Universidad Politécnica de Cartagena

industriales
etsii UPCT

CARACTERIZACIÓN Y MONITORIZACIÓN DEL COMPORTAMIENTO DEL BRAZO ROBOT MODULAR

Titulación: I.T.I. esp. Electrónica Industrial.
Intensificación: Automática.
Alumna: Cristina Díaz Díaz
Director: José Luis Muñoz Lozano

Cartagena, 9 de Marzo de 2012

Índice

1. Introducción acerca del proyecto.

1.1. Antecendetes.

Vivimos en un mundo cuya tecnología avanza a pasos agigantados. La historia de lo acontecido con la tecnología durante el siglo XX muestra lo imposible que hubiera sido a fines del siglo XIX vaticinar el futuro. En el caso de la robótica, ésta también ha experimentado un rápido crecimiento y en la actualidad encontramos proyectos muy ambiciosos relacionados con la robótica doméstica hasta el punto de alcanzar visiones semejantes a las que podemos encontrar en cualquier film de ciencia ficción.

En un principio, la robótica surgió como una respuesta a los trabajos de esfuerzo para el hombre, sin embargo, en el último siglo parece haberse propuesto suplantar al hombre en todos sus aspectos y es que uno de los ámbitos en los que se han conseguido grandes avances ha sido en la robótica humanoide, llegando a conseguir robots que pueden simular los sentimientos de las personas.

En mi caso, la tecnología y, en concreto, la robótica siempre han sido algo que ha despertado mi atención. Quizás, este fue el motivo principal para decidir entrar en Ingeniería Electrónica; Sin embargo, no ha sido hasta mi último año de carrera, cuando he podido tener un primer contacto con este mundo a través de la optativa de Robótica Industrial. El hecho de que mi proyecto de final de carrera esté orientado hacia la robótica industrial ha sido principalmente el poder profundizar en un tema que me resulta tan interesante como éste y poder tener, así, contacto directo con un equipo, poder investigar acerca de su funcionamiento, su morfología, su método de comunicación con el medio, etc. y no quedar meramente con la teoría, que, aunque necesaria, a veces resulta difícil de relacionar con la realidad que conocemos. Digo esto debido a que el contenido que se estudió durante la optativa, en principio pudo parecerme abstracto, pero con el desarrollo de este proyecto he podido ir relacionando la teoría con el robot físico, de modo que ésta se hace mucho más fácil de comprender.

1.2. Objetivos.

El proyecto se centra principalmente en el desarrollo de una aplicación que interactúa con el robot, monitorizando en cada momento los valores que adopta éste en cuanto a sus variables eléctricas características (corriente, tensión), así como a variables mecánicas (velocidad, aceleración, posición, etc.).

La aplicación pretende alcanzar un seguimiento en tiempo real, es decir, la actualización automática de los datos, de forma que se tenga un conocimiento constante a lo largo del tiempo de cuál es la situación del robot.

Asimismo, se ha pretendido llevar a cabo un estudio en relativa profundidad del equipo de trabajo, no sólo del sistema robótico, sino de todo su sistema de comunicaciones y método de funcionamiento de éste.

1.3. Descripción de la memoria.

Como ya se ha mencionado, el proyecto se ha desarrollado conforme a dos procesos distintos:

En primer lugar, ha sido necesario un proceso de familiarización con el equipo de trabajo a través de los correspondientes manuales suministrados por el fabricante, así como la obligada profundización en conceptos relacionados con la robótica y términos vinculados a la ingeniería de comunicaciones, ya que resultan imprescindibles para la comprensión del funcionamiento del sistema. Como resultado, en la memoria de este proyecto van a quedar desarrollados y expuestos los estudios realizados referentes tanto a la cinemática del robot que van desde la obtención de las matrices de transformación homogénea o la obtención de los parámetros de Denavit-Hartenberg hasta la resolución del problema cinemático directo a través de éstos algoritmos, como al protocolo de comunicaciones en que está basada la conexión PC-robot.

Y, en segundo lugar, el proceso de desarrollo de la aplicación que interactuaría con el robot. La interfaz ha sido desarrollada con el software de trabajo Microsoft Visual Studio, empleando el lenguaje de programación C++. Aunque en un primer momento se planteó el objetivo de una directa comunicación aplicación-robot, debido a la falta de material de trabajo, información o la complejidad de ésta, finalmente se optó por el desarrollo de una aplicación secundaria que simularía el envío de datos desde el robot a la aplicación principal.

La aplicación desarrollada está basada en la programación de ventanas, de forma que al ejecutarla se nos muestra una serie de ventanas, con información distribuida según unos criterios (que se explicarán más adelante) en varias pestañas donde el usuario puede leer la información útil que proporciona el robot.

Siguiendo con este planteamiento, la memoria se ha divido en seis apartados de contenido, más un séptimo correspondiente a la bibliografía.

Este, el primer apartado, es una breve introducción al proyecto, en la que se mencionan los motivos de elección de este trabajo, como se ha estructurado y los objetivos finales a conseguir.

El siguiente capítulo, desarrolla una breve introducción al mundo de la robótica, cómo ésta ha ido evolucionando a lo largo de la historia. Este capítulo es un preámbulo a lo que será la presentación del equipo de trabajo objeto de este proyecto.

A continuación, le siguen dos capítulos en los que se condensa la información que ha sido extraída y estudiada sobre el robot, su composición física, su método de comunicación con el medio, la programación que emplea, las librerías, así como otras herramientas de utilidad que suministra el fabricante, y un capítulo dedicado a la comunicación a través del bus CAN.

El capítulo siguiente es una descripción de la aplicación desarrollada, donde se detallan los aspectos característicos de la herramienta, así como el software empleado para su creación.

Por último, se incluyen unas conclusiones extraídas del trabajo y la bibliografía de la cual se ha podido extraer información necesaria para el desarrollo de este proyecto.

2. Introducción a la robótica.

2.1. Definición de robot industrial.

Existen ciertas dificultades a la hora de establecer una definición formal de lo que es un robot industrial. La primera de ellas surge de la diferencia conceptual entre el mercado japonés y el euro-americano de lo que es un robot y lo que es un manipulador. Así, mientras que para los japoneses un robot industrial es cualquier dispositivo mecánico dotado de articulaciones móviles destinado a la manipulación, el mercado occidental es más restrictivo, exigiendo una mayor complejidad, sobre todo en lo relativo al control. En segundo lugar, y centrándose ya en el concepto occidental, aunque existe una idea común acerca de lo que es un robot industrial, no es fácil ponerse de acuerdo a la hora de establecer una definición formal. Además, la evolución de la robótica ha ido obligando a diferentes actualizaciones de su definición.

La definición más comúnmente aceptada posiblemente sea la de la Asociación de Industrias Robóticas (RIA), según la cual:

"Un robot industrial es un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas".

Esta definición, ligeramente modificada, ha sido adoptada por la Organización Internacional de Estándares (ISO) que define al robot industrial como:

"Manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas".

Se incluye en esta definición la necesidad de que el robot tenga varios grados de libertad. Una definición más completa es la establecida por la Asociación Francesa de Normalización (AFNOR), que define primero el manipulador y, basándose en dicha definición, el robot:

Manipulador: *"Mecanismo formado generalmente por elementos en serie, articulados entre sí, destinado al agarre y desplazamiento de objetos. Es multifuncional y puede ser gobernado directamente por un operador humano o mediante dispositivo lógico".*

Robot: *"Manipulador automático servo-controlado, reprogramable, polivalente, capaz de posicionar y orientar piezas, útiles o dispositivos especiales, siguiendo trayectoria variables reprogramables, para la ejecución de tareas variadas. Normalmente tiene la forma de uno o varios brazos terminados en una muñeca. Su unidad de control incluye un dispositivo de memoria y ocasionalmente de percepción del entorno. Normalmente su uso es el de realizar*

una tarea de manera cíclica, pudiéndose adaptar a otra sin cambios permanentes en su material".

Así, observamos como característica común en todas las definiciones anteriores la aceptación del robot industrial como un brazo mecánico con capacidad de manipulación y que incorpora un control más o menos complejo.

2.2. Evolución histórica del robot.

Por siglos el ser humano ha construido máquinas que imitan las partes del cuerpo humano. Ejemplos claros de esto son, por ejemplo, los antiguos egipcios, que unían brazos mecánicos a las estatuas de sus dioses; los griegos, que construían estatuas que operaban con sistemas hidráulicos; o los europeos durante los siglos XVII y XVIII, que construyeron muñecos mecánicos muy ingeniosos que tenían algunas características de robots, como es el caso de Jacques de Vauncansos quien construyó varios músicos de tamaño humano a mediados del siglo XVIII.

En 1805, Henri Maillardert construyó una muñeca mecánica que era capaz de hacer dibujos. Ya durante la época de la revolución industrial hubo otras invenciones mecánicas, creadas por mentes de igual genio, muchas de las cuales estaban dirigidas al sector de la producción textil. Entre ellas se puede citar la hiladora giratoria de Hargreaves (1770), la hiladora mecánica de Crompton (1779), el telar mecánico de Cartwright (1785) o el telar de Jacquard (1801), entre otros.

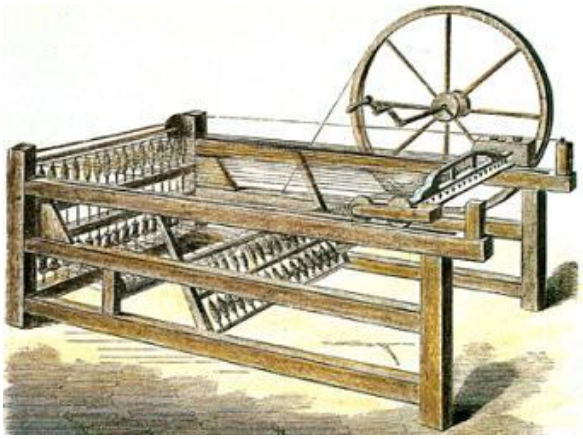


Figura2.1.- Hiladora de Hargreaves (1770).

Ya más tarde, el desarrollo del brazo artificial multi-articulado, o manipulador, es lo que llevará al desarrollo de los robots actuales. El inventor estadounidense George Devol

desarrolló en 1954 un brazo primitivo que se podía programar para realizar tareas específicas. En 1975, el ingeniero mecánico estadounidense Víctor Scheinman, cuando estudiaba la carrera en la Universidad de Stanford, en California, desarrolló un manipulador polivalente realmente flexible conocido como Brazo Manipulador Universal Programable (PUMA, siglas en inglés). El PUMA era capaz de mover un objeto y colocarlo en cualquier orientación en un lugar deseado que estuviera a su alcance. El concepto básico multi-articulado del PUMA es la base de la mayoría de los robots actuales.

El desarrollo en la tecnología, donde se incluye el desarrollo de las poderosas computadoras electrónicas, los actuadores de control retroalimentados, transmisión de potencia a través de engranajes y la tecnología en sensores han contribuido a flexibilizar los mecanismos autómatas para desempeñar tareas dentro de la industria. Son varios los factores que intervienen para que se desarrollaran los primeros robots en la década de los años 50. La investigación en inteligencia artificial desarrolló maneras de emular el procesamiento de información humana con computadoras electrónicas e inventó una variedad de mecanismos para probar sus teorías. En cuanto a robots móviles, en un principio, los robots se movían gracias a una rueda y fueron utilizados para llevar a cabo investigaciones sobre conducta, navegación, y planeo de ruta. Más adelante se intentó con robots de múltiples piernas, los más desarrollados en aquella época son los de seis y cuatro patas debido a que eran más estables, lo que los hace más fáciles para trabajar. Actualmente se están desarrollando robots bípedos capaces de guardar el equilibrio correctamente.

En cuanto a la evolución de la robótica móvil cabe citar que el primer robot móvil de la historia, pese a sus muy limitadas capacidades, fue ELSIE (Electro-Light-Sensitive Internal-External), construido en Inglaterra en 1953. ELSIE se limitaba a seguir una fuente de luz utilizando un sistema mecánico realimentado sin incorporar inteligencia adicional.

En 1968, apareció SHAKEY del SRI (Standford Research Institute), que estaba provisto de una diversidad de sensores así como una cámara de visión y sensores táctiles y podía desplazarse por el suelo. El proceso se llevaba en dos computadores conectados por radio, uno a bordo encargado de controlar los motores y otro remoto para el procesamiento de imágenes.

En los años setenta, la NASA inició un programa de cooperación con el Jet Propulsion Laboratory para desarrollar plataformas capaces de explorar terrenos hostiles. El primer fruto de esta alianza sería el MARS-ROVER, que estaba equipado con un brazo mecánico tipo STANFORD, un dispositivo telemétrico láser, cámaras estéreo y sensores de proximidad.

En los ochenta aparece el CART del SRI que trabaja con procesado de imagen estéreo, más una cámara adicional acoplada en su parte superior. También en la década de los ochenta, el CMU-ROVER de la Universidad Carnegie Mellon incorporaba por primera vez una rueda timón, lo que permite cualquier posición y orientación del plano.

Figura 2.2.- Robot SHAKEY (1968).

En la actualidad, la robótica se debate entre modelos sumamente ambiciosos, como es el caso del IT, diseñado para expresar emociones, el COG, también conocido como el robot de cuatro sentidos, el famoso SOUJOURNER o el LUNAR ROVER, vehículo de turismo con control remotos, y otros mucho más específicos como el CYPHER, un helicóptero robot de uso militar, el guardia de trafico japonés ANZEN TARO o los robots mascotas de Sony.

En el campo de los robots antropomorfos (androides) se debe mencionar el P3 de Honda que mide 1.60m, pesa 130 Kg y es capaz de subir y bajar escaleras, abrir puertas, pulsar interruptores y empujar vehículos.

Figura 2.3.- Evolución de los robots serie- P de Honda.

En general la historia de la robótica la podemos clasificar en cinco generaciones. Las dos primeras, ya alcanzadas en los ochenta, incluían la gestión de tareas repetitivas con autonomía muy limitada. La tercera generación incluiría visión artificial, en lo cual se ha avanzado mucho en los ochenta y noventa. La cuarta incluye movilidad avanzada en exteriores e interiores y la quinta entraría en el dominio de la inteligencia artificial en lo cual se está trabajando actualmente.

2.3. Clasificación.

A la hora de llevar a cabo la clasificación de los robots, éstos se pueden agrupar atendiendo a distintos criterios. A continuación, se van a exponer algunas de las pautas que se siguen:

a) Según la fuente de energía utilizada por las diferentes partes de un robot para su movimiento:

- **Robots eléctricos:**

Obtienen la energía para su funcionamiento de un generador de energía eléctrica, ya sea continua o alterna.

- **Robots neumáticos**:

Obtienen la energía para su correcto funcionamiento de un generador de aire comprimido, el cual es capaz de mover las diferentes partes móviles del robot.

Figura 2.4.- Robot accionado neumáticamente

- **Robots hidráulicos:**

Son movidos gracias a la energía contenida en un fluido en estado líquido (generalmente aceites).

- **Robots movidos por un motor de combustión:**

Obtienen la energía necesaria para su funcionamiento de un motor de combustión.

b) Según la cronología, tenemos los robots clasificados en:

- 1ª Generación:

Manipuladores. Son sistemas mecánicos multifuncionales con un sencillo sistema de control, bien manual, de secuencia fija o de secuencia variable.

- 2ª Generación:

Robots de aprendizaje. Repiten una secuencia de movimientos que ha sido ejecutada previamente por un operador humano. El modo de hacerlo es a través de un dispositivo mecánico. El operador realiza los movimientos requeridos mientras el robot le sigue y los memoriza.

- 3ª Generación:

Robots con control sensorizado. El controlador es una computadora que ejecuta las órdenes de un programa y las envía al manipulador para que realice los movimientos necesarios.

- 4ª Generación:

Robots inteligentes. Son similares a los anteriores, pero además poseen sensores que envían información a la computadora de control sobre el estado del proceso. Esto permite una toma inteligente de decisiones y el control del proceso en tiempo real.

c) Según su arquitectura, se clasifican en:

- Poliarticulados:

En este grupo están los robots de muy diversa forma y configuración cuya característica común es la de ser básicamente sedentarios (aunque excepcionalmente pueden ser guiados para efectuar desplazamientos limitados) y estar estructurados para mover sus elementos terminales en un determinado espacio de trabajo según uno o más sistemas de coordenadas y con un número limitado de grados de libertad. En este grupo se encuentran los manipuladores, los robots industriales, los robots cartesianos y se emplean cuando es preciso abarcar una zona de trabajo relativamente amplia o alargada, actuar sobre objetos con un plano de simetría vertical o reducir el espacio ocupado en el suelo.

- Móviles:

Son robots con gran capacidad de desplazamiento, basados en carros o plataformas y dotados de un sistema locomotor, pudiendo ser este de tipo rodante, aéreo, acuático, etc.

Siguen su camino por telemando o guiándose por la información recibida de su entorno a través de sus sensores. Estos robots aseguran el transporte de piezas de un punto a otro de una cadena de fabricación. Guiados mediante pistas materializadas a través de la radiación electromagnética de circuitos empotrados en el suelo, o a través de bandas detectadas fotoeléctricamente, pueden incluso llegar a sortear obstáculos y están dotados de un nivel relativamente elevado de inteligencia.

- Androides:

Son robots que intentan reproducir total o parcialmente la forma y el comportamiento cinemática del ser humano. Actualmente los androides son todavía dispositivos muy poco evolucionados y sin utilidad práctica, y destinados, fundamentalmente, al estudio y experimentación. Uno de los aspectos más complejos de estos robots, y sobre el que se centra la mayoría de los trabajos, es el de la locomoción bípeda. En este caso, el principal problema es controlar dinámica y coordinadamente en el tiempo real el proceso y mantener simultáneamente el equilibrio del robot.

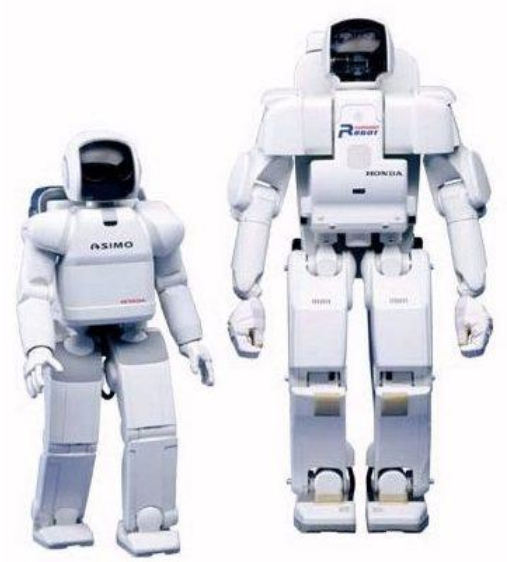


Figura 2.5.- Robots androides.

- Zoomórficos:

Los robots zoomórficos, que considerados en sentido no restrictivo podrían incluir también a los androides, constituyen una clase caracterizada principalmente por sus sistemas de locomoción que imitan a los diversos seres vivos. A pesar de la disparidad morfológica de sus posibles sistemas de locomoción es conveniente agrupar a los robots zoomórficos en dos categorías principales: caminadores y no caminadores. El grupo de los robots zoomórficos no caminadores está muy poco evolucionado. Los experimentados efectuados en Japón basados en segmentos cilíndricos biselados acoplados axialmente entre sí y dotados de un movimiento relativo de rotación. Los robots zoomórficos caminadores multípedos son muy numeroso y están siendo experimentados en diversos laboratorios con vistas al desarrollo posterior de verdaderos vehículos terrenos, piloteando o autónomos, capaces de evolucionar

en superficies muy accidentadas. Las aplicaciones de estos robots serán interesantes en el campo de la exploración espacial y en el estudio de los volcanes.

- Híbridos:

Corresponden a aquellos de difícil clasificación cuya estructura se sitúa en combinación con alguna de las anteriores ya expuestas, bien sea por conjunción o por yuxtaposición. Por ejemplo, un dispositivo segmentado articulado y con ruedas, es al mismo tiempo uno de los atributos de los robots móviles y de los robots zoomórficos.

d) Según el modo en que realizan sus movimientos, tenemos:

- De control en lazo abierto:

El programa que controla el movimiento de los diferentes componentes del robot se realiza en un posicionamiento punto a punto en el espacio.

- De control en lazo cerrado:

Este tipo de control permite dos formas de trabajo diferentes:

1. Gobierno por configuración, donde el control de los movimientos de los elementos del robot se realiza en función de sus ejes. Los desplazamientos pueden realizarse punto a punto o con trayectoria continua.
2. Gobierno por sensor: Los movimientos se establecen en función de la respuesta que van proporcionando uno o varios sensores. Estos pueden ser de diferentes tipos como por ejemplo, visión o distancia .

Además de estos, existen otros muchos criterios de clasificación, pero su explicación resulta prescindible en esta memoria.

3. El brazo modular.

3.1. Introducción.

El brazo modular de Robotnik es un brazo robot formado a partir de los servoaccionamientos modulares *PowerCube* de Schunk. Estos accionamientos integran motorreductor, etapa de potencia y controlador, de forma que el brazo resultante no requiere de un armario de control externo. En su lugar, la conexión externa del brazo se reduce a la alimentación a 24 VDC y la de comunicación a través de CAN bus.

Los módulos trabajan como controladores distribuidos. El controlador maestro (PC de control) es el encargado de generar la secuencia de programa y el envío de referencias a cada uno de los ejes del sistema de articulaciones.

El control de corriente, velocidad y de posición, tiene lugar en cada uno de los módulos, así como las operaciones de supervisión de temperatura y control de parada. También existe la posibilidad de cerrar los lazos de control (corriente, velocidad y posición) en el controlador externo.

Este tipo de brazo, que integra la etapa de potencia y control, es el más adecuado para su instalación en plataformas móviles o androides. Entre las ventajas que ofrece este tipo de brazos, destacamos:

- Alimentación a 24 VDC. No requiere cargar con un pesado inversor.
- Control a través de bus CAN. A diferencia de un brazo robot industrial, solamente se requiere un ordenador con una tarjeta de comunicación CAN, en lugar de armario de control.

3.2. Hardware.

3.2.1. Elementos mecánicos del robot.

Físicamente, la estructura del robot está formada por seis módulos o elementos con motorreductor y controlador, además de las conexiones y el cableado correspondiente. Estos elementos constituyen los seis eslabones que conforman la cadena cinemática del brazo modular:

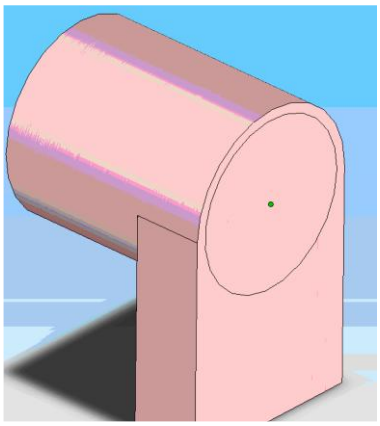


Figura3.1.-Elemento 1.

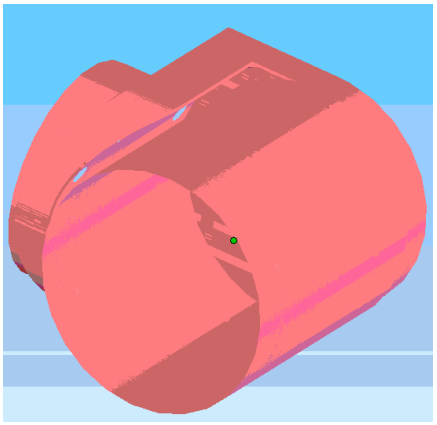


Figura 3.2.- Elemento 2.

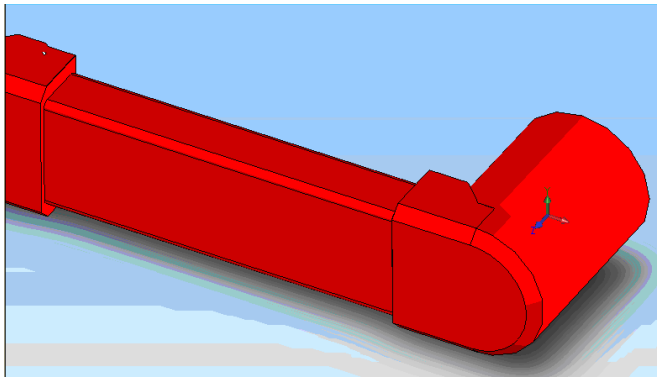


Figura 3.3.- Elemento 3.

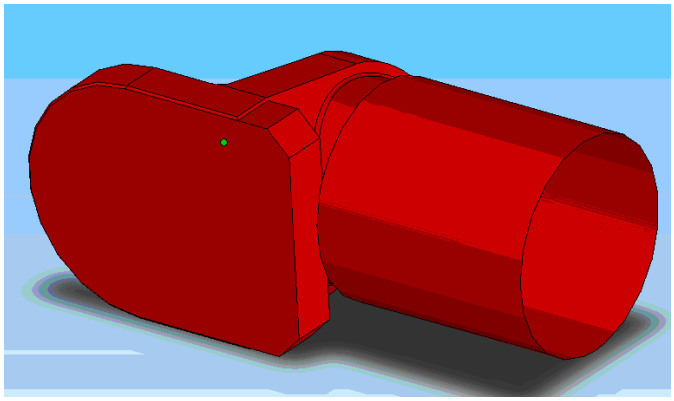


Figura 3.4.- Elemento 4.

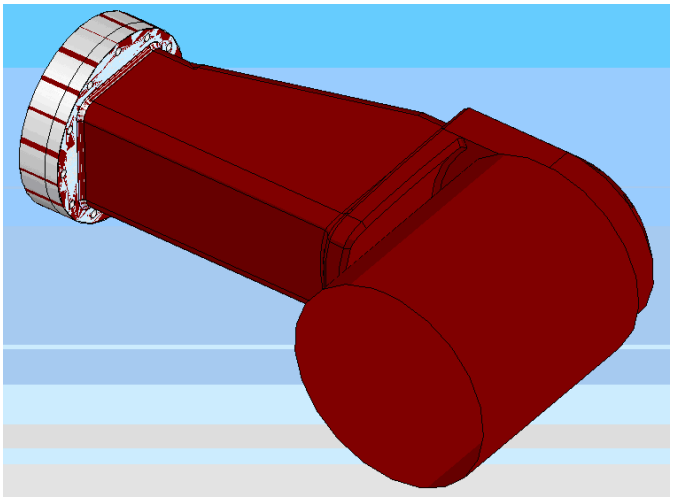


Figura 3.5.- Elemento 5.

