las clemas entre sí según corresponda. La razón de no utilizar una única alimentación para todo el robot ha sido para ofrecer una mayor inmunidad contra el ruido. Los servomotores consumen una potencia variable en función del esfuerzo mecánico que se les exige, esa variación puede producir caídas de tensión bastante importantes, que podrían hacer que la electrónica se desprogramase.

Si en lugar de utilizar la CT6811 como módulo maestro se utilizase otra tarjeta se podría introducir el *bus de alimentación* TTL por el *bus de control*. El motivo es que los módulos esclavos están preparados para recibir la tensión TTL por el puerto D, pero la CT6811 no.

También hay que decir que el módulo esclavo cortocircuita las líneas GND de los buses de alimentación, de esta forma se establece una única referencia de masa.

En la figura 4.12 se ha representado un croquis de la estructura final del robot con la localización de cada tarjeta electrónica. Además se han numerado los servomotores y se han asociado en tres grupos de cuatro unidades, cada uno de los cuales se ha hecho corresponder con cada una de las tres tarjetas esclavas (BT6811). La correspondencia es sencilla; cada servo lleva asociado un código formado por una letra y un número. La letra identifica la tarjeta esclava y el número indica el número de entrada a la que ha sido conectado. Así por ejemplo el servomotor *SERVO C2* se ha conectado a la entrada 2 de la tarjeta *BT6811 C*. Al final se ha obtenido lo siguiente:

1. La tarjeta esclava *BT6811 A* tiene conectados cuatro servomecanismos, tres de los cuales forman la pata delantera derecha y el cuarto forma la tercera articulación de la pata trasera derecha. Para recordar: El servomotor A1 se conecta a la primera entrada de la BT6811 A (motor 1) y forma la primera articulación de la pata delantera derecha, el servomotor A2 se conecta a la segunda entrada (motor 2) y forma la segunda articulación, el servomotor A3 se conecta a la tercera entrada (motor 3) y forma la tercera articulación. Finalmente el servomotor A4 se conecta a la cuarta entrada (motor 4) y forma la tercera articulación de la pata trasera derecha.

2. La tarjeta esclava *BT6811 B* vuelve a tener cuatro servomecanismos conectados, tres de los cuales forman la pata delantera izquierda y el cuarto forma la segunda articulación de la pata trasera derecha. La asociación es como la expresada antes.

3. Por último la tarjeta esclava BT6811 C tiene tres servomotores que forman la pata trasera izquierda y un cuarto que forma la primera articulación de la pata trasera derecha.

Al final la red está compuesta por tres tarjetas esclavas y una maestra, entre todas ellas controlan los doce motores del robot. Aunque no se ha implementado, se ha considerado ampliar el sistema con cuatro servomecanismos más, dos de ellos se utilizarían para poner al robot una cola móvil y los otros dos para hacer lo mismo con la cabeza. Dado como se ha concebido la red y la estructura, añadir un módulo esclavo para controlar esos cuatro nuevos motores es una tarea inmediata, tan sólo habrá que preocuparse de ampliar el software, cuestión que se aborda en el siguiente capítulo.

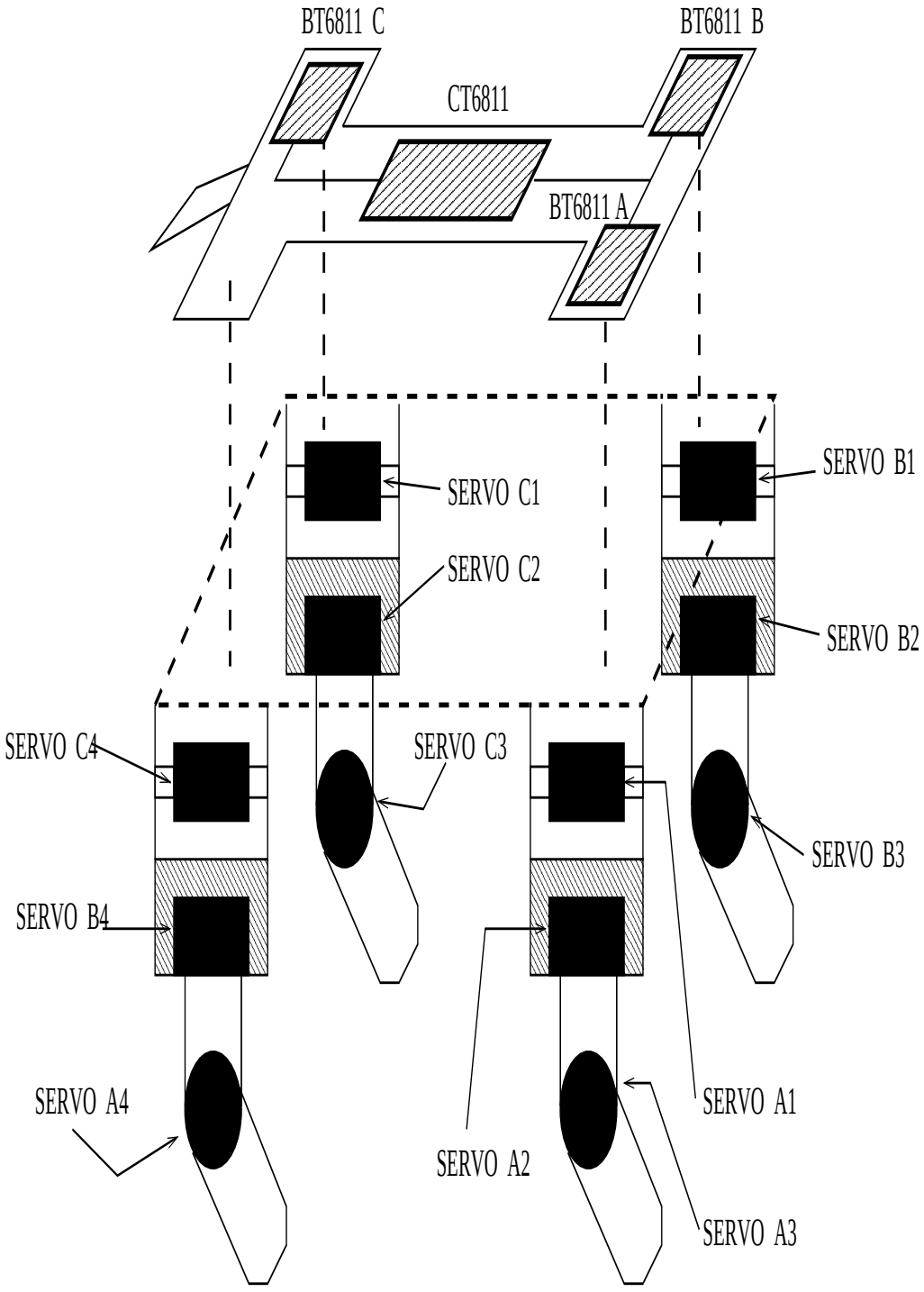


Figura 4.12: Diagrama de conexión de los motores.

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.

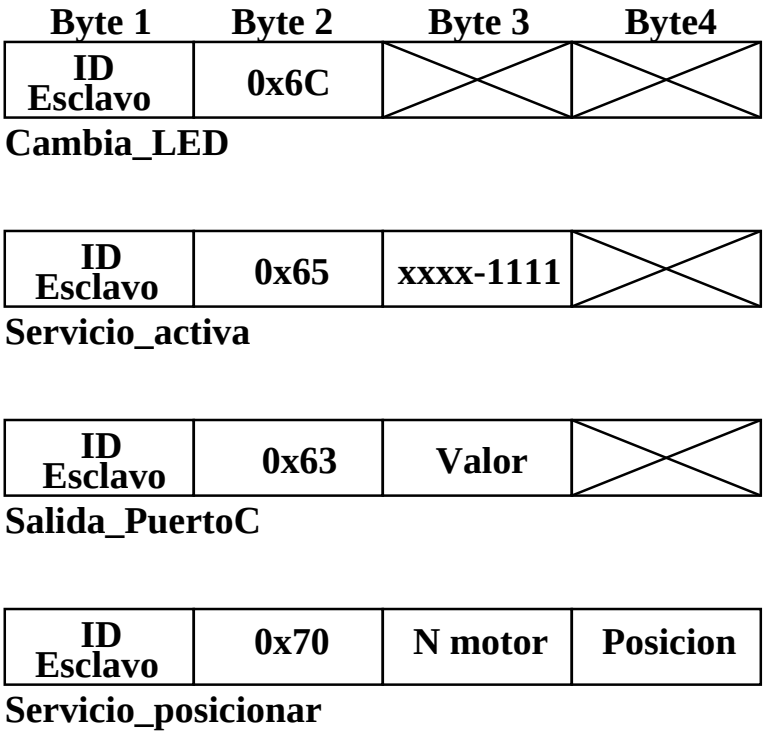


Figura 5.1: Tramas de control.

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

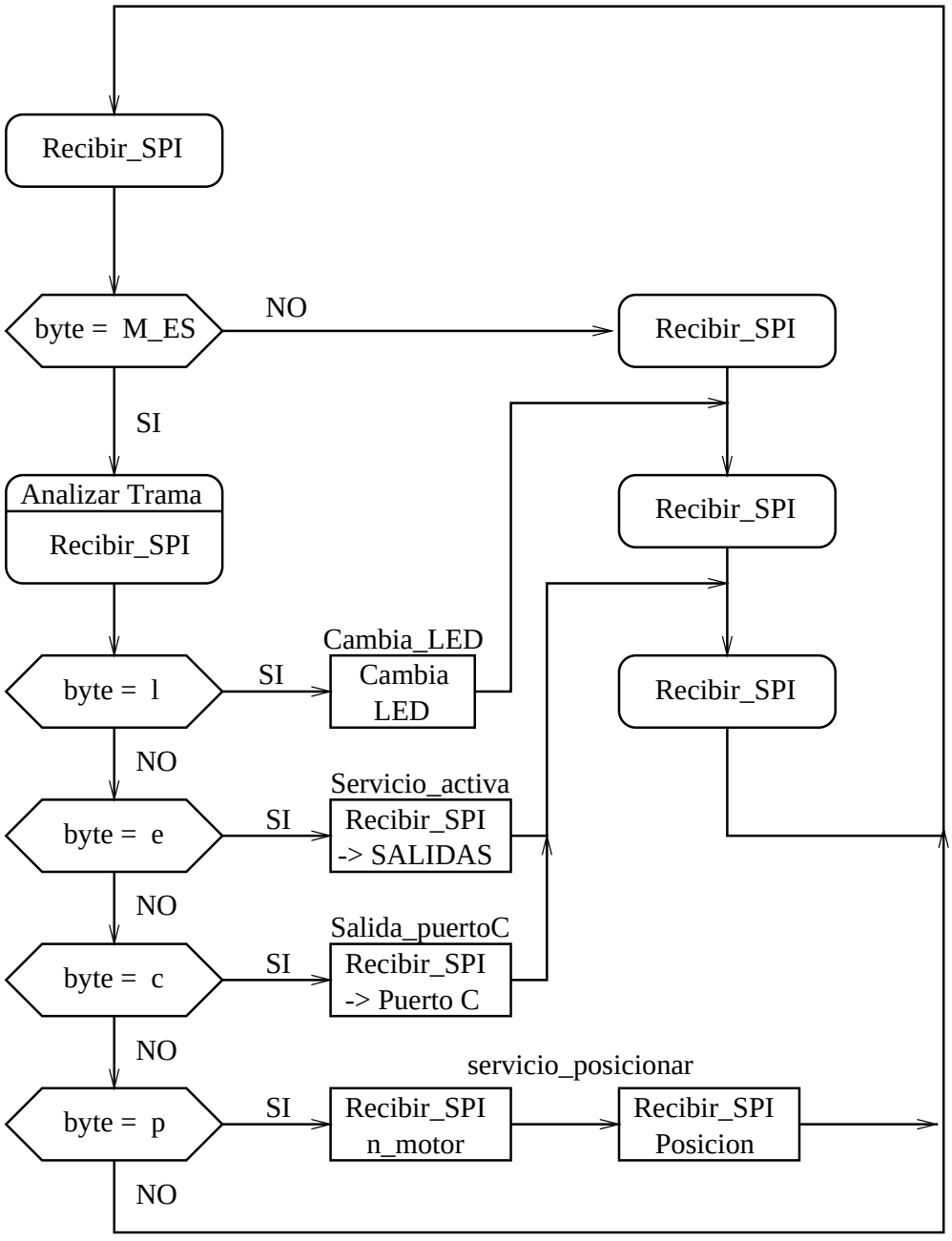


Figura 5.2: Diagrama de recepción de tramas en el módulo esclavo.

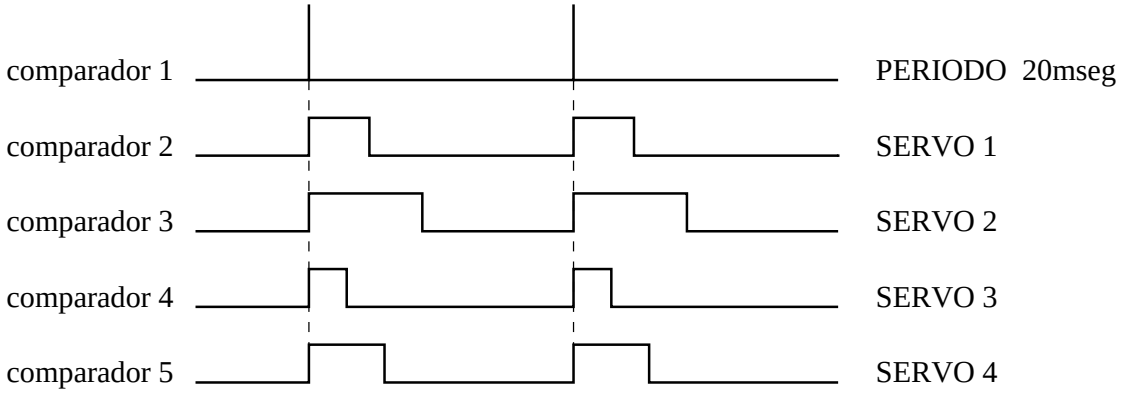


Figura 5.3: Señales de PWM de los cuatro servos.

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-

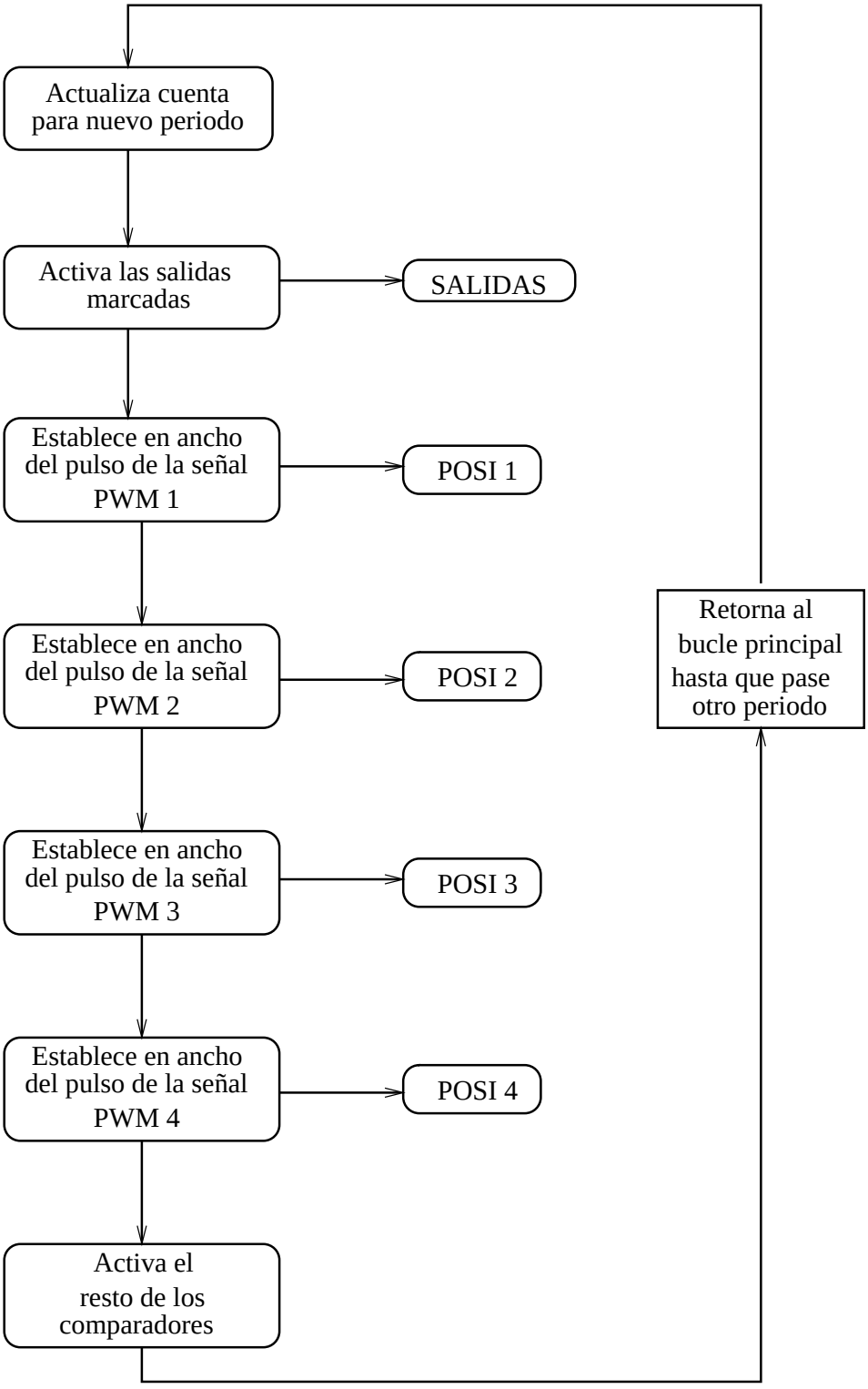


Figura 5.4: Rutina de interrupción del comparador 1.

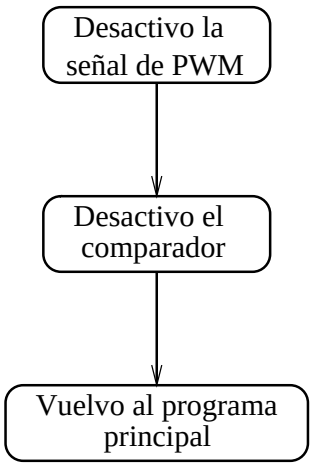


Figura 5.5: Rutina de interrupción de los comparadores 2, 3, 4 y 5.

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

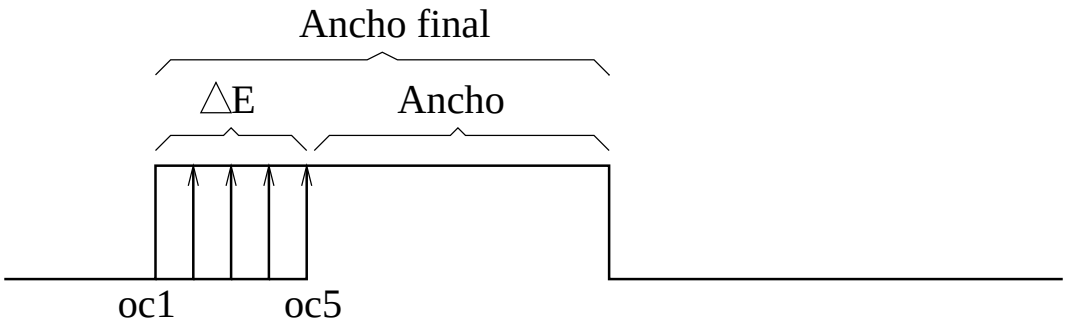


Figura 5.6: Primer error en el PWM debido a la activación anticipada de las salidas.

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.

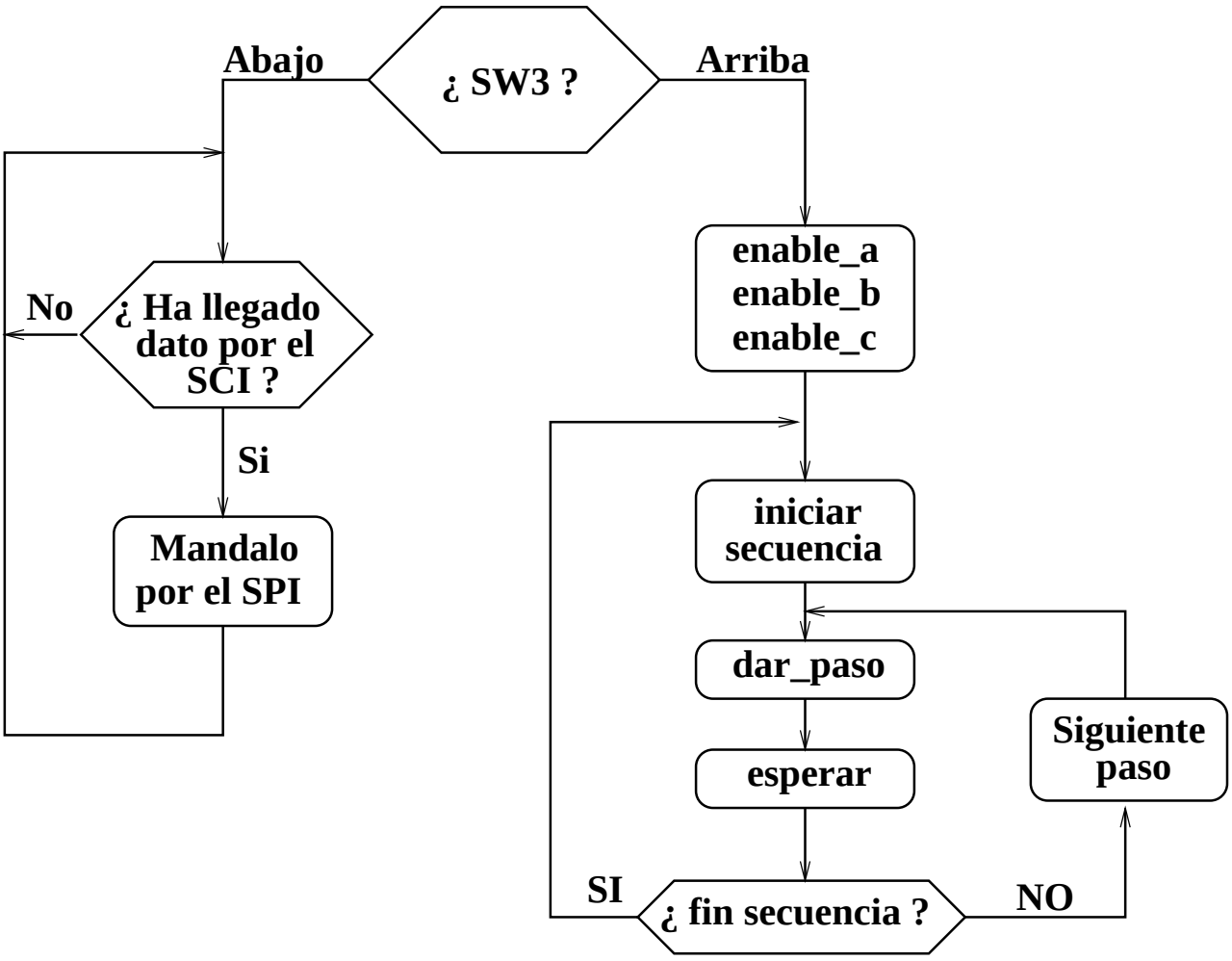


Figura 5.7: Diagrama del programa maestro.

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 1.8^o. 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.

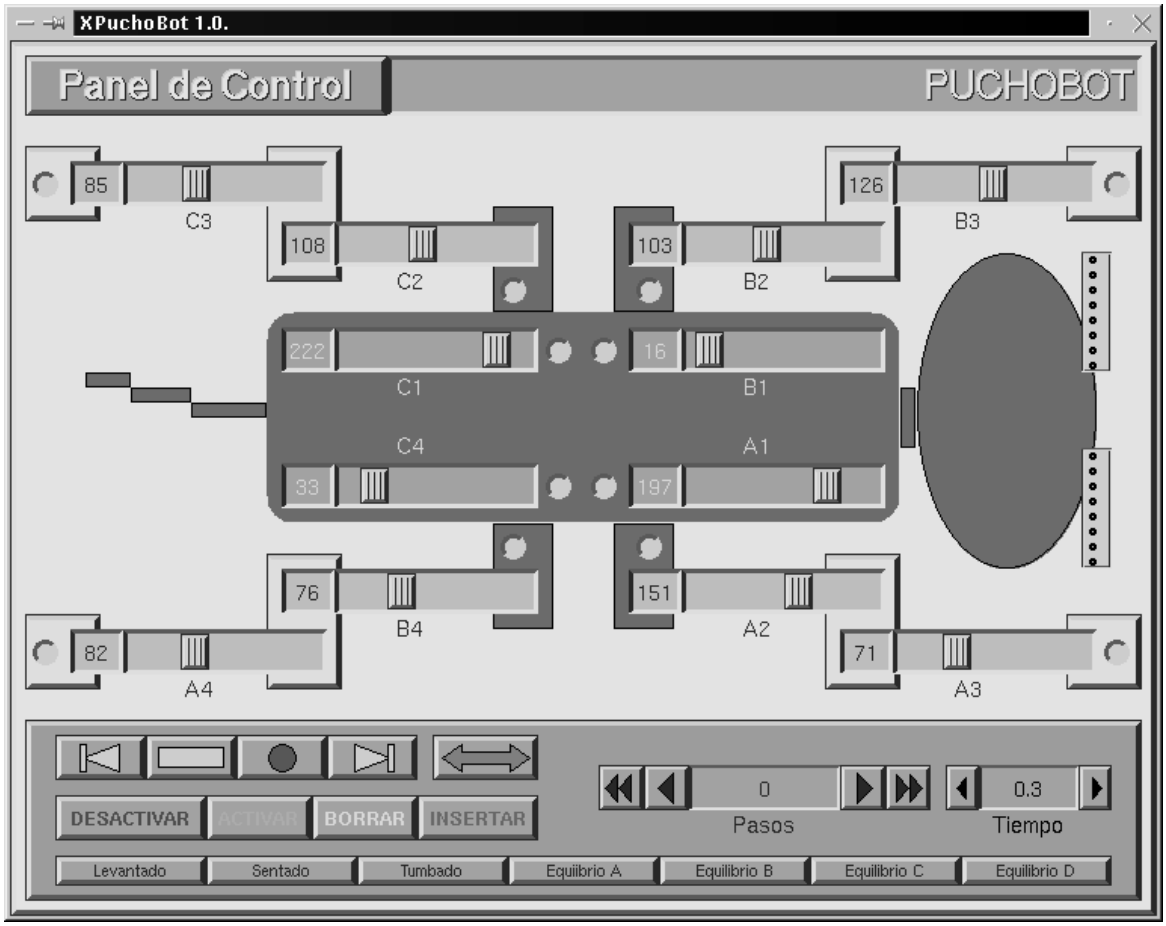


Figura 5.8: Interfaz gráfica del programa generador de secuencias: XPucho.

