DISEÑO DE UN ROBOT HEXAPODO
Hardware y Software de control

MEMORIA DEL PROYECTO

REALIZADO POR
Alberto Sánchez Espinosa
Carlos López Cardelo

DIRIGIDO POR
Pere Ponsa

ROBOTICA INDUSTRIAL

Escola Universitària Politécnica de Vilanova i la Geltrú
UPC - EUPVG

Escola Universitària Politécnica de Vilanova i la Geltrú
UPC - EUPVG

Reunido el Tribunal examinador en el dia de la fecha, costituido por:

D._____

D._____

D._____

para juzgar el proyecto **DISEÑO DE UN ROBOT HEXAPODO**
de los alumnos D. Alberto Sánchez Espinosa y D. Carlos López Cardelo,
dirigido por D. Pere Ponsa,

ACORDO POR _____ OTORGAR LA CALIFICACIÓN DE

_____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPONENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

Vilanova i la Geltrú, a de de

El Presidente El Secretario El Vocal

Fdo.: Fdo.: Fdo.:

INDICE DE CONTENIDOS

1. Introducción

Se ha llevado a cabo el diseño de un robot móvil articulado con patas y controlado por ordenador.
El objetivo de este trabajo es realizar el diseño de un robot experimental con complejidad reducida.

Se probarán con diversas estructuras mecánicas, servomecanismos de posicionamiento, electrónica de generación de señales y software jerarquizado y modular.

Se compararán diferentes estructuras mecánicas, componentes electrónicos y arquitecturas de software para el diseño de un robot móvil de 6 patas.

El trabajo se divide en 6 grandes capítulos y dos apéndices, en los que se describen los componentes que serán utilizados para la construcción del robot y la valoración económica que supondrá la construcción del robot HEXAPODO.

En el primer capítulo se declararán los objetivos de este trabajo y la justificación del mismo.

En el segundo capítulo se tratará de realizar una visión genérica de la robótica móvil, comparando los tipos de robots móviles existentes hoy en día en el mercado y situando el ROBOT HEXAPODO dentro de este campo.

En el resto de capítulos se entrará de lleno en el diseño de un robot móvil de 6 patas. A lo largo de estos 4 capítulos se describirá tanto el diseño previo, como el diseño de las estructuras mecánicas y electrónicas, así como también el software que se implementará sobre el robot.
De ese modo, tendremos los primeros pasos para la construcción de un robot móvil HEXAPODO.

Hay que reseñar, que este es un proyecto en el que se dan los primeros pasos para la construcción de un robot móvil. En este proyecto, debido a la falta de tiempo y de recursos, no se define el diseño final de un robot hexapodo, simplemente es una guía o un camino a seguir, mediante el cual, después de seguir investigado y realizando diversas pruebas con distintos componentes tanto electrónicos como mecánicos y de software, podemos realizar la construcción del robot.

2. Objetivos y justificación del trabajo

El principal objetivo de este trabajo es el diseño y construcción de un robot móvil con patas y complejidad reducida.

Este objetivo plantea esfuerzos de investigación en los siguientes campos:

- Diseño de estructuras mecánicas poliarticuladas, capaces de posicionar de manera eficiente el sistema robot (maximización del espacio alcanzable frente a minimización de colisiones entre los elementos).

- Estudio y búsqueda de sistemas mecánicos de servocontrol (minimización del tamaño, peso y coste frente a optimización del control en el sentido de fuerza ejercida y reducción del error).

- Arquitecturas de control (autonomía frente a control remoto).

- Diseño de la electrónica de control y de los interfaces (estructuras digitales para el mantenimiento de las señales de posicionamiento y elección de dispositivos de entrada/salida para el control desde el software).

- Sistemas software estructurados en niveles de abstracción, capaces de planificar eficientemente movimientos complejos del robot (arquitecturas reactivas frente a arquitecturas jerarquizadas).

-Interfaz para el manejo del robot por el usuario (facilidad de uso frente a potencia. Acceso a todos los niveles de control del robot, desde configuración del sistema a la simulación de los futuros movimientos y del comportamiento previsto).

En la implantación de todos estos sistemas se han encontrado los problemas y restricciones enumerados a continuación:

- Escasez de documentación sobre los dispositivos de servocontrol utilizados, y en general, sobre los robots de este tipo.

- Limitación de tiempo y de personal para el proyecto, debida principalmente a la extensión y la diversidad de los campos abarcados.

3. Antecedentes

La documentación existente en el campo de los robots móviles con patas es relativamente escasa, y se limita al desarrollo de sistemas dirigidos a objetivos muy concretos.

En este capítulo nos disponemos a dar una rápida visión de los robots móviles existentes hoy en día, y a describir el robot hexapodo.

3.1 Clasificación de los robots móviles

Existen varias formas de clasificar robots. Desde un primer punto de vista, podemos dividir éstos en *experimentales y aplicados.*

La primera categoría incluye aquellos robots de propósito general diseñados para realizar un conjunto bastante amplio de experiencias, es decir, aquellos pensados con un estricto enfoque de investigación.

La segunda categoría abarca los robots construidos con algún propósito específico (industrial, de exploración, etc...).
En esta clasificación, el ROBOT HEXAPODO, entraría en el primer grupo, debido a que su construcción persigue la creación de una línea de investigación en principio sólo limitada por la estructura física del robot, estando abierta a cualquier tipo de experiencias que se adapten a esta restricción.

Desde otro punto de vista, se pueden encontrar robots *móviles o estaticos.*
Los robots industriales son fundamentalmente estáticos, es decir, incapaces de desplazarse libremente por un entorno no limitado.
El ROBOT HEXAPODO es un robot móvil, por lo que a partir de ahora la clasificación se centrará en este tipo de dispositivos.

3.1.1. Robots móviles

Dentro de los robots móviles, se encuentra una primera división en robots *autónomos y no autónomos*. Los primeros portan todo el software y hardware de control sobre la estructura mecánica. Esto les da un rango de alcance limitado únicamente por la duración de las fuentes de alimentación que utilicen, pero encarece y produce una mayor complejidad en el sistema.

El ROBOT HEXAPODO que se va a diseñar, es un robot *no-autónomo*. Es gobernado por un ordenador externo al que se comunica a través de un bus de señales de datos y control. Las fuentes de alimentación son así mismo externas.

Desde un segundo punto de vista, los robots móviles pueden clasificarse atendiendo al medio de locomoción que utilicen.
Los robots con patas permiten desplazamientos más eficientes sobre terrenos de cualquier tipo (rugosos, con obstáculos o desniveles,...), además de ofrecer un control de estabilidad más completo y requerir menor potencia.
Los robots con otro tipo de locomoción (ruedas, orugas, ...) simplifican el posicionamiento y los cálculos necesarios para el mismo.

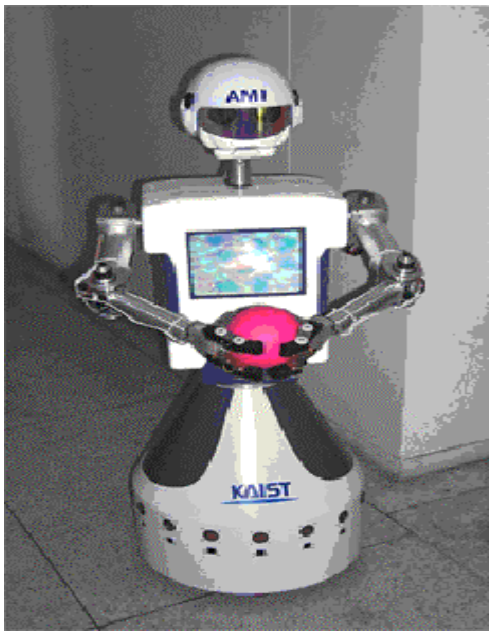


Figura 2.1.: Robot móvil de locomoción mediante ruedas

A continuación describimos los tipos de robots móviles con pata que existen en el mercado:

3.1.1.1. ROBOTS DE UNA SOLA PATA

El ejemplo más representativo es un robot desarrollado por el Instituto de Robótica y el Departamento de Informática de la Universidad americana Carnegie-Mellon.
Este dispositivo salta sobre su única extremidad, buscando continuamente la estabilidad dinámica, ya que la estática es imposible en general sobre menos de cuatro patas. Supone un gran esfuerzo en el desarrollo del sistema de control.

3.1.1.2. ROBOTS BÍPEDOS

Los robots de la serie BIPER, diseñados en la Universidad de Tokyo, pueden caminar lateralmente, avanzar y retroceder, simulando más o menos aproximadamente el modo de andar humano.

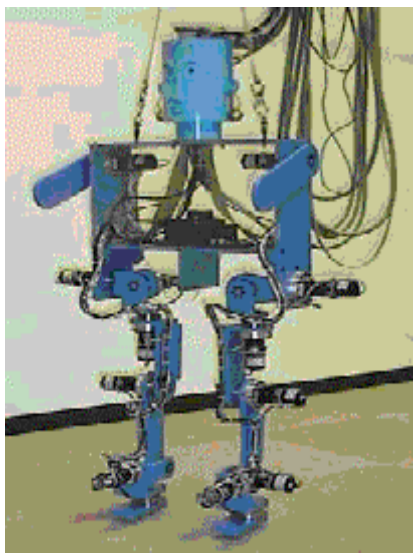


Figura 2.2.: Robot bípedo

3.1.1.3. ROBOTS CUADRÚPEDOS

Este tipo tiene más representantes, tanto fuera de España como en nuestro país. En el Instituto Tecnológico de Tokyo fue construido un vehículo de cuatro patas dotado de sensores táctiles y detector de posturas. Cada pata tiene 3 grados de libertad.
El control se realiza desde un microordenador que asegura la existencia de un triángulo de apoyo sobre 3 de las patas continuamente.
Por otra parte, en España, el *RIMHO* (Robot de Intervención en Medios HOstiles), diseñado conjuntamente por el Instituto de Automática Industrial (*IAI*) y el Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (*CIEMAT*) soporta nuevos algoritmos para conseguir que el robot conserve la estabilidad mientras camina, así como para la detección de inestabilidades y la planificación de modos de caminar.

Figura 2.3.: Robots cuadrúpedos

3.1.1.4. ROBOTS HEXAPODOS

En la Universidad Estatal de Ohio ha sido desarrollado un robot con 6 patas que permite al operador tomar decisiones estratégicas, independientemente del posicionado particular de cada extremidad.
Está basado en un sistema de control consistente en 13 computadores de a bordo Intel 86/30.
En el Instituto de Robótica de la Carnegie-Mellon (EEUU) ha sido desarrollado el *(Autonomous mobile expliratory robot),* un robot hexápodo dirigido por programas específicamente diseñados para optimizar el movimiento de avance sobre terrenos abruptos.