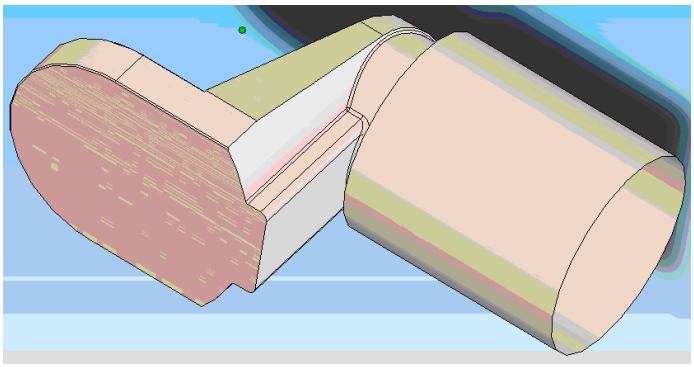


Figura 3.6.- Elemento 6.

Cada uno de los módulos posee las siguientes características:

- Cada eje contiene su propio controlador y servo amplificador.
- Motores sin escobillas.
- Eje pasante que permite el paso interior de los cables.
- Encóder absoluto para posicionamiento y control de velocidad.
- Requerimientos de potencia de 20A y 24Vdc (Potencia para todo el brazo de 480W).
- Freno magnético integrado en todos los ejes.
- Arquitectura abierta (control de alto nivel sobre el movimiento de cada eje).
- Construcción en aluminio para minimización del peso.
- Controlador maestro con drivers disponibles para múltiples sistemas operativos.

Además, los elementos y el brazo se ajustan a los requerimientos de cada aplicación, pudiendo elegir el brazo íntegro o rediseñar los elementos de unión con los parámetros de cada aplicación:

Las propiedades mecánicas más importantes del robot son:

- Hasta 7 grados de libertad (en el mismo bus).
- Alcance de 400 a 1300mm.
- Capacidad de carga útil en función del diseño (9kg en máxima extensión)
- Peso reducido: 25Kg con 6gdl.
- Velocidad máxima de 57grados/s en la base y 360grados/s en la muñeca.
- Motores servo de corriente alterna y frenos electromagnéticos.
- Repetitividad posicional de 0.5mm.

3.2.2. Sistema eléctrico.

En la siguiente figura se muestra una visión superficial de los principales componentes del sistema eléctrico del brazo robot:

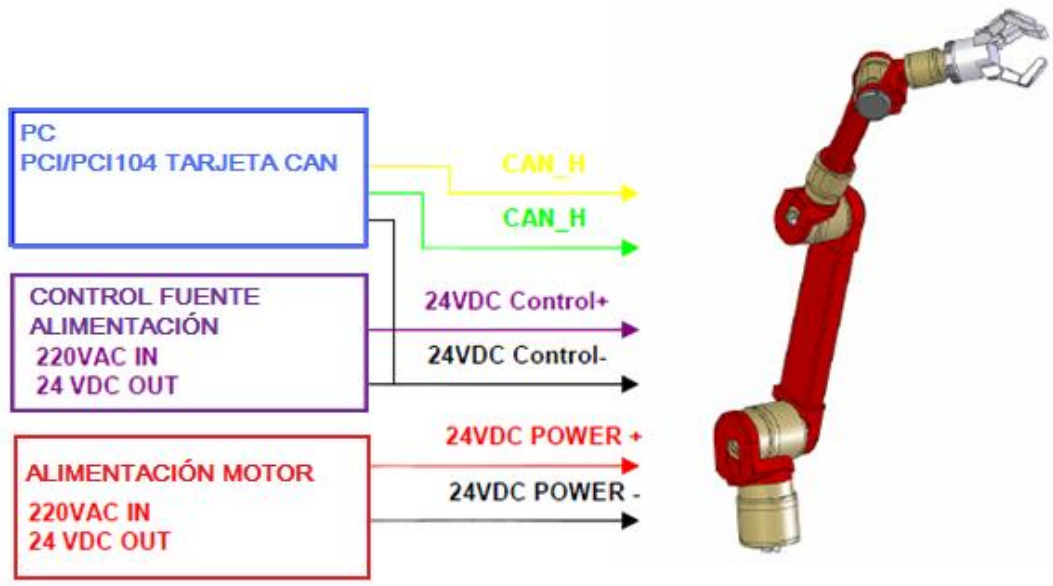


Fig. 3.7. – Principales componentes del sistema eléctrico.

3.2.2.1. Fuente de alimentación principal.

El brazo robot necesita una alimentación de 24 Vdc y 20A. Para evitar posibles ruidos en la alimentación, ésta puede estar repartida en dos suministradores independientes, uno para el

control y otro para la alimentación del motor. El BUS CAN de masa debe estar conectado junto la masa de control.

El brazo robot puede estar alimentado por baterías. Para cargarlas, es recomendable desconectarlas del robot y después conectarlas al cargador. Nunca conectar el cargador al robot, ya que el equipo electrónico puede sufrir daños.

Si el robot no va a ser usado durante un largo período de tiempo, conviene desconectarlo.

3.2.2.2. Secuencia de inicio.

Para iniciar el brazo robot siempre se seguirán los siguientes pasos:

1. Conectar la fuente de alimentación.
2. Asegurarse de que el botón de parada de emergencia no está pulsado.
3. Apretar el botón de reinicio.
4. En este momento, se deberá encender el PC controlador. El PC se se pone en marcha y LinuxRT OS se inicia. El sistema está configurado para comenzar el control de los componentes de forma automática.
5. Durante el inicio, el sistema operativo intentará obtener la dirección IP desde el router por DHCP (Protocolo de configuración de host dinámico). Si el router no está conectado, el sistema está configurado para reintentarlo, de modo que el inicio se retrasará más de un minuto.

No existe la posibilidad de conectarse al sistema de forma remota o local y empezar a trabajar con él.

3.2.2.3. Parada de emergencia.

El robot modular consta de un botón para paradas de emergencia que corta la alimentación de todos los módulos para evitar daños en los actuadores o en el entorno del robot.

El botón de parada de emergencia también debe ser accionado cuando se va a realizar cualquier tipo de trabajo con el robot, como cambiar el actuador del final del brazo, agregar nuevos componentes, en resumen, cualquier tarea durante la cual un movimiento inesperado del robot pueda resultar peligroso.

Para salir del estado de parada de emergencia, se deberán seguir los siguientes pasos:

1. Comprobar que la situación de peligro que causó la parada ha finalizado.

2. Tirar del botón de parada de emergencia.
3. Pulsar el botón Reinicio del panel de control.

3.3. Software.

3.3.1. Arquitectura del sistema.

3.3.1.1. Arquitectura JAUS.

La arquitectura de control del Brazo Robot Modular está basado en la arquitectura JAUS, más concretamente en la versión de código abierto (OpenJAUS). La arquitectura JAUS es la componente base e implementa la comunicación y sincronización de los componentes en una compleja red de trabajo.

Este tipo de arquitectura tiene las siguientes ventajas:

- Reduce el tiempo de desarrollo e integración del software.
- Permite actualizar el sistema con nuevas capacidades.
- Interoperabilidad.

El brazo modular está controlado por las siguientes componentes:

a) Robot: integrado por los componentes responsables de la coordinación de las actividades dentro del sistema robot.
b) Manipulador Primitivo: componente a cargo de controlar el movimiento y control del brazo.

La comunicación entre los componentes del brazo modular se lleva a cabo a través de una memoria compartida, cada componente es ejemplo de una clase que se ejecuta en tiempo real con una frecuencia determinada.

3.3.1.2.Comunicación entre componentes.

La comunicación entre componentes es realizada mediante JAUS a través de un sistema de intercambio de mensajes. Los mensajes vienen definidos de forma previa, pero el usuario

tiene la posibilidad de definir sus propios mensajes para transmisiones específicas del sistema.

Existen dos métodos de envío de mensajes, el método estándar y el basado en las conexiones del servicio.

Para la transmisión en modo estándar, la dirección de los componentes involucrados es dada y el mensaje se envía a través del canal de comunicación pertinente. El segundo caso es una prestación más de la arquitectura JAUS diseñada para la transmisión periódica de mensajes.

3.3.2. Topología de redes.

Existen dos tipos de redes utilizadas: CAN bus y Ethernet.

3.3.2.1. Red Ethernet.

El robot modular puede trabajar independiente con un único PC. Para aplicaciones remotas o altamente distribuidas con varios equipos, la arquitectura JAUS soporta la comunicación a través de la red Ethernet. Las redes pueden operar de forma cableada o de forma inalámbrica.

En este tipo de redes, la arquitectura JAUS se convierte en pieza fundamental.

3.3.2.2.Bus CAN.

El brazo modular tiene una red CAN para interconectar los módulos del brazo con el PC de control. Los buses son inicialmente asignados como se observa en la siguiente tabla:

CAN 1	CAN 0
Brazo derecho – Módulo Power Cube 1	Brazo izquierdo – Módulo Power Cube 1
Brazo derecho – Módulo Power Cube 2	Brazo izquierdo – Módulo Power Cube 2
Brazo derecho – Módulo Power Cube 3	Brazo izquierdo – Módulo Power Cube 3
Brazo derecho – Módulo Power Cube 4	Brazo izquierdo – Módulo Power Cube 4
Brazo derecho – Módulo Power Cube 5	Brazo izquierdo – Módulo Power Cube 5

Brazo derecho – Módulo Power Cube 6	Brazo izquierdo – Módulo Power Cube 6
{Opcional Servo 2 Pinza}	{Opcional Servo 2 Pinza}

Tabla 3.1 – Configuración Red CAN

Los detalles sobre el funcionamiento del Bus CAN que incorpora el robot se pueden encontrar en el *Anexo A.*

3.3.3. Programación del sistema.

El software de control del brazo robot modular ha sido diseñado utilizando C++ como lenguaje de programación y el compilador gnu.

3.3.3.1. Estructura de archivos.

Los ficheros fuentes del software del robot están en la carpeta /opt/MODULAR. Dentro de este directorio principal, se encuentran las siguientes subcarpetas:

- /bin carpeta de ejecutables
- /config archivos de configuración de componentes
- /Build archivos objeto
- /docs código fuente
- /include encabezamiento de archivos
- /lib librerías
- /src carpetas de archivos con código fuente (*.cpp, *.c)

3.3.3.2. Librería m5api.

Para mover el brazo modular como una cadena cinemática, sus motores deben ser movidos de forma independiente. Para ello, se utiliza la librería m5api, administrada por Schunk, que puede ser compilada bajo diferentes sistemas operativos:

Sistema operativo	Forma	Compiladores Soportados
MS Windows 9x/NT/2000/XP	Librería de enlace dinámico (DLL)	Visual C/C++ Visual Basic National Instruments LAbWindows CVI
SuSe Linux 6.4	Código Abierto	GNU C/C++
QNX 4.25	Código Abierto	Watcom C/C++ v.10

Tabla 3.2 – Sistemas operativos soportados

3.3.3.2.1. Funciones definidas dentro de la librería m5api.

Las funciones que se encuentran definidas dentro de la librería aparecen listadas y ordenadas según su ámbito de funcionalidad en las siguientes tablas:

Abrir y cerrar la interfaz de comunicación	Función	Descripción
	Pcube_openDevice	Abre la interfaz. Genera una ID de mecanismo válida.
	Pcube_closeDevice	Cierra la interfaz de comunicaciones.

Funciones administrativas	Función	Descripción
	Pcube_getModuleIdMap	Recupera el número de módulos PowerCube encontrados en el bus. Al mismo tiempo visualiza la dirección física de los módulos en ID lógicas, las cuales se guardarán en orden ascendente en el array correspondiente.
	Pcube_updateModuleMap	Actualiza el mapeo.
	Pcube_getModuleCount	Recupera el número de módulos conectados al bus.
	Pcube_getModuleType	Recupera el tipo de módulo especificando dos ID, una asociada al mecanismo y otra al módulo. Mecanismo de rotación: TYPE_MOD_ROTARY=0X0F Mecanismo lineal: TYPE_MOD_LINEAR=0XF0
	Pcube_getModuleVersion	Recupera la versión del sistema operativo del módulo. El resultado es un número hexadecimal.
	Pcube_getModuleSerialNo	Recupera el número de serie de un módulo especificando dos ID, una asociada al mecanismo y otra a su módulo.
	Pcube_getDllVersion	Recupera la versión de la DLL que se está ejecutando.
	Pcube_configFromFile	Configura el sistema completo en un archivo Ini. El nombre del archivo es un parámetro de llamada.
	Pcube_serveWatchdogAll	Refresca el perro guardián en todos los módulos conectados si éste está activado. Sólo es válido para CAN.

	Pcube_getDefSetup	Recupera el funcionamiento por defecto del módulo especificando las ID asociadas al mecanismo y al módulo. El resultado es una palabra de instalación, que más adelante se detallará.
	Pcube_getDefBaudRate	Recupera el valor por defecto de la velocidad de transmisión del módulo. Sólo está disponible para CAN y redes RS232. El resultado es un valor entre 0 y 5: CAN: 0=50 1=250 2=500 4=1000 kbit/seg RS232: 0=1200 1=2400 2=4800 4=9600 5=38400 bit/seg
	Pcube_setBaudRateAll	Recupera la disposición por defecto del módulo especificando las ID asociadas al mecanismo y al módulo. El resultado es una palabra de instalación (setup) que más adelante se explicará con detalle.
	Pcube_getDefGearRatio	Recupera el valor de la relación de transmisión por defecto.
	Pcube_getDefLinearRatio	Recupera el factor de conversión de un movimiento rotatorio a lineal.
	Pcube_getDefCurRatio	Recupera el factor por defecto para la conversión de corriente en dígitos a Amperios.
	Pcube_getDefBreakTimeOut	Recupera el retraso por defecto entre el final del movimiento y el descanso.
	Pcube_getDefIncPerTurn	Recupera el valor por defecto para el número de incrementos por rotación del motor.

Recuperación de Posición	Función	Descripción
	Pcube_getPos	Recupera la posición actual del módulo especificando las ID asociadas al mecanismo y módulo. El resultado es la posición en radianes (módulo giratorio) o en metros (módulo lineal).
	Pcube_getPosInc	Recupera la posición actual del módulo especificando las ID asociadas al mecanismo y módulo. El resultado es la posición en incrementos.
	Pcube_getPosCountInc	Recupera el valor actual del contador especificando las ID asociadas al mecanismo y módulo. El resultado es el valor del contador en incrementos (posición sin offset)

Recuperación de Velocidad	Función	Descripción
	Pcube_getVel	Recupera la velocidad actual especificando las Id asociadas al mecanismo y al módulo. El resultado es la velocidad real en rad/s (rotación) o en m/s (módulos lineales)
	Pcube_getVelInc	Recupera la velocidad especificando las Id asociadas al mecanismo y al módulo. El resultado es la velocidad real en incrementos/s.
	Pcube_getIPolVel	Recupera la velocidad interpolada especificando las Id asociadas al mecanismo y al módulo. El resultado es la velocidad interpolada en rad/s (rotación) o m/s (lineal).

Recuperación de corriente	Función	Descripción
	Pcube_getCur	Recupera la información actual acerca de la corriente especificando las ID asociadas al mecanismo y al módulo. El resultado es la corriente actual en A.
	Pcbe_getCurInc	Recupera la información actual acerca de la corriente especificando las ID asociadas al mecanismo y al módulo.

		El resultado es la corriente en dígitos.

Recuperación de la distancia de remolque	Función	Descripción
	Pcube_getDeltaPos	Recupera la distancia actual de remolque especificando las ID asociadas al mecanismo y al módulo. El resultado es la distancia actual en radianes o metros.
	Pcube_getDeltaPosInc	Recupera la distancia actual de remolque especificando las ID asociadas al mecanismo y al módulo. El resultado es la distancia actual en incrementos.

Recuperación de estado del módulo	Función	Descripción
	Pcube_getModuleState	Recupera el estado actual del módulo especificando las Id asociadas al mecanismo y al módulo. El resultado es la palabra de estado de módulo, que más adelante se explicará con detalle.
	Pcube_getStateDioPos	Recupera una combinación de información acerca del estado de módulo, posición y estado digital I/O. El resultado es el estado del módulo, la actual posición en radianes o metros y el estado de las I/Os digitales.

Recuperación de posición síncrona	Función	Descripción
	Pcube_savePosAll	Este comando fuerza a todos los módulos conectados a guardar su posición actual al mismo tiempo. La ID de mecanismo es un parámetro necesario. Sólo está disponible para CAN.
	Pcube_getSavePos	Recupera el valor de la posición guardado durante la llamada a la función Pcube_getPosAll especificando las ID asociadas al mecanismo y al módulo. El resultado es el valor guardado en radianes o metros. Sólo disponible para CAN.

Configuración	Función	Descripción
	Pcube_getDefConfig	Recupera la configuración por defecto del módulo especificando las ID asociadas al mecanismo y al módulo. El resultado es la palabra de configuración, que más adelante se explicará con detalle.
	Pcube_getConfig	Recupera la configuración actual del módulo especificando las ID asociadas al mecanismo y al módulo. El resultado es la palabra de configuración que fue guardada con la última llamada de Pcube_setConfig. Después de alimentar este valor coincide con el valor por defecto.
	Pcube_setConfig	Establece la configuración actual del módulo, especificando las ID asociadas al mecanismo y al módulo y una nueva palabra de configuración.

E/S Digitales	Función	Descripción
	Pcube_getDefDioData	Recupera el estado digital de las entradas/salidas digitales especificando las ID asociadas al mecanismo y al módulo. El resultado es el estado de la entrada/salida.
	Pcube_getDioData	Recupera el estado actual de la entrada/salida especificando las ID asociadas al mecanismo y al módulo. El resultado es el estado acutal.
	Pcube_setDioData	Establece el estado actual de la entrada/salida especificando las ID asociadas al mecanismo, al módulo y una palabra de estado válida de entrada/salida.

Coeficientes del lazo de PID	Función	Descripción
	Pcube_getDefA0	Recupera el valor por defecto del coeficiente A0 del lazo PID.
	Pcube_getA0	Recupera el valor actual del coeficiente A0 del lazo PID. Después de alimentar, este valor coincide con el valor por defecto.
	Pcube_setA0	Establece el valor actual del coeficiente A0 del lazo PID (rango 1…12).
	Pcube_getDefC0	Recupera el valor por defecto del coeficiente C0 del lazo PID.
	Pcube_getC0	Recupera el valor actual del coeficiente C0 del lazo PID. Después de alimentar, este valor coincide con el valor por defecto.
	Pcube_setC0	Establece el valor actual del coeficiente C0 del lazo PID (rango 1…12).
	Pcube_getDefDamp	Recupera el valor por defecto del coeficiente de amortiguamiento del lazo PID.
	Pcube_getDamp	Recupera el valor actual del coeficiente de amortiguamiento del lazo PID. Después de alimentar, este valor coincide con el valor por defecto.
	Pcube_setDamp	Establece el valor actual del coeficiente de amortiguamiento del lazo PID (rango 1…4).
	Pcube_recalcPIDParams	Actualiza el lazo PID y crea nuevos coeficientes válidos. Esta función debe ser llamada después de que los parámetros A0, C0 o Coeficiente de Amortiguamiento hayan sido modificados.

Offset de posición	Función	Descripción
	Pcube_getDefHomeOffset	Recupera el offset por defecto al inicio especificando las ID asociadas al mecanismo y al módulo. El resultado es el offset por defecto al inicio en radianes o metros. El offset de inicio es el valor de la posición en la posición de inicio.
	Pcube_getHomeOffset	Recupera el offset actual de inicio especificando las ID asociadas al mecanismo y al módulo. Después de alimentar, este valor coincide con el valor por defecto.
	Pcube_getHomeOffsetInc	Recupera el valor actual de offset de inicio especificando las ID asociadas al mecanismo y al módulo. El resultado es el valor actual de offset en incrementos
	Pcube_setHomeOffset	Establece el offset actual de inicio especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en radianes o metros.
	Pcube_setHomeOffsetInc	Establece el offset actual de inicio especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en incrementos.

Velocidad de búsqueda de blanco	Función	Descripción
	Pcube_getDefHomeVel	Recupera el valor por defecto de la velocidad especificando las ID asociadas al mecanismo y al módulo. El resultado es la velocidad de búsqueda por defecto en rad/s o m/s. Esta velocidad es usada durante la operación de aproximación al blanco.
	Pcube_getHomeVel	Recupera el valor actual de la velocidad búsqueda especificando las ID asociadas al mecanismo y al módulo. El resultado es la velocidad en rad/s o m/s. Después de la alimentación del sistema, este valor coincide con el valor por defecto.
	Pcube_getHomeVelInc	Recupera el calor actual de la velocidad de búsqueda especificando las ID asociadas al mecanismo y al módulo. El resultado es la velocidad en Incrementos/s.
	Pcube_setHomeVel	Establece la velocidad de aproximación al blanco actual especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en rad/s o m/s.
	Pcube_setHomeVelInc	Establece la velocidad de aproximación al blanco actual especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en incrementos/s.

Rango de operación: Posición mínima.	Función	Descripción
	Pcube_getDefMinPos	Recupera el valor de posición mínima por defecto especificando las ID asociadas al mecanismo y al módulo. El resultado es la posición mínima en radianes o metros. Este parámetro es usado como límite para el rango de operación. Valores menores que éste serán limitados por el mínimo establecido.
	Pcube_getMinPos	Recupera el valor actual de posición mínima especificando las ID asociadas al mecanismo y al módulo. El resultado será la posición mínima en radianes o metros. Después de alimentar el sistema, este valor coincide con el valor por defecto.
	Pcube_getMinPosInc	Recupera el valor actual de posición mínima especificando las ID asociadas al mecanismo y al módulo. El resultado es la posición mínima en incrementos.
	Pcube_setMinPos	Establece el valor actual de posición mínima especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en radianes o metros.
	Pcube_setMinPosInc	Establece el valor actual de posición mínima especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en incrementos.

Rango de operación: Posición máxima.	Función	Descripción
	Pcube_getDefMaxPos	Recupera el valor de posición máxima por defecto especificando las ID asociadas al mecanismo y al módulo. El resultado es la posición máxima en radianes o metros. Este parámetro es usado como límite para el rango de

		operación. Valores mayores que éste serán limitados por el máximo establecido.
	Pcube_getMaxPos	Recupera el valor actual de posición máxima especificando las ID asociadas al mecanismo y al módulo. El resultado será la posición máxima en radianes o metros. Después de alimentar el sistema, este valor coincide con el valor por defecto.
	Pcube_getMaxPosInc	Recupera el valor actual de posición máxima especificando las ID asociadas al mecanismo y al módulo. El resultado es la posición máxima en incrementos.
	Pcube_setMaxPos	Establece el valor actual de posición máxima especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en radianes o metros.
	Pcube_setMaxPosInc	Establece el valor actual de posición máxima especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en incrementos.

Máxima velocidad	Función	Descripción
	Pcube_getDefMaxVel	Recupera el valor de velocidad máxima por defecto especificando las ID asociadas al mecanismo y al módulo. El resultado es la velocidad máxima en rad/s o m/s. Este parámetro es usado como límite. Valores mayores que éste serán limitados por el máximo establecido.
	Pcube_getMaxVel	Recupera el valor actual de velocidad máxima especificando las ID asociadas al mecanismo y al módulo. El resultado será la velocidad máxima en rad/s o m/s. Después de alimentar el sistema, este valor coincide con el valor por defecto.
	Pcube_getMaxVelInc	Recupera el valor actual de velocidad máxima especificando las ID asociadas al mecanismo y al módulo. El resultado es la velocidad máxima en incrementos/s.
	Pcube_setMaxVel	Establece el valor actual de velocidad máxima especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en rad/s o m/s.
	Pcube_setMaxVelInc	Establece el valor actual de velocidad máxima especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en incrementos/s.

Máxima aceleración	Función	Descripción
	Pcube_getDefMaxAcc	Recupera el valor de aceleración máxima por defecto especificando las ID asociadas al mecanismo y al módulo. El resultado es la velocidad máxima en rad^2/s o m^2/s. Este parámetro es usado como límite. Valores mayores que éste serán limitados por el máximo establecido.
	Pcube_getMaxAcc	Recupera el valor actual de aceleración máxima especificando las ID asociadas al mecanismo y al módulo. El resultado será la aceleración máxima en rad^2/s o m^2/s. Después de alimentar el sistema, este valor coincide con el valor por defecto.
	Pcube_getMaxAccInc	Recupera el valor actual de aceleración máxima especificando las ID asociadas al mecanismo y al módulo. El resultado es la aceleración máxima en incrementos/s^2.
	Pcube_setMaxAcc	Establece el valor actual de aceleración máxima especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en rad^2/s o m^2/s.
	Pcube_setMaxAccInc	Establece el valor actual de aceleración máxima especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en incrementos/s^2.

Máxima corriente	Función	Descripción
	Pcube_getDefMaxCur	Recupera el valor de corriente máxima por defecto especificando las ID asociadas al mecanismo y al módulo. El resultado es la corriente máxima en Amperios. Este parámetro es usado como límite para la corriente máxima de motor utilizada durante la operación.
	Pcube_getMaxCur	Recupera el valor actual de corriente máxima especificando las ID asociadas al mecanismo y al módulo. El resultado será la corriente máxima en Amperios. Después de alimentar el sistema, este valor coincide con el valor por defecto.
	Pcube_setMaxCur	Establece el valor actual de aceleración máxima especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en Amperios.

Movimiento de rampa con especificación de posición	Función	Descripción
	Pcube_movePos	Comienza un movimiento de rampa del módulo especificando las ID asociadas al mecanismo y al módulo. La posición tope viene dada en radianes o metros.
	Pcube_movePosExtended	Comienza un movimiento de rampa del módulo especificando las ID asociadas al mecanismo y al módulo. La posición tope viene dada en radianes o metros. Los resultados de esta función son el estado, la posición actual y el estado digital de las E/S.