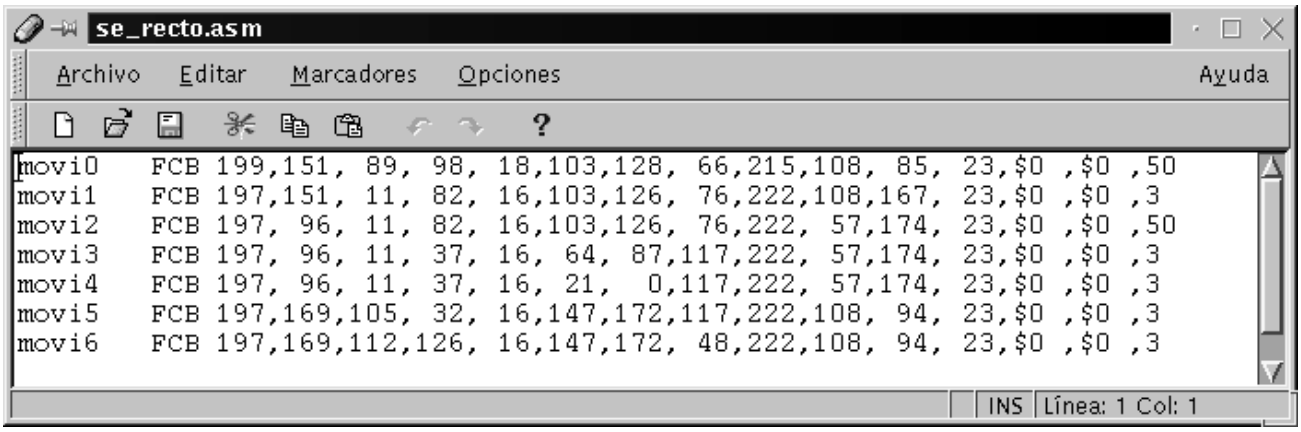


Figura 5.9: Fichero obtenido con la función *Convertir.*

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.

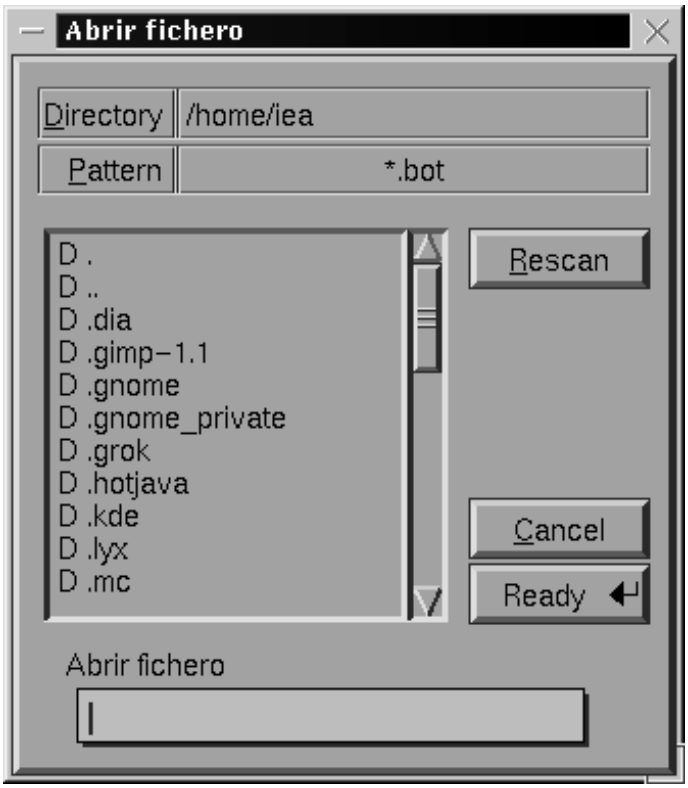


Figura 5.11: Pantalla de selección de ficheros para abrir, guardar o convertir.

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

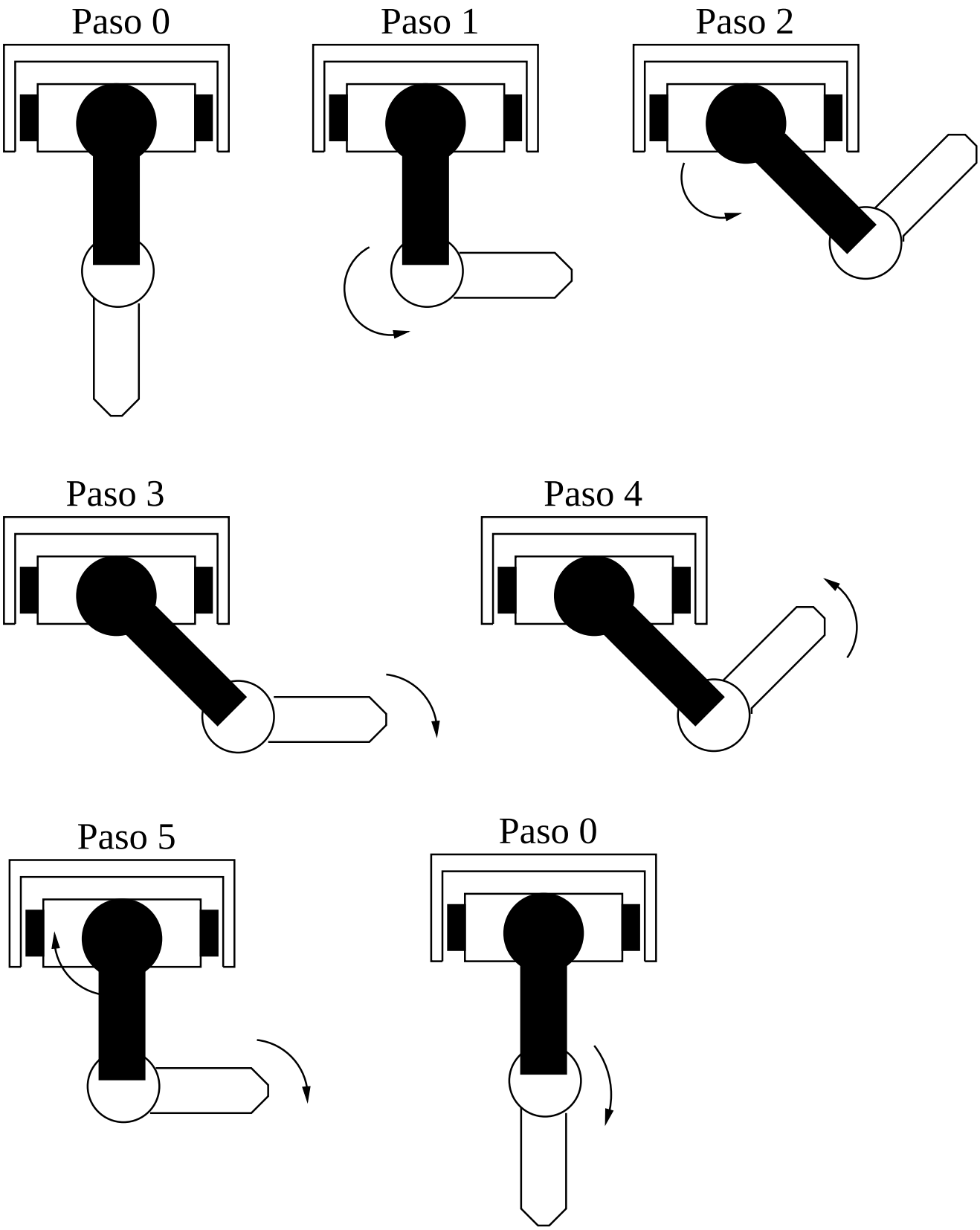


Figura 5.13: Diagrama de los pasos para configurar la secuencia *saludo*.

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.

Capítulo 6

Conclusiones y líneas futuras

6.1 Conclusiones y reflexión del autor

Al principio del proyecto, durante su planificación (página 9), se determinaron los objetivos básicos que debía cumplir el robot. Haciendo memoria, fueron los siguientes:

1. Desarrollar una plataforma móvil articulada que incluya ocho motores.
2. La estructura mecánica tiene que apoyarse en cuatro extremidades.
3. Hay que ser capaces de controlar la posición de los ejes de los motores.
4. El sistema electrónico de control tiene que ser modular y basado en la familia de microcontroladores de Motorola 68hc11 o 68302.
5. El robot tiene que ser capaz de realizar los movimientos básicos, es decir ir recto, girar y hacer algún gesto gracioso.

Repasando todo lo realizado hasta ahora, se puede comprobar cómo el resultado final ha superado con creces las expectativas iniciales. Si al principio del proyecto se planteaba la construcción de un cuadrúpedo que tuviese una funcionalidad mínima, al final se ha conseguido que el robot sea mucho mejor de lo pensado originalmente. Para lograr esto, ha sido necesario introducir algunos cambios en las especificaciones originales, cambios que a continuación se van a describir, indicando qué mejoras se han conseguido con ellos:

1. Al principio se planificó usar ocho motores para realizar los movimientos básicos, pero después de una primera versión se optó por ampliar el número a doce y así permitir muchos más movimientos. Por ejemplo, mientras que con la anterior estructura se giraba sin control, mediante la nueva se puede controlar el radio de giro, es decir se ha conseguido un movimiento mucho más general. Pero no sólo se ha producido esa mejora en lo concerniente a la mecánica, sino que los propios motores se han cambiado por otros mejores. En concreto seis servomecanismos Futaba 3003 se han sustituído por el modelo Futaba 9404. Estos ofrecen más par y velocidad de movimiento, aunque como todo lo bueno, cuestan más que los anteriores. Pero aún así, merece la pena, sobre todo cuando se aprecia que el robot tiene más fuerza y se mueve con mayor soltura que en su primera versión. Además, la sustitución, ha permitido implementar acciones como levantarse, arrastrarse y sentarse.

2. En el planteamiento original, toda esta mecánica estaba acompañada de una electrónica modular que permitía posicionar los ejes de los motores en los ángulos correctos, haciendo que el robot se moviera adecuadamente. Dicha electrónica estaba formada por dos módulos esclavos, que se encargaban de mover los motores, y por un módulo maestro que controlaba a los anteriores. Se planteó que los módulos esclavos estuvieran formados por una tarjeta electrónica, basada en el microcontrolador MC68hc11 de Motorola, y diseñada a medida para este proyecto. Por el contrario, el módulo maestro estaría basado en una tarjeta ya disponible, también basada en el MC68hc11, y sólo en el caso de que no sirviera se procedería a la construcción de otra. Al final, en este apartado no ha habido muchos cambios, todo lo contrario, la planificación inicial ha permitido cumplir los objetivos propuestos, incluso superarlos.

   El diseño del módulo esclavo ha sido un éxito: se ha conseguido tener en un espacio muy reducido una electrónica capaz de mover cuatro servomecanismos. Además, el programa desarrollado a medida para dicho módulo permite que únicamente indicando la posición en donde se quiere situar el eje del motor, la tarjeta se encargue de moverlo y mantenerlo en dicha posición de forma autónoma. La buena realización de los dos puntos anteriores, la tarjeta y el programa, ha permitido que la red de control no esté saturada de información y que el módulo maestro sea muy sencillo de implementar. Esto ha evitado el tener que diseñar otra tarjeta electrónica, ahorrando tiempo y dinero al proyecto. Además, cuando se modificó el robot para ampliarlo a doce motores, no hubo ningún problema por parte de la electrónica, tan sólo hubo que añadir un módulo esclavo más. Si la red de control no se hubiese planteado como modular, lo anterior habría supuesto rehacer una parte importante del sistema. Incluso, y basado en las estimaciones hechas en el capítulo cuatro, la red puede ampliarse hasta 82 módulos esclavos manteniendo la velocidad de movimiento actual.

3. Uno de los grandes retos del proyecto ha sido buscar las secuencias de movimiento del robot. En un principio no se dió mucha importancia a este punto, y se pensó, que una vez construida la estructura y la red de control, no habría problema para buscar las posiciones y las secuencias adecuadas para moverlo correctamente. Pero esto no fue así: en un robot de este tipo, articulado y con doce motores, no es posible intuir las secuencias de movimiento. Por eso, la realización de una herramienta de ayuda software era algo imprescindible. Y así se ha hecho. Utilizando herramientas disponibles en Linux se ha desarrollado un software que permite programar los pasos del robot de una manera ordenada, fotograma a fotograma. Tan grande ha sido la utilidad aportada por esta herramienta, denominada XPucho, que muchas de las mejoras y líneas propuestas para este proyecto se basan en ella.

   El software anterior se completó con otro pequeño programa que permite controlar desde el ordenador al robot. En lugar de ir cargando y ejecutando las secuencias según se quiera que el robot haga una cosa u otra, este programa permite mediante cursores ir ejecutando diferentes movimientos predeterminados. Si bien no estaba dentro de los objetivos iniciales del proyecto, y tampoco sirve como herramienta de trabajo, sí es útil para poder hacer demostraciones. Seguro que será utilizada una vez que se quiera grabar secuencias de movimiento autónomo en el robot.

Figura 6.1: Robot cuadrúpedo de LEGO Mindstorms[11].

4. También se especificó que el robot tenía que ser autónomo, es decir, que se pudiese mover sin necesidad de tener el PC conectado a él o, lo que es lo mismo, que la electrónica interna permitiese generar o reproducir las secuencias de movimiento. Esto no ha sido difícil de implementar, sobre todo porque la herramienta de programación XPUCHO permite convertir una secuencia de movimiento almacenada en el PC, en un archivo compatible con el ensamblador del MC68hc11. Esto significa que, una vez obtenidas las secuencias de movimiento, se pueden ir grabando en el módulo maestro para que sea éste el que las reproduzca sin necesidad de la presencia del PC. En nuestro caso, el robot tiene dos modos de funcionamiento: mediante el primero lo controlamos desde el PC utilizando el software de análisis descrito en el punto tres; en el segundo modo, el robot se comporta de forma autónoma, y en concreto lo que hace es ir caminando hacia adelante.

Por último, durante la elaboración del proyecto se ha podido comprobar cómo algunos de los robots investigados en la etapa de búsqueda de datos, capítulo primero, han evolucionado y se han hecho mucho más conocidos y asequibles. Por ejemplo se mencionó la reciente aparición del perro robot de Sony Aibo, pues bien, en todo este tiempo ha crecido su página WEB y han surgido en Internet bastantes foros al respecto. Pero tal vez la evolución más espectacular ha sido el Lego MindStorm, que si bien al inicio de este proyecto no estaba muy extendido, se puede considerar como uno de los grandes aciertos de Lego. Ha sorprendido incluso a los mismos creadores, que a pesar de enfocarlo, al principio, a un público joven, rápidamente se dieron cuenta que eran los profesionales los que lo estaban utilizando en mayor cantidad. Ejemplos de este incremento se encuentran en la página oficial de LEGO Mindstorms [11] o en otras muchas fácilmente localizables[1], como por ejemplo *http://member.nifty.ne.jp/mindstorms.* En la foto 6.1 se puede ver uno de los kits ofrecidos por esta compañía.

Todo este auge en la robótica 'casera', por distinguirla de la industrial, hace que los contenidos del proyecto sean novedosos, sobre todo para aquellas personas que se quie-

[1]En mi caso he utilizado el buscador *google* y como palabra de búsqueda *mindstroms.*

ran introducir en el mundillo de los robots articulados. Además, como el robot se ha construido y ha funcionado superando las espectativas iniciales, el proyecto adquiere un valor añadido en lo que a experiencia se refiere. Lo presentado aquí no es un estudio teórico de robótica, sino un manual práctico de cómo hacer un robot cuadrúpedo. Por eso, y siguiendo la línea de otros proyectos realizados por el autor, este trabajo se publicará en la página WEB de Microbótica S.L. [10], esperando que sirva de base a otros muchos proyectos que se encuadren en el marco de la robótica, y aunque tal vez no se utilice la estructura de un cuadrúpedo, tal vez sí se pueda usar la red de control o algunas de las características mecánicas mencionadas en el capítulo 2.

Para terminar doy las gracias a todas las personas que me han ayudado en la realización del proyecto, ya sea por los consejos técnicos que me han dado como por la ayuda moral que me ha permitido seguir adelante en los momentos de flaqueza.

6.2 Líneas futuras de desarrollo

Hasta obtener el último prototipo del robot ha sido necesario realizar muchas modificaciones, la mayor parte de ellas impuestas por la imposibilidad de realizar ciertos movimientos considerados como imprescindibles. Pero aún habiendo solventado todos ellos, todavía quedan mejoras por hacer, las cuales se han organizado en tres grupos:

1. El primero es el de las mejoras mecánicas, en él se exponen las mejoras o cambios relacionadas con la morfología del robot. En este punto es importante tener los pies en el suelo, al hablar de mejoras se puede divagar mucho, sobre todo después de ver robots como el de SONY (ver figura 1.7, página 12), pero hay que ser conscientes de nuestras posibilidades y de nuestros recursos. Por ese motivo las mejoras expuestas son las más fáciles o cercanas al autor del proyecto.

2. El segundo grupo de mejoras hace referencia al software. Al igual que antes se pueden plantear muchas, pero la ventaja es que ahora no depende tanto de la tecnología sino del tiempo de que uno quiera dedicarle.

3. Por último hay una serie de mejoras que según la opinión del autor son las más importantes. Se trata de todo aquello relacionado con el entorno del robot. Es decir la posibilidad de que el robot pueda recibir información del entorno que le rodea, o lo que es lo mismo, dotarle de sensores.

Con estas propuestas se espera conseguir motivar a otras personas para continuar con el proyecto, y quién sabe, al final tal vez tengamos la mascota artificial que suele salir en las películas de ciencia ficción y que nos hace soñar con vivir el futuro.

Mejoras mecánicas

1. Se podrían cambiar los motores ya que los utilizados al final han sido más caros y más pesados que los previstos inicialmente. Esto hace que, considerando el factor coste, al empezar a diseñar un nuevo robot uno pueda buscar motores entre una gama más amplia. La consecuencia de esto son cambios en la morfología del robot

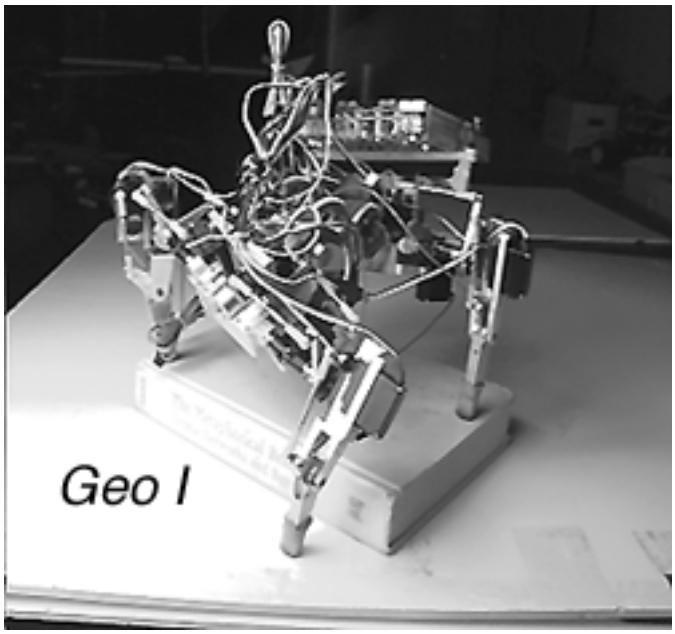


Figura 6.2: Foto del robot GEO I.

ya que no todos los motores tienen la forma de los servomecanismos. El peso del motor es difícil reducirlo ya que los servomecanismos utilizan materiales ligeros tipo plástico. En este aspecto se pueden considerar optimizados sobre todo si se tiene en cuenta que su diseño es específico para modelos aeronáuticos.

2. Otra mejora sería utilizar materiales mixtos para realizar la estructura del robot, de esa forma se puede reducir el coste, el peso y el tiempo de desarrollo final. Es más fácil cortar el plástico que el aluminio y por otro lado es más resistente el aluminio plegado que el plástico pegado. En el perro se ha utilizado aluminio, pero se podrían haber hecho piezas de plástico (la madera no se recomienda). Por ejemplo, las articulaciones del robot, al ser piezas simples que incluso no necesitan doblarse, podrían haber sido de plástico. Por el contrario el hombro es una pieza difícil de obtener en plástico, sobre todo por la cantidad de pliegues que hay que hacer y la resistencia que debe tener, por eso no se recomienda tocarla.

3. Por último otra mejora previsible sería cambiar la morfología del robot pero esta vez para adaptar o mejorar los movimientos del mismo. Por ejemplo se puede plantear una estructura que tenga todos los motores en el cuerpo y que mediante engranajes transmita el movimiento hasta las articulaciones de las patas. En la foto 6.2 se puede ver un ejemplo.

Mejoras software

1. Los programas que se han desarrollado son los básicos para poder probar el robot y ver que todo funciona. Mediante ellos se pueden buscar las secuencias de movimiento, reproducirlas y mandarlas al robot, pero aún así, con todo esto, todavía quedan cosas por hacer. Lo más destacable es que ahora mismo no se puede recibir infor-

mación del entorno, por lo que el PC no sabe si el robot realmente se ha movido o no. Es decir, no se ha implementado la recepción de datos provenientes del robot, el control del robot se hace en bucle abierto. Por lo tanto una mejora sería adaptar los programas para que el PC pueda recibir datos enviados desde el robot.

2. En línea con lo anterior, se plantea la siguiente mejora: Cuando se programó el módulo maestro para que enviase las tramas por la red de control, hubo que introducir pausas para dar tiempo a los motores a llegar a la posición indicada. Pues bien, una mejora muy buena es que sea el propio módulo esclavo el que gestione este tiempo, de manera que cuando se cumpla avise al módulo maestro para que éste, si quiere, le envíe nuevas tramas. La forma de gestionar esos tiempos por parte de los módulos esclavos puede realizarse de dos formas:

   (a) Mediante espera activa. Se calcula un promedio y ese es el tiempo que se espera en todos los casos. Método parecido al que hay ahora.

   (b) Leyendo la posición del eje del motor o lo que es lo mismo, cerrando el bucle. De esta forma el módulo esclavo tendrá conocimiento de cuándo ha llegado el motor a su posición y dará luz verde a la recepción de una nueva trama para el mismo. Este método es mejor que el anterior pero más complejo de realizar.

3. Otra mejora reside en la forma de programar el robot. Ahora la programación del mismo es visual mediante barras deslizadoras, pero se podría mejorar si se utiliza un modelo virtual del robot. En la figura 6.3 se puede ver un ejemplo aplicado a un brazo. Con ese modelo la programación sería más fácil ya que no se necesitaría tener conectado el robot para ver la posición final de los motores. Incluso se podría hacer, en caso de tener ganas y paciencia, un modelado físico del mismo para ver si sería capaz de moverse o no, sin necesidad de probar con el prototipo real. Esto último es algo que se ha venido haciendo frecuentemente en la robótica, aunque por un motivo diferente, la no existencia del robot real. Lo que aquí se propone es lo contrario, mueve el robot virtual, facilita la programación y el estudio mediante el PC, y asegura en todo momento que lo que el modelo virtual hace también lo puede hacer el real.