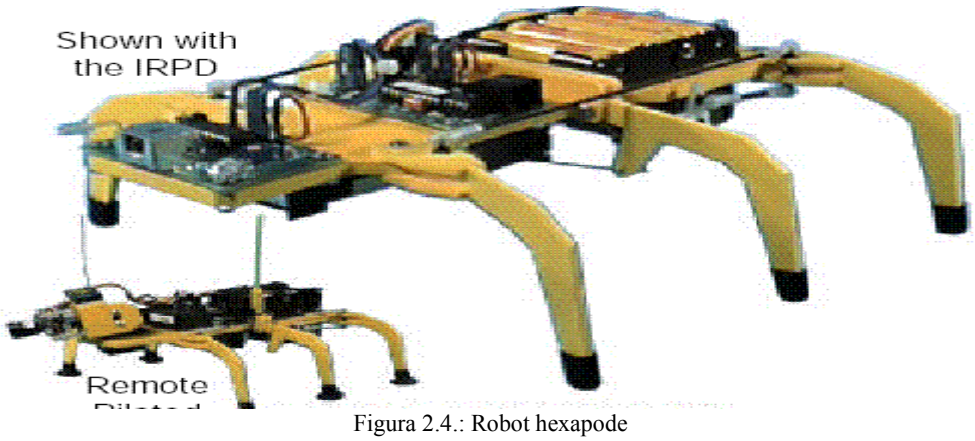


Figura 2.4.: Robot hexapode

3.1.2. Tipo de control

Una última clasificación distinguiría entre robots con control *jerárquico* y robots con control *reactivo*.
Los primeros implican un tiempo de proceso mayor, pues se basan en un sistema fuertemente jerarquizado en el que los niveles de control más altos generan trayectorias que son expandidas y planificadas por los niveles intermedios, y ejecutadas por los niveles inferiores.
Los robots con control *reactivo,* tienen implantados comportamientos más o menos simples que exhiben reaccionando a la influencia del medio.
Carecen de una planificación detallada a priori de sus trayectorias. Esto les da más flexibilidad y les evita tiempo de cálculo, pero hace que no puedan encontrar solución para algunas órdenes de alto nivel.

Ésta última es una tendencia seguida por los conocidos *MOBOTs* del MIT: la distribución del sistema total en pequeños autómatas capaces de funciones sencillas de control, interconectados entre sí, y que en su conjunto permiten funciones mucho más complejas e incluso similares a las de algunos insectos.

En la categoría de robots *jerárquicos,* entraría RHEX: su sistema de control software consiste en un único programa dividido en módulos de distinto nivel de abstracción que gobierna todas las funciones del robot, desde las más básicas hasta las puramente estratégicas.

A continuación se incluye un cuadro resumen de la clasificación de los robots móviles:

<table>
<tr><th colspan="5">Classificación Robots</th></tr>
<tr><td colspan="4">Experimentales</td><td>Aplicados</td></tr>
<tr><td colspan="4">- Robot Hexapodo
- Enfoque experimental</td><td>- Aplicación en la industria y exploración</td></tr>
<tr><td colspan="3">Mòvils</td><td>Estàticos</td><td rowspan="3"></td></tr>
<tr><td colspan="3">- Robot Hexàpodo</td><td>- Industriales
- Incapaces de moverse en entorno no limitado</td></tr>
<tr><td>Autonomos</td><td>No autonomos</td><td>Con patas</td><td>Con otro tipo de locomoción</td></tr>
<tr><td>Soft i hard sobre estructura mecànica</td><td>Governdo por ordenador externo
Alimentación externa</td><td>Desplazameintos más eficients
Control estabilidad compleja</td><td>Posicionamento sencillo
Calculos más simples</td><td colspan="2"></td></tr>
</table>

4. Diseño preliminar

En este capítulo se detalla el diseño que se estudió para llevar a cabo la construcción del ROBOT HEXAPODO. En el mismo se exponen los problemas encontrados en la elección de la configuración mecánica y electrónica del robot, y las soluciones a las que dieron lugar.

4.1. Construcción y distribución de las patas

Existen varias configuraciones posibles para un robot con patas:

- Aquellas que contemplan menos de cuatro patas presentan problemas más propios de la Ingeniería de Control que de la Computación.

Dentro de los robots de más de tres patas, los que nos interesan adoptan la forma de cuadrúpedos o hexápodos.

- Los robots cuadrúpedos y hexápodos permiten un mayor desarrollo de los aspectos de planificación del movimiento.

En ellos los problemas derivados de la estabilidad están relativamente simplificados o son susceptibles de simplificación, con lo que el diseño del hardware de control se hace más sencillo. Esto añade cierta complejidad al software, pero esa es la orientación que se pretendía dar al proyecto.

- No hay ninguna ventaja en usar una configuración pentagonal (en todo caso puede presentar inconvenientes debido a la más compleja simetría).

- Robots de más de seis patas sólo son útiles en campos mucho más específicos, además de presentar un mayor coste.

Un concepto importante a la hora de elegir el número de patas del robot es el de *estabilidad estática*.
Este término se refiere a la capacidad del robot para permanecer estable (sin caerse) cuando no está en movimiento.
Es más fácil mantener la estabilidad estática en un robot hexápodo que en uno cuadrúpedo por un motivo muy sencillo: hay más patas libres para reposicionar el cuerpo del robot mientras éste se apoya en tres de ellas.
En un robot cuadrúpedo, sin embargo, estamos obligados a utilizar un algoritmo de avance en el que las patas vayan alternándose, puesto que sólo una está disponible una vez apoyado el robot en las otras tres.

Por todos estos motivos, se escogió la configuración hexápoda.

Dentro de esta configuración, surgieron de forma natural dos distribuciones de las patas alrededor del cuerpo del robot.

La primera de ellas, la denominada *bilateral* se puede observar en la figura 4.1.a.
Esta distribución presenta una simetría a lo largo del eje longitudinal del robot.
Tiene la ventaja de una mayor simplicidad a la hora de programar los movimientos, debido a que el robot está preparado físicamente para facilitar el avance en direcciones paralelas al eje de simetría.

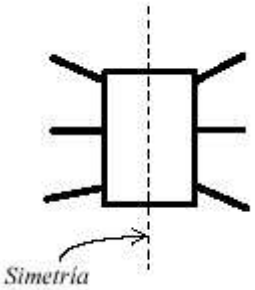


Figura 4.1.a.: Configuración bilateral de las patas de un robot hexapodo

El inconveniente fundamental es que el sistema ofrece una mayor limitación en el movimiento en otras direcciones.
Esto se puede obviar implantando un nuevo movimiento, el de *giro*, para cambiar de orientación explícitamente, pero ello complica más el software.

La distribución *radial* que aparece en la figura 4.1.b. fue la finalmente escogida para el ROBOT HEXAPODO.
No sufre el problema comentado, puesto que cualquier dirección es exactamente igual a cualquier otra.
En dos dimensiones, esta distribución es totalmente simétrica, y eso lleva a un software más genérico y más simple.

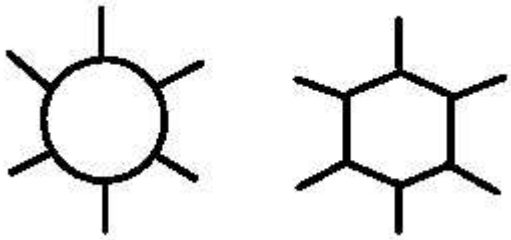


Figura 4.1.b.: Configuración radial para las patas de un robot hexápodo. La figura de la derecha es la configuración escogida finalmente para el ROBOT HEXAPODO.

No necesita movimientos específicos de *giro* puesto que es capaz de desplazarse en cualquier dirección sin tener previamente que cambiar de orientación (el robot es omnidireccional), aunque este tipo de movimiento puede implantarse para casos específicos si es necesario.

4.2. Estructura de las patas

Condicionadas por la elección de configuración hexápoda y distribución radial, las patas debían contar con la mayor movilidad y accesibilidad dentro de sus espacios de trabajo.
Desde el primer momento se pensó en utilizar servos para las articulaciones, lo cual llevó a estudiar diseños que poseyeran únicamente articulaciones de rotación.
En la figura 4.3 aparece el esquema de una pata con dos grados de libertad.

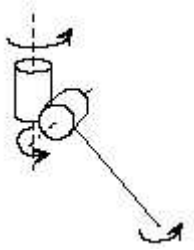


Figura 4.3: Una pata con dos grados de libertad que presenta problemas de deslizamiento

Esta estructura permite posicionar el extremo de la pata en cualquier punto de la superficie de una esfera cuyo centro está en el interior de la primera articulación, pero provoca deslizamientos indeseables.
Para posicionar completamente el extremo de una pata hacen falta seis grados de libertad: tres para especificar la *posición* y tres para especificar la *orientación*.
Sin embargo, puesto que el extremo de la pata se considerará puntual, no hará falta especificar su orientación. Por consiguiente, son necesarios solamente tres grados de libertad.

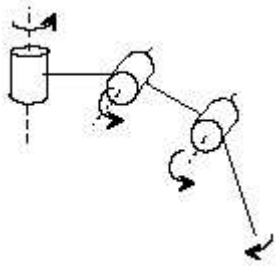


Figura 4.4.: Esquema de los tres grados de libertad de una pata de RHEX.

El esquema de una pata con esta característica aparece en la figura 4.4.

Estos tres grados de libertad aparecen en los dos tipos de estructuras mostradas en la figura 4.5 (además de otras como la de *pantógrafo*, no considerada aquí).

Figura 4.5.:Dos posibles estructuras para las patas de un robot.
(a) paralelogramo deformable, y (b) actuación in-situ

De ellas se escogió la estructura de *actuación in-situ*, en la cual los motores se sitúan directamente en las articulaciones. Esto permite una mayor simplicidad en la función que relaciona posición del servo con posición de la articulación (de hecho ésta es inmediata salvo algún factor de offset en ciertas articulaciones). Presenta sin embargo el inconveniente de un mayor esfuerzo de la estructura, aunque éste es perfectamente soportable con los materiales escogidos.

El otro tipo (*paralelogramo deformable*) consiste en un paralelogramo cuyos ángulos pueden variar mediante servos situados lejos de la articulación.
A pesar de evitar un peso importante en la misma situando el centro de gravedad de la pata más cercano al robot, este diseño acentúa las posibles holguras que con toda seguridad aparecerán en la estructura.
Además genera una mayor complejidad en la función anteriormente indicada.

5. Diseño mecánico.

En este capítulo se detallan los componentes mecánicos que dan lugar a la estructura del robot. Éstos son tres fundamentales: los actuadores, las patas y el cuerpo.

Los actuadores implementan las articulaciones. Son servomecanismos de rotación que permiten el posicionado de cada articulación de forma absoluta y evitan los errores cometidos (por esfuerzos o falta de tiempo) mediante un bucle interno de control.
Las patas se han de diseñar para soportar tres de estos actuadores en el menor espacio posible (ya que a mayor tamaño las holguras propias de estos servos se pueden amplificar, aparte del mayor peso) y permitir que la fuerza ejercida por éstos se propagara de la forma más eficiente hacia el extremo.
El cuerpo, finalmente, debe ser construido pensando en minimizar las colisiones entre patas sin evitar por completo el solapamiento de las zonas alcanzables por cada una.

Seguidamente se explican con más detalle estos componentes estructurales.

5.1. Los actuadores

Deben ser escogidos para mover las articulaciones del ROBOT HEXAPODO servos de radiocontrol. Éstos son muy apropiados por lo siguiente:
- Al contrario que los motores paso a paso, los servomecanismos son reguladores que fuerzan una posición. Así, si se les desvía de ella, tienden siempre a volver, minimizando el error. El control lo hacen típicamente proporcional, absorbiendo más corriente de la fuente de alimentación cuanta más distancia los separa de la posición deseada.
- La relación fuerza ejercida/consumo es relativamente alta.
- Los hay de diversos tamaños y pesos.
- Las órdenes que reciben siguen un formato PWM estándar, fácilmente generable.
- Son de relativamente bajo coste.
- Permiten la lectura de la posición a través de sus potenciómetros.

