ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Análisis del estado, configuración y puesta en marcha de un mecanismo de manipulación avanzado brazo-mano.

AUTOR: Guillermo Martínez-Fortún Martínez
DIRECTOR: José Luis Muñoz Lozano

2009

Índice

1.1.-Preámbulo

El tema de la robótica siempre me ha parecido muy interesante, creo que en la actualidad la evolución para este sector es bastante fuerte. Durante mi época de recopilar información conocí proyectos muy interesantes, como por ejemplo la NASA, para integrar robóticos gigantes en los transbordadores que tienen en el espacio, o los robots que imitan el movimiento humano. Es muy fácil comprobar cómo periódicos y revistas muestran una perspectiva del sector en pleno auge científico.

Fig. 1.1 - Muestra de un modelo japonés con gran parecido humano.

Como muestra la imagen de la figura 1.1, la ingeniería está invadiendo cada vez más sectores de la sociedad actual, parece como si estuviese interesada en sustituir en todas las tareas al hombre. Elegí esta foto en particular porque trata de un intento por parte de los japoneses de introducir elementos con alto parecido humano en la recepción de invitados.

Su finalidad actual es explicar los servicios que se ofrecen en la empresa, apoyado por una serie de movimientos de sus brazos que la hagan parecer más humana, o lo que es lo mismo agradable al trato con los clientes. Aunque ahora mismo su función no es la de un robot propiamente dicho, ya que esta más pensado más como un elemento que hace una tarea repetitiva, en un futuro se pueden implementar un movimiento de andar y hacer un robot que acompañe y ayude a personas mayores a realizar tareas diarias.

La robótica ahora mismo esta intentando introducirse en un mundo, por muchos sectores diferentes, como por ejemplo el comentado arriba, para el apoyo de tareas para personas con edad avanzada o con problemas de movilidad. Este problema se ve acentuado en una sociedad como Japón, un país que se encuentra con un problema de envejecimiento de una población y

la necesidad de encontrar una solución para el problema. Otro sector es el aeroespacial, es muy interesante poder encontrar herramientas dinámicas que sean capaces de trabajar en entornos muy extremos, sustituyendo la mano de obra humana. Otro sector que lo veo muy interesante, y que se está planteando en Estados Unidos es el de introducir robots para la recogida de alimentos en el campo, ya que actualmente escasea la mano de obra, y se encuentran con un problema creciente, de pérdidas de cosechas, y con el consiguiente coste de dinero. Creo que esto muestra un claro ejemplo de la importancia de la robótica, invertir en ella es futuro, ya que tiene la ventaja de poder adaptarse a muchas tareas diferentes.

Fig. 1.2.- Foto tomada del periódico el País, donde muestra la implantación en el transbordador espacial internacional por parte del laboratorio espacial japonés Kibo

Nosotros podíamos hablar de robótica enfocados a trabajos muy complejos como por ejemplo el citado arriba en el espacio como un caso muy interesante. En la actualidad nos podríamos encontrar tareas que no sean repetitivas, que necesiten mucha precisión, necesidad por parte del sistema de tener una capacidad de adaptarse a cambios imprevistos muy alta, o justamente en lo contrario. Creo que lo importante es pensar que la cantidad de posibles soluciones que tiene un problema está en la imaginación del investigador. Lo importante es seguir trabajando en ello.

Aunque he hablado generalmente de la robótica, me gustaría centrarme en el estado de las investigaciones sobre las manos robóticas. Estas manos imitan los movimientos humanos, teniendo bastante versatilidad y una capacidad de adaptarse a muchos procesos. Aunque actualmente la utilidad de una mano robótica es nula, y se tiene vista que aún queda mucho camino para introducirlas en algún sector en el que se pueda sacar rendimiento, creo que en un futuro serán una herramienta imprescindible. El motivo principal de su falta de utilidad es que disponemos de herramientas mucho más específicas que cumplen las tareas encomendadas con una calidad muy alta y con costes mucho menores. Pero quien sabe si en un futuro aparecen procesos mucho más complejos, y las máquinas actuales se quedan desfasadas, ese puede ser el momento de pensar en nuevas vías, como puede ser las manos robóticas.

Fig. 1.3.- Muestra de la capacidad de una mano robótica para realizar tareas de precisión

Creo que la última afirmación que hago en el párrafo anterior se puede demostrar con las experiencias que muestra la historia; inventos que no tienen una utilidad clara en una época, y que para la siguiente sean objetos imprescindibles. Durante mi estudio de los inventos a lo largo de la historia, me lleve una sorpresa al comprobar que la historia está llena de giros imprevistos, cosas que antes no tenían un uso claro, y han revolucionado siglos después la sociedad. Los ejemplos que más me han impresionado son la rueda, la brújula, y los autómatas programables que citaré a continuación.

La rueda es un ejemplo claro de invento revolucionario, en la sociedad actual es impensable vivir sin ella. Este invento produjo un cambio muy grande para el ser humano, cambio la forma de transportar objetos pesados. Este invento se puede considerar como ejemplo de lo que puede hacer una idea, generar un elemento muy potente y sencillo, y adaptarlo para convertirlo en un elemento básico y necesario para la sociedad.

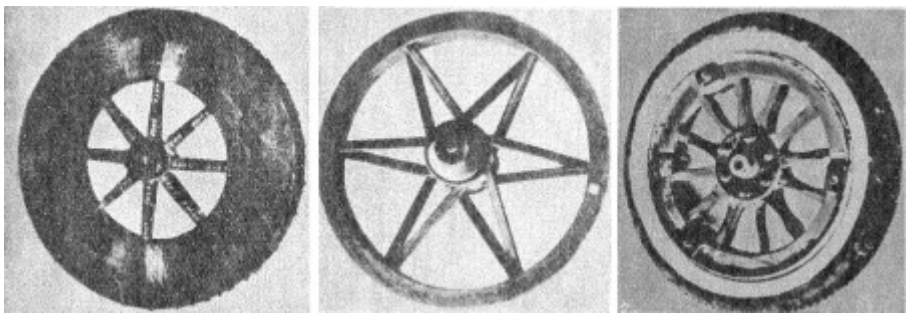


Fig. 1.4.- La rueda apareció alrededor del 3500 A.c. Los grabados muestran la evolución de una rueda del 700 A.c. (Asiria), la rueda acampanada de Leonardo y un neumático con clavos de comienzos del siglo XX

Este objeto al principio se usó como si fuesen engranajes para complementar a máquinas de hilado. Al cabo del tiempo se comprendió su potencial para transportar objetos, y se adapto para ser usada para por vías y carreteras, llevando una revolución a la sociedad, y rápidamente siendo adaptada por esta para su uso en la mayoría de tareas, actualmente es impensable prescindir de este objeto, tanto para el transporte de personas, como de objetos a lo largo y ancho del mundo.

Otro ejemplo es la brújula, me parece que es un elemento que, aunque se vea relegado a segundo plano por culpa de los sistemas modernos, es un invento que ha sido, y aún sigue siendo muy usado, sobre todo para los barcos y aviones como apoyo a los sistemas digitales. La historia demuestra que las cosas se pueden mejorar y reinventar para obtener así un objeto fiable y sencillo. Este invento empezó siendo un elemento que se basaba en unos campos un poco inestables, pero una sociedad como la China lo estudió y lo mejoró, hasta tener la idea de brújula tal cual la conocemos hoy.

Los historiadores no tienen muy claro cuál fue el proceso para descubrir este invento claramente, pero la mayoría están de acuerdo en que lo desarrollaron los chinos. Algunos historiadores creen que había una serie de instrumentos que usaban los navegantes del mediterráneo en el cual aprovechaban los campos magnéticos para situarse. Dicen que los árabes los adoptaron para ellos, y que por su expansión, fue llevado a China, que fue donde se desarrolló de verdadera brújula como la conocida actualmente.

Fig. 1.5.- El Astrolabio fue desarrollado por los árabes en el siglo XII. Está basado en el principio del cuadrante.

El tercer invento, y que más relacionado está con lo que es la robótica, son los autómatas programables. Los autómatas eran piezas de ingeniería, complejas y muy curiosas si se estudia su funcionamiento, no sería tan interesante hablar de ellos si no fuese porque añadieron cosas como tarjetas

perforadas, la máquina de calcular, y algo que podemos llamar “memoria artificial” de computador.

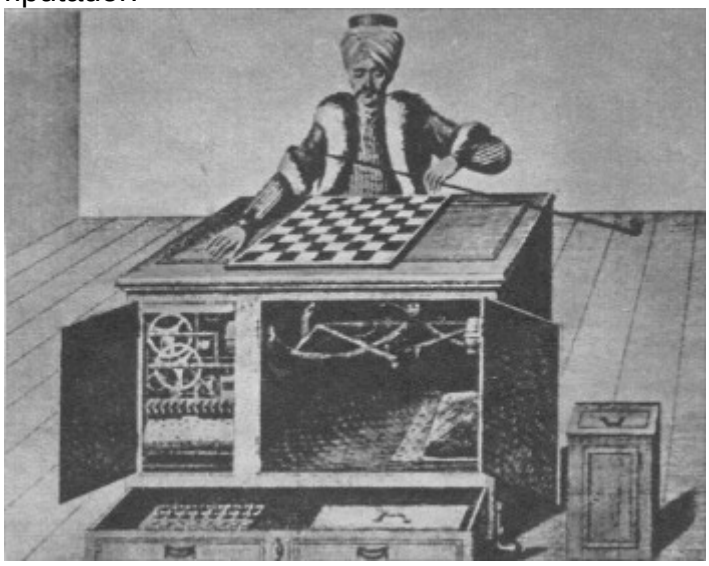


Fig. 1.6.- El turco era un buen ajedrecista, pero poco amigo de las "trampas" en incurrían sus adversarios. Su creador fue el barón von Kempelen. El grabado capta una escena de la presentación hecha por su inventor antes de cada enfrentamiento de su "pupilo"

El primer invento de este tipo pertenece a Jacques Vaucanson, quien, después de haber construido un revolucionario telar mecánico, asombró al mundo de su tiempo con un prodigioso juguete, el que expuso en París, en 1738: el célebre ánade o pato, de tamaño, natural, que nadaba, aleteaba, se alisaba las plumas, tragaba agua, picoteaba e ingería alimento y después de algún tiempo evacuaba lo tragado, en forma de materia amorfa. Su principio de funcionamiento era un complejo sistema de relojería.

Pronto aparecieron competidores igualmente hábiles: Pierre y Louis Droz, suizos, creadores de un "escribiente", quien escribía con hermosa caligrafía unas cuantas palabras, después de mojar la pluma en el tintero, y de un "dibujante", que ejecutaba con elegante trazo un retrato del rey Luis XV, con una perfección tal que su fabricante fue sometido a un proceso por brujería; afortunadamente salió absuelto. Casi igualmente famoso fue el "Turco" construido por el barón van Kempelen, quien jugaba ajedrez y solía ganarles a sus contrincantes, volcando el tablero cuando el adversario infringía las reglas del juego. En realidad los movimientos de este autómata los pensaba un ajedrecista escondido en su interior, era un invento que activo la mente de muchos otros inventores, y que se intentase buscar otros inventos similares.

Un ejemplo de lo que produjo en la sociedad el invento anterior, es el llamado “El Ajedrecista”, “El Ajedrecista” fue un autómata construido en 1912 por Leonardo Torres Quevedo. “El Ajedrecista” hizo su debut durante la Feria de París de 1914, generando gran expectación entre el público. Este artilugio usaba electroimanes bajo el tablero de ajedrez para mover las piezas y jugaba automáticamente hasta el final con un rey y una torre contra un rey contra un contrincante desde cualquier posición sin ninguna intervención humana.

Ninguno de estos ingeniosos juguetes, sin embargo, podía "pensar", como lo haría un hombre. El primer inventor que dio un paso en esa dirección sería Blas Pascal, quien en el siglo XVII inventó la primera máquina de calcular. Su hermana escribiría más tarde: "Este instrumento fue considerado una maravilla por haber reducido a máquina una ciencia que reside en el espíritu, transformando las formas de cumplir todas las operaciones, con absoluta seguridad, sin saber ninguna regla aritmética, sin ficha, ni lapicero, sin necesidad de razonamiento." En esas pocas palabras se define toda una revolución tecnológica: posibilidad de cálculo infalible, la sustitución de la mente humana por la máquina. El invento de Pascal fue perfeccionado más tarde por Grillet de Roven (1678), y luego por el filósofo Leibniz y por Poleni (1709)

Considero con los ejemplos citados pueden definir un denominador común, y es que todo puede sufrir un proceso de mejora, y que a la larga, pueden llegar a objetos bastante útiles. La rueda tuvo un principio totalmente diferente al actual, su utilidad era un engranaje de una máquina de hilar. Otro ejemplo, es el de las brújulas, suponiendo que los historiadores no se equivoquen, muestra como un artefacto que usaba unos principios bastante útiles, pero que no estaba del todo bien diseñado, fue tomado por una sociedad y mejorado hasta como lo conocemos hoy. Y que decir sobre la historia de los autómatas, fue una evolución exponencial donde uno empezó a desarrollar y enseguida la gente fue obteniendo elementos más complejos e interesantes.

Creo que queda claro que la investigación no es un objetivo, si no un camino que se debe de recorrer, intentando obtener unos frutos y pensando en que los resultados están abiertos a que otras personas lo retomen y se puedan convertir en una idea revolucionaria. En el caso de la robótica, el caso es aún más claro en el caso de la robótica. Yo creo que uno de los futuros más importantes de la robótica es en el espacio. Arriesgar un robot humanoide en vez de una persona es una cosa muy interesante. A parte de las ventajas de precisión que tiene una máquina frente a cualquier tarea desarrollada por un humano.

1.2.- Objetivos

Este proyecto me fue planteado con una serie de objetivos para mostrarme por donde iba a tomar mi trabajo.

Lo primero y más importante era familiarizarme con la mano robótica Shadow, leerme toda la información que suministraba el fabricante, comprobar los complementos que vienen en el maletín, e intentare entender el programa para la adquisición de datos.

La primera parte del proyecto venia complementada por el estudio del protocolo de transmisiones del bus CAN, ya que tanto la mano citada, como el brazo que vería después trabajaban con tarjetas CAN. Esto era leerme bibliografía recomendada por parte del profesor, intentando entender las

tramas de datos que se enviaban y recibían entre el ordenador y la mano o brazo.

La segunda parte del proyecto era intentar entender el funcionamiento del brazo robótico Robotnik, disponía de una serie de manuales proporcionados por la empresa, otros desarrollados por compañeros del departamento que estaban trabajando también sobre el brazo. La intención era que intentase programar en C++ el movimiento de algún motor del robot. Esto se conseguía leyendo sobre una serie de principios como puede ser el estudio cinemático directo o inverso que usaba el fabricante, o estudiar el significado de los sistemas de coordenadas que usaba el fabricante para desarrollar su programa en particular. Luego debía de intentar programar mediante un entorno de desarrollo integrado o programando sobre un editor de texto un sencillo programa de control del robot.

Otro problema que se planteo durante el proyecto era intentar conseguir instalar el brazo en cualquier ordenador que usase un sistema operativo basado en Linux

Y como objetivo final, aunque sería el objetivo que iría desarrollando durante todo el proyecto, era aprender sobre principios de robótica, conocer el estado del sector en la actualidad, e intentar comprimir en un documento, las especificaciones mecánicas de la mano y del brazo.

1.3.- Descripción de la memoria

He dividido el proyecto en 5 partes, cada una centrada en una parte. La introducción, que es la que nos encontramos, la he dividido en tres partes, la primera contiene los motivos por los que elegí este proyecto, lo segundo los objetivos que tenía al acabar el proyecto, y lo tercero este índice rápido para explicar el proyecto

El segundo bloque es una introducción rápida sobre el estado de la robótica antropomorfa por una serie de universidades y organismos que explican el estado de este sector, mostrando muchas ramas abiertas, cada una con un objetivo diferente, como puede ser robots capaces de andar, robots con capacidad de mostrar sentimientos, entre otros muchos ejemplos.

El tercer bloque contiene toda la parte de hardware, donde intento explicar tanto para el brazo como para la mano, su funcionamiento. He intentado introducir su similitud con la parte humana que corresponden, intentando mostrar las ventajas de porque esta forma y no otra. Luego he intentado introducir un poco la empresa que lo suministra, y por ultimo introducir las especificaciones físicas y mecánicas del elemento en cuestión.

El cuarto bloque trata del software, esto contiene, los programas que han sido suministrados por parte del fabricante, como de la plataforma o sistema operativo Linux en el que trabaja el brazo. En la última parte he hecho una breve introducción a los diferentes programas que he considerado que me pueden servir para programar en C++ para el control de motores del robot.

El quinto bloque explica el trabajo que he ido desarrollando a lo largo de todo este tiempo, intentando que sea lineal con respecto a mi trabajo en el laboratorio.

El sexto y último bloque muestra las conclusiones que he sacado del trabajo con este proyecto. Tanto como las expectativas, como las cosas que he sacado en claro después de estos meses de trabajo.

2.1-Antecedentes

La palabra "robot" tiene su origen en el vocablo checo "robota" que significa "servidumbre", "trabajo forzado", o esclavitud, y hace referencia especialmente a los llamados "trabajadores alquilados" que vivieron en el Imperio Austrohúngaro hasta 1848.

Un robot es un dispositivo electrónico generalmente mecánico, que desempeña tareas automáticamente, ya sea de acuerdo a supervisión humana directa, a través de un programa predefinido o siguiendo un conjunto de reglas generales, utilizando técnicas de inteligencia artificial. Generalmente estas tareas reemplazan, asemejan o extienden el trabajo humano, como ensamblado en líneas de manufactura, manipulación de objetos pesados o peligrosos, trabajo en el espacio, etc.

Dado que la definición anterior es una definición muy general y no muy técnica, a continuación vamos a dar unas cuantas definiciones dadas por diferentes entidades de normalización.

Según la norma ISO, se define robot 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”.

Según la Asociación de industrias robóticas (RIA) un robot es un “manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas”.

Por último, según la norma AFNOR, se define robot como: “Manipulador automático servo-controlado, reprogramable, polivalente, capaz de posicionar y orientar piezas, útiles o dispositivos especiales, siguiendo trayectorias 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”.

2.2.- Reseña histórica de los robots

En este apartado expondremos algunos hechos y acontecimientos que marcaron en algún momento de la historia la evolución de la robótica.

Los autómatas, o máquinas semejantes a personas, ya aparecían en los relojes de las iglesias medievales, y los relojeros del siglo XVIII eran famosos por sus ingeniosas criaturas mecánicas. Jacques de Vauncansos construyó varios músicos de tamaño humano a mediados del siglo XVIII. Esencialmente se trataba de robots mecánicos diseñados para un propósito específico: la diversión.

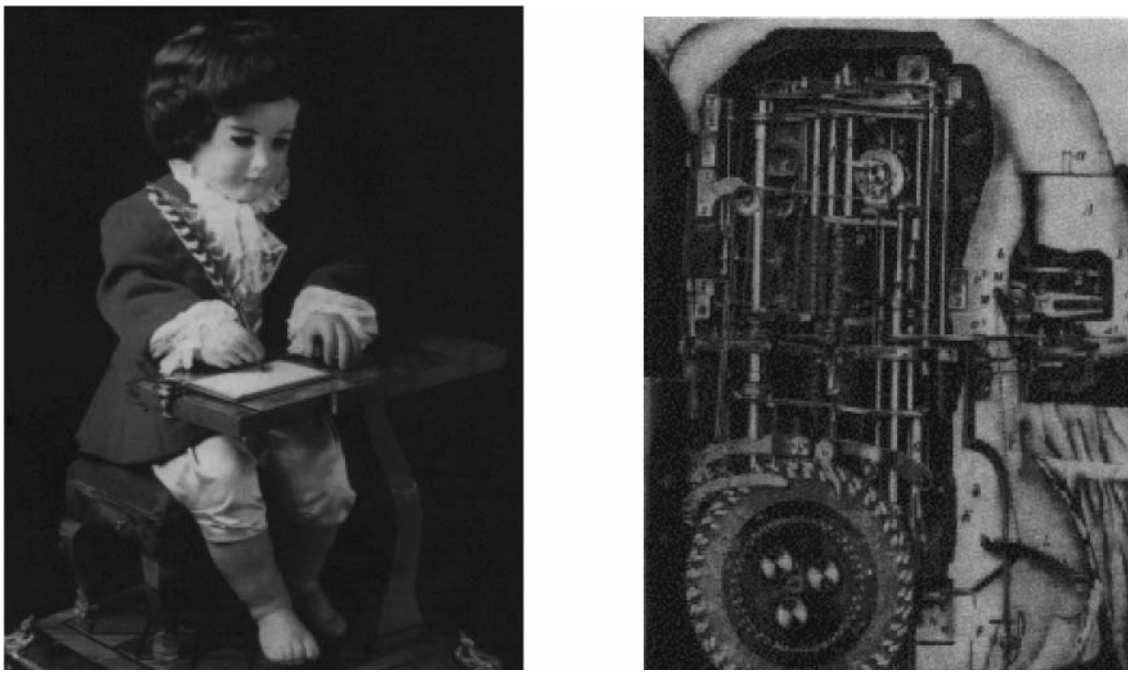


Fig.2.1: Escriba de Jacques Droz, que hacía bellos trazos de caligrafía y mojaba la pluma en tinta

En 1805, Henri Maillardert construyó una muñeca mecánica que era capaz de hacer dibujos. Cabe mencionar que estas creaciones mecánicas de forma humana deben considerarse como inversiones aisladas que reflejan el genio de hombres que se anticiparon a su época, y ni mucho menos eran creaciones generalizadas, como ocurre hoy en día.

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.

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.

También en la actualidad se está desarrollando la inteligencia artificial, que trata de conseguir que los ordenadores simulen en cierta manera la inteligencia humana. Se acude a sus técnicas cuando es necesario incorporar en un sistema informático, conocimiento o características propias del ser humano.

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.

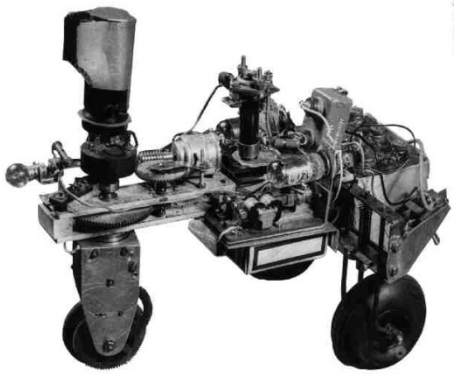


Fig.2.2: Foto Robot ELSIE.

En 1968, apareció SHACKEY 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.

2.3.-Clasificación de los robots

Se puede establecer la clasificación de los diferentes robots atendiendo a diferentes criterios, como pueden ser:

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.
- 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.

El nivel de inteligencia propio de cada robot:

- Dispositivos de manejo manual controlados por una persona, que necesitan de un operador humano para su mando.
- Robots de secuencia fija, programados de tal manera que siempre realizan las mismas acciones, sin tener en cuenta las alteraciones que se pudieran producir en el sistema. La secuencia de funcionamiento no puede ser modificada.
- Robots de secuencia variable, donde un operador humano modifica la secuencia con anterioridad al inicio de un ciclo.
- Robots de repetición: Son aquellos que se limitan a repetir una secuencia de movimientos, previamente ejecutada por un operador humano, haciendo uso de un controlador manual o un dispositivo auxiliar.
- Robots controlados numéricamente, en los que el operador proporciona un programa de movimientos y acciones.
- Robots inteligentes, los cuales son capaces de actuar de forma distinta ante estímulos externos aleatorios, y todo ello sin que intervenga el operador humano.

Según su morfología:

- Androides o antropomorfos: Los androides son robots que tienen forma y actuación similares a las de los seres humanos. Hoy en día existen robots de todas las formas y tamaños por lo que la clasificación de androide es cuanto menos ambigua, ya que los robots se pueden parecer a los humanos en un gran número grados, ya sea tanto por su

morfología como por su forma de comportamiento. Más adelante se profundizará más sobre este tipo de robots.

- Robots zoomórficos: Robots caracterizados principalmente por su sistema de locomoción que imita a diversos seres vivos. Los androides también podrían considerarse robots zoomórficos, ya que los seres humanos son seres vivos.

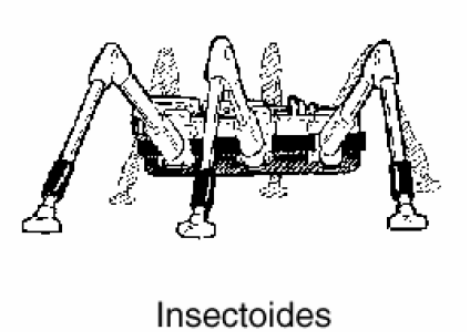


Fig. 2.3: Robot zoomórfico.

Según su utilidad:

- Robots industriales: Los robots industriales son artilugios mecánicos y electrónicos destinados a realizar de forma automática determinados procesos de fabricación o manipulación. Son en la actualidad los más frecuentemente utilizados.

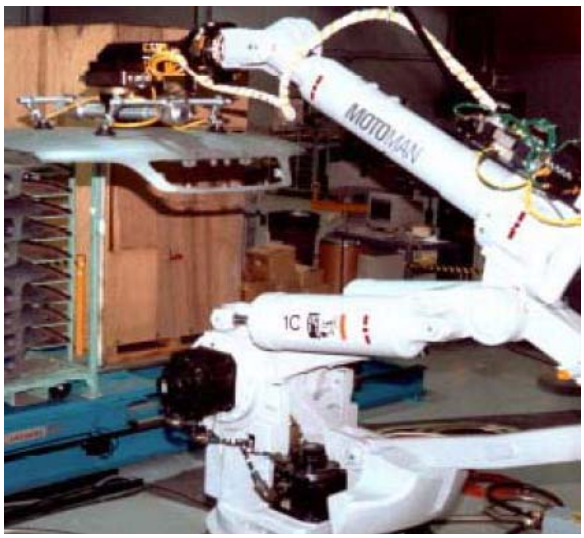


Fig. 2.4: Robot industrial utilizado en la industria del automóvil

- Robots médicos: Los robots médicos son, fundamentalmente, prótesis para disminuidos físicos que se adaptan al cuerpo y están dotados de potentes sistemas de mando. Con ellos se logra igualar con precisión los movimientos y funciones de los órganos o extremidades que suplen.
- Micro-robots: Con fines educacionales, de entretenimiento o investigación, existen numerosos robots de formación o micro-robots a

un precio muy asequible y, cuya estructura y funcionamiento son similares a los de aplicación industrial.

- Robots tele-operados: se controlan remotamente por un operador humano. Cualquiera que sea su clase, los robots tele-operados son generalmente muy sofisticados y extremadamente útiles en entornos peligrosos tales como aquellos en los que hay residuos químicos o para la desactivación de bombas.

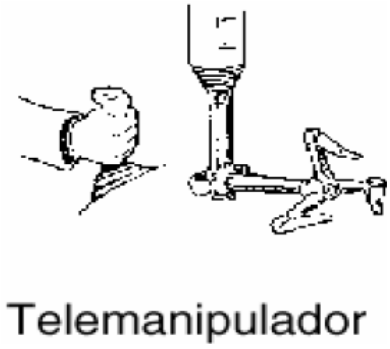


Fig. 2.5: Robot tele-operado.

Según los grados de libertad de movimiento del robot:

Esta clasificación hace referencia a la cantidad de movimientos que puede realizar un determinado robot o máquina y a los lugares donde es capaz de actuar.

Existen robots con gran variedad de grados de movimientos, desde los más elementales con pocos grados de libertad (3 ó 4 a lo sumo), hasta robots móviles con una gran libertad de movimiento, pudiendo considerarse que éstos poseen un elevado número de grados de libertad.

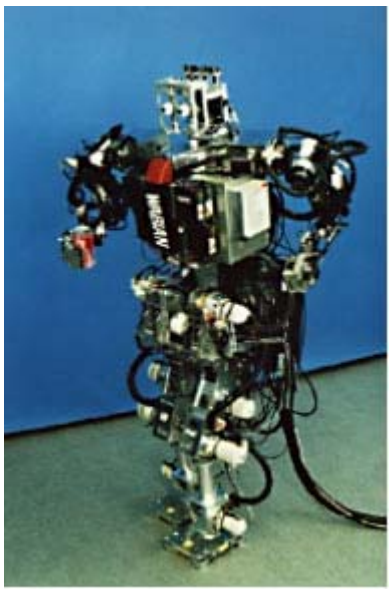
2.4.-Estructuras robóticas antropomorfas en la actualidad

En la actualidad hay un gran número de investigadores intentando desarrollar estructuras robóticas que se asemejen cada vez más al ser humano tanto en el comportamiento como en la forma. Se describen a continuación algunas de las estructuras de este tipo más famosas y punteras:

WABIAN –RII: Sucesor de WABOT 1, el primer robot con una estructura antropomorfa completa y desarrollada en la universidad Waseda de Tokio.

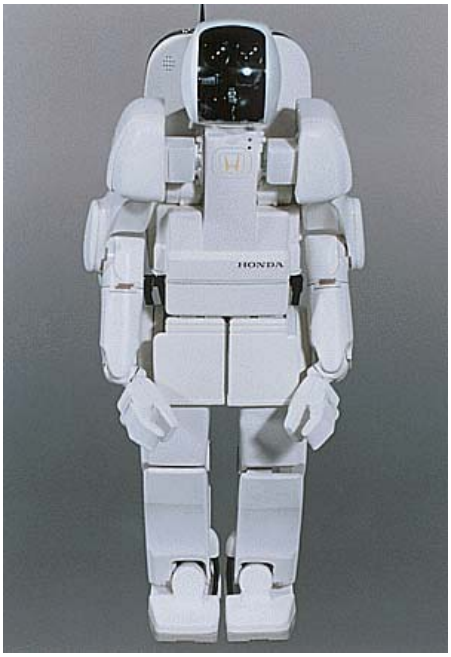
Una vez creados WABOT 1 y 2, esta universidad entró en competición con Honda en la creación de robots bípedos cuyo medio de locomoción fuera lo más parecido posible al medio de locomoción humano. Las principales características de este robot son:

Nombre	Wabian-RII
Origen del nombre	Acrónimo de Waseda Bípeda humanoid
Propósito	1. Clarificar el control de movimiento humano desde el punto de vista de la robótica. 2. Establecer una base tecnológica para construir robots personales en el futuro.
Inspiración	Trabajo de Ichiro Kato
Altura	1.83 m
Anchura	70 cm (en los hombros)
Peso	127 Kg.
Visión	2 cámaras de color digitales
Sensores	27 encóders, 2 sensores de fuerza/par de seis ejes y un girómetro de tres ejes.
Composición del chasis	Aleaciones de aluminio, fibra de carbono, SUPER-DURALMIN
Baterías	Ninguna
Suministro eléctrico externo	100V AC monofásico, 200V AC trifásico
Coste	370000$ (componentes)



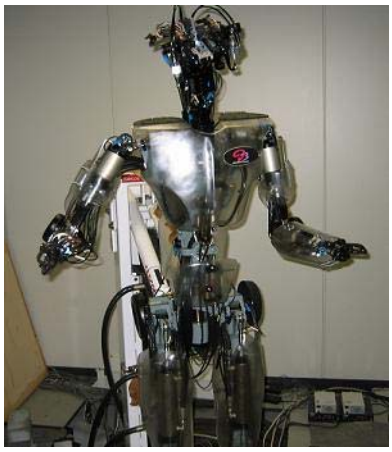
HONDA P3: De un proyecto millonario nació en 1996 el primer prototipo creado por HONDA. El tercer prototipo es el P3, capaz de caminar hasta una puerta, abrirla y atravesarla.