4. Todavía más avanzada sería una aplicación para programar el robot gestualmente, mediante la cual las secuencias de movimiento se buscarían directamente moviendo el robot real con las manos. El procedimiento sería colocar manualmente el robot en la posición deseada y luego dar la orden al PC de leer el estado de los motores del robot, almacenándose así dicha información como si fuese un fotograma. Finalmente, otra vez se movería manualmente el robot hasta colocarlo en la siguiente posición. Se volvería a grabar en el ordenador y así sucesivamente hasta completar la secuencia deseada de movimiento. La ventaja de este método es que contiene a los demás, es decir, se puede tener el robot virtual a la vez que el real, y se puede usar cualquiera de los dos para buscar las secuencias de movimiento. Esta idea no es nueva, en Jurasic Park[2] se usó para animar a los dinosaurios. Al principio sólo se tenían

[2]Jurasic Park: Película de cine del año 1993 dirigida por Steven Spielberg y escrita por Michael Crichton. En ella se cuenta como unos científicos clonan dinosaurios para crear un parque temático. En la fase de

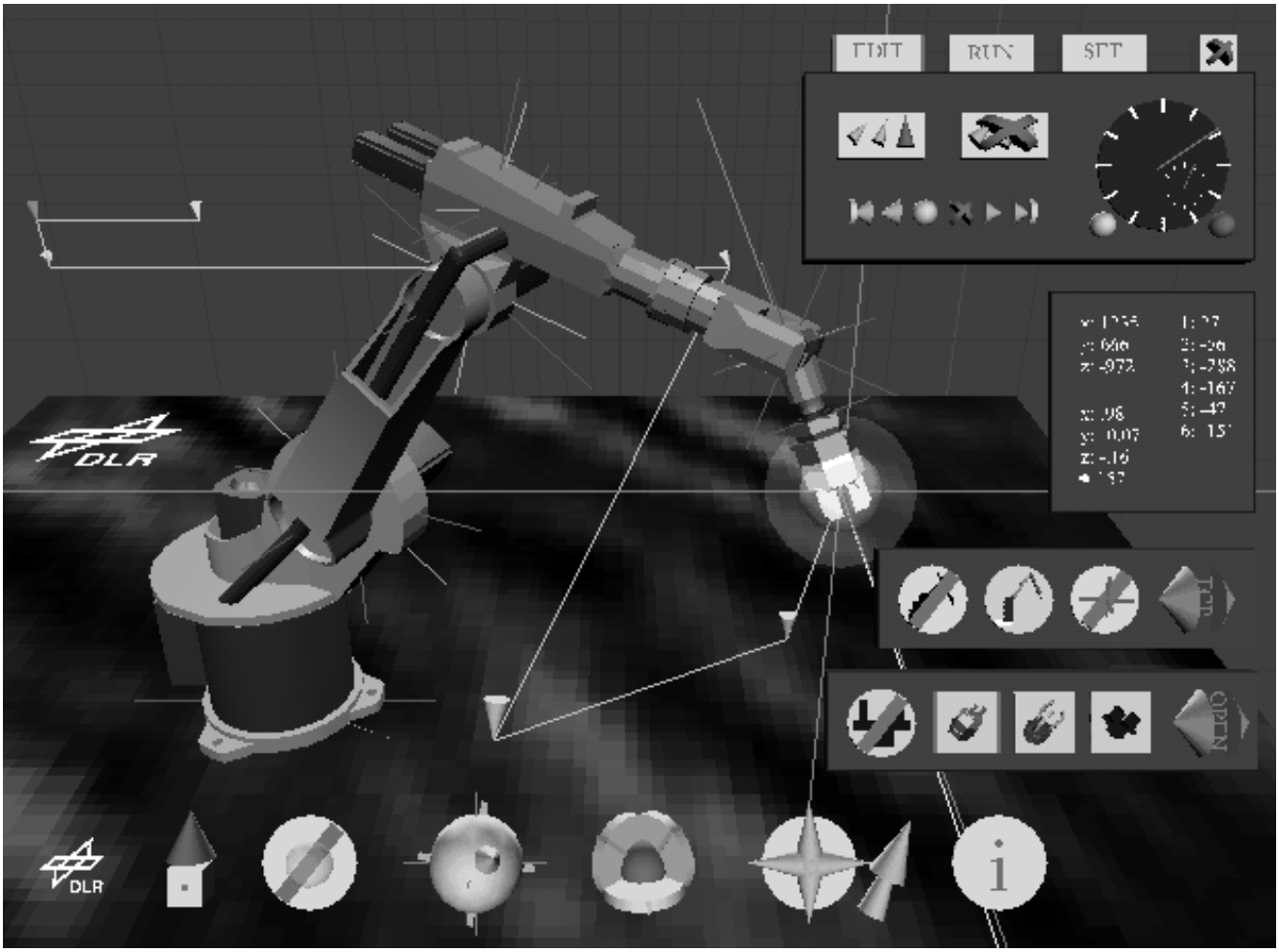


Figura 6.3: Ejemplo de un programador virtual de movimientos.

modelos virtuales en el ordenador pero pronto fue necesario utilizar el modelo robot. La razón de ello estaba en que los equipos de animadores no eran capaces de crear las secuencias y movimientos correctos sin usar la forma tradicional del cine. Es decir, utilizando maquetas y grabando fotograma a fotograma. Tal vez esta mejora sea una de las más avanzadas, y no sólo por el software, sino también porque en los motores o en las articulaciones hay que añadir algún sensor que nos informe del ángulo en el que se encuentra el eje del motor. En nuestro caso esto se ve facilitado mucho por los motores y la electrónica utilizados. Los motores ya tienen incorporado dicho sensor, por lo que tan sólo hace falta soldar un cable y sacarlo al exterior para tener la lectura del ángulo. Y la electrónica está a su vez preparada para leer la información anterior, que es analógica, y transformarla a digital. Una vez en ese estado se puede transmitir al PC. En resumen, con poco esfuerzo hardware pero mucho de programación se puede diseñar un prototipo de robot autónomo con programación gestual.

pruebas el parque sufre un sabotaje y los dinosaurios quedan en libertad, empezando una trama de acción que no para hasta el final. Lo destacable de esta película fue la innovación en efectos especiales al crear todos los dinosaurios con modelos virtuales en el ordenador.

Interactuando con el entorno

Hasta ahora, todas las mejoras que se han propuesto, tratan de perfeccionar el robot, ya sea mejorando el movimiento o facilitando la búsqueda de secuencias. Ninguna de las mejoras anteriores ha considerado la relación del robot con el entorno. Es decir, una vez que se dispone de una plataforma que anda, gira y es fácilmente programable, queda por establecer la utilidad de la misma. La respuesta es difícil, ya que, tal y como está el robot ahora no dispone de sensores que capten información del entorno. Sin estos, no es posible realizar aplicaciones avanzadas y por eso una mejora imprescindible sería conectarle toda una batería de sensores. Mediante ellos se pueden obtener datos como temperatura ambiental, intensidad de luz, distancia a los objetos, imágenes y un sin fin de medidas más. Una vez que se cuente con alguno de ellos es más fácil pensar en una utilidad para el robot. Por ejemplo, mediante células fotoeléctricas se podría guiar al robot en función de la intensidad de luz recibida. Mediante ultrasonidos podría detectar obstáculos y evitar tropezarse o chocarse contra la pared. Más avanzado sería colocarle una pequeña cámara para poder detectar objetos, algo parecido a lo que hace AIBO, que localiza una pelota de ping-pong y le pega una patada.

Aún así, con el tema de los sensores hay que tener mucho cuidado y analizar bien las situaciones. Si se coloca una cámara alguien tendrá que procesar la información. Si lo hace el propio robot tendrá que tener una electrónica más potente, si lo hace el PC la señal de vídeo habrá que transmitirla por radio o tener conectado el robot al PC mediante un cable.

Independientemente de todo, en cuanto se ponga un sensor, por muy simple que sea habrá que pensar en modificar el software para permitir comunicaciones de ida y vuelta. Esto no es difícil, toda la electrónica esta preparada para ello, incluso la CT6811 y la BT6811 están preparadas para conectarle directamente algunos tipos de sensores sencillos. Como pueden ser bumpers, CNY70 (infrarrojos), LDR (células fotoeléctricas) y sensores de temperatura.

Apéndice A

Manual de usuario de la BT6811

A.1 Introducción

La BT6811 es una tarjeta microcontroladora basada en el MC68hc11 de Motorola. Su objetivo es controlar de forma autónoma cuatro servomecanismos, permitiendo una conexión sencilla de los mismos y facilitando su alimentación.

Para desarrollar esta placa se ha partido del modelo CT6811, anteriormente desarrollado y se han eliminado los componentes que no se van a utilizar. Al final se ha obtenido una placa con unas dimensiones de 48 mm x 58 mm.

Los elementos que se han eliminado con relación a su antecesora han sido: el dispositivo de comunicaciones MAX232, los puertos B, C, Control, el jack de alimentación, los switches de configuración y algunos jumpers de configuración. Por contra, se han introducido los siguientes elementos: Una clema para alimentar los motores, un jumper para poner la alimentación de los motores interna, cuatro conexiones directas para servomecanismos y un puerto mixto, que saca al exterior el puerto C y el E.

Se ha conseguido rutar la placa a una sola cara, logrando así abaratar los costes de fabricación y permitir que la placa se pueda construir artesanalmente. En el caso de soldar a mano se hará por la cara de atrás y la tarjeta no necesitará taladros metalizados. Una consecuencia de esto, es que se han perdido algunas compatibilidades de puertos con relación a la CT6811, pero esto no es importante debido a la nueva función que ha de desarrollar esta placa. En los capítulos siguientes se irán detallando todas las funciones.

A.2 Modos de funcionamiento

Realmente la BT6811 está pensada para funcionar en modo autónomo, dentro de esta modalidad se pueden distinguir dos tipos de modos.

1. Arrancando en Bootstrap (jumper JP1 puesto): en este caso al alimentar la BT6811 se ejecuta un programa interno del MC6811 que se queda esperando a recibir un programa por el puerto serie. Se puede utilizar esto para volcar programas desde el PC [1], o desde cualquier otro dispositivo serie, como por ejemplo una calculadora

[1] Utilizando un adaptador especial ya que la BT6811 no lleva MAX y por lo tanto no puede recibir señales del PC.

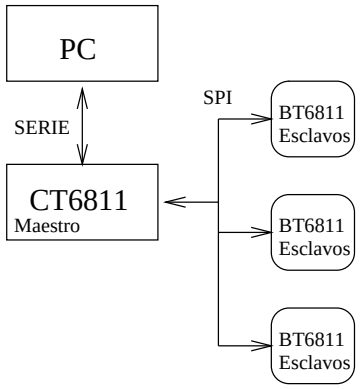


Figura A.1: Red de tarjetas BT6811.

HP. Para pasar al modo autónomo desde el *bootstrap* hay que colocar el jumper JP2. Así, cada vez que se haga un reset se le indica al programa *bootstrap* que ejecute el programa grabado en la EEPROM interna. El inconveniente de este método es que al poner el jumper JP2 se pierden las comunicaciones serie asíncronas (SCI) de la BT6811.

2. Arrancando en Single-Chip (jumper JP1 quitado): en este caso al introducir la alimentación en la tarjeta se ejecuta el programa apuntado por el vector de reset. Debido a que este vector se encuentra en la posición $FFFF, será necesario disponer de un microcontrolador modelo E2 (que tiene 2K de EEPROM situados en el tramo $F800-$FFFF). Al grabar el programa en la EEPROM hay que asegurarse de poner el vector de reset apuntando al principio del programa. Con este método no se coloca el jumper JP2 y no se pierden las comunicaciones serie asíncronas.

En ambos casos, se ha pensado en la posibilidad de montar una red de tarjetas BT6811. Para ello se puede utilizar el SPI o el SCI, cuyos pines están en el PUERTO D. La red estará formada por un conjunto de BT6811 esclavas y un dispositivo maestro, como por ejemplo una CT6811. Con la estructura planteada se puede tener un PC comunicándose con una CT6811 por el puerto serie y ésta a su vez controlando la red de BT6811 por el puerto SPI. Ver figura A.1

Otra necesidad diferente es poder cargar o grabar programas desde el PC en el microcontrolador 6811 de la BT6811 sin tener que quitarlo del zócalo. Es decir usar la BT6811 como entrenadora para ejecutar el "downmcu", el "ctdialog" o el "mcboot". Como la tarjeta no lleva un MAX232 habrá que proporcionarle uno, las alternativas son dos: o se construye un adaptador o se utiliza una tarjeta CT6811 que ya lo lleva incorporado. En ambos casos la conexión del adaptador se hará por el puerto D de la BT6811.

En este manual se hace referencia al segundo caso, es decir a utilizar la CT6811 como interfaz entre el PC y la placa BT6811. Lo que se hará es conectar los puertos D de ambas tarjetas con un cable de bus estándar, quitar el microcontrolador de la CT6811 y alimentar con 5 voltios ambas tarjetas. El PC se conecta mediante el cable telefónico al conector de la CT6811. A partir de este momento se pueden ejecutar todos los programas citados antes. La única diferencia es que ahora no hay "reset software", por lo que habrá que pulsar el reset de la BT6811 de forma manual. Salvo este pequeño detalle todo lo demás se mantiene como en la CT6811. Ver figura A.2

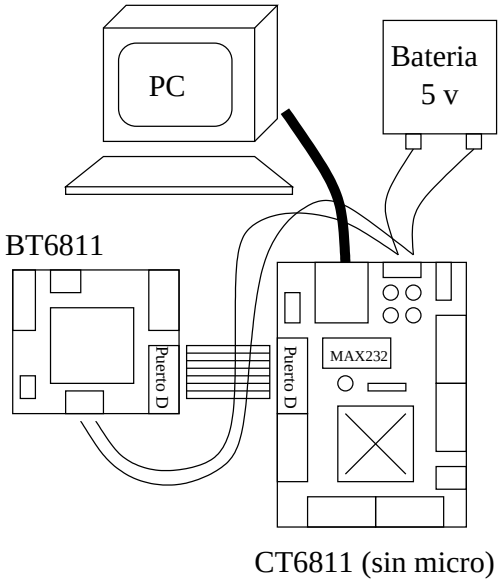


Figura A.2: Esquema de conexión de la BT6811 en modo entrenadora.

Es importante que se quite el micro de la CT6811 para poder cargar los programas en la BT6811. Si no se quita aparecerá un conflicto entre los programas de BOOTSTRAP de cada micro. Esto último se puede generalizar a una red de BT6811, si las tenemos todas conectadas entre sí por un cable de BUS normal y arrancamos todos los micros en BOOTSTRAP para ser cargados desde el adaptador del MAX, resultará que no funcionará nada. Los programas *bootstrap* de cada BT6811 interferirán la línea TX y RX corrompiendo la información. Debido a esto cada BT6811 conectada a una red habrá que reprogramarla aisladamente.

A.3 Jumpers de configuración de la BT6811

La BT6811 tiene seis jumpers de configuración, se procede a describir la función de cada uno de ellos. Ver figuraA.3

1. JP1 (#B/S): Es el encargado de seleccionar el modo de funcionamiento del microcontrolador. Si el jumper esta puesto se configura el modo “bootstrap”, por el contrario si esta quitado se configura el modo “single-chip”. Para poder utilizar éste último será necesario disponer del vector de reset en EEPROM, como ocurre con los modelos de microcontroladores MC68hc11E2. El jumper estará puesto.

2. JP2 (Go EEPROM): Este jumper tiene sentido cuando el JP1 está conectado, es decir cuando la BT6811 esta configurada en modo “bootstrap”. Si se conecta el JP2, cada vez que se pulse “reset” en la BT6811 se ejecutará el programa que esté grabado en la EEPROM interna. Si por el contrario se deja quitado, al recibir el “reset” se ejecutará el programa “bootstrap” y por lo tanto la placa estará funcionando en modo entrenador, esperando a recibir el programa a ejecutar por el puerto serie.

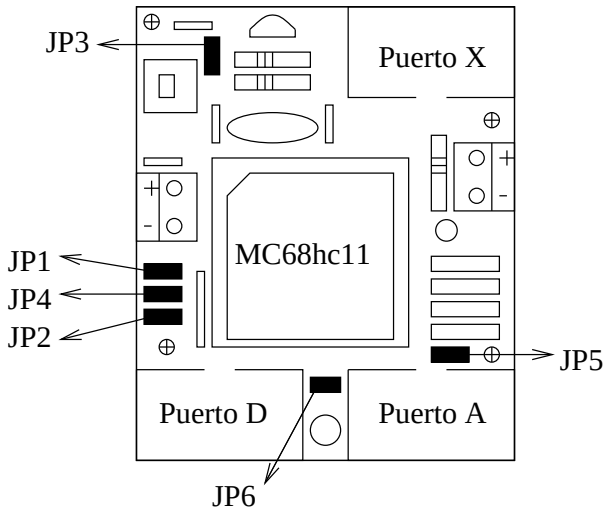


Figura A.3: Disposición de los jumpers en la BT6811.

3. JP3 (LVI): Este jumper esta conectado inicialmente. Su función es proteger el programa grabado en la EEPROM, haciendo un reset del sistema cada vez que la alimentación baje de 4,5v. Por eso si se utilizan aplicaciones que inducen ruido eléctrico en la alimentación, será aconsejable quitar el jumper. Por ejemplo, es el caso de utilizar la placa para mover motores estando configurada para alimentarlos internamente.

4. JP4 (EXP): Su función es permitir sacar por el puerto D la alimentación TTL. Esto es muy útil a la hora de conectar otros periféricos a la BT6811. Inicialmente está quitado, debido a que el periférico que se suele conectar es una CT6811 y ésta no puede recibir ninguna señal de alimentación por dicho puerto.

5. JP5 (Vcc ON): Su utilidad es permitir llevar la alimentación TTL a los motores, de manera que no sea necesario conectar una fuente externa (batería, pila, etc.) para poder moverlos. Inicialmente esta quitado, por lo que habrá que utilizar la segunda fuente para poder mover los motores.

6. JP6 (SS): ecto está puesto, su utilidad es facilitar el control en las redes de BT6811 mediante el SPI. Al quitar el jumper JP6, la línea SS (Slave Select) que proviene del maestro no se conecta al SS del esclavo. De esta forma podemos hacer que otros pines del micro maestro se puedan conectar al SS del esclavo, pudiendo activar cada esclavo sin interferir en los demás.

A.4 Localización del resto de componentes

Ya se han descrito los jumpers de configuración, ahora se describen el resto de componentes de la BT6811. Ver figura A.4

- LED: Esta conectado al bit_4 del puerto B. Este puerto esta mapeado en la dirección $1004 del microcontrolador MC68hc11. Si se pone a uno dicho bit, entonces se encenderá el LED.

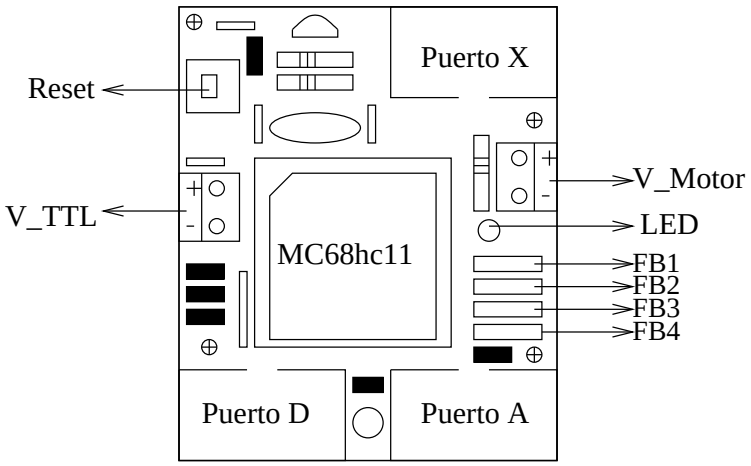


Figura A.4: Resto de componentes de la BT6811.

- Reset: Cada vez que se pulse el "pulsador" se estará haciendo un reset del MC68hc11. El programa en ejecución se parará y volverá a inicializarse el sistema. El modo de arranque será el configurado por medio de los jumpers JP2 y JP1.
- V_TTL: Es la clema por donde se introduce la tensión de alimentación TTL (5v) necesaria para la electrónica.
- V_Motor: Es la clema por donde se introduce la tensión de alimentación de los motores. Si se conecta el jumper JP5 se evita usar esta clema, al permitir que la tensión TTL (la introducida por V_TTL) sea la que alimente los motores.
- Puerto D: Es un bus de expansión, por él van las señales de comunicaciones serie y las del SPI.
- Puerto A: Es un bus de expansión, por él van las señales del puerto A del MC68hc11 junto con V_TTL y GND. Este puerto es totalmente compatible con el de la CT6811.
- Puerto X: Es un bus de expansión mixto. Por él se sacan al exterior las señales del puerto C y del puerto E del MC68hc11. El puerto E es de entrada mientras que el C es bidireccional, hay que tener en cuenta esto para evitar cortocircuitar dos salidas. Por ejemplo si se quiere introducir una señal por el pin PX1, hay que configurar el bit del puerto C correspondiente a PX1 como entrada. Luego se podrá leer el valor bien por el puerto E o por el C. El estado predeterminado del Puerto C y del Puerto E es de entrada, por lo que no habrá ningún conflicto en el momento de arranque del microcontrolador.Ver tabla A.1
- FB1, FB2, FB3, FB4: Son jumpers triples donde se van a conectar los servomecanismos. El FB1 se controla con el bit_0 del puerto B del MC68hc11, el FB2 con el bit_1, el FB3 con el bit_2 y el FB4 con el bit_3.

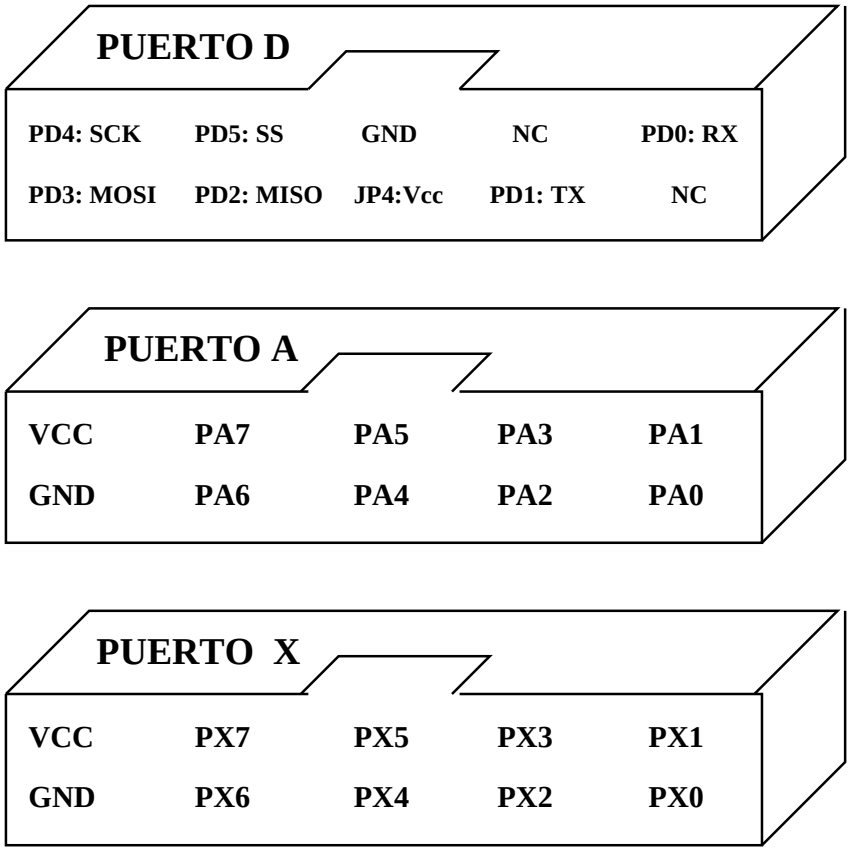
Puerto B	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
$1004	–	–	–	LED	FB4	FB3	FB2	FB1

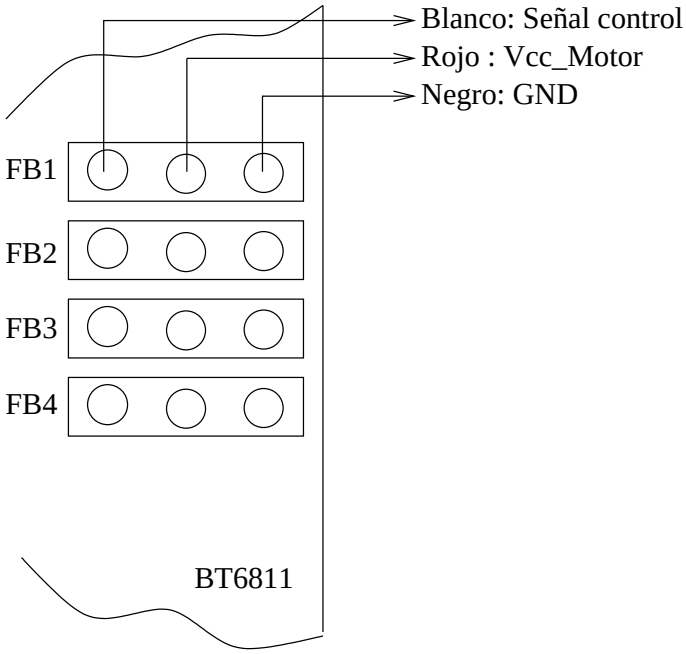
PUERTO X	Puerto C (bidireccional D)	Puerto E (entada D/A)
PX0	PC0	PE3
PX1	PC1	PE7
PX2	PC2	PE2
PX3	PC3	PE6
PX4	PC4	PE5
PX5	PC5	PE1
PX6	PC6	PE4
PX7	PC7	PE0

Tabla A.1: Correspondencia de bits en el Puerto X.

- El resto de componentes son el reloj y sus condensadores asociados, el propio MC68hc11, condensadores de filtrado de alimentación y resistencias de polarización.

A.5 Pines de los buses de expansión





A.6 Utilización de la BT6811

La BT6811 se puede usar como entrenadora, o como producto final (autónomo), tanto en modo aislado como en red. Su diseño está pensado para los modos autónomos y dentro de ellos el de red. Para usarse como entrenadora se debe disponer de un adaptador serie como el max232, éste será externo y permitirá unir la tarjeta con el puerto serie del PC. Como la CT6811 incorpora uno se puede utilizar ésta como adaptador.

En este apartado se van a describir mediante ejemplos todas las formas de uso. Se empezará por el modo entrenador y se terminará con el modo autónomo en red.

A.6.1 BT6811 trabajando como entrenadora

La utilidad de usar la BT6811 como entrenadora radica en la posibilidad de programar la tarjeta sin necesidad de sacar el microcontrolador de su zócalo, detalle importante ya que en algunas ocasiones se tiene un difícil acceso a él. Como ya se ha dicho con anterioridad, para poder usar este modo se necesita un adaptador externo que permita conectar el puerto serie de la BT6811 con el puerto serie del PC. En el mercado existen varios integrados que realizan esta función, uno de ellos es el MAX232, que es el que se ha utilizado en la CT6811. Por esta razón se ha diseñado la BT6811 para poder ser conectada al PC usando la CT6811 como adaptador. La única condición es sacar el microcontrolador del zócalo de la CT6811 y unir ambas tarjetas por el puerto D mediante un cable bus estándar de diez hilos. En la figura A.2 se describe un esquema de conexión.

Para configurar la BT6811 en modo entrenador hay que colocar los jumpers de la siguiente forma:

1. Jumper JP1 puesto para seleccionar el modo Bootstrap.

2. Jumper JP2 quitado para permitir la ejecución del programa Bootstrap.

3. Los jumpers JP3, JP5 y JP6 no influyen en la configuración del modo entrenador.

4. Jumper JP4 desconectado ya que el modo auxiliar de la BT6811 será la CT6811.

Con la estructura y la configuración planteadas la BT6811 se comporta como una CT6811 pero sin la función de reset software. Esto hace que la mayoría de los procedimientos y ejemplos que se describen en el manual de la CT6811 sean válidos para la BT6811. A pesar de esto se describe a continuación cómo enviar programas a la tarjeta y cómo dejarlos autónomos, pero conviene recordar que se pueden obtener ejemplos más prácticos en el manual de la CT6811.

