Parte II

Implementación de un gusano transversal

Capítulo 7

Software: modelo virtual

7.1. Introducción

Antes de pasar a la construcción de un robot gusano, es necesario probar las indeas introducidas en la parte teórica. Para ello se ha construido un modelo de gusano virtual, con 6 articulaciones y 5 segmentos, pero que está pensado trabajar con gusanos de cualquier longitud.

Este programa permite probar el algoritmo de ajuste, experimentar con diferentes funciones de contorno, ver los vectores de estado y generar secuencias de movimiento a partir de la propagación de la función de contorno.

Además de ser una herramienta para la investigación a nivel teórico, es el programa encargado de generar un fichero con las secuencias que se enviarán al prototipo mecánico.

El emplear un programa virtual permite desligarse del gusano físico. Es decir, las secuencias generadas son válidas para cualquier gusano, puesto que no dependen de ninguno en concreto. Tiene la limitación de 6 articulaciones, puesto que el interfaz está diseñado para 6 articulaciones, sin embargo el diseño interno está pensado para ser muy facilmente ampliable a cualquier número de articulaciones.

Para más información sobre la programación de interfaces gráficos usando `GTK+` consultar [29]. Un manual bastante bueno para la programación en C es [30] y para la programación bajo Linux/Unix [31].

7.2. Herramientas software

El entorno de desarrollo creado para Cube está constituido por dos aplicaciones (ver figura 7.1), cada una asociada a un tipo de gusano diferente, virtual y real:

1. `Cube-virtual`. Entre otras muchas funciones que se describen con más detalle en este capítulo, permite actuar sobre un gusano transversal virtual, estableciendo el estado de sus articulaciones y generando secuencias a partir de funciones de contorno parametrizadas.

2. `Cube-fisico`. Permite controlar directamente los servos que componen el gusano real, así como generar secuencias creadas por el usuario y leer secuencias del gusano virtual.

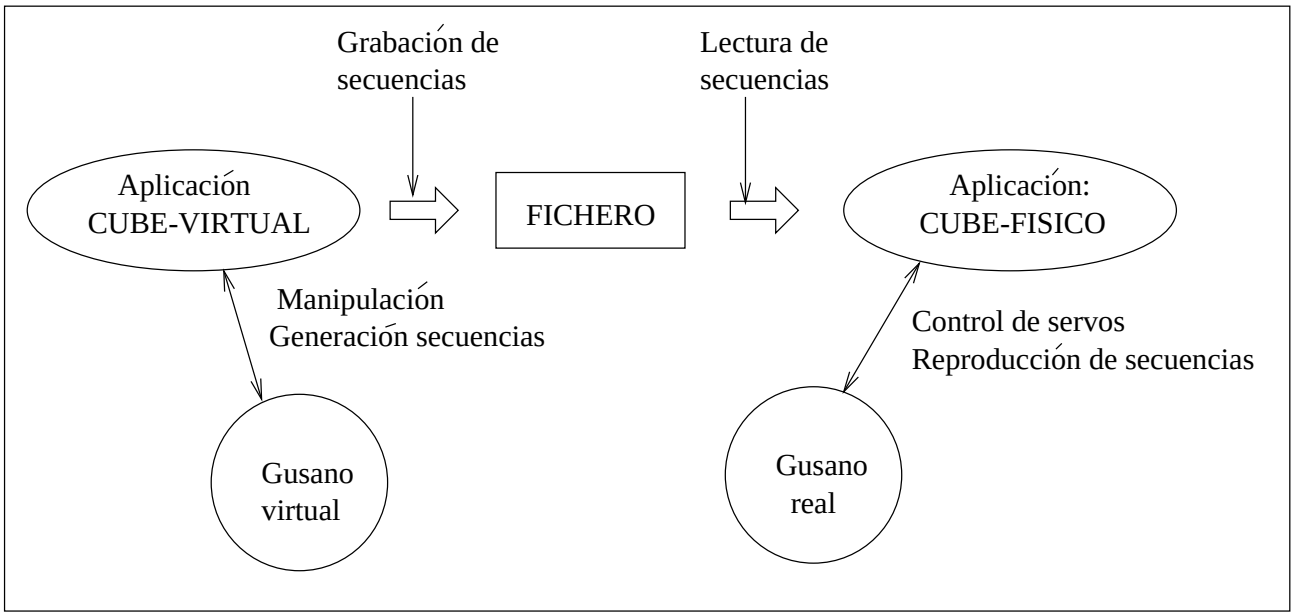


Figura 7.1: Entorno software desarrollado para cube

Para el estudio del avance de un gusano transversal se utiliza la aplicación `cube-virtual`. Mediante ella se pueden establecer los parámetros de la función de contorno: amplitud, longitud de onda y velocidad de propagación que modelan el avance del gusano. En la pantalla se puede ver cómo evolucionan los estados internos para comprender mejor los mecanismos de movimiento. Una vez que se tiene una función de contorno apropiada, el diseñador puede grabar la secuencia en un fichero para luego envíaserla al gusano y reproducirla empleando el programa `cube-físico`, descrito con más detalla en el capítulo 10.

Todo el software se ha desarrollado bajo Linux, y utilizando un interfaz gráfico basado en la librería `GTK+`. El lenguaje de programación empleado es el C y el compilador usado es el `GCC` (GNU Cross Compiler).

7.3. Descripción de la aplicación `cube-virtual`

El programa **cube-virtual** permite realizar las siguientes tareas, sobre un gusano transversal de 6 articulaciones y 5 segmentos:

1. Obtener el vector de estado $E(t)$ del gusano, en un instante
2. Mover individualmente cualquiera de las articulaciones, reflejándose los cambios en el vector de estado
3. Mostrar tres funciones de contorno diferentes y cambiar sus parámetros: amplitud, longitud de onda y velocidad de propagación.

4. Aplicar el algoritmo de ajuste a todo el gusano o a ciertas articulaciones particulares
5. Calcular el error de una articulación así como su ángulo inicial
6. Opción de reproducción *play* para mover la función de contorno.
7. Opción de *enganche* del gusano a la función de contorno, de manera que cada vez que varíen los parámetros de la función de contorno o que ésta se desplace, el gusano virtual lo hará de la misma manera. Esto permite calcular secuencias de vectores de estado.
8. Opción de grabación de una secuencia en un fichero, de manera que pueda ser leída por la aplicación `cube-físico`, presentada en capítulos posteriores, para mover el gusano.

En la figura 7.3 se muestra la apariencia del programa `cube-virtual`. En la pantalla aparece una función de contorno sinusoidal y el gusano está ajustado a ella.

7.4. Arquitectura software

El programa cube-virtual está constituido por 7 módulos, relacionados como se muestra en la figura 7.3. Los módulos son:

- **`Interfaz`**: Dibujo del interfaz
- **`Vectores`**: Operaciones con vectores de estado
- **`Calc`**: Operaciones con puntos en dos dimensiones
- **`Func`**: Manipulación e implementación de las funciones de contorno
- **`Cube`**: Manipulación de Cube
- **`Cube-virtual`**: Programa principal y funciones asociadas (callback) a los botones de la interfaz

A continuación se presentan los módulos con más detalle.

7.4.1. Módulo `calc`

Este módulo realiza operaciones con puntos en dos dimensiones. En él se define el tipo *calc_punto2d_t* y funciones para realizar operaciones con él. Este tipo se define a continuación:

```
typedef struct calc_punto2d_s {
  double x;
  double y;
} calc_punto2d_t;
```

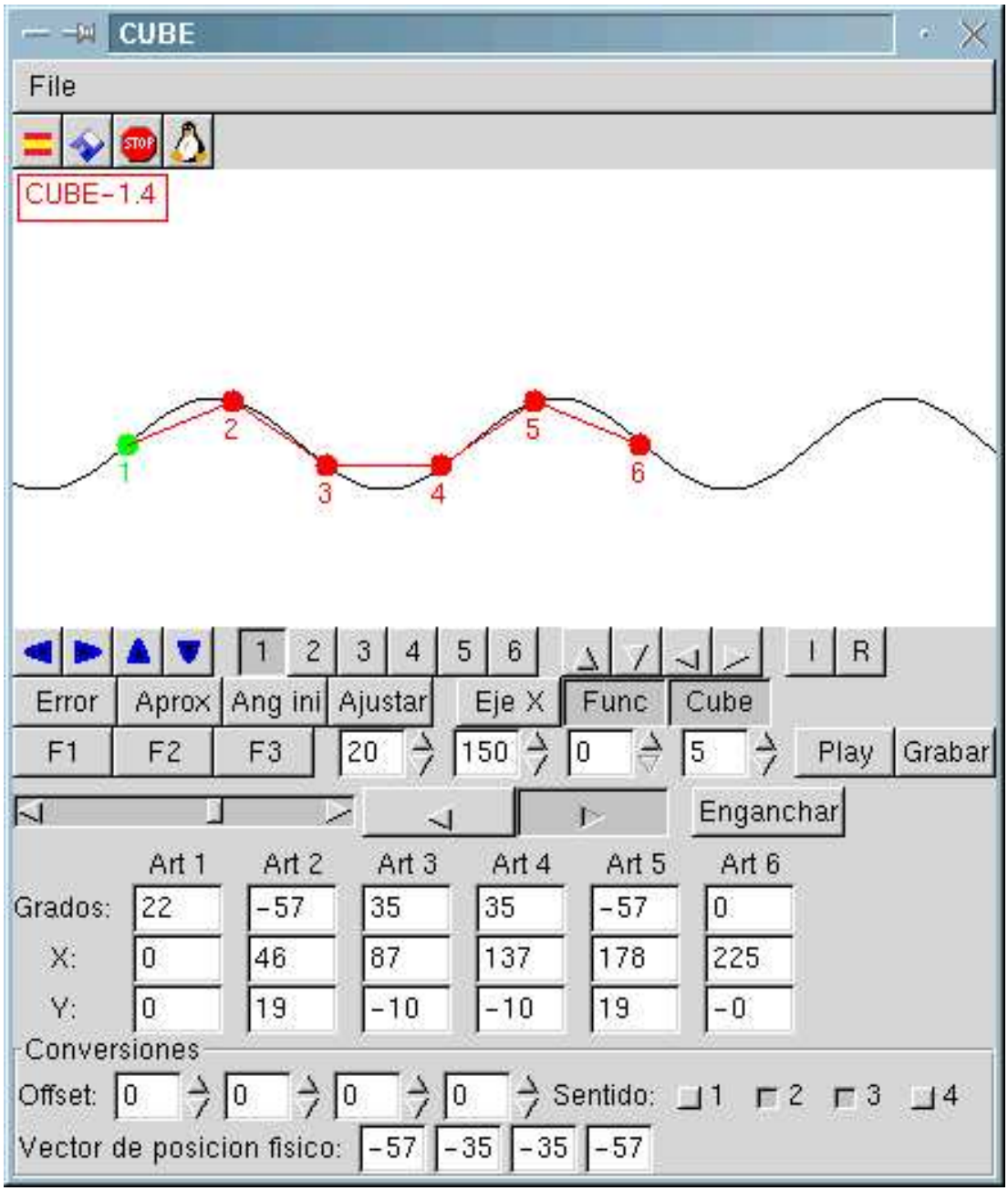


Figura 7.2: Aspecto del programa cube-virtual

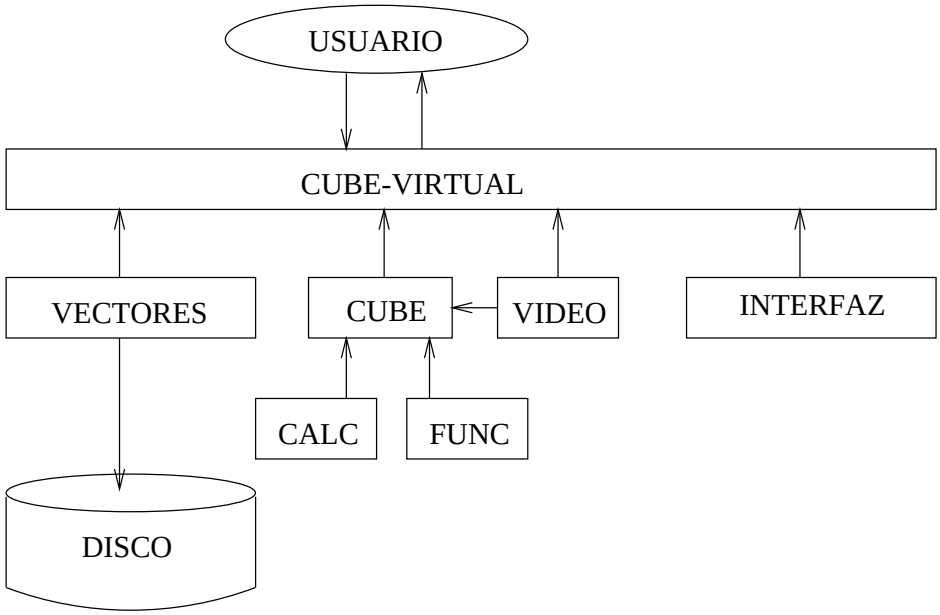


Figura 7.3: Módulos del programa *cube-virtual*

Lo que está definiendo son las coordenadas (x, y) de cada punto. Las operaciones permitidas sobre este tipo vienen definidas por la **interfaz del módulo**:

- **`double calc_angulo(calc_punto2d_t centro, calc_punto2d_t extremo);`**

  Calcular el ángulo que forma el segmento que une los dos puntos con la horizontal.

- **`double calc_angulo_aprox(calc_punto2d_t centro, calc_punto2d_t extremo, double (*func_contorno)(double));`**

  Calcular el ángulo de aproximación de un punto con respecto a un centro y una función de contorno.

- **`double calc_angulo_ajuste(calc_punto2d_t centro, calc_punto2d_t extremo, double (*func_contorno)(double));`**

  Calcular el ángulo que hay que rotar el punto indicado, con respecto al centro, para que se ajuste a la función de contorno. Esta función es donde se implementa el algoritmo de ajuste para una articulación.

- **`double calc_distancia(calc_punto2d_t p1, calc_punto2d_t p2);`**

  Calcular la distancia entre dos puntos

- **`void calc_rotar_punto(calc_punto2d_t centro, calc_punto2d_t extremo, double ang, calc_punto2d_t *rotado);`**

  Obtener un nuevo punto a partir de la rotación del extremo con respecto a un centro un ángulo determinado.

- **`double calc_error_ajuste(calc_punto2d_t centro, calc_punto2d_t extremo, double (*funcion)(double));`**

  Calcular el error de ajuste de un punto con respecto a un extremo y una función de contorno.

7.4.2. Módulo `func`

En este módulo se definen las funciones de contorno de interés y funciones para poder manipular los diferentes parámetros: amplitud, longitud de onda, frecuencia e instante de tiempo. La i**nterfaz del módulo** es:

- **`void func_amplitud_set(double amp);`**

  Establecer el parámetro amplitud

- **`void func_long_onda_set(double lamda);`**

  Establecer el parámetro longitud de onda

- **`void func_tiempo_set(double time);`**

  Establecer el instante de tiempo

- **`void func_frec_set(double frecuencia);`**

  Establecer el parámetro frecuencia

- **`double func_periodo_get();`**

  Obtener el valor del periodo, que se calcula a partir de los parámetros anteriores

- **`double func_onda_sinusoidal(double x);`**

  Implementación de una función sinusoidal genérica. A partir del valor x se devuelve el valor de y, pero en función de los parámetros establecidos.

- **`double func_nula(double x);`**

  Función nula, que devuelve 0 para cualquier valor de x

- **`double func_recta(double x);`**

  Función recta, con una pendiente de 1

- **`double func_ventana(double x);`**

  Esta función devuelve el propio valor de x si se encuentra dentro de una ventana definida mediante parámetros y 0 en el caso contrario. Se emplea para generar funciones no periódicas, como por ejemplo un sólo lóbulo de una función sinusoidal.

De todas las funciones de contorno definidas en este módulo, la que es de interés es la **onda sinusoidal**, que permite modelar los estados internos del gusano.

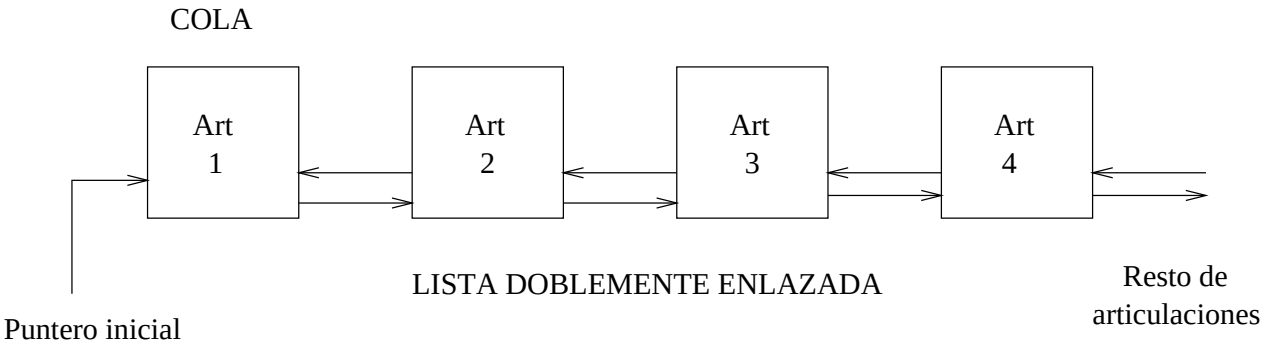


Figura 7.4: Estructura de datos empleada para modelar el gusano

7.4.3. Módulo cube

El módulo cube trabaja con otro tipo abstracto de datos: **la articulación**. La definición del tipo es la siguiente:

```
typedef struct articulacion_s {
  gint _numero;
  gint fi;
  calc_punto2d_t coord;
  gint  sentido;
} articulacion_t;
```

Los campos de los que consta son:

- **numero**: Número de la articulación
- ***fi***: Ángulo de estado de la articulación φ
- ***coord***: Coordenadas (x, y)
- ***sentido***: Sentido de colocación de la articulación. Este parámetro influye en el signo del ángulo φ, y depende de cómo se haya implementado el gusano físicamente y de la orientación de la articulación (ver apartado 7.6).

El gusano virtual se define como una lista doblemente encadenada de articulaciones, de manera que para trabajar con un gusano más largo sólo hay que añadir un nuevo nodo a la lista. La lista comienza por la cola, que es la información que almacena el módulo. También se encuentra definida la **articulación activa**, que es la que el usuario tiene seleccionada para poder moverse.

En la figura 7.4 se muestra gráficamente la estructura de datos empleada para modelar el gusano virtual.

El resto de funciones de interfaz nos permiten realizar diferentes operaciones sobre las articulaciones, olvidándonos de los detalles internos.

- **`void cube_init();`**

  Inicialización del módulo y de todas las estructuras de datos asociadas. Se crea la lista con las articulaciones y se inicializa.

- **`void cube_inicializar_gusano();`**

  Inicializacion de la lista encadenada. Se utiliza para hacer un 'reset' y volver al estado inicial. La función cube_init() llama a esta función, pero antes crea todas las estructuras de datos necesarias. Cube_inicializar_gusano() sólo establece los valores iniciales sobre las estructuras ya creadas.

- **`int cube_long_get();`**

  Devolver la longitud del gusano. Esta función se emplea para poder independizar el resto de módulos de la longitud real del gusano, de manera que sólo cambiando la longitud en este módulo no haya que modificar el resto.

- **`void cube_cola_get_xy(double *x,double *y);`**

  Devolver las coordenadas (x, y) de la articulación de la cola

- **`void cube_cola_set_xy(double x,double y);`**

  Establecer las coordenadas (x, y) de la cola. Se recalculan las coordenadas del resto de articulaciones. Esta función sirve para situar el gusano en cualquier posición de la pantalla, manteniendo las articulaciones en el mismo estado.

- **`void cube_func_contorno_set(double (*func_contorno)(double));`**

  Establecer la función de contorno activa. Se pasa como parámetro un puntero a la función de contorno a emplear. Estas funciones se encuentran definidas en el módulo `func.c`.

- **`void cube_rotar(int centro, int ang, int sentido);`**

  Rotar el gusano respecto de una articulación centro. La rotación se realiza desde la articulación centro hacia la derecha o la izquierda, según cómo se especifique en el parámetro sentido. El funcionamiento normal es que se rote siempre la parte derecha de la articulación centro. El valor de este parámetro puede ser DERECHA o IZQUIERDA

- **`void cube_info_art(int num_art);`**

  Obtener información sobre la articulación indicada. Esta información se imprime en la consola. El propósito de esta función es la depuración del código.

- **`double cube_error_get(int nart);`**

  Devolver el error de aproximación de la articulación especificada. Se devuelve en tanto por ciento.

- **`void cube_ajustar_art(int num_art);`**

  Aplicar el algoritmo de ajuste a la articulación especificada, utilizando como función de contorno la activa. Se aplica el algoritmo sin calcular el ángulo inicial, es decir, se aplica a

partir del estado en el que se encuentra la articulación. Es responsabilidad del que llama a esta función el establecer el ángulo inicial para mejorar la velocidad del algoritmo.

- **`void cube_rotar_aprox(int nart);`**

  Situar la articulación indicada en el ángulo inicial de aproximación con respecto a la función de contorno activa.

- **`void cube_rotar_aprox_rep(int nart);`**

  Esta función es igual que la anterior pero se realiza más de una pasada, de manera que el ańgulo todavía está más próximo al final. El objetivo de esta función es realizar pruebas sobre el tiempo de convergencia.

- **`void cube_ajustar(int nart);`**

  Ajustar el gusano a la función de contorno activa. Primero se calcula el ángulo inicial y después se aplica el algoritmo de ajuste.

- **`double cube_articulacion_get_pos(int nart);`**

  Obtener el ángulo de estado φ de la articulación especificada

- **`void cube_articulacion_set_pos(int nart, double pos, int sentido);`**

  Establecer el ángulo de estado de una articulación y propagar en el sentido indicado

- **`void cube_articulacion_get_xy(int nart, double *x, double *y);`**

  Obtener las coordenadas (x, y) de una articulación.

7.4.4. Módulo vectores

Este módulo trabaja con **vectores de estado** del gusano. La información sobre el estado se encuentra en la estructura de datos del gusano, en el módulo cube, pero este módulo permite trabajar con vectores de estado ya formados, a partir de la información del gusano.

La estructura de datos empleada es:

```
typedef struct vector_pos_s {
  double pos[NUM_ART];
  unsigned int te;
} vector_pos_t;
```

Los campos son los siguientes:

- **`pos`**: Vector con los ángulos de estado φ.
- **`te`**: Tiempo de espera

El campo de tiempo de espera permite definir el tiempo que debe permanecer el gusano en ese estado, antes de pasar al siguiente. Con ello se pueden definir secuencias de movimiento, que serán enviadas al gusano físico.

El **interfaz del módulo** es:

- **`double vector_distancia(vector_pos_t *v1, vector_pos_t *v2);`**

  Calcular la distancia entre dos vectores. La distancia se define como:

  $$d(v,w) = MAX\{|w_1 - v_1|, |w_2 - v_2|, ..., |w_N - v_N|\}$$

  siendo $w = (w_1, w_2, ..., w_N)$ y $v = (v_1, v_2, ..., v_N)$ los vectores de estado. Esta distancia nos da una idea de cúal es la articulación que está más alejada del estado final w.

- **`unsigned long vector_tiempo_trans(vector_pos_t *v1, vector_pos_t *v2, double ct);`**

  Calcular el tiempo de tránsito entre dos estados, empleando una constante de tiempo. Esta función no se utiliza en el programa cube-virtual y se explicará más adelante.

- **`void vector_suma(vector_pos_t *v1, vector_pos_t *v2, vector_pos_t *suma);`**

  Calcular la suma de dos vectores de estado. La suma se define como:

  $$v + w = (v_1 + w_1, v_2 + w_2, ..., v_N + w_N)$$

- **`void vector_producto(vector_pos_t *v1, vector_pos_t *v2, vector_pos_t *prod);`**

  Calcular el producto de dos vectores, definido como:

  $$v.w = (v_1 w_1, ..., v_N w_N)$$

- **`vector_pos_t *vector_new();`**

  Crear un vector nuevo, inicializado a cero

- **`vector_pos_t *vector_crear(double a1, double a2, double a3, double a4, unsigned int te);`**

  Crear un vector con los valores indicados

- **`void vector_free(vector_pos_t *vector);`**

  Liberar la memoria tomada por un vector

- **`int vector_save(FILE *f, vector_pos_t *v);`**

  Almacenar el vector en un fichero. Se almacena en una línea ASCII, con la siguiente codificación: $[v1, v2, v3, v4], te$. Donde $v1 - v4$ son los estados de las articulaciones 1-4 y te el tiempo de espera asociado a ese estado. Los estados de la cabeza y la cola no se graban puesto que son articulaciones "virtuales" que en el gusano real no existen.

- **`int vector_load(FILE *f, vector_pos_t *v);`**

  Leer un vector de un fichero y devolverlo. Esta función no se usa en cube-virtual, pero sí en los programas que se presentan más adelante.

- **`void vector_print(vector_pos_t *v);`**

  Función de depuración para imprimir un vector de estado en la consola

7.4.5. Módulo `video`

Este es el módulo encargado de dibujar el gusano virtual, en una pantalla gráfica situada encima de los botones. Además permite variar parámetros de dibujo, como por ejemplo si hay que mostrar el eje x o la función de contorno.

- **`GtkWidget *video_init();`**

  Inicialización de la pantalla de video. Se crea el área de dibujo, se configura y se asignan los eventos necesarios. Todas las variables locales del módulos también se inicializan

- **`void video_origenx_inc(int inc);`**

  Incrementar coordenada x del origen de coordenadas

- **`void video_origeny_inc(int inc);`**

  Incrementar coordenada y del origen de coordenadas

- **`void video_origen_reset();`**

  Llevar el origen a su situación inicial

- **`void video_refrescar();`**

  Redibujar toda la información de la pantalla, en función del nuevo estado del gusano y del interfaz

- **`void video_set_func_draw(double (*func)(double));`**

  Establecer la función de contorno que se dibujará

- **`void video_set_art_activa(int nart);`**

  Indicar cuál es la articulación activa. Se pinta con otro color

- **`void video_show_ejex(gboolean dibujar);`**

  Mostrar/ocultar el eje x

- **`void video_show_func(gboolean dibujar);`**

  Mostrar/ocultar la función de contorno

- **`void video_show_cube(gboolean dibujar);`**

  Mostrar/ocultar el gusano virtual

7.4.6. Módulo `interfaz`

Este módulo crea y dibuja el interfaz que el usuario puede ver. Todo el estado interno del interfaz: estado de los botones, contenido de los cuadros de datos,etc. se sitúa dentro de una variable del tipo *main_state_t*:

```
typedef struct main_state_s {
  GtkWidget *win_main;          /* Ventana principal */
  GtkTooltips *sugerencias;     /* Sugerencias        */
  GtkAccelGroup *accel_group;   /* Teclas aceleracion */
  GtkObject *adj1;              /* Valor asociado al potenciometro  */

  /*-- Valores de los diferentes parametros de la onda -- */
  GtkObject *amp_adj;   /* Amplitud          */
  GtkObject *lamda_adj; /* Longitud de onda  */
  GtkObject *time_adj;  /* Tiempo            */
  GtkObject *frec_adj;  /* Frecuencia        */

  /* ---- Botones Toggle  ----- */
  GtkWidget *toggle_ejex;
  GtkWidget *toggle_func;
  GtkWidget *toggle_cube;
  GtkWidget *toggle_enganche;

  /* --- LCD's ---- */
  GtkWidget *lcd_grados[6];
  GtkWidget *lcd_x[6];
  GtkWidget *lcd_y[6];

  /*--- Valores para el vector de offset ---*/
  GtkObject *a_adj[4];

  /*--- Lcd para vectores fisicos ----*/
  GtkWidget *vector_lcd[4];

  /*--- Valores para los botones de sentido --*/
  GtkWidget *sentido[4];

} main_state_t;
```

Sólo existe una función de interfaz, que lo construye, lo dibuja y devuelve el estado en una variable del tipo anterior.

- **`void create_window1(main_state_t *estado);`**

7.4.7. Programa principal: `cube_virtual`

Este módulo realiza todas las inicializaciones de los módulos anteriores, llama a **create_window1()** para dibujar el interfaz, inicializa su propio estado interno y le pasa el control al motor de GTK+, que gestiona las señales y eventos producidos.

Cada vez que ocurre algún evento, como la pulsación de un botón, se llama a la función de retrollamada correspondiente, implementada en el módulo principal. La función asociada a un evento (callback function) determina, según el estado del programa, qué acciones hay que tomar.

7.5. Manejo del programa

7.5.1. Descripción del interfaz

En el interfaz se pueden distinguir 4 partes, que se muestran en la figura 7.5:

1. **Pantalla de vídeo**. Zona donde se representa el gusano virtual y la función de contorno
2. **Botones de interfaz**. Conjunto de botones, barras de desplazamiento y entrada de parámetros que permiten realizar operaciones sobre el gusano virtual o la función de contorno.
3. **Estado de las articulaciones**. Se muestra el ángulo de estado de cada articulación φ así como las coordenadas x, y.
4. **Conversiones**. Conversiones que se aplican para obtener el vector de estado físico.

En las siguientes secciones se explica en detalle cada una de estas partes.

7.5.2. La pantalla de vídeo

La figura 7.6 muestra el aspecto de la pantalla de vídeo. En ella podemos ver las tres componentes que se visualizan:

- Eje X
- Función de contorno
- Gusano virtual

Cada una de estas partes se puede visualizar u ocultar, según que los botones *Eje x*, *Func* y *Cube* estén o no activados (definidos en la página 133).

Las articulaciones se visualizan como círculos rojos, con su número asociado. La articulación activa se muestra con un círculo verde y es sobre la que actúan ciertas partes del interfaz, como la barra de posicionamiento.