Movimiento de rampa con especificación de posición, velocidad y aceleración	Función	Descripción
	Pcube_moveRamp	Comienza un movimiento de rampa del módulo especificando las ID asociadas al mecanismo y al módulo. La posición tope viene dada en radianes o metros, la velocidad en rad/s o m/s, y la aceleración en rad^2/s o m^2/s.
	Pcube_moveRampInc	Comienza un movimiento de rampa del módulo especificando las ID asociadas al mecanismo y al módulo. La posición viene dada en incrementos, la velocidad en incrementos/s y la aceleración en incrementos $/s^2$.
	Pcube_moveRampExtended	Comienza un movimiento de rampa del módulo especificando las ID asociadas al mecanismo y al módulo. La posición tope viene dada en radianes o metros, la velocidad en rad/s o m/s, y la aceleración en rad^2/s o m^2/s. El resultado de esta función es el estado, la posición actual y el estado digital de E/S.

Movimiento de velocidad constante	Función	Descripción
	Pcube_moveVel	Comienza un movimiento de velocidad constante. La velocidad de objetivo es especificada en rad/s o m/s.
	Pcube_moveVelInc	Comienza un movimiento de velocidad constante. La

		velocidad de objetivo es especificada en incrementos/s.
	Pcube_moveVelExtended	Comienza un movimiento de velocidad constante. La velocidad de objetivo es especificada en rad/s o m/s. Los resultados son estado, actual posición y estado digital de E/S.

Movimiento de intensidad constante	Función	Descripción
	Pcube_moveCur	Comienza un movimiento de intensidad constante. La corriente es especificada en Amperios.
	Pcube_moveCurInc	Comienza un movimiento de intensidad constante. La corriente se especifica en Dígitos.
	Pcube_moveCurExtended	Comienza un movimiento de intensidad constante. La corriente es especificada en Amperios. Los resultados son estado, actual posición y estado digital de E/S.

Movimiento con especificación de posición y tiempo	Función	Descripción
	Pcube_moveStep	Comienza un movimiento hacia la posición fijada especificada en radianes o metros. El tiempo para el recorrido es especificado en ms.
	Pcube_moveStepInc	Comienza un movimiento hacia la posición fijada especificada en incrementos. El tiempo para el recorrido es especificado en ms.
	Pcube_moveStepExtended	Comienza un movimiento hacia la posición fijada especificada en radianes o metros. El tiempo para el recorrido es especificado en ms. El resultado de esta función es el estado, la posición actual y el estado digital de E/S.

Distancia máxima de remolque	Función	Descripción
	Pcube_getDefMaxDeltaPos	Recupera el valor de distancia máxima de remolque por defecto especificando las ID asociadas al mecanismo y al módulo. El resultado es la distancia máxima de remolque en radianes o metros. Este parámetro es usado como límite para la distancia máxima de remolque. Si se supera este valor, se generará un error y los motores pararán.
	Pcube_getMaxDeltaPos	Recupera el valor actual de distancia máxima de remolque especificando las ID asociadas al mecanismo y al módulo. El resultado será la distancia máxima en radianes o metros. Después de alimentar el sistema, este valor coincide con el valor por defecto.
	Pcube_getMaxDeltaPosInc	Recupera el valor actual de distancia máxima especificando las ID asociadas al mecanismo y al módulo. El resultado es la distancia máxima en incrementos.
	Pcube_setMaxDeltaPos	Establece el valor actual de distancia máxima de remolque especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en radianes o metros.
	Pcube_setMaxDeltaPosInc	Establece el valor actual de distancia máxima de remolque especificando las ID asociadas al mecanismo y al módulo y el nuevo valor máximo en incrementos.

Aceleración del movimiento de rampa	Función	Descripción
	Pcube_setRampAcc	Establece la aceleración para el movimiento de rampa especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en rad^2/s o m^2/s.
	Pcube_setRampAccInc	Establece la aceleración especificando las ID asociadas al mecanismo y al módulo y el nuevo valor en Incrementos/s^2.

Objetivo	Función	Descripción
	Pcube_homeModule	Comienza una operación de aproximación al objetivo especificando las ID asociadas al mecanismo y al módulo.
	Pcube_homeAll	Comienza una operación de aproximación al objetivo con todos los módulos conectados al bus. Sólo disponible para CAN.

Parada rápida	Función	Descripción
	Pcube_haltModule	Lleva a cabo una rápida detención del módulo especificado a través de las ID asociadas al mecanismo y al módulo.
	Pcube_haltAll	Lleva a cabo una parada rápida de todos los módulos conectados al bus. Sólo disponible para el bus CAN.

Reseteo del estado del módulo	Función	Descripción
	Pcube_resetModule	Lleva a cabo el reseteo de un módulo determinado especificando las ID asociadas al mecanismo y al módulo. Un reseteo puede limpiar las señales de error en el estado del módulo. Si un error es permanente, el reseteo es ignorado.
	Pcube_resetAll	Lleva a cabo el reseteo de todos los módulos conectados al bus. Sólo disponible para el bus CAN.

Movimiento de rampa con especificación de posición fijada	Función	Descripción
	Pcube_movePos	Comienza un movimiento de rampa del módulo especificando las ID asociadas al mecanismo y al módulo. La posición fijada viene dada en radianes o metros. Antes de esto, se llama a las funciones que determinan la velocidad y la aceleración de aproximación al objetivo a través de las funciones correspondientes.
	Pcube_movePosInc	Comienza un movimiento de rampa del módulo especificando las ID asociadas al mecanismo y al módulo. La posición viene dada en incrementos.

Datos del sensor de fuerza de torsión SCHUNK	Función	Descripción
	Pcube_getDataSCHUNK_FTC	Recupera los datos de fuerza y torsión del sensor. El

		resultado son tres valores de fuerza (x, y, z) y tres valores de torsión (x, y, z) o tres de translación (x, y, z) y tres valores de rotación (x, y, z) así como el estado del sensor.
	Pcube_setNullSCHUNK_FTC	Anula el sensor. El resultado es el estado del sensor.

Tabla 3.3.- Funciones integradas en la librería m5api.

3.3.3.3.Configuración de los módulos.

Algunas de las funciones anteriormente descritas generarán o necesitarán como parámetros de entrada determinadas "palabras", que son valores predefinidos de forma que tienen un significado u otro para las funciones y que permiten configurar el sistema de la forma que el usuario desee.

3.3.3.3.1. Palabra de configuración.

La palabra de configuración es el resultado de llamar a las funciones Pcube_getDefConfig y Pcube_getConfig y un parámetro de la función Pcube_setConfig. En la tabla que se muestra a continuación, vienen definidos los valores que puede tomar esta palabra y sus posibles significados.

Valor	Definición	Descripción
0x00000008L	CONFIGID_MOD_BRAKE_PRESENT	1 = Actualmente interrumpido.
0x00000010L	CONFIGID_MOD_BRAKE_AT_POWERON	0 = La interrupción se realizará tras la alimentación.
0x00000020L	CONFIGID_MOD_SWR_WITH_ENCODERZERO	1 = Encoder Index used for Homing
0x00000040L	CONFIGID_MOD_SWR_AT_FALLING_EDGE	1 = Homing finishes on falling edge of homing switch
0x00000080L	CONFIGID_MOD_CHANGE_SWR_TO_LIMIT	1 = Homing switch converts to limit switch after Homing is finished
0x00000100L	CONFIGID_MOD_SWR_ENABLED	1= El interruptor de aproximación está habilitado.
0x00000200L	CONFIGID_MOD_SWR_LOW_ACTIVE	1= El interruptor de aproximación es activo a nivel bajo.
0x00000400L	CONFIGID_MOD_SWR_USE_EXTERNAL	1= El interruptor externo de aproximación será utilizado.
0x00000800L	CONFIGID_MOD_SW1_ENABLED	1= El interruptor limitador 1 está habilitado.
0x00001000L	CONFIGID_MOD_SW1_LOW_ACTIVE	1= El interruptor limitador 1 es activo a

		nivel bajo.
0x00002000L	CONFIGID_MOD_SW1_USE_EXTERNAL	1= El interruptor limitador externo será utilizado.
0x00004000L	CONFIGID_MOD_SW2_ENABLED	1= El interruptor limitador 2 está habilitado.
0x00008000L	CONFIGID_MOD_SW2_LOW_ACTIVE	1=El interruptor limitador 2 es activo a nivel bajo.
0x00010000L	CONFIGID_MOD_SW2_USE_EXTERNAL	1= El interruptor limitador externo 2 será utilizado.
0x00020000L	CONFIGID_MOD_LINEAR	1= Módulo de tipo lineal.
0x00080000L	CONFIGID_MOD_ALLOW_FULL_CUR	0= La corriente máxima instaurada mediante la función Pcube_moveCur será limitada a la corriente nominal.
0x00100000L	CONFIGID_MOD_M3_COMPATIBLE	1= El modulo es MoRSE3 compatible.
0x00200000L	CONFIGID_MOD_LINEAR_SCREW	1= El modulo es lineal cuyo actuador es un tornillo sin fin.
0x00800000L	CONFIGID_MOD_DISABLE_ON_HALT	1= Si existe error se deshabilita la corriente.
0x01000000L	CONFIGID_MOD_WATCHDOG_ENABLE	1= El perro guardián está habilitado. El perro guardián comienza después de la recepción de la primera señal de control.
0x02000000L	CONFIGID_MOD_ZERO_MOVE_AFTER_HOK	1= Después de alcanzar el objetivo, el módulo vuelve automáticamente a su posición cero.
0x04000000L	CONFIGID_MOD_DISABLE_ACK	1= Los mensajes no son reconocidos. Los comandos aún son respondidos. Sólo para CAN.
0x08000000L	CONFIGID_MOD_SYNC_MOTION	1= Habilita los 35omandos de movimiento sincronizado.

Tabla 3.4.- Palabra de configuración.

3.3.3.3.2. Palabra de instalación.

La palabra de instalación es el resultado de llamar a la función Pcube_getDefSetup. Al igual que en el caso anterior, a continuación se muestran los diferentes valores y definiciones posibles:

Value	Define	Descripción
0x00000001L	SETUPID_MOD_ENCODER_FEEDBACK	No se utiliza.
0x00000002L	SETUPID_MOD_RESOLVER_FEEDBACK	No se utiliza.
0x00000004L	SETUPID_MOD_ABSOLUTE_FEEDBACK	No se utiliza.
0x00000008L	SETUPID_MOD_4IN_4OUT	1= El conector es configurado para 4 señales de E/S.
0x00000010L	SETUPID_MOD_3IN_ENCODER_IN	1= El conector es configurado como codificador de entrada.
0x00000020L	SETUPID_MOD_3IN_ENCODER_OUT	1= El conector es configurado como codificador de salida.
0x00000040L	SETUPID_MOD_RS232	1= El módulo es configurado para

		comunicación RS232.
0x00000200L	SETUPID_MOD_CAN	1= El módulo es configurado para comunicación CAN.
0x00000400L	SETUPID_MOD_PROFIBUS	1= El modulo es configurado para comunicación vía Profibus.
0x00000800L	SETUPID_MOD_USE_M3ID	1= CAN identifica los módulos MoRSE3 que están activados.
0x00001000L	SETUPID_MOD_USE_M4ID	1= CAN identifica los módulos MoRSE4 que están activados.
0x00002000L	SETUPID_MOD_USE_CANOPEN	1= El modulo es configurado para CAN abierto.
0x00008000L	SETUPID_MOD_USE_SW2_AS_ENABLE	1= La entrada por el interruptor limitador 2 es utilizada como señal de habilitación.
0x00010000L	SETUPID_MOD_USE_SW2_AS_BRAKE	1= La entrada por el interruptor limitador 2 es utilizada para realizar la interrupción.
0x00020000L	SETUPID_MOD_ERROR_TO_OUT0	1= Error señalizado en la salida 0.

Tabla 3.5.- Palabra de instalación.

3.3.3.4. Librería Roboop.

El paquete ROBOOP es una herramienta de programación de robots de C++ orientada a objetos adecuada para la síntesis y simulación de modelos robóticos de manipulación en entornos adecuados a las características de Matlab de tratamiento de matrices. En este paquete viene incluida una clase denominada Robot que proporciona la implementación de la cinemática, dinámica y dinámica linealizada de manipuladores robóticos.

3.3.3.5. Otras configuraciones, programas y ejemplos suministrados.

3.3.3.5.1. M5test (para Linux).

Este programa lleva a cabo un análisis básico de los PowerCubes.

3.3.3.5.2. Test1.lin (para Linux).

Este programa implementa el detector de colisiones Vcollide, que reconoce los posibles choques que se pueden producir entre los distintos eslabones del robot modular.

3.3.3.5.3. Test2.lin (para Linux).

El programa Test1.lin implementa la detección rápida de choques del robot.

3.3.3.5.4. Vcollide201 (Windows).

Este programa implementa la librería Vcollide. Éste incluye una gran cantidad de demos sobre la detección de choques entre diferentes objetos.

3.3.3.5.5. CubeVB (Windows).

Este programa muestra la funcionalidad básica de los PowerCubes y cómo pueden ser programados a través de Visual Basic.

3.3.3.5.6. PowerCube Software.

Se trata de la herramienta más importante para la configuración y análisis de los distintos módulos o PowerCubes. Este software permite, además, que el usuario mueva y configure los módulos a través de unas sencillas ventanas de control.

En nuestro caso, a través del archivo de configuración de los módulos, accedemos a una ventana con diferentes pestañas donde se nos permite manipular las distintas propiedades del robot. La ventana principal consta de cinco pestañas: Identificación, Ajustes del controlador, Ajustes generales, Ajustes entrada/salida y Ajustes eléctricos.

Si ahora las analizamos, comenzando por la primera de ellas, Identificación, observamos que, podemos modificar los parámetros iniciales del robot, es decir, el tipo de módulo del que se trata, sus valores máximos y mínimos distintas propiedades del robot, como por ejemplo, valor máximo de rotación del motor, o el rango de movimiento del módulo (posición).

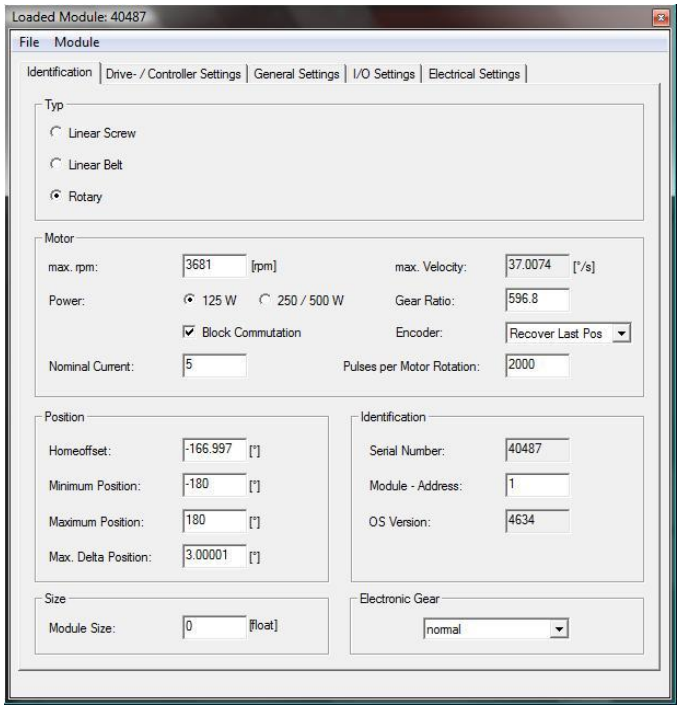


Figura 3.10.- PowerCube Software.

Si cambiamos de pestaña, observamos que podemos elegir también los valores máximos y mínimos de la velocidad y aceleración del módulo desde la pestaña de Ajustes del controlador (Drive/Controller Settings), el modo de comunicación con el robot a través de la pestaña de Ajustes generales (General Settings) o establecer las propiedades eléctricas referidas a tensión o corriente en la pestaña de Ajustes eléctricos (Electrical Settings):

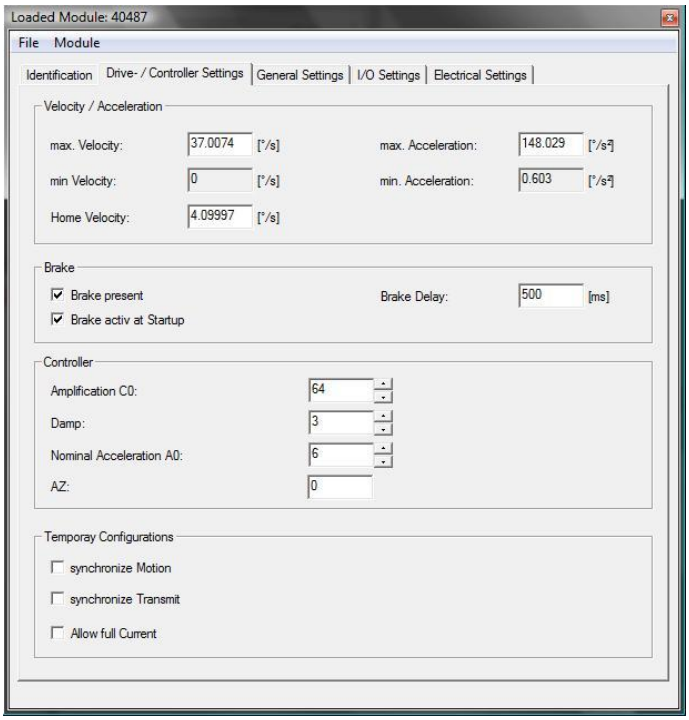


Figura 3.11. – Drive Settings.

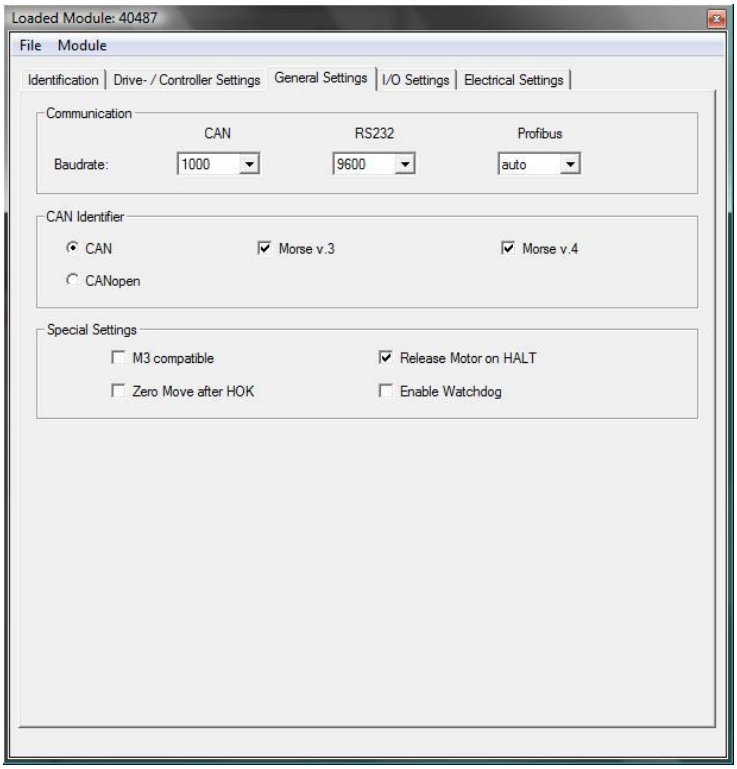


Figura 3.12.- General Settings

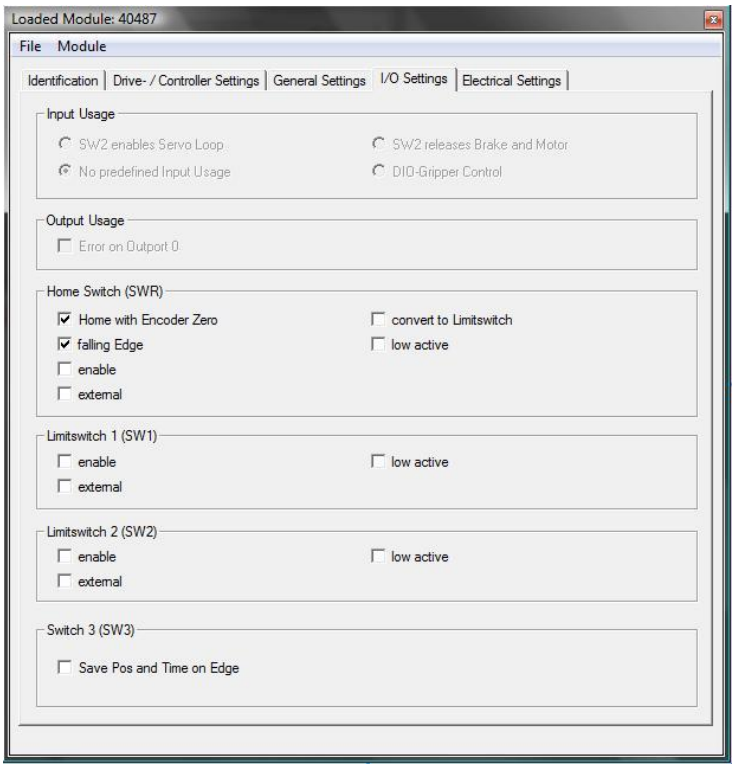


Figura 3.13.- E/O Settings

3.3.4. Control sobre LINUX.

Lo que se describe a continuación es una introducción al sistema operativo LINUX, pues el pc de control del robot trabaja bajo este entorno y resulta oportuno tener unos conocimientos respecto a su procedencia, sus características más importantes o las distribuciones que pueden utilizarse.

3.3.4.1.Introducción.

La mayor parte de los ordenadores que existen en la actualidad están diseñados de forma que puedan ejecutar diversas tareas o programas. Estos programas pueden ir desde un procesador de textos, a un programa para la animación de gráficos tridimensionales o distintos tipos de juegos. Para su correcto funcionamiento deben ser además capaces de acceder a los recursos de que dispone el ordenador, como por ejemplo escribir o leer datos en un disco duro, mostrar un gráfico por pantalla, etc. Es evidente, que si cada programa actuase de una forma independiente, existirían graves problemas y conflictos, puesto que, por ejemplo, tendrían libertad para escribir sus datos sobre los de otro, etc.

Para solucionar este tipo de problemas se desarrollaron los Sistemas Operativos, los cuales aportan unos mecanismos y reglas básicas de funcionamiento, de forma que los programas puedan acceder a los recursos del ordenador de una forma adecuada. Aunque ésta fue la funcionalidad inicial de los sistemas operativos, con el tiempo se han añadido otras muchas, como la ejecución de programas, el control de la memoria del ordenador, la creación y control de interfaces gráficas de usuario, etc.

En la actualidad existen una gran cantidad de sistemas operativos dependiendo del tipo de ordenador en el que se va a ejecutar. Por ejemplo para los PC uno de los sistemas operativos más difundidos en Microsoft Windows. Otros posibles sistemas operativos para este tipo de ordenadores son Solaris, OS/2, BeOS, Microsoft DOS, o uno de los sistemas operativos más poderosos y en rápida expansión para PC, LINUX.

3.3.4.2.¿Qué es LINUX?

Linux es un sistema operativo gratuito y de libre distribución inspirado en el sistema Unix, escrito por Linus Torvalds con la ayuda de miles de programadores en Internet. Unix es un sistema operativo desarrollado en 1970, una de cuyas mayores ventajas es que es fácilmente portable a diferentes tipos de ordenadores, por lo que existen versiones de Unix para casi todos los tipos de ordenadores, desde PC y Mac hasta estaciones de trabajo y superordenadores.

Al contrario que otros sistemas operativos, como por ejemplo MacOS (Sistema operativo de los Apple Macintosh), Unix no está pensado para ser fácil de emplear, sino para ser sumamente flexible. Por lo tanto Linux no es en general tan sencillo de emplear como otros sistemas operativos, aunque, se están realizando grandes esfuerzos para facilitar su uso. Pese a todo la enorme flexibilidad de Linux y su gran estabilidad (y el bajo coste) han hecho de este sistema operativo una opción muy a tener en cuenta por aquellos usuarios que se dediquen a trabajar a través de redes, naveguen por Internet, o se dediquen a la programación. Además el futuro de Linux es brillante y cada vez más y más gente y más y más empresas (entre otras IBM, Intel, Corel) están apoyando este proyecto, con lo que el sistema será cada vez más sencillo de emplear y los programas serán cada vez mejores.

3.3.4.3.Características de LINUX.

Las que se muestran a continuación, son una lista detallada de las principales características que posee LINUX:

a) **32 Bits:** Gracias a los 32 bits, el sistema operativo es rápido eficaz, seguro y fiable, sin que una aplicación pueda causar problemas a las otras, al no tener que guardar compatibilidad con los sistemas operativos anteriores de 16 bits. En la actualidad ya soporta 64 bits.

b) **Multitarea:** El ordenador puede estar haciendo varias cosas a la vez y no tendrás que esperar a que acabe una para hacer otra. La multitarea está controlada por el sistema operativo, no por las aplicaciones, por lo que, a diferencia de otros sistemas operativos, nunca se quedará parado por culpa de una mala aplicación que consuma todos los recursos del ordenador.

c) **Multiusuario:** En sistemas operativos como MAC OS o Windows, el usuario que trabaja con el ordenador es el único que lo usa; sin embargo, en Linux puede haber varias personas

usando el ordenador, compartiendo el microprocesador y estarán en el mismo ordenador. Linux garantiza la privacidad y la seguridad de los datos entre usuarios.

d) **POSIX:** Aunque para los usuarios normales esto importa poco, POSIX es un estándar de la industria que asegura una calidad mínima en ciertas partes del sistema operativo y asegura su compatibilidad a nivel de código, es decir, los programas POSIX que funcionan en otros Unix no tendrán problema para compilarse y ejecutarse en Linux. Para muchas empresas esto es muy importante a la hora de decantarse por un sistema operativo u otro (por eso Windows NT es compatible POSIX).

e) **Compatibilidad:** En Linux debemos tener en cuenta que:

a. **Ficheros:** Linux no tiene ningún problema para leer cualquier tipo de disco y usar su contenido, ya que existen Suites como OpenOffice o Corel WordPerfect que permiten leer y usar ficheros de aplicaciones comunes como pueden ser Word o Excel. Por otra parte, cuando se trabaja en red, Linux es capaz de entenderse y de mediar entre todo tipo de redes, permitiendo entornos heterogéneos sin ningún problema.

b. **Programas:** Se pueden ejecutar programas de otros sistemas operativos: para MAC existe basilisk2, capaz de crear un Macintosh virtual y ejecutar MacOs para M68K sin problemas; para Windows existen varios programas que permiten hacer funcionar programas de Windows, como por ejemplo, Crossoffice para entornos de oficina o WineX para juegos. Si el programa es para MS-Dos existe DosEmu, un emulador de MS-Dos donde podrás ejecutar a pantalla completa, como en la realidad, o en ventana de X Windows, cualquier programa para este sistema operativo. Además de estos, existen vmware (comercial) y bosh que crean PC virtuales donde ejecutar cualquier sistema operativo.

f) **Estabilidad:** Linux es robusto, no se colgará el sistema operativo (si una aplicación está mal hecha por supuesto que se colgará, pero no afectará al resto del sistema, no habrá que reiniciar el ordenador porque un programa lo ha colgado).

g) **Es libre:** Es decir, no costará nada, no hay que pagar licencias, se puede copiar e instalar donde se desee sin problemas, pero lo más importante es que se dispone del código fuente, esto significa que si un día surge un problema del sistema operativo no habrá que esperar que su creador le dé una solución y cree un service pack para el sistema operativo, si no que el propio usuario puede solucionar el problema.

h) **Soporte:** Las empresas que venden cd's de Linux como Mandrake, SUSE, o RedHat ofrecen soporte técnico, pero además, en internet existen diversidad webs y programadores que ofrecen solución a problemas.

i) **Adaptación:** Linux es uno de los sistemas operativos que más rápido evoluciona, se adapta al mercado y soluciona los problemas rápidamente.

j) La mayor dificultad que se le puede atribuir al Sistema Operativo Linux es la **dificultad de configuración**. No es exacto, simplemente no hay botones, se hace todo por ficheros de configuración ASCII. Sin embargo, actualmente las distribuciones incluyen su propio GUI (Interfaz Gráfica para el Usuario) para la configuración del equipo, aunque esta depende de cada distribución.

