

UNIVERSIDAD DE CASTILLA - LA MANCHA

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS INDUSTRIALES**

CIUDAD REAL

PROYECTO FIN DE CARRERA Nº 11-09-201240

**DISEÑO, CONSTRUCCIÓN Y CONTROL DE UNA
SERPIENTE ROBÓTICA**



Autor:
ANDRÉS SAN - MILLÁN RODRÍGUEZ

Director:
VICENTE FELIU BATLLE

JUNIO DE 2011

Índice general

1. INTRODUCCIÓN	1
1.1. Motivación	1
1.2. Objetivos	3
1.3. Estado del arte	3
2. MECÁNICA	12
2.1. Primera generación	12
2.2. Segunda generación. Módulos ORM	14
2.3. Tercera generación. Robot serpiente autónomo	15
2.4. Sensores y actuadores	16
3. MÉTODO DE CONTROL	18
3.1. Locomoción serpentinoide	19
3.2. Algoritmos de implementación	22
3.2.1 Solución manual	24
3.2.2 Cinemática inversa	24
3.2.3 Osciladores sinusoidales	24
4. IMPLANTACIÓN DEL CONTROL	28
4.1. Programas desarrollados	29
4.1.1. Primera generación	29
4.1.2. Segunda generación	30
4.1.3. Tercera generación	30
4.2. Hardware de control	33

4.2.1. Tarjeta controladora JVM-Servo 28	33
4.2.2. Tarjeta controladora Pololu Micro Serial Servo	34
4.2.3. Módulos de comunicación inalámbrica Xbee	35
4.2.4. Adaptador Xbee Explorer USB	36
4.2.5. Dispositivo digitalizador	37
4.3. Electrónica del robot	38
4.3.1. Alimentación de la electrónica	42
5. ESPECIFICACIONES TÉCNICAS Y SIMULACIONES	45
5.1. Especificaciones técnicas	45
5.2. Simulaciones desarrolladas	47
6. EJEMPLOS DE APLICACIÓN	59
6.1. Tareas de reconocimiento	60
6.2. Tareas de inspección	61
7. CONCLUSIONES	63
7.1. Conclusiones	64
7.2. Problemas encontrados y propuestas de mejora	65
7.3. Colaboración en el software de robótica V-REP	66
Bibliografía	68
Anexos	70
A. PLANOS CONSTRUCTIVOS	1
B. DIAGRAMA DE CONEXIONADO DEL ROBOT	1
C. DESCRIPCIÓN DEL SOFTWARE DESARROLLADO	1

C.1. Control mediante voz	2
C.1.1. Módulo <i>ApiDeclarations</i>	3
C.1.2. Módulo <i>Funciones</i>	7
C.1.3. Módulo <i>ModuloUSB</i>	12
C.1.4. Módulo <i>Variables</i>	19
C.1.5. Formulario <i>FrmPuerto</i>	19
C.1.6. Formulario <i>frmMain</i>	21
C.2. Programa para el control del robot teleoperado	31
C.2.1. Módulo <i>Representa</i>	32
C.2.2. Módulo <i>Funciones</i>	35
C.2.3. Módulo <i>Funcionamiento</i>	37
C.2.4. Módulo <i>Constantes</i>	39
C.2.5. Módulo <i>Acciones</i>	39
C.2.6. Formulario <i>Form1</i>	39
D. HARDWARE UTILIZADO	1
D.1. Tarjeta controladora JVM-Servo 28	2
D.2. Tarjeta controladora Pololu Micro Serial Servo	11
D.3. Módulos de comunicación inalámbrica Xbee	20
D.4. Adaptador Xbee Explorer USB	31
D.5. Minicámara espía inalámbrica S130347	33
D.6. Servomotores Futaba S3003	36
E. DESCRIPCIÓN DEL SOFTWARE UTILIZADO	1
E.1. Google Sketchup	2
E.1.1. Sketchy Physics	3
E.2. V-REP	4
E.3. JVM-Servo	5
E.4. X-CTU	9

Índice de figuras

1.1. Fibroscopio	2
1.2. Módulos MY1	4
1.3. Robot MAKRO	5
1.4. Robot Aiko	6
1.5. Robot Anna Konda	6
1.6. Robot Piko	6
1.7. Robot del CMU	8
1.8. Omnitread	9
1.9. Desarrollo del robot ACM	10
2.1. Máquina de prototipado de PCB	13
2.2. Primer robot fabricado	13
2.3. Simulación de los módulos ORM	14
2.4. Módulo Autónomo	16
2.5. Robot serpiente	16
2.6. Servo Futaba S3003	17
2.7. Detalle de la cámara utilizada	17
3.1. Configuración 1D y 3D	19
3.2. Curva serpentinoide y ángulo de serpenteo	20
3.3. Definición de los parámetros de la curva	21
3.4. Curvas serpentinoides discretas	23

3.5. Espacios de formas para un modelo continuo y otro discreto	23
3.6. Comparación entre osciladores sinusoidales y curva serpentinoide	25
3.7. Modos de desplazamiento	27
4.1. Controladora de servos y programa de control	29
4.2. Estructura de control	31
4.3. Visualización del programa desarrollado	31
4.4. Tarjeta controladora JVM-Servo 28	33
4.5. Tarjeta controladora Pololu Micro Serial Servo	34
4.6. Módulo Xbee	35
4.7. Adaptador Xbee Explorer USB	36
4.8. Digitalizadora Zolid	37
4.9. Señal digital a 38400 bps	39
4.10. Comparación entre una señal digital y su descomposición de Fourier	41
4.11. Portapilas bh443d y características dimensionales	44
5.1. Tiempo de servicio en función de la descarga	47
5.2. Configuración de los módulos alineados y movimiento de oruga	49
5.3. Evolución de la posición con el tiempo	50
5.4. Trayectoria recorrida	50
5.5. Configuración de los módulos desfasados 90°	51
5.6. Evolución de la posición con el tiempo	51
5.7. Trayectoria recorrida	52
5.8. Desplazamiento lateral	52
5.9. Evolución de la posición con el tiempo	52
5.10. Trayectoria recorrida	53
5.11. Comparativa entre movimientos	54
5.12. Movimiento en círculo	54
5.13. Evolución de la posición con el tiempo	55

5.14. Trayectoria recorrida	55
5.15. Movimiento de rotación	56
5.16. Evolución de la posición con el tiempo	56
5.17. Trayectoria recorrida	57
5.18. Comparativa entre métodos de cambio de trayectoria	58
6.1. Fotograma del vídeo de la semana de la robótica en la ETSII	60
6.2. Fotograma del vídeo de inspección de tuberías	61
7.1. Pantalla del simulador V-REP	67
C.1. Interfaz principal del programa	2
C.2. Interfaz de control del puerto serie	2
C.3. Esquema del programa	3
C.4. Interfaz principal del programa	31
C.6. Esquema del programa	31
D.1. Aspecto de la tarjeta controladora JVM-Servo	2
D.2. Aspecto de la tarjeta controladora Pololu Micro Serial Servo	11
D.3. Aspecto de los módulos Xbee	20
D.4. Aspecto del dispositivo Xbee Explorer USB	31
D.5. Aspecto de la cámara	33
D.6. Aspecto del receptor	34
D.7. Servomotor Futaba S3003	36
D.8. Señales de control de un servomotor	36
E.1. Interfaz de usuario de Sketchup	3
E.2. Interfaz de usuario de JVM-Servo	5
E.3. Interfaz del programa X-CTU	9

Índice de tablas

3.1. Parámetros de la curva serpentinoide	20
4.1. Características del transistor utilizados	40
4.2. Comparativa entre pilas comerciales	42
4.3. Comparativa entre baterías recargables	43
5.1. Presupuesto y costos de fabricación	47
D.1. Características de la cámara	35
E.1. Comparativa entre programas de diseño	2

Agradecimientos

Nos esse quasi nanos, gigantium humeris insidentes, ut possimus plura eis et remotiora videre, non utique proprii visus acumine, aut eminentia corporis, sed quia in altum subvenimur et extollimur magnitudine gigantea.

Somos como enanos a hombros de gigantes. Podemos ver más, y más lejos que ellos, no por alguna distinción física nuestra, sino porque somos levantados por su gran altura.

Bernardo de Chartres

Gracias a todos los compañeros de fatigas que estuvieron a mi lado en este viaje, bien durante largos periodos de tiempo o solo breves instantes, pero siempre junto a mí en el momento oportuno.

Gracias a todos los profesores que supieron transmitirme su entusiasmo y sus conocimientos, despertando en mí la curiosidad y el interés por nuevas ramas del saber.

Gracias a Juan González, por inspirar este proyecto y hacer amigable un campo fascinante y difícil como la robótica.

Gracias a Pedro Hungría, por toda esa multitud de piezas que tuvo que fabricarme, (la máquina probablemente no lo agradezca tanto).

Y muchas gracias sobre todo a Vicente Feliu, por sus valiosas revisiones, sus amables consejos y su acertada dirección, sin la cual este proyecto no habría sido posible, y de haberlo sido, nunca de forma tan satisfactoria.

Por supuesto gracias a mi familia, pues su cariño, interés y apoyo han sido fundamentales a lo largo de toda mi vida y mi carrera. Gracias por esa capacidad para escuchar atentamente interminables relatos sobre robots y serpientes (seguramente llegaron a dudar de si estudiaba ingeniería o biología).

Si he llegado hasta aquí es gracias a que me aupé en los hombros de grandes personas, sin cuya ayuda y trabajo nunca habría podido alcanzar este momento. El camino continúa y por desvíos que tenga o largo que sea guardaré siempre en mi memoria un agradable recuerdo de todos y cada uno de los momentos compartidos.

A todos Muchas Gracias.

Memoria



E.T.S DE INGENIEROS INDUSTRIALES



1

INTRODUCCIÓN

1.1. Motivación

1.2. Objetivos

1.3. Estado Del arte

En este primer capítulo se expondrán los motivos del diseño y posterior desarrollo del robot ORM, el primer robot serpiente desarrollado dentro del Grupo de Robótica, Automática y Mecatrónica (GRAM) de la Escuela Técnica Superior de Ingenieros Industriales de la UCLM en Ciudad Real. El proyecto ha sido desarrollado durante el curso 2010-2011, realizado por Andrés San Millán Rodríguez bajo la supervisión de Vicente Feliu Batlle.

1.1. Motivación

El presente proyecto final de carrera representa una continuación al programa de iniciación a la investigación que acompañaba a las becas de excelencia de la Junta de Comunidades de Castilla la Mancha.

Para justificar la elección del tipo de robot deberían contestarse a las siguientes preguntas: ¿Para qué son buenos los robots serpiente? ¿Dónde podrían usarse?

Si se examinan las distintas áreas de aplicación destacan algunas donde los robots serpiente tendrán un gran impacto.

- **Inspección.**

Muchas técnicas de inspección en la industria y en medicina se basan en mecanismos de base fija como boroscopios, videoscopios y fibroscopios. Estos dispositivos se usan principalmente para inspeccionar cavidades que no pueden observarse directamente. Las distintas aplicaciones incluyen mantenimiento de motores aéreos, control de calidad en manufactura, y monitorización de procesos e inspección en plantas químicas. Los boroscopios simples han demostrado su utilidad, pero nuevos dispositivos articulados, autopropulsados y capaces de seguir trayectorias complejas podrían tener muchas otras aplicaciones

Otra necesidad real es la inspección de los tubos de refrigeración de las centrales eléctricas, que pueden llegar a medir 18m de longitud y sólo 10mm de diámetro

Las plantas químicas tienen enormes y complejas redes de tuberías que suelen requerir inspecciones o búsquedas de atascos. El procedimiento de adivinación más apertura es una técnica muy cara incluso sin que se tengan en cuenta los costes asociados al tiempo de parada.

Los dispositivos móviles de tuberías o “cerdos de tuberías” son muy utilizados, pero su utilidad es limitada. Un robot serpiente resultaría de gran ayuda. Con un dispositivo serpentinoide las inspecciones in-situ y la localización de precisión podrían ver reducidos significativamente tanto sus costes como sus tiempos de parada

Conscientes de sus limitaciones, los desarrolladores de equipamiento de inspección están muy interesados en los dispositivos autopropulsados. Sin embargo la industria es altamente competitiva y es difícil obtener información sobre sus actividades. Aun así su interés se pone de manifiesto en el creciente número de patentes sobre videoscopios autopropulsados. Ninguno de estos dispositivos se encuentra maduro todavía



Figura 1.1 Fibroscopio

- **Reconocimiento.**

El reconocimiento ofrece algunas nuevas aplicaciones para los robots serpiente. La capacidad para dirigir pequeños ojos y oídos de forma teleoperada ofrece atractivas posibilidades para las fuerzas y cuerpos de seguridad. La capacidad de penetración en vegetación densa y terrenos accidentados en general podría aportar información imposible de obtener de otro modo

1.2. Objetivos

El objetivo principal del proyecto es el de diseñar y construir un robot serpiente lo más compacto posible con capacidad de autopropulsión y desplazamiento en un entorno tridimensional. El control de la locomoción será realizado mediante un PC de forma completamente inalámbrica

Como se partió del diseño mecánico de los módulos Y1 desarrollados por Juan González (Juan González, 2008) se decidió que el robot debía cumplir los siguientes objetivos principales:

- Rediseño completo de la mecánica del robot.
- Diseño de una estructura completamente modular para reducir costes de fabricación y aumentar la flexibilidad de operación y montaje.
- Diseño de una interfaz de control amigable y flexible, y exploración del control mediante comandos de voz.
- Adición de una cámara de video para poder realizar tareas de inspección.

1.3. Estado del arte

El desarrollo de robots serpiente no es un proyecto exclusivo de esta universidad, pudiéndose encontrar gran variedad de robots desarrollados en otras universidades por equipos de desarrollo completos.

Cabe destacar que precisamente debido a esta diversidad no existe una descripción normalizada de las distintas características de cada prototipo, pudiéndose encontrar solamente los aspectos más significativos de cada robot puestos de manifiesto en la amplia variedad de artículos publicados por sus autores,

Algunos ejemplos se listan a continuación ordenados según el país al que pertenecen:

- **España:**

Cube revolutions (U. Autónoma de Madrid)

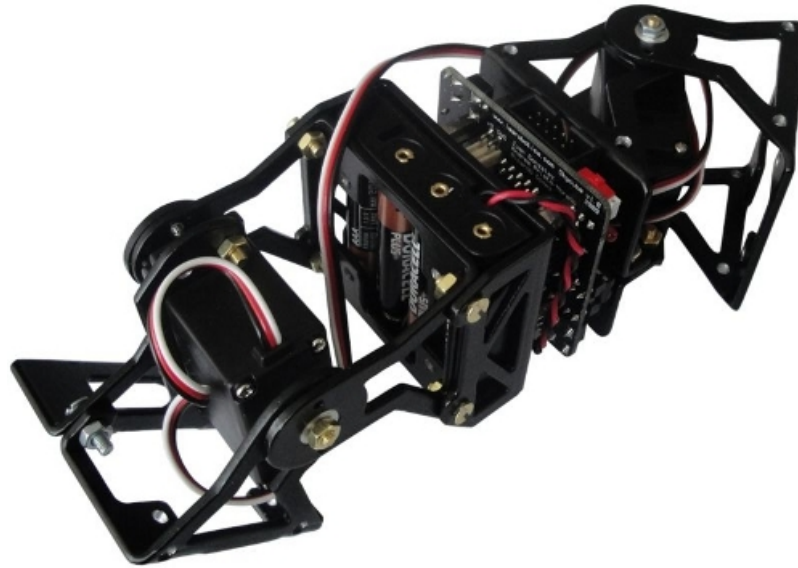


Figura 1.2 Módulos MY1

Características técnicas:

- Arquitectura modular fácilmente reconfigurable.
- Sección transversal: 52 x 52 mm.
- Longitud de cada módulo: 72,4 mm.
- Actuadores: Servomotores Futaba 3003 (Par: 3,2 Kg-cm Velocidad: 0,23 seg/60°).
- Número de grados de libertad por módulo: 1.
- Número de módulos usualmente empleados: 8.
- Alimentación: mediante umbilical.
- Control: Desde PC mediante umbilical.
- Sensores: Ninguno.
- Material de fabricación: PVC expandido.

Otros datos de interés:

El robot Cube revolutions se encuentra formado por los llamados módulos Y1 desarrollados con motivo de la realización de la tesis doctoral del profesor Juan González (Juan González 2008). Las últimas versiones del robot se encuentran formadas por los módulos MY1 mostrados en la Figura 1.2, funcionan alimentados con pilas y su control se realiza mediante movimientos preprogramados en la tarjeta Skycube desarrollada como hardware libre.

- **Alemania:**

MAKRO (I. Fraunhofer de Sistemas Autónomos Inteligentes)



Figura 1.3 Robot MAKRO

Características técnicas:

- Diseño no modificable.
- Sección transversal: 16 cm de diámetro.
- Longitud total del robot: 150 cm.
- Peso aproximado: 30 Kg.
- Número de grados de libertad por módulo: 3.
- Número de módulos empleados: 5.
- Número de GDL totales: 22 (4x3 en uniones + 5x2 en las ruedas).
- Alimentación: Baterías.
- Control: Desde PC mediante WLAN / autónomo mediante planificación de misiones.
- Sensores: Encoders en ruedas y articulaciones, una cámara estereoscópica y dos escáneres láser en cada extremo del robot, además de inclinómetros bidimensionales y sensores infrarrojos y ultrasónicos de proximidad en cada segmento.
- Material de fabricación: Metal.

Otros datos de interés:

El robot se encuentra equipado con un Pentium MMX a 166MHz con 2GB de disco duro, tarjeta gráfica para procesamiento de imágenes, tarjeta Ethernet para comunicaciones gíreles un microcontrolador SIEMENS C167 en cada segmento encargados de la coordinación articular. Es el único robot de los que aquí se presentan con aplicación industrial real en la actualidad.

- **Noruega:**

Aiko, Anna Konda, Piko (N.T.N.U. Trondheim)



Figura 1.4 Robot Aiko



Figura 1.5 Robot Anna Konda



Figura 1.6 Robot Piko

Características técnicas (AIKO):

- Diseño no modificable.
- Sección transversal: 15 cm de diámetro.
- Longitud total del robot: 150 cm.
- Peso aproximado: 7 Kg.
- Número de grados de libertad por módulo: 2.
- Actuadores: Motores eléctricos DC (Par: 2,5Nm).
- Número de GDL totales: 20.
- Alimentación: Umbilical.
- Control: Desde PC mediante Umbilical.
- Sensores: Táctiles en la superficie del robot.
- Material de fabricación: Metal.

Características técnicas (Anna Konda):

- Diseño no modificable.
- Sección transversal: 27 cm de diámetro.
- Longitud total del robot: 3 m.
- Peso aproximado: 75 Kg.
- Número de grados de libertad por módulo: 2.
- Actuadores: Cilindros hidráulicos (presión de funcionamiento: 100 bar).
- Número de GDL totales: 20.
- Alimentación: Umbilical.
- Control: Desde PC mediante Umbilical.
- Sensores: Táctiles en la superficie del robot.
- Material de fabricación: Metal.
- Velocidad de Desplazamiento: 20-30 cm/s.

Otros datos de interés:

El robot utiliza microcontroladores AVR Atmega128 para coordinar el movimiento de las articulaciones, el control central se encuentra en un ordenador desde el que se dirige el robot.

- **EEUU:**

Modular Snake Robot (U. Carnegie Mellon)



Figura 1.7 Robot del CMU

Características técnicas:

- Diseño modular.
- Sección transversal: 6,2 cm de diámetro.
- Longitud total del robot: 84 cm.
- Peso aproximado: 1,3 Kg.
- Número de grados de libertad por módulo: 1.
- Actuadores: Servomotores Hitec HS-5955TG modificados.
- Número de GDL totales: 18.
- Alimentación: Umbilical.
- Control: Desde PC mediante Umbilical.
- Sensores: Cámara en la cabeza del robot.
- Material de fabricación: Aluminio.
- Velocidad de Desplazamiento: 10 cm/s.

Otros datos de interés:

Cada módulo posee un microprocesador Amtel para el control de las posiciones angulares, lo que permite que sólo dos cables recorran el robot por completo, sin embargo las primeras versiones necesitaban un número mayor de cables (tres para cada servomotor).

Omnitread (U. Michigan)



Figura 1.8 Omnitread

Características técnicas:

- Diseño modular difícilmente modificable.
- Sección transversal: 8,2x8,2 cm.
- Longitud total del robot: 94 cm.
- Peso aproximado: 4 Kg.
- Número de grados de libertad por módulo: 2.
- Actuadores: Fuelles neumáticos (pueden adoptar 33° en cualquier dirección).
- Número de GDL totales: 13 (6x2 en uniones + 1 el motor de las orugas).
- Alimentación: Umbilical.
- Control: Desde PC mediante Umbilical.
- Sensores: Cámara en la cabeza del robot.
- Material de fabricación: metal.
- Velocidad de Desplazamiento: 15 cm/s.

Otros datos de interés:

Para el control del robot se necesitan 3 operadores manejando 6 joysticks (uno por cada grado de libertad de las articulaciones). El segmento central alberga el motor encargado de transmitir potencia a todas las orugas que recubren su cuerpo.

- **Japón:**

Active Cord Mechanism (Lab. Robótica Hirose Fukushima)

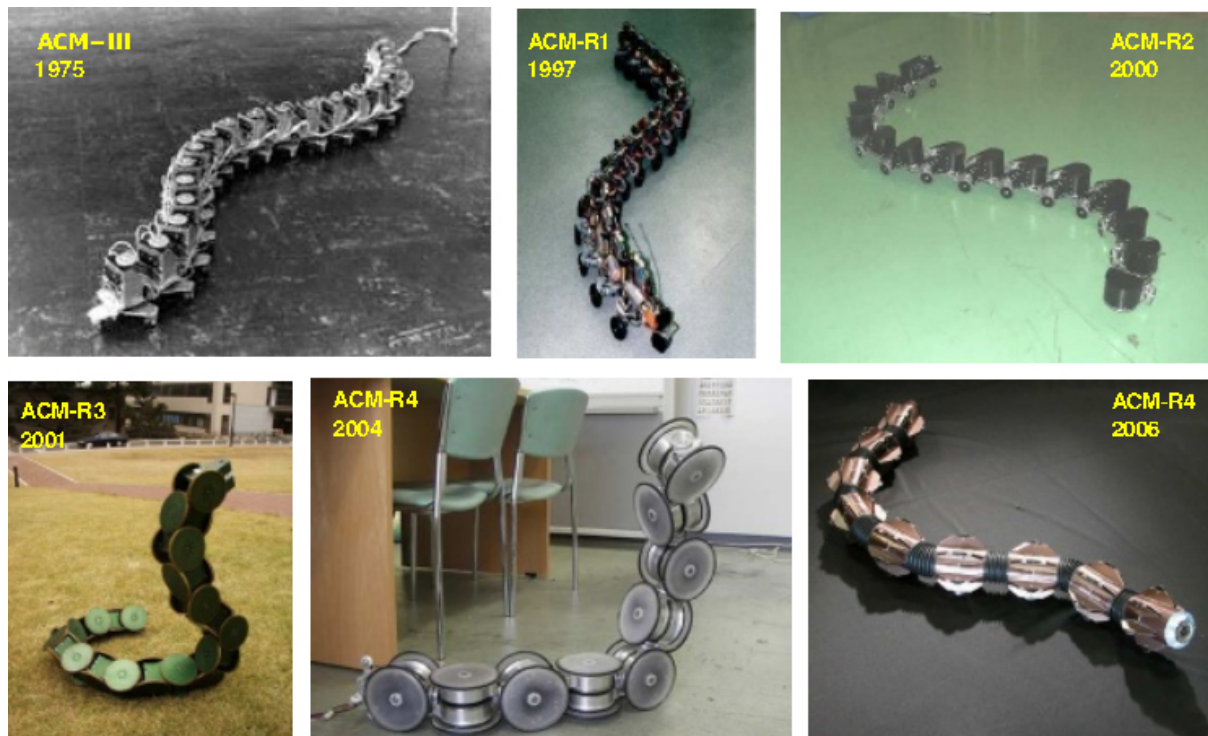


Figura 1.9 Desarrollo del robot ACM

Características técnicas:

- Diseño modular fácilmente modificable.
- Sección transversal: 8 cm diámetro.
- Longitud total del robot: 175 cm.
- Peso aproximado: 7,5 Kg.
- Número de grados de libertad por módulo: 2.
- Actuadores: Servomotores eléctricos (Par: 9Nm Velocidad 70°/seg).
- Número de GDL totales: 18.
- Alimentación: Baterías.
- Control: Teleoperado desde unidad de control.
- Sensores: Cámara en la cabeza del robot.
- Material de fabricación: metal.
- Velocidad de Desplazamiento: 80 cm/s.

Otros datos de interés:

El robot está dotado de ruedas pasivas montadas sobre aletas que le permiten gran velocidad de desplazamiento tanto en tierra como en el agua, así mismo cuenta con un microprocesador en cada módulo que le permite identificar el número de módulos de los que está compuesto para adaptar su sistema de locomoción según convenga.

2

MECÁNICA

2.1. Primera generación

2.2. Segunda generación. Módulos ORM

2.3. Tercera generación. Robot serpiente autónomo

2.4. Sensores y actuadores

En este capítulo se detallarán las distintas etapas de diseño seguidas para la elaboración del robot serpiente y se mostrarán los distintos diseños considerados para cada una de las versiones del mismo.

2.1. Primera generación

En primer lugar y como una primera aproximación al diseño y elaboración de robots serpiente se optó por la fabricación de los módulos Y1 (Juan González, 2008) introducidos en el apartado dedicado al estado del arte, esta decisión se tomó ya que su diseño es relativamente simple y están disponibles de forma Open Source.

El robot elaborado se encuentra constituido por tres módulos Y1 idénticos como puede observarse en la Figura 2.2, para más detalles consultar el anexo A correspondiente a planos constructivos.

Tras realizar una evaluación de las herramientas disponibles para la realización de los módulos se optó por utilizar la máquina de prototipado de PCB pero substituyendo el material de trabajo por un material compuesto de alta rigidez y baja densidad denominado cartón prensado.

2.1. Primera generación



Figura 2.1 Máquina de prototipado de PCB

Con esta decisión se redujo el coste de elaboración del prototipo, pero fue necesaria una adaptación de los planos originales, ya que el grosor de las piezas resultantes era algo menor que el de las utilizadas en el modelo original.

Una vez fabricadas las piezas se fabricó un primer robot de tres segmentos de longitud y controlado mediante cable. Con este modelo disponible se acometió la tarea de realizar un simulador de la física del robot para posteriormente validar futuros diseños antes de su fabricación.

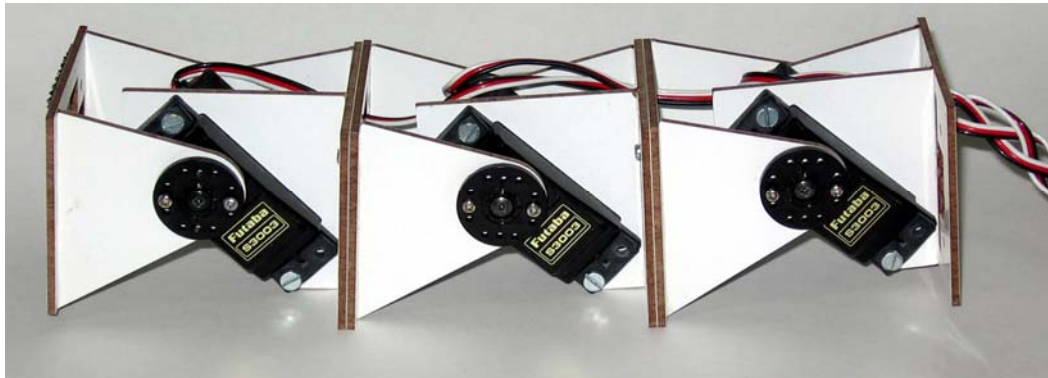


Figura 2.2 Primer robot fabricado

2.2. Segunda generación. Módulos ORM

Como primera experiencia de diseño se acometió la tarea de crear unos módulos de menores dimensiones y de mayor funcionalidad. Esto se consiguió con el diseño de los módulos ORM, cuya principal característica es su elevada simetría (su planta es de forma octogonal y su perfil es completamente simétrico).

La forma de estos módulos, como se comprobó con las simulaciones, permite que el robot opere de forma idéntica independientemente de su orientación relativa en el plano, lo que posibilita que aunque ruede por una pendiente siempre se controle igual.

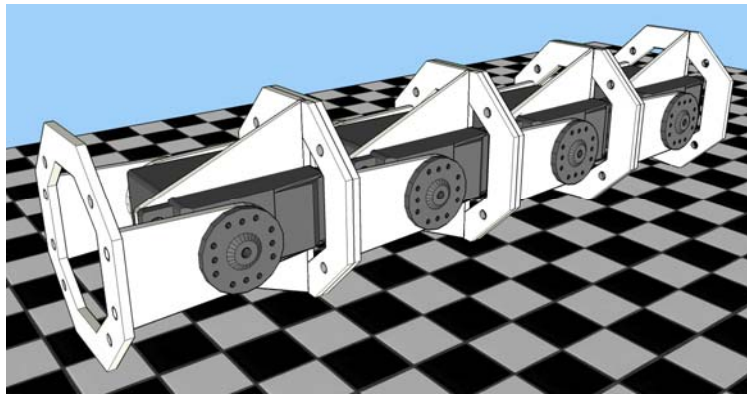


Figura 2.3 Simulación de los módulos ORM

Este diseño de módulos completamente simétricos supone una mejora con respecto a los módulos Y1 ya que la capacidad de locomoción se ve beneficiada debido a su forma, pero impiden la inclusión de baterías o electrónica de control por su reducido tamaño.

Debido a esta razón solamente se construyó un módulo para validar sus características dimensionales, ya que se decidió acometer la tarea de diseñar un prototipo completo con baterías y sistema de control lo antes posible.

El programa utilizado para el diseño de estos módulos fue Google SketchUp, la elección de este programa de diseño se debió fundamentalmente a que se distribuye de forma gratuita, su curva de aprendizaje es pequeña para realizar diseños sencillos, y presenta un motor de física que permite realizar simulaciones simples.

Precisamente el motor de física proporcionado por el plugin SketchyPhysics fue el utilizado para realizar las simulaciones con este modelo del robot.

La principal ventaja de la combinación de Google Sketchup y SketchyPhysics en las primeras etapas de diseño es que las simulaciones físicas se convierten en una extensión natural e intuitiva de los modelos tridimensionales realizados.

Sin embargo el uso de este simulador presenta serios inconvenientes que determinarán su substitución en las siguientes etapas de desarrollo del robot, entre los que cabe destacar:

- Solamente pueden simularse cuerpos físicos convexos sencillos,
- No existe la posibilidad del control del coeficiente de rozamiento entre cuerpos,
- Solamente pueden programarse movimientos muy sencillos.
- No es posible la inclusión de sensores que determinen cambios en el comportamiento del robot.
- No se pueden obtener datos cuantitativos de las simulaciones realizadas como distancia recorrida o trayectoria seguida por el robot.

Para más información acerca del programa, así como para ver una comparativa entre los distintos paquetes de CAD existentes puede consultarse el Anexo E dedicado a software utilizado.

2.3. Tercera generación. Robot serpiente autónomo

La etapa de desarrollo de los módulos ORM fue muy breve ya que enseguida se puso de manifiesto que sus reducidas dimensiones impedían la fabricación de un robot autónomo, por lo que se procedió al rediseño completo de los módulos.

Para diseñar los módulos que constituyen el prototipo final del robot se tuvieron en cuenta los siguientes aspectos:

- Necesidad de incorporar la alimentación de cada servomotor.
- Necesidad de incorporar la controladora de cada servomotor.
- Necesidad de incorporar un sistema de comunicación inalámbrico.
- Realizar un diseño modular con posibilidad de intercambiabilidad.

Inicialmente el diseño se enfocó de forma que cada módulo fuese completamente independiente de los demás, dotado con su propio sistema de comunicación y con su propia alimentación, para que el robot estuviese formado por un enjambre de módulos cooperativos.

No obstante la inclusión de un dispositivo de radiofrecuencia y un controlador de servomotores en cada módulo disparaba el precio del prototipo, por lo que aunque el diseño mecánico de los módulos admite esta configuración, el robot presenta un control centralizado.

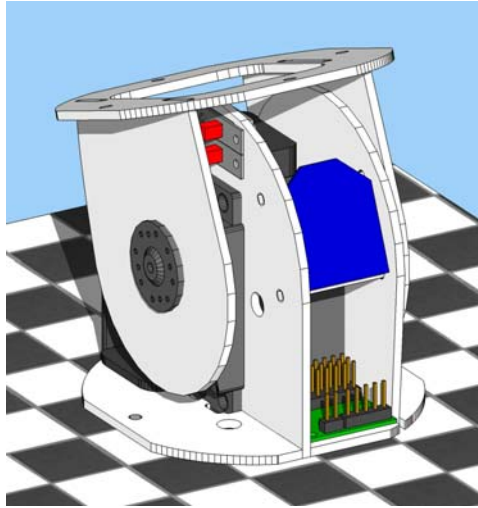


Figura 2.4 Módulo autónomo

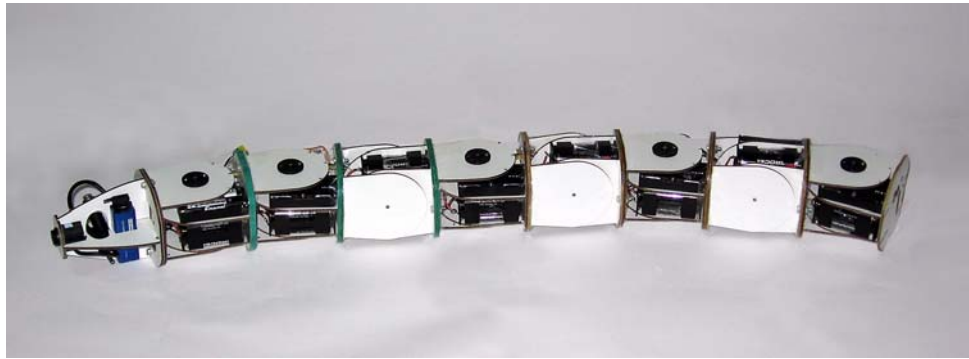


Figura 2.5 Robot serpiente

2.4. Sensores y actuadores

Los actuadores utilizados para el movimiento del robot son un total de ocho servomotores de clase Futaba 3003, estos actuadores se eligieron entre la amplia variedad disponible en el mercado ya que al ser servomotores de aeromodelismo su precio es bastante reducido (aprox. 10 € la unidad).

La decisión de utilizar servomotores vino determinada por el hecho de que ya presentan integrado todo el sistema de control y lazos de realimentación, de forma que pueden tratarse en primera aproximación como una caja negra donde una consigna de entrada determina la posición angular adoptada de forma instantánea. Una descripción más detallada de los servomotores puede encontrarse en el Anexo D.



Figura 2.6 Servo Futaba 3003

En cuanto a los sensores, el único presente en el robot es la cámara inalámbrica situada en el módulo de cabeza y cuya principal tarea es la retransmisión de imágenes al ordenador desde el que se controla el robot para que el usuario pueda ver el entorno por donde se desplaza el robot y realizar las tareas de inspección pertinentes.

Como extra cabe destacar que la cámara presenta un micrófono incorporado, lo que capacita al robot para llevar a cabo posibles tareas de reconocimiento. La descripción técnica completa de la cámara se encuentra en el Anexo D.



Figura 2.7 Detalle de la cámara utilizada

3

MÉTODO DE CONTROL

3.1. Locomoción serpentinoide

3.2. Algoritmos de implementación

3.2.1. Solución manual

3.2.2. Cinemática inversa

3.2.3. Osciladores sinusoidales

En este capítulo se enumerarán las distintas metodologías existentes para abordar el problema de la locomoción y el control de un robot ápedo, así mismo se detallará la solución escogida para el control del robot realizado.

El problema del control: Dado un robot con N articulaciones, se trata de encontrar las funciones $\varphi_i(t)$ que deciden cómo tiene que variar con el tiempo el ángulo de cada articulación para lograr el movimiento deseado del robot.

3.1 Locomoción serpentinoide

La primera idea que viene a la mente al contemplar el robot, es que de acuerdo a su diseño, éste puede encuadrarse dentro del marco de los robots bio-inspirados, por lo que en primer lugar debería de investigarse como resuelve la naturaleza el problema del movimiento en aquellos organismos que se desplazan reptando, tales como orugas o serpientes.

El hecho de considerar dos animales tan aparentemente dispares viene determinado por las dos posibles configuraciones que puede adoptar el robot desarrollado: con todos los módulos orientados de igual forma (configuración 1D como las orugas), o con los módulos alternando 90° (configuración 3D como las serpientes)



Figura 3.1 Configuraciones 1D y 3D

Tras consultar la amplia variedad de bibliografía especializada existente (ver el apartado bibliografía) puede sintetizarse que ambos animales se desplazan mediante la propagación de una serie de ondas sinusoidales a lo largo de su cuerpo.

Centrándonos en las serpientes, ya que la configuración 3D es más útil para nuestros propósitos de inspección, y según la tesis del profesor Hirose (Shigeo Hirose 1993) la función que mejor describe este movimiento y que mejor puede adaptarse a un robot es la curva serpentinoide.

3.2. Algoritmos de implementación

La curva serpentinoide es, por definición aquella cuyo ángulo de curvatura varía sinusoidalmente con la distancia a lo largo de la curva

Su **curvatura** está dada por la ecuación:

$$K(s) = -\frac{2\pi k}{l} \alpha \sin\left(\frac{2\pi k}{l} s\right) \quad 3.1$$

Donde:

l	Longitud de la curva. $l > 0$
s	Distancia a lo largo de la curva. $s \in [0, l]$
k	Número de ondulaciones. $k > 0$
α	Ángulo de serpenteo. $\alpha \in [0, 121]$

Tabla 3.1 Parámetros de la curva serpentinoide

La curvatura $K(s)$ se define como el ritmo de cambio del vector unidad tangente a la curva, es decir:

$$K(s) = \frac{d\alpha_s}{ds} \quad 3.2$$

A su vez el ángulo que forma la tangente a la curva en el punto s con el eje x se denota por α_s . Hirose demostró que la expresión de α_s viene dada por la ecuación:

$$\alpha_s = \alpha \cos\left(\frac{2\pi k}{l} s\right), \quad 3.3$$

donde el parámetro α se denomina ángulo de serpenteo y se define como el ángulo que forma la tangente a la curva que pasa por el punto $s=0$ y determina la forma que tendrá la curva.

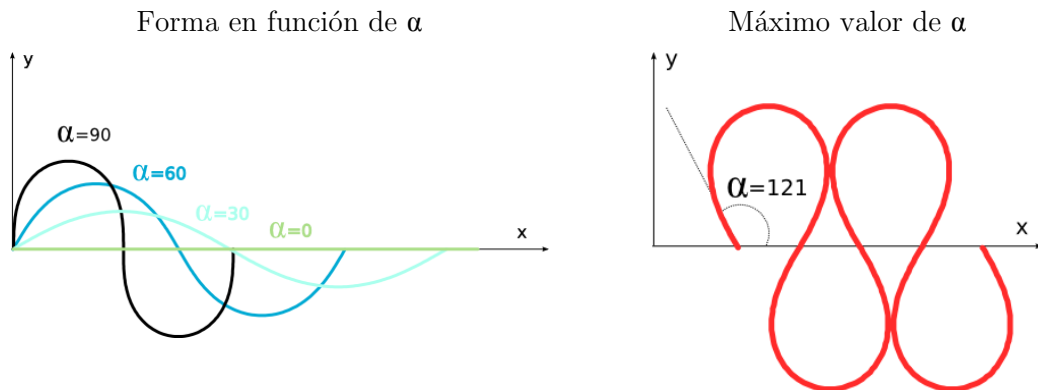


Figura 3.2 Curva serpentinoide y ángulo de serpenteo

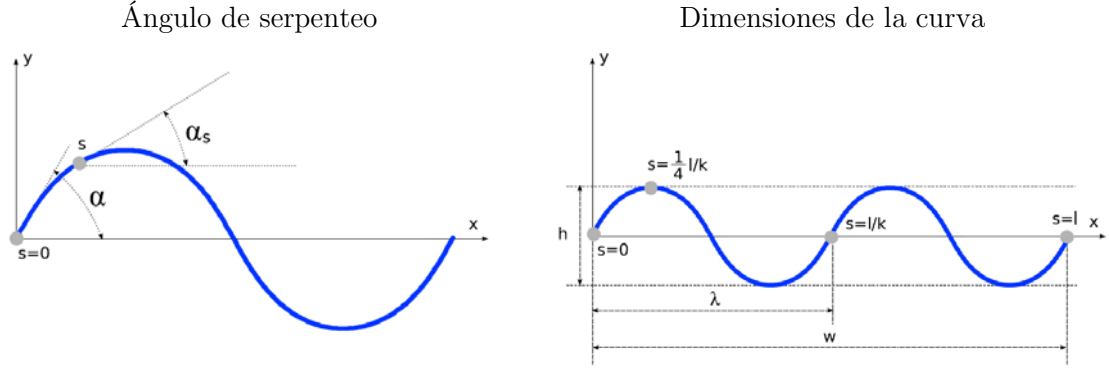


Figura 3.3 Definición de los parámetros de la curva

La forma de la curva serpentinoide se muestra en las Figuras 3.2 y 3.3. Es una curva periódica formada por k ondulaciones donde el periodo espacial es l/k y la sección de la curva entre $s=0$ y $s=l/k$ se denomina ondulación.

Por último la onda serpentinoide se define como una curva serpentinoide donde la curvatura $K(s)$ depende también de la fase ϕ :

$$K(s, \phi) = -\frac{2\pi k}{l} \alpha \sin\left(\phi + \frac{2\pi k}{l} s\right) \quad 3.4$$

Las coordenadas cartesianas x e y de los puntos de la onda se calculan mediante la integración de cosenos y senos respectivamente de α_s :

$$x(s, \phi) = \int_0^s \cos\left(\alpha \cos\left(\phi + \frac{2\pi k}{l} s\right)\right) ds \quad 3.5$$

$$y(s, \phi) = \int_0^s \sin\left(\alpha \cos\left(\phi + \frac{2\pi k}{l} s\right)\right) ds \quad 3.6$$

Estas integrales no presentan solución analítica y deben resolverse numéricamente.

Nuevamente tomando como base el desplazamiento de las serpientes, se puede encontrar que el movimiento se consigue mediante la superposición de dos ondas de forma transversal, una en el plano horizontal y otra en el plano vertical.

La sincronización de estas dos ondas es lo que determina los distintos modos de desplazamiento que pueden verse en las serpientes, así como otros movimientos que no se encuentran presentes en la naturaleza.

3.2. Algoritmos de implementación

A partir del trabajo de observación y modelizado de la locomoción en serpientes realizado por el profesor Hirose puede observarse que las ecuaciones del movimiento reflejan la sincronización entre ondas comentada anteriormente:

$$x(s) = sJ_0(\alpha) + \frac{4l}{\pi} \sum_{m=1}^{\infty} \frac{(-1)^m}{2m} J_{2m}(\alpha) \sin\left(m\pi \frac{s}{l}\right) \quad 3.7$$

$$y(s) = \frac{4l}{\pi} \sum_{m=1}^{\infty} (-1)^{m-1} \frac{J_{2m-1}(\alpha)}{2m-1}(\alpha) \sin\left(\frac{2m-1}{2} \pi \frac{s}{l}\right) \quad 3.8$$

Donde $x(s)$ e $y(s)$ representan el desplazamiento en las direcciones x e y respectivamente medido a lo largo de la longitud del cuerpo curvado s ; l es la longitud del cuerpo; y $J(\cdot)$ son funciones de Bessel.

Hirose descubrió que el movimiento de las serpientes en superficies de fricción constante se encuentra muy próximo a las ondas serpentinoideas anteriores, así mismo estas curvas son significativamente distintas a las sinusoidales o a las clotoideas.

3.2 Algoritmos de implementación

Ya se ha visto que puede lograrse el movimiento del robot combinando dos ondas serpentinoideas perpendiculares con cierto desfase entre ellas, pero existe el problema de cómo adaptar estas ondas a parámetros articulares del robot.

Los parámetros que caracterizan la onda serpentinoidea ya se han comentado anteriormente. Estos son: ángulo de serpenteo α , número de ondulaciones k y desfase ϕ .

Puesto que los robots ápodos son por naturaleza discretos necesitamos realizar una aproximación discreta de la curva serpentinoidea. Su definición será por tanto la misma que para la curva continua, pero considerando que la curvatura K varía de forma discreta con la variable i en lugar de una variación de forma continua como lo hacía con s .

La variable discreta i hace referencia al número de módulos que componen el robot, con lo que tendremos que la longitud total l será $l = ML$, siendo M el número total de módulos y L la longitud de cada módulo

Todo esto puede verse de forma más clara en la Figura 3.4 donde se representa una curva serpentinoidea discreta con los siguientes parámetros $\alpha=60$, $k=1$ y 8 módulos, además se comparará con la misma curva continua.

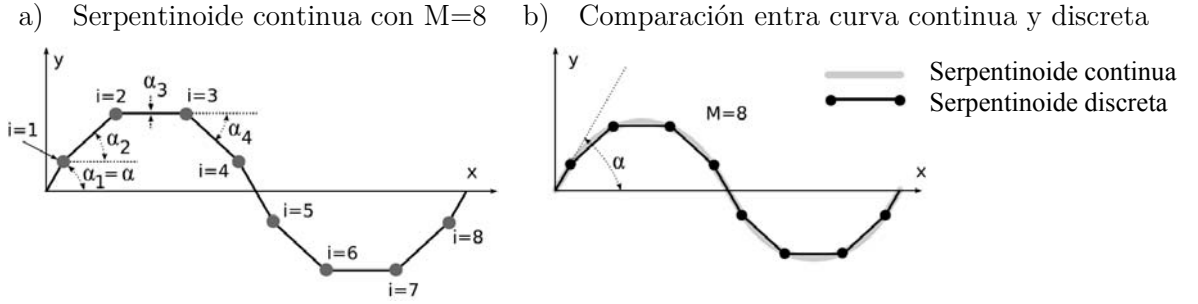


Figura 3.4 Curvas serpentinoides discretas

Una vez definidas las curvas serpentinoides tanto continua como discreta y sus parámetros característicos es necesario definir la relación existente entre ambas en función de sus variables.

Para la curva continua los parámetros variables son el ángulo de serpenteo α y el número de ondulaciones k , ya que se consideran los demás parámetros constantes, mientras que para la curva discreta las variables son el ángulo de serpenteo y la variable M_u definida como el número de ondulaciones dividido por el número total de módulos.

Puesto que cada curva depende de dos variables podemos dibujar el espacio de formas que se obtienen para distintas combinaciones de sus variables representadas en los ejes de ordenadas y abscisas obteniéndose el resultado que se muestra en la Figura 3.5

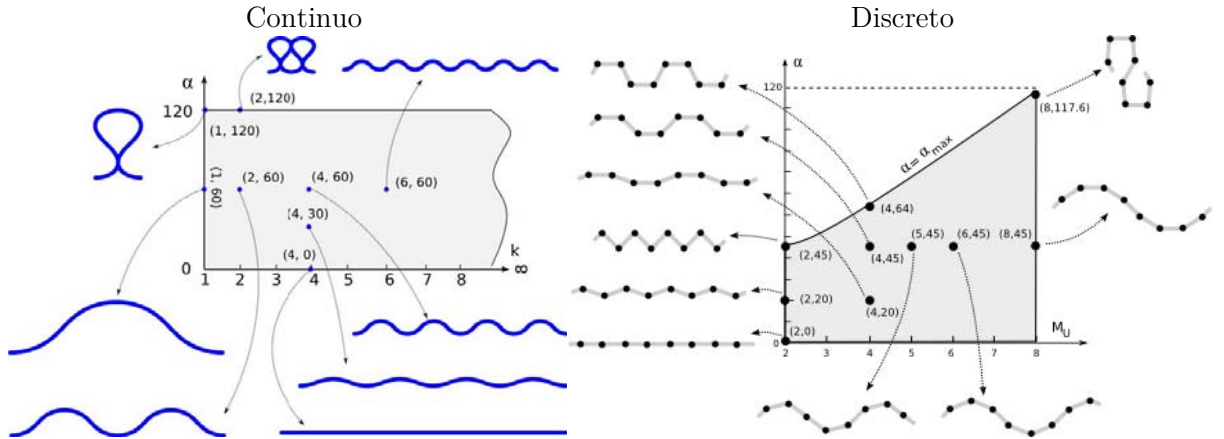


Figura 3.5 Espacios de formas para un modelo continuo y otro discreto

Una vez que se obtiene la curva deseada para el modelo discreto es necesario conocer los valores angulares que debe adoptar cada articulación del robot, si se pueden obtener estos valores articulares para cada instante de tiempo, la locomoción del robot se conseguirá empleando ondas serpentinoides en las que se variará el desfase con el tiempo para lograr la propagación de la onda a lo largo del cuerpo del robot.

3.2. Algoritmos de implementación

Según (Juan González, 2008) existen diferentes soluciones para abordar esta tarea entre las que destacan:

3.2.1. Solución manual

Esta forma de encarar el problema consiste en la elaboración de una serie de tablas en las que se encuentra el valor del ángulo que debe adoptar cada articulación en cada instante. De esta forma el programa que ejecuta el robot recorre la lista de valores y cuando llega al final comienza nuevamente, produciendo un movimiento cíclico.

Operar de esta manera, aunque sencillo cuando se trata con un número pequeño de articulaciones, resulta ineficaz para casos con gran número de articulaciones como el que nos ocupa, además resulta imposible una modificación en tiempo real de la onda generada.

3.2.2. Cinemática inversa

En este caso se trata de aplicar las mismas técnicas que se emplean para calcular los ángulos que deben adoptar las articulaciones de los robots manipuladores para alcanzar la posición y orientación deseada.

Sólo que en este caso, en lugar de tomar como referencia la base del robot se toma como referencia el módulo del extremo y éste se mueve conforme una función sinusoidal.