Para descargar un programa desde el PC se pueden utilizar los programas *downmcu*, *ctload* y *mcboot*. En este caso se va a emplear el *downmcu* por ser el más conocido y disponible para las plataformas MsDos y Linux. Ahora hay que mencionar una diferencia fundamental con respecto al proceso de carga con la entrenadora CT6811. En dicha entrenadora existe la función de reset software que permite que el PC pueda *resetear* al microcontrolador justo antes del proceso de carga. En la BT6811 esa función no existe por lo que el proceso de reset tendrá que ser manual. El método para ello es pulsar el botón de reset de la BT6811 e inmediatamente después pulsar el botón de ENTER del PC para ejecutar el programa *downmcu*. Resumiendo:

1. Teclear en la línea de comandos: downmcu *miprograma* -com1

2. Pulsar el botón de reset de la BT6811

3. Pulsar el botón de ENTER del PC para ejecutar el programa *downmcu*.

Miprograma es el nombre del programa con extensión s19, es decir ya compilado, que se quiere enviar a la BT6811. Si la conexión se ha realizado por el puerto serie dos, habrá que poner como parámetro *-com2*.

Otros programas que pueden utilizarse en modo entrenador son el *mcboot* y el *ctdialog*. El primero consiste en un pequeño terminal que manda por el puerto serie todo lo que recibe del teclado y envía a la pantalla todo lo que recibe por el puerto serie. La forma de utilizarlo es la siguiente:

1. Teclear en la línea de comandos: *mcboot*

2. Pulsar F4 para seleccionar el puerto serie adecuado.

3. Finalmente pulsar F1 para cargar programas en la RAM interna. En el proceso de carga habrá que hacer lo mismo que con el *downmcu*, pulsar el reset de la BT6811 y luego pulsar ENTER para enviar el programa al micro.

Mediante el *mcboot*, una vez cargada la aplicación deseada en la BT6811, se devuelve el control al terminal del *mcboot*. Por eso si el programa enviado a la BT6811 permite recibir carácteres por el puerto serie se le pueden enviar con tan solo pulsar las teclas en el *mcboot*. A su vez si la BT6811 envía datos por el puerto serie estos aparecerán en la pantalla del *mcboot*.

Por último, en el modo entrenador hay otro programa que adquiere mucha importancia a la hora de buscar aplicaciones autónomas. Este programa es el *ctdialog*, que como se acaba de decir a pesar de ejecutarse en modo entrenador es el que permite, entre otras funciones, dejar grabados los programas permanentemente en la EEPROM interna, de forma que se obtienen aplicaciones autónomas del PC.

Para ejecutarlo en MSDOS hay que hacer lo siguiente:

1. downmcu ctserver -com1

2. ctdialog -com1

Si el puerto serie utilizado es el segundo hay que poner -com2 y tener presente siempre que la BT6811 no dispone de reset software por lo que habrá que hacerlo manual al utilizar el *downmcu*. En el *ctdialog* no es necesario ya que opera sobre el servidor *ctserver* previamente cargado en el microcontrolador.

Para ejecutarlo en Linux no hace falta el paso primero, con poner ctdialog -com1 bastará, aunque ahora sí es necesario pulsar el botón de reset de la BT6811 justo antes de pulsar el ENTER. Una vez que se ha cargado el programa aparecerá en pantalla lo que se muestra en la figura A.5. A partir de ese momento se puede grabar un programa en la memoria EEPROM[2] mediante la orden siguiente:

1. **ms** 1035 0 (Sólo para el micro E2 y sólo en MSDOS)

2. **eeprom** miprograma

Después de terminar la ejecución de la orden (aparece un mensaje de OK), el microcontrolador de la BT6811 tendrá grabado en su memoria *EEPROM* el programa. Esta memoria no es volátil, permaneciendo inalterado su contenido independientemente del estado de las baterías. Esta característica hace posible las aplicaciones autónomas que se describen en los siguientes apartados.

A.6.2 BT6811 en modo autónomo aislado

Al decir que la BT6811 esta funcionando en modo autónomo se hace referencia a que el programa de la aplicación ya se encuentra grabado en la EEPROM interna del microcontrolador, por lo que no es necesario utilizar el PC para descargar los programas y probar. Es decir no hace falta utilizar el adaptador que antes había sido necesario incorporar.

¿ Qué aplicaciones se pueden hacer con una BT6811 ?. La respuesta una vez más: depende de la imaginación del lector, pero se pueden citar ejemplos de cómo dotar de

[2]La memoria EEPROM actua como una EPROM pero que se borra eléctricamente. La particularidad de una EPROM es que sus contenidos se graban electricamente pero se borran con luz ultravioleta, por eso se mantienen independientemente de si hay o no tensión eléctrica. Con la EEPROM se da un paso más ya que se sustituye la luz ultravioleta por unos pulsos de tensión especiales, de forma que el proceso de borrado y escritura se puede realizar sin necesidad de agentes externos y en un tiempo menor. Esta memoria hace que la familia de microcontroladores MC68HC11 haya tenido una aceptación muy grande en la industria, pues de una manera muy fácil permite programar aplicaciones que no necesiten de mecanismos externos para retener en memoria el programa, considerándose así micros autónomos o preprogramados.

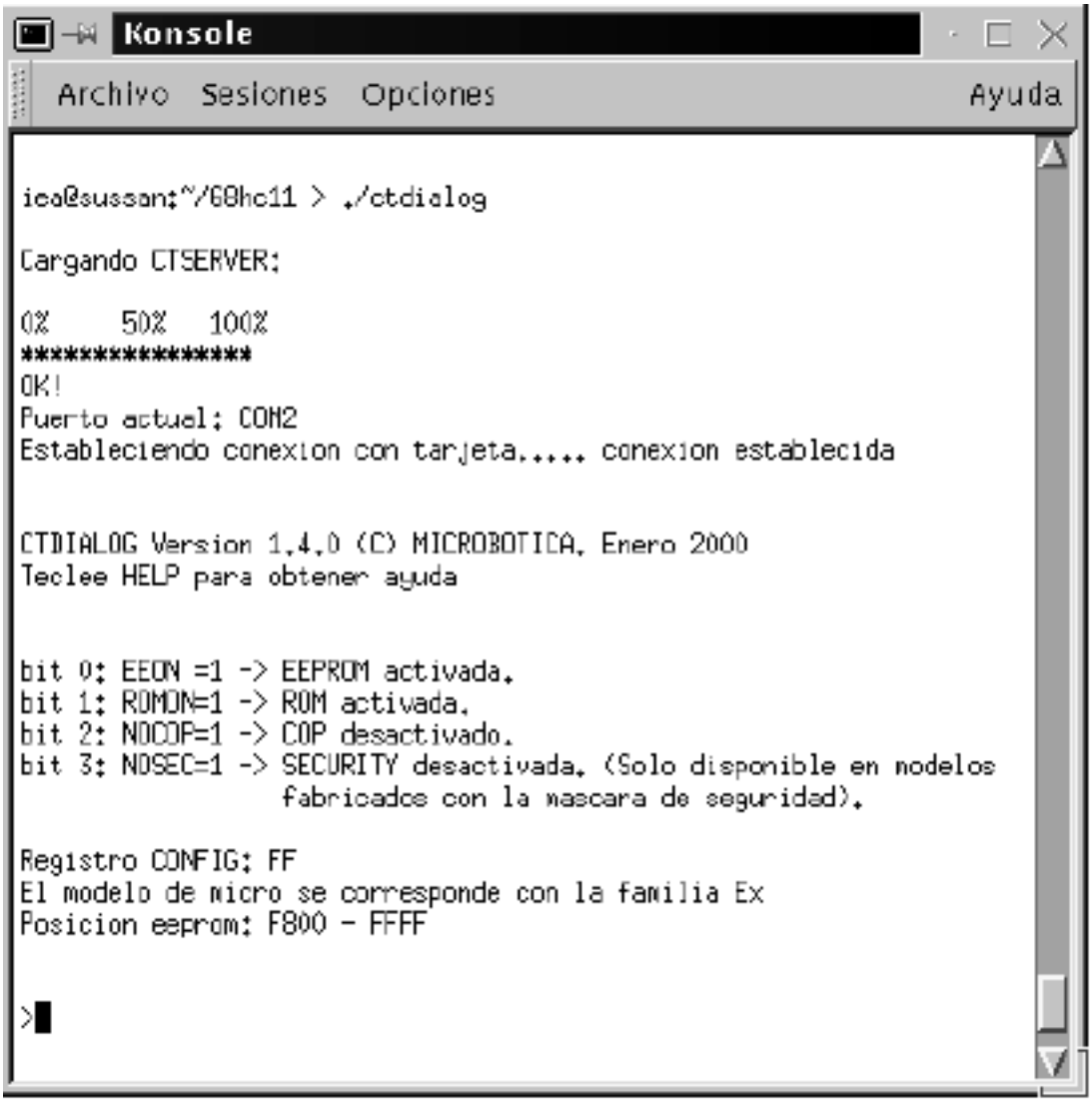


Figura A.5: Programa CTDIALOG bajo Linux.

movimiento a un robot o a una maqueta. Con una BT6811 se podrían mover alternativamente cuatro motores, ocho luces y tal vez emitir alguna señal acústica de vez en cuando. Se recuerda que el motivo principal del desarrollo de esta tarjeta es el poder controlar hasta cuatro servomecanismos Futaba para la realización de robots autónomos. Además tiene suficiente capacidad para gestionar otros recursos como los anteriormente citados : leds, sonidos, bumpers, etc.

Las ventajas de usar la tarjeta en este modo son:

1. Tiene un tamaño reducido y un coste inferior a la CT6811 ya que sólo utiliza los componentes imprescindibles, el resto los elimina.

2. Ya viene preparada para conectar cuatro servomecanismos Futaba.

El lector puede considerar que estas características no son suficientes para justificar la utilización de la BT6811 en lugar de la CT6811 y en parte lleva razón. Se recuerda que para programarla se necesita una CT6811 para realizar todo lo descrito en el punto anterior. Y si ya se dispone de una CT6811, es más fácil construirse el adaptador para los motores que la propia BT6811. Entonces, ¿ dónde está la verdadera utilidad ?. La respuesta se encuentra en las redes de control distribuido, en donde conceptos como la modularidad, el coste y el tamaño toman bastante importancia. En el apartado siguiente se explicarán ejemplos de esto.

Para configurar la BT6811 en modo autónomo (independientemente de si es aislado o en red) se pueden emplear dos métodos. El primero se usa cuando el microcontrolador que lleva la tarjeta es un MC68HC11A1 y el segundo se emplea cuando el microcontrolador es un MC68HC11E2. La diferencia entre ambos se encuentra en que en el primer caso el microcontrolador tiene situada la EEPROM en la dirección $B600 y la única forma de acceder a ella nada más activar la placa es mediante el modo *bootstrap* y la colocación de un jumper. Por el contrario en el segundo caso el microcontrolador tiene situada la EEPROM en el rango $F800 - $FFFF por lo que el vector de reset puede programarse en ella y por ello se habilita el modo de arranque *single-chip*. Mediante este modo nada más activar la tarjeta se ejecuta el programa indicado por el *vector de reset*.

Configuración del modo autónomo para el microcontrolador 68HC11A1:

1. Jumper JP1 puesto para configurar el modo *bootstrap*.

2. Jumper JP2 puesto para permitir el salto a la EEPROM y ejecutar así el programa grabado en ella.

3. Los demás jumpers no tienen importancia en la selección del modo aunque sí en el resto de funcionalidades.

Configuración del modo autónomo para el microcontrolador 68HC11E2:

1. Jumper JP1 quitado para configurar el modo *single-chip*.

2. Jumper JP2 quitado.

3. Vector de reset grabado en la EEPROM y apuntando al inicio del programa.

4. El resto de jumpers no tienen importancia en la selección del modo aunque sí en el resto de funcionalidades.

Para terminar el apartado se explicará mediante un ejemplo cómo obtener una aplicación autónoma. Lo importante del ejemplo no es la funcionalidad del mismo sino el proceso que se va a seguir para tener el sistema funcionando, sobre todo porque en el apartado siguiente todo lo explicado aquí se dará por aprendido. Además, se va a considerar sólo el segundo caso, es decir cuando el microcontrolador de la BT6811 es un MC68HC11E2.

El ejemplo pretende que en cuanto se enchufe la BT6811, empiece a parpadear su LED. El código del programa sería el siguiente:

```
* Programa que hace parpadear el LED de la BT6811

PORTB  EQU $04

    ORG $F800  * Micro E2

inicio         * Inicio del programa
    LDX #$1000

bucle
    LDAA PORTB,X
    EORA #$10
    STAA PORTB,X
    BSR  pausa
    BRA  bucle

pausa
    LDY  #$FFFF
sigue
    DEY
    CPY  #$0
    BNE  sigue
    RTS

* Vector de interrupción del reset

    ORG $FFFE
v_reset FDB #inicio
```

Lo importante del programa es ver cómo se ha configurado el vector de reset (últimas dos líneas) apuntando al inicio del programa, de manera que cuando se arranque en *single-chip* se empiece a ejecutar la instrucción situada en la dirección guardada en el vector. Esa instrucción tiene que coincidir con el inicio del programa. Si no se configura correctamente,

el programa empezará a ejecutarse por cualquier lado siendo impredecible su comportamiento. En caso de utilizar un microcontrolador MC68HC11A1 este vector no se utiliza, por lo que no hará falta configurarlo y en cambio habrá que considerar que cada vez que se haga un reset y se tenga la placa configurada correctamente se empezará a ejecutar la instrucción del programa que se encuentra en la línea $B600.

El programa anterior hay que compilarlo y mediante el *ctdialog* grabarlo en la EEPROM interna. La forma de realizar esto último se ha explicado en el apartado anterior.

1. Para compilar : as11 miprograma.asm
2. Ejecutar el *ctdialog* : > ctdialog -com1[3]
3. Para grabar : **eeprom** miprograma

Una vez grabado el programa se configura la BT6811 y si todo ha ido bien, cada vez que se pulse el reset o se enchufe la tarjeta el LED empezará a parpadear. A continuación se explicará el modo autónomo en red que, como ya se ha dicho, es con el que se obtiene la máxima utilidad de la BT6811.

A.6.3 BT6811 en modo autónomo en red

La BT6811 como sistema de control aislado tiene unas ventajas limitadas, pero cuando se utilizan varias tarjetas para formar un sistema distribuido de control éstas aumentan. Ya no sólo es importante el tamaño, sino el precio y la funcionalidad de cada unidad. Suponga por un momento que desea construir un robot que tenga cuatro servomecanismos y que sea autónomo. Mediante una BT6811 aislada lo puede hacer, pero ¿qué ocurre si más tarde desea ampliar el robot ?, es decir, ponerle más motores. Es evidente que necesitará ampliar el hardware y lo más probable es que utilice otra BT6811. Ahora bien, el software hay que cambiarlo, ahora hay dos tarjetas en lugar de una: ¿cómo repartir el programa?, ¿ cómo hacerlo en futuras ampliaciones?. Son preguntas que sirven para demostrar la utilidad de la red de control. Si cada vez que se introduce un módulo hay que rectificar todos los programas, algo se está haciendo mal. Además, cuando se construye un robot es muy frecuente que a medida que se van obteniendo buenos resultados se vayan adquiriendo nuevos retos. Y está claro que si se sigue por el camino de modificar todo cada vez que se introduce un nuevo elemento, mal camino se ha escogido.

¿ Cómo se puede evitar lo anterior ?. La respuesta está en la modularidad. La mayoría de los proyectos se pueden subdividir en módulos más o menos independientes y con funcionalidades parecidas o totalmente dispares. De forma que para retocar un módulo no haga falta atacar a los otros. Además si se decidiese a añadir más, lo más seguro es que no se necesite reprogramar todo el sistema, o por lo menos de eso se trata. Todos estos módulos no deben actuar por su cuenta, sino que tienen que guardar un sincronismo entre ellos, deben intercambiar información o por lo menos tener uno maestro que controle a los demás. Sería como construir una plataforma móvil partiendo de bloques ya establecidos, es decir, utilizando un módulo motriz, un módulo sensorial, un módulo actuador y para

[3]Considerar que se necesita el adaptador, lo del reset software y la diferencia entre el método en linux y msdos. es decir todo lo explicado en 6.6.1.

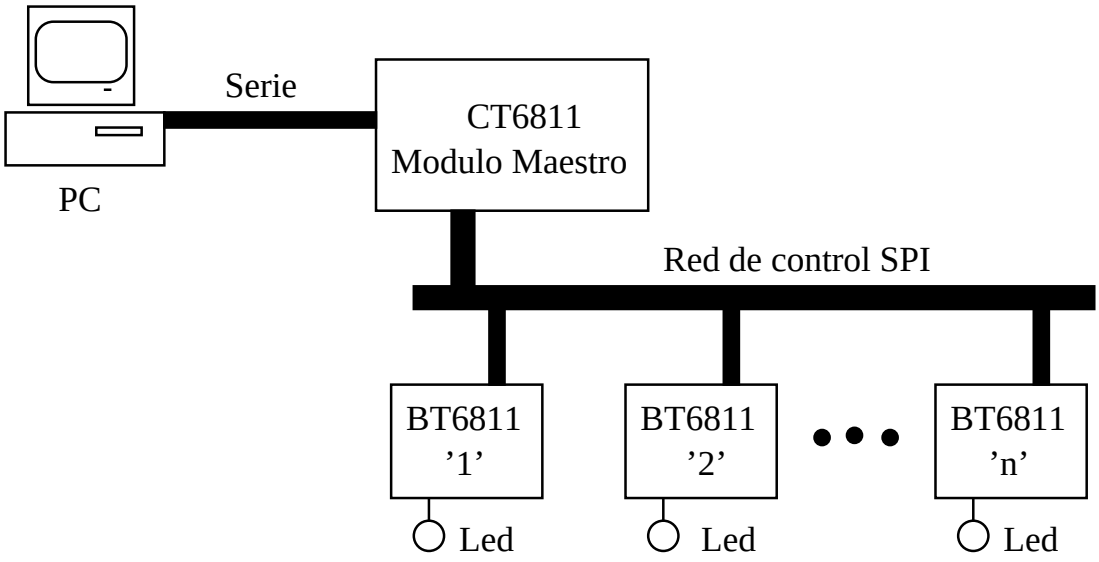


Figura A.6: Representación de la red de BT6811.

tener todo comunicado un módulo maestro que gestione los anteriores. Si una vez desarrollado hay que añadir algún nuevo elemento, tan sólo hay que añadirlo a la estructura y modificar el módulo de gestión, los demás deberían permanecer inalterados. Para lograr esto se requiere una buena planificación del proyecto y una descripción buena de los interfaces entre módulos.

Debido a que los ejemplos suelen ser más útiles que las palabras expondremos uno. Aprovechando el LED de la BT6811 se diseñará una red de control que permita desde un módulo maestro encender y apagar cada LED de cada BT6811.

Lo primero que hay que hacer es pensar en la aplicación desde un punto de vista general: el esquema de la figura A.6 servirá de ayuda. Se utilizará el PC para mandar las órdenes de encender y apagar el LED de cada BT6811. Habrá un módulo maestro que recibirá por el puerto serie los datos del PC y después los enviará por la red de control. A esta red están conectadas todas las BT6811 que uno quiera y a pesar de que todas ellas recibirán la misma orden, no habrá conflicto. Mediante un identificador cada BT6811 podrá saber si la orden va o no dirigida a ella y en el caso afirmativo procederá a encender o apagar su LED. En este ejemplo todos los módulos van a tener la misma función, pero puede haber otros casos en los que no. En estos, habrá que estudiar bien qué órdenes se van a transmitir por la red. Cuanto mejor se especifiquen éstas, más detallado se tendrá el protocolo final de la red y menos modificaciones se tendrán que hacer a largo plazo.

En este ejemplo todos los módulos BT6811 tendrán la misma función: Encender y apagar su LED cuando se lo indique el módulo maestro. Con esta decisión se puede especificar el protocolo de la red o, lo que es lo mismo, el formato de las tramas y el procedimiento para enviarlas por la red.

Cada trama tendrá dos bytes, el primero identificará la BT6811 a la cual va dirigida la orden y el segundo será la propia orden, es decir, encender o apagar el LED, ver la tabla A.2. El identificador **Id** será un número entre 1 y 9, y el código de orden, **n_orden,** será el carácter **'a'** para encender el LED y el carácter **'s'** para apagarlo.

Dicho lo anterior se va a proceder a programar el módulo BT6811. Con hacer un pro-

Identificador de destino	orden
1 Byte	1 Byte
Id	n_orden

Tabla A.2: Especificaciones de la trama de control.

grama general bastará y para añadir más módulos tan sólo habrá que ir cambiando el valor de una etiqueta, en concreto la indicada en el programa por **M_ID**. El código del programa se detalla a continuación.

```
* ........................................
* . BTgen. Version 1.0                    .
* ........................................
* . Modulo general de la red de BT6811    .
* ........................................

*******************************************
* Identificador de la BT6811              *
* Incrementar una unidad en cada .nueva BT *
*******************************************

M_ID  equ '1'

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

* Puertos del 68hc11

PORTB equ $04

* Registros del SPI

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

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

LED  equ $10 ; Bit donde se encuentra el LED
```

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

        ORG $F800    * Programa para un E2

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

inicializar
        LDS #$FF     * puntero de pila
        LDX #$1000   * acceder a registros

*--- Configuración del SPI (como esclavo)

        LDAA  #$3C    * SS de entrada
        STAA  DDRD,X

        LDAA  #$40    * Colector cerrado
        STAA  SPCR,X  * Modo esclavo el SPI


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

inicio
        BSR  recibir_spi
        CMPA #M_ID        * verifica el ID
        BEQ  analizar     * SI->lee orden
        BSR  recibir_spi  * .NO->no hagasñada
        BRA  inicio

analizar BSR  recibir_spi * lee la orden
         CMPA #'a'        * ¿ Es a ?
         BEQ  led_on      * SI -> enciende led
         CMPA #'s'        * ¿ Es b ?
         BEQ  led_off     * SI -> apaga led
         BRA  inicio

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

```
*.............................
*. Rutina enciende el LED     .
*.............................

led_on
        BSET PORTB,X LED
        BRA  inicio

*.............................
*. Rutina apaga el LED        .
*.............................

led_off
        BCLR PORTB,X LED
        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  * espera dato
        LDAA  SPDR,X
        RTS

*****************************
* vectores de interrupcion   *
*****************************

         ORG $FFFE          * vector del reset
v_reset  FDB  #inicializar

         END
```

El programa no es complicado: el bucle principal lo único que hace es esperar a recibir el primer byte de cada trama enviada por el bus de control (formado por el SPI). Este byte (**Id**) identifica a la tarjeta a la que se ha enviado la orden, por eso lo siguiente que hace el bucle principal es comparar el byte con su identificador (**M_ID**). Si no coinciden ambos el bucle lee el siguiente byte de la trama, pero no lo utiliza, vuelve al bucle principal a esperar otro principio de trama. Por el contrario si ambos identificadores son iguales, el bucle lee el siguiente byte, lo analiza y en función de su contenido enciende o apaga el LED

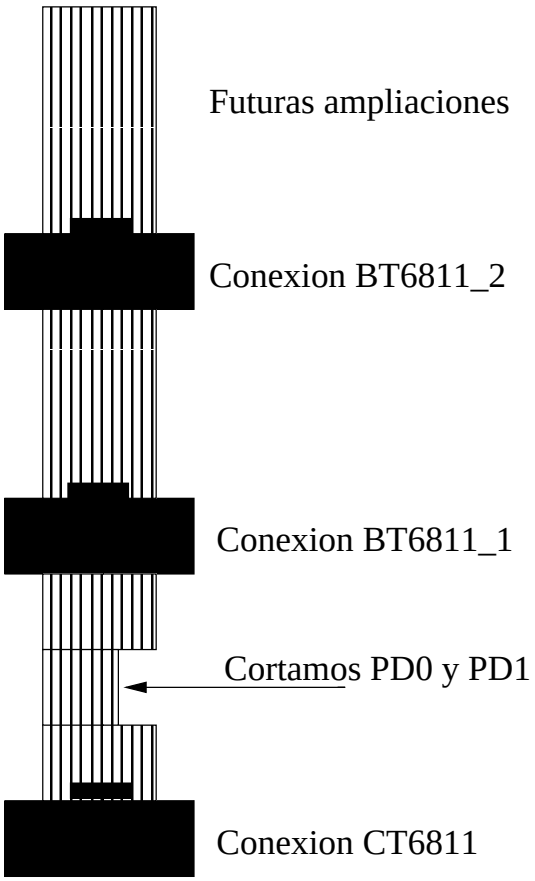


Figura A.7: Cable para conectar los distintos módulos entre sí.

de la BT6811. Una vez terminado esto vuelve al bucle principal para seguir esperando una nueva trama. Tal vez la parte más complicada es la de recibir los datos por el bus SPI ya que para ello hay que configurarlo de una manera especial, en concreto en los módulos esclavos hay que programarlo con los parámetros siguientes: colector cerrado, señal SS de entrada y modo esclavo. Todo esto se hace en la inicialización del programa. Por último, como el microcontrolador que se está utilizando es el MC68HC11E2 pondremos el vector de reset apuntando al principio del programa para poder arrancar en modo *single-chip*[4].

