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

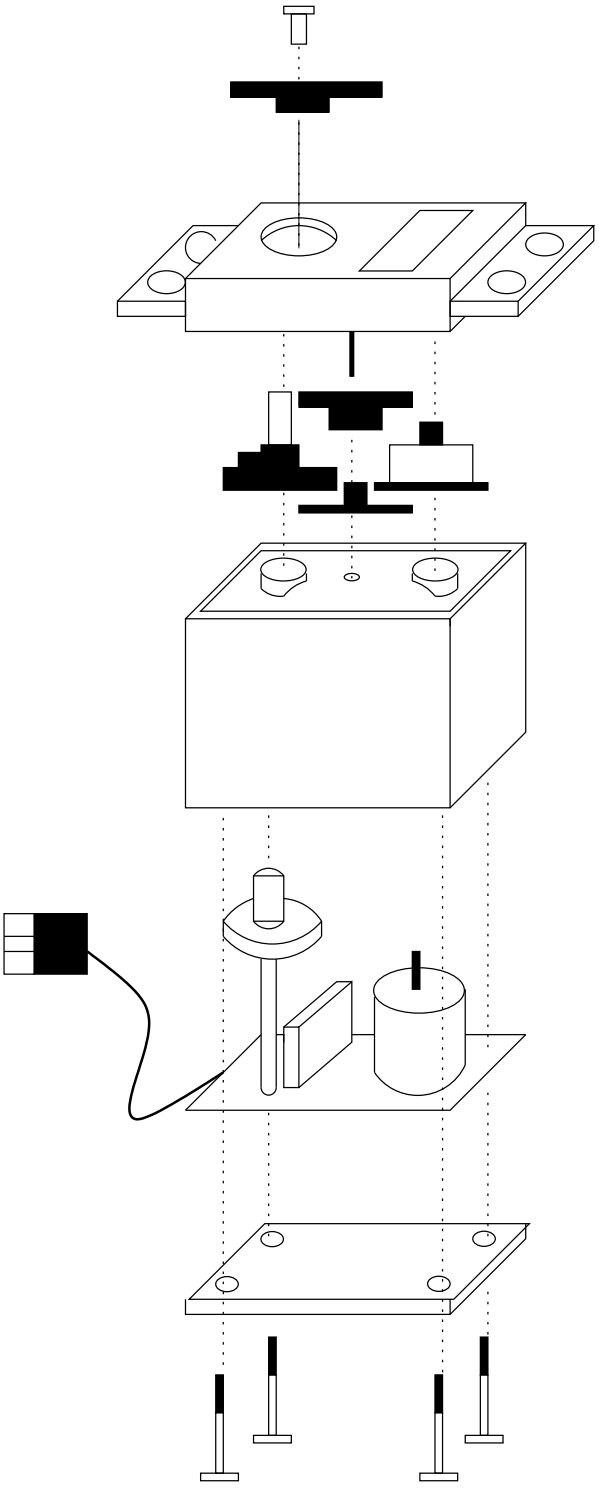

Figura 3.1: Estructura del Futaba S3003.