Aunque los robots ápodos pueden considerarse como manipuladores con un elevado grado de redundancia, y con este algoritmo las tablas de ángulos se generan automáticamente, resulta una forma ineficaz de resolver el problema debido al elevado coste computacional que conllevan los algoritmos iterativos de resolución.

3.2.3. Osciladores sinusoidales

Esta solución tiene un origen bio-inspirado. La existencia de los centros generadores de patrones, o CGP fue documentada por primera vez por el doctor Wilson en su obra “The central nervous control of flight in a locust” (Wilson, 1965) donde estudia el vuelo de las langostas.

Básicamente el modo de actuación de los CGP consiste en “módulos de control autónomos” existentes en todos los animales, estos módulos son los encargados de generar los movimientos rítmicos propios de la locomoción. En esta configuración los estímulos del cerebro no se encargarían directamente del movimiento, sino de la modulación de los CGP

Aunque un modelo matemático exacto de los CGP existentes en los animales conlleva una gran complejidad, puede adoptarse como solución una primera aproximación en forma de función senoidal simple.

3.2. Algoritmos de implementación

De hecho el método empleado para el control del robot consiste precisamente en esto, cada módulo oscila conforme a una función senoidal y el control de sus parámetros (amplitud, frecuencia y desfase) es controlado por el usuario que actúa de “cerebro”.

Esta aproximación de los CGP por osciladores sinusoidales puede utilizarse ya que si se observan las siguientes imágenes obtenidas mediante simulación (Figura 3.6), un modelo discreto en el que las articulaciones sigan un patrón sinusoidal reproducen con gran fidelidad la forma de las ondas serpentinoides.

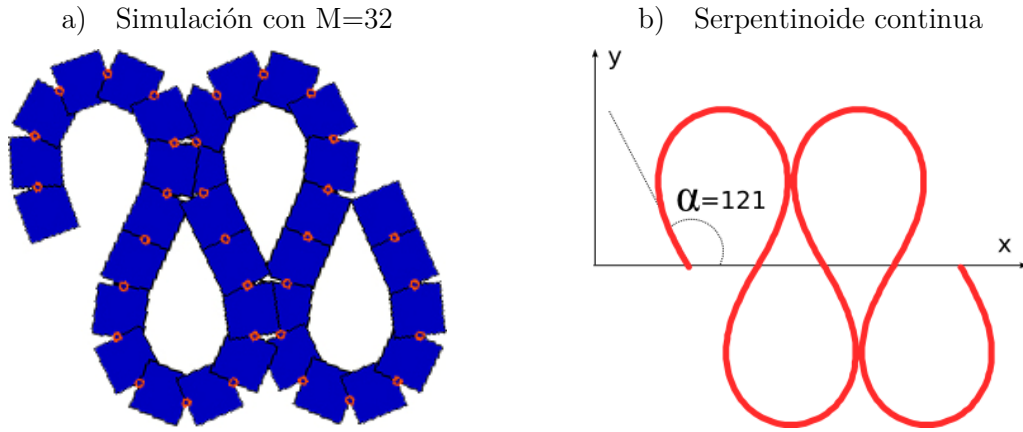


Figura 3.6 Comparación entre osciladores sinusoidales y curva serpentinoide

La ecuación a la que atienden las posiciones angulares de los servomotores presenta la siguiente forma:

$$\theta_i = A \sin(\omega t + \phi_i) \quad 3.9$$

Donde θ_i es el ángulo adoptado por la articulación, siendo el valor correspondiente a cero aquel en el que los segmentos formarían una línea recta.

La amplitud de la onda $A \in [0^\circ, 90^\circ]$. La velocidad de propagación de la onda se representa por ω , Φ representa el desfase por módulo y el parámetro i indica la posición que ocupa cada módulo

En la simulación de la Figura 3.6 los parámetros adoptados son $A = 45^\circ$ y $\Phi = 22,5^\circ$.

El motivo del gran parecido entre los resultados de aplicar osciladores sinusoidales en los servomotores y la onda serpentinoide resulta evidente si se repasa la definición de la onda serpentinoide (ec. 3.1), en ella se indica que la curvatura básicamente varía de forma senoidal y esto es justamente lo que hacemos al variar la posición de las articulaciones de forma senoidal, ya que la posición angular de las articulaciones se corresponde con la definición de la curvatura (ec. 3.2)

3.2. Algoritmos de implementación

Para lograr un desplazamiento tridimensional se deben combinar dos ondas serpentinoides situadas de forma transversal, por lo que se utilizarán dos ondas de control para los osciladores sinusoidales, una vertical y otra horizontal.

Las ecuaciones que determinan las posiciones de las articulaciones a lo largo del tiempo para un robot compuesto por M segmentos se muestran a continuación:

$$\theta_i = A_v \sin(\omega_v t + \phi_v i + \Delta\phi_v) \quad 3.10$$

$$\theta_j = A_h \sin(\omega_h t + \phi_h j + \Delta\phi_h) \quad 3.11$$

Cuyos parámetros son: por un lado para la onda vertical: posición que ocupa el servomotor (i), amplitud de la onda (A_v), velocidad de la onda (ω_v), desfase por módulo (Φ_v), desfase de la onda ($\Delta\Phi_v$). Y para la onda horizontal: posición que ocupa el servomotor (j), amplitud de la onda (A_h), velocidad de la onda (ω_h), desfase por módulo (Φ_h), desfase de la onda ($\Delta\Phi_h$).

En este punto es necesario realizar un resumen de los modos de desplazamiento existentes en el robot a fin de definir cada caso y evitar posibles ambigüedades.

Cuando el robot se encuentra ensamblado en la configuración en la que todos sus módulos tienen la misma orientación (configuración unidimensional) solamente admite la aplicación de una onda serpentinoide para efectuar su desplazamiento, a este modo de desplazamiento se le denominará movimiento de oruga.

Cuando el robot se encuentra ensamblado en la configuración tridimensional (con la orientación de sus módulos alternando 90°) admite la aplicación de una sola onda serpentinoide o de dos ondas serpentinoides de forma transversal.

A la aplicación de una sola onda serpentinoide sobre la configuración tridimensional se la denominará desplazamiento en línea recta.

Las distintas combinaciones de dos ondas serpentinoides sobre la configuración tridimensional producen diversos métodos de desplazamiento que se muestran en las cuatro últimas ilustraciones de la Figura 3.7. En dicha figura pueden verse las denominaciones de cada tipo de movimiento y los parámetros empleados para conseguirlos.

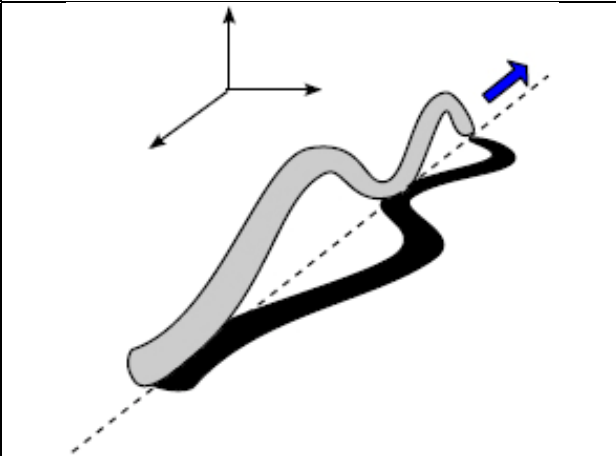
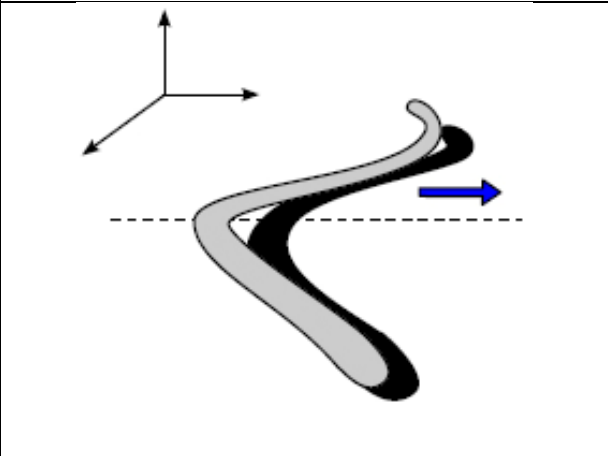
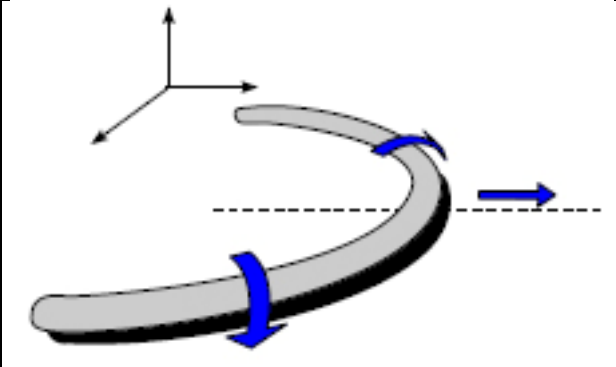
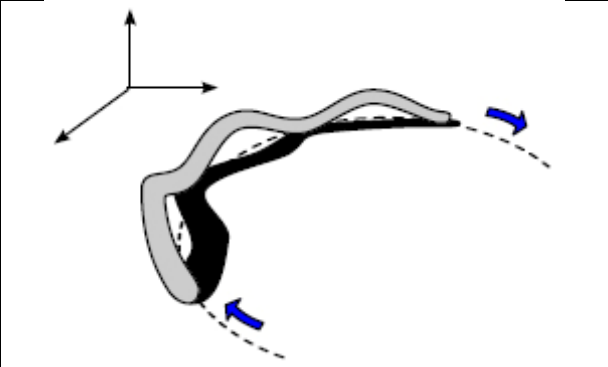
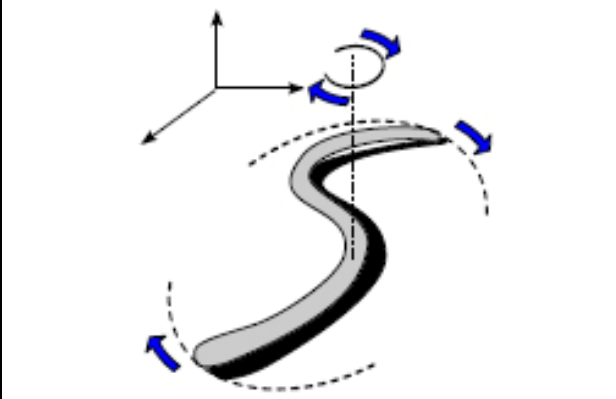
Línea recta	Desplazamiento lateral
	
$A_h = 0, \omega_h = 0, \Phi_h = 0, \Delta\Phi_h = 0$ $A_v = 20, \omega_v = 1, \Phi_v = 60, \Delta\Phi_v = 0$	$A_h = 45, \omega_h = 1, \Phi_h = 60, \Delta\Phi_h = 0$ $A_v = 45, \omega_v = 1, \Phi_v = 60, \Delta\Phi_v = 0$
Rodar	Círculo
	
$A_h = 20, \omega_h = 1, \Phi_h = 0, \Delta\Phi_h = 0$ $A_v = 20, \omega_v = 1, \Phi_v = 0, \Delta\Phi_v = 90$	$A_h = 180/M = 0, \Phi_h = 0, \Delta\Phi_h = 90$ $A_v = 20, \omega_v = 1, \Phi_v = 60, \Delta\Phi_v = 0$
Rotación	
	
$A_h = 45, \omega_h = 1, \Phi_h = 30, \Delta\Phi_h = 0$ $A_v = 20, \omega_v = -1, \Phi_v = 30, \Delta\Phi_v = 90$	

Figura 3.7 Modos de desplazamiento

4

IMPLANTACIÓN DEL CONTROL

4.1. Programas desarrollados

4.1.1. Primera generación

4.1.2. Segunda generación

4.1.3. Tercera generación

4.2. Hardware de control

4.2.1. Tarjeta controladora JVM-Servo 28

4.2.2. Tarjeta controladora Pololu Micro Serial Servo

4.2.3. Módulos de comunicación inalámbrica Xbee

4.2.4. Adaptador Xbee Explorer USB

4.2.5. Dispositivo digitalizador

4.3. Electrónica del robot

En este capítulo se describirán brevemente cuáles fueron los programas desarrollados para realizar la implantación del algoritmo de control necesario para llevar a cabo la locomoción del robot serpiente. Así mismo se describirán las características técnicas de los distintos componentes electrónicos utilizados en la construcción del mismo.

4.1 Programas desarrollados

Uno de los objetivos planteados para la realización del proyecto fue la creación de un programa de control capaz de permitir un manejo intuitivo y sencillo del robot pese a la complejidad que supone la sincronización de todas sus articulaciones.

Puesto que como se indicó anteriormente se construyeron diversas generaciones de robots fue necesario el diseño de diversos programas de control. Los aspectos más relevantes se citan a continuación.

4.1.1. Primera generación

Debido a que el primer robot utilizaba una tarjeta controladora de servos modelo JVM-Servo 28 los primeros programas tuvieron que adoptar el enfoque de la solución manual comentado anteriormente.

Las ventajas de esta tarjeta son múltiples: Posee conexión USB, lo que facilita su uso de forma plug & play, es capaz de controlar hasta 28 servomotores de manera simultánea y cuenta con un programa de control gratuito desarrollado por el propio fabricante de la tarjeta.

El principal inconveniente es que el programa de control se basa en la definición de tablas de posiciones y tiempos de ejecución.

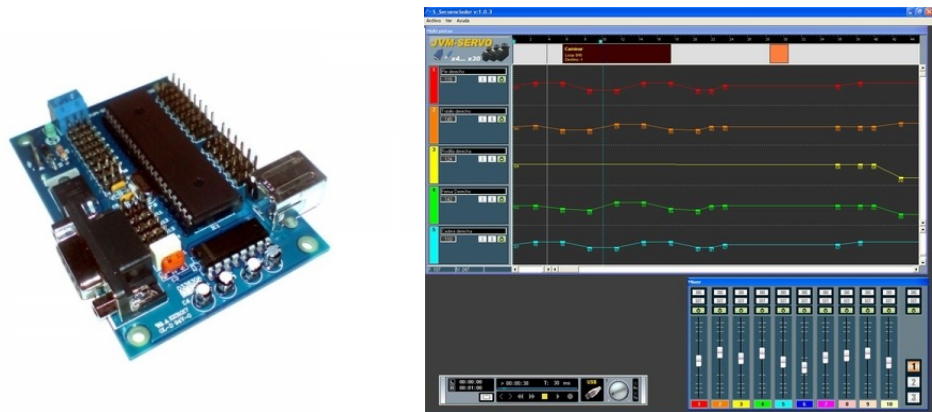


Figura 4.1 Controladora de servos y programa de control

Tras rechazar la posibilidad de controlar el robot mediante esta técnica se desarrolló un programa en Visual Basic que permitía una mayor flexibilidad al calcular de forma automática los valores para cada articulación a partir de los parámetros de la onda sinusoidal.

La principal razón para utilizar Visual Basic en el desarrollo de este programa fue la existencia de rutinas ya existentes en distintos repositorios de internet capaces de controlar la comunicación vía puerto USB.

Una vez conseguido el movimiento sincronizado de todas las articulaciones del robot, se realizaron diversos experimentos a fin de mejorar la interfaz de usuario del programa.

Entre las diversas versiones realizadas, cabe destacar dos:

- El control del robot mediante comandos de voz, para lo cual se desarrolló una aplicación utilizando el “*Speech Recognition SDK*” proporcionado por Microsoft.

Esta opción se descartó por dos motivos fundamentales: los comandos de voz reconocibles son solamente en inglés y es necesario un gran tiempo de calibración del sistema de reconocimiento de voz. Además el reconocimiento de los distintos comandos se ve fuertemente interferido por ruidos ambientales.

- El control del robot mediante teclado, es decir, utilizando como entradas solamente las teclas de dirección del teclado, el programa de control selecciona el desfase necesario entre las ondas, así como su velocidad y amplitud. Esta opción resultó no ser válida en el modelo final del robot debido a la gran variedad de formas de desplazamiento.

4.1.2. Segunda generación

El diseño de los módulos de la segunda generación no llegó a poseer un prototipo físico, por lo que en lo que a su programación se refiere todos los esfuerzos se centraron en el desarrollo de los algoritmos de control en el simulador de distribución gratuita compuesto por el programa “*Google SketchUp*” y por su motor de física “*Sketchyphysics*”.

4.1.3. Tercera generación

Para el desarrollo de los programas de control del modelo final del robot tuvieron que tenerse en cuenta varios aspectos: la inclusión de una cámara inalámbrica de la que debía ser posible obtener imágenes, la utilización de una nueva tarjeta controladora de servos y la utilización de dispositivos de radiofrecuencia conectados mediante puerto serie rs-232 que utilizan su propio protocolo de comunicación

Todos estos aspectos determinaron la necesidad de rehacer desde cero el programa de control de una forma más estructurada. Un diagrama de bloques en el que se sintetiza la estructura de control completa puede verse en la Figura 4.2:

4.1. Programas desarrollados

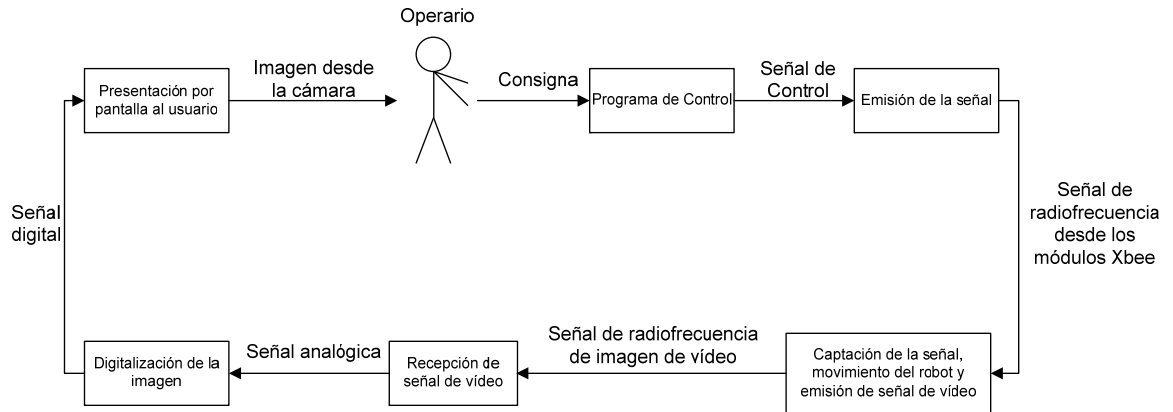


Figura 4.2 Estructura de control

Como puede apreciarse en la Figura 4.2 el lazo de realimentación lo cierra el operario que controla el robot utilizando como señal de error la diferencia entre la imagen obtenida por la cámara del robot, que le indica dónde se encuentra y la posición a la que desea ir, definida por el objetivo de la tarea que pretenda realizar.

De esta forma la función que debe llevar a cabo el programa de control es la sincronización de los motores del robot, que vendrá determinada por la elección del operario de un determinado modo de locomoción de entre una serie de opciones disponibles

El nuevo programa de control se realizó en Visual Basic por dos razones fundamentales: la principal es su facilidad para la programación y el manejo del puerto serie, y la segunda por su capacidad para desarrollar aplicaciones con interfaz gráfico de una manera rápida y sencilla.

En el nivel más alto se encuentra la capa destinada a la presentación de la interfaz gráfica al usuario, que muestra el aspecto de la Figura 4.3.

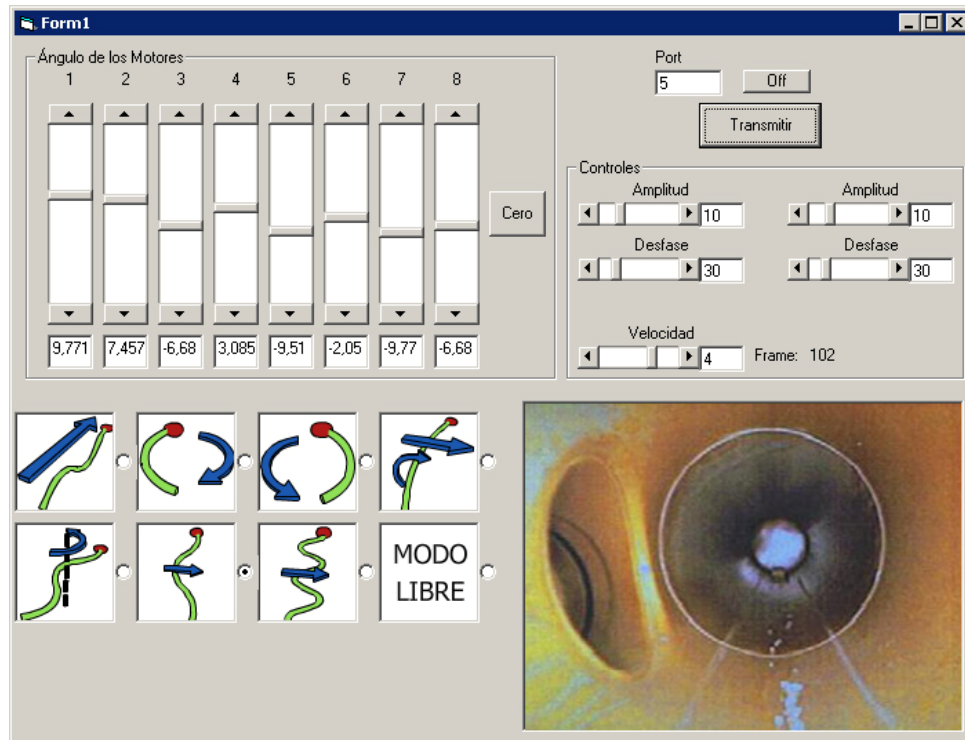


Figura 4.3 Visualización del programa desarrollado

Como puede observarse en la Figura 4.3 en la interfaz gráfica se muestra por un lado la imagen que el robot capta por la cámara (esta imagen se sitúa en la parte inferior derecha del programa), el resto de la pantalla se encuentra ocupada por las distintas entradas que debe introducir el operario y cuya función se detalla a continuación:

Los dos botones y el cuadro de texto situado en la parte superior derecha son los elementos necesarios para elegir la salida de puerto serie donde se encuentra conectada la tarjeta de radiofrecuencia Xbee y para decidir si ésta se encontrará enviando datos o no.

Los controles de la parte central derecha se corresponden con los parámetros modificables que caracterizan las ondas sinusoidales que recorren la serpiente, recordemos que según se combinen éstos se obtendrá una forma de locomoción distinta.

Como sería laborioso recordar las combinaciones adecuadas para obtener el mecanismo de locomoción deseado, esta tarea la realiza el panel seleccionable que se encuentra en la parte inferior izquierda de la pantalla. Aquí el operario puede identificar cada forma de locomoción por su icono. Tras seleccionar una opción, el programa modifica los parámetros de las ondas para obtener el mecanismo de locomoción deseado.

Además para dotar al programa de mayor flexibilidad puede elegirse una opción de “modo libre” que permite al operario definir los parámetros de las ondas de forma manual.

Por último, en la parte superior izquierda de la pantalla se encuentran una serie de barras deslizantes que controlan la posición angular de los servomotores individualmente. Esas barras deslizantes, además de proporcionar información gráfica de las posiciones de las articulaciones, pueden servir para mover cada servomotor de forma manual si se encuentra seleccionada la opción “modo libre”.

Esta función resulta de gran utilidad para aquellas ocasiones en las que con el robot inmóvil se quiere orientar de forma precisa la cámara, ya que al encontrarse ésta en el extremo del robot el movimiento de los dos últimos segmentos permite mantener estable el campo de visión durante los desplazamientos y, cuando el robot se encuentra detenido, apuntar hacia cualquier objetivo de interés.

Por debajo de esta apariencia gráfica se ejecutan distintas subrutinas encargadas de la comunicación por puerto serie, de la obtención de las posiciones de los servomotores para cada instante, del acondicionamiento de estos datos para el envío a la tarjeta controladora en el protocolo propio de ésta y de la recepción de la imagen y de su salida por pantalla.

Todas estas subrutinas pueden consultarse en el Anexo C.

4.2 Hardware de control

Para controlar el robot se utilizaron dos tipos distintos de tarjetas controladoras de servomotores y módulos de comunicación inalámbrica.

4.2.1. Tarjeta controladora JVM-Servo 28

Las principales características de esta tarjeta se detallan a continuación:

- No requiere alimentación en la parte lógica (electrónica) cuando está conectada al USB.
- Circuito de alimentación externo protegido contra inversiones de polaridad (para la conexión RS232).
- Alimentación externa a de 7,5V a 30V
- Velocidad RS232 configurable por software.
- Velocidades soportadas: 9600 (defecto) 19200, 38400, 57600 y 115200 baudios.
- Existe un software ya desarrollado para poder controlarla

Estas características, sobre todo la facilidad de conexión mediante USB, propiciaron que se eligiese como la primera placa de control para desarrollar el primer prototipo controlado mediante umbilical.

Pero sus grandes dimensiones (53x84x15), así como la necesidad de alimentación a 7,5 V hicieron que resultase imposible su incorporación a un robot de reducidas dimensiones



Figura 4.4 Tarjeta controladora JVM-Servo 28

4.2.2. Tarjeta controladora Pololu Micro Serial Servo

Las principales características de esta tarjeta se detallan a continuación:

- Soporta hasta 8 servomotores
- Dimensiones: 23.11 x 23.11 x 13
- Resolución 0.5 μ s (unos 0.05°)
- Rango: 250 – 2750° μ s
- Tensión de la alimentación de la lógica: 5 – 16V
- Tensión de la línea de datos: 0 / 5 V
- Frecuencia de pulsos: 50 Hz
- Velocidades soportadas: 1200 – 38400 baudios (detectados automáticamente)
- Consumo de corriente 5 mA (valor medio)



Figura 4.5 Tarjeta controladora Pololu

Las reducidas dimensiones de esta tarjeta controladora de servos (en la fecha de compra la más pequeña del mercado), junto con su capacidad de comunicación por puerto serie, la convierten en ideal para una aplicación como la que nos ocupa, en la que la señal transmitida por radiofrecuencia será de tipo puerto serie y el espacio disponible es pequeño.

Para más detalles sobre esta tarjeta consultar el Anexo D.

4.2.3. Módulos de comunicación inalámbrica Xbee

El hardware empleado para comunicar de forma inalámbrica el robot con el ordenador de control fueron los módulos Xbee comercializados por MaxStream.

Las principales funcionalidades y características de estos módulos se listan a continuación:

- Soporta el protocolo ZigBee
- Velocidad de transmisión: 20-250 Kbps
- Rango de alcance 10-75
- Hasta 255 dispositivos simultáneos
- Banda de frecuencia 2,4 GHz
- Tensión de alimentación: 2,8 – 3,4 V
- Consumo de corriente: 45 mA

Además de las características mencionadas anteriormente, hay que tener en cuenta que estos módulos tienen la gran ventaja de ser capaces de autoorganizarse en forma de red, escaneando automáticamente cada cierto tiempo en busca de nuevos dispositivos y elaborando mapas de rutas óptimas de transmisión de datos.



Figura 4.6 Módulo Xbee

La configuración de estos módulos puede realizarse mediante dos procedimientos:

- Operación API: este procedimiento resulta muy potente ya que permite controlar completamente la transmisión de datos. Para trabajar en este modo es necesario utilizar comandos de bajo nivel propios de los módulos. Trabajar de esta forma permite:
 - Transmitir datos simultáneamente a varios dispositivos.
 - Identificar la dirección de origen de cada paquete recibido.
 - Recibir informes sobre el éxito o fallo en la transmisión de paquetes.
- Operación en modo transparente: éste es el modo de operación en el que vienen configurados los módulos de forma predeterminada. Además fue el modo elegido para operar en la transmisión de datos a nuestro robot.

La principal ventaja de trabajar de esta forma es que su utilización es completamente transparente a nuestro programa de control, podemos conectar el módulo Xbee al puerto USB del ordenador mediante un adaptador, pero éste sencillamente lo detectará como una conexión puerto serie adicional.

4.2.4. Adaptador Xbee Explorer USB

Cabe destacar la importancia de este componente a la hora de trabajar con los módulos Xbee, ya que éstos reciben los comandos de control mediante el estándar rs-232 típico de puerto serie, pero aunque los niveles lógicos de éste van de -12V a 12V, los módulos Xbee deben recibir unos niveles de tensión comprendidos entre 0V y 3,3V.

Además resulta especialmente relevante el hecho de que los ordenadores portátiles que precisamente serían los más indicados para controlar un robot de estas características, no suelen presentar puerto serie, en su lugar disponen de varios conectores USB.



Figura 4.7 Adaptador Xbee Explorer USB

Por estos motivos el empleo de este componente resulta esencial tanto para adaptar los niveles de tensión como para facilitar la conexión Xbee – ordenador.

4.2.5. Dispositivo digitalizador

Las principales características de este dispositivo se muestran a continuación:

- Interfaz Plug & Play USB 2.0
- Detección automática NTSC/PAL
- Resoluciones de vídeo:

NTSC: 720x480 / 30FPS

PAL: 720x576 / 25FPS

- Entrada de vídeo externa: Vídeo compuesto y S-Vídeo
- Requisitos mínimos:

Windows 2000/XP/Vista

400 MB de espacio libre en disco para instalación de software

Monitor de 16-bits con resolución 1024x768

DirectX 9



Figura 4.8 Digitalizadora Zolid

El uso de una tarjeta digitalizadora se convierte en parte imprescindible del acondicionamiento de la señal de vídeo, ya que si recordamos las características de la cámara utilizada por la serpiente, ésta utiliza un aparato receptor de vídeo cuya salida es una señal analógica de vídeo compuesto que la mayoría de los ordenadores no son capaces de reconocer.

Mediante la incorporación de una tarjeta digitalizadora se consigue una interfaz de conversión entre señales de vídeo sencilla y de bajo coste, ya que es capaz de enviar la imagen captada por la cámara a través del USB del ordenador a cualquier aplicación.

Para el caso que nos ocupa fue necesario desinstalar la cámara que incorpora el ordenador de control (un portátil) y realizar un código capaz de manejar la señal de vídeo y mostrarla por pantalla. Una vez que ya disponemos de la señal de vídeo sería sencillo implementar diversas aplicaciones de procesamiento de imagen según la necesidad requerida.

4.3 Electrónica del robot

El objetivo de este capítulo es complementar los detalles ya proporcionados sobre el diseño mecánico de los módulos que componen el robot con los detalles acerca de la electrónica que contienen. Así como detallar los problemas encontrados al trabajar con los componentes comerciales indicados en el apartado anterior.

La tarea del diseño y la construcción de los módulos del robot se enfocó de una manera distinta a la habitual: en lugar de diseñar un chasis mecánico sobre el que ajustar y acomodar la electrónica, lo que se hizo fue crear modelos tridimensionales de los componentes electrónicos y de los servomotores con un alto grado de detalle y precisión, acomodarlos de forma que ocupasen el menor espacio posible y diseñar el chasis de forma que sujetase firmemente todos los componentes.

Esta forma de abordar el diseño del robot vino condicionada fundamentalmente porque el uso de componentes comerciales como el dispositivo Xbee y la controladora de servos impide el diseño de circuitos con especificaciones personalizadas, como su forma o grado de miniaturización.

La relativa simplicidad del esquema de funcionamiento del robot, consistente en la recepción de una consigna desde el ordenador central, la generación de señales de control para los servomotores y su envío a los mismos, determina una estructuración de la electrónica en dos bloques claramente diferenciados: por una parte se encuentra el módulo Xbee encargado de la recepción de señal, y por otra parte la controladora de servos encargada del posicionamiento de los mismos.

No obstante, si se presta atención a la descripción presentada anteriormente de estos componentes, se comprobará que aunque trabajan con el mismo protocolo de comunicación (comunicación RS-232), el nivel de tensión utilizado por ambos es distinto.

Este inconveniente hace necesario incluir algún tipo de interfaz convertidora de niveles de tensión entre los 3,3 V utilizados por la Xbee y los 5-16V empleados por la controladora de servos.

Aunque existen diversas opciones para abordar este problema como son convertidores especiales para estos niveles lógicos, amplificadores operacionales, o incluso ciertos montajes en forma de divisor de tensión con dos resistencias, la opción elegida se basa en el uso de un transistor empleado en corte y saturación.

La elección escogida se basa por una parte en el escaso espacio disponible en los módulos y por otra en el sencillo sistema de implementación, como puede verse en el esquema de conexionado presentado en los anexos. Para realizar este montaje solamente se necesita un transistor y una resistencia de $1K\Omega$.

El diagrama de conexión completo del robot se encuentra en el Anexo B.

4.3 Electrónica del robot

Los posibles problemas que podrían aparecer debido a las frecuencias de conmutación, ya que el protocolo de comunicación RS-232 es propio de electrónica digital y los transistores típicamente se emplean en electrónica analógica, no son tales como se demostrará en el siguiente análisis:

El protocolo utilizado de comunicación entre Xbee y tarjeta controladora de servos es una señal tipo puerto serie con un bitrate de 38400 bps. Esta medida quiere decir que se transmitirán 38400 bits por segundo.

Puesto que un bit es básicamente una onda cuadrada, el peor caso posible sería la transmisión de una trama de la forma 101010... como la que se muestra en la Figura 4.9

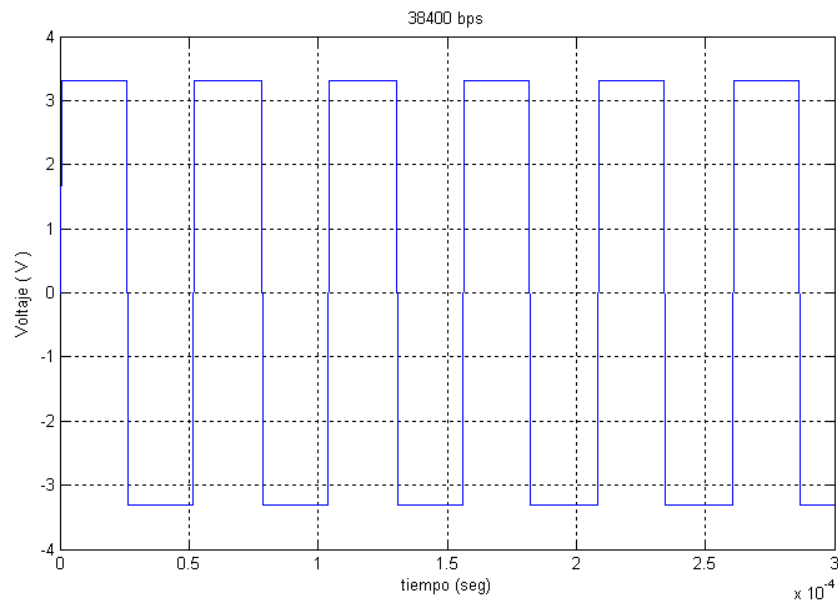


Figura 4.9 Señal digital a 38400 bps

Este sería el peor caso porque supone el empleo de la frecuencia de conmutación más elevada, ya que cualquier otro paquete de datos supondría la generación de una onda cuadrada de igual o menor frecuencia.

Si se analiza la ficha técnica del transistor empleado (transistor de propósito general NPN BC237B), pueden verse algunos datos de interés mostrados en la Tabla 4.1.

En dicha tabla puede verse que la frecuencia de transición es de 100 MHz, Típicamente el transistor deberá operar por debajo de esta frecuencia para evitar cualquier tipo de problemas. No obstante, si se toma una frecuencia de operación mucho menor, como por ejemplo 1MHz, el transistor seguirá realizando su función adecuadamente como se verá a continuación.

4.3 Electrónica del robot

Para analizar si el transistor puede conmutar lo suficientemente rápido como para reproducir la onda cuadrada de una señal a 38400 bps, puede realizarse el análisis de Fourier de dicha señal. Si la frecuencia de operación del transistor es superior al armónico de mayor frecuencia de la descomposición de Fourier el transistor será válido.

NPN general purpose transistors

BC237; BC237B

CHARACTERISTICS

$T_{amb} = 25\text{ }^{\circ}\text{C}$ unless otherwise specified.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
I_{CBO}	collector cut-off current	$I_E = 0; V_{CB} = 30\text{ V}$	–	–	15	nA
		$I_E = 0; V_{CB} = 30\text{ V}; T_J = 150\text{ }^{\circ}\text{C}$	–	–	5	μA
I_{EBO}	emitter cut-off current	$I_C = 0; V_{EB} = 5\text{ V}$	–	–	100	nA
h_{FE}	DC current gain	$I_C = 0.1\text{ mA}; V_{CE} = 5\text{ V}; \text{ see Fig.2}$	–	250	–	
h_{FE}	DC current gain BC237 BC237B	$I_C = 2\text{ mA}; V_{CE} = 5\text{ V}; \text{ see Fig.2}$	120 200	– –	460 460	
V_{CEsat}	collector-emitter saturation voltage	$I_C = 100\text{ mA}; I_B = 5\text{ mA}; \text{ note 1}$	–	–	600	mV
V_{BEsat}	base-emitter saturation voltage	$I_C = 100\text{ mA}; I_B = 5\text{ mA}$	–	–	1200	mV
V_{BE}	base-emitter voltage	$I_C = 2\text{ mA}; V_{CE} = 5\text{ V}$	580	–	700	mV
C_c	collector capacitance	$I_E = I_B = 0; V_{CB} = 10\text{ V}; f = 1\text{ MHz}$	–	11	–	pF
C_e	emitter capacitance	$I_C = I_E = 0; V_{EB} = 0.5\text{ V}; f = 1\text{ MHz}$	–	1.5	–	pF
f_T	transition frequency	$I_C = 10\text{ mA}; V_{CE} = 5\text{ V}; f = 100\text{ MHz}$	100	–	–	MHz
F	noise figure	$I_C = 200\text{ }\mu\text{A}; V_{CE} = 5\text{ V}; R_S = 2\text{ k}\Omega; f = 1\text{ kHz}; B = 200\text{ Hz}$	–	–	10	dB

Note

1. Pulse test: $t_p \leq 300\text{ }\mu\text{s}; \delta \leq 0.02$.

Tabla 4.1 Características del transistor utilizado

La frecuencia de una señal cuadrada de 38400 bps es 19200 Hz. Aplicando un desarrollo en serie de Fourier hasta el armónico 26 se obtiene la gráfica de la Figura 4.10.

En dicha gráfica puede verse que se reproduce la onda cuadrada con gran fidelidad, y que el armónico de mayor frecuencia exigiría al transistor una frecuencia de conmutación superior o igual a 979200 Hz (inferior al MHz que suponíamos frecuencia de operación del transistor).

De hecho, para seguir en un rango de operación inferior a la frecuencia de transición de 100 MHz del transistor, se debería continuar la descomposición de Fourier hasta el armónico 2604.

A un resultado similar se llegaría si se realizases el análisis en base al teorema del muestreo de Nyquist-Shannon.

Hay que destacar que la forma de onda que reproduce el análisis de Fourier no es exactamente la onda cuadrada, y que a medida que se adhieren más términos a las series ésta se va aproximando a la onda cuadrada, dado que las oscilaciones se vuelven más rápidas y más pequeñas, pero los picos no disminuyen.

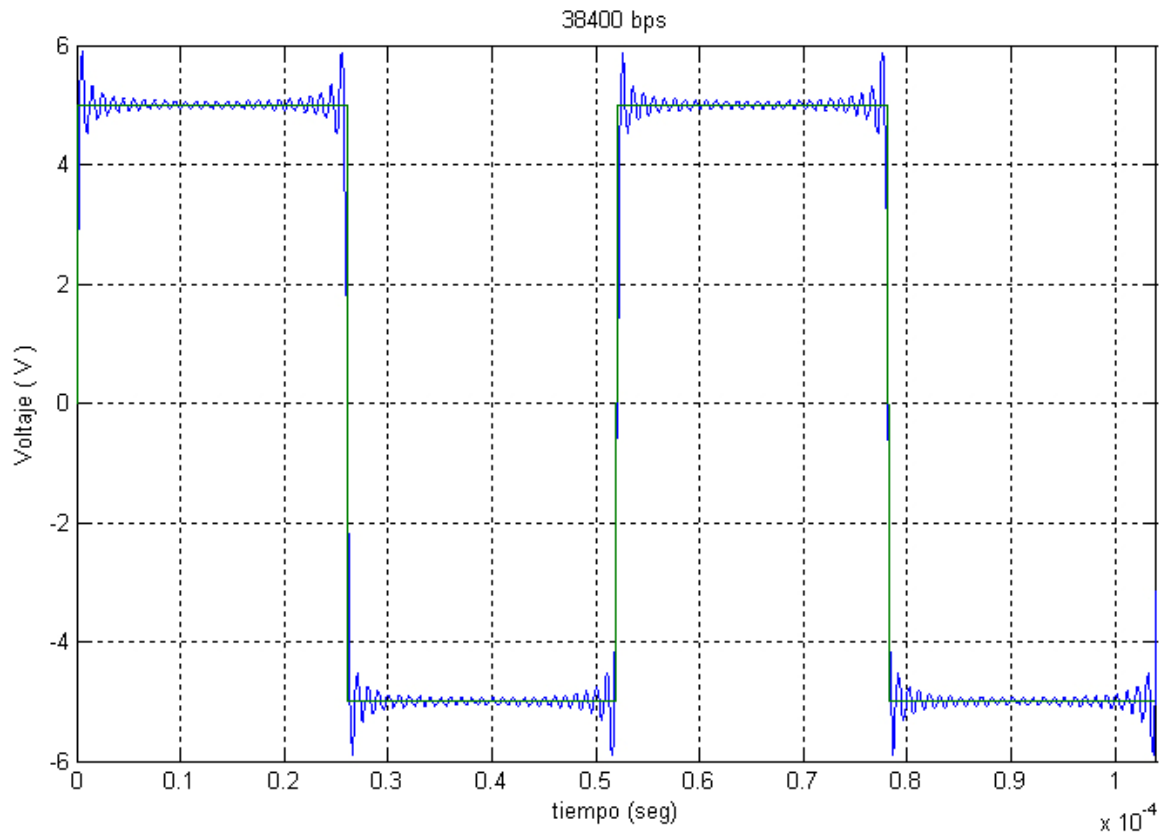


Figura 4.10 Comparación entre una señal digital y su descomposición de Fourier

Estos picos en la serie de Fourier de la función cuadrada nunca desaparecen reciben el nombre de fenómeno de Gibbs, nombrado así por el físico Americano Josiah Willard Gibbs, y ocurren en señales discontinuas, siempre que la señal tenga fuertes saltos como en este caso.

A pesar de la existencia del fenómeno de Gibbs, tras realizar el montaje experimental del circuito propuesto, la comunicación entre la tarjeta Xbee y la controladora de servos se produce sin problemas como puede deducirse del correcto funcionamiento del robot.

4.3 Electrónica del robot

4.3.1. Alimentación de la electrónica

Por último hay que señalar que para la elección de la alimentación del robot existen diversas opciones entre las que destacan las pilas tradicionales y los distintos tipos de baterías existentes.

A continuación se muestra una tabla comparativa entre las pilas comerciales más comunes:

Designación:	CR1025	LR1	LR03	LR6	6LR61
Imagen:					
Clasificación:	Moneda Litio	Alcalina	Alcalina	Alcalina	Alcalina
Sistema Químico:	Litio / Dióxido de Manganeso (Li/MnO ₂)	Zinc-Dióxido de Manganeso (Zn/MnO ₂)	Zinc- Dióxido de Manganeso (Zn/MnO ₂)	Zinc- Dióxido de Manganeso (Zn/MnO ₂)	Zinc- Dióxido de Manganeso (Zn/MnO ₂)
Voltaje Nominal:	3,0 V	1,5 V	1,5 V	1,5 V	9,0 V
Carga Eléctrica	30 mAh	1000 mAh	1200 mAh	2900 mAh	600 mAh
Peso Típico	0,7 g	9,0 g	11,5 g	23,0 g	45,6 g
Volumen Típico:	0,2 cm ³	3,3 cm ³	3,8 cm ³	8,1 cm ³	21,1 cm ³
Precio:	0,50 €	1,00 €	0,40 €	0,60 €	1,75 €

Tabla 4.2 Comparativa entre pilas comerciales

En cuanto a baterías recargables se refiere, existen gran variedad de alternativas en el mercado: Baterías de plomo-ácido, baterías de níquel-hidruro metálico (Ni-Mh) baterías de iones de litio (Li-ion) y baterías de polímero de litio (LiPo).

Entre esta variedad las opciones que se consideraron como más adecuadas para su utilización como fuente de energía fueron las de LiPo y las de Li-ion debido a su bajo costo y peso, así como por su alta densidad de energía.

4.3 Electrónica del robot

Una tabla comparativa entre las dos opciones consideradas se muestra a continuación:

Designación:	E603048	H083448
Imagen:		
Clasificación:	Batería Recargable de litio - polímero	Batería Recargable Prismática de ion-litio
Sistema Químico:	Litio - Polímero	Ion - Litio
Voltaje Nominal:	3,7 V	3,7 V
Carga Eléctrica	900 mAh	1250 mAh
Peso Típico	16,8 g	37 g
Dimensiones Típicas:	48x30x6 mm	48x34x8
Precio:	7,90 € + Gastos de Envío	5 € + Gastos de envío

Tabla 4.3 Comparativa entre baterías recargables

Finalmente, para la alimentación del robot se optó por la elección de pilas alcalinas, debido a la necesidad que presentan las baterías de circuitos electrónicos de control de carga y descarga, la necesidad de cargadores especiales, y el elevado tiempo de carga necesario para el funcionamiento del robot, lo cual es un inconveniente para un prototipo en el que se van a realizar múltiples pruebas.

Además, como ventaja adicional, las pilas son fáciles de substituir ya que se encuentran fácilmente en cualquier centro comercial. Otra ventaja es su facilidad de uso.

Para la elección del modelo concreto de pilas a utilizar la principal característica que se tuvo en cuenta fue el tamaño, ya que el espacio disponible en el robot es reducido. Las pilas de gran tamaño aumentan, además, el peso del robot dificultando su movimiento.

Por último también se tuvo en cuenta el precio, ya que alimentar el robot con pilas de botón resultaría demasiado caro debido a su escasa capacidad de almacenamiento de energía.

Estos criterios solamente nos ofrecen dos opciones posibles, pilas de 9 V tipo, y pilas tipo AAA de 1,5 V. Entre estos dos modelos se escogieron estas últimas, ya que para el uso de las pilas de 9 V habría que implementar un sistema de división de voltaje para suministrar con una misma fuente de energía 3 V a la electrónica de las Xbee por un lado y 6 V a los servomotores. Esta opción resulta compleja ya que los servomotores en su funcionamiento producen gran cantidad de ruido electrónico que afecta al funcionamiento de las Xbee.

4.3 Electrónica del robot

Puesto que se optó por el uso de pilas de tipo AAA, resulta necesaria la fijación mecánica de las baterías al robot. Para esta tarea se utilizaron portapilas modelo bh443d que permiten una fácil sujeción, además de facilitar el acceso a sus bornes de conexión para soldar las mismas.

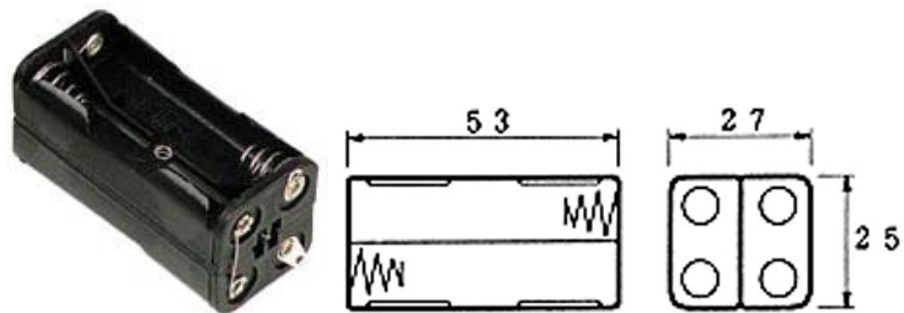


Figura 4.11 Portapilas bh443d y características dimensionales

5

ESPECIFICACIONES TÉCNICAS Y SIMULACIONES

5.1. Especificaciones técnicas

5.2. Simulaciones desarrolladas

En este capítulo se detallan las especificaciones técnicas del prototipo fabricado y se exponen los resultados obtenidos mediante simulación de los distintos algoritmos de locomoción implementados en los programas de control.

5.1 Especificaciones técnicas

En este primer apartado se detallan las características del robot ordenándolas con una estructura similar a la presentada en el capítulo dedicado al estado del arte

Especificaciones técnicas:

- Diseño modular fácilmente modificable.
- Sección transversal: 7,2x7,2 cm.
- Longitud total del robot: 68 cm.
- Peso aproximado: 1,35 Kg.
- Número de grados de libertad por módulo: 1.

- Actuadores: Servomotores Futaba 3003 (Par: 3,2 Kg-cm Velocidad: 0,23 seg/60°).
- Número de GDL totales: 8.
- Alimentación: Pilas modelo AAA.
- Control: Desde PC mediante comunicación inalámbrica.
- Sensores: Cámara y micrófono en la cabeza del robot.
- Material de fabricación: Cartón Prensado.
- Velocidad de Desplazamiento: 4 cm/s.
- Máxima inclinación abordable: 30°.
- Fuerza de arrastre en una superficie horizontal: 25 N.
- Autonomía: 40 minutos.
- Radio de alcance: 30 m.
- Precisión de la orientación de la cámara: $\pm 2^\circ$.

Para el cálculo de la autonomía del robot hay que tener en cuenta que los servomotores no consumen la misma cantidad de corriente durante todo el tiempo y aunque en la ficha técnica de los servomotores Futaba S3003 se indique un consumo de 7,2 mA (Anexo D), esto es para el modo inactivo. Para un funcionamiento sin carga consumen 50 mA, y para carga nominal consumen 200 mA, en realidad el consumo de los servomotores utilizados será superior, ya que la carga a la que están sometidos está más próxima al funcionamiento bloqueado, pero se supondrán 200 mA por simplicidad.

Si con este valor de consumo observamos las gráficas de tiempo de servicio en función de la descarga de las pilas utilizadas se puede estimar la autonomía del robot.