3.3.4.4.Distribuciones de LINUX.

Como ya se ha comentado, Linux es un sistema operativo de código abierto, con lo que varios grupos de programadores han modificado el sistema y han sacado nuevas versiones de este sistema operativo, estas versiones reciben el nombre de distribuciones.

Figura 5.2.- Distribuciones más importantes de LINUX.

Una distribución Linux es una distribución de software basada en el núcleo Linux que incluye determinados paquetes de software para satisfacer las necesidades de un grupo específico de usuarios, dando así origen a ediciones domésticas, empresariales y para servidores. Por lo general están compuestas, total o mayoritariamente, de software libre, aunque a menudo incorporan aplicaciones o controladores propietarios.

Además del núcleo Linux, las distribuciones incluyen habitualmente las bibliotecas y herramientas del proyecto GNU y el sistema de ventanas Xwindow System. Dependiendo del tipo de usuarios a los que la distribución esté dirigida, se incluye también otro tipo de software como procesadores de texto, hoja de cálculo, reproductores multimedia, herramientas administrativas, etcétera. En el caso de incluir herramientas del proyecto GNU, también se utiliza el término distribución GNU/Linux.

4. Estudio cinemático del robot.

4.1. Cinemática del robot.

Para poder llevar a cabo el estudio de la cinemática del robot, debemos tener conocimiento previo de algunos conceptos que se desarrollan a continuación.

4.2. Grado de libertad.

Mecánicamente, un robot está formado por una serie de elementos o eslabones unidos mediante articulaciones que permiten un movimiento relativo entre cada dos eslabones consecutivos. La constitución física de la mayor parte de los robots industriales guarda cierta similitud con la anatomía del brazo humano, por lo que en ocasiones, para hacer referencia a los distintos elementos que componen el robot, se usan términos como cuerpo, brazo, codo y muñeca.

El movimiento de cada articulación puede ser de desplazamiento, de giro o de una combinación de ambos. Cada uno de estos movimientos independientes que puede realizar cada articulación con respecto a la anterior, se denomina grado de libertad.

El número de grados de libertad del robot viene dado por la suma de los grados de libertad de las articulaciones que lo componen. Puesto que, como se ha indicado, las articulaciones empleadas son únicamente las de rotación y prismática con un solo con grado de libertad cada una, el número de grados de libertad del robot suele coincidir con el número de articulaciones de que se compone.

4.3. Parámetros de Denavit-Hartenberg.

La configuración estándar del brazo robot modular es de 6-DOF (seis grados de libertad), RRR (tres grados de libertad rotacional) para posición y una muñeca RPR (Rodar-Paso-Rodar).

Originalmente, un movimiento rígido requiere seis coordenadas para su definición, tres para determinar la posición y tres para establecer la orientación, de forma que un movimiento relativo entre n cuerpos requiere 6^n coordenadas.

Para reducir el número de coordenadas independientes, empleamos las convenciones de Denavit-Hartenberg, que reducen el número de coordenadas a cuatro. Las condiciones que se establecen a la hora de situar los sistemas de referencia para una correcta determinación de estos parámetros son:

1. El eje X_{i+1} debe ser perpendicular al eje Z_i.
2. El eje X_{i+1} debe cortar al eje Z_i en un punto.

Antes de aplicar el método D-H es importante tener en cuenta los siguientes comentarios:

- Se parte de una configuración cualesquiera del robot, si bien es aconsejable colocarlo en una posición sencilla de analizar.
- Se numeran los eslabones, asignando el 0 para la base y *n-1* para el último eslabón, siendo *n* el número de grados de libertad del robot.
- El sistema de coordenadas ortonormal dextrógiro de la base (x_0, y_0, z_0) se establece con el eje z_0 localizado a lo largo del eje de movimiento de la articulación 1 y apuntando hacia fuera del hombro del brazo del robot.
- El sistema de referencia de cada eslabón se coloca al final del mismo, en el extremo de la articulación a la cual está conectado el eslabón siguiente.
- El ángulo o desplazamiento de cada eslabón siempre se mide tomando como base el sistema de referencia del eslabón anterior.

Teniendo en cuenta las normas y condiciones para el análisis del robot para la obtención de sus parámetros de Denavit-Hartenberg, procedemos al estudio correspondiente. Cada sistema de coordenadas se establece sobre las siguientes reglas:

D_i: es la distancia medida desde el origen del sistema *i-1*, a lo largo del eje Z_{i-1} hasta la intersección del eje Z_{i-1} con el eje X_i:

$$D = dist(O_{i-1}, X_i \cap Z_{i-1})$$

A_i: es la distancia de separación entre los orígenes de los sistemas de referencia *i-1* e *i*, medida a lo largo del eje X_i hasta la intersección con el eje Z_{i-1}.

$$A = dist(\, O_i, X_i \cap Z_{i-1})$$

θ_i: es el ángulo de la articulación desde el eje X_{i-1} hasta el eje X_i, medido respecto del eje Z_{i-1}, usando la regla de la mano derecha.

$$\theta = (Z_{i-1} \hat{} Z_i) \,\big|\, X_i$$

α_i: es el ángulo que separa los ejes Z_i y Z_{i-1}, medido respecto del eje X_i

$$\alpha = (X_{i-1}\hat{}X_i) \mid Z_{i-1}$$

En nuestro caso, y apoyándonos en el software de MATLAB para robots (al cual se dedicará el Capítulo 5), tomamos como posición inicial del robot la que se muestra en la *Figura 3.14*, obteniendo los resultados mostrados en la tabla siguiente:

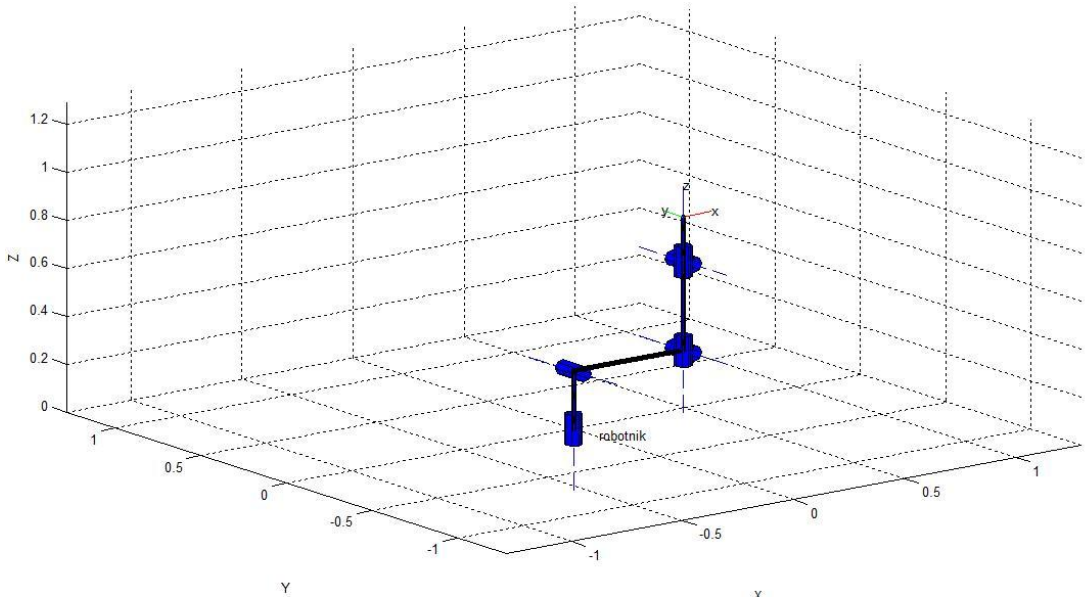


Figura 3.14.- Posición 1.

	D	**A**	**θ**	**α**
Eslabón 1	l_1	0	θ_1	π/2
Eslabón 2	0	l_2	θ_2	0
Eslabón 3	0	0	θ_3	- π/2
Eslabón 4	l_4	0	θ_4	π/2
Eslabón 5	0	0	θ_5	- π/2
Eslabón 6	l_6	0	θ_6	0

Tabla 3.7.- Parámetros Denavit-Hartenberg. Modelo 1.

Donde los valores de las incógnitas de la tabla se corresponden con las medidas de los eslabones del robot, que son los siguientes (en mm):

$$l_1 = 241.1$$
$$l_2 = 495$$
$$l_4 = 372$$
$$l_6 = 183.2$$

Estos parámetros pueden variar en función de la posición que se tome inicialmente para el estudio. Vemos como varían éstos para la siguiente posición en la que el brazo se encuentra completamente vertical:

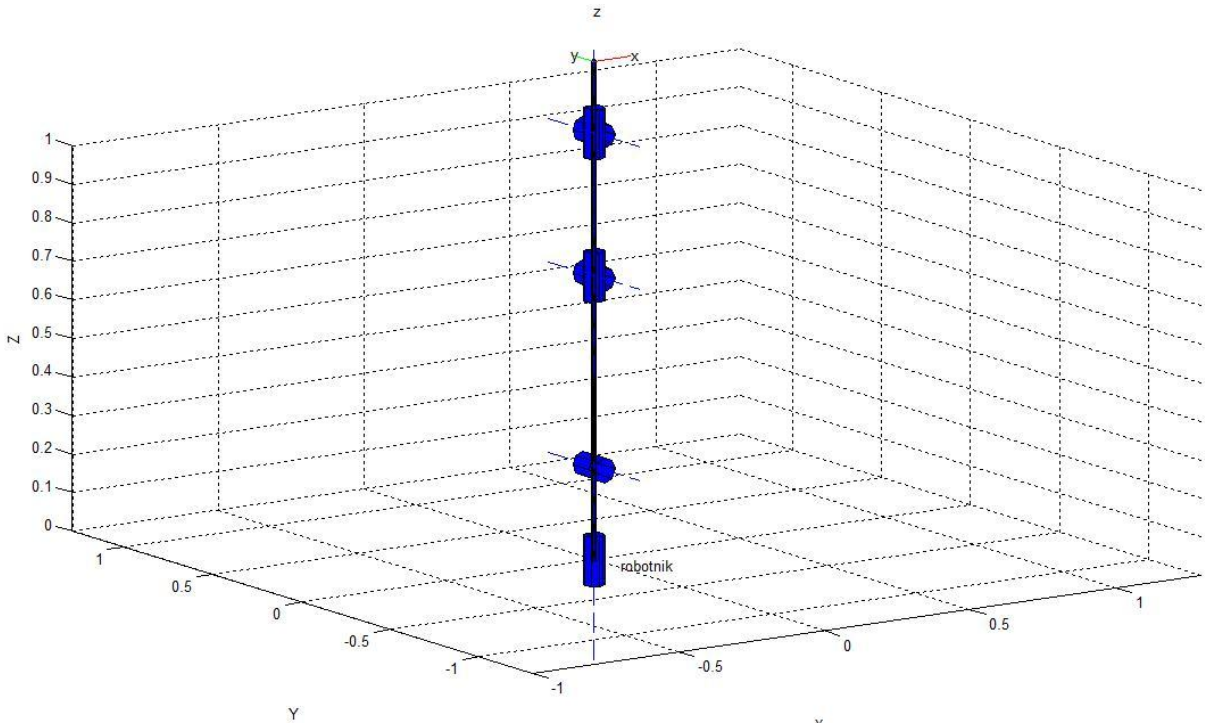


Figura 3.15.- Posición 2.

Los parámetros que se obtienen son parecidos a los obtenidos anteriormente, con ligeras diferencias:

	D	**A**	**θ**	**α**
Eslabón 1	l_1	0	0	π/2
Eslabón 2	0	l_2	0	0
Eslabón 3	0	0	0	- π/2
Eslabón 4	l_4	0	0	π/2
Eslabón 5	0	0	0	- π/2
Eslabón 6	l_6	0	0	0

Tabla 3.8.- Parámetros Denavit-Hartenberg. Modelo 2.

4.4. Espacio de trabajo del robot.

Las siguientes figuras muestran el espacio de trabajo del robot acorde con el conjunto de rangos límites de cada uno de los elementos. El conjunto del robot ha sido limitado para evitar posibles colisiones entre los propios componentes del sistema (restricciones mecánicas, ej.: el mecanismo 5 puede colisionar si el ángulo es mayor de 90º), pero también para reducir la cantidad de soluciones cinemáticas redundantes (ej.: los elementos 4 y 6 pueden rotar infinitamente, pero esto sólo incrementaría el número de posibles soluciones, incrementando la complejidad).

Elemento	Mín	Máx
q1	-180º	180º
q2	-126º	126º
q3	-100º	100º
q4	-180º	180º
q5	-90º	90º
q6	-180º	180º

Tabla 3.9.- Límites establecidos para los eslabones del robot.

Para algunas aplicaciones (ej.: atornillar con el 6º mecanismo) tiene sentido sobrepasar los rangos programados. Si esto fuese necesario, es importante tener en cuenta que el cableado interno impone también una restricción en el número de giros.

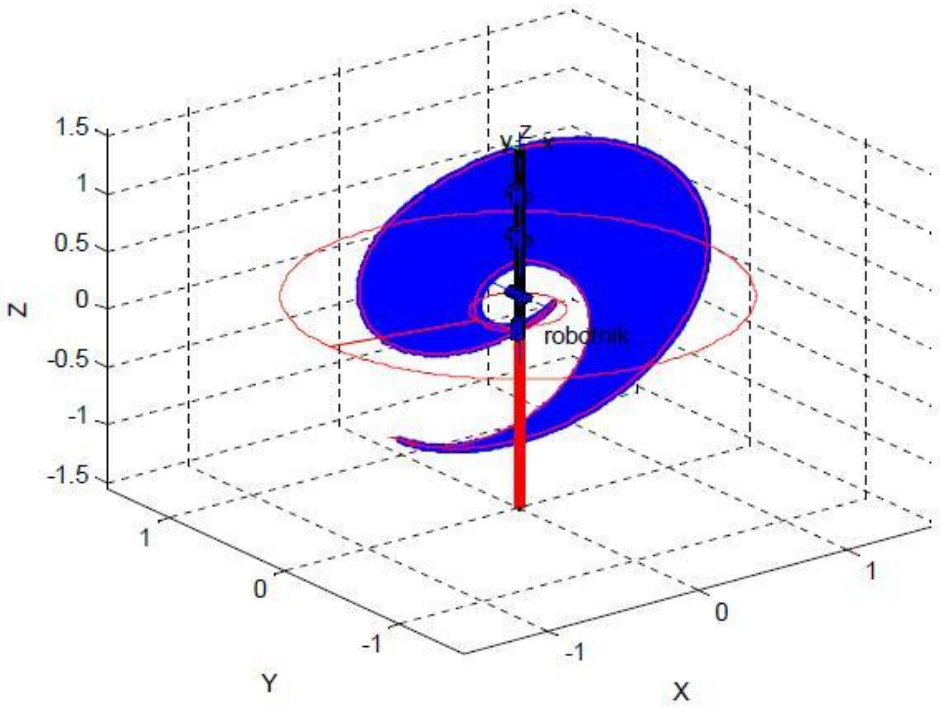


Figura 3.16.- Espacio de trabajo del robot.

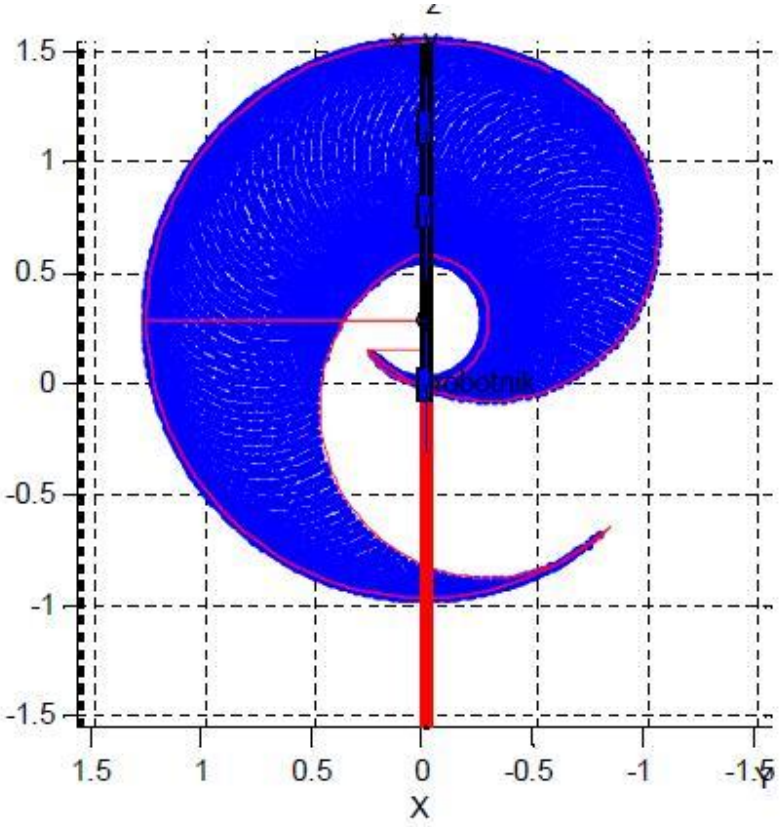


Figura 3.17.- Espacio de trabajo del robot (2).

4.5. Estudio de la cinemática del robot.

La cinemática del robot estudia el movimiento del mismo con respecto a un sistema de referencia. La cinemática se interesa por la descripción analítica del movimiento espacial del robot como una función del tiempo y, en particular, por las relaciones entre la posición y la orientación de la herramienta del robot con los valores que toman sus coordenadas de sus articulaciones.

Existen dos problemas fundamentales a resolver con respecto a la cinemática del robot:

a) **Cinemática directa:** Consiste en determinar la posición y orientación del extremo final del robot con respecto al sistema de la base del robot a partir de conocer los valores de las articulaciones y los parámetros geométricos.
b) **Cinemática inversa:** Resuelve la configuración que debe adoptar el robot para una posición y orientación conocidas del extremo.

4.5.1. Resolución del problema cinemático directo.

Para solventar el problema cinemático directo se utiliza fundamentalmente el álgebra vectorial y matricial para representar y describir la localización de un objeto en el espacio tridimensional con respecto a un sistema de referencia fijo, ya que, para robots con más de dos grados de libertad resulta difícil aplicar métodos geométricos.

En este caso vamos a resolver el problema a través de dos métodos diferentes:

- Método de matrices de transformación homogénea.
- Algoritmo Denavir-Hartenberg.

4.5.1.1. Método de matrices de transformación homogénea.

Dado que un robot se puede considerar como una cadena cinemática formada por objetos rígidos o eslabones unidos entre sí mediante articulaciones, se puede establecer un sistema de referencia fijo situado en la base del robot y describir la localización de cada uno de los eslabones con respeto a dicho sistema de referencia. De esta forma, el problema cinemático directo se reduce a encontrar una matriz de transformación homogénea **T** que relacione la posición y orientación del extremo del robot respecto del sistema de referencia fijo situado en la base del mismo. Esta matriz **T** será función de las coordenadas articulares.

En general, un robot de *n* grados de libertad está formado por *n* eslabones unidos por *n* articulaciones, de forma que cada par articulación-eslabón constituye un grado de libertad. A cada eslabón se le puede asociar un sistema de referencia solidario a él y, empleando las transformaciones homogéneas, es posible representar las rotaciones y traslaciones relativas entre los distintos eslabones que componen el robot. De este modo, también resulta posible la representación de forma total o parcial la cadena cinemática del robot a través del producto de las diferentes matrices obtenidas para los diferentes eslabones del robot.

Así, comenzaremos a deducir las diferentes matrices que pueden asociarse a nuestro robot en cuestión:

La primera transformación, ${}^{0}A_{1}$, describe la posición y orientación del sistema de referencia solidario al primer eslabón con respecto al sistema de referencia de solidario a la base. En este caso, la transformación se constituye de un movimiento de rotación en torno al eje *z* respecto del sistema de referencia fijo de la base y una traslación a lo largo del nuevo eje *z*, de forma que tenemos:

$$ {}^{0}A_{1} = \begin{bmatrix} C\theta & -S\theta & 0 & 0 \\ S\theta & C\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

A continuación, el sistema realiza una rotación a través del eje *x* y una traslación a lo largo del nuevo eje *y:*

$$ {}^{1}A_{2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha & -S\alpha & 0 \\ 0 & S\alpha & C\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & l_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

La siguiente transformación que nos encontramos es un giro sobre el eje *x:*

$$ {}^{2}A_{3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\omega & -S\omega & 0 \\ 0 & S\omega & C\omega & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Después, tenemos un giro sobre el eje *z* y una traslación sobre el nuevo eje *z:*

$$ {}^{3}A_{4} = \begin{bmatrix} C\varphi & -S\varphi & 0 & 0 \\ S\varphi & C\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Para la matriz ${}^{4}A_{5}$ tenemos un giro sobre el eje *x*:

$$ {}^{4}A_{5} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\omega & -S\omega & 0 \\ 0 & S\omega & C\omega & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Por último, tenemos una rotación alrededor del eje *z* y una traslación sobre el nuevo eje *y*, quedándonos:

$$ {}^{5}A_{6} = \begin{bmatrix} C\psi & -S\psi & 0 & 0 \\ S\psi & C\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & l_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Resultando que la relación entre el sistema de coordenadas de la base y del extremo final del robot queda definida por una matriz de transformación homogénea **T** función de las coordenadas articulares:

$$\boldsymbol{T} = {}^{0}A_{6} = {}^{0}A_{1} \cdot {}^{1}A_{2} \cdot {}^{2}A_{3} \cdot {}^{3}A_{4} \cdot {}^{4}A_{5} \cdot {}^{5}A_{6}$$

4.5.1.2.Algoritmo Denavit-Hartenberg.

En general, una matriz de transformación homogénea queda definida por 6 grados de libertad, el método de Denavit-Hartenberg, permite, en eslabones rígidos, reducir éste a 4 con la correcta elección de los sistemas de coordenadas.

Estas 4 transformaciones básicas consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del elemento *i-1* con el elemento *i.* Las transformaciones en cuestión son las siguientes:

1. Rotación alrededor del eje Z_{i-1} un ángulo θ_i.
2. Traslación a lo largo del eje Z_{i-1} una distancia d_i;
3. Traslación a lo largo del eje X_i una distancia a_i;
4. Rotación alrededor del eje X_i un ángulo α_i.

Donde las transformaciones se refieren al sistema móvil.

Dado que el producto de matrices no es conmutativo, las transformaciones se han de realizar en el orden indicado. De este modo se tiene que:

$${}^{i-1}A_{i} = Rotz(\theta_i) \cdot T(0,0,d_i) \cdot T(a_i,0,0) \cdot Rotx(\alpha_i)$$

Y realizando el producto obtenemos que:

$${}^{i-1}A_{i} = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde $\theta_i, \alpha_i, d_i, a_i$ son los parámetros D-H del eslabón *i.*

De este modo, podemos obtener de forma sencilla las matrices de transformación para cada uno de los eslabones. Téngase en cuenta que estas transformaciones sólo son válidas en el caso de que los sistemas de rotación hayan sido definidos conforme a las normas expuestas anteriormente (convenciones Denavit-Hartenberg).

Siguiendo este método y conforme a los parámetros obtenidos en el primer apartado, obtenemos las matrices de transformación correspondientes para nuestro caso. Según la *tabla 2*, tenemos los siguientes resultados:

	D	**A**	**θ**	**α**
Eslabón 1	l_1	0	0	π/2
Eslabón 2	0	l_2	0	0
Eslabón 3	0	0	0	- π/2
Eslabón 4	l_4	0	0	π/2
Eslabón 5	0	0	0	- π/2
Eslabón 6	l_6	0	0	0

Tabla 3.11.- Parámetros Denavit-Hartenberg.

Y conforme a estos resultados obtenidos a través del estudio realizado de los parámetros Denavit Hartenberg correspondiente al robot, calculamos las matrices:

$$^{0}A_{1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^{1}A_{2} = \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{2}A_{3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^{3}A_{4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{4}A_{5} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^{5}A_{6} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Y el resultado es la matriz de transformación **T** que nos relaciona, una vez más, el sistema fijo de la base con el extremo final del robot a través del producto de las matrices:

$$\boldsymbol{T} = {}^{0}A_{6} = {}^{0}A_{1} \cdot {}^{1}A_{2} \cdot {}^{2}A_{3} \cdot {}^{3}A_{4} \cdot {}^{4}A_{5} \cdot {}^{5}A_{6}$$

4.5.2. El problema de la cinemática inversa.

Como ya se ha introducido, el problema de la cinemática inversa consiste en obtener la posición de los distintos ejes del robot a partir de la posición y orientación cartesianas del efector final del robot. Resolver la cinemática inversa es un proceso complejo debido a las siguientes razones:

- Pueden existir múltiples soluciones (existen 8 configuraciones posibles en el robot modular: brazo derecho/izquierdo, codo arriba/abajo, muñeca izquierda/derecha);
- Pueden existir infinitas soluciones (ej.: cuando el centro de la muñeca está alineado con el primer eje).
- Ninguna solución es posible con la estructura cinemática del robot.