5.2. Las patas.

El dibujo simplificado de una pata aparece en la figura 5.1. Las tres piezas metálicas que la componen se han de diseñar ajustando sus dimensiones para conseguir el mejor compromiso entre dos objetivos contrapuestos:

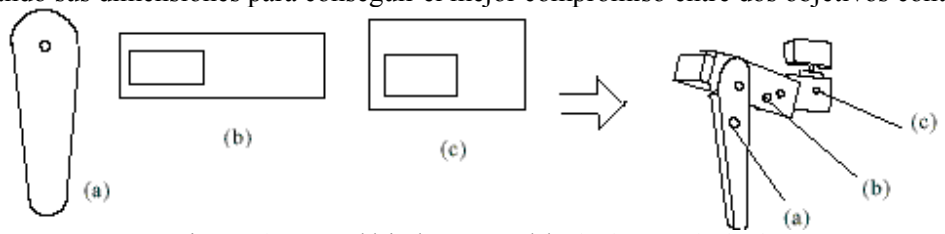


Figura 5.1. : Ensamblaje de una pata del ROBOT HEXAPODO

- Espacio de trabajo lo más amplio posible.

- Holguras mínimas y estructura compacta.

En un primer diseño, la pata adoptaba la configuración mostrada en la figura 5.2. Como se puede observar, ésta pertenece a la estructura en paralelo deformable.

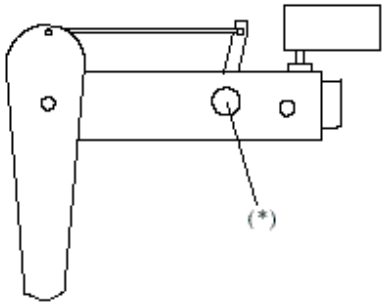


Figura 5.2: Primer diseño de una pata del ROBOT HEXAPODO.

Este tipo de estructura presenta ciertas características que la hacen inadecuada. Las holguras pueden ser amplificadas hasta grados inaceptables. Así mismo, el control se puede ver dificultado por la necesidad de mapear las posiciones deseadas a las posiciones reales del servo marcado en el esquema. Éste debería ser el encargado de mover la tercera articulación. La pata debía tener unas dimensiones mayores que en el diseño definitivo, con lo que el rango de alcanzabilidad seria aumentado, a costa de mayores imprecisiones.

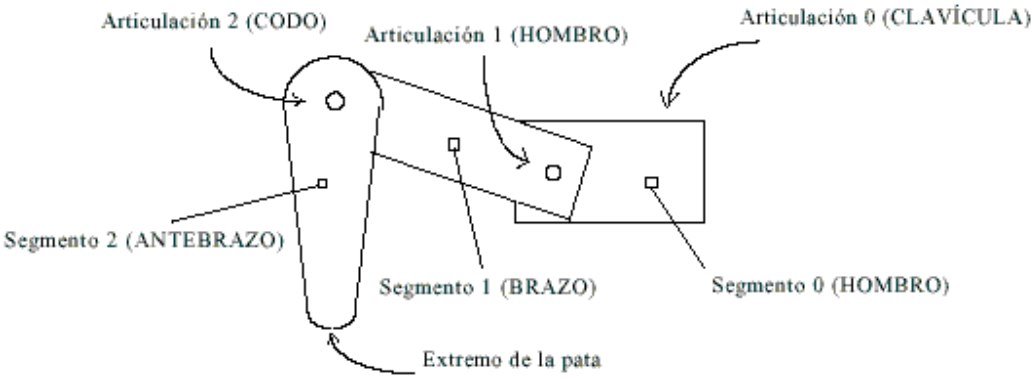


Figura 5.3: Etiquetado de elementos en una pata.

En la estructura escogida finalmente, las articulaciones y los "segmentos" o piezas de la pata se etiquetan y numeran tal y como indica la figura 5.3.

5.3. El cuerpo

El diseño del cuerpo se debe hacer teniendo en cuenta el diseño previamente realizado de las patas. Basándose en la distribución radial, se ha optimizado en varios sentidos:

1º) Sus dimensiones no deben ser demasiado grandes, para evitar futuras diferencias entre la realidad y el modelo cinemático del robot debidas a holguras e imprecisiones. Tampoco deben ser tan pequeñas como para permitir un gran solapamiento en los espacios de trabajo de cada pata, aunque una cierta compartición de espacio sí es deseable.

2º) Se ha preparado para contemplar la posibilidad de un estado especial del ROBOT HEXAPODO en el que éste tenga todas sus patas recogidas.

El resultado es un hexágono con "muescas" donde las patas pueden recogerse, ahorrando gran cantidad de espacio. En el mismo hay lugar para alojar la tarjeta controladora.
Un esquema del cuerpo puede verse en la figura 5.4.

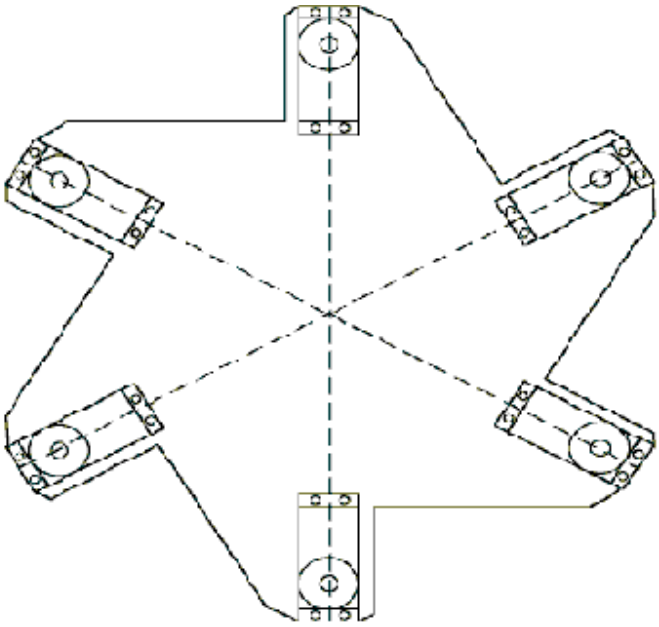


Figura 5.4.: Planta del cuerpo de RHEX, fabricado de una sola pieza, mostrando los servos de las clavículas.

6. Electrónica de control.

En este capítulo se describe el aspecto electrónico del control del ROBOT HEXAPODO.
Los circuitos deben ser capaces de generar y mantener las ondas que los servos aceptan como señales de posicionamiento.

Estas ondas, llamadas PWM, codifican un dato numérico que indica al servo la posición absoluta a la que debe desplazarse y en la que debe permanecer mientras la onda no cambie.
Se escogió el integrado i8253 para generar estas ondas, debido a su versatilidad y facilidad de programación.
Finalmente se debe diseñar la llamada *tarjeta controladora*, que incluye toda la lógica de selección de servos y generación de ondas PWM.

También se detallan en este capítulo los problemas encontrados con la alimentación de servos y controladora, y cómo se deberían solucionar.

Por último se presenta un estudio con en el que se enumeran las características de dos tarjetas de entrada/salida.
Finalmente se escogió la tarjeta PC-LabCard PCL-812PG por las razones que posteriormente se indican.

6.1. La tarjeta controladora

La controladora, situada sobre el ROBOT HEXAPODO, recibe órdenes de posicionado individual de servos que transforma en ondas PWM gracias a los integrados i8253. Se basa en la señal de reloj proveniente del ordenador de 2MHz.
Dentro de este subcapítulo se explican los fundamentos de las ondas PWM, la lógica de la tarjeta controladora y los distintos bucles de control electrónico implantados en ésta.

6.1.1. Generación de ondas PWM.

La señal PWM de posicionamiento de un servo es generable por un integrado i8253. Este dispositivo programable puede dar como salida hasta 3 ondas independientes de muy diversos tipos.
Para generar una onda PWM hará falta utilizar sus tres contadores conectados en cascada. En la figura 6.1 se aprecia esta conexión.

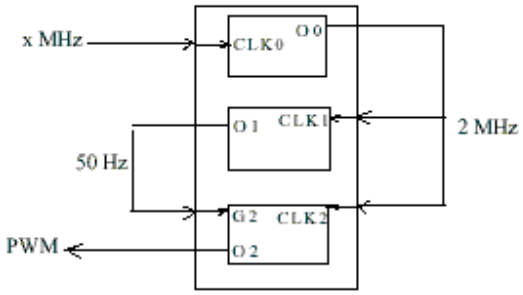


Figura 6.1: Generación de una onda PWM.

- El primer contador (contador 0) se encarga de convertir la onda cuadrada del reloj externo (x MHz) a 2 MHz, que será la frecuencia base por la que se regirán los otros dos contadores. Para ello será programado en modo 3 (*Square Wave Mode*) o *modo divisor de frecuencia*.

- El segundo contador (contador 1) será el encargado de generar la señal gate del tercero. Esta señal deberá ser cuadrada, con una frecuencia de 50 Hz. Ésta es la frecuencia de la onda PWM. Para que cumpla esta misión, se programará también en modo 3, con una palabra de cuenta de 2e6/50 = 40000.

- Por último, el tercer contador (contador 2) generará la onda PWM. Se programará en modo 1 (Hardward Retriggerable One-Shot). En este modo, cada vez que se reciba un pulso alto en el gate se recargará la palabra de cuenta y se la irá decrementando en 1 por cada pulso de reloj. Durante este tiempo, la salida estará baja. Al terminar la cuenta, la salida pasará a estado alto hasta volver a recibir un pulso en el gate. De esta forma, cambiando la palabra de cuenta se puede reprogramar el ancho del pulso positivo de la PWM.

El valor que hay que cargar en el tercer contador será el nº de pulsos de longitud 0.5 microsegundos (1/2e6) que permanezca la onda en estado bajo. Es decir, en realidad no se programa la duración del pulso positivo, sino la del negativo. Sin embargo esto no presenta
ningún problema, ya que tras cargar la cuenta, a partir del siguiente flanco ascendente en la salida se obtendrá la onda PWM deseada. Con ello se ahorra un inversor. Este "truco" se puede ver en la figura 6.2.

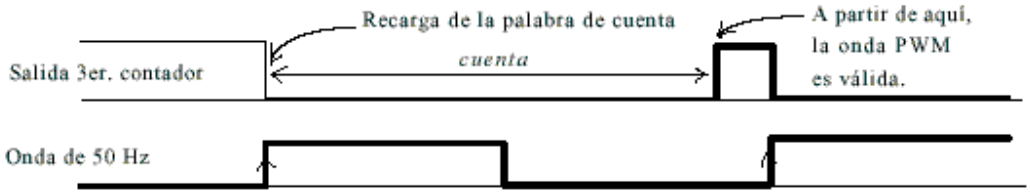


Figura 6.2: La onda PWM generada por nuestro i8253.

Dado que habrá que programar 18 servos (6 patas con 3 servos por pata) harán falta 7 integrados i8253. Uno de ellos generará la señal PWM de 50 Hz, ya que el reloj de 2 MHz se obtendrá directamente desde el PC. Posteriormente se utilizarán sus otros contadores para funciones auxiliares.
Cada uno de los 6 integrados restantes estará asociado a una pata. Y cada uno de sus contadores, a una articulación de esa pata (coincidentes numéricamente, es decir, el contador 0 manejará la clavícula y así sucesivamente).