El lector puede preguntarse cómo se conectan los módulos entre sí, la respuesta es sencilla, o se construye un cable que una los pines MOSI, SS, CLK y GND de todos los módulos incluyendo el maestro, o se construye un cable tipo bus con tantos conectores de salida como módulos tenga la red. Esta última opción es la recomendada en el ejemplo, aunque se podría tomar la precaución de cortar los cables correspondientes al puerto serie (PD1 y PD0) en el enganche al módulo maestro, de esa forma se evitarán conflictos con los puertos serie de los módulos esclavos. En la figura A.7 se representa cómo construirse el cable de conexión del bus SPI. Por último, para conectar el PC al módulo esclavo se utilizará el cable serie proporcionado en la CT6811. La alimentación del sistema se introducirá, por ejemplo, por el jack de la CT6811 y se distribuirá por toda la red uniendo con

[4]En el apartado 'A.2: modos de funcionamiento' y en el 'A.6.2: funcionamiento en modo autónomo aislado' hay más información sobre el modo de arranque *single-chip*.

un cable todas las salidas positivas de las clemas V_TTL de cada BT6811 (ver figura A.4).

Ya se tiene construida la red y el programa general de los módulos esclavos. Ahora se tiene que grabar éste en cada módulo, para lo que se utiliza toda la teoría explicada en el apartado anterior. Al final se configurará cada BT6811 en modo autónomo aislado, pero con un par de añadidos por utilizar la red SPI, en concreto:

1. Jumper JP1 quitado para configurar el modo *single-chip*.
2. Jumper JP2 quitado.
3. Jumper JP4 quitado ya que en la red se va a colocar una CT6811.
4. Jumper JP6 puesto ya que el control por el SPI va a utilizar la señal SS.
5. El resto de jumpers no tienen importancia en la aplicación.

Al grabar el programa en cada BT6811 hay que acordarse de modificar la etiqueta M_ID que identifica a cada módulo. Se empieza dándole el valor uno al primer módulo y se va incrementando a medida que se van grabando más módulos. En caso de dejar el mismo valor para todos, ocurrirá que todos los LEDs se encenderán o apagarán a la vez.

Ahora queda programar el módulo maestro, que como ya se ha dicho es una CT6811. Como va a estar conectado al PC se puede elegir entre tener grabado el programa en la memoria EEPROM o volcarlo cada vez que se utilice la aplicación. Para facilitar las cosas se hará de esta segunda forma, es decir se dejará configurada en modo entrenador[5]. Se usará el MCBOOT para enviar el programa maestro a la CT6811 y luego tecleando una serie de órdenes se verá como el LED de los módulos esclavos cambia de estado. A continuación se detalla el código del programa maestro.

```
*********************************************
*     Programa para el modulo maestro       *
*********************************************
*  El programa recibe  ordenes por el puerto*
* serie, las interpreta y manda las tramas  *
* adecuadas  por la  red_SPI para  que  le  *
* llegue la orden a la BT6811 adecuada.     *
*********************************************

* --- Puertos de entrada y salida

PORTA   EQU $0

* --- Registros del SCI

BAUD    EQU $2B
```

[5]Para ver la configuración en modo *'entrenador'* recurrir al *'manual de usuario de la CT6811'*. Este se encuentra disponible en la WEB de Microbótica S.L. [10].

```
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  * micro E2

inicio  LDS   #$FF    * pila
        LDX   #$1000  * registros

*---- Configuracion del SCI

wait    BRCLR SCSR,X $40 wait

        LDAA  #$22      * Poner 22 para  7812 baudios
        STAA  BAUD,X    * Poner 30 para  9600 baudios
        LDAA  #$0
        STAA  SCCR1,X   * 8 bits de datos
        LDAA  #$0C      *.No usar interrupciones del SCI
        STAA  SCCR2,X   * Activar receptor y transmisor

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

        LDAA  #$3C
        STAA  DDRD,X
        LDAA  #$50      * Colector cerrado
        STAA  SPCR,X    * Activa en modo maestro el SPI

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

bucle_prin
        JSR  leer_car    * leo carácter por el puerto se-
rie
        JSR  enviar_car  * hace eco del caracter
        JSR  enviar_spi  * mando carácter al bus spi
```

```
        BRA  bucle_prin

*.................................
*. 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 envia un dato por el SPI .
*.......................................

enviar_spi
        LDAB  PORTD,X
        ANDB  #$DF
        STAB  PORTD,X    *  activarse al esclavo

        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

*********************************
*   Vector de interrupcion      *
*********************************

        ORG $FFFE
v_reset FDB #inicio
```

```
END
```

El programa maestro tampoco es difícil de comprender y se podría reducir más, ya que al utilizar el modo entrenador el puerto serie se auto configura, igual que la pila, por lo que se podían haber ahorrado todas esas líneas de configuración. Las líneas de configuración del SPI vuelven a ser claves, aunque ahora se ha configurado como colector cerrado y como maestro.

Después de la configuración se ejecuta el bucle principal que lo único que hace es esperar a recibir datos del PC. En cuanto recibe uno hace eco y lo envía por el bus SPI a los módulos esclavos. Por lo tanto si con el MCBOOT se teclea '1' y luego 'a' se debería encender el LED del módulo esclavo cuyo identificador M_ID es igual a uno. Si se hace lo mismo pero tecleando un '2' debería pasar lo mismo pero en el módulo esclavo cuyo M_ID fuera dos. Para apagar el LED de la BT6811 hay que pulsar primero el '1' y luego 's'.

Si una vez terminada la aplicación se quiere añadir un nuevo módulo habrá que hacer lo siguiente:

1. Se incrementa el valor M_ID del programa de los módulos esclavos (BT6811).
2. Se graba dicho programa en el nuevo módulo y se configura en modo autónomo en red.
3. Se conecta el módulo al bus SPI y a la alimentación de 5 voltios.
4. El resto de módulos se dejan como están.

Tal como se dijo al principio del ejemplo al ser un sistema modular no ha hecho falta tocar ninguno de los módulos ya existentes para ampliar el sistema. El lector puede considerar que este ejemplo es muy sencillo, pero en la realidad hay muchos sistemas de control que se adaptan al ejemplo. De todas formas la BT6811 todavía admite más complejidad en la red. Por ejemplo, en lugar de usar la señal SS común para toda la red se puede independizar (quitar el jumper JP6). En su lugar se conectará una salida del módulo maestro, como puede ser algún pin del puerto B. Haciendo esto se consigue una comunicación de ida y vuelta entre el maestro y el esclavo, de forma que se puede intercambiar información en ambos sentidos. Además, en lugar de ser la trama la que identifica el destino es el propio maestro el que lo hace. En este caso al incluir un nuevo módulo hay que retocar el módulo maestro para que lo reconozca, pero por el contrario, este nuevo módulo puede tener una funcionalidad distinta sin que afecte a la red. Como podrá apreciar el lector las posibilidades son muchas y cuanto más practique más irán surgiendo. Para adquirir experiencia se recomienda leerse el capítulo relacionado con el SPI del libro *Microcontrolador MC68hc11: Fundamentos, recursos y programación* [9].

Apéndice B

Manual de usuario de la CT6811

La finalidad de este manual es dar una idea global de la CT6811, sin entrar en detalles demasiado técnicos, para que el usuario sea capaz de manejarla y sacar el máximo provecho de ella. Este manual no es un curso de aprendizaje del 68hc11, sino una guía para poder usar la CT6811.

B.1 Primeros pasos

A continuación se enumeran los pasos para ponerla en funcionamiento en modo entrenador sin necesidad de conocer nada más.

1. Asegúrese de que los jumpers JP1, JP2, JP3 y JP8 estén colocados.
2. El jumper JP5 deberá estar quitado. Si la tensión de alimentación es inferior a 5v el jumper JP6 deberá estar quitado también. El estado predeterminado es quitado.
3. El jumper triple JP4 estará situado en la posición RST (a la derecha).
4. El jumper triple JP7 estará situado en la posición ON (abajo).
5. Conecte la tarjeta al PC por medio del adaptador y del cable.
6. Alimente la tarjeta con una tensión comprendida entre 4.5 y 6 voltios (tensión nominal 5v). (Se aconseja leer el apartado 7 de este manual).
7. Teclee downmcu ctserver -com1 (-com2).
8. Teclee ctdialog -com1 (-com2).

El software para el control de la CT6811 se ha desarrollado para las plataformas MSDOS y LINUX, todos los programas tienen una interfaz muy parecida aunque no idéntica, por eso según cual sea la plataforma escogida por el lector puede haber alguna que otra diferencia. Los ejemplos de este manual se refieren a la plataforma LINUX por ser esta la escogida para la realización de este proyecto.

El manual de usuario que se incluye a continuación ha sido escrito por Cristina Doblado Alcázar, Juan González Gómez, Juan José San Martín y el propio autor de este proyecto, por eso y contando con la autorización de mis compañeros incluyo el manual para completar la información del capítulo 4 dedicado a la electrónica de control.

B.2 Presentación de la tarjeta CT6811

La CT6811 (figura 4.3, página 73)es una tarjeta para el desarrollo de aplicaciones hardware y software basadas en el microcontrolador 68HC11. No sólo sirve para el desarrollo de prototipos, sino que también está pensada para funcionar en sistemas terminados. Se trata sobre todo de una tarjeta muy versátil, que se puede emplear como:

1. **Tarjeta entrenadora**: Conectándose a ordenadores PC a través del puerto serie. Es posible enviar programas desde el PC a la CT6811 para que los ejecute. De esta manera, la tarjeta es muy útil para la depuración de programas en fase de pruebas y para aprender a programar el 68HC11 desde un punto de vista práctico: todos los programas creados sobre el 'papel' se pueden ejecutar en un 68HC11 'de verdad'.

2. **Tarjeta autónoma de control**: La CT6811 funciona en modo autónomo, es decir, sin conectarse al PC. Cada vez que se encienda la tarjeta, se ejecutará el programa que previamente se habrá grabado en la memoria EEPROM del 68HC11. Conectando periféricos a través de los puertos de expansión, se consiguen desarrollar sistemas inteligentes de control: control de las luces de una casa, control de la maqueta de un tren, manejo de microbots, alarmas...

3. **Periférico inteligente del PC**: También es posible utilizar la tarjeta para comunicar el PC con el mundo exterior. La CT6811 actuaría como un periférico conectado al PC a través del puerto serie. Este periférico 'inteligente' puede recibir órdenes del PC y actuar en consecuencia. Se pueden realizar aplicaciones del tipo: digitalización de señales y presentación en el PC, diseño de joysticks no convencionales, control de luces de la casa a través del PC, control de maquetas de tren visualizando en el PC la situación de los trenes, semáforos...

B.2.1 Características de la CT6811

1. Microcontrolador **68HC11**: Por ello incorpora todas las características de este micro:
   (a) 512 Bytes de EEPROM en los modelos A1 y A8. En el modelo E2 hay 2048 bytes de EEPROM
   (b) 256 Bytes de memoria RAM interna
   (c) Temporizador de 16 bits
   (d) 3 capturadores de entrada
   (e) 5 comparadores con salida hardware
   (f) Un acumulador de pulsos
   (g) Comunicaciones serie asíncronas
   (h) Comunicaciones serie Síncronas
   (i) 8 canales de conversión analógico digital
   (j) Interrupciones en tiempo real
   (k) 4 puertos de E/S

2. **Bus Expansión**: Bus de expansión dividido en 6 puertos de 10 bits cada uno. Desde estos puertos se tiene acceso a todas las patas del micro.

3. **Switches de configuración**: 2 switches de configuración del modo de arranque de la placa (bootstrap, single chip, expanded, special test) y 2 switches disponibles para aplicaciones del usuario

4. **Led de pruebas** conectado al bit 6 del puerto A. Este led se puede conectar y desconectar por medio de un jumper

5. **Pulsador de reset/pruebas IRQ:** Pulsador para inicializar la tarjeta. Este pulsador se puede utilizar también, cambiando un jumper, como un pulsador de propósito general conectado a la entrada de interrupciones IRQ.

6. **Niveles VRH y VRL** configurables a VCC y masa respectivamente mediante 2 jumpers

7. Modos de funcionamiento **entrenador/autónomo** configurables mediante un jumper

8. **Alimentación** a través de clemas o conector tipo jack

9. **Conexión directa al PC** por medio de un cable tipo teléfono

10. **Reset software y hardware** de la placa. El jumper JP7 permite anular el reset software.

B.2.2 Diagrama de bloques de la CT6811

El núcleo central de la tarjeta CT6811 está constituido por el microcontrolador 68HC11A1 de Motorola. Los demás bloques surgen como una extensión del 68HC11, configurándolo para funcionar adecuadamente y adaptándolo para las necesidades del usuario. La tarjeta se puede dividir en 8 bloques (ver figura B.1): el corazón de la placa (68HC11), el circuito de reloj, el circuito de inicialización o reset, el circuito de pruebas, el circuito de comunicaciones con el PC, la configuración del sistema, los puertos de expansión y la alimentación del sistema completo. A continuación se explica cada bloque:

1. **Microcontrolador 68HC11A1**: Se trata de un microcontrolador de 8 bits. Además de una CPU que le permite ejecutar instrucciones y realizar operaciones aritmético lógicas, dispone en el propio chip de memorias ROM, RAM y EEPROM, así como una serie de periféricos integrados que hacen de este micro una herramienta muy potente.

2. **Circuito de reloj**: Este circuito permite que el micro obtenga la señal de reloj necesaria para funcionar. Está constituido por un cristal de 8MHZ, que es el valor máximo que permite el 68HC11. Además, con este valor, el micro es capaz de comunicarse con el PC a las velocidades de 1200, 2400 ó 9600 baudios.

3. **Circuito de reset**: El circuito de reset permite la correcta inicialización del 68HC11. Se ha diseñado de tal forma que es posible realizar un reset por 'hardware', apretando un pulsador, o bien realizar un reset por software desde el PC, cuando la CT6811

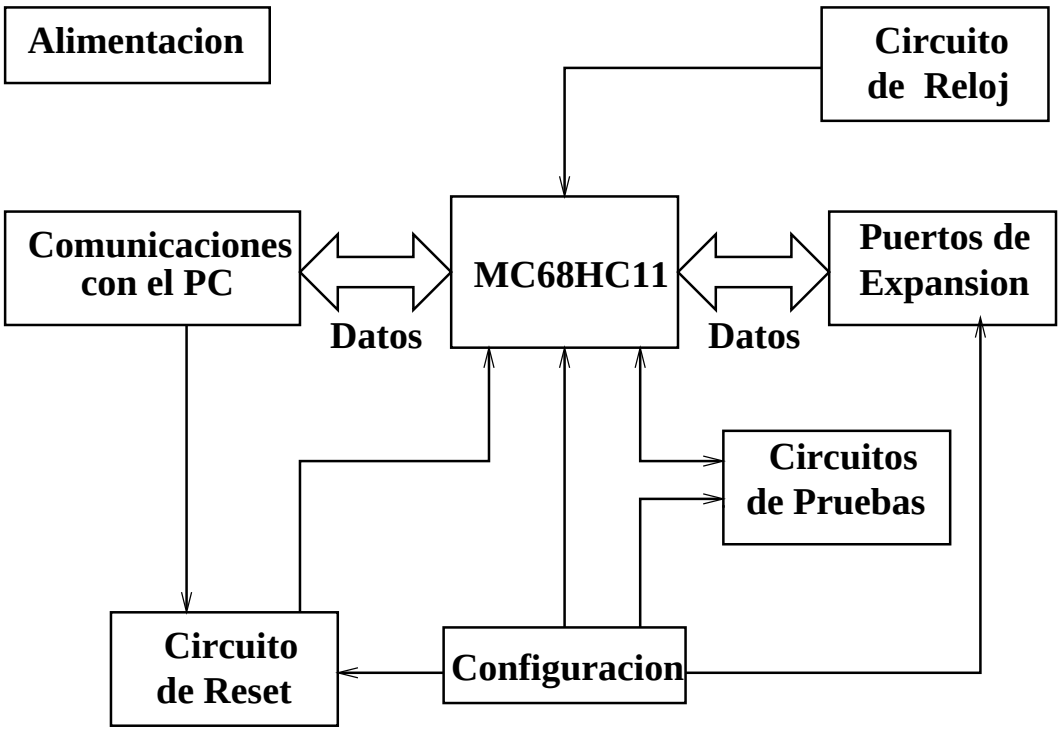


Figura B.1: Diagrama de bloques de la CT6811.

está funcionando en modo entrenador. El reset software es muy útil cuando se están desarrollando microbots, pues cada vez que hay que cargar un programa nuevo en el microbot, no es necesario desplazarse hasta él y apretar el botón de reset, sino que directamente desde el PC se puede llevar a cabo.

4. **Configuración**: La parte de configuración de la CT6811 está constituida por 6 jumpers y 4 switches. Dos de los switches permiten configurar el 68HC11 para trabajar en cualquiera de los 4 modos para los que está diseñado: bootstrap, single chip, expanded y special. Los otros 2 switches están a disposición del usuario a través de los puertos de expansión para configurar tarjetas periféricas conectadas a la CT6811. Los 6 jumpers permiten configurar la CT6811 para realizar diferentes funciones, que se comentarán más adelante.

5. **Circuito de pruebas**: La CT6811 dispone de un led conectado al bit 6 del puerto A del 68HC11. Este led, que se puede conectar y desconectar a través de un jumper, es muy útil para la realización de software de pruebas. El pulsador de la tarjeta, se puede configurar mediante un jumper para actuar bien como pulsador de reset o bien como un pulsador normal conectado a la entrada de interrupción IRQ del 68HC11. De esta forma, es posible realizar pequeños programas de prueba que lean el pulsador y actúen en consecuencia.

6. **Comunicaciones con el PC**: Esta es una de las partes más importante. A través del circuito de comunicaciones es posible comunicarse con el PC para intercambiar información. Las comunicaciones se realizan a través del puerto serie del PC, mediante

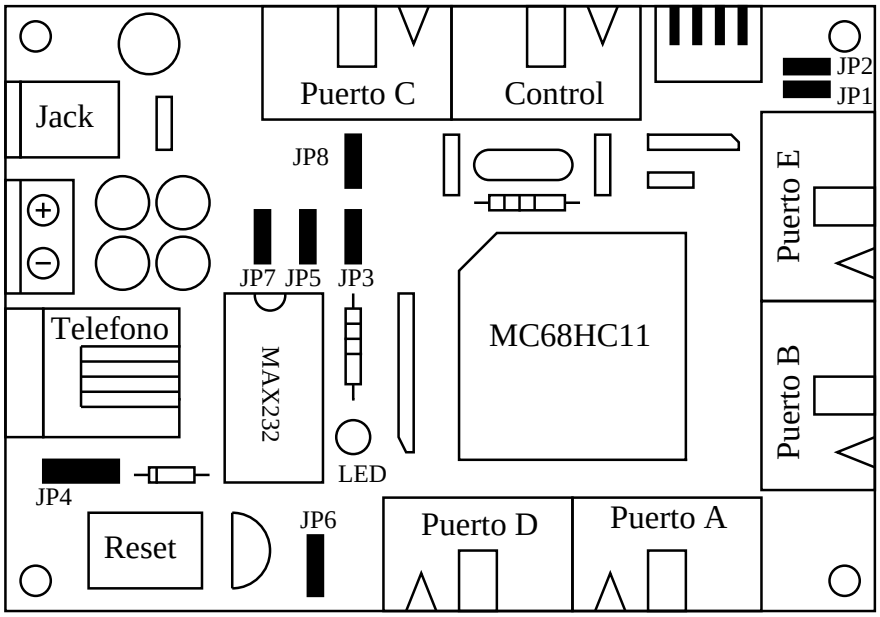


Figura B.2: Componentes de la CT6811.

la norma RS-232. La velocidad máxima compatible con el PC es de 9600 baudios, aunque el 68HC11 es capaz de transmitir a mucha más velocidad.

7. **Puertos de expansión**: La CT6811 dispone de 6 puertos de expansión donde es posible conectar los diferentes periféricos. Todas las señales del 68HC11, salvo las correspondientes al reloj (EXTAL y XTAL), son accesibles desde los puertos de expansión. Cinco de los puertos coinciden con los 5 puertos del 68HC11: A,B,C,D y E. El sexto puerto está destinado a llevar las señales de control del 68HC11.

8. **Alimentación**: La tensión de alimentación de la CT6811 oscila entre 4.5 y 5.5 voltios. La tarjeta dispone de un conector hembra de tipo jack cilíndrico para la conexión de un transformador y dos bornas ajustables mediante tornillos para la conexión de cables de alimentación, provenientes por ejemplo de pilas o baterías.

B.2.3 Aspecto físico y situación de los componentes

En el diseño de la CT6811 se han tenido en cuenta las siguientes condiciones: Los componentes no pueden ser SMD, tienen que ser normales para permitir una sustitución fácil. Hay que facilitar la ampliación del sistema, para ello hay que dividir el bus en grupos funcionales y permitir un acceso sencillo a ellos. Por último hay que buscar el menor tamaño posible..

En la figura B.2 se aprecia la disposición de los componentes de la CT6811, el tamaño final es de 8.2 x 6.4 cm. Estas dimensiones hacen que sea una herramienta muy buena para el diseño de microbots y sistemas empotrados. Además el sistema se ha pensado para ampliarse verticalmente, de manera que las futuras expansiones no aumenten la superficie sino el volumen.

De todos los componentes de la CT6811 los que más pueden interesar al usuario son los siguientes:

- Los jumpers: JP1, JP2, JP3, JP4, JP5, JP6, JP7, JP8
- Puertos de expansión: Puerto A, Puerto B, Puerto C, Puerto D, Puerto E y Control.
- Conectores de alimentación: Jack y clema de alimentación.
- Switches de configuración.

B.3 Modos de funcionamiento

La tarjeta CT6811 se ha diseñado para trabajar en dos modos diferentes: modo autónomo y modo entrenador. Estos modos se configuran mediante los switches de control y uno de los jumpers (JP5).

Funcionamiento en modo entrenador

En esta configuración, la CT6811 se conecta a un PC a través del puerto serie. Los programas se escriben y se compilan en el PC y mediante un programa de comunicaciones serie se envían a la entrenadora para ser ejecutados. También es posible en este modo utilizar el PC como si fuese un terminal tonto de la tarjeta, empleando el teclado del PC para enviar datos a la CT6811 y utilizando el monitor del PC para visualizar datos generados en la CT6811.

El modo entrenador también se emplea para realizar programas de autotest de la tarjeta y presentar los resultados en el monitor del PC.

La configuración típica en modo entrenador tiene todos los jumpers colocados excepto el JP5, el JP4 situado a la derecha (RST) y el JP7 abajo (ON). Los switches 1 y 2 deben estar hacia abajo. Además la CT6811 tiene que estar conectada al PC a través del cable de teléfono y mediante un transformador o baterías alimentada a una tensión entre 4.5 y 6 voltios.

Funcionamiento en modo autónomo

En este modo de funcionamiento la CT6811 al ser inicializada (pulsar botón de reset o alimentar la tarjeta) comienza a ejecutar el programa que se encuentra en la memoria EEPROM interna del micro. Este programa ha sido grabado previamente en modo entrenador utilizando un software especial.

La CT6811 se diseñó pensando en los microbots, pequeños robots móviles y buscando un sistema pequeño y autónomo de control. Por eso este modo adquiere un gran protagonismo en las aplicaciones con la tarjeta. Para configurarla hay dos alternativas dependiendo del modelo de microcontrolador que posea la tarjeta:

1. Lo típico es disponer del 68HC11A1 y entonces todos los jumpers deberán estar colocados, el JP4 situado a la derecha y los switches abajo. Estando la CT6811 alimentada al apretar el pulsador se ejecutará el programa residente en la eeprom: Lo malo es

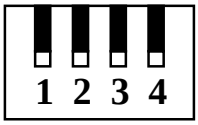
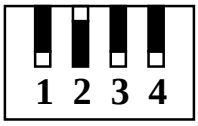
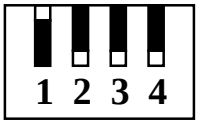
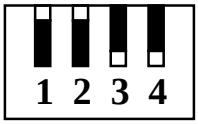
SWITCH 1 y 2	MODO	DESCRIPCION
	**BOOTSTRAP**	Ejecucion programa BootStrap Modo especial Sin acceso a memoria externa
	**SINGLE CHIP**	Vector interrupcion en ROM Sin acceso a memoria externa
	**SPECIAL TEST**	Acceso a memoria externa Modo especial Vector interrupcion en ROM
	**EXPANDED**	Vector interrupcion en RAM o ROM Acceso a memoria externa

Figura B.3: Configuración de los modos del 68hc11 mediante los switches de la CT6811.

que al estar puesto el jumper JP5 se están perdiendo las comunicaciones serie, salvo que después del arranque se libere dicho jumper.

2. Cada vez es más común disponer del 68HC11E2, microcontrolador que tiene 2Kbytes de memoria EEPROM situados en las posiciones $F800 - $FFFF. Para arrancar el modo autónomo se puede hacer lo mismo que antes, o también arrancar la placa con el switch 2 arriba y con el *vector de reset* apuntando al principio del programa (esto último se hace en el programa de la aplicación). La ventaja es que el jumper JP5 puede estar quitado y con ello no se pierden las comunicaciones serie.

B.4 Configuración de la tarjeta

La tarjeta CT6811 dispone de switches y jumpers para que el usuario la configure según sus necesidades. Existen 4 switches y 8 jumpers. De los 4 primeros, dos se utilizan para configurar los modos de funcionamiento del 68HC11 y los otros dos quedan disponibles para las aplicaciones del usuario.

B.4.1 Configuración de los modos del micro

El 68HC11 puede funcionar en 4 modos diferentes. Utilizando los mismos nombres que figuran el manual de 68HC11, los modos son: *single chip*, *special bootstrap*, *expanded* y *special test*. Los dos primeros son los más utilizados y en caso de disponer de un microcontrolador 68HC11A1 se puede reducir el número al modo *Bootstrap*.

Single_Chip: En este modo de funcionamiento, el mapa de memoria del 68HC11 está constituido por la memoria RAM, la memoria EEPROM , los registros de control y la memoria ROM. Este modo está pensado para funcionar cuando existe un programa grabado en la ROM, de tal manera que al arrancar se comience a ejecutar el programa indicado por los vectores de interrupción que se encuentran en la ROM.

Expanded: Además del mapa de memoria del modo single chip, es posible acceder al resto de las posiciones de memoria conectando memorias externas. El precio a pagar es que se pierden 2 puertos de E/S el puerto B y el puerto C. En este modo se puede utilizar la memoria ROM interna, pero también es posible deshabilitar esta ROM y acceder a los vectores de interrupción que se encuentren en una memoria externa.