La articulación número 1 se encuentra situada en la coordenada $x = 0$, $y = 0$. Inicialmente el gusano se encuentra situado sobre el eje x.

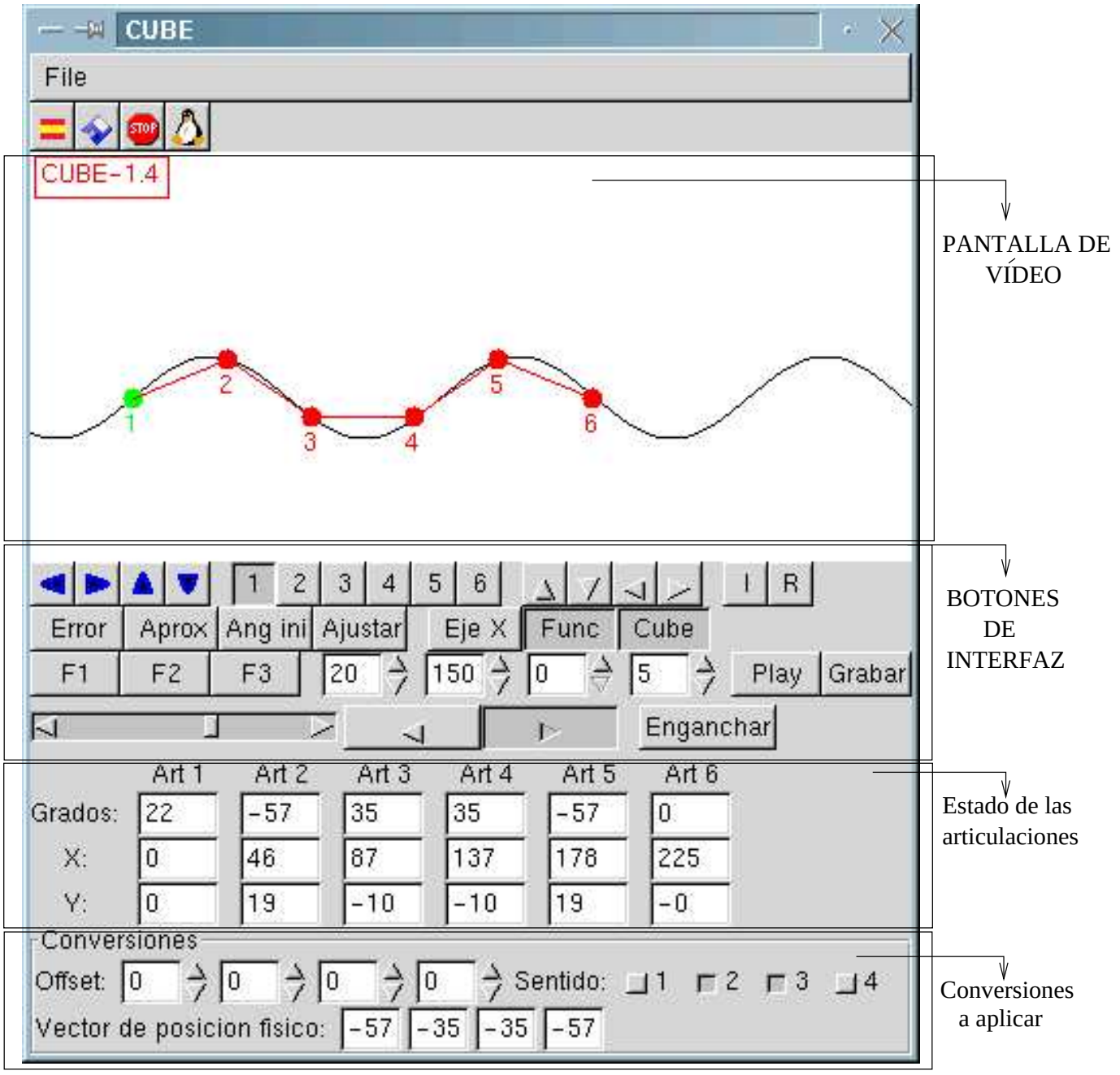


Figura 7.5: Diferentes partes del interfaz del programa cube_virtual

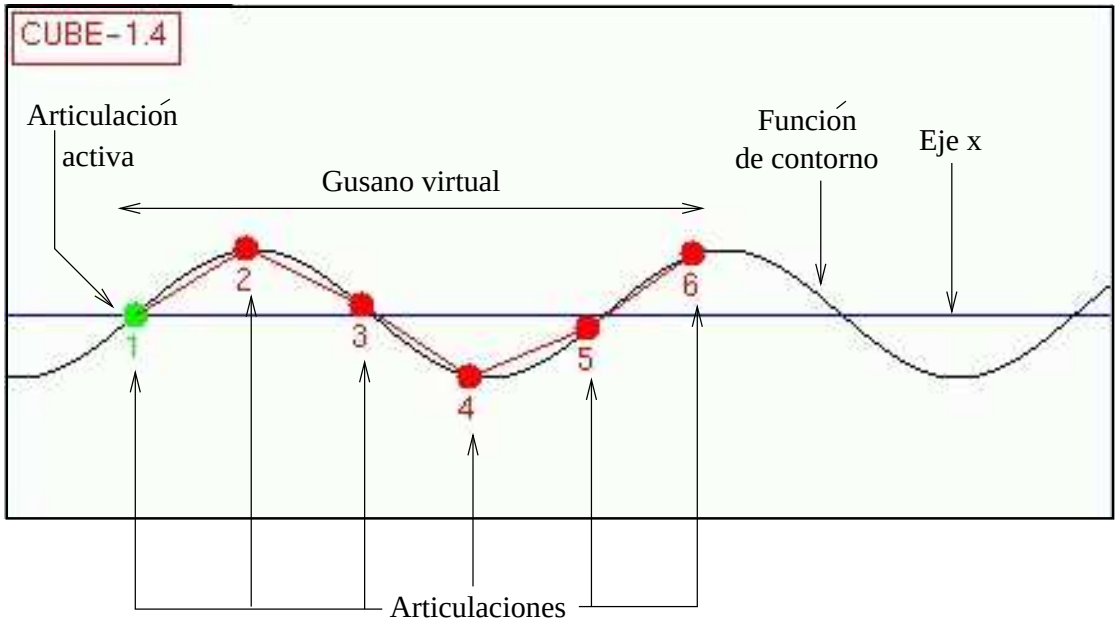


Figura 7.6: La pantalla de vídeo

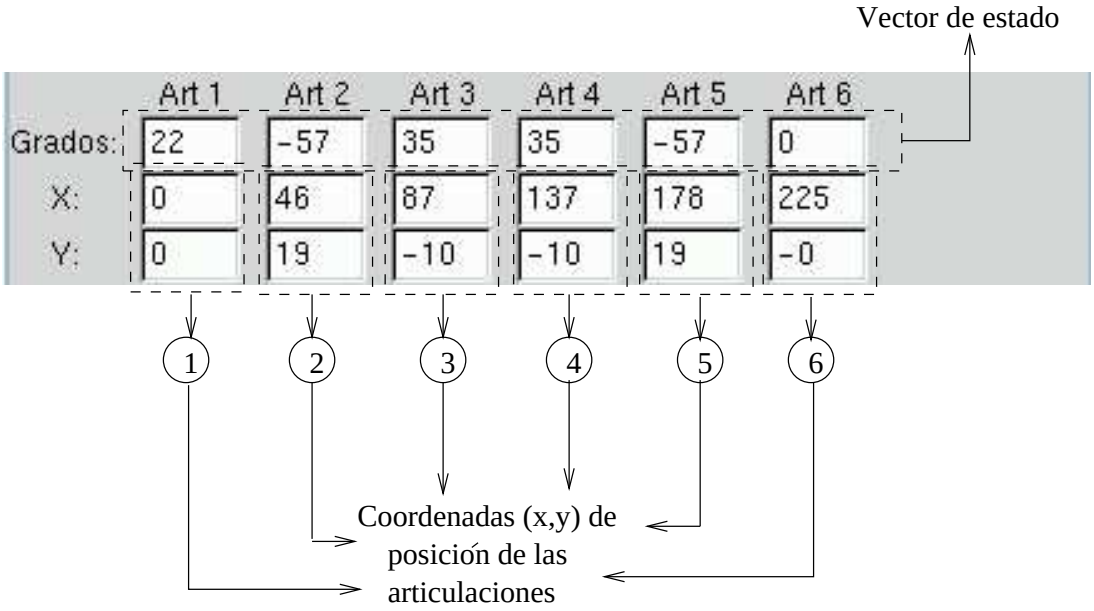


Figura 7.7: Estado de las articulaciones

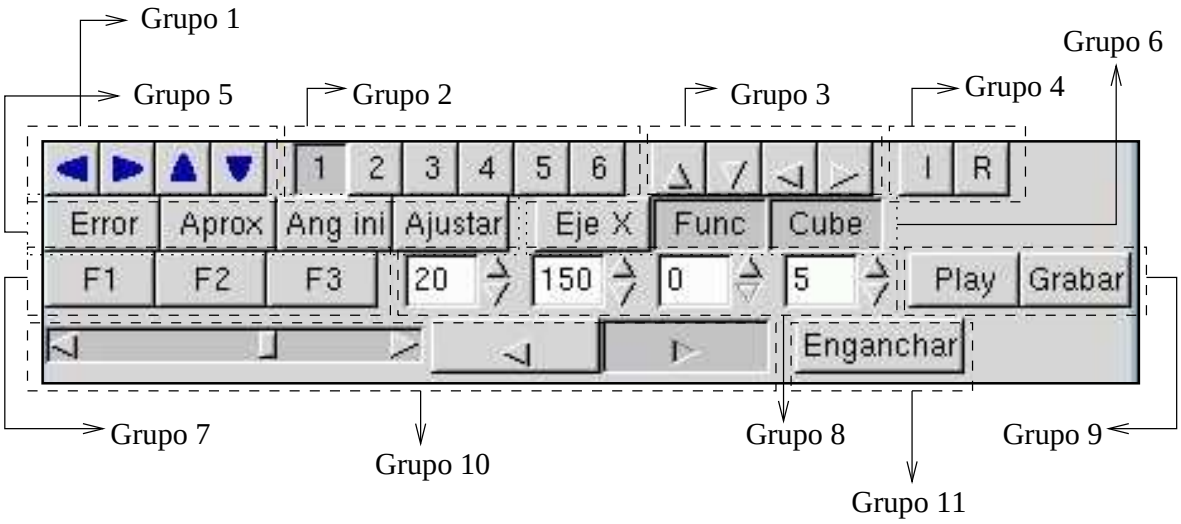


Figura 7.8: Grupos de botones en el interfaz

7.5.3. Estado de las articulaciones

La parte del interfaz que muestra el estado de cada una de las articulaciones aparece en la figura 7.7. El vector de estado está en grados sexagesimales, y cada componente representa el ángulo de estado φ en el que se encuentra cada articulación.

Además del estado se representa también un vector X, que contiene todas las coordenadas x de las articulaciones y un vector Y con las coordenadas y. Mirando estos vectores transversalmente, componente a componente, se obtienen los pares (x, y) de coordenadas de cada articulación.

Es muy interesante conocer las coordenadas de las articulaciones, sobre todo las coordenas x para saber en todo momento la longitud del gusano longitudinal asociado y si se está cumpliendo o no la condición de que la longitud permanezca constante mientra la función de contorno se desplaza. Cuanto más constante sea esta longitud, mejor avanzará el gusano.

La mejor manera de conocer la longitud es fijándose en la coordenada x de la articulación 6, cuando se encuentra apoyada sobre el suelo (esto es, que y=0). La teoría dice que esta coordenada x debería ser constante siempre que y=0, es decir, siempre que no esté siendo atravesada por la función de contorno.

7.5.4. Botones del interfaz

Existen muchos botones en el interfaz, clasificados en diferentes grupos según su funcionalidad, como se muestra en la figura 7.8. En total hay 11 grupos, pero el grupo 11 está constituido por un único botón.

1. **Grupo 1**: Desplazamiento del origen de coordenadas.

   El origen de coordenadas está situado sobre la articulación 1, cuando se encuentra en reposo. Mediante estos botones se cambia el origen, desplazándose todos los elementos

que se dibujan en la pantalla: gusano, eje x y función de contorno. Con ello podemos centrar en la pantalla la información que nos interese ver. Normalmente estos botones no se usarán.

2. **Grupo 2**: Selección de la articulación activa.

   De este grupo de botones sólo uno puede estar activo cada vez. Permiten seleccionar cuál es la articulación activa, que se dibuja en la pantalla como un círculo verde en vez de rojo. Sobre la articulación activa actúan botones de otros grupos.

3. **Grupo 3**: Desplazamiento del gusano

   Este grupo permite mover el gusano, variando las coordenadas (x, y) de todas las articulaciones de manera que el estado permanece inalterado, pero la posición varía.

4. **Grupo 4**: Información y Reset

   Formado por dos botones, uno de reset (R) y otro de información (I):

   - **Reset**: Llevar el gusano al estado inicial, con la articulación 1 situada en el origen de coordenadas y asignando el estado (0,0,0,0,0,0). El origen de coordenadas se sitúa en su posición inicial en la pantalla.
   - **Info**: Devuelve información sobre la articulación activa. La información se imprime en la consola. Es un botón de depuración. La información se presenta como en el ejemplo siguiente:

     ```
     [2] (46.359,18.730) pos: -57
          Ángulo con eje x: 22.000
     ```

     Donde [2] indica que la articulación activa es la dos. La información entre paréntesis son las coordenadas (x, y), y pos es el ángulo de estado φ. También se indica el ángulo que forma con el eje x (θ)

5. **Grupo 5**: Parámetros del algoritmo de ajuste

   Este grupo está relacionado con el algoritmo de ajuste. Está constituido por 4 botones, pensados sobre todo para hacer pruebas con los conceptos teóricos.

   - **Error**: Imprime en la consola el error de aproximación de la articulación activa, expresado en tanto por ciento. Este error nos indica cómo de bien está ajustada esa articulación a la función contorno. Un ejemplo es:

     ```
     [2]: 12.00%
     ```

     Donde el parámetro entre corchetes es la articulación activa y el otro es el error.
   - **Aprox**: Aproximar la articulación activa a la función de contorno. Se aplica el algoritmo de ajuste sólo para la articulación activa. Esta opción es muy útil para ver cómo evoluciona el algoritmo cuando se aplica a las diferentes articulaciones, desde la 1 hasta la 6.

- **Ang ini**: Situar la articulación anterior a la activa en el ángulo inicial de aproximación, de manera que la articulación activa se encuentra más cerca de la función de contorno. Esto permite evaluar cuánto de bueno es el algoritmo de calculo de la primera aproximación, antes de empezar a iterar.
- **Ajustar**: Aplicar el algoritmo de ajuste completo, desde la articulación activa hasta la cabeza del gusano (articulación 6). Si la articulación activa es la 1 el gusano completo se ajusta a la función de contorno.

6. **Grupo 6**: Selección de los elementos a visualizar

   Estos botones permiten elegir cuáles de los tres elementos que se pueden representar en la pantalla se quieren visualizar:

   - **Eje x**: Visualizar o no el eje x. Se visualiza como una línea azul
   - **Func**: Función de contorno
   - **Cube**: Gusano virtual

   Se trata de botones independientes que pueden estar en cualquiera de los dos estados ON/OFF por lo que se pueden presentar en cualquiera de las 8 combinaciones: desde no visualizar nada hasta visualizarlo todo (opción por omisión).

7. **Grupo 7**: Selección de la función de contorno

   Permite seleccionar entre tres funciones de contorno diferentes. Por omisión se han implementado tres tipos de funciones de contorno, pero es muy fácil representar cualquier otra sin más que añadirla en el módulo `func.c`.

   - **F1**: Función sinusoidal completa
   - **F2**: Función sinusoidal rectificada, sólo con parte positiva
   - **F3**: Función sinusoidal multiplicada por una ventana de una cierta anchura, de manera que sólo se muestra un "trozo" de la señal, como por ejemplo un lóbulo positivo.

   Las funciones de contorno que se emplean se muestran en la figura 7.9.

8. **Grupo 8**: Parámetros de la función de contorno

   En los cuadros de este grupo se muestran los 4 parámetros de las funciones de contorno que se pueden variar. Afectan a las tres funciones definidas: F1, F2 y F3. Los parámetros son, por orden de izquierda a derecha:

   - **Amplitud** (A)
   - **Longitud de onda** (λ)
   - **Instante de tiempo** (t)
   - **Velocidad de propagacion** (v)

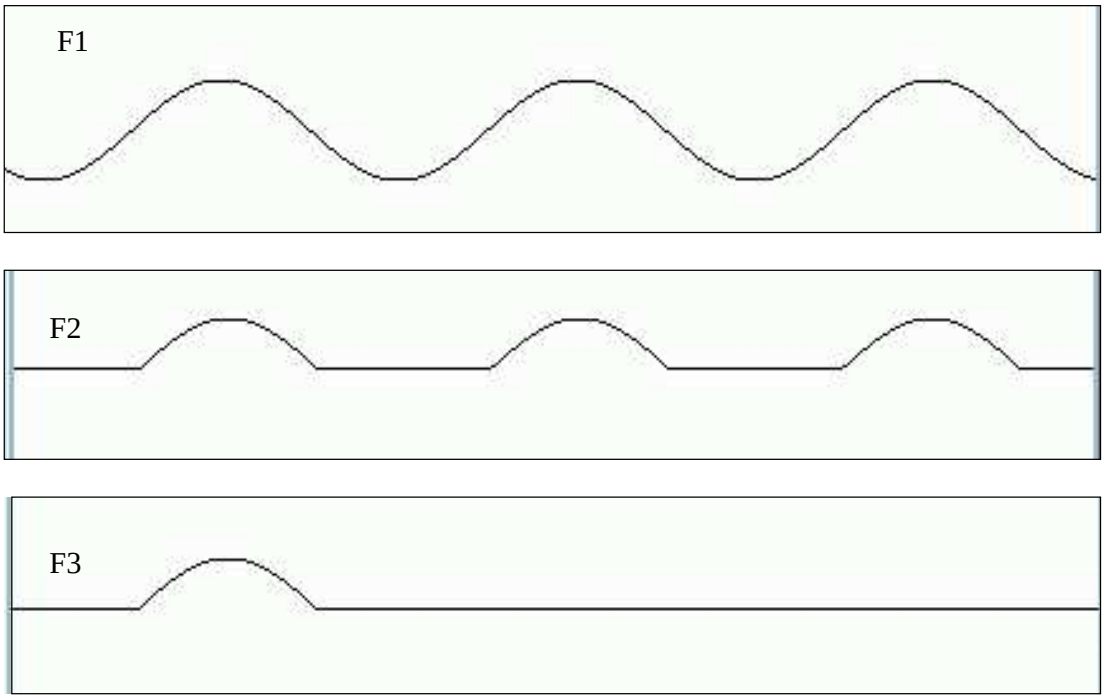


Figura 7.9: Las tres funciones de contorno empleadas

Cuanto menor sea la longitud de onda, más "lóbulos" sinusoidales recorrerán el gusano. Cuanto mayor sea, mejor se ajusta el gusano a la función de contorno y más se cumple la condición de que la longitud del gusano longitudinal asociado permanezca constante.

El parámetro tiempo nos define el instante en el que se encuentra la onda, ya que se va propagando. Inicialmente se toma $t = 0$. Al aumentar este parámetro puede verse cómo la onda "avanza".

La velocidad de propagación nos da una idea del número de instantes de tiempo que tarda la onda en recorrer una longitud igual a la longitud de onda λ.

$$v = \frac{\lambda}{T}$$

Fijada una λ y dos velocidades v_1 y v_2, con $v_1 < v_2$, en el instante $t = 0$ las ondas son iguales pero en el instante $t = 1$, la onda con velocidad v_2 ha avanzado más que la onda con velocidad v_1.

Por ello, el parámetro velocidad, nos está indicando cuántos estados diferentes va a tener el gusano al recorrer una longitud igual a λ. Si v es alto, se generan pocos estados intermedios y si es bajo muchos. Para poner un ejemplo, con un valor fijo de $\lambda = 150$, que es la longitud de onda que se tiene por defecto y con $v_1 = 5$ y $v_2 = 10$, se obtienen en el primer caso 30 estados diferentes y 15 en el segundo. Esto es porque en el primer caso la distancia igual a λ se recorre en 30 instantes de tiempo y en 15 en el segundo caso.

A menor velocidad de propagación, mayor número de estados internos y por tanto mayor es la resolución temporal, es decir, tenemos más "fotogramas" del avance del gusano.

9. **Grupo 9**: Generación de secuencias

   Este grupo comprende dos botones relacionados con la evolución temporal del gusano.

   - **Play**: Realizar una "reproducción" de la evolución de los estados internos del gusano, con la función de contorno activa y con sus parámetros establecidos. **Se entiende por reproducción la siguiente secuencia de acciones**:
     a) Aplicar el algoritmo de ajuste al gusano, con respecto de la función de contorno establecida
     b) Incrementar el parámetro tiempo
     c) Recalcular la nueva función de contorno para el nuevo instante temporal
     d) Volver al punto (a)
   - **Grabar**: Realizar una reproducción sólo hasta que la onda avance una distancia igual a la longitud de onda, de manera que el estado inicial es igual al estado final, y grabar los vectores de estado en un fichero.

   En la figura 7.10 se muestran un gusano ajustado a una función de contorno en diferentes instantes.

10. **Grupo 10**: Movimiento de las articulaciones

    Este grupo comprende dos botones de dirección, izquierda y derecha y una barra de desplazamiento.

    - **Barra de desplazamiento**: Permite establecer el ángulo de estado de la articulación activa. De esta manera podemos situar manualmente el gusano en cualquier estado
    - **Botón derecha**: Establecer el sentido de propagación del giro de la articulación activa. Por defecto siempre es el derecho. Esto quiere decir que al mover una articulación, las articulaciones del lado izquierdo quedarán fijas y sólo se mueven las del lado derecho.
    - **Botón izquierda**: Establecer el sentido de propagación hacia la izquierda.

    En la figura 7.11 se ha dibujado un ejemplo para comprender mejor el fenómeno de la propagación hacia la derecha o hacia la izquierda. Se parte del gusano que está en el estado del dibujo superior. El dibujo que está debajo muestra lo que ocurre cuando se cambia el ángulo de la articulación 3 dejando fijo el lado izquierdo y sólo moviendo el derecho. En el dibujo de la parte inferior aparece lo que ocurre si la propagación es hacia la izquierda. El resultado es que el gusano se encuentra en el mismo estado (la forma es la misma) pero la posición ha variado.

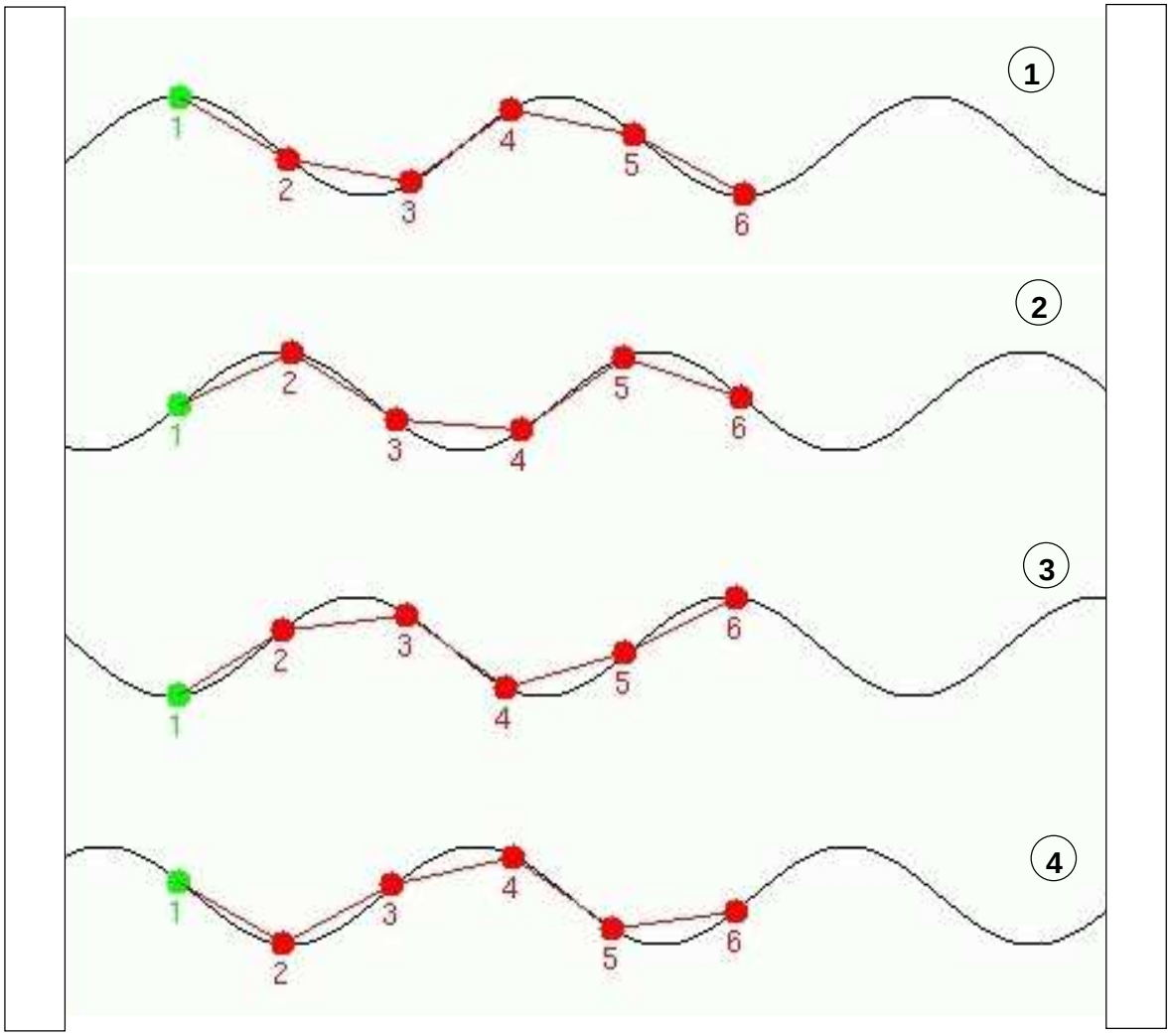


Figura 7.10: Secuencia de 4 estados

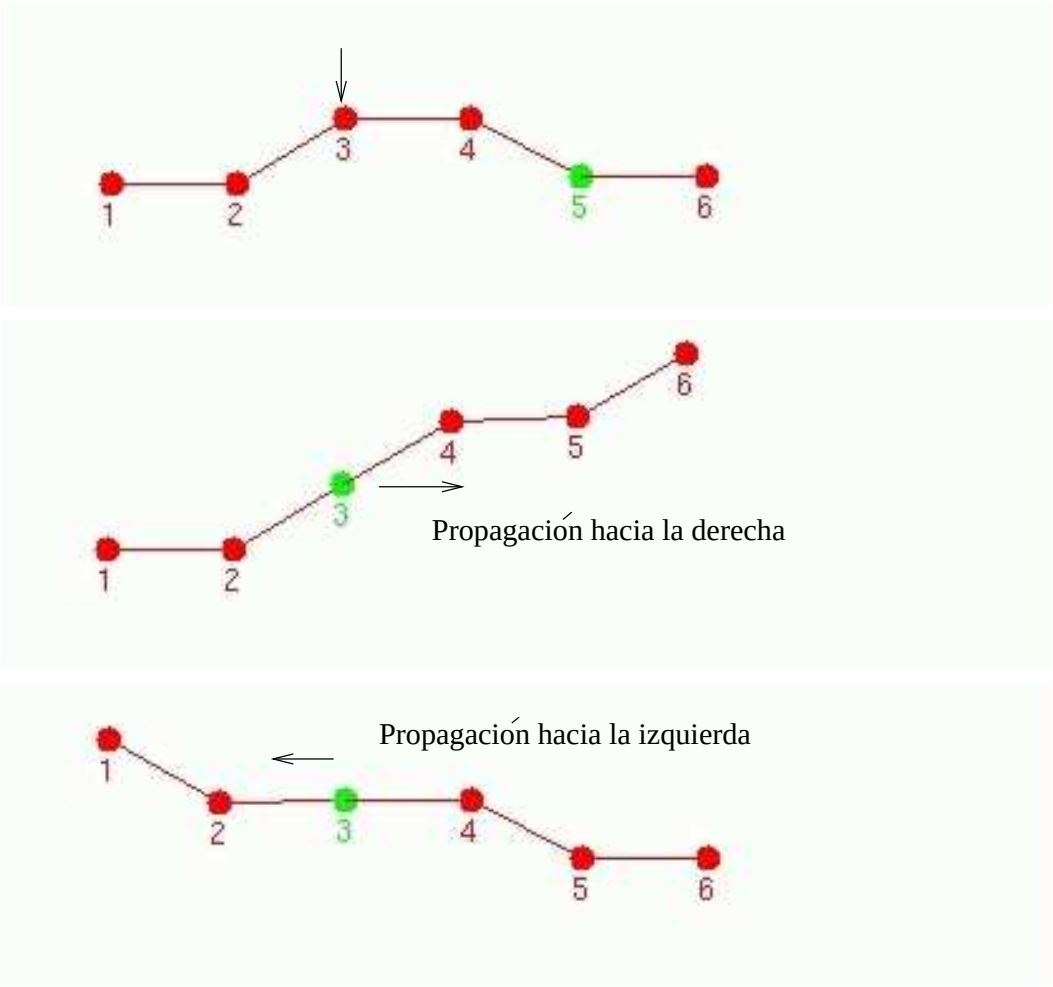


Figura 7.11: Propagación del estado hacia la ezquierda y la derecha

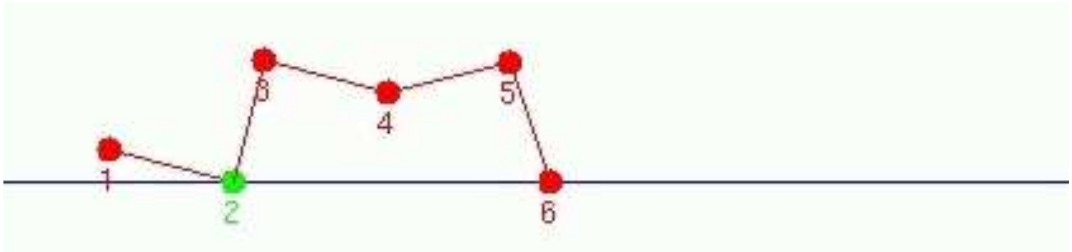


Figura 7.12: Gusano en un estado alcanzado “manualmente”

En la figura 7.12 se ha situado manualmente el gusano virtual en un estado, moviendo cada una de las articulaciones mediante la barra de desplazamiento.