Nombre	Honda P3
Origen del nombre	Prototype 3
Propósito	Descubrir una nueva perspectiva de la movilidad para una compañía de automóviles.
Altura	1.6 m
Peso	130 Kg.
Visión	Cámaras de video digitales
Sensores	Girómetros, sensores de aceleración, sensores de fuerza de 6 ejes.
Composición del chasis	
Baterías	Ni-Zn de 25 minutos de duración
Suministro eléctrico externo	136 V DC
Estado del proyecto	En desarrollo (a partir del año 2000 proyecto ASIMO)



DB: Creado con el fin de conocer mejor el cuerpo humano y el control que ejerce el cerebro sobre el mismo (forma de resolver la redundancia presente en el mismo), buscando imitarlo y así conseguir una mejor interacción robot-humano. Las principales características de este robot son:

Nombre	DB
Origen del nombre	Dynamic Brain
Propósito	Herramienta experimental para explorar cuestiones del control motor biológico, teorías de la neurociencia, aprendizaje y otros aspectos relacionados con el aprendizaje del cuerpo humano.
Altura	1.90 m
Peso	90 Kg.
Visión	2 ojos de dos grados de libertad, cada uno con dos cámaras para emular la visión humana.
Sensores	4 cámaras, un sistema de recubrimiento con sensores de velocidad angular y aceleración de tres ejes, 26 sensores de carga y sensores de posición para todos los grados de libertad.
Composición del chasis	Aluminio de bajo peso.
Baterías	Ninguna
Coste	1000000 $

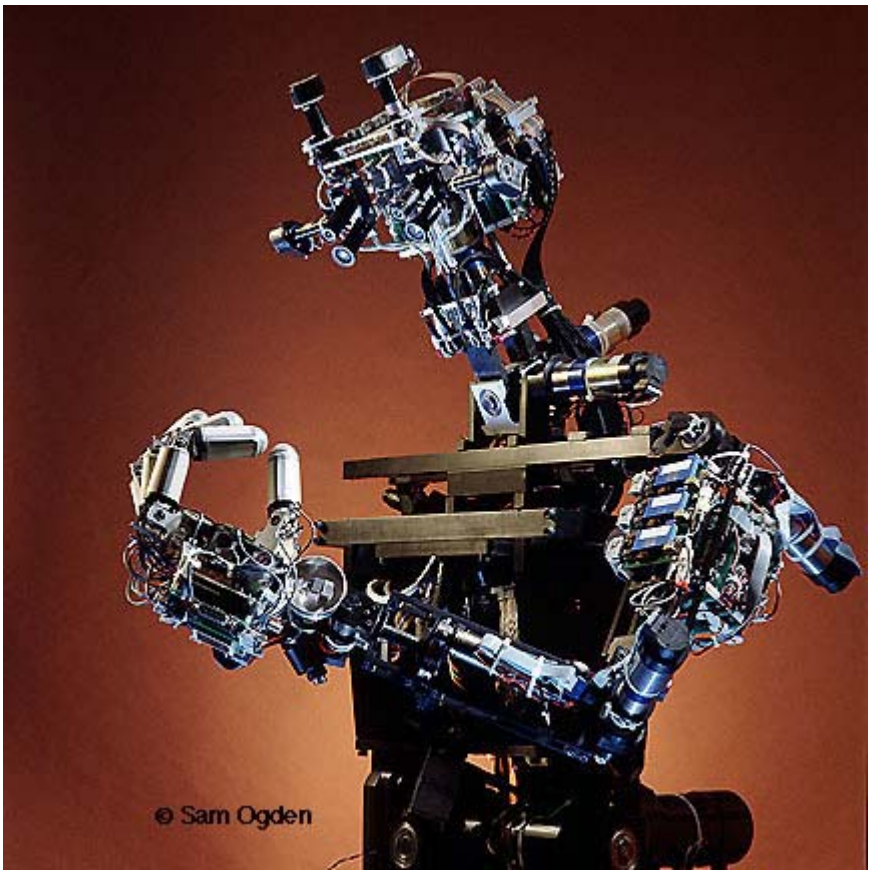


JACK: Robot diseñado para imitar el comportamiento de un humano e interactuar con él. Las principales características de este robot son:

Nombre	Jack (Informalmente)
Origen del nombre	Obtenido de una canción de Robbie William's
Propósito	Estudiar interacciones continuas hombre/robot y los elementos de hardware/software necesarios para una integración global.
Altura	Aproximadamente 1.6 m
Peso	Aproximadamente 60 Kg.
Visión	Visión estereoscópica con cámaras de vídeo CCD.
Sensores	Micrófono de audición estéreo, sensores en las articulaciones.
Composición del chasis	Principalmente aluminio

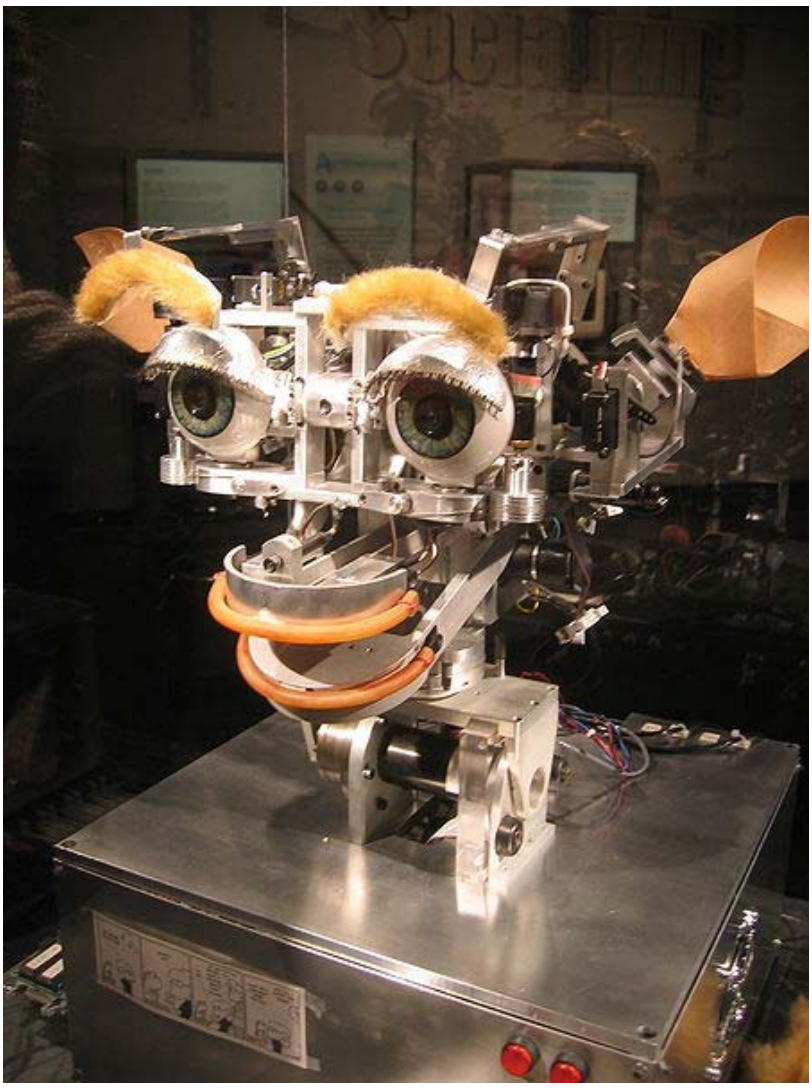
COG: Robot antropomorfo diseñado en el MIT, tras el desarrollo de varios robots con forma de animales e insectos. Las principales características de este robot son:

Nombre	Cog
Origen del nombre	Hace referencia al término “COGnitive”
Propósito	Estudiar aspectos de desarrollo físico, morfológico, integración senso-motora, e interacción social. Estudiar modelos de inteligencia humana.
Altura	Aproximadamente 1.72 m (incluye 86 cm de base)
Visión	4 cámaras en color
Sensores	2 micrófonos en las orejas, sistema inercial de 3 ejes, array de sensores táctiles, sensores cinemáticos en cada articulación, sensores de fuerza, sensores fines de carrera y sensores de temperatura en los motores.
Baterías	Ninguna
Alimentación externa	+/- 12V, +/-24V, +/-5V



KISMET: Robot desarrollado por el MIT capaz de expresar emociones. Las principales características de este robot son:

Nombre	Kismet
Origen del nombre	Traducción al turco de “cara”
Propósito	Aprendizaje social y estudio de las interacciones sociales entre hombre y robot.
Altura	38 cm.
Peso	De 4 a 7 Kg.
Visión	3 cámaras digitales.
Sensores	3 micrófonos, 21 encóders
Material de la carcasa	Aluminio
Baterías	Ninguna
Alimentación externa	6A 24V, 16A 12V, 5V (para la electrónica)



SIG: Robot antropomorfo para estudiar aspectos relacionados con la visión artificial. Las principales características de este robot son:

Nombre	Sig
Origen del nombre	“Symbiotic Intelligence Group”, nombre del grupo de investigación encargado.
Propósito	Estudiar la integración de la inteligencia artificial en un entorno con múltiples sensores y un gran número de grados de libertad. Servir de prototipo para un humanoide completo.
Visión	Cámara digital.
Sensores	4 micrófonos para escucha estereofónica.
Material de la carcasa	Aluminio

UC3M-HUMANOID (Rh-1): Robot antropomorfo con 21 grados de libertad desarrollado en la Universidad Carlos III de Madrid capaz de reproducir la forma de andar humana y realizar tareas conjuntas con humanos u otros robots.

- El robot humanoide de la Universidad Carlos III de Madrid es un robot autónomo capaz de caminar por los diferentes tipos de ambientes interiores y exteriores, y coopera con otros seres humanos y robots en entornos reales de trabajo en colaboración.
- El robot humanoide Rh-0 (Rh y su actualización-1) es de tamaño completo (1,3 m de altura, unos 50 Kg. de peso) del robot. El robot está equipado con dos procesadores a bordo y sistema de comunicación de bus CAN y dos baterías de a bordo que permite a los 30 minutos de autonomía.
- El sistema sensorial del robot se divide en dos partes: los sensores para la locomoción (inclinómetros y acelerómetros) y la interacción de los sensores (cámara y micrófonos).

- Este prototipo llamado REEM-B pertenece a Pal Technology Robotics, con sede en Barcelona. El robot tiene un aspecto humanoide con metro y medio de altura, capaz de reconocer rostros, comunicarse con las personas, caminar e incluso subir escaleras o sentarse
- Este ingenio, que pesa 60 kilos tiene una autonomía de unas dos horas, mucho mayor que la mayoría de robots hasta ahora producidos, puede también caminar con estabilidad e incluso cargar hasta doce kilos de peso, por lo que se trata de un robot sofisticado pensado para poder ayudar en asuntos domésticos, sobre todo a personas con alguna discapacidad física o psíquica.
- Reem-B tiene la capacidad de archivar en su sistema información datos sobre los humanos y está programado para evitar obstáculos o mantener una conservación, entre otras habilidades.

3.1.-Introducción

Este bloque se va a centrar en el estudio del hardware con el que he trabajado. Explicar en detalle todos los componentes en los que consta el proyecto. Intentar explicar al detalle el brazo, la mano, y el protocolo de comunicaciones CAN.

Desde el punto 3.2 hasta el punto 3.7 intentaré detallar todos los puntos que me parecen interesantes para describir el funcionamiento de la mano.

El punto 3.3 contendrá toda la información sobre el brazo, el primer apartado describirá el parecido con el brazo humano, el segundo apartado hablará sobre la empresa Robotnik, y el tercer apartado hablará sobre las características mecánicas de este.

El punto 3.4 hablará sobre el protocolo de transmisión de datos, que en nuestro caso, todo esta basado en bus CAN conectado a una tarjeta CAN insertada en el ordenador.

3.2.- Mano Shadow

3.2.1- Introducción a la mano

La mano humana es la más perfecta y versátil herramienta de la naturaleza. Aunque roedores como las ardillas y aves como los loros pueden manipular objetos con sorprendente habilidad, es en nuestro propio grupo, el de los primates, donde podemos encontrar instrumentos semejantes.

Todos los primates tienen la capacidad de agarrar con sus manos, lo cual es una adaptación muy útil para la vida arborícola, pero muchos de ellos sólo ejercen fundamentalmente una presión fuerte y poco precisa, agarrando los objetos entre la palma de la mano y la totalidad de los dedos.

Para poder asir objetos con delicadeza y manipularlos con precisión se requiere que el pulgar sea un dedo de características diferentes a las de los demás, con capacidad de rotar lateralmente con respecto a la mano y de oponerse a cada uno de los demás dedos. Los grandes simios antropomorfos, como orangutanes, gorilas y chimpancés, presentan un pulgar oponible y pueden ejercer una pinza de precisión, pero mucho más torpemente que el hombre.

Ello es debido a la escasa longitud relativa del pulgar con respecto a los demás dedos, que impide que se pueda enfrentar una yema contra la otra. El hombre tiene un pulgar mucho más largo con respecto al resto de la mano que los otros antropomorfos. Esto se ha producido más bien debido a un acortamiento del resto de la mano que a un crecimiento del pulgar. Los orangutanes, que viven en los árboles, tienen los dedos no pulgares muy largos para poder agarrarse a las ramas con mayor eficacia. Los antropomorfos que se desplazan a cuatro patas (gorilas y chimpancés), andan sobre los nudillos

de las manos porque aún siguen conservando una mano larga, recuerdo de sus orígenes arbóreos (es útil conservar una buena capacidad de trepar a los árboles, por ejemplo, para escapar de un peligro momentáneo).

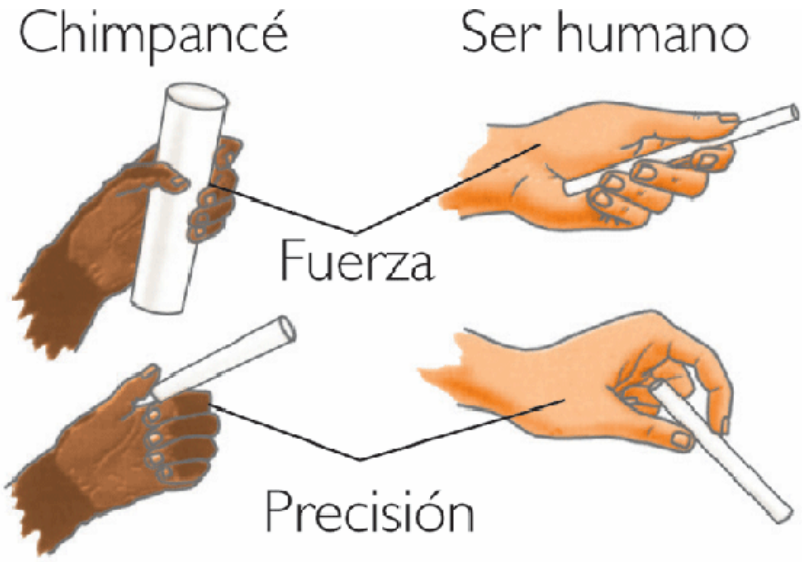


Fig 3.1.- Ejemplo de la precisión de la mano humana frente a la del chimpancé

3.2.2.-Anatomía de la mano humana.

La mano humana se conecta a la muñeca a través de la palma y está dotada de veinte grados de libertad accionados por cerca de cuarenta músculos.

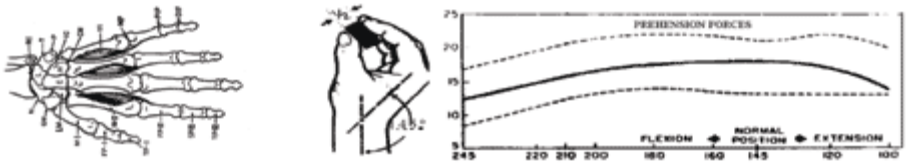


Figura 3.2. La mano humana. Estructura ósea y efecto de la orientación de la muñeca en la fuerza de agarre.

La estructura ósea de la mano se muestra en figura 1. En la misma figura se muestra el efecto de la orientación de la muñeca en la fuerza de agarre. Cada dedo está compuesto por tres falanges, a excepción del dedo pulgar, que solo tiene dos. El dedo pulgar esta fijo por debajo de los otros dedos y puede realizar los movimientos de cierre y rotación (circonducción), debido a la gran movilidad de su metacarpo, como es mostrado en la figura 3.3.

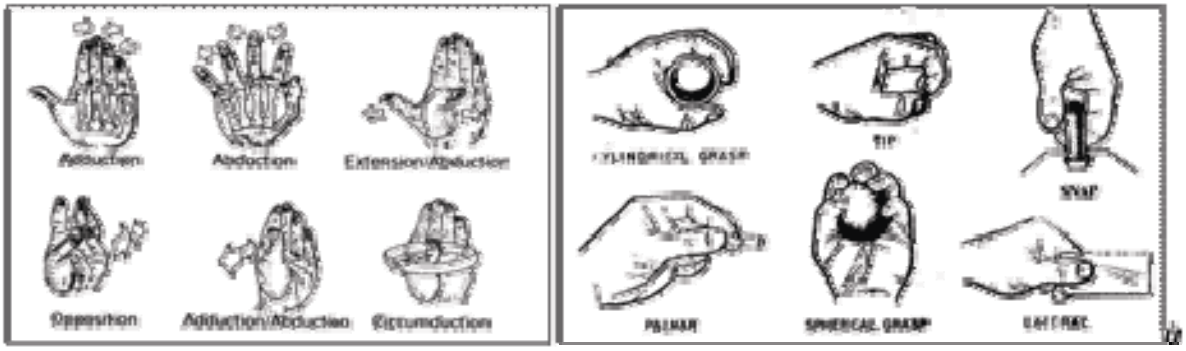


Figura 3.3. La mano humana. Movimientos característicos de los dedos. Configuraciones de agarre.

Esto permite variar la orientación del plano en que se desarrolla el movimiento de doblado y extensión del dedo pulgar, propiedad a través de la cual es posible oponer el dedo pulgar a los otros dedos.

Con el término abducción se entiende el movimiento de salida del dedo del eje del brazo, como es mostrado en la figura 3.3. El movimiento de Extensión/Abducción es la capacidad de extensión del pulgar hacia la parte exterior y flexión hacia el interior de la palma. El término Oposición se define como la capacidad de unión de las puntas del pulgar y el meñique. La Aducción/Abducción es la capacidad de acercamiento y alejamiento del pulgar de la palma, cuando ambos se encuentran en un mismo plano. La gran cantidad de músculos y juntas que están presentes en la mano permiten esta gran variedad de configuraciones de agarre. En 1919, Schlesinger desarrolló una clasificación de la taxonomía para el estudio de la destreza de las manos humanas y la planificación de prótesis. Este autor agrupó en seis categorías las estrategias de agarre de la mano humana: agarre cilíndrico (Cylindical Grasp), de punta (Tip), de gancho (Hook or Snap), de palma (Palmar), esférico (Spherical Grasp) y de lado (Lateral), como es mostrado en la figura 2.

3.2.3.-Definición de actuador

Para introducir el objetivo de una mano, creo que es interesante definir un término muy interesante, y que para mí puede ser uno de los objetivos a conseguir de una mano, convertirse en un actuador de un proceso industrial. La definición de actuador es la de: dispositivo capaz de generar una fuerza a partir de líquidos, de energía eléctrica y gaseosa. El actuador recibe la orden de un regulador o controlador y da una salida necesaria para activar a un elemento final de control como lo son las válvulas.

Para mí sería interesante introducir esto como una posible vía para obtener rentabilidad del uso de una mano robótica, la ventaja que tiene frente a los anteriores es su alta precisión y libertad de movimiento. Aunque ahora mismo es inviable, ya que la mayoría de procesos están bien diseñados, y el costo es tan bajo que hablar de mano robótica es una locura.

Fig. 3.4- Diferentes ejemplos de actuadores actualmente disponibles en los procesos de producción.

Como en la figura 3.4. se muestra una serie de actuadores, los actuadores son herramientas de distinta forma, cada uno para una acción en particular. Lo interesante de un actuador es que sea capaz de realizar la tarea encomendada. El problema es que la mayoría de actuadores tienen un uso muy limitado, o lo que es lo mismo, no pueden tener muchas opciones de trabajo.

Una mano robótica es un claro ejemplo de actuador perfecto, tiene mucha libertad de movimiento, es capaz de adaptarse a muchas actividades diferentes, y tiene gran sensibilidad a la hora de agarrar objetos frágiles. El problema es la falta de fuerza de esta, y el coste de trabajo que conlleva.

Aunque actualmente está demostrado que la cosa va por otro camino de investigación, por ejemplo, actualmente, el doctor Honghai Liu, y el profesor Xiangyang Zhu están intentando crear una mano capaz de realizar íntegramente las acciones que solo puede realizar una mano humana. Su búsqueda es para integrar las manos robóticas en tareas médicas, como puede ser una prótesis para personas con discapacidad.

Hay muchas otras aplicaciones, pero en el camino de la biomedicina y el sector espacial son el futuro. Realizar tareas con una precisión muy alta, y a la vez con gran capacidad para adaptarse a los problemas que puedan surgir. Aunque nunca debemos de dejar de lado que el sector de la industria es un sector que produce beneficios, y puede ser que un futuro se demande este tipo de elemento.

3.2.4.- Descripción de la empresa Shadow Robot Company Ltd.

La empresa Shadow Robot Company Ltd. es una empresa del reino unido que ha creado un modelo de mano muy parecida a la humana capaz de imitar la mayoría de movimientos de esta, permitiendo el control de esta en particular o añadiéndola a alguna plataforma con otros componentes.

Fig. 3.5.- Muestra de la mano Shadow

Como se puede comprobar en la figura 3.5, la mano dispone de 40 tendones conectados a un sistema de músculos neumáticos capaces de controlar con bastante precisión la fuerza aplicada, dando bastante utilidad para trabajar con objetos de poco peso o muy frágiles.

La libertad de movimiento viene acompañada de unos sensores de tacto capaces de sentir la más mínima variación. Permitiendo en todo momento poder tener monitorizada la fuerza que se está aplicando a este objeto.

En particular, la mano Shadow es capaz de imitar el movimiento y la sensibilidad de la mano humana, solo que tiene una ventaja, y es que tiene menos fuerza que la mano humana, así que se puede usar en entornos humanos sin riesgos para la seguridad.

3.2.5.-La mano Shadow

Como hemos citado anteriormente, la mano humana es una herramienta muy potente, en nuestro grupo de investigación disponemos de la mano Shadow, esta mano es igual a una mano diestra de tres dedos. Dispone de un avanzado sistema robótico derivado de la mano. Reproduce un subconjunto de manos humanas de once grados de libertad. Ha sido diseñada para asemejarse en fuerza y sensibilidad de movimiento a una mano humana. Todas las medidas de las partes antropomorfas corresponden a una típica mano humana masculina.

Cada uno de los dedos posee cuatro articulaciones (aunque sólo tres grados de libertad): Distal (la más próxima a la punta del dedo), media, proximal (unida a la palma) y la unión de aducción/abducción (que funciona a través de la proximal). Esto no sucede en el dedo pulgar en donde hay cinco articulaciones (con cinco grados de libertad). Disponemos de tres dedos: pulgar, índice y anular. Esto hace que el número de articulaciones que posee

nuestra mano sea de trece (cinco en el pulgar y cuatro en cada uno de los otros dos dedos), aunque sólo vamos a utilizar (supongo que de momento) once.

3.2.5.1.-Mecánica.

A pesar de que la mano puede disponer de cinco dedos, en la versión reducida de la misma sólo tiene tres, siendo estos los dedos índice, anular y pulgar. Estos son suficientes para el agarre del que se pretende. Mientras que el pulgar tiene cinco grados de libertad, el índice y el anular tienen tres.

Índice (*First*).
Anular (*Third* o *Ring*).
Pulgar (*Thumb*).

3.2.5.2.-Dimensiones

La mano ha sido diseñada para asemejarse lo más posible a la mano media de una mano masculina. La distancia de los dedos, desde la punta hasta la mitad del nudillo es de 98mm. El tamaño del pulgar es de 105mm. El tamaño de la palma (desde la mitad del nudillo hasta el eje de la muñeca) es de 90mm. La anchura de la palma es de 85mm. El grosor de la palma es de 24mm.

La mano de tres dedos tiene un peso de 350 gramos.

3.2.5.3.-Velocidad

Hay algunas diferencias en las velocidades de movimiento de las diferentes partes de la mano. También, diferentes métodos de movimiento producen diferentes velocidades máximas. De forma aproximada, se sitúa la velocidad del movimiento en la mitad de la velocidad de un humano. Por ejemplo, el tiempo para cerrar la mano es de 0.2 segundos aproximadamente.

3.2.5.4.-Materiales

El sistema está compuesto de diferentes tipos de metales y plásticos.

- Muñeca fija: Aluminio. ***(de esto no estoy seguro)***
- Palma: Acetil, aluminio y poliuretano.
- Dedos: Acetil, aluminio, las uñas de policarbonato y también se utiliza el poliuretano.
- Base: acetil, caucho y latón.

3.2.5.5.-Fuerza

Las medidas aproximadas de los máximos esfuerzos de torsión son:

- Muñeca puede soportar 1.5Nm.
- Distal (próxima a la punta del dedo): 0.5Nm máximo (entre dedos y pulgar)
- Proximal (próxima a la palma de la mano): 1Nm máximo (entre dedos y pulgar)

3.2.5.6.-Consumo eléctrico.

El consumo eléctrico de la mano Shadow viene demandado por el bus CAN, que tiene requiere una señal eléctrica de 8 V y un consumo máximo de 1 A. La mano original Shadow, con actuadores neumáticos tenía un consumo eléctrico mayor por parte de las válvulas.

3.2.5.7Actuadores.

Aunque, tal y como ya se ha dicho, la mano Shadow inicialmente utiliza como actuadores sistemas neumáticos (músculos neumáticos), la mano que la UPCT posee va a utilizar motores eléctricos que, de momento, quedan fuera de este informe. Dado el diseño de inspiración biológica de la mano, será necesario un par de motores eléctricos para cada articulación con el fin de moverlos de una forma oponente, tal y como sucede en una mano humana. La mano (tanto la original como esta de tres dedos) utiliza tres modos distintos de actuación sobre las articulaciones: la ya citada actuación oponente que permite un control total de la misma, un único motor con un muelle de retorno para el movimiento de aducción y abducción y una actuación acoplada para la articulación entre las falanges distal y media. La tabla 1 muestra el mecanismo de actuación de cada articulación.

Unión	Conexión	Rango	Mecanismo de actuación*
Dedos índice y anular			
1	Distal-Medial	-20 … +90	Par acoplado*
2	Medial-Proximal	0 … +90	
3	Proximal-Nudillo	-20 … +90	Par
4	Nudillo-Palma	-25 … +25	Uno con muelle
Dedo pulgar			
1	Distal-Medial	-20 … +90	Par
2	Medial-Proximal 1	-40 … +40	Par
3	Medial-Proximal 2	-15 … +15	Par
4	Proximal-Palma 1	-15 … +80	Par
5	Proximal-Palma 2	-60 … +60	Par

Tabla 1. Datos sobre las articulaciones.

Par acoplado: Las dos uniones están acopladas de tal forma que el ángulo de la unión 2 es menor que el de la unión 1.

Par: Dos músculos antagónicos.

Uno con muelle: Un músculo con retorno de muelle.

Las uniones Distal-Medial de los dedos, a excepción del pulgar, están acopladas de forma similar a los dedos humanos, de forma tal que el ángulo de la unión medial es siempre mayor o igual al ángulo de la unión distal. Esto permite tensar la falange medial mientras la falange distal está estirada.

La mano de tres dedos está movida, pues, por 22 tendones. Están etiquetados de la siguiente manera:

3.2.5.8.-Sensorización de la posición.

Cada articulación posee un sensor de posición y en cada punta del dedo hay un sensor táctil. La mano transmite de forma continua mensajes con los valores de estos sensores.

La posición de cada articulación es detectada por un sensor de posición angular (de efecto HALL especial –patentado por Shadow), con una resolución típica de 0.2 grados, anexo a cada unión. La salida del sensor varía con el desplazamiento angular de la unión, ofreciendo una variación a la salida de 2 V para una variación en la rotación de 90º. En el punto medio, el sensor ofrece una salida de 2.5 V aproximadamente.

La información de cada sensor es convertida utilizando un convertidor analógico–digital de 12 bits situado en las proximidades del sensor. El valor digital es enviado a través de bus SPI hasta un PIC situado en la palma, el cual convierte las señales recogidas y las envía por el bus CAN. La velocidad de muestreo es configurable hasta un máximo de 180Hz.

Otros sensores que aparecen permiten monitorizar factores tales como la aducción/abducción o el movimiento de flexión de los dos metacarpios. Estos sensores tienen su conversión A/D situada en la palma de la mano.

El pulgar posee dos movimientos distintos en la falange media. El sensor más cercano a la palma mide el movimiento de aducción/abducción, el más alejado mide la flexión de la falange media. Para medir la rotación del pulgar se emplean dos sensores para obtener una mayor precisión.

La articulación 5 del pulgar posee dos sensores (diferenciados por el offset). Sin embargo, en su disposición habitual el primer sensor proporciona información suficiente sobre la articulación.

Cada mensaje que genera el bus CAN con la información de los sensores de posición consiste en un identificador (ID) y 8 bytes de datos. Los datos contienen los valores obtenidos de 4 sensores, utilizando 2 bytes por sensor. Es posible conocer a qué sensor pertenecen los datos utilizando el ID del mensaje. La tabla 2 muestra un ejemplo de un mensaje.