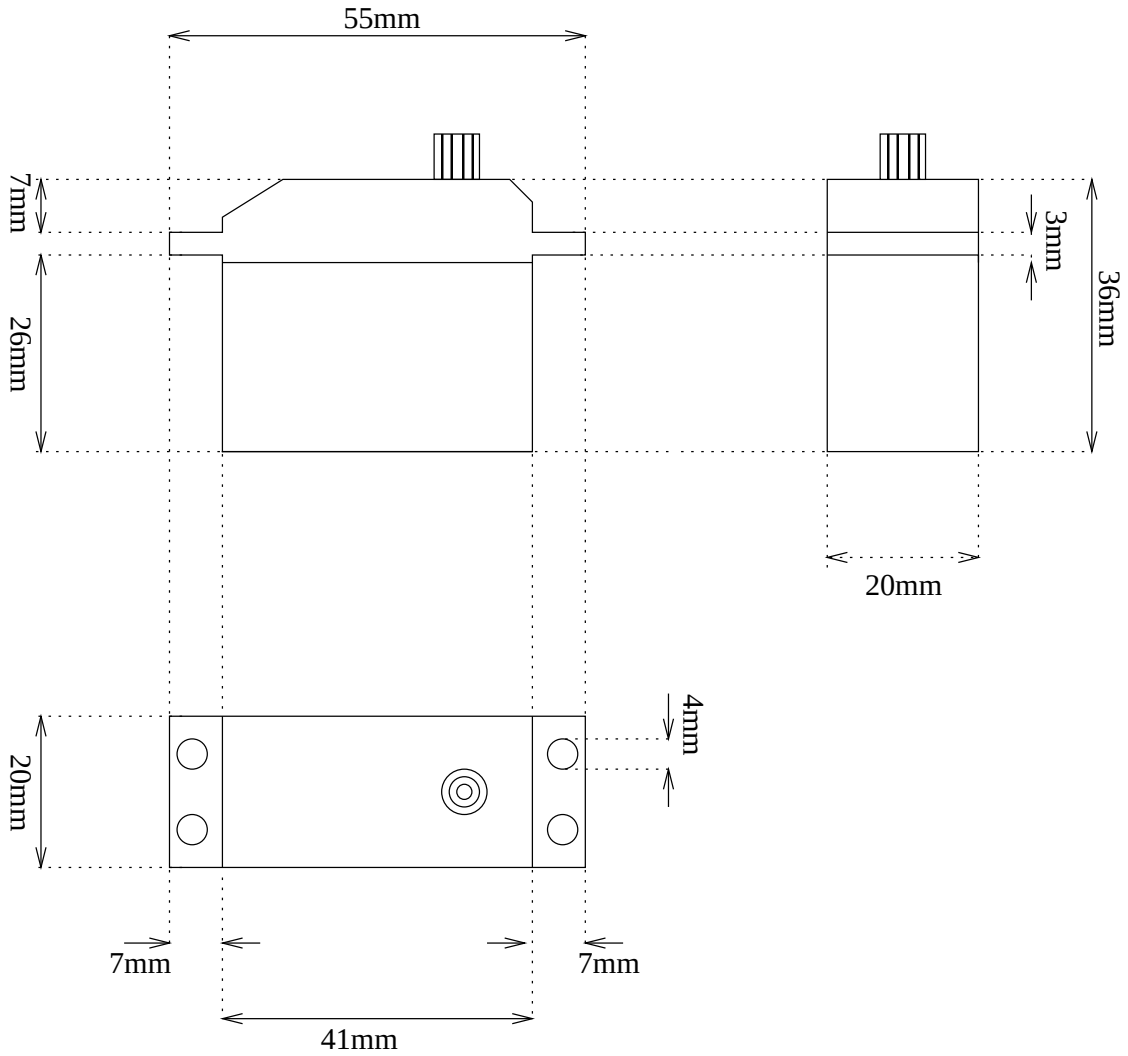

Figura 3.2: Dimensiones del Futaba S3003.