11. **Grupo 11**: Botón de *enganche*

    Denominaremos enganche a la asociación de los estados internos del gusano con el estado de la función de contorno. La función de contorno está caracterizada por los parámetros descritos anteriormente: amplitud, longitud de onda, tiempo y velocidad. Tanto los estados de la función de contorno como los del gusano interno se pueden alterar individualmente sin que afecten uno al otro. Sin embargo al apretar el estado de enganche, se asocian ambos estados, de manera que al cambiar el estado de la función de contorno se cambia el estado del gusano.

    Una vez que el gusano está enganchado a la función de contorno, podemos emplear los botones del grupo 8 para actuar sobre el gusano. Aumentando el parámetro tiempo, el estado interno avanza. Los botones del grupo 9, *play* y *grabar*, lo que hacen es primero "enganchar" el gusano y luego variar el parámetro tiempo. El "enganche" no sólo es a nivel temporal sino que funciona con cualquiera de los otros parametros: amplitud y longitud de onda.

En la figura 7.13 se muestra un gusano que está enganchado a una función de contorno sinusoidal y cómo al variar la amplitud el gusano va modificando su estado.

7.5.5. Interfaz para conversiones

Este interfaz se utiliza para relacionar el modelo virtual de gusano con el modelo físico. Los conceptos se tratan en la sección 7.6. Está constituido por tres partes:

- **Correccion de offset**: Este grupo permite establecer las correcciones de offset necesarias para las articulaciones reales (articulaciones números 2-5).
- **Corrección de sentido**: Botones de estado para corregir el sentido de la articulación
- **Vector de estado físico**: Presenta el estado del vector físico real, de aplicación para el gusano físico concreto.

7.5.6. Generación de secuencias

Para generar una secuencia de movimiento en el gusano virtual para luego ser grabada en un fichero y leida por la aplicación cube-físico hay que seguir los siguientes pasos:

1. Seleccionar la función de contorno, utilizando los botones F1, F2 ó F3. Normalmente se utilizará la que viene por omisión que es una onda sinusoidal.
2. Establecer los parámetros de la onda: amplitud, longitud de onda y velocidad de propagación. Para ello se utiliza el grupo 8 de los botones de interfaz

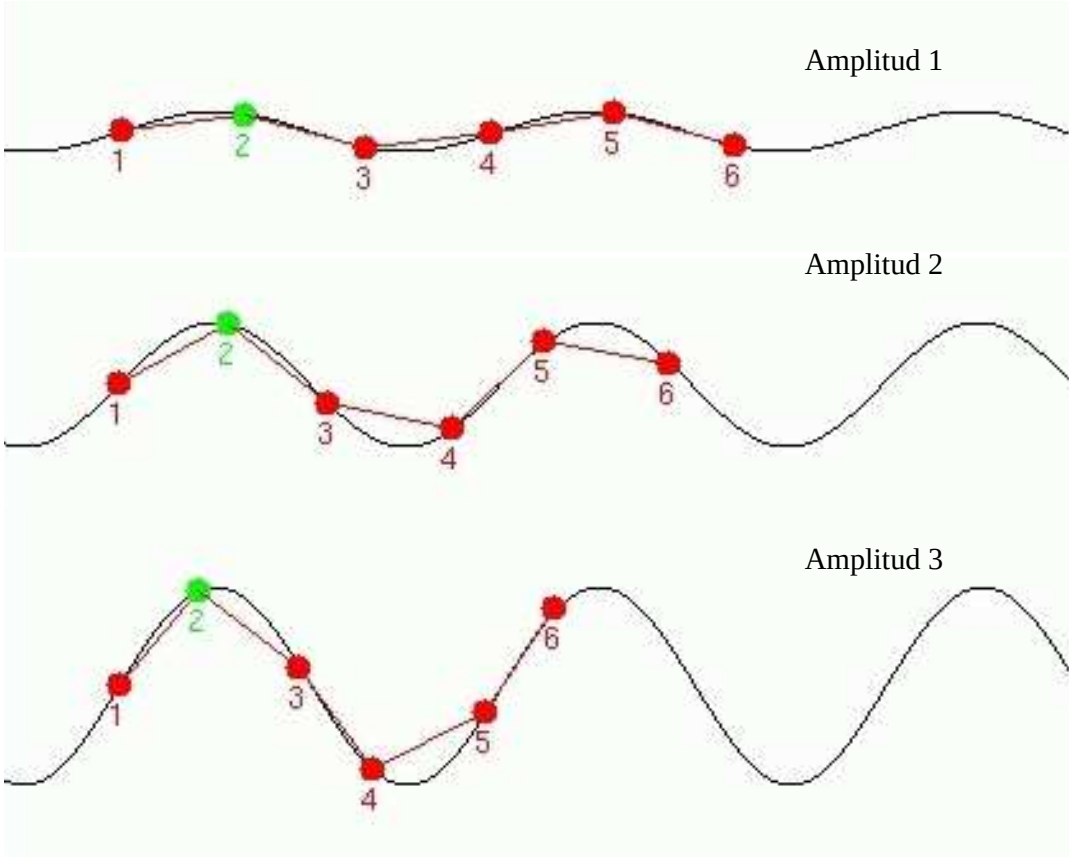


Figura 7.13: Gusano enganchado a una función de contorno a la que se le varía la amplitud

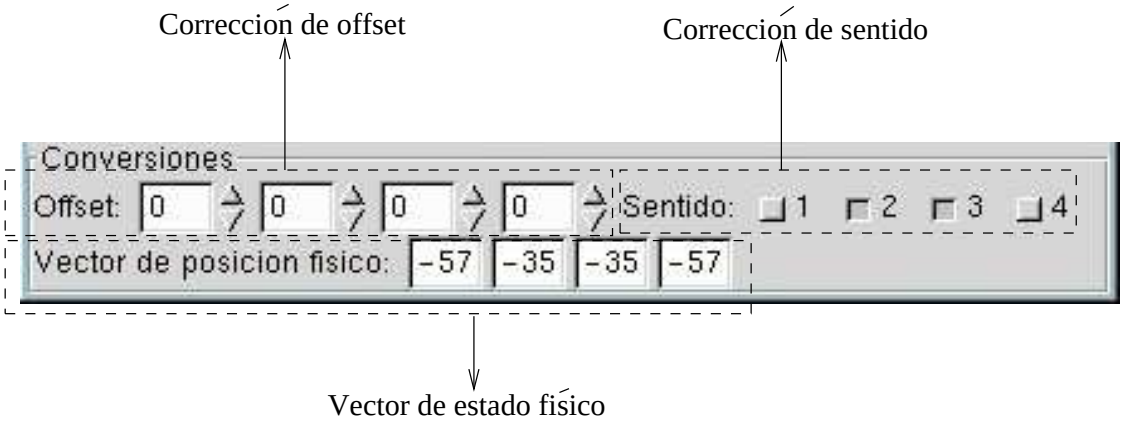


Figura 7.14: Interfaz para las correcciones del estado

3. Probar la secuencia en el gusano virtual, bien automáticamente pulsando el botón de play o bien manualmente, incrementando el tiempo usando el grupo 8.

4. Pulsar el botón de grabación “grabar”. A continuación se indica el fichero en el que se quiere grabar la secuencia. Se grabará un periodo de la misma.

7.6. Gusano virtual y gusano físico

Entendemos por **gusano virtual** el modelo de gusano transversal y su simulación, y por **gusano físico** la implementación en un gusano real. Los cálculos de los estados internos se realizan sobre el gusano virtual para luego ser llevados al gusano físico. Por ello es necesario establecer la relación entre los dos modelos.

7.6.1. El estado del gusano virtual

El estado del gusano virtual queda caracterizado por el vector de estado (E), que tiene como componentes los angulos de todas las articulaciones. El modelo virtual presentado en el software cube-virtual está constituido por 6 articulaciones y 5 segmentos. Sin embargo, dos de las articulaciones, la 1 y la 6 son virtuales, puesto que en el modelo físico no existen. Estas articulaciones sirven para determinar la posición del gusano virtual con respecto al eje x, para un mismo estado, pero realmente no están definiendo un estado interno, sino una orientación.

El conocer las coordenadas (x, y) de las articulaciones del gusano virtual es interesante para evaluar las diferentes funciones de contorno que se emplean para generar el movimiento. Nos permite por ejemplo conocer la longitud del gusano longitudinal asociado, observando la coordenada x de la articulación 6, cuando se encuentra sobre el eje x. Para un correcto avance esta longitud debe permanecer constante.

En la figura 7.15 se muestran dos gusanos en los que lo único en que difieren es en el estado de la articulación virtual 1.

7.6.2. El estado del gusano físico

El estado del gusano físico viene determinado por los ángulos de sus servomecanismos, o de sus articulaciones “reales”. En este caso son 4, por lo que el vector de estado físico (E’) tiene sólo 4 componentes. Para el gusano físico no nos interesan las coordenadas (x, y), puesto que no podemos actuar sobre ellas, ni tampoco determinarlas. Sólo disponemos del vector de estado físico.

7.6.3. Conversiones entre los estados

De nada serviría el modelo virtual si no se pudiese aplicar a un gusano físico. Para ello es necesario hacer una serie de conversiones. **La primera conversión es una proyección**, para pasar de un vector de estado de 6 componentes a otro de 4. Recordemos que las articulaciones de

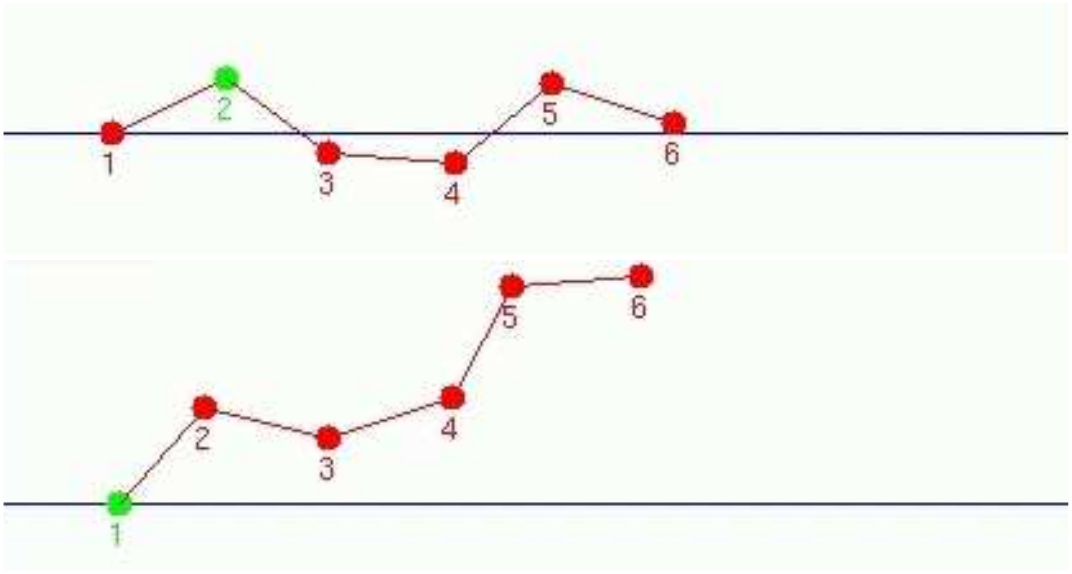


Figura 7.15: Gusanos con diferente estado de la articulacion 1

los extremos (la 1 y la 6) son “virtuales” y que no tienen sentido en un gusano físico. Se utilizan en el modelo virtual para facilitar los cálculos. Esta proyección la definimos por la función $\Pi(E)$. De esta manera, dado un vector $E = (v_1, v_2, v_3, v_4, v_5, v_6)$, se obtiene el vector físico $E' = (v_2, v_3, v_4, v_5)$:

$$E' = \Pi(E)$$

Sin embargo es necesario hacer más conversiones. El gusano virtual es perfecto, pero el físico hay que “calibrarlo”. Por imprecisiones mecánicas, el estado físico (0,0,0,0) puede no corresponderse exactamente con el estado (0,0,0,0,0,0). Para que la correspondencia sea lo más precisa posible, hay que aplicar un vector que compense las imprecisiones mecáncias. Este vector lo denominaremos vector de offset O. Si la calibración mecánica fuese perfecta, el vector de offset sería O=(0,0,0,0). De esta manera, el vector de estado físico lo obtenemos aplicando una **segunda conversión**, sumando el vector de offset:

$$E' = \Pi(E) + O$$

Pero queda una tercera conversión. El gusano físico se puede implementar de muchas formas. Los servos pueden estar orientados con el eje de salida apuntando hacia un lado o hacia otro, dependiendo de cómo se construya. En el gusano virtual se supone que todos los servos están orientados de la misma manera, pero en el real no tiene por qué cumplirse. Si la orientación del servo es la contraria que la de la articulación virtual, hay que cambiar el signo del ángulo al hacer la conversión.

En la figura 7.16 se muestra una articulación virtual en un estado φ y dos servos en el mismo estado. El de la izquierda tiene la misma orientación que la articulación virtual, por lo que se

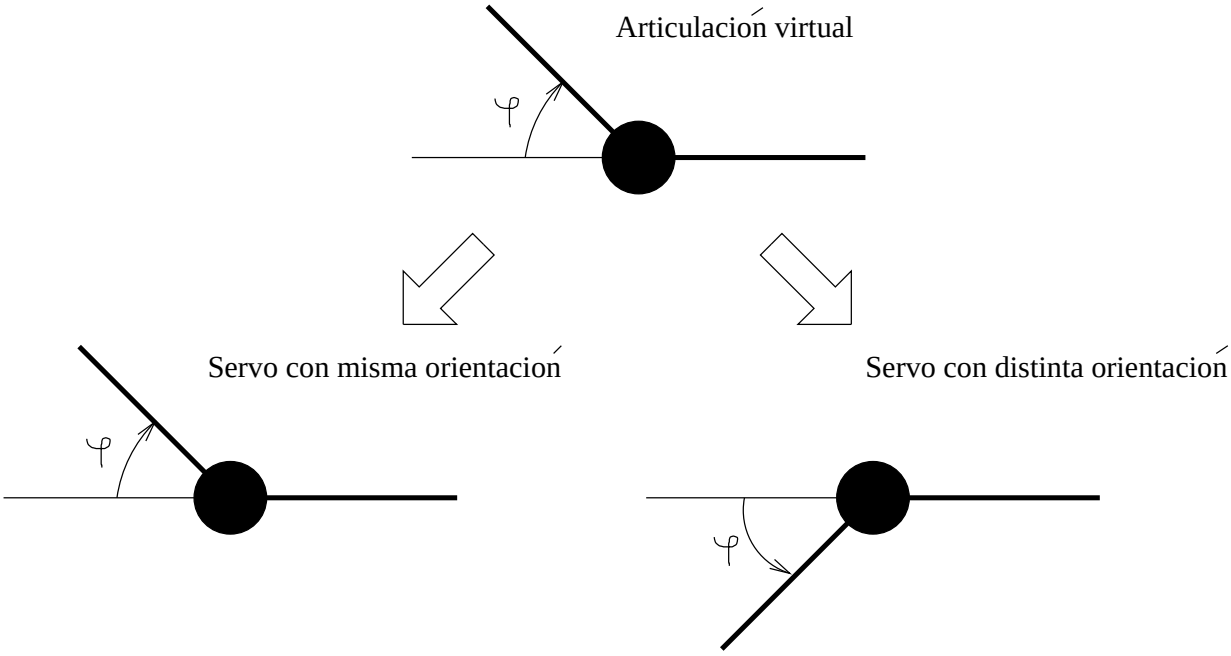


Figura 7.16: Articulaciones virtuales y servos reales

encuentran en la misma posición. Sin embargo el de la derecha está en la otra orientación por lo que al aplicarle el mismo ángulo φ el segmento izquierdo baja en vez de subir. Para que se encontrase en la misma posición que la articulación virtual habría que situarlo en el ángulo $-\varphi$. Por este motivo es necesario realizar una **tercera conversión** cambiando el signo de algunas componentes del vector de estado físico. Esta conversión viene determinada por la función $S(E')$, que toma un vector físico y devuelve otro aplicando las correcciones de signo pertinentes. De esta manera, la conversión final entre los vectores de estado virtuales y físicos es la siguiente:

$$E' = S(O + \Pi(E)) \tag{7.1}$$

donde:

- E': es el vector físico que determina el estado del gusano físico
- E: Es el estado del gusano virtual
- O: Es la corrección de offset
- S: Es la corrección del sentido
- Π: Es la función de proyección para pasar de 6 a 4 componentes.

Cuando se “graba” una secuencia de estados en un fichero, con la opción “grabar” del grupo 9, se están grabando los estados físicos, con las correcciones aplicadas. Estas correcciones se definen en la parte del interfaz conversiones, como se explicó en la sección 7.5.5.

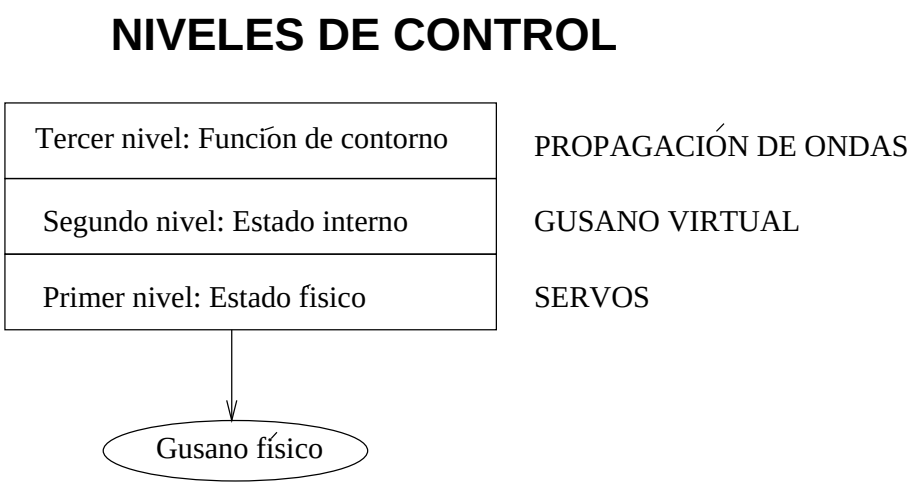


Figura 7.17: Los tres niveles de control del gusano

7.7. Estados y niveles de control

Imaginemos que partimos ahora de un gusano físico que queremos controlar. El **primer nivel de control**, el más bajo o más asociado al gusano físico, está constituido por secuencias de **vectores de estado físicos**, que le indican al gusano en qué estado establecer sus articulaciones. En este nivel, hay que especificar las posiciones de cada **servo**. Para generar una secuencia hay que utilizar algún tipo de software que permita situar los servos en un estado, grabar ese estado, obtener el siguiente estado, grabarlo, etc. Una vez grabados todos los estados, al reproducirlos se genera el movimiento deseado.

A este nivel no se tiene información sobre la coordinación de los diferentes servos. Generar una secuencia nueva es un proceso que requiere paciencia. Y el autor lo sabe bien puesto que la primera versión que se hizo de Cube no pasaba de este primer nivel de control. El movimiento conseguido no era malo, pero como no se satisfacía correctamente la propiedad de que se mantuviese constante la longitud del gusano longitudinal asociado, los servos sufrían mucho y el movimiento no era uniforme y dependía mucho de la superficie, ya que al no cumplirse dicha condición había puntos que se tenían que arrastrar.

El **segundo nivel** viene representado por el **gusano virtual** y la asociación entre su estado y el estado físico. En este nivel es posible situar, también a mano, las articulaciones del gusano virtual y generar el vector físico asociado. Pero al disponer de información sobre las coordenadas (x, y) de las articulaciones se pueden tener en cuenta otros factores, como la longitud del gusano longitudinal asociado. El resultado es una mejor coordinación entre todas las articulaciones y el independizarse del gusano físico: ahora para generar una secuencia no es necesario tener un gusano físico, sino que se trabaja con el virtual y finalmente se prueba con el real.

Hay un **tercer nivel**, representado por la **función de contorno** y el gusano virtual "enganchado". Ahora para generar una secuencia no nos importa el estado del gusano porque se genera automáticamente. Sólo hay que preocuparse de qué parametros emplear para definir la función de contorno: Amplitud, longitud de onda, velocidad y tiempo. La Amplitud y la longitud de on-

da determinan la contracción del gusano, y por tanto la velocidad de avance. También se puede controlar la áltura máxima y mínima alcanzada por el gusano.

Fijados unos parámetros de amplitud, longitud de onda y velocidad, con sólo variar el factor tiempo, se generan automáticamente los estados internos del movimiento y a partir de éstos los estados físicos que hay que aplicar al gusano real. El resultado es que estamos a otro nivel de abstracción. Un nivel en el que no importan los estados internos, pues se generan automáticamente.

7.8. Resumen

Existen dos herramientas software para manejar a Cube: **cube-virtual** que manje un gusano virtual y **cube-fisico** que maneja uno real. Las secuencias generadas por el gusano virtual pueden ser cargadas en la aplicación cube-fisico y que el gusano real las reproduzca.

Este capítulo se centra en la aplicación cube-virtual, que permite manipular el modelo virtual y generar secuencias a partir de diferentes funciones de contorno. Se ha mostrado la arquitectura software, con todos los módulos software desarrollados y cómo se han combiando para lograr la aplicación final. También se ha mostrado con detalle cómo funciona el programa, para que el diseñador lo pueda utilizar para sus propósitos.

El gusano virtual y el físico son diferentes, y aunque las secuencias de uno valen para el otro, es necesario realizar una serie de conversiones. A partir de los **vectores de estado virtuales** se obtienen los **vectores de estado físicos**, que se graban en un fichero se pueden ser leídos por la aplicación cube-físico.

Finalmente se ha hecho una reflexión sobre los diferentes niveles de abstracción del gusano y cómo con este programa se puede llegar al nivel superior: controlar el gusano mediante funciones de contorno y sus parámetros asociados, haciendo abstracción de todos los detalles intermedios y de bajo nivel.

Capítulo 8

Estructura mecánica

8.1. Introducción

En este capítulo se aborda la construcción mecánica de un gusano transversal, sobre el que se pueda aplicar el modelo de gusano virtual presentado en el capítulo 7. La estructura mecánica debe cumplir los siguientes requisitos:

1. **Modularidad**. El gusano se debe construir mediante módulos iguales, de manera que sea muy fácil su ampliación.
2. **Simplicidad**. Que las piezas sean sencillas y fáciles de obtener y que exista el menor número posible de piezas diferentes.
3. **Economía**. Las piezas deben ser lo más baratas posibles y los materiales fáciles de encontrar en el mercado.
4. **Robustez**. Buscar que la estructura sea lo más resistente posible y que las piezas se acoplen sólidamente.

Un libro muy bueno de robótica en general, más orientado hacia temas prácticos que teóricos es [28].

8.2. Descripción de los prototipos

8.2.1. Primer prototipo: cube-1.0

El primer diseño, **Cube 1.0,** se realizó en madera de 2mm y 3mm de espesor. Este material es muy bueno para cortar y taladrar, pero tiene un inconveniente que no se había previsto en un principio: se ensucia mucho y no es posible lavarlo, salvo que las piezas se barnicen. Esta primera versión, además, no era modulable. Se pensó sólo para cuatro articulaciones. Si se querían añadir más, había que diseñar piezas nuevas.

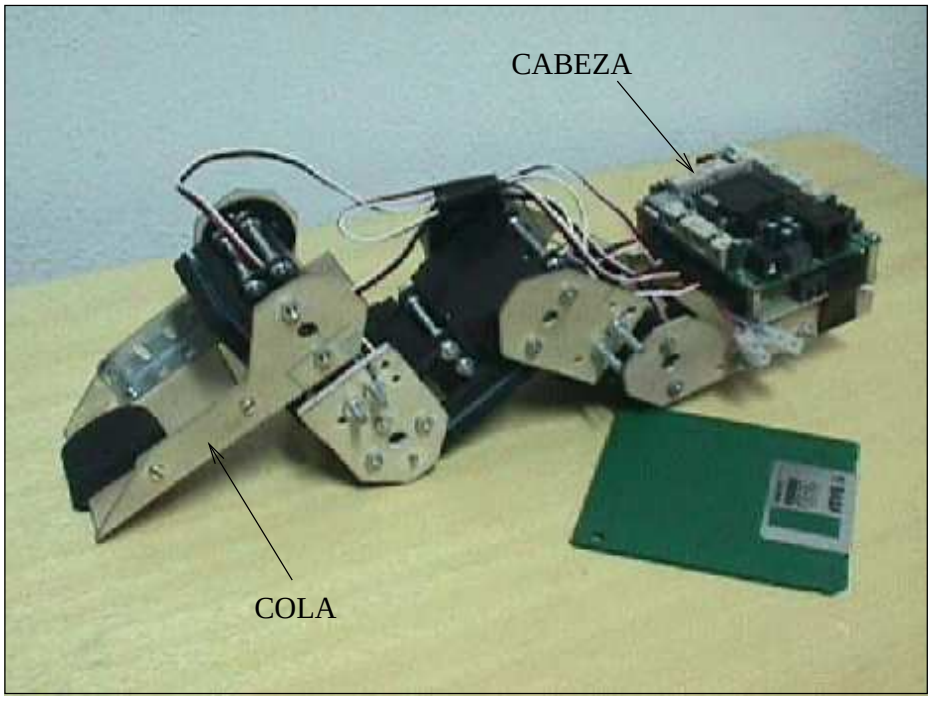


Figura 8.1: Primera version del gusano: cube-1.0

Este primer prototipo permitió comprobar la viabilidad mecánica del proyecto, así como encontrar fallos para ser mejorados en la versión 2.0, que es la que se presenta en este proyecto.

En la figura 8.1 se muestra cube-1.0, al lado de un disquete de 3.5” para apreciar su tamaño real. Se han señalado con flechas la cola y la cabeza.

8.2.2. Segundo prototipo: cube-2.0

Para construir las diferentes piezas de **Cube 2.0**, se ha empleado un material plástico, transparente, que se corta muy fácilmente con una segueta, los taladros se realizan con facilidad y es lo suficientemente robusto. Estas mismas piezas se pueden realizar en aluminio, pero en ese caso no se cumple el requisito de la *simplicidad*, puesto que es necesario disponer de una sierra especial para cortarlas, y taladrarlas no es tan sencillo.

En la foto de la figura 8.3 se muestran las dos estructuras. Aparentemente son muy parecidas, pero la versión en plástico está mucho más elaborada. Cube 2.0 está pensado para ser clonado y ampliado fácilmente. Se puede ver en la figura 8.2.

A lo largo de este capítulo se hará referencia a Cube 2.0, salvo que se indique lo contrario.

8.3. Servomecanismo futaba 3003

El gusano está diseñado basado en los servomecanismos Futaba 3003, por lo que siempre que se haga referencia a la palabra servo o servomecanismo hay que tener en mente este modelo.