Este problema puede ser resuelto de diferentes formas, por ecuación cinemática (métodos geométricos) o, la solución más común, ecuación cinemática diferencial Jacobiana, aunque en esta memoria no se va a desarrollar dicho método.

5. Simulación mediante MATLAB.

5.1. Herramienta para la simulación de sistemas robotizados. Ámbito de aplicación.

La simulación de sistemas robotizados está íntimamente ligada a la potencia computacional de los procesadores de cálculo. El gran avance producido con los microprocesadores actuales ha permitido obtener paquetes de simulación dinámica capaces de simular el comportamiento dinámico de casi cualquier mecanismo multicuerpo. Estos paquetes incorporan librerías de articulaciones y fuerzas que permiten al usuario construir su modelo en un tiempo relativamente corto. Estos paquetes son utilizados en los centros de investigación y en las empresas de tecnología para el diseña de prototipos mecánicos y es debido a sus altas prestaciones, que su coste económico es elevado.

En la red, sin embargo, podemos encontrar paquetes de demostración de software especializado en la simulación de robots clásicos. El inconveniente de éstos es que suelen ser cerrados desde el punto de vista del código fuente y, por tanto, están limitados a las capacidades que el programador haya incorporado antes de su publicación.

En otro nivel encontramos otro tipo de herramientas diseñadas para el análisis de sistemas robotizados que se presentan con el código fuente accesible al usuario. La filosofía de código abierto pretende ampliar continuamente las capacidades de ese código, permitiendo al usuario su contribución a éste. Este es el caso de la Robotics Toolbox de MATLAB de Peter I. Corke y a cuya presentación se procederá en este apartado.

5.2. Robotics Toolbox de MATLAB.

La Robotics Toolbox de MATLAB proporciona varias funciones que son útiles en robótica. Su primera versión fue desarrollada en 1996 por Peter I. Corke y se puede obtener de manera libre desde MathWorks.

La Toolbox ofrece muchas funciones que son útiles en robótica y referidas al ámbito de la cinemática y dinámica, así como la generación de trayectorias del robot. Además, resulta muy útil tanto para la simulación como para analizar los resultados de los experimentos con robots reales.

Esta Toolbox se basa en un método muy general de la representación de la cinemática y la dinámica de manipuladores unidos en serie. Estos parámetros se encapsulan en objetos Matlab. El usuario puede crear cualquier robot compuesto por eslabones unidos en serie, sin embargo, la aplicación también proporciona diferentes ejemplos reales, tales como el Puma 560 y el grupo de Stanford. La Toolbox también ofrece funciones para la manipulación y la conversión entre tipos de datos tales como vectores, transformaciones homogéneas y cuaternios, que son necesarios para representar posición y orientación tridimensional.

Las rutinas se desarrollan de una manera directa, lo que permite una fácil comprensión, tal vez a expensas de la eficiencia computacional.

5.3. Funciones de utilidad de la Robotics Toolbox.

Como ya se ha mencionado, la Toolbox incorpora diferentes funciones de gran utilidad, las cuales se muestran en las siguientes tablas y se procederán a comentar aquellas más ilustrativas a la hora de aplicación en casos reales.

Transformaciones Homogéneas	
`angvec2tr`	Ángulo/vector
`eul2tr`	Ángulo de Euler para la transformación homogénea
`oa2tr`	Vector de orientación y aproximación para la transformación homogénea
`rpy2tr`	Ángulos de Roll/pitch/yaw (alabeo/cabeceo/guiñada) de la transformación
`tr2angvec`	Conversión de una transformación homogénea o matriz de rotación a forma vectorial/angular.
`tr2eul`	Conversión de una transformación homogénea o matriz de rotación en ángulos de Euler
`t2r`	Conversión de una transformación homogénea en una submatriz de rotación
`tr2rpy`	Matriz de rotación para ángulos de giro Roll/pitch/yaw
`trotx`	Transformación homogénea para una rotación sobre el eje X
`troty`	Transformación homogénea para una rotación sobre el eje Y
`trotz`	Transformación homogénea para una rotación sobre el eje Z
`transl`	Establece o extrae la componente traslacional de una transformación homogénea
`trnorm`	Transformación homogénea normalizada
`trplot`	

Matrices de Rotación	
`angvecr`	Convierte la forma angular/vectorial en matriz de rotación
`eul2r`	Convierte los ángulos de Euler en matriz de rotación
`oa2r`	Convierte el vector de orientación y aproximación en una transformación homogénea

rotx	Matriz de rotación para una rotación sobre el eje X
roty	Matriz de rotación para una rotación sobre el eje Y
rotz	Matriz de rotación para una rotación sobre el eje Z
rpy2r	Conversión de los ángulos Roll/Pitch/Yaw en matriz de rotación
r2t	Conversión de una matriz de rotación en transformación homogénea

Generación de Trayectorias	
ctraj	Genera una trayectoria cartesiana
jtraj	Genera la trayectoria espacial de una articulación
trinterp	Interpola transformaciones homogéneas

Generación de Manipuladores Unidos en Serie	
link	Construye un objeto de unión (eslabón)
nofriction	Elimina la fricción de un objeto robot
perturb	Modifica aleatoriamente los parámetros de la dinámica
robot	Construye el objeto robot
showlink	Muestra el eslabón/robot en detalle

Gráficos	
drivebot	Permite guiar a un robot de forma gráfica
plot	Traza/anima un robot

Dinámica	
accel	Calcula la dinámica directa
cinertia	Calcula la matriz de inercia del manipulador
coriolis	Calcula la fuerza de coriolis/centrípeta
fdyn	Dinámica directa (movimiento generado por fuerzas dadas)
friction	Fricción de la articulación
gravload	Calcula la carga gravitacional
inertia	Calcula la matriz de inercia del manipulador
itorque	Calcula la torsión de inercia
rne	Dinámica inversa (las fuerzas son calculadas por el movimiento dado)

Cinemática	
diff2tr	
fkine	Calcula la cinemática directa
ftrans	Transformación fuerza/momento
ikine	Calcula la cinemática inversa
ikin560	Calcula la cinemática inversa para el Puma 560
jacob0	Calcula la Jacobiana en base a un marco coordinado
jacobn	Calcula la Jacobiana en el efector final
tr2diff	Convierte una transformación homogénea en un vector de movimiento diferencial
tr2jac	Convierte la transformación homogénea en Jacobiana

Tabla 3.6.- Funciones de la Robotics Toolbox de MATLAB.

En estas tablas se muestra las funciones más importantes. A continuación, se utilizarán algunas de ellas de forma que podamos aplicarlas a nuestro robot particular. Para ello, primero definimos las dimensiones de los distintos eslabones de nuestro robot en Matlab haciendo uso del siguiente código:

```
%Robotnik Modular Robot Arm Kinematics
d1=0.2411;
a2=0.495;
d4=0.372;
d6=0.1832;
```

Y creamos los eslabones mediante el uso del comando *'Link'*:

```
%L =LINK([alpha A theta D sigma], CONVENTION)
L1=link([pi/2 0 0 d1 0], 'standard');
L2=link([0 a2 0 0 0], 'standard');
L3=link([-pi/2 0 0 0 0], 'standard');
L4=link([pi/2 0 0 d4 0], 'standard');
L5=link([-pi/2 0 0 0], 'standard');
L6=link([0 0 0 d6 0], 'standard');
```

En la estructura de esta función, el parámetro sigma determina si la articulación es rotatoria o prismática. Si éste toma el valor 0 la articulación será rotatoria, para cualquier otro caso, prismática. El resto de los parámetros (*alpha, A, theta, D*) inicializa el modelo cinemático basado en los parámetros de Denavit-Hartenberg, desarrollado anteriormente.

Por defecto, el programa emplea las convenciones Denavit-Hartenberg estándar, pero pueden ser modificadas a través de la modificación del argumento '*CONVENTION*', el cual puede tomar los valores 'modified' o 'standard' (por defecto).

Una vez establecido el modelo cinemático, creamos el robot con la función '*robot*' que unirá los distintos eslabones:

```
% Create robot
r = robot({L1 L2 L3 L4 L5 L6},'robotnik');
```

Por último, definimos la posición inicial que tomará nuestro robot cuando lo mostremos en pantalla de forma gráfica. Según esto, establecemos el valor de inicio de cada articulación:

```
% Define initial position
q0 = [0 pi/2 -pi/2 0 0 0];
```

Y mostramos por pantalla:

```
% Plot robot
plot(r,q0);
```

De forma global, el programa en MATLAB nos queda de la siguiente forma:

```
%Robotnik Modular Robot Arm Kinematics
d1=0.2411;
a2=0.495;
d4=0.372;
d6=0.1832;
%L =LINK([alpha A theta D sigma], CONVENTION)
L1=link([pi/2 0 0 d1 0], 'standard');
L2=link([0 a2 0 0 0], 'standard');
L3=link([-pi/2 0 0 0 0], 'standard');
L4=link([pi/2 0 0 d4 0], 'standard');
L5=link([-pi/2 0 0 0], 'standard');
L6=link([0 0 0 d6 0], 'standard');
% Create robot
r = robot({L1 L2 L3 L4 L5 L6},'robotnik');
% Define initial position
q0 = [0 pi/2 -pi/2 0 0 0];
% Plot robot
plot(r,q0);
```

Y el resultado que se obtiene se muestra en la siguiente imagen:

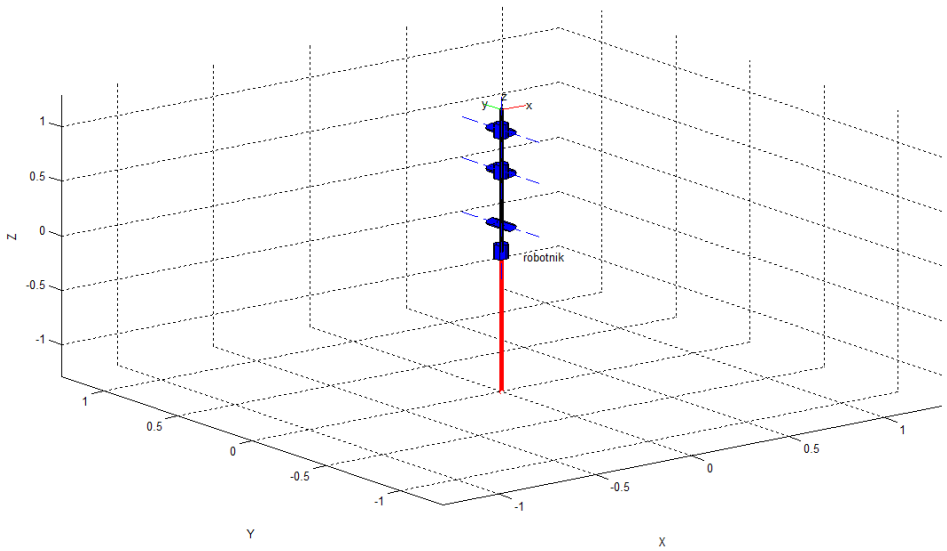


Figura 3.8. – Posición 1.

Gráficamente, tenemos la posibilidad de mostrar el robot y modificar su posición de forma directa a través del comando '*drivebot*' de forma que nos aparece un cuadro como el siguiente, sobre el cual podemos alterar la posición de las distintas articulaciones deslizando las distintas barras que aparecen en la pantalla referidas a cada una de las articulaciones:

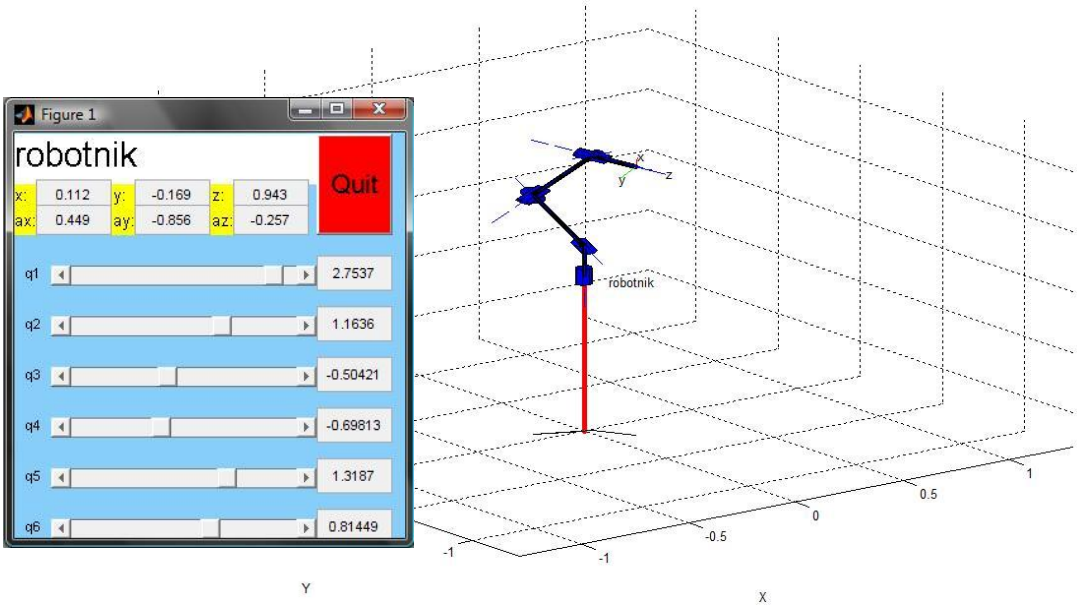
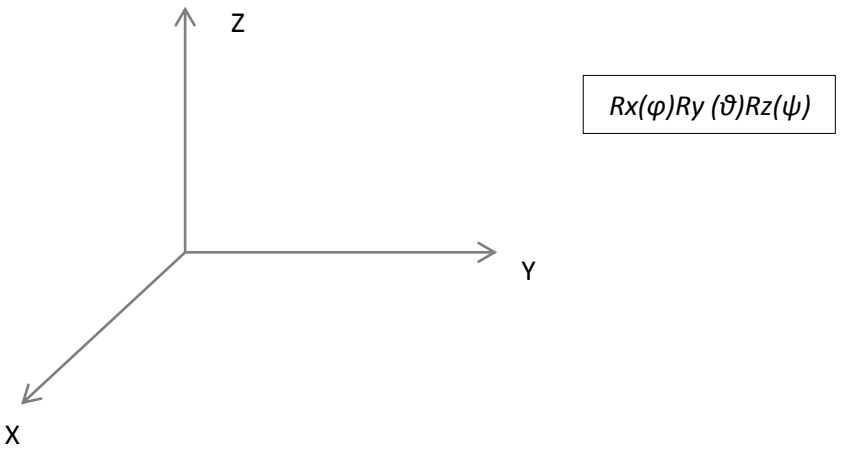


Figura 3.9.- Uso del comando 'drivebot' en MATLAB.

Otras funciones de utilidad son aquellas que resuelven los problemas relacionados con las matrices de rotación y las transformaciones homogéneas.

Veamos, un ejemplo de esto con la función '*eul2tr*' que transforma los ángulos de Euler Roll/Pitch/Yaw en una matriz de transformación o, su semejante, '*eul2r*', que realiza la transformación a una matriz de rotación. Hay que señalar el hecho de que los ángulos siempre vendrán dados en radianes y no en grados. La función realiza la conversión siguiendo:




```
>> T = eul2tr(pi, pi/2, 2*pi/3)
```

```
T =
   -0.0000    0.0000   -1.0000         0
   -0.8660    0.5000    0.0000         0
    0.5000    0.8660    0.0000         0
         0         0         0    1.0000
```

```
>> R = eul2r(pi, pi/2, 2*pi/3)

R =
   -0.0000    0.0000   -1.0000
   -0.8660    0.5000    0.0000
    0.5000    0.8660    0.0000
```

La Toolbox también resulta útil para trabajar con matrices de transformación homogénea. Como ya se explicó en el capítulo anterior, estas matrices son utilizadas para describir la localización de sólidos en un espacio n-dimensional, expresando las relaciones entre coordenadas cartesianas. La gran innovación que introdujeron estas matrices es que permiten representar en una única matriz la posición y la orientación del cuerpo.

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix}$$

Donde:
- R- matriz de rotación
- p- matriz de traslación
- f- matiz de perspectiva
- w- matriz de escaldo

Es importante notar que el producto de estas matrices en general no es conmutativo.

Para crear una traslación pura basta con utilizar la función “transl”:

```
TR = TRANSL(X, Y, Z)
>> transl (0.5, 0.0, 0.0)
```

En el caso de querer generar una rotación se utiliza “rot*e*” donde *e* es el eje de rotación

```
TR = ROTY (theta)
>> roty (pi/2)
```

O si se desea implementar una combinación de ellos simplemente se multiplican las matrices :

```
>> t = transl (0.5, 0.0, 0.0) * roty(pi/2)
```

6. Desarrollo de la aplicación en C++.

Como ya se ha introducido al inicio de esta memoria, uno de los objetivos principales de este proyecto se centraba en la creación de una interfaz a través de la cual el usuario pudiese visualizar el estado del robot. Para llevar a cabo este propósito, por facilidad, se decidió que la mejor opción de trabajo era emplear el software Microsoft Visual Studio 2010 por la simplicidad que ofrecía a la hora de la programación con ventanas.

6.1. Microsoft Visual Studio 2010.

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Microsoft Visual Studio 2010 incorpora características que hacen que todo el proceso de desarrollo sea más sencillo, desde el diseño a la implementación. Concretamente, la versión de 2010 es un exhaustivo paquete de herramientas de administración del ciclo de vida de las aplicaciones para equipos. Tanto al crear soluciones nuevas como al mejorar las aplicaciones ya existentes, Visual Studio 2010 admite un número cada vez mayor de plataformas y tecnologías (incluidos los sistemas informáticos en cloud y en paralelo).

Figura 6.1.- Visual Studio 2010.

Toda la interfaz ha sido rediseñada para que fuese menos recargada que en las versiones anteriores. Y es que una queja frecuente de muchos usuarios de Microsoft Visual Studio ha sido precisamente la cantidad de elementos que distraían más que ayudaban.

El entorno de desarrollo de Microsoft Visual Studio 2010 difiere ligeramente dependiendo del lenguaje elegido. La configuración para C++, por ejemplo, activa atajos de teclado especiales, mientras que el entorno para Visual Basic da más relevancia al diseño de los formularios.

El abanico de lenguajes soportados por Microsoft Visual Studio va desde clásicos como Visual Basic o C++ hasta otros más recientes, como C# y F#. Con Microsoft Visual Studio también puedes diseñar aplicaciones Silverlight, la alternativa de Microsoft a Adobe Flash.

6.2. Visual C++ .NET : Windows Forms

Un formulario Windows representa la conocida ventana utilizada en los Sistemas Operativos de tipo Windows.

Por otra parte, un control es aquel elemento situado dentro de una ventana o formulario y que permite al usuario de la aplicación Windows interactuar con la misma, para introducir datos o recuperar información.

Dentro de .NET, las ventanas clásicas Windows reciben la denominación de Windows Forms o WinForms.

Los controles más comunes y que se van a emplear en nuestra aplicación, se van a exponer de forma clara a continuación, añadiendo el correspondiente código para poder identificarlo así en el código de la herramienta desarrollada.

6.3. Desarrollo de la clase *Robot.*

El objetivo que se persigue con el desarrollo de una clase '*Robot*' es la de ofrecer la posibilidad de creación de un objeto "robot", que venga definido por una serie de parámetros que coinciden con los distintos parámetros que se pueden obtener del propio robot a partir de su librería ya implementada m5api y comentada en capítulos anteriores.

La aplicación que se ha desarrollado, así como la clase creada, tienen el objetivo de solventar, a modo de simulación, el problema de comunicación PC-Robot que surgió durante el progreso de este proyecto. El problema surge debido a la imposibilidad de realizar la comunicación directa con el programa de C++ y el robot de trabajo. Lo que se pretendió entonces fue la creación de un formulario y un tipo de datos a los que se recurriera en caso de no poder establecer un lazo de comunicación con el brazo.

Dicho de otro modo, y en términos de programación, lo que se ha llevado a cabo es la definición de unas funciones con el mismo nombre que las originales y existentes en la librería m5api y que, gracias a la sobrecarga de funciones que permite C++, se utilizan en caso de no conseguir el acceso directo al robot. Así, en función de los parámetros de llamada, el PC accede a las funciones de la librería m5api o a las definidas junto a la clase *Robot*.

La clase se estructura como sigue:

```
ref class Robot
{
 public:
        Robot(void);


        double defvelocidad;
        double defcorriente;
        double defposicion;
        double velocidad;
        double corriente;
        double posicion;
        double aceleracion;
};
```

Asimismo, esta clase lleva asociadas algunas de las funciones descritas, que recogen y establecen los valores de las variables que conforman la clase:

```
Private:
      double PCube_setVel (System::String^ sControlText);
      double PCube_setVelDef (System::String^ sControlText);
      double PCube_setPos (System::String^ sControlText);
      double PCube_setPosDef (System::String^ sControlText);
      double PCube_setCur (System::String^ sControlText);
      double PCube_setCurDef (System::String^ sControlText);
      double PCube_setDefMaxAcc (System::String^ sControlText);
      double PCube_getVel (&Robot r);
      double PCube_getPos (&Robot r);
      double PCube_getVel (&Robot r);
      double PCube_getCur (&Robot r);
```

En este caso, las variables miembro de la clase son públicas, puesto que debe poder accederse a ellas desde ambos formularios creados y, aun existiendo otros métodos para poder compartir variables entre formularios de un mismo programa, éstos complican demasiado el código y exceden mis conocimientos de C++.

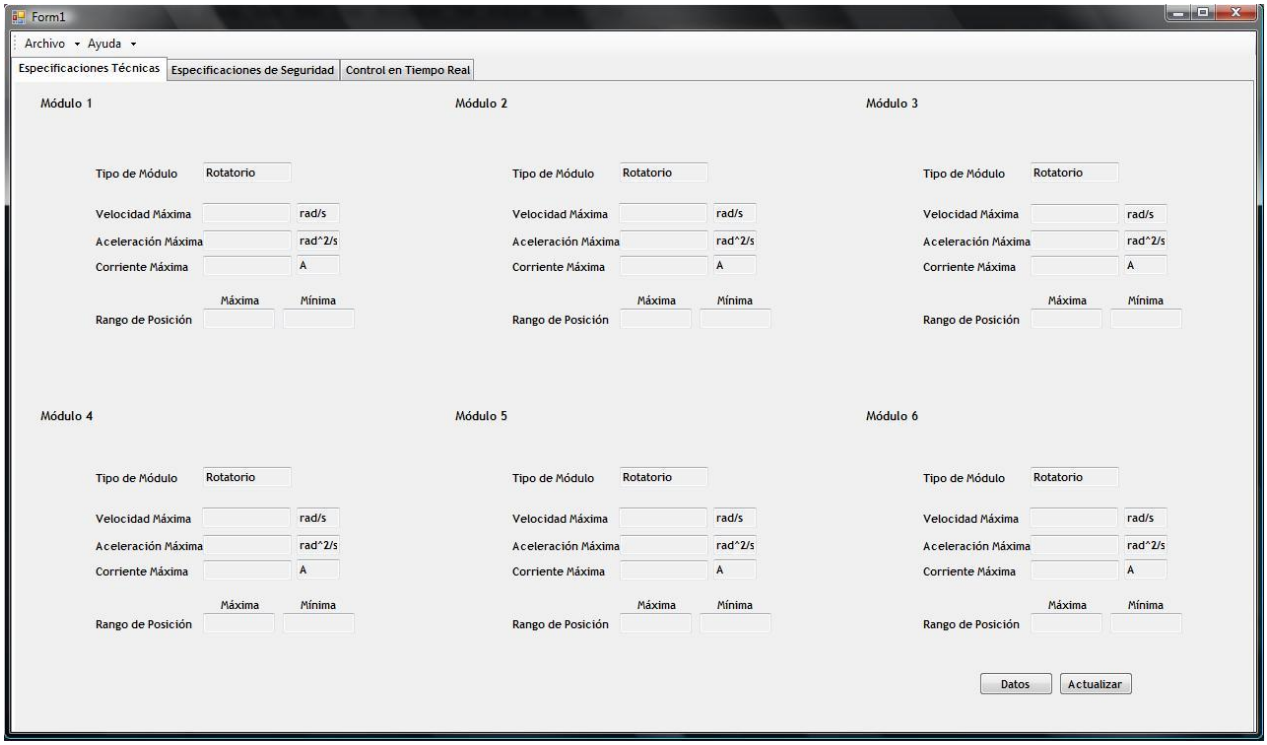
6.4. Aplicación.

En este apartado se tratará de explicar lo más claramente posible cuál es la constitución del programa diseñado, las partes de que consta y su funcionamiento, además de anexarse el código correspondiente.

La herramienta desarrollada consta de dos ventanas o formularios de Windows, Form1 y Form2, correspondientes a la aplicación principal y un formulario de datos, respectivamente.