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

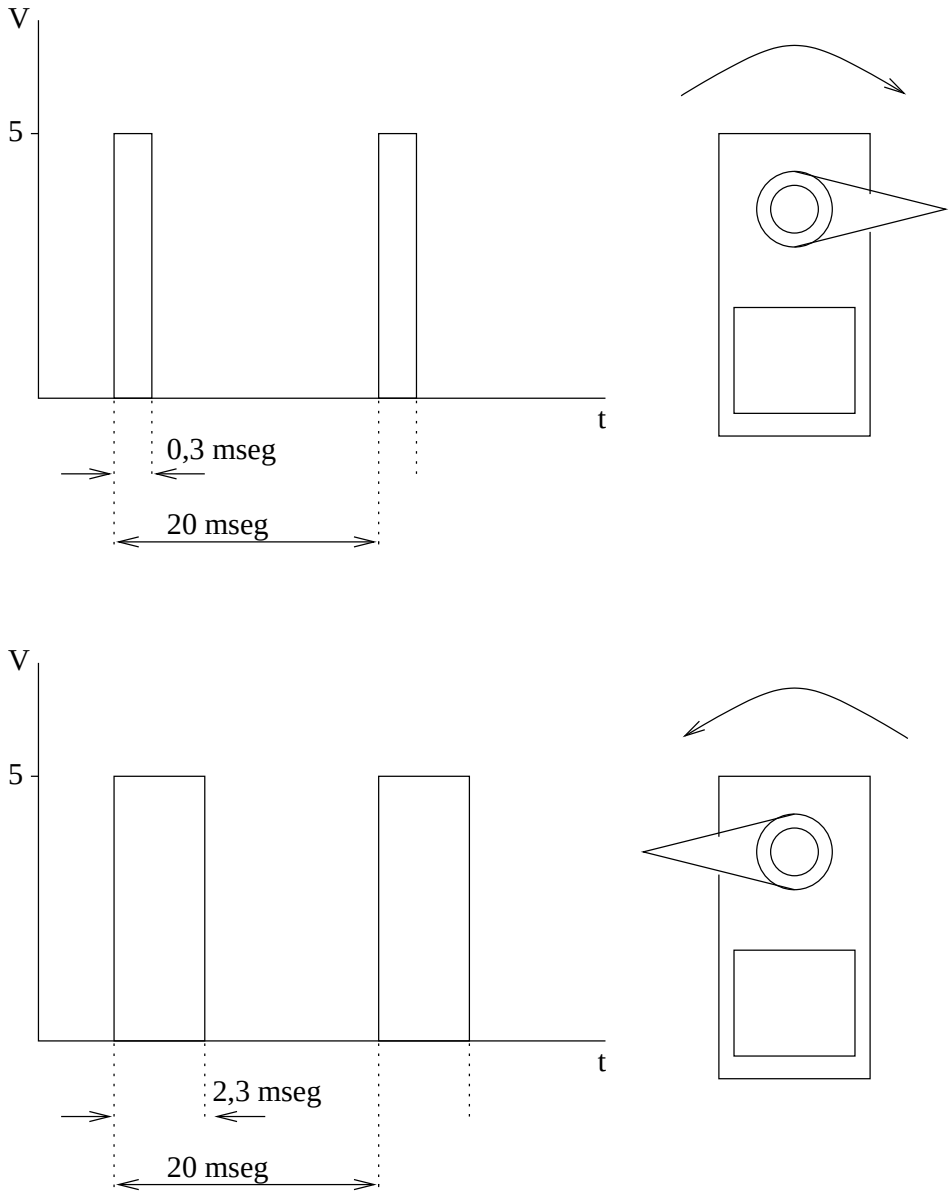

Figura 3.3: Señal de control del Futaba S3003.

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

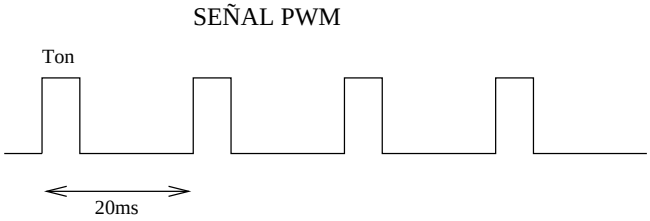

Figura 3.4: Señal PWM para el control de un Futaba

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\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.

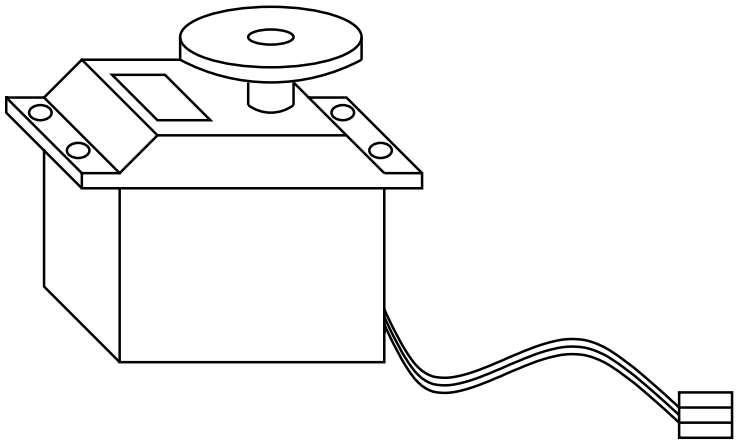

Figura 3.5: Servomecanismo Futaba.