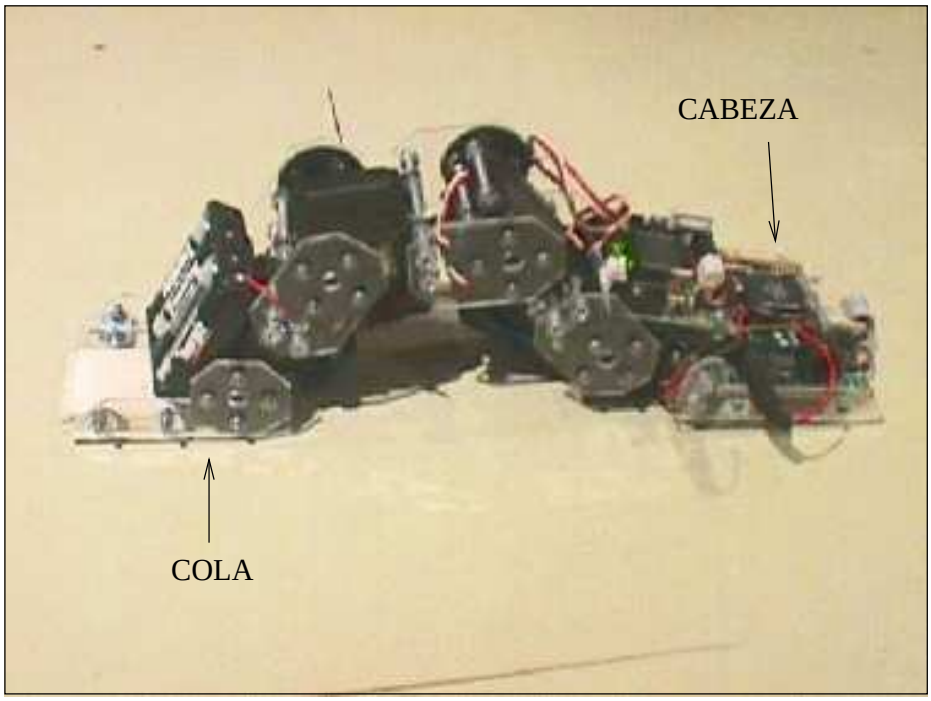


Figura 8.2: Foto de Cube 2.0

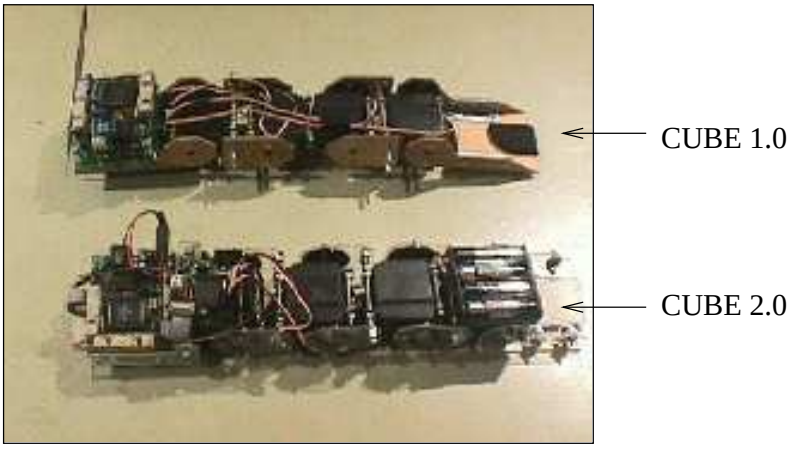


Figura 8.3: Foto con las estructuras mecánicas de Cube 1.0 y Cube 2.0

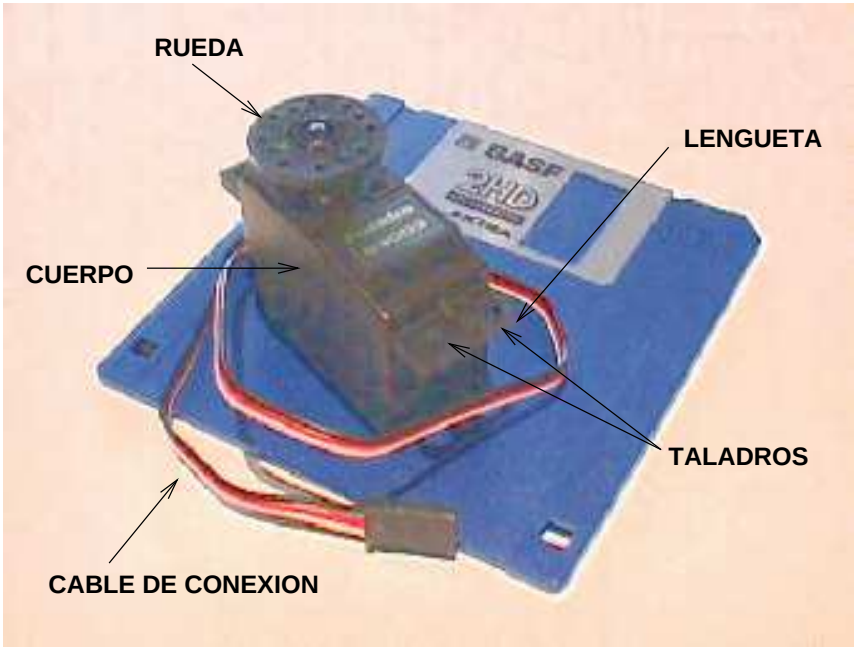


Figura 8.4: Foto de un servo modelo futaba 3003

En la figura 8.4 se muestra una foto, tomada junto a un disquete de 3.5 pulgadas, donde se puede apreciar el reducido tamaño del servo.

Estos servos tienen un **cuerpo** y un eje de salida con una **rueda** atornillada a él, de tal manera que el cuerpo y la rueda rotan uno respecto del otro. Dentro del cuerpo se encuentran los engranajes, el circuito de control y el motor. La rueda es extraible, quitando el tornillo que lo sujeta al eje. En ella se realizan los taladros necesarios para unirse al resto de las piezas, como se explica más adelante.

En el cuerpo existen dos **lengüetas** que sobresalen, con dos taladros cada una. Esto permite atornillarlo a cualquier estructura.

.

8.4. Modelo mecánico

CUBE está constituido por tres partes: **cabeza**, **cuerpo** y **cola**. La cabeza y la cola son las partes terminales del gusano, mientras que el cuerpo puede crecer cuanto se quiera, ya que está compuesto por módulos iguales que se repiten. En la figura 8.5 se muestran todas las partes. Sólo existen tres tipos de piezas diferentes, denominadas piezas de tipo A, B y C.

8.4.1. Cabeza

La cabeza está formada por dos piezas de tipo C, unidas a la base rectangular por medio de cuatro escuadras metálicas y al cuerpo del futaba por dos varillas roscadas de 9.5cm de longitud.

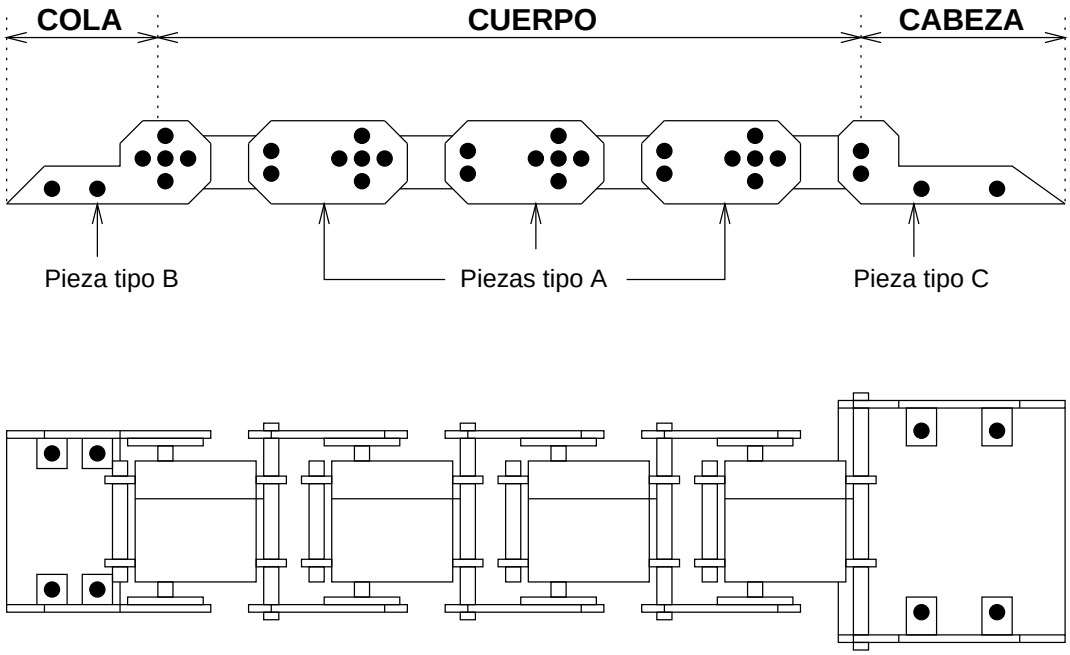


Figura 8.5: Planta y alzado de Cube

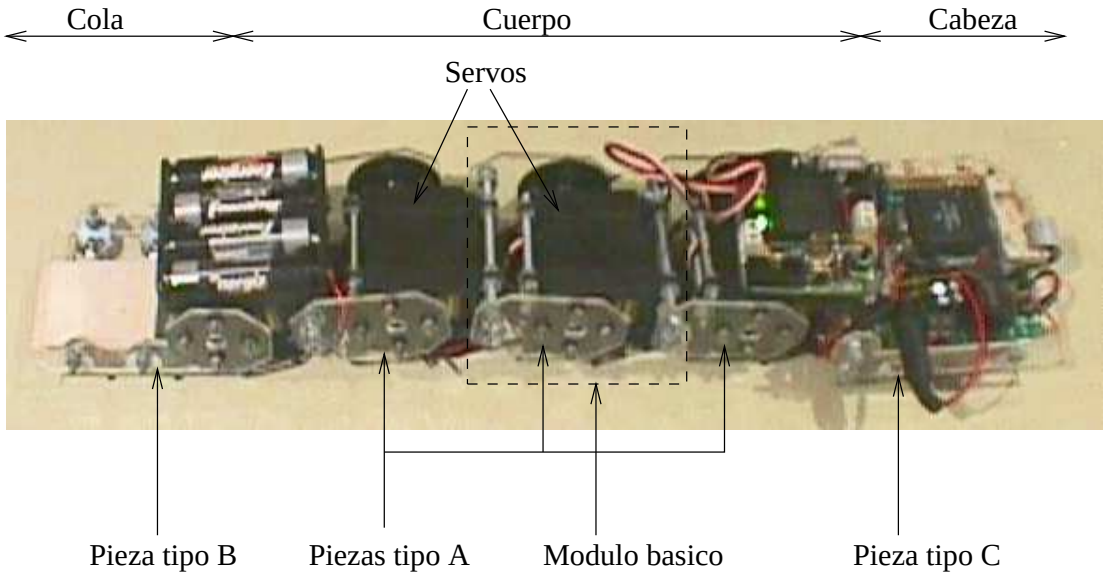


Figura 8.6: Foto de la estructura de cube

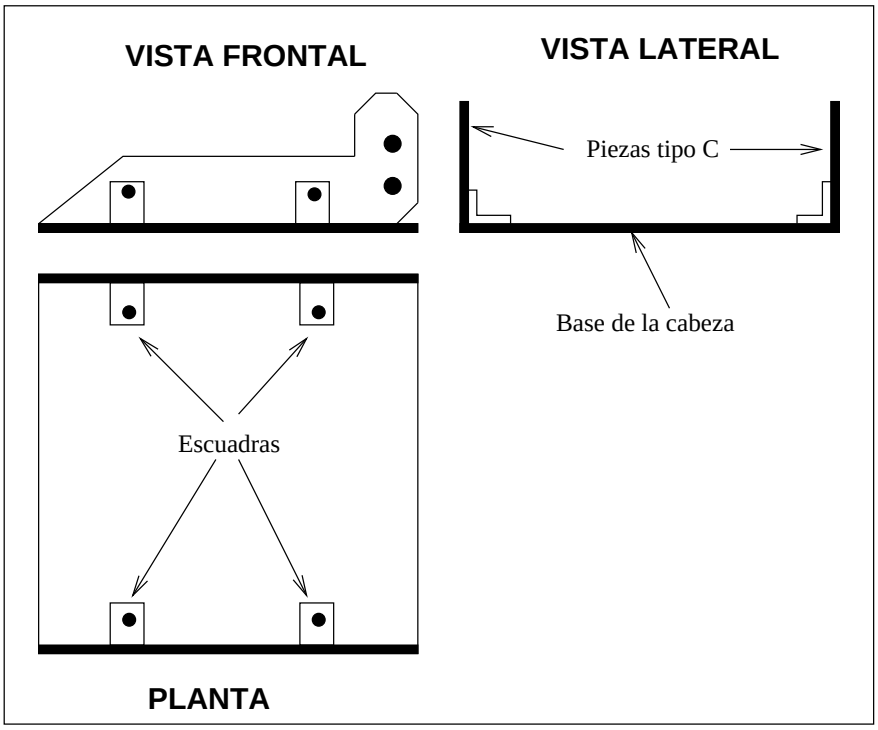


Figura 8.7: Diferentes vistas de la cabeza de Cube

En la figura 8.7 se muestran las diferentes vistas y en la 8.8 aparece en perspectiva. En ambas figuras no se muestras las varillas roscadas, pero sí sus taladros

8.4.2. Cola

La cola es muy similar a la cabeza, pero es más pequeña y está unida al eje de salida del futaba, en vez de a su cuerpo, como lo está la cabeza. Las piezas de tipo B se unen a la base por medio de dos escuadras y al eje de la articulación mediante cuatro tornillos. Ver figuras 8.9 y 8.10.

8.4.3. Cuerpo

El cuerpo es la parte más importante de Cube. Se caracteriza por estár formado por módulos iguales, denominados **módulos básicos**, lo que permite que sea extensible. Cada módulo básico está constituido por una articulación de doble eje y dos piezas de tipo A, que se emplean para unirse al siguiente módulo.

8.4.3.1. Módulos básicos

El módulo básico está formado por una articulación de doble eje y dos piezas de tipo A atornilladas a cada eje de salida de la articulación, como se muestra en las figuras 8.11 y 8.12.

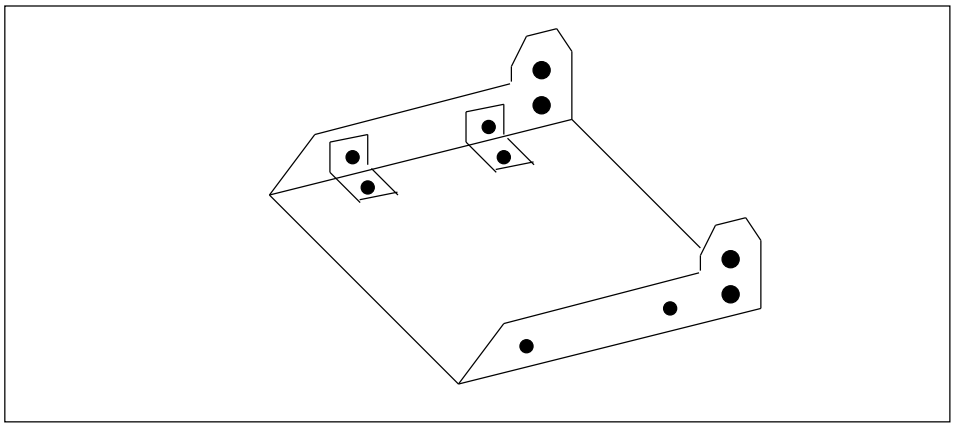


Figura 8.8: La cabeza vista en perspectiva

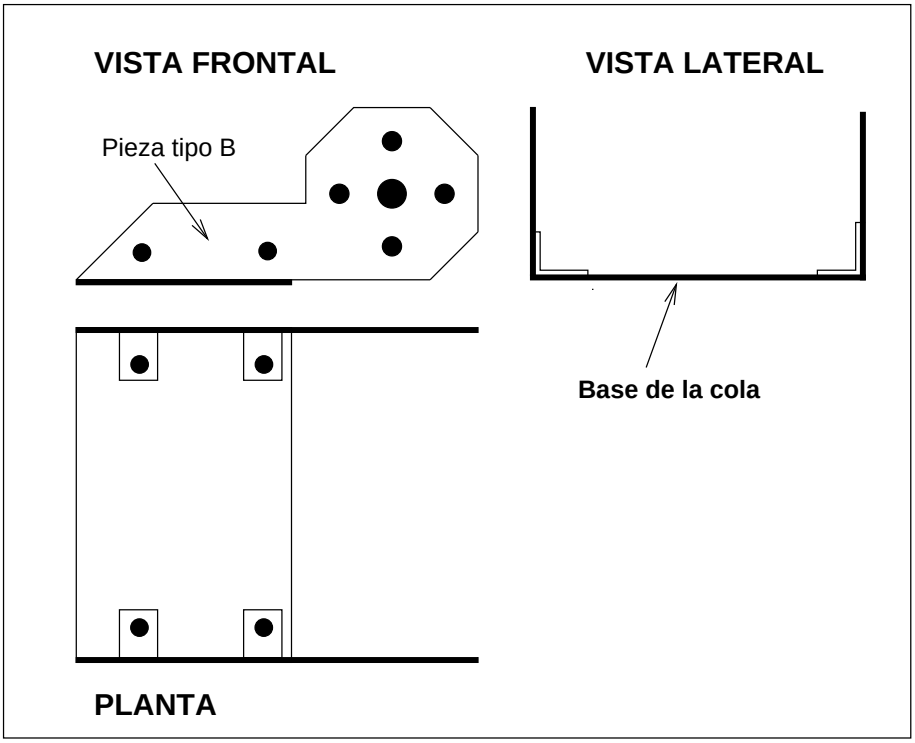


Figura 8.9: Diferentes vistas de la cola de Cube

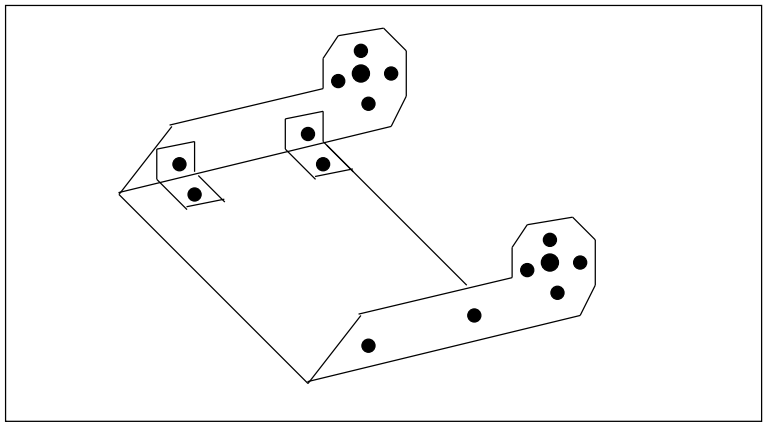


Figura 8.10: La cola de Cube vista en perspectiva

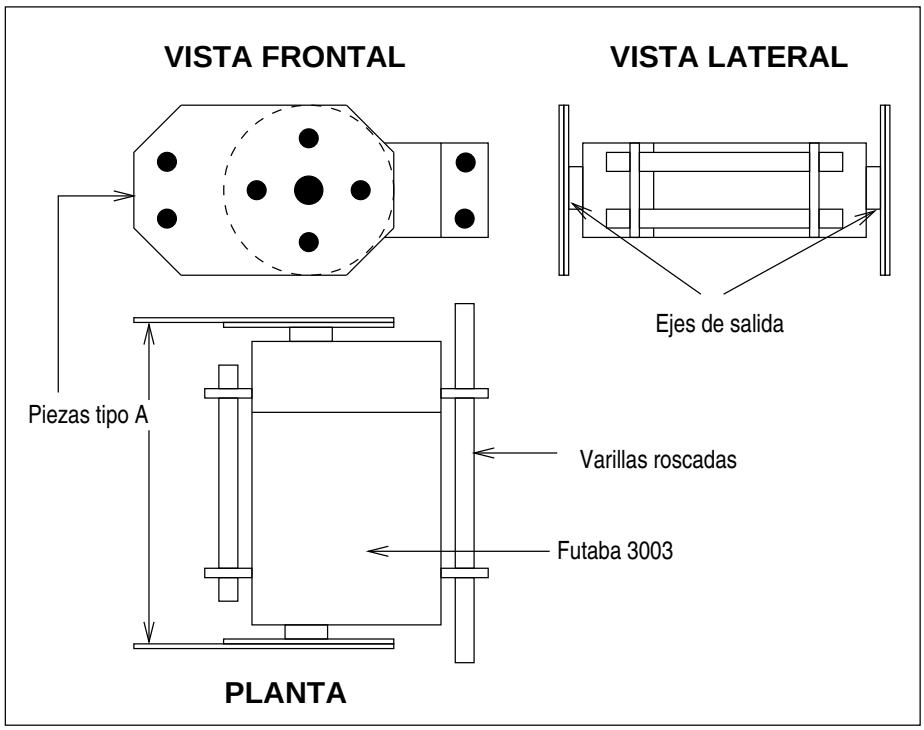


Figura 8.11: Módulo básico, en diferentes vistas

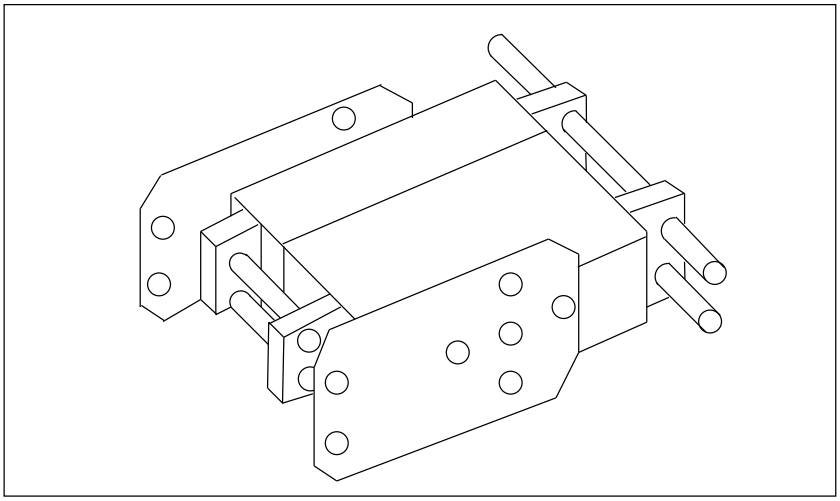


Figura 8.12: Módulo básico visto en perspectiva

Estos módulos se pueden unir unos a otros formando el cuerpo entero del gusano. Las piezas de tipo A se unen al cuerpo del siguiente módulo mediante dos varillas roscadas de 7.5cm.

8.4.3.2. Articulaciones de doble eje

Para la realización de Cube, como también de otros ingenios articulados, es muy útil el disponer de una articulación con doble eje. Uno de los ejes es el que está conectado al servo y el otro es un falso eje, que no ejerce fuerza pero permite que la articulación sea simétrica y que el peso que soporta la articulación se reparta de igual manera entre ambos ejes, mejorándose la estabilidad mecánica (Ver figura 8.13). Para construir una articulación de estas con un servo de tipo futaba 3003, que sólo posee un eje de salida, es necesario añadir una **pieza con el falso eje**. Esta pieza se puede construir a partir de los elementos que componen el propio futaba 3003.

El servo 3003 está constituido por un motor de corriente continua, un circuito de control, engranajes, una carcasa que lo soporta, una tapa superior donde se encuentra el eje de salida, una rueda que se adapta al eje de salida y una tapa inferior.

Tanto la carcasa, como las tapas superior e inferior y los engranajes se pueden conseguir en el mercado por separado. Para construir la pieza del falso eje es necesaria la tapa superior, la inferior, un engranaje con el eje de salida, la rueda y un tornillo de 4mm de diámetro y 2cm de longitud (Figura 8.14). El tornillo es necesario para que el engranaje se apoye sobre la tapa inferior y que el eje de salida no se pueda desplazar verticalmente. Una vez que se tiene esta pieza montada, hay que utilizar cinta aislante para unir la tapa inferior y la superior. Por medio de las varillas roscadas se une el servo con la pieza, consiguiéndose una articulación de soble eje muy sólida y muy sencilla de construir.

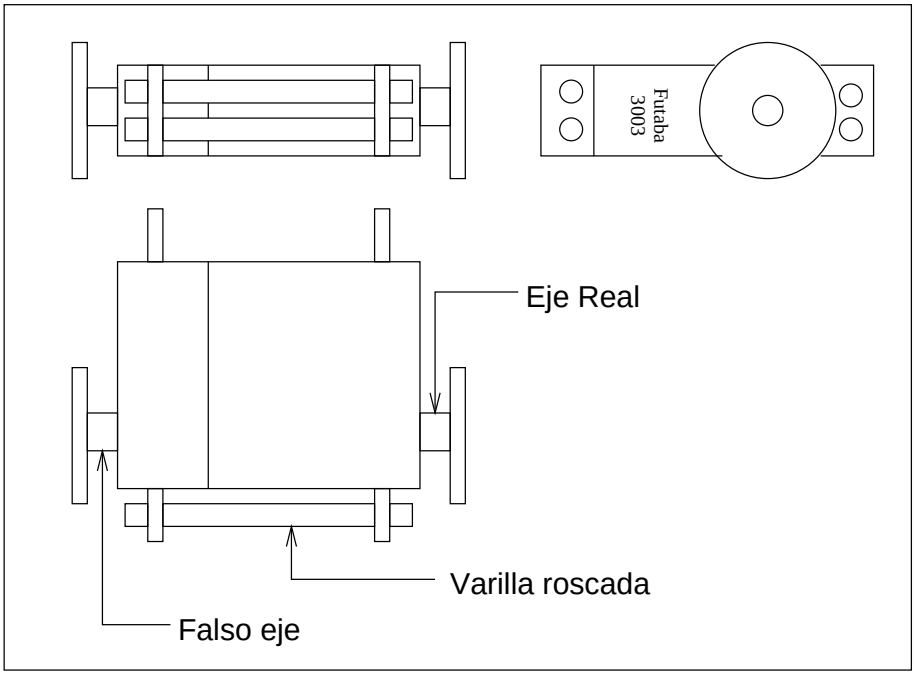


Figura 8.13: Diferentes vistas de una articulación con doble eje

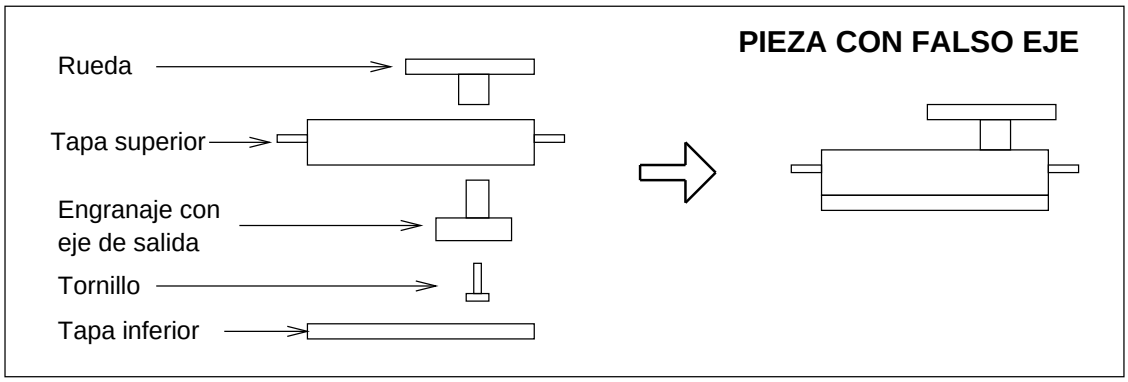


Figura 8.14: Construcción de la pieza con falso eje

8.5. Construcción de Cube

A continuación se listan todos los pasos que hay que seguir para construir la estructura mecánica de cube 2.0. Para la colocación de la electrónica en la estructura, consultar el capiítulo 9. En el apéndice A hay un listado completo de todas las piezas y materiales necesarios.

1. Construir todas las **piezas de plástico** y hacer los taladros correspondientes. Para ello tomar como plantillas los planos de las piezas adjuntos a esta memoria. Las piezas necesarias son:

   a) Seis piezas de tipo A, para la unión de los módulos básicos

   b) Dos de tipo B, para la cola

   c) Dos de tipo C para la cabeza

   d) Base de la cabeza

   e) Base de la cola

2. Cortar las **varillas roscadas** para obtener:

   a) Ocho de 4.6cm, para construir las articulaciones de doble eje

   b) Seis de 7.5cm, para la unión de los módulos básicos

   c) Dos de 9.5cm, para unir el cuerpo con la cabeza

   En la figura 8.15 se muestran todas las piezas necesarias.

3. Construir las **articulaciones de doble eje**. Para ello seguir los siguientes pasos:

   a) Montar la pieza con falso eje (apartado 8.4.3.2 y figura 8.14). Las piezas necesarias se pueden conseguir en una tienda de aeromodelismo donde vendan los servos Futaba. Las piezas necesarias son:

      1) Rueda del servo
      2) Tapa superior
      3) Engranaje con eje de salida
      4) Tapa inferior
      5) Tornillo de 4mm de diametro y 2cm de longitud

   b) Colocar cinta aislante negra rodeando la tapa superior y la inferior para que queden sujetas. Esta unión es provisional, nos sirve para que no se desmonte la pieza mientras la unimos al servo mediante las varillas roscadas.

   c) Unir el servo y la pieza con falso eje. Son necesarias dos varillas de 4.6cm y dos de 7.5cm. Para el futaba que se une con la cabeza son necesarias dos varillas de 9.5cm, en vez de las de 7.5cm. Por cada varilla corta hay que emplear cuatro tuercas, dos en la parte interior y dos en la exterior como se muestra en la figura 8.16. Por cada varilla larga se emplean seis tuercas.