ID del mensaje **Nodo base +N**	Byte 0 L0	Byte 1 H0	Byte 2 L1	Byte 3 H1	Byte 4 L2	Byte 5 H2	Byte 6 L3	Byte 7 H3
	Sensor N=H0*256+L0		Sensor N+1=H1*256+L1		Sensor N+2=H2*256+L2		Sensor N+3=H3*256+L3	

Tabla 2. Contenido de un mensaje con información sensorial.

3.2.5.9.-.-Valores de calibración de los sensores táctiles

Estos sensores proporcionan unos valores de salida en el rango 0-1. Estos valores pueden ser incrementados si la carga es muy grande.

3.2.5.10.-Descripción de los sensores táctiles

Los sensores táctiles integrados en la Shadow utilizan “Quantum Tunnelling Composite (QTC)” como medio sensible. El QTC permite gran sensibilidad y un gran rango de medida.

Estos sensores táctiles están disponibles en dos formas diferentes, una para utilizarse en el pulgar y otra en los dedos. También están disponibles en dos tamaños, de 34 ó 22 elementos táctiles individuales (“Tactel”). Estas unidades individuales se encuentran distribuidas sobre la superficie del sensor. El pulgar y el resto de dedos tienen el mismo número de estos elementos, pero dispuestos de forma diferente. Cada una de estas células es sensible en un rango de cargas que va desde 0.1 N a 25 N.

3.2.5.11.-Características de los sensores táctiles

Electrónica compacta: Todo lo necesario en un solo chip (Amplificadores operacionales, convertidores A/D y D/A, drivers SPI,…)

Suministro eléctrico

Suministro eléctrico: 5 V regulados (opcional: 6-9 V no regulados)

Comunicación: Vía SPI. Opcionalmente se puede utilizar en bus CAN utilizando otro SPI y otro convertidor A/D.

Sensores táctiles: Mide fuerzas en un rango de 0.1N a 25N. Hay 22 ó 34 células individuales en cada sensor. Lectura rápida de los mismos (lectura completa en 50 Hz).

3.2.5.12.-Descripción mecánica de los sensores

Materiales

La carcasa de los sensores está fabricada con dos tipos de poliuretano. La región sensible se encuentra bajo una capa de poliuretano blando (de 1 ó 2mm). Esta parte es azul habitualmente. La parte superior del sensor está compuesta de poliuretano rígido, y habitualmente es blanca o negra.

Distribución de los sensores

Los sensores contienen 34 ó 22 células individuales distribuidas uniformemente sobre la superficie sensible. Estas células son de aproximadamente 3.3mm. de diámetro, y su forma ha sido diseñada para obtener la máxima cobertura sin puntos muertos.

3.2.5.13.-Especificaciones electrónicas

Forma de operación

Las células táctiles se comportan como si estuvieran situadas en el interior de una matriz, que se comprueba fila por fila. Para mostrar una célula, se utiliza la ayuda de un SWITCH, que selecciona una fila a mostrar. La selección de la columna se hace en función de la que se esté leyendo en ese momento.

Con la célula táctil sin carga el valor de la tensión leída será debido al convertidor D/A. Cuando la presión incrementa, lo hace la conductancia del sensor, pudiendo llegar a los 5 V. La tensión es amplificada por una PGA antes de ser muestreada por el convertidor A/D.

3.2.5.14.--Electrónica

Controlador del bus CAN en placa electrónica.

Sensorización en la palma: 7 convertidores A/D distribuidos a lo largo de la palma proveen de 26 canales activos de 12 bits.

3.2.5.15.-Microcontroladores

Se emplean PIC18F4580 para realizar un control embebido a través del sistema robótico. Todos los microcontroladores están conectados al bus CAN.

3.2.5.16.-Buses de comunicación.

El sistema de la mano se comunica con elementos exteriores a través de bus CAN. A través de este bus es posible acceder a los datos de los sensores, los componentes, la configuración y los puntos de consigna (setpoints) de los controladores.

Además, entre ciertos componentes del sistema, se utiliza una conexión interna basada en un bus SPI (Serial Peripheral Interfase). Este bus es usado, para la transferencia de información entre circuitos integrados.

3.2.5.17.-Configuración de las comunicaciones

El protocolo usado en el bus CAN permite una gran variedad de configuraciones específicas del sistema. Esto incluye:

Habilitar o deshabilitar un componente del robot.

Establecer velocidades de transmisión de los sensores.

Cambiar la dirección CAN usada por un componente.

Resetear un componente.

…

El bus está conectado con la placa principal presente en la parte trasera de la mano. El micro presente en ella se comunica con cada dedo a través de tres buses SPI de forma individual para cada dedo, y también tiene dos canales análogos conectados al pulgar, que posee un mayor número de conexiones.

El bus SPI de cada dedo se comunica con el convertidor A/D presente en la falange proximal (la más cercana a la palma), que se encarga de convertir las tensiones ofrecidas por los sensores situados en las distintas uniones de los dedos y con el PIC presente en la falange distal (la más alejada de la palma). El bus SPI se encarga también de comunicar la señal digital con el PIC situado en la palma, encargado de recolectar toda la información de los sensores. A dicho PIC también llega directamente la información de los sensores de aducción/abducción.

La placa principal presente en la palma está formada en realidad por tres placas interconectadas, proporcionando juntas las siguientes funciones:

Conversión de los 8V de funcionamiento del bus CAN a los 5V necesarios en la electrónica presente en los dedos.

Recogida de toda la información procedente de los sensores y de los convertidores A/D.

Transmitir toda esta información a través del bus CAN.

La comunicación de los sensores (táctiles y posicionales) se realiza mediante el protocolo de comunicaciones CAN. La mano dispone de un botón reset para reiniciar la comunicación CAN (figura 3). También dispone de unos LEDs que indican el estado del bus. En la siguiente figura se muestra la ubicación de estos elementos.

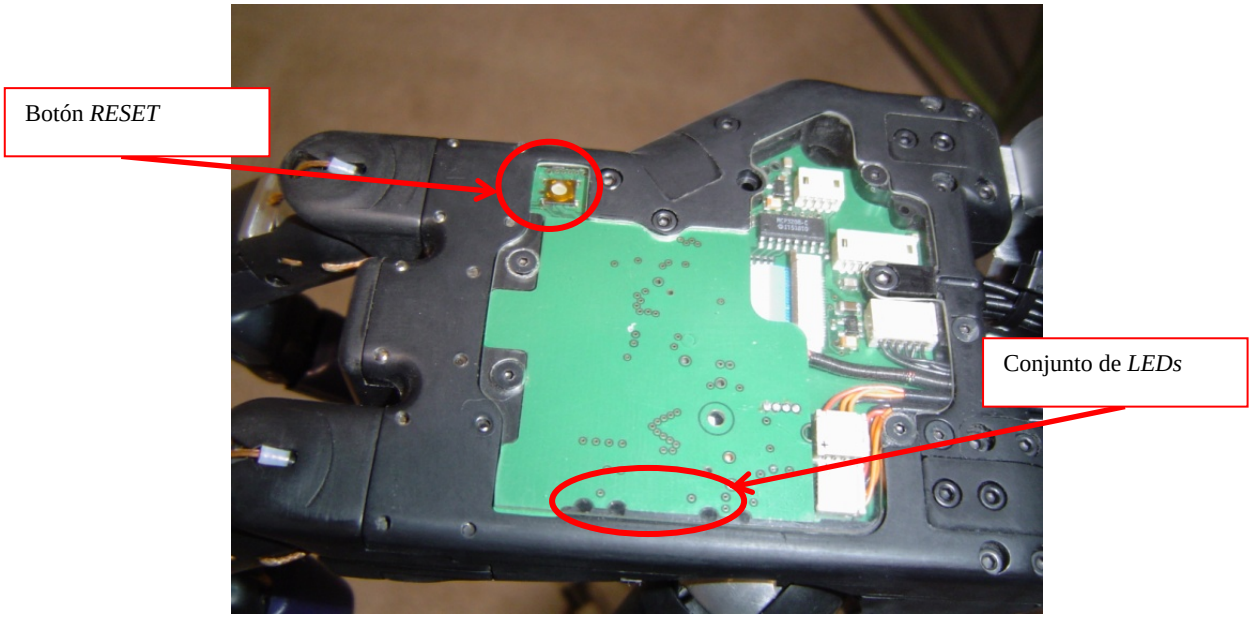


Figura 3.6.- Foto de la parte superior de la mano

3.2.6.-Investigaciones

La mano shadow está totalmente diseñada para fines de investigación, aún la utilidad de este elemento es bastante limitado. Ya que las actividades para las que se usa la robótica actualmente necesitan herramientas que ya van integradas en el extremo de las máquinas. Pero en un futuro se plantea introducirla en la vida de las personas con discapacidad, personas ancianas, y gente que necesite de una ayuda extra, y que añadiendo elementos agradables para el ser humano (una mano es mucho más agradable que una serie de tubos metálicos)

Actualmente hay varias universidades como la de Bielefeld que está centrada en proyectos de biorobótica, sobre todo en la parte de la búsqueda de objetos mediante el tacto. O como la universidad de Carnegie Mellon, que busca agarrar objetos de diferente masa y fragilidad.

La NASA se esta centrando en desarrollar el uso de la mano para realizar tareas precisas y delicadas en entornos diversos.

Aunque la mayoría de proyectos están centrados en la neurociencia, y la biorobótica.

Las ventajas que ven los investigadores en este campo es la posibilidad que tiene la mano a la hora de trabajar en entornos peligrosos, como por ejemplo con zonas con alto nivel de radiación o en entornos difíciles de acceso como laberintos de pasillos estrechos. Se plantea el uso de este tipo de manos en la medicina, como apoyo para el quirófano, u otros apoyos para el paciente. En los casos más extremos se está hasta para desactivar bombas.

También se puede usar para ayudar en la rehabilitación de pacientes, debido a las ventajas que plantea tener una mano con tanta libertad y sensibilidad.

También se puede usar para testear el efecto en una mano humana determinados objetos, como bolígrafos, guantes u otros elementos, sin poner en riesgo a ninguna persona, por problemas de ergonomía.

Las posibilidades de una mano así son muchas, lo importante es que se siga invirtiendo tiempo en la investigación, para que en un futuro se pueda usar para algo concreto que ayude a las personas, o a un entorno industrial.

3.3.- Brazo Robotnik

3.3.1.-Introducción al brazo

El ser humano está formado por 4 extremidades, 2 usadas para la movilidad, y otras 2 usadas para la realización de diversas actividades. El ser humano tiene una ventaja muy grande frente a otros animales, uno es el bipedismo, y otra es la longitud de sus extremidades, que aunque no sean las más destacables de la naturaleza, tienen ventaja por la flexibilidad y longitud frente a otras especies.

Los brazos es una extremidad que define claramente al ser humano, los hemos desarrollado en habilidad debido a que pasamos de usarlos para desplazarnos, a andar erguidos y usarlos para otras actividades.

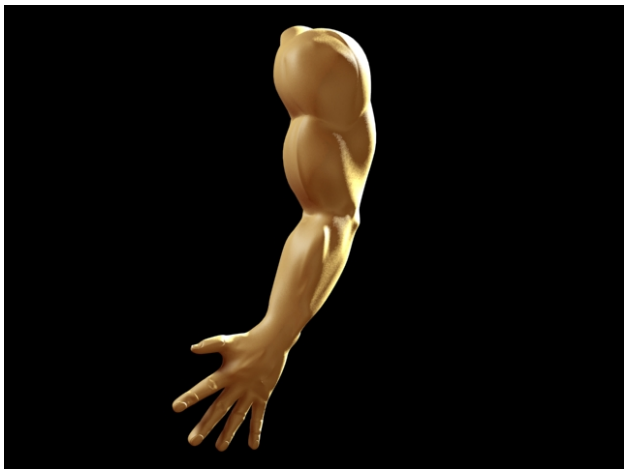


Figura 3.7.-. Muestra de un brazo humano

3.3.2.- Movimiento tridimensional

El movimiento del cuerpo humano, evidentemente, se realiza en tres dimensiones: cada segmento del cuerpo va asumiendo posiciones y orientaciones en el espacio a medida que se realiza el movimiento. Sin embargo, la comprensión del movimiento es más simple si se realiza en dos dimensiones, proyectando el movimiento en un plano adecuado.

En el caso de las extremidades superiores, la referencia de movimientos se define con respecto a una posición arbitraria, denominada posición anatómica, que corresponde a tener el brazo al costado del tórax con la palma de la mano hacia el frente. A partir de esta posición se identifican los movimientos de la cadena formada por los segmentos de la extremidad superior.

Tabla 1. Definición de segmentos del modelo del brazo humano.

SEGMENTO	DEFINICIÓN
Tórax	Origen del sistema articulado (Tierra).
Clavícula	Desde la articulación esterno-clavicular hasta la acromio-clavicular.
Húmero	Desde la articulación gleno-humeral hasta la articulación de codo.
Antebrazo	Desde el codo hasta la muñeca.
Mano	A partir de la muñeca.

Estos segmentos se unen por medio de cuatro articulaciones que proveen un total de nueve grados de libertad. Las articulaciones que se han incluido en el modelo son: la articulación esterno-clavicular que permite 2 grados de libertad, el hombro que permite 3 grados de libertad y funcionalmente es la unión de las articulaciones acromio-clavicular y gleno-humeral; el codo y la muñeca que permiten cada una 2 grados de libertad.

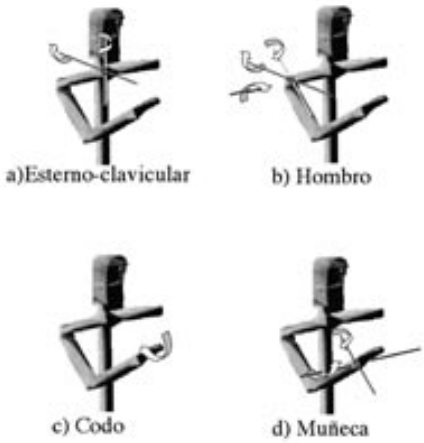


Figura 3.8.-.-Grados de giro de un brazo humano

3.3.3.- Introducción al uso de los brazos robóticos

Los brazos robóticos han sido implementados muy rápidamente en la industria. Actualmente vemos que su uso es muy amplio para introducirlos como una solución a un problema, y esa solución no es la única. Por ejemplo, centrándonos en un sector muy conocido, en el sector de la automoción, necesitamos un robot que no se mueva en el proceso, y a la vez tenga muchos grados de libertad debido a la deformación de la chapa o la necesidad de meterse por huecos en la carrocería. Otro ejemplo sería un robot de pintura de chapa, que debe de moverse por una superficie grande, pero no necesita tener una herramienta con mucha libertad de movimiento.

Figura 3.9.- Muestra de unos brazos trabajando en el sector de la automoción

Nuestro brazo robot está pensado en una manipulación con gran libertad de movimiento, pero sin necesidad de que se mueva por una superficie. Para hablar del robot voy a introducir una serie de términos que son interesantes conocer.

3.3.4.- Grados de libertad

Mecánicamente, un robot esta 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. De este modo son posibles los seis tipos diferentes de articulaciones. Cada uno de los 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.

El empleo de diferentes combinaciones de articulaciones en un robot, da lugar a diferentes configuraciones, con características a tener en cuenta tanto en el diseño y construcción del robot como en su aplicación. Las combinaciones más frecuentes son con tres articulaciones y que son las más importantes a la hora de posicionar su extremo en un punto en el espacio.

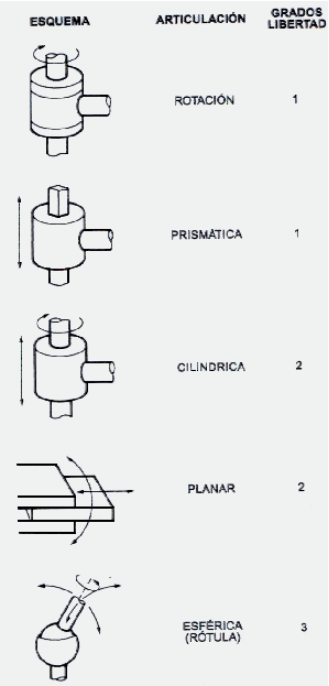


Figura 3.10.- Muestra de diferentes uniones móviles

Puesto que para posicionar y orientar un cuerpo de cualquier manera en el espacio son necesarios seis parámetros, tres para definir la posición y tres para la orientación, si se pretende que un robot posicione y oriente su extremo (y con el la pieza o herramienta manipulada) de cualquier modo en el espacio, se precisara al menos seis grados de libertad.

En la práctica, a pesar de ser necesarios los seis grados de libertad comentados para tener total libertad en el posicionado y orientación del extremo del robot, muchos robots industriales cuentan con solo cuatro o cinco grados de libertad, por ser estos suficientes para llevar a cabo las tareas que se encomiendan. Existen también casos opuestos, en los que se precisan más de seis grados de libertad para que el robot pueda tener acceso a todos los puntos de su entorno. Así, si se trabaja en un entorno con obstáculos, el dotar al robot de grados de libertad adicionales le permitirá acceder a posiciones y orientaciones de su extremo a las que, como consecuencia de los obstáculos, no hubieran llegado con seis grados de libertad. Otra situación frecuente es dotar al robot de un grado de libertad adicional que le permita desplazarse a lo largo de un carril aumentando así el volumen de su espacio al que puede acceder.

Cuando el número de grados de libertad del robot es mayor que los necesarios para realizar una determinada tarea se dicen que el robot es redundante.

3.3.5.- Sistemas de referencia

Para estudiar en robótica la posición del robot, primero debemos de aprender en que consisten términos como el sistema de referencia, ya que con este se determina la posición de nuestro brazo robot.

En física clásica un sistema de referencia se define por un par (P, E), donde el primer elemento P' es un punto de referencia arbitrario, normalmente perteneciente a un objeto físico, a partir del cual se consideran las distancias y las coordenadas de posición. El segundo elemento E es un conjunto de ejes de coordenadas. Los ejes de coordenadas tienen como origen de coordenadas en el punto de referencia (P), y sirven para determinar la dirección y el sentido del cuerpo en movimiento (o expresar respecto a ellos cualquier otra magnitud física vectorial o tensorial).

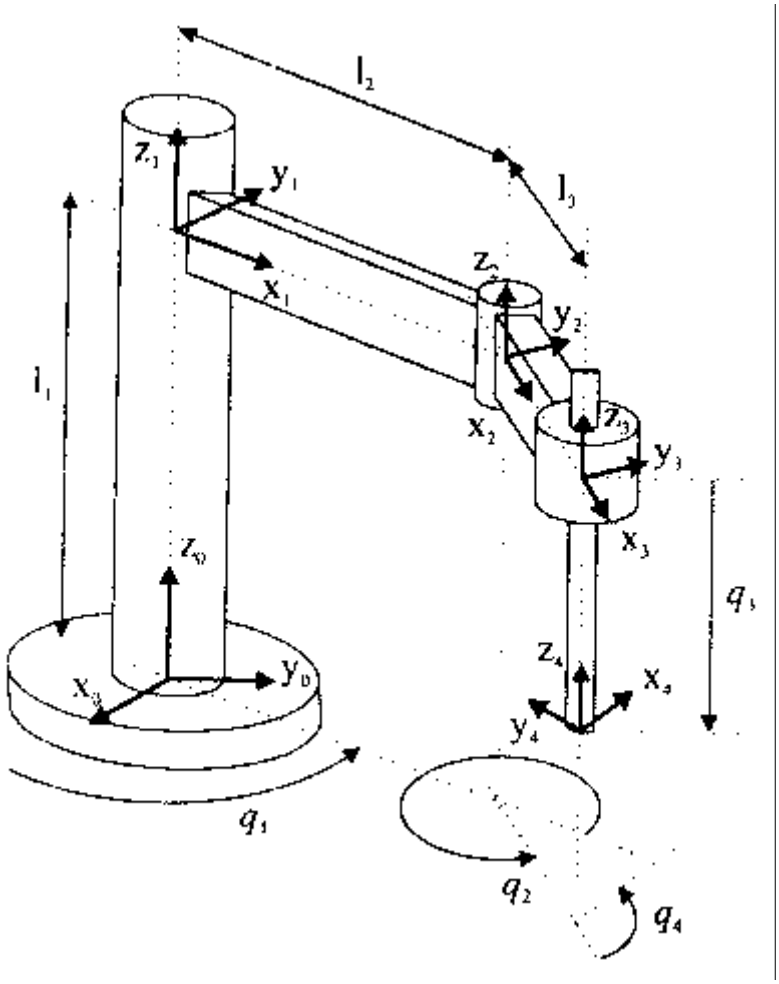


Figura 3.11.- Muestra de un brazo robótico con diferentes ejes de coordenadas

Un tercer elemento es el origen en el tiempo, un instante a partir del cual se mide el tiempo. Este instante acostumbra a coincidir con un suceso concreto. En cinemática el origen temporal coincide habitualmente con el inicio del movimiento que se estudia.

Estos tres elementos: punto de referencia, ejes de coordenadas y origen temporal, forman el sistema de referencia. Para poder utilizar un sistema de referencia, sin embargo, se necesitan unas unidades de medida que nos sirvan para medir. Las unidades son convencionales y se definen tomando como referencia elementos físicamente constantes. A un conjunto de unidades y sus relaciones se le llama sistema de unidades. En el Sistema Internacional de Unidades o S.I., se utiliza el metro como unidad del espacio y el segundo como unidad del tiempo.

3.3.6.-Cinemática del robot

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

A partir del sistema de referencia se describen las trayectorias, las velocidades y las aceleraciones de este. La velocidad es la rapidez con la que cambia de posición. Y la aceleración es la rapidez con la que cambia de velocidad.

Existen dos problemas fundamentales a resolver en la cinemática del robot; el primero es el problema cinemático directo, que consiste en determinar cuál es la posición y orientación del extremo del extremo final de robot, con respecto a un sistema de coordenadas que se toma como referencia, conocidos los valores de las articulaciones, y los parámetros geométricos de los elementos del robot; y el otro es el problema cinemático inverso, que resuelve la configuración que debe e adoptar el robot para una posición y orientación del extremo conocidas.

Para resolver estos problemas se usan modelos matemáticos matriciales para obtener la solución al problema, y ver que variables debemos de aplicar para obtener el resultado.

3.3.7.- Especificaciones del brazo robótico

El brazo robótico integra servoaccionamientos modulares, que integran motorreductores, etapa de potencia y controlador, de forma que el brazo resultante no requiere de un armario de control externo. La conexión del brazo se reduce a 2 hilos de alimentación y 2 hilos CAN bus.

Figura 3.12.-Esta figura muestra el brazo en su total extensión

Los elementos y el brazo se ajusta a los requerimientos de cada aplicación, pudiendo elegir el brazo integro, o rediseñar los elementos de unión con los parámetros de cada aplicación:

El brazo contiene
- Hasta 7 grados de libertad (en el mismo bus)
- Alcance de 400 a 1300mm
- Capacidad de carga util 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 serv o de corriente alterna y frenos electromagnéticos.
- Repetitividad posicional de 0.5mm
- Electrónica de control mediante tarjeta CAN-PCI desde el ordenador personal

3.3.7.1.-Especificaciones técnicas

El brazo robotizado esta formado por elementos modulares con motorreductor y controlador

Este brazo integra etapa de potencia y control, es el más adecuado para su instalación en unidades móviles

Las caracteristicas de control son:

- Alimentación a 24Vdc
- Control mediante bus CAN: solo necesita una tarjeta CAN y un equipo PC

La mecánica que lo compone es:

- Los motoreductores
- Los elementos de enlace entre módulos
- Los elementos de ensamble con el chasis
- El elemento de ensamblaje con el efector final
- Conectorizaciones
- Cableado

Complementos adicionales:

- Tarjeta CAN-PCI
- Drivers Linux para la tarjeta CAN
- Drivers Linux y Windows de los módulos
- Software de control (integra el control completo del brazo bajo Linux o Windows).

Los módulos trabajan como controladores distribuidos. El controlador superior o maestro puede ser un PC bajo la plataforma Windows. El controlador maestro se usa para generar la secuencia de programa o referencias, y para realizar el envió de este paso a paso a cada eje del sistema.

El control de corriente, velocidad de rotación y de posición tiene lugar en cada uno de los módulos, de igual forma que las operaciones de supervisión, de temperatura y de control de parada. No se requiere que el controlador superior tenga capacidades de tiempo real.

Figura 3.13.- Brazo Robotnik con una mano incluida

Características técnicas de los módulos:

- Cada eje contiene su propio controlador y servo amplificador
- Motores sin escobillas
- Eje pasante que permite el paso interior de los cables
- Encoder absoluto para posicionamiento y control de velocidad
- Monitorización de rango, límites, temperatura y corriente

- 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 multiples sistemas operativos

3.4.- Protocolo de comunicaciones CAN

3.4.1.-Introducción a las comunicaciones

El ser humano desde sus comienzos interactuó con un medio hostil, con especies muy peligrosas, y con una desventaja muy clara, como la falta de garras, dientes afilados, o una velocidad para moverse superior a sus presas. Para eso tuvo que usar su inteligencia y buscar herramientas con las que facilitar todas las tareas para igualarse a otras especies. La mayoría de cosas eran parecidas a lo que la naturaleza usaba, por ejemplo, como pueden ser los cuchillos, parecidos a los dientes de los depredadores más voraces.

Estas herramientas, útiles, y demás artículos se hacían a pequeña escala y para una utilidad muy determinada, y después de muchas horas de trabajo y esfuerzo. Durante la evolución de la sociedad se fue mejorando la forma de crear cosas. Hasta que en un momento de la historia se dio un salto industrial, se comprobó que si se usaban máquinas para realizar determinadas tareas repetitivas, se obtenían artículos en mucho menos tiempo y con una calidad bastante alta como para competir con artículos hechos a mano. Abaratando los precios y obteniendo una mayor velocidad de desarrollo de la sociedad.

Estos procesos industriales tenían un problema, y es que a diferencia de una persona, no tenían la capacidad para saber si lo que estaban haciendo estaba bien, un ejemplo es una sartén, si la chapa está mal en el proceso de doblado obtener una sartén deforme, que no es capaz de contener el aceite, convirtiéndose en un objeto inútil para lo que se ha diseñado, produciendo un gasto de materia prima. Lo primero que se hacía era depender de personas que iban controlando que todo estuviese bien. El inconveniente era que en muchos casos, se producían accidentes debido al cansancio, falta de atención, u otros motivos, haciendo que el proceso productivo se ralentizase.

Necesitaban algo que les proporcionase información del proceso que estaban realizando, y que alguien o algo pudiese actuar si ocurría alguna alteración. Esta información debía estar escalada, estar a tiempo real, y ser fiable. Se idearon los sensores, que son elementos que toman datos del proceso, como un termómetro, o sensor de posición entre otros muchos. También y muy importante, los actuadores, que son los que modifican el proceso, como por ejemplo un cilindro neumático, o un motor.

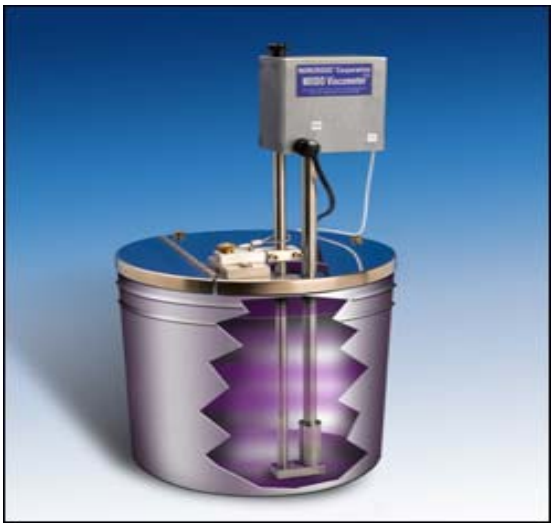


Fig. 3.14.- Un tanque de gas cerrado y con sensores en la parte superior

En el principio se usaron cuadros repletos de contactores electromecánicos con interconexión directa a cada elemento, mediante una línea de cable de dos puntos. Se utilizaban cableados de maniobra a 24 o 48 Voltios y líneas de corriente de 4-20mA. La informática se encontraba en un punto bastante poco desarrollado, con computadores generalmente usados para tareas administrativas. Los inconvenientes de estos cuadros eran muchos y variados, entre ellos la cantidad de cables, la dificultad de mantenimiento, y cuando se instalaba un nuevo componente había que reconstruir el sistema. La toma de datos era casi manual, y con una complejidad de trabajo bastante alta.

En la actualidad cada vez nos encontramos con procesos más complejos, con multitud de sensores y actuadores, que trabajan con multitud de entradas y salidas. También la necesidad cada vez más de centralizar el control de cualquier proceso. Esto demanda unos sistemas en continua evolución, que sean rápidos y robustos.

En la actualidad nos da dos opciones de trabajo, una es la conexión punto por punto, y la otra es la del bus de datos.

La conexión punto por punto es un método que se usa cuando disponemos de pocos equipos. Es un método para procesos sencillos, pero implica gran cableado, una dificultad a la hora de repararla y altos costes de instalación y mantenimiento.

La otra opción es crear un cable común para todos los equipos, que fuese rápido y seguro, y se idearon los llamados buses de campo. Trabajan con señales digitales, con pocos cables, una estructura de red seriada. Los nodos tienen un sistema inteligente para enviar los datos por el bus y que sea comprendido por los otros nodos con métodos de control de errores.

3.4.2.- Transmisión de datos y protocolo OSI

Como hemos citado anteriormente, los buses de Campo es una idea que se centra en la eliminación de cables inútiles, aumentando el rendimiento del proceso y disminuyendo los costes.

A principios de la década de los 80 el desarrollo de redes sucedió con desorden en muchos sentidos. Se produjo un enorme crecimiento en la cantidad y el tamaño de las redes. A medida que las empresas tomaron conciencia de las ventajas de usar tecnologías de conexión, las redes se agregaban o expandían a casi la misma velocidad a la que se introducían las nuevas tecnologías de red.

Para introducir el siguiente párrafo vamos a hablar de lo que es la comunicación, ya que ese fue el problema más importante que se enfrentaron en aquella época. Los ordenadores parten de que solo entienden un idioma, que es el código binario. Cada bit puede significar una cosa, por ejemplo, supongamos un caso de transmisión de datos entre dos ordenadores conectados a un mismo cable, donde enviamos un paquete de 16 bits por un cable de dos puntos. Supongamos que lo enviamos a una velocidad determinada, donde los primeros 2 bits son para avisar al otro ordenador que está a punto de empezar la trama, los 8 siguientes son los datos que se desea transmitir, y el resto son código de error. Esto puede ser una trama sencilla de datos. Imaginemos que el otro ordenador no tiene la misma configuración, la velocidad de recepción es la mitad que la de transmisión es la mitad que la de recepción Se perderían la mitad de bits. Supongamos otro posible error, el primer ordenador toma como bits para avisar que va a trasmitir los 2 primeros y el segundo toma como bits para iniciar la transmisión los 4 primeros. Esto es una de las muchas opciones configurables, que hace que sea imposible crear un sistema común de transmisión, ya que cada fabricante crea sus productos y sus propios sistemas de transmisión.