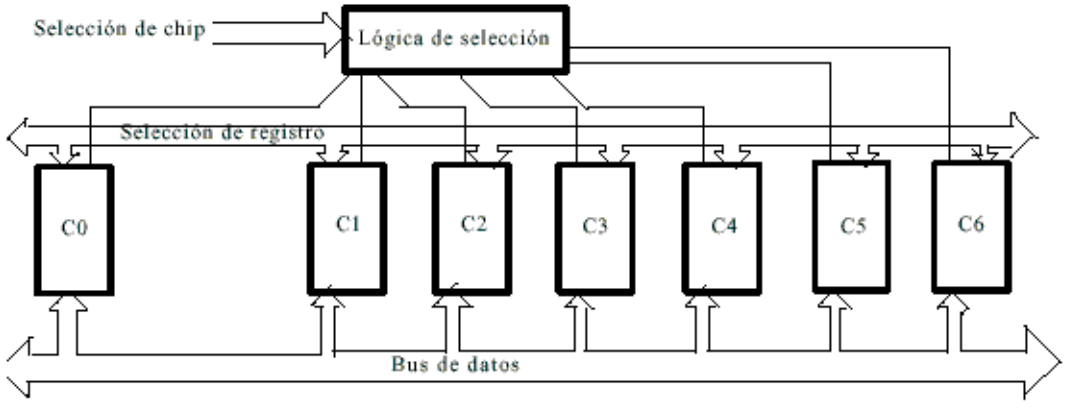


Figura 6.3: Lógica de selección de chips y registros.

Por supuesto, la tarjeta controladora deberá disponer así mismo de una lógica de selección de integrado. Ésta se puede ver esquemáticamente en la figura 6.3.

La lógica de selección recibe un nº binario que le indica el chip a activar. Decodifica este número y activa el **chip select** de ese chip, únicamente. El bus de datos y el bus que lleva el nºde registro al que se quiere acceder de ese chip actúan siempre sobre todos los integrados. El **chip select** inhibido de aquellos no seleccionados impide que las operaciones posteriores afecten a los mismos.

La resolución obtenida en la longitud del ancho positivo de la PWM depende del reloj base de los contadores. Con el que se ha utilizado, de 2 MHz, se pueden enviar pulsos positivos de anchura comprendida entre 0.5 microsegundos (para un valor de la palabra de cuenta de 39999) y 19999.5 microsegundos (para una palabra de cuenta de 1), a incrementos de 0.5 microsegundos (correspondientes a decrementos de 1 unidad en la palabra de cuenta). Las duraciones típicas del pulso positivo de la PWM que admiten los servos utilizados están entre 600 y 2600 microsegundos aproximadamente, valores a los cuales los cálculos anteriores se adecúan perfectamente.

6.1.2 El bucle de control.

El bucle de control está muy simplificado. No se pueden obtener las verdaderas posiciones de los servos, ya que eso supondría una electrónica más compleja y la inclusión de conversores A/D y/o lógica de multiplexión analógica. Ello conlleva un control en bucle abierto.

Dadas las características de los servos en cuanto a corrección automática del error, esto no supone un problema demasiado acuciante, aunque lo tocante al software de control se complica debido a la falta de información real sobre las posiciones de las articulaciones. Esta información se suple con un modelo interno del robot almacenado en la memoria del ordenador que se mantiene sincronizado en todo momento con el robot físico.

A pesar de todo ello, se pretendió suplir esta carencia de realimentación proporcionando formas de recuperar información desde el robot. Así, el ROBOT HEXAPODO dispone de un doble bucle que implanta algunas características adicionales que permiten verificar de forma limitada el cumplimiento de las órdenes enviadas. Esta información es fundamentalmente digital, por lo que no necesita una electrónica demasiado compleja que añadir a la ya comentada.

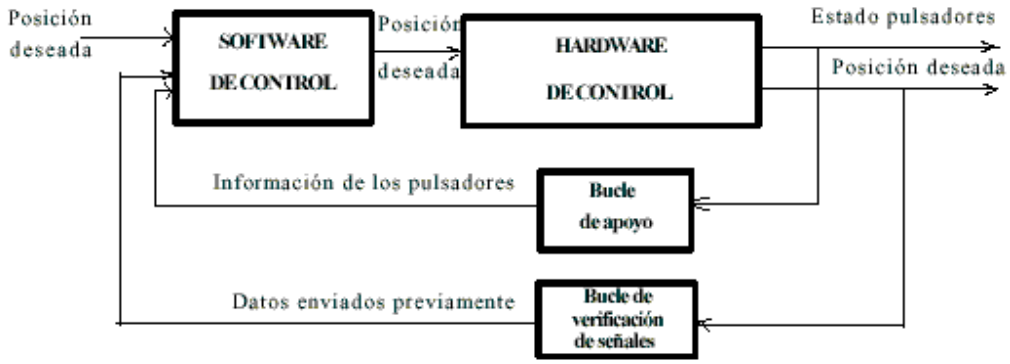


Figura 6.4: Los dos bucles de control implantados en RHEX.

Este doble bucle se descompone en los denominados *bucle de apoyo y bucle de verificación de señales*, esquematizados en la figura 6.4.

El **bucle de apoyo** consiste en la realimentación del estado de los pulsadores situados en los extremos de las patas. Éstos indican al programa de control qué patas están realmente apoyadas en el suelo y cuales no, permitiendo realizar ajustes de las posiciones de las mismas.

En la implantación de este sistema se pueden encontrado varios inconvenientes:

- La baja calidad de los pulsadores existentes en el mercado, unida a los errores cometidos en las dimensiones de las patas durante su fabricación manual hacen que los datos obtenidos no sean suficientemente fiables.

- El problema básico encontrado es que un pulsador no esté activado aún estando la pata apoyada en el suelo (debido a una posición no vertical y/o a su baja sensibilidad). La solución adoptada para evitarlo puede ser seguir bajando la pata hasta que el pulsador diera señal

Por tanto, este bucle de control solamente se aprovecha en la búsqueda de triángulosde apoyo del robot.
El **bucle de verificación de señales** es de suma utilidad para evitar movimientos descontrolados del robot, debidos principalmente a la fuga o desvirtuación de las señales enviadas a la tarjeta controladora desde el ordenador.

Se basa en la monitorización del contenido de los buses de datos y de selección de registro de la tarjeta controladora. Así, el software es capaz de detectar cuándo un dato enviado no ha sido recibido correctamente y en ese caso, de corregirlo o advertir al operario. Este bucle no es completo, es decir, los datos que vuelven desde la tarjeta controladora hacia el PC para su monitorización no están verificados y pueden llevar errores inducidos.

6.1.3. Las fuentes de alimentación.

Se han utilizado dos fuentes de alimentación lineales para la electrónica externa al ordenador. Una de ellas alimenta los 18 servos. El consumo de éstos es variable dependiendo de los errores encontrados entre sus posiciones ideales y las reales. Es de vital importancia que la fuente utilizada para los servos tenga potencia suficiente para proporcionar este amperaje sin perder el nivel de tensión, ya que con tensiones fuera de los límites necesarios para la alimentación de los servos, el comportamiento de los mismos se hace impredecible.

La otra fuente es necesaria para aislar a la tarjeta controladora de las influencias (picos de corriente) de la alimentación de los servos. Proporciona 5 voltios y 1 amperio, energía más
que suficiente para los 7 i8253 y la lógica de selección.

Debido al cambio de carga a la salida de ambas fuentes, éstas no pueden ser conmutadas, pues las fuentes de éste tipo se apagan cuando la carga varía.

Otra posibilidad no ha entrado en el ámbito de este proyecto: situar las fuentes o baterías sobre el mismo robot. Ello es debido a su influencia en el sobrepeso del robot y a su excesivo tamaño en el caso de las fuentes, añadido a que éstas necesitan de todas formas una conexión a la red eléctrica.

6.2 La tarjeta de E/S.

Se han hecho comparaciones con dos tarjetas de entrada y salida conectadas al PC: la I/O8255 y la PC-LabCard modelo PCL-812PG. Los mejores resultados se han obtenido con la segunda, que es la utilizada finalmente.

En la tabla que se muestra a continuación, se enumeran las ventajas e inconvenientes
de ambas.

TARJETA I/O 8255:

VENTAJAS	INCONVENIENTES
- Bajo coste - Menor tamaño - 48 bits de E/S reconfigurables - Los 3 contadores de un Intel 8253 disponibles	- Reloj dependiente del resto del hardware del ordenador - Mapeado inadecuado de los pines de los conectores - No dispone de conversores A/D, aunque éstos no son indispensables - Distinto comportamiento del reloj de la tarjeta dependiendo del modelo de ordenador en el que se instala

TARJETA PC-LABCARD:

VENTAJAS	INCONVENIENTES
-Mayor facilidad de programación -Disponibilidad de conversores A/D y D/A -Reloj interno basado en un cristal de cuarzo, que proporciona una frecuencia fija y exacta de 2 MHz -Distribución de los pines en los conectores de forma modular	-Mayor tamaño y coste - 16 bits de salida digital y 16 bits de entrada digital no reconfigurables -Sólo un contador de un 8253 libre para el programador

6.3. Comunicación entre tarjetas.

El esquema de los distintos buses de datos que comunican la tarjeta de E/S y la controladora puede observarse en la figura 6.4.

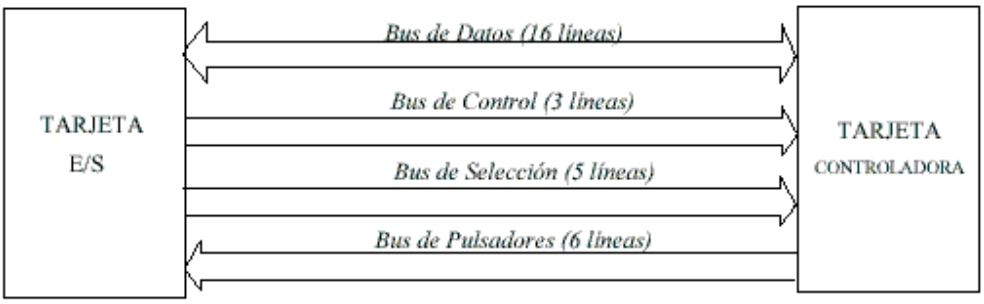


Fig. 6.4.: Buses de comunicación entre el ordenador y el ROBOT HEXAPODO.

Los buses son cuatro:

- BUS DE DATOS (16 líneas): envía datos de 8 bits hacia la controladora para programar los i8253 de la misma, y recibe el último dato de 8 bits enviado.

- BUS DE CONTROL (3 líneas): contiene las señales que controlan el funcionamiento de la controladora. Las líneas son:

> **Escritura (WR):** un pulso bajo en esta señal hace que el último dato enviado a la controladora reprograme un registro de uno de los i8253.
>
> **Reloj (CLK):** cuadrada periódica de 2MHz indispensable para el funcionamiento de los i8253.
>
> **Habilitación del reloj (E_CLK):**esta señal permite eliminar la señal CLK de la entrada de todos los i8253 de la controladora. Es útil para hacer que los servos dejen de ejercer control en situaciones de sobreesfuerzo.

- BUS DE SELECCIÓN (5 líneas): tres de ellas seleccionan el integrado i8253 al que afectará la próxima señal de escritura. Las dos restantes seleccionan un registro dentro del mismo.

- BUS DE PULSADORES (6 líneas): este bus unidireccional envía el estado de los pulsadores (pulsados o no) hacia el ordenador en todo momento. Una señal por cada pata.

7. Estructura del Software.

El software de control se ha distribuido jerárquicamente en diversos módulos. Cada uno de ellos implementa funciones con objetivos coincidentes dentro de su nivel de abstracción. Esta orientación diverge del enfoque *reactivo* con el que se han diseñado muchos de los robots con patas actuales, entre ellos los robots tipo insecto.

Dentro de la jerarquía del software se han distinguido tres niveles de abstracción:

- *Nivel EJECUTOR ,* el más cercano al hardware. Es el encargado de traducir órdenes básicas de posicionado de servos o de los extremos de las patas a señales utilizadas para programar los i8253 de la tarjeta controladora.