Como puede verse en la Figura 5.1 el tiempo de servicio se encuentra entorno a 1 hora, pero tras la realización de numerosas pruebas se llegó a la conclusión que la autonomía real se encuentra próxima a los 40 minutos, ya que tras media hora de utilización el voltaje de las baterías disminuye por debajo del límite admisible por los servomotores.

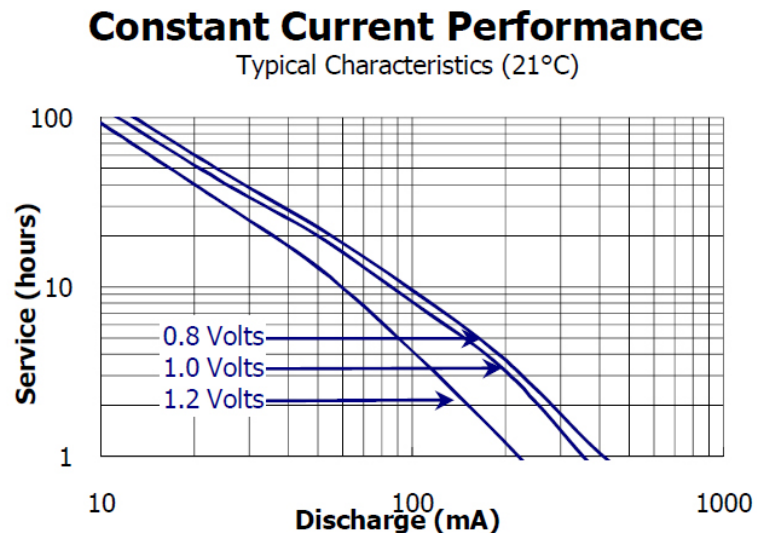


Figura 5.1 Tiempo de servicio en función de la descarga

Otros datos de interés:

Para el control del robot se necesitan 1 persona manejando el programa de control principal. El costo de fabricación del robot es el menor posible utilizando componentes comerciales, encontrándose entorno a los 170 €.

Cantidad	Descripción	Precio Unitario	Subtotal
8	Servomotores Futaba s3003	11	88
2	Módulos Xbee	16	32
1	Pololu Micro Serial Servo	14	14
1	Xbee USB Adapter	17	17
1	Mini cámara inalámbrica	20	20
Total:			171

Tabla 5.1 Presupuesto y costo de fabricación

5.2 Simulaciones desarrolladas

En este apartado se exponen los resultados obtenidos en las simulaciones de los distintos algoritmos de locomoción, a partir de los cuales puede obtenerse una medida de la precisión que se conseguirá con el robot a la hora de seguir una trayectoria determinada o alcanzar una posición concreta.

Para la realización de las simulaciones que se presentan en este apartado se utilizó el software de simulación V-REP (Virtual Robot Experimentation Platform), dicho simulador se encuentra enfocado principalmente a la robótica y puede utilizarse para simular, probar, evaluar o comunicarse con un sistema robótico completo.

Estas características lo hacen mucho más útil que otros simuladores como la combinación del programa “*Google SketchUp*” con su motor de física “*SketchyPhysics*”, que solamente puede emplearse para pruebas cualitativas de robots sencillos.

Algunas de las principales características de V-R EP son las siguientes:

- Detección de colisiones, permitiendo control de interferencia entre geometrías de forma rápida y sencilla
- Capacidad de cálculo de mínima distancia, permitiendo el cálculo para cualquier tipo de geometría
- Algoritmos especializados dedicados a cálculos de cinemática directa e inversa
- Algoritmos de planificación de trayectorias tanto para entidades holonómicas como no holonómicas
- Módulo dedicado al cálculo de la dinámica de sistemas físicos que permite simular interacciones entre objetos de forma realista, soportando los motores físicos *Bullet* y *ODE*.
- Simulación de una amplia Variedad de sensores como son: sensores de proximidad, par, contacto y cámaras de vídeo, además de capacidad de procesamiento de imagen
- Simulación de operaciones de corte de materiales o de pintura.
- Control distribuido, lo que permite tener un número ilimitado de programas ejecutándose en paralelo

Las razones principales para la utilización de este nuevo software de simulación son varias, entre las que destacan:

- Su versión de evaluación es completamente funcional y gratuita
- Se ha empleado con éxito en la evaluación de otras plataformas robóticas reales como el robot ACM-R5 del profesor Shigeo Hirose.
- Se comprobó la existencia de una elevada concordancia entre los resultados experimentales y las simulaciones

Las características de las simulaciones, así como las suposiciones empleadas se detallan a continuación:

- Los cuerpos rígidos que se simulan tienen la misma forma que el diseño real (se tienen en cuenta tanto cuerpos cóncavos como convexos)
- El tiempo de muestreo en la simulación es de 0,05 s
- El motor de física empleado es el proporcionado por *Bullet*

- Los coeficientes de fricción entre el robot y el suelo son lo suficientemente elevados para que no exista deslizamiento
- Los motores se controlan mediante un control en posición PID con los parámetros que el simulador ofrece por defecto, por lo que se comportan como cajas negras al igual que los servomotores Futaba empleados
- Los motores se encuentran configurados de forma que su velocidad máxima y su momento angular máximo les permitan desplazamientos instantáneos a cualquier posición angular sin importar la carga que soporten.

Los resultados obtenidos en las distintas simulaciones son los siguientes:

5.2.1. Configuración de módulos alineados

Con la configuración de módulos alineados se consigue desplazamiento en una dimensión lo cual resulta apropiado para la inspección del interior de conductos que no presenten secciones acodadas. La evolución de la posición con el tiempo puede verse en la Figura 5.3 y la trayectoria recorrida en la Figura 5.4.

Conviene recordar que con esta configuración de módulos se consigue el movimiento de oruga ya definido al final del capítulo 3.

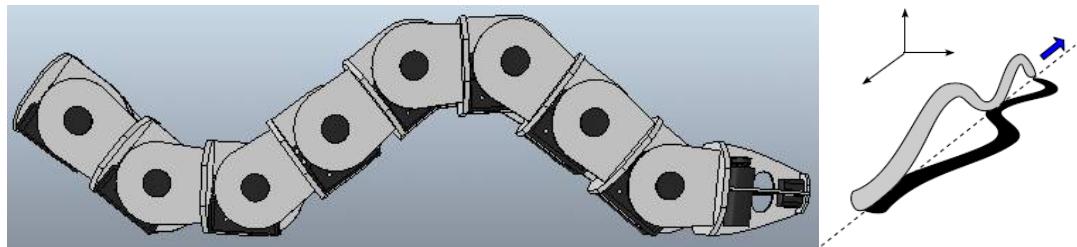


Figura 5.2 Configuración de módulos alineados y movimiento de oruga

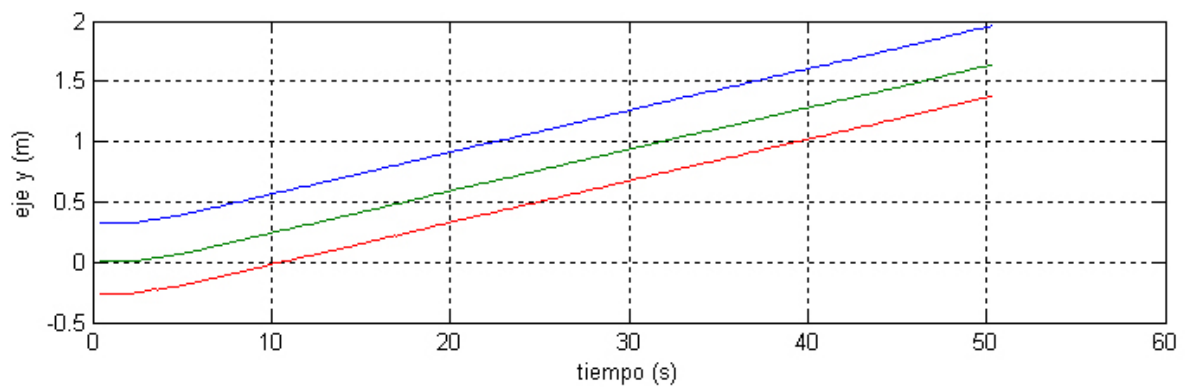
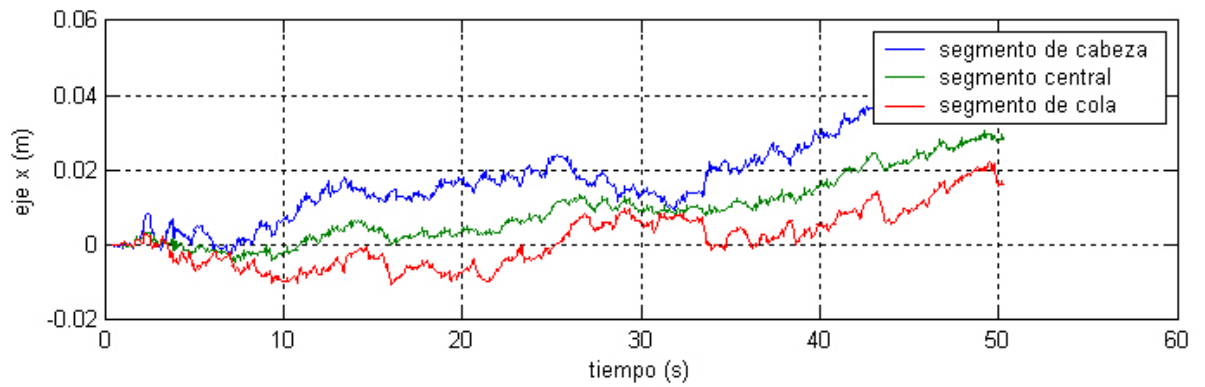


Figura 5.3 Evolución de la posición con el tiempo

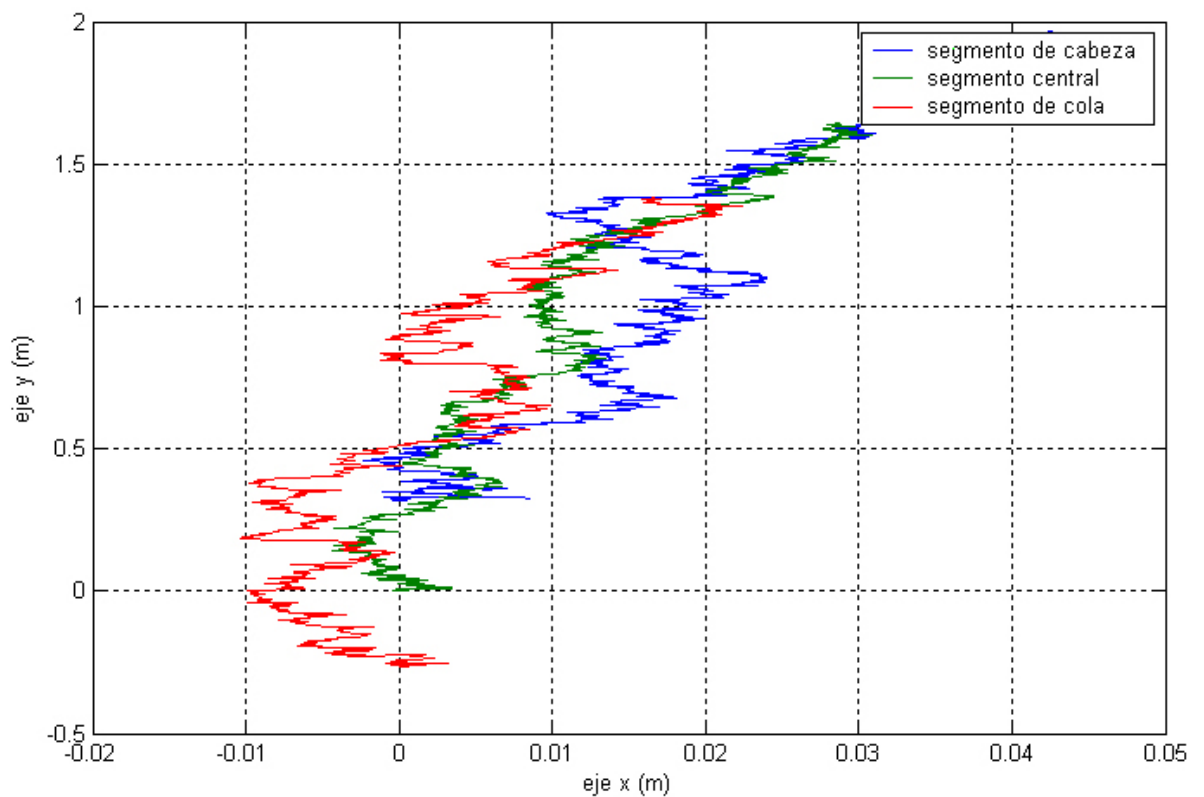


Figura 5.4 Trayectoria recorrida

5.2.2. Configuración de módulos desfasados 90°

Con esta configuración existen diversos mecanismos de locomoción como se indicó en el al final del capítulo 3.

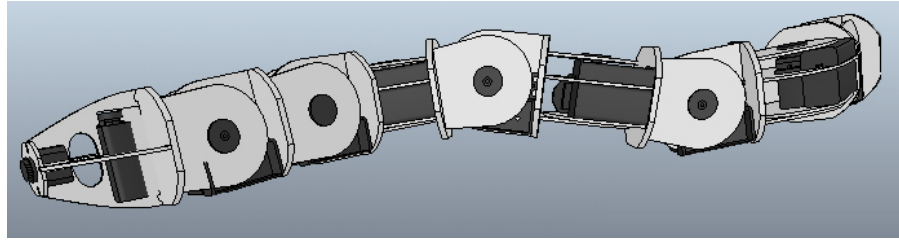


Figura 5.5 Configuración de módulos desfasados 90°

5.2.2.1. Desplazamiento en línea recta

Este modo de desplazamiento es análogo a la forma de desplazamiento que se produce con los módulos alineados, pero al ser menor el número de servomotores útiles resulta mucho menos eficaz. La evolución de la posición con el tiempo puede verse en la Figura 5.6 y la trayectoria recorrida en la Figura 5.7

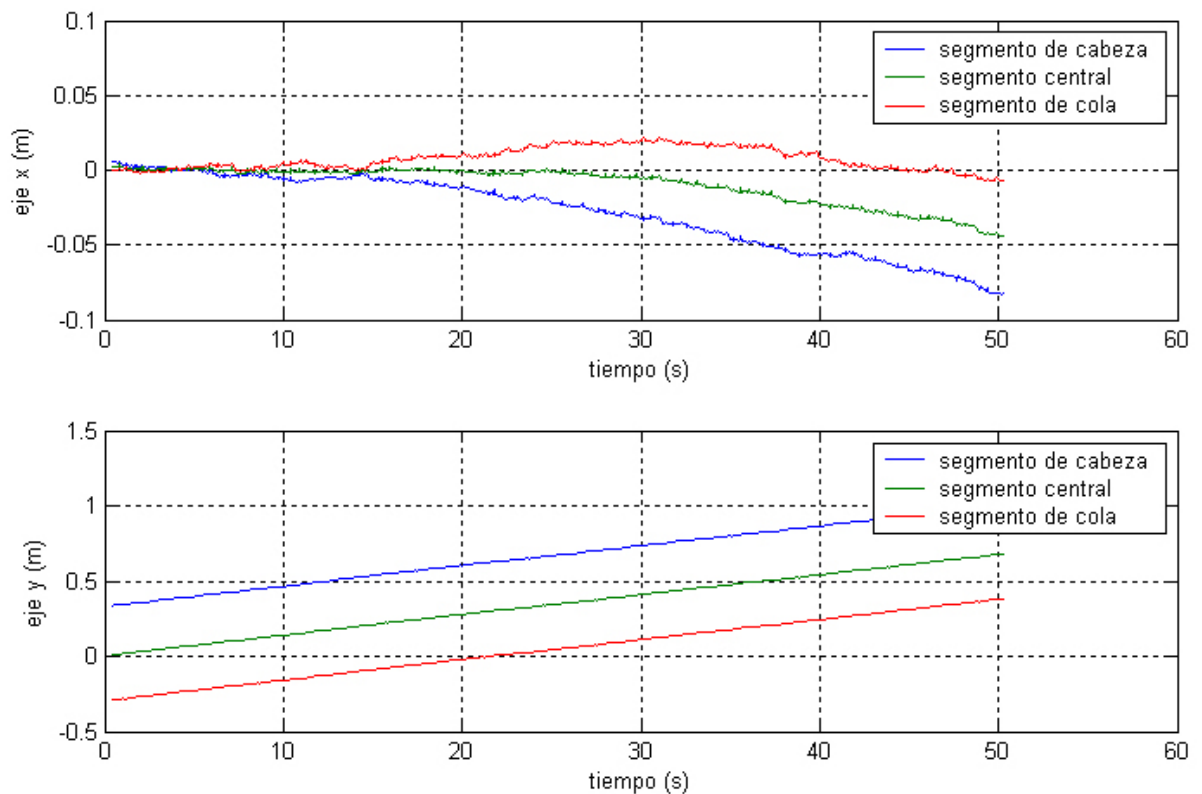


Figura 5.6 Evolución de la posición con el tiempo

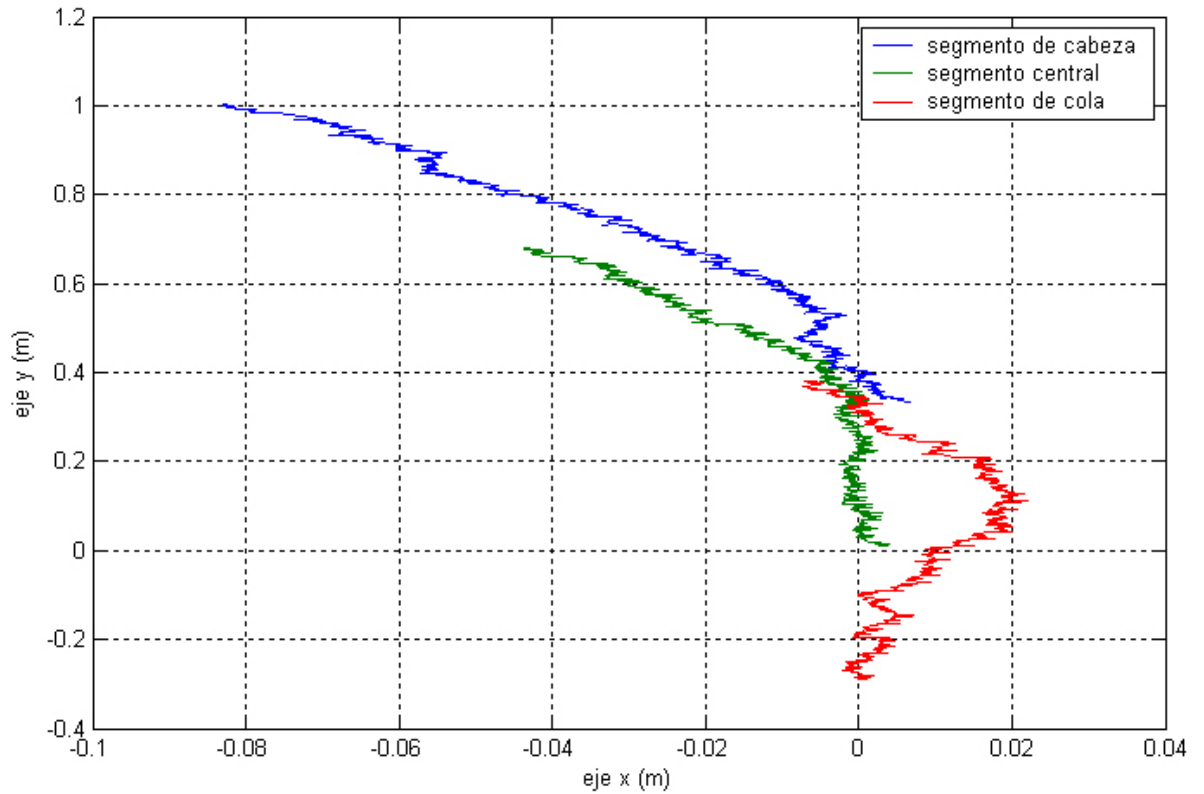


Figura 5.7 Trayectoria recorrida

5.2.2.2. Movimiento Lateral

Este modo de desplazamiento es el que utilizan las serpientes para desplazarse largas distancias en terrenos arenosos y se caracteriza por su gran economía energética, ya que la serpiente solamente debe elevar una parte pequeña del cuerpo cada vez.

El esquema de este movimiento es el que se muestra en la Figura 5.8.

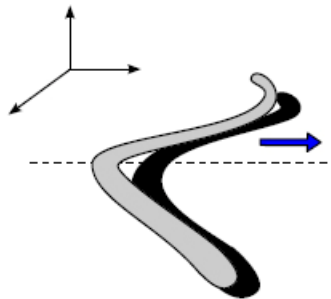


Figura 5.8 Desplazamiento lateral

La evolución de la posición con el tiempo puede verse en la Figura 5.9 y la trayectoria recorrida en la Figura 5.10.

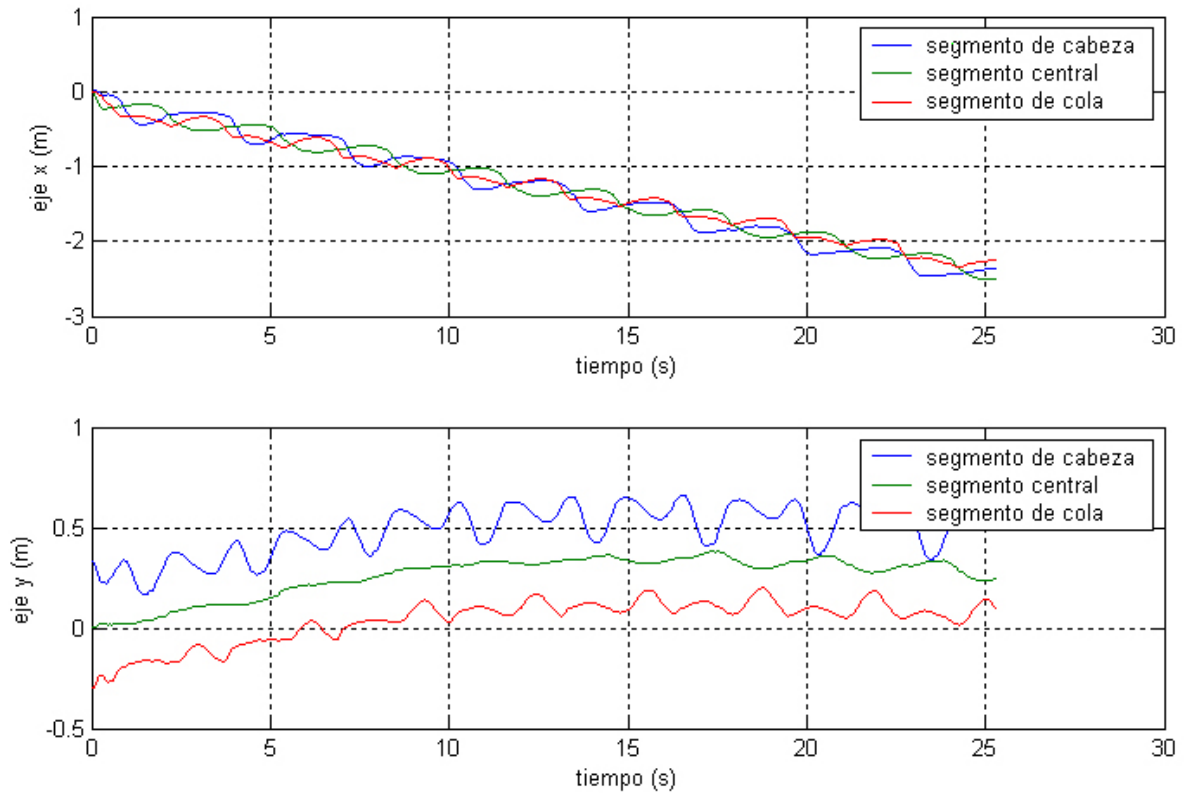


Figura 5.9 Evolución de la posición con el tiempo

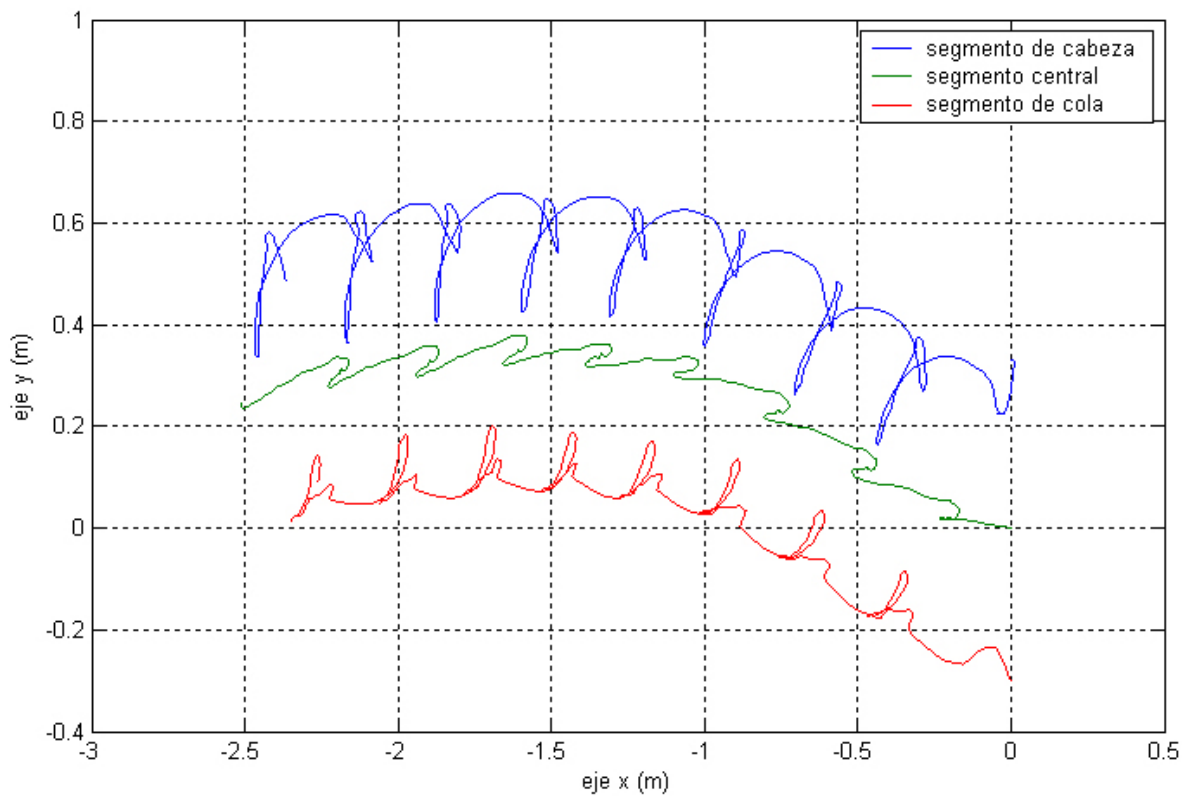


Figura 5.10 Trayectoria recorrida

A la vista de los resultados obtenidos puede comprobarse que este método de desplazamiento es idóneo para viajar campo a través realizando tareas de reconocimiento, puesto que recorre mayor distancia que los dos mostrados anteriormente en menor tiempo. Además, como se observa en la figura 5.11, es el que menos tiempo mantiene partes del cuerpo elevadas y a menor altura.

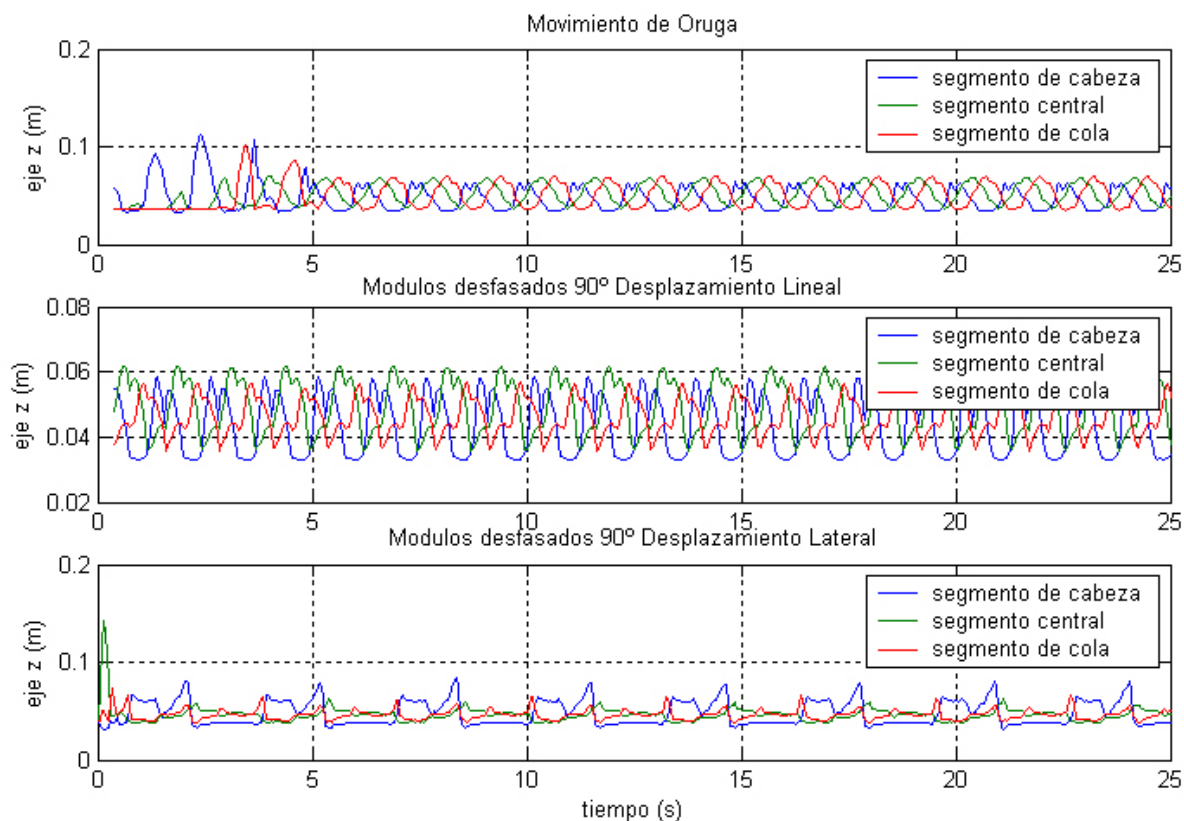


Figura 5.11 Comparativa entre movimientos

5.2.2.3. Movimiento en Círculo

Este modo de desplazamiento se utiliza principalmente para variar la dirección de desplazamiento con radios de curvatura que pueden controlarse, aunque su eficacia es limitada.

El esquema de este movimiento es el que se muestra en la Figura 5.12.

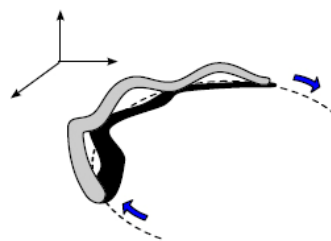


Figura 5.12 Movimiento en círculo

La evolución de la posición con el tiempo puede verse en la Figura 5.13 y la trayectoria recorrida en la Figura 5.14.

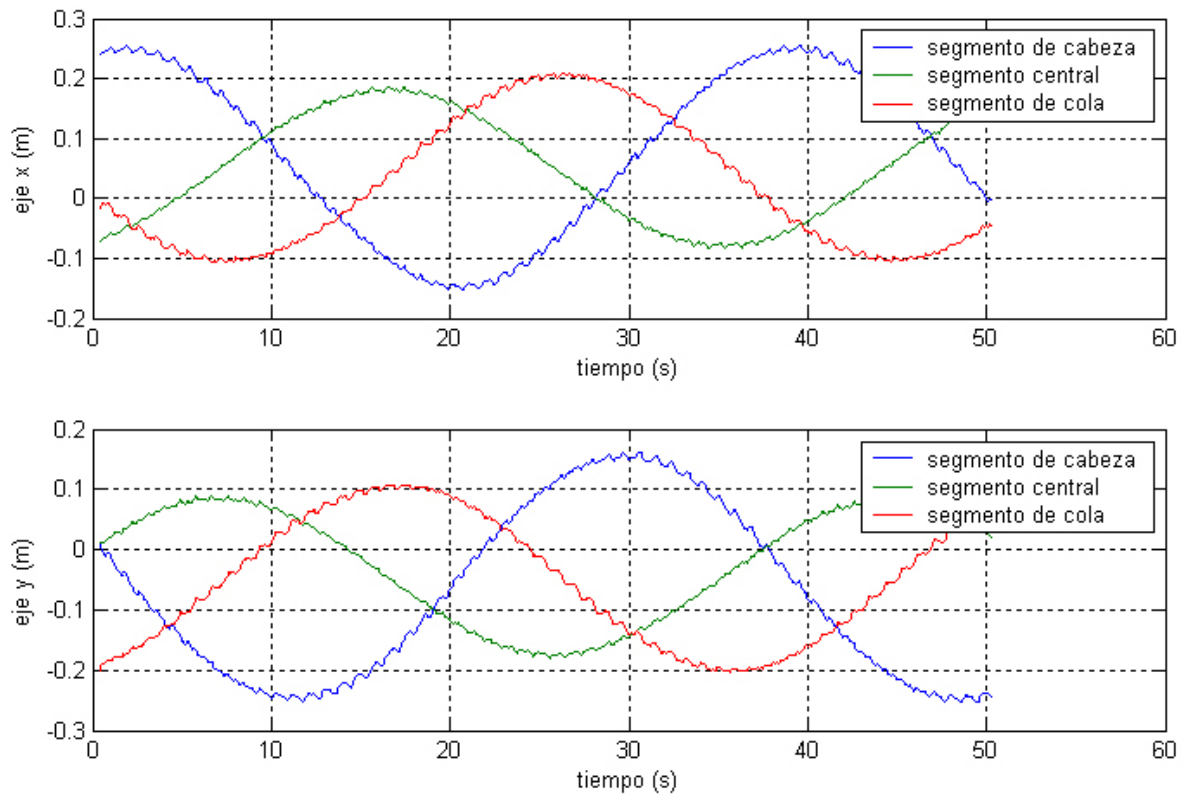


Figura 5.13 Evolución de la posición con el tiempo

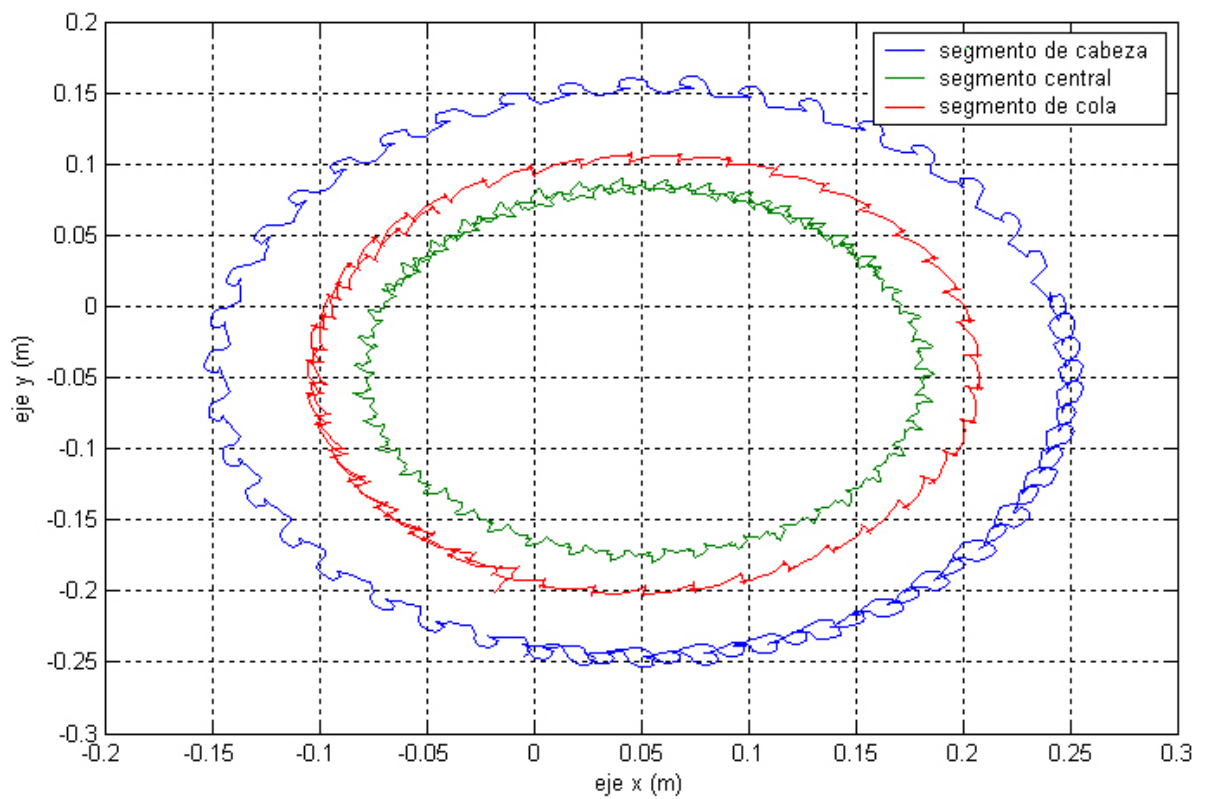


Figura 5.14 Trayectoria Recorrida

5.2.2.4. Movimiento de Rotación

Este modo de desplazamiento se utiliza para variar la dirección de desplazamiento del robot con un radio de curvatura igual a cero, resulta más eficaz que el movimiento en círculo, pero como contrapartida presenta un movimiento más caótico y requiere de mayor espacio disponible para ejecutar la maniobra.

El esquema de este movimiento es el que se muestra en la Figura 5.15.

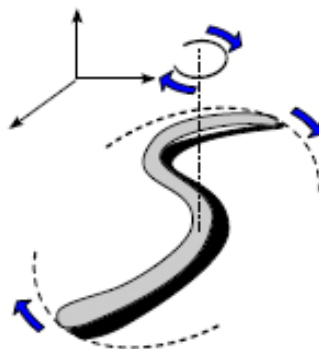


Figura 5.15 Movimiento de rotación

La evolución de la posición con el tiempo puede verse en la Figura 5.16 y la trayectoria recorrida en la Figura 5.17.

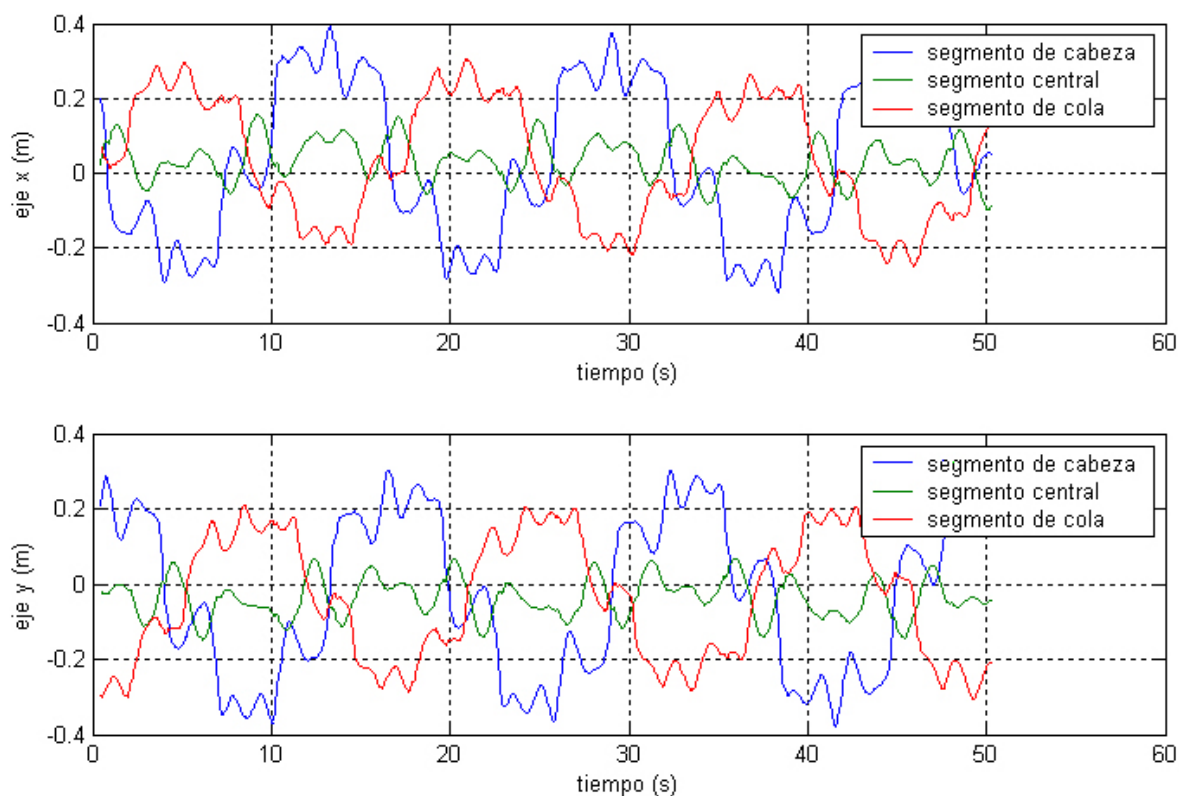


Figura 5.16 Evolución de la posición con el tiempo

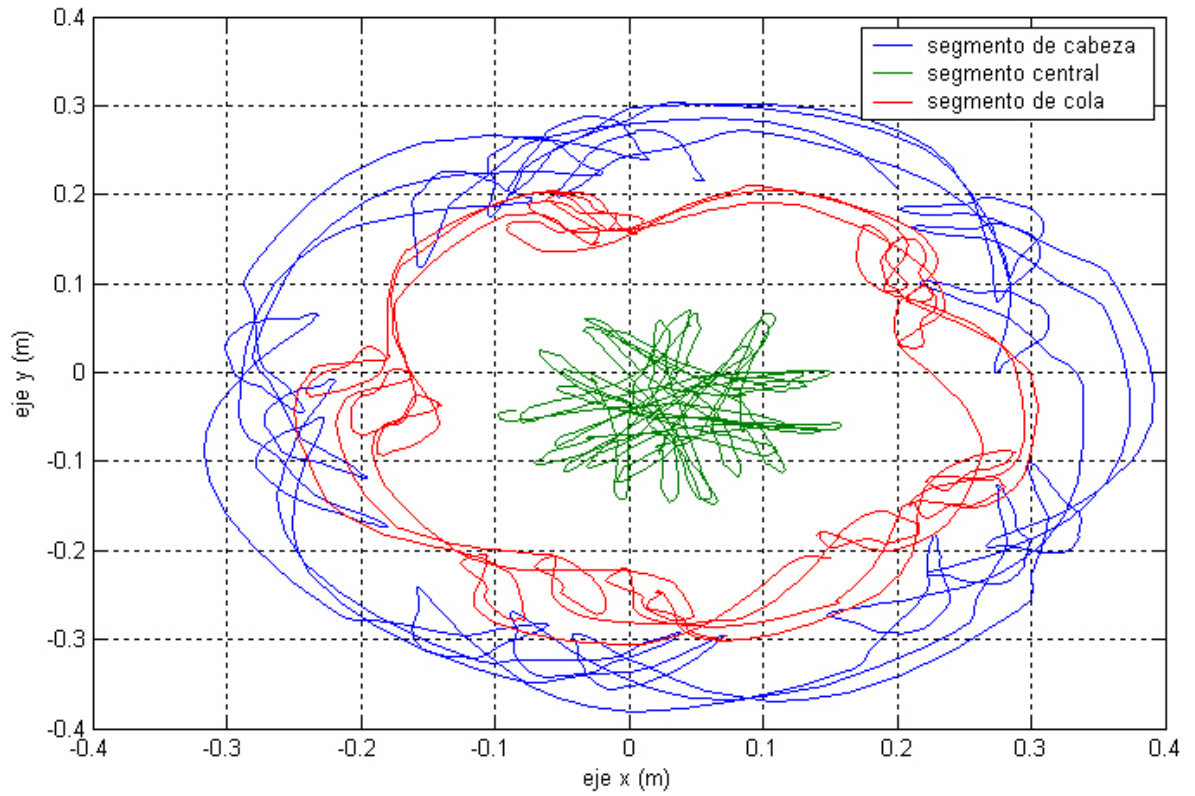


Figura 5.17 Trayectoria Recorrida

De nuevo la mayor eficacia de este método con respecto a las distintas alternativas puede basarse en la altura y el tiempo que permanecen elevados los distintos módulos del robot, además de que en el mismo tiempo el método de cambio de dirección ha conseguido realizar el triple de vueltas que el movimiento en círculo. La comparativa de alturas y tiempos de elevación puede verse en la Figura 5.18.

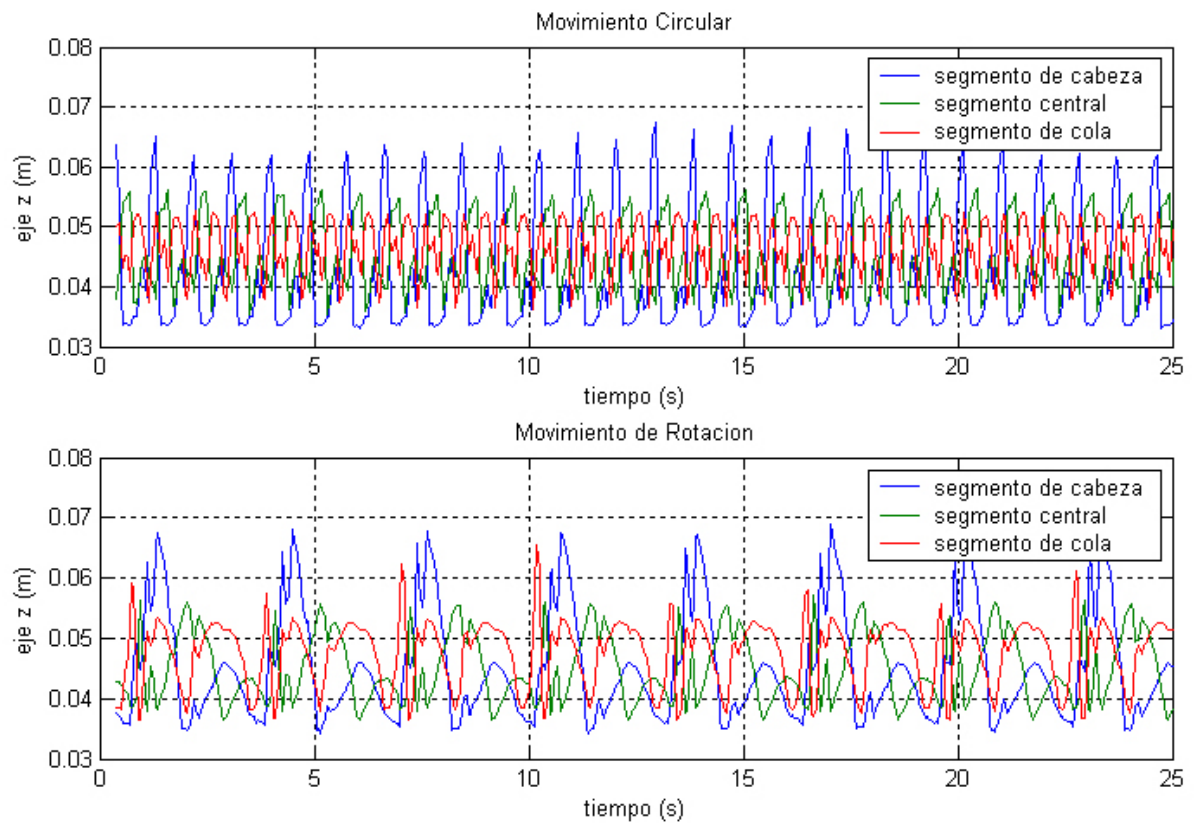


Figura 5.18 Comparativa entre métodos de cambio de trayectoria

6

EJEMPLOS DE APLICACIÓN

6.1. Tareas de reconocimiento

6.2. Tareas de inspección

En este capítulo se realizará una descripción de los experimentos realizados con la intención de conseguir aplicar el robot a tareas de tipo industrial, así como la demostración práctica que se realizó con motivo de la celebración de la 1ª Semana de la Robótica y Automática de la Escuela Técnica Superior de Ingenieros Industriales de Ciudad Real.

En primer lugar es necesario detallar que se entiende como tareas de inspección y que características son las propias de las tareas de reconocimiento.

En este documento se considera que las tareas de inspección son aquellas que se limitan a la observación del estado de conductos de canalización o tuberías desde su interior, además se considerará que la mayor parte de los sistemas de tuberías se encontrarán formados por tramos rectilíneos siendo las secciones acodadas, secciones especiales que requerirían un trato diferente del necesario para el resto de la conducción.

Mientras que las tareas de reconocimiento implican el desplazamiento del robot sobre un entorno bidimensional, como puede ser el suelo de un edificio o cualquier tipo de terreno moderadamente llano que se encuentre en campo abierto. Esta definición se encuentra orientada a aplicaciones que puedan resultar útiles a las fuerzas y cuerpos de seguridad debido a su potencial para situar ojos y oídos en terreno hostil o accidentado de forma encubierta.

6.1 Tareas de reconocimiento

En el caso de las tareas de reconocimiento, se realizaron diversas demostraciones en distintos momentos del desarrollo del robot, destacando la 1ª Semana de la Robótica y Automática de la Escuela Técnica Superior de Ingenieros Industriales de Ciudad Real, realizada en Octubre de 2010.

En ella se demostró que la configuración más eficaz para el desplazamiento por terreno abierto es aquella en la que los módulos se encuentran girados 90º unos respecto a otros.

Esta configuración es la más conveniente, ya que permite el desplazamiento y la orientación del robot de cualquier forma que se desee en el plano sobre el que nos desplazamos.