1. Form1.

Como ya se ha descrito, el Form1 constituye la base del programa desarrollado. Este es el formulario en el que se muestran todos los valores procedentes del robot y se estructura de la siguiente forma:



La ventana principal está dividida en tres pestañas, que muestran diferentes parámetros, siempre agrupadas en torno a criterio común:

1. Especificaciones Técnicas:

Es la pestaña que visualizamos inicialmente por defecto. En esta ventana se pueden observar todas las variables que por defecto lleva establecido el robot. Son características

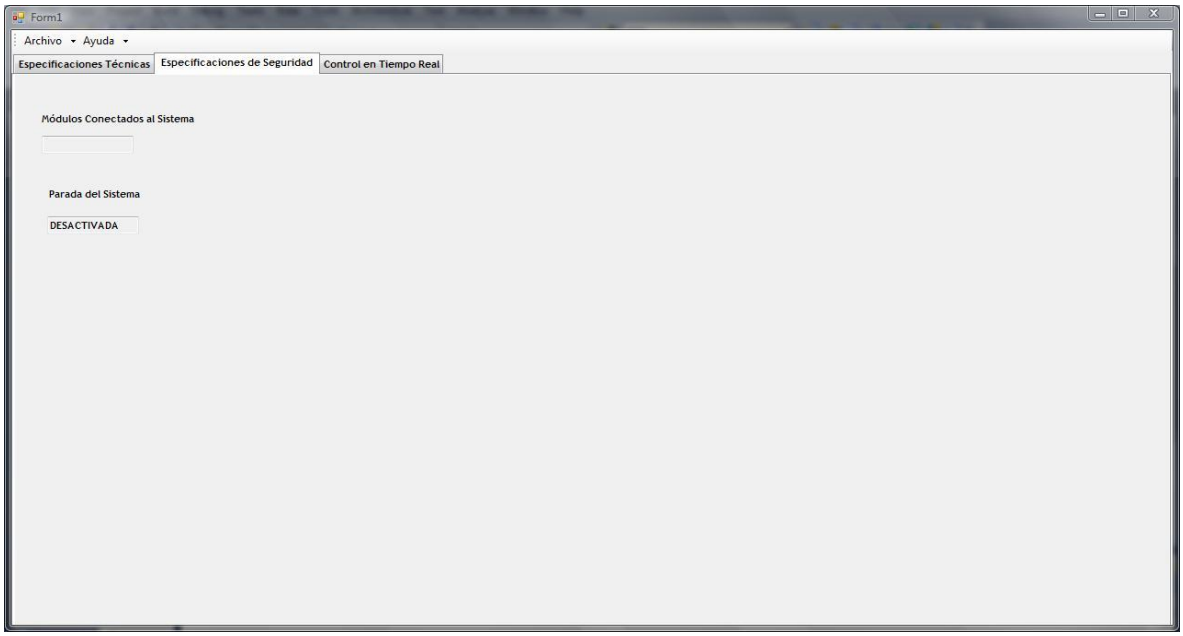
definidas previamente al uso del equipo y que no han sido alteradas por el usuario, aunque, como veremos más adelante, estas pueden modificarse.

Internamente, la pantalla se encuentra seccionada en seis zonas, cada una de ellas referente a un módulo distinto. Así, las variables se encuentran agrupadas según el módulo al que hacen referencia. Las variables de las que aquí se habla son las siguientes: Tipo de módulo, Velocidad Máxima, Aceleración Máxima, Corriente Máxima y Rango de Posición, ésta última definida por unos valores extremos mínimo y máximo. El valor que toman estas variables se muestra junto a su nombre, a través de controles TextBox en los que aparece la cifra numérica correspondiente a la variable.

Además de esto, la pantalla de Especificaciones Técnicas incluye dos botones, Datos y Actualizar, cuya función se explicará con detalle más adelante.

2. Especificaciones de Seguridad:

Este apartado es mucho más simple que el anterior. Contiene dos variables que informan del estado del equipo y que ayudan a mantener la seguridad del equipo por la información que proporcionan. Estas variables son: Módulos Conectados al Sistema, la cual indica el número de módulos detectados en el bus; y Parada de Emergencia, la cual identifica el momento en que se acciona el botón de parada de emergencia. Ésta última puede tomar dos valores, DESACTIVADA, cuando el sistema actúa con normalidad; y, ACTIVADA, en caso de haberse activado la parada.

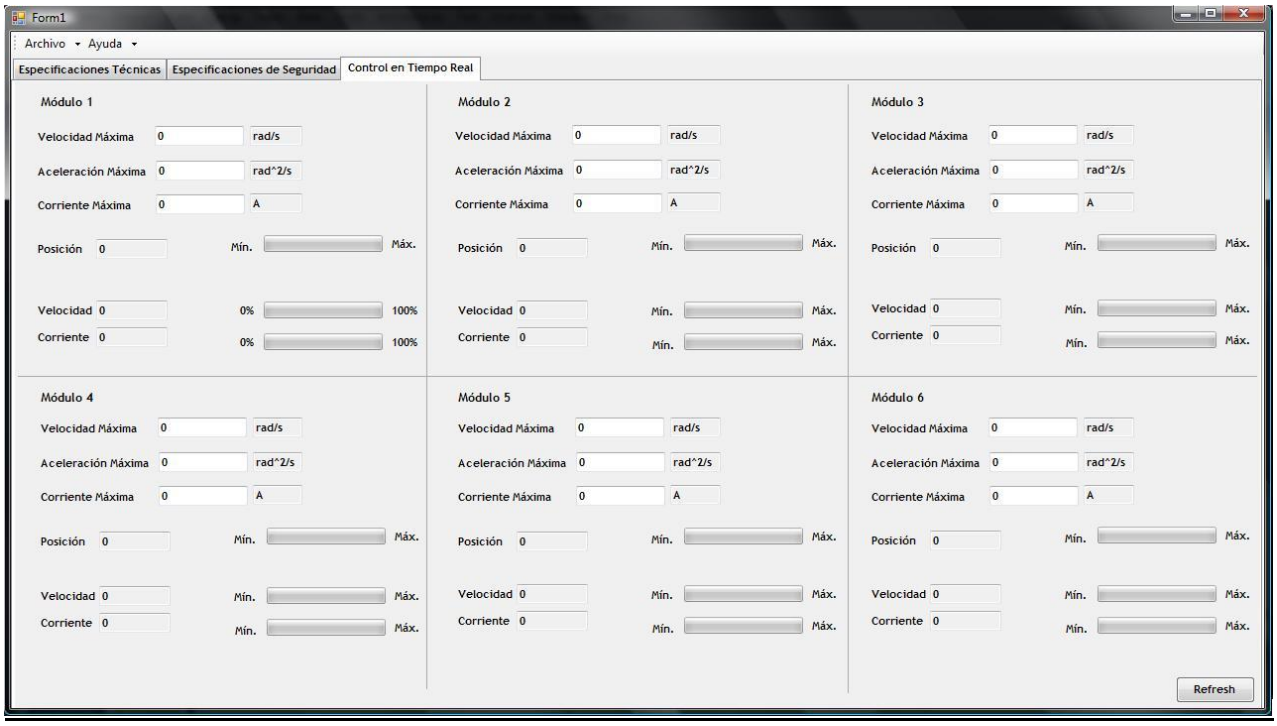


3. Control en Tiempo Real:

Este es, quizás, el apartado más importante de la aplicación. Es la ventana en la que se muestran las variables y sus valores en tiempo real. La aplicación está desarrollada de forma que cada segundo se actualice el valor de las diferentes magnitudes, aunque, por supuesto, este es sólo un valor de referencia establecido con la finalidad de comprobar el correcto funcionamiento al tiempo que se evita el colapse del procesador y puede ser modificada según las necesidades del usuario y el trabajo que esté realizando con el robot. Para operaciones muy minuciosas que requieran de un seguimiento exhaustivo del proceso, este tiempo puede reducirse hasta valores de décimas de segundo.

En este caso, al igual que el apartado de Especificaciones Técnicas, la pantalla se encuentra seccionada en seis zonas, esta vez delimitadas debido a la concentración de información de la pantalla, según el módulo al que se refiere la información mostrada. Las variables que se presentan en pantalla son las siguientes: Velocidad Máxima, Aceleración Máxima, Corriente Máxima, Posición, Velocidad y Corriente.

Como se aprecia, en esta ventana vuelven a aparecer las variables Velocidad Máxima, Aceleración Máxima y Corriente Máxima. Esto se debe a que, en el apartado Especificaciones Técnicas, las variables que se muestran son las variables por defecto, aquellas que vienen establecidas para el robot y, una vez que se muestran, su valor no cambia, no se actualizan. Sin embargo, en Control en Tiempo Real, se muestran las variables según van siendo modificadas por el usuario a lo largo del tiempo.



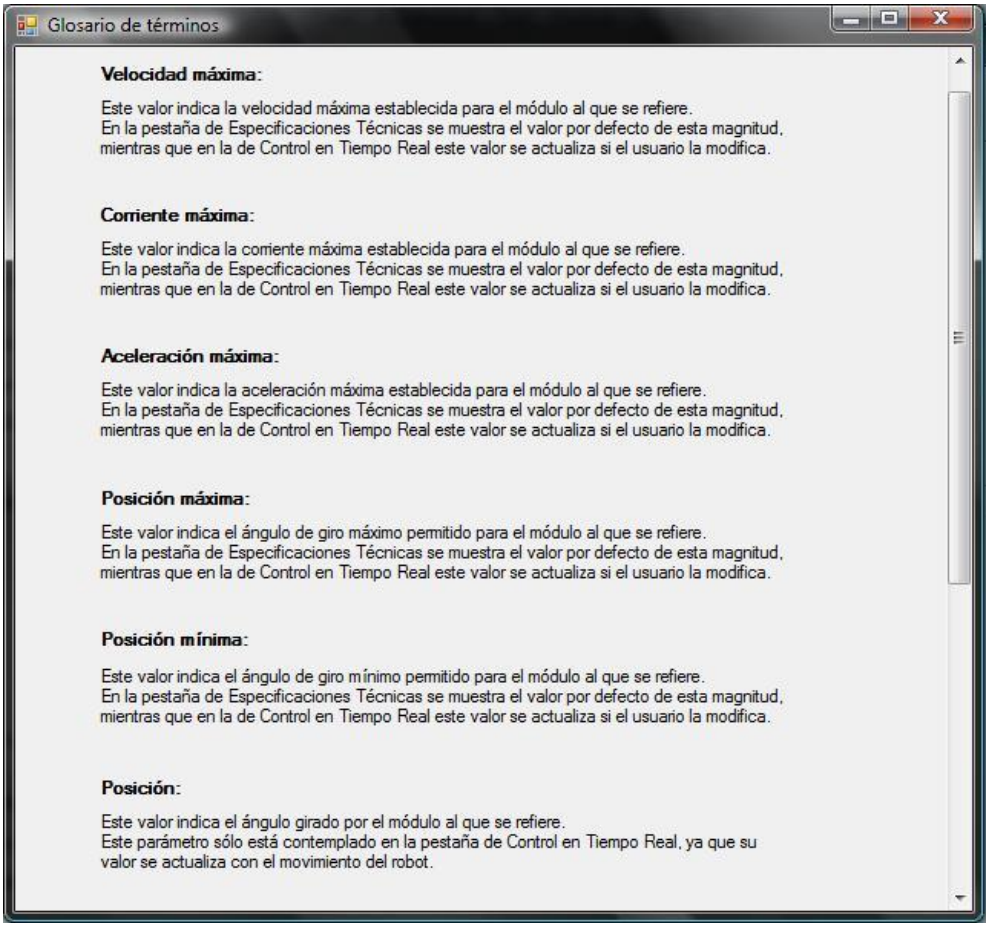
Además, las magnitudes Posición, Velocidad y Corriente muestran el valor de la posición, velocidad y corriente de cada uno de los módulos en cada instante de tiempo. Son las variables que describen el movimiento del robot y su comportamiento en general. En cuanto a estos parámetros, no sólo se muestra el valor que toma la magnitud en cada momento, sino que este valor se muestra en relación al valor máximo que hay definido para ese parámetro en ese momento a través de una barra de progresión. Un ejemplo para que quede completamente claro este último aspecto, podría un caso en que la Velocidad Máxima tuviera un valor de 3rad/s, mientras que el valor de la Velocidad un valor de 1rad/s, este parámetro es un tercio respecto de la máxima establecida, luego la barra se llenaría 1/3.

Por otra parte, esta pantalla incorpora un botón, Refresh, que como su nombre indica, sirve para refrescar la información presentada en los diferentes TextBox que existen en la ventana de forma manual por alguna circunstancia, aunque como se advierte, es innecesario ya que el proceso de actualizar la información se realiza automáticamente.

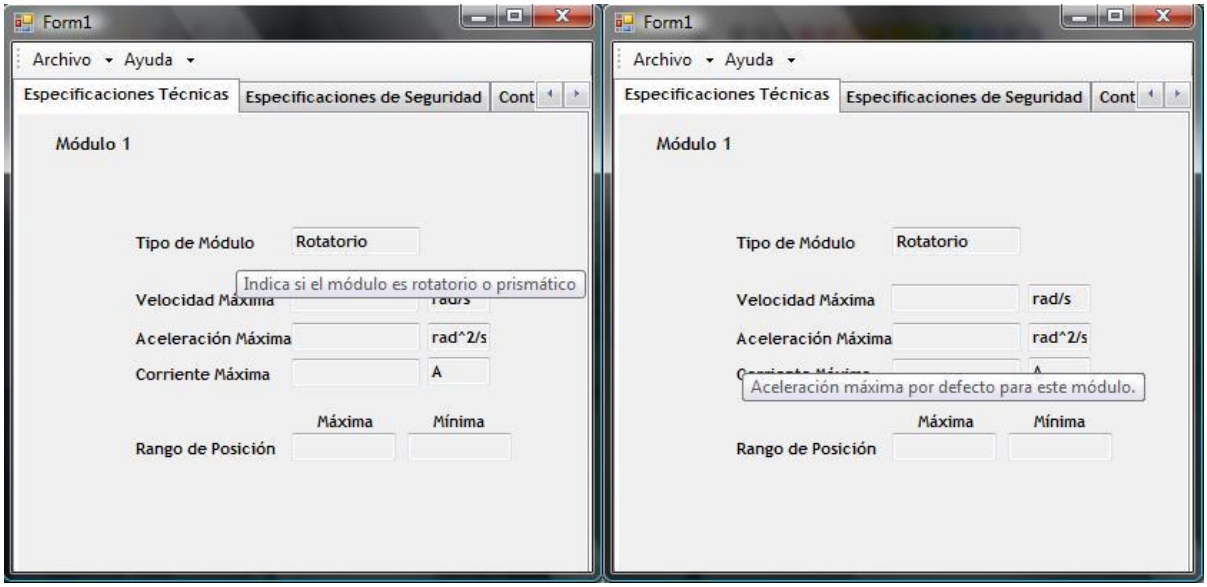
Ahora, observemos que en la parte superior de este Form tenemos una barra de menú desplegable con dos secciones, Archivo y Ayuda.

Si presionamos en el primer de los apartados, nos aparece la opción “Cerrar” que cierra por completo toda la aplicación y las ventanas que ésta hay podido abrir durante la ejecución.

El desplegable de Ayuda nos ofrece la posibilidad de acceder a un Glosario de Términos, en el que se explican todos los parámetros que aparecen en las distintas pestañas de esta aplicación.



Existe, además de esta opción de ayuda, otra ayuda adicional, que resulta de situar el ratón encima de los elementos de la pantalla y que consiste en un cuadro de texto emergente con una definición breve y concisa de cuál es la función o significado del control señalado.



2. Form2.

Como se comentó en la introducción de esta memoria, la intención, en un principio, era que el formulario anterior se comunicase de forma directa con el robot con el que se ha estado trabajando, sin embargo, debido a la falta de información y material para trabajar con CAN, se planteó opción de crear un segundo formulario, Form2, el cual simula ser el robot y enviar datos al formulario.

Desde un principio se mantuvo la idea de que este proyecto sólo desarrollaría la interfaz que monitorizaría el comportamiento del robot, de forma que la comunicación y el envío de información desde la interfaz al robot no se tratarían en este proyecto.

Establecidos los límites de trabajo de este segundo formulario, se va a proceder a desarrollar su contenido y funcionalidad.

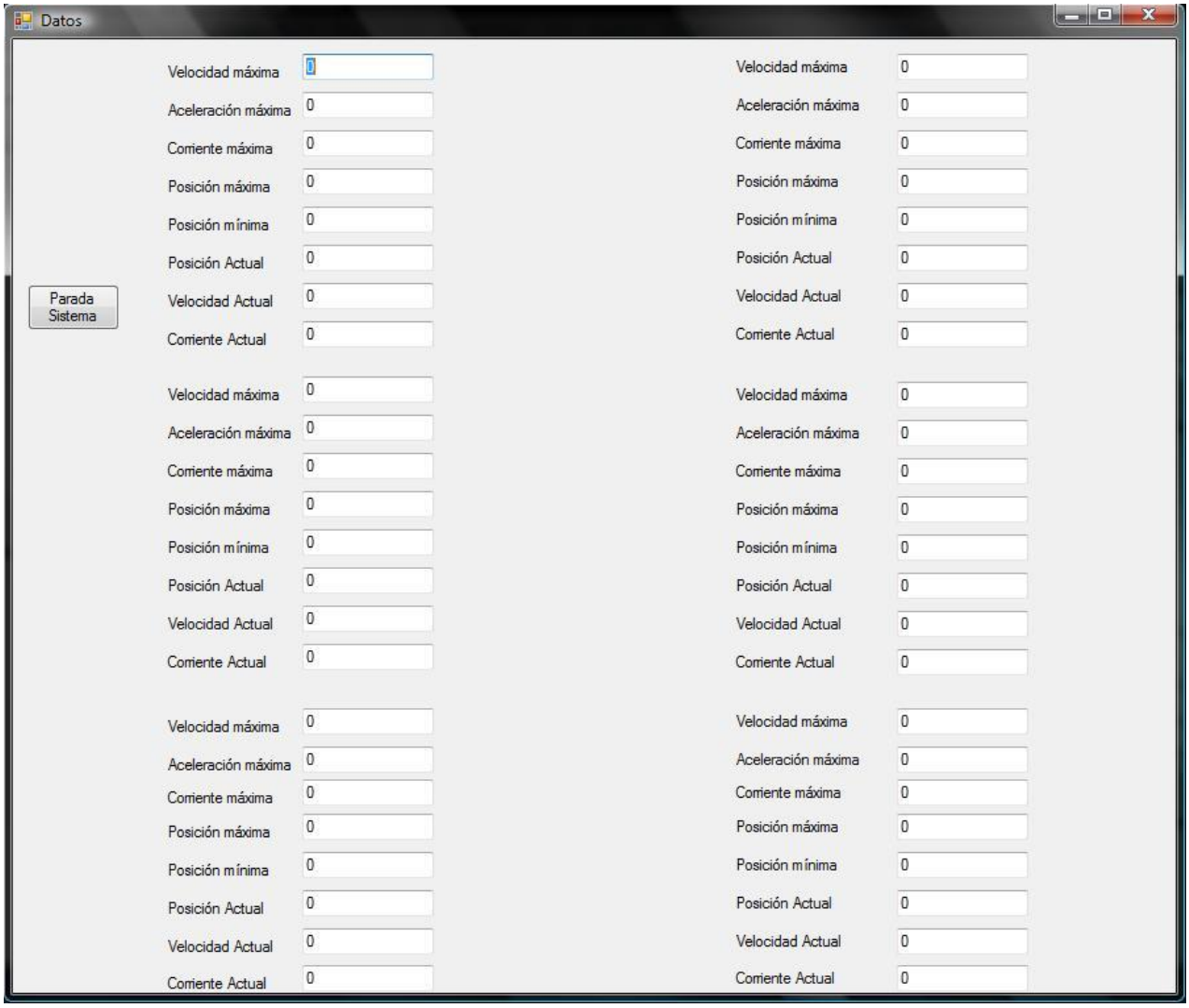
En primer lugar, debemos volver al Form1, a la pestaña de Especificaciones Técnicas. Ya se señaló que en esta ventana existían dos botones, Datos y Actualizar, pero no se explicó la funcionalidad de éstos.

Por una parte, el botón Datos abre el formulario secundario. Este es un formulario no modal, lo que significa que se abre pero podemos seguir trabajando sobre el primero mientras el Form2 permanece en el fondo de la pantalla.

En este formulario, de nuevo, vuelven a aparecer todas las variables ya descritas, todas. La finalidad es que a través de éste, puedan modificarse los datos, como si del propio robot se tratase, y éstos puedan mostrarse en nuestra aplicación. Además de las variables, contiene un botón de Parada de Emergencia que viene a representar el botón real que viene con el equipo robótico.

La pantalla es muy simple, sólo contiene los TextBox asociados a cada variable, donde se pueden escribir los valores que se desee enviar a la ventana principal. Estos TextBox están inicializados con el valor 0 y programados para que en el momento que se cambie el valor de uno de ellos, se guarde y se transmita esta información al Form1. Además, sólo admiten valores numéricos, de modo que si se introduce cualquier otro tipo de caracteres, la aplicación genera un error al intentar la conversión del carácter a número.

La apariencia de este formulario denominado con el nombre “Datos” es la siguiente:



De nuevo, las variables separadas en seis regiones según los módulos, con una etiqueta para indicar la variable de que se trata y un TextBox con el valor que adoptan.

Por su parte, el botón “Actualizar” realiza una actualización manual de los valores de la pantalla Especificaciones Técnicas. Dicho botón está programado para pasar los valores de un Form a otro y es el único método de hacerlo en esta pestaña, ya que, como se ha explicado, ésta muestra únicamente los valores por defecto que, en teoría, permanecerán inalterados durante el funcionamiento del robot.

7. Conclusiones.

A forma de conclusión y como ya se comentó ya al comienzo de la memoria, este proyecto se ha dividido en varias partes diferenciadas. Por una parte, el estudio cinemático realizado al robot, en el cual se ha determinado el área de trabajo del mismo, así como sus parámetros de Denavit-Hartenberg y la resolución del problema cinemático directo a través de estos; la simulación del posicionamiento del robot gracias a las herramientas de trabajo proporcionadas por el fabricante, así como por la aplicación para MATLAB, que permite la visualización directa del robot; y el desarrollo de una aplicación en C++ para realizar una simulación de cómo sería la comunicación con el robot.

En el transcurso del desarrollo de este proyecto, he podido poner en práctica los conceptos aprendidos en la asignatura de Robótica de 3º de Electrónica. Este trabajo me ha permitido encontrar el significado práctico de los procesos matemáticos de caracterización de robots y entender la necesidad de éstos para el correcto control de los sistemas robóticos en entornos industriales. Los problemas de la cinemática directa, así como el estudio del funcionamiento de las partes mecánicas del sistema (velocidad, posición o corriente en el motor, librerías de control) han sido algunas de las partes estudiadas en mayor profundidad, completando el proyecto con la simulación de la comunicación con el robot al no haberse podido llevar a cabo la comunicación directa desde el PC-Maestro con el Bus CAN y la aplicación desarrollada en C++, lenguaje de programación estudiado muy superficialmente en algunas de las asignaturas relacionadas con la informática de la carrera.

Por otra parte, el principal problema que se me ha presentado durante el desarrollo de mi trabajo ha sido la escasez de bibliografía referente a la programación de formularios en C++. Concretamente, mi problema surgió al intentar comunicar ambos formularios, pues mis conocimientos sobre C++ no eran demasiado amplios y no existe demasiada información referida a este tema. En internet, fácilmente es posible encontrar numerosos ejemplos sobre cómo solventar el problema en el resto de los lenguajes con los que trabaja el programa Visual de forma bastante sencilla, sin embargo, cuando se trata de C++, la situación se complica bastante.

En conclusión, poder plasmar en la práctica, con un robot real, los conceptos teóricos estudiados a los largo de la carrera, ha resultado de gran ayuda para la comprensión y entendimiento de éstos, pues la teoría sin la práctica, suele resultar demasiado abstracta, sobre todo cuando se trata de teoría física o matemática, como sucede en este caso.

8. Bibliografía consultada.

Para el desarrollo de esta memoria, se han utilizado los manuales proporcionados por el fabricante de Robotnik Brazo Modular.

Libros:

"*Fundamentos de Robótica*", Antonio Barrientos.

"*Prácticas de robótica utilizando Matlab*", Satarén Pazmiño, Roque J.

"*Robótica manipuladores y robots móviles*", Ollero Baturone, Aníbal.

Sitios Web:

http://www.robotnik.es/es/productos/brazos-roboticos/brazo-modular

http://petercorke.com/Robotics_Toolbox.html

http://www.schunk-modular-robotics.com/left-navigation/service-robotics/modular-robotics.html

http://www.roboticspot.com/especial/historia/his2004a.php

http://www.cplusplus.com/doc/tutorial/

http://www.winprog.org/tutorial/start.html

http://148.202.12.20/~cin/robotic/tarease/dh/dh.htm

http://www.angelfire.com/extreme/greynosom/archivos/Cinematica_Robot.pdf

http://proton.ucting.udg.mx/robotica/r166/r81/r81.htm

http://icaro.eii.us.es/descargas/Tema4_parte3.pdf

http://omarsanchez.net/Documents/Cinem%C3%A1ticainversadelosmanipuladores.pdf

http://es.wikipedia.org/wiki/Bus_CAN

http://es.wikipedia.org/wiki/Inteligencia_artificial

ANEXO A: Funcionamiento del Bus CAN y sus herramientas de trabajo.

1. Protocolo de comunicaciones CAN.

1.1. Introducción.

El bus CAN (Controller Area Network) es un protocolo de comunicaciones serie desarrollado por la firma alemana Robert Bosch GmbH que soporta de forma eficiente el control distribuido en tiempo real con un muy alto nivel de seguridad.

CAN es un protocolo basado en mensajes que fue desarrollado inicialmente para aplicaciones específicas en los automóviles y por lo tanto la plataforma del protocolo es resultado de las necesidades existentes en el área de la automoción. En electrónica de automóviles, las unidades de control de motores, sensores y sistemas antideslizamiento, etc. son conectados mediante bus CAN con tasas de transferencia de bits de hasta 1 Mbit/sec. Al mismo tiempo que son efectivos para construir el cuerpo electrónico del interior del vehículo (lámparas, ventanas automáticas, etc.) reemplazando los sistemas de cableado requeridos de otra forma.

1.2. Conceptos básicos.

Como ya se ha mencionado, CAN es un protocolo basado en una topología bus para la transmisión de mensajes en entornos distribuidos que, además, ofrece una solución a la gestión de la comunicación entre múltiples CPUs.

El protocolo de comunicaciones CAN es un protocolo de comunicaciones normalizado, con lo que se simplifica y economiza la tarea de comunicar subsistemas de diferentes fabricantes sobre una red común o bus. El procesador anfitrión (host) delega la carga de comunicaciones a un periférico inteligente, por lo tanto el procesador anfitrión dispone de mayor tiempo para ejecutar sus propias tareas y, al tratarse de una red multiplexada, reduce considerablemente el cableado y elimina las conexiones punto a punto.