- *Nivel PLANIFICADOR DE TRAYECTORIAS,* o nivel intermedio. Planifica las órdenes enviadas desde el nivel más alto y las traduce a comandos básicos del nivel Ejecutor. Implementa pseudoconcurrencia y detecta posibles colisiones sobre un modelo simplificado del robot.

-*Nivel PLANIFICADOR DE MOVIMIENTOS,* el nivel más alto en la pirámide de control. Genera las órdenes adecuadas para que el robot se levante, avance o se recoja, enviándolas a los niveles inferiores para su descomposición en trayectorias de articulaciones individuales y su ejecución.

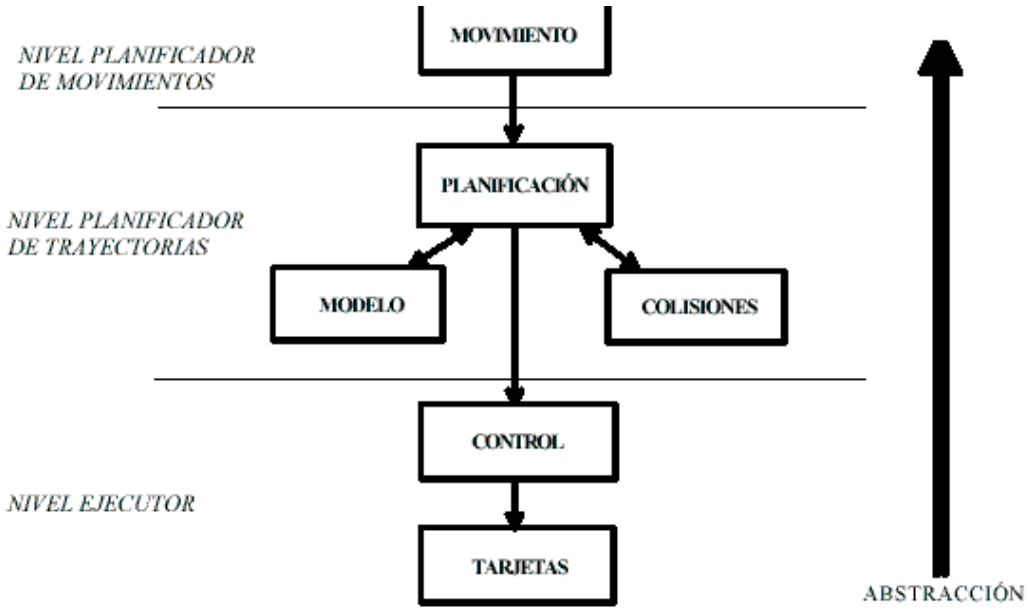


Figura 7.1: Distribución jerárquica del software de control

Esta distribución se puede ver esquemáticamente en la figura 7.1. En la misma se observan los módulos software más importantes que residen en cada nivel. Las flechas entre módulos indican el flujo de información entre los mismos para su proceso.

Seguidamente se explican con detalle estos módulos y otros que no aparecen en la figura anterior.

7.1. Nivel EJECUTOR.

Este nivel contiene los módulos que permiten la traducción de órdenes elementales de posicionamiento de articulaciones a comandos para la programación de los servos de la tarjeta controladora. No se ocupa de la detección de colisiones entre las patas al efectuar estos movimientos. Permite sin embargo la manipulación a bajo nivel de los buses, y operaciones sencillas sobre los datos de los mismos.

Este nivel se compone de dos módulos básicos: el de CONTROL proporciona el verdadero núcleo del nivel Ejecutor. El de TARJETAS es un interfaz que hace transparente al módulo de CONTROL el tipo de tarjeta de E/S que se está utilizando.

7.1.1 Módulo de TARJETAS.

El software está preparado para trabajar con dos tarjetas de E/S comerciales. Este módulo hace transparente a los demás el manejo de cualquiera de ellas. Una vez iniciado indicando qué tarjeta se utilizará, el envío de órdenes se hará con las mismas funciones, independientemente de cuál esté activada. Internamente, el módulo ejecuta las órdenes de una forma u otra según qué tarjeta sea el destino.

Proporciona funciones para el envío de un dato de 8 bits a uno de los integrados i8253 de la tarjeta controladora, para la iniciación del módulo y para la desactivación de la controladora. Ésta última función simplemente elimina el último dato que se envió a la misma y deselecciona todos los integrados.

Además, dispone de una función que devuelve el estado de los 6 pulsadores del robot comprimidos en un octeto.

7.1.2. Módulo de CONTROL.

Éste es el más importante del nivel Ejecutor. Proporciona varios grupos de funciones:

- De iniciación de la tarjeta controladora.

- De calibrado de los servos.

- De mapeado de los integrados i8253.

- De posicionado básico de las articulaciones (articular y cartesiano),incluyendo interpolaciones lineales muy simples.

- De retardo, con resolución de milisegundos, independiente del ordenador en el que se ejecute.

- De lectura de pulsadores, a más alto nivel que la proporcionada por el módulo de TARJETAS.

Seguidamente se explican con más detalle estos grupos.

7.1.2.1 FUNCIONES DE INICIACIÓN

Sólo una función pertenece a este grupo. Es de ejecución obligatoria antes de cualquier operación efectuada en cualquier otro módulo superior a éste. Inicia el reloj de 50 Hz de la tarjeta controladora, carga en memoria los datos sobre el calibrado de los servos, programa inicialmente los i8253 para situar las patas en posición de reposo (recogidas), y por último, mapea los integrados, según su posición por defecto sobre la tarjeta.

7.1.2.2. FUNCIONES RELATIVAS AL CALIBRADO DE LOS SERVOS

Es necesario almacenar en alguna parte la información que permita saber qué ancho de pulso positivo de la PWM es necesario para llevar cada servo a una posición determinada.
Puesto que el control del movimiento se hace básicamente en bucle abierto, se presenta un problema: la generación de esta información.

El problema se ha resuelto teniendo en cuenta una de las conclusiones que se encontraron tras investigar servos individuales de distintas marcas: su posicionamiento es lineal. Es decir, hay una relación lineal entre ancho de pulso positivo de la onda PWM y posición del servo. Así pues, si conseguimos saber qué ancho de pulso es necesario para situar el servo a 0 grados, qué ancho de pulso es necesario para situarlo en el tope contrario y qué ancho es necesario para situarlo a 90 grados, podremos saber qué ancho es necesario para situarlo en cualquier posición, y qué rango de movimiento tiene (en grados).

Las posiciones de 0 y 90 grados se escogieron por su facilidad de comprobación visual, ya que el calibrado (la recogida de estos datos para cada servo) se realiza "a mano". Es decir, una persona es la encargada de ir variando manualmente el ancho del pulso hasta que estime que el servo se ha situado en estas posiciones.

7.1.2.3. FUNCIONES PARA EL MAPEADO DE LOS INTEGRADOS.

La tarjeta controladora se situaría físicamente sobre el cuerpo del robot, intentando minimizar la distancia desde los conectores hasta los servos. A pesar de ello, puede haber problemas con estas conexiones.

Para evitarlos en lo posible, estas funciones permiten asociar a una pata concreta uno de los integrados i8253, de forma que las órdenes enviadas al mismo se refieran a esa pata.
Esto no cambia el cableado de la tarjeta (como es obvio). Solamente hace que al especificar un número de chip para referirse a una pata, éste sea traducido al verdadero número que está asociado a esa pata. Es decir, hay un número de chip lógico (el que se especifica al mapear los chips sobre las patas) y otro físico (el chip 1 siempre estará asociado a la pata 2, por ejemplo).
Una vez realizado un mapeado, todas las referencias al chip lógico serán "desviadas" hacia el correspondiente chip físico.

7.1.2.4. FUNCIONES DE POSICIONEMIENTO BÁSICO DE LAS ARTCULACIONES

Permiten posicionar articulaciones individuales (sin control sobre si la posición es alcanzable o no) o el extremo de una pata (en coordenadas cartesianas, con la misma salvedad). Tampoco esperan a que las articulaciones lleguen efectivamente a sus posiciones de destino. Sin embargo, existen funciones auxiliares de este grupo que calculan el tiempo estimado de llegada, basándose en las posiciones de inicio, final y en la velocidad media de un servo.

Otras funciones de este grupo permiten generar posiciones intermedias entre origen y destino (interpolando linealmente) y ejecutar los movimientos en tres velocidades distintas. Esto es una aproximación bastante burda a las tareas realizadas por el módulo de PLANIFICACIÓN, y son sustituidas por ellas en la mayor parte de los casos.

Además, existe la posibilidad de recuperar las últimas posiciones enviadas a los servos, ya que éstas se guardan internamente.

Y por último, hay funciones auxiliares para acotar una posición dentro de los valores permitidos por la calibración de un servo, así como para calcular una medida positiva de la distancia entre dos posiciones de una pata (que será mayor cuanto más distancia deban recorrer los servos para pasar de una a otra).

7.1.2.5. FUNCIONES PARA LA GENERACIÓN DE RETARDOS.

Sólo hay una función en este grupo, y es muy simple: mediante el reloj interno del ordenador, y por interrupciones, realiza retardos con resolución de milisegundos. Esta función se diseña porque los retardos generables mediante funciones del compilador de C son dependientes del ordenador.

7.1.2.6. FUNCIONES RELACIONADAS CON LOS PULSADORES.

Basadas en la análoga del módulo de TARJETAS, pueden realizar esperas activas hasta que una pata pise el suelo (hasta que su pulsador se active) o hasta que deje de pisarlo.
Así mismo, permiten de forma cómoda saber si un pulsador concreto está activado.

7.2. Nivel PLANIFICADOR DE TRAYECTORIAS.

Este nivel de la jerarquía descompone los movimientos generados por el nivel superior (PLANIFICADOR DE MOVIMIENTOS) en trayectorias individuales asociadas a cada articulación, y las ejecuta en pseudoconcurrencia enviando las órdenes de posicionado oportunas al nivel inferior (EJECUTOR).

Así mismo, dentro de este nivel se sitúan los módulos que implementan el modelo interno del robot, con las funciones que permiten acceder al mismo.

Los módulos más importantes que lo componen son el de COLISIONES (trata las colisiones entre patas del modelo y/o de RHEX) y el de PLANIFICACIÓN (descompone los movimientos en trayectorias articulares).

7.2.1. Módulo de COLISIONES.

En este módulo se encuentran las funciones necesarias para la detección eficiente de colisiones entre las patas del robot

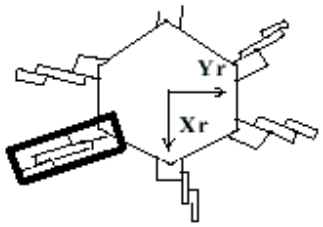


Figura 7.2: Rectángulo límite de la pata 1.

Puesto que los espacios de trabajo de las patas permiten una facilidad de posicionamiento sin colisiones relativamente grande, se puede optar por simplificar el cálculo a costa de una menor precisión.
Las colisiones se detectan basándose en los "rectángulos límites", explicados en el siguiente apartado.

Las funciones de este módulo pertenecen a tres grupos diferenciados:
- De "rectángulos límite".
- De análisis de futuros movimientos.
- De optimización de la detección de colisiones.

7.2.1.1. FUNCIONES RELATIVAS A LOS “RECTANGULOS LIMITE”

El rectángulo límite de una pata está contenido en el plano OXY del sistema R, y es el rectángulo de menor área que contiene en su interior la proyección ortogonal en el plano OXY del sistema R de todos los segmentos de la pata.