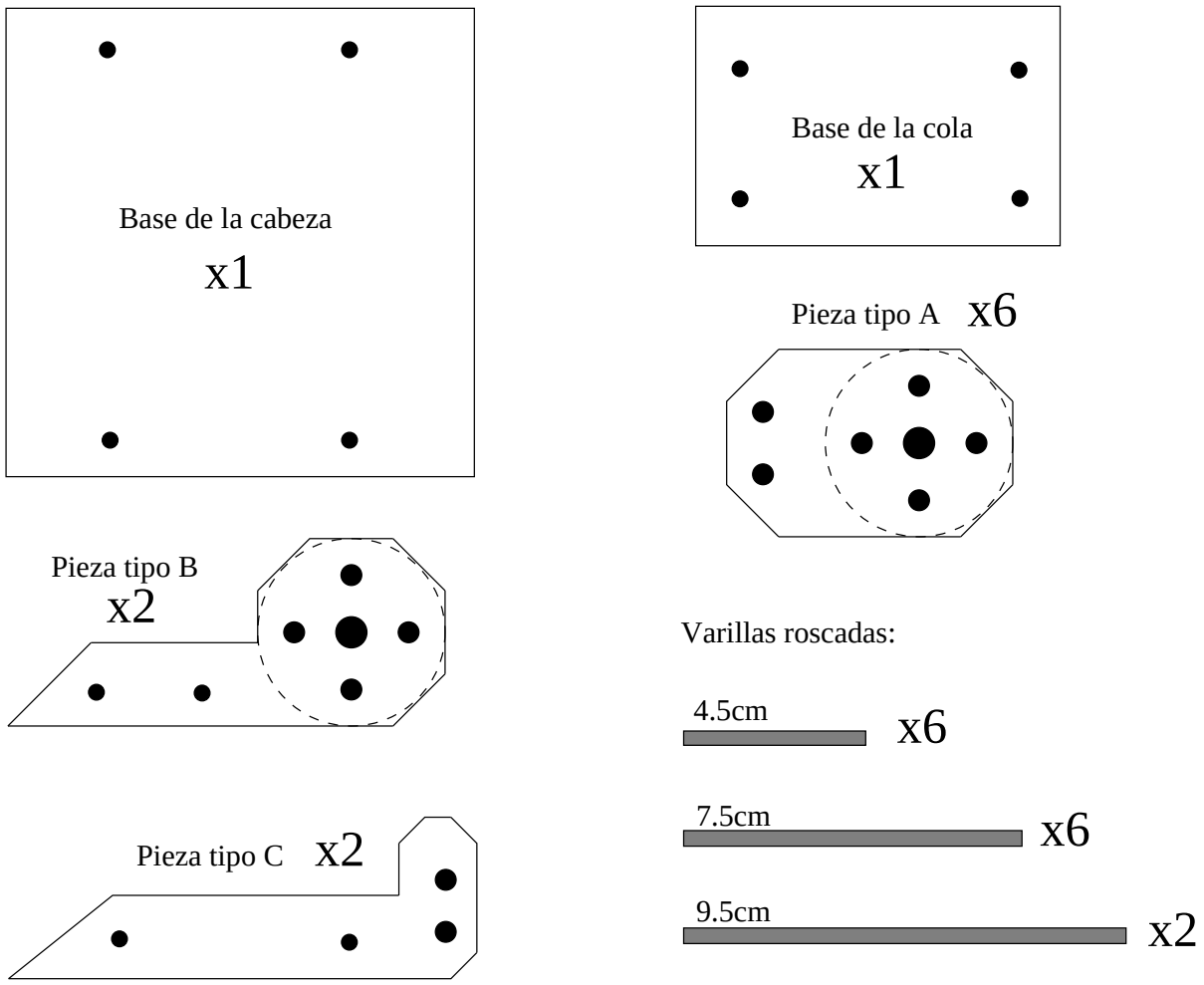


Figura 8.15: Todas las piezas necesarias para construir Cube-2.0

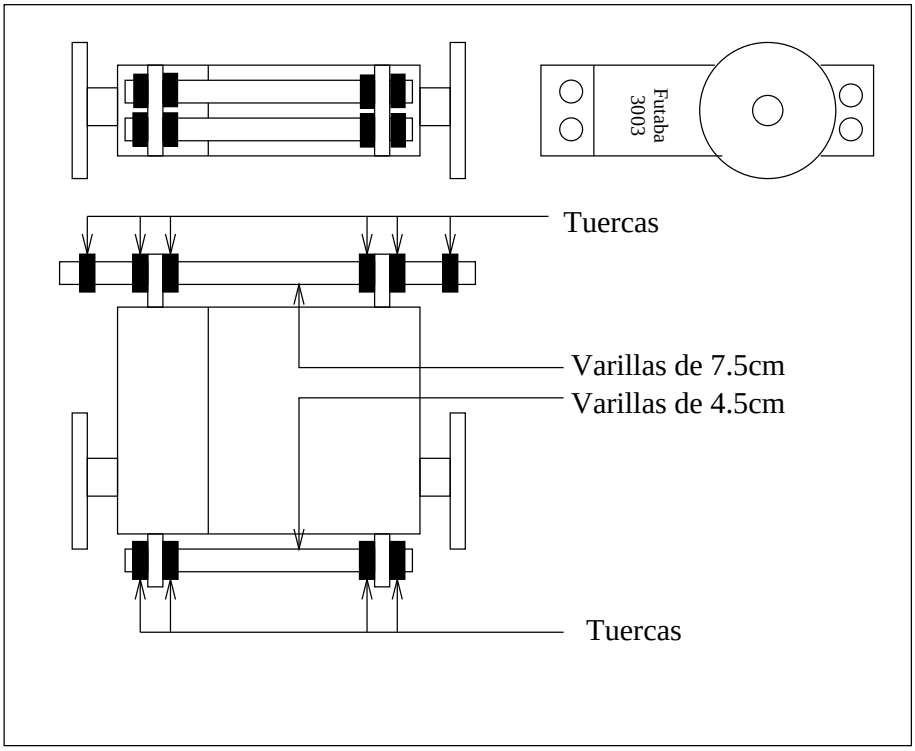


Figura 8.16: Aspecto de la articulación de doble eje una vez montada

4. Realizar los taladros en las ruedas de los servos. Para ello utilizar la plantilla de las piezas de tipo A, a las que irán atornillas las ruedas. Hay que realizar 4 taladros para el anclaje a las piezas de tipo A y uno mayor para insertar el tornillo que une la rueda con el eje del servo. También hay que limar el saliente que hay para que se puedan colocar las piezas tipo A en el siguiente paso (ver figura 8.17)

5. Atornillar las piezas de tipo A a las ruedas de los futabas. Para ello es preciso limar el saliente que hay en la cara exterior de las ruedas. Hay que atornillar las seis piezas de tipo A y las dos de tipo B.

6. Calibrar el gusano. Colocar cuatro ruedas en los falsos ejes, tres ruedas, con sus piezas A enganchadas en los servos 2, 3 y 4, y una rueda con la pieza B al servo 1 (ver figura 8.18). Con el programa **`cube-fisico`** situar todas las articulaciones en el estado de reposo (cero grados). Este estado se corresponde con el del gusano en posición horizontal. Asegurarse de que el falso eje se sitúa en el mismo estado que el eje normal y que el recorrido que realizan es el mismo. Atornillar las ruedas a los servos.

7. Unir todos los módulos del cuerpo, mediante las piezas tipo A. Apretar bien las tuercas. El servo 1 debe llevar atornilladas las dos piezas de tipo B. En la figura 8.19 sólo se han dibujado las tuercas necesarias para la unión.

8. Montar la base de la cola, usando las escuadras para su unión a las piezas de tipo B.

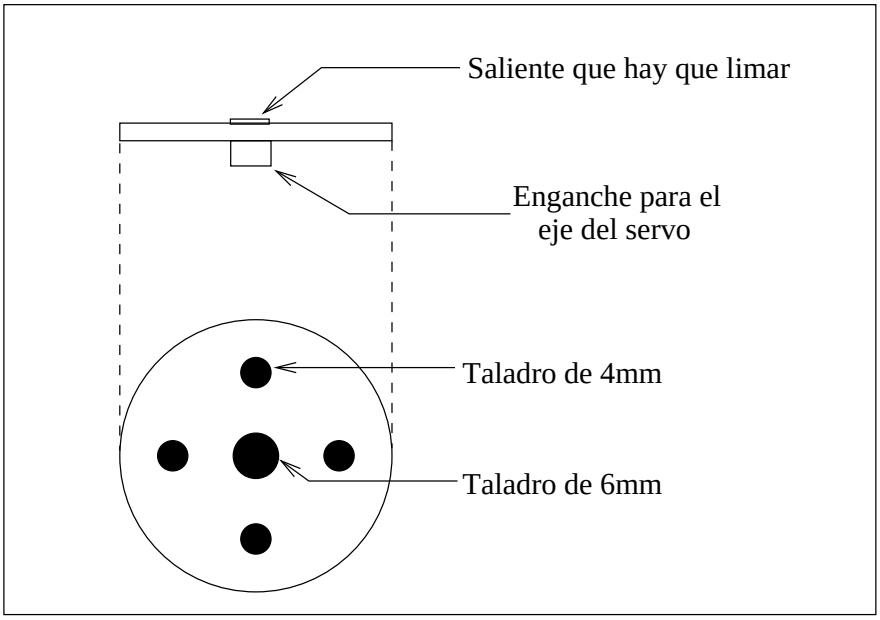


Figura 8.17: Rueda del futaba, con los taladros que hay que realizar

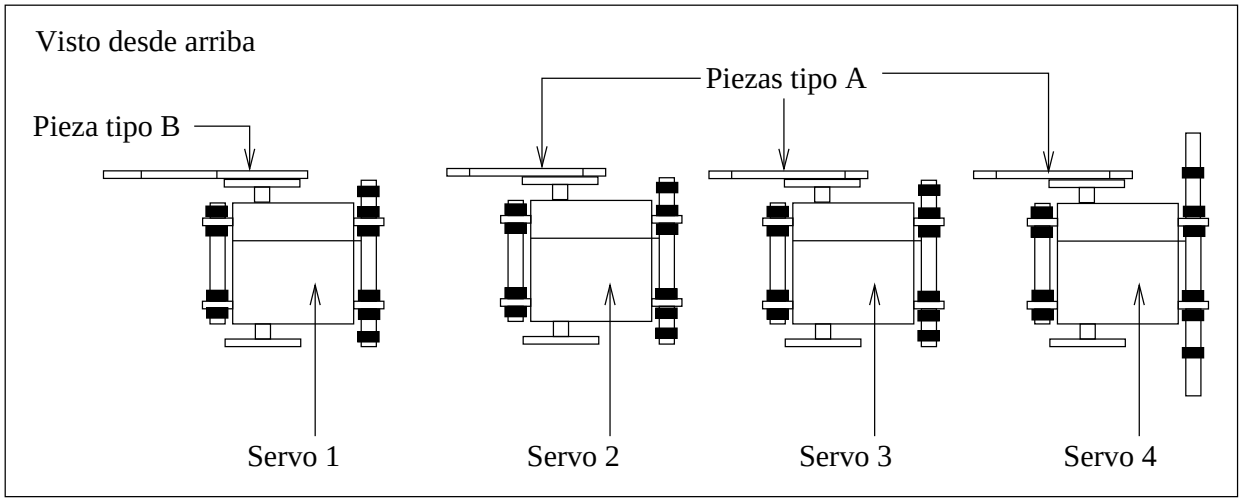


Figura 8.18: Paso 6 del montaje

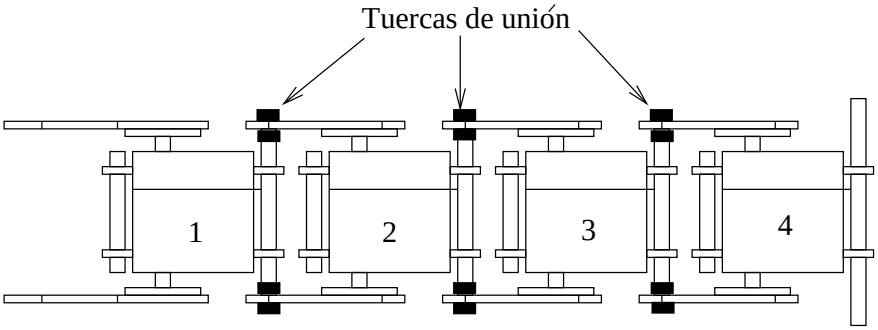


Figura 8.19: Paso 7: Unión de los módulos básicos.

9. Montar la cabeza, uniéndola a las piezas de tipo C mediante cuatro escuadras.

10. Atornillar la cabeza al cuerpo.

11. Pegar los rectángulos de alfombrilla de ratón dos a dos, de manera que se obtengan 4 bloques más gordos.

12. Unir los bloques de las alfombrillas a los módulos básicos mediante velcro, de manera que se puedan poner y quitar. Estos bloques sirven de base al gusano, para que se apoye correctamente sobre la superficie. En la figura 8.20 se muestra la estructura terminada, sin la electrónica.

8.6. Resumen

Se han desarrollado dos prototipos. El primero, **Cube-1.0**, estaba realizado en madera y no era ampliable, sin embargo fue muy útil para determinar la viabilidad del proyecto. El segundo prototipo, **Cube-2.0**, que es el que se presenta en este proyecto, está realizado en piezas de plástico transparente de 2mm de grosor, que se corta y taladra muy facilmente. Está diseñado para ser ampliable a cualquier número de servos puesto que está compuesto por la unión de unos módulos básicos que son iguales.

La estructura del gusano está formada por tres partes diferentes: **cabeza**, **cuerpo** y **cola**. La cabeza está pensada para llevar la electrónica y por ello es más grande que la cola. El **cuerpo** está constituido por módulos iguales que se interconectan mediante piezas de tipo A. Los módulos están construidos a partir de **servos con doble eje**, que se construyen uniendo un servo del tipo **Futaba 3003** con una pieza con falso eje. Esta pieza se puede construir a partir de las propias piezas que componen el servo y se pueden conseguir en cualquier tienda especializada.

Con todas estas piezas se ha mostrado cómo hay que construir el gusano. Especial cuidado hay que tener en la **calibración**, de manera que cuando el software le indique al gusano que se sitúe en la posición de reposo, el gusano se encuentre en posición horizontal.

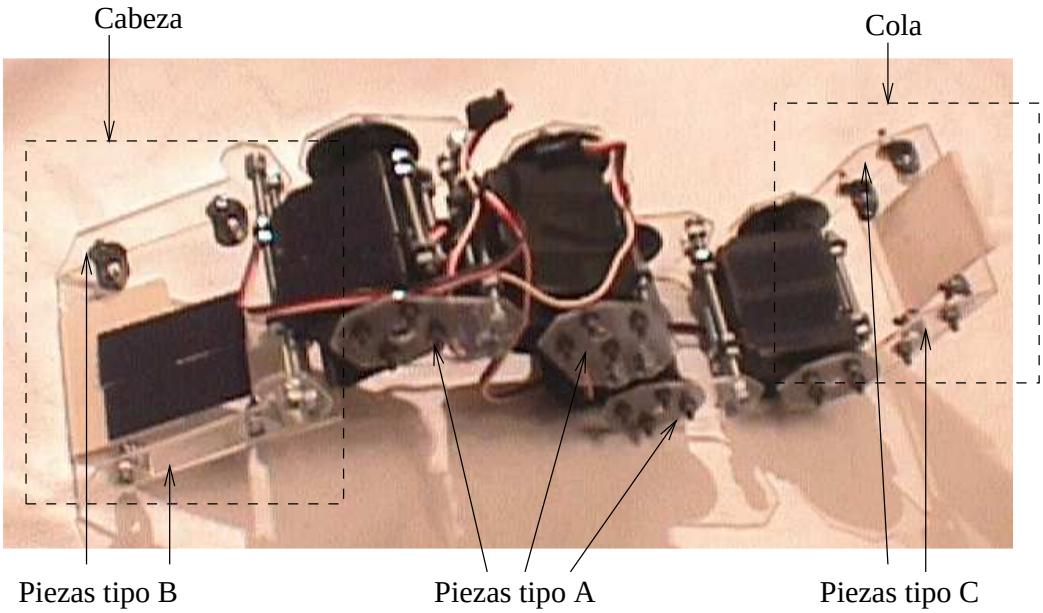


Figura 8.20: Estructura final de Cube-2.0, sin la electrónica

Capítulo 9

Electrónica

9.1. Introducción

La parte mecánica ya está resuelta en el capítulo 8. Ahora es necesario añadir una electrónica que pueda mover los servos y que se pueda comunicar con el PC para recibir órdenes. En este capítulo se presenta una electrónica que cubre esos dos aspectos.

El gusano tienen una serie de particularidades que hacen que la electrónica tenga que cumplir una serie de requisitos. Con ellos se plantearán las alternativas de diseño, justificándose la solución adoptada. Esta solución está basada en redes de microcontroladores. La electrónica, además, se tiene que adaptar correctamente a la estructura mecánica del gusano.

En [19] se encuentran los esquemas de las tarjetas empleadas en *Cube*, así como los manuales de usuario. En [16] se emplean estas mismas tarjetas para controlar un perro robot. Para la programación del 68hc11 se puede consultar [21] y finalmente se puede encontrar información muy útil en [28].

9.2. Requisitos

Hay una serie de requisitos que debe cumplir la electrónica integrada en *Cube*:

1. Debe poder controlar **gusanos de cualquier longitud**. Cuantas menos limitaciones mejor.
2. Debe permitir que el **gusano sea autónomo**, aunque en este proyecto se utilice el PC para implementar todos los cálculos.
3. Debe poder **conectarse al PC**, para utilizar software de calibrado, carga de programas, prueba de secuencias, etc, aunque también pueda ser autónomo.
4. Debe ser **fácil de conseguir** en el mercado.
5. Debe ser muy **flexible y modular**, de modo que permita incorporar mejoras futuras.

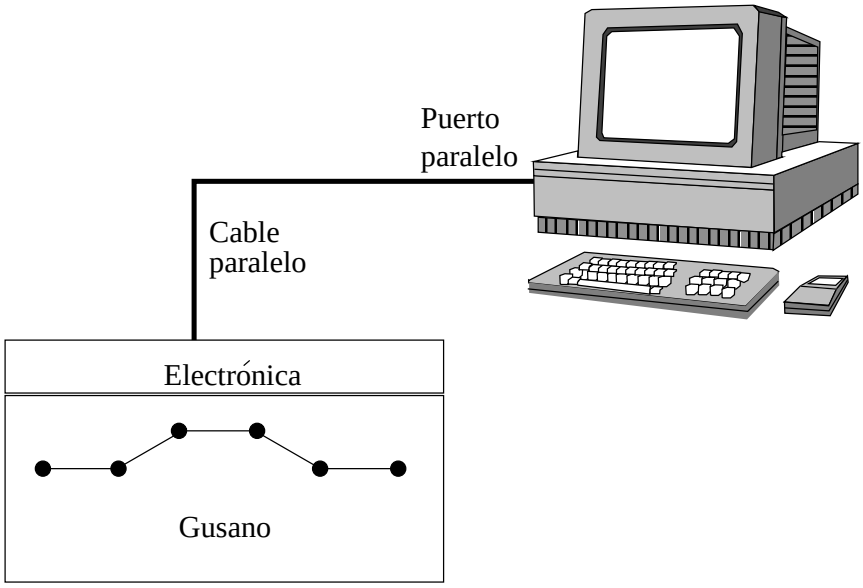


Figura 9.1: Alternativa 1: PC + electrónica por el puerto paralelo

9.3. Alternativas de diseño

Son muchas las alternativas que se pueden tomar a la hora de implementar un robot. La empleada por el autor está basada en la experiencia adquirida durante estos últimos 5 años y en toda la electrónica y software que ya han sido diseñados y probados con éxito en otros robots[1]. No obstante es bueno presentar otras alternativas.

1. Para controlar un robot, más o menos complejo, se puede pensar en utilizar el **PC conectado por el puerto paralelo**. En este caso la electrónica del robot sólo está constituida por la etapa de potencia de los servos y la necesaria para adaptar niveles. Esta solución es sencilla, pero no cumple con todos los requisistos del apartado 9.2:

   a) Los pines de salida del puerto paralelo son pocos. Por cada pin se puede controlar un servo mediante PWM, por lo que en total podríamos controlar unos 15 servos. El gusano en este caso tiene una limitación electrónica en la longitud que puede llegar a tener.

   b) El sistema nunca podrá ser autónomo. Siempre será necesario disponer de un PC.

   c) Hay que emplear un cable de puerto paralelo estándar, que tiene muchos hilos (25), que es pesado, caro y no puede ser muy largo (unos 4m como máximo)

2. Otra solución es utilizar un **microcontrolador que se comunique con el PC por medio del puerto serie**. Ésta opción ofrece más ventajas que la del puerto paralelo:

[1]Toda esta tecnología ha sido desarrollada por los miembros de Microbótica, S.L, en su etapa de estudiantes, de los que el autor forma parte.

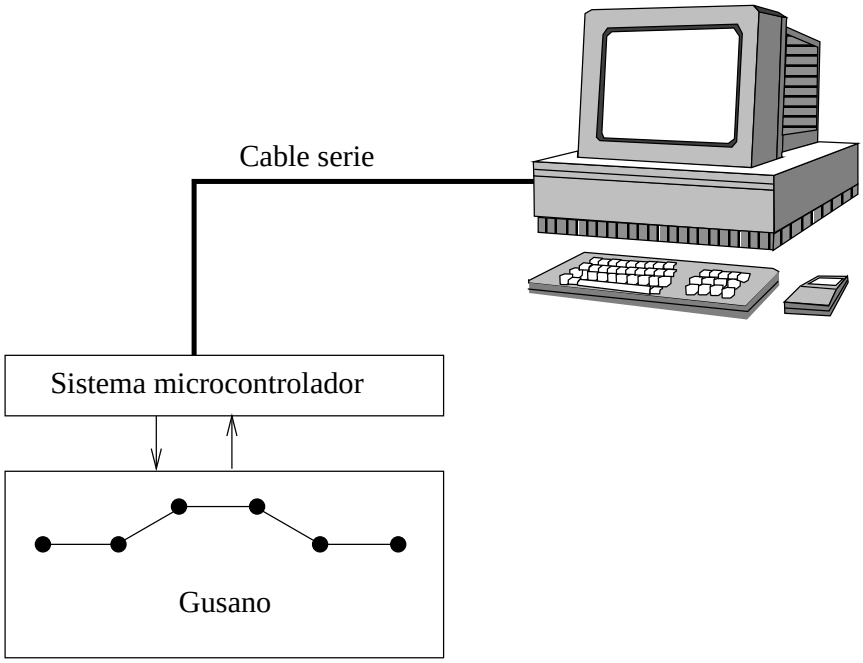


Figura 9.2: Alternativa 2: PC + microcontrolador por puerto serie

a) Sólo son necesarios tres hilos para realizar la comunicación (Tx, Rx y GND).

b) El PC envía la información de control al microcontrolador y es éste el que genera las señales de PWM. Los microcontroladores disponen de recursos para poder generar estas señales de forma fácil y precisa. **El PC se hace así independiente de los servos**. Si estos cambian por otros, sólo hay que modificar el programa del microcontrolador, pero el PC sigue "viendo" el mismo sistema. No es necesario modificar el software.

c) Es posible hacer que el propio micro gobierne el gusano, de forma que sea autónomo.

d) La longitud del cable de conexión entre el PC y el microcontrolador puede ser mayor (la norma RS232 especifica un máximo de 12m)

Sin embargo, el sistema todavía está limitado a los servos que se puedan controlar con el microcontrolador y por tanto no escala bien. Esta es la solución que se empleó en **Cube 1.0**, que si bien funciona muy bien, no permite insertar muchas más articulaciones. Para poder ampliarlo había que rehacer toda la electrónica.

3. La alternativa empleada en **Cube 2.0** se basa en una **red de microcontroladores**, que ha sido desarrollada en el PFC de D. Andrés Prieto-Moreno Torres[16] e implementada con éxito en la construcción de Puchobot, un perro robot. La ventaja de esta alternativa es la **escalabilidad**. Si se añaden más articulaciones al gusano sólo hay que añadir más nodos a la red de control. El número todal de servos que se pueden tener vendrá determinado por la velocidad de la red y el tráfico que en ella se genere. Utilizando direcciones de nodos de

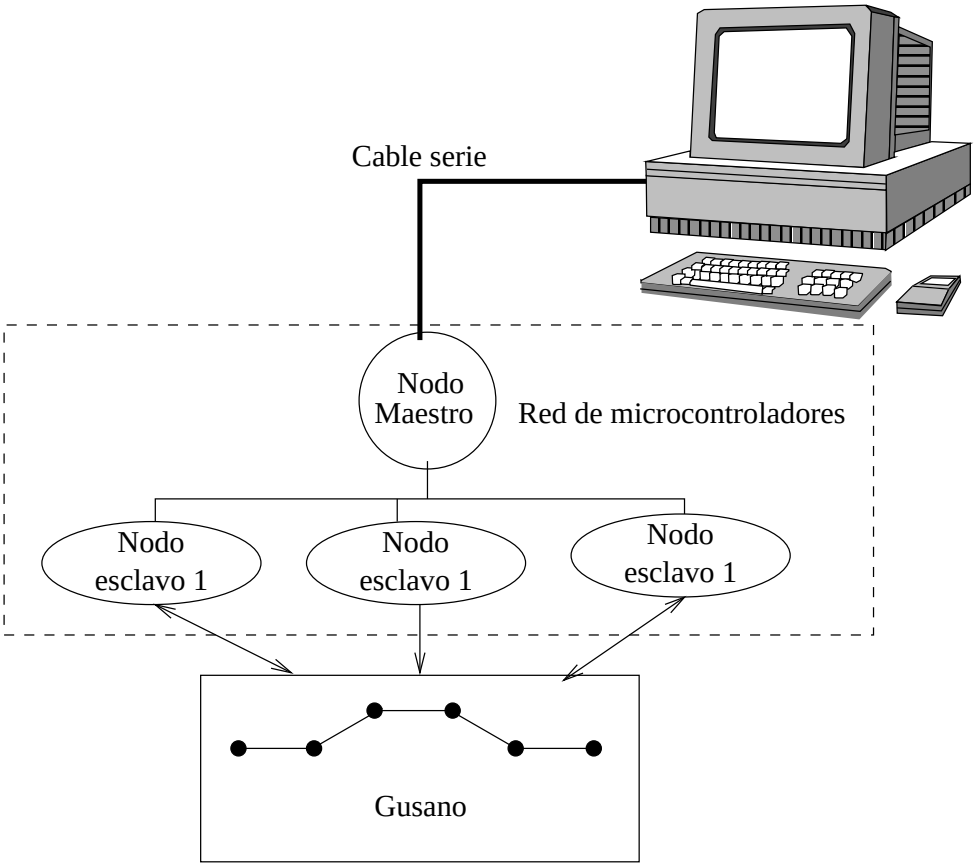


Figura 9.3: Alternativa 3: Red de microntroladores, conectada al PC por puerto serie

la red de 8 bits, se pueden conectar hasta 256 nodos y cada nodo es capaz de controlar 4 servos, por lo que en teoría se podrían conectar hasta ¡¡256*4=1024 servos!!.

9.4. Arquitectura hardware

La red de microcontroladores es del tipo **maestro-esclavo**. Existe un nodo más importante (**nodo maestro**) que es el que envía las distintas órdenes a los demás nodos (**nodos esclavos**). La conexión entre los nodos es serie síncrona.

El nodo maestro, además, es el que se conecta al PC vía interfaz serie asíncrono, según la norma RS-232. Si se quiere trabajar en modo autónomo, el programa se ejecuta en el maestro, actuando sobre los diferentes servos. Si lo que se quiere es un modo no autónomo, el maestro simplemente hace de puente entre el PC y el resto de nodos, pasando la información de un lugar a otro.

Para el nodo maestro se utiliza la **tarjeta CT6811**, basada en el microcontrolador 68hc11 de Motorola. Los nodos esclavos se implementan con la **tarjeta BT6811**, que permite controlar hasta 4 servos del tipo Futaba 3003. Se puede encontrar abundante información sobre estas

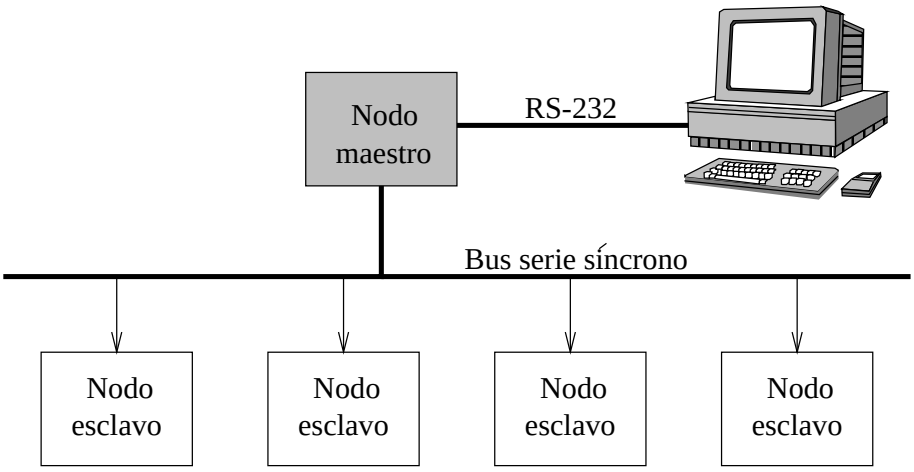


Figura 9.4: Arquitectura hardware

tarjetas en [16] y [19], así como en el CD que acompaña al proyecto.

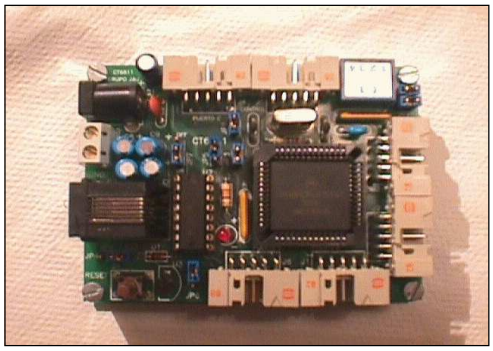
Puesto que en la implementación del gusano se han utilizado cuatro articulaciones, sólo hace falta una BT6811 para los servos y una CT6811 para el nodo maestro.

9.5. Tarjetas CT6811 y BT6811

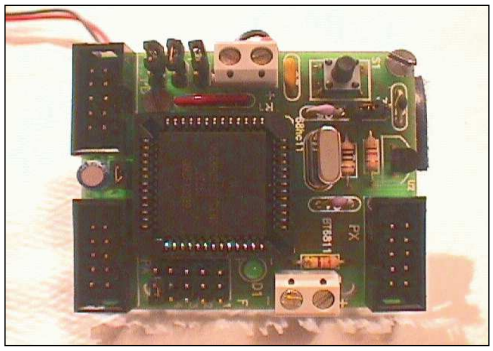
9.5.1. Características

Ambas tarjetas utilizan el microcontrolador 68hc11E2 de motorola, que tiene las siguientes características:

- **CPU** de 8bits. Capaz de direccionar hasta 64Kbytes
- **Reloj**: 8Mhz
- **Velocidad de bus**: 2Mhz
- **Memoria RAM**: 256bytes
- **Memoria EEPROM**: 2Kbytes
- **Temporizador de 16bits**
- **5 Comparadores con salida hardware** (Con los que es muy sencillo generar una señal PWM para el control de los servos)
- **3 capturadores de entrada**
- Un **puerto serie asíncrono** (SCI)



CT6811



BT6811

Figura 9.5: Las tarjetas CT6811 y BT6811

- Un **puerto serie síncrono** (SPI)
- **8 canales A/D**
- **5 puertos de entrada/salida**: A, B, C, D y E.