En la Figura 6.1 se muestra una imagen perteneciente al vídeo de demostración realizado para dicho acto (el video se encuentra incluido en la documentación en formato digital adjuntada con la memoria del proyecto)



Figura 6.1 Fotograma del vídeo de la semana de la robótica en la ETSII

6.2 Tareas de inspección

La tarea de inspección de tuberías puede considerarse la que tiene una aplicación industrial más rentable en un futuro inmediato debido a la cercanía a la universidad de la empresa REPSOL.

Tras realizar varios experimentos con distintas configuraciones del robot, se comprobó que para tuberías de pequeño diámetro la configuración más eficaz para el desplazamiento es aquella en la que los módulos que conforman el robot se encuentran dispuestos con la misma orientación.

Esta configuración permite un desplazamiento más rápido en tuberías de pequeña sección e incluso abordar tramos de tuberías con cierta pendiente, ya que el robot gana fuerza de tracción al tener la posibilidad de contactar con lados opuestos del interior de la tubería con varias partes del cuerpo.

Abordar la inspección de tuberías verticales no fue posible con nuestro prototipo ya que la escasa fuerza motriz de los servomotores unida al elevado peso del sistema de alimentación reducía notablemente su capacidad de adherencia a la tubería. No obstante, esto es fácilmente solucionable con la incorporación de un sistema de baterías y servomotores de gama alta como se verá en el capítulo dedicado a conclusiones.

Una imagen perteneciente al vídeo grabado durante los experimentos de inspección puede verse en la Figura 6.2 (el video se encuentra incluido en la documentación en formato digital adjuntado con la memoria del proyecto)

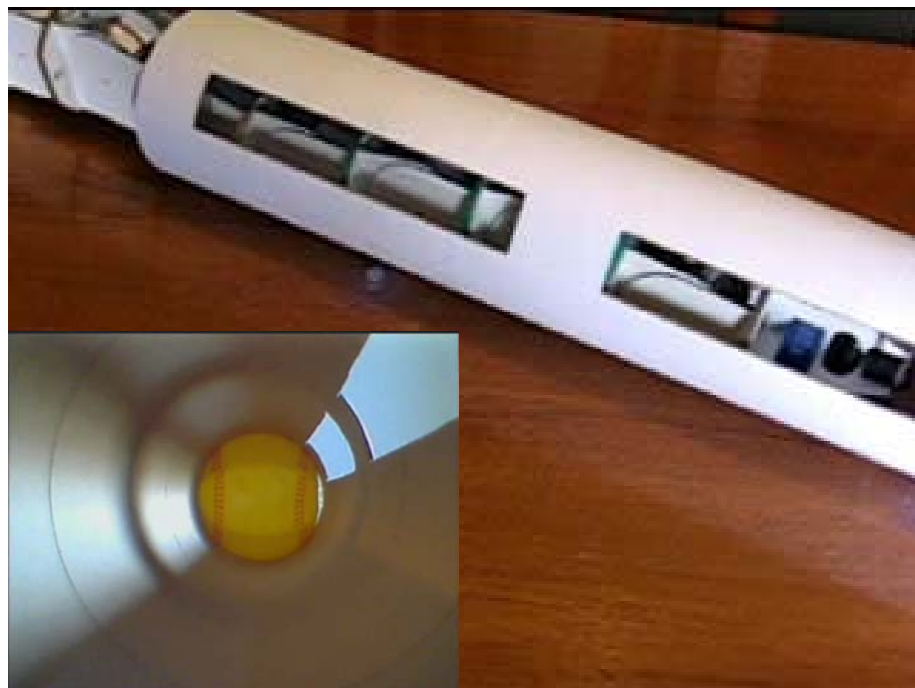


Figura 6.2 Fotograma del vídeo de inspección de tuberías

6.2. Tareas de inspección

Como puede verse en el video del experimento, el robot es capaz de descubrir atascos en las tuberías e incluso solucionarlos, aunque en un principio su diseño se orientase sólo a la inspección y no al desatasco.

7

CONCLUSIONES

7.1. Conclusiones

7.2. Problemas encontrados y propuestas de mejora

7.3. Colaboración en el software de robótica V-REP

En este capítulo se exponen las conclusiones a las que puede llegarse a partir de los resultados obtenidos tanto en las simulaciones como en los experimentos realizados con el prototipo.

Así mismo, se detallarán los aspectos más problemáticos y las distintas propuestas de mejora.

Por último, se realizará una breve descripción de un nuevo simulador dedicado a la robótica que el autor pudo probar en los últimos meses de desarrollo del robot y en el cual ha podido participar activamente incluyendo su robot como parte del catálogo de robots disponibles para su experimentación.

7.1 Conclusiones

En primer lugar, es necesario poner de manifiesto que si se compara con el resto de robots ápodos existentes en el marco actual de universidades, el robot fruto de este proyecto supone un éxito debido a varias razones.

La principal es el reducido coste de fabricación del robot, ya que al utilizar servomotores propios del mundo del aeromodelismo y una electrónica de control lo más sencilla posible, su precio total de fabricación se encuentra entorno a los 170 €.

Si se compara con el presupuesto destinado a algunos prototipos como los mostrados anteriormente en la sección correspondiente al estado del arte, vemos que éstos superan con facilidad cantidades diez veces mayores a los 170 € de nuestro robot.

Sin olvidar que dichos prototipos cuentan con equipos completos de desarrollo compuestos por varias personas con años de experiencia en el campo de la robótica.

Por otra parte hay que destacar que se han logrado cumplir todos los objetivos propuestos con la elaboración del robot. Se ha logrado realizar un diseño modular y flexible, se ha desarrollado una interfaz de control simple y amigable y, sobre todo, se ha conseguido que el robot sea completamente autónomo y pueda dirigirse sin ayuda de cables.

Este último punto también marca una diferencia importante con respecto al resto de prototipos existentes, pues a la fecha de la redacción de este proyecto, salvo contadas excepciones, la gran mayoría de prototipos necesitan ser operados mediante un umbilical que les transmita energía y señales de control.

Incluso algunos de los que se encuentran dotados de cierta autonomía no pueden controlarse más allá de la ejecución de unas pocas secuencias de movimiento preprogramadas, lo que los convierte en inviables para su aplicación a la inspección de entornos desconocidos o cambiantes.

Finalmente, hay que destacar que, pese a las discrepancias obtenidas entre los resultados teóricos obtenidos en los simuladores y los resultados obtenidos mediante experimentación, fruto principalmente del reducido par de los servomotores empleados, se ha conseguido experimentar con éxito diversos algoritmos de locomoción y discernir cuáles de éstos resultarían más útiles en dos de las posibles tareas que podría realizar el robot (locomoción serpentina para tareas de reconocimiento y locomoción con módulos alineados para tareas de inspección).

Además se han podido realizar múltiples demostraciones del funcionamiento del robot, siendo la más importante la realizada para la 1ª Semana de la Robótica y Automática de la Escuela Técnica Superior de Ingenieros Industriales de Ciudad Real.

7.2 Problemas encontrados y propuestas de mejora

En cuanto al mecanismo de locomoción, se comprobó que el más eficaz, debido a su bajo coste computacional y posibilidad de gran flexibilidad, lo que facilita el desarrollo de futuras aplicaciones de control automático, es el producido mediante generadores centrales de patrones (GCP), en las configuraciones de módulos alternados 90° para el desplazamiento en tierra y módulos alineados para el desplazamiento por sistemas de tuberías de reducido diámetro.

No obstante los principales problemas encontrados se ponen de manifiesto cuando se comparan los movimientos realizados por el robot en las simulaciones y los movimientos realizados en los experimentos reales, ya que los movimientos realizados por el robot son menos ágiles que los que se pueden observar en las simulaciones. Esto es debido a los siguientes problemas:

- Los servomotores utilizados carecen del par necesario para controlar los segmentos de la serpiente de manera precisa. Esta deficiencia, asociada al elevado peso del sistema de alimentación, impide la realización de actividades como por ejemplo la inspección de tuberías verticales.

La solución a este problema es sencilla, bastaría con substituir las pilas por un sistema de alimentación mediante baterías de LiPo que presentan un peso inferior, además de reemplazar los servomotores utilizados por otros con un mayor par como los servomotores especialmente diseñados para robótica HSR-8498HB creados por la compañía Hitec que presentan un par máximo de 9 Kg-cm frente a los 4,1 Kg-cm de los Futaba empleados.

- La longitud del robot es insuficiente para realizar maniobras complejas de locomoción como la estabilización de la cámara mientras se avanza o ascender por tuberías enroscándose a la periferia exterior de las mismas.

La solución de este inconveniente se solucionaría añadiendo más módulos, lo que conferiría un mayor número de grados de libertad al robot y una mayor capacidad de maniobra.

- El material con el que se encuentra confeccionado el robot es demasiado frágil para soportar caídas o funcionamiento en entornos agresivos.

Resultaría conveniente cambiar el material del robot por otro con similares características en cuanto a su reducido peso pero con un mejor comportamiento frente a esfuerzos mecánicos como por ejemplo el aluminio.

- La actual arquitectura de control electrónico necesita la existencia de demasiados cables recorriendo toda la estructura de control, lo que provoca su frecuente ruptura por desgaste.

En un robot de esta naturaleza, en el que el desplazamiento se consigue con el movimiento simultáneo de todas sus articulaciones, resultaría conveniente equipar cada segmento con una controladora de servomotores para que así el número de cables que recorren el robot pueda reducirse a los dos encargados de enviar la señal de control desde la tarjeta Xbee a las distintas controladoras conectadas en serie (conexión tipo daisy chain)

7.3 Colaboración en el software de robótica V-REP

Debido a que el programa se encuentra en constante desarrollo y su autor proporciona continuamente versiones con nuevas funcionalidades se tuvo la oportunidad de colaborar en el proyecto enviando un modelo 3D del robot que ocupa este proyecto, así como una serie de algoritmos de locomoción para evaluación por parte del autor del software.

Recientemente el robot desarrollado ha sido incluido como parte de las bibliotecas Standard de robots con las que cuenta el programa. Pudiendo cualquier usuario que utilice alguna Versión igual o superior a la 2.5.3 comprobar el funcionamiento de dos algoritmos distintos de locomoción en el prototipo de robot serpiente.

Como curiosidad cabe destacar que este robot se encuentra en la categoría de robots móviles situado entre algunos de los robots citados en el estado del arte, así como otros muy conocidos, los cuales se listan a continuación:

- ACM-R5
- Roller-Walker
- Asti (equivalente en el simulador del robot ASIMO)
- DR12
- DR20
- E-Puck
- K-Junior
- Khepera 3
- Kuka Youbot

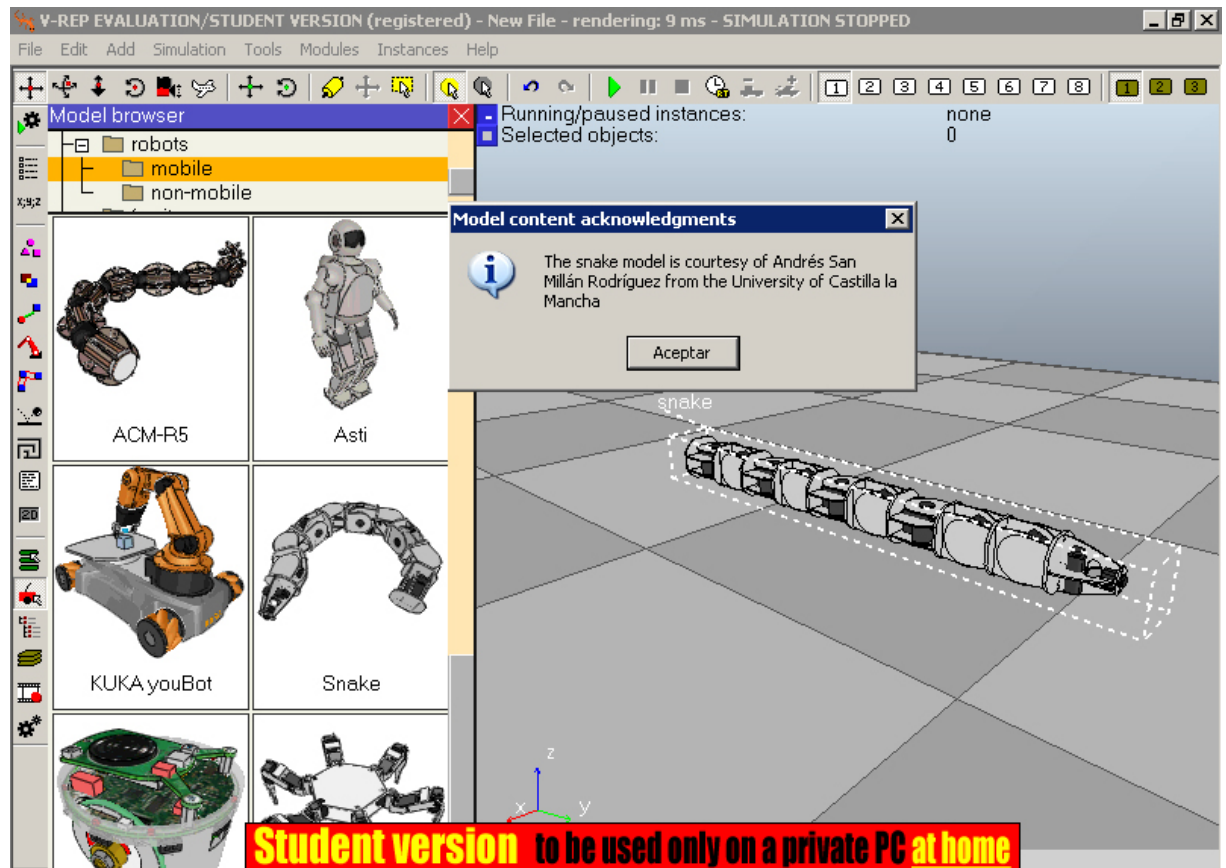


Figura 7.1 Pantalla del simulador V-REP con el robot serpiente incluido en el programa

BIBLIOGRAFÍA

- [1] Dino Accoto, Piero Castrataro, and Paolo Dario. Biomechanical Analysis of Oligochaeta biologically Crawling. (2004)^o *Journal of Theoretical Biology*, 230^o, pp. 49-.
- [2] A. Alonso-Puig. Application of Waves Displacement Algorithms for the Generation of Gaits in an all terrain Hexapod. In *Climbing and Walking Robots*, pp 343–348, September 2005.
- [3] Aksel Andreas Transeth, Nathan van de Wouw, Alexey V. Pavlov, João Pedro Hespanha, Kristin Ytterstad Pettersen. Tracking control for snake robot joints. In *Proceedings of IROS'2007*. pp.3539~3546.
- [4] J. Borenstein, M. Hansen, and H. Nguyen. The OmniTread OT-4 Serpentine Robot for Emergencies and Hazardous Environments. In *Proceeding of the International Joint Topical Meeting: Sharing Solutions for Emergencies and Hazardous Environments*, 12 February 2006.
- [5] L. Chen, Y. Wang, and S. Ma. Studies on lateral rolling locomotion of a snake robot. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 5070– 5074, April 2004.
- [6] G.S Chirjkjian and J.W Burdick. Kinematically optimal hyperredundant manipulator configurations. *IEEE transactions on Robotics and Automation*, 11:794–806, 1995.
- [7] A. Crespi, A. Badertscher, A. Guignard, and A. J. Ijspeert. An amphibious robot capable of snake and lamprey-like locomotion. In *Proceedings of the 35th international symposium on robotics*, 2004.
- [8] A. Crespi, A. Badertscher, and A. Guignard. Swimming and crawling with an amphibious snake robot. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3035–3039, 2005

- [9] A. Degani, A. Shapiro, H. Choset, and M. Mason. A dynamic single actuator verticalvertical climbing robot. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2901–2906, November 2007.
- [10] Dobrolyubov A.I. The mechanism of locomotion of some terrestrial animals by travelling waves of deformation (1986) *Journal of Theoretical Biology*, 119, pp.457-466.
- [11] A. H. Cohen and P.J Holmes. The Nature of the Couplings Between Segmental Oscillators of the Lamprey Spinal Generator for Locomotion: A Mathematical Model. *Journal of Mathematical Biology*, 13:345–369, 1982.
- [12] K. Dowling. *Limbless Locomotion: Learning to Crawl with a Snake Robot*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1997.
- [13] S. Hirose. *Biologically Inspired Robots (Snake-like Locomotor and Manipulator)*. Oxford Science Press, 1993.
- [14] S. Hirose, A. Morishima, and S. Tukagosi. Design of Practical Snake Vehicle: Articulated Body Mobile Robot KR-II. In *Proc. 5th Int. Conf. Advanced Robotics*, volume 1, pp 833–838, 1991.
- [15] J. González-Gómez. Modular Robotics and Locomotion: Application to Limbless Robots Universidad Autónoma de Madrid, November 2008.
- [16] Shugen Ma. Analysis of Creeping Locomotion of a Snake-like Robot. *Int. J. of Advanced Robotics*. 2000
- [17] Shugen Ma, Yoshihiro Ohmameuda and Kousuke Inoue. Dynamic Analysis of 3-dimensional Snake Robots. In *Proc. 2004 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'04)*, pp.767---772, Sendai, Japan 2004.9,
- [18] D. M. Wilson. The central nervous control of flight in a locust. *Journal of Experimental Biology*, 38:471–490, 1961.
- [19] V. Zykov, A. Chan, and H. Lipson. Molecubes: An Open-Source Modular Robotics Kit. In *Proc. of the IEEE Int. Conf. on Intelligents Robots and systems*, 2007.

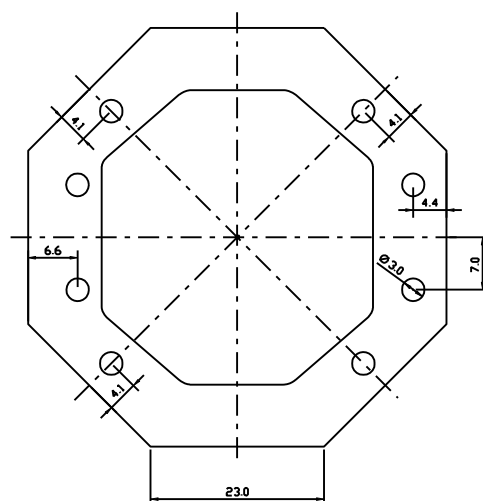
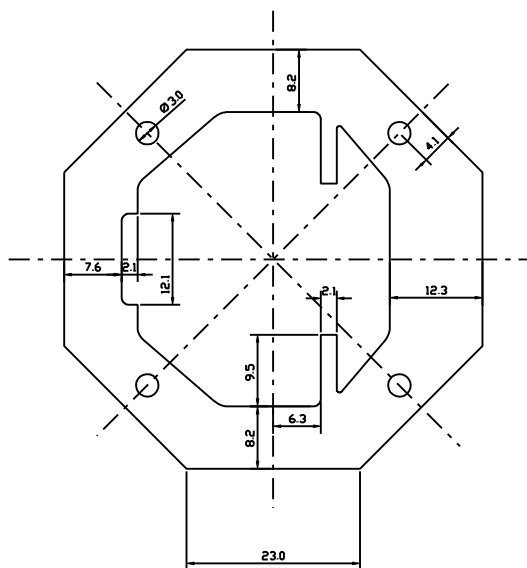
Anexos

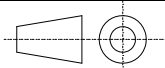
- [A] PLANOS CONSTRUCTIVOS
- [B] DIAGRAMA DE CONEXIONADO DEL ROBOT
- [C] DESCRIPCIÓN DEL SOFTWARE DESARROLLADO
- [D] HARDWARE UTILIZADO
- [E] DESCRIPCIÓN DEL SOFTWARE UTILIZADO

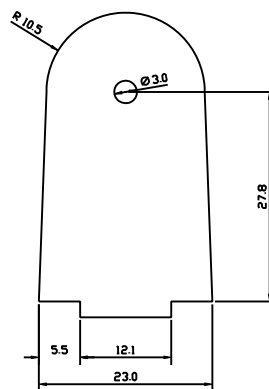
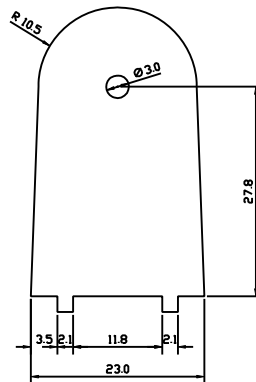


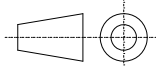
PLANOS CONSTRUCTIVOS

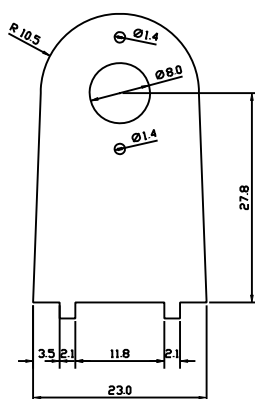
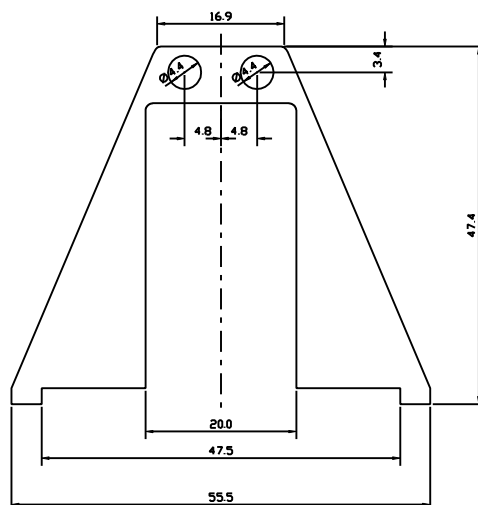
En este anexo se van a mostrar los planos diseñados para su simulación y fabricación de las distintas generaciones de robots realizadas. Debido a que el diseño de las piezas se hizo orientado específicamente a su fabricación en láminas de cartón prensado de 2,1 mm de espesor maquinado en un máquina de fresado de PCB's no se incluyen las vistas de planta y perfil, por se éstas innecesarias.

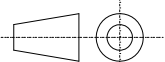


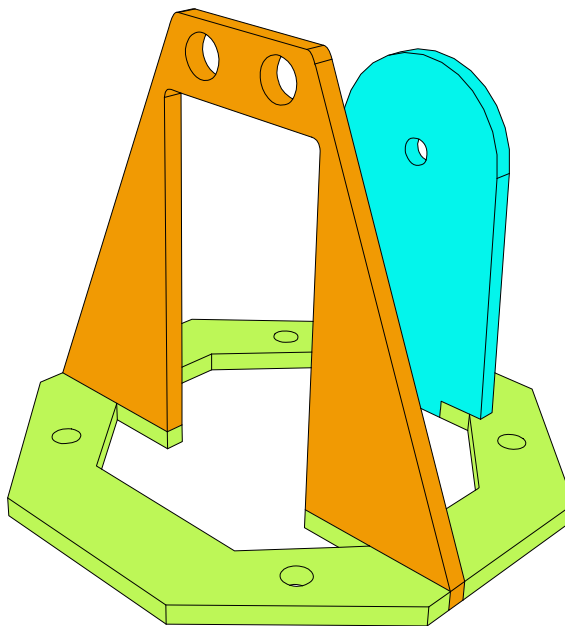
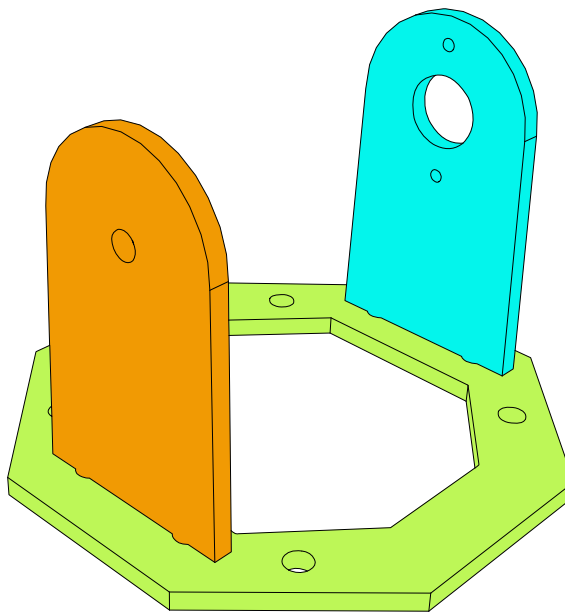
Modificaciones				MÓDULO ORM: PIEZAS BASE		EDICION		
		Tol. gen	Escala					
			1:1					
			Fecha	Nombre	E.T.S. Ingenieros Industriales 	Hoja nº		
		Dibujad.	1/6/11	Andrés San Millán Rodríguez		1		
		Compr.						
		Sustituye a:			Sustituido por:		Nº Hojas	
		Plano nº 01.01					13	



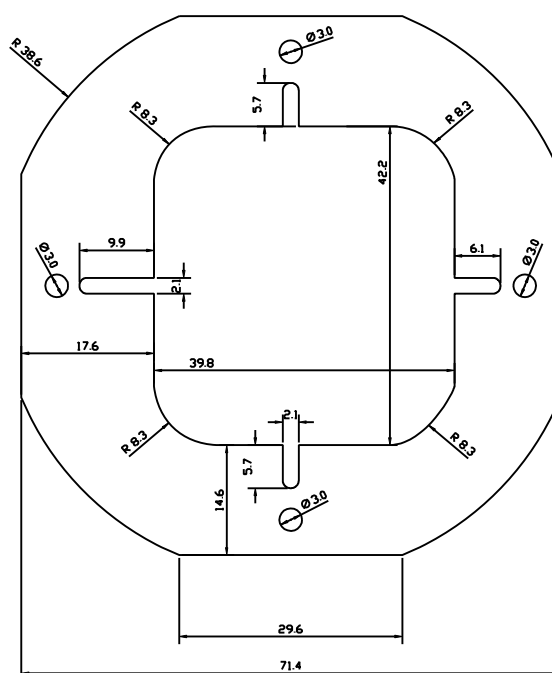
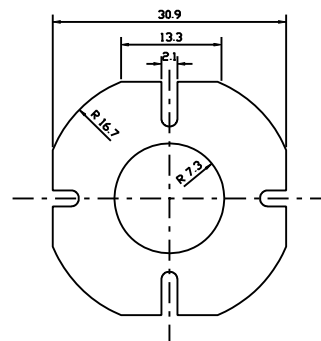
Modificaciones				MÓDULO ORM: PIEZAS LATERALES		EDICION	
		Tol. gen	Escala				
			1:1				
			Fecha	Nombre	E.T.S. Ingenieros Industriales	Hoja nº	
		Dibujad.	1/6/11	Andrés San Millán Rodríguez		2	
		Compr.					
		Sustituye a:			Sustituido por:		Nº Hojas
		Plano nº 01.02					13

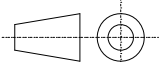


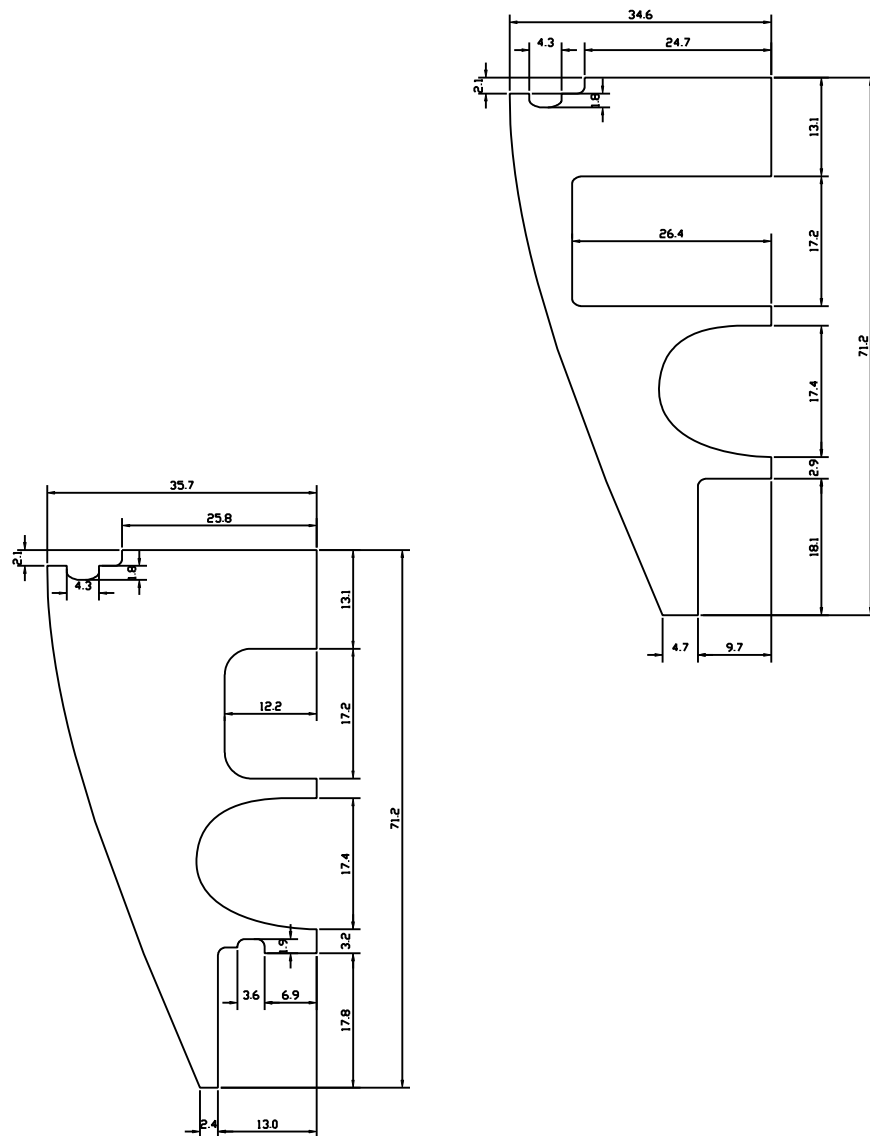
Modificaciones				MÓDULO ORM: SOPORTE DEL SERVO SOPORTE DE LA CORONA		EDICION	
		Tol. gen	Escala				
			1:1	E.T.S. Ingenieros Industriales 		Hoja nº	
		Dibujad.	Fecha			3	
		Compr.	1/6/11				
Sustituye a:		Sustituido por:		Nº Hojas			
Plano nº 01.03						13	



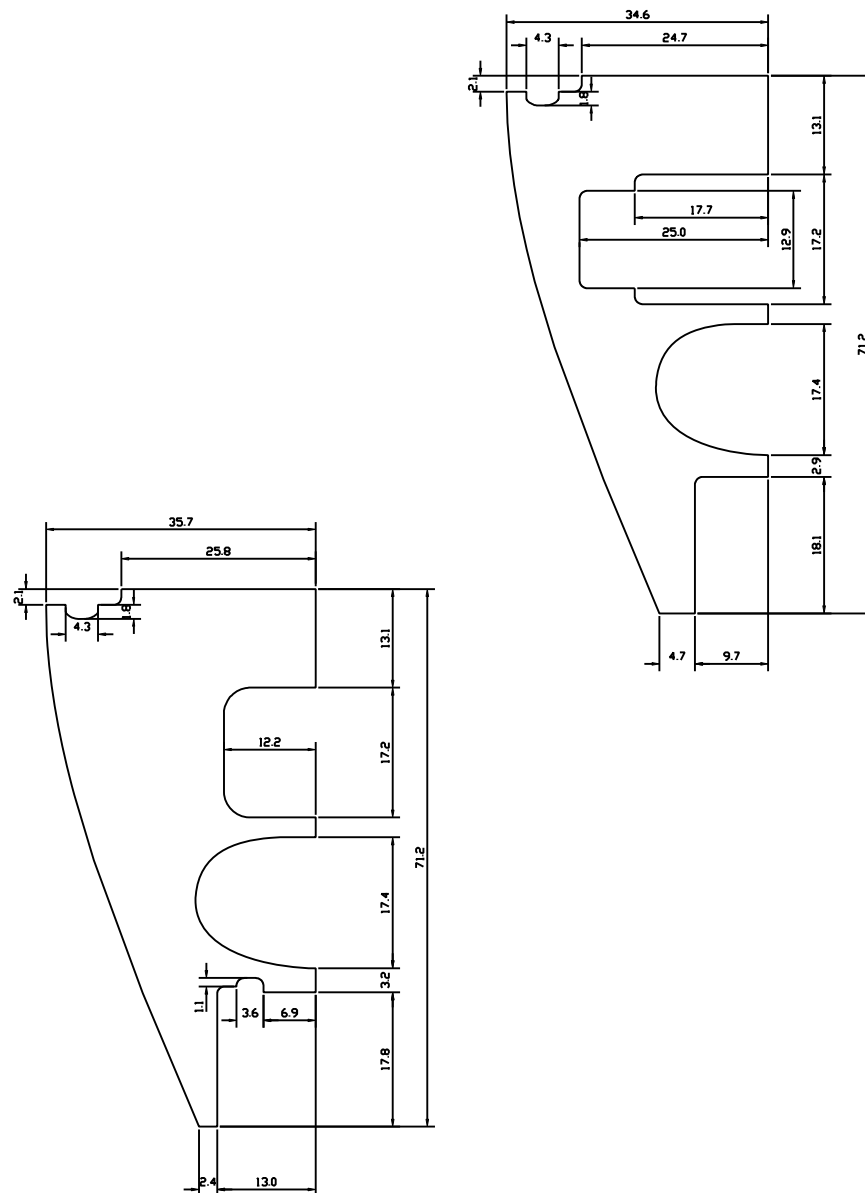
Modificaciones				MÓDULO ORM: VISTA GENERAL		EDICION	
		Tol. gen	Escala				
			-				
			Fecha	Nombre	E.T.S. Ingenieros Industriales	Hoja nº	
		Dibujad.		Andrés San Millán Rodríguez		4	
		Compr.					
		Sustituye a:			Sustituido por:		Nº Hojas
		Plano nº 01.04					13



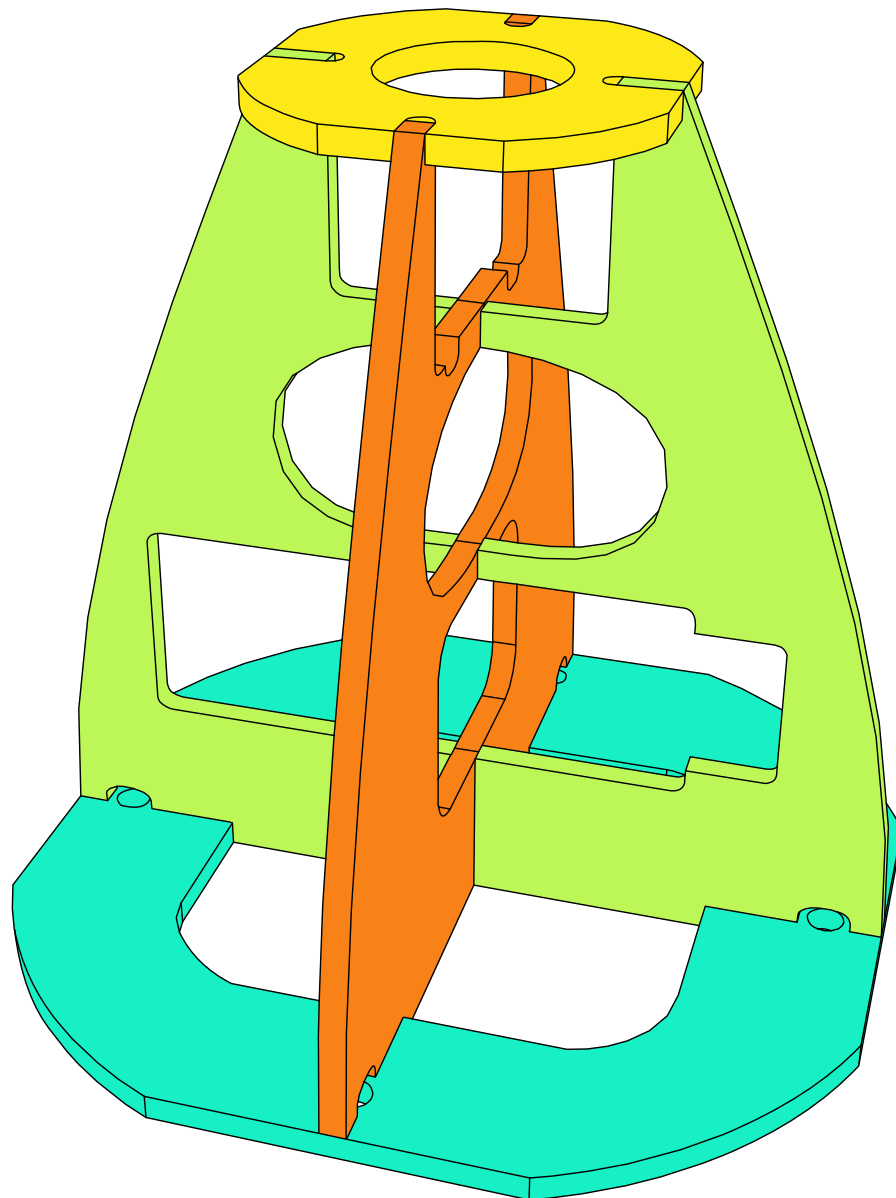
Modificaciones				MÓDULO DE CABEZA: PIEZA BASE Y SOPORTE DE LA CÁMARA			EDICION			
		Tol. gen	Escala							
			1:1							
			Fecha	Nombre	E.T.S. Ingenieros Industriales 			Hoja nº		
		Dibujad.	1/6/11	Andrés San Millán				5		
		Compr.		Rodríguez				Nº Hojas		
		Sustituye a:			Sustituido por:					
		Plano nº 02.01			13					

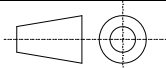


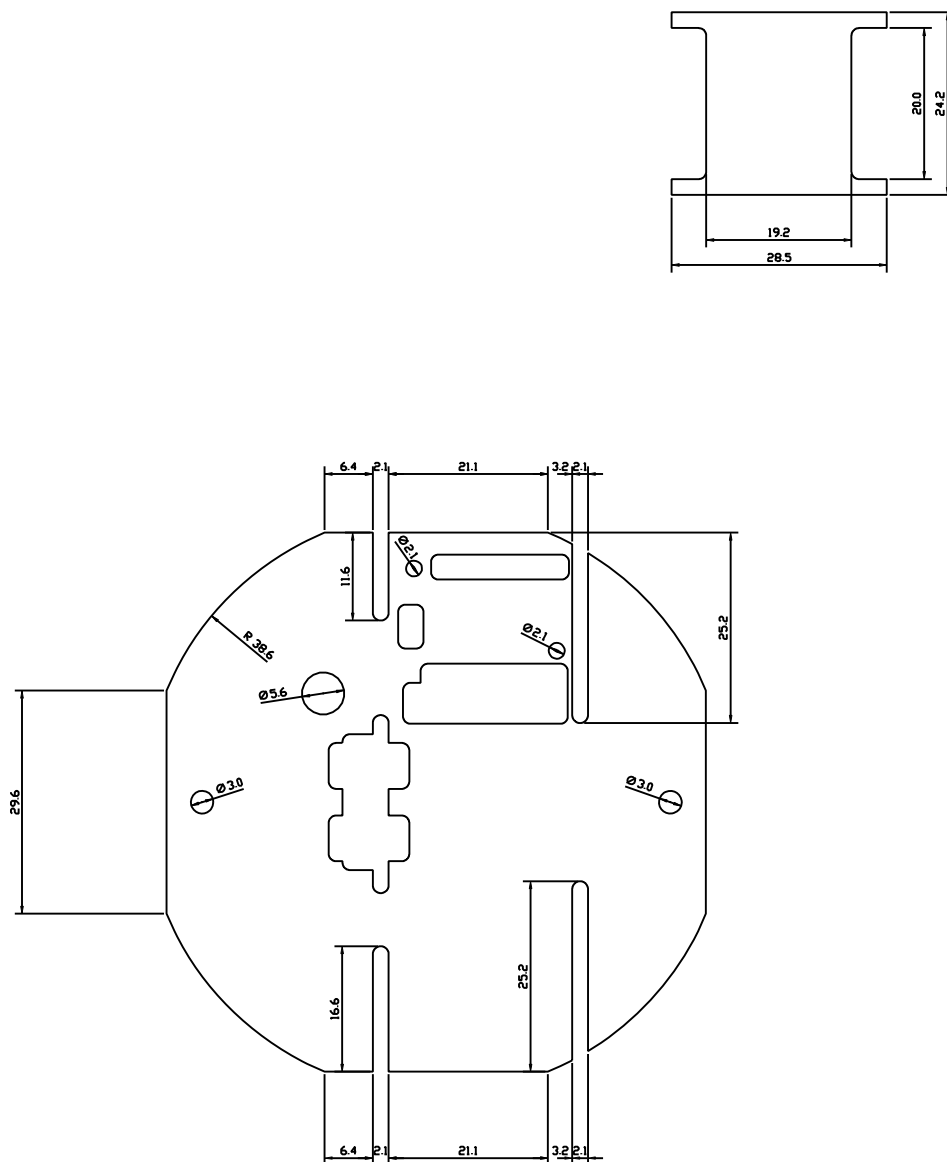
Modificaciones				MÓDULO DE CABEZA: PIEZAS DE SUJECCIÓN 1 Y 2		EDICION	
		Tol. gen	Escala				
			1:1	E.T.S. Ingenieros Industriales			
		Dibujad.	Fecha			Hoja nº	
		Compr.	1/6/11			6	
		Sustituye a:			Sustituido por:		Nº Hojas
		Plano nº 02.02					13

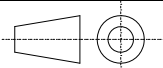


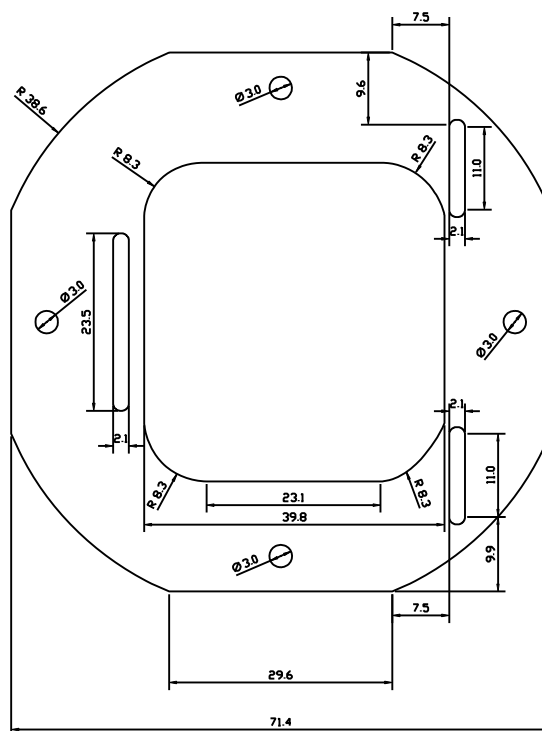
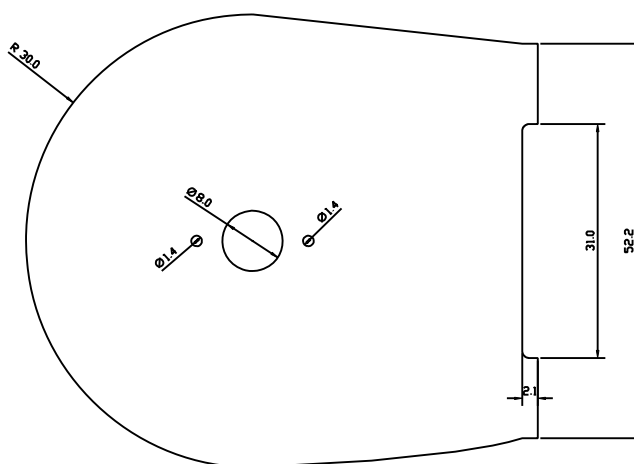
Modificaciones				MÓDULO DE CABEZA: PIEZAS DE SUJECCIÓN 3 Y 4		EDICION		
		Tol. gen	Escala					
			1:1					
			Fecha	Nombre	E.T.S. Ingenieros Industriales	Hoja nº		
		Dibujad.	1/6/11	Andrés San Millán Rodríguez		7		
		Compr.						
		Sustituye a:			Sustituido por:			Nº Hojas
		Plano nº 02.03						13



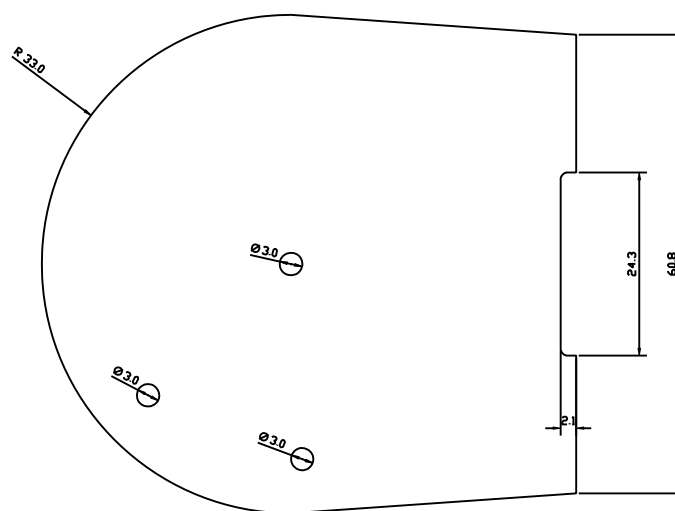
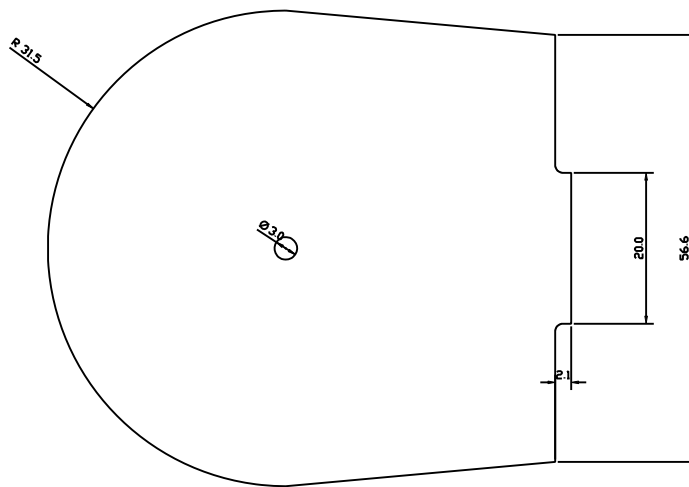
Modificaciones				MÓDULO DE CABEZA: VISTA GENERAL		EDICION	
		Tol. gen	Escala				
			-				
			Fecha	Nombre	E.T.S. Ingenieros Industriales 	Hoja nº	
		Dibujad.	1/6/11	Andrés San Millán Rodríguez		8	
		Compr.					
		Sustituye a:			Sustituido por:		Nº Hojas
		Plano nº 02.04					13

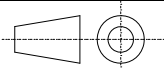


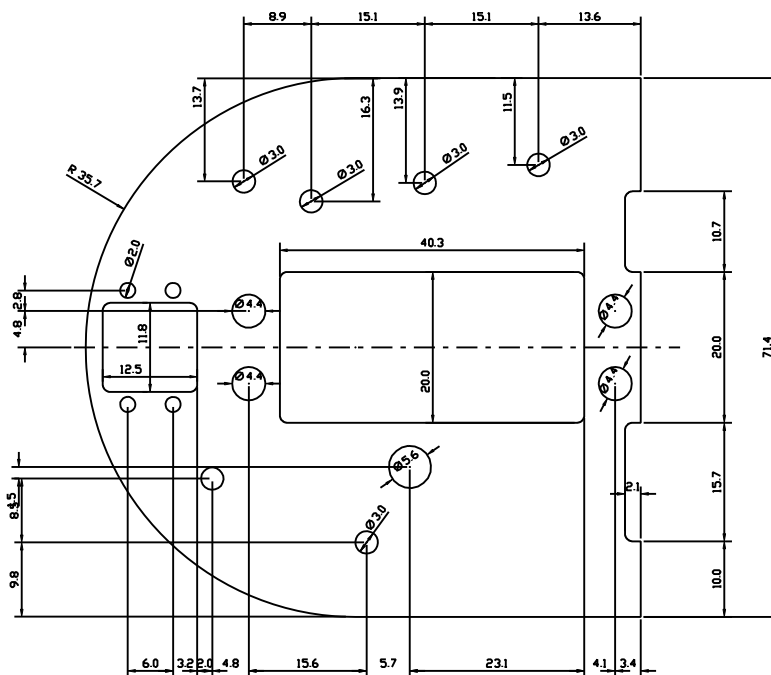
Modificaciones				MÓDULO AUTÓNOMO: PIEZA BASE 1 Y SOPORTE DE XBEE		EDICION	
		Tol. gen	Escala				
			1:1				
			Fecha	Nombre	E.T.S. Ingenieros Industriales 	Hoja nº	
		Dibujad.	1/6/11	Andrés San Millán Rodríguez		9	
		Compr.					
		Sustituye a:			Sustituido por:		Nº Hojas
		Plano nº 03.01					13

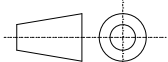


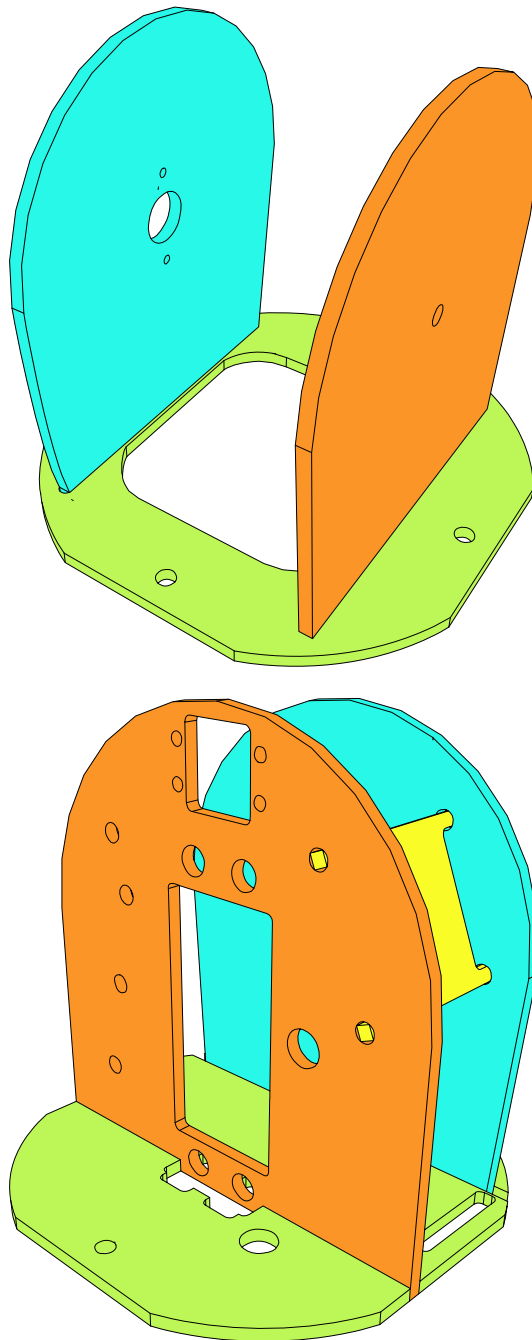
Modificaciones				MÓDULO AUTÓNOMO: PIEZA BASE 2 Y SOPORTE DE LA CORONA		EDICION		
		Tol. gen	Escala					
			1:1					
			Fecha	Nombre	E.T.S. Ingenieros Industriales 	Hoja nº		
	Dibujad.	1/6/11	Andrés San Millán Rodríguez			10		
	Compr.							
		Sustituye a:			Sustituido por:			Nº Hojas
		Plano nº 03.02						13

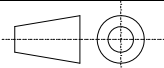


Modificaciones				MÓDULO AUTÓNOMO: PIEZAS LATERALES		EDICION		
		Tol. gen	Escala					
			1:1					
			Fecha	Nombre	E.T.S. Ingenieros Industriales 	Hoja nº		
		Dibujad.	1/6/11	Andrés San Millán Rodríguez		11		
		Compr.						
		Sustituye a:			Sustituido por:			Nº Hojas
		Plano nº 03.03						13



Modificaciones			MÓDULO AUTÓNOMO: SOPORTE DEL SERVO	EDICION	
		Tol. gen	Escala		
			1:1		
		Fecha	Nombre	E.T.S. Ingenieros Industriales 	
		Dibujad.	Andrés San Millán		
		Compr.	Rodríguez	Hoja nº	
		Sustituye a:		Sustituido por:	
		Plano nº 03.04		Nº Hojas	
				13	



Modificaciones				MÓDULO AUTÓNOMO: VISTA GENERAL		EDICION	
		Tol. gen	Escala				
			-				
			Fecha	Nombre	E.T.S. Ingenieros Industriales	Hoja nº	
		Dibujad.	1/6/11	Andrés San Millán Rodríguez		13	
		Compr.					
		Sustituye a:			Sustituido por:		Nº Hojas
		Plano nº 03.05					13

B

DIAGRAMA DE CONEXIONADO DEL ROBOT

En este anexo se muestra el diagrama completo del cableado que conecta los distintos componentes que forman parte del robot



DESCRIPCIÓN DEL SOFTWARE DESARROLLADO

En este anexo se detallarán las características de los distintos programas desarrollados para controlar el robot serpiente. Principalmente diversas aplicaciones creadas para controlar mediante puerto serie la tarjeta controladora de servos Pololu Micro Serial Servo

C.1 Control mediante voz

Tras unos primeros programas desarrollados cuyo principal objetivo era la comprobación del buen funcionamiento de los servomotores y la tarjeta controladora se procedió a la elaboración de un primer programa completo de control.