Se puede observar un ejemplo de rectángulo límite en la figura 7.2

Como se ve fácilmente, el chequeo de colisiones entre dos patas consecutivas (no existe otro caso posible de colisión entre dos patas dadas las dimensiones del robot) se reduce al cálculo de intersecciones entre dos rectángulos contenidos en el mismo plano. Esto conlleva 4*6 = 24 cálculos de intersección semirrecta-rectángulo, que se pueden simplificar si se usa un sistema de coordenadas en el que uno de los rectángulos límite tenga los lados paralelos a los ejes.
Así, el cálculo de la intersección de las semirrectas que forman los lados del otro rectángulo límite con este rectángulo se reducen en gran medida.

La función perteneciente a este grupo devuelve las coordenadas del rectángulo límite de una pata.

7.2.1.2. FUNCIONES PARA EL ANALISI DE MOVIMIENTOS

Hay una única función en este grupo. Se encarga de devolver información sobre las colisiones que ocurrirían si una articulación de una pata efectuara cierto movimiento. Uno de sus parámetros le indica si la detección de colisiones se realiza sobre el modelo o sobre el robot.

Esta función calcula las intersecciones entre los rectángulos límite de la pata que efectúa el movimiento y los de las dos patas adyacentes. Informa a su regreso de con qué patas de estas dos colisionaría (y si llegaría a hacerlo), además de indicar si el movimiento provocaría que la articulación que lo va a efectuar se saliera del rango posible de movimiento, contemplado en los datos de calibración del servo. Toda esta información ocupa una tripleta de enteros: dos para indicar con qué patas colisionaría y otro que indica si la posición está fuera de rango.

7.2.1.3. FUNCIONES PARA LA OPTIMIZACIÓN DE LOS CALCULOS

Debido a que incluso la simplificación que supone el uso de rectángulos límite no es capaz de evitar la relativamente grande cantidad de cálculos de intersección (además supone la aplicación de una matriz homogénea para rotar los rectángulos hasta ser paralelos a un sistema de referencias), se ha previsto la posibilidad de optimizar algunas partes del cálculo.
Esto se puede llevar a cabo siempre que haya memoria disponible para ello, puesto que la optimización consiste en almacenar en memoria los resultados de ciertos cálculos, previamente generados. Concretamente, se optimiza el cálculo de la dimensión longitudinal del rectángulo límite (el más largo de los dos lados, el único que es variable: depende de los valores que tengan los ángulos de las articulaciones de hombro y codo de la pata). Para cada pareja de valores de articulación (hombro,codo) se almacena en una gran tabla la longitud del rectángulo límite.

7.2.2. Módulo de PLANIFICACIÓN

Este es, con diferencia, el módulo más complejo y potente de todo el software del ROBOT HEXAPODO. En esencia, realiza una planificación de varias trayectorias (secuencias de posiciones
asociadas a una articulación de una pata) simulando concurrencia, con posibilidad de sincronizarlas en distintos instantes de tiempo.
El módulo está jerarquizado internamente.

La primera capa, más básica, permite:

- Diseñar y ejecutar en pseudoconcurrencia planes (conjuntos de varias trayectorias)
- Sincronizar las trayectorias de un plan, pero sólo en su finalización. Es decir, hacer que finalicen al mismo tiempo.
- Realizar interpolaciones lineales entre dos posiciones cartesianas del extremo de una pata.
- Especificar los puntos de las trayectorias en coordenadas articulares o cartesianas (en este caso, del extremo de una pata).
- Especificar las trayectorias mediante un pseudolenguaje muy simple en el que cada punto de una de ellas va acompañado de un código que le dice al programa si éste es cartesiano, articular, etc.
- Consultar el estado de cada trayectoria una vez finalizado el plan (información sobre colisiones y ajustes de posiciones fuera de rango).

La segunda capa es una extensión de la anterior. Añade las siguientes características:
- Manejo de *superplanes*. Éstos son una ampliación de los planes del nivel anterior, que incorporan:
    - Sincronización en cualquier punto de una trayectoria.

    - Especificación de posiciones relativas, tanto en coordenadas articulares como cartesianas.
    - Realización de interpolaciones entre posiciones articulares o cartesianas.

- Incorporación de un pseudolenguaje más potente para la especificación de trayectorias, que contempla las nuevas posibilidades de los superplanes.
- Información posterior a la ejecución del superplan mucho más completa y específica que la proporcionada en la capa básica.

Las funciones que componen este módulo son muchas y muy diversas, por lo que a continuación se explican los conceptos fundamentales en que se basan y no cada función en particular.

7.2.2.1. PLANES

Un plan consta de 18 trayectorias, una por cada articulación de cada pata, que pueden estar definidas o no. A la hora de ejecutarlo, debe haber al menos una definida.
Hay varias funciones que permiten la definición completa de una trayectoria. Se puede crear una trayectoria mediante la llamada a funciones que añaden puntos, entre los cuales puede haber una interpolación lineal o no. Esta forma supone la adición de puntos uno a uno.
O bien puede crearse directamente una trayectoria mediante un pseudolenguaje básico. Este lenguaje no incorpora palabras clave. Es simplemente una lista de puntos.

Estos puntos son conjuntos de 4 valores: los 3 primeros especifican unas coordenadas.
El cuarto, si es 0, indica que las coordenadas son articulares. Si es mayor que 0, por ejemplo Q, indica que las coordenadas son cartesianas y se refieren al extremo de la pata. Además, entre el punto anterior de la trayectoria y ese punto, habrá que realizar una interpolación lineal de Q pasos.
De esta forma, se definen 6 posibles trayectorias, que internamente se expanden a las 18 (ya que los puntos de la lista contienen los 3 valores de las articulaciones de la pata).

En la ejecución de un plan, puede especificarse que se sincronice la finalización de todas las trayectorias. También se especifica en dónde se ejecutará el plan: en el modelo, en el robot o en ambos. Además, se establece el nivel de detección de colisiones.
Éste puede ser de 4 tipos:

Nivel 0: se ignorarán las colisiones entre patas y las posiciones fuera de rango. La ejecución del plan continuará a pesar de las mismas. Hay que destacar la diferencia existente entre *posiciones fuera de rango y posiciones inalcanzables*. Las primeras son acotadas al tope más cercano del servo. Las segundas sí provocarán error, ya que suponen situar la pata en una posición imposible.

Nivel 1: si hay alguna colisión, detiene la trayectoria asociada a la articulación que la ha provocado. Las demás trayectorias continúan en ejecución, y los ajustes de posiciones fuera de rango siguen siendo ignorados (acotados).

Nivel 2: sigue ignorando las posiciones fuera de rango, pero cualquier colisión provocará que el plan se detenga por completo.

Nivel 3: el más restrictivo. Cualquier colisión o acotación de posición hará que el plan se detenga.

Tras la detención (con éxito o no) de un plan, se puede consultar el estado en que se quedó cualquiera de las trayectorias. La información proporcionada es idéntica a la de la función de análisis de movimientos.
De este modo pueden averiguarse las causas que provocaron la detención.
Otra facilidad de que se dispone a la hora de ejecutar un plan es la especificación de una función de tratamiento del teclado. Cada vez que se pulse una tecla durante la ejecución del plan, será llamada esta función. Lo que ésta

realice no influirá en la reanudación de la ejecución, salvo en que se le permite indicarle al planificador que se debe terminar la misma.
Esto es útil si queremos disponer, por ejemplo, de una tecla que aborte inmediatamente la ejecución de un plan.

7.2.2.2. PSEUDOCONCURRENCIA

La ejecución en paralelo de las distintas trayectorias de un plan se realiza mediante un algoritmo. Previamente a la ejecución del plan, todos los puntos de las trayectorias han sido expandidos y/o traducidos a coordenadas articulares.
El algoritmo es como sigue:

1.- Enviar a los servos (o al modelo, o a ambos) todas las posiciones de las trayectorias que no estén en espera. Al comienzo del plan, ninguna trayectoria está en espera, con lo que se lanzarán al mismo tiempo todas las posiciones iniciales. Si previamente al envío de alguna posición a una articulación se detecta que habrá una colisión o acotación, se actúa en consecuencia con el nivel establecido de tratamiento de colisiones o ajustes.

2.-Tras enviar una posición a un servo, esa trayectoria se queda en estado de espera, y almacena el tiempo estimado que tardará el servo en situarse en la posición especificada.

3.-Se busca el menor tiempo de espera que se haya calculado.

4.-Se resta ese tiempo de todos los tiempos de espera de todas las trayectorias en espera, quedando aquellas que tuvieran ese tiempo mínimo preparadas para seguir ejecutándose.

5.-Se llama a la función de tratamiento del teclado si se detecta alguna tecla pulsada.

6.-Se vuelve a 1 a menos que todas las trayectorias hayan terminado o se haya recibido orden de terminar la ejecución desde la función de tratamiento del teclado.

7.2.2.3. SUPERPLANES

Dotan al módulo de planificación de una mayor potencia. Básicamente son la ejecución consecutiva en el tiempo de varios planes. Los puntos de unión de éstos constituyen los puntos de sincronización del superplan.

7.3 Nivel PLANIFICADOR DE MOVIMIENTOS

Este nivel contiene los módulos necesarios para implantar el modo de andar (*gait* en inglés) del ROBOT HEXAPODO, así como el resto de movimientos de alto nivel (levantamiento, reposo, cambio de altura). Todo ello lo ofrece el módulo de MOVIMIENTO, descrito a continuación.

7.3.1. Funciones relativas a los triangulos de apoyo

Un triángulo de apoyo está formado por los extremos de tres patas del robot (o del modelo) que lo mantienen estable sin necesidad de apoyar las otras patas. Puede haber varios triángulos de apoyo distintos para una misma

posición global del ROBOT HEXAPODO. El que una pata esté apoyada en el suelo se chequea verificando su pulsador.

Existe una función que chequea si tres patas dadas forman triángulo de apoyo. Esto se hace comprobando si la proyección vertical (siguiendo el eje OZ del sistema R) del centro de masas del robot sobre el plano que forma el triángulo está en el interior del mismo.
El centro de masas se considera coincidente con el origen del sistema R, pero la función que devuelve sus coordenadas puede modificarse en diseños futuros para contemplar un cálculo más preciso de éste (dependiente de los pesos de las patas y su distribución en cada momento).

7.3.1.1. FUNCIONES PARA EL MOVIMIENTO DE ALTO NIVEL

Como ya se ha explicado, existen cuatro movimientos de alto nivel:

Levantamiento: partiendo de una postura recogida (todas las articulaciones con valores de reposo, idénticos a los que tienen justo después de iniciar los módulos de CONTROL y/o de MODELO SIMPLIFICADO), se pretende que el robot se levante verticalmente hasta quedar a la distancia del suelo especificada por el operario. Esto es un movimiento que se realiza con las 6 patas al mismo tiempo (para minimizar el esfuerzo y de ahí el consumo de los servos). Éstas empiezan situándose en sus respectivos puntos de sobreapoyo. Una vez todas estén ahí, comienzan a descender siguiendo la vertical de apoyo (en una trayectoria cartesiana relativa e interpolada) hasta llegar a los puntos de apoyo.

Reposo: partiendo del robot en pie, las patas comienzan a subir de nuevo por la vertical de apoyo hasta llegar a sus puntos de sobreapoyo. De allí, se recogen hasta sus posiciones de reposo a la vez, sin interpolar. Si al principio del movimiento de reposo el robot no estaba en pie, se hace una llamada a la función de levantamiento.