Además de éstas, CAN dispone de diferentes propiedades, entre las que se destacan:

- Priorización de mensajes.

- Garantía de tiempos de latencia, es decir, en el tiempo que tarda un paquete en viajar desde el origen hasta el destino.
- Flexibilidad en la configuración.
- Recepción por multidifusión (*multicast*) con sincronización, de modo que todos los nodos pueden acceder al bus de datos simultáneamente.
- Sistema robusto en cuanto a consistencia de datos.
- Sistema multimaestro.
- Detección y señalización de errores.
- Retransmisión automática de tramas erróneas
- Distinción entre errores temporales y fallas permanentes de los nodos de la red, así como desconexión autónoma de nodos defectuosos.

1.3. Capas del bus CAN.

De acuerdo con el modelo de referencia OSI, la arquitectura de capas de CAN es la siguiente: capa física, capa de enlace de datos y una capa adicional denominada supervisor.

1.3.1. Modelo de referencia OSI de ISO.

El modelo de referencia OSI surge debido a la necesidad de normalización y estandarización de protocolos de comunicación para poner, así, solución al problema del intercambio de datos entre los distintos equipos informáticos desarrollados por los diferentes fabricantes.

El objetivo de este modelo es, por tanto, permitir la comunicación entre aplicaciones que se ejecutan en diferentes equipos, sin importar las diferencias entre estos componentes físicos.

Para conseguir esto, el modelo de referencia OSI propone una arquitectura de capas que define el comportamiento del subsistema de comunicación de los equipos. Este modelo regula el comportamiento externo de cada capa, dando libertad al fabricante para su implementación interna.

1.3.1.1. Funcionamiento.

Cada capa del modelo desempeña una función determinada según un protocolo definido e intercambia datos con la capa de igual nivel del sistema remoto siguiendo unas normas:

- Cada capa ofrece una serie de servicios a la capa superior exclusivamente.
- Cada capa solicita servicios a la capa inmediatamente inferior.

Gracias a este mecanismo, la implementación de cada una de las capas se encuentra totalmente encapsulada. De este modo, cada fabricante puede realizar su propia implementación interna de las capas sin afectar al comportamiento externo, que será compatible con otras implementaciones.

1.3.1.2. Misión de las capas.

El modelo OSI se encuentra definido por las siguientes capas:

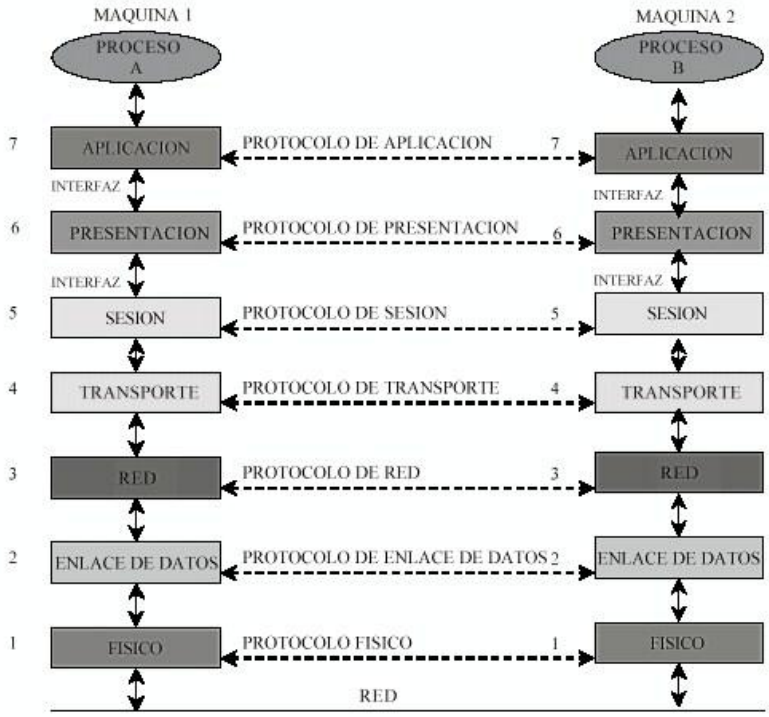


Fig.1.1.- Capas, protocolos e interfaces del modelo OSI

- Capa física (1): detalla las especificaciones eléctricas y mecánicas para garantizar la transmisión del flujo de bits serie entre equipos.
- Capa de enlace (2): realiza operaciones de detección y corrección de errores, garantizando la fiabilidad de la transmisión.
- Capa de red (3): responsable del encaminamiento de paquetes a través de la red de nodos intermedios.
- Capa de transporte (4): provee de una comunicación fiable extremo a extremo, proporcionando una sesión transparente y libre de errores.
- Capa de sesión (5): establece los mecanismos necesarios para el control y la sincronización de conversaciones entre equipos de distintos sistemas.
- Capa de presentación (6): se ocupa de la representación de los datos durante la transferencia entre aplicaciones.
- Capa de aplicación (7): proporciona servicios y funciones para que los procesos de aplicación tengan acceso al entorno OSI y se puedan comunicar con otras aplicaciones.

1.3.1.3. Arquitectura de capas de CAN.

En relación al modelo OSI explicado anteriormente, tenemos que, para el caso del bus CAN, sólo se implementan los niveles uno y dos, es decir, "Nivel Físico" y "Nivel de Enlace de Datos", aunque también consta de una capa adicional encargada de la gestión y control del nodo, denominada capa supervisor, y, en algunos casos, también posee una capa de aplicación.

1.3.1.3.1. Capa física.

La capa física es responsable de la transferencia de bits entre los distintos módulos que componen la red. Define aspectos como niveles de señal, codificación, sincronización y tiempos en que los bits se transfieren al bus.

En la especificación original de CAN, la capa física no fue definida, permitiendo diferentes opciones para la elección del tipo de utilidad y niveles eléctricos de transmisión. Las características de las señales eléctricas en el bus fueron establecidas más tarde por el estándar ISO 11898.

La especificación CiA (CAN in AUTOMATION) complementó las definiciones respecto al medio físico y conectores. Así, se determinó para CAN el uso de dos cables (CAN_H y CAN_L) o par diferencial o trenzado, aunque también existen versiones para un cable único. En el caso de empleo de par trenzado, la capa física más empleada determina que los módulos conectados al bus interpreten dos niveles lógicos denominados:

a) Dominante: la tensión diferencial (CAN_H - CAN_L) es del orden de 2'0 V con CAN_H = 3'5V y CAN_L = 1'5V (nominales).
b) Recesivo: la tensión diferencial (CAN_H - CAN_L) es del orden de 0V con CAN_H = CAN_L = 2'5V (nominales).

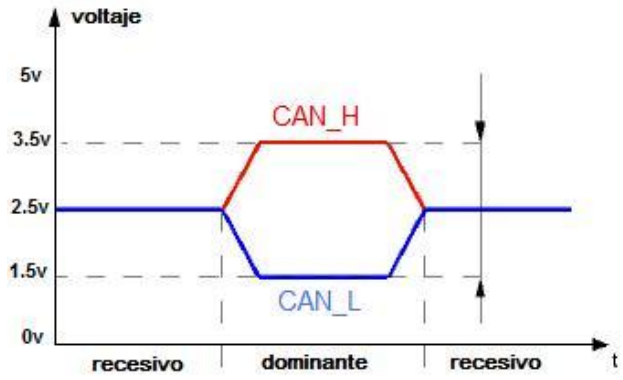


Figura 1.2. – Niveles de tensión para CAN.

Otro aspecto importante de esta comunicación es la velocidad con la que se transmiten los mensajes a través de la red. Lo normal es tener buses de corta longitud para así conseguir un funcionamiento óptimo, pero, si por el contrario, tenemos redes de largas longitudes, esto irá mermando nuestra velocidad de transmisión debido a los retardos en la línea, impedancias, tolerancias de los osciladores, etc. Para atenuar estas carencias se colocan en los extremos del bus impedancias de carga para una mayor estabilidad.

1.3.1.3.2. Capa de enlace de datos.

Esta capa define las tareas independientes del método de acceso al medio. Además, debido a que una red CAN brinda soporte para procesamiento en tiempo real a todos los sistemas que la integran, el intercambio de mensajes que demanda dicho procesamiento requiere de un sistema de transmisión a frecuencias altas y retrasos mínimos. En redes multimaestro, la técnica de acceso al medio es muy importante ya que todo nodo activo tiene los derechos para controlar la red y acaparar los recursos. Por lo tanto, la capa de enlace de datos define el método de acceso al medio así como los tipos de tramas para el envío de mensajes.

La capa de enlace de datos se encuentra dividida a su vez en dos subcapas: por una parte, la capa de Control de Enlace Lógico (LLC), y por otra, la capa de Control de Acceso al Medio (MAC).

1.3.1.3.3. Capa de enlace lógico (LLC).

La capa de enlace lógico tiene diferentes funciones, entre las que se destacan:

- Filtración de mensajes.
- Proporción de servicios durante la transferencia de datos y durante la petición de datos remotos.
- Proporción de medios para el restablecimiento y notificación de la sobrecarga del bus.
- Aceptación de los mensajes recibidos de la capa MAC.

1.3.1.3.4. Capa de control del acceso al medio (MAC).

La capa de control de acceso representa el núcleo del protocolo CAN. Entre las distintas funciones que desempeña esta capa se encuentran:

- Representación de los mensajes recibidos a la subcapa LLC y aceptación de éstos para ser transmitidos a dicha subcapa.
- Es responsable de la trama de mensajes, arbitraje, reconocimiento, detección de errores y señalización.
- En esta subcapa se decide si el bus está libre para comenzar una nueva transmisión o si la recepción acaba de comenzar.

1.3.1.3.5. Capa supervisor.

La sustitución del cableado convencional por un sistema de bus serie presenta el problema de que un nodo defectuoso puede bloquear el funcionamiento del sistema completo. Cada nodo activo transmite una bandera de error cuando detecta algún tipo de

fallo y puede ocasionar que un nodo defectuoso pueda acaparar el medio físico. Para eliminar este riesgo, el protocolo CAN define un mecanismo autónomo para detectar y desconectar un nodo defectuoso del bus; dicho mecanismo se conoce como aislamiento de fallos.

1.3.1.3.6. Capa de aplicación.

Existen diferentes estándares que definen la capa de aplicación; algunos son muy específicos y están relacionados con sus campos de aplicación.

<table>
<tr><th>Especificación</th><th colspan="2">Capa OSI</th></tr>
<tr><td>Especificado por el diseñador del sistema</td><td colspan="2">Capa de Aplicación</td></tr>
<tr><td rowspan="6">Especificación Protocolo CAN

ISO 11898</td><td rowspan="2">Capa de Enlace de Datos</td><td>Enlace de Control Lógico</td></tr>
<tr><td>Acceso Contro del Medio</td></tr>
<tr><td rowspan="3">Capa Física</td><td>Señalización Física</td></tr>
<tr><td>Acceso Medio Físico</td></tr>
<tr><td>Interfaz dependiente del medio</td></tr>
<tr><td colspan="2">Medio de transmisión</td></tr>
</table>

Fig.1.3.- Arquitectura de capas CAN

1.4. Transferencia de mensajes.

Los mensajes son una sucesión de ‘0’ y ‘1’ que están representados por distintos niveles de tensión en los cables del CAN Bus y que se llaman “bit”. CAN utiliza mensajes de estructura predefinida para la gestión de la comunicación llamados “tramas” (*frame*).

1.4.1. Formatos de las tramas.

Existen dos tipos de formatos diferentes, los cuales difieren en la longitud del campo identificador. Las tramas con una longitud de 11 bits para el identificador son denominadas

como tramas estándares (*standard frame*) definidas en la especificación CAN 2.0A, mientras que aquellas con 29 bits para el campo del identificador son llamadas tramas extendidas (*extended frame*) y están definidas en la especificación CAN 2.0B.

1.4.2. Tipos de tramas.

La transferencia de mensajes es manifestada y controlada por cuatro tipos de trama diferentes: de datos (data frame), remota (remote frame), de error (error frame) y de sobrecarga (overload frame).

Tanto la trama de datos como la remota pueden ser utilizadas en ambos formatos, estándar y extendido. Asimismo, ambas se separan de tramas precedentes mediante una trama especial denominada “espacio entre tramas”.

1.4.2.1. Trama de datos.

La trama de datos es la más común, y es aquella que transmite información de un nodo a cualquiera de los restantes. La trama de datos está formada por campos que proporcionan información adicional sobre los mensajes definidos en CAN:

- Campo de inicio de trama (SOF): marca el inicio de la trama de datos, y consiste únicamente en un bit ‘dominante’.
- Campo de arbitraje: se utiliza para priorizar los mensajes en el bus. Puede estar formado por 12 o 32 bits
- Campo de control: formado por 6 bits. El formato de este campo varía dependiendo del tipo de identificador que se emplee (estándar o extendido). Este campo contiene el DLC (longitud del código de datos).
- Campo de datos: el número de bytes del campo de datos viene definido por los cuatro bits pertenecientes al DLC (longitud del código de datos) del campo de control. El campo de datos contiene la información que va a ser transmitida dentro de la trama de datos
- Campo de CRC: se trata de15 bits y un delimitador CRC (código de redundancia cíclica) que es utilizado por los receptores para detección de errores de transmisión.
- Campo de confirmación (ACK). El nodo receptor indica recepción correcta del mensaje, poniendo un bit dominante en el flag ACK de la trama.
- Campo de final de trama (EOF).

En la siguiente figura se muestra la descomposición en los diferentes campos de la trama de datos para los formatos extendido y estándar:

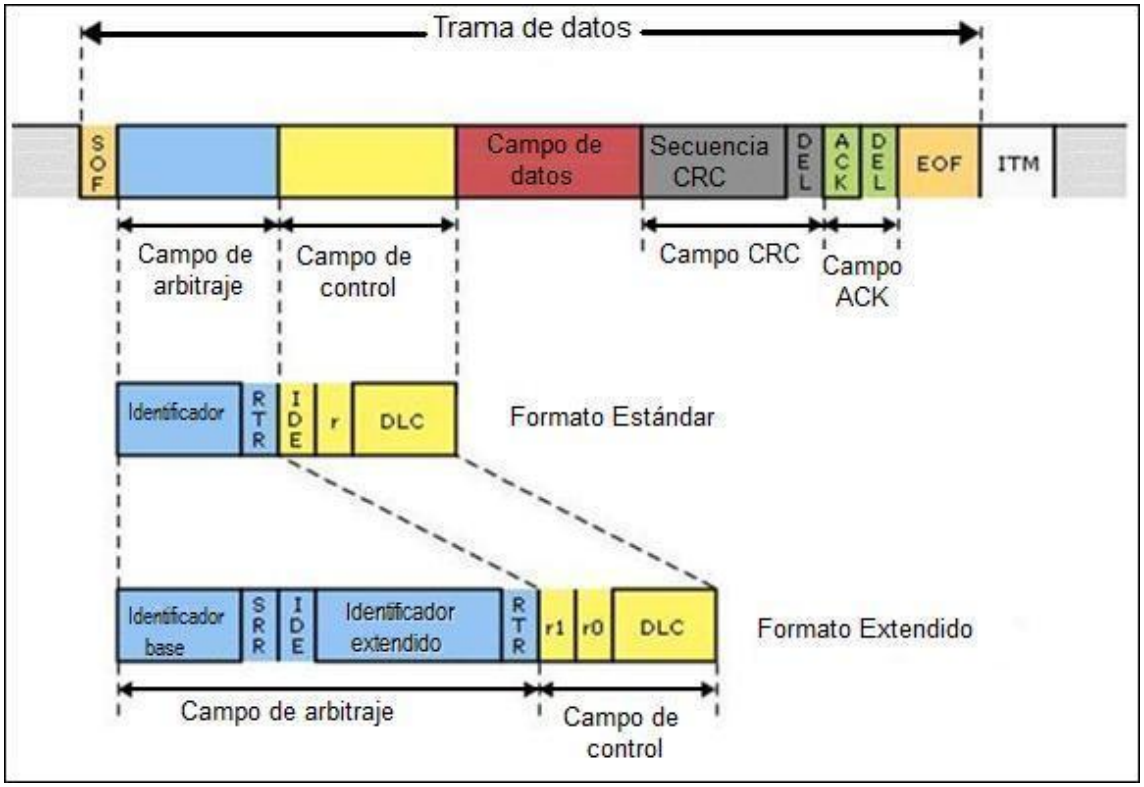


Fig. 1.4. –Trama de datos para formato estándar y formato extendido.

Como se observa en la figura, existen ciertas diferencias en la trama de datos para formato estándar y formato extendido que se van a proceder a comentar:

1. La primera diferencia aparece en el campo de arbitraje. En la trama estándar tenemos los 11 bits de identificación y un bit RTR; mientras que en la trama extendida tenemos los 29 bits de identificación, 1 bit para definir el mensaje como trama extendida, un bit SRR no usado y un bit RTR.

En el formato estándar, los 11 bits de identificador se corresponden con el identificador base (*base* ID) del formato extendido. Estos bits son transmitidos en orden desde ID-28 a ID-18, con ID-18 como el bit menos significativo.

En el formato extendido, por el contrario, el identificador está constituido por dos secciones, el identificador base (*Base ID*), con 11 bits distribuidos de igual forma que en el formato estándar (ID-28 a18), y el identificador extendido (*extended ID*) que consiste en 18 bits (ID-17 a 0).

También se aprecia que el bit IDE (*Identifier Extension Bit*) pertenece al campo de arbitraje en el caso del formato extendido, mientras que en el estándar se encuentra integrado en el campo de control.

2. La segunda diferencia se encuentra en el campo de control. Este campo se constituye por 6 bits diferentemente distribuidos para cada formato. Como ya se ha mencionado, en el formato estándar se encuentra incluido el bit de IDE además de un bit reservado r0. En el formato extendido, no se encuentra el bit de IDE, sino que en su lugar aparece un segundo bit reservado r1. El resto del campo está ocupado por el DLC (longitud del código de datos), que indica el número de bits del campo de de datos.

1.4.2.2. Trama remota.

Una estación que actúa como receptora de datos puede iniciar una transmisión de los datos respectivos a su nodo a través del envío de una trama remota. Esta trama existe para ambos formatos de transmisión de datos, estándar y extendido. En ambos casos está compuesto por seis campos diferentes, a saber: campo de inicio de trama, campo de arbitraje, campo de control, campo CRC, campo de reconocimiento y campo de fin de trama.

En la figura se muestra la distribución de la trama remota:

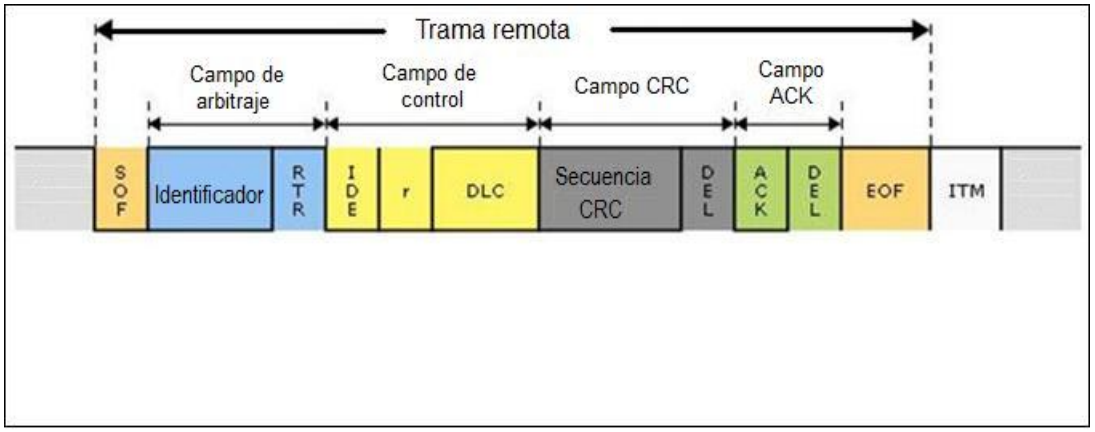


Fig.1.5.- Trama Remota.

Al contrario que en el caso de la trama de datos, en este tipo de trama no existe campo de datos, independientemende del valor del código DLC, el cual puede tomar cualquier valor entre cero y ocho. Este campo pertenece únicamente a la trama de datos. Para identificar si se trata de una trama de datos o remota se emplea el bit RTR que tomará un valor dominante para indicar que es una trama de datos y valor recesivo para trama remota.

1.4.2.3. Trama de error.

La trama de error está dividida en dos campos diferentes. El primer campo viene dado por la superposición de banderas de error y el segundo pertenece al delimitador de errores. En la siguiente figura se muestra su distribución:

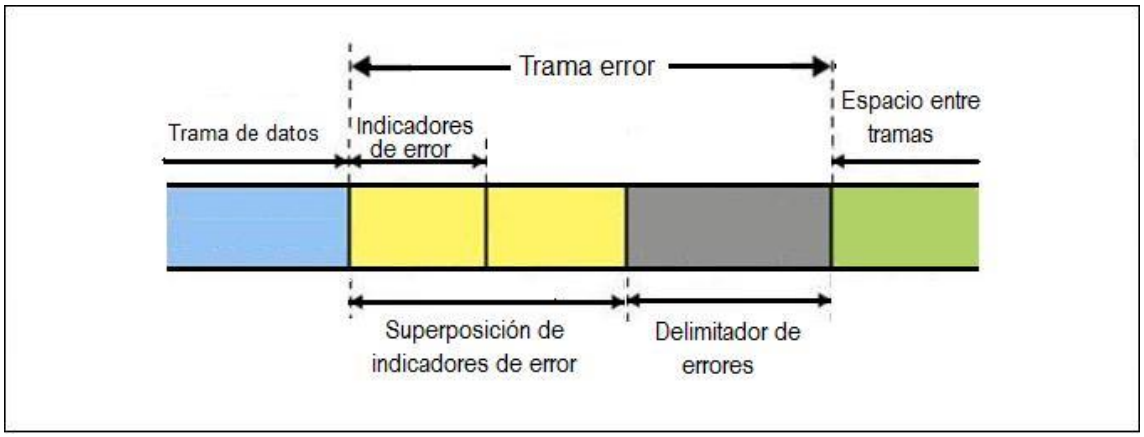


Fig. 1.6.- Trama de error.

Indicadores de Error: existen dos tipos de indicador de errores, el indicador activo y pasivo.

a) Indicador de error activo: consiste en seis bits consecutivos con valor dominante.

b) Indicador de error pasivo: consiste en seis bits consecutivos con valor recesivo, a menos que se sobreescriban por bits dominantes de otros nodos.

Una estación en error activo, detectando una condición de error, la indica transmitiendo un indicador de error activo. La forma de los indicadores de Error violan la ley del bit de relleno aplicada a todos los campos desde el Inicio de Trama (SOF) hasta el delimitador CRC o destruye el formato fijo de los campos ACK o Fin de Trama. En consecuencia, todas las demás estaciones detectan una condición de error y comienzan la transmisión de un indicador de error. Entonces, la secuencia de bits dominantes resulta ser la superposición de los diferentes indicadores de error transmitidos por las estaciones de forma individual. La longitud total de esta secuencia varía entre un mínimo de seis y un máximo de doce bits.

Una estación en error pasivo detectando una condición de error, trata de indicarlo transmitiendo un indicador de error pasivo. La estación en error pasivo espera seis bits consecutivos de igual polaridad, comenzando al inicio del Indicador de Error Pasivo. El indicador de error pasivo se completa cuando estos seis bits iguales han sido detectados.

Delimitador de Error: Consiste en ocho bits de valor recesivo.

Después de la transmisión de un Indicador de Error, cada estación envía bits recesivos y monitorea el bus hasta que detecta un bit recesivo. Después comienza a transmitir siete bits recesivos más.

1.4.2.4. Trama de sobrecarga.

La trama de sobrecarga contiene dos campos de bits: indicación de sobrecarga y delimitador de sobrecarga:

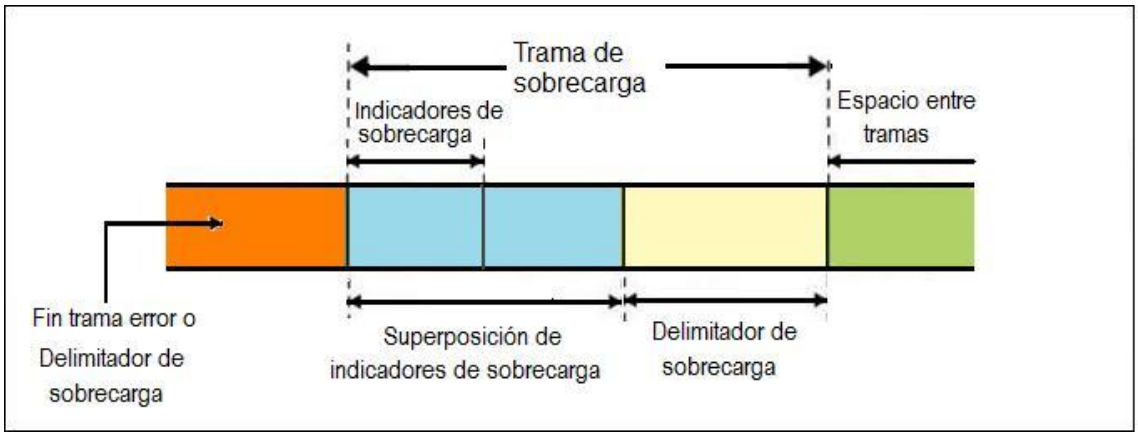


Fig. 1.7. – Trama de sobrecarga

Existen tres tipos de condición de sobrecarga:

a) La condición interna del receptor, el cual requiere un retardo de la próxima Trama de Datos o Trama Remota.

b) Detección de un bit dominante en el primer y segundo bit de intermisión.

c) Si un nodo CAN detecta un bit dominante en el octavo bit (el último bit) de un Delimitador de Error o un Delimitador de Sobrecarga, comenzará la transmisión de un cuadro de Sobrecarga (no una Trama de Error). El contador de Error no se verá incrementado.

El comienzo de una Trama de Sobrecarga debido a la condición a) de sobrecarga está solamente autorizado en el primer tiempo de bit de la intermisión esperada, mientras que la Trama de Sobrecarga por la condición b) y condición c) empiezan un bit después de ser detectado el bit dominante.