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

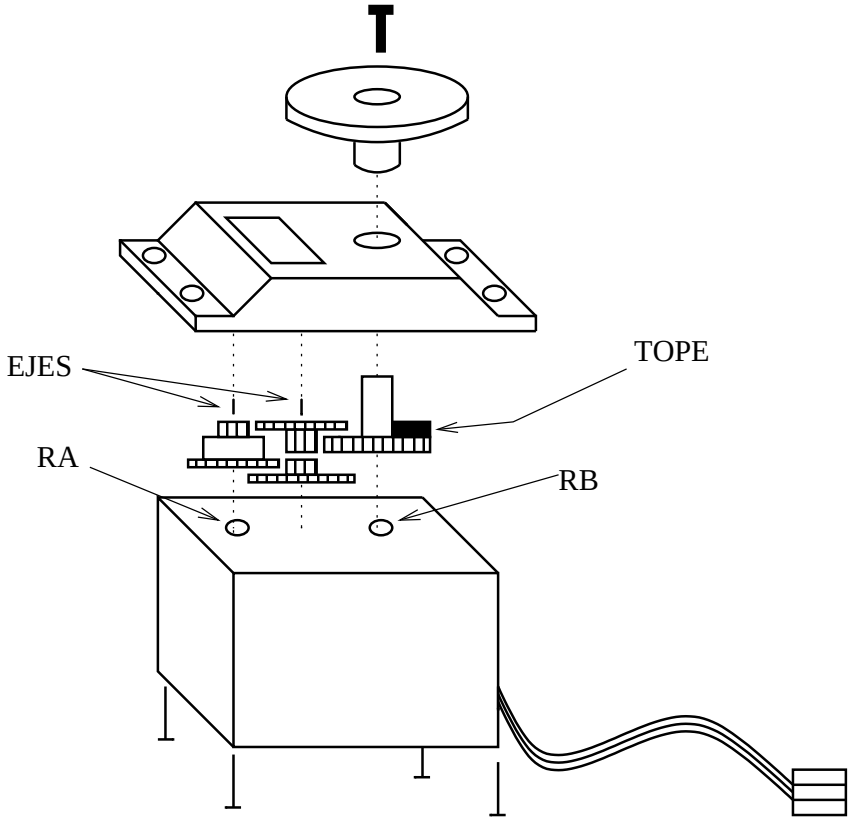

Figura 3.6: Descomposición de un servomecanismo.

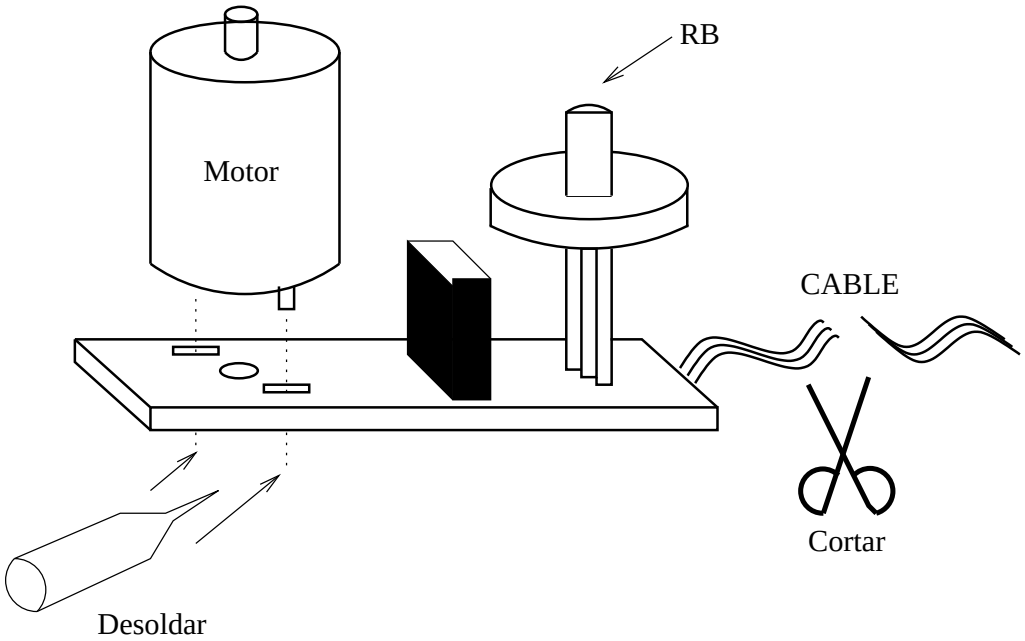

Figura 3.7: Circuito electrónico de un servomecanismo.

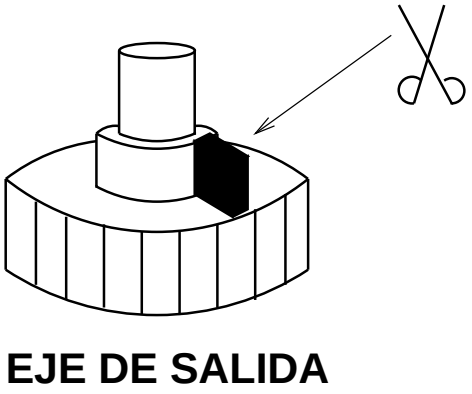

Figura 3.8: Tope mecánico del servomecanismo.

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.