Avance: se desarrolla en un bucle. Antes y después de cada paso, el robot deberá encontrarse en pie. Durante cada paso, se hará que avance cierta distancia en la dirección especificada. El bucle se repetirá hasta agotar la distancia que se le ordenó avanzar o hasta que haya algún error, lo que antes suceda. El algoritmo que sigue en su avance es el de *tripode alternante,* que se muestra a continuación a grandes rasgos:

1.-Calcular la mínima distancia que se pueden desplazar las patas en la dirección de avance. Si ésta es mayor que la distancia que el robot debe avanzar, escoger ésta última como mínima distancia.

2.-Levantar las patas pares y situarlas a esa distancia. Mientras, el robot se apoya en las patas impares.

3.-Desplazar todas las patas en la dirección contraria esa distancia. Como todas se encuentran en el suelo, conseguirán desplazar el cuerpo del robot en la dirección de avance, puesto que empujarán contra el suelo.

4.-Levantar las patas impares y situarlas de forma que el robot de nuevamente en pie. Las patas pares no deben realizar este movimiento puesto que tras los pasos 2 y 3 ya están en esa posición.

5.-Sumar a la distancia total recorrida, esa mínima distancia.

6.-Si aún no se ha recorrido toda la distancia especificada inicialmente, volver al primer paso del algoritmo.

APENDICE A

Servomecanismos de control

En este apéndice se muestra una descripción completa de los servos utilizados en el diseño del ROBOT HEXAPODO.

A.1. Aspectos mecánicos.

Existen multitud de modelos de servos para radiocontrol. Las dimensiones de los escogidos se pueden ver en la figura A.1.
Estos servos ejercen una fuerza de 3'0 Kg/cm y tardan 0'22 seg. en moverse 60º.

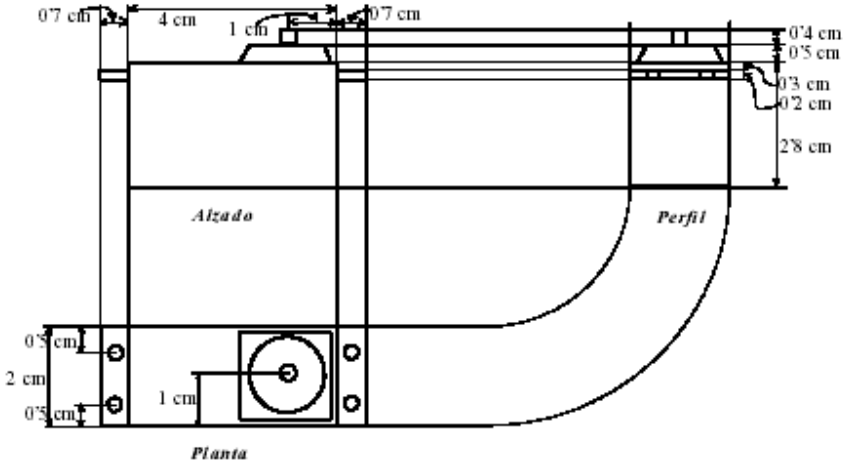


Figura A.1: Dimensiones de los servos del ROBOT HEXAPODO.

A.2. Aspectos electrónicos

Los servos trabajan con un rango de alimentación relativamente amplio (desde unos 5 voltios hasta 8 voltios, aproximadamente). A mayor voltaje de alimentación, mayor velocidad y fuerza en la respuesta.
El posicionamiento se realiza enviando al servo una onda digital con formato PWM (*Pulse Width Modulation*).

Este formato es una variante de una onda cuadrada que se diferencia de ésta en que la duración del pulso positivo (o negativo) es variable, manteniendo siempre la misma frecuencia. Esta duración codifica el dato que porta la onda. En el caso de los servos, la duración es proporcional a la posición absoluta que deberá tomar el eje.

La resolución obtenida en la longitud del ancho positivo de la PWM depende del reloj base de los contadores. Con el que se ha utilizado, de 2 MHz, se pueden enviar pulsos positivos de anchura comprendida entre 0.5 microsegundos (para un valor de la palabra de cuenta de 39999) y 19999.5 microsegundos (para una palabra de

cuenta de 1), a incrementos de 0.5 microsegundos (correspondientes a decrementos de 1 unidad en la palabra de cuenta).

No es recomendable utilizar valores de la palabra de cuenta superiores a 39999 (ni 0, que es equivalente a 65536), ya que hacen que la duración del pulso negativo alcance o supere los 20 milisegundos, que es el período básico de la onda PWM de 50 Hz. Dado que la señal gate no afecta al contador mientras éste no termine su cuenta, estos casos "degenerados" provocan pulsos positivos anormales, cuya longitud es de 40000-F*0.5 microsegundos (donde F es la palabra de cuenta, F>=40000).

Esto no es de ninguna utilidad, ya que esta longitud estará comprendida entre 20000 (F=40000) y 7232 (F=0, que en realidad es 65536) microsegundos, valores abarcados por los casos "no degenerados"; y además hacen que la onda PWM cambie su período por otro mayor, de 40 milisegundos (25 Hz).
Las duraciones típicas del pulso positivo de la PWM que admiten los servos utilizados están entre 600 y 2600 microsegundos aproximadamente, valores a los cuales los cálculos anteriores se adecúan perfectamente.

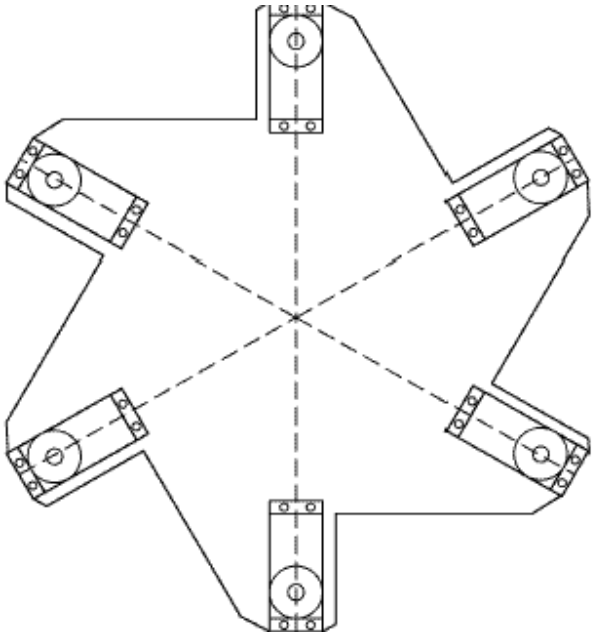
APENDICE B

Estructura mecánica del ROBT HEXAPODO: (Patas)

Las tres piezas que se ensamblan para formar una pata de RHEX se han fabricado en aluminio de 1 mm de espesor. Se han recortado manualmente de ángulos en "L" de este metal para dotarlas de mayor resistencia.

Estructura mecánica del ROBT HEXAPODO: (Cuerpo)

En el esquema inferior se muestra el plano de la planta del cuerpo de RHEX. Éste se hace de una sóla pieza sobre una plancha de aluminio de 1 a 2 mm de espesor, recortando las ventanas apropiadas para insertar los servos de las clavículas.



APENDICE C

Descripción del i8253

El integrado i8253 tiene las siguientes características generales:

- Tres contadores decrementales independientes de 16 bits.
- Entrada de reloj de frecuencia comprendida entre 0 y 2'6 MHz.
- Varios modos de programación de los contadores.
- Cuentas binarias o en BCD.
- Alimentación de 5 v.
- Señales compatibles TTL.

A continuación se ofrece un resumen de la hoja de datos del integrado, en el que se destacan los puntos más interesantes.

Funciones

El i8253 es un elemento multi-temporizador de propósito general que puede ser tratado como una "caja negra" desde el software del sistema.

Algunos de los usos más típicos del i8253 son los siguientes:

- Generador de ondas de frecuencia programable.
- Contador de eventos.
- Divisor de frecuencias.
- Reloj de tiempo real.
- Generador de pulsos.
- Controlador complejo de motores.

Patillas

Bus de datos: son 8 líneas bidireccionales. Pueden estar en tres estados: entrada (escritura o programación del chip), salida (latcheada) y en alta impedancia (chip deshabilitado).

Lectura: Un nivel bajo en esta patilla hará aparecer en el bus el dato contenido en uno de los buffers de uno de los contadores, previamente seleccionado. No se permite leer el registro de control.

Escritura: Un nivel bajo en esta patilla hará que el dato del bus sea escrito en uno de los registros internos (3 buffers para los contadores, cada uno de 16 bits, y el registro de control, de 8 bits).

A0-A1 nª de registro: Especifican (tanto en lectura como escritura) el registro al que se accederá:

A0	A1	Registro
0	**0**	Buffer contador 0
0	**1**	Buffer contador 1
1	**0**	Buffer contador 2
1	**1**	Registro de control

Chip Select: Mientras esta patilla permanezca a nivel bajo, las operaciones de E/S con el i8253 estarán bloqueadas (bus de datos en alta impedancia).

APENDICE D

Descripción de la tarjeta I/S E/O

Esta tarjeta comercial de E/S tiene las siguientes características generales:

- Un integrado i8253 cuyos tres contadores están disponibles para su programación.
- Dos integrados i8255 que proporcionan entre ambos 48 bits de E/S, divididos en grupos (A y B) y registros (A,B y C) para cada integrado.
- Una señal de reloj teórica de 2'386333 MHz, aunque en realidad no es estables en ordenadores PC de gama alta
- Display interno compuesto de 16 leds indicadores del estado de algunas líneas de E/S digitales.

Debido sin embargo a los problemas que se enumeran a continuación, ha sido preferible utilizar la tarjeta PCLabCard PCL-812PG:

- Reloj dependiente del resto del hardware del ordenador.
- Mapeado inadecuado de los pines de los conectores.
- No dispone de conversores A/D, aunque éstos no son indispensables.
- Distinto comportamiento del reloj de la tarjeta dependiendo del modelo de ordenador en el que se instala.

Descripción de la tarjeta PCLabCard

A continuación se ofrece una descripción general de las funciones de esta tarjeta utilizadas por el software de comunicación con el ROBOT HEXAPODO.

Características principales:

- 16 entradas analógicas con conversores A/D de resolución 12 bits.
- 2 salidas analógicas con conversores D/A de resolución 12 bits.
- Rango de las señales analógicas entre -5v y +5v.
- Conversores A/D por aproximaciones sucesivas y D/A por multiplicador monolítico.

Velocidad de conversión A/D máxima de 30 KHz. Tiempo de establecimiento del dato del conversor D/A de 30 mseg.

- Transferencia de datos analógicos por espera activa, interrupción o DMA.
- 16 bits de entrada digital compatible TTL.
- 16 bits de salida digital compatible TTL.
- Un dispositivo i8253 con 1 contador libre para el usuario.
- Reloj interno accesible de 2 MHz.

APENDICE E

Valoración Económica

Componentes	Precio un.	Unidades	P.V.P.
Servo	6,39	18	115,02
Pieza Al (Cuerpo)	14,09	1	14,09
Patas Al (3 piezas)	0,61	6	3,66
Micro i8253	9,86	7	69,02
Tarjeta I/O	69,99	1	69,99
Led	0,15	16	2,4
Cableado	0,35	18	6,3
Tornilleria	0,22	30	6,6
Fuente alimentación	3,589	1	3,589
Fuente alimentación	2,658	1	2,658
Minirru. SS5GL2 OMRON	0,48	6	2,88

TOTAL: 296,207

El precio final del diseño aproximadamente seria de 296.207 euros.