Lo citado anteriormente se puede resumir en que al principio cada fabricante planteaba sus propias normas para la transmisión de datos, esto produjo dificultades a la hora de transmitir datos entre dispositivos de diferentes proveedores. Estos sistemas se llamaban sistemas cerrados donde era muy difícil añadir dispositivos que no estuviesen pensados para esa red en particular. Las diferentes organizaciones intentaron plantear un sistema común para todas las redes de transmisión a base de hacer su sistema el mejor por encima de los demás. Pero entonces en 1977 la Organización Internacional de Estandarización (ISO) se planteó un sistema común a todos que se llamó OSI (Open Sistem Interconnection).

Este método intento plantear 7 capsulas aisladas, donde cada fabricante era libre de administrarlas como quisiese, pero cada una debía de tener una serie de cosas que la hiciese comunes a todas las demás, y así trabajar con otros fabricantes

Fig.3.15.- Protocolo de capas

Cada capa plantea una parte determinada de la comunicación de datos. El nivel físico, se centra en el cable en sí y todo lo que le rodea, de que material es, la tasa de transmisión, la codificación. La capa de enlace se encarga de comprobar que ha llegado el mensaje sin errores. La capa de red se encarga de buscar el camino por el que se van a enviar los datos. La capa de transporte comunica las capas inferiores con las superiores, divide en bloques o paquetes los datos, comprueba que no falta ningún paquete (para recibir el mensaje completo). La capa de sesión se encarga de avisar que hay una transmisión, de decir cuando se puede transmitir. La capa de presentación se encarga de traducir los datos que recibe a unos que pueda comprender. La capa de aplicación es la cabeza de todo, es la que se va a encargar de convertir los datos en algo útil para trabajar con ellos. Ya que si se transmiten datos sin ningún fin, es como no hacer nada.

3.4.3.- Buses de campo

En los buses de campo se trabaja con señales digitales, con pocos cables, una estructura de red seriada. Los nodos tienen un sistema inteligente para enviar los datos por el bus y que sea comprendido por los otros nodos con métodos de control de errores.

Los buses de campo trabajan en serie, destinados para aplicaciones en tiempo real, se puede usar para controlar tanto procesos sencillos, como procesos complejos. Los buses eliminan cableado entre todos los componentes del proceso, como ayuda a integrar componentes de diferentes fabricantes sin que se produzcan incompatibilidades.

Aparte permiten enviar datos rápidamente y con bastante seguridad. Esto es interesante ya que podemos tener elementos separados espacialmente o en zonas de difícil acceso.

El crecimiento del uso de los buses de campo se ha debido principalmente a la rápida evolución de los microprocesadores, y al abaratamiento entre estos.

En un bus nos podemos encontramos los siguientes dispositivos conectados:

Nivel de entradas/salidas: es el nivel más próximo al proceso, es con el equipo con el que opera la empresa, suelen ser los sensores y actuadotes para la toma de medidas y realización de actividades

Nivel de campo y proceso: integra pequeños automatismos (PLC's,…) en subredes o 'islas'. En el nivel más alto podemos encontrar uno o varios autómatas modulares actuando como maestros de la red o maestros flotantes.

Nivel de control: enlaza las células de trabajo o zonas de trabajo. A este nivel se sitúan los autómatas de gama alta y los ordenadores dedicados al diseño, control de calidad, programación, etc.

3.4.4.-El bus CAN

CAN es un protocolo de comunicaciones desarrollado por la firma alemana Robert Bosch GmbH, basado en una topología de bus para la transmisión de mensajes en ambientes distribuidos. Además ofrece una solución a la gestión de la comunicación entre múltiples unidades centrales de proceso. Este protocolo se desarrollo sobre todo para el sistema ECU de algunas marcas de coches (mercedes-BENZ)

El protocolo CAN (Controller Area Network), desarrollado a mitad de los ochenta, nació para proporcionar una comunicación serie robusta. Sus principales aplicaciones son la automoción, equipos de testeo, equipos médicos...

El protocolo CAN es un protocolo normalizado, así que es compatible con la mayoría de componentes de la industria.

Es un sistema multimaestro, cada ordenador decide si necesita enviar datos, evitando así tener que tener un ordenador reservado para controlar la comunicación entre los diferentes componentes.

3.4.5.- Ventajas del protocolo CAN

Es posible destacar las siguientes propiedades del protocolo del bus CAN:

Priorización de mensajes.
Garantía de los tiempos de retardos.
Flexibilidad en la configuración
Recepción múltiple con tiempos de sincronización
Robustez en sistemas de amplios datos
Multimaestro
Detección de error y señalización
Retransmisión automática de mensajes corruptos tan pronto como el bus está libre de nuevo.
Distinción entre errores temporales y fallos permanentes de nodos y desconexión automática de nodos defectuosos.

3.4.6.- Capas del bus CAN

CAN es un protocolo de comunicaciones serie que soporta control distribuido en tiempo real con un alto nivel de seguridad y multiplexación.

El establecimiento de una red CAN para interconectar los dispositivos electrónicos internos de un vehículo tiene la finalidad de sustituir o eliminar el cableado. Las ECUs (sensores, sistemas antideslizantes, etc.) se conectan mediante una red CAN.

De acuerdo al modelo de referencia OSI (Open Systems Interconnection), la arquitectura de protocolos CAN incluye tres capas: física, de enlace de datos y aplicación, además de una capa especial para gestión y control del nodo llamada capa de supervisor.

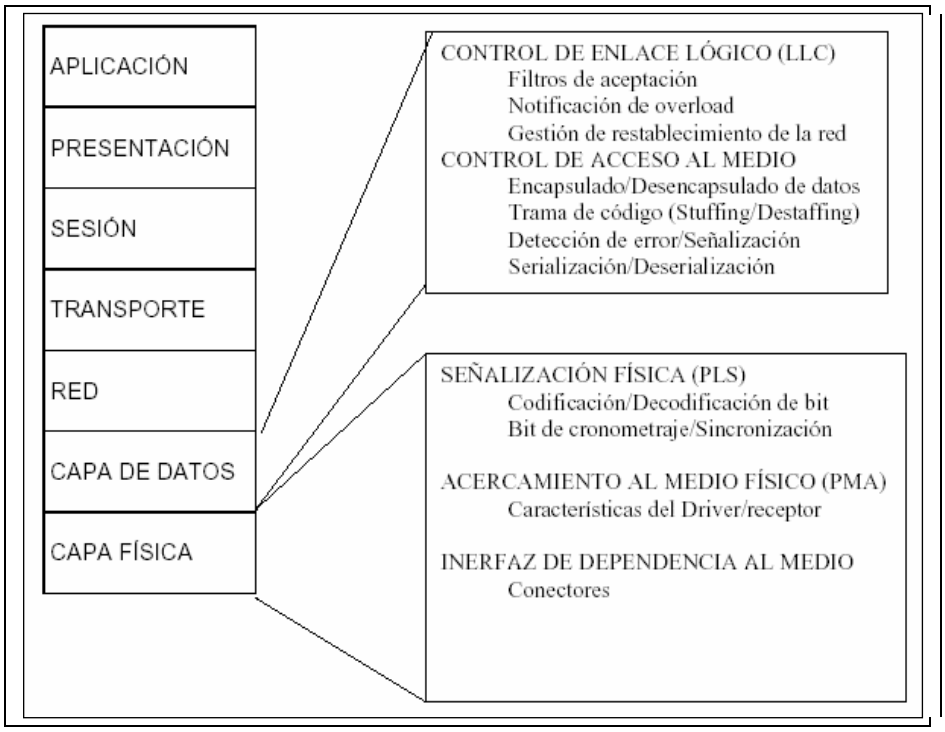


Fig. 3.16.- Capas que integran el bus CAN

Capa física: define los aspectos del medio físico para la transmisión de datos entre nodos de una red CAN, los más importantes son niveles de señal, representación, sincronización y tiempos en los que los bits se transfieren al bus. La especificación del protocolo CAN no define una capa física, sin embargo, los estándares ISO 11898 establecen las características que deben cumplir las aplicaciones para la transferencia en alta y baja velocidad.

Capa de enlace de datos: 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

Cuando un nodo necesita enviar información a través de una red CAN, puede ocurrir que varios nodos intenten transmitir simultáneamente. CAN resuelve lo anterior al asignar prioridades mediante el identificador de cada mensaje, donde dicha asignación se realiza durante el diseño del sistema en forma de números binarios y no puede modificarse dinámicamente. El identificador con el menor número binario es el que tiene mayor prioridad.

El método de acceso al medio utilizado es el de Acceso Múltiple por Detección de Portadora, con Detección de Colisiones y Arbitraje por Prioridad de Mensaje (CSMA/CD+AMP, Carrier Sense Multiple Access with Collision Detection and Arbitration Message Priority). De acuerdo con este método, los nodos en la red que necesitan transmitir información deben esperar a que el

bus esté libre (detección de portadora); cuando se cumple esta condición, dichos nodos transmiten un bit de inicio (acceso múltiple). Cada nodo lee el bus bit a bit durante la transmisión de la trama y comparan el valor transmitido con el valor recibido; mientras los valores sean idénticos, el nodo continúa con la transmisión; si se detecta una diferencia en los valores de los bits, se lleva a cabo el mecanismo de arbitraje.

CAN establece dos formatos de tramas de datos (data frame) que difieren en la longitud del campo del identificador, las tramas estándares (standard frame) con un identificador de 11 bits definidas en la especificación CAN 2.0A, y las tramas extendidas (extended frame) con un identificador de 29 bits definidas en la especificación CAN 2.0B.

Para la transmisión y control de mensajes CAN, se definen cuatro tipos de tramas: de datos, remota (remote frame), de error (error frame) y de sobrecarga (overload frame). Las tramas remotas también se establecen en ambos formatos, estándar y extendido, y tanto las tramas de datos como las remotas se separan de tramas precedentes mediante espacios entre tramas (interframe space).

En cuanto a la detección y manejo de errores, un controlador CAN cuenta con la capacidad de detectar y manejar los errores que surjan en una red. Todo error detectado por un nodo, se notifica inmediatamente al resto de los nodos.

Capa de 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 error 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.

Capa de aplicación: Existen diferentes estándares que definen la capa de aplicación; algunos son muy específicos y están relacionados con sus campos de aplicación. Entre las capas de aplicación más utilizadas cabe mencionar CAL, CANopen, DeviceNet, SDS (Smart Distributed System), OSEK, CANKingdom.

El número máximo de nodos no está limitado por la especificación básica y depende de las características de los transceptores, las especificaciones de buses de campo lo limitan a 32 o 64 en una red sin repetidores.

En un bus según ISO 11898 se considera bit dominante aquel en que la tensión diferencial entre las líneas del bus V CAN_H V CAN_L supera los +0.9 V y un bit recesivo cuando la diferencia es inferior a +0.5 V. Las tensiones nominales en estado dominante son 3.5 V para CAN_H y 1.5 V para CAN_L

respecto a la masa de referencia, en estado recesivo son de 2.5 V en ambas líneas, figura B.3.

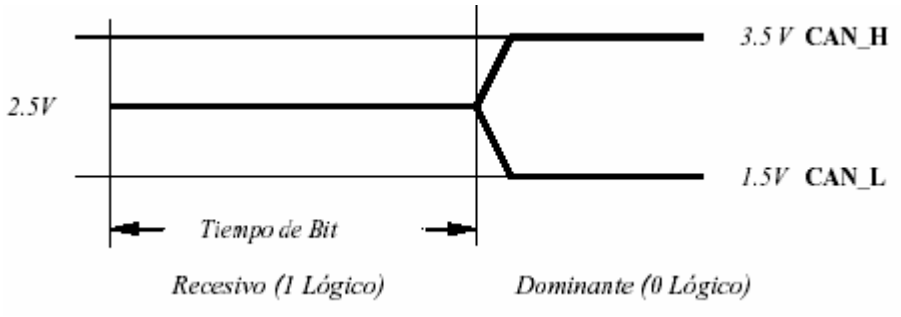
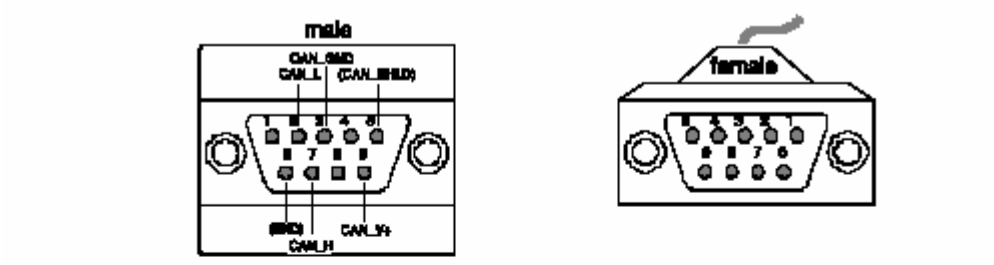


Figura 3.17.-Niveles eléctricos del bus CAN

En los buses CAN normalizados los conectores, figura B.4, usualmente utilizados son el SubD-9, el mini de 5 patillas o el tipo regleta.



Patilla	Señal	Descripción
1	-	Reservado
2	CAN_L	Línea CAN_L (para bit dominante a nivel bajo)
3	CAN_GND	Masa de CAN
4	-	Reservado
5	(CAN_SHLD)	Masa opcional para pantalla
6	(GND)	Masa de CAN opcional
7	CAN_H	Línea CAN_H (para bit dominante a nivel alto)
8	-	Reservado (línea de error)
9	(CAN_V+)	Alimentación, por si se necesitan aislar el bus (optoacopladores)

Figura 3.18.- Conector para el bus CAN

3.4.7.- Características de los Mensajes en CAN

La información del bus se envía en un formato fijo de mensajes con diferente pero limitada longitud. Cuando el bus está libre cualquier unidad conectada puede comenzar a trasmitir un mensaje nuevo.

En los sistemas CAN, un nodo no utiliza toda la información que se procesa. Esto tiene varias consecuencias importantes:

Flexibilidad del sistema: Los nodos se pueden añadir a la red CAN sin realizar ningún cambio en el software o hardware de cualquier nodo y de la capa de aplicación.

Ruta de los mensajes: El contenido de un mensaje se especifica con un identificador. El identificador no indica el destino del mensaje, pero describe el significado de los datos, por lo tanto todos los nodos de la red pueden decidir por filtraje de los mensajes si los datos han de ser descartados o ser procesados.

Multicasting: Como consecuencia del concepto de filtrado de mensajes, cualquier cantidad de nodos pueden recibir y actuar bajo el mismo mensaje simultáneamente.

Robustez de los datos: Dentro de la red CAN se garantiza que un mensaje sea simultáneamente aceptado o bien por todos lo nodos o por ninguno. Además la robustez de los datos del sistema se alcanza gracias al concepto de multicasting y al manejo de errores.

La velocidad de transmisión del CAN puede ser diferente en distintos sistemas. Pero para un sistema en concreto la velocidad de transmisión debe ser uniforme y fija.

El identificador define la prioridad de un mensaje durante el acceso al bus.

Cuando se envía una trama remota se está realizando una petición de datos que podría necesitar a otro nodo enviando la correspondiente trama de datos. La trama de datos y la correspondiente trama remota se nombran con el mismo identificador.

Cuando el bus está libre, cualquier unidad puede comenzar a trasmitir un mensaje. Si dos o más unidades comienzan a trasmitir un mensaje al mismo tiempo, el conflicto de acceso al bus es resuelto por arbitraje utilizando el identificador. El mecanismo de arbitraje garantiza que ni la información ni el tiempo se pierdan. Si una trama de datos y una trama remota con el mismo identificador se inician al mismo tiempo, la trama de datos prevalece sobre la trama remota. Durante el arbitraje todos los transmisores comparan el nivel del bit trasmitido con el nivel que se lee en el bus. Si estos niveles son iguales, la unidad puede enviar. Cuando un bit recesivo se envía y se monitoriza un bit dominante, entonces la unidad pierde arbitraje y debe retirarse sin enviar ningún otro bit.

Con el fin de alcanzar la máxima seguridad en la transferencia de datos se implementan en todos los nodos del CAN medidas especiales para la detección de errores, señalización y autochequeo.

Detección de error

Para la detección de errores se siguen las siguientes medidas:

Monitorización (los transmisores comparan los niveles de bit que se envían con el nivel de bit que hay en el bus)

Código de redundancia cíclica (CRC)

Bit Stuffing

Chequeo de la trama de mensaje

Mecanismo de detección de errores

Los mecanismos de detección de errores tienen las siguientes propiedades:

Detección de errores globales

Detección de los errores locales a trasmitir

Detección de más de cinco errores distribuidos aleatoriamente en el mensaje.

Detección de errores con una longitud menor de 15.

Señalización de error y tiempo de restablecimiento

Los mensajes corruptos son reconocidos por cualquier nodo que detecta un error. Estos mensajes se abortan y se retransmiten automáticamente. El tiempo de restablecimiento desde la detección de un error hasta el comienzo del próximo mensaje es como mucho de 31 bits, si no hay más errores.

Limitación de fallos

Los nodos CAN son capaces de distinguir entre perturbaciones cortas y fallos permanentes. Estos nodos por defecto se desconectan.

El enlace de la comunicación serie CAN es un bus al que se le puede conectar un número de unidades. Este número no tiene un límite teórico. En la práctica el número total de unidades estará limitado por el tiempo de retardo y /o cargas eléctricas de la línea del bus.

El bus se compone de un único canal que transporta bits. Desde esta resincronización de los datos se puede derivar la información. La manera en que este canal se implementa no se fija en esta especificación. Por ejemplo si es de un único hilo, dos hilos diferenciales, fibra óptica, etc.

El bus puede tener uno o dos valores lógicos: dominante o recesivo. Durante la transmisión simultanea de bits dominantes o recesivos, el resultado del estado lógico del bus será dominante.

Todos los receptores comprueban la fiabilidad del mensaje recibido, reconociendo un mensaje aceptable y desechando un mensaje inaceptable.

Para reducir el consumo de potencia del sistema, cualquier nodo CAN puede entrar en modo sleep sin ningún tipo de actividad interna y sin la conexión de los drivers. El modo sleep finaliza cuando se activa el bus o debido a condiciones internas del sistema. Cuando se despierta, la actividad interna se reinicializa, aunque la subcapa de control de acceso al medio se esperará hasta que el oscilador del sistema se estabilice y entonces esperará a que se haya sincronizado la actividad del bus.

Por regla general las aplicaciones con velocidades de transmisión superior a los 125kbit/s necesitan para obtener un adecuado bit de cronometraje de resonadores cerámicos.

3.4.8.- Descripción de las tramas de mensajes CAN

El protocolo CAN define cuatro diferentes tipos de mensajes. La primera y más común es la trama de datos (Data Frame), figura B.5. Ésta es utilizada cuando un nodo trasmite información a cualquiera de los otros nodos del sistema. La segunda es la trama remota (Remote Frame), figura B.6, que es básicamente una trama de datos con el bit RTR a 1. Los otros dos tipos de tramas son para el manejo de errores. Uno se llama trama de error y la otra trama de overload o sobrecarga. Las tramas de error son generadas por nodos que detectan cualquiera de los errores de protocolo definidos por el CAN. Los errores de overload son generados por nodos que necesitan más tiempo para procesar los mensajes ya recibidos.

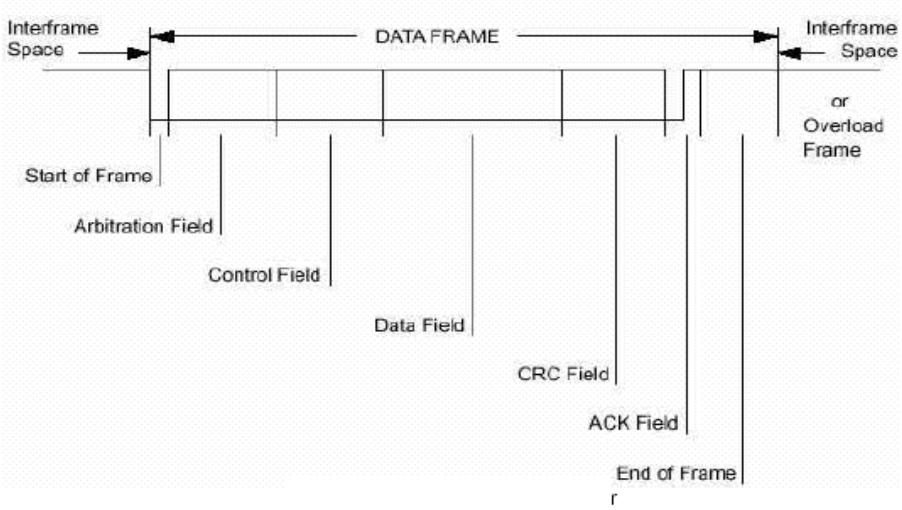


Figura 3.19.-Trama de datos estándar

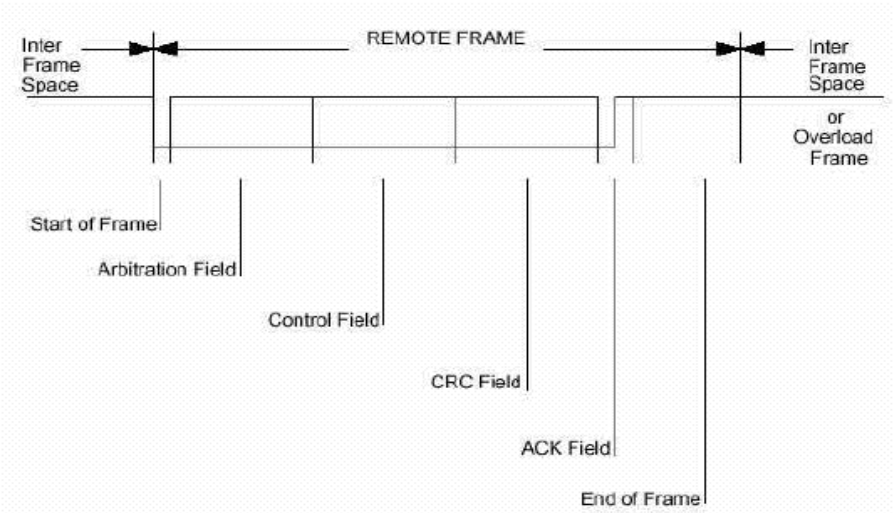


Figura 3.20.-Trama remota

La trama de datos consiste en campos que proporcionan información adicional sobre los mensajes definidos según las especificaciones del CAN. Dentro de la trama de datos están los campos de arbitraje, los campos de control, campos de datos, campo de CRC, campo de ACK y final de trama.

El campo de arbitraje se utiliza para priorizar los mensajes en el bus. Éste consiste en 12 bits (11 bits de identificación y un bit RTR) o 32 bits (29 bit de identificación, 1 bit para definir el mensaje como trama de datos extendida, un bit SRR que no es utilizado, y un bit RTR), dependiendo de si la trama es extendida o estándar se utiliza uno u otro.

Como se ha dicho previamente, la petición de trasmisión remota (RTR) se realiza cuando un nodo necesita que se le envíe información de otro nodo.

Para ello, se inicia la trama remota con el identificador de la trama de datos requerida. El bit RTR en el campo de arbitraje se utiliza para diferenciar entre trama remota y trama de datos. Si el bit RTR es recesivo, el mensaje es remoto, si es dominante el mensaje es una trama de datos.

El campo de control consiste en seis bits. El de mayor peso es el bit IDE (significa trama extendida) que deberá ser dominante para una trama de datos estándar. Este bit determina si el mensaje es una trama estándar o extendida, figura B.7.

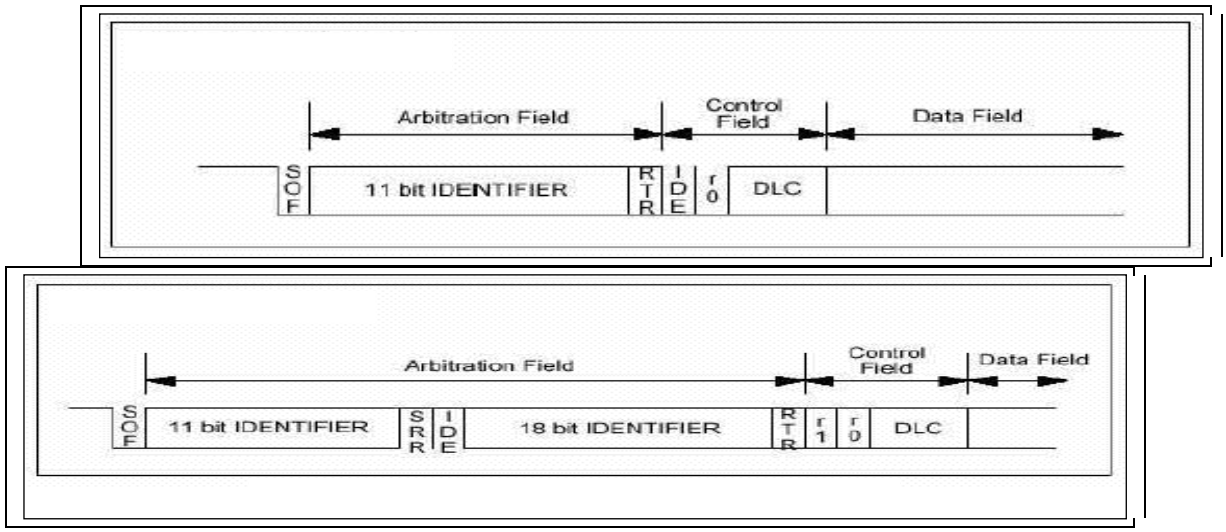


Figura 3.21.- Trama de datos estándar (arriba) y trama de datos extendida (abajo)

En tramas extendidas, este bit es RB1 y está reservado. El próximo bit es RB0 y también es reservado. Los cuatro de menor peso son la longitud de los datos (DLC). Estos bits determinan la cantidad de bytes que se van a incluir en el mensaje. Hay que tener en cuenta que las tramas remotas no tienen campo de datos, a pesar de contener el valor de los bits del DLC.

El campo de datos consiste en un número de bytes de datos determinado por el valor de bits del DLC.

El campo CRC consiste en un campo de 15 bits y un delimitador CRC, utilizado para nodos receptores que determinan si han ocurrido errores de trasmisión.

El campo de confirmación (ACK) se utiliza para indicar si el mensaje se ha recibido correctamente. Cualquier nodo que haya recibido correctamente el mensaje, a pesar de si el nodo procesa o descarta los datos, pone un bit dominante en el flag ACK de la trama.

Los dos últimos mensajes son tramas de error y tramas de overload, figuras B.8 y B.9 respectivamente. Cuando un nodo detecta uno de los muchos tipos de errores definidos por el protocolo CAN, aparecerá una trama de Error. Las tramas de overload indican a la red que el nodo que envía la trama de overload no está preparado para recibir un mensaje adicional en ese momento, o que la intermisión ha sido violada.

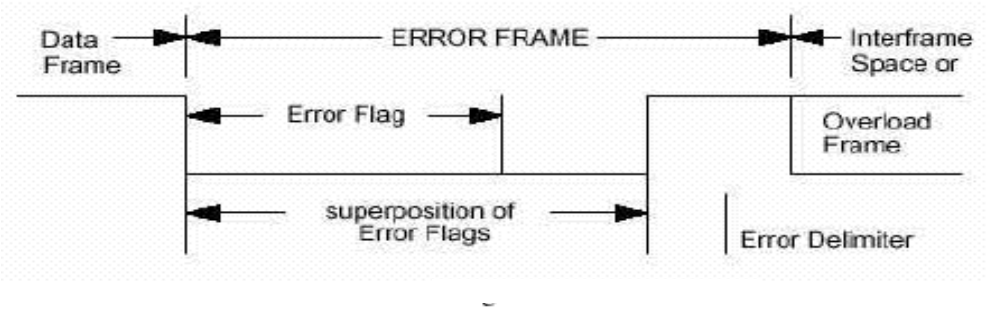


Figura 3.22.- Trama de error

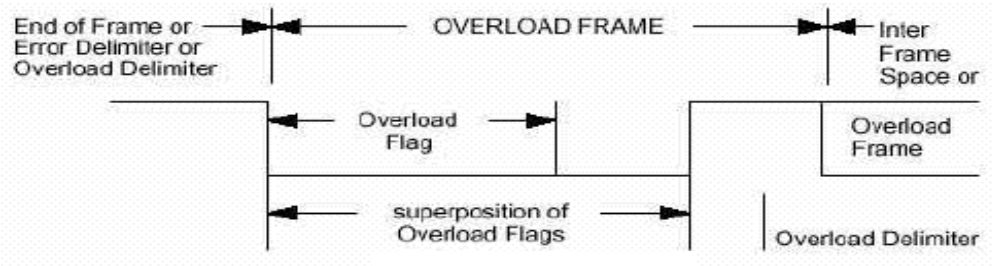


Figura 3.23.- Trama de overload

3.3.9.- Detección de errores

Como se ha comentado anteriormente, hay cinco condiciones de error, que son definidas en el protocolo CAN y tres estados de error, según el tipo y número de condiciones de detección de error. La siguiente sección describe cada uno de ellos con más detalle.

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.

Aunque al menos un nodo no reciba correctamente el mensaje, éste es reenviado después del apropiado tiempo de intermisión.

Error de ACK

En el campo de acknowledge de un mensaje, el nodo trasmisor comprueba si el flag 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.

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.

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 flag ACK, no se genera bit de error ya que se está produciendo un arbitraje o ACK normal.

Error de stuff

El protocolo CAN utiliza el método “No Retorno a Cero (NRZ). Quiere decir, que el nivel de bit toma lugar en el bus durante todo el tiempo de bit. CAN es también asíncrono, y se utiliza un bit de stuffing 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 stuffing). El nodo receptor lo utilizará para sincronización, pero ignorará el bit de stuff 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 stuffing habrá sido violado. Un error de stuff aparece, así pues se envía una trama de error, y el mensaje se repite.

Error de estado

La detección de errores se hace pública a todos los otros nodos a través de tramas de error o flags de error. La trasmisión de un mensaje erróneo se aborta y la trama se repite tan pronto el mensaje puede otra vez ganar arbitraje sobre la red. Además, cada nodo tiene uno de los tres estados de error, Error-Activo, Error-Pasivo o Bus-off.