Special_Bootstrap: Este modo difiere del single chip en que los vectores de interrupción no se encuentran en la memoria ROM de 8K sino que se encuentran en otra memoria ROM, llamada la ROM de arranque. Al arrancar en este modo, automáticamente comienza a ejecutarse el programa BOOTSTRAP que se encuentra en esta ROM.

Special_Test: Igual que el modo Bootstrap con la salvedad de que se puede acceder a memoria externa. Este modo se utiliza para realizar pruebas de fábrica. En este modo especial se tiene acceso a determinados registros de control que en otros modos están protegidos.

Estos modos de funcionamiento del micro se pueden configurar mediante 2 switches en la CT6811. En los modos de trabajo Entrenador y autónomo de la CT6811, se utiliza siempre el modo bootstrap del 68HC11. Los modos special test y expanded se pueden utilizar si se conecta a la CT6811 una tarjeta periférica con memoria, como por ejemplo la CT3216. El modo single chip se utiliza cuando se ha cambiado el microcontrolador, utilizando en lugar del modelo A1, el modelo E2, que permite tener grabado el vector de reset en la EEPROM interna.

Para configurar los modos del micro, sólo se utilizan los switches 1 y 2. Estos switches actúan directamente sobre los pines MODA y MODB del 68HC11, poniéndolos a VCC o a GND. En la figura B.3 se encuentra resumida la información sobre la configuración de los modos.

B.4.2 Configuración de los jumpers

Existen 8 jumpers que permiten configurar aspectos del 68HC11 y de la CT6811.

- **Jumpers JP1 y JP2**: Configuración de los niveles VRH y VRL del micro. Estos jumpers se utilizan para fijar los niveles de tensión de las entradas VRH y VRL del 68HC11. El jumper JP1 actúa sobre VRH. Si está colocado, la pata VRH está cortocircuitada con VCC. Si no se coloca, esta tensión se puede fijar desde el exterior, a través del puerto E. El jumper JP2 actúa sobre VRL. Si está colocado, la pata VRL se cortocircuita con GND. En caso contrario queda accesible desde el exterior a través del puerto E. Estos dos jumpers sólo se utilizan cuando se trabaja con el conversor A/D del 68HC11. Los jumpers normalmente estarán colocados. Sólo para aplicaciones muy específicas habrá que quitarlos y acceder a VRL y VRH desde el puerto E.

- **Jumper JP3**: Conexión/desconexión del LED de pruebas. El diodo LED está conectado al bit 6 del puerto A del 68HC11 cuando el jumper JP3 está colocado. Si se quita, el LED quedará desconectado y el bit 6 del puerto A podrá ser utilizado para otros propósitos que no sean encender el led.

- **Jumper JP4**: Configuración del pulsador: RESET/IRQ. El jumper JP4 se trata de un jumper triple, que puede estar colocado en la zona de la izquierda, cortocircuitando los dos pines situados más a la izquierda, o puede estar colocado en la zona de la derecha, cortocircuitando los dos pines situados más a la derecha. Cuando el jumper está situado en la zona de la derecha (posición marcada como RST en la tarjeta), el pulsador actúa sobre la entrada de reset del 68HC11. Por tanto, en esta posición, cada vez que se apriete el pulsador, se realizará un reset hardware de la CT6811. Cuando el jumper está situado en la posición de la izquierda (marcada como IRQ), el pulsador se conecta a la entrada de interrupción enmascarable IRQ del 68HC11. De esta forma, podemos utilizar el pulsador en nuestros programas para realizar pruebas. Si el jumper está en esta posición, sólo se puede realizar un reset de la CT6811 a través del PC, utilizando el reset software.

- **Jumper JP5**: Configuración modo de funcionamiento de la CT6811: Entrenador o autónomo. Este es uno de los jumpers más importantes. Nos permiten decidir el funcionamiento de la CT6811: modo autónomo o modo entrenadora conectada al PC. Cuando el jumper está quitado, la tarjeta está configurada en modo entrenador. Todo el software habrá que cargarlo desde el PC. Sin embargo, cuando este jumper está puesto, al pulsar reset, la CT6811 comenzará a ejecutar el programa que tiene grabado en la EEPROM . No necesita del PC para cargar ningún software. El inconveniente de tenerlo puesto es que se pierden las comunicaciones serie, por eso hay veces que es mejor adaptar un pulsador a JP5 y arrancar el programa pulsando primero reset y luego el pulsador nuevo. Mediante este truco se recuperan las comunicaciones serie. Si se quiere evitar hacer todo esto hay que cambiar el modelo de microcontrolador, en lugar de utilizar el A1 hay que poner el E2. Para arrancar en modo autónomo configuraremos la tarjeta en modo *single_chip* sin necesidad de utilizar el jumper JP5.

Nota importante: El jumper JP5 lo que está haciendo es cortocircuitar o no los pines TX y RX del 68HC11 a través de una resistencia de pull-up. Por ello, si estamos trabajando en modo autónomo, NO FUNCIONARÁN LAS COMUNICACIONES CON EL PC. Hay veces que es mejor adaptar un pulsador a JP5 y arrancar el programa pulsando primero reset y luego el pulsador nuevo. Mediante este truco se recuperan las comunicaciones serie. Si se quiere evitar hacer todo esto hay que cambiar el modelo de microcontrolador, en lugar de utilizar el A1 hay que poner el E2. Ahora, para arrancar en modo autónomo se configura la tarjeta en modo *single_chip* mediante los switches y por lo tanto no se tiene que utilizar el jumper JP5.

- **Jumper JP6**: Conexión/desconexión del LVI. Este jumper conecta o desconecta el dispositivo MC34064 a la entrada de reset. Cuando está conectado, el dispositivo MC34064 previene que se pueda borrar el contenido de la memoria EEPROM del 68HC11. Cuando se detectan niveles de tensión por debajo de los niveles normales de funcionamiento del 68HC11, es decir por debajo de 4.5 voltios, el MC34064 realizará

JUMPER	FUNCIÓN	CONECTADO	QUITADO
JP1	Actuar sobre VRH	VRH = Vcc	VRH accesible desde el bus E
JP2	Actuar sobre VRL	VRL = GND	VRL accesible desde el bus E
JP3	Actuar sobre el LED	LED conectado a PA6	LED desconectado
JP4	Usos del pulsador	Izquierda: IRQ	Derecha: Reset
JP5	Modo funcionamiento	Modo Autónomo	Modo Entrenador
JP6	Actuar sobre LVI	LVI conectado	LVI desconectado
JP7	Reset Software On/Off	Arriba: Reset_Soft Off	Abajo: Reset_Soft On
JP8	Actuar sobre XIRQ	Modo normal	Sólo para EPROM del E9

Tabla B.1: Significado de los jumpers de la CT6811.

un reset del 68HC11. Esto ocurre al encender y apagar la CT6811. Sin embargo, puede ocurrir que mientras la CT6811 esté funcionando, aparezca un pico de caída de tensión y el micro sufra un reset. Por ello se da la opción de desconectar el MC34064 quitando el jumper JP6.

- **Jumper JP7**: Conexión/Desconexión del Reset Software. Ya se ha comentado la posibilidad de poder realizar un reset software (desde el PC) de la CT6811. Esta señal de reset se canaliza por el DTR del puerto serie, esto provoca que algunos programas de comunicaciones no puedan comunicarse con la CT6811. Por eso se ha añadido este jumper que permite separar el DTR de la señal de reset, de forma que se pierde el reset software pero se gana el poder utilizar otros programas de comunicaciones. Si el jumper está colocado abajo (on) estará activado el reset software, si está colocado arriba (off) estará desactivado.

- **Jumper JP8**: Liberación de la XIRQ. La tarjeta CT6811 se puede utilizar con otros modelos de la familia del microcontrolador 68hc11A1. Uno de ellos es el modelo 68hc11E9, que dispone de 12Kbytes de EPROM. Para poder programar esta memoria no volátil es necesario introducir 12v por la entrada XIRQ, que como se verá más adelante se encuentra en el bus de control. Para poder introducir esta tensión sin interferir los circuitos de la tarjeta CT6811es necesario quitar el jumper JP8. Esto sólo se hará cuando se vaya a programar la EPROM, una vez finalizada la programación se liberará la XIRQ y se volverá a poner el jumper.

B.5 Puertos de expansión

La tarjeta CT6811 dispone de 6 puertos de expansión (ver figura B.4). Cinco de estos puertos coinciden con los cinco puertos del 68HC11 y se han denominado igual: puerto A, puerto B, puerto C, puerto D y puerto E. El sexto puerto contiene otras señales de interés del 68HC11.

En la figura B.4 está representado uno de los conectores de los puertos, que está constituido por dos filas paralelas de 5 pines, a los cuales se podrán conectar cables planos de tipo bus de 10 bits. Se utilizan este tipo de cables por los siguientes motivos:

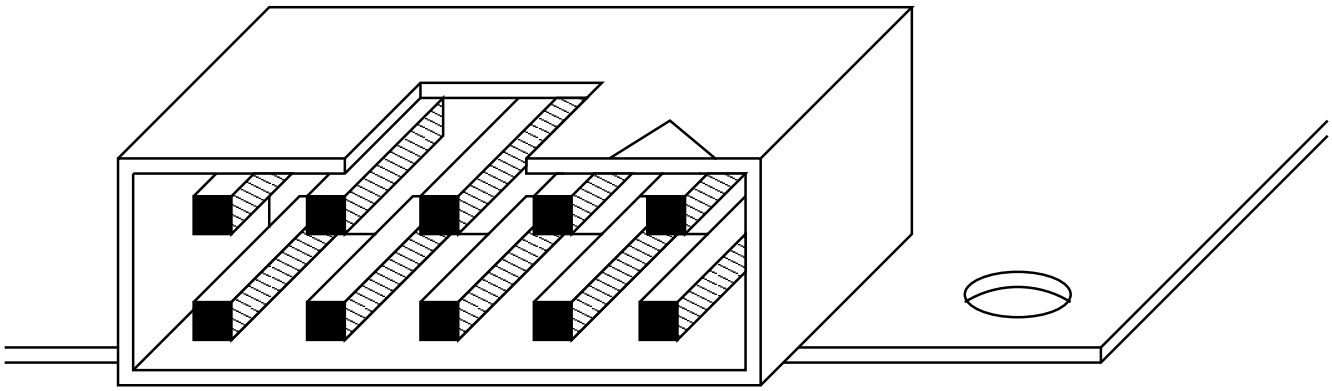


Figura B.4: Conector de los puertos de expansión.

Fiabilidad: Se trata de un cable muy resistente ya que al estar todo integrado en un sólo tipo de cable, se reduce el ruido y el peligro de cortocircuitos.

Multiplicidad: Es posible situar muchos conectores en el mismo cable, lo que lo hace especialmente útil para la conexión de varios periféricos al mismo puerto de expansión.

En la figura B.5 se detalla la correspondencia entre los pines del conector y los del microcontrolador. A continuación se describe cada puerto de la CT6811.

1. **Puerto A**: Este puerto está constituido por los 8 bits del puerto A del 68HC11 más un pin de VCC y otro pin de GND ambos para poder llevar alimentación a los periféricos de una forma cómoda.

2. **Puerto B**: Igual que el puerto A pero con el puerto B del 68HC11. El puerto B del 68HC11 sirve también como parte alta del bus de direcciones en caso de ampliación del micro con memoria externa. Cuando funciona en modo no expandido, el bit PB0 se corresponde con A8, PB1 con A9, ..., PB7 con A15.

3. **Puerto C**: A este puerto de expansión se lleva el puerto C del 68HC11. Este puerto funciona como un puerto normal de E/S cuando no hay conectada memoria externa. Cuando se conecta memoria externa, este puerto lleva el byte bajo del bus de direcciones multiplexado con el bus de datos. El bit PC0 se corresponde con AD0, PC1 con AD1,..., PC7 con AD7.

4. **Puerto D**: Este puerto está constituido por el puerto D del 68HC11 junto con las señales TXPC, RSTPC, RXPC y GND. El puerto D del 68HC11 dispone de 6 bits, que están compartidos con el SCI y el SPI del 68HC11: PD0/RX, PD1/TX, PD2/MISO, PD3/MOSI, PD4/SCK, PD5/SS. La señales TXPC y RXPC se utilizan para conectar a la CT6811 otros dispositivos que utilicen la norma de comunicaciones RS-232. Se denominan TXPC y RXPC porque están cortocircuitados con las señales TX y RX provenientes del PC. Si por ejemplo se quiere conectar una calculadora HP a la CT6811, habrá que hacerlo a través de las señales TXPC y RXPC. Por TXPC se transmite la información hacia la CT6811; por RXPC se recibe la información de la CT6811. La señal RSTPC está reservada para usos futuros. No conectar nada en este pin.

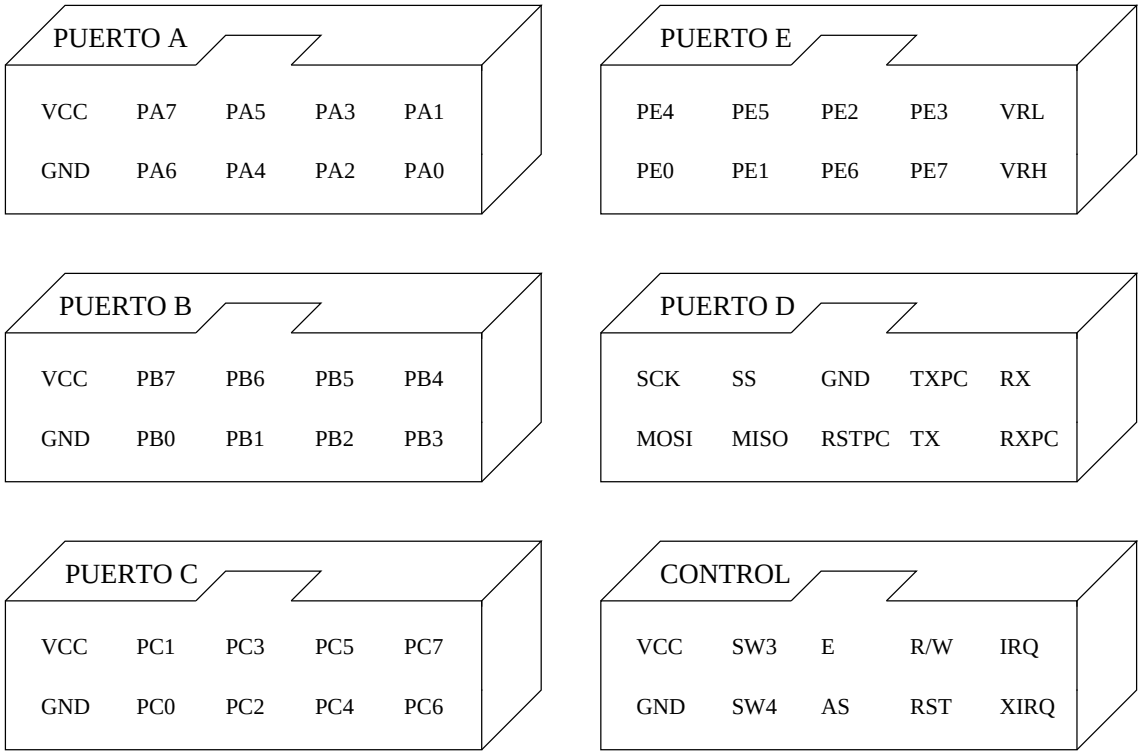


Figura B.5: Señales de los puertos de expansión. Los puertos se miran como se indica en la figura B.4.

5. **Puerto E**: Constituido por el puerto E del 68HC11. Este puerto en el 68HC11 tiene una doble función: puerto de entrada de 8 bits ó 8 canales de conversión A/D. Las señales VRL y VRH se utilizan para introducir las señales de referencia alta y baja del conversor. Si los jumpers JP1 y JP2 están colocados, por estos podemos obtener la alimentación: VRH=VCC, VRL=GND. Si no están colocados estos jumpers, podremos introducir las tensiones de referencia necesarias para nuestra aplicación.

6. **Control**: Por este puerto se han 'recopilado' una serie de señales del 68HC11 para el control. Por los pines SW3 y SW4 se sacan las señales correspondientes al estado de los switches de usuario 3 y 4.

B.6 Alimentación

La CT6811 tiene una tensión nominal de alimentación de 5 voltios, aunque se puede alimentar sin ningún problema entre 4.5 y 5.5 voltios. Dispone de dos conectores diferentes de alimentación para alimentar la placa bien por medio de un transformador entre 4.5 y 5.5 voltios conectado a la red o bien directamente a través de pilas o baterías.

La alimentación a través del transformador se emplea cuando se está utilizando la CT6811 como una entrenadora. En este caso, puesto que la entrenadora tiene que estar junto al PC, lo mejor es alimentar mediante el transformador. Sin embargo, cuando está funcionando en modo autónomo, es más cómodo utilizar una batería independiente.

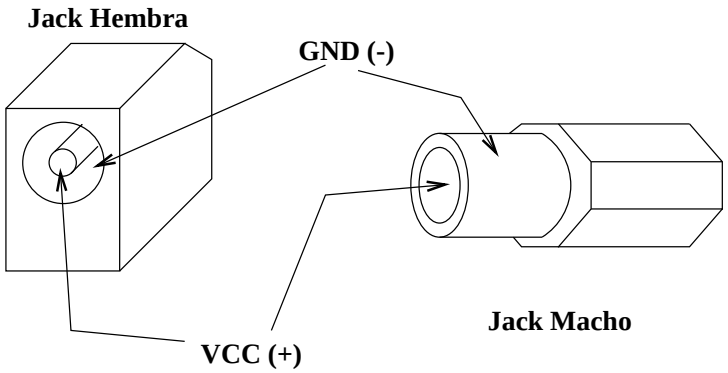


Figura B.6: Jack de alimentación.

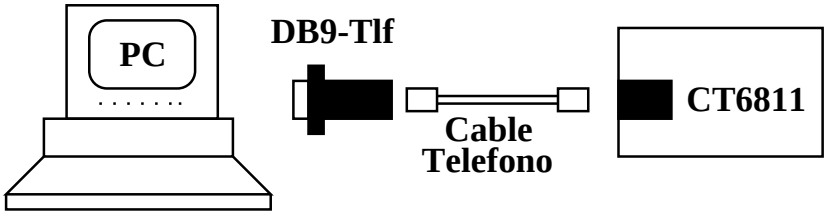


Figura B.7: Conexión PC - CT6811.

En la figura B.6 se ha representado el conector hembra de alimentación y el jack macho cilíndrico. Es importante hacer notar que **la parte exterior del jack macho debe ser GND** (negativa), mientras que **la parte interior debe ser VCC** (positiva). En la mayoría de los transformadores comerciales, la polaridad se puede invertir cambiando un switch. Antes de conectar un transformador asegurarse de que la polaridad es la correcta: Masa por la parte exterior(-), alimentación (+) por la parte interior.

Si se desea utilizar las clemas de alimentación la polaridad se encuentra indicada en la CT6811. La etiqueta Vcc significa que por ahí hay que introducir la tensión positiva (+), la etiqueta GND indica que por ahí hay que introducir la negativa o masa (-).

Los terminales de VCC de las bornas y el conector jack se encuentran cortocircuitados, lo mismo que los terminales de GND. Por ello, **nunca se debe alimentar la CT6811 mediante un transformador y baterías a la vez**. O se utiliza el transformador, o se utilizan las bornas.

B.7 Conexión al PC

La conexión al PC se realiza por medio de dos elementos: un cable de teléfono de 4 hilos y un conector DB9-TLF que tiene una entrada para el cable de teléfono por un lado y por el otro lado se conecta directamente a un puerto serie del PC.

El cable de teléfono empleado es de 4 hilos, pero no todos los cables sirven. Existen dos tipos de cables según cómo estén situados los conectores de los extremos (ver figura). Un cable es válido cuando las pestañas de los conectores de teléfono están situadas en el mismo sentido, es decir, o ambas coinciden con la línea marcada en la superficie del cable, o ambas no lo hacen, pero nunca una si y otra no.

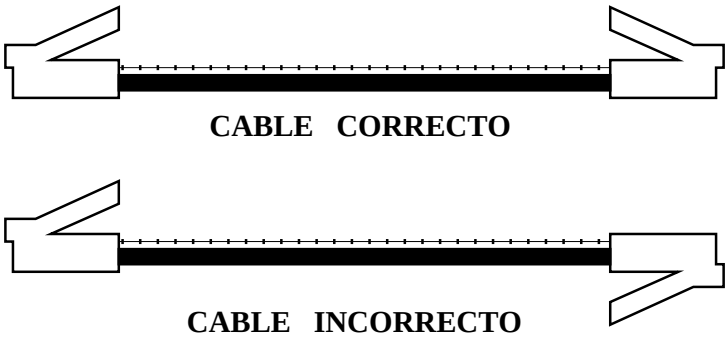


Figura B.8: Cable de teléfono para unir la CT6811 con el PC.

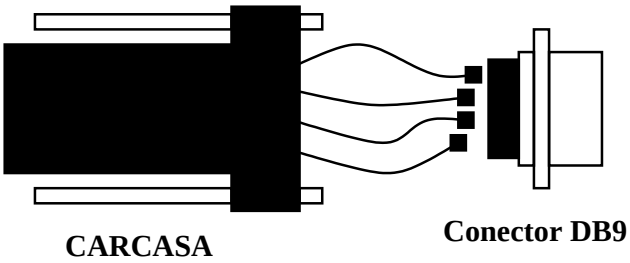


Figura B.9: Conector DB9 - Teléfono.

El cable utilizado es de 4 hilos, por ellos viajarán las señales: TX, RX, GND y DTR. La señal DTR es la utilizada para realizar el reset software de la tarjeta CT6811. El conector DB9-TLF tiene por un lado un conector hembra DB9 y por el otro una hembra de teléfono de 4 vías. Estos conectores se venden desmontados y en su interior hay cuatro cables que tendrán que introducirse en los pines correcto de la salida DB9 del mismo. Generalmente los colores son los indicados en la figura B.10. Pero lo que realmente interesa es realizar las conexiones internas correctamente. Si al desmontar el conector los colores son iguales se puede realizar la asignación tal y cómo indica la figura. Por ejemplo el cable rojo se insertará en el pin 5 del conector DB9. Si los colores no son iguales se tiene que revisar la asignación pin a pin. Por ejemplo, un cable azul que sale del pin 6 de la carcasa (conector teléfono) se tiene que unir con el PIN 2 del conector DB9 (PC).

COLOR	PIN DB9	CARCASA (CABLE)	FUNCIÓN
NEGRO	2	6	RX PC
AMARILLO	3	1	TX PC
VERDE	4	5	RESET
ROJO	5	2	GND

Tabla B.2: Significado de las señales del conector.

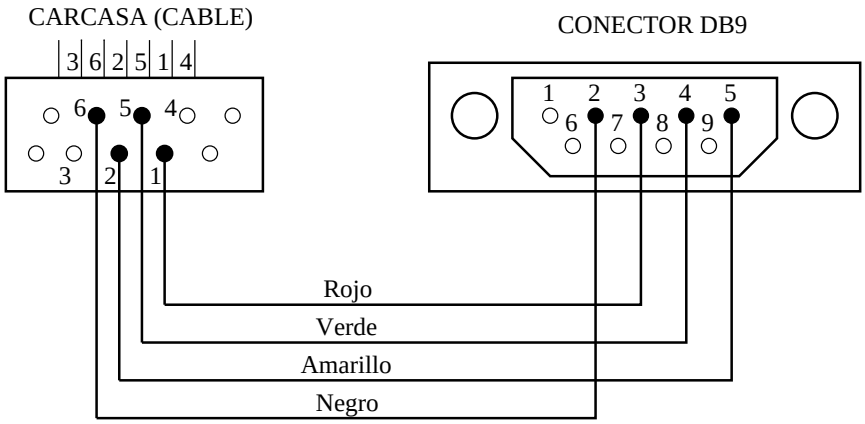


Figura B.10: Unión cable - PC.

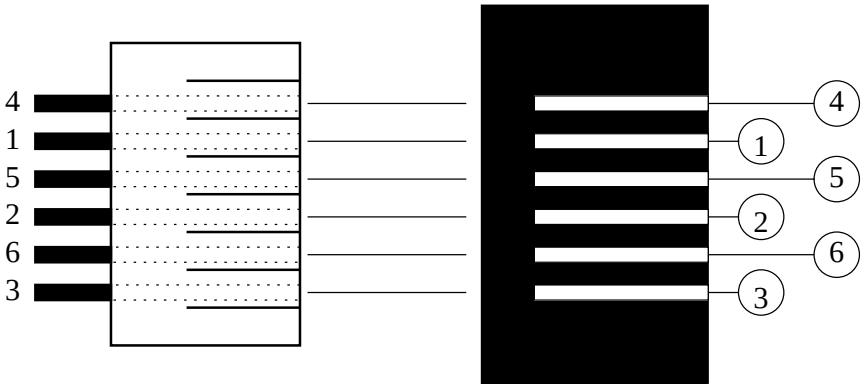


Figura B.11: Unión cable - CT6811.

B.8 Pruebas de funcionamiento

B.8.1 Probando la CT6811 en modo autónomo.

La CT6811 se entrega con un pequeño programa de prueba para comprobar que todo funciona adecuadamente. El programa simplemente hace parpadear el led de la CT6811 intermitentemente. Para ejecutar el programa de pruebas siga los siguientes pasos:

- Coloque los jumpers JP5, JP3. Situar el jumper JP4 en la posición más de la derecha, indicada en la CT6811 como RST. Los demás jumpers pueden estar colocados o quitados, es indiferente.
- Sitúe los switches 1 y 2 hacia abajo. Los switches 3 y 4 pueden estar en cualquier posición.
- Alimente la CT6811, bien con un transformador, bien con pilas.
- Apriete el pulsador para hacer un reset.

Se tendrá que encender intermitentemente el led de la CT6811. Si no funciona correctamente, revise los puntos anteriores y vuélvalo a intentar. Si sigue sin funcionar, puede ser que la EEPROM se haya desprogramado. Haga las pruebas que se indican en el apartado siguiente.