APENDICE F

Ejemplo de programación

Para la programación del robot, se utilizará el lenguaje C.
A continuación se muestra un ejemplo de programación de un robot hexapodo en lenguaje C:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <signal.h>
#include "robi.h"
/* ----------- */
static int opt_r=0;
static int fd=0;
/* ----------- */
/* ----------- */
void help()
{
   printf("pprobi -- Software de control para el robot andante\n\
USO: pprobi [-h] [parport-device]\n\
\n\
OPCIONES:\n\
        -h HELP\n\
        -r puesta a cero de los pins del port paralelo y exit\n\
    \n\
El device es por defecto /dev/parport0 \n\
");
#ifdef VERINFO
   puts(VERINFO);
#endif
exit(0);
}
/* Evento : todo apagado y fin */
void offandexit(int code)
{
   robi_setdata(fd,0);
   set_terminal(0);
   exit(0);
}
/* ----------- */
int main(int argc, char **argv)
{
   int state,bpat,alternate;
   char *dev;
   /*Lo siguiente es utilizado por getopt: */
   int ch;
      extern char *optarg;
      extern int optind;
      extern int opterr;
   opterr = 0;
   while ((ch = (char)getopt(argc, argv, "hr")) != -1) {
      switch (ch) {
      case 'h':
```

```
            help(); /*no hay ruptura, desde help no se vuelve */
        case 'r':
            opt_r=1;
            break;
        case ¿':
            fprintf(stderr, "serialtemp ERROR: Opción no diponible. Ayuda : -h.\n");
            exit(1);
        /*sin acción */
        }
    }
    if (argc-optind < 1){
        /* sin argumentos */
        dev="/dev/parport0";
    }else{
        /* el usuario a asignado un argumento */
        dev=argv[optind];
    }
    fd=robi_claim(dev); /* robi_claim autochequeo de errores */
    /* Se captura las señales INT y TERM y se ponen todas la lineas de datos a cero antes de
     * terminar */
    signal(SIGINT, offandexit);
    signal(SIGTERM, offandexit);

    /* Inicialización de la linea de datos parpprt a cero: */
    robi_setdata(fd,0);
    set_terminal(1); /* set_terminal tiene su propio sistema de detección de errores */
    state=0;
    alternate=0;
    if (opt_r){
        offandexit(1);
    }
        while(1){
        ch=getchoice();
        if (ch!=0) state=ch;
        if (ch == ' '){
            printf("Stop\n");
            robi_setdata(fd,0);
            usleep(500*1000);
        }
        if (ch == 'q'|| ch == 'x'){
            printf("Quit\n");
            break;
        }
        if (state=='l'){
            /*derecha */
            printf("Desplazamiento a la derecha\n");
            walkright(fd);
        }
        if (state=='h'){
            /*left */
            printf("desplazamiento a la izquierda\n");
            walkleft(fd);
        }
        if (state=='j'){
            printf("marcha atras\n");
            walkback(fd);
        }
        if (state=='k'){
            if (alternate){
                printf("desplazamiento hacia a\n");
                walkstraight_a(fd);
            }else{
```

```
                printf("esplazamiento hacia b\n");
                walkstraight_b(fd);
            }
            alternate=(alternate +1) %2;
        }
    }
    /* Si llegado aqui se ha pulsado q */
    set_terminal(0);
    return (0);
}
```

==== robi.c =====

```
/* vim: set sw=8 ts=8 si : */
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <sys/types.h>
#include <sys/time.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <linux/ppdev.h>
#include <sys/ioctl.h>
#include <termios.h>
#include "robi.h"
/* como printf pero se abandona el programa*/
static int die(const char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    vprintf(fmt, ap);
    va_end(ap);
    exit(1);
}
/* Se coge un caracter de stdin
 * Si el caracter leido es diferente de cero, se devuelve no cero
 * Asignación de las flechas:
 * <- = h
 * -> = l
 * v = j
 * ^ = k
 */
int getchoice()
{
    int c;
    char s[20];
    if (fgets(s,20,stdin)){
        c=s[0];
        switch (c){
            case 0x1b: /* ESC */
                if (s[1] == 0x5b){
                    /* si se pulsan las flechas */
                    switch (s[2]){
                        case 0x41: /*flecha arriba*/
                            c='k';
                            break;
                        case 0x42: /*dflecha abajo*/
                            c='j';
                            break;
                        case 0x44: /*tecla l */
```

```
                    c='h';
                    break;
                case 0x43: /*tecla r */
                    c='l';
                    break;
                default:
                    c=0;
            }
        }else{
            c=0;
        }
        break;
    case ' ':
    case 'h':
    case 'j':
    case 'k':
    case 'l':
    case 'q':
    case 'x':
        break;
    default:
        c=0;
  }
  return(c);
 }
 return(0);
}
/* Activa el terminal en modo no canonico
 * o se resetea es terminal.
 * Uso: set_terminal(1) para modo canonico
 */
int set_terminal(int canonical)
{
 static struct termios originalsettings;
 struct termios newsettings;
 static int origok=0; /* sActivo si originalsettings es valido */
 if (canonical){
  /* Guarda los parametros y se activa el modo canonico*/
  tcgetattr(fileno(stdin),&originalsettings);
  newsettings=originalsettings;
  newsettings.c_lflag &= ~ICANON;
  newsettings.c_lflag &= ~ECHO;
  newsettings.c_cc[VMIN]=0; /* do not block */
  newsettings.c_cc[VTIME]=1; /* 100 ms */
  if (tcsetattr(fileno(stdin),TCSANOW,&newsettings) !=0){
   die("ERROR: No se han podido activar los atributos de stdin\n");
  }
  origok=1;
 }else{
  if (origok){
   /* Recuperando los datos programados */
   tcsetattr(fileno(stdin),TCSANOW,&originalsettings);
  }
 }
 return(0);
}
/* abre el dispositivo /dev/parportX y lo captura.
 * Uso: fd=robi_claim("/dev/parport0");
 * El valor devuelto es un descriptor de fichero usado por otras
 * funciones como robi_setdata */
int robi_claim(char *dev)
{
```

```
    int fd,i;
    fd = open(dev, O_RDWR );
        if (fd < 0) {
                die("ERROR: no se puede comunicar con el dispositivo %s\n",dev);
        }
    i=0;
    /* wNecesitamos permiso y no activar las lineas de control */
    /*ioctl(fd, PPEXCL, &i)&&die("ERROR: fallo en la busqueda de permisos\n");*/
    ioctl(fd, PPCLAIM, &i)&&die("ERROR: fallo de comunicación con
parport\n");
    return(fd);
}
/* desplazamiento a la izquierda
 */
int walkleft(int fd)
{
    /* Las patas B al suelo*/
    robi_setdata(fd,LEGBD);
    usleep(400 *1000);
    /* all A legs 1 step */
    robi_setdata(fd, LEGB1 | LEGB3 );
    usleep(1100 *1000);
    /* Patas A al suelo, enfriando B*/
    robi_setdata(fd,LEGAD);
    usleep(400 *1000);
    robi_setdata(fd,0);
    usleep(1000 *1000);
    return(0);
}
/* Desplazamiento a la derecha
 */
int walkright(int fd)
{
    /* patas A al suelo */
    robi_setdata(fd,LEGAD);
    usleep(500 *1000);
    robi_setdata(fd,  LEGA3 | LEGAD);
    usleep(300 *1000);
    /* all A legs 1 step */
    robi_setdata(fd, LEGA1 | LEGA3 );
    usleep(1100 *1000);
    /* patas B al suelo , enfriamiento de A*/
    robi_setdata(fd,LEGBD);
    usleep(400 *1000);
    robi_setdata(fd,0);
    usleep(1000 *1000);
    return(0);
}
/* Dando un paso  atras con 3 patas */
int walkstraight_a(int fd)
{
    /* patas A al suelo*/
    robi_setdata(fd,LEGAD);
    usleep(800 *1000);
    /* all A legs 1 step */
    robi_setdata(fd, LEGA1 | LEGA2 | LEGA3 );
    usleep(1000 *1000);
    /* patas B al suelo, se enfria A*/
    robi_setdata(fd,LEGBD);
    usleep(500 *1000);
    robi_setdata(fd,0);
    usleep(1200 *1000);
```

```
    return(0);
}
/*Desplazamiento hacia delante*/
int walkstraight_b(int fd)
{
    /* patas B al suelo */
    robi_setdata(fd,LEGBD);
    usleep(400 *1000);
    /* topas la patas B una paso */
    robi_setdata(fd,LEGB1 | LEGB2 | LEGB3);
    usleep(1000 *1000);
    /* A down and cool */
    robi_setdata(fd,LEGAD);
    usleep(800 *1000);
    robi_setdata(fd,0);
    usleep(1200 *1000);
    return(0);
}
/* Desplazamiento con todas las patas un paso hacia atras */
int walkback(int fd)
{
    /* primeras patas A hacia delante/
    robi_setdata(fd,LEGAD);
    usleep(800 *1000);
    /* Todas las patas B dan un paso en el aire*/
    robi_setdata(fd, LEGB1 | LEGB2 | LEGB3 );
    usleep(500 *1000);
    /* primeras patas B al suelo, se enfria A*/
    robi_setdata(fd,LEGBD);
    usleep(500 *1000);
    /*todas la patas A un paso en el aire*/
    robi_setdata(fd,LEGA1 | LEGA2 | LEGA3);
    usleep(500 *1000);
    /* A down and cool */
    robi_setdata(fd,LEGAD);
    usleep(800 *1000);
    robi_setdata(fd,0);
    usleep(1000 *1000);
    return(0);
}
/*---------*/
/* Se escribe un patron de bits en el puerto
 * Uso: rc=robi_setdata(fd,bitpat);
 * Valor devuelto 0
int robi_setdata(int fd,unsigned char bitpat)
{
    int rc;
    rc=ioctl(fd, PPWDATA, &bitpat);
    return(rc);
}
```

==== robi.h =====

```
/* vim: set sw=8 ts=8 si et: */
#ifndef H_ROBI
#define H_ROBI 1
#define VERINFO "version 0.2"
/* la primera cosa que necesitas hacer: */
extern int robi_claim(char *dev);
/* Se escribe un patron de bits en el puerto: */
extern int robi_setdata(int fd,unsigned char bitpat);
```

```
/* entrada y funciones del terminal */
extern int set_terminal(int canonical);
extern int getchoice();
extern int walkstraight_a(int fd);
extern int walkstraight_b(int fd);
extern int walkback(int fd);
extern int walkleft(int fd);
extern int walkright(int fd);
/* Asignación de datos a las patas:
 * A1------=------B1
 *        =
 *        =
 * B2------=------A2
 *        =
 *        =
 * A3------=------B3
 * Pin para poner las patas A al suelo= AD
 * Pin para poner las patas A al suelo= BD
 * Puerto paralelo  Pata
 * -------------------------
 * data 0          A1
 * data 1          A2
 * data 2          A3
 * data 3          AD
 * data 4          B1
 * data 5          B2
 * data 6          B3
 * data 7          BD
 */
#define LEGA1 1
#define LEGA2 2
#define LEGA3 4
#define LEGAD 8
#define LEGB1 16
#define LEGB2 32
#define LEGB3 64
#define LEGBD 128
#endif
```

BIBLIOGRAFÍA

John J. Craig: "**Introduction to Robotics, Mechanics and Control"**

Addison-Wesley, 1989

K. S. Fu, R.C. González, C. S. G. Lee: "**Robótica, control, detección e inteligencia"**

McGraw Hill, 1988

Sandler: "**Robotics: designing the mechanisms for automated machinary"**

Prentice Hall,1991

S. Bennet: **"Real-Time computer control: an introduction"**

Prentice Hall,1988

Direcciones de internet:

www.rambal.com

www.microbotica.es

www.microbot.tripod.cl

www.superrobótica.com