La tarjeta **CT6811** viene ya preparada para ser conectada directamente a un PC, incorporando los circuitos de adaptación entre niveles TTL/RS232. Puede funcionar como una tarjeta entrenadora, en la que se puede cargar un programa en la memoria RAM o bien como un sistema autónomo, ejecutando el programa que tiene grabado en la memoria EEPROM.

La tarjeta **BT6811** está pensada fundamentalmente para funcionar como un sistema autónomo. Tiene dos entradas para la alimentación, una para los servos y otra TTL para la electrónica.

Ambas tarjetas se alimentan con una tensión continua entre 4.5 y 6 voltios. En la figura 9.5 se muestra una foto. El tamaño de la BT6811 es aproximadamente la mitad que el de la CT6811.

9.5.2. Descripción

En la figura 9.6 se muestran todos los elementos del interfaz de la tarjeta CT6811. Dispone de 6 conectores acodados, de 10 pines, desde los que se puede acceder a los puertos de E/S del 68hc11. Especial interés tiene el puerto D que es donde se encuentran los pines del SPI, que permiten conectar la tarjeta a la red de microcontroladores. La alimentación se puede realizar bien por clemas o bien por un conector de tipo jack. Para la conexión al PC se utiliza un conector hembra teléfonico, de cuatro vías. El cable es similar al de los teléfonos. Tiene la ventaja de que es muy ligero y muy fácil de construir. Para la conexión al puerto serie del PC se emplea un adaptador DB9 a teléfono.

Existen unos microinterruptores para configurar el modo de arranque del microcontrolador. Para el gusano se empleará el modo *bootstrap*, situando todos los switches hacia abajo. La configuración de los jumpers es la siguientes:

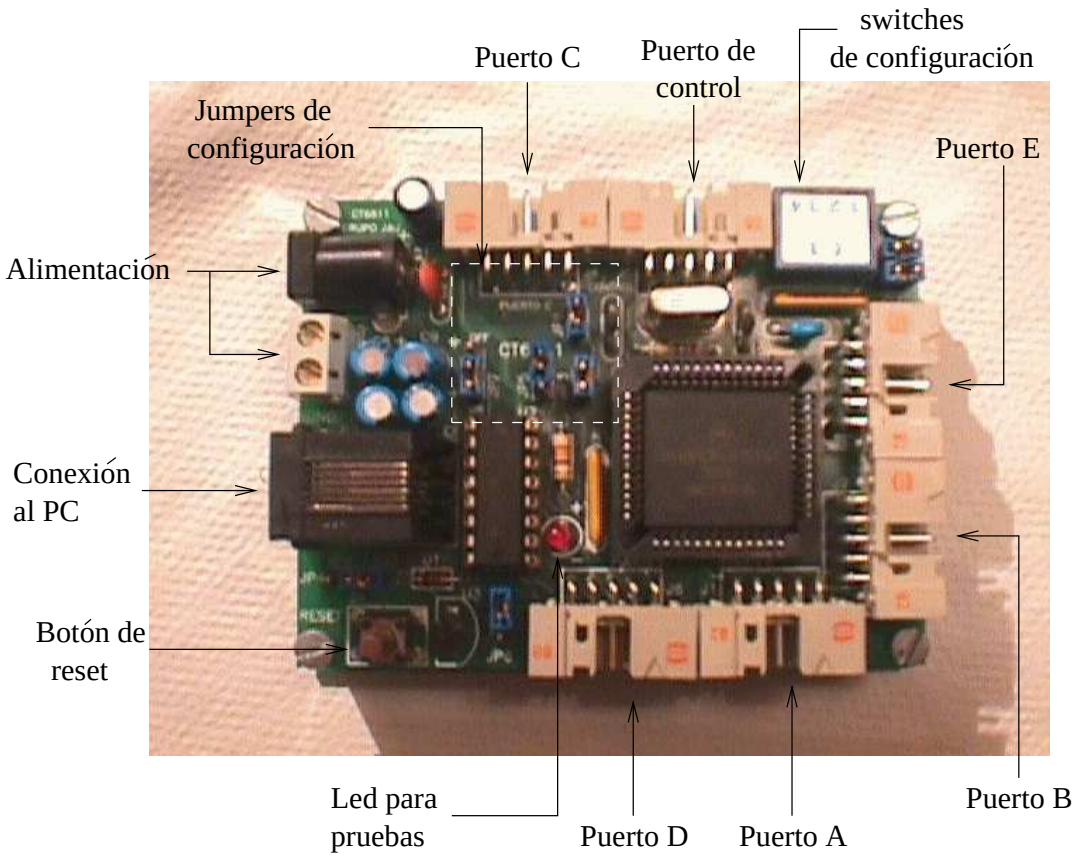


Figura 9.6: Elementos de la tarjeta CT6811

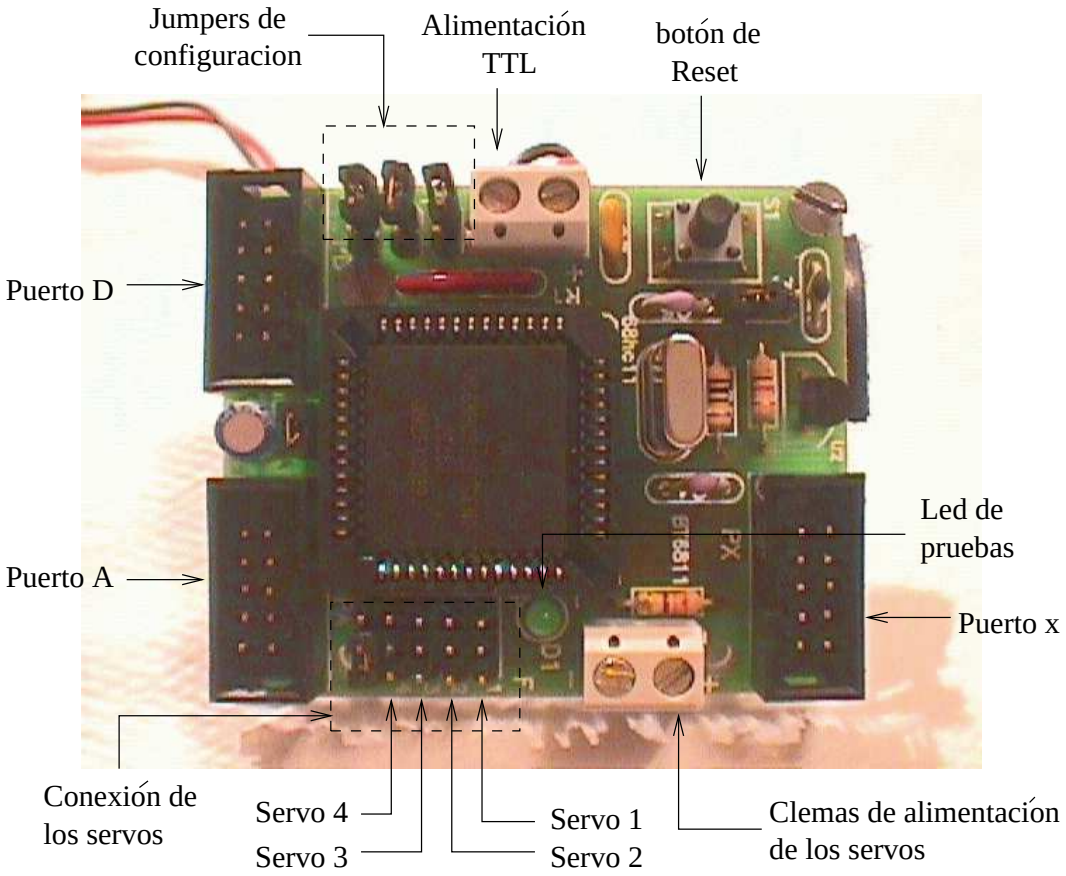


Figura 9.7: Elementos de la tarjeta BT6811

- Los Jumpers **JP3**, **JP8**, **JP2**, **JP1**, **JP7** deben estar puestos
- Los jumpers **JP5** y **JP6** quitados
- El jumper **JP7** en posición **ON**
- El jumper **JP4** en posición **RST**

El botón de *reset* permite reinicializar la placa. Esta operación también la puede realizar el software que corre en el PC, activando la señal DTR de la norma RS232. La CT6811 dispone tambień de un led conectado al bit 6 del puerto A para realizar pruebas y depurar el software.

En la figura 9.7 están los elementos de interés de la tarjeta BT6811. Dispone también de un botón de *reset* para reinicialización del software, pero a diferencia de la CT6811, este *reset* debe ser manual y no se puede hacer por software. Hay dos clemas para la alimentación: una para la de la circuitería y otra para la de los servos. Es posible alimentar tanto los servos como la electrónica a partir de las mismas clemas, sin embargo es recomendable separar ambas alimentaciones para evitar ruido en la del micro.

En el puerto D se encuentran los pines del *SPI* y es donde se conecta la CT6811 maestra y el resto de BT6811 esclavas.

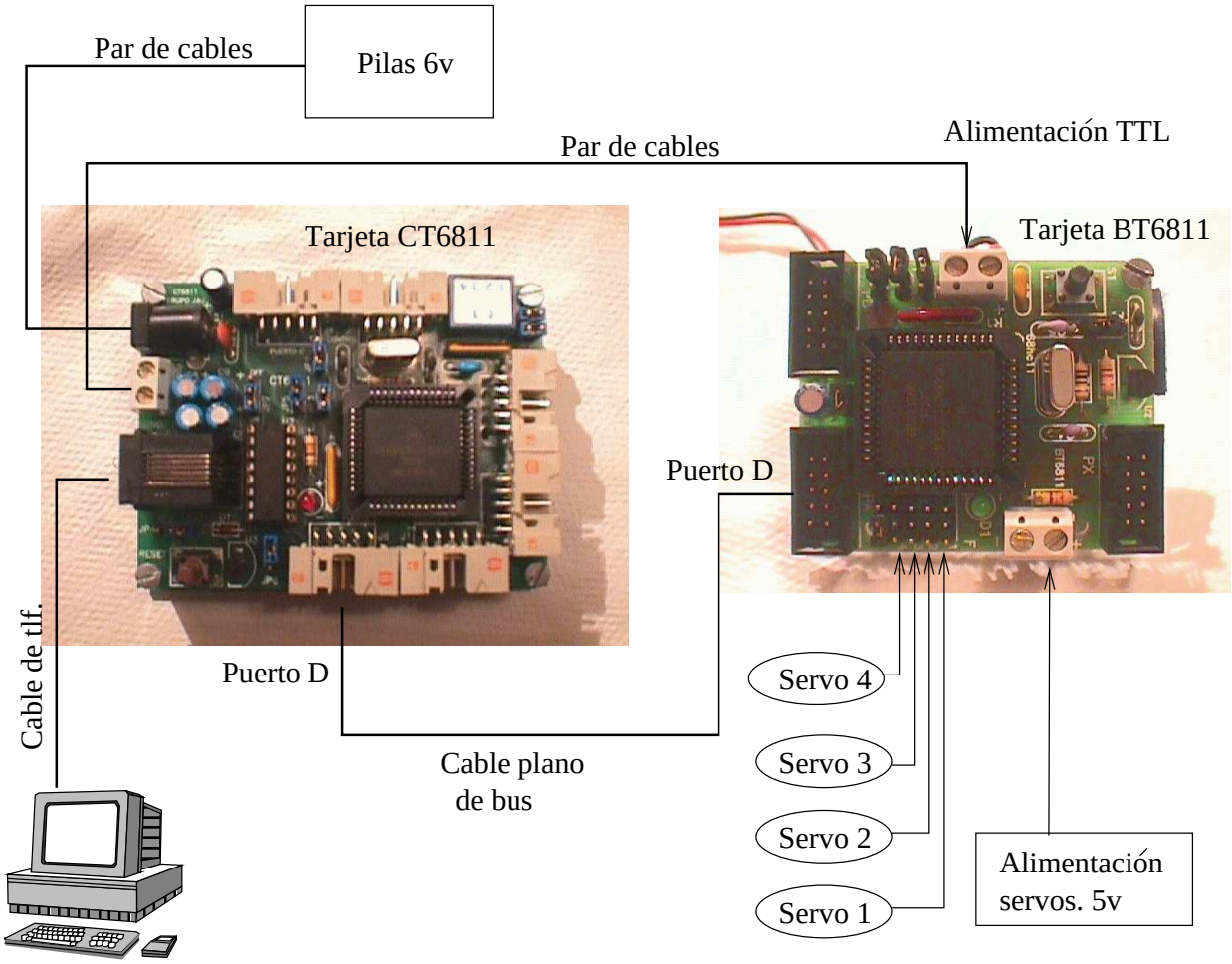


Figura 9.8: Interconexión de las tarjetas CT6811 y BT6811 en Cube

La configuración de los jumpers para trabajar con el gusano debe ser:

- Jumper **JP6** activado (colocado)
- El resto deben estar desactivados

9.5.3. Interconexión de las tarjetas

En la figura 9.8 se han representado las conexiones entre ambas tarjetas, así como las conexiones con el resto de partes de Cube. Los servos se conectan a los conectores **F1**-**F4** de la BT6811. La alimentación TTL se toma de la propia CT6811, mediante un par de cables, que unen ambas clemas. La CT6811 la toma de las pilas, a través de un cable que termina en un conector de tipo jack macho. La alimentación de los servos se obtiene de una fuente de alimentación externa.

Mediante un cable plano de bus, de 10 hilos, se conectan los puertos D de ambas tarjetas, quedando conectadas a la misma red. A través de un cable de teléfono se conecta la CT6811 al

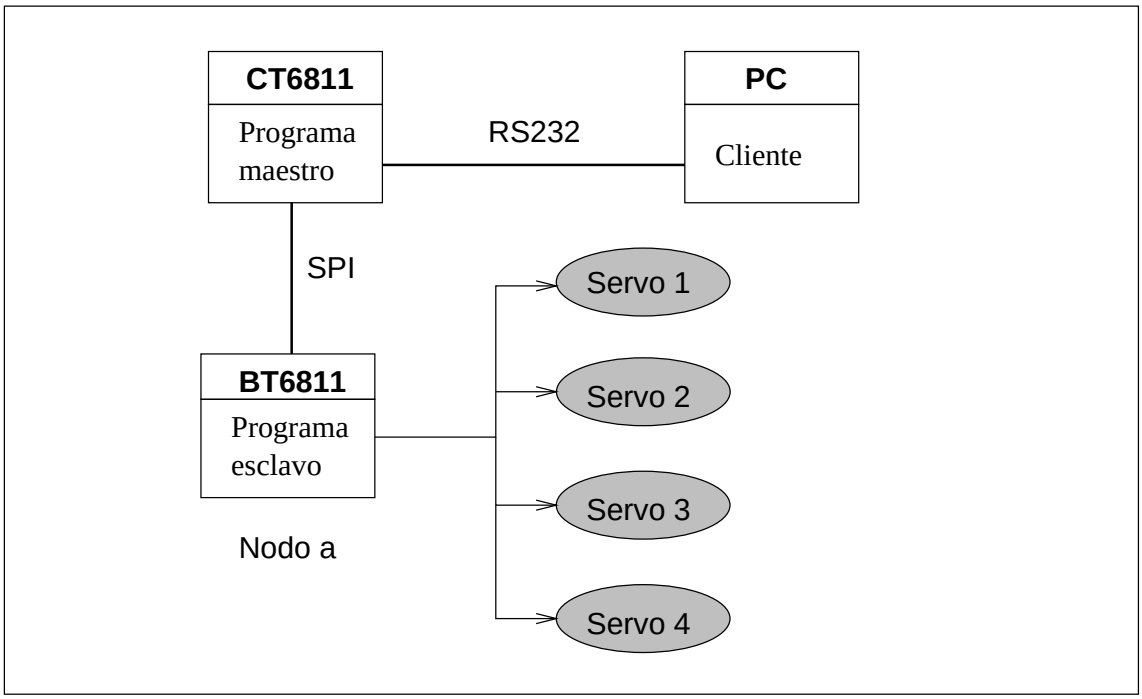


Figura 9.9: Arquitectura software de la spinet

PC.

9.6. Red de microcontroladores: spinet

9.6.1. Arquitectura software

Puesto que los microcontroladores de la red se encuentran conectados por el SPI (Serial Peripheral Interface), que es un bus serie síncrono, la red se ha denominado **spinet**.

En la figura 9.9 se muestra la arquitectura software empleada con la *spinet* de Cube. Esta red está constituida por un nodo maestro y uno esclavo. En cada nodo se está ejecutando un software diferente:

1. **Nodo esclavo**: Programa *futA.asm*. Se encarga de la recepción de paquetes por la red y de ejecutar los comandos que recibe. Mediante interrupciones se generan las señales PWM de los servos activos que está controlando. Este programa se encuentra grabado en la memoria EEPROM del 68hc11.

2. **Nodo maestro**: Programa *maestro.asm*. Es un programa que hace de puente entre los nodos de la red y el cliente en el PC, pasando la información recibida por la norma RS232 y enviándosela a los nodos de la red, a través de bus SPI. Este programa lo carga el cliente del PC en la RAM del 68hc11.

Los listados de ambos programas se incluyen en el apéndice C.

En el PC se ejecuta un programa cliente que carga el programa *maestro* en la ram del nodo maestro y a través del cual accede a todos los servicios que le ofrece la *spinet*.

9.6.2. Servicios ofrecidos por la spinet

Los servicios que ofrece la *spinet* son:

1. **Test**. Cambiar el estado del led del nodo indicado. Es muy útil para comprobar si la red está funcionando y para depurar el software.
2. **Activar**. Activar o desactivar los servos indicados de cualquier nodo.
3. **Control**. Enviar un valor a los puertos C de los nodos esclavos. Esto permite que cada nodo esclavo se puede conectar a un periférico al que puede mandar información de control.
4. **Posicionamiento**. Posicionar cualquier servo de un nodo.

Estos servicios se implementan mediante tramas que se envían desde el PC al nodo maestro y éste a su vez las reenvía al resto de los nodos.

Para acceder a estos servicios hay que especificar el nodo esclavo y el servo dentro del nodo sobre el que se quiere actuar, para el caso del servicio 4. Cada nodo esclavo tiene una dirección en la *spinet*. En el caso de Cube **sólo existe un nodo que tiene la dirección 61h**[2]. Los servos están numerados desde el 1 hasta el 4.

9.7. Adaptación a la estructura mecánica

La electrónica que se tiene es:

- Tarjeta CT6811
- Tarjeta BT6811
- Cables de interconexión

y hay que acoplarla a Cube. En la figura 9.10 aparece la estructura mecánica junto con la electrónica y las pilas, sin colocar en el gusano. Un problema importante es que el gusano está constituido por partes móviles por lo que hay muy poco espacio. La CT6811 se ha situado en la cabeza del gusano, encima de su base. La sujección se puede realizar con gomas elásticas o mejor haciendo una base de plástico de las mismas dimensiones que la CT6811, atornillarla a ella y la base unirla a la cabeza mediante velcro. Este enganche es muy robusto y permite quitar y poner la tarjeta cuantas veces se quiera.

[2]El valor hexadecimal 61h se corresponde con el carácter ascii 'a'. Con esto se indica que es el nodo 'a'.

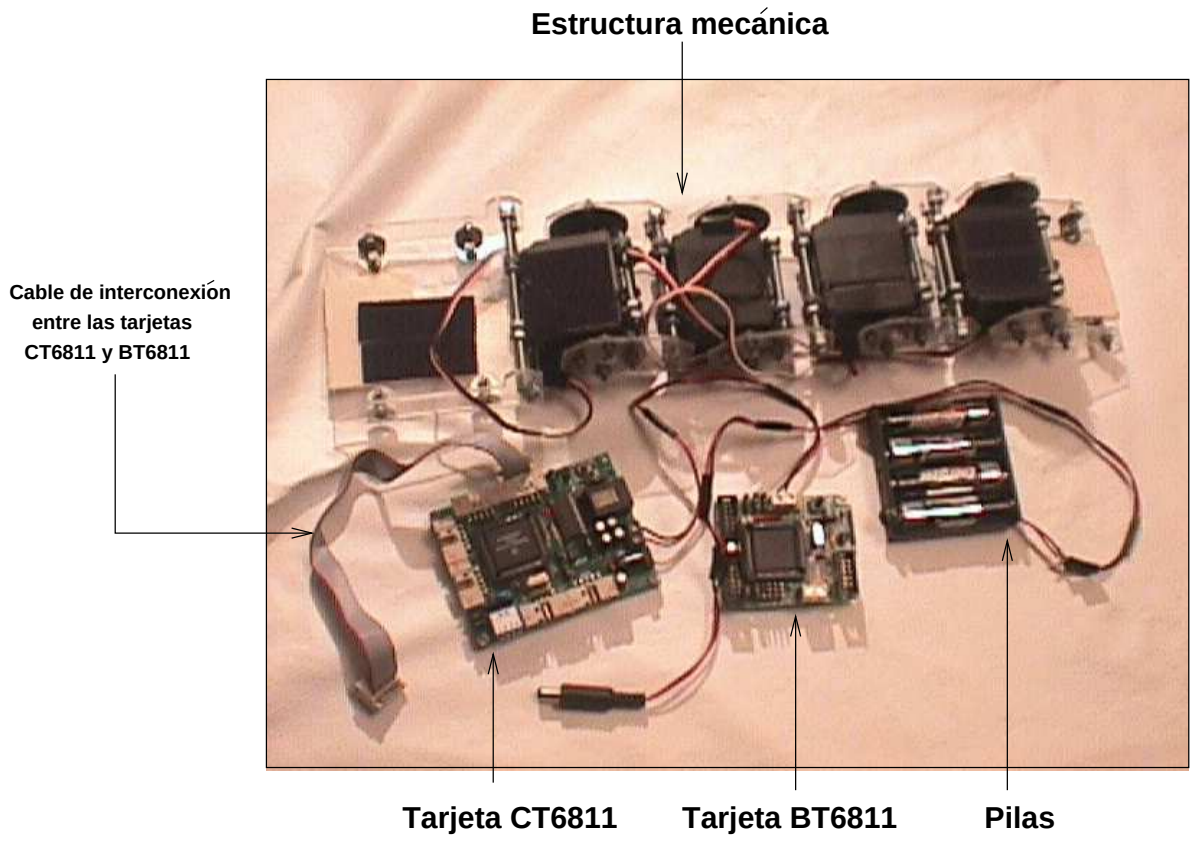


Figura 9.10: La estructura mecánica y la electrónica de cube.

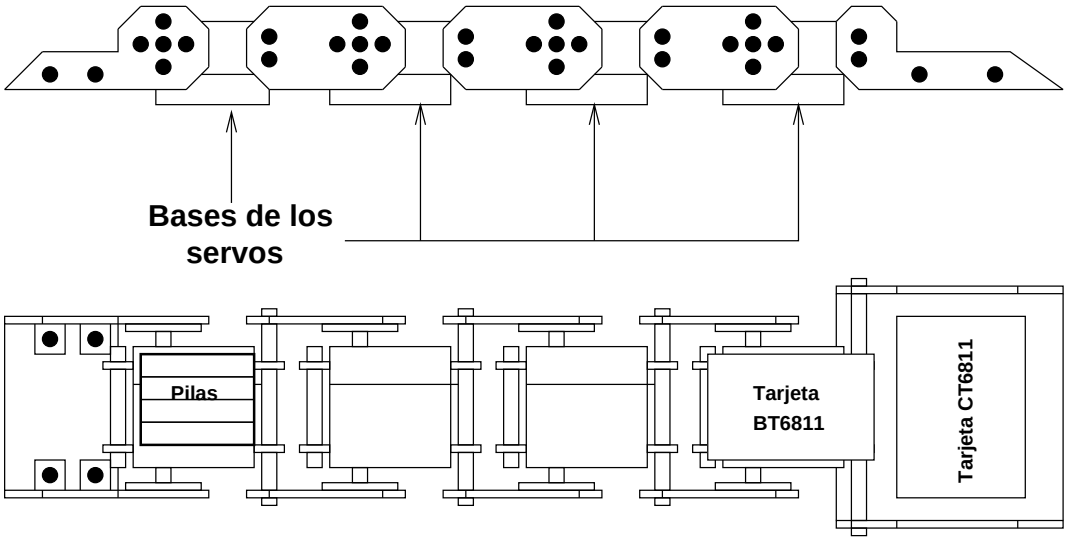


Figura 9.11: Situación de la electrónica y las pilas en Cube

Para la BT6811 se hace algo parecido. Se construye la base de plástico y se atornilla a ella. Esta base se adhiere al futaba 4 mediante velcro. El futaba 4 es el que se encuentra más cerca de la cabeza, de esta manera los cables de interconexión entre las tarjetas pueden ser muy cortos.

Para la alimentación de la electrónica se emplean cuatro pilas de tipo AA, de 1.5v, consiguiéndose en total 6v. El portapilas empleado tiene las medidas justas para situarse sobre cualquiera de los cuatro servos. Se ha empleado el número 1, el más cercano a la cola, para contrarrestar el peso de la CT6811 y la BT6811 que están más cerca de la cabeza. Del portapilas salen los cables de alimentación que terminan en un conector jack macho. Este conector es el que se conecta a la tarjeta CT6811.

La alimentación de los servos se obtiene de una fuente de alimentación externa, o una batería. Tiene que ser externa porque no hay sitio para colocarla dentro de la estructura del gusano.

Los cables de los futabas se conectan a la BT6811 y para evitar enredos se llevan por la parte inferior de cube, introduciéndose entre las bases de los servos. En la figura 9.11 se muestra un esquema de la situación de toda la electrónica y las pilas. Y en la figura 9.12 se presenta el gusano final en el que se identifican los elementos y se dibujan las conexiones principales.

9.8. Alimentación

La alimentación completa del gusano es la siguiente:

- **Electrónica**
  - *Voltaje*: 6v (Pilas)
  - *Consumo*: 40mA

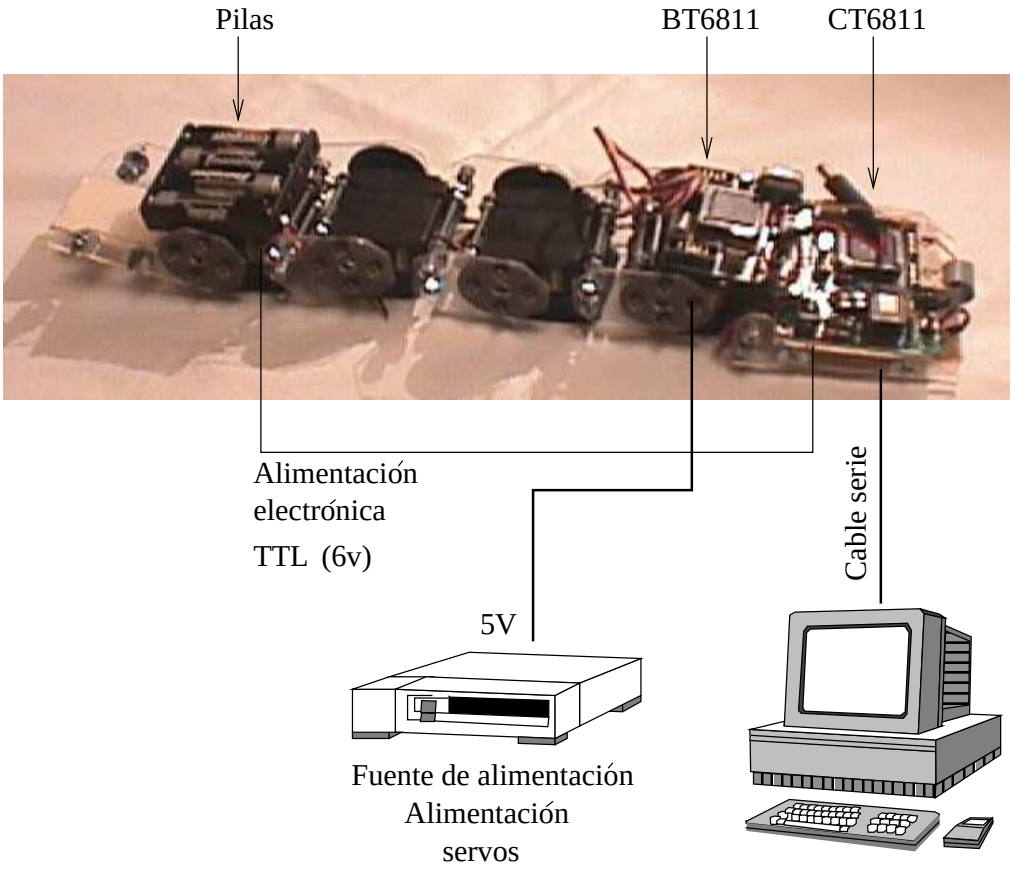


Figura 9.12: Gusano final, con las principales interconexiones

- **Servos**:
  - *Voltaje*: 5v (Fuente de alimentacion externa)
  - *Concumo*: unos 200mA por servo. En total 800mA.

9.9. Resumen

Para cumplir con los requisitos de que el gusano sea ampliable, autónomo, conectable al PC, electrónica fácil de conseguir y económica se ha elegido la alternativa de emplear una **red de microcontroladores**, muy probada en otros robots. La electrónica está constituida por un nodo maestro, la **tarjeta CT6811** y un nodo esclavo, **tarjeta BT6811**, que puede controlar los cuatro servos de las cuatro articulaciones. Al tratarse de una red es muy fácilmente ampliable, se pueden implementar gusanos de mayor número de articulaciones.