Error activo

Un nodo con error activo puede activamente participar en la comunicación del bus, incluyendo el envío de flag de error activo. El flag de error viola las reglas de bit stuffing y causa a todos los otros nodos el envío de un flag de error, llamado “Error Echo Flag”, como respuesta. Un flag de error activo, y el subsiguiente flag de eco de error puede causar doce bits dominantes consecutivos en el bus; seis del flag de error activo y de cero a seis del flag de eco de error, dependiendo de cuando cada nodo detecta un error en el bus. Un nodo es error activo cuando el contador de error de trasmisión y el contador de error de recepción es más bajo de 128. El error activo es el modo de operación normal, permitiendo al nodo trasmitir y recibir sin restricciones.

Error pasivo

Un nodo se pone en error pasivo cuando o el contador de error de trasmisión o el contador de error de recepción se excede de 127. El nodo de error pasivo no está permitido para trasmitir flags de error activo, pero además, los flags de error trasmitido consisten en seis bits recesivos. Si el nodo de error pasivo es el único que esta trasmitiendo en el bus, el flag se error pasivo violará las reglas de bit stuffing y el nodo receptor responderá con flags de error del mismo (el error pasivo o error activo dependen de su estado de error). Si el nodo de error pasivo en cuestión no es el único que trasmite (por ejemplo durante el arbitraje) o si es un receptor, el flag de error pasivo no tendrá efecto en el bus, debido a la naturaleza recesiva del flag de error. Cuando un nodo de error pasivo trasmite un flag de error pasivo y detecta un bit dominante, debe ver el bus como libre durante ocho bits adicionales después de una intermisión antes de reconocer el bus como disponible. Después de este tiempo, intentará retrasmitir.

Bus off

Un nodo entra en estado bus-off cuando el contador de error de trasmisión es mayor de 255 (la recepción de errores no puede motivar que un nodo pase al estado de bus off). En este modo, el nodo no puede enviar o recibir mensajes, mensajes de confirmación, o trasmitir tramas de errores de cualquier clase.

Este es el tipo de limitación que se produce. Hay una recuperación del bus que se define en el protocolo CAN y permite a un nodo que está bus-off recuperarse, volver a error activo, y comenzar a trasmitir otra vez si desaparece el fallo.

4.1.-Introducción

En el siglo XXI nos encontramos con una sociedad muy evolucionada, con niveles de tecnología que en las novelas de hace unos siglos ni se aproximaban. Somos capaces de enviar satélites al espacio. Somos capaces de ver a tiempo real lo que ocurre en cualquier punto del planeta. Esto viene apoyado por la informática, elementos que suprimen tareas repetitivas como cálculos, rutinas de operación entre muchas cosas más.

La informática en la actualidad es tan común e importante, que se ha implantado en todas las tareas del día, siendo un apoyo importante, nos encontramos con pequeños microprocesadores en todo el ámbito industrial, y del día a día. Hoy en día ya es impensable realizar muchas actividades sin el uso del ordenador. Algunos de los ejemplos son los siguientes.

-En la industria se puede usar para controlar el proceso desde tu casa mediante una conexión a los equipos que se encuentran en la fábrica.
-En el desarrollo de proyectos de investigación, los datos tomados son tratados por el ordenador, por ejemplo cada segundo durante una semana, con una precisión tremenda, y dando en pantalla los resultados a tiempo real.
-En el deporte, los ordenadores analizan los movimientos del deportista, y optimizan estos para aumentar sus resultados.
-En las casas del futuro se habla de un ordenador central conectado al teléfono, donde podremos operar desde el móvil y a distancia para controlar todos los procesos que ocurren en la casa

Fig. 4.1.- Ejemplo de casa con un sistema de control central

La expansión de los ordenadores se debe a dos cosas muy importantes; la primera es el abaratamiento de estos, tanto en su versión sobremesa, como en la actualidad de su versión portátil. La segunda es el aumento de la velocidad de trabajo, con el consiguiente, aumento de la resolución de

problemas complejos en un tiempo mínimo. Aparte, en la actualidad un ordenador, por ejemplo, puede realizar varias tareas complejas a la vez, crear complejas imágenes en tres dimensiones y modificarlas, u operaciones de cálculo muy complejas en muy poco tiempo. Esto hace indispensable su uso para ahorrar mucho tiempo en actividades de diferentes ámbitos de la industria y en ambientes domésticos.

El inconveniente de los microprocesadores que controlan los ordenadores es que ellos solo entienden un lenguaje, que es el código binario, que es un código sencillo de unos y ceros, este lenguaje como se puede ver, no ofrece muchas ventajas, ya que está muy limitado. Según se ha evolucionado en el tiempo, se han inventado herramientas que ayudan a que el ordenador y el programador sean capaces de entenderse. Se empezó programando en binario, con un lenguaje muy difícil, y que necesitaba de un nivel de conocimientos de programación muy altos, y de muchas horas de trabajo. Luego se consiguió llegar al lenguaje ensamblador, que es un lenguaje que es intermedio entre el nuestro y el suyo, son instrucciones sencillas que escribe el programador, y que mediante un traductor o compilador, convierte en tramas binarias. Luego se fueron mejorando las herramientas de programación hasta obtener lenguajes con un parecido muy grande al lenguaje humano, y con grandes posibilidades de operación.

El problema es que la mayoría de programas que se diseñaban se necesitaba de una persona con altos conocimientos de informática, que solo se dedicase a eso, y teniendo en cuenta que cuando se actualizaban los programas, se debía de estudiar casi todo de nuevo. A partir de esto diferentes empresas se plantearon crear una herramienta que permitiese usar el ordenador por cualquier persona, una herramienta sencilla, que con un nivel bajo de conocimientos de informática se pudiese trabajar. Con esta idea se buscaba impulsar la venta de ordenadores, ya que si conseguían una herramienta estable de trabajo capaz de ser usada por todos, se obtendría unos beneficios considerables. Esta es la idea de sistema operativo, un entorno que sea fácil de usar por un usuario, y que aparte este programa gestione los recursos del ordenador para mejorar el rendimiento de este. Aparte este sistema operativo tendrá programas que le darán mayor funcionalidad a este como por ejemplo un procesador de textos, o una calculadora.

El inconveniente de un sistema operativo es que se debe de pensar en programas que sean estables para ese entorno, ya que cada sistema operativo tiene sus propios traductores y rutinas de ejecución, que hacen que el programa deba de cumplir una serie de cosas a la hora de ejecutar todas sus instrucciones para que el sistema operativo ejecute el programa y funcione sin fallos. Eso solo se hace creando el programa exclusivamente para el sistema operativo en cuestión. Actualmente en el mercado hay varias opciones, entre ellas Windows, GNU/Linux y Mac Os. La forma de instalar un programa en cada sistema operativo es diferente, pero parte de un archivo que lleva la información para instalar o ejecutar en este.

Las ventajas e inconvenientes de cada uno de los sistemas operativos podrían ser las siguientes.

-El sistema operativo Mac OS es un sistema operativo pensado para una marca determinada de ordenadores, funciona perfectamente con ordenadores de la marca Apple, está totalmente diseñado para sacar el máximo partido a sus ordenadores, pero ya el resto de ordenadores baja mucho el rendimiento de este sistema. Esta opción se desecha, ya que no buscamos un sistema operativo que solo dependa de una pieza determinada.

-La segunda opción es la del sistema operativo Windows, este sistema ofrece muchas ventajas, ya que casi todos los programas están pensados para esta plataforma, obteniendo una increíble cantidad de posibilidades a la hora de usar programas. Es compatible con todos los ordenadores del mercado. El inconveniente, más importante, es que es un sistema operativo privado, que depende de muchas licencias, con su consiguiente precio.

La tercera, y la seleccionada es la del sistema GNU/Linux, el inconveniente es la incompatibilidad con algunos programas usados en el entorno de investigación e industrial. En cambio ofrece muchas ventajas como el código abierto, ahorrando así licencias y pudiendo modificar éste a nuestro gusto, un sistema operativo estable y en continua mejora, y con un consumo de recursos muy pequeño. Aparte de que dispone de un sistema para evitar que se quede bloqueado el ordenador, siendo un peligro para cualquier actividad donde necesitamos que no haya fallos.

Fig. 4.2. Logos de diferentes sistemas operativos

4.2.- Linux

4.2.1.-Historia de Linux

En los años 1953 IBM sacó su primer ordenador. Este ordenador trabajaba con lenguaje máquina. Este ordenador supuso el inicio de la comercialización de los ordenadores para los usuarios. Se sabía que el potencial del mercado era muy grande, pero en ese momento era difícil alcanzar al usuario común, debido a la dificultad del uso de los ordenadores. Esto produjo que se centrase la investigación en buscar un sistema que fuese sencillo para interactuar con el ordenador.

Surgieron varias alternativas de sistemas operativos, una de ellas fue el sistema UNIX en 1969. Los diseñadores pertenecían a un grupo de investigación de Bell y AT&T pero que trabajaban por libre, sin patrocinio por parte de la empresa. Su idea era hacer en código abierto un sistema operativo que todo el mundo podía consultar y participar. Surgió la idea del sistema

operativo Multics. Era un inicio bastante prometedor, pero al final no les salió tan bien el sistema operativo, ya que el rendimiento era muy bajo, se necesitaba muchos recursos del ordenador para obtener algún resultado.

Entonces se plantearon volver a escribir el código, desarrollando el sistema de ficheros que conocemos ahora, con cosas como el sistema operativo multitarea en sí, aparte de agregar un intérprete de órdenes y un pequeño conjunto de programas para realizar alguna tarea en particular. Esto dio a la luz el sistema operativo UNIX. Para demostrar su utilidad tuvieron que añadir un procesador de textos y algunos programas más, para así obtener el patrocinio por parte de los laboratorios Bell. Convirtiendo a UNIX en un sistema privado.

En 1972 se volvió a escribir el código de UNIX de nuevo, pero esta vez en lenguaje C para aumentar la compatibilidad con los equipos del mercado. Se fue mejorando hasta tal punto que fue fácilmente comercializado para muchos equipos. En 1983, Richard Stallman publicó en los diferentes medios de comunicación que deseaba crear un sistema operativo parecido a UNIX, pero volviendo a la idea de la colaboración entre programadores de modo altruista y libre.

Plantearon un sistema que fuese totalmente compatible con UNIX, solo que a diferencia de UNIX, este nuevo sistema llamado GNU tenía una licencia que permitía la distribución gratuita. Esto fue la idea base y revolucionaria que fue apoyada por Richard Stallman. Para demostrar su apoyo fundó una organización en 1985 que se iba a encargar a partir ese momento de la organización de todo tipo de licencias libres.

En 1990 el sistema GNU ya disponía de su propio editor de textos, un compilador y la mayoría de bibliotecas que había en UNIX. El problema es que faltaba un núcleo (kernel).

El núcleo (kernel) es la parte más importante de un sistema operativo, es el software encargado de unir los procesos o tareas con el Hardware, aparte de decidir el orden de uso de los dispositivos. El núcleo aparte facilita la tarea a la hora de programar, ya que oculta mucho código complejo, dando una interfaz fácil y rápida. Por lo que hace al núcleo el centro o piedra angular para desarrollar un sistema operativo.

GNU disponía de un núcleo, pero este producía muchos problemas a la hora de trabajar con el sistema UNIX (idea de compatibilidad total con este sistema). Debían de emular a este núcleo para poder trabajar con él, produciendo una disminución de su rendimiento. Este núcleo del que hablaban era uno que habían desarrollado en el MIT, este núcleo se llamaba TRIX que fue distribuido libremente por sus creadores. El problema es que este núcleo necesitaba unos equipos caros para trabajar con él, y se planteó cambiar al núcleo Mach desarrollado en la CMU, se le renombró con el nombre de HURD, y se trabajó sobre él hasta que el proyecto se estancó debido a conflictos entre los programadores.

En 1991 Linus Torvalds empezó a escribir el núcleo llamado Linux, éste fue publicado con licencia GPL. Esto produjo que en el proyecto se uniesen pronto muchos programadores desde Internet, obteniendo en poco tiempo un núcleo compatible con Linux. En 1992 se consiguió un núcleo totalmente compatible con UNIX, al que fue pronto unido al sistema GNU obteniendo un sistema operativo libre y funcional. Este sistema operativo se le llamó GNU/Linux o distribución Linux.

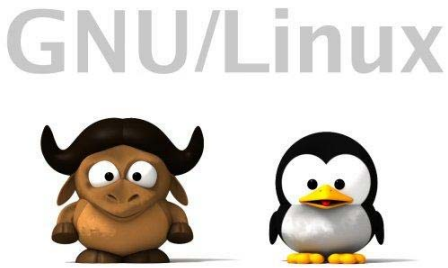


Fig. 4.3. Caricatura de los logotipos de GNU y de Linux

Debido a la estabilidad de GNU/Linux se empezó a ver cada vez más programas escritos para GNU funcionando en UNIX, el motivo era que los programas normalmente funcionaban mejor en Linux que sus versiones equivalentes en UNIX debido a que había muchas personas detrás que se encargaban de revisar cada día los programas. A estos programas se les llama herramientas GNU, que se han exportado a sistemas operativos como Windows o MAC OS.

4.2.2.- Distribuciones de Linux

Aunque GNU/Linux es un sistema operativo completo, existen diferentes variantes que se han desarrollado paralelamente incorporando una serie de paquetes de programas instalados de fábrica para satisfacer a un sector de los usuarios con unas demandas específicas.

Las distribuciones más conocidas de Linux son Ubuntu, Debian, Fedora, Gentoo, Suse, Mandriva. Cada una lleva un gestor de paquetes diferente, y la administración y sus actualizaciones las recibe por parte de un servidor diferente.

Fig. 4.4.- Diferentes logotipos de las distribuciones Linux disponibles

Aunque GNU/Linux se desarrolle con el núcleo Linux, hay diferentes distribuciones que usan otros núcleos, como el citado Hurd o, en Debian GNU/kFreeBSD, Debian GNU/NetBSD, Nexenta OS o GNU-Darwin.

4.2.2.1- Debian

Debian, o Proyecto Debian, es una comunidad conformada por desarrolladores y usuarios, que mantiene un sistema operativo GNU basado en software libre precompilado y empaquetado, en un formato sencillo en múltiples arquitecturas de computador y en varios núcleos.

Fig. 4.5.- Logotipo de la distribución Debian

Debian nace como una apuesta por separar en sus versiones el software libre del software no libre. El modelo de desarrollo del proyecto es ajeno a motivos empresariales o comerciales, siendo llevado adelante por los propios usuarios, aunque cuenta con el apoyo de varias empresas en forma de infraestructuras. Debian no vende directamente su software, lo pone a

disposición de cualquiera en Internet, aunque sí permite a personas o empresas distribuir comercialmente este software mientras se respete su licencia.

Como todas las versiones de Linux, dispone de una versión estable, que es la que no tiene casi errores, y es suficiente estable, esta distribución se llama Lenny

La versión en pruebas, que es la más actualizada, tiene pocos errores, pero a veces no es estable se llama Squeeze. La ventaja de esta versión es que todo el software esta actualizado.

La versión inestable, que es la que se está ahora mismo modificando para obtener una versión estable se llama Sid, es muy inestable y esta plagada de errores, pero es lo que batallan los desarrolladores para mejorar el sistema operativo.

4.2.3.-Entornos de escritorio

El entorno de escritorio es un conjunto de programas que hacen más fácil la interacción entre el usuario y el ordenador. Este conjunto de programas ofrece una serie de iconos, barras de herramientas y comandos (como arrastrar y soltar).

Cada escritorio tiene una serie de características y comportamientos particulares. Aunque algunos de ellos han cogido algunas características de escritorios existentes. Para Linux podemos destacar el escritorio GNOME y KDE

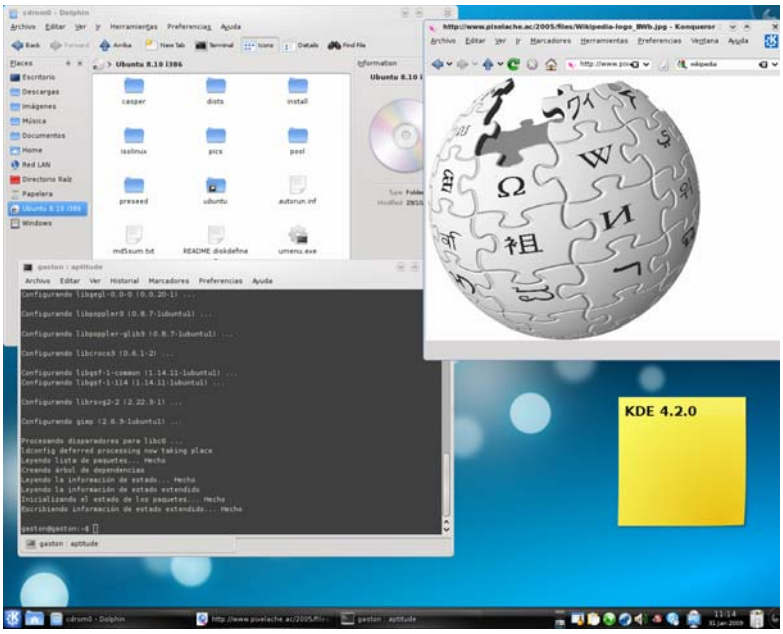


Fig. 4.6.- Un ejemplo de escritorio ejecutando varias aplicaciones

4.2.3.1.- GNOME

El entorno de escritorio GNOME fue desarrollado para sistemas operativos UNIX y compatibles (como puede ser Linux), tiene licencia libre

El Proyecto GNOME, según sus creadores, provee un gestor de ventanas «intuitivo y atractivo» y una plataforma de desarrollo para crear aplicaciones que se integran con el escritorio. El Proyecto pone un gran énfasis en la simplicidad, usabilidad y eficiencia. Otros objetivos del proyecto son:

-La libertad para crear un entorno de escritorio que siempre tendrá el código fuente disponible para reutilizarse bajo una licencia de software libre.

-El aseguramiento de la accesibilidad, de modo que pueda ser utilizado por cualquiera, sin importar sus conocimientos técnicos y discapacidad física.

-Hacer que esté disponible en muchos idiomas. En el momento está siendo traducido a más de 100 idiomas.

-Un ciclo regular de liberaciones y una estructura de comunidad disciplinada.

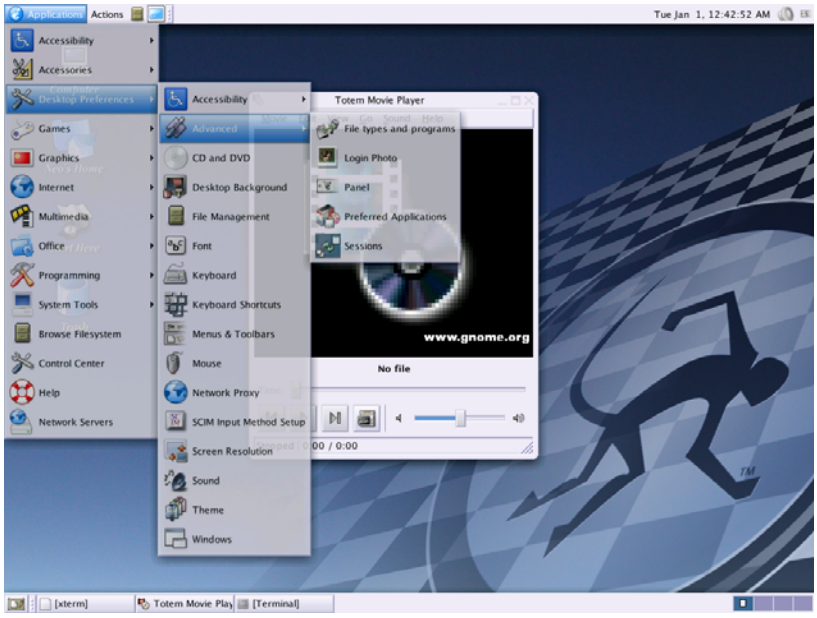


Fig. 4.7.- Escritorio GNOME

Cuando inicie una sesión gráfica verá el escritorio GNOME. Este escritorio cuenta con iconos para acceder rápidamente a aplicaciones y con menús que le permiten iniciar programas. También le permite usar varios espacios de trabajo, cada uno como un escritorio independiente de los demás —aunque es fácil pasar aplicaciones de un espacio de trabajo a otro, pues basta arrastrarlas con el ratón.

El botón izquierdo del ratón normalmente permite elegir una opción de un menú o activa un icono. El botón derecho tiene diversas aplicaciones de acuerdo al contexto —por ejemplo sobre los iconos permite configurarlos—, el botón del centro permite pegar el texto que se hubiera seleccionado con el ratón, si su ratón sólo tiene 2 botones, puede «emular» el botón del centro oprimiendo simultáneamente el izquierdo y el derecho. Para seleccionar un

texto se pasa por encima del mismo con el puntero del ratón mientras se mantiene presionado el botón izquierdo.

El escritorio GNOME es bastante configurable: puede configurar los menús, los iconos, las tipografías, el fondo, el protector de pantalla, el tema, el administrador de ventanas, sonido, la interacción con las ventanas y muchos otros detalles de acuerdo a su gusto. Para hacer algunas de las configuraciones puede emplear opciones de los menús GNOME, el ratón —por ejemplo para administrar los iconos que hay sobre el escritorio—, el programa «Centro de control GNOME» y eventualmente los archivos de configuración de X-Windows.

El gestor de ventanas empleado decorará cada ventana con botones que le permitirán cerrar, maximizar o minimizar. Las ventanas minimizadas se verán en la parte inferior del escritorio GNOME —más precisamente en el panel que no necesariamente está en la parte inferior, porque puede reubicarse de acuerdo a su gusto.

Entre los menús de GNOME, existen múltiples herramientas: calculadora gcalc; editor de texto sencillo gnotepad; calendario gnomecal, procesador de textos, hoja de cálculo, reproductor multimedia y muchas otras.

4.3.-Definición de entorno de desarrollo integrado

Un entorno de desarrollo integrado (IDE) es un conjunto de herramientas que usa un programador para desarrollar sus programas. Cada IDE puede centrarse en un lenguaje de programación, o en varios para trabajar.

Un IDE contiene el editor de código, un compilador, un depurador y un constructor de interfaces gráficas. Los IDE pueden ser una aplicación completa, o pueden ser parte de aplicaciones existentes. El lenguaje Visual Basic, por ejemplo, puede ser usado dentro de las aplicaciones de Microsoft Office, lo que hace posible escribir sentencias Visual Basic en forma de macros para Microsoft Word.

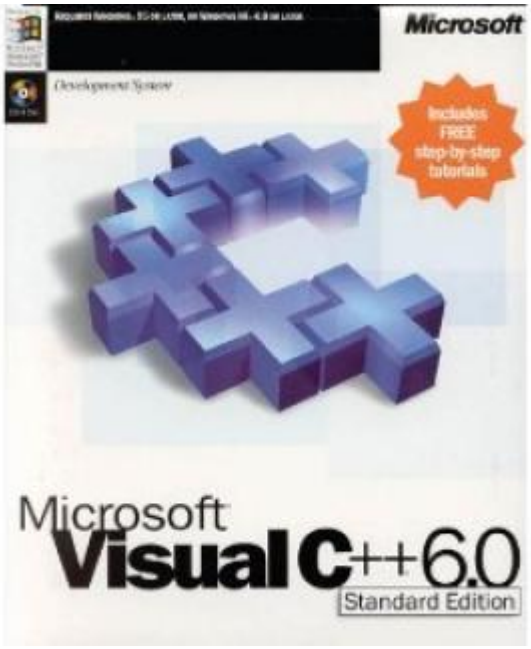


Fig. 4.8.- Un entorno de desarrollo integrado privado

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++,Python, Java, C#, Delphi, Visual Basic , etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto, como es el caso de Smalltalk u Objetive-C.

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, al que mediante plugins se le puede añadir soporte de lenguajes adicionales.

4.3.1- Anjuta

Anjuta (actualmente Anjuta DevStudio) es un entorno integrado de desarrollo(IDE) para programar en los lenguajes C, C++, Java y Python, en sistemas GNU/Linux. Su principal objetivo es trabajar con GTK y en el *escritorio* GNOME, además ofrece un gran número de características avanzadas de programación. Anjuta es software libre, liberado bajo la licencia GLP. Incluye un administrador de proyectos, asistentes, plantillas, depurador interactivo y un poderoso editor que verifica y resalta la sintaxis escrita.

Fue en 1999 cuando Naba Kumar dio a conocer su primera versión alfa de Anjuta, llamado así en honor a su novia a quien se lo dedica. Su objetivo es desarrollar un entorno de desarrollo integrado (IDE) para GNOME que al margen de la potente línea de comandos para el desarrollo en Linux, podía facilitar la creación de programas utilizando las bibliotecas de funciones GTK. Después de que Anjuta 1.2 apareciese en 2003, Anjuta2 se planteó como una versión próxima con nuevas características, pero el desacuerdo con parte del equipo terminó en una escisión renombrando Anjuta2 a *Scaffold* (literalmente,

andamio). Scaffold era desarrollado sobre todo por el equipo anterior de gIDE mientras que Anjuta 2.0 ahora se construía encima del viejo código de Anjuta pero con una nueva arquitectura.

De todas formas, Naba siempre ha creído en implementar un sistema de extensiones y la portabilidad de algunas de las viejas características como características fundamentales para Anjuta 2.0. Se reutilizó algo del anterior Anjuta2 y el diseño actual continua utilizando totalmente en GTK+.

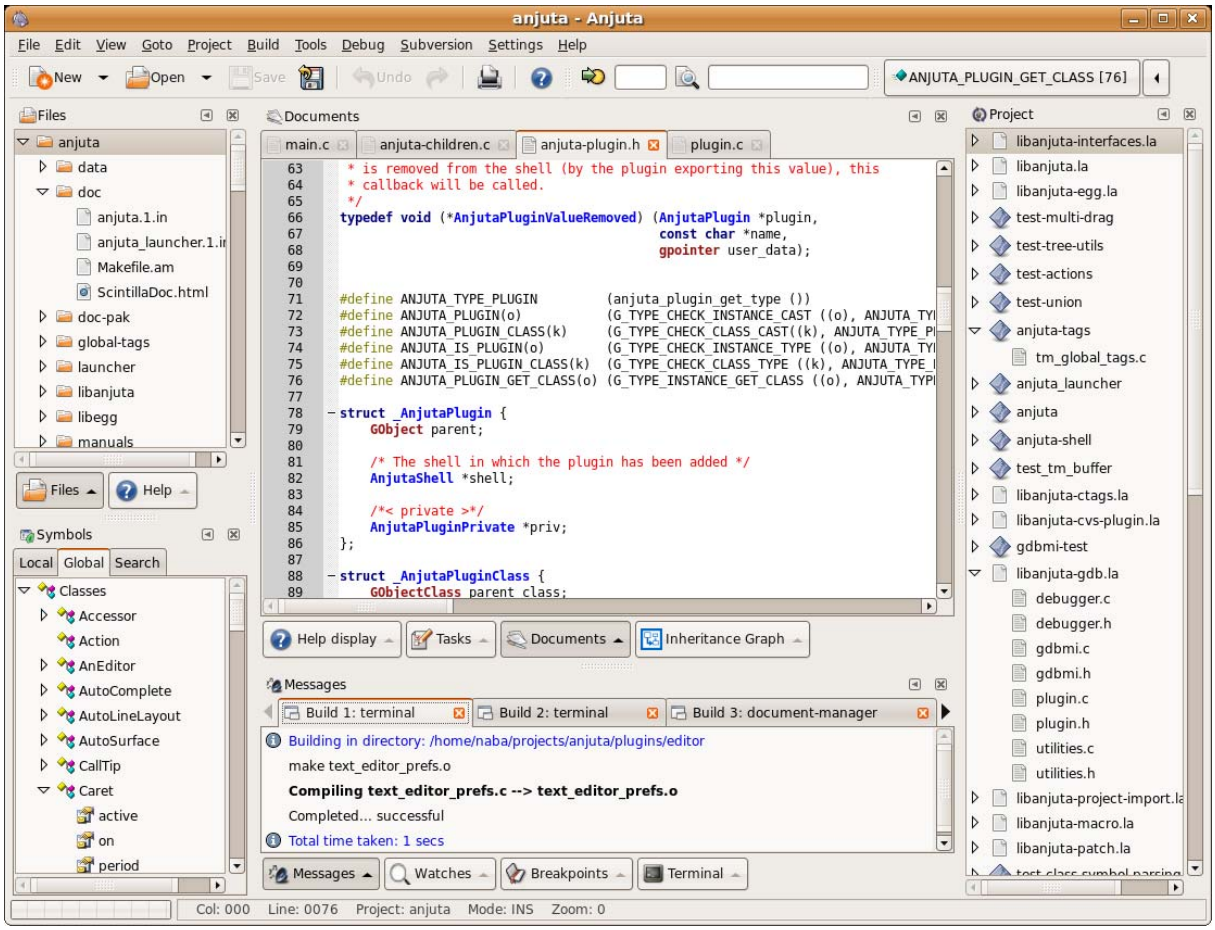


Fig. 4.9.- Entorno IDE Anjuta

A finales de 2004 , el código base alcanzó una cierta estabilidad y otros desarrolladores se interesaron otra vez en el proyecto. Ahora muchas de las funcionalidades del entorno IDE están funcionando, mejorando cada día. Después de alrededor de un año, Anjuta 2.0 esta finalmente preparado por primera vez.

4.3.2.- Glade

Glade (o Glade Interface Designer, que significa Diseñador de interfaz Glade) es una herramienta de desarrollo visual de interfaces gráficas mediante GTK/GNOME. Es independiente del lenguaje de programación y predeterminadamente no genera código fuente sino un archivo XML. La posibilidad de generar automáticamente código fuente fue descontinuada desde Glade-3.