La primera tarea que se acometió fue el desarrollo de una aplicación que permitiese controlar el robot mediante comandos de voz.

Para poder emprender esta tarea el primer paso fue instalar el llamado Microsoft Speech SDK (software developer kit) que permite desarrollar aplicaciones basadas en el reconocimiento de voz, aunque solamente reconoce palabras en inglés.

Una vez instalado el componente anterior se programó a Visual Basic 6 un programa con la interfaz de usuario que se muestra en las Figuras C.1 y C.2.

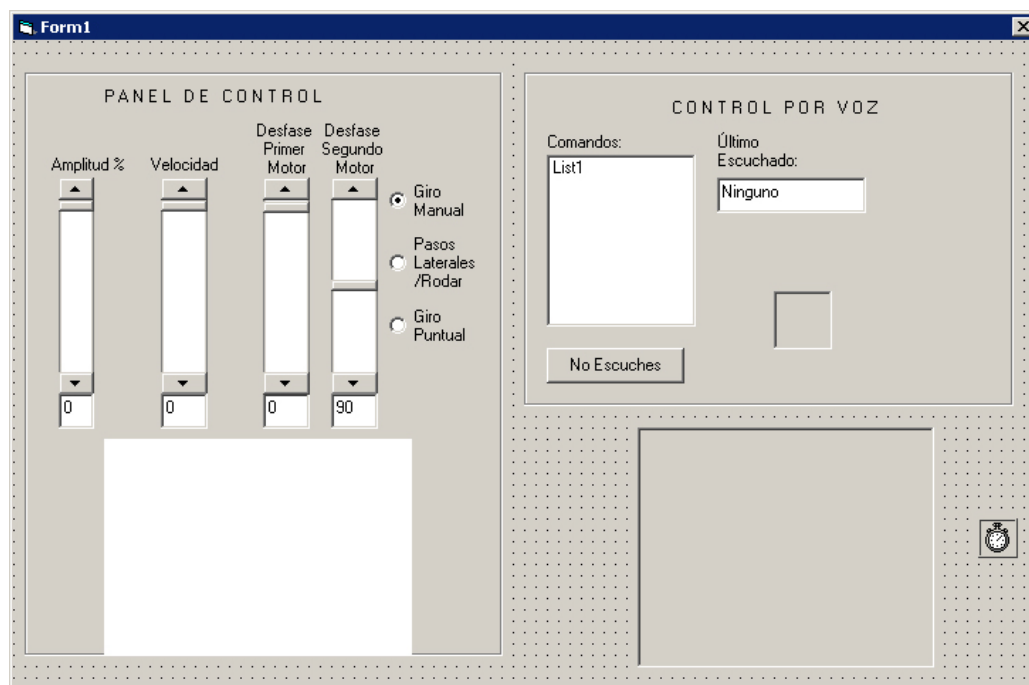


Figura C.1 Interfaz principal del programa

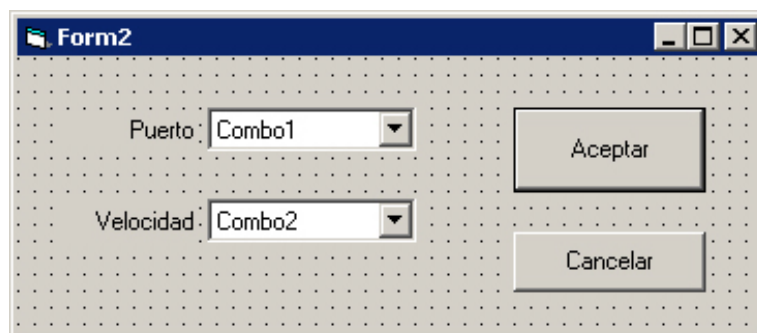


Figura C.2 Interfaz de control del puerto serie

Este programa fue una de las primeras versiones que se utilizó para controlar el robot y su funcionamiento estaba basado en la tarjeta JVM-Servo por lo que admite comunicación por USB y mediante puerto serie como puede apreciarse en el esquema de la Figura D.3

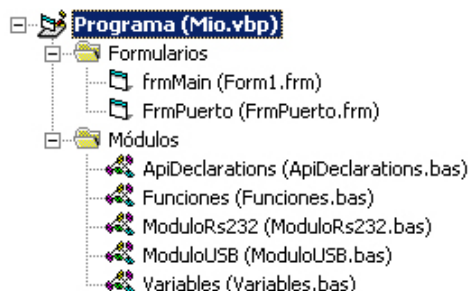


Figura C.3 Esquema del programa

Según el esquema representado en la Figura C.3, puede comprobarse que el programa se estructura en dos ventanas o formularios que presentan la información al usuario y que se encargan de llevar el hilo principal de ejecución del programa y en una serie de módulos encargados de subrutinas específicas como la declaración de variables o el control de la comunicación vía serial o USB.

Aunque no se analizarán los pormenores del código hay que destacar como puntos importantes que el programa permite el control del robot mediante las teclas de dirección como si de un coche de radiocontrol se tratase ya que para el robot para el que estaba pensado el programa (la primera generación compuesta sólo por tres módulos) no se habían desarrollado aún algoritmos de locomoción, y que el programa también es capaz de mostrar por pantalla las imágenes recibidas mediante una cámara.

Como resultado de las experiencias realizadas con el programa se concluyó que el control por voz es ineficaz debido a la necesidad de calibración que requiere el motor de reconocimiento de voz y a la influencia que ejercen los sonidos ambientales en la fiabilidad del sistema.

A continuación se detallará el código empleado en cada módulo del programa, así como el empleado en los formularios:

C.1.1 Módulo *ApiDeclaratios*:

```
*****
'API constants, listed alphabetically
*****

Public Const DIGCF_PRESENT = &H2
Public Const DIGCF_DEVICEINTERFACE = &H10
Public Const FORMAT_MESSAGE_FROM_SYSTEM = &H1000
Public Const GENERIC_READ = &H80000000
Public Const GENERIC_WRITE = &H40000000
Public Const FILE_SHARE_READ = &H1
Public Const FILE_SHARE_WRITE = &H2
Public Const HidP_Input = 0
```

Descripción del software desarrollado

Public Const HidP_Output = 1

Public Const HidP_Feature = 2

Public Const OPEN_EXISTING = 3

'User-defined types for API calls, listed alphabetically

Public Type GUID

 Data1 As Long

 Data2 As Integer

 Data3 As Integer

 Data4(7) As Byte

End Type

Public Type HIDD_ATTRIBUTES

 Size As Long

 VendorID As Integer

 ProductID As Integer

 VersionNumber As Integer

End Type

'Windows 98 DDK documentation is incomplete.

'Use the structure defined in hidpi.h

Public Type HIDP_CAPS

 Usage As Integer

 UsagePage As Integer

 InputReportByteLength As Integer

 OutputReportByteLength As Integer

 FeatureReportByteLength As Integer

 Reserved(16) As Integer

 NumberLinkCollectionNodes As Integer

 NumberInputButtonCaps As Integer

 NumberInputValueCaps As Integer

 NumberInputDataIndices As Integer

 NumberOutputButtonCaps As Integer

 NumberOutputValueCaps As Integer

 NumberOutputDataIndices As Integer

 NumberFeatureButtonCaps As Integer

 NumberFeatureValueCaps As Integer

 NumberFeatureDataIndices As Integer

End Type

'If IsRange is false, UsageMin is the Usage and UsageMax is unused.

Descripción del software desarrollado

'If IsStringRange is false, StringMin is the string index and StringMax is unused.

'If IsDesignatorRange is false, DesignatorMin is the designator index and DesignatorMax is unused.

Public Type HidP_Value_Caps

UsagePage As Integer

ReportID As Byte

IsAlias As Long

BitField As Integer

LinkCollection As Integer

LinkUsage As Integer

LinkUsagePage As Integer

IsRange As Long

IsStringRange As Long

IsDesignatorRange As Long

IsAbsolute As Long

HasNull As Long

Reserved As Byte

BitSize As Integer

ReportCount As Integer

Reserved2 As Integer

Reserved3 As Integer

Reserved4 As Integer

Reserved5 As Integer

Reserved6 As Integer

LogicalMin As Long

LogicalMax As Long

PhysicalMin As Long

PhysicalMax As Long

UsageMin As Integer

UsageMax As Integer

StringMin As Integer

StringMax As Integer

DesignatorMin As Integer

DesignatorMax As Integer

DataIndexMin As Integer

DataIndexMax As Integer

End Type

Public Type SP_DEVICE_INTERFACE_DATA

cbSize As Long

InterfaceClassGuid As GUID

Flags As Long

Reserved As Long

Descripción del software desarrollado

End Type

Public Type SP_DEVICE_INTERFACE_DETAIL_DATA

 cbSize As Long

 DevicePath As Byte

End Type

Public Type SP_DEVINFO_DATA

 cbSize As Long

 ClassGuid As GUID

 DevInst As Long

 Reserved As Long

End Type

'API functions, listed alphabetically

Public Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long

Public Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName As String, ByVal dwDesiredAccess As Long, ByVal dwShareMode As Long, ByRef lpSecurityAttributes As Long, ByVal dwCreationDisposition As Long, ByVal dwFlagsAndAttributes As Long, ByVal hTemplateFile As Long) As Long

Public Declare Function FormatMessage Lib "kernel32" Alias "FormatMessageA" (ByVal dwFlags As Long, ByRef lpSource As Any, ByVal dwMessageId As Long, ByVal dwLanguageZid As Long, ByVal lpBuffer As String, ByVal nSize As Long, ByVal Arguments As Long) As Long

Public Declare Function HidD_FreePreparedData Lib "HID.dll" (ByRef PreparedData As Long) As Long

Public Declare Function HidD_GetAttributes Lib "HID.dll" (ByVal HidDeviceObject As Long, ByRef Attributes As HIDD_ATTRIBUTES) As Long

'Declared as a function for consistency,

'but returns nothing. (Ignore the returned value.)

Public Declare Function HidD_GetHidGuid Lib "HID.dll" (ByRef HidGuid As GUID) As Long

Public Declare Function HidD_GetPreparedData Lib "HID.dll" (ByVal HidDeviceObject As Long, ByRef PreparedData As Long) As Long

Public Declare Function HidP_GetCaps Lib "HID.dll" (ByVal PreparedData As Long, ByRef Capabilities As HIDP_CAPS) As Long

Public Declare Function HidP_GetValueCaps Lib "HID.dll" (ByVal ReportType As Integer, ByRef ValueCaps As Byte, ByRef ValueCapsLength As Integer, ByVal PreparedData As Long) As Long

Public Declare Function lstrcpy Lib "kernel32" Alias "lstrcpyA" (ByVal dest As String, ByVal source As Long) As String

Public Declare Function strlen Lib "kernel32" Alias "strlenA" (ByVal source As Long) As Long

Public Declare Function ReadFile Lib "kernel32" (ByVal hFile As Long, ByRef lpBuffer As Byte, ByVal nNumberOfBytesToRead As Long, ByRef lpNumberOfBytesRead As Long, ByVal lpOverlapped As Long) As Long

Public Declare Function RtlMoveMemory Lib "kernel32" (dest As Any, src As Any, ByVal Count As Long) As Long

```
Public Declare Function SetupDiCreateDeviceInfoList Lib "setupapi.dll" (ByRef ClassGuid As GUID, ByVal  
hWndParent As Long) As Long
```

```
Public Declare Function SetupDiDestroyDeviceInfoList Lib "setupapi.dll" (ByVal DeviceInfoSet As Long) As  
Long
```

```
Public Declare Function SetupDiEnumDeviceInterfaces Lib "setupapi.dll" (ByVal DeviceInfoSet As Long,  
ByVal DeviceInfoData As Long, ByRef InterfaceClassGuid As GUID, ByVal MemberIndex As Long, ByRef  
DeviceInterfaceData As SP_DEVICE_INTERFACE_DATA) As Long
```

```
Public Declare Function SetupDiGetClassDevs Lib "setupapi.dll" Alias "SetupDiGetClassDevsA" (ByRef  
ClassGuid As GUID, ByVal Enumerator As String, ByVal hWndParent As Long, ByVal Flags As Long) As  
Long
```

```
Public Declare Function SetupDiGetDeviceInterfaceDetail Lib "setupapi.dll" Alias  
"SetupDiGetDeviceInterfaceDetailA" (ByVal DeviceInfoSet As Long, ByRef DeviceInterfaceData As  
SP_DEVICE_INTERFACE_DATA, ByVal DeviceInterfaceDetailData As Long, ByVal  
DeviceInterfaceDetailDataSize As Long, ByRef RequiredSize As Long, ByVal DeviceInfoData As Long) As  
Long
```

```
Public Declare Function WriteFile Lib "kernel32" (ByVal hFile As Long, ByRef lpBuffer As Byte, ByVal  
nNumberOfBytesToWrite As Long, ByRef lpNumberOfBytesWritten As Long, ByVal lpOverlapped As Long)  
As Long
```

C.1.2 Módulo *Funciones*

Option Explicit

' Función que crea un timer

```
Declare Function SetTimer Lib "user32" (ByVal hWnd As Long, ByVal nIDEvent As Long, ByVal uElapse As  
Long, ByVal lpTimerFunc As Long) As Long
```

' Función que detiene el timer iniciado

```
Declare Function KillTimer Lib "user32" (ByVal hWnd As Long, ByVal nIDEvent As Long) As Long
```

Dim Buffer As String

Dim Retraso As Boolean 'Variable que indica que se cumplio el tiempo

' Función Callback que se dispara al iniciar el timer

```
Sub TimerProc(ByVal hWnd As Long, ByVal nIDEvent As Long, ByVal uElapse As Long, ByVal lpTimerFunc  
As Long)
```

 TimeOut = True

```
End Sub
```

```
Function AbrirPuerto(MostrarError As Boolean) As Boolean
```

On Error GoTo Rutina

'cerramos el puerto por si esta abierto

```
If frmMain.PuertoSerie.PortOpen Then frmMain.PuertoSerie.PortOpen = False
```

'Puerto a abrir

```
frmMain.PuertoSerie.CommPort = Puerto 'el numero del puerto
```

' 115200 baud, sin paridad, 8 bytes, y un bit de stop.

```
If Vel_RS232 = "" Then Vel_RS232 = "9600"
```

```
frmMain.PuertoSerie.Settings = Vel_RS232 & ",N,8,1"
```

' lectura completa del buffer

```
frmMain.PuertoSerie.InputLen = 0
```

Descripción del software desarrollado

```
' Abrimos el puerto
frmMain.PuertoSerie.PortOpen = True
'Flag indicador de que se ha abierto el puerto serie con exito
AbrirPuerto = True
Exit Function
```

```
Rutina:
AbrirPuerto = False
If MostrarError Then
    MsgBox Error & "... Abrir puerto", vbInformation, "Error"
End If
End Function
```

```
Sub CerrarPuertoSerie()
On Error GoTo Rutina
' Cerramos el puerto serie
frmMain.PuertoSerie.PortOpen = False
Rutina:
End Sub
```

```
'Timer de retraso para el envio de datos
Sub retraso15(ByVal hwnd As Long, ByVal nIDEvent As Long, ByVal uElapse As Long, ByVal lpTimerFunc
As Long)
    Retraso = True
End Sub

'Lee las posicion del servo seleccionado
Function LeerPosicion(nServo As Integer, MostrarError As Boolean) As Byte
On Error GoTo Rutina
'Abre el puerto y si lo consigue lee las posiciones
If PuertoAbierto Then
    frmMain.PuertoSerie.Output = "P" & Chr$(nServo)
    'Creamos un timer de 15 ms para que se envíen los datos
    SetTimer frmMain.hwnd, 0, 15, AddressOf retraso15
    Retraso = False
    While Retraso = False
        DoEvents
    Wend
    KillTimer frmMain.hwnd, 0
    Buffer = ""
    Timeout = False
    'Encendemos el timer de timeout
    SetTimer frmMain.hwnd, 0, 300, AddressOf TimerProc
    'Esperamos a que se envíen todos los datos
```

Descripción del software desarrollado

```
While frmMain.PuertoSerie.OutBufferCount > 0 And TimeOut = False
    DoEvents
Wend
'Esperamos a que entren los datos
While frmMain.PuertoSerie.InBufferCount = 0 And TimeOut = False
    DoEvents
Wend
Do
    DoEvents
    Buffer = frmMain.PuertoSerie.Input
    If Buffer <> "" Then
        LeerPosicion = Val(Asc(Buffer))
    End If
    Loop Until InStr(Buffer, vbCr) Or TimeOut = True
'Matamos el timer de TimeOut
KillTimer frmMain.hwnd, 0
'Si se habilita el TimeOut puede ser porque no se detecta la placa
If TimeOut Then
    If MostrarError Then
        MsgBox "Error de Time Out... Leer posicion", vbInformation, "Error"
    End If
End If
End If
Exit Function
Rutina:
    MsgBox Error & "... Leer posicion", vbInformation, "Error"
End Function

Sub EnviarPosicion()
    On Error GoTo Rutina
    Dim CadenaEnvio As String
    Dim tt As Integer
    Dim TmpC As Integer
    'Modo RS232
    If ModoRS232 = RS232 And PuertoAbierto Then
        'Lanzamos al puerto el comando a los servos *****
        If PuertoAbierto Then
            Dim Buffer As String
            'Generamos la cadena de envio
            CadenaEnvio = "T"
            'Envio los valores de los servos en modo multiple
            For tt = 0 To 27
```

Descripción del software desarrollado

```
        CadenaEnvio = CadenaEnvio & Chr(frmMain.VS1(tt).Value)
    Next tt
    'Enviamos la posicion a los servos
    frmMain.PuertoSerie.Output = CadenaEnvio
End If
End If
Exit Sub
Rutina:
    MsgBox Error & "... Enviar posicion", vbInformation, "Error"
End Sub

Sub EnviarPosicionServo(nServo As Integer, Posicion As Byte)
On Error GoTo Rutina
Dim CadenaEnvio As String
Dim t As Integer
'Modo RS232
If ModoRS232 = RS232 And PuertoAbierto Then
    'Lanzamos al puerto el comando a los servos *****
    If PuertoAbierto Then
        Dim Buffer As String
        'Enviamos la posicion a los servos
        frmMain.PuertoSerie.Output = "S" & Chr$(nServo) & Chr$(Posicion)
    End If
End If
Exit Sub
Rutina:
    MsgBox Error & "... Enviar posicion Servo", vbInformation, "Error"
End Sub

Sub EnviarComandoDeInicio(nServo As Integer, Posicion As Byte)
On Error GoTo Rutina
If ModoRS232 = RS232 And PuertoAbierto Then
    'Lanzamos al puerto el comando a los servos *****
    If PuertoAbierto Then
        frmMain.PuertoSerie.Output = "I" & Chr$(nServo) & Chr(Posicion)
        Buffer = ""
        TimeOut = False
        'Encendemos el timer de timeout
        SetTimer frmMain.hwnd, 0, 300, AddressOf TimerProc
        'Esperamos a que se envíen todos los datos
        While frmMain.PuertoSerie.OutBufferCount > 0 And TimeOut = False
            DoEvents
        End While
    End If
End If
Exit Sub
Rutina:
    MsgBox Error & "... Enviar comando de inicio", vbInformation, "Error"
End Sub
```

Descripción del software desarrollado

```
Wend
'Esperamos a que entren los datos
While frmMain.PuertoSerie.InBufferCount = 0 And TimeOut = False
    DoEvents
Wend
Do
    DoEvents
    Buffer = frmMain.PuertoSerie.Input
Loop Until InStr(Buffer, vbCr) Or TimeOut = True
'Matamos el timer de TimeOut
KillTimer frmMain.hwnd, 0
If TimeOut Then
    MsgBox "Error de Time Out en el comando de inicio", vbInformation, "Error"
End If
End If
Exit Sub
Rutina:
    MsgBox Error & "... Enviar comando de inicio", vbInformation, "Error"
End Sub

Sub VelocidadRS232(Velocidad As Integer)
On Error GoTo Rutina
Dim CadenaEnvio As String
Dim t As Integer
'Modo RS232
If ModoRS232 = RS232 And PuertoAbierto Then
    'Lanzamos al puerto el comando a los servos *****
    If PuertoAbierto Then
        Dim Buffer As String
        'Enviamos la posicion a los servos
        frmMain.PuertoSerie.Output = "A" & Chr$(Velocidad)
        Buffer = ""
        TimeOut = False
        'Encendemos el timer de timeout
        SetTimer frmMain.hwnd, 0, 300, AddressOf TimerProc
    Do
        DoEvents
        Buffer = frmMain.PuertoSerie.Input
    Loop Until InStr(Buffer, vbCr) Or TimeOut = True
    'Matamos el timer de TimeOut
    KillTimer frmMain.hwnd, 0
```

```
If TimeOut Then
    MsgBox "Error de Time Out enviando la velocidad al puerto serie", vbInformation, "Error"
End If
End If
Exit Sub
Rutina:
    MsgBox Error & "... Enviando velocidad de inicio", vbInformation, "Error"
End Sub
```

C.1.3 Módulo *ModuloUSB*

```
Option Explicit
Dim Capabilities As HIDP_CAPS
Dim DataString As String
Dim DetailData As Long
Dim DetailDataBuffer() As Byte
Dim DeviceAttributes As HIDD_ATTRIBUTES
Dim DevicePathName As String
Dim DeviceInfoSet As Long
Dim HidDevice As Long
Dim LastDevice As Boolean
Dim MyDeviceInfoData As SP_DEVINFO_DATA
Dim MyDeviceInterfaceDetailData As SP_DEVICE_INTERFACE_DETAIL_DATA
Dim MyDeviceInterfaceData As SP_DEVICE_INTERFACE_DATA
Dim Needed As Long
Dim OutputReportData(65) As Byte
Dim DispositivoDetectado As Boolean
Dim PreparsedData As Long
Dim Result As Long
Dim TimeOut As Boolean

'Valores del HID
Const VID = &H1562 '&H8568 (30 v2)
Const PID = &H2312 '&H4865 (30 v2)
'Variable que nos indica si esta detectada la placa
Public DetectBoard As Boolean

Sub DetectarUSB(MostrarError As Boolean)
    Dim HidGuid As GUID
    Dim MemberIndex As Long
    LastDevice = False
```

DetectBoard = False

'HidD_GetHidGuid

'Get the GUID for all system HIDs.

'Returns: the GUID in HidGuid.

'The routine doesn't return a value in Result

'but the routine is declared as a function for consistency with the other API calls.

Result = HidD_GetHidGuid(HidGuid)

'SetupDiGetClassDevs

'Returns: a handle to a device information set for all installed devices.

'Requires: the HidGuid returned in GetHidGuid.

DeviceInfoSet = SetupDiGetClassDevs(HidGuid, vbNullString, 0, (DIGCF_PRESENT Or
DIGCF_DEVICEINTERFACE))

'Begin with 0 and increment until no more devices are detected.

MemberIndex = 0

Do

 'The cbSize element of the MyDeviceInterfaceData structure must be set to

 'the structure's size in bytes. The size is 28 bytes.

 MyDeviceInterfaceData.cbSize = LenB(MyDeviceInterfaceData)

 Result = SetupDiEnumDeviceInterfaces(DeviceInfoSet, 0, HidGuid, MemberIndex, MyDeviceInterfaceData)

 'Call DisplayResultOfAPICall("SetupDiEnumDeviceInterfaces")

 If Result = 0 Then LastDevice = True

 'If a device exists, display the information returned.

 If Result <> 0 Then

 MyDeviceInfoData.cbSize = Len(MyDeviceInfoData)

 Result = SetupDiGetDeviceInterfaceDetail(DeviceInfoSet, MyDeviceInterfaceData, 0, 0, Needed, 0)

 DetailData = Needed

 'Store the structure's size.

 MyDeviceInterfaceDetailData.cbSize = Len(MyDeviceInterfaceDetailData)

 'Use a byte array to allocate memory for

 'the MyDeviceInterfaceDetailData structure

 ReDim DetailDataBuffer(Needed)

 'Store cbSize in the first four bytes of the array.

 Call RtlMoveMemory(DetailDataBuffer(0), MyDeviceInterfaceDetailData, 4)

 'Call SetupDiGetDeviceInterfaceDetail again.

 'This time, pass the address of the first element of DetailDataBuffer

 'and the returned required buffer size in DetailData.

Descripción del software desarrollado

```
Result = SetupDiGetDeviceInterfaceDetail(DeviceInfoSet, MyDeviceInterfaceData,
VarPtr(DetailDataBuffer(0)), DetailData, Needed, 0)
'Convert the byte array to a string.
DevicePathName = CStr(DetailDataBuffer())
'Convert to Unicode.
DevicePathName = StrConv(DevicePathName, vbUnicode)
'Strip cbSize (4 bytes) from the beginning.
DevicePathName = Right$(DevicePathName, Len(DevicePathName) - 4)
HidDevice = CreateFile(DevicePathName, GENERIC_READ Or GENERIC_WRITE,
(FILE_SHARE_READ Or FILE_SHARE_WRITE), 0, OPEN_EXISTING, 0, 0)
'Set the Size property to the number of bytes in the structure.
DeviceAttributes.Size = LenB(DeviceAttributes)
Result = HidD_GetAttributes(HidDevice, DeviceAttributes)
'Find out if the device matches the one we're looking for.
If (DeviceAttributes.VendorID = VID) And (DeviceAttributes.ProductID = PID) Then
    DetectBoard = True 'Variable de deteccion de la placa controladora de servos
Else
    DetectBoard = False 'Variable de deteccion de la placa controladora de servos
    'If it's not the one we want, close its handle.
    Result = CloseHandle(HidDevice)
End If
End If
'Keep looking until we find the device or there are no more left to examine.
MemberIndex = MemberIndex + 1
Loop Until (LastDevice = True) Or (DispositivoDetectado = True)
'Si se detecta la controladora y se capturan las propiedades
If DetectBoard Then
    Dim ppData(29) As Byte
    Dim ppDataString As Variant
    'Prepared Data is a pointer to a routine-allocated buffer.
    Result = HidD_GetPreparedData(HidDevice, PreparedData)
    'Copy the data at PreparedData into a byte array.
    Result = RtlMoveMemory(ppData(0), PreparedData, 30)
    ppDataString = ppData()
    'Convert the data to Unicode.
    ppDataString = StrConv(ppDataString, vbUnicode)
    Result = HidP_GetCaps(PreparedData, Capabilities)

    Debug.Print Capabilities.OutputReportByteLength
    Debug.Print Capabilities.InputReportByteLength
Else
    If MostrarError Then
        MsgBox "No se detecta la controladora... Revise la conexión USB", vbCritical, "Error"
```

Descripción del software desarrollado

```
End If
End If
End Sub

Private Function LeerUSB(ByRef ByteValue As Byte) As Boolean
Dim Count
Dim NumberOfBytesRead As Long
Dim ReadBuffer() As Byte
Dim UBoundReadBuffer As Integer
'The ReadBuffer array begins at 0, so subtract 1 from the number of bytes.
ReDim ReadBuffer(Capabilities.InputReportByteLength - 1)
'Pass the address of the first byte of the read buffer.
Result = ReadFile(HidDevice, ReadBuffer(0), CLng(Capabilities.InputReportByteLength),
NumberOfBytesRead, 0)
'Debug.Print NumberOfBytesRead
If Result Then
For Count = 1 To UBound(ReadBuffer)
'Add a leading 0 to values 0 - Fh.
If Len(Hex$(ReadBuffer(Count))) < 2 Then
ByteValue = ReadBuffer(Count)
Else
ByteValue = ReadBuffer(Count)
End If
Debug.Print ByteValue
Next Count
LeerUSB = True
End If
End Function

Private Function EscribirUSB() As Boolean
Dim Count As Integer
Dim NumberOfBytesRead As Long
Dim NumberOfBytesToSend As Long
Dim NumberOfBytesWritten As Long
Dim ReadBuffer() As Byte
Dim SendBuffer() As Byte
If Capabilities.OutputReportByteLength <= 0 Then
Exit Function
End If
'The SendBuffer array begins at 0, so subtract 1 from the number of bytes.
ReDim SendBuffer(Capabilities.OutputReportByteLength - 1)
'The first byte is the Report ID
SendBuffer(0) = 0
```

Descripción del software desarrollado

```
'The next bytes are data
For Count = 1 To Capabilities.OutputReportByteLength - 1
    SendBuffer(Count) = OutputReportData(Count - 1)
Next Count
NumberOfBytesWritten = 0
Result = WriteFile(HidDevice, SendBuffer(0), CLng(Capabilities.OutputReportByteLength),
NumberOfBytesWritten, 0)
If Result Then EscribirUSB = True
End Function

'Lee las posiciones de los servos
Function LeerPosicion(nServo As Integer) As Byte
On Error GoTo Rutina
'Leemos la posicion de los servos
OutputReportData(0) = Asc("P")           'Cabecera ("P")
OutputReportData(1) = nServo             'Nº de Servo al que accedemos
If EscribirUSB = False Then Exit Function 'Escribimos la consula de la posicion
If LeerUSB(LeerPosicion) = False Then Exit Function 'Devuelve la posicion en Valor2
Exit Function
Rutina:
    MsgBox "Error... al leer las posiciones", vbExclamation, "Error"
End Function

Sub EnviarPosicion()
'On Error GoTo Rutina
Dim TmpC As Integer
OutputReportData(0) = Asc("T")           'cabecera (todos=T)
'Preparacion de los datos de cada servo al bufer TX del USB
For TmpC = 1 To 3
    OutputReportData(TmpC) = frmMain.VS1(TmpC - 1).Value 'Preparacion del buffer
Next TmpC
'si la escritura del usb falla intentamos capturar el HID
If EscribirUSB = False Then Call DetectarUSB(False)
Exit Sub
Rutina:
    MsgBox "Error en el sub Enviarposición", vbInformation, "Error"
End Sub

Sub EnviarPosicionServo(nServo As Integer, Posicion As Byte)
On Error GoTo Rutina
OutputReportData(0) = Asc("S")           'cabecera (Servo =S)
OutputReportData(1) = nServo             'Numero del servo
OutputReportData(2) = Posicion           'Preparacion del buffer
'si la escritura del usb falla intentamos capturar el HID
```

Descripción del software desarrollado

If EscribirUSB = False Then Call DetectarUSB(False)

Exit Sub

Rutina:

MsgBox "Error en la funcion EnviarPosiciónServo..."

End Sub

Sub EnviarComandoDeInicio(nServo As Integer, Posicion As Byte)

On Error GoTo Rutina

OutputReportData(0) = Asc("I") 'Cabecera (I)

OutputReportData(1) = nServo 'Nº de Servo

OutputReportData(2) = Posicion 'Preparacion del dato de la posicion

'Escribimos los datos en el puerto USB

Call EscribirUSB

Exit Sub

Rutina:

MsgBox "Error en el sub EnviarComandoInicio..."

End Sub

'Lee el parametro de Ancho del PWM

Function LeerAnchoPWM(Puerto As Integer) As Byte

On Error GoTo Rutina

OutputReportData(0) = Asc("R") 'Cabecera ("R")

OutputReportData(1) = Puerto 'Nº de puerto

If EscribirUSB = False Then Exit Function 'Escribimos la consula de la posicion

If LeerUSB(LeerAnchoPWM) = False Then Exit Function 'Devuelve el valor del ancho PWM

Exit Function

Rutina:

MsgBox Error & "...Leyendo el parametro de Ancho del PWM"

End Function

'Lee el parametro de inicio del PWM

Function LeerInicioPWM(Puerto As Integer) As Byte

On Error GoTo Rutina

OutputReportData(0) = Asc("Q") 'Cabecera ("Q")

OutputReportData(1) = Puerto 'Nº de puerto

If EscribirUSB = False Then Exit Function 'Escribimos la consula de la posicion

If LeerUSB(LeerInicioPWM) = False Then Exit Function 'Devuelve el valor del inicio PWM

Exit Function

Rutina:

MsgBox Error & "... Leyendo el parametro de inicio del PWM"

End Function

Sub EnviarInicioPWM(Puerto As Integer, valor As Byte)

Descripción del software desarrollado

On Error GoTo Rutina

OutputReportData(0) = Asc("O") 'cabecera (PWM sIni=O)

OutputReportData(1) = Puerto 'puerto

OutputReportData(2) = valor 'Valor del puerto

'si la escritura del usb falla intentamos capturar el HID

If EscribirUSB = False Then Call DetectarUSB(False)

Exit Sub

Rutina:

 MsgBox Error & "... Enviando el parametro de Inicio del PWM"

End Sub

Sub EnviarAnchoPWM(Puerto As Integer, valor As Byte)

On Error GoTo Rutina

OutputReportData(0) = Asc("N") 'cabecera (PWM sfin=N)

OutputReportData(1) = Puerto 'puerto

OutputReportData(2) = valor 'Valor del puerto

'si la escritura del usb falla intentamos capturar el HID

If EscribirUSB = False Then Call DetectarUSB(False)

Exit Sub

Rutina:

 MsgBox Error & "... Enviando el parametro de Ancho del PWM"

End Sub

'Lee la version del controlador

Function LeerVersion() As String

On Error GoTo Rutina

OutputReportData(0) = Asc("Z") 'Cabecera ("Z")

OutputReportData(1) = 13 'Bit 1

If EscribirUSB = False Then Exit Function 'Escribimos la consola de la posicion

'Leemos los caracteres del ID

Dim cadena As Byte

Dim aa As Integer

For aa = 0 To 20

 If LeerUSB(cadena) = False Then Exit Function 'Devuelve cada caracter leído

Next aa

Exit Function

Rutina:

 MsgBox "Error... al leer la ID", vbExclamation, "Error"

End Function

```
Sub VelocidadRS232(Velocidad As Integer)
On Error GoTo Rutina
'If ModoComunicacion = USB And PlacaDetectada Then
If ModoRS232 = USB Then
    Dim TmpC As Integer
    Dim BuffUSB As Long
    OutputReportData(0) = Asc("A")           'cabecera (A)
    OutputReportData(1) = Velocidad          'Velocidad
    'si la escritura del usb falla intentamos capturar el HID
    If EscribirUSB = False Then Call DetectarUSB(False)
End If
Exit Sub
Rutina:
    MsgBox Error & "... Enviar velocidad RS232"
End Sub
```

```
Function CerrarUsb() As Boolean
'Close the open handle to the device.
Result = CloseHandle(HidDevice)
'Free memory used by SetupDiGetClassDevs
Result = SetupDiDestroyDeviceInfoList(DeviceInfoSet)
'Libera el PreparsedData
Result = HidD_FreePreparsedData(PreparsedData)
End Function
```

C.1.4 Módulo *Variables*

```
'Funciones de configuracion de las comunicaciones
Global Const RS232 As Integer = 0
Global Const USB As Integer = 1
```

```
Public TimeOut As Boolean           ' Variable de time out
Public PuertoAbierto As Boolean      ' Indicador de que el puerto esta abierto
Public ModoRS232 As Integer         ' Modo USB=false o RS232=true

Public Puerto As Integer             'Puerto COM de comunicacion serie
Public Vel_RS232 As String           'Velocidad del puerto
Public VelocidadPuertoAnt As String
```

C.1.5 Formulario *FrmPuerto*

```
Option Explicit
Private Sub CargarPropiedades()
```

Descripción del software desarrollado

```
Dim i As Integer, Settings As String, Offset As Integer
' Carga las configuraciones de puerto
Combo1.Clear
For i = 1 To 16
    Combo1.AddItem "COM" & Trim$(Str$(i))
Next i
' Carga las configuraciones de velocidad
Combo2.Clear
Combo2.AddItem "9600"
Combo2.AddItem "19200"
Combo2.AddItem "38400"
Combo2.AddItem "57600"
Combo2.AddItem "115200"
' Establece la configuración predeterminada
Settings = frmMain.PuertoSerie.Settings
'Seleccionamos los valores actuales del puerto
Combo1.ListIndex = Puerto - 1
Combo2.Text = Vel_RS232
'Cargamos la variable de velocidad anterior
VelocidadPuertoAnt = Vel_RS232
End Sub

Private Sub Command1_Click()
'Grabamos los datos de configuracion en el INI
Call EscribeINI("Conf.ini", "Settings", "Puerto", Trim(Str((Combo1.ListIndex + 1))))
Call EscribeINI("Conf.ini", "Settings", "Baudios", Trim(Combo2.Text))
'Velocidad del puerto serie
Vel_RS232 = Combo2.Text
'Numero de puerto
Puerto = (Combo1.ListIndex + 1)
' Enviamos el valor de velocidad del puerto serie
If ModoRS232 = RS232 Then Call ModuloRs232.VelocidadRS232(Combo2.ListIndex)
' Enviamos tambien el cambio de la velocidad por el USB
If ModoRS232 = USB Then Call ModuloUSB.VelocidadRS232(Combo2.ListIndex)
' Miramos si se ha cambiado la velocidad del puerto
If VelocidadPuertoAnt <> Vel_RS232 Then
    ' Actualizamos el valor de la velocidad
    VelocidadPuertoAnt = Combo2.Text
    MsgBox "Inicialice su controladora y pulse aceptar...", vbInformation, "Inicialización..."
End If
'Apertura del puerto
If ModoRS232 = RS232 Then
```

```
PuertoAbierto = ModuloRs232.AbrirPuerto(True) 'Abrimos el puerto serie
Else
    Call ModuloUSB.CerrarUsb
    Call ModuloUSB.DetectarUSB(True)          'Buscamos la controladora
End If
'Leemos las posiciones
Call frmMain.Leer_Posiciones
Unload Me
End Sub
```

```
Private Sub Command2_Click()
Unload Me
End Sub
```

```
Private Sub Form_Load()
' Carga la configuración actual de las propiedades
Call CargarPropiedades
End Sub
```

C.1.6 Formulario *frmMain*

```
'CAMARA STUFF'
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal
wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
Private Declare Function capCreateCaptureWindow Lib "avicap32.dll" Alias "capCreateCaptureWindowA"
(ByVal lpszWindowName As String, ByVal dwStyle As Long, ByVal X As Long, ByVal Y As Long, ByVal
nWidth As Long, ByVal nHeight As Long, ByVal hwndParent As Long, ByVal nID As Long) As Long
Private mCapHwnd As Long
Private Const CONNECT As Long = 1034
Private Const DISCONNECT As Long = 1035
Private Const GET_FRAME As Long = 1084
Private Const COPY As Long = 1054
Private Declare Function GetTickCount Lib "kernel32" () As Long
Dim LastTime As Long
Dim tcount As Long
Dim Velocidad_motor As Long
Option Explicit
'CAMARA STUFF'
Dim radian As Double
Dim indice As Integer
Dim Mot1 As Byte
Dim Mot2 As Byte
Dim Mot3 As Byte
```

Descripción del software desarrollado

Const PI = 3.141592654

Dim Interactuar As Boolean

Dim gMymenu As Long

Private Sub UpdateList()

Dim i As Long

Dim command As String

Dim description As String

Dim category As String

Dim Flags As Long

Dim action As String

List1.Clear

tcount = Vcommand1.CountCommands(gMymenu)

For i = 1 To tcount

Vcommand1.GetCommand gMymenu, i, command, description, category, Flags, action

List1.AddItem command

Next i

End Sub

Private Sub Dicho_Change()

If Dicho.Text = "forward" Then

Label1.Caption = "1"

End If

'*****'

If Dicho.Text = "backward" Then

Label1.Caption = "-1"

End If

'*****'

If Dicho.Text = "go" Then

Velocidad.Value = 60

Velocidad_Display.Text = "60"

End If

'*****'

If Dicho.Text = "stop" Then

Velocidad.Value = 0

Velocidad_Display.Text = "0"

End If

'*****'

If Dicho.Text = "left" Then

If VS1(1).Value < 222 Then

VS1(1).Value = VS1(1).Value + 20

End If

End If

*****'

If Dichotext = "right" Then

If VS1(1).Value > 22 Then

VS1(1).Value = VS1(1).Value - 20

End If

End If

End Sub

Private Sub Escucha_Click()

If Escucha.Caption = "Escucha" Then

Vcommand1.Activate gMymenu

Escucha.Caption = "No Escuches"

List1.Enabled = True

Else

Vcommand1.Deactivate gMymenu

Escucha.Caption = "Escucha"

List1.Enabled = False

End If

End Sub

Sub Leer_Posiciones()

Dim bb As Integer

Dim TmpValor As Byte

Dim Time_Out As Boolean

For bb = 0 To (VS1.Count - 1)

'Establecemos los Faders

If ModoRS232 = RS232 Then

If PuertoAbierto Then

'Leemos la posicion de cada servo

TmpValor = ModuloRs232.LeerPosicion(bb, False)

'Filtro por si no leemos la posicion

If TmpValor = 0 Then TmpValor = 1

'Establecemos el valor leido

VS1(bb).Value = TmpValor

Else

'Valor por defecto 1 (asi nos damos cuenta del fallo)

VS1(bb).Value = 1

End If

Else

If DetectBoard Then

'Leemos la posicion de cada servo

```
    TmpValor = ModuloUSB.LeerPosicion(bb)
    'Filtro por si no leemos la posicion
    If TmpValor = 0 Then TmpValor = 120
    'Establecemos el valor leido
    'VS1(bb).Value = TmpValor'
Else
    'Valor por defecto 1 (asi nos damos cuenta del fallo)
    VS1(bb).Value = 1
End If
End If
'Inicializamos los labels de valor de pista
Lval(bb).Caption = VS1(bb).Value
'Inicializamos los labels de valor de pista
LNum(bb).Caption = Trim(Str(bb + 1))
Next bb
End Sub

Private Sub Amplitud_Change()
Amplitud_Display = Amplitud.Value
End Sub

Private Sub Amplitud_Display_Change()
Amplitud.Value = Amplitud_Display
End Sub

Private Sub Desfase1_Change()
Desfase1_Display = Desfase1.Value
End Sub

Private Sub Desfase1_Display_Change()
Desfase1.Value = Desfase1_Display
End Sub

Private Sub Desfase2_Change()
Desfase2_Display = Desfase2.Value
End Sub

Private Sub Desfase2_Display_Change()
Desfase2.Value = Desfase2_Display
End Sub

Private Sub Option2_Click(Index As Integer)
If Option2(0).Value = True Then
```

```
Manual_o_no.Caption = "0"
```

```
Desfase1.Value = 0
```

```
Desfase1_Display = 0
```

```
Desfase2.Value = 90
```

```
Desfase2_Display = 90
```

```
End If
```

```
If Option2(1).Value = True Then
```

```
Manual_o_no.Caption = "1"
```

```
Desfase1.Value = 90
```

```
Desfase1_Display = 90
```

```
Desfase2.Value = 0
```

```
Desfase2_Display = 0
```

```
End If
```

```
If Option2(2).Value = True Then
```

```
Manual_o_no.Caption = "1"
```

```
Desfase1.Value = 90
```

```
Desfase1_Display = 90
```

```
Desfase2.Value = 180
```

```
Desfase2_Display = 180
```

```
End If
```

```
End Sub
```

```
Private Sub Tempo_Timer()
```

```
indice = indice + ((11 - Amplitud.Value / 10) * Label1.Caption)
```

```
Text6 = indice
```

```
radian = indice * PI / 180
```

```
Text2 = radian
```

```
Mot1 = (Amplitud.Value * Sin(radian)) + 132
```

```
Mot2 = (Amplitud.Value * Sin(radian + (Desfase1.Value * PI / 180)) * Manual_o_no.Caption) + 122
```

```
Mot3 = (Amplitud.Value * Sin(radian + (Desfase2.Value * PI / 180))) + 122
```

```
Texto_Motor(0).Text = Mot1
```

```
Texto_Motor(1).Text = Mot2
```

```
Texto_Motor(2).Text = Mot3
```

```
End Sub
```

```
Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
```

```
'*****'
```

```
If KeyCode = vbKeyRight Then
```

```
If VS1(1).Value > 22 Then
```

```
VS1(1).Value = VS1(1).Value - 10
```

```
End If
```

```
End If
```

Descripción del software desarrollado

*****'

```
If KeyCode = vbKeyLeft Then
If VS1(1).Value < 222 Then
VS1(1).Value = VS1(1).Value + 10
End If
End If
```

*****'

```
If KeyCode = vbKeyUp Then
Label1.Caption = "1"
End If
```

*****'

```
If KeyCode = vbKeyDown Then
Label1.Caption = "-1"
End If
```

*****'

```
End Sub

Private Sub Text1_KeyUp(KeyCode As Integer, Shift As Integer)
Select Case KeyCode
Case vbKeyUp
Label1.Caption = "0"
Case vbKeyDown
Label1.Caption = "0"
End Select
End Sub
```

```
Private Sub Texto_Motor_Change(Index As Integer)
Call Personalizado(Index)
End Sub
```

```
Private Sub Vcommand1_CommandRecognize(ByVal ID As Long, ByVal CmdName As String, ByVal Flags
As Long, ByVal action As String, ByVal NumLists As Long, ByVal ListValues As String, ByVal command As
String)
Dicho.Text = command
End Sub

Private Sub Form_Unload(Cancel As Integer)
Vcommand1.ReleaseMenu gMymenu
End Sub
```

```
Private Sub Velocidad_Change()
Velocidad_Display = Velocidad.Value
Velocidad_motor = Sgn(Velocidad) * (101 - Abs(Velocidad))
Tempo.Interval = Abs(Velocidad_motor)
Text1 = Velocidad_motor
```

Descripción del software desarrollado

End Sub

Private Sub Velocidad_Display_Change()

Velocidad.Value = Velocidad_Display

End Sub

Private Sub Command1_Click()

'Envia la posicion de los faders a todos los servos

If ModoRS232 = RS232 Then

Call ModuloRs232.EnviaPosicion

Else

Call ModuloUSB.EnviaPosicion

End If

End Sub

Private Sub Command2_Click()

Dim Canal As Integer

For Canal = 0 To (VS1.Count - 1)

'Establecemos la posicion de inicio a cada canal

If ModoRS232 = RS232 Then

Call ModuloRs232.EnviaComandoDeInicio(Canal, VS1(Canal).Value)

Else

Call ModuloUSB.EnviaComandoDeInicio(Canal, VS1(Canal).Value)

End If

Next Canal

End Sub

Private Sub Command3_Click()

'Presentamos el formulario del puerto serie

FrmPuerto.Show vbModal

End Sub

Private Sub Form_Load()

'CAMARA STUFF'

mCapHwnd = capCreateCaptureWindow("WebcamCapture", 0, 0, 0, 640, 480, Me.hwnd, 0)

DoEvents

SendMessage mCapHwnd, CONNECT, 0, 0

Timer10.Enabled = True

'CAMARA STUFF'

indice = 0

'Leemos los parametros de inicio

' Cargamos el modo de comunicacion (USB=false, RS232=True)

Descripción del software desarrollado

```
ModoRS232 = Val(Funciones.LeeINI("Conf.ini", "Settings", "Modo", "0"))
'Puerto seleccionado del RS232
Puerto = Val(LeeINI("Conf.ini", "Settings", "Puerto", "1"))
Vel_RS232 = LeeINI("Conf.ini", "Settings", "Baudios", "9600")
' Segun el modo de comunicacion selecciona los items y abrimos el puerto
If ModoRS232 = RS232 Then
    Option1(1).Value = True    'Rs232
    PuertoAbierto = ModuloRs232.AbrirPuerto(True) 'Abrimos el puerto serie
Else
    Option1(0).Value = True    'Usb
    Call ModuloUSB.DetectarUSB(True)          'Buscamos la controladora por USB
End If
'Presenta el formulario
Me.Show
'Leemos las posiciones y cargamos los controles
Call Me.Leer_Posiciones
'Esto da permiso para modificar el valor a los faders
Interactuar = True
'CONTROL POR VOZ'
Vcommand1.Initialized = 1
gMymenu = Vcommand1.MenuCreate(App.EXENAME, "state1", 4)
Vcommand1.Enabled = 1
Vcommand1.AddCommand gMymenu, 1, "right", "when you say" + "right", "listen list", 0, ""
Vcommand1.AddCommand gMymenu, 1, "left", "when you say" + "left", "listen list", 0, ""
Vcommand1.AddCommand gMymenu, 1, "forward", "when you say" + "forward", "listen list", 0, ""
Vcommand1.AddCommand gMymenu, 1, "backward", "when you say" + "backward", "listen list", 0, ""
Vcommand1.AddCommand gMymenu, 1, "go", "when you say" + "go", "listen list", 0, ""
Vcommand1.AddCommand gMymenu, 1, "stop", "when you say" + "stop", "listen list", 0, ""
UpdateList
Vcommand1.Activate gMymenu
'CONTROL POR VOZ'
End Sub

Private Sub Option1_Click(Index As Integer)
'Permiso para poder cambiar las opciones
If Interactuar Then
    'Valor de la variable de modo
    ModoRS232 = Abs(CBool(Option1(0).Value))
    ' Guardamos el modo de comunicacion (USB=false, RS232=True)
    Call Funciones.EscribeINI("Conf.ini", "Settings", "Modo", Trim(Str(ModoRS232)))
    MsgBox "Inicialice su controladora y pulse aceptar...", vbInformation, "Inicialización..."
    'Abrimos el puerto seleccionado e intentamos comunicar
```

```
If ModoRS232 = RS232 Then
    PuertoAbierto = ModuloRs232.AbrirPuerto(True) 'Abrimos el puerto serie
Else
    Call ModuloUSB.CerrarUsb
    Do While Funciones.Esperar(1)
        Loop
        Call ModuloUSB.DetectarUSB(True)          'Buscamos la controladora por el USB
    End If
    'Leemos las posiciones
    Call Me.Leer_Posiciones
End If
End Sub
```

```
Private Sub VS1_Change(Index As Integer)
    Call VS1_Scroll(Index)
End Sub
```

```
Private Sub VS1_Scroll(Index As Integer)
    'Presentamos el valor en el label del canal correspondiente
    Lval(Index).Caption = VS1(Index).Value
    'Permiso de escritura al USB
    If Interactuar Then
        'Si el check1 esta habilitado no dejara enviar la posicion
        If Check1.Value = 0 Then
            If ModoRS232 = RS232 Then
                'Enviamos el valor al USB (Numero de servo, posicion)
                Call ModuloRs232.EnviarPosicionServo(Index, VS1(Index).Value)
            Else
                'Enviamos el valor al USB (Numero de servo, posicion)
                Call ModuloUSB.EnviarPosicionServo(Index, VS1(Index).Value)
            End If
        End If
    End If
End Sub
```

```
Private Sub Personalizado(Index As Integer)
    'Presentamos el valor en el label del canal correspondiente
    'Permiso de escritura al USB
    If Interactuar Then
        'Si el check1 esta habilitado no dejara enviar la posicion
        If Check1.Value = 0 Then
            If ModoRS232 = RS232 Then
```

Descripción del software desarrollado

```
'Enviamos el valor al USB (Numero de servo, posicion)
Call ModuloRs232.EnviaPosicionServo(Index, (Texto_Motor(Index).Text))
Else
'Enviamos el valor al USB (Numero de servo, posicion)
Call ModuloUSB.EnviaPosicionServo(Index, Texto_Motor(Index).Text)
End If
End If
End If
End Sub

'CAMARA STUFF'
Private Sub Timer10_Timer()
'Get the picture from camera.. the main part
SendMessage mCapHwnd, GET_FRAME, 0, 0
SendMessage mCapHwnd, COPY, 0, 0
Picture1.Picture = Clipboard.GetData
Clipboard.Clear
LastTime = GetTickCount
End Sub

Sub STOPCAM()
DoEvents: SendMessage mCapHwnd, DISCONNECT, 0, 0
Timer10.Enabled = False
End Sub
'CAMARA STUFF'
```

C.2 Programa para el control del robot teleoperado

Tras la experiencia conseguida con el programa destinado al control del primer prototipo mediante la voz, se decidió realizar un programa más estructurado y capaz de controlar un robot compuesto por gran cantidad de articulaciones.