Esta red, denominada, ***spinet***, ofrece una serie de servicios a un cliente que se ejecute en el PC de manera que este software hace abstracción de los detalles del moviento de los servos y se limita a establecer la posición en la que deben situarse.

Esta electrónica se tiene que adaptar a la estructura del gusano. Para ello se han se emplean unas **bases de plástico** que se enganchan al gusano utilizando **velcro**. De esta forma es muy fácil quitar y poner las placas. La alimentación es doble: son necesarias **pilas** para la electrónica, que van en el propio gusano, enganchadas también con velcro, y una **fuente de alimentación externa** para los servos, puesto que las baterías no caben físicamente en el gusano.

Capítulo 10

Software: gusano físico

10.1. Introducción

El software que se describe en este capítulo es el que se utiliza en el PC para acceder al gusano físico y mover sus diferentes servos. Nos permite realizar calibraciones mecánicas y lo más importante, reproducir secuencias creadas, bien por el software del gusano virtual o bien secuencias manuales.

Información sobre la programación en C se puede encontrar en [30] y para el desarrollo de interfaces gráficos con GTK+ es muy útil [29].

10.2. Descripción

El programa creado se denomina `cube-físico`, en oposición al software `cube-virtual`, que controla un gusano virtualo, presentado en el capítulo 7. Está diseñado para controlar un gusano real, al que denominaremos **gusano físico**. En la figura 10.1 se muestra el aspecto del programa. Permite realizar las siguientes operaciones:

1. **Mover los servos independientemente**, mediante barras de desplazamiento.
2. **Establecer el estado de los servos** especificando su vector de estado físico.
3. **Generar secuencias de movimiento manuales**, que se puedan reproducir, grabar y cargar.
4. **Lectura de secuencias de movimientos** de un fichero, bien generadas por el propio programa o por la aplicación **cube-virtual.**
5. **Editar secuencias** ya creadas, insertando o eliminando estados.

En realidad este programa sirve para manejar cuatro servos aislados, independientemente de que se encuentren en un gusano. Es posible generar secuencias para cualquier otro robot, que esté constituido por 4 servos. Si además se cargan las secuencias generadas por el gusano virtual, se consigue el movimiento en el gusano físico, y se pueden analizar los diferentes estados intermedios.

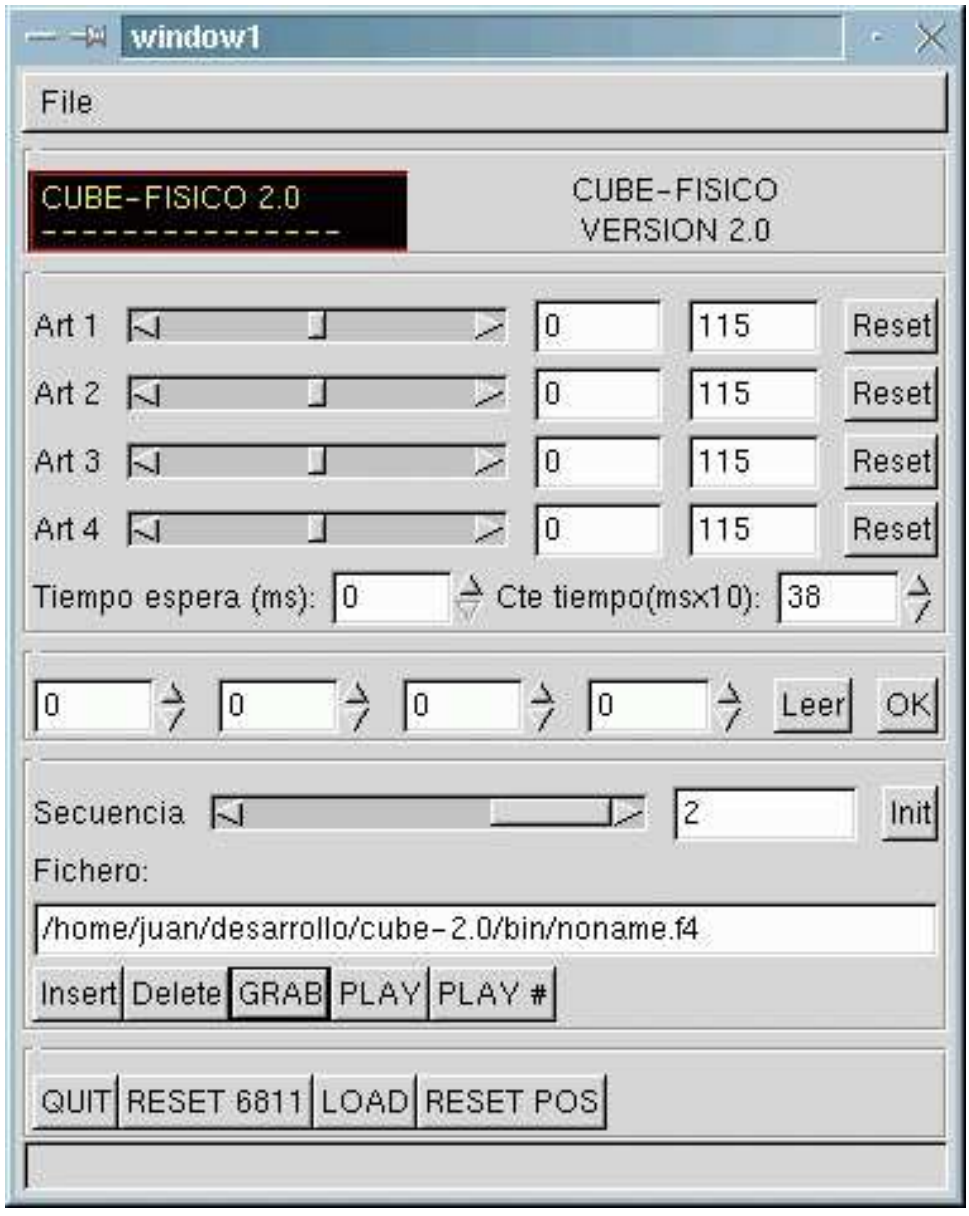


Figura 10.1: Aspecto del programa cube-fisico

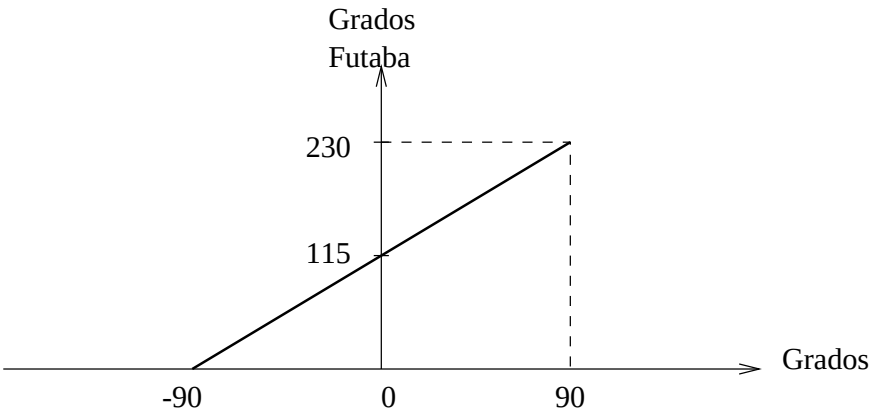


Figura 10.2: Gráfica de conversión entre grados y grados futaba

10.3. Servos y secuencias

10.3.1. Control de los servos

El control local de cada servo se realiza en los nodos esclavos de la *spinet*. La información de control la envía el PC a la tarjeta maestra y ésta lo reenvía a los notos esclavos. Los servicios a los que tenemos acceso son los que se especificaron en el apartado 9.6.2.

A estos servicios se accede mediante las funciones del módulo *spinet*, descrito más adelante. Las tramas se generan y se envían al nodo maestro, por el puerto serie.

El servicio de posicionamiento ofrecido por la red utiliza **grados futaba**. El rango total de giro del servo, 180 grados, se divide en 230 partes. Cada parte es un grado futaba.

En los programas **cube-físico** y **cube-virtual** se trabaja con grados sexagesimales[1] que son más intuitivos. El módulo *spinet* tiene una función para realizar la conversión. En la figura 10.2 se muestra la gráfica de conversión. La relación es:

$$gf = \frac{115}{90} grados + 115 \tag{10.1}$$

en donde gf son los **grados futaba**.

10.3.2. Secuencias

10.3.2.1. Parámetros temporales

El estado de los servos queda determinado en un instante de tiempo por el **vector de estado físico**, que tiene cuatro componentes, una para cada articulación, expresadas en grados:

$$E(t) = (\varphi_1, \varphi_2, \varphi_3, \varphi_4)$$

[1] A partir de ahora, cada vez que se escriba grados se entenderá que se trate de grados sexagesimales, salvo que se indique expresamente lo contrario.

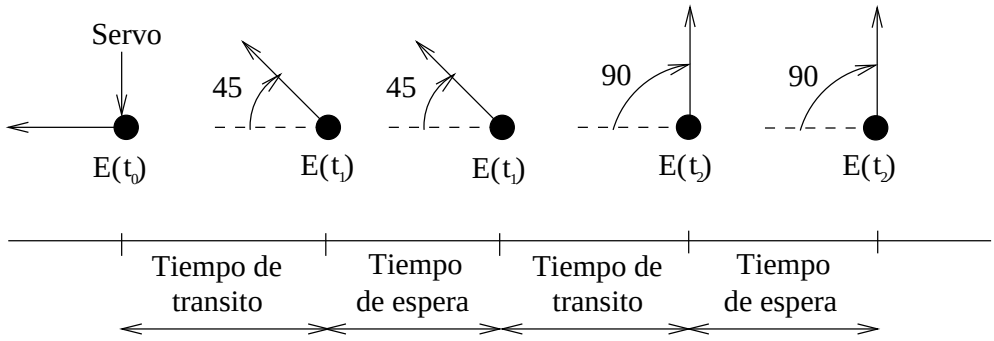


Figura 10.3: Ejemplo de tiempo de tránsito y de tiempo de espera en la evolución de un servo

Una secuencia es una serie de vectores de estado, cada uno de los cuales define el estado en un instante t_i:

$$Sec = \{E(t_1), E(t2), ..., E(t_i), E(t_j), ..., E(t_M)\}$$

Los intervalos de tiempo $t_j - t_i$ están constituidos por dos componentes:

- t_t**: Tiempo de tránsito**. Tiempo que tarda el conjunto de servos en alcanzar el estado definido por $E(t_j)$.
- t_e**: Tiempo de espera**. Tiempo que debe permanecer el sistema en el estado $E(t_j)$, antes de pasar al estado $E(t_{j+1})$.

En la figura 10.3 se ha dibujado un ejemplo de una secuencia para un servo en la que se pueden ver las componentes t_t y t_e. La secuencia está compuesta por tres estados diferentes:

$$sec = \{E(t_0), E(t_1), E(t_2)\}$$

Los estados son $E(t_0) = 0$, $E(t_1) = 45$, $E(t_2) = 90$. El tiempo de espera se puede hacer diferente para cada estado, de manera que por ejemplo permanezca 2 segundos en el estado $E(t_1)$ y 3 en el $E(t_2)$.

El **tiempo de espera** es un parámetro determinado por el usuario. Si $t_e = 0$, se obtienen *movimientos continuos*, es decir, que no hay esperas en cada estado, sino que se va evolucionando continuamente de un estado hasta otro.

El tiempo de tránsito es una característica del futaba, que depende de su velocidad y del ángulo girado:

$$t_t = \tau.\triangle\varphi \tag{10.2}$$

siendo:

- τ**: Constante de tiempo del futaba**. Es el tiempo que tarda el futaba en recorrer un grado. Se expresa en mseg.
- $\triangle\varphi$**: Ángulo que debe recorrer el servo**. Se expresa en grados.

Por las características dadas por el fabricante, se tiene que para un servo futaba, $\tau = 3{,}8$mseg.

La **constante de tiempo** τ es un parámetro estadístico que depende de la carga del servo, de la alimentación, etc. Por ello hay que estimarlo, llamando al valor estimado τ_e. El fabricante proporciona un valor nominal cuando el servo no está cargado y se alimenta con la tensión nominal. Suponiendo que $t_e = 0$, es decir, que se quiere una trayectoria continua, puede ocurrir lo siguiente:

1. $\tau_e >> \tau$. Se está sobreestimando la constante de tiempo. El tiempo de tránsito estimado es mayor de lo que debería ser. El servo alcanza la posición final antes de lo que hemos previsto, por lo que se para en ese estado antes de pasar al siguiente. El resultado es que la trayectoria no es continua sino que se realiza a "saltos".

2. $\tau_e << \tau$**.** Se está subestimando la constante de tiempo. El tiempo de tránsito estimado es menor de lo que debería ser. Antes de alcanzar la posición final se le está indicando al servo que pase a la siguiente. La articulación no se llega a posicionar en los estados que se le indican. El movimiento es continuo pero diferente del que se le ha especificado, ya que hay posiciones que no se alcanzan nunca.

Se llega a la siguiente conclusión:

> ***Hay que buscar un compromiso entre la exactitud y la continuidad del movimiento***.

Cuanto mayor sea la exactitud, la continuidad será peor y cuanto mayor continuidad en el movimimento menor será la exactitud.

El parámetro τ_e es fijable por el usuario, de manera que busque el compromiso entre *exactitud* y *continuidad* que más se ajuste a su aplicación. Esto también es muy útil para la depuración de secuencias, ya que fijando un τ_e grande, se puede ir viendo la trayectoria paso a paso y analizar los problemas.

10.3.2.2. Estimación del tiempo de tránsito

Para un sistema con más de una articulación, el tiempo de tránsito estimado viene determinado por el incremento de ángulo $\triangle\varphi$ más grande. Puesto que la constante de tiempo se estima igual para todos los servos, el tiempo de tránsito mayor es debido al incremento de ángulo más grande.

Se define la distancia entre dos vectoes de estado físico como el mayor de los incrementos angulares, en valor absoluto. Sean $E_1(t_i)$ y $E_2(t_j)$ dos vectores físicos de cuatro componentes, definidos de la forma:

$$E_1 = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$$
$$E_2 = \{\beta_1, \beta_2, \beta_3, \beta_4\}$$

la distancia se define como:

$$d(E_1, E_2) = Max\left\{|\alpha_1 - \beta_1|, |\alpha_2 - \beta_2|, |\alpha_3 - \beta_3|, |\alpha_4 - \beta_4|\right\}$$

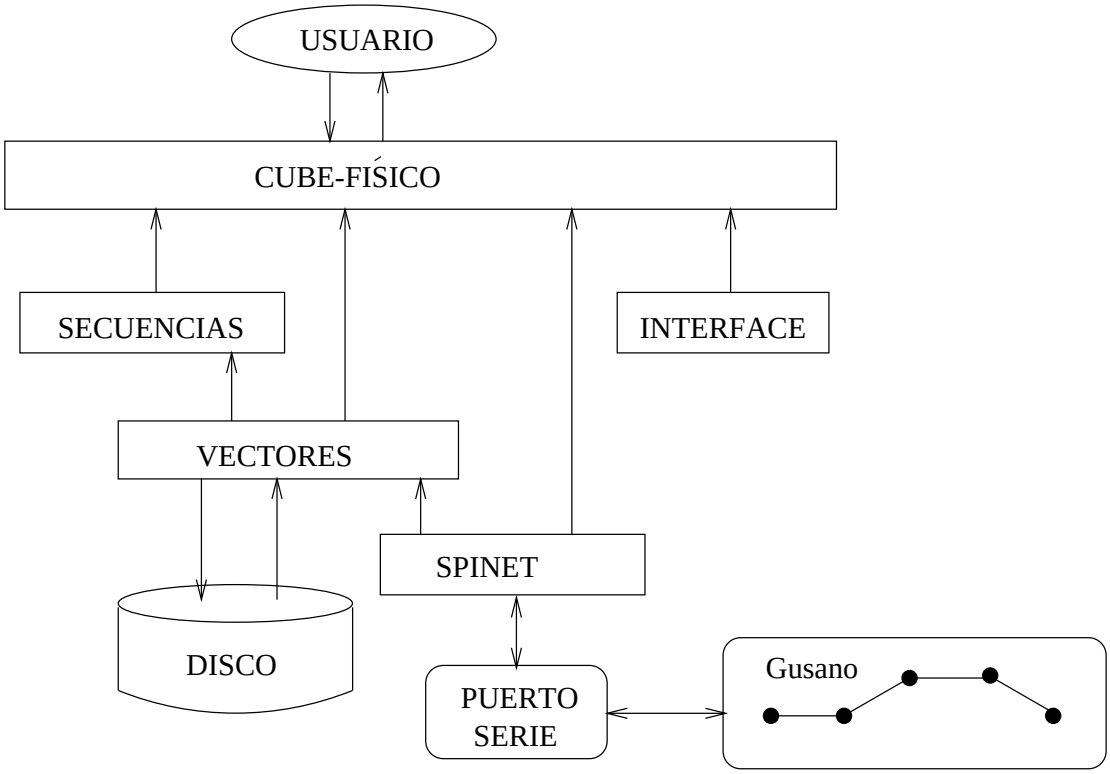


Figura 10.4: cf_arquitectura

De esta forma, el tiempo de tránsito estimado entre los estados E_1 y E_2 es:

$$t_{t_e} = \tau_e.d(E_1, E_2) \tag{10.3}$$

que está estimando el tiempo que tarda el gusano en alcanzar el estado E_2 a partir del estado E_1. El valor por omisión empleado en el programa `cube-fisico` para el parámetro τ_e es de 3.8ms, pero puede ser modificado por el usuario.

10.4. Arquitectura software

La aplicación **`cube-físico`** está formado por los siguientes módulos:

- **`interface`**: Dibujo del interfaz
- **`spinet`**: Acceso a la red de microcontroladores
- **`secuencias`**: Gestión de secuencias
- **`vectores`**: Operaciones con vectores de estado. Es el mismo módulo que el empleado en *cube-virtual*, pero con funcionalidad añadida.

- **`cube-fisico`**: Programa principal

En la figura 10.4 se muestra la relación entre todos los módulos. Los módulos software se representan con un rectángulo. De ellos parten unas flechas que indican los módulos que están haciendo uso de sus funciones de interfaz.

10.4.1. Módulo spinet

Este módulo implementa todos los servicios ofrecidos por la red de microcontroladores (*spinet*). En todas las llamadas a su interfaz hay que especificar la dirección del nodo de la red. De todos los servicios ofrecidos, en el programa `cube-fisico` sólo se utilizan los que permiten posicionar los servos, especificando la posición en grados. El módulo vectores es el único que realiza llamadas a este módulo:

- **`void spinet_cambiar_led(byte dir);`**

  Servicio de cambio del estado del led.

- **`void spinet_activar(byte dir, byte mask);`**

  Activar los servos indicados en la máscara.

- **`void spinet_puertoc(byte dir, byte valor);`**

  Enviar un valor al puerto C. Esta función se utiliza para acceder a través de la red a recursos que no sean leds.

- **`void spinet_posicionar(byte dir, byte motor, byte pos);`**

  Posicionar el servo en la posición indicada. Las posiciones vienen indicadas en grados futaba.

- v**`oid spinet_pos_sistema(byte dir, byte fut1, byte fut2, byte fut3, byte fut4);`**

  Posicionar los 4 servos en las posiciones indicadas. Esta función se utiliza para no tener que llamar 4 veces seguidas a la función *spinet_posicionar()*. Las posiciones se especifican en grados futaba.

- **`void spinet_set_pos_grados(byte dir,byte fut, double pos);`**

  Establecer la posición de un servo, en grados.

- **`void spinet_set_pos_sistema_grados(byte dir,double fut1, double fut2, double fut3, double f4);`**

  Establecer la posicion de los cuatro servos, en grados.

- **`double spinet_to_grados_fut(double grados);`**

  Convertir de grados a grados futaba.

10.4.2. Módulo vectores

Este es el mísmo módulo que el realizado para la aplicación `cube-virtual`. Sin embargo, el interfaz está ampliado, puesto que incluye funciones para posicionar los servos a partir de un vector de estado físico. Se trabaja con el tipo abstracto de datos `vector_pos_t`, que representa vectores de estados físicos y el tiempo de espera.

```
typedef struct vector_pos_s {
  double pos[NUM_ART];
  unsigned int te;
} vector_pos_t;
```

La descripción de este tipo de datos así como del interfaz se encuentra en la sección 7.4.4. La nueva función de interfaz incorporada es:

- **`void vector_posicionar(vector_pos_t *v);`**

  Enviar el vector de posición especificado a la red de microcontroladores para que se posicionen los servos.

10.4.3. Módulo secuencias

Este módulo nos permite trabajar con secuencias. Una secuencia es una lista de vectores de estado, con un tiempo de espera asociado. Este módulo nos permite anadir vectores, eliminarlos, grabar secuencias a disco, etc. Las secuencias se almacenan en una lista doblemente encadenada, gestionada por el propio módulo y que es transparente al módulo que lo usa.

- **`void sec_init();`**

  Inicialización del módulo. Se crea la lista y se inicializan las variables internas. Es obligatorio que se llame a esta función antes de utilizar cualquier otra. En cualquier momento se puede volver a llamar a esta función para volver al estado inicial.

- **`int sec_num_vectores();`**

  Devolver el número de vectores de estado que hay en la secuencia.

- **`vector_pos_t *sec_get_vector(int numvect);`**

  Devolver un puntero al vector solicitado. El vector se identifica mediante un número entero comprendido entre 0 y el mayor número, devuelto por *sec_get_vector()*.

- **`void sec_anadir_vector(vector_pos_t *v);`**

  Añadir un vector al final de la lista.

- **`void sec_situar_vector(int numvect, vector_pos_t *v);`**

  Situar un vector en la posición indicada. El vector que ocupaba esa posición se elimina.

- **`void sec_insertar_vector(int numvect, vector_pos_t *v);`**

  Insertar un vector en la posición indicada. La secuencia aumenta en un vector.

- **`void sec_delete_vector(int numvect);`**

  Borrar un vector. La secuencia se reduce en un vector.

- **`void sec_grabar_vectores(FILE *f);`**

  Grabar la secuencia de vectores de un fichero.

- **`int sec_load_vectores(FILE *f);`**

  Leer la secuencia de vectores de un fichero.

10.4.4. Módulo interface

Este módulo se encarga de crear y visualizar todo el interfaz. Solo tiene una función en el interfaz. La función devuelve una estructura con información sobre el interfaz: puntero a los botones, a los cuadros de diálogo, etc.

- **`void create_window1 (main_state_t *estado);`**

10.4.5. Programa principal cube-fisico

Este es el programa principal que implementa toda la funcionalidad y que se encarga de llamar al resto de módulos. Todo el control del interfaz, las funciones de evento asociadas a los botones (callbacks), se encuentran implementados en este módulo.

10.5. Manejo del programa

10.5.1. Descripción del interfaz de usuario

En el interfaz hay 5 partes, mostradas en la figura 10.5:

1. **Menú**. En esta parte se encuentra el menu File y una pantalla para mostrar información al usuario.
2. **Movimiento de los servos**: Posicionamiento independiente de los servos
3. **Vector de estado físico**: Establecer un vector de estado que deben cumplir los servos
4. **Control de sucuencias**: Interfaz para trabajar con secuencias de estados
5. **General**: Funciones generales

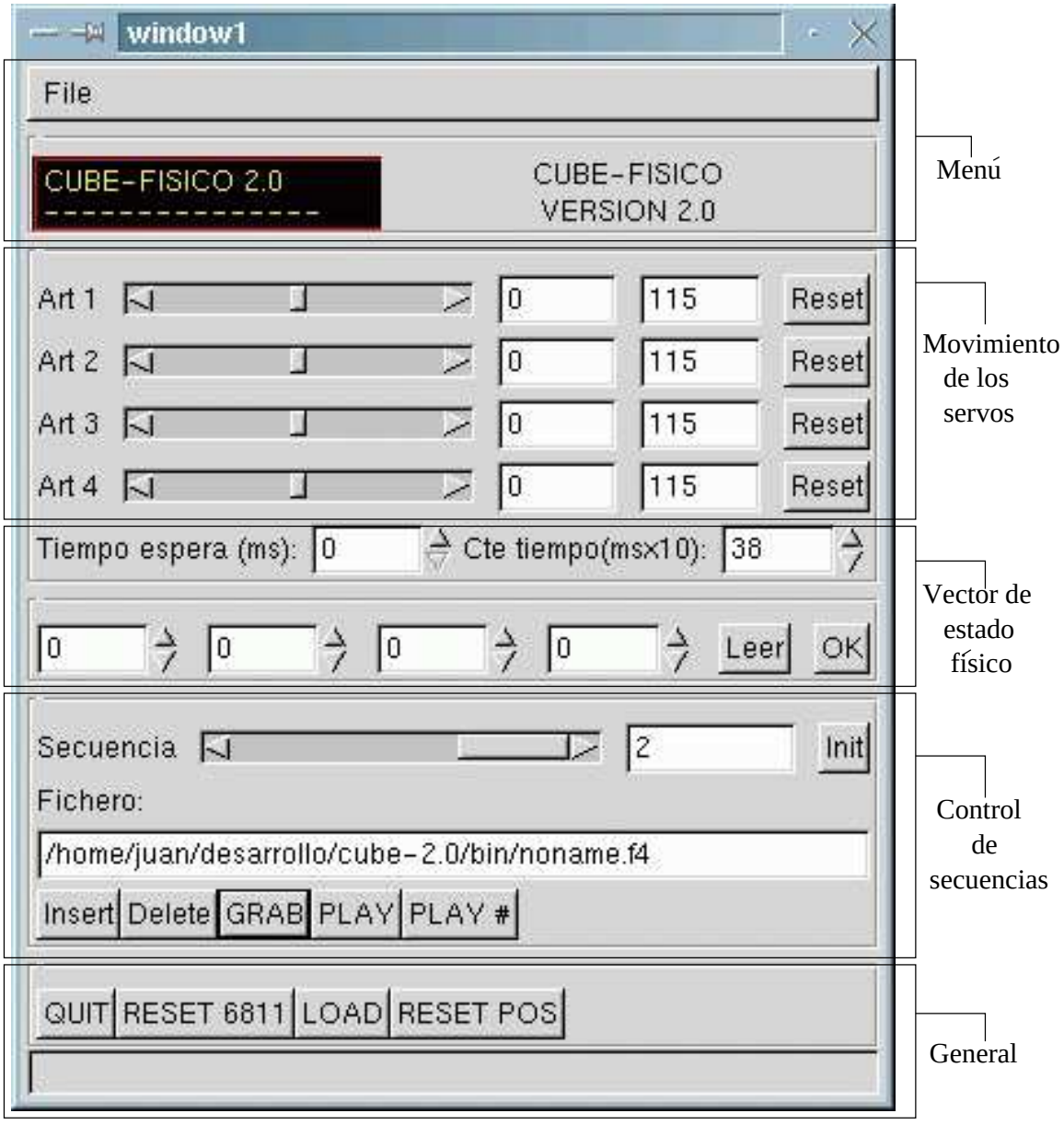


Figura 10.5: Diferentes partes del interfaz

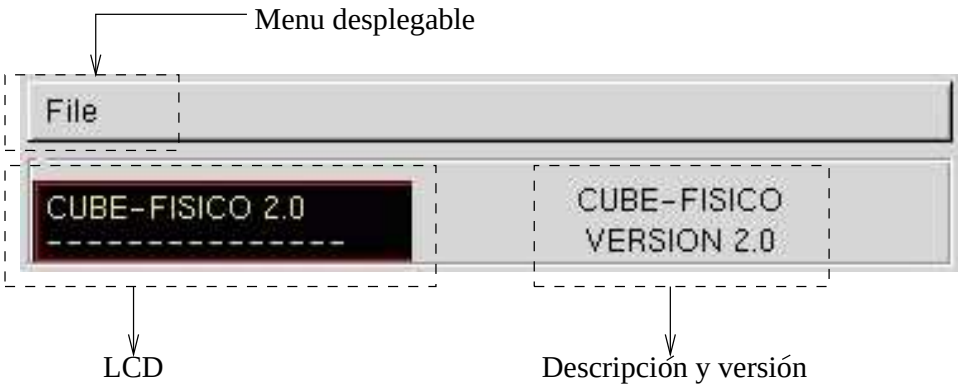


Figura 10.6: Parte superior del interfaz

10.5.2. Menú

Los diferentes elementos de esta parte del interfaz se presentan en la figura 10.6. Son:

- **Menú desplegable** (Menú File). Dispone de las siguientes opciones:
  - **New**: Crear una secuencia nueva. Se establece el nombre *noname.f4* por defecto.
  - **Open**: Abrir una secuencia nueva y cargarla en memoria.
  - **Save**: Grabar la secuencia actual, con el nombre asignado.
  - **Save as**: Grabar la secuencia con un nombre nuevo.
  - **Quit**: Salir del programa.

  La extensión por defecto para los ficheros de secuencias es .f4, haciéndose referencia a que los vectores de estado que se están manejando tienen 4 componentes y están pensados para los servos Futaba.
- **Descripción y versión del programa.**
- **Pantalla** donde se presentan mensajes al usuario. Inicialmente contiene el número de versión. Aqui se presentan mensajes de error que puedan ocurrir durante la ejecución.

10.5.3. Movimiento de los servos

Todos los elementos se muestran en la figura 10.7. Esta parte del interfaz permite controlar los servos por separado y visualizar su estado, tanto en grados como en grados futaba. Los diferentes elementos son:

- **Barras de desplazamiento**. Permiten situar cada servo en cualquiera de sus posiciones. Se puede desplazar el cursor con el ratón o se puede pinchar sobre las flechas para mayor precisión.