Si todo ha funcionado correctamente, la CT6811 ha funcionado en modo autónomo. En el siguiente apartado se va a probar la CT6811 en modo entrenador.

B.8.2 Probando la CT6811 en modo entrenador.

En este apartado se va a cargar un programa en la CT6811 desde el PC utilizando el *downmcu*. Para ello, primero debe configurar la CT6811 adecuadamente:

- Coloque el jumper JP3, JP8 y el JP4 sitúelo en la posición más a la derecha indicada por la etiqueta RST. El jumper JP5 debe estar quitado y el JP7 colocado en la posición de ON (abajo). Los demás jumpers se pueden encontrar conectados o desconectados.
- Sitúe los switches 1 y 2 hacia abajo. Los switches 3 y 4 pueden estar en cualquier posición.
- Conecte la CT6811 al puerto serie del ordenador (COM1 ó COM2) utilizando el cable proporcionado.
- Alimente la CT6811 por medio de un transformador o por medio de pilas

Ahora la CT6811 se encuentra dispuesta para funcionar en modo entrenador. En el PC debe disponer de los siguientes ficheros: el programa *downmcu*, que probablemente esté en */usr/local/bin*[1] y el *ledp.s19* en el directorio de trabajo. Estos son los mínimos ficheros necesarios para realizar estas pruebas.

[1] Localización por defecto en las máquinas Linux, en MSDOS estará en el path o en el directorio de trabajo.

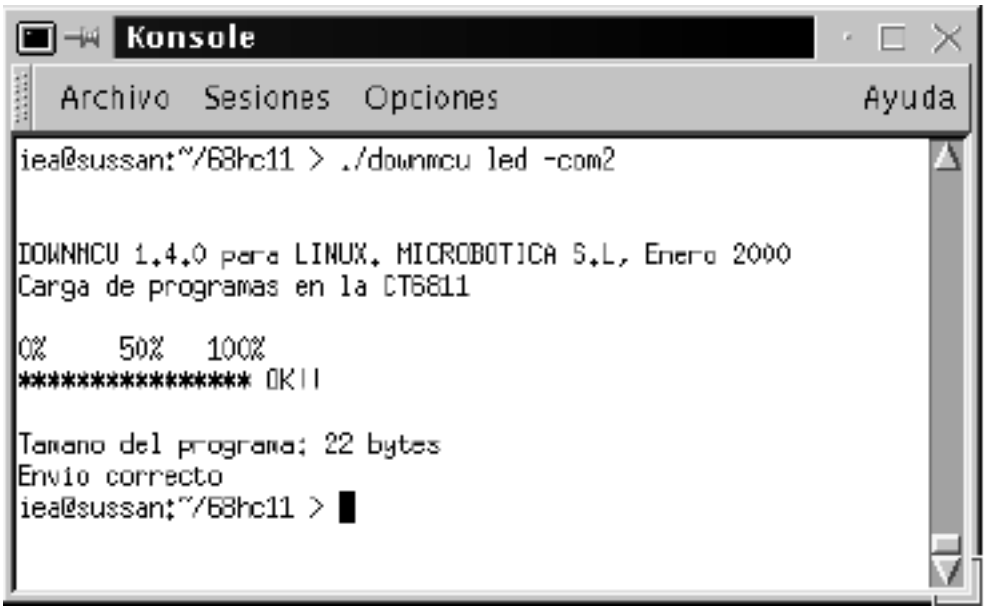


Figura B.12: Carga del programa LEDP.S19 en la CT6811 utilizando el *downmcu*.

Primero se va a cargar el programa ledp.s19 en la CT6811. Este programa simplemente hace parpadear el led de la tarjeta. La extensión .S19 indica que se trata de un programa ejecutable por el 68HC11. El programa en ensamblador, por si lo quiere ver, se encuentra en el fichero LEDP.ASM. En la figura B.12 se muestra lo que hay que teclear y los resultados obtenidos. Se ha supuesto que la CT6811 se encuentra conectada en el puerto COM2. Si se encuentra en el COM1, cambie el parámetro -com2 por -com1 (¡Siempre en minúsculas!).

El programa *downmcu* implementa el reset software automáticamente, por eso el jumper JP7 deberá estar puesto en la posición indicada con *ON* en la CT6811. Si se coloca al contrario se desactiva el reset software y cada vez que los programas pidan hacer un reset se deberá pulsar el botón situado en la CT6811 y para que éste funcione el jumper JP4 deberá estar situado a la derecha (posición RST).

Si el proceso de carga ha ido bien se apreciará que el LED de la CT6811 está parpadeando y significa que se ha podido enviar un programa a la memoria RAM correctamente. En el siguiente capítulo se escribirán los pasos necesarios para poder grabar los programas en la EEPROM y obtener sistemas autónomos.

B.9 Desarrollo de programas para la CT6811

En este apartado se van a dar unos ejemplos de programación de la CT6811. Para ello se va a utilizar un software concreto: el ensamblador AS11 de Motorola y los programas *downmcu* y *ctdialog*.

Se utilizan las versiones para LINUX, pero se recuerda que también se dispone del software para MSDOS, el cual se maneja de forma similar al explicado aquí.

B.9.1 Filosofía de trabajo

La filosofía empleada para el manejo de la CT6811 es la siguiente. En el PC se programan las aplicaciones para la CT6811 en ensamblador del 68HC11. Estos programas fuente tienen extensión .ASM. Utilizando el ensamblador FREEWARE AS11 de Motorola, los programas se compilan, obteniéndose archivos ejecutables con extensión .S19. Estos son los que se envían a la CT6811.

Para enviar programas a la CT6811 se pueden utilizar varios programas. En los ejemplos de más adelante se utiliza el programa DOWNMCU v.10. Una vez que se ha enviado el programa, el 68HC11 comienza su ejecución. Se puede cargar programas en la CT6811 tantas veces como se quiera. Los programas así cargados son almacenados en la RAM interna del 68HC11.

Una vez que se tiene la aplicación depurada, se graba en la memoria EEPROM del 68HC11 para que se quede permanentemente. Ahora la CT6811 no necesita del PC. Funciona en modo autónomo. Para grabar programas en la EEPROM interna se pueden utilizar varios programas: Pc-bug, ctdialog... En esta sección se utilizará el programa *ctdialog*. Este programa, además de permitir grabar la EEPROM , permite ver el contenido de la memoria del 68HC11, cambiar valores de la memoria, desensamblar ...

B.9.2 Un ejemplo completo

Se presenta un programa fuente que se va a compilar para después ser cargado en la CT6811 y finalmente grabado en la EEPROM . El programa hace parpadear un led y sus fuentes se puede ver en la figura B.13.

Lo primero que hay que hacer es escribir el programa con algún editor de textos, para luego grabarlo con la extensión ASM (que indica que es el código fuente); en este caso el programa se llamará *LED.ASM*. El siguiente paso es compilar el programa para obtener el archivo LED.S19, que es el fichero ejecutable y por lo tanto entendible por el microcontrolador. Para ello se utiliza el ensamblador AS11 de Motorola de la siguiente forma:

```
> as11 led.asm
```

Se obtiene el archivo *led.s19* que se trata de un archivo de texto y por tanto se puede editar (¡Pero no se debe modificar!).

Para probar si funciona el programa se cargará directamente en la CT6811, para ello se utiliza el programa *downmcu* de la forma explicada en la sección anterior. En la figura B.14 se muestran los comandos para compilar el fichero y para luego enviarlo a la CT6811. Se ha supuesto que la CT6811 se encuentra conectada en el puerto serie COM2, si se quiere utilizar el puerto COM1 basta con cambiar el parámetro -com2 por -com1 (¡en minúsculas!). En cuanto se recibe el mensaje de 'ENVIO CORRECTO' el programa ya se estará ejecutando en el 68HC11 y se podrá ver cómo parpadea el LED.

Hasta ahora, todos los programas enviados al 68HC11 se han almacenado en la RAM interna del micro, esto significa que al quitar la alimentación estos se perderán. A pesar de que se pueden cargar todas las veces que se quiera, es necesario de disponer de algún mecanismo para poder dejarlos permanentemente grabados la tarjeta independientemente del estado de la alimentación. Esto se consigue enviando los programas a la memoria EEPROM interna del 68hc11, la cual una vez grabada mantiene los datos aunque se apague

led.asm

Archivo Editar Marcadores Opciones Ayuda

```
*.....................................
*. LED                          2000 .
*.....................................
*. Programa de ejenplo para la CT6811 .
*.....................................

        ORG $0

init    LDAA $1000
        EORA #$40
        STAA $1000

        LDY #$FFFF
bucle   DEY
        CPY #$0
        BNE bucle
        BRA init

        END
```

INS Linea: 6 Col: 1

Figura B.13: Listado del programa LED.ASM

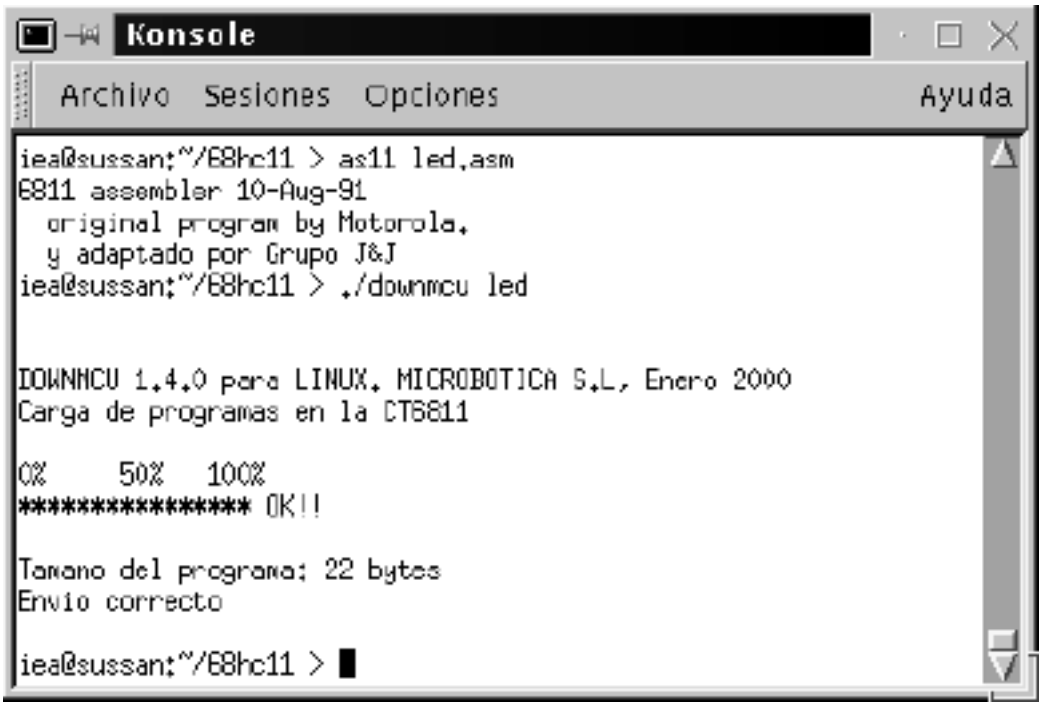


Figura B.14: Proceso para compilar y enviar el programa a la CT6811.

la tarjeta. Para poder enviar los programas a dicha memoria hay que utilizar un software distinto del *downmcu*, en concreto el *ctdialog*.

Antes de nada, hay que modificar el programa fuente para indicar que se quiere situar en la memoria EEPROM . Si el programa a grabar en la EEPROM utilizase variables, habría que situar las variables en la RAM interna y dejar sólo el código en la EEPROM . Como el programa *led* no tiene variables, no hay que preocuparse, pero si que hay que cambiar la directiva ORG, que indica el comienzo del programa. En vez de comenzar en la dirección $0000, ahora deberá comenzar en la dirección $b600, que es el comienzo de la memoria EEPROM en el modelo de microcontrolador MC68HC11A1[2]. El nuevo programa creado se denomina *lede.asm* para diferenciarlo del anterior. Una vez compilado este programa, de forma similar a como se ha hecho con el programa *led.asm*, se obtiene el fichero *lede.s19*, que será el que se utilice para grabar la EEPROM .

El programa que se va a emplear es el *ctdialog* (figura B.15). El proceso de carga de este programa depende de la plataforma sobre la que se ejecuta el software. En el caso de LINUX es suficiente con llamar al programa de la siguiente forma:

> ctdialog -com2

Al igual que antes si en lugar de estar conectada la CT6811 al segundo puerto serie estuviera en el primero se pondría como parámetro *-com1*. En el caso de MSDOS hay que hacer un proceso más largo; Primero hay que enviar a la CT6811 el programa CTSERVER y después se ejecuta el programa CTDIALOG.EXE.

> downmcu ctserver -com2

[2]Si el modelo de microcontrolador es el MC68HC11E2 hay que poner ORG $F800.

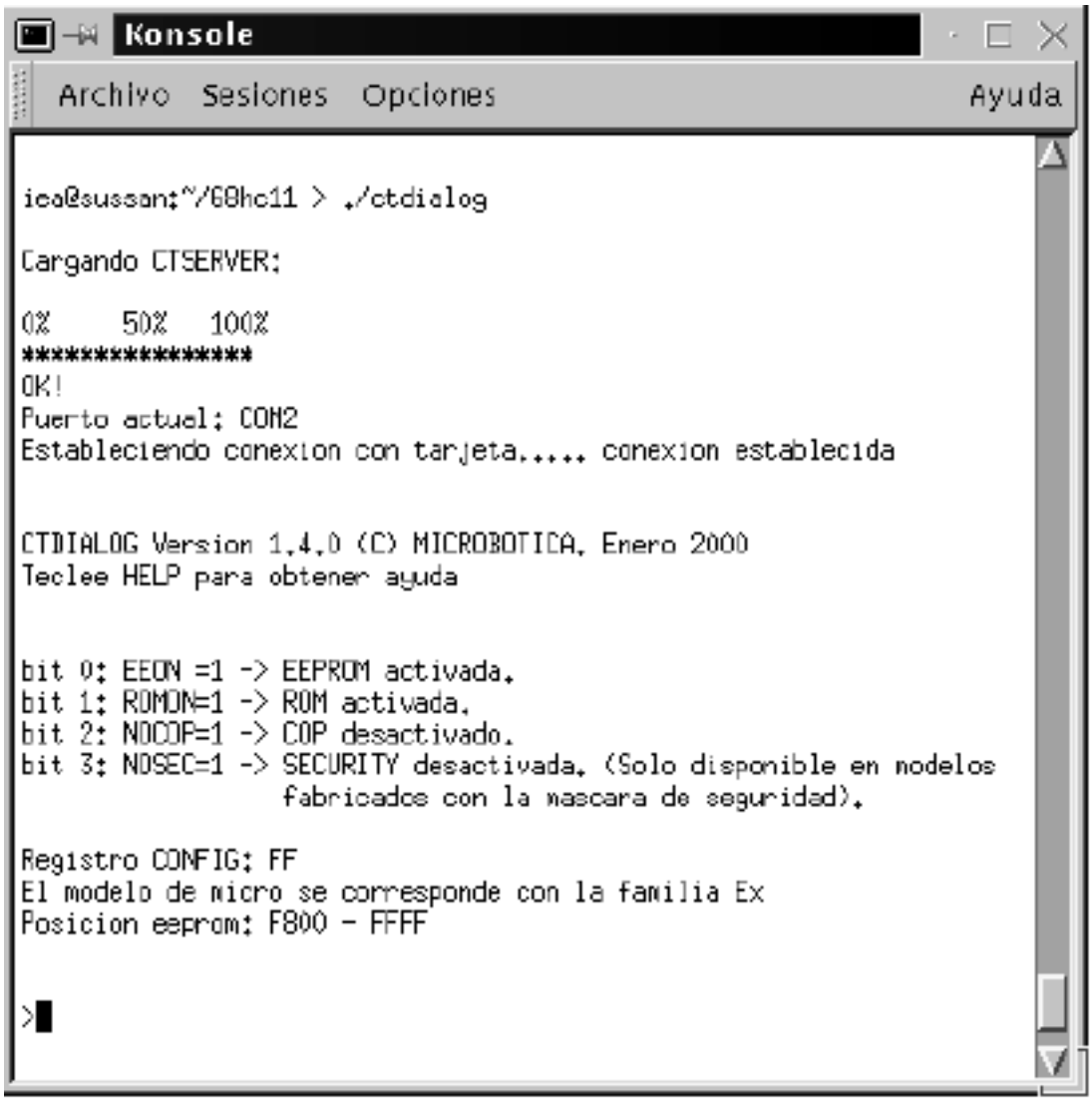


Figura B.15: Programa CTDIALOG ejecutado en una CT6811 con un microcontrolador E2.

> ctdialog -com2

En ambos casos el programa *ctdialog* permite grabar la EEPROM del 68HC11, pero también sirve para ver el contenido de la memoria, modificarla, desensamblar , ejecutar programas, reconfigurar el micro, etc.

Para grabar el programa *lede.s19* en la EEPROM hay que utilizar el comando ***'eeprom'*** seguido del nombre del fichero:

>**eeprom** lede

Para probar el programa, se puede configurar la CT6811 en modo autónomo (conectando el jumper JP5) y hacer un reset. También es posible realizar un 'salto' a la EEPROM desde el *ctdialog*, para ello se utiliza el comando *'g'* seguido de la dirección a donde se quiere saltar:

>**g** b600

Al realizar el salto, el 68HC11 comienza a ejecutar el programa almacenado en la EEPROM . El servidor que estaba en la RAM interna se deja de ejecutar y por tanto se pierde la conexión del *ctdialog* con la CT6811. Para volver a ejecutar el programa habrá que salir utilizando el comando *'**quit**'*, volver a cargar el servidor en caso de estar en MSDOS (bajo LINUX no hace falta) y ejecutar el *ctdialog* otra vez.

El *ctdialog* dispone de otros comandos y entre ellos una ayuda en línea que muestra un resumen de todos:

>**help**

Otro comando interesante es *'**ms**'* que permite escribir un byte (segundo argumento) en una posición de memoria (primer argumento). Por ejemplo, la instrucción siguiente encenderá el LED de la CT6811.

>**ms** 1000 40

El valor 1000 es la dirección en hexadecimal del Puerto A y el valor 40 es el valor en hexadecimal del byte que hace que el bit PA6 se ponga a uno encendiendo LED. Para apagarlo hay que poner en la dirección 1000 el valor 0, es decir:

>**ms** 1000 0

Por último hay que hacer una puntualización sobre el uso del *ctdialog* bajo MSDOS y con una tarjeta que incorpore el microcontrolador 68hc11E2 en lugar del A1. En este caso, antes de grabar la EEPROM hay que desprotegerla, para ello teclear:

>**ms** 1035 0

>**eeprom** 'programa'

La primera instrucción desprotege la EEPROM y la segunda la graba. Si lo que se quiere es borrarla en lugar de utilizar *'**eeprom**'* hay que poner *'**bulk**'*.

B.10 68HC11 y comunicaciones serie: El programa MCBOOT.

El programa ***mcboot*** es un programa de comunicaciones para comunicarse con la tarjeta CT6811. Además de funcionar como terminal de comunicaciones también permite cargar programas en la RAM interna del 68HC11.

En la depuración de programas y pruebas de periféricos es muy útil poder contar con el teclado y pantalla del PC. Utilizando programas de comunicaciones serie en el 68HC11 se va a poder utilizar la pantalla del PC para representar datos y el teclado del PC para enviar datos al 68HC11.

B.10.1 Programa de ejemplo scihola.asm

A continuación se presenta un pequeño programa para probar las comunicaciones serie. El programa *scihola.asm*, cuyo código está en la figura B.16, simplemente espera a que se pulse una tecla en el PC y responde enviando la cadena "HOLA COMO ESTAS...". Existen dos subrutinas de comunicaciones serie: **enviar** para enviar un carácter por el puerto serie y **leer_car** para esperar a que venga un carácter por el puerto serie. Estas rutinas están relacionadas con el puerto serie del 68HC11 y no se explican en este manual. El bucle principal está constituido por una llamada a la subrutina de *leer_car* para que el micro se quede esperando recibir un carácter por el puerto serie. Una vez recibido el carácter, se envía por el puerto serie la cadena especificada utilizando la función **send_cad.**

Este programa se puede enviar a la CT6811 utilizando el programa de comunicaciones *mcboot*. Antes de cargar el programa hay que seleccionar el puerto en el que está conectada la CT6811, el predeterminado es el COM2, pero pulsado la tecla **F4** se puede cambiar a otro. **Asegúrese de que NO está activado el reset software. Para ello pulse F7 hasta que aparezca en la pantalla DTR OFF, ya que siempre que esté DTR ON la CT6811 estará *reseteada* y no se podrán cargar programas.**

Para cargar un programa pulse la tecla **F1** e introduzca el nombre del fichero .S19 a cargar. En este caso será el programa *scihola*, el cual envía al PC la cadena “hola como estas...” cada vez que recibe por el puerto serie la tecla pulsada en el PC.

B.10.2 Programa de ejemplo menú

Con el siguiente ejemplo se pretende mostrar la posibilidad de realizar programas de prueba que ofrezcan diferentes opciones al usuario. En concreto, el programa *menu.asm* presenta en pantalla dos opciones. Según la opción que elija el usuario el micro cambiará el estado del led o volverá a imprimir el menú. A partir de este programa modelo el usuario puede ir practicando y realizando otros más complejos.

```
* +---------------------------------------------+
* | MENU                        MICROBOTICA, 2000 |
* |---------------------------------------------|
* |  Programa ejemplo para ser ejecutado en la   |
* | tarjeta CT6811. Este programa se debe cargar |
* |en la RAM interna del 6811.                   |
* |                                              |
* | Ejemplo de como manejar un menu de opciones  |
* |para programas interactivos con el usuario    |
* +---------------------------------------------+

CR      equ  13    * Retorno de carro
LF      equ  10    * Avance de linea

* Registros del SCI

BAUD    equ  $2B
```

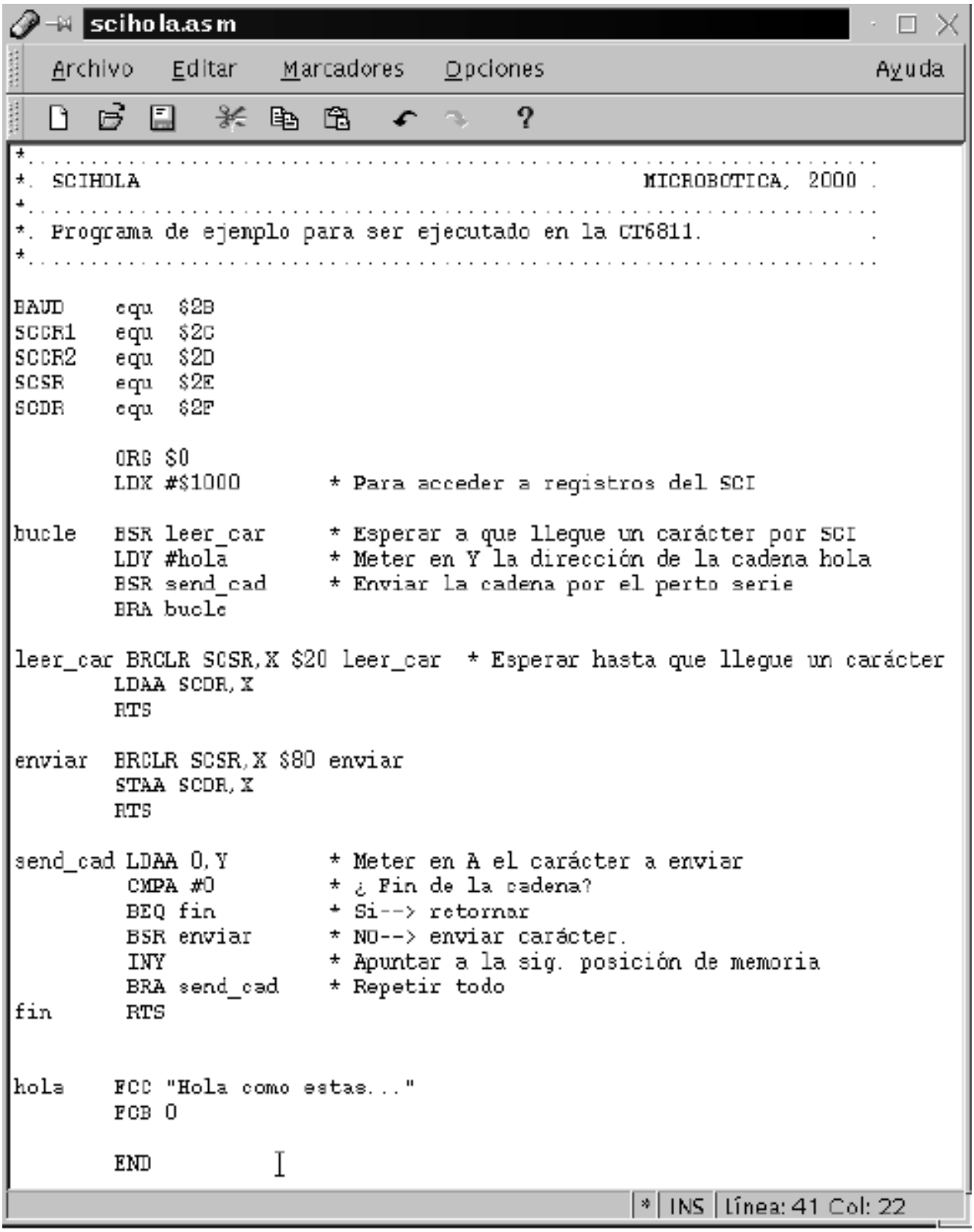


Figura B.16: Código del programa SCIHOLA.

```
SCCR1   equ  $2C
SCCR2   equ  $2D
SCSR    equ  $2E
SCDR    equ  $2F

        ORG $0        * programa para la RAM
        LDX #$1000    * acceder a registros

bucle   LDY #menu
        BSR send_cad   * Sacar menu
wait    BSR leer_car   * Leer tecla
        CMPA #'1'
        BEQ opcion1    * Tecla '1'-> Opcion 1
        CMPA #'2'
        BEQ opcion2    * Tecla '2'-> Opcion 2
        BRA wait

opcion1 LDAA $1000
        EORA #$40      * Cambiar estado bit 6 puerto A
        STAA $1000
        BRA wait

opcion2 BRA bucle      * Volver a sacar el menu


*+-----------------------------------------------------+
*| Rutina par leer un caracter del puerto serie (SCI)  |
*| La rutina espera hasta que llegue algun caracter    |
*| ENTRADAS:Ninguna.                                   |
*| SALIDAS: El acumulador A contiene caracter recibido |
*+-----------------------------------------------------+
leer_car BRCLR SCSR,X $10 leer_car   * esperar caracter
         LDAA SCDR,X
         RTS

*+-------------------------------------------------+
*| Enviar un caracter por el puerto serie (SCI)    |
*| ENTRADAS: Acumulador A contiene caracter a enviar |
*| SALIDAS:Ninguna.                                |
*+-------------------------------------------------+
enviar  BRCLR SCSR,X $80 enviar
        STAA SCDR,X
        RTS

*+-------------------------------------------------+
*| Enviar una cadena de caracteres por el puerto SCI |
```

```
*| La cadena debe terminar con el caracter 0         |
*| ENTRADAS: Y contiene direccion cadena a enviar    |
*| SALIDAS: Acumulador A contiene caracter recibido  |
*+---------------------------------------------------+
send_cad LDAA 0,Y       * Meter en A caracter a enviar
         CMPA #0        * ¿Fin de la cadena?
         BEQ fin        * Si--> retornar
         BSR enviar     * .NO--> enviar caracter.
         INY            * Apuntar a la sig. posicion
         BRA send_cad   * Repetir todo
fin      RTS


*+--------+
*| DATOS  |
*+--------+
menu    FCB CR,LF,LF
        FCC "       MENU DE OPCIONES"
        FCB CR,LF
        FCC "       ================"
        FCB CR,LF,LF
        FCC "  1.- Cambiar de estado el LED"
        FCB CR,LF
        FCC "  2.- Sacar este menu"
        FCB CR,LF
        FCC "  Opcion: "
        FCB 0

        END
```

B.10.3 El MCBOOT y la RAM externa

El *mcboot* tiene otras funciones hasta ahora no mencionadas ya que están desactivadas en la versión de Linux, pero no en la de MSDOS, por eso conviene mencionarlas:

F2 Cargar programas en la RAM externa. Para utilizarla se debe disponer de la ampliación de memoria externa para la CT6811 y además disponer en el directorio donde se encuentre el *mcboot* el cargador *loader.s19*. La utilización de esta opción es igual que **F1**.