Este hecho unido al cambio de hardware (ya que hay que recordar que la versión final del robot se controla con la tarjeta Pololu en lugar de la JVM utilizada en la primera versión) determinaron la completa reescritura del programa de control.

El nuevo programa presenta una interfaz de usuario como la mostrada en la figura C.4.

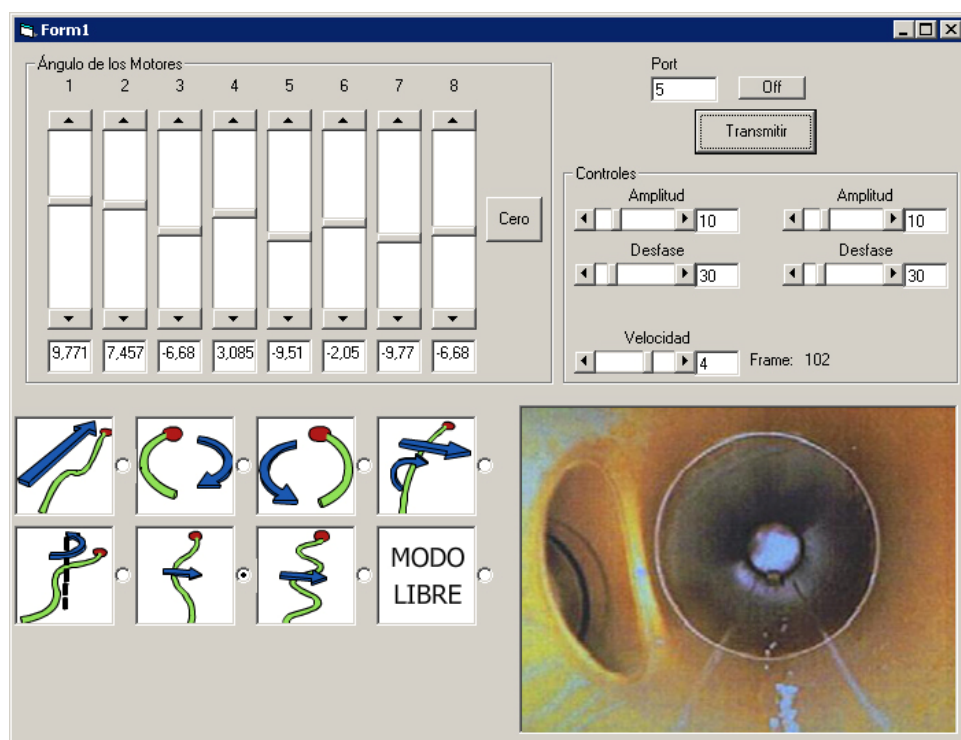


Figura C.4 Interfaz principal del programa

Puesto que la funcionalidad de cada elemento que se muestra por pantalla se describe en la sección 4.1, el análisis del programa se centrará en el código empleado y su estructura, dicha estructura es la que se muestra en la Figura C.5

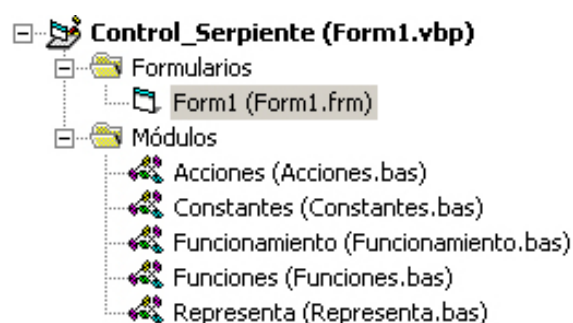


Figura C.5 Esquema del programa

En este programa hay que destacar el intento por conseguir modularidad y claridad en la programación, esto puede verse en su estructura, presenta gran cantidad de módulos encargados de realizar cada una de las subrutinas como el control de las barras deslizantes, el cálculo de las posiciones de los servomotores o la comunicación puerto serie.

El hecho de que solamente sea necesario establecer comunicación mediante puerto serie simplifica enormemente el código a utilizar comparado con el programa desarrollado para el control por voz, donde para el uso del puerto USB fue necesario adaptar rutinas y códigos preexistentes en repositorios de Internet, los cuales no siempre se encuentran libres de fallos.

A continuación se muestran los códigos correspondientes a cada uno de los módulos del programa y al formulario principal:

C.2.1 Módulo *Representa*

```
Public Sub Cambia_Slider_Motor(Index)
```

```
If Index = 0 Then
```

```
Form1.Motor_Display(Index).Text = ((Form1.Motor(Index).value - 2325) / 1750) * 90
```

```
ElseIf Index = 1 Then
```

```
Form1.Motor_Display(Index).Text = ((Form1.Motor(Index).value - 2700) / 1750) * 90
```

```
ElseIf Index = 2 Then
```

```
Form1.Motor_Display(Index).Text = ((Form1.Motor(Index).value - 2625) / 1750) * 90
```

```
ElseIf Index = 3 Then
```

```
Form1.Motor_Display(Index).Text = ((Form1.Motor(Index).value - 2625) / 1750) * 90
```

```
ElseIf Index = 4 Then
```

```
Form1.Motor_Display(Index).Text = ((Form1.Motor(Index).value - 2625) / 1750) * 90
```

```
ElseIf Index = 5 Then
```

```
Form1.Motor_Display(Index).Text = ((Form1.Motor(Index).value - 2775) / 1750) * 90
```

```
ElseIf Index = 6 Then
```

```
Form1.Motor_Display(Index).Text = ((Form1.Motor(Index).value - 2575) / 1750) * 90
```

```
ElseIf Index = 7 Then
```

```
Form1.Motor_Display(Index).Text = ((Form1.Motor(Index).value - 2575) / 1750) * 90
```

```
End If
```

```
End Sub
```

```
Public Sub Cambia_Display_Motor(Index)
```

```
If Form1.Motor_Display(Index).Text > 45 Then
```

```
Form1.Motor_Display(Index).Text = 45
```

```
ElseIf Form1.Motor_Display(Index).Text < -45 Then
```

```
Form1.Motor_Display(Index).Text = -45
```

```
Else
```

```
If Index = 0 Then
```

```
Form1.Motor(Index).value = Math.Round(Form1.Motor_Display(Index).Text * 1750 / 90 + 2325)
```

```
ElseIf Index = 1 Then
```

```
Form1.Motor(Index).value = Math.Round(Form1.Motor_Display(Index).Text * 1750 / 90 + 2700)
    ElseIf Index = 2 Then
Form1.Motor(Index).value = Math.Round(Form1.Motor_Display(Index).Text * 1750 / 90 + 2625)
    ElseIf Index = 3 Then
Form1.Motor(Index).value = Math.Round(Form1.Motor_Display(Index).Text * 1750 / 90 + 2625)
    ElseIf Index = 4 Then
Form1.Motor(Index).value = Math.Round(Form1.Motor_Display(Index).Text * 1750 / 90 + 2625)
    ElseIf Index = 5 Then
Form1.Motor(Index).value = Math.Round(Form1.Motor_Display(Index).Text * 1750 / 90 + 2775)
    ElseIf Index = 6 Then
Form1.Motor(Index).value = Math.Round(Form1.Motor_Display(Index).Text * 1750 / 90 + 2575)
    ElseIf Index = 7 Then
Form1.Motor(Index).value = Math.Round(Form1.Motor_Display(Index).Text * 1750 / 90 + 2575)
    End If
End If
End Sub
```

```
Public Sub Amplitud_Horizontal_Cambio_Display()
    If Form1.Amplitud_Horizontal_Display.Text > 45 Then
Form1.Amplitud_Horizontal_Display.Text = 45
    ElseIf Form1.Amplitud_Horizontal_Display.Text < 0 Then
Form1.Amplitud_Horizontal_Display.Text = 0
    Else
Form1.Amplitud_Horizontal.value = Math.Round(Form1.Amplitud_Horizontal_Display.Text)
    End If
End Sub
```

```
Public Sub Amplitud_Horizontal_Cambia()
    Form1.Amplitud_Horizontal_Display.Text = Form1.Amplitud_Horizontal.value
End Sub
```

```
Public Sub Amplitud_Vertical_Cambio_Display()
    If Form1.Amplitud_Vertical_Display.Text > 45 Then
Form1.Amplitud_Vertical_Display.Text = 45
    ElseIf Form1.Amplitud_Vertical_Display.Text < 0 Then
Form1.Amplitud_Vertical_Display.Text = 0
    Else
Form1.Amplitud_Vertical.value = Math.Round(Form1.Amplitud_Vertical_Display.Text)
    End If
End Sub
```

```
Public Sub Amplitud_Vertical_Cambia()
    Form1.Amplitud_Vertical_Display.Text = Form1.Amplitud_Vertical.value
```

End Sub

Public Sub Desfase_Horizontal_Cambio()

Form1.Desfase_Horizontal_Display.Text = Form1.Desfase_Horizontal.value

End Sub

Public Sub Desfase_Horizontal_Cambio_Display()

If Form1.Desfase_Horizontal_Display.Text > 180 Then

Form1.Desfase_Horizontal_Display.Text = 180

ElseIf Form1.Desfase_Horizontal_Display.Text < 0 Then

Form1.Desfase_Horizontal_Display.Text = 0

Else

Form1.Desfase_Horizontal.value = Math.Round(Form1.Desfase_Horizontal_Display.Text)

End If

End Sub

Public Sub Desfase_Vertical_Cambio()

Form1.Desfase_Vertical_Display.Text = Form1.Desfase_Vertical.value

End Sub

Public Sub Desfase_Vertical_Cambio_Display()

If Form1.Desfase_Vertical_Display.Text > 180 Then

Form1.Desfase_Vertical_Display.Text = 180

ElseIf Form1.Desfase_Vertical_Display.Text < 0 Then

Form1.Desfase_Vertical_Display.Text = 0

Else

Form1.Desfase_Vertical.value = Math.Round(Form1.Desfase_Vertical_Display.Text)

End If

End Sub

Public Sub Velocidad_Cambio_Display()

If Form1.Velocidad_Display.Text > 10 Then

Form1.Velocidad_Display.Text = 10

ElseIf Form1.Velocidad_Display.Text < -10 Then

Form1.Velocidad_Display.Text = -10

Else

Form1.Velocidad.value = Math.Round(Form1.Velocidad_Display.Text)

End If

If Form1.Velocidad_Display.Text = 0 Then

Form1.Tic.Interval = 0

Else: Form1.Tic.Interval = Math.Abs(100 / Form1.Velocidad_Display.Text)

End If

End Sub

Public Sub Velocidad_Cambio()

Form1.Velocidad_Display.Text = Form1.Velocidad.value

End Sub

C.2.2 Módulo *Funciones*

Function SendPololu(num As Integer, value As Integer)

' num is 0-7

' value is 500-5500

' Asc(Mid(cmd, 1, 1)) & "." & Asc(Mid(cmd, 2, 1)) & "." & Asc(Mid(cmd, 3, 1)) & "." & Asc(Mid(cmd, 4, 1))
& "." & Asc(Mid(cmd, 5, 1)) & "." & Asc(Mid(cmd, 6, 1))

If Form1.MSComm1.PortOpen = True Then

cmd = Chr(&H80) ' start byte - 0x80 ' always

cmd = cmd & Chr(&H1) ' Device ID - 0x01 ' always

cmd = cmd & Chr(&H4) ' command - 0x04 ' we want command 4: Set Position, Absolute (2 data bytes)

cmd = cmd & Chr(num) ' servo num - 0x.. ' 00-07

' data1 - 0x.. ' upper 7 bits - range is 500 through 5500

' data2 - 0x.. ' lower 7 bits

strbinary = "0000000000000000" & dec2bin(value)

lsb = "0" & Mid(strbinary, Len(strbinary) - 6)

msb = "0" & Mid(strbinary, Len(strbinary) - 12, 6)

cmd = cmd & Chr(bin2dec(msb))

cmd = cmd & Chr(bin2dec(lsb))

Form1.MSComm1.Output = cmd & vbNewLine

' Sleep 60

DoEvents

End If

End Function

Function hex2dec(sHexStr) As Integer

hex2dec = CLng("&H" & Trim(sHexStr))

End Function

Function dec2hex(val As Integer) As String

dec2hex = Hex(val)

End Function

Function pad(val As String, Optional length As Integer = 1, Optional char As String = "0") As String

' pads a string with chars until len = length

Do While Len(val) < length

val = char & val

```
Loop
pad = val
End Function
```

```
Function dec2bin(mynum As Variant) As String
' from http://cuinl.tripod.com/Tips/decimaltobinary.htm
Dim loopcounter As Integer
If mynum >= 2 ^ 31 Then
    dec2bin = "Number too big"
    Exit Function
End If
Do
    If (mynum And 2 ^ loopcounter) = 2 ^ loopcounter Then
        dec2bin = "1" & dec2bin
    Else
        dec2bin = "0" & dec2bin
    End If
    loopcounter = loopcounter + 1
Loop Until 2 ^ loopcounter > mynum
End Function
```

```
Function bin2dec(ByVal BinValue As String) As Long
' from http://cuinl.tripod.com/Tips/math5.htm
Dim lngValue As Long
Dim X As Long
Dim k As Long
k = Len(BinValue) ' will only work with 32 or fewer "bits"
For X = k To 1 Step -1 ' work backwards down string
    If Mid$(BinValue, X, 1) = "1" Then
        If k - X > 30 Then ' bit 31 is the sign bit
            lngValue = lngValue Or -2147483648# ' avoid overflow error
        Else
            lngValue = lngValue + 2 ^ (k - X)
        End If
    End If
Next X
bin2dec = lngValue
End Function
```

C.2.3 Módulo *Funcionamiento*

Public Sub Modo_Funcionamiento(Index)

 If Form1.Modo_Funcion(0).value = True Then

 Form1.Velocidad_Display.Text = 0

 Form1.Tiempo.Caption = 0

 Form1.Amplitud_Horizontal_Display.Text = 20

 Form1.Desfase_Horizontal_Display.Text = 120

 Form1.Amplitud_Vertical_Display.Text = 0

 Form1.Desfase_Vertical_Display.Text = 0

 Form1.Modo.Caption = 0

 End If

 If Form1.Modo_Funcion(1).value = True Then

 Form1.Velocidad_Display.Text = 0

 Form1.Tiempo.Caption = 0

 Form1.Amplitud_Horizontal_Display.Text = 20

 Form1.Desfase_Horizontal_Display.Text = 120

 Form1.Motor_Display(2).Text = 45

 Form1.Motor_Display(4).Text = 45

 Form1.Motor_Display(6).Text = 45

 Form1.Modo.Caption = 1

 End If

 If Form1.Modo_Funcion(2).value = True Then

 Form1.Velocidad_Display.Text = 0

 Form1.Tiempo.Caption = 0

 Form1.Amplitud_Horizontal_Display.Text = 20

 Form1.Desfase_Horizontal_Display.Text = 120

 Form1.Motor_Display(2).Text = -45

 Form1.Motor_Display(4).Text = -45

 Form1.Motor_Display(6).Text = -45

 Form1.Modo.Caption = 2

 End If

 If Form1.Modo_Funcion(3).value = True Then

 Form1.Velocidad_Display.Text = 0

 Form1.Tiempo.Caption = 0

 Form1.Amplitud_Horizontal_Display.Text = 45

 Form1.Desfase_Horizontal_Display.Text = 60

 Form1.Amplitud_Vertical_Display.Text = 20

 Form1.Desfase_Vertical_Display.Text = 120

 Form1.Modo.Caption = 3

 End If

 If Form1.Modo_Funcion(4).value = True Then

 Form1.Velocidad_Display.Text = 0

```
Form1.Tiempo.Caption = 0
Form1.Amplitud_Horizontal_Display.Text = 10
Form1.Desfase_Horizontal_Display.Text = 30
Form1.Amplitud_Vertical_Display.Text = 10
Form1.Desfase_Vertical_Display.Text = 30
Form1.Modo.Caption = 4
End If
If Form1.Modo_Funcion(5).value = True Then
    Form1.Velocidad_Display.Text = 0
    Form1.Tiempo.Caption = 0
    Form1.Amplitud_Horizontal_Display.Text = 45
    Form1.Desfase_Horizontal_Display.Text = 60
    Form1.Amplitud_Vertical_Display.Text = 20
    Form1.Desfase_Vertical_Display.Text = 120
    Form1.Modo.Caption = 5
End If
If Form1.Modo_Funcion(6).value = True Then
    Form1.Velocidad_Display.Text = 0
    Form1.Tiempo.Caption = 0
    Form1.Amplitud_Horizontal_Display.Text = 20
    Form1.Desfase_Horizontal_Display.Text = 30
    Form1.Amplitud_Vertical_Display.Text = 20
    Form1.Desfase_Vertical_Display.Text = 30
    Form1.Modo.Caption = 6
End If
If Form1.Modo_Funcion(7).value = True Then
    Form1.Velocidad_Display.Text = 0
    Form1.Tiempo.Caption = 0
    Form1.Amplitud_Horizontal_Display.Text = 0
    Form1.Desfase_Horizontal_Display.Text = 0
    Form1.Amplitud_Vertical_Display.Text = 0
    Form1.Desfase_Vertical_Display.Text = 0
    Form1.Modo.Caption = 7
End If
End Sub
```

C.2.4 Módulo *Constantes*

Global Const PI = 3.141592654

'PARA LA WEBCAM

Global mCapHwnd As Long

Global Const CONNECT As Long = 1034

Global Const DISCONNECT As Long = 1035

Global Const GET_FRAME As Long = 1084

Global Const COPY As Long = 1054

C.2.5 Módulo *Acciones*

Public Sub Puesta_a_Cero()

Form1.Motor_Display(0).Text = 0

Form1.Motor_Display(1).Text = 0

Form1.Motor_Display(2).Text = 0

Form1.Motor_Display(3).Text = 0

Form1.Motor_Display(4).Text = 0

Form1.Motor_Display(5).Text = 0

Form1.Motor_Display(6).Text = 0

Form1.Motor_Display(7).Text = 0

End Sub

Public Sub Pasa_Tiempo()

Form1.Tiempo.Caption = Form1.Tiempo.Caption + 5 * Form1.Velocidad

If Form1.Tiempo.Caption > 360 Then

Form1.Tiempo.Caption = Form1.Tiempo.Caption - 360

ElseIf Form1.Tiempo.Caption < -360 Then

Form1.Tiempo.Caption = Form1.Tiempo.Caption + 360

End If

End Sub

C.2.6 Formulario *Form1*

Private Declare Function SendMessage Lib "USER32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wParam As Long, ByVal lParam As Any) As Long

Private Declare Function capCreateCaptureWindow Lib "avicap32.dll" Alias "capCreateCaptureWindowA" (ByVal lpszWindowName As String, ByVal dwStyle As Long, ByVal X As Long, ByVal Y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hwndParent As Long, ByVal nID As Long) As Long

Private Sub cmdOnOff_Click()

If Form1.cmdOnOff.Caption = "Off" Then

```
If Form1.MSComm1.PortOpen = True Then
    Form1.MSComm1.PortOpen = False
End If
Form1.MSComm1.CommPort = Form1.txtPort.Text
Form1.cmdOnOff.Caption = "On"
Else
    If Form1.MSComm1.PortOpen = True Then
        Form1.MSComm1.PortOpen = False
    End If
    Form1.MSComm1.PortOpen = True
    Form1.cmdOnOff.Caption = "Off"
End If
End Sub

Private Sub Form_Load()
    MSComm1.CommPort = 5
    Camara.Width = 320 * Screen.TwipsPerPixelX
    Camara.Height = 240 * Screen.TwipsPerPixelY
    STARTCAM
End Sub

Sub STARTCAM()
    mCapHwnd = capCreateCaptureWindow("WebcamCapture", 0, 0, 0, 640, 480, Me.hwnd, 0)
    DoEvents
    SendMessage mCapHwnd, CONNECT, 0, 0
End Sub

Private Sub Transmission_Click()
    If Transmission.Caption = "Transmitir" Then
        Tic.Enabled = True
        Transmission.Caption = "No Transmitir"
    Else
        Tic.Enabled = False
        Transmission.Caption = "Transmitir"
    End If
End Sub

Private Sub Motor_Change(Index As Integer)
    Call Representa.Cambia_Slider_Motor(Index)
End Sub

Private Sub Motor_Display_Change(Index As Integer)
    Call Representa.Cambia_Display_Motor(Index)
```

Descripción del software desarrollado

End Sub

Private Sub Amplitud_Horizontal_Change()

 Call Representa.Amplitud_Horizontal_Cambia

End Sub

Private Sub Amplitud_Horizontal_Display_Change()

 Call Representa.Amplitud_Horizontal_Cambio_Display

End Sub

Private Sub Amplitud_Vertical_Change()

 Call Representa.Amplitud_Vertical_Cambia

End Sub

Private Sub Amplitud_Vertical_Display_Change()

 Call Representa.Amplitud_Vertical_Cambio_Display

End Sub

Private Sub Velocidad_Change()

 Call Representa.Velocidad_Cambio

End Sub

Private Sub Velocidad_Display_Change()

 Call Representa.Velocidad_Cambio_Display

End Sub

Private Sub Desfase_Horizontal_Change()

 Call Representa.Desfase_Horizontal_Cambio

End Sub

Private Sub Desfase_Horizontal_Display_Change()

 Call Representa.Desfase_Horizontal_Cambio_Display

End Sub

Private Sub Desfase_Vertical_Change()

 Call Representa.Desfase_Vertical_Cambio

End Sub

Private Sub Desfase_Vertical_Display_Change()

 Call Representa.Desfase_Vertical_Cambio_Display

End Sub

Private Sub Cero_Click()

 Call Acciones.Puesta_a_Cero

Descripción del software desarrollado

```
Tiempo.Caption = 0
End Sub

Private Sub Tic_Timer()
    Call Acciones.Pasa_Tiempo
    If Modo.Caption = 0 Then
        Motor_Display(0).Text = Senoide(Amplitud_Horizontal.value, -Tiempo.Caption, 0,
Desfase_Horizontal.value)
        Motor_Display(1).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 0, Desfase_Horizontal.value)
        Motor_Display(3).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 1, Desfase_Horizontal.value)
        Motor_Display(5).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 2, Desfase_Horizontal.value)
        Motor_Display(7).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 3, Desfase_Horizontal.value)
    End If
    If Modo.Caption = 1 Then
        Motor_Display(0).Text = Senoide(Amplitud_Horizontal.value, -Tiempo.Caption, 0,
Desfase_Horizontal.value)
        Motor_Display(1).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 0, Desfase_Horizontal.value)
        Motor_Display(3).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 1, Desfase_Horizontal.value)
        Motor_Display(5).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 2, Desfase_Horizontal.value)
        Motor_Display(7).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 3, Desfase_Horizontal.value)
    End If
    If Modo.Caption = 2 Then
        Motor_Display(0).Text = Senoide(Amplitud_Horizontal.value, -Tiempo.Caption, 0,
Desfase_Horizontal.value)
        Motor_Display(1).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 0, Desfase_Horizontal.value)
        Motor_Display(3).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 1, Desfase_Horizontal.value)
        Motor_Display(5).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 2, Desfase_Horizontal.value)
        Motor_Display(7).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 3, Desfase_Horizontal.value)
    End If
    If Modo.Caption = 3 Then
        Motor_Display(0).Text = Senoide(Amplitud_Horizontal.value, -Tiempo.Caption, 0,
Desfase_Horizontal.value)
        Motor_Display(1).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 0, Desfase_Horizontal.value)
        Motor_Display(2).Text = Senoide(Amplitud_Vertical.value, Tiempo.Caption, 0, Desfase_Vertical.value)
        Motor_Display(3).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 1, Desfase_Horizontal.value)
        Motor_Display(4).Text = Senoide(Amplitud_Vertical.value, Tiempo.Caption, 1, Desfase_Vertical.value)
        Motor_Display(5).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 2, Desfase_Horizontal.value)
        Motor_Display(6).Text = Senoide(Amplitud_Vertical.value, Tiempo.Caption, 2, Desfase_Vertical.value)
        Motor_Display(7).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 3, Desfase_Horizontal.value)
    End If
    If Modo.Caption = 4 Then
        Motor_Display(0).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 0, Desfase_Horizontal.value)
        Motor_Display(1).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 1, Desfase_Horizontal.value)
```

```
Motor_Display(2).Text = Senoide2(Amplitud_Vertical.value, Tiempo.Caption, 1, Desfase_Vertical.value)
Motor_Display(3).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 2, Desfase_Horizontal.value)
Motor_Display(4).Text = Senoide2(Amplitud_Vertical.value, Tiempo.Caption, 2, Desfase_Vertical.value)
Motor_Display(5).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 3, Desfase_Horizontal.value)
Motor_Display(6).Text = Senoide2(Amplitud_Vertical.value, Tiempo.Caption, 3, Desfase_Vertical.value)
Motor_Display(7).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 4, Desfase_Horizontal.value)
End If
If Modo.Caption = 5 Then
    Motor_Display(0).Text = Senoide(Amplitud_Horizontal.value, -Tiempo.Caption, 0, Desfase_Horizontal.value)
    Motor_Display(1).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 0, Desfase_Horizontal.value)
    Motor_Display(2).Text = Senoide2(Amplitud_Vertical.value, Tiempo.Caption, 0, Desfase_Vertical.value)
    Motor_Display(3).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 1, Desfase_Horizontal.value)
    Motor_Display(4).Text = Senoide2(Amplitud_Vertical.value, Tiempo.Caption, 1, Desfase_Vertical.value)
    Motor_Display(5).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 2, Desfase_Horizontal.value)
    Motor_Display(6).Text = Senoide2(Amplitud_Vertical.value, Tiempo.Caption, 2, Desfase_Vertical.value)
    Motor_Display(7).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 3, Desfase_Horizontal.value)
End If
If Modo.Caption = 6 Then
    Motor_Display(0).Text = Senoide(Amplitud_Horizontal.value, -Tiempo.Caption, 0, Desfase_Horizontal.value)
    Motor_Display(1).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 0, Desfase_Horizontal.value)
    Motor_Display(2).Text = Senoide2(Amplitud_Vertical.value, Tiempo.Caption, 0, Desfase_Vertical.value)
    Motor_Display(3).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 1, Desfase_Horizontal.value)
    Motor_Display(4).Text = Senoide2(Amplitud_Vertical.value, Tiempo.Caption, 1, Desfase_Vertical.value)
    Motor_Display(5).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 2, Desfase_Horizontal.value)
    Motor_Display(6).Text = Senoide2(Amplitud_Vertical.value, Tiempo.Caption, 2, Desfase_Vertical.value)
    Motor_Display(7).Text = Senoide(Amplitud_Horizontal.value, Tiempo.Caption, 3, Desfase_Horizontal.value)
End If
If Modo.Caption = 7 Then
End If
SendPololu 0, Motor(0).value
SendPololu 1, Motor(1).value
SendPololu 2, Motor(2).value
SendPololu 3, Motor(3).value
SendPololu 4, Motor(4).value
SendPololu 5, Motor(5).value
SendPololu 6, Motor(6).value
SendPololu 7, Motor(7).value
'*****
'PARA REPRESENTAR LA IMAGEN DE LA CAMARA
SendMessage mCapHwnd, GET_FRAME, 0, 0
SendMessage mCapHwnd, COPY, 0, 0
```

Descripción del software desarrollado

```
Camara.Picture = Clipboard.GetData
```

```
Clipboard.Clear
```

```
'*****'
```

```
End Sub
```

```
Private Sub Modo_Funcion_Click(Index As Integer)
```

```
    Call Funcionamiento.Modo_Funcionamiento(Index)
```

```
End Sub
```

```
'SENOIDES PARA EL MOVIMIENTO*****'
```

```
Private Function Senoide(Amplitud, frame, N_Motor, Desfase)
```

```
    Senoide = Amplitud * Math.Sin(frame * PI / 180 + N_Motor * Desfase * PI / 180)
```

```
End Function
```

```
Private Function Senoide2(Amplitud, frame, N_Motor, Desfase)
```

```
    Senoide2 = Amplitud * Math.Cos(frame * PI / 180 + N_Motor * Desfase * PI / 180)
```

```
End Function
```



HARDWARE UTILIZADO

En este anexo se detallarán las características y toda la información disponible sobre los distintos elementos de hardware utilizados en el robot, tanto elementos electrónicos como sistemas de actuación

D.1 Tarjeta controladora JVM-Servo 28

Esta tarjeta controladora se empleó para manejar la primera generación de robot serpiente, su principal característica es que es capaz de controlar hasta 28 servomotores simultáneamente y presenta el aspecto mostrado en la Figura D.1.



Figura D.1 Aspecto de la tarjeta controladora JVM-Servo 28

Según la descripción de su desarrollador: *“Esta herramienta es ideal para la animación con servo-motores, creación de robots y/o proyectos diversos donde se requiere sincronizar la posición de hasta 28 servos de RC simultáneamente.”*

Esta controladora es capaz de comunicarse tanto por puerto serie como por USB y reconoce por cual de ellos le llegan los datos. El rango de trabajo de los servos es de 400 a 2400us a 50Hz de forma que se le puede conectar cualquier servo standard con la norma de conexión Futaba (señal-positivo-negativo).

No necesita instalación de drivers de ningún tipo, viene implementado el HID en el propio PIC y el sistema operativo Windows la detecta automáticamente.

Características de la controladora de servos:

- Testeada en Windows XP (Home y Pro), Windows Vista, Windows 2000 y 2003 Server
- Versión 2.0 del USB compatible
- No requiere alimentación en la parte lógica (electrónica) cuando esta conectada al USB
- Circuito de alimentación externo protegido contra inversiones de polaridad (para la conexión RS232)
- Alimentación externa a partir de 7.5 hasta 30v. max.
- Velocidad RS232 configurable por software
- Velocidades RS232 soportadas: 9600 (defecto) 19200, 38400, 57600 y 115200 baudios.
- Librerías de desarrollo en Visual Basic 6 y Visual Basic 2005

JVM-SERVO

HARDWARE USB/RS232 (28 servos)

MANUAL DE MONTAJE



Diseño de: J.V.M

INDICE

1- MONTAJE DEL HARDWARE.....	PAG 2
▪ LISTA DE MATERIALES	
▪ INSTRUCCIONES PARA EL MONTAJE	
▪ PINEADO DEL MICROCONTROLADOR	
2- ESQUEMA ELECTRONICO.....	PAG 3
3- CIRCUITO IMPRESO.....	PAG 4
▪ CONEXIONADO	
▪ DIAGRAMA DEL CABLEADO Y CONEXIONES	
4- FUNCIONES IMPLEMENTADAS.....	PAG 6
5- PUESTA EN MARCHA.....	PAG 7

1-MONTAJE DEL HARDWARE

LISTA DE MATERIALES

- R1 - 10k 1/4 w.
- R2 - 330 ohm. 1/4 w.
- C1 - condensador de 470nf/100v (plaqueta)
- C3 - condensador de 100nf/100v (plaqueta)
- C4, C5 - condensadores de 22p (disco)
- C6, C7, C8, C9 - condensadores de 10uf/25v (electrolíticos)
- Q1 - Cristal de cuarzo de 12 MHz
- LED1 - diodo led de 3mm
- IC1 - Microcontrolador PIC18F4550
- IC2 - Conversor RS232-TTL MAX232CPE
- IC3 - Regulador LM7805
- D1 - Diodo 1N4007
- USB - conector usb de tipo b para c.i.
- SERIE - conector hembra DB9 para c.i.
- X2 - regleta de ci de 2 pins tipo clema
- 28 - contactos de 3 pins (conectores servos)
- 1 - contacto de 2 pins (Alimentación +5/9v)
- zócalo de 40 pins
- Circuito impreso serigrafiado

INSTRUCCIONES PARA EL MONTAJE

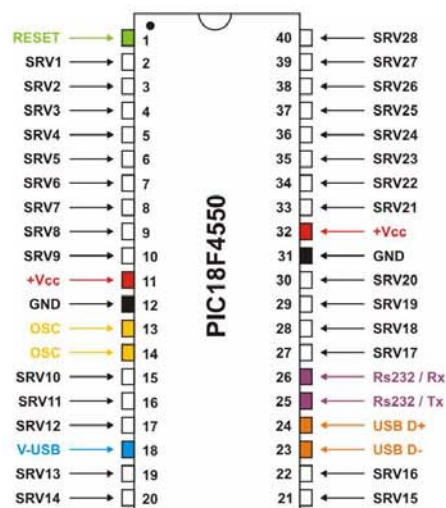
El montaje de los componentes es muy sencillo y para un correcto ensamblaje del kit de piezas recomendamos disponer de:

- Alicates de corte para electrónica
- Alicates de punta para la manipulación de los pines
- Estaño en relación 60/40
- Soldador de lápiz de 30W

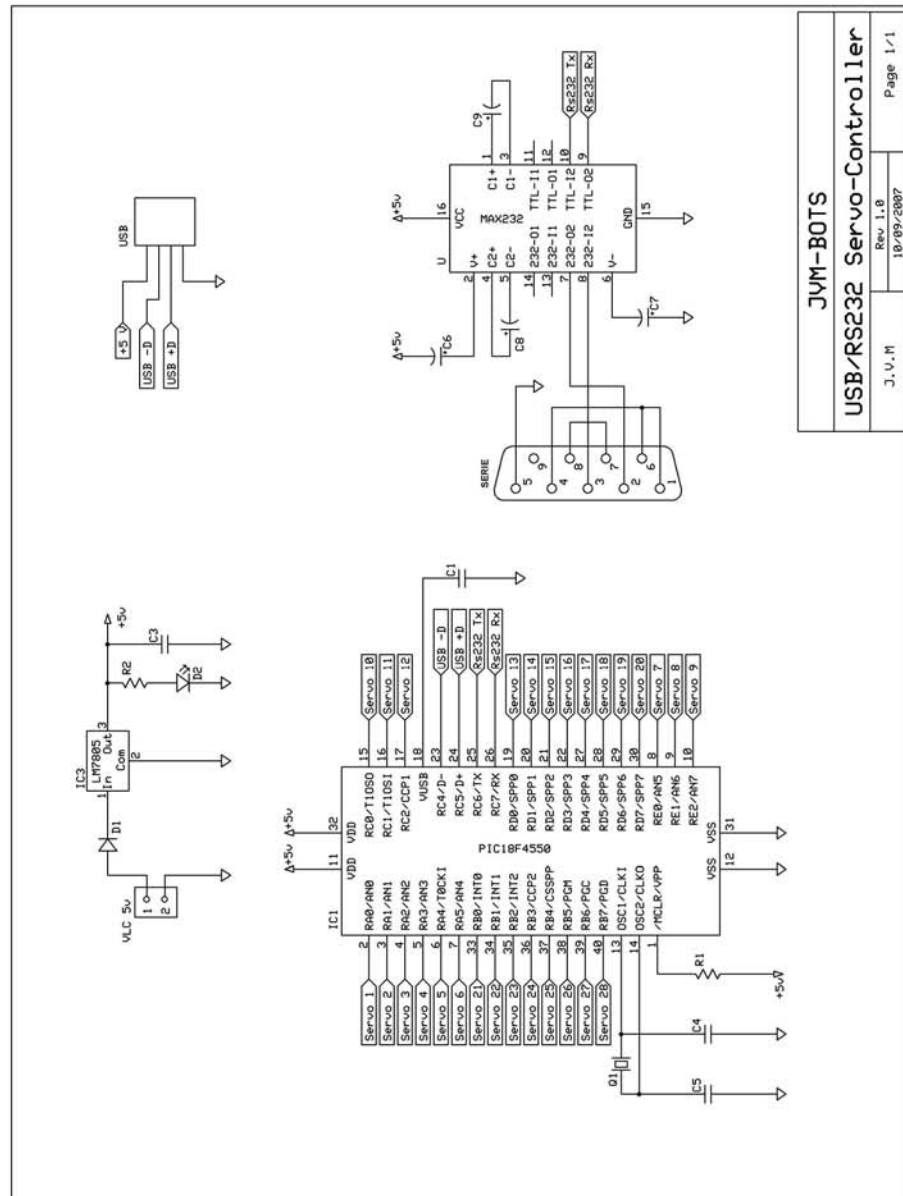
Colocar el zócalo del circuito integrado de 40 pines (si lo desea) o en su defecto el circuito integrado.

Soldar el resto de componentes pasivos, resistencias, condensadores, terminales de contacto tal y como se muestra en la lista de componentes y la imagen inferior.

PINEADO DEL MICROCONTROLADOR

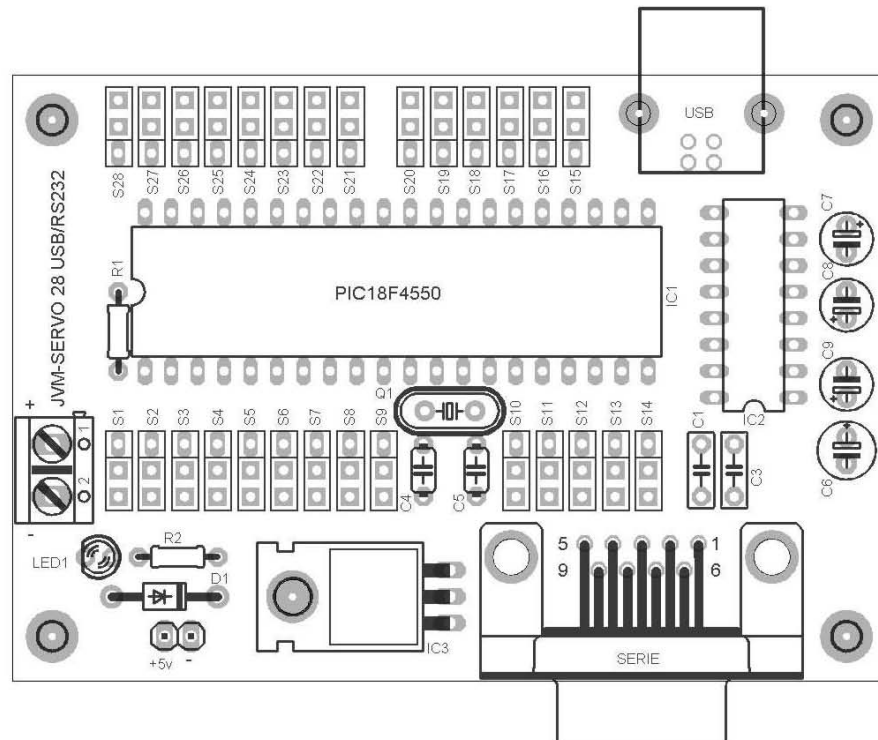


2-ESQUEMA ELECTRONICO



3-CIRCUITO IMPRESO

IDENTIFICACION DE COMPONENTES POR LA VISTA SUPERIOR



CONEXIONADO

La alimentación del microcontrolador está separada de la alimentación de los servos. Para los servos existe un terminal de tipo regleta en el cual suministraremos la tensión adecuada según el modelo de servo que empleemos en nuestro proyecto. Es aconsejable utilizar servos similares para evitar consumos y alteraciones de movimiento inesperados.

La parte de lógica de la controladora es alimentada de dos formas: Cuando está **conectada al puerto USB** del PC y por medio del **circuito de alimentación estabilizado**. Este circuito de alimentación permite que se alimente con una tensión superior o igual a 7.5v hasta un máximo de 30v.

El circuito de alimentación estabilizado también está provisto de un diodo en la entrada, de esta forma está **protegido de posibles inversiones de polaridad**.

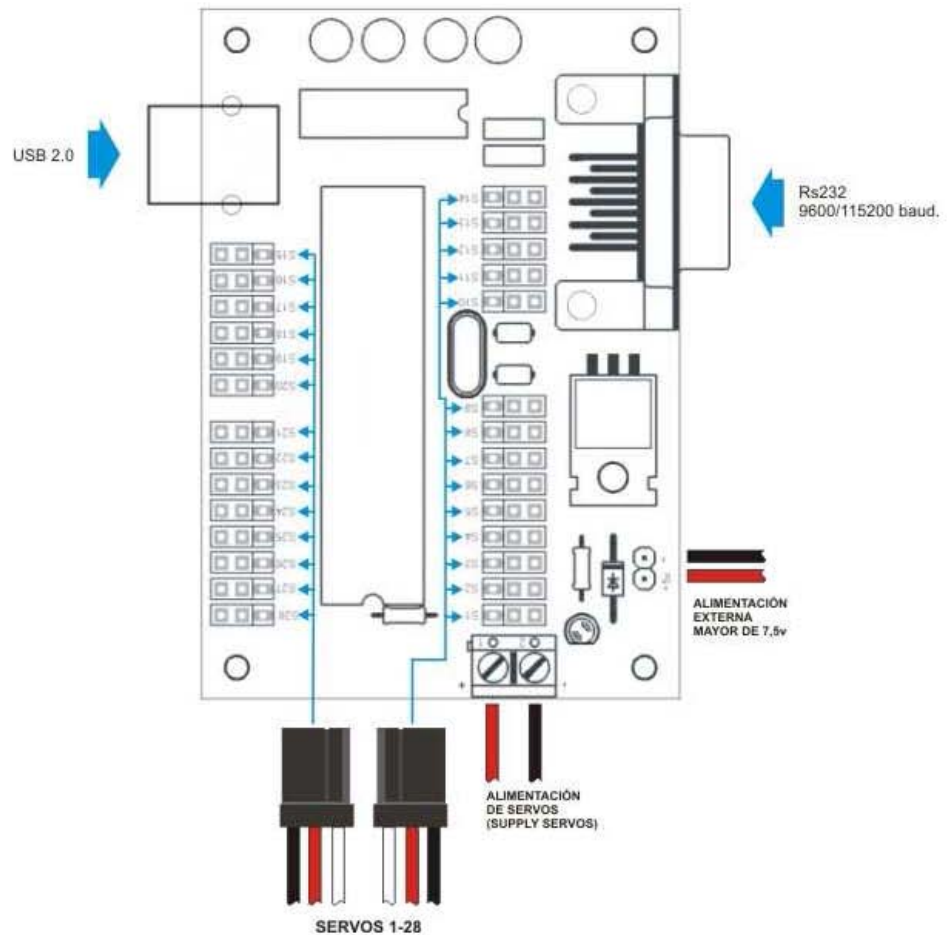
Se empleará el circuito de alimentación estabilizado cuando utilicemos la controladora para ser comandada por el puerto RS232, ya que cuando el microcontrolador es conectado por el USB al PC se alimenta gracias al suministro de energía que le proporciona el propio USB y de esta forma no es necesario tener conectada la alimentación hacia el circuito de alimentación estabilizado.

Alimentado de cualquiera de las formas en el circuito se iluminará el led LED1 indicándonos que ya tiene suministro de corriente, en el caso de estar conectado por USB al PC por primera vez, el dispositivo tendrá que ser detectado por Windows. Espere a que el sistema reconozca el nuevo dispositivo de hardware.

- ⚠ Extreme el cuidado** a la hora de conectar la alimentación a los servos, una inversión de polaridad puede dañarlos irreversiblemente.
- ⚠ Preste especial atención** en la posición de las conexiones de los servos, observe que el pin de señal está claramente identificado.

DIAGRAMA DEL CABLEADO Y CONEXIONES

En la imagen inferior podemos observar el posicionado de los conectores de los servos, las alimentaciones y los dos puertos de comunicaciones.



LEYENDA

- Cables rojos -> Alimentaciones positivas
- Cables negros -> Alimentaciones negativas
- Cables blancos -> señal del servo
- Flechas azules -> Comunicaciones

NOTA: Tenga cuidado con las posiciones del cableado, especialmente las alimentaciones y los conectores de los servos.

4-FUNCIONES IMPLEMENTADAS

PROPIEDADES HID (USB)

La controladora tiene los siguientes parámetros de identificación HID:

- HID= 1562
- VID= 2312

PROPIEDADES RS232 (Puerto serie)

La velocidad de comunicación serie de la controladora se configura por software mediante un comando de establecimiento. Las velocidades soportadas son:

- 9600 baudios
- 19200 baudios
- 38400 baudios
- 57600 baudios
- 115200 baudios

Los datos son todos enviados/recibidos a 8 bits sin paridad y un bit de stop (8N1).

FORMATO DE LOS COMANDOS

El formato de la comunicación esta definido por la lectura del primer byte de la trama, este primer byte decide el parámetro que se va a interpretar y establece la longitud de toda la trama de datos.

Los comandos son de dos tipos: normal de 3 bytes y extendido de 29 bytes.

Ejemplo para el comando normal:

Byte1 - Byte2 - Byte3

El Byte1 decide el comando, el segundo corresponde al número de servo y el tercero a la posición de dicho servo.