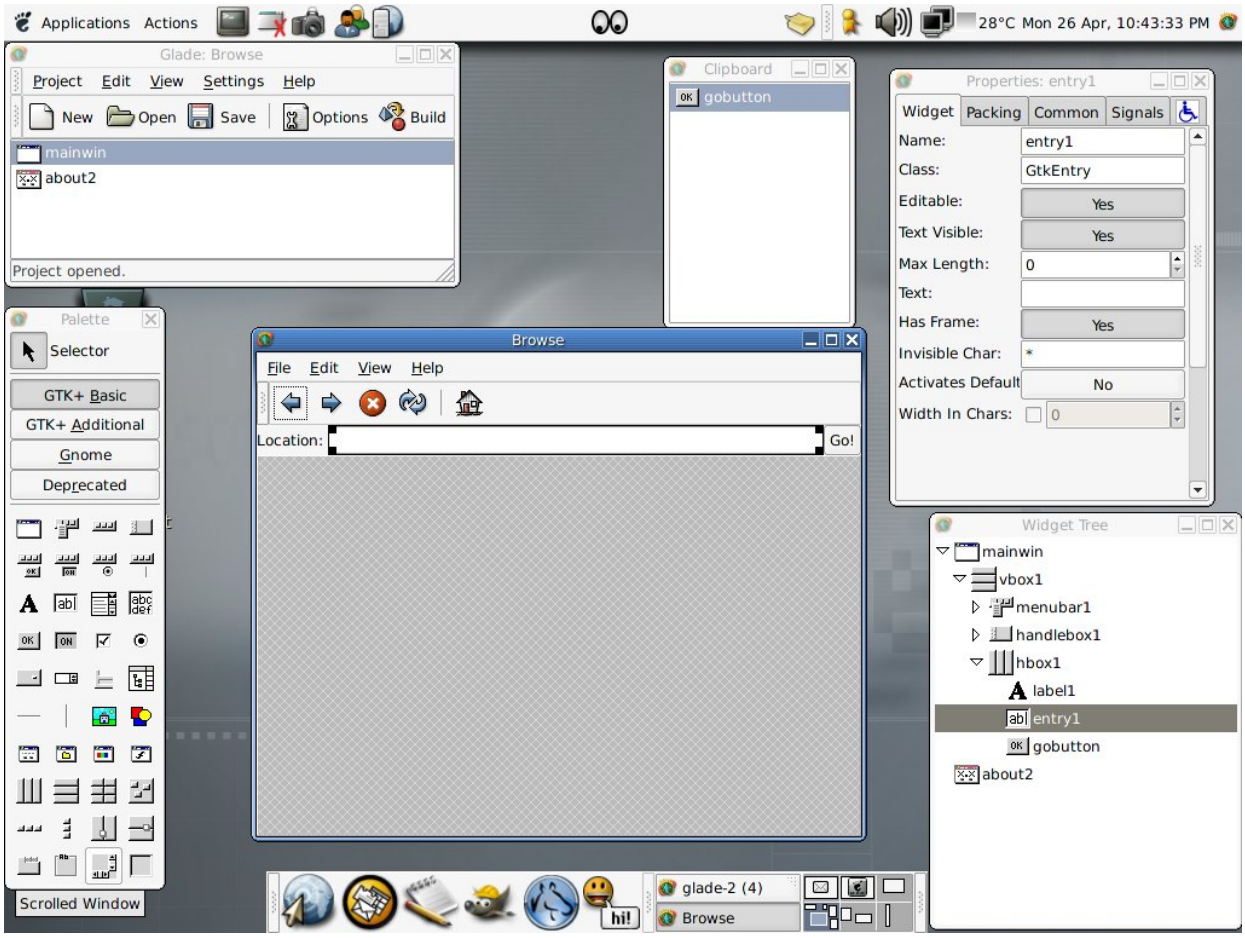


Fig. 4.10. Programa IDE Glade

Aunque tradicionalmente se ha utilizado de forma independiente, está totalmente integrado en el recientemente liberado Anjuta 2. Cuenta con tres versiones, la primera para GTK+ 1 y las otras para GTK+ 2. Se encuentra bajo la licencia GPL. Para QT existe un proyecto similar, QtDesigner.

4.3.3.-Kdevelop

KDevelop es un entorno de desarrollo integrado para sistemas GNU/Linux y otros sistemas Unix, publicado bajo licencia GPL, orientado al uso bajo el entorno gráfico KDE, aunque también funciona con otros entornos, como Gnome.

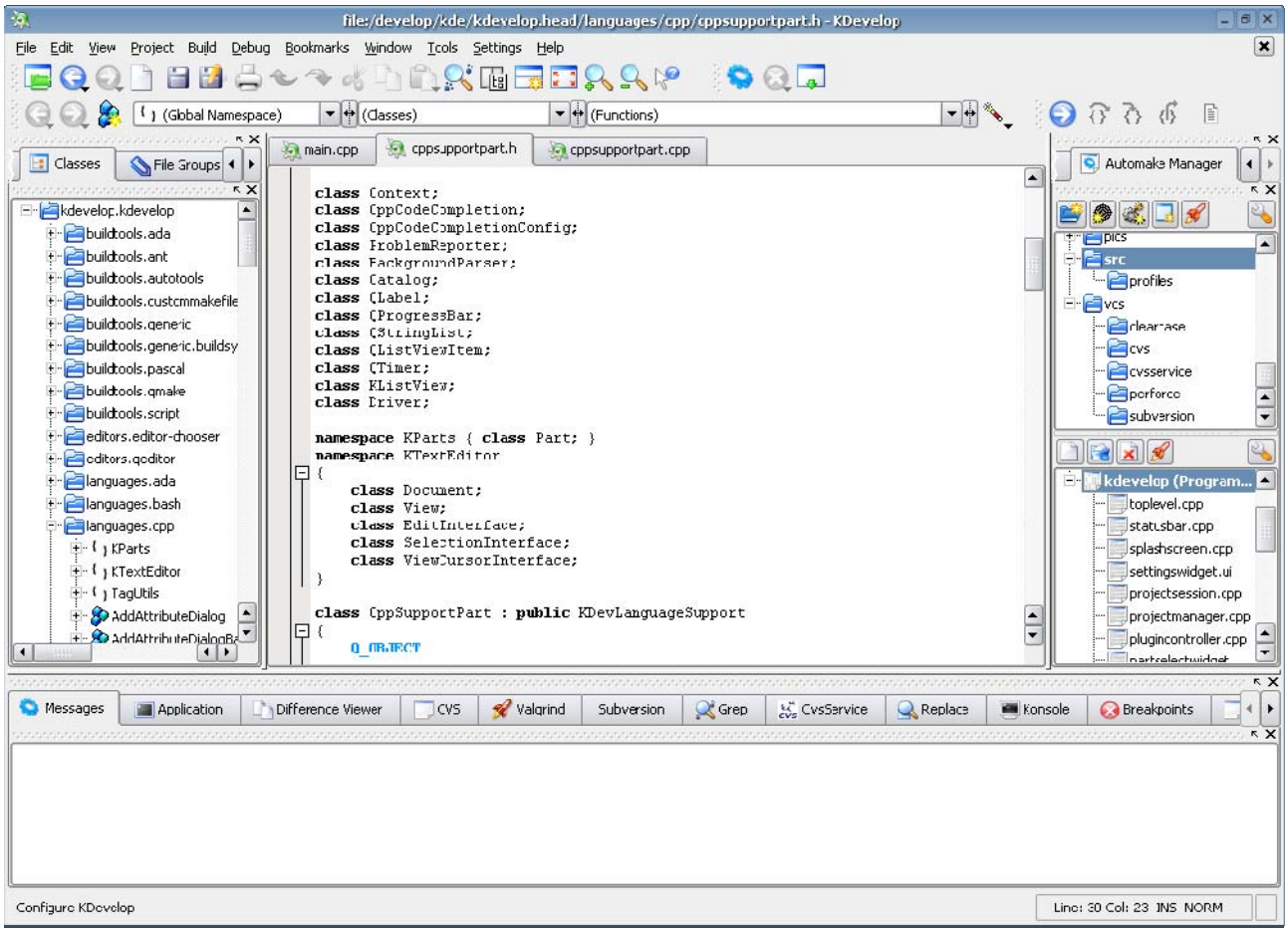


Fig.4.11.- Entorno Kdevelop

El mismo nombre alude a su perfil: KDevelop - KDE Development Environment (Entorno de Desarrollo para KDE).

KDevelop 3.0 ha sido reconstruido completamente desde los cimientos, se dio a conocer junto con KDE 3.2 en diciembre de 2003.

A diferencia de muchas otras interfaces de desarrollo, KDevelop no cuenta con un compilador propio, por lo que depende de gcc para producir código binario.

Su última versión se encuentra actualmente bajo desarrollo y funciona con distintos lenguajes de programación como C, C++, Java, Ada, SQL, Python, Perl y Pascal, así como guiones para el intérprete de comandos Bash.

4.3.4.- Sun studio 12

El software Sun Studio proporciona compiladores y herramientas para C, C + +, Fortran y el desarrollo en Solaris, OpenSolaris, y las plataformas Linux, con el apoyo de múltiples núcleos x86 y SPARC basados en sistemas Sun Studio Software - Herramientas avanzadas para C, C + + y Fortran Desarrolladores

Sun Sun Studio 12 Update 1 es la última versión de producción de software Sun Studio. Disponible para la mayoría de las distribuciones de Linux. Esta versión contiene nuevas características y mejoras que simplifican el desarrollo de múltiples núcleos, incrementar el rendimiento de aplicaciones en tiempo de ejecución y mejorar la productividad del desarrollador.

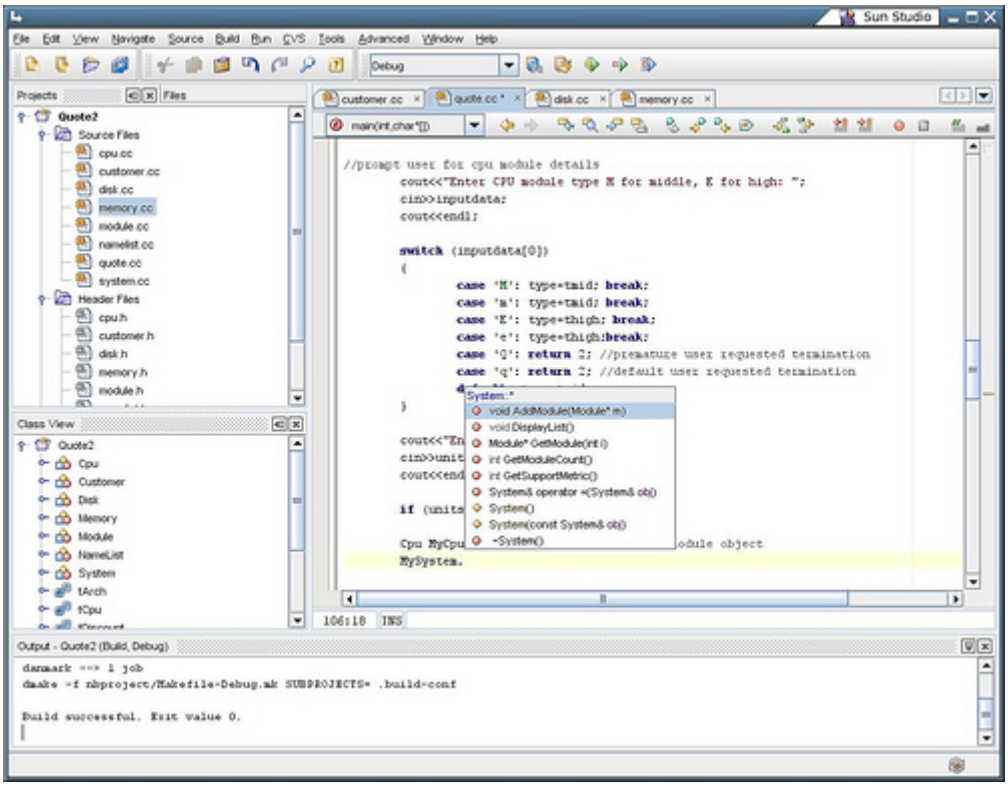


Fig. 4.12.- Entorno Sun Studio

4.3.5.- Visual C++

Visual C++ (también conocido como MSVC, Microsoft Visual C++) es un entorno de desarrollo integrado (IDE) para lenguajes de programación C, C++ y C++/CLI. Esta especialmente diseñado para el desarrollo y depuración de código escrito para las API's de Microsoft Windows, DirectX y la tecnología Microsoft .NET Framework.

Visual C++ hace uso extensivo del framework Microsoft Foundation Classes (o simplemente MFC), el cual es un conjunto de clases C++ para el desarrollo de aplicaciones gráficas en Windows. El IDE cuenta con herramientas poderosas como el IntelliSense, RemoteDebuging, Editar y Continuar, y Texto Resaltado. A parte, cuenta con una versión Express, llamada Microsoft Visual C++ Express Edition, la cual es gratuita y se puede descargar desde el sitio de Microsoft. El lenguaje de programación utilizado por esta herramienta, de igual nombre está basado en C++, y es compatible en la mayor parte de su código con este lenguaje, a la vez que su sintaxis es exactamente igual. En algunas ocasiones esta incompatibilidad impide que otros compiladores, sobre todo en otros sistemas operativos, funcionen bien con código desarrollado en este lenguaje. Algunas de las nuevas

implementaciones es que el llamado código administrado (managed code), hace uso de una de las mejores herramientas dentro de .NET, el recolector de basura (garbage collector).

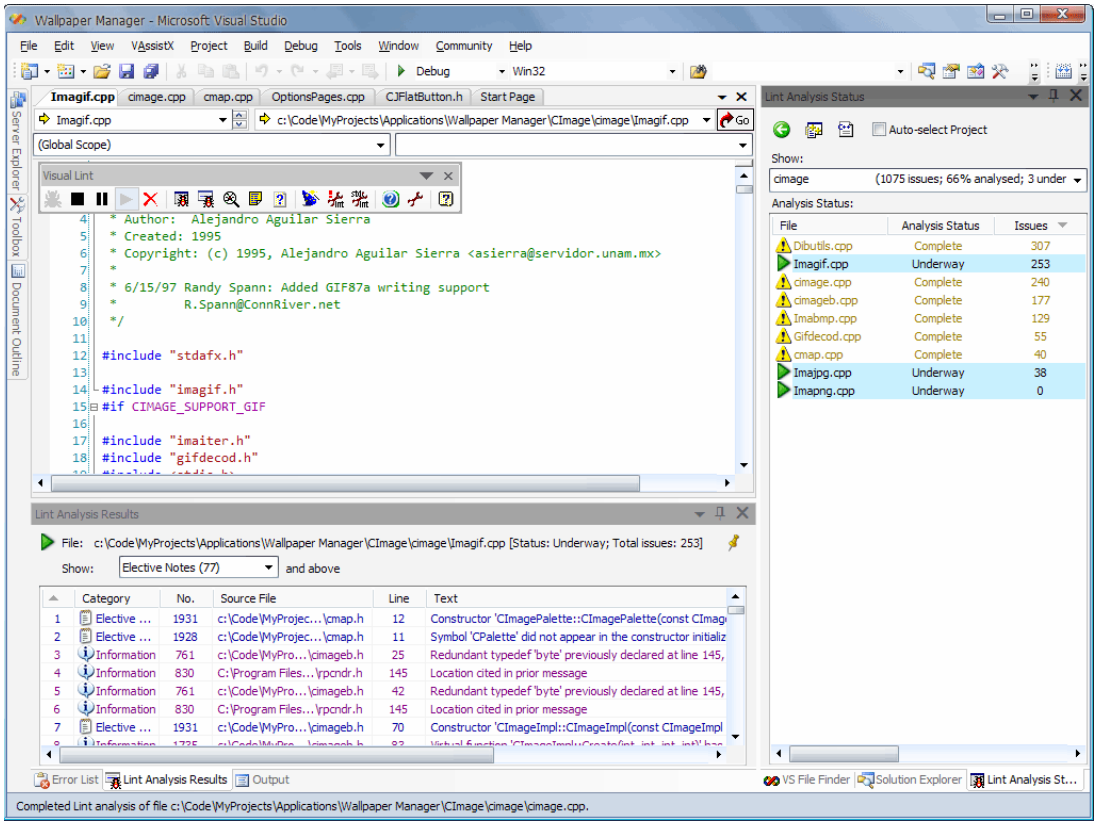


Fig. 4.13.- Entorno Visual Studio

4.4.-Definición de ventana en informatica

Uno de mis objetivos para el proyecto era desarrollar una ventana grafica para intentar controlar los motores del brazo robótico. Para introducir esto creo que tengo que definir claramente que es una ventana en informática.

En informática, una ventana es un área visual, normalmente de forma rectangular, que contiene algún tipo de interfaz de usuario, mostrando la salida y permitiendo la entrada de datos para uno de varios procesos que se ejecutan simultáneamente. Las ventanas se asocian a interfaces gráficas, donde pueden ser manipuladas con un puntero. La idea fue desarrollada en el Xerox PARC.

Una [[interfaz se superponen, una está encima de la otra, con la parte tapada de la ventana de abajo no visible. De todas maneras, muchos programas con interfaces de texto, como Emacs, permiten su división en áreas denominadas también ventanas. La parte de un sistema de ventanas que controla esto se denomina gestor de ventanas.

Las ventanas son una característica (o widget) de muchas interfaces gráficas de usuario (sobre todo las de WIMP). CDE (para VMS), X Window

System (para sistemas GNU y Unix), Microsoft Windows y el Open Windows de IBM son identificados por esta característica.

Muchas aplicaciones con las que es posible trabajar con más de un archivo a la vez, como un editor de imágenes, ponen cada archivo en una ventana separada de manera que todos los archivos están visibles a la vez. Normalmente hay una distinción entre la ventana principal de la aplicación y sus ventanas hijas, así que a veces una aplicación de este tipo fuerza a las ventanas a minimizarse en la parte inferior de la ventana principal, en vez del lugar preparado para eso por el sistema operativo.

.

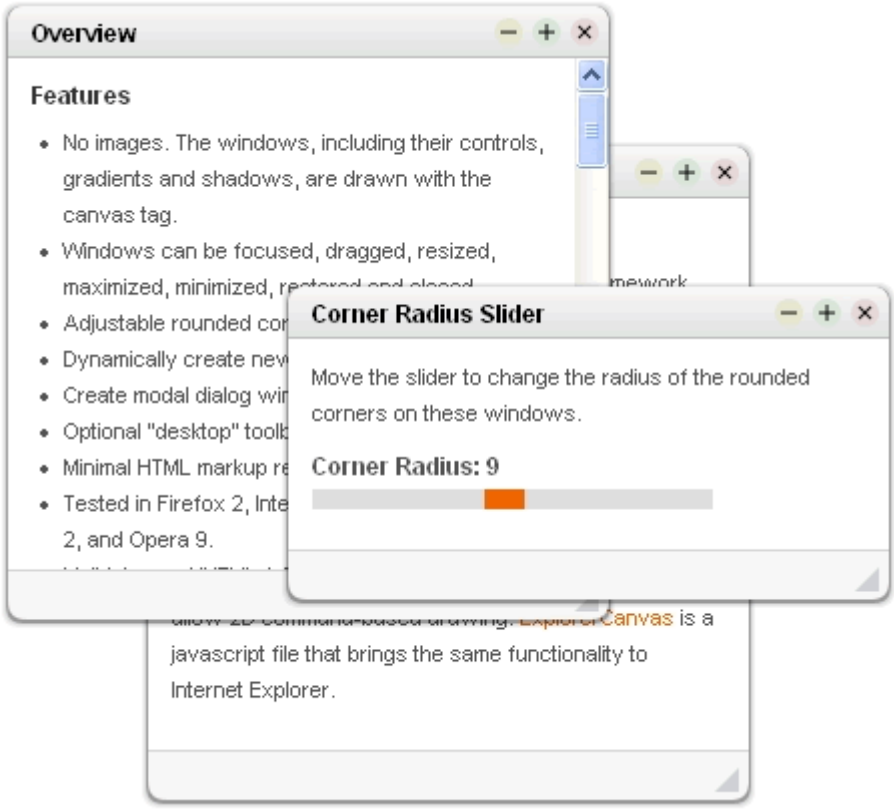


Fig. 4.14.- Muestra de varios ejemplos de ventanas abiertas

Sistema de ventana

Los sistemas de ventanas se encargan de gestionar las prioridades, la apariencia y demás, en los sistemas basados en Unix y Linux no tienen un sistema de ventanas estándar. En muchos administradores de ventanas para X11, la apariencia y comportamiento de las ventanas puede especificarse en la opción de preferencias o en los archivos de configuración.

Propiedades de ventana

Las propiedades de las ventanas vienen marcadas por el administrador de ventanas utilizado, estas tienen un amplio conjunto de propiedades que a menudo pueden ser modificadas por el usuario:

-Su tamaño.

-Maximizar en el eje vertical, horizontal, o ambos.

-Minimizar (normalmente las oculta y pone un enlace en la barra de tareas o dock).

-Visible en todos los escritorios. Si el administrador de ventanas soporta escritorios virtuales, esto hace que la ventana sea visible en todos los escritorios.

-Dejar sólo la barra de título y oculta el resto de la ventana.

-Visibilidad de las barras de herramientas de la ventana.

-Transparencia (si el administrador de ventanas lo soporta).

-Siempre arriba, que evita que la ventana sea tapada por otras.

-Borde - presencia y apariencia.

Apariencia de la barra de título.

Normalmente es posible mover las ventanas en X11 manteniendo pulsada una tecla modificadora (como Alt) y arrastrando cualquier parte de la ventana, o usando la barra de título. Muchos administradores de ventanas pueden agrupar ventanas para que actúen como una sola, y los administradores de ventanas en 3D, como Metisse y Project Looking Glass permiten cambiar las propiedades tridimensionales de una ventana, (como su rotación en los ejes Y o Z).

Tipos de ventanas

-Los administradores de ventanas a menudo ofrecen distintos tipos de ventanas, de acuerdo con sus propiedades.

-Ventanas de aplicación/documento - el tipo normal de ventanas, que contienen documentos o datos de aplicaciones

-Ventanas de utilidad, que flotan encima del resto y ofrecen herramientas o información sobre la aplicación

-Cuadros de diálogo: ventanas que dan información al usuario o se la piden

-Inspectores: ventanas que están siempre encima de otras ventanas de la misma aplicación. Se usan para mostrar las propiedades de un elemento

-Ventana de aviso: que son las que salen mayormente

Foco

El administrador de ventanas necesita saber qué ventana se desea utilizar. Por ejemplo, si dos ventanas permiten la entrada de texto, el usuario debe decirle al ordenador dónde debe colocar la entrada del teclado. Esto se denomina poner el foco en una ventana. A veces es necesario hacer clic en una ventana para usarla, pero en algunos gestores de ventanas "el foco sigue al puntero", al mover el ratón sobre otra ventana esta se activa para su uso. Los objetos dentro de la ventana requieren otro clic para activarlos, para diferenciar los sitios en los que se puede introducir información.

Modalidad

La modalidad de una ventana es como mantiene el foco respecto a las demás ventanas del sistema. De acuerdo a su modalidad se clasifican en:

Ventana no modal

Permite alternar el foco a cualquier otra ventana presente dentro del entorno gráfico. Es el tipo más común de modalidad.

Ventana modal respecto a una aplicación

Permite alternar el foco a otras ventanas del sistema, pero no a la ventana que le da origen ("ventana madre") hasta que se toma una acción sobre ella. Normalmente se utilizan para confirmar una acción del usuario, un ejemplo típico sería un administrador de archivos que detiene una acción del usuario pidiendo confirmación como "¿Desea eliminar éste archivo? y las opciones de Aceptar y Cancelar (O Borrar / No Borrar)".

Ventana modal respecto al sistema

Similar a la anterior, pero no cede el foco a ninguna otra aplicación hasta que se toma determinada acción sobre ella. Por regla general sólo deben surgir cuando existe un evento a nivel del sistema completo que exija atención inmediata del usuario, por ejemplo "¿Desea apagar el equipo?".

5.1.-Introducción

Esta parte intenta mostrar mi trabajo lo más linealmente posible con mi evolución en el laboratorio de I+D+I. La primera parte que desarrollé fue el estudio de la mano Shadow, comprobando los valores que me daba e intentando comprobar que su funcionamiento es correcto. Mientras estaba en esta parte estuve viendo los avances que se llevaban con la instalación del brazo robótico sobre Linux que llevaba Pascual. Luego fue la parte de programar, primero en ventanas y luego más adelante, intente trabajar con todos los sistemas de entorno de desarrollo para el control de motores, aunque no obtuve ningún resultado.

Debido a un problema con la placa base del ordenador con el que estaba trabajando no pude terminar la parte de la memoria de la mano Shadow. Me hubiese gustado hacer unos estudios un poco más completos para enfrentarme con la mano.

Y agradezco mucho a Pascual su manual tan completo de instalación de los drivers del brazo robótico que he añadido al final de esta memoria.

5.2.1-Experiencia personal con la mano Shadow

Mi primera parte en el proyecto fue familiarizarme con el funcionamiento de la mano Shadow, la primera idea era comprobar que los sensores funcionasen, y ver qué valores mostraban a la salida, para luego unirla al brazo robot conociendo los resultados y pudiendo calibrar estos a la vez. Aparte de familiarizarme con el contenido del maletín.

La mano Shadow viene contenida en un maletín de aluminio protector, que contiene, aparte de la mano, un adaptador de corriente, una serie de piezas de plástico con diferentes ángulos para estudiar la salida de los sensores de posición, aparte de un rollo de hilo para sustituir los tendones dañados.

El software consta de una biblioteca proporcionada por el fabricante National Instrument, con las funciones usadas para el bus CAN, y un programa sencillo para la adquisición de datos de la mano Shadow, este programa se ejecuta bajo símbolo de sistema (símil a MS-DOS). Este fichero genera un fichero de texto “.txt” donde almacena los datos del último experimento. Este software fue proporcionado por el profesor en un archivo de datos.

5.2.2.-Conexión de la mano

Para conectar la mano al ordenador es necesario disponer de una tarjeta PCI-CAN. Aparte de un adaptador de corriente que le da la tensión que necesita la mano para funcionar correctamente.

El sistema operativo para realizar todas las mediciones es Windows XP, y el programa para trabajar en C++ es el DEV C++.

5.2.3.- Estudio del programa para la adquisición de datos

La siguiente parte de este bloque, fue estudiar por los archivos que venían para trabajar con la mano y que iba a usar para realizar mis experimentos, comprobar que entendía el funcionamiento, el programa ejecutaba lo que el usuario necesitaba, y luego ejecutar una serie de experimentos para estudiar los resultados.

Al abrirlo por primera el documento de C++ me encuentro dos partes, la parte nican.h, que es la biblioteca, y el ejecutable transmit_receive_same_port.cpp, que realiza la transmisión de los datos entre la mano y el ordenador. Los siguientes puntos muestran los resultados que obtuve al estudiarlos.

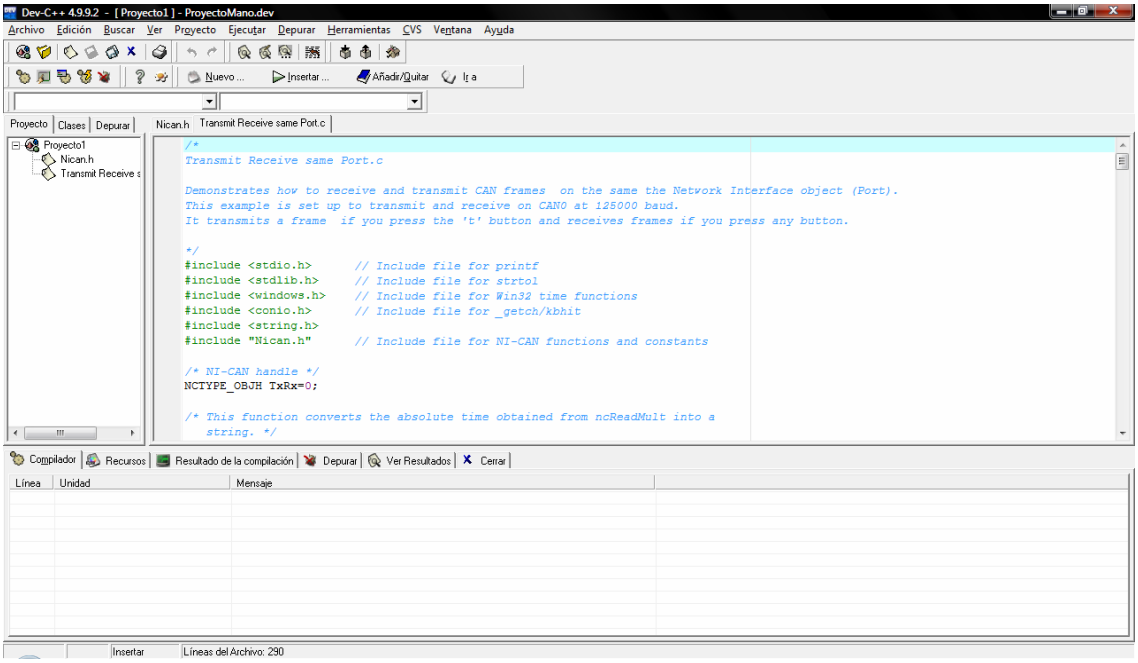


Fig. 5.-1.- Muestra de la los programas para la adquisición de datos

5.2.4.-Nican.h

Este programa es proporcionado por National Instrument, es una biblioteca de C++ que nos sirve para definir las principales funciones en el trabajo de elementos con bus CAN. Esta configurada para trabajar en modo dinámico, o lo que es lo mismo, que pueda ser usada por diferentes programas sin que haya un conflicto debido a que tiene llamadas a funciones de diferentes sitios.

Es una biblioteca que se usa para todos los productos de National Instrument que trabajan con el bus CAN. Esta biblioteca se actualiza cada cierto tiempo para corregir los errores y a la vez proporcionar nuevas funciones más potentes.

Una de las cosas que se centra la biblioteca es explicarle al ordenador que es el bus CAN, primero detecta el numero de dispositivos de National Instrument que se encuentran este momento instalados en el PC, en nuestro caso solo esta instalada una tarjeta de adquisición con una salida de bus CAN. Aparte le define el tamaño de las tramas del bus CAN, tales como las de envío y recepción de datos, como las diferentes de error. Otra de las partes mas importantes de esta biblioteca es que tiene una parte de configuración donde define un retraso en el envío de dos tramas consecutivas, esto debemos de configurar en programas complejos donde ese valor ya que se puede acumular y producir un tiempo de envío y recepción muy alto

5.2.5.- Transmit_receive_same_port.cpp

Este programa contiene la programación necesaria para realizar la transmisión. En resumen este programa se divide en los siguientes pasos:

- Primero establece la configuración para transmitir y recibir datos, como por ejemplo tiempos de respuesta, longitud de trama, etc.
-Crea el objeto para recibir los datos y estructurarlos en una información tratable,
-Se produce un envío de datos por parte del ordenador hacia la mano, con la petición de inicio de transmisión, a lo que la mano responde con una serie de datos con las lecturas de los sensores, y cuando estos datos han sido procesados, se destruye la información y se cierra el programa.

En el siguiente esquema se muestran los pasos

Formas de comunicación “programación”:
- Canal (Channel API).
- Trama (Frame API).