F3 GAIA. Esta opción necesita las condiciones anteriores, pero en lugar de enviar cualquier programa carga un software especial que permite ver los registros internos, volcar datos, poner *break points*, etc. El *ctdialog* ha venido a sustituir esta opción del *mcboot*.

F6 Volcar programas. Mediante esta opción se puede volcar un fichero de texto por el puerto serie indicado. La diferencia con **F1** está en que **F6** envía todo el fichero, mien-

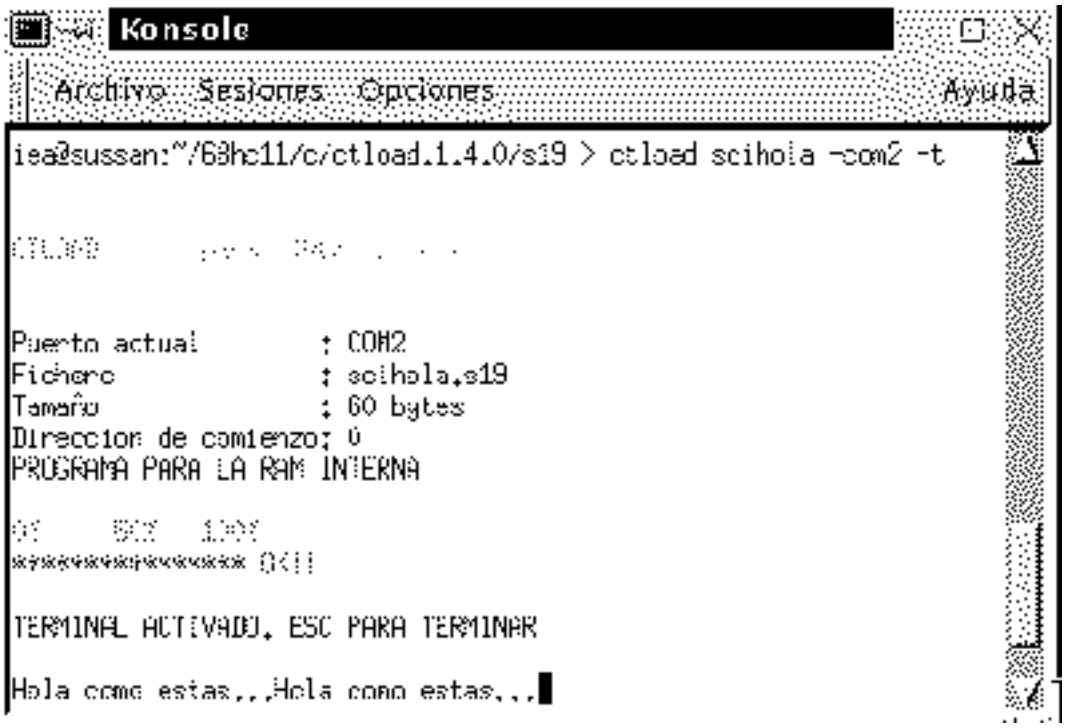


Figura B.17: Programa *ctload* actuando de terminal.

tras que **F1** sólo la parte correspondiente al código, dejando sin enviar las cabeceras y los bytes de control.

En Linux, para enviar programas a la RAM externa se puede utilizar el *ctload* (ver figura B.17), que funciona igual que el *downmcu*, solo que es capaz de distinguir si el programa es para la RAM interna o externa, cargándolo donde corresponda. Además este programa tiene la opción '**-t**' que permite después de cargar el programa dejar la consola como un terminal. Es decir, todo lo que se reciba se verá en pantalla y todo lo que se teclee se enviará por el puerto serie.

B.11 Utilización de la CT6811 con la familia 68HC11

La tarjeta CT6811 incorpora como microcontrolador principal el MC68HC11A1, pero este puede ser sustituido por otros de la familia, en concreto por los modelos de las familias A, D y E. En la tabla B.3 se muestran los modelos que los autores han utilizado en alguna aplicación de la CT6811.

Todos estos modelos tienen los mismos recursos internos, 8 conversores, SPI, SCI, capturadores, comparadores, etc, únicamente se diferencian en el tipo y cantidad de la memoria disponible.

La tarjeta CT6811, como ya se ha dicho, trae de serie el MC68HC11A1. Es un microcontrolador fácilmente obtenible y al tener EEPROM interna se puede utilizar en modo autónomo, aunque para ello sea necesario arrancar en BOOTSTRAP con el jumper JP5 puesto. Es decir se perderán las comunicaciones serie con el PC. Un truco para resolver esto es arrancar el sistema con un pulsador conectado en el lugar del jumper, o mejor aún,

MODELO	RAM	ROM	EPROM	EEPROM
MC68HC11A8	256	8K	0	512
MC68HC11A1	256	0	0	512
MC68HC11A0	256	0	0	0
MC68HC811E2	256	0	0	2048
MC68HC11E1	512	0	0	512
MC68HC711E9	512	0	12K	512

Tabla B.3: Modelos de microcontroladores compatibles con la CT6811.

con un pequeño sistema que cortocircuite dicho jumper justo después de hacer un reset para acto seguido liberar la conexión.

Un inconveniente no ajeno al programador es la capacidad de memoria disponible en el sistema. Inicialmente la CT6811 tiene 256 bytes de RAM y 512 de EEPROM, aunque existe un módulo de ampliación de memoria que añade 32K de RAM y 32 de EEPROM externas. El inconveniente de la ampliación es la pérdida del puerto B y del puerto C, es decir un gran número de entradas y salidas. Por eso los autores aconsejan sustituir el modelo de microcontrolador, se ahorra espacio, no se pierden puertos y la fiabilidad será mayor, el sistema se reduce en componentes. Además la tarjeta CT6811 ha sido diseñada para servir de entrenadora primero y luego de producto final. Otras tarjetas, como TRANTOR, también desarrollada por el GRUPO J&J[3] incorporan memoria RAM y EEPROM externas. TRANTOR ha sido pensada principalmente como entrenadora de software. Una vez desarrollado el software se le encargaría a Motorola un MC68HC11A8 con el programa grabado en ROM. Esto tiene el inconveniente del precio, por eso una opción bastante buena es utilizar microcontroladores MC68HC711E9 o MC68HC811E2.

El MC68HC811E2 tiene 2K de memoria EEPROM situados desde la posición $F800 hasta la $FFFF, con lo cual los vectores de interrupción están en EEPROM, incluido el de reset. Esta característica hace que el modo autónomo sea mucho mejor. Ahora se tienen dos posibilidades para acceder a dicho modo. La primera es hacer un reset en BOOTSTRAP y utilizar el jumper JP5, la segunda es configurar el vector de interrupción del reset para que apunte al principio del programa en EEPROM. Se configura la CT6811 para arrancar en modo Single Chip y al hacer un reset el programa grabado en EEPROM empezará a funcionar independientemente de la situación del jumper JP5, es decir se conservan las comunicaciones serie con el PC.

El MC68HC711E9 tiene otras características interesantes. La primera es que el tamaño de la RAM y de la EEPROM coincide. Esto es útil a la hora de probar software. Con los modelos anteriores los programas de la EEPROM podían ser mayores que los de la RAM, por lo tanto no se podían probar en ella. La segunda característica es que incluye 12K de EPROM situadas en las posiciones $D000-$FFFF. Ahora también se puede inicializar el vector de reset, por lo que todo lo referente al modo autónomo del MC68HC811E2 sigue siendo válido salvo una pequeña diferencia. En este caso la memoria es EPROM y sólo se puede grabar una vez, a no ser que se disponga del modelo con ventana (OTP) para poder borrarla. Es útil disponer del modelo con ventana para desarrollar software y luego

[3]El Grupo J&J es como se hacían llamar los autores de la CT6811 antes de formar parte de la empresa Microbótica.

grabar las versiones definitivas en modelos sin ventana. Otra solución es utilizar tarjetas con memoria externa y cuando el programa este terminado traspasarlo a los modelos E9. Las soluciones son variadas y la más adecuada dependerá del tipo, tiempo, coste, etc... de la aplicación o desarrollo a implementar.

Por último mencionar la utilidad del MC68HC11A0. En este caso sólo se puede usar la memoria RAM, en concreto los 256 bytes. ¿ Por qué no tiene EEPROM?, la respuesta está en que son modelos del MC68HC11A1 que tienen estropeada dicha memoria. En lugar de retirarlos del mercado se les desactiva la misma y se vende como un MC68HC11A0. Los autores los han utilizado en tarjetas insertadas en el PC, donde el microcontrolador era programado desde los programas de aplicación del PC. Según la aplicación se reprogramaba en el acto la tarjeta, a modo de periférico programable. Al tener la memoria del PC no era imprescindible la EEPROM y debido a que al principio el coste de este modelo era significativamente menor que el del resto, se podía prescindir de un modelo con EEPROM. Actualmente no hay tanta diferencia y lo anterior se puede cuestionar, sin olvidar que la CT6811 con un MC68HC11A0 solamente se podría usar en modo entrenadora, no como producto final.

Para más información sobre la familia del Motorola MC68HC11 se recomienda acudir al manual de referencia técnica de Motorola. 'MC68HC11 REFERENCE MANUAL'.

B.12 Características del 68HC11

1. En la figura B.18 se representa la numeración del zócalo PLCC de 52 pines del microcontrolador.

2. En la figura B.19 se representa el patillaje del microcontrolador de 48 pines con encapsulado DIP.

3. En la figura B.20 se representa el patillaje del microcontrolador de 52 pines con encapsulado PLCC.

4. En la figura B.21 se representa el diagrama de bloques del MC68HC11A1.

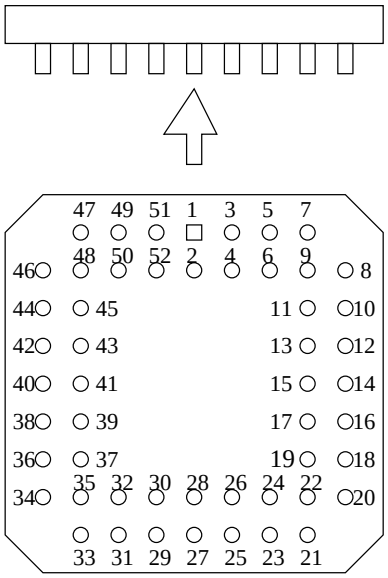


Figura B.18: Numeración del zócalo PLCC de 52 pines (vista inferior).

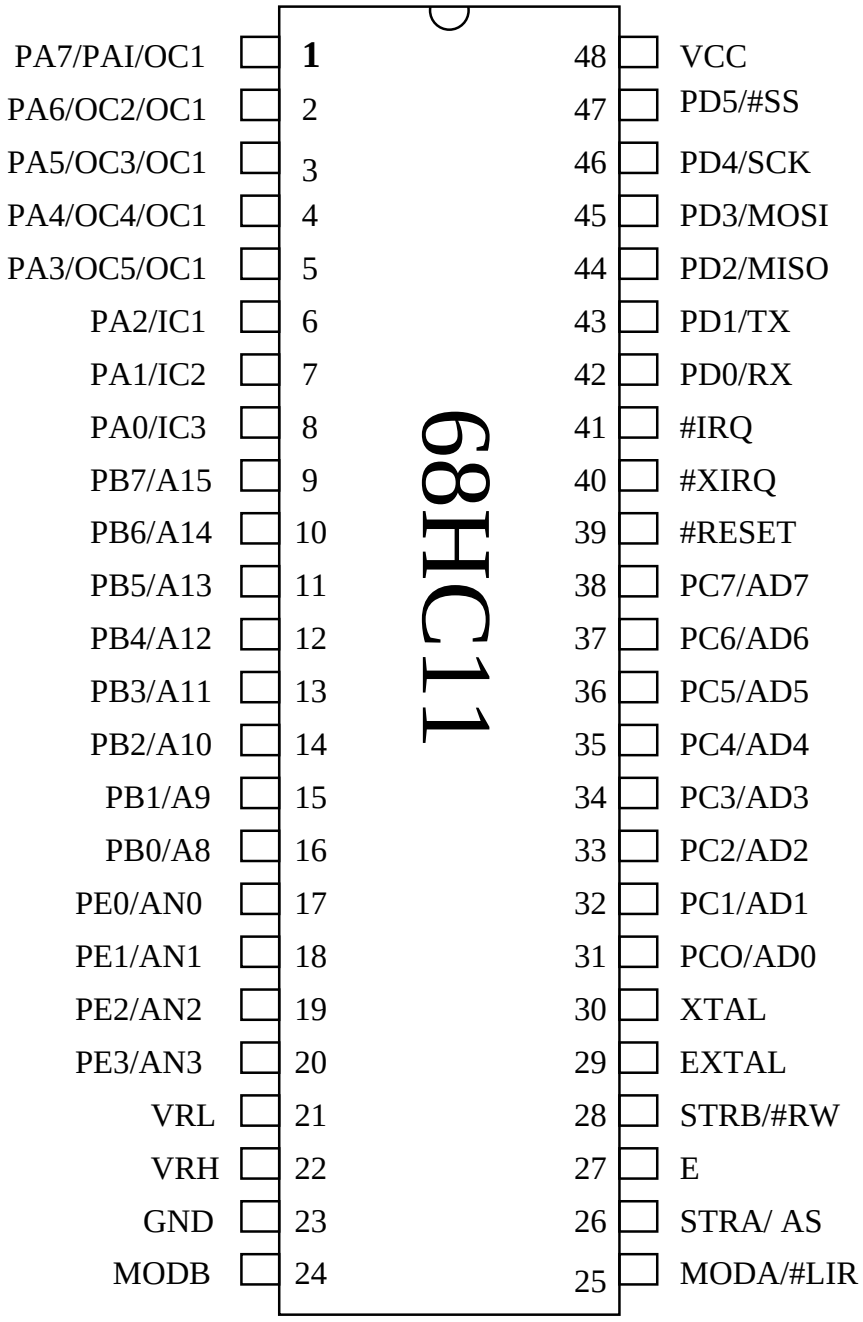


Figura B.19: Patillaje del 68HC11 en formato DIP.

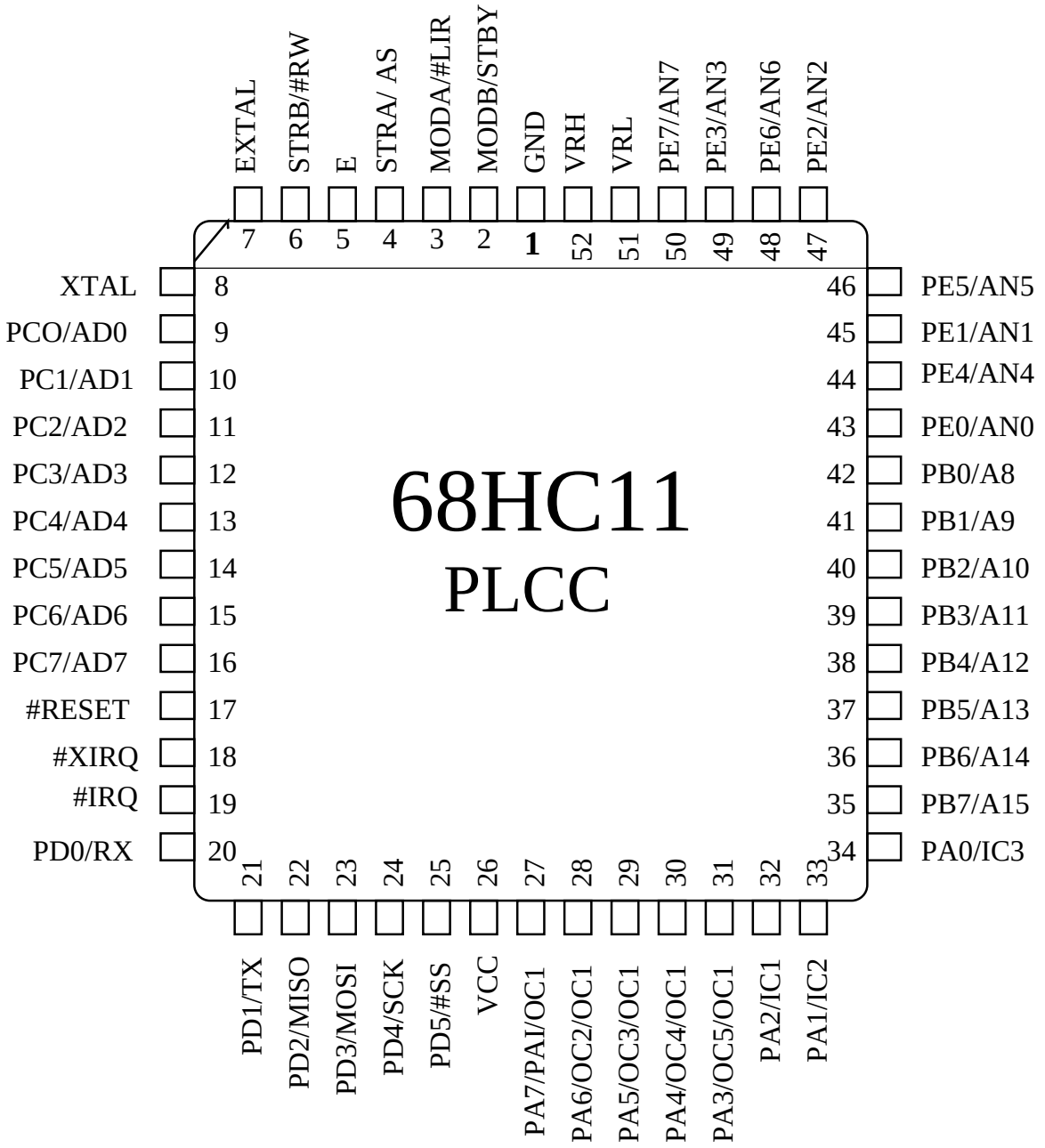


Figura B.20: Patillaje del 68HC11 en formato PLCC:

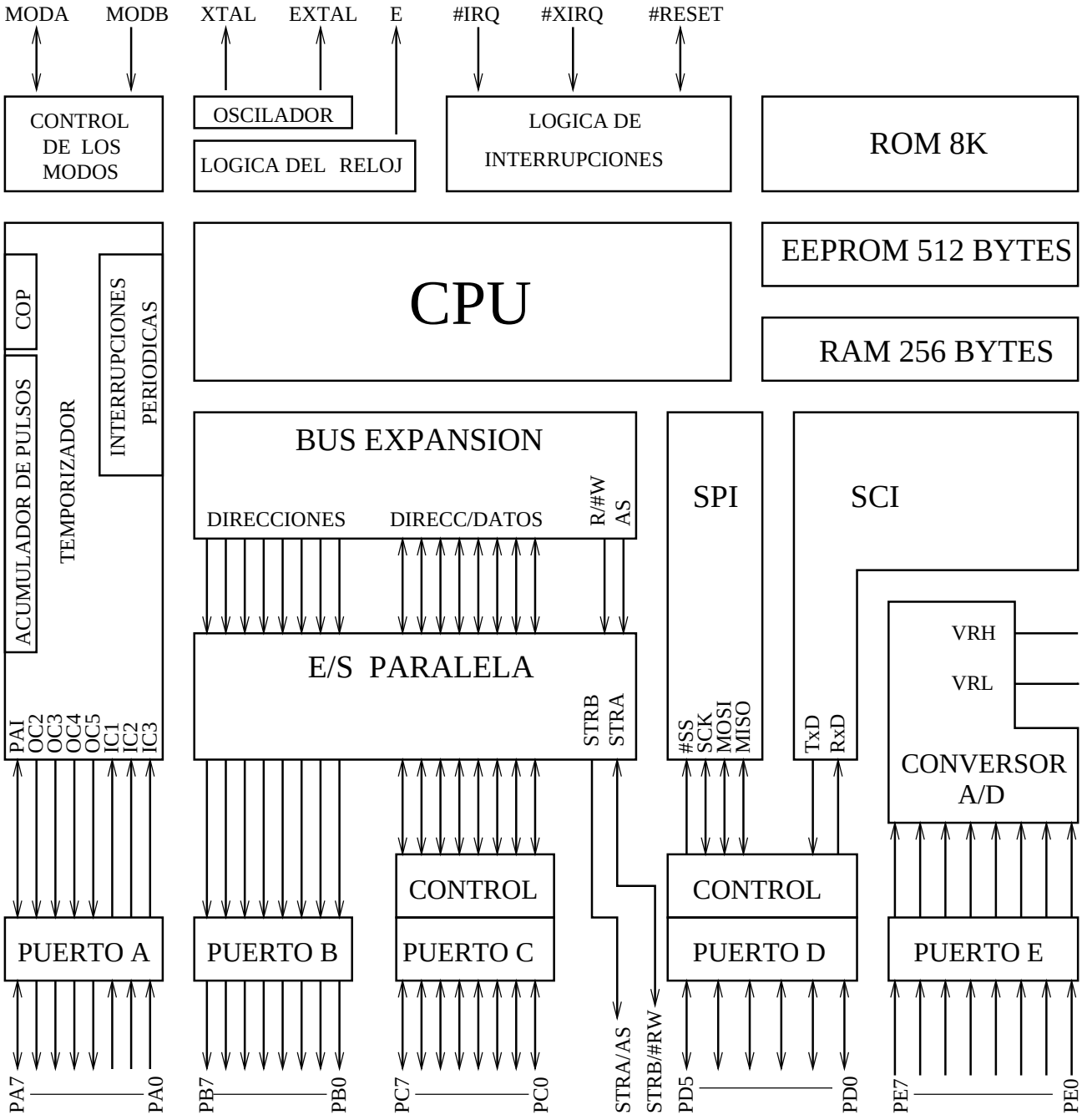


Figura B.21: Diagrama de bloques del MC68HC11A1.

Bibliografía

[1] Robótica Industrial. Mikell P.Groover, Mitchell Weiss, Roger N. Nagel y Nicholas G. Odrey. Editorial Mc Graw Hill.

[2] Forms Library. A Graphical User Interface Toolkit for X. 1996-1998 by T.C. Zhao and Mark Overmars.

[3] COG: http://www.ai.mit.edu/projects/cog

[4] ROBOCUP: http://www.robocup.org

[5] AESS: http://aess.upc.es

[6] Manual Técnico de Microbótica. Autor Microbótica S.L. Correo electrónico: info@microbotica.es.

[7] Artículo tecnológico de TechWeb: "Sony Proposes Open Architecture For Robots". (06/12/98 Yoshiko Hara, EE Times). Localización electrónica: www.techweb.com/news/story/TWB19980612s0008

[8] Gran Larousse Universal. Plaza & Janes. Tomos 19 (insectos) y tomo 22 (mamíferos).

[9] Microcontrolador MC68hc11: Fundamentos, recursos y programación. Autores: Cristina Doblado Alcázar, Andrés Prieto-Moreno Torres, Juan José San Martín Mazzucconi y Juan González Gómez. Edita: Microbótica S.L. (info@microbotica.es)

[10] Microbótica S.L. Empresa relacionada con la robótica de la cual forma parte el autor. 'www.microbotica.es', 'info@microbotica.es'

[11] Página oficial del LEGO Mindstorms: *http://mindstorms.lego.com* .

[12] Pathfinder: William R. Newcott en la revista National Geographic. Número 3 de la versión española. Septiembre 1998, página 2.

[13] Torre Bot. Autor Microbótica S.L. Descrita en el artículo 'Vigilante' del Tomo III de la Biblia de la CT6811: Recopilación de dodumentos. Información localizable en la página web de Microbótica S.L.[10]