COMANDOS DE ESCRITURA (Tx)

1- Comando "S": Establecer la posición de un servo (3 bytes)

Byte1 = S
Byte2 = N° servo
Byte3 = Posición servo

2- Comando "I": Establecer la posición de inicio de un servo (3 bytes)

Byte1 = I
Byte2 = N° servo
Byte3 = posición servo

3- Comando "T": Establecer la posición a los 28 servos en un solo comando extendido (29 bytes)

Byte1 = T
Byte2 = posición servo nº 1
Byte3 = posición servo nº 2
Byte4 = posición servo nº 3
...
Byte29 = posición servo nº 28

4 -Cambio de velocidad del puerto serie

Byte1 = A
Byte2 = Velocidad

Velocidad corresponde al nuevo parámetro, estos son los valores:

- Velocidad 0 = 9.600 baudios
- Velocidad 1 = 19.200 baudios
- Velocidad 2 = 38.400 baudios
- Velocidad 3 = 57.600 baudios
- Velocidad 4 = 115.200 baudios

La controladora devuelve un byte de respuesta con el comando ASCII 13

COMANDOS DE LECTURA (Rx)

1- Lectura de posiciones de los servos (2 bytes)

Byte1 = P
Byte2 = N° servo

La controladora devuelve un byte con la posición y un byte con el comando ASCII 13

5-PUESTA EN MARCHA

Con la controladora se suministran dos ejemplos de programación en Basic (Visual Basic 6 y/o Visual 2005). Para empezar a trabajar y configurar las comunicaciones podemos abrir cualquiera de los dos ejemplos.

En la controladora solo se configuran las propiedades de velocidad del puerto serie, para ello tiene un menú en las **PROPIEDADES RS232**. Dentro de este formulario encontrara dos menús de opciones: **Numero de puerto serie** por el cual se comunicara con el hardware y **Velocidad de la conexión RS232**.

Para proceder con el arranque siga los pasos siguientes:

ARRANQUE DEL SOFTWARE Y CONFIGURACION

- 1- Conecte la alimentación de los servos en el borne regleta.
- 2- Conecte un servo en el canal 1
- 3- Conecte la controladora por USB al PC

Pasados unos instantes la controladora estará enviando los pulsos modulados al servo lo que provocara que el eje del servo se desplace al centro de su recorrido.

- 4- Arranque el ejemplo de programación
- 5- Seleccione la opción USB (en caso de que no este seleccionada) que encontrara debajo de la imagen del hardware
- 6- Compruebe que el servo se mueve cuando desplaza el fader nº 1

Lo errores de conexión o de funcionamiento han sido detallados de forma que si aparece alguno de ellos en pantalla deberá proceder como se indica.

Cuando tiene la controladora funcionando en modo USB es posible indicarle las propiedades del puerto serie accediendo al formulario **PROPIEDADES RS232**. Cuando cambie la velocidad RS232 se enviara a la controladora de forma que será establecida en ese nuevo valor.

El valor de la velocidad RS232 por defecto es de 9600 baudios y puede ser cambiada en los siguientes valores:

- 9600 baudios (defecto)
- 19200 baudios
- 38400 baudios
- 57600 baudios
- 115200 baudios

Es posible que el software de su pc este configurado a una velocidad RS232 diferente del que hay configurado en la controladora, en ese caso la controladora mostrara repetidos errores de conexión serie y no podrá conectar en el modo RS232.

Para solucionar esto conéctese directamente a la controladora por USB y proceda a establecer el valor como le hemos indicado anteriormente.

D.2 Tarjeta controladora Pololu Micro Serial Servo

Esta tarjeta controladora se empleó para manejar la primera generación de robot serpiente, su principal característica es que es capaz de controlar hasta 28 servomotores simultáneamente y presenta el aspecto mostrado en la Figura D.2.



Figura D.2 Aspecto de la tarjeta controladora Pololu Micro Serial Servo

Según la descripción de su fabricante: “*Con tan solo 0,91’’x0,91’’ la controladora Pololu Micro Serial Servo es la controladora serial de servomotores más pequeña que encontrará. Con control individual de posición y velocidad de hasta 8 servos y 12 bits de resolución, este pequeño dispositivo ofrece una serie de características avanzadas a un precio sorprendentemente asequible.*”

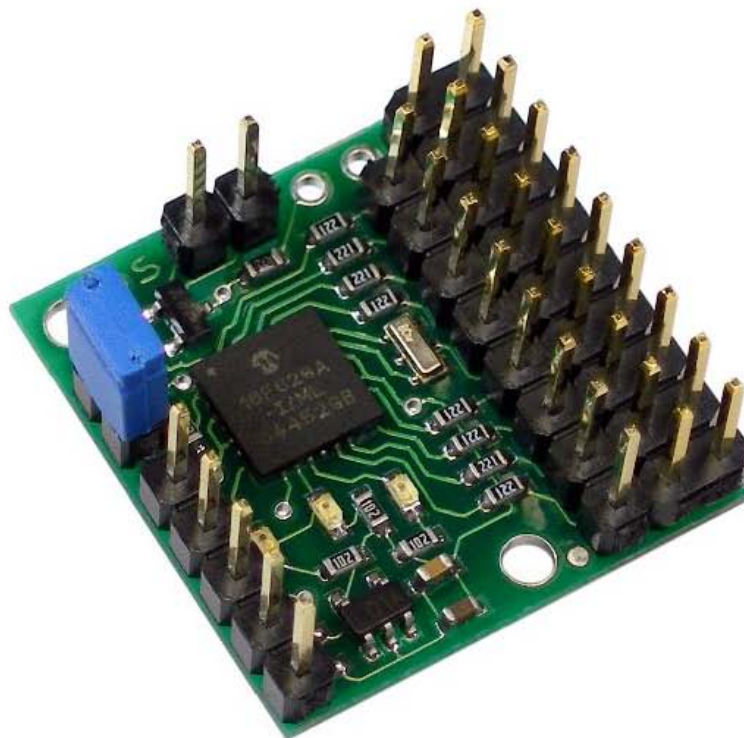
Características de la controladora de servos:

- Tamaño de la PCB: 0,91’’ x 0,91’’
- Puertos para Servos: 8
- Resolución: 0.5 μ s (unos 0.05°)
- Rango: 250 – 2750° μ s
- Tensión de la alimentación de la lógica: 5 – 16V
- Tensión de la línea de datos: 0 / 5 V
- Frecuencia de pulsos: 50 Hz
- Velocidades soportadas: 1200 – 38400 baudios (detectados automáticamente)
- Consumo de corriente 5 mA (valor medio)



Micro Serial Servo Controller

User's Guide



Specifications

PCB size.....	0.91" x 0.91"
Number of servo ports.....	8
Pulse width range.....	0.25-2.75 ms
Resolution.....	0.5 microsecond (~0.05 degree)
Supply voltage.....	5-16 V
I/O voltage.....	0 and 5 V
Baud rate.....	1200-38400 (auto detect)
Current consumption.....	5 mA (average)



© 2005
<http://www.pololu.com/>

SSC03A



Important Safety Warning

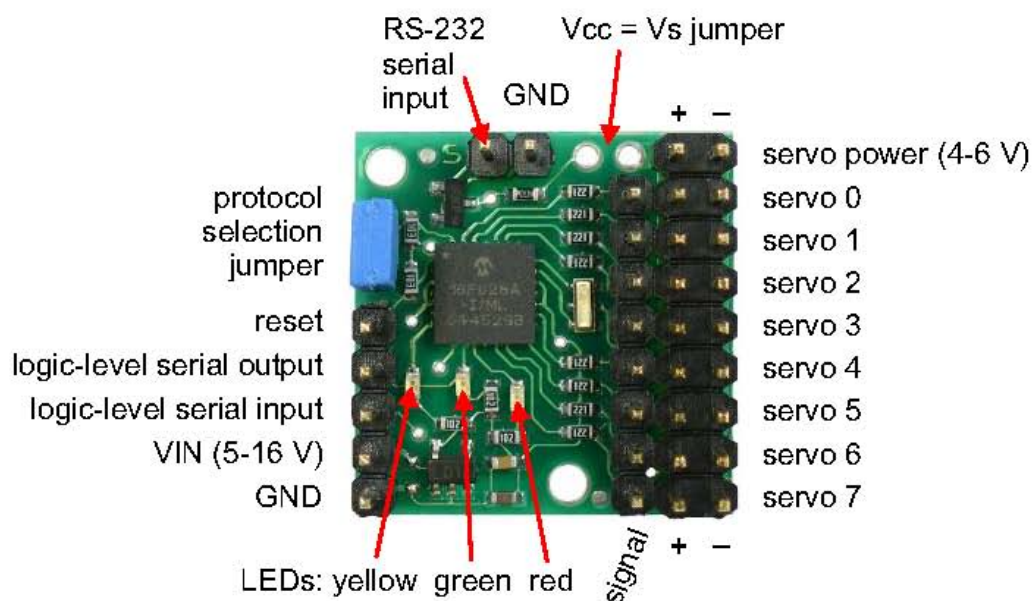
The servo controller module is not intended for young children! Younger users should use this module only under adult supervision. **By using this product, you agree not to hold Pololu liable for any injury or damage related to the use or to the performance of this product. This product is not designed for, and should not be used in, applications where the malfunction of the product could cause injury or damage.**

Contacting Pololu

You can check the Pololu web site at <http://www.pololu.com/> for the latest information about the servo controller, including color pictures, application examples, and troubleshooting tips.

We would be delighted to hear from you about your project and about your experience with our servo controller. You can contact us through our online feedback form or by email at support@pololu.com. Tell us what we did well, what we could improve, what you would like to see in the future, or anything else you would like to say!

Servo Controller Layout and Pinout



Connecting the Servo Controller

The servo controller connections are shown on the facing page, and most of the pins are labeled on the back of the servo controller. All square pads are ground.

Power

The servos are typically powered by their own supply consisting of a 4- or 5-cell rechargeable battery pack with a nominal voltage of 4.8-6.0 volts. This supply can be connected to the upper-right corner of the board, or to any of the servo 0 through servo 6 power pins (the connection between the servo 7 pins and the other pins is not designed to carry the current for all of the servos). The servo supply should be capable of providing several amps of current. If many servos will be straining simultaneously, the total current can be close to 10 amps. In such applications, you should use the servo controller PCB just for signals and separately wire up the power and ground pins of the servos individually.

The servo controller needs its own power, which can be 5-16 volts. This supply should be applied to the lower-left corner of the board, at the VIN and GND pins. If the servo battery pack is above 5 V, it can also power the servo controller. If the servo battery is under 5 V (a 4-cell pack), the servo controller can be powered by connecting across the Vcc = Vs jumper (and leaving VIN disconnected).

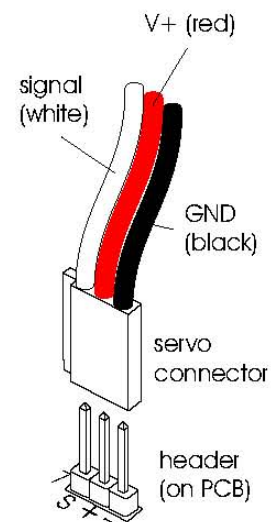
Please note that both the servo controller and most servos can be destroyed by applying power incorrectly (either too high of a voltage or reversed voltage).

Control signals

The servo controller requires a logic-level (0-5 V) serial input connected to the logic-level serial input or an RS-232-level serial input at the RS-232 pin. **Do not use both serial inputs at the same time!** The servo controller echoes all serial data out of the serial output pin, and the reset line can be brought low to reset the servo controller. In most applications, the reset input and serial output can be left disconnected.

Servos

When connecting servos, be careful because it is possible to plug in a servo backwards. **Make sure to connect your servos correctly, or they may be destroyed.** The signal (usually white or yellow) wire should be toward the inside of the board, and the black wire should be closest to the edge of the board.



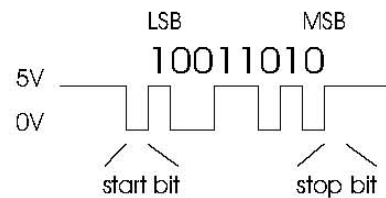
Using the Servo Controller

The Pololu servo controller performs the processor-intensive task of simultaneously generating 8 independent servo control signals. The servo controller can generate pulses from 0.25 ms to 2.75 ms, which is greater than the range of most servos, and which allows for a servo operating range of over 180 degrees.

Internally, the servo controller maintains a servo position value that is two times the pulse width, measured in microseconds. Thus, the 1.5 ms neutral position, which is 1500 microseconds long, is represented internally as 3000. The internal values range from 500 to 5500. Various interface modes allow the user to set the position value for each servo in multiple ways, which are described below.

Serial Input

The serial commands sent to the servo controller must be sent eight bits at a time, with no parity and one stop bit (sometimes abbreviated 8N1). Logic-level serial input must be *non-inverted*, meaning that a zero is sent as a low voltage, and a one is sent as a high voltage, as shown in the diagram to the right. Any inverted serial input, whether at logic levels or at RS-232 levels, may be connected to the RS-232 serial input.



When you turn on power with your serial input connected, you should either see all of the LEDs or just the yellow LED turn on. After the servo controller turns on and determines the communication mode (see below), it waits for a serial input to determine the baud rate. If the input line is low, it turns all LEDs on and waits for the line to go high, which is the idle state for the serial line. Once the line is detected to be high, only the yellow LED is turned on, and the servo controller waits for a serial input. If the detected baud rate is too high, the red LED will turn on and the green LED will flash quickly. If the serial rate is too slow, the red LED will turn on and the yellow LED will flash. From this point on, the servo controller behavior depends on the communication mode. **Once you choose a baud rate, all subsequent transmissions must be at that same baud rate.**

Indicator LEDs

The green LED indicates serial activity: it should flicker whenever the servo controller receives data. The yellow LED indicates a warning regarding position: either the absolute or neutral position you have requested is out of range, or a combination of neutral, range, and 7-bit or 8-bit position caused the internal position variable to go out of range. The position will just be limited to the max or min, and the yellow LED will go out when all requested positions are in range. The red LED indicates a fatal error that prevents further operation (for example, a fatal error could be caused by invalid serial input).

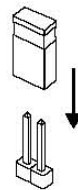


Interface Options

You can communicate with the servo controller using one of two communication protocols. One of the two interface modes is chosen based on the state of mode selection jumper when the servo controller is powered up; you cannot change modes without resetting the servo controller.

Pololu Mode: The default mode, when the jumper is open (no shorting block), is a Pololu protocol used for controlling multiple serial devices. In this mode, the servo controller can be on the same serial line as other devices such as our Dual Serial Motor Controller. This mode also allows access to all of the special features of the servo controller, such as setting speeds, ranges, and neutral settings.

Mini SSC II Mode: This mode is set by placing the shorting block over the two jumper pins. This setting allows the servo controller to respond to the protocol used by the Mini SSC II servo controller made by Scott Edwards Electronics. This protocol is more simple, but it only allows the user to specify the desired servo positions in only one way. In this mode, the servo controller is not compatible with other Pololu serial peripheral products.



Mini SSC II Mode

Baud Rate. The available baud rate range in this mode is approximately 500-10k baud, but the Mini SSC II only works at 2400 or 9600 baud. If you want to put a Mini SSC II servo controller on the same serial line as your Pololu 8-servo controller, you must use one of the two baud rates that the Mini SSC II can support.

Protocol. To set the servo position, send a sequence of three bytes. The first byte is a synchronization value that must always be 255. Byte 2 is the servo number, and it can be 0-254. Byte 3 is the position to which you want the servo to move, also 0-254.

start byte = 0xFF	servo number, 0x00-0xFE	Servo position, 0x00-0xFE
-------------------	-------------------------	---------------------------

Two motion ranges are available in this mode. Each Pololu servo controller responds to 16 servo numbers. Addressing the lower 8 will move them within an approximately 90 degree range, while addressing the upper 8 servo numbers will give twice the range. You can set the group of servo numbers to which your servo controller responds (see “Setting and Checking the Servo Numbers”). For example, if your servo controller is set to servo numbers 0, it responds to servo numbers 0-15, and sending the command sequence [255, 10, 254] will move servo 2 all the way to one extreme of its range in 180 degree mode. If you send servo numbers that are not recognized, the servo controller will ignore the command. Up to sixteen servo controllers can be connected on one serial line to control up to 128 servos independently.



Pololu Mode

In this mode, there are several options for controlling your servos. As mentioned at the beginning of this section on page 4, the servo controller holds an internal variable for each servo, the value of which ranges from 500 to 5500, where the number corresponds to the pulse length in increments of half of a microsecond. The various commands deal with setting these internal values. With *absolute* commands, you simply set the value for each servo. In 7- and 8-bit modes, you set *neutral*, *range*, and *direction* parameters for each servo; then, when you send a 7- or 8-bit position command, the servo controller combines all of the parameters to obtain the actual servo position. Whether you use absolute commands or not, you can individually control the speed of each servo and whether the servo is on or off (most servos shut off when they receive no pulses). This section describes the interface details.

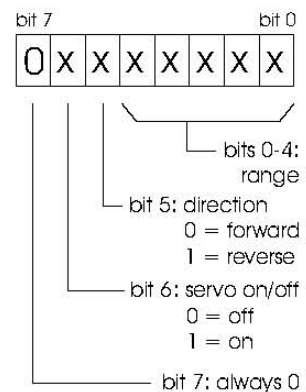
Baud Rate. The available baud rate range in this mode is approximately 2,000-40,000 baud. It is possible for the servo controller to operate at rates as high as 57600 baud, but the success of the attempt will depend on the exact speed of your serial control source. The servo controller is not committed to standard baud rates, so if you have a device that can transmit at a maximum rate of 45,000 baud, try it, and it might work.

Protocol. To communicate with the servo controller, send sequences of five or six bytes. The first byte is a synchronization value that must *always* be 0x80 (128). Byte 2 is the Pololu device type number, which is 0x01 for the 8-servo controller. Byte 3 is one of six values for different commands to the controller; the commands are discussed below. Byte 4 is the servo to which the command should apply. Bytes 5 and possibly 6 are the data values for the given command. **In every byte except the start byte, bit seven must be clear. Thus, the range of values for bytes 2-6 is 0-0x7F (0-127).**

start byte = 0x80	device ID = 0x01	command	servo num	data 1	data 2
-------------------	------------------	---------	-----------	--------	--------

Command 0: Set Parameters (1 data byte)

- Bit 6 specifies whether a servo is on or not; a 1 turns the servo on, and a 0 (default) turns it off.
- Bit 5 sets the direction the servo moves, which only applies to 7- and 8-bit position commands. If the bit is 0 (default), a larger position number causes the output pulse to get bigger; if the bit is 1, a larger position number will make the output pulse shorter.
- Bits 0-4 set the range through which the servo moves in 7- and 8-bit commands. A larger value will give a larger range, and setting the range to 0 will make the servo always stay at neutral. Given the same range setting, an 8-bit position command will move the servo through twice the range of a 7-bit position command. The default range setting is 15, which will give a range of approximately 180 degrees in 8-bit commands and 90 degrees in 7-bit commands.



Pololu Mode (continued)

Command 1: Set Speed (1 data byte)

This command allows you to set the speed at which the servo moves. If the speed is set to 0 (default), the output pulse will instantly change to the set position. If the speed value is nonzero, the pulse changes gradually from the old position to the new position. With a speed of 1, the pulse width changes at 50 microseconds per second; the maximum speed of 6.35 ms per second is achieved with a speed setting of 127.

Command 2: Set Position, 7-bit (1 data byte)

When this command is sent, the data value is multiplied by the range setting for the corresponding servo and adjusted for the neutral setting. This command can be useful in speeding up communications since only 5 total bytes are sent to set a position. Setting a servo position will automatically turn the servo on.

Command 3: Set Position, 8-bit (2 data bytes)

This command is just like the 7-bit version, except that two data bytes must be sent to transfer 8 bits. Bit 0 of data 1 contains the most significant bit (bit 7), and the lower bits of data 2 contain the lower seven bits. (Bit 7 in data bytes must always be 0.) Setting a servo position will automatically turn the servo on.

Command 4: Set Position, Absolute (2 data bytes)

This command allows direct control of the internal servo position variable. Neutral, range, and direction settings do not affect this command. Data 2 contains the lower 7 bits, and Data 1 contains the upper bits. The range of valid values is 500 through 5500. Setting a servo position will automatically turn the servo on.

Command 5: Set Neutral (2 data bytes)

Setting neutral only applies to 7- and 8-bit commands. The neutral value sets the middle of a range, and corresponds to a 7-bit position value of 63.5 or an 8-bit position value of 127.5. The neutral position is an absolute position just like in command 4, and setting the neutral position will move the servo to that position. The default value is 3000. It may be useful to change neutral if you change servos and need to calibrate your system, or if you cannot get your mechanical linkages to just the right lengths.

Tip: Setting neutral and servo direction can be useful if you have a device, such as a walking robot, that has multiple symmetrical structures on two sides of a chassis. Instead of determining a sequence of positions for each leg individually, you can design a single leg, and then use the same position values for other legs, changing only the neutral position and direction as necessary.



Setting and Checking the Servo Numbers

The micro-servo controller has the unique feature of allowing the user to set the servo numbers to which the controller responds. By default, the servo controller responds to servo numbers 0-7 (in Pololu mode), but you can set it to respond to numbers 8-15, 16-23, all the way to 120-127. (In Mini SSC II mode, the servo controller would respond to numbers 0-15, 16-31, all the way through 240-254.) This feature is useful if you want to use more than one servo controller at a time to control up to 128 independent servos.

To set the servo numbers, put the servo controller in Pololu mode (shorting block removed from J1) and send the serial sequence [128, 2, *<servonums>*], where *<servonums>* is a number from 0 through 15. A setting of 0 will make the servo controller respond to servo numbers 0-7 (in Pololu mode), a setting of 1 will make it respond to servo numbers 8-15, and so on.

start byte = 0x80	change servo numbers = 0x02	new setting, 0x00-0x10
-------------------	-----------------------------	------------------------

Upon receiving the command, the servo controller will turn on the red and yellow LEDs and quickly flash the green LED *<servonums>* + 1 times. The green LED will thus quickly flash 1-16 times. The green LED will then pause for approximately 1 second before flashing again. The 8-servo controller must be reset (power turned off and back on) before it can be used.

If you want to just see the servo numbers setting without changing it, use the above command, but use the value 16 for *<servonums>*. The servo number settings will remain unchanged, but the green LED will flash to indicate the servo numbers, as described above.

The configuration of the servo controller should only be done once per servo controller, with only one servo controller on the serial line (otherwise, all units would get set to the same number, and the whole point is to give them unique numbers!). The number is stored in a location that can be written thousands of times, but the number should not be written every time your system is powered up.



D.3 Módulos de comunicación inalámbrica Xbee

Esta tarjeta controladora se empleó para manejar la primera generación de robot serpiente, su principal característica es que es capaz de controlar hasta 28 servomotores simultáneamente y presenta el aspecto mostrado en la Figura D.3.

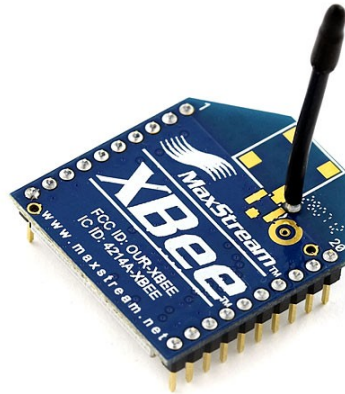


Figura D.3. Aspecto de los módulos Xbee

Según la descripción de su fabricante: *“Los módulos XBEE y XBEE-PRO 802.15.4 OEM RF son soluciones integradas para proveer de conectividad inalámbrica punto-punto a una gran variedad de dispositivos. Estos módulos usan el protocolo de red IEEE 802.15.4 para un rápido trabajo en red puerto-puerto o punto-multipunto. Están diseñados para aplicaciones de alto rendimiento que requieren baja latencia y tiempo de comunicación predecibles.”*

Características de los módulos Xbee:

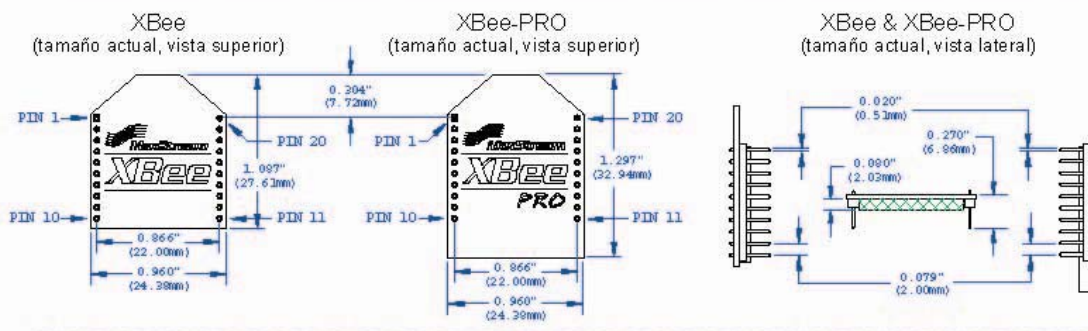
- Sin necesidad de configuración para comunicaciones RF “out-of-the-box”
- Huella Xbee común para gran variedad de módulos de RF
- Rápida tasa de datos de Rf de 250 Kbps al nodo final
- 2.4 GHz para desarrollo en todo el mundo
- Modo “sleep” soportado para alargar la vida de las baterías
- iDigi ® compatible

Puesto que la hoja de características técnicas de estos dispositivos es un documento de una amplia extensión y la información referente a las primeras versiones de desarrollo se encuentra fragmentada debido a las sucesivas versiones que diferentes fabricantes desarrollaron en un breve espacio de tiempo, solo se mostrarán las páginas más relevantes de la versión “Series 2” que es la utilizada en el robot

Modulos de RF version OEM XBee & XBee-PRO 2.4 GHz

Especificaciones		XBee 	XBee-PRO 
Rendimiento	Alcance en ambientes interiores/espacios urbanos	hasta 100' (30 m)	hasta 300' (100 m)
	Alcance de RF en Línea de Visión para ambientes exteriores	hasta 300' (100 m)	hasta 4000' (1200 m)
	Potencia de Salida en Transmisión	1 mW (0 dBm)	60 mW (18 dBm), 100 mW EIRP
	Régimen RF de datos	250,000 bps	250,000 bps
	Sensibilidad del Receptor	-92 dBm (1% PER)	-100 dBm (1% PER)
Requerimientos de Potencia	Suministro de Voltaje	2.8 – 3.4 V	2.8 – 3.4 V
	Corriente de Transmisión (típico)	45 mA @ 3.3 V	270 mA @ 3.3 V
	Corriente de Recepción (típico)	50 mA @ 3.3 V	55 mA @ 3.3 V
	Corriente Power-Down	< 10 µA	< 10 µA
Información General	Frecuencia	ISM 2.4 GHz	ISM 2.4 GHz
	Dimensiones	0.960" x 1.087" (2.438cm x 2.761cm)	0.960" x 1.297" (2.438cm x 3.294cm)
	Temperatura de Operación	-40 to 85° C (Industrial)	-40 to 85° C (Industrial)
	Opciones de Antena	Conector U.FL, Antena Chip, o alambre de antena	Conector U.FL, Antena Chip, o alambre de antena
Trabajo en Red y Seguridad	Topologías permitidas en la Red	Punto a Punto, Punto a Multipunto, Igual a Igual y Mesh	Punto a Punto, Punto a Multipunto, Igual a Igual y Mesh
	Número de Canales	16 Canales de Secuencia Directa (software seleccionable)	12 Canales de Secuencia Directa (software seleccionable)
	Capas de Filtración de la Red	PAN ID & Direcciones 64-bit	PAN ID & Direcciones 64-bit
Aprobaciones de la Agencia	FCC Parte 15.247	OUR-XBEE	OUR-XBEEPRO
	Industry Canada (IC)	4214A-XBEE	4214A-XBEEPRO
	Europe	ETSI	ETSI

Las especificaciones están sujetas a cambios sin previo aviso.

Dibujos Mecánicos


MaxStream.
 355 South 520 West, 180
 London, UT 84042
 © 2005 MaxStream, Inc.

Para lograr lo mejor en Soluciones Inalámbricas de Datos y Soporte Técnico, sírvase contactar con MaxStream, Inc Se Habla Español

teléfono: (866) 765-9885 (Para llamados gratis dentro de U.S.A. & Canadá)
 (801) 765-9885 (Para llamadas desde cualquier parte del mundo)
 fax: (801) 765-9895
 web: www.maxstream.net (Tenemos un canal para CHAT en vivo así como muchos otros recursos para brindarle el mejor servicio)

1.5. Pin Signals

Figure 1-03. XBee Series 2 RF Module Pin Number
(top sides shown - shields on bottom)

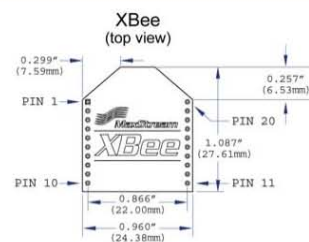


Table 1-02. Pin Assignments for the XBee Series 2 Modules
(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / <u>CONFIG</u>	Input	UART Data In
4	DIO12	Either	Digital I/O 12
5	<u>RESET</u>	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI / DIO10	Either	PWM Output 0 / RX Signal Strength Indicator / Digital I/O
7	PWM / DIO11	Either	Digital I/O 11
8	[reserved]	-	Do not connect
9	DTR / <u>SLEEP_RQ</u> / DIO8	Either	Pin Sleep Control Line or Digital I/O 8
10	GND	-	Ground
11	DIO4	Either	Digital I/O 4
12	<u>CTS</u> / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	<u>ON</u> / <u>SLEEP</u> / <u>DIO9</u>	Output	Module Status Indicator or Digital I/O 9
14	[reserved]	-	Do not connect
15	Associate / DIO5	Either	Associated Indicator, Digital I/O 5
16	<u>RTS</u> / DIO6	Either	Request-to-Send Flow Control, Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0 / ID Button	Either	Analog Input 0, Digital I/O 0, or Node Identification

Design Notes:

- Minimum connections: VCC, GND, DOUT & DIN
- Minimum connections to support serial firmware upgrades: VCC, GND, DIN, DOUT, RTS & DTR
- Signal Direction is specified with respect to the module
- Module includes a 30k Ohm resistor attached to RESET
- Several of the input pull-ups can be configured using the PR command
- Unused pins should be left disconnected

1.6. Electrical Characteristics

Table 1-03. DC Characteristics of the XBee Series 2 (VCC = 2.8 - 3.4 VDC)

Symbol	Parameter	Condition	Min	Typical	Max	Units
V _{IL}	Input Low Voltage	All Digital Inputs	-	-	0.2 * VCC	V
V _{IH}	Input High Voltage	All Digital Inputs	0.8 * VCC	-	0.18 * VCC	V
V _{OL}	Output Low Voltage	I _{OL} = 2 mA, VCC >= 2.7 V	-	-	0.18 * VCC	V
V _{OH}	Output High Voltage	I _{OH} = -2 mA, VCC >= 2.7 V	0.82 * VCC	-	-	V
I _{IN}	Input Leakage Current	V _{IN} = VCC or GND, all inputs, per pin	-	-	0.5 uA	uA

2.1.4. Transparent Operation

RF modules that contain the following firmware versions will support Transparent Mode: 1.0xx (coordinator) and 1.2xx (router/end device).

When operating in Transparent Mode, the modules act as a serial line replacement. All UART data received through the DIN pin is queued up for RF transmission. When RF data is received, the data is sent out the DOUT pin. The module configuration parameters are configured using the AT command mode interface. (See RF Module Operation --> Command Mode.)

When RF data is received by a module, the data is sent out the DOUT pin.

Serial-to-RF Packetization

Data is buffered in the serial receive buffer until one of the following causes the data to be packetized and transmitted:

1. No serial characters are received for the amount of time determined by the RO (Packetization Timeout) parameter. If RO = 0, packetization begins when a character is received.
2. Maximum number of characters that will fit (72) in an RF packet is received.
3. The Command Mode Sequence (GT + CC + GT) is received. Any character buffered in the serial receive buffer before the sequence is transmitted.

2.1.5. API Operation

API (Application Programming Interface) Operation is an alternative to the default Transparent Operation. The frame-based API extends the level to which a host application can interact with the networking capabilities of the module. RF modules that contain the following firmware versions will support API operation: 1.1xx (coordinator) and 1.3xx (router/end device).

When in API mode, all data entering and leaving the module is contained in frames that define operations or events within the module.

Transmit Data Frames (received through the DIN pin (pin 3)) include:

- RF Transmit Data Frame
- Command Frame (equivalent to AT commands)

Receive Data Frames (sent out the DOUT pin (pin 2)) include:

- RF-received data frame
- Command response
- Event notifications such as reset, associate, disassociate, etc.

The API provides alternative means of configuring modules and routing data at the host application layer. A host application can send data frames to the module that contain address and payload information instead of using command mode to modify addresses. The module will send data frames to the application containing status packets; as well as source, and payload information from received data packets.

The API operation option facilitates many operations such as the examples cited below:

- > Transmitting data to multiple destinations without entering Command Mode
- > Receive success/failure status of each transmitted RF packet
- > Identify the source address of each received packet

To implement API operations, refer to the API Operation chapter 6.

2.2. Modes of Operation

2.2.1. Idle Mode

When not receiving or transmitting data, the RF module is in Idle Mode. During Idle Mode, the RF module is also checking for valid RF data. The module shifts into the other modes of operation under the following conditions:

- Transmit Mode (Serial data in the serial receive buffer is ready to be packetized)
- Receive Mode (Valid RF data is received through the antenna)
- Sleep Mode (End Devices only)
- Command Mode (Command Mode Sequence is issued)

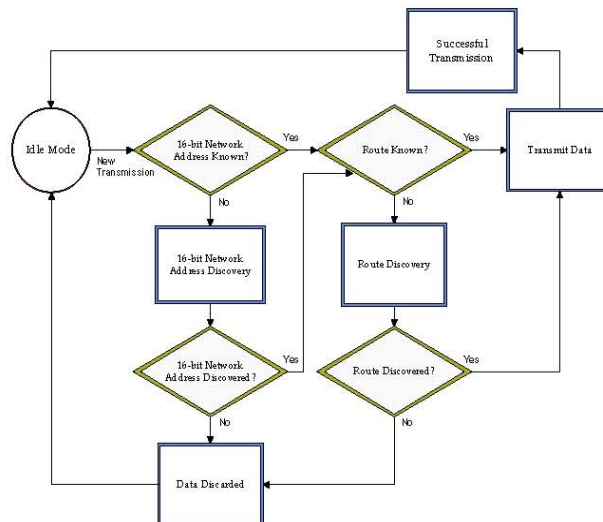
2.2.2. Transmit Mode

When serial data is received and is ready for packetization, the RF module will exit Idle Mode and attempt to transmit the data. The destination address determines which node(s) will receive the data.

Prior to transmitting the data, the module ensures that a 16-bit network address and route to the destination node have been established.

If the 16-bit network address is not known, network address discovery will take place. If a route is not known, route discovery will take place for the purpose of establishing a route to the destination node. If a module with a matching network address is not discovered, the packet is discarded. The data will be transmitted once a route is established. If route discovery fails to establish a route, the packet will be discarded.

Figure 2-04. Transmit Mode Sequence



When data is transmitted from one node to another, a network-level acknowledgement is transmitted back across the established route to the source node. This acknowledgement packet indicates to the source node that the data packet was received by the destination node. If a network acknowledgement is not received, the source node will re-transmit the data. See Data Transmission and Routing in chapter 3 for more information.

2.2.3. Receive Mode

If a valid RF packet is received, the data is transferred to the serial transmit buffer

2.2.4. Command Mode

To modify or read RF Module parameters, the module must first enter into Command Mode - a state in which incoming serial characters are interpreted as commands. Refer to the API Mode section for an alternate means of configuring modules.

AT Command Mode

To Enter AT Command Mode:

Send the 3-character command sequence "+++" and observe guard times before and after the command characters. [Refer to the "Default AT Command Mode Sequence" below.]

Default AT Command Mode Sequence (for transition to Command Mode):

- No characters sent for one second [GT (Guard Times) parameter = 0x3E8]
- Input three plus characters ("+++") within one second [CC (Command Sequence Character) parameter = 0x2B.]
- No characters sent for one second [GT (Guard Times) parameter = 0x3E8]

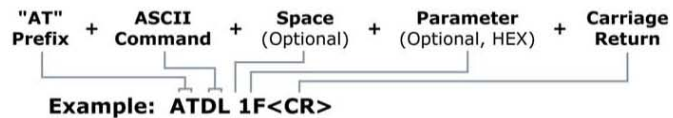
All of the parameter values in the sequence can be modified to reflect user preferences.

NOTE: Failure to enter AT Command Mode is most commonly due to baud rate mismatch. Ensure the 'Baud' setting on the "PC Settings" tab matches the interface data rate of the RF module. By default, the BD parameter = 3 (9600 bps).

To Send AT Commands:

Send AT commands and parameters using the syntax shown below.

Figure 2-05. Syntax for sending AT Commands



To read a parameter value stored in the RF module's register, omit the parameter field.

The preceding example would change the RF module Destination Address (Low) to "0x1F". To store the new value to non-volatile (long term) memory, subsequently send the WR (Write) command.

For modified parameter values to persist in the module's registry after a reset, changes must be saved to non-volatile memory using the WR (Write) Command. Otherwise, parameters are restored to previously saved values after the module is reset.

System Response. When a command is sent to the module, the module will parse and execute the command. Upon successful execution of a command, the module returns an "OK" message. If execution of a command results in an error, the module returns an "ERROR" message.

To Exit AT Command Mode:

1. Send the ATCN (Exit Command Mode) command (followed by a carriage return).
[OR]
2. If no valid AT Commands are received within the time specified by CT (Command Mode Timeout) Command, the RF module automatically returns to Idle Mode.

For an example of programming the RF module using AT Commands and descriptions of each configurable parameter, refer to the "Examples" and "XBee Series 2 Command Reference Tables" chapters.

2.2.5. Sleep Mode

Sleep modes allow the RF module to enter states of low-power consumption when not in use. To enter Sleep Mode, one of the following conditions must be met (in addition to the module having a non-zero SM parameter value):

- Sleep_RQ (pin 9) is asserted
- The module is idle (no data is transmitted or received) for the time defined by the ST (Time before Sleep) parameter.

Table 2-01. Sleep Mode Configurations (Router / End Device Firmware Only)

Sleep Mode Setting	Transition into Sleep Mode	Transition out of Sleep Mode (wake)	Characteristics	Related Commands	Power Consumption
SM=1	Assert (high) Sleep_RQ (pin 9)	De-assert (0V) Sleep_RQ (pin 9)	Pin/Host controlled	SM	< 1uA
SM=4	Automatic transition to sleep mode as defined by the ST parameter	Transition occurs after the cyclic sleep time interval elapses. The time interval is defined by the SP (Cyclic Sleep Period) parameter.	RF module wakes after a pre-determined time interval to detect if RF data is present.	SM, ST, SP, SN	< 1uA

The SM command is central to setting Sleep Mode configurations. By default, sleep modes are disabled (SM=0) and the module remains in Idle/Receive Mode. When in this state, the module is constantly ready to respond to serial or RF activity.

ZigBee Protocol: Sleep Modes

In the ZigBee protocol, sleep modes are only supported on end devices. See section 4.3, "Sleep Mode Operation" for more information.

Pin/Host Controlled Sleep

Pin sleep puts the module to sleep and wakes it from sleep according to the state of Sleep_RQ (pin 9). When Sleep_RQ is asserted (high), the module will finish any transmit or receive operations, and then enter a low power state. The module will not respond to either serial or RF activity while in sleep.

To wake a module operating in pin sleep, de-assert Sleep_RQ (pin 9). The module will wake when Sleep_RQ is de-asserted and is ready to transmit or receive when the CTS line is low. When the module wakes from pin sleep, it sends a transmission to its parent router or coordinator (called a poll request) to see if it has buffered any data packets for the end device. The module will continue to poll its parent for data while it remains awake. If the parent receives an RF data packet destined for one or more of its end device children, it will transmit the packet to the end device upon receipt of a poll request. See section 4.3, "Sleep Mode Operation" for more information.

Cyclic Sleep

Cyclic sleep allows modules to wake periodically to check for RF data and sleep when idle. When the SM parameter is set to 4, the module is configured to sleep for the time specified by the SP parameter. After the SP time expires, the module will wake and check for RF or serial data. To check for RF data, the module sends a transmission to its parent router or coordinator (called a poll request) to see if its parent has any buffered data packets for the end device. If the parent has data for the module, the module will remain awake to receive the data. Otherwise, the module will return to sleep. (See section 4.3, "Sleep Mode Operation" for more information.)

If serial or RF data is received, the module will start the ST timer and remain awake until the timer expires. While the module is awake, it will continue to send poll request messages to its parent to check for additional data. The ST timer will be restarted anytime serial or RF activity occurs. The module will resume sleep when the ST timer expires.

When the module wakes from sleep, it asserts On/Sleep (pin 13) to provide a wake indicator to a host device. If a host device wishes to sleep longer than SP time or to wake only when RF data arrives, the SN command can be used to prevent On/Sleep from asserting for a multiple of SP time. For example, if SP = 20 seconds, and SN = 5, the On/Sleep pin will remain de-asserted (low) for up to 100 seconds.

If CTS flow control is enabled, CTS (pin 12) is asserted (0V) when the module wakes and de-asserted (high) when the module sleeps, allowing for communication initiated by the host if desired.

3. ZigBee Networks

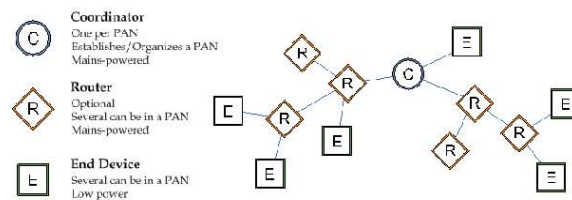
3.1. ZigBee Network Formation

A ZigBee Personal Area Network (PAN) consists of one coordinator and one or more routers and/or end devices. A ZigBee Personal Area Network (PAN) is created when a coordinator selects a channel and PAN ID to start on. Once the coordinator has started a PAN, it can allow router and end device nodes to join the PAN.

When a router or end device joins a PAN, it receives a 16-bit network address and can transmit data to or receive data from other devices in the PAN. Routers and the coordinator can allow other devices to join the PAN, and can assist in sending data through the network to ensure data is routed correctly to the intended recipient device. When a router or coordinator allows an end device to join the PAN, the end device that joined becomes a child of the router or coordinator that allowed the join.

End devices, however can transmit or receive data but cannot route data from one node to another, nor can they allow devices to join the PAN. End devices must always communicate directly to the parent they joined to. The parent router or coordinator can route data on behalf of an end device child to ensure it reaches the correct destination. End devices are intended to be battery powered and can support low power modes.

Figure 3-01. Node Types / Sample of a Basic ZigBee Network Topology



The network address of the PAN coordinator is always 0. When a router joins a PAN, it can also allow other routers and end devices to join to it. Joining establishes a parent/child relationship between two nodes. The node that allowed the join is the parent, and the node that joined is the child. The parent/child relationship is not necessary for routing data.

3.1.1. Starting a ZigBee Coordinator

When a coordinator first comes up, it performs an energy scan on multiple channels (frequencies) to select an unused channel to start the PAN. After removing channels with high detected energy levels, the coordinator issues an 802.15.4 beacon request command on the remaining, low energy level channels. Nearby routers or coordinators that have already joined a PAN respond to the beacon request frame with a small beacon transmission indicating the PAN identifier (PAN ID) that they are operating on, and whether or not they are allowing joining. The coordinator will attempt to start on an unused PAN ID and channel. After starting, the coordinator may allow other devices to join its PAN.

3.1.2. Joining a Router

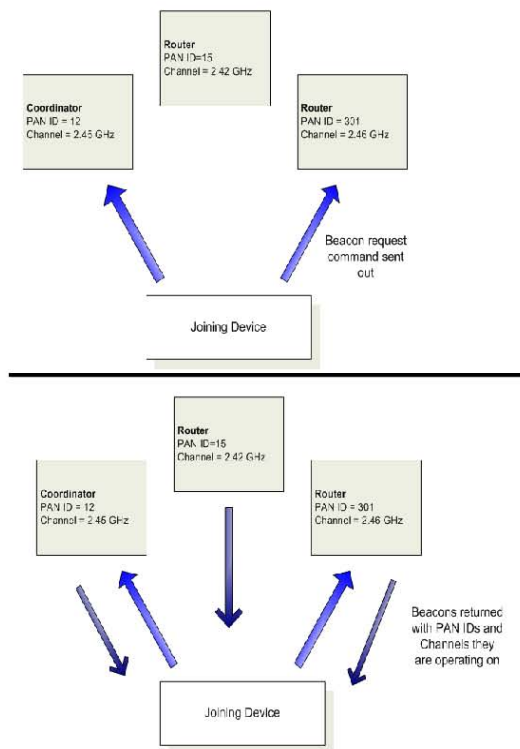
When a router first comes up, it must locate and join a ZigBee PAN. To do this, it issues an 802.15.4 beacon request command on multiple channels to locate nearby PANs. Nearby routers and coordinators that have already joined a PAN respond to the beacon request frame with a small beacon transmission, indicating which channel and PAN ID they are operating on. The router listens on each channel for these beacon frames. If a valid PAN is found from one of the received beacons, the router issues a join request to the device that sent the beacon. If joining succeeds, the router will then receive a join confirmation from the device, indicating the join was successful.

Once the router joins the PAN, it can communicate with other devices on the PAN and allow new devices to join to it.

3.1.3. Joining an End Device

When an end device first comes up, it must also locate and join a PAN. End devices follow the same process as a router to join a PAN. Once the end device has successfully joined a PAN, it can communicate with other devices on the PAN. However, since end devices cannot route data, they must always communicate directly with their parent and allow the parent to route data in its behalf.

Figure 3-02. Demonstration of Beacon Request and Beacon transmissions that take place during joining.



D.4 Adaptador Xbee Explorer USB

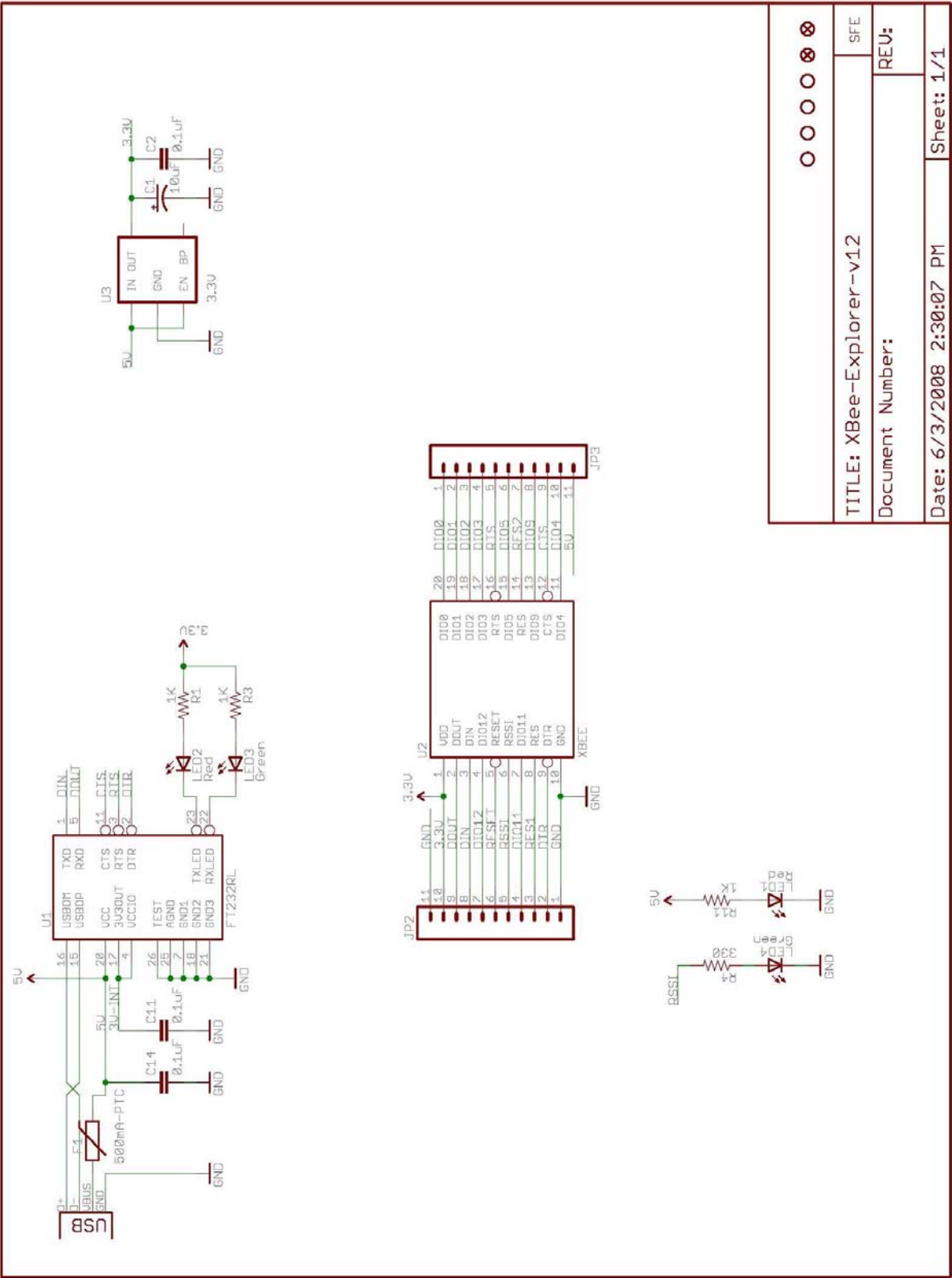
Este dispositivo resulta necesario para poder conectar el módulo Xbee que realiza la función de coordinador al PC, así como para poder configurar el firmware del resto de módulos Xbee para que permanezcan configurados de forma permanente en modo “end device”. Su aspecto es el que se muestra en la Figura D.4.



Figura D.4. Aspecto del dispositivo Xbee Explorer USB

Según la descripción de su fabricante: *“El XBee explorer USB permite conectar y utilizar cualquier módulo XBee directamente mediante un puerto USB. Es tan sencillo como conectar un módulo XBee, pinchar un cable mini USB al PC y tendremos acceso a los pines TX/RX del XBee y estará listo para funcionar.”*

La principal característica de este componente es que básicamente se comporta como una interfaz de conversión entre el puerto USB y el módulo Xbee, su utilización es extremadamente sencilla ya que opera en modo transparente una vez instalados los drivers que podemos encontrar en la página Web del fabricante: <http://www.ftdichip.com/FTDrivers.htm>



D.5 Minicámara espía inalámbrica S130347

La cámara empleada utilizada para sensorizar al robot no es un artículo especializado de robótica lo que impide su integración completa en el diseño del robot, no obstante debido a su pequeño tamaño y a sus buenas prestaciones el resultado obtenido es bueno, su aspecto, así como una idea de sus dimensiones puede verse en la Figura D.5.