Funciones de comunicación.
- Configuración:
  - ncConfig();

Apertura:
ncOpenObject();
Envío de datos:
ncWrite();
Recepción de datos:
ncReadMult();
Cierre:
ncCloseObject();

El programa que usamos es el siguiente

```
/*****************************************************************************/
Transmit Receive same Port.c

Demonstrates how to receive and transmit CAN frames  on the same the Network
Interface object (Port).
This example is set up to transmit and receive on CAN0 at 125000 baud.
It transmits a frame  if you press the 't' button and receives frames if you press any
button.

*/
#include <stdio.h>      // Include file for printf
#include <stdlib.h>     // Include file for strtol
#include <windows.h>    // Include file for Win32 time functions
#include <conio.h>      // Include file for _getch/kbhit
#include <string.h>
#include "Nican.h"      // Include file for NI-CAN functions and constants

/* NI-CAN handle */
NCTYPE_OBJH       TxRx=0;

/* This function converts the absolute time obtained from ncReadMult into a
  string. */
void AbsTimeToString(NCTYPE_ABS_TIME *time, char *TimeString, int cont)
{

  SYSTEMTIME       stime;
  FILETIME         localftime;

  FileTimeToLocalFileTime((FILETIME *)(time), &localftime);
  FileTimeToSystemTime(&localftime, &stime);
        sprintf(TimeString, "%d\n%02d:%02d:%02d.%03d", cont,
       stime.wHour, stime.wMinute, stime.wSecond,
       stime.wMilliseconds);
}

/* Print a description of an NI-CAN error/warning. */
void PrintStat(NCTYPE_STATUS Status, char *source)
{
```

```
  char StatusString[1024];

  if (Status != 0)
  {
        ncStatusToString(Status, sizeof(StatusString), StatusString);
    printf("\n%s\nSource = %s\n", StatusString, source);

        if (Status < 0)
    {
      // On error, close object handle, then exit.
      ncCloseObject(TxRx);
      exit(1);
    }
  }
}

int main ()
{
        NCTYPE_STATUS           Status;
        NCTYPE_ATTRID           AttrIdList[8];
        NCTYPE_UINT32           AttrValueList[8];
        NCTYPE_CAN_FRAME            Transmit;
        NCTYPE_UINT32           Baudrate = 1000000;
        NCTYPE_CAN_STRUCT       ReceiveBuf[150];
        NCTYPE_UINT32           ActualDataSize=0;
        u32                     i;
        u16                     j;
         u8                      k;
        char                    Interface[6] = "CAN0";
        char                    output[15];
        char                    CharBuff[50];
        int                     ch;
/* Set XTD = 1 to send extended ID frames */
        int                     XTD = 0;
        char                    data[6];
        int                     id = 0;
        char                    data_length = 6;
        FILE                    *salida;

// Configure the CAN Network Interface Object
        AttrIdList[0] =       NC_ATTR_BAUD_RATE;
        AttrValueList[0] =     Baudrate;
        AttrIdList[1] =       NC_ATTR_START_ON_OPEN;
        AttrValueList[1] =     NC_TRUE;
        AttrIdList[2] =       NC_ATTR_READ_Q_LEN;
        AttrValueList[2] =     150;
        AttrIdList[3] =       NC_ATTR_WRITE_Q_LEN;
        AttrValueList[3] =     0;
        AttrIdList[4] =       NC_ATTR_CAN_COMP_STD;
        AttrValueList[4] =     0;
```

```
        AttrIdList[5] =       NC_ATTR_CAN_MASK_STD;
        AttrValueList[5] =    NC_CAN_MASK_STD_DONTCARE;
        AttrIdList[6] =       NC_ATTR_CAN_COMP_XTD;
        AttrValueList[6] =    0;
        AttrIdList[7] =       NC_ATTR_CAN_MASK_XTD;
        AttrValueList[7] =    NC_CAN_MASK_XTD_DONTCARE;


        Status = ncConfig(Interface, 8, AttrIdList, AttrValueList);
                if (Status < 0)
                {
                  PrintStat(Status, "ncConfig");
                }


/* open the CAN Network Interface Object */
        Status = ncOpenObject(Interface, &TxRx);
                if (Status < 0)
                {
                        PrintStat(Status, "ncOpenObject");
                }

/*Populate data array*/

                data[0]=0;
                data[1]=0;
                data[2]=0;
                data[3]=0;
                data[4]=0;
                data[5]=1;

/* print out the instructions to the I/O window */
        printf("\n\ninitialized successfuly on CAN0 ... \n\nPress any key to receive
frames");
        printf("\n\n Press 't' to transmit a frame and receive frames");
        printf("\n\n Press 'q' to quit\n\n");

/*Transmit and receives frames on key hit until the user aborts*/
        salida = fopen("Salida.txt","w+");
        do
        {
                ch = _getch();

                if (ch == 't')
                {
                        memcpy (Transmit.Data, data, data_length);
                        Transmit.DataLength = data_length;
                        Transmit.IsRemote = 0;
```

```
/* If XTD Flag is 1, set the 30th bit to initialize extended Frame Format */
                        if (XTD>0)
                                {
                                Transmit.ArbitrationId = id | 0x20000000;
                                }
                                else
                                {
                                Transmit.ArbitrationId = id;
                                }

                Status= ncWrite(TxRx, sizeof(Transmit), &Transmit);

                        if (Status < 0)
                        {
                        PrintStat(Status, "ncWrite");
                        }

                }

                Status = ncReadMult(TxRx, sizeof(ReceiveBuf),(void *)ReceiveBuf,
&ActualDataSize);
                if (Status < 0)
                        {
                                PrintStat(Status, "ncReadMult");
                        }


// Get the frame size
                printf("%d \n",ActualDataSize);
                ActualDataSize = ActualDataSize/sizeof(NCTYPE_CAN_STRUCT);
                printf("%d \n",ActualDataSize);

/* if frames were received, display them */
                        if (ActualDataSize >= 1)
                        {
                                for (i=0; i<ActualDataSize; i++)
                                {

// check, if frame is a data frame
                        if (ReceiveBuf[i].FrameType == NC_FRMTYPE_DATA)
                        {

// convert returned data into string
                                                if (ReceiveBuf[i].ArbitrationId == 293)
                                                {

        AbsTimeToString(&ReceiveBuf[i].Timestamp, &output[0], i);

                                                //printf("%s     ", output);
```

```
                                        //fprintf(salida,"%s     ", output);

                                        sprintf (&CharBuff[0], "%8.8X",
ReceiveBuf[i].ArbitrationId);
                                        printf("%s     ", CharBuff);
                                        fprintf(salida,"%s     ", CharBuff);

                                        sprintf (&CharBuff[0], "%s","CAN Data
Frame");
                                        printf("%s     ", CharBuff);
                                        fprintf(salida,"%s     ", CharBuff);

                                        sprintf (&CharBuff[0], "%1d",
ReceiveBuf[i].DataLength);
                                        printf("%s     ", CharBuff);
                                        fprintf(salida,"%s     ", CharBuff);

                                        for (j=0; j<ReceiveBuf[i].DataLength;
j++)
                                        //for (j=7; j<0; j--)
                                {
                                        //sprintf (CharBuff, " %02X",
ReceiveBuf[i].Data[j]);
                                                //printf("%s", CharBuff);
                                                //fprintf(salida,"%s", CharBuff);
                                                printf("
%02X",ReceiveBuf[i].Data[j]);
                                }
                                printf("\n");
                                        fprintf(salida,"\n");
                                        break;
                                        }
                        }

// check, if frame is a remote frame
                        if (ReceiveBuf[i].FrameType == NC_FRMTYPE_REMOTE )
                        {
// display the returned frame
                                AbsTimeToString(&ReceiveBuf[i].Timestamp,
&output[0], i);
                                printf("%s     ", output);
                                sprintf (&CharBuff[0], "%8.8x",
ReceiveBuf[i].ArbitrationId);
                                printf("%s     ", CharBuff);
                                        sprintf (&CharBuff[0], "%s  ", "Remote
Frame");
                                        printf("%s     ", CharBuff);
                                sprintf (&CharBuff[0], "%1d",
ReceiveBuf[i].DataLength);
                                printf("%s     ", CharBuff);
```

```
                                        sprintf (&CharBuff[0], " %s", "No
Data\n");
                                        printf("%s", CharBuff);;
                        }
                                }
                        }

        Sleep(100);
        } while (ch != 'q');

        fclose(salida);

/* Close the Network Interface Object */
        Status = ncCloseObject(TxRx);
                if (Status < 0)
                {
                        PrintStat(Status, "ncCloseObject");
                }
return 0;
}
```

//////////////////////////////////// FIN CÓDIGO //

Lo primero que definimos son las bibliotecas que vamos a usar para implementar nuestro código. La primera función obtiene la hora del sistema, y luego la guarda en una cadena. La siguiente es que si ocurre un error, o lo que es lo mismo, si la variable Status que vigila que el programa se esta ejecutando sin errores y si hay un error nos da un -1, si esto ocurre nos cierra el programa que estemos ejecutando.

Despues de definir estas funciones, empezamos con nuestra función main, esta función es el esqueleto de nuestro programa. Primero definimos nuestros vectores AttrIdList AttrValueList, estos vectores van a guardar la configuración para la transmisión por el bus CAN. Comprobamos que nuestro programa va bien (con la variable Status). Ahora creamos el objeto, este objeto esta definido por los datos que recibimos de la mano, el programa dispone de un vector llamado data de 6 variables, cada valor de data define los datos que queremos recibir, siendo el valor 0 para los sensores tactiles, el valor 1 para los sensores de posición del dedo 1, y asi respectivamente hasta el valor 5. Como estamos comprobando el funcionamiento de la mano, ponemos todos los datos a 1, y así nos aseguramos de que no estamos filtrando los datos de algún dedo.

Despues de haber creado el objeto, nos disponemos a enviarle un mensaje a la mano, este mensaje le pedirá la información sobre los sensores. Esta ventana será una ventana en MSDOS donde si pulsamos 't' empezará la comunicación y con la q cerramos el programa. El programa ahora espera a que el usuario pulse la tecla t para empezar la transmisión.

Si pulsamos la 't', comprueba el programa que no hay ningún problema (variable Status), y empieza el envio y la recepción de datos. Despues de recibir los datos de la mano, estudiamos cuanto es su tamaño, lo copiamos a nuestro objeto y los dividimos en paquetes.

Con un bucle if indicamos al programa que paquete de datos queremos que nos muestre por pantalla, algunos de estos paquetes corresponden a los sensores, en este caso se encontraran los paquetes entre la división 40 y 58.

Nosotros hemos modificiado el programa para poder almacenar los datos en un archivo simple de texto (.txt), este archivo guardará todos los valores que obtengamos en una sesión de transmisión de datos.

5.2.6.-Experimento de la mano shadow

Lo primero que hice cuando obtuve el maletín fue conectar la mano al ordenador. Abriendo con el programa Dev C++ el código que me fue proporcionado y que he citado arriba. Lo compilé y ejecuté y vi si el ordenador recibía datos por parte de la mano. Después de cada ejecución del programa de transmisión de datos tenía que pulsar el botón de reseteo para borrar los datos del buffer y empezar de nuevo. Para recordarte eso la mano dispone de un diodo Led azul que debe de estar apagado a la hora de la transmisión de los datos.

La siguiente parte que me dedique fue a mirar las piezas de plástico que proporcionaba el fabricante con la mano y ver donde se colocaban. Estas piezas vienen en una serie de bolsas de plástico con una pegatina que indica en que unión se deben de colocar. Se usan para realizar una serie de experimentos con los sensores de posición de efecto Hall, estas piezas de plástico al colocarlas en la unión correspondiente podemos hacer que se encuentre en un ángulo determinado, entre ellos por ejemplo podemos encontrarnos piezas a 90º, 45º, 20º, 0º.

Después, y con el manual en la mano me dediqué a comprobar los datos que me proporcionaba el fabricante que había tomado él por su cuenta, y comprobar que me daban los mismo resultados.

Para esta parte confeccioné una tabla de datos donde iba a ir apuntando los datos que iba a recibir por parte del ordenador. Para facilitarme a la hora de identificar los dedos y la uniones, me centré en nombrarlos con respecto a la mano humana. Así que para el dedo con 5 uniones sería el dedo gordo, y los otros dedos por cercanía serían el índice y anular.

Después de colocar mi primera pieza en la mano y dejarla totalmente rígida, me dedique a introducir en el programa de transmisión la ID de la unión y empecé a recibir los datos por parte de la mano obteniendo la siguiente tabla. Y así hice con todas las uniones.

DEDO	UNION	ID	OFFSET	POSICION	VALOR	VALOR SHADOW
INDICE	1 DISTAL	0X40	0	**-20**	fb10	**INDICE**
				0	e41d	EC4D
				22,5	cce9	C8F1
				45	9cb1	95FD
				67,5	B601	6248
				90	4BC5	38F1
	2 MEDIA		2	**0**	6b16	6747
				22,5	8639	870D
				45	9beb	9B14
				67.5	68AF	B3C1
				90	C746	C389
	3 PROXIMAL		4			
				0	1a76	CDA2
				22,5	9a10	9DAC
				45	6072	63E 3
				67,5	3cb5	3EB1
				90	15C4	1D8E
	4 ADUCCION-ABDUCCION		6	**-25**	7FC6	8077
				0	73C3	7369
				25	354F	320F
ANULAR	1 DISTAL	0X44	0	**-20**	No varia nada	**ANULAR**
				0		D01D
				22,5		A457
				45		70A8
				67,5		4614
				90		2380
	2 MEDIA		2	**0**		66AA
				22,5		7F8D
				45		9F17
				67.5		B4A2
				90		C63D
	3 PROXIMAL		4	**-20**		
				0		BCAF
				22,5		8C4C
				45		5D91
				67,5		2877
				90		DBB
	4 ADUCCION-ABDUCCION		6	**-25**	00f5	84A8
				0		7A0E
				25		935
PULGAR	1 DISTAL	0X50	4	**-20**	4B43	**GORDO**

				0	5DA3	6947
				22,5	76F3	80F1
				45	9859	9748
				67,5	B1C4	A77C
				90	beb8	AF51
	2 MEDIA		6	**-40**	9229	
				-30	8FFE	8E8D
				-15	8C2F	87AE
				0	7F6D	7C5A
				15	771E	70B0
				30	6F4F	6C3F
				40	6F20	
	3 MEDIA	0X54	0	**-15**	0320	7417
				0	0320	8E 87
				15	0320	943C
	4	0X58	0	**-15**	4745	
				0	45ac	64C0
				22,5	4552	845D
				45	45d7	
				67,5	4645	
				80	4555	
	5	0X58	2	**-60**	ba53	B740
				-30	A599	A538
				0	8881	86E 1
				30	6a2e	684C
				60	49b1	4AFC

Tabla de datos obtenida de la mano

La primera columna muestra el nombre del dedo, la segunda el nombre de la unión, la tercera es la dirección que tenemos que poner en nuestro programa, la cuarta es el ángulo de la unión (obtenido con la pieza de plástico). El quinto y el sexto son los valores que hemos obtenido y los datos que les ha dado al fabricante respectivamente. Los datos que he obtenido son a base de recibir una cantidad de casos, después seleccionar un caso más o menos que se encuentre central a todo lo recibido.

En ese primer estudio obtuve como resultados que los dedos gordo e índice nos daba un valor diferente a la del fabricante, pero con una cierta coherencia, en cambio con el dedo anular obteníamos un valor constante sin importar la posición de este dedo.

La segunda parte fue ejecutar una prueba para comprobar que nuestra mano era capaz de detectar un movimiento, ya que con las piezas de plástico, el valor que nos daba era estático. Me centre en cada dedo por separado. Configuré el programa para cada dedo y empecé un movimiento aleatorio, y

con la tabla comprobé que los resultados se aproximaban a los de la tabla, pero sin usar ángulos determinados.

La tercera parte era colocar cada dedo en un ángulo inicial, y mover en un movimiento aleatorio, y volver a la posición inicial. Lo realicé varias veces con diferentes ángulos iniciales, y realicé para cada junta. Obtuve unos valores coherentes para los dedos gordo e índice. Pero para el dedo anular no he recibido nada.

En esta dos últimas partes obtuve como resultado que los dedos gordo e índice eran capaces de recibir los datos de nuestro movimiento con suficiente rapidez. También pude observar que de los 4 dígitos en hexadecimal que recibíamos, los dos primero eran bastante fiables para estudiar la posición del dedo en cuestión, ya que se mantenían estáticos cuando el dedo no se movía, y tenían un desplazamiento lineal según íbamos desplazando el dedo. Los problemas que identifiqué, son que de los dos últimos dígitos hexadecimales que recibíamos de cada unión variaban mucho, haciendo difícil la lectura de los datos y encontrar la posición exacta del dedo. Para el dedo anular me encontré con que no tenía ninguna sensibilidad al movimiento, eso significa que el dato obtenido era estático por más que moviese el dedo en cualquier sentido.

5.3.1.-Ejemplos de programación

Esta parte esta sacada de diferentes libros de programación que he usado durante la carrera, y aunque considero que no tendría mucha importancia, fui indagando hasta encontrar cosas que pueden ser útiles en un futuro si se sigue intentando desarrollar el proyecto. Estos ejemplos de programación están totalmente creados y editados por mí, los motivos son varios, uno de ellos es para entender un poco la transferencia entre los programas y las librerías, otra parte para intentar buscar bloques de programas que me pudiesen ser útiles para el desarrollo del control de motores.

El programa está formado por dos partes, la primera parte es la del programa principal, está compuesto de un bucle sencillo que hace que se repita el programa main hasta que introduzcamos un valor negativo o igual a 0, que en ese caso se saldrá del bucle y cerraría el programa. En el bucle llama a funciones que están dentro de la biblioteca “menú.h”.

```
/*Este programa main mostrará un menu inicial con todos
los ejemplos que puede hacer*/
#include <iostream.h>
#include <stdlib.h> //para los comandos sistem
#include <stdio.h>
#include "menu.h"  //contiene el menu sencillo
```

```
int main()
{
  int a;

   do{       //Si no le damos a 0, el programa sigue ejecutandose
   a=menu();
   system("cls");
   cout << "\n";
          switch(a)
          {
          case 0:
               {break;}
          case 1:
              {holamundo();
              break;}
          case 2:
              {calc();
              break;}
          case 3:
              {bucleif();
              break;}
           case 4:
              {bucleswich();
              break;}
           case 5:
              {descubrir();
              break;}
           case 6:
              {bucletecla();
              break;}
           default:
               {cout << "Tecla erronea";
               break;}
          }
   cout << "\n";
   if(a<0||a>0)
   {system ("pause");}
   system("cls");
   }while(a>0);

   return 0;
}
```

Esta es la segunda parte de mi programa, esta parte es un estudio práctico de todos los conocimientos que he visto útiles a la hora de programar, aunque no he puesto todo, pero he puesto bucles, operaciones matemáticas y lo más interesante, una función que te descubre la tecla en código numérico que has pulsado, muy interesante para programar, y la otra que está a la espera de que pulses cualquier tecla.

La estructura es sencilla, primero el menú principal, y la segunda parte las funciones particulares.

```
#include <stdio.h> //Para printf
#include<iostream.h>
#include <conio.h> //Para getch
/*Aqui mostramos el menu por pantalla*/

int menu() ////////////////////
{
   int a;
   cout << "Menu de programa:\n";
   cout << "Eliga la opcion introduciendo el numero por teclado\n\n";
   cout << "1.- Hola mundo\n";
   cout << "2.- Calculadora\n";
   cout << "3.- bucle if\n";
   cout << "4.- bucle swich\n";
   cout << "5.- Descubrir tecla\n";
   cout << "6.- Bucle tecla\n";
   cout << "0.- Fin de programa\n";
   cout << "Introduce la opcion que desea ejecutar\n";
   cin >> a;
   return a;
}



//Programas sencillos de entrada salida de datos
int holamundo() ////////////////////////////
{
 cout<<"Hola mundo\n";

 return 0;
}

int calc() ////////////////////////////
{
    int a,b;
    cout << "Introdudce el primer valor: ";
    cin >> a;
    cout << "introduce el segundo valor: ";
    cin >> b;
    cout << "SUMA= " << a+b << "\nRESTA= " << a-b << "\n\n";

    return 0;
}

//Programas de bucles

int bucleif() ////////////////////////////
```

```
{
   int a;
   cout << "Introduce un valor: ";
   cin >> a;

   if(a>0&&a<=10)
        {cout << "Esta entre 1 y 10\n";}
   else if(a<0)
        {cout << "Es negativo\n";}
   else
        {cout << "El valor es nulo o mayor de 10\n";}

   return 0;
}

void bucleswich() ////////////////////////////////
{
   int a;
   cout << "Introduce un valor: ";
   cin >> a;

   switch(a)
   {case 1:
        {cout << "Uno\n";
        break;}
   case 2:
        {cout << "dos\n";
        break;}

   default: cout << "Solo se contar hasta 2\n \n";
   }
}

int descubrir() ///////////////////////////////
{char tecla;

printf("Pulsa la tecla de la que quieres conocer su codigo: ");
tecla = getch();

printf("\nEl codigo de la tecla pulsada es: %d\n",tecla);

return 0;
}

int bucletecla() ////////////////////////////////
{   int tecla;

   do{

   cout << "1";
```

```
    if(kbhit())
    tecla = getch();
} while(tecla != 113);

    return 0;
}
```

5.4.- Manual de instalación de Linux

El brazo robótico Robotnik esta configurado para trabajar desde una distribución GNU/Linux con todo el software en código libre. Como ya he citado anteriormente, las ventajas de este sistema operativo son muchas y muy variadas. El problema es que la mayoría de ordenadores del mercado no suelen venir con este sistema operativo de serie, es necesario instalarlo por nuestra cuenta.

Para trabajar mejor, e intentar acostumbrarme al trato con este sistema operativo, me centré en instalarlo en mi ordenador personal.

Busque algún manual de internet y me marco una serie de pautas para instalarlo. Todo lo que necesitaba era disponer de un CD que ofrece Linux para instalar el Linux, llamado live CD. Este CD es un instalable bastante sencillo e intuitivo con las aplicaciones básicas para instalar.

5.4.1- La Instalación.

Lo primero que hacemos es irnos a la página web oficial de Ubuntu y buscar el enlace para descargar el Live CD. Este archivo que nos descargamos contiene una imagen del CD, que con cualquier programa de grabación podemos copiar dentro de un CD.

Lo siguiente es configurar nuestra BIOS del sistema para que reconozca el CD y arranque desde el directamente, en vez que cargue los datos desde el disco duro.

Reiniciamos de nuevo y al arrancar nos aparecerá una pantalla con diferentes opciones. Nos saldrá una pantalla de menú mostrada a continuación. Para más facilidad podemos cambiar el idioma a castellano pulsando la tecla F2 y seleccionando desde el menú el castellano.

Fig. 5.2.- Pantalla de instalación desde el CD

Seleccionamos la primera opción, nos saldrá una pantalla de carga del sistema operativo.

Fig. 5.3.- Pantalla de carga del CD de instalación

Cuando tengamos cargado el Live CD en nuestro ordenador, nos saldrá el escritorio de GNU/Linux, con las aplicaciones mínimas, para comprobar su funcionamiento en general y la compatibilidad con nuestro hardware. Si estamos seguros de que queremos instalar Linux, en el escritorio aparecerá un

icono que nos permitirá realizar la instalación; pinchamos en él y empezaremos el proceso de instalación.

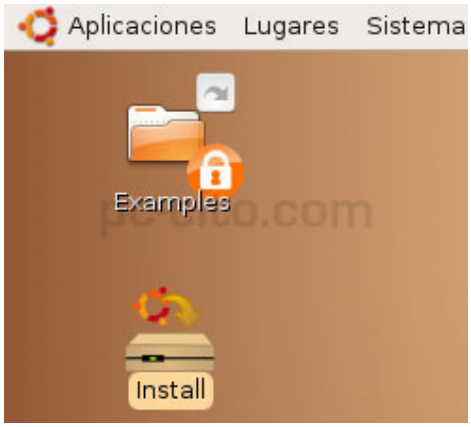


Fig 5.4.- Escritorio de muestra de Linux con el icono de instalación

Las primeras opciones las he omitido por la poca importancia que tienen frente a la instalación. La primera es la de configuración del idioma de nuestro futuro sistema operativo. Lo buscamos de la lista y le damos a siguiente. El siguiente menú es para seleccionar la franja horaria, buscamos en el mapa nuestra zona y a siguiente. La siguiente ventana nos permite decir la distribución de nuestras teclas.

Las siguientes opciones ya son más enfocadas a la configuración de Linux. Los datos que nos piden ahora son el nombre de usuario y la contraseña que vamos a tener a partir de ese momento hasta que lo cambiemos. Después de esta serie de ventanas ya nos pide seleccionar la partición en la que vamos a instalar el sistema operativo. Hay diferentes formas de hacerlo para distribuir las diferentes particiones. Si disponemos del sistema operativo Windows, este tiene una herramienta para configurar las particiones del disco duro. Hay programas en el mercado que ofrecen unas herramientas muy potentes, para administrar las particiones, crear copias de seguridad y todo lo relacionado con los discos duros. El ejemplo de un programa que podemos usar es el Partition Magic.

Seleccionamos la última opción, que es la que nos permite manualmente ver cuáles son las particiones de las que disponemos, y vemos cual es la que nosotros hemos tomado dedicada a Linux.

Fig 5.5.- Pantalla general de instalación

Al darle a aceptar, nos mostrara muy gráficamente un dibujo con los diferentes discos duros y como están particionados. Abajo te vienen los datos de cada partición en particular. Nosotros hemos creado una partición en particular para Linux, solo tenemos que identificarla y seleccionarla.

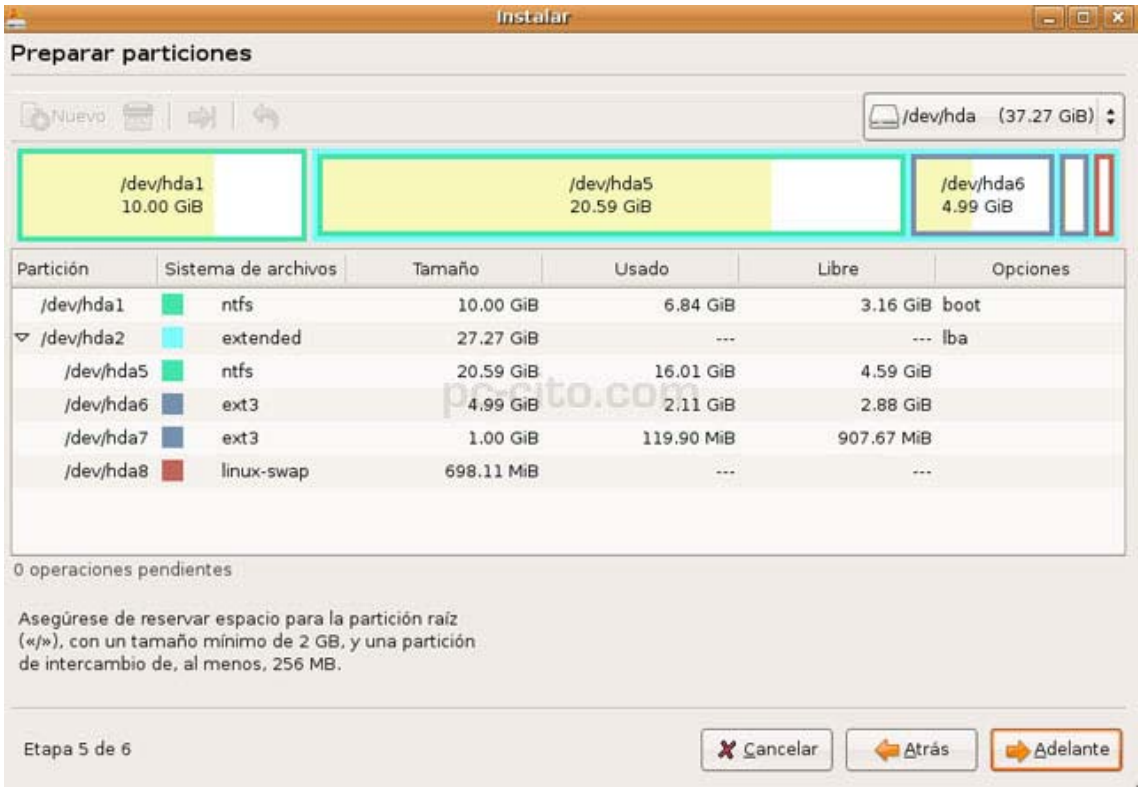


Fig. 5.6.- Pantalla de configuración manual de particiones

Cuando ya tengamos la tabla de particiones damos a adelante y asignaremos. Para esto hay que tener en cuenta:

/ ----> La partición en la que va a ir el sistema operativo, también conocido como directorio “root”.

/home ---> Ésta partición será como “Mis documentos” en Windows. Allí se guardarán todos los datos del usuario.

swap ----> Esta partición es una especia de memoria de intercambio para cuando necesite una memoria auxiliar

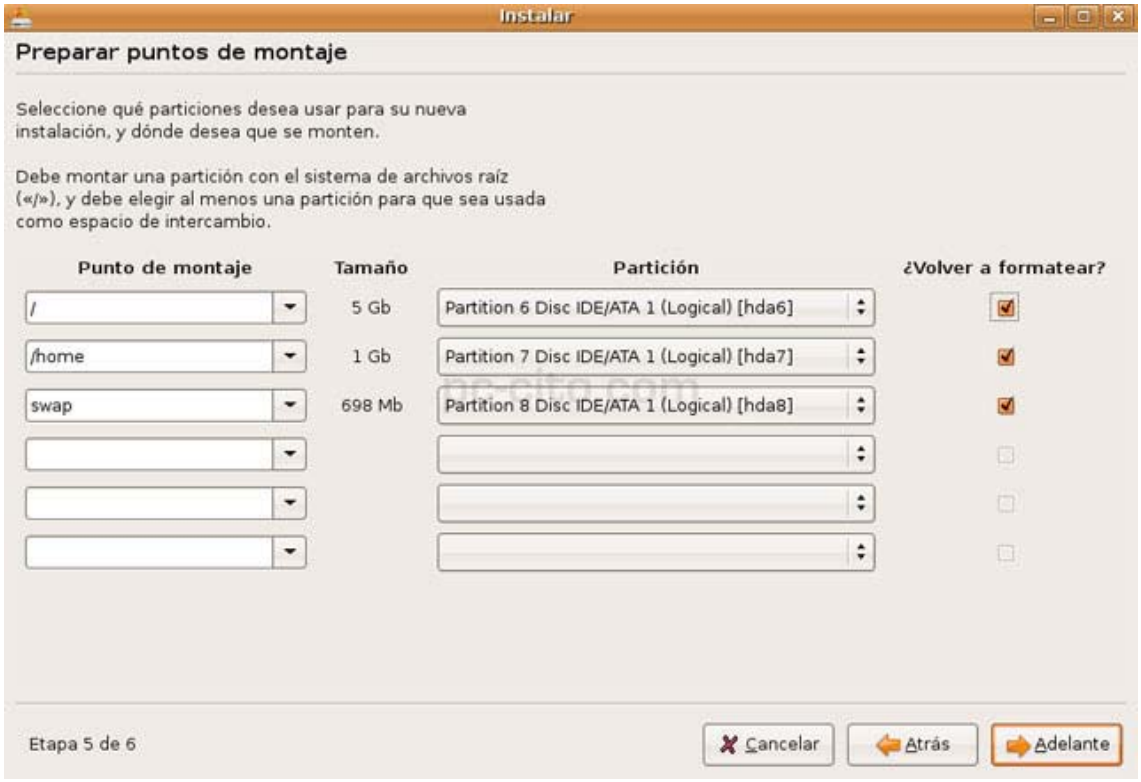


Fig. 5.7.- Distribución de las carpetas

Ahora tendremos una pantalla de confirmación donde está todo lo que hemos ido haciendo anteriormente. Si no está todo correcto damos a atrás. Si está todo correcto damos a Instalar y el equipo comenzará a instalar el Sistema Operativo.