Indicador de sobrecarga: Se trata de seis bits dominantes. Su estructura coincide con la del indicador de error activo.

La forma del campo de indicador de sobrecarga destruye la disposición del campo de intermisión. Como consecuencia, el resto de las estaciones detectan una condición de sobrecarga e inician la transmisión del indicador correspondiente. En caso de que haya un bit

dominante detectado en el tercer bit de la intermisión, éste se interpretará como el inicio de trama.

Delimitador de Sobrecarga: Consiste en ocho bits recesivos. El Delimitador de Sobrecarga tiene el mismo formato que el Delimitador de Error. Después de la transmisión de un Indicador de Sobrecarga, la estación monitoriza el bus hasta que detecta una transición de bit dominante a recesivo. En ese momento cada estación del bus ha terminado de enviar su Indicador de Sobrecarga y todas las estaciones comienzan la transmisión de seis bits recesivos más en coincidencia.

1.4.2.5. Espaciado entre tramas.

Las tramas de datos y remotas se encuentran separadas de las tramas precedentes, sea cual sea el tipo de las mismas, por un campo de bits denominado espacio entre tramas. En contraste, las tramas de sobrecarga y error no están separadas por un espacio entre trama, ni si quiera en el caso de múltiples tramas de sobrecarga.

El espacio entre tramas contiene los campos de intermisión y bus libre y, para las estaciones de " error pasivo" que han sido transmisoras del mensaje último mensaje, suspensión de la transmisión.

Para estaciones no definidas como de error pasivo o que han sido receptoras:

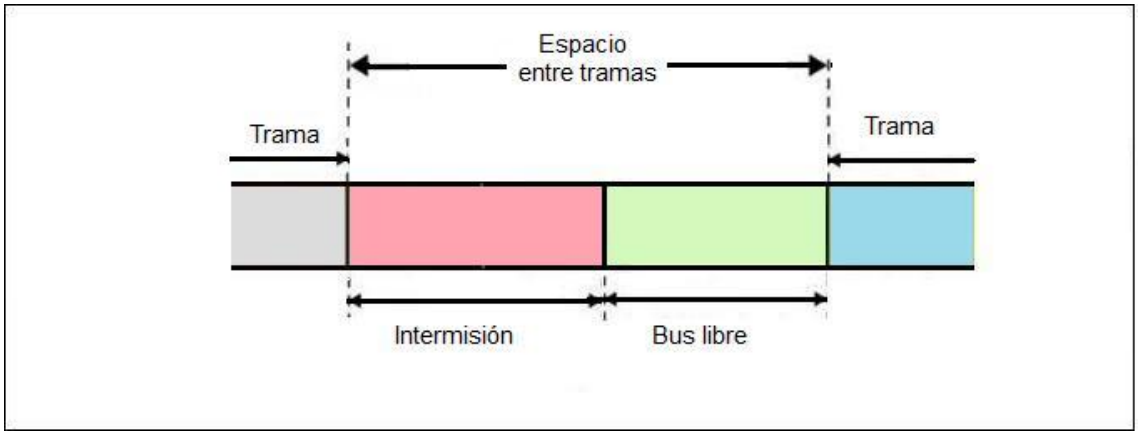


Figura 1.8.- Espacio entre tramas, caso a).

Para estaciones de error pasivo que han sido transmisoras, tenemos:

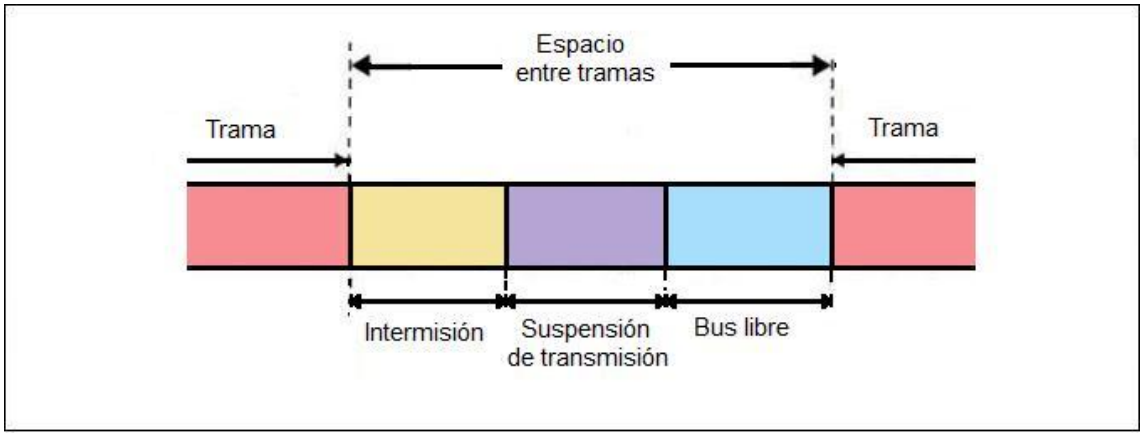


Figura 1.9.- Espacio entre tramas, caso b).

Intermisión: Consiste en 3 bits recesivos. Durante la intermisión no está permitido que ninguna estación comience la transmisión de una trama de datos o de una trama remota. La única acción que se debe tomar es señalar una condición de sobrecarga.

Bus libre: El período de bus libre puede ser de una longitud arbitraria. El bus se identifica como libre y cualquier estación que tenga algo para transmitir puede acceder al bus. Un mensaje que esté pendiente de transmisión durante la transmisión de otro mensaje comienza en el primer bit siguiente de intermisión.

Transmisión suspendida: Después de que una estación en estado de "error pasivo" transmita un mensaje, envía 8 bits recesivos a continuación de la intermisión, antes de comenzar a transmitir el futuro mensaje o haber reconocido el bus libre. Mientras comienza una transmisión (causada por otra estación) la estación se transformará en la receptora de este mensaje.

1.4.3. Codificación.

Los segmentos de trama: comienzo de tramas, campo de arbitraje, campo de control, campo de datos y secuencia CRC son codificados a través del método del bit de relleno. Cada vez que un transmisor detecta 5 bits consecutivos de igual valor en el paquete de bits que será transmitido, éste inserta automáticamente un bit complementario en el paquete de bits que realmente se está transmitiendo.

Los restantes campos de bits de la trama de datos y de la trama remota (delimitador, CRC, ACK, y fin de trama) tienen una estructura fija y no son codificados a través del método de bit de relleno.

El paquete de bits en un mensaje está codificado de acuerdo al método de no retorno a cero, esto significa que durante el total de tiempo de bit el nivel de bit ya era dominante o recesivo.

1.4.4. Manejo de errores.

Haciendo referencia a lo mencionado anteriormente, nos encontramos con la coexistencia de cinco tipos de error definidos para CAN, y tres estados de error posibles para la unidad, según el tipo y número de condiciones de detección de error.

1.4.4.1. Detección de errores.

A continuación se describen los posibles tipos de errores que define el protocolo de comunicaciones CAN:

a) Error CRC:

El valor de 15 bits de CRC es calculado por el nodo trasmisor y este valor de 15 bits es trasmitido al campo CRC. Todos los nodos de la red reciben este mensaje, calculan un CRC y verifican que los valores de CRC coinciden. Si el valor no coincide, entonces se produce un error de CRC y se genera una trama de error.

b) Error de ACK:

En el campo de reconocimiento de un mensaje, el nodo trasmisor comprueba si el indicador ACK, que ha sido enviado como bit recesivo, contiene un bit dominante. Este bit dominante reconocerá que al menos un nodo ha recibido correctamente el mensaje. Si este bit es recesivo, significa que ningún nodo ha recibido correctamente el mensaje. Por lo tanto se genera una trama de error y el mensaje original se repetirá después de pasar el apropiado tiempo de intermisión.

c) Error de forma:

Si cualquier nodo detecta un bit dominante en uno de los siguientes cuatro segmentos del mensaje: final de trama, espacio entre tramas, delimitador ACK o delimitador CRC, el protocolo CAN define esto como una violación de la forma y se genera un Error de forma. El mensaje original es reenviado después del correspondiente tiempo de intermisión.

d) Error de bit:

Ocurre un error de bit si un trasmisor envía un bit dominante y detecta un bit recesivo, o si envía un bit recesivo y detecta un bit dominante cuando monitoriza el nivel del bus actual y lo compara con el bit que acaba de enviar. Cuando el trasmisor envía un bit recesivo y se detecta un bit dominante durante el arbitraje o en el indicador ACK, no se genera bit de error ya que se está produciendo un arbitraje o ACK normal.

e) Error de relleno:

El protocolo CAN utiliza el método “No Retorno a Cero” (NRZ). Esto quiere decir que el nivel de bit toma lugar en el bus durante todo el tiempo de bit. CAN es asíncrono y utiliza un bit de relleno para permitir a los nodos receptores sincronizarse recuperando la información de la señal de reloj de la parte de la trama de datos. Los nodos receptores se sincronizan con la transición de recesivo a dominante. Si hay más de cinco bits de la misma polaridad en la fila, CAN automáticamente pondrá un bit de polaridad opuesta en la trama de datos (bit de relleno). El nodo receptor lo utilizará para sincronización, pero ignorará el bit de relleno para el propósito del dato. Si entre el comienzo de trama y el delimitador CRC, se detectan seis bits con la misma polaridad, el bit de relleno habrá sido violado de modo que se envía una trama de error y el mensaje se repite.

1.4.4.2. Error de confinamiento.

Con respecto al fallo de confinamiento, una unidad puede estar en uno de estos tres estados:

- Error Activo.
- Error Pasivo.
- Bus Apagado (*Bus-off*).

Una unidad en "error activo" puede tomar parte normalmente en la comunicación del bus y enviar un indicador de error activo cuando un error ha sido detectado.

Una unidad en "error pasivo" no debe enviar un indicador de error activo. La unidad toma parte en la comunicación del bus pero cuando un error ha sido detectado solo un indicador pasivo de error es enviado. Después de la transmisión, una unidad en error pasivo deberá esperar antes de iniciar una nueva comunicación.

Una unidad en bus apagado no está autorizada a tener influencia sobre el bus.

1.5. Descripción del sofware de comunicación.

En este apartado se va a describir tanto la interfaz C que incorpora CAN para interactuar con el controlador, como herramientas útiles que pueden ser empleadas para comprobar el correcto comportamiento de CAN de forma cómoda y directa.

1.5.1. Librería C de interfaz para DOS y Windows 3.11

Para el manejo de CAN, éste tiene integrado una interfaz C que realizará la comunicación con el driver controlador de CAN. Actualmente, la interfaz C está implementada para varios tipos de tarjeta CAN, si bien es cierto que existen ciertas restricciones dependiendo del tipo empleado aunque no se entrará en ese aspecto.

1.5.1.1. Introducción.

La librería C que se va a describir trabaja bajo DOS y Windows 3.11, por lo que resulta necesario tener una cierta noción del sistema operativo con que se trabaja.

DOS es una familia de sistemas operativos para PC. El nombre son las siglas de disk operating system ("sistema operativo de disco"). Fue creado originalmente para computadoras de la familia IBM PC e inicialmente contaba con una interfaz de línea de comandos en modo texto o alfanumérico, vía su propio intérprete de órdenes, command.com.

La más popular de sus variantes es la perteneciente a la familia MS-DOS, de Microsoft, suministrada con buena parte de los ordenadores compatibles con IBM PC como sistema operativo independiente o nativo, hasta la versión 6.22 (bien entrados los 90), frecuentemente adjunto a una versión de la interfaz gráfica Ms Windows de 16 bits, como las 3.1x.

En las versiones nativas de Microsoft Windows, basadas en NT (y éste a su vez en OS/2 2.x) (véase Windows NT, 2000, 2003, XP o Vista) MS-DOS desaparece como sistema operativo (propiamente dicho) y entorno base, desde el que se arrancaba el equipo y sus procesos básicos y se procedía a ejecutar y cargar la inferfaz gráfica o entorno operativo de Windows. Todo vestigio del mismo queda relegado, en tales versiones, a la existencia de un simple

intérprete de comandos, denominado Símbolo del Sistema, ejecutado como aplicación mediante cmd.exe, a partir del propio entorno gráfico

1.5.1.2. Funciones de la interfaz de programación.

A continuación se describen con detalle las funciones definidas en la librería que actúa de interfaz con el controlador CAN.

<table>
<tr><td colspan="2">Inicialización</td></tr>
<tr><td>Función</td><td>Descripción</td></tr>
<tr><td>canHwInit()</td><td>Esta función debe ser llamada al principio como la primera función de la librería.</td></tr>
<tr><td>canEnableIrq()</td><td>Por medio de esta función, se establece el nivel de interrupción del módulo CAN.</td></tr>
<tr><td>canDisabkeIrq()</td><td>A través de esta función, el nivel de interrupción previamente programado queda deshabilitado. Esta función solo debe ser llamada si la función canEnableIrq() fue llamada con éxito.</td></tr>
<tr><td>canInit()</td><td>Inicializa la interfaz de CAN y establece la velocidad de transmisión para la red según la siguiente tabla:
<table>
<tr><td>baudios [HEX]</td><td>kbits/s</td></tr>
<tr><td>0</td><td>1000</td></tr>
<tr><td>1</td><td>666.6</td></tr>
<tr><td>2</td><td>500</td></tr>
<tr><td>3</td><td>333.3</td></tr>
<tr><td>4</td><td>250</td></tr>
<tr><td>5</td><td>166</td></tr>
<tr><td>6</td><td>125</td></tr>
<tr><td>7</td><td>100</td></tr>
<tr><td>8</td><td>66.6</td></tr>
<tr><td>9</td><td>50</td></tr>
<tr><td>A</td><td>33.3</td></tr>
<tr><td>B</td><td>20</td></tr>
<tr><td>C</td><td>12.5</td></tr>
<tr><td>D</td><td>10</td></tr>
</table></td></tr>
<tr><td>canInitEx()</td><td>Inicializa la interfaz de CAN y se establece la velocidad de transmisión a través de una variable específica, bt0_bt1. En esta variable, el contenido de los registros BTR0 y BTR1 del controlador de CAN se codifican según:
bt0_bt1= (BTR0 x 256) + BTR1.</td></tr>
</table>

Lectura y escritura de datos	
Función	**Descripción**
canEnableId()	A través de esta función se activan los identificadores Rx y RTR
canDisableId()	Desactiva el identificador para transferencias
canSetAcceptanceEx()	Con esta función se establece un filtro para identificadores de 29 bits.
canRead()	Se produce la lectura de la trama Rx del identificador.
canReadEx()	Se procede a la lectura de la trama Rx (formato extendido).
canWrite()	Las tramas Tx se transmiten.
canWriteEx()	Las tramas Tx se transmiten (formato extendido).

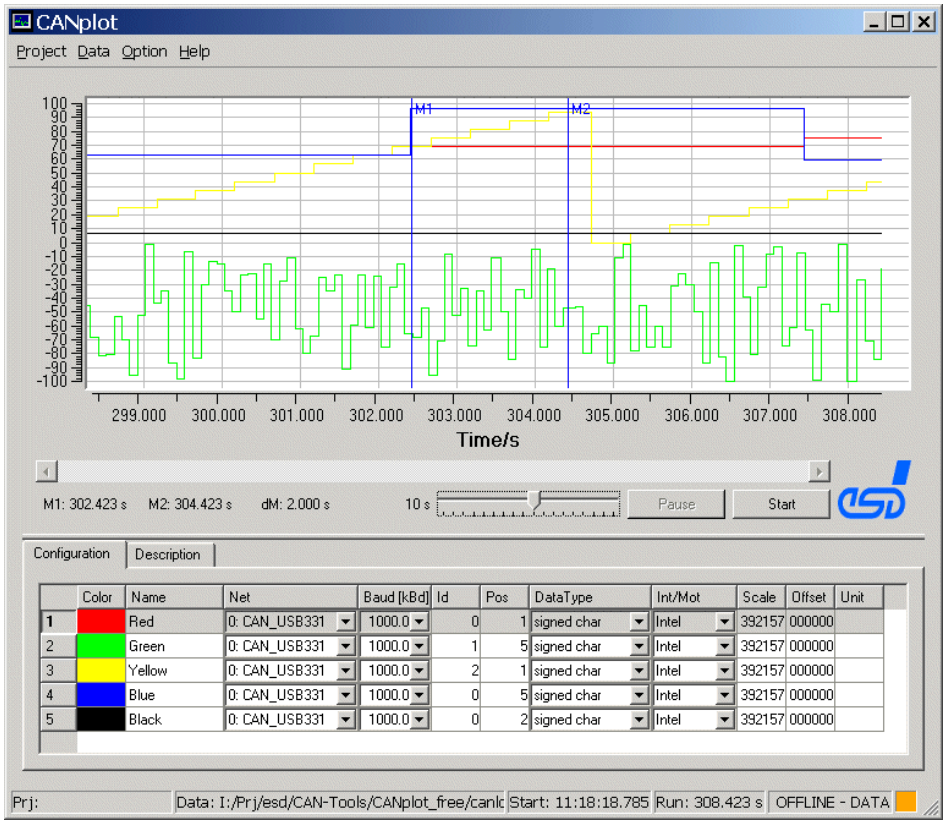
1.5.2. Algunas herramientas de trabajo para CAN.

1.5.2.1. CANplot.

Se trata de una herramienta para la representación cronológica del proceso de envío de datos a través de CAN. Concretamente, este programa ofrece los siguientes servicios:

- Representación de datos en forma de gráficos.
- Selección de bytes de datos para ser representados dentro de sus respectivas posiciones dentro del campo de datos.

Su interfaz resulta bastante sencilla de utilizar, ya que se presenta de forma muy intuitiva para el usuario:



1.5.2.2. CANscope.

CANscope es un programa de menú controlado empleado para monitorizar y probar redes CAN. Se trata de un dispositivo para registrar y evaluar los niveles de tensión del bus CAN. CANscope ofrece un módulo de grabación robusto y un programa software fácil de utilizar.

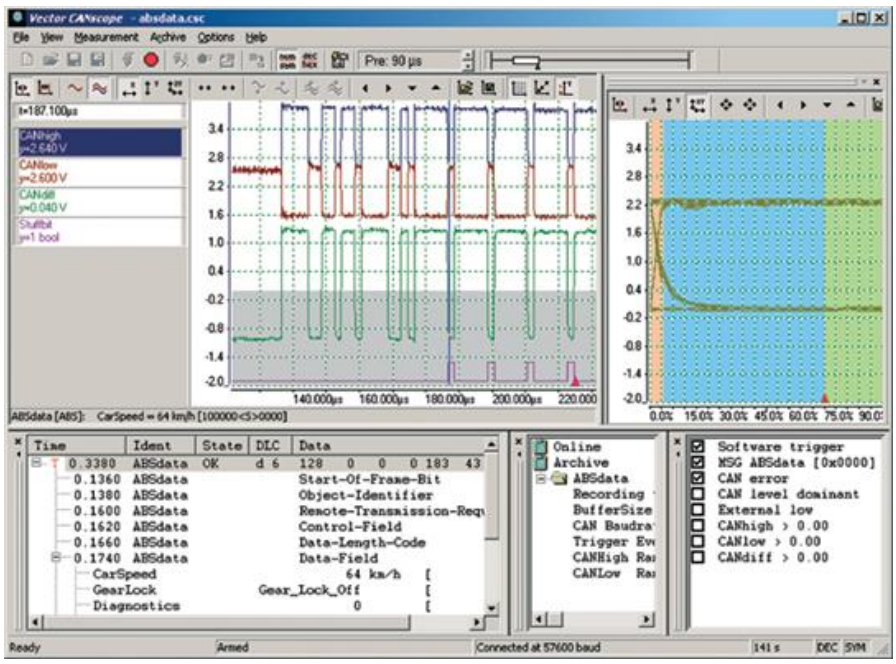
Esta aplicación permite al desarrollador analizar los efectos en:

- Varios tipos de cable.
- Varias longitudes de cable.
- Terminaciones del bus.
- Fallas.
- Influencias EMC (Compatibilidad Electromagnética).
- Errores en los controladores CAN.

Además, posee funciones que permiten al usuario trabajar con el bus CAN, tales como:

- Osciloscopio para muestrear el nivel del bus.
- Diagrama visual que permite evaluar la calidad de la señal.
- Diferentes vistas para comparar las curvas de tensión.

La ventana de trabajo es la que se muestra a continuación:

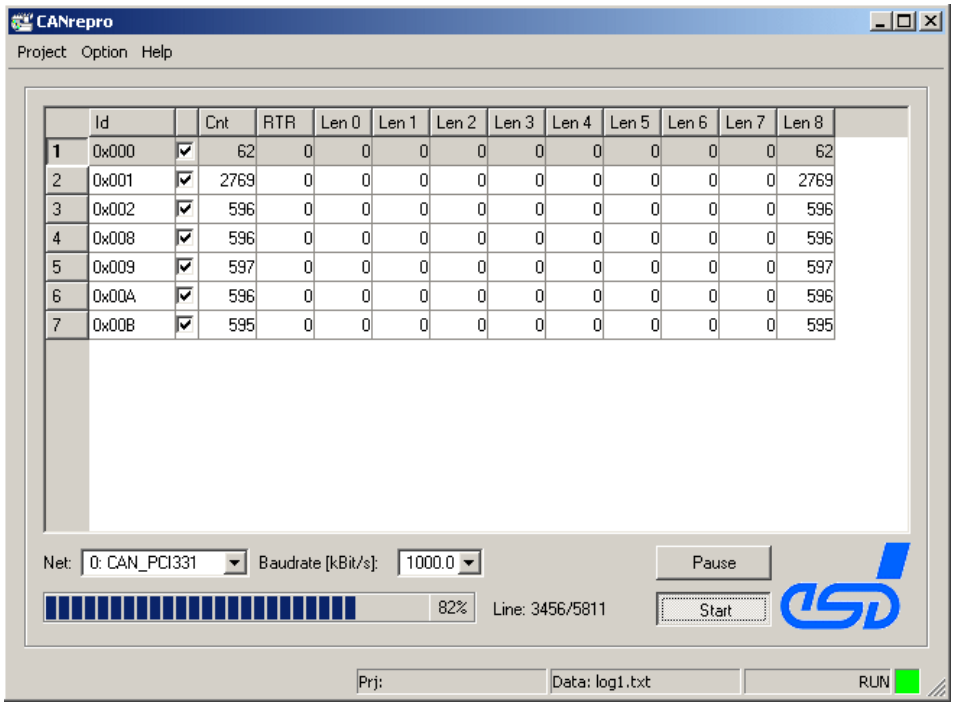


1.5.2.3. CANrepro.

Se trata de una herramienta para la representación de datos que han sido grabados previamente por el programa de seguimiento CANscope. Este programa ofrece varias posibilidades al usuario:

- Reproducción de tramas de mensajes grabadas previamente.
- Reproducción de datos con secuencia y temporización originales.
- Selección individual de datos.

La interfaz de esta herramienta puede parecer, a priori, algo más compleja que las anteriores, sin embargo, comentados los aspectos menos intuitivos, resulta muy sencillo la interactuación:



Algunos detalles importantes de la ventana de visualización

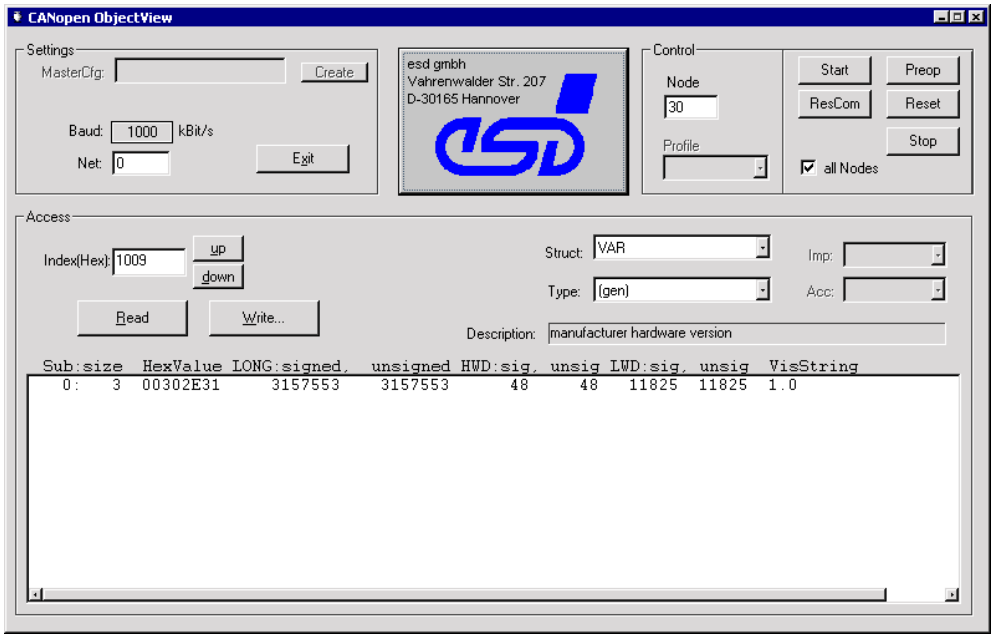
Las columnas más significativas indican:

- La segunda columna indica si la trama es estándar o extendida.
- Id(Hex): el identificador en valor hexadecimal.
- Id(Dec): el identificador en valor decimal.
- Checkbox: se marca si se pretende reproducir la trama correspondiente.
- Cnt: contador de tramas con el identificador correspondiente.

1.5.2.4. COBview.

Por último, esta herramienta permite al usuario analizar y diagnosticar los nodos conectados al bus CAN. Este programa ofrece:

- Detección y visualización de los dispositivos de CAN conectados a la red.
- Permite los accesos para lectura y escritura.


CANopen ObjectView
Settings
MasterCfg:
Create
Baud:
1000
kBit/s
Net:
0
Exit
esd gmbh
Vahrenwalder Str. 207
D-30165 Hannover
Control
Node
30
Profile
Start
Preop
ResCom
Reset
Stop
all Nodes
Access
Index(Hex):
1009
up
down
Read
Write...
Struct:
VAR
Type:
(gen)
Imp:
Acc:
Description:
manufacturer hardware version
Sub:size HexValue LONG:signed, unsigned HWD:sig, unsig LWD:sig, unsig VisString
0: 3 00302E31 3157553 3157553 48 48 11825 11825 1.0