Figura D.5. Aspecto de la cámara

La siguiente descripción se encuentra sacada de forma textual de la página Web donde se puede realizar su compra (<http://www.superinventos.com/S130347.htm>):

La microcámara espía inalámbrica es un kit que incluye una diminuta cámara de vídeo de alta resolución, (mas de 380 líneas de TV), un microtransmisor de vídeo y audio de 50 mW, incorporado en la misma carcasa de la cámara, receptor, los adaptadores de corriente para poder conectar a la red tanto el receptor como la cámara y un adaptador de pila de 9 voltios para la microcámara. Es decir incluye todo lo necesario para utilizar la minicámara en sistemas de vigilancia, espionaje, documentación, control de acceso y seguridad, ...

La instalación de la microcámara en cualquier ambiente es bien sencilla, tan solo hay que colocarla donde deseemos y conectarle la alimentación que más nos convenga:

- Adaptador de corriente de 8 v cuando se sitúe en un lugar fijo.
- Cuando se utiliza de forma móvil, llevándola encima por ejemplo, o colocándola en lugares donde no dispongamos de enchufe, utilizaremos pila 9 voltios alcalina (No incluida), cuya duración es de aproximadamente 4 a 5 horas.

Para recibir la señal de la microcámara, solo hay que conectar el receptor a un televisor o a un videograbador doméstico mediante el cable de vídeo y audio que trae al efecto, y conectar el alimentador DC. El receptor ofrecerá a la salida la señal de vídeo compuesto y

audio para poder aplicarse a cualquier TV, monitor de vídeo, VCR, ordenador con digitalizadora, etc.



Figura D.6. Aspecto del receptor

Las inmejorables prestaciones del equipo hacen que su rango de aplicaciones sea muy amplio. Resulta ideal como cámara espía de instalación inmediata, ya que su diminuto tamaño le permitirá ocultarla en cualquier lugar por pequeño que este sea. No menos importante que la transmisión de imágenes es el sonido.

La microcámara S130347 dispone de un micrófono incorporado que suministra una calidad de sonido, fundamental para labores de espionaje y vigilancia. Además, al ser inalámbrica y funcionar con pilas, no hay cables a la vista lo que facilita todavía mas su colocación. Puede esconderla en la corbata, en un paquete de tabaco, en un monedero o en cualquier otro elemento de uso cotidiano sin que nadie sospeche jamás que esta siendo espiado por una cámara en color.

No hay una cámara en color inalámbrica más pequeña. A pesar de su pequeño tamaño, el transmisor tiene la misma potencia que un vídeo sender normal, llegando a los 50 - 200 metros en el exterior. En el interior el rango viene determinado principalmente por el número de obstáculos que tiene que atravesar. En las pruebas realizadas por nuestros técnicos, se lograron distancias de más de 60 metros, atravesando paredes y techos.

Dadas las características tan particulares de esta microcámara y sobre todo debido a su reducido tamaño, la posibilidad de camuflarla u ocultarla en un objeto se convierte en una labor sencilla, tan solo hay que dejar volar un poco la imaginación y sobre todo utilizar objetos cotidianos que permitan no llamar la atención. De esta manera podremos ocultarla en objetos tan variados como un sombrero, una corbata, un maletín, un archivador de oficina, un peluche, etc...

CARACTERÍSTICAS:

Elemento de imagen	Sensor CMOS Color de 1/3
Sistema de TV	PAL
Resolución	380 líneas de TV.
Frecuencia de escaneado	50 Hz
Iluminación mínima	3 lux
Micrófono	Incorporado en cámara
Potencia de salida	50 mW
Frecuencia de salida	900MHz-1200MHz
Alimentador cámara	9 v /80 mA(pila), 8 v. (adaptador)
Alimentador receptor	12 v. DC
Duración de la pila 9v	4 a 5 horas
Peso aproximado kit	850 gr.

Tabla D.1 Características de la cámara

Contenido del Kit

- Minicámara inalámbrica color con sonido 50 mW.
- Receptor de audio y vídeo.
- Cable conexión audio vídeo.
- Adaptador para pila de 9v.
- Alimentador para minicámara
- Alimentador para receptor.

D.6 Servomotores Futaba S3003

Los servomotores utilizados para el movimiento del robot, son los producidos por la compañía Futaba y su principal uso es el de la industria del aeromodelismo, por lo que presentan un precio reducido debido a su elevada producción. Su aspecto es el de la Figura D.7.



Figura D.7. Servomotor Futaba S3003

Los servos se controlan aplicando una señal PWM por su cable de control. Las señales PWM (Pulse Width Modulation, Modulación por anchura de pulso) son digitales (pueden valer 0 ó 1).

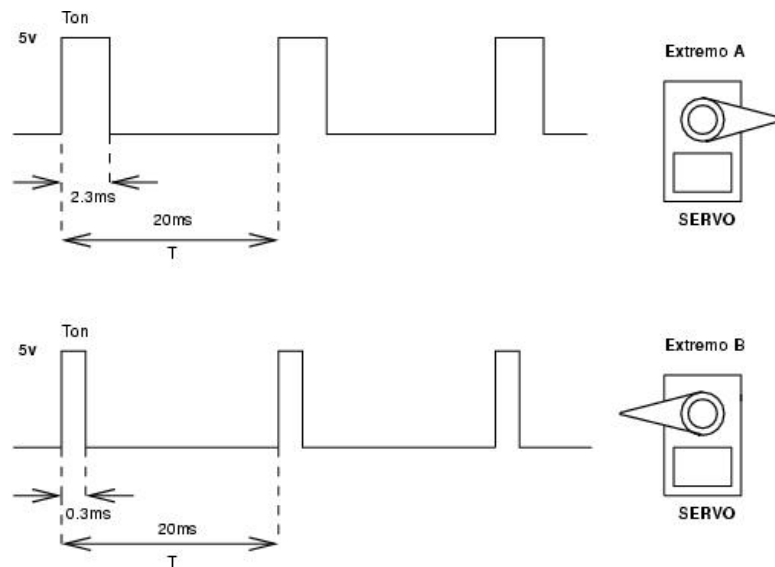


Figura D.8. Señales de control de un servomotor

Para posicionar el servo hay que aplicar una señal periódica, de 50Hz (20ms de periodo). La anchura del pulso determina la posición del servo. Si la anchura es de 2.3ms, el servo se sitúa en un extremo y si la anchura es de 0.3ms se sitúa en el opuesto. Cualquier otra anchura entre 0.3 y 2.3 sitúa el servo en una posición comprendida entre un extremo y otro. Por ejemplo, si queremos que se sitúe exactamente en el centro, aplicamos una anchura de 1.3ms.

Cuando se deja de enviar la señal, el servo entra en un estado de reposo, y por tanto se podrá mover con la mano. Mientras se le aplique la señal, permanecerá fijado en su posición, haciendo fuerza para permanecer en ella.

Las características técnicas de este servomotor se detallan a continuación:

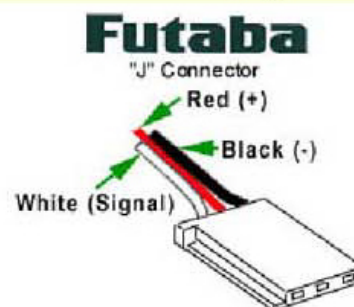
- Engranajes de Nylon.
- Un año de garantía.
- Voltaje de alimentación entre 4,8 y 6,0 V
- Par: 3,2 Kg-cm a 4,8 V y 4,1 Kg-cm a 6,0 V
- Velocidad de giro: 0,23 seg / 60º a 4,8 V y 0,19 seg / 60º a 6,0 V
- Dimensiones: 40 x 20 x 36 mm
- Peso: 37 g
- Conector tipo J.



...S3003 FUTABA SERVO...

Detailed Specifications

Control System:	+Pulse Width Control 1520usec Neutral	Current Drain (4.8V):	7.2mA/idle
Required Pulse:	3-5 Volt Peak to Peak Square Wave	Current Drain (6.0V):	8mA/idle
Operating Voltage:	4.8-6.0 Volts	Direction:	Counter Clockwise/Pulse Traveling 1520- 1900usec
Operating Temperature Range:	-20 to +60 Degree C	Motor Type:	3 Pole Ferrite
Operating Speed (4.8V):	0.23sec/60 degrees at no load	Potentiometer Drive:	Indirect Drive
Operating Speed (6.0V):	0.19sec/60 degrees at no load	Bearing Type:	Plastic Bearing
Stall Torque (4.8V):	44 oz/in. (3.2kg.cm)	Gear Type:	All Nylon Gears
Stall Torque (6.0V):	56.8 oz/in. (4.1kg.cm)	Connector Wire Length:	12"
Operating Angle:	45 Deg. one side pulse traveling 400usec	Dimensions:	1.6" x 0.8" x 1.4" (41 x 20 x 36mm)
360 Modifiable:	Yes	Weight:	1.3oz. (37.2g)





DESCRIPCIÓN DEL SOFTWARE UTILIZADO

En este anexo se detallarán las características de los distintos programas utilizados para el desarrollo del presente proyecto. Fundamentalmente han sido programas pertenecientes al ámbito del diseño y la simulación, al campo de la programación de aplicaciones informáticas y a la programación de microcontroladores.

E.1 Google Sketchup

Antes de comenzar a describir las características del programa seleccionado para realizar los diseños de las distintas piezas que componen el robot es necesario realizar una comparativa entre las alternativas existentes en el mercado.

La tabla C.1 muestra de forma esquemática un resumen del abanico de posibilidades.

	Autodesk 3ds max 9 SP2	Autodesk Maya complete 8.5	Blender 3D 2.45	SolidWorks	Sketchup Pro 6	Sketchup 6 (Free Versión)	Autodesk Softimage Mod Tool	Autodesk XSI	AC3D	Solid Edge
Mercado	Juegos	Juegos - Películas	Juegos - Películas	Modelado mecánico	Educación	Educación	Juegos	Juegos y efectos visuales	Juegos	Modelado mecánico
Precio Final	5,000 €	2,500 €	Gratis	7,500 €	331 €	Gratis	Gratis	2600 €	80 €	3578 €
Suscripción Anual	Si	Si	No	Si	No	No	Si	Si	No	Si
Plataformas										
Versión de prueba	Si	Si	No	Si	Si	Si	Si	Si	Si	Si
Idiomas	FR, EN	EN, JPN	EN	EN	EN ESP	EN ESP	EN	EN, JPN	EN	EN ESP
Idiomas incorporados	No	No	Todos incluidos	No	No	No	No	No	No	No
Popularidad en la industria	Muy buena	Muy buena	Baja	Buena	Normal	Normal	Norma	Muy buena	Normal	Muy buena
Tiempo de aprendizaje	< 2 meses	< 3 meses	< 3 meses	< 3 meses	< 1 mes	< 1 mes	< 2 meses	< 3 meses	< 2 mes	< 3 meses
Soporte para usuarios particulares	Regular	Bueno	No tiene	Regular	Bueno	Bueno	No tiene	Bueno	Bueno	Bueno
Interfaz	Limpia y potente	Flexible, potente, pero no intuitiva	Poco intuitiva	Práctica e intuitiva	Flexible y muy intuitiva	Flexible y muy intuitiva	Práctica e intuitiva	Flexible, potente, pero no intuitiva	Flexible y muy intuitiva	Potente y flexible.
Documentación	Buena	Excelente	Buena	Buena	Regular	Regular	Buena	Excelente	Buena	Excelente
DVD de entrenamiento	Buena	Muy buena	Regular	Buena	Excelente	Excelente	No tiene	No tiene	No tiene	Buena

Tabla E.1 Comparativa entre programas de diseño

Finalmente se eligió Sketchup como herramienta de diseño debido a su corto tiempo de aprendizaje, reducido precio, y porque incorpora un plugin de simulación física (Sketchyphysics) que fue el utilizado en las primeras simulaciones realizadas para validar los diseños mecánicos

SketchUp fue diseñado para usarlo de una manera intuitiva y flexible, facilitando ampliamente su uso en comparación con otros programas de modelado 3D. Cualquier persona, desde un niño hasta un adulto, pueden de manera muy sencilla aprender a utilizar esta herramienta para diseño tridimensional.

El programa también incluye en sus recursos un tutorial en vídeo para ir aprendiendo paso a paso cómo se puede ir diseñando y modelando el propio ambiente. Se recomienda a los

principiantes tener paciencia para seguir las explicaciones del tutorial (está disponible la versión en español). Como una introducción, el tutorial tiene una importancia básica para comprender lo sencillo, práctico y poderoso que en definitiva resulta ser este programa.

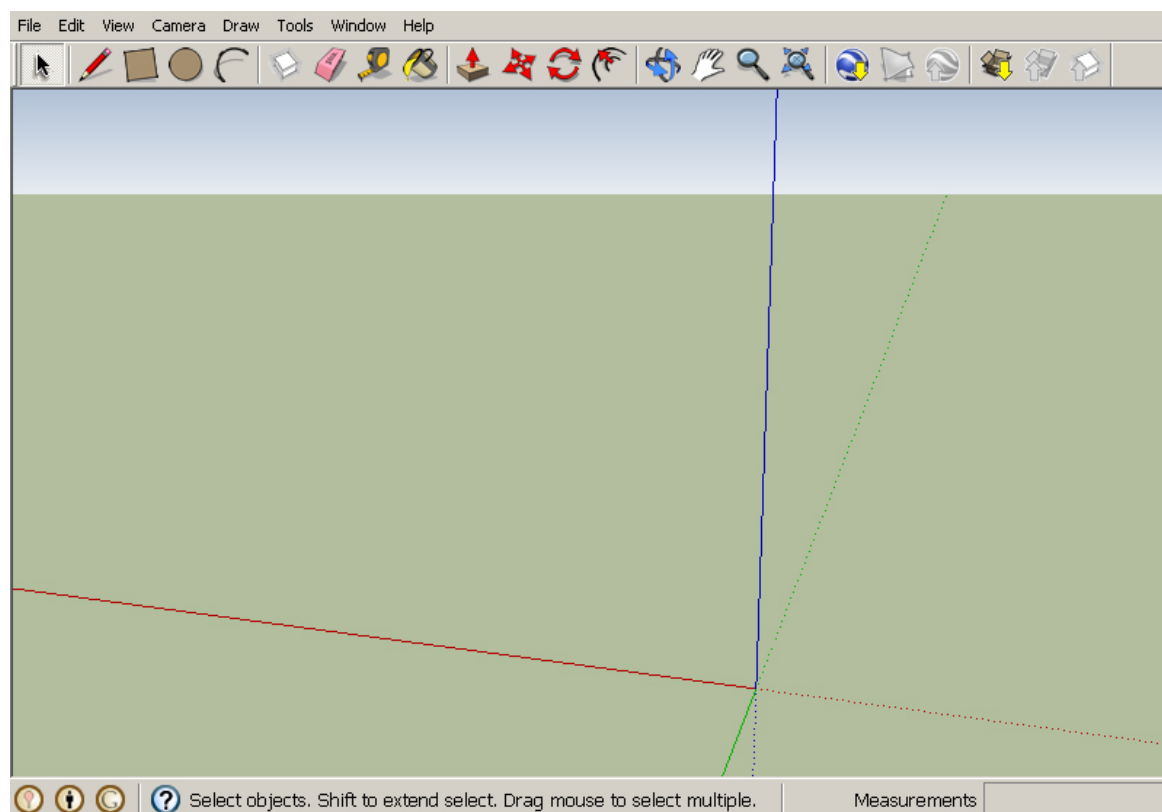


Figura E.1 Interfaz de usuario de Sketchup

SketchUp permite diseñar modelos en 3D de edificios, coches, personas y cualquier objeto o artículo dentro de la imaginación del diseñador o dibujante. Además, para mayor comodidad, el programa incluye una galería de objetos, texturas e imágenes para descargar.

E.1.1 SketchyPhysics

SketchyPhysics es un plugin para Google SketchUp que ejecuta simulaciones basadas en la física del mundo real. Esto significa que, durante la simulación, las leyes de la física newtoniana se aplican a los objetos agrupados, además cuenta con interactividad en tiempo de ejecución (el usuario es capaz de arrastrar objetos mientras se está ejecutando la simulación).

Asimismo es compatible con dispositivos de entrada como el teclado o joysticks que permiten controlar articulaciones, propulsores, emisores de partículas o imanes. También es posible el uso de scripts Ruby para realizar acciones más avanzadas.

El plugin SketchyPhysics usa el motor de física Newton de Newton Game Dynamics para llevar a cabo las simulaciones.

E.2 V-REP

V-REP (de Virtual Robot Experimentation Platform) es el otro programa utilizado para realizar las simulaciones físicas del robot, a diferencia de Sketchup en este programa solamente se importaron los modelos ya diseñados, porque aunque posee capacidades de edición de primitivas no resulta tan intuitivo y potente como la alternativa ofrecida por Google.

Este programa es comercial, aunque tiene una versión de estudiantes gratuita que se utilizó para trabajar; para realizar las simulaciones utiliza tanto los motores físicos Bullet como ODE (Open Dynamics Engine) y posee capacidades mucho más potentes y orientadas hacia la robótica que Sketchyphysics.

Entre sus funcionalidades destacan:

- Detección de colisiones, permitiendo control de interferencia entre geometrías de forma rápida y sencilla
- Capacidad de cálculo de mínima distancia, permitiendo el cálculo para cualquier tipo de geometría
- Algoritmos especializados dedicados a cálculos de cinemática directa e inversa
- Algoritmos de planificación de trayectorias tanto para entidades holonómicas como no holonómicas
- Simulación de una amplia Variedad de sensores como son: sensores de proximidad, par, contacto y cámaras de vídeo, además de capacidad de procesamiento de imagen
- Simulación de operaciones de corte de materiales o de pintura.
- Control distribuido, lo que permite tener un número ilimitado de programas ejecutándose en paralelo
- Capacidad de recolección y exportación de datos para posterior análisis

E.3 JVM-Servo

Este programa fue el utilizado para controlar los servomotores de la primera generación de robot serpiente, funciona junto con la tarjeta controladora de servos JVM-Servo 28 y presenta un interfaz de usuario como la mostrada en la Figura E.2.



Figura E.2 Interfaz de usuario de JVM-Servo

El programa está destinado para el control de hasta 28 servos de RC enviando la posición a 28 servos simultáneamente a través del puerto USB.

La herramienta permite crear los puntos de posición gráficamente y basándose en el tiempo/cuadro transcurrido, de forma que se convierte en un modo muy sencillo de crear interpolaciones graficas para animaciones complejas. Las líneas y puntos son del color perteneciente a la pista así como los valores numéricos que se muestran en la pantalla. También dispone de ayudas visuales que agilizaran la comprensión de los comandos, manipuladores, puntos y posiciones.

Una de las características más notables de la herramienta es la potencia que ofrece la posibilidad de desarrollar las posiciones de movimiento en tiempo real, es decir, podemos poner en modo de reproducción el secuenciador e ir creando las coordenadas de posición en tiempo real sin necesidad de detenerlo. El programa recalcula siempre la posición en la que se encuentra el cursor y la envía para que el microcontrolador procese la posición al servo correspondiente.

3- ENTORNO IDE

Para una mejor versatilidad de manejo y adaptación a los diferentes tamaños de pantalla soportados actualmente en el mercado, el software está dividido en tres ventanas principales: Multi-pistas, Mixer y el navegador. Cada una de ellas se encarga de unas funciones específicas.

En la imagen de la derecha identificamos cuál es cada una de ellas:

La ventana multi-pistas es la principal se encarga de mostrar las pistas del secuenciador, entre otros controles, dispone de zoom biaxial para conseguir una visualización perfecta de la zona de trabajo, control de cada pista y visualización de coordenadas del ratón.

La pantalla de navegación (navegador) nos permite desplazarnos por la secuencia además de ofrecer el control de la posición y de la reproducción. En esta ventana encontramos los controles comunes de reproducción como son:

PLAY, STOP, FF y REW, también se dispone de unas funciones específicas para desplazar el cursor de reproducción frame a frame o situarnos en una posición establecida numéricamente.

La pantalla de Mixer nos muestra la posición (en modo gráfico) de cada pista en cada momento y posición del cursor. En esta ventana también tenemos otros controles adicionales para una mejor flexibilidad a la hora de componer nuestras secuencias de movimientos, On-Off de pista, establecer posición de inicio... etc.



A continuación vamos a detallar el significado y función de cada botón en el entorno IDE

3.1 - MIXER

El mezclador es una ventana que puede ser redimensionada horizontalmente.

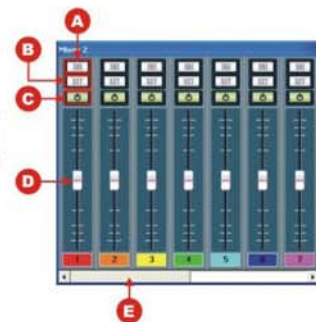
El botón **A** corresponde a la función INI, al pulsarlo se establece la posición actual en la que se encuentre el servo de ese canal como posición de inicio, de forma que cuando iniciemos el hardware el servo se posicionará en ese valor.

El botón **B** sirve para enviar la posición actual en la que se encuentra el cursor del soft, esta función es útil cuando tenemos deshabilitada la opción "Enviar la posición continuamente en play" en las opciones del programa.

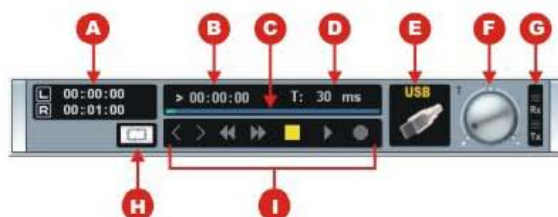
Botón **C**: sirve para deshabilitar el envío de posición al canal correspondiente.

Botón **D**: fader para establecer o para visualizar la posición actual de la pista correspondiente.

Botón **E**: Es un control de scroll que permite navegar por los canales, aparece en caso de que el número de canales cargados exceda el largo de la ventana del mezclador.



3.2 – NAVEGADOR



La zona del punto **A**: muestra la posición de las líneas L y R. El valor está separado en tres pares de dígitos para una mejor precisión del ajuste. Puede establecer un nuevo valor haciendo doble clic en cualquiera de los tres dígitos.

El punto **B**: Nos muestra el contador del cursor y por lo tanto muestra la posición actual. Dicha posición también puede cambiarse haciendo doble clic en cualquiera de los tres pares de dígitos.

El punto **C**: nos señala la barra de navegación interactiva, esta barra está en escala y comprende desde el inicio (posición frame 0) hasta el punto creado más lejano de nuestra secuencia. Situándonos sobre él y por medio del ratón podemos situar el cursor en la posición que deseemos de la secuencia.

El punto **D**: Indica el valor actual del tiempo (tempo), en la imagen podemos observar que el programa está establecido para enviar la posición al hardware cada 50 milisegundos en el modo PLAY.

El punto **E**: Nos indica en qué modo de comunicación está establecido el software USB o RS232. Este parámetro se podrá modificar desde las opciones del programa en una futura versión.

El punto **F**: Corresponde al botón que regula el tiempo de ejecución (tempo), para variarlo sitúese con el ratón sobre él, pulse el botón derecho y manteniéndolo pulsado desplácelo hacia arriba o hacia abajo. El rango de trabajo es desde 0 hasta 500 milisegundos.

El punto **G**: muestra los indicadores de comunicación y sirven para mostrar el flujo de datos con el hardware.

El punto **H**: es el botón de activación de bucle (loop). Al pulsarlo le indicamos al programa que ha de permanecer en bucle entre las líneas L y R. Una peculiaridad a tener en cuenta es que si la línea L está situada en una posición más avanzada que la línea R y el botón de bucle activado, lo que ocurrirá es que: cuando el cursor alcance la posición de la línea R saltará inmediatamente hasta la posición de la línea L (no hará bucle, efectuará un único salto de cursor).

El punto **I**: nos muestra los botones de control que detallamos abajo:

CTRL	FUNCION
<	Retroceso del cursor frame a frame.
>	Avance del cursor frame a frame.
<<	Retroceso rápido del cursor.
>>	Avance rápido del cursor.
■	Stop, detención inmediata del cursor en modo play y/o situación a la posición cero cuando no está en play y el cursor no está en cero.
▶	Play
●	Habilita/deshabilita el modo de grabación en tiempo real.

Nota: El navegador tiene la propiedad 'siempre visible' para que permanezca en primer plano, esto es útil para poder situarlo en cualquier sitio de la pantalla, siempre se mostrará arriba.

3.3 – MULTI-PISTAS

Punto **A**: Indicador escalado con forma de regla para tener la referencia visual de la posición actual.

Punto **B**: Zona destinada a las secuencias.

Punto **C**: Control de pistas vertical. Este control nos permite visualizar las pistas que permanecen ocultas por un exceso de zoom vertical.

Punto **D**: Control de Zoom vertical. Ajusta el número de pistas que se van a visualizar verticalmente.

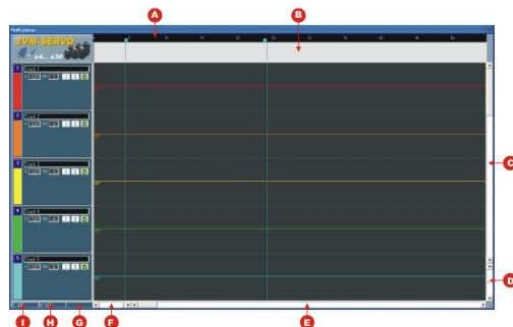
Punto **E**: Permite el desplazamiento horizontal por la zona de puntos.

Punto **F**: Control del zoom horizontal. Ajusta la visión para expandir o contraer la zona que pretendemos visualizar.

Punto **G**: Indicador de la posición X del punto que tengamos seleccionado.

Punto **H**: Valor vertical de la pista. Este valor está comprendido entre 0 y 256 y es útil para saber el valor del punto que vamos a crear, ya sea con doble clic o con la tecla "C" del PC.

Punto **I**: Nos indica la posición frame en la que se encuentra el ratón, es útil para saber el valor del frame en el punto que vamos a crear, ya sea con doble clic o con la tecla "C" del PC.



E.4 X-CTU

X-CTU es una aplicación basada en Windows proporcionada por Digi. Este programa fue diseñado para interactuar con los archivos de firmware que se encuentran en los productos Digi RF y proporcionar una interfaz gráfica fácil de utilizar por el usuario final.

El uso de este programa resulta fundamental para configurar apropiadamente los módulos Xbee, así como para poder programar la memoria no volátil que llevan incorporada mediante comandos específicos de estos módulos. Una vez ejecutado el programa presenta el aspecto de la Figura E.3.

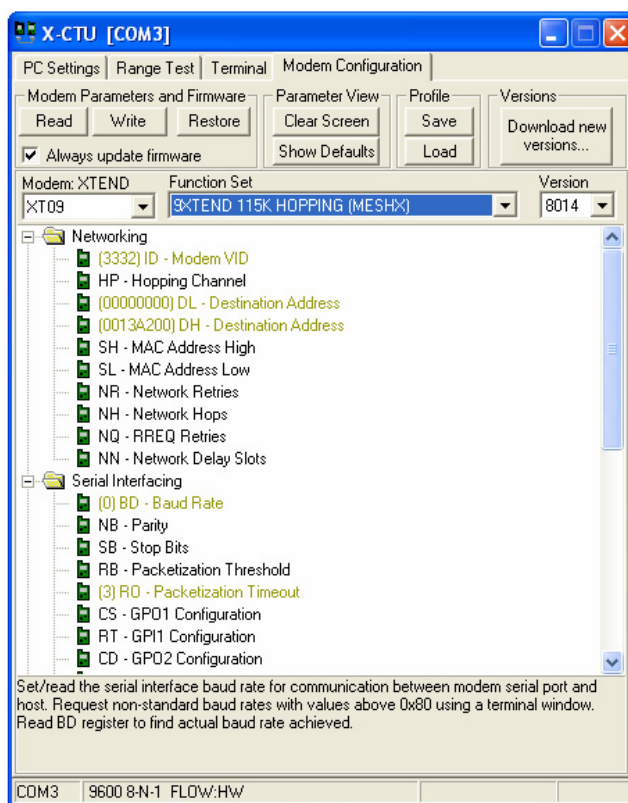


Figura E.3 Interfaz del programa X-CTU

La descripción completa de las funcionalidades de cada una de las pestañas puede encontrar se en su manual de usuario.

Introduction

This User's Guide is intended to discuss the functions of Digi's X-CTU software utility. Each function will be discussed in detail allowing a better understanding of the program and how it can be used.

X-CTU is a Windows-based application provided by Digi. This program was designed to interact with the firmware files found on Digi's RF products and to provide a simple-to-use graphical user interface to them.

X-CTU is designed to function with all Windows-based computers running Microsoft Windows 98 SE and above. X-CTU can either be downloaded from Digi's Web site or an installation CD. When properly installed it can be launched by clicking on the icon on the PC desktop (see Figure 1) or selecting from the Start menu (see Figure 2).



Figure 1



Figure 2

When launched, you will see four tabs across the top of the program (see Figure 3). Each of these tabs has a different function. The four tabs are:

PC Settings: Allows a customer to select the desired COM port and configure that port to fit the radios settings.

Range Test: Allows a customer to perform a range test between two radios.

Terminal: Allows access to the computers COM port with a terminal emulation program. This tab also allows the ability to access the radios' firmware using AT commands (for a complete listing of the radios' AT commands, please see the product manuals available online).

Modem Configuration: Allows the ability to program the radios' firmware settings via a graphical user interface. This tab also allows customers the ability to change firmware versions.

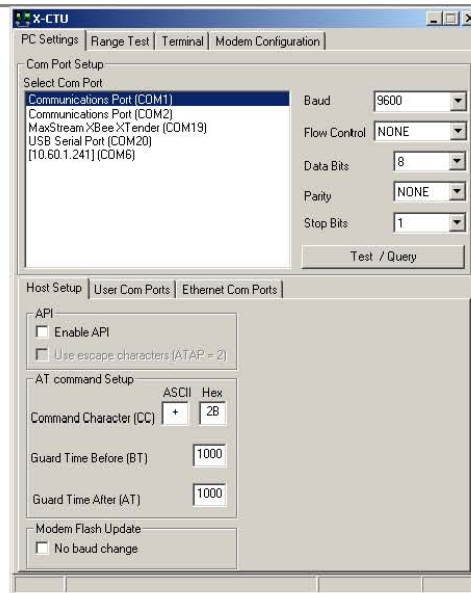


Figure 3

PC Settings Tab

When the program is launched, the default tab selected is the "PC Settings" tab. The PC Settings tab is broken down into three basic areas: The COM port setup, the Host Setup, and the User Com ports.

COM port setup:

The PC settings tab allows the user to select a COM port and configure the selected COM port settings when accessing the port. Some of these settings include:

Baud Rate:	Both standard and non-standard
Flow Control:	Hardware, Software (Xon/Xoff), None
Data bits:	4, 5, 6, 7, and 8 data bits
Parity:	None, Odd, Even, Mark and Space
Stop bit:	1, 1.5, and 2

To change any of the above settings, select the pull down menu on the left of the value and select the desired setting. To enter a non-standard baud rate, type the baud rate into the baud rate box to the left.

The **Test / Query** button is used to test the selected COM port and PC settings. If the settings and COM port are correct, you will receive a response similar to the one depicted in Figure 4 below.



Figure 4

Host Setup:

The Host Setup tab allows the user to configure how the X-CTU program is to interface with a radio's firmware. This includes determining whether API or AT command mode will be used to access the module's firmware as well as the proper command mode character and sequence.

By default, the Host Settings are as follows:

API mode:	not enabled (Not checked)
Command mode Character:	+ (ASCII) 2B (Hex).
Before Guard Time:	1000 (1 Sec)
After Guard Time:	1000 (1 Sec)

This is the default value of our radios. If this is not the value of the AT, BT, or GT commands of the connected radio, enter the respective value here.

User COM ports:

The user COM port option allows the user to "Add" or "Delete" a user-created COM port. This is only for temporary use. Once the program has closed, the user-created COM port will disappear and is no longer accessible to the program.

Range Test Tab

The range test tab is designed to verify the range of the radio link by sending a user-specified data packet and verifying the response packet is the same, within the time specified. For performing a standard range test, please follow the steps found in most Quick Start or Getting Started Guides that ship with the product.

Packet Data and Size

By default, the size of the data packet sent is 32 bytes. This data packet specified can be adjusted in either size or the text sent.

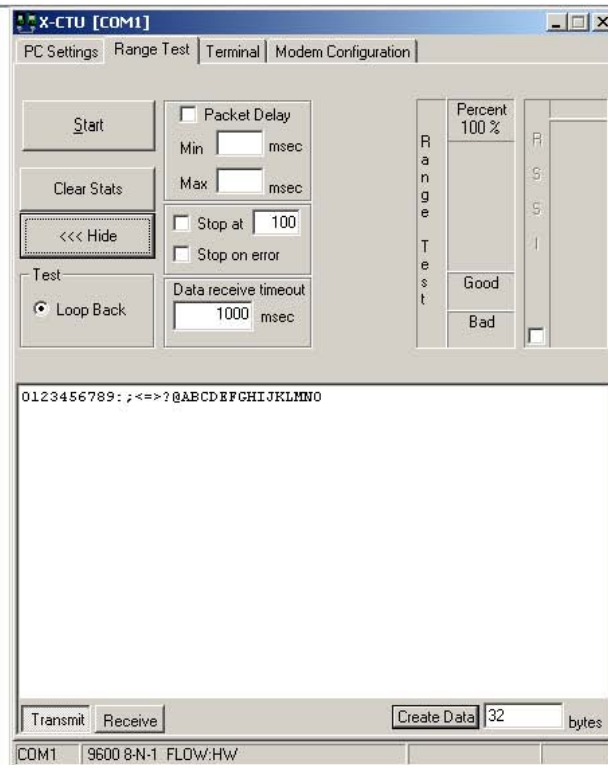


Figure 5

To modify the size of the packet sent, change the value next to the "Create Data" box and click on the "Create Data" button (see Figure 5). If you want to change the data sent, delete the text in the transmit window and place in your desired text.

By modifying the text, data packet size, packet delay and the data receive timeout; the user is able to simulate a wide range of scenarios.

RSSI:

The RSSI option of the X-CTU allows the user to see the RSSI (Received Signal Strength Indicator) of a received packet when performing a range test.

API Function:

The X-CTU also allows the user to test the API function of a radio during a range test.

To perform a range test with the API function of the radio, follow the steps outlined below:

- 1: Configure the Base with API enabled and a unique 16 bit or 64 bit source address.
- 2: Configure the remote radio with a unique source address and set the Destination address to equal the Base radio's source address.
- 3: Enable the API option of the X-CTU on the PC Settings tab and connect the base radio to the PC (See Figure 3).
- 4: Connect the red loopback adapter to the remote radio and place them a distance apart.
- 5: Enter either the 16 bit or 64 bit destination address of the remote radio into the Destination Address box on the Range Test tab (See figure 6).
- 6: Create a data packet of your choosing by typing in the data in the Transmit box
- 7: To start a Range test, click on Start.

You will notice the TX failures, Purge, CCA, and ACK messages will increment accordingly while the range test is performed.

To stop a range test, click on the Stop button.

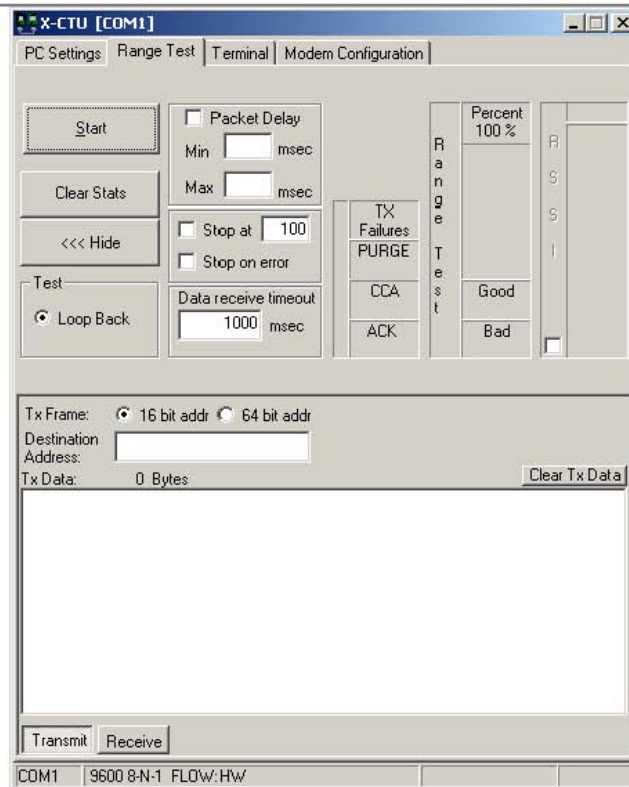


Figure 6

The Terminal Tab

The Terminal tab has three basic functions:

- Terminal emulator
- Ability to send and receive predefined data packets (Assemble packet)
- Ability to send and receive data in Hex and ASCII formats (Show/Hide hex)

The main terminal window

The main white portion of this tab is where most of the communications information will occur while using X-CTU as a terminal emulator. The text in **blue** is what has been typed in and directed out to the radio's serial port while the **red** text is the incoming data from the radio's serial port (see Figure 7).

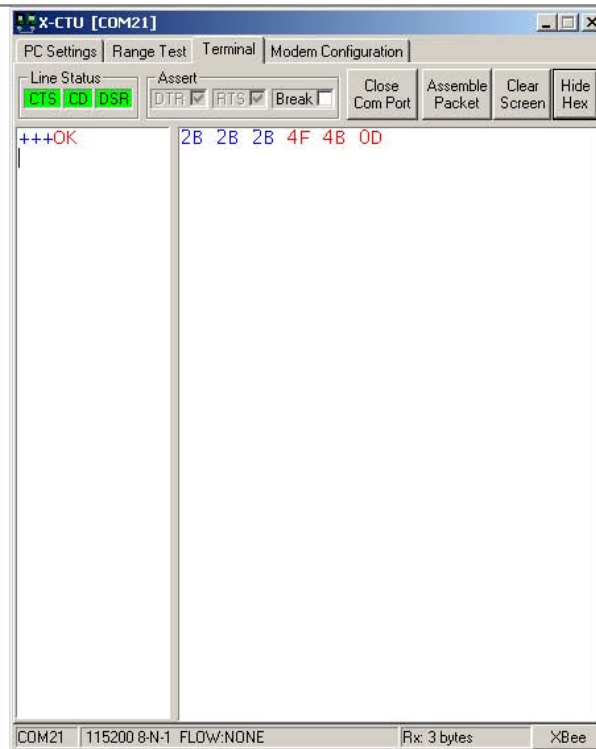


Figure 7

Assemble Packet

The Assemble Packet option on the Terminal tab is designed to allow the user to assemble a data packet in either ASCII or Hex characters. This is accomplished by selecting the Assemble packet window and choosing either ASCII (default) or Hex. Once selected, the data packet is assembled by typing in the desired characters as depicted in Figure 8.

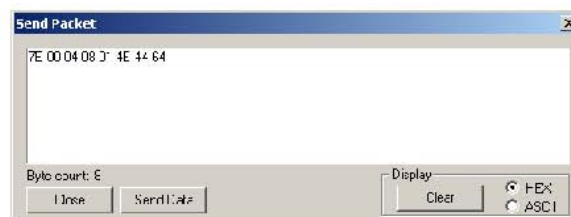


Figure 8

The **Line Status** indicators depicted in Figure 5 shows the status of the RS-232 hardware flow control lines. Green indicates the line is asserted while black indicates de-asserted.

The **Break** option is for engaging the serial line break. This can be accomplished by checking or asserting the Break option. Asserting the Break will place the DI line high and prevent data from being sent to the radio.

Modem Configuration tab

The Modem configuration tab has four basic functions:

- 1: Provide a Graphical User Interface with a radio's firmware
- 2: Read and Write firmware to the radio's microcontroller
- 3: Download updated firmware files from either the web or from a compressed file
- 4: Saving or loading a modem profile

Reading a radio's firmware

To read a radio's firmware, follow the steps outlined below:

- 1: Connect the radio module to the interface board and connect this assembly or a packaged radio (PKG) to the PC's corresponding port (IE: USB, RS232, Ethernet etc.).
- 2: Set the PC Settings tab (see Figure 3) to the radio's default settings.
- 3: On the Modem Configuration tab, select "Read" from the Modem Parameters and Firmware section (see Figure 9).

Making changes to a radio's firmware

Once the radio's firmware has been read, the configuration settings are displayed in three colors (see Figure 10):

- Black – not settable or read-only
- Green – Default value
- Blue – User-specified

To modify any of the user-settable parameters, click on the associated command and type in the new value for that parameter. For ease of understanding a specific command, once the command is selected, a quick description along with its limits is provided at the bottom of the screen. Once all of the new values have been entered, the new values are ready to be saved to the radio's non-volatile memory.

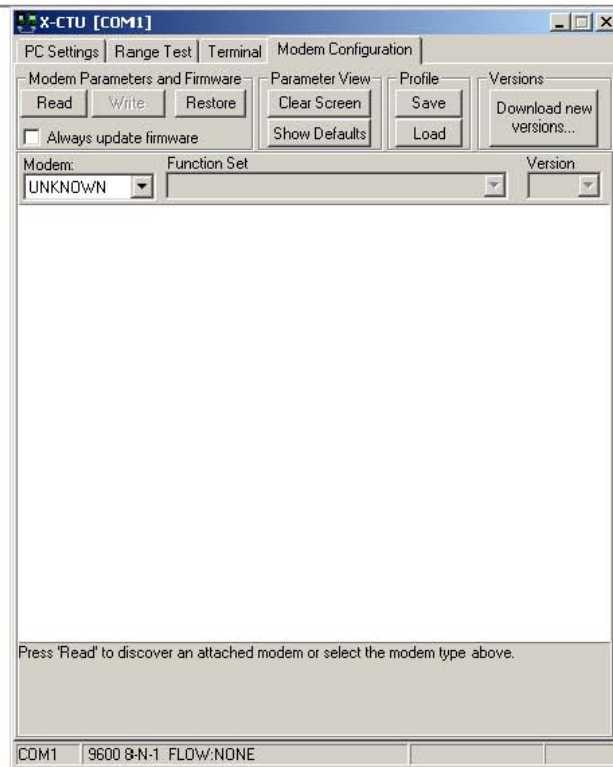


Figure 9

Writing firmware to the Radio

To write the parameter changes to the radio's non-volatile memory, click on the Write button located in the Modem Parameters and Firmware section (see Figure 10)

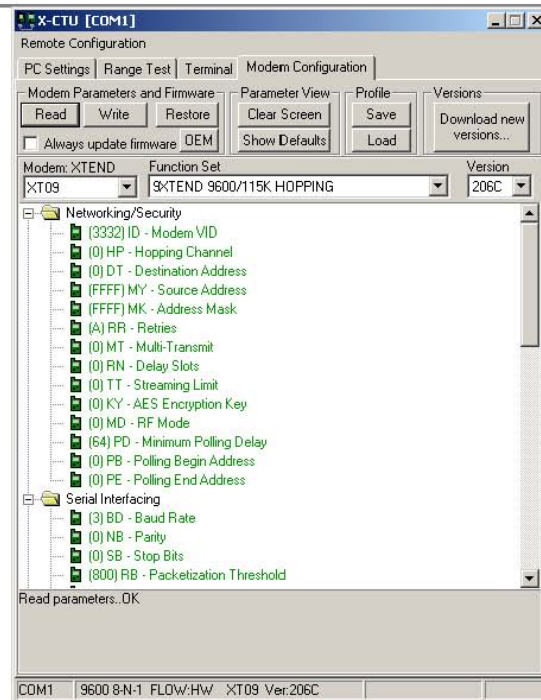


Figure 10

Downloading Updated Firmware Files

Another function of the Modem Configuration tab is allowing the user to download updated firmware files from either the web or install them from a disk or CD. This is accomplished by following the steps below:

- 1: Click on the Download New Versions... option under the Version section
- 2a: Click on Web for downloading new firmware files from the web
- 2b: Click on the File when installing compressed firmware files from a CD or saved file (see Figures 11 and 12)
 - 2bi: Browse to the location the file is saved at and click on Open (see Figure 13)
- 3: Click on OK and Done when prompted

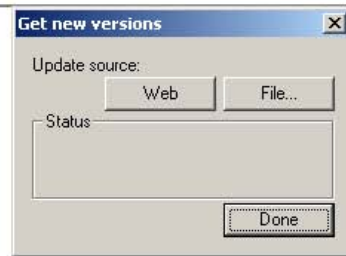


Figure 11



Figure 12

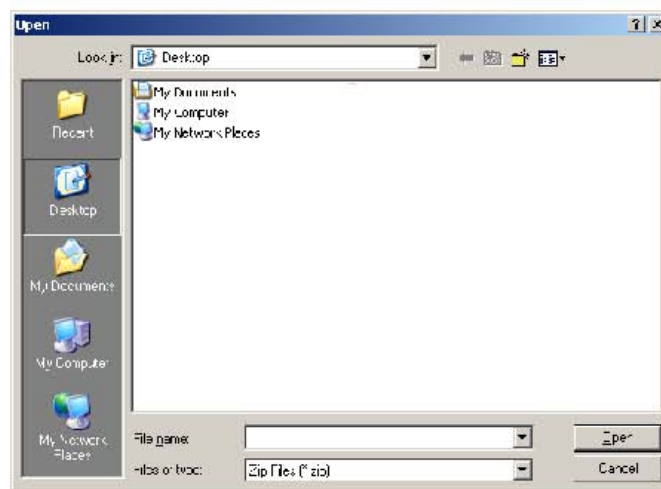


Figure 13

Modem Profiles

The X-CTU has the ability to save and write saved modem profiles or configuration to the radio. This function is useful in a production environment when the same parameters need to be set on multiple radios.

How to save a profile:

- 1: Set the desired settings within the radio's firmware as described in the Making changes to the radios firmware section
- 2: Click Save in the Profile section
- 3: Type in the desired name of this profile in the File Name box (see Figure 14)
- 4: Browse to the location where you wish to save your profile
- 5: Click Save

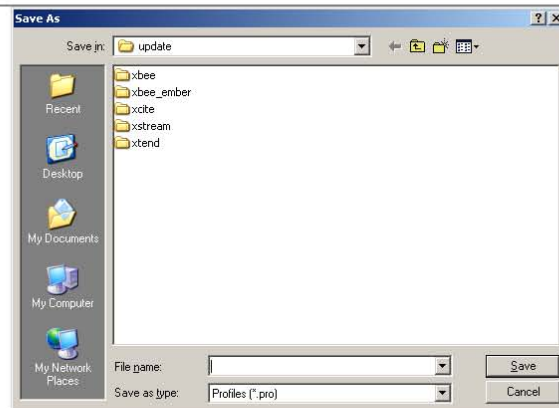


Figure 14

How to load a saved profile:

- 1: Click on Load from the profile section
- 2: Browse to the location of the file and click on the desired file (see Figure 15)
- 3: Click Open

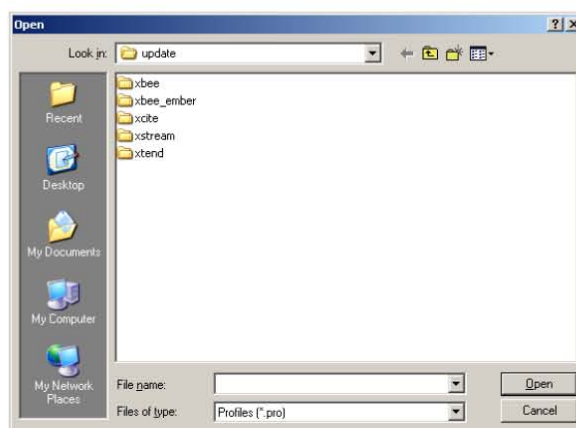


Figure 15

To save the loaded profile to the radio once you have loaded the file, follow the steps outlined in the [Writing firmware to the radio](#) section above.

To find out how to load the saved profiles in a production environment from a DOS prompt, please follow the steps outlined in Digi's online Knowledgebase at

<http://www.maxstream.net/support/knowledgebase/article.php?kb=126>

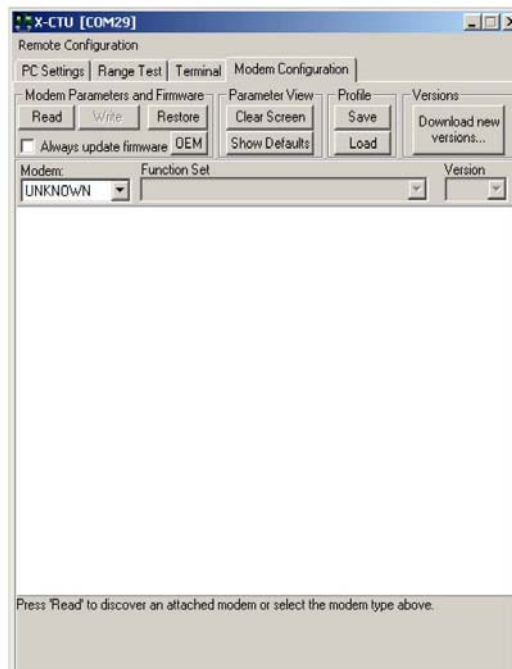
Remote Modem Management

XBee 802.15.4 modules with firmware version 1xCx and above, XBee ZNet 2.5 modules, and XBee ZB modules offer the ability to be configured with over the air commands. With the addition of this new feature, the user is able to configure remote radio parameters with X-CTU or API packets. To use the remote configuration tool, the following is required:

- The radio connected to the PC must be in API mode
- The remote radio must be associated or within range of the base radio

To access remote radios through X-CTU's Modem Configuration tab, perform the steps below:

- Enable API on the PC Settings tab
- Verify the COM port selection and settings
- On the Modem Configuration tab, select the Remote Configuration option on the top left corner of the program



- Select Open Com port
- Select Discover