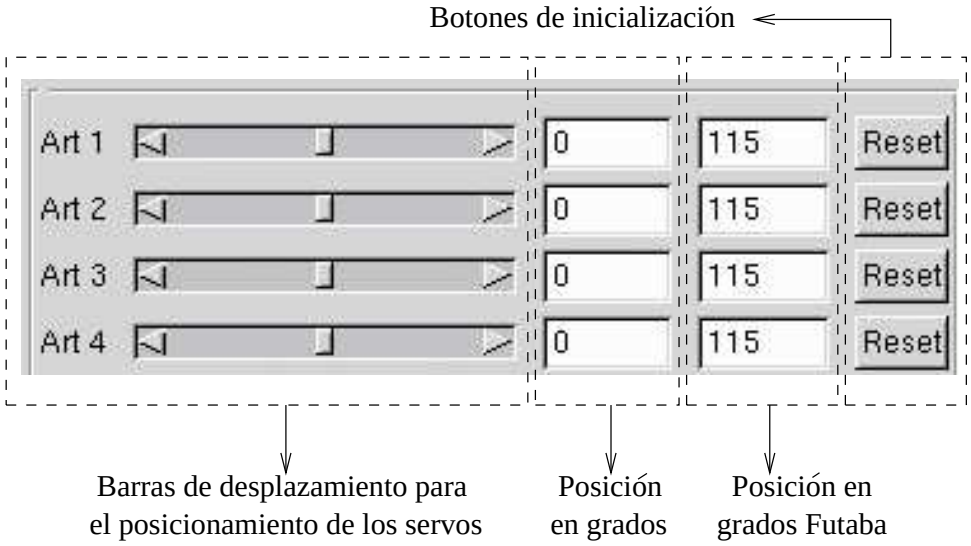


Figura 10.7: Posicionamiento de los servos

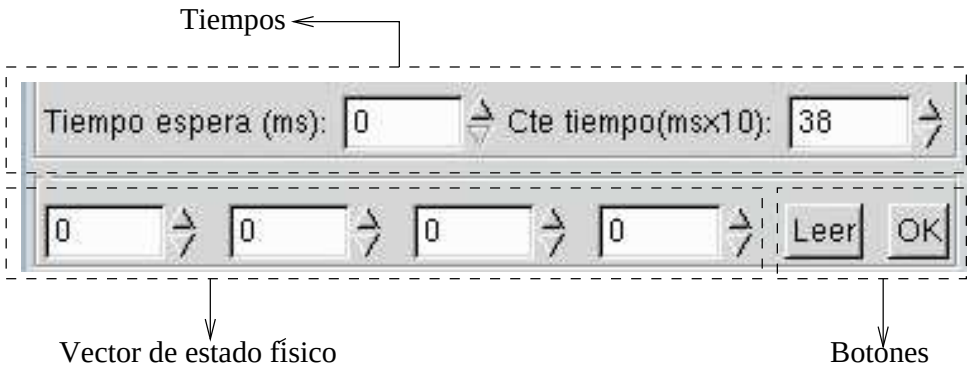


Figura 10.8: Parte del interfaz relacionada con los vectores de estado físicos

- **Visualización en grados**. Se muestra el estado de cada servo en grados
- **Visualización en grados futaba**. Idem pero en grados futaba
- **Botones de inicialización**. Llevan cada servo a su estado 0.

Esta parte del interfaz es la más "vistosa" porque resulta curioso el ver moverse los servos al actuar sobre las barras de desplazamiento. También es una forma de llevar el gusano a un estado extraño, adoptando formas diferentes y viendo qué estados son inestables.

10.5.4. Vector de estado físico

Esta parte permite la entrada de vectores de estado y controla los tiempos y parámetros asociados a la reproducción de secuencias. Las partes son (figura 10.8):

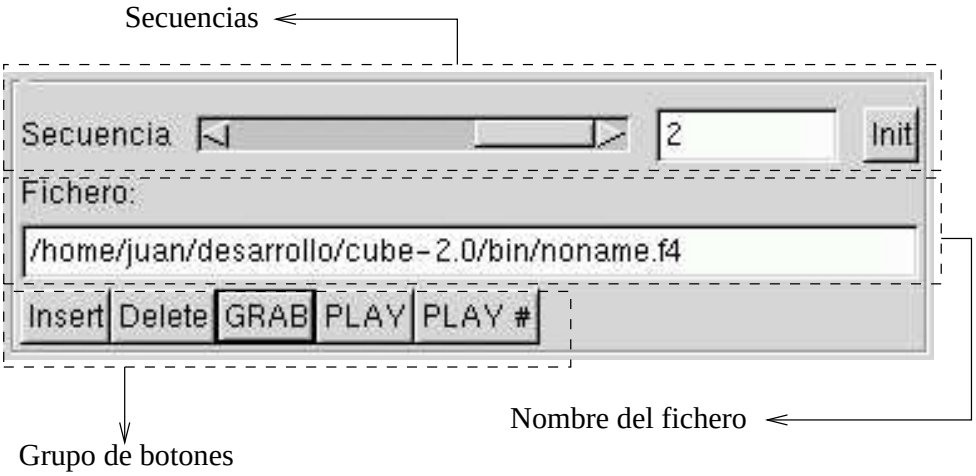


Figura 10.9: El interfaz para el control de las secuencias

- **Vector de estado**: Aquí se introduce el estado para cada servo, especificándolo en grados. Se queda introducido, pero no se hace nada con él. Para enviárselo al gusano hay que pulsar el botón de OK.
- **Botones**:
  - **Boton de OK**: Se utiliza para enviar el vector de estado actual al gusano físico.
  - **Botón Leer**: Se lee el vector de estado de las barras de desplazamiento y se visualiza.
- **Tiempos**.
  - Establecer el tiempo de espera, en milisegundos, para el vector de estado actual
  - Establecer la constante de tiempo τ, en centésimas de segundo[2].

10.5.5. Control de secuencias

Esta es la parte que nos permite generar, editar y reproducir secuencias. Está constituido por tres partes:

- **Nombre del fichero actual**, con el que estamos trabajando. En caso de no haber definido ninguno, se toma por defecto noname.f4, en el directorio desde donde se haya ejecutado la aplicación.
- **Navegación por los vectores**:

[2]Se utiliza centésimas de segundo para que el número sea entero. El valor por omisión es de 3.8ms = 38 centésimas.

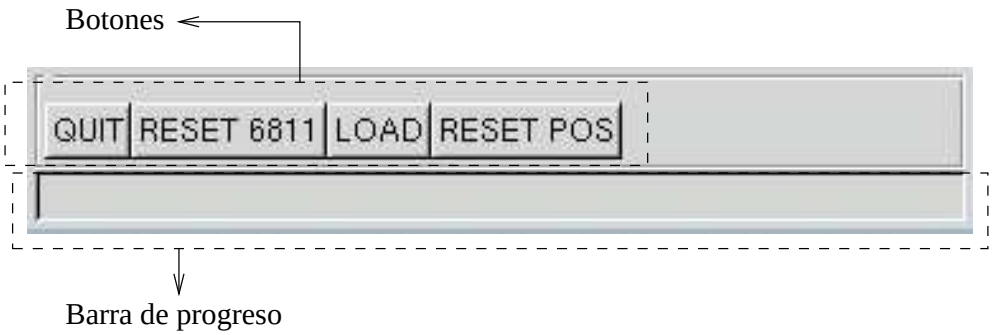


Figura 10.10: Botones generales

- La **barra de desplazamiento** nos permite navegar por los vectores que componen la secuencia: ir hacia adelante, hacia atras, etc. Cada vez que se cambia la posición de esta barra, se actualizan las barras de desplazamiento que indican la posición de los servos de acuerdo con el nuevo vector y se envían al gusano físico.
- **Número de vector actual**: Indica el vector de estado en el que nos encontramos.
- **Botón de inicio**: Permite ir directamente al vector de estado 0.

- **Grupo de botones**:
  - **Insert**: Insertar en la secuencia el vector determinado por las barras de desplazamiento. Se inserta en la posición actual, desplazando los vectores a posiciones superiores.
  - **Borrar**: Eliminar el vector actual.
  - **Grab**: Añadir un vector en la posición actual y avanzar. Cuando se está creando una secuencia manual es el botón que más se emplea.
  - **Play**: Reproducir la secuencia actual. Se comienza por el vector 0 y cuando se llega al último se vuelve a comenzar.
  - **Play #**: Reproducir sólo una vez. Se comienza por el vector 0 y se reproduce hasta llegar al último.

10.5.6. General

Esta parte del interfaz está formada por (figura 10.10)

- **Barra de progreso**: Indica el estado de carga del programa servidor para el micro maestro de la red.
- **Botones**:
  - **Quit**: Salir del programa
  - **Reset 6811**: Hacer un reset de la placa maestra.

- **Load**: Cargar el programa *maestro* en la placa maestra
- **Reset Pos**: Ir al estado (0,0,0,0).

10.5.7. Manejando las articulaciones

Para probar el gusano lo primero es mover individualmente los servos. Para ellos hacemos lo siguiente:

1. Alimentar la electrónica con las pilas y los servos mediante una fuente externa.
2. Conectar el gusano al PC.
3. Apretar el botón de reset de la tarjeta BT6811. Se debe encender el led verde.
4. Apretamos el botón de LOAD, para cargar el programa *maestro* en la RAM de la CT6811 y tener acceso a la *spinet*.
5. Mediante las barras de desplazamiento se podrán mover los servos.

10.5.8. Reproducción de secuencias

Para reproducir una secuencia previamente generada, bien por el programa *cube-fisico* o por *cube-virtual*, hay que hacer lo siguiente:

1. Seguir los pasos 1-4 del apartado 10.5.7.
2. Seleccionar la opción OPEN del menú File.
3. Seleccionar la secuencia que se quiere reproducir.
4. Apretar el botón de *play* para una reproducción continua o el de *play #* para sólo una vez.

10.5.9. Generación de secuencias

1. Seguir los pasos 1-4 del apartado 10.5.7.
2. Seleccionar el estado en el primer instante, bien mediante las barras de desplazamiento o escribiendo el vector de estado físico.
3. Introduccir el valor del tiempo de espera. Por defecto es cero.
4. Pulsar el botón *grab* para grabar el estado en la secuencia.
5. Volver al paso 2.

Mediante la barra de desplazamiento de las secuencias se puede navegar por los diferentes estados, insertando nuevas posiciones o cambiando los datos.

10.6. Resumen

Para controlar el **gusano físico** es necesario un software que envíe la información sobre la posición de los servos a la red de microcontroladores. Esta información la puede introducir directamente el usuario, mediante un interfaz amigable, para hacer calibraciones o generar secuencias de movimiento manuales. Las **secuencias** también se pueden cargar desde un fichero externo, generado por la aplicación *cube-virtual*.

Para la **reproducción de las secuencias** hay que tener en cuenta dos parámetros muy importantes, el **tiempo de tránsito** y el **tiempo de espera**. El tiempo de espera lo define el usuario, y nos está indicando el tiempo que queremos que el gusano permanezca en un estado determinado. El tiempo de tránsito depende del servo y mide el tiempo que tardan los sevos en posicionarse en un nuevo estado. Este parámetro es estadístico y hay que estimarlo a partir de la **constante de tiempo** del servo y de la **distancia entre los vectores de estado físicos**. Existe un **compromiso** entre *continuidad* del movimiento y *exactitud* en las posiciones. El diseñador debe seleccionar el valor de la constante de tiempo estimada que más se ajuste a su aplicación.

El programa **cube-fisico** dispone de un interfaz muy sencillo que permite cargar, generar, reproducir y manipular las secuencias que se envían al gusano físico.

Capítulo 11

Conclusiones y líneas futuras

11.1. Conclusiones

En este apartado analizaremos todo lo que se ha conseguido en relación con los **objetivos planteados inicialmente**, que fueron:

1. **Diseño y construcción de un robot tipo gusano**, que se desplace horizontalmente y en línea recta de forma análoga a como lo hacen los gusanos, mediante **ondas transversales** que se propagan desde la cola hasta la cabeza.

2. Desarrollo del **cálculo automático de las secuencias de movimiento** necesarias para que se desplace en función de la onda aplicada.

3. Diseño de un **software de alto nivel**, para un ordenador PC, **que pueda generar las secuencias** de movimiento del gusano según cómo sea la forma de la onda que lo recorra, su periodo y su amplitud.

4. Estudio de alternativas para la incorporación de **mecanismos de giro**.

Este proyecto nació como un reto personal del autor. Junto con mis compañeros de Microbótica[1], desarrollamos una electrónica y mecánica que nos permitió construir microbots móviles y con patas. Sin embargo no se tenía experiencia en el campo de los *ápodos*, o robots sin patas.

Para comprobar la viabilidad del proyecto, se desarrolló un primer prototipo, **Cube-1.0**, con 4 articulaciones y la electrónica necesaria. Se implementó un software para reproducir una secuencia generada manualmente, que hacía que el gusano se desplazase. **Primer objetivo cumplido**.

Se empezó a estudiar la manera de generar las secuencias automáticamente, obteniéndose una primera versión del **algoritmo de ajuste**, que permite que el gusano adopte la forma de cualquier función. Desplazando esta función, conocida como **función de contorno**, y ajustando el gusano a ella, se conseguía obtener los diferentes estados que permitían que el gusano avanzace. **Segundo objetivo cumplido**. Se implementó en un software que calculaba las secuencias y las enviaba al gusano. **Tercer objetivo cumplido**.

[1] Más conocidos en la ETSI telecomunicación de la UPM como Grupo J&J: Andrés, Juanjo, Cristina y Juan.

Finalmente se realizó un estudio sobre los mecanismos de giro. **Cuarto y último objetivo cumplido**.

En la **versión final de este proyecto**, los objetivos no sólo se han cumplido sino que se han generado unos **resultados intermedios que abren unas líneas de trabajo muy interesantes** en el campo de los *ápodos*:

1. La **estructura mecánica** de *Cube-1.0* era muy artesanal y estaba orientada sólo a un gusano de 4 articulaciones. Este prototipo se modificó para obtener un diseño modular, que permitiese construir gusanos de cualquier longitud, utilizando muy pocas piezas diferentes, sencillas de realizar y muy económicas. El prototipo final, **Cube-2.0,** tiene 4 articulaciones, las mismas que *Cube-1.0*, sin embargo el no tener más articulaciones es ahora una cuestión puramente económica: sólo hay que comprar más servos y clonar las piezas ya existentes. No hay que rediseñar la estructura, aunque sí hay que modificar el interfaz de usuario del software.

2. La **electrónica** de *Cube-1.0* permitía mover los 4 servos a través de un microcontrolador, además de facilitar el control desde un PC. Para conseguir un prototipo ampliable, se introdujo una **red de microcontroladores** de manera que la electrónica fuese también ampliable para poder mover un número mucho mayor de servos y poder construir gusanos de mucha mayor longitud. Igual que la ampliación mecánica, la ampliación electrónica es ahora un problema puramente económico, pero no técnico.

3. Para programar el **software** e implementar los algoritmos de movimiento se desarrolló un **modelo de gusano transversal**. Este modelo se fue simplificando hasta llegar a las ideas básicas que permitieron comprender mejor los mecanismos de movimiento y dieron pie al desarrollo de un **modelo virtual de gusano**, en el PC, sobre el que se puede interactuar, hacer cálculos, obtener los vectores de estado de las articulaciones y generar secuencias automáticas de movimiento, en función de los parámetros de las funciones de contorno. Sobre este modelo fue posible aplicar todos los algoritmos de avance, así como deducir otras propiedas muy importantes de estos robots.

4. No sólo se ha desarrollado un modelo detallado de los **gusanos transversales** de cualquier longitud, sino también se ha hecho para los **gusanos longitudinales**, sobre el que es más fácil trabajar a nivel teórico. Ambos modelos se han relacionado y se han obtenido los parámetros fundamentales que los caracterizan, así como las reglas que se deben cumplir para que se desplacen.

5. Para el estudio de los mecanismos de giro se ha desarrollado un **modelo de gusano tridimensional** que se ha relacionado con los dos modelos anteriores y lo que ha permitdo no sólo obtener las ideas para que gire, sino para que describa cualquier trayectoria indicada. La implementación de estas ideas implica el rediseño de la estructura mecánica, por lo que no se ha llevado a la práctica, pero abre una nueva vía de trabajo en la realización de un gusano tridimensional virtual.

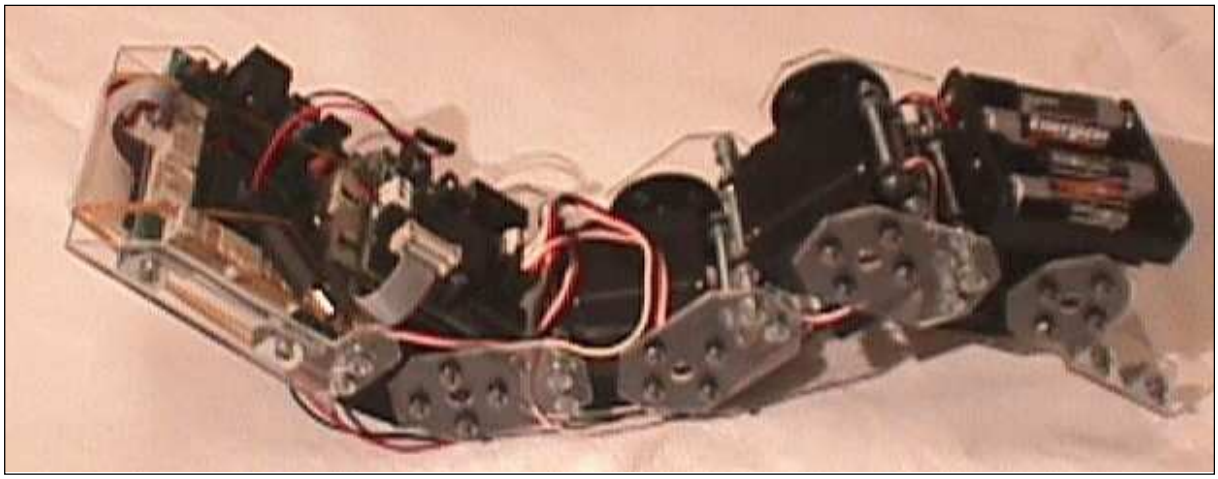


Figura 11.1: Prototipo final, Cube-2.0

6. Para la calibración y las pruebas mecánicas y eletrónicas, se ha diseñado un **software** que permite mover directamente los servos, a través de un **interfaz gráfico amigable**, de manera que se pueden generar secuencias manuales que no sólo sean de desplazamiento. Este mismo programa lee las secuencias generadas automáticamente por el gusano virtual.

Lo obtenido finalmente, a modo de resumen es lo siguiente:

1. Un **gusano físico**, de cuatro articulaciones, que se desplaza en línea recta y que es totalmente ampliable, tanto a nivel mecánico como a nivel electrónico (ver figura 11.1).

2. Un **software** que permite controlar los servos desde el PC, generando secuencias manuales, que se pueden grabar en ficheros y cargarlas para su posterior reproducción(ver figura 11.2).

3. Un **entorno de trabajo virtual** para profundizar en los mecanismos de movimiento sin necesidad de tener el gusano físico. Se generan secuencias automáticas de moviento que se pueden probar en el propio robot virtual o se pueden grabar en un fichero para su reproducción por el software del punto 2.

4. Un **modelo** detallado de los **gusanos transversales**, **longitudinales** y **tridimensionales**, obteniéndose ecuaciones y propiedades muy importantes.

El movimiento de avance logrado, calculado con el gusano virtual y ejecutado por el gusano físico, es muy bueno. Se pueden ver perfectamente las ondas que lo recorren y cómo va avanzando sin apenas arrastrarse, lo que le permite desplazarse por cualquier tipo de superficie. Puede superar obstáculos que no sean de demasiada altura y subir por superficies inclinadas.

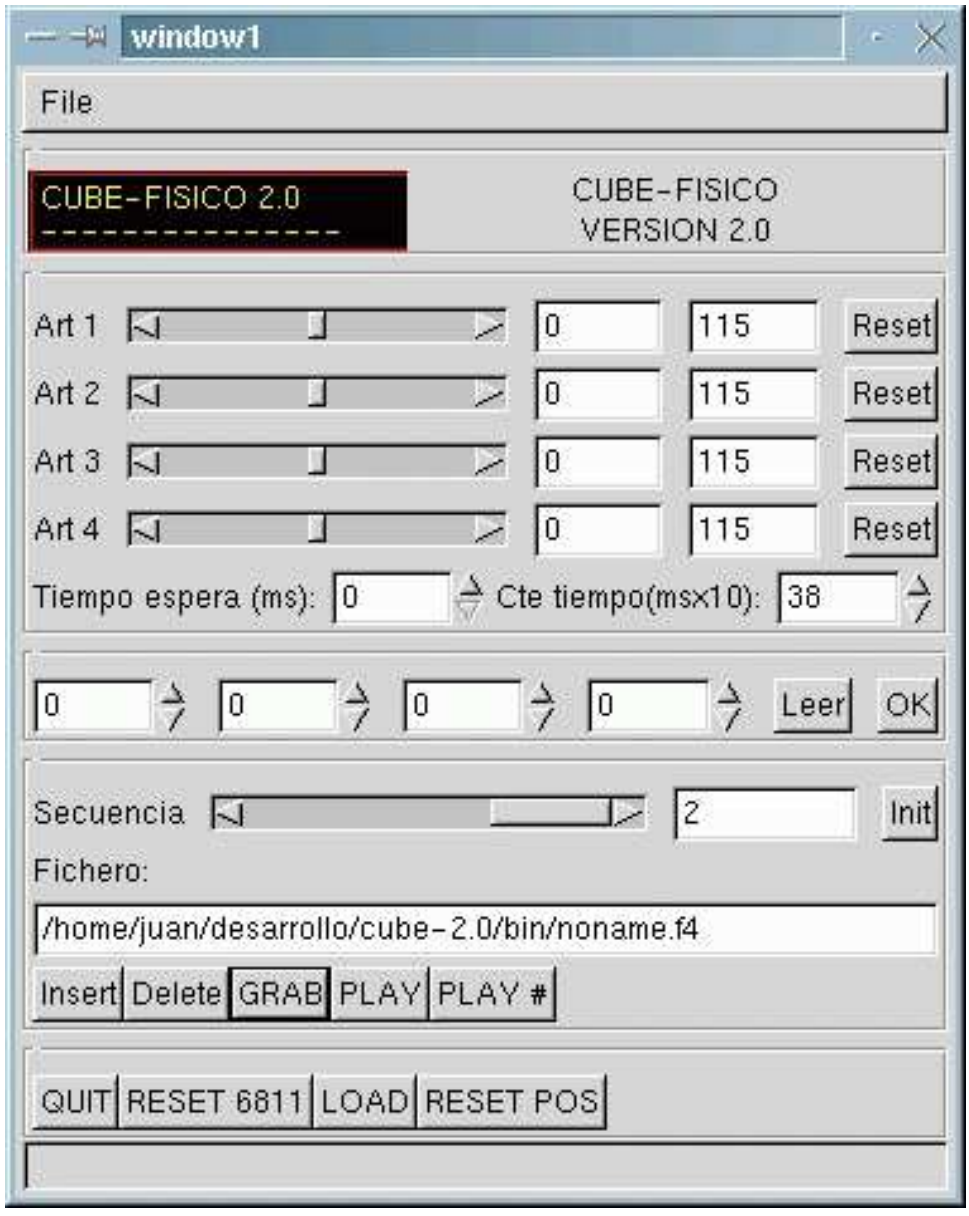


Figura 11.2: Programa `cube-fisico` para el control del gusano físico

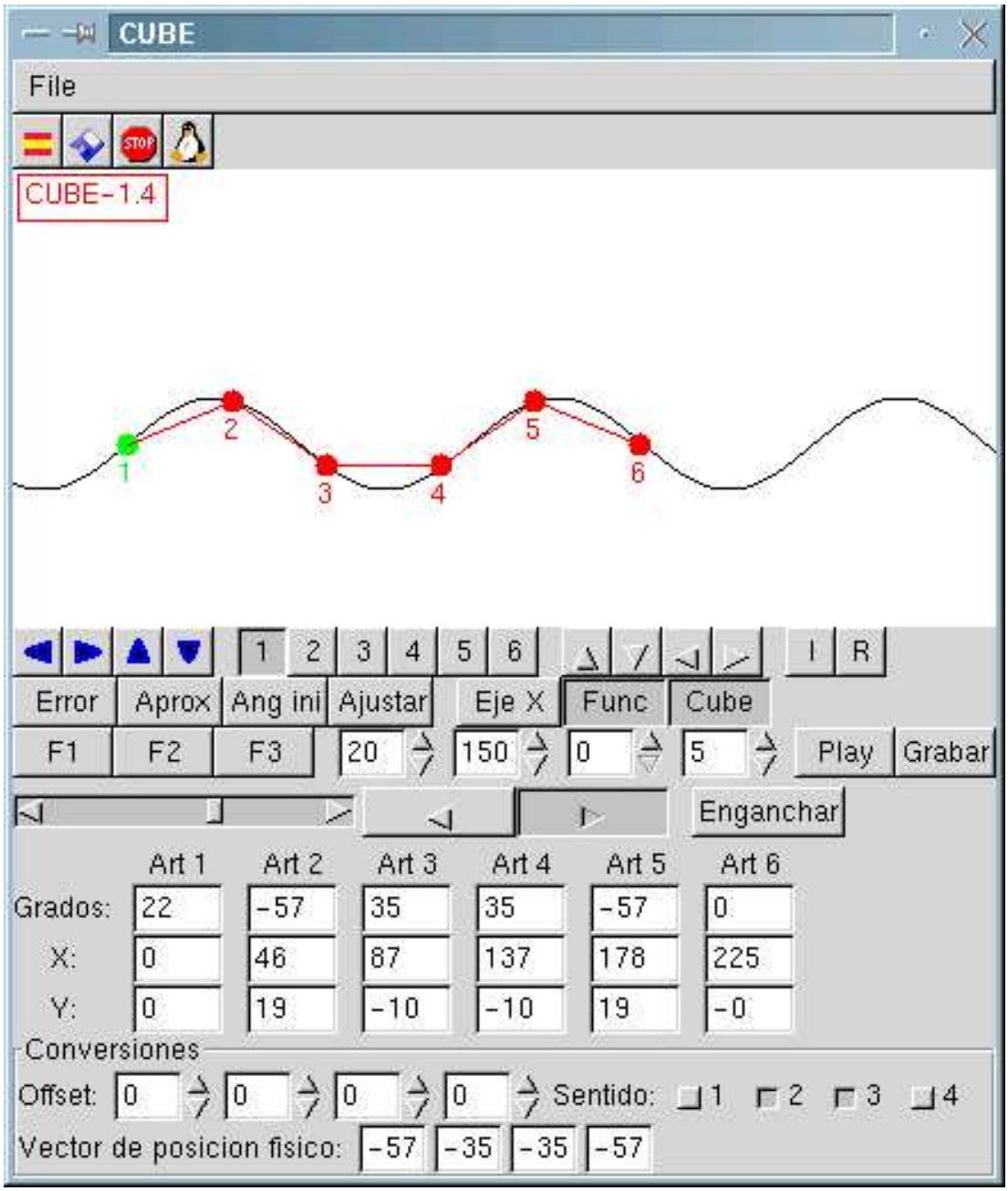


Figura 11.3: Entorno de trabajo virtual, para un gusano transversal

11.2. Líneas futuras

El proyecto abre nuevas líneas de trabajo, en diferentes niveles:

1. **Mecánica**: Desarrollar una estructura modulable para los **gusanos tridimensionales**. Los módulos básicos deben ser sencillos e iguales, constituidos por dos servos que permitan la orientación del módulo en cualquier dirección del espacio. La misma electrónica desarrollada sirve para estos módulos. En la nasa están trabajando en este tipo de estructuras [18].

2. **Electrónica**: Dotar al nodo maestro de mayor potencia de cálculo para que no haga falta disponer de un PC para el movimiento del gusano físico. Para este propósito pueden utilizarse placas microcontroladores con **Linux empotrado**, que además permiten adaptar casi directamente el software desarrollado en el PC al propio sistema empotrado. Para más información consultar [32].

3. **Software**:

   a) **Unificación de los programas `cube-fisico` y `cube-virtual`** en un único entorno que permita acceder al gusano físico a partir del virtual.

   b) **Entorno de trabajo virtual para gusanos tridimensionales**, que permita generar secuencias de movimiento que sean reproducidas en el gusano físico.

   c) **Interfaz web**, para cambiar los parámetros de movimiento del gusano a través de un navegador. Esto es posible si se utiliza para el *nodo maestro* un sistema empotrado como el indicado en el punto 2. Conectando el gusano a una red Ethernet de 10Mbits, y desde un ordenador de esa red, que se puedan cambiar los paráameros de movimiento. Se puede encontrar más información en [33].

4. **Nivel teórico**: Hacer un estudio para que el gusano tridimensional puede **seguir trayectorias** no sólo en el plano xy, sino también **en un espacio tridimensional**, realizando trayectorias más complejas como por ejemplo helicoidales que le permitan trepar o avanzar por cilindros de un diámetro determinado.

El trabajar con sistemas empotrados con un sistema operativo dentro es fundamental para desarrollar robots autónomos complejos donde la coordinación de los servos no es trivial y se necesita un software más potente que genere las secuencias.

El construir un gusano tridimensional es muy útil no sólo en el campo de los ápodos. En realidad se trata de un “segmento orientable” que puede tener muchas otras aplicaciones como por ejemplo introducir una cámara en rincones poco accesibles o construir robots con miembros de este tipo, como pulpos o calamares. En ellos los miembros no sólo sirven para desplazarse sino para otras funciones: reconocimiento del terreno, arragarse a objetos, trepar, etc.

Este proyecto pretende dejar establecidas las bases para trabajar con este tipo de robots, partiendo de uno más simple y que se ha implementado con éxito.