Fig. 5.8.- Pantalla donde muestra todas las opciones que hemos ido configurando

Una vez haya terminado nos saldrá una ventana, seleccionaremos reiniciar ahora, el equipo se reiniciará y nos pedirá que retiremos el CD de instalación. La instalación ya ha concluido. Ahora cada vez que iniciemos el sistema nos saldrá una pantalla para elegir qué SO cargar.

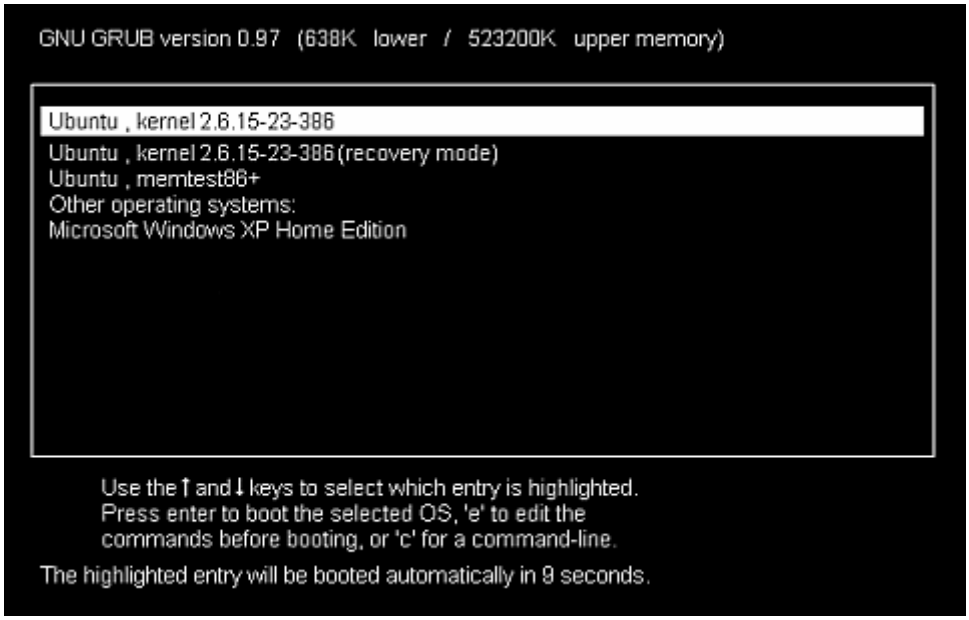


Fig 5.9.- Pantalla de selección del sistema operativo

Seleccionamos la primera opción. El sistema se cargará y dará paso a una pantalla de acceso a Linux, en la que deberemos de teclear nuestro usuario y contraseña. Ya después de todo esto seremos capaces de abrir el Linux.

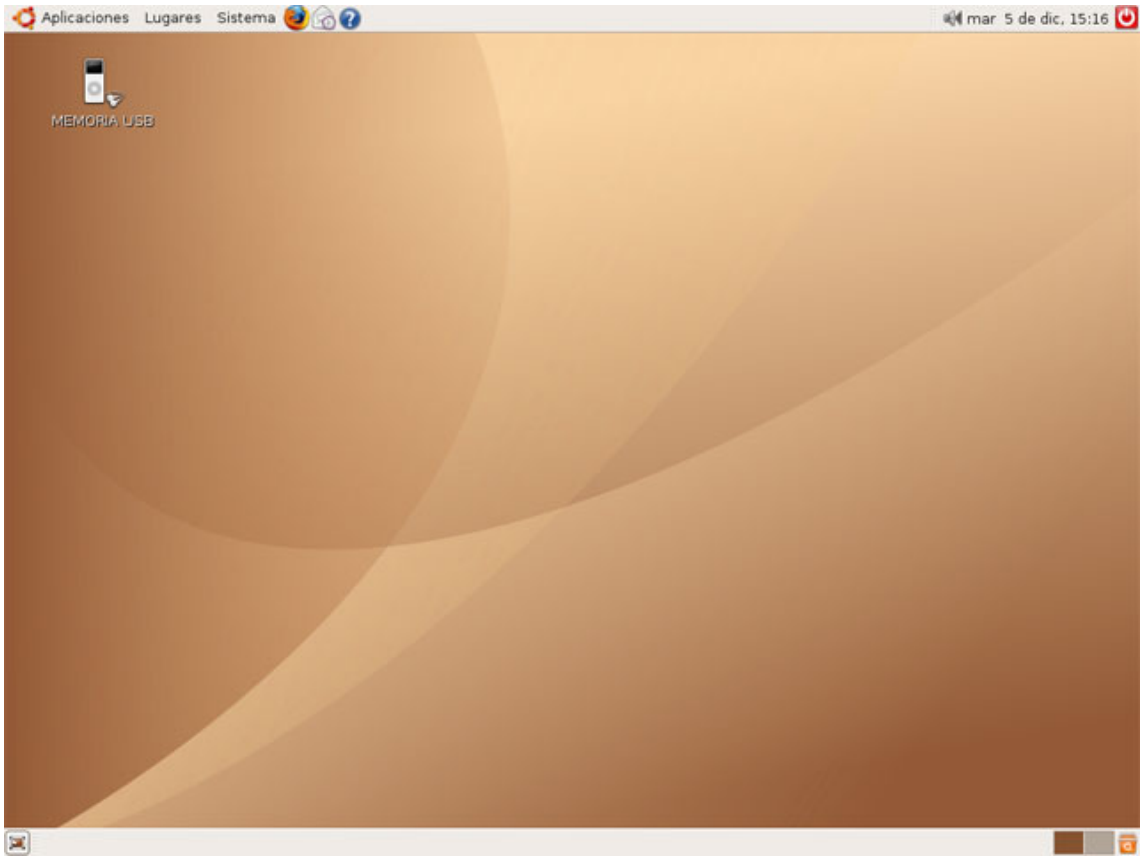


Fig 5.10.- Escritorio de Linux Ubuntu

5.5.- Introducción a la programación en ventanas

Una opción que se me planteó en el proyecto fue intentar programar algo para controlar el robot. Para eso una opción muy potente, y a la vez muy difícil, es programar directamente en un editor de texto y generar un código sin ayuda de ningún programa. Las ventajas son muchas, ya que el límite es tu imaginación, el inconvenientes es que las horas de trabajo son muchas.

La ventaja más importante es que los programas son independientes de la máquina en la que se ejecutan (o deberían serlo), el acceso a los dispositivos físicos se hace a través de interfaces, y nunca se accede directamente a dispositivos físicos. Esta es una de las principales ventajas para el programador, no hay que preocuparse por el modelo de tarjeta gráfica o de impresora, la aplicación funcionará con todas, y será el sistema operativo el que se encargue de que así sea.

Un problema que podemos encontrarnos el acceso a los recursos; un recurso es todo aquello que puede ser usado por una o varias aplicaciones. Existen recursos físicos, que son los dispositivos que componen el ordenador, como la memoria, la impresora, el teclado o el ratón y recursos virtuales o lógicos, como los gráficos, los iconos o las cadenas de caracteres.

Por ejemplo, si nuestra aplicación requiere el uso de un puerto serie, primero debe averiguar si está disponible, es decir, si existe y si no lo está usando otra aplicación; y después lo reservará para su uso. Esto es porque este tipo de recurso no puede ser compartido.

5.5.1.- La función principal

La función de entrada de un programa en ventanas es "WinMain", en lugar de la conocida "main" usada en la programación sencilla. Aunque la definición de esta función cambia muy poco de una aplicaciones a otras. Se divide en tres partes claramente diferenciadas:

declaración, inicialización y bucle de mensajes.

Parámetros de entrada de WinMain

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)

La función WinMain tiene cuatro parámetros de entrada:

-hInstance es un manipulador para la instancia del programa que estamos ejecutando. Cada vez que se ejecuta una aplicación, Windows crea una Instancia para ella, y le pasa un manipulador de dicha instancia a la aplicación.

-hPrevInstance es un manipulador a instancias previas de la misma aplicación. Como Linux es multitarea, pueden existir varias versiones de la misma aplicación ejecutándose, varias instancias. En sistemas operativos antiguos, este parámetro nos servía para saber si nuestra aplicación ya se estaba ejecutando, y de ese modo se podían compartir los datos comunes a todas las instancias. Pero eso era antes, ya que en Win32 usa un segmento distinto para cada instancia y este parámetro es siempre NULL, sólo se conserva por motivos de compatibilidad.

-lpszCmdParam, esta cadena contiene los argumentos de entrada del comando de línea.

-nCmdShow, este parámetro especifica cómo se mostrará la ventana. Para ver sus posibles valores consultar valores de nCmdShow. Se recomienda no usar este parámetro en la función ShowWindow la primera vez que se ésta es llamada. En su lugar debe usarse el valor SW_SHOWDEFAULT.

Primer ejemplo de ventana básica

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpszCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;
WNDCLASS WndClass;
WndClass.style = CS_HREDRAW | CS_VREDRAW;
```

```
WndClass.lpfnWndProc = WindowProcedure;
WndClass.cbClsExtra = 0;
WndClass.cbWndExtra = 0;
WndClass.hbrBackground = (HBRUSH) GetStockObject(LTGRAY_BRUSH);
WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
WndClass.hInstance = hInstance;
WndClass.lpszClassName = "NUESTRA_CLASE";
WndClass.lpszMenuName = NULL;
RegisterClass(&WndClass);
hWnd = CreateWindow(
"NUESTRA_CLASE",
"Ventana de Ejemplo",
WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT,
CW_USEDEFAULT,
320,
200,
HWND_DESKTOP,
NULL,
hInstance,
NULL
);
ShowWindow(hWnd, SW_SHOWDEFAULT);
while(TRUE == GetMessage(&Message, 0, 0, 0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}
```

En la primera zona hemos declarado las variables que necesitamos para nuestra función WinMain, que como mínimo serán tres:

-HWND hWnd, un manipulador para la ventana principal de la aplicación. Ya sabemos que nuestra aplicación necesitará al menos una ventana.

-MSG Message, una variable para manipular los mensajes que lleguen a nuestra aplicación.

-WNDCLASS WndClass, una estructura que se usará para registrar la clase particular de ventana que usaremos en nuestra aplicación.

Inicialización

Esta zona se encargará de registrar la clase, crear la ventana y visualizarla en pantalla.

Para registrar la clase primero hay que rellenar adecuadamente la estructura WNDCLASS, que define algunas características que serán comunes a todas las ventanas de una misma clase, como color de fondo, icono, menú por defecto, el procedimiento de ventana, etc. Después hay que llamar a la

función RegisterClass. A continuación se crea la ventana usando la función CreateWindow, que nos devuelve un manipulador de ventana.

Pero esto no muestra la ventana en la pantalla. Para que la ventana sea visible hay que llamar a la función ShowWindow. La primera vez que se llama a ésta función, después de crear la ventana, se puede usar el parámetro nCmdShow de WinMain como parámetro o mejor aún, como se recomienda por Windows, el valor SW_SHOWDEFAULT.

Bucle de mensajes

Este es el nucleo de la aplicación, permanece en este bucle mientras la función GetMessage retorne con un valor TRUE.

```
while(GetMessage(&Message, 0, 0, 0)) {
TranslateMessage(&Message);
DispatchMessage(&Message);
}
```

Este bucle no es recomendable, aunque se usa muy habitualmente. La razón es que la función GetMessage puede retornar tres valores: TRUE, FALSE o -1. El valor -1 indica un error, así que en este caso se debería abandonar el bucle. Durante mis pruebas descubrí un bucle mucho más estable:

```
while(TRUE == GetMessage(&Message, 0, 0, 0)) {
TranslateMessage(&Message);
DispatchMessage(&Message);
}
```

La función TranslateMessage se usa para traducir los mensajes de teclas virtuales a mensajes de carácter. Las teclas virtuales son las teclas de función, las teclas de las flechas, y las combinaciones de teclas de caracteres o números con las teclas ALT y CONTROL.

Los mensajes traducidos se reenvian a la lista de mensajes del proceso, y se recuperarán con las siguientes llamadas a GetMessage.

La función DispatchMessage envía el mensaje al procedimiento de ventana, donde será tratado adecuadamente. El próximo capítulo está dedicado al procedimiento de ventana, y al final de él estaremos en disposición de crear nuestro primer programa.

5.5.2. El procedimiento de ventanas

Cada ventana tiene una función asociada, ésta función se conoce como procedimiento de ventana, y es la encargada de procesar adecuadamente todos los mensajes enviados a una determinada clase de ventana. Es la responsable de todo lo relativo al aspecto y al comportamiento de una ventana. Normalmente, estas funciones están basadas en una estructura "switch" donde cada "case" corresponde aun determinado tipo de mensaje.

Sintaxis

```
LRESULT CALLBACK WindowProcedure(
HWND hwnd, // Manipulador de ventana
UINT msg, // Mensaje
WPARAM wParam, // Parámetro palabra, varía
LPARAM lParam // Parámetro doble palabra, varía
);
```

-hwnd es el manipulador de la ventana a la que está destinado el mensaje.
-msg es el código del mensaje.
-wParam es el parámetro de tipo palabra asociado al mensaje.
-lParam es el parámetro de tipo doble palabra asociado al mensaje.

Prototipo de procedimiento de ventana

```
LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM,
LPARAM);
```

Implementación del procedimiento de ventana simple

```
/* Esta función es llamada por la función del API DispatchMessage() */
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg,
WPARAM wParam, LPARAM lParam)
{
switch (msg) /* manipulador del mensaje */
{
case WM_DESTROY:
PostQuitMessage(0); /* envía un mensaje WM_QUIT a la cola de
mensajes */
break;
default: /* para los mensajes de los que no nos ocupamos */
return DefWindowProc(hwnd, msg, wParam, lParam);
}
return 0;
}
```

En general, habrá tantos procedimientos de ventana como programas diferentes y todos serán distintos, pero también tendrán algo en común: todos ellos procesarán los mensajes que lleguen a una ventana.

En este programa que intente ejecutar sólo procesa un tipo de mensaje, se trata de WM_DESTROY que es el mensaje que se envía a una ventana cuando se recibe un comando de cerrar, ya sea por menú o mediante el icono de aspa en la esquina superior derecha de la ventana.

Este mensaje sólo sirve para informar a la aplicación de que el usuario tiene la intención de abandonar la aplicación, y le da una oportunidad de dejar las cosas en su sitio: cerrar ficheros, liberar memoria, guardar variables, etc.

Incluso, la aplicación puede decidir que aún no es el momento adecuado para abandonar la aplicación. En el caso del ejemplo, efectivamente cierra la aplicación, y lo hace enviándole un mensaje WM_QUIT, mediante la función PostQuitMessage. El resto de los mensajes se procesan en el caso "default", y simplemente se cede su tratamiento a la función del API que hace el proceso por defecto para cada mensaje, DefWindowProc.

Este es el camino que sigue el mensaje WM_QUIT cuando llega, ya que el proceso por defecto para este mensaje es cerrar la aplicación.

5.5.3- Introducción de menus

Ahora que ya sabemos hacer el esqueleto de una aplicación de ventanas, veamos elprimer medio para comunicarnos con ella.

Ese medio es el llamado menú, que se trata de una ventana un tanto especial, del tipo pop-up, que contiene una lista de comandos u opciones entre las cuales el usuario puede elegir.

Cuando se usan en una aplicación, normalmente se agrupan varios menús bajo una barra horizontal, (que no es otra cosa que un menú), dividida en varias zonas o ítems.

Cada ítem de un menú que tenga asociado un comando o un valor, es decir todos menos los separadores y aquellos que despliegan nuevos menús, tiene asociado un identificador. Este valor se usa por la aplicación para saber qué opción se activó por el usuario, y decidir las acciones a tomar en consecuencia.

Existen varias formas de añadir un menú a una ventana, yo use una que parecía sencilla y a la vez estable.

Usando las funciones para inserción objeto a objeto

Este es el sistema más rudimentario, pero según estuve leyendo, era un método muy potente y fácil de usar:

l. Empezaremos viendo éste sistema porque ilustra mucho mejor la estructura de los menús.

Primero definamos algunas constantes:

```
#define CM_PRUEBA 100
#define CM_SALIR 101
```

Y añadamos la declaración de una función en la zona de prototipos:

```
void InsertarMenu(HWND);
```

Al final del programa añadimos la definición de esta función:

```
void InsertarMenu(HWND hWnd)
{
HMENU hMenu1, hMenu2;
hMenu1 = CreateMenu(); // Manipulador de la barra de menú
hMenu2 = CreateMenu(); // Manipulador para el primer menú pop-up
AppendMenu(hMenu2, MF_STRING, CM_PRUEBA, "&Prueba"); // Item 1 del
menú
AppendMenu(hMenu2, MF_SEPARATOR, 0, NULL); // Item 2 del menú
(separador)
AppendMenu(hMenu2, MF_STRING, CM_SALIR, "&Salir"); // Item 3 del
menú
AppendMenu(hMenu1, MF_STRING | MF_POPUP, (UINT)hMenu2,
"&Principal"); // Inserción del menú pop-up
SetMenu (hWnd, hMenu1); // Asigna el menú a la ventana hWnd
}
```

Y por último, sólo nos queda llamar a nuestra función, insertaremos ésta llamada justo antes de visualizar la ventana.

```
InsertarMenu(hWnd);
ShowWindow(hWnd, SW_SHOWDEFAULT);
```

La cosa mas curiosa que me encontré durante mi estudio de este tipo de programación fue la opción de HMENU. HMENU es un tipo de manipulador especial para menús. Necesitamos dos variables de este tipo, una para manipular la barra de menú, hMenu1. La otra para manipular cada uno de los menús pop-up, en este caso sólo uno, hMenu2. De momento solo me plantee hacer una barra de menú con un único elemento que será un menú pop-up. Después veremos como implementar menús más complejos.

Para crear un menú usaremos la función CreateMenu, que crea un menú vacío. Para ir añadiendo ítems a cada menú usaremos la función AppendMenu. Esta función tiene varios argumentos, el primero es el menú donde queremos insertar el nuevo ítem. El segundo son las opciones o atributos del nuevo ítem, por ejemplo MF_STRING, indica que se trata de un ítem de tipo texto, MF_SEPARATOR, es un ítem separador y MF_POPUP, indica que se trata de un menú que desplegará un nuevo menú pop-up.

El siguiente parámetro puede tener distintos significados:

- Puede ser un identificador de comando, éste identificador se usará para comunicar a la aplicación si el usuario selecionó un determinado ítem.
- Un manipulador de menú, si el ítem tiene el flag MF_POPUP, en éste caso hay que hacer un casting a (UINT).
- O puede ser cero, si se trata de un separador.

El último parámetro es el texto del ítem, cuando se ha especificado el flag MF_STRING, más adelante veremos que los ítems pueden ser también bitmaps. Normalmente se trata de una cadena de texto. Pero hay una peculiaridad interesante, para indicar la tecla que activa un determinado ítem

de un menú se muestra la letra correspondiente subrayada. Esto se consigue insertando un '&' justo antes de la letra que se quiere usar como atajo, por ejemplo, en el ítem "&Prueba" esta letra será la 'P'.

Por último SetMenu, asigna un menú a una ventana determinada. El primer parámetro es el manipulador de la ventana, y el segundo el del menú.

5.6.- Instalación del brazo robótico

Esta parte pertenece a mi compañero Pascual, que se encargó de toda la parte de instalación y de desarrollar este manual, tuve el privilegio de comprobar algunos pasos que realizó para tener instalado el brazo en el disco duro que compró la universidad, para intentar no depender siempre del fabricante. Este manual ha sido cedido y me han dicho que debería de formar parte de mi proyecto.

Después de tener una instalación Linux Debian instalada en tu ordenador se procede a lo siguiente para instalar todo lo necesario para mover el brazo robótico.

5.6.1. Instalación del kernel de tiempo real.

a) Montar el cd de robotnik (Robotnik Modular Robot Arm Installation v1.0) y copiar los siguientes ficheros con los comandos siguientes:

. $ sudo cp /media/cdrom/SW-Linux/Linux-RT_Kernel/patch-2.6.18-rt6 /usr/src
. $ sudo cp /media/cdrom/SW-Linux/Linux-RT_Kernel/linux-2.6.18.tar.bz2 /usr/src
. $ sudo cp /media/cdrom/SW-Linux/Linux-RT_Kernel/ciclictest-v0.11.tar.bz2.tar /opt

b) Instalar el patch del kernel 2.6.18 para tiempo real:

. $ sudo cd /usr/src
. $ sudo tar xjvf linux-2.6.18.tar.bz2
. $ sudo rm linux *(si existe este directorio)*
. $ sudo ln -sf /usr/src/linux-2.6.18 linux
. $ sudo cd /usr/src/linux
. $ sudo patch -p1 < ../patch-2.6.18-rt6

c) Configurar el kernel:

. $ sudo make menuconfig

O bien:

. $ sudo make xconfig

d) Comprobar que las siguientes opciones del kernel estén habilitadas o deshabilitadas y corregirlas:

- energy spare options Disabled
- ACPI Enabled
- APM Disabled
- high resolution timers support Enabled
- eliminar CONFIG_DEBUG_PREEMPT de .config

- Joystick options Enabled
- Load/Unload modules dynamically Enabled
- Processor type and features:
    - High Resolution Timer support
        - 1000 us Resolution
        - No symmetric Multiproc support
        - No generic x86 support
    - Preemption mode: Complete Preemption (Real Time)
- Kernel 2.6 –> Device Drivers → Input Device Support
    <*> Joysticks Interface
    <*> Event Interface
    [*] Joysticks
- USB Support →
    <*> Support for Host-side USB
    [*] USB device filesystem
    <*> EHCI HCD (USB 2.0) support
    <*> OHCI HCD support
    <*> UHCI HCD (most Intel and VIA) support
    <*> USB Human Interface Device (full HID) support
        [*] HID input layer support

En caso de tener problemas con la configuración del sistema hay una copia de un fichero de configuración del kernel funcionando en:

*/media/cdrom/Software_Linux/Linux_RT_Kernel/.config_*

e) Compilar el kernel con uno de los dos procedimientos siguientes:

Método 1. Configura automáticamente el inicio con grub, generando automáticamente el fichero initrd.

. $ sudo make-kpkg clean
. $ sudo make-kpkg - -append_to_version v1 kernel_image - -initrd modules_image

(Se crea un paquete llamado linux-imageXXX.deb en /usr/src)

. $ sudo dpkg -i linux-image-2.6.18-rt6v1.deb

(En Debian 4.0 la imagen puede crearse también con update_initramfs - -c -k 2.6.18-rt6, en /boot/initrd-img-2.6.18-rt6))

Para compilar e instalar solo los módulos: (saltar este paso si se ha hecho lo anterior)

. $ sudo make-kpkg modules
. $ sudo make-kpkg modules-image

Método 2. La imagen creada ha de copiarse a mano en /boot/, el fichero initrd ha de crearse manualmente y el inicio con grub ha de configurarse también manualmente.

. $ sudo make clean
. $ sudo make bzImage
. $ sudo make modules
. $ sudo make modules_install
. $ sudo mkinitrd - -o /boot/initrd.img-2.6.18-rt6

. $ sudo cd ./arch/i386/boot/
. $ sudo cp vmlinuz /boot/vmlinuz-2.6.18-rt6

. $ sudo vim /boot/grub/menu.lst

```
title     Debian GNU/Linux, kernel 2.6.18-rt6v1
root     (hd0,0)
kernel /boot/vmlinuz-2.6.18-rt6v1 root=/dev/hda1 ro
initrd   /boot/initrd.img-2.6.18-rt6v1
savedefault
boot

title     Debian GNU/Linux, kernel 2.6.18-rt6v1 (recovery mode)
root     (hd0,0)
kernel /boot/vmlinuz-2.6.18-rt6v1 root=/dev/hda1 ro single
initrd   /boot/initrd.img-2.6.18-rt6v1
savedefault
boot
```

f) Comprobar la actualización:

. $ sudo update-grub

g) Reiniciar el sistema:

. $ sudo reboot

h) Comprobar el núcleo instalado:

. $ sudo uname -a

Linux modular 2.6.18-rt6v1 #1 PREEMPT Lun Dec 1 00:00:00 CET 2008 i686 GNU/Linux

i) Comprobar si el Real Time System se ha iniciado correctamente. El siguiente mensaje debe aparacer en el fichero /var/log/dmesg:

hrtimers: Switched to high resolution mode CPU 0

j) Comprobar la funcionalidad y rendimiento del Real Time System. Debe obtenerse un reducido valor de jitter, dependiendo de la placa base y del microprocesador:

. $ sudo cd /opt
. $ sudo tar xvf cyclictest-v0.11.tar.bz2.tar
. $ sudo cd cyclictest
. $ sudo make
. $ sudo ./cyclictest -p 80 -t 5 -n -I 1000

0.15 0.04 0.01 3/150 3348
T:0 (3344) P:80 I:1000 C: 1000 Min: 1 Act: 2 Avg: 1 Max: 8
T:1 (3345) P:79 I:1500 C: 668 Min: 1 Act: 2 Avg: 1 Max: 6
T:2 (3346) P:78 I:2000 C: 501 Min: 1 Act: 5 Avg: 1 Max: 6
T:3 (3347) P:77 I:2500 C: 401 Min: 1 Act: 4 Avg: 1 Max: 6
T:4 (3348) P:76 I:3000 C: 334 Min: 1 Act: 1 Avg: 1 Max: 5

Este ejemplo comienza 5 procesos y muestra el jitter mínimo, actual y máximo en microsegundos. El máximo valor es de 8 ms. Pueden esperarse valores de jitter máximo entre 5 y 50 us. Si aparecen valores mucho más altos, el sistema no está trabajando correctamente.

Para comprobar completamente el sistema se recomienda leer el fichero RT_PREEMPT_HOWTO.htm del directorio */media/cdrom/Software_Linux/Linux_RT_Kernel/docs.*

Los resultados obtenidos son:

T:4 (3473) P:76 I:3000 C: 336 Min: 15 Act: 23 Avg: 19 Max: 28

5.6.2. Instalación de los drivers del brazo robótico.

Las librerías a instalar son open source y algunas están disponibles en internet. Están recopiladas en el CD1 de Robotnik. La instalación se realizará siempre en el directorio /opt. Antes hay que crear los directorios:

. $ sudo mkdir /opt/modular
. $ sudo mkdir /opt/modular/lib

Para compilarlas es necesario tener los headers del núcleo -linux-headers, (en ningún caso es es necesario tener los fuentes del núcleo -linux-

sources). Los headers han de ser los correspondientes al núcleo que se está ejecutando mientras se compila (en nuestro caso, kernel rt).

a) Driver ESDCAN: controla la tarjeta del bus CAN (error en Ubuntu 8.04.1)

Copiar la librería en /opt y desempaquetarla:
. $ sudo cp /(CD1)/.../esdcan_3.tgz /opt
. $ sudo cd /opt
. $ sudo tar zxvf esdcan_3.tgz

Comprobar la versión del compilador gcc (3.3 ó 3.4):
. $ gcc -v

Editar el fichero config.mk con las siguientes opciones:
GET_OS=linux
TARGET_ARCH=x86
BOARD=pci331
BOARD_NAME=CAN_PCI331
NTC_LIB_VERSION=3.1.0
KERNELPATH=/lib/modules/'uname -r'/build

Compilarla:
. $ sudo cd esdcan-pci331-linux-x86-3.5.0-k26
. $ sudo make clean
. $ sudo make

Copiar el módulo creado en el directorio kernel/drivers:
. $ sudo cp ./src/esdcan-pci331.ko /lib/modules/2.6.18-rt6v1/kernel/drivers

Configurar el sistema para cargarla automáticamente durante el inicio:
. $ sudo vi /etc/modules
esdcan-pci331 (añadir esta línea)

. $ sudo update-modules
. $ sudo depmod

Comprobar que el módulo se ha cargado correctamente:
. $ sudo insmod ./esdcan-pci331.ko
. $ sudo modprobe esdcan-pci331
. $ less /var/log/messages
esd CAN driver: CAN_PCI331
esd CAN driver: baudrate not set
...
esd CAN driver: version 5.5.0.09 successfully loaded
Crear los nodos en el directorio /dev/
. $ cd /dev/
. $ sudo mknod - -mode=a+rw can0 c 50 0
. $ sudo mknod - -mode=a+rw can1 c 50 1

*. $ ls -l /dev/can**

Nota: sólo si se ha cambiado el kernel de la versión 2.4 a la 2.6

. $ cd /opt/esdcan-pci331-linux-x86-3.5.0-k26/lib32
. $ sudo cp libntcan.so.3.1.0 /usr/local/lib
. $ sudo cp libntcan.a /usr/local/lib

Copiar la librería recién creada en el directorio lib de modular:
. $ sudo cp /opt/esdcan-pci331-linux-x86-3.5.0-k26/lib32/libntcan.a /opt/modular/lib

Observaciones:

- En Ubuntu, con un kernel 2.6.25, la librería esdcan-pci331-linux-x86-3.5.0-k26 no compila correctamente debido a que los ficheros fuente esdcan.c y otros incluyen linux/config.h y linux/ioctl32.h, ficheros que ya no se encuentran entre los headers de ese núcleo. Para usar el bus can se deberá usar otra librería, como la esdcan-usb331-linux-2.6.x-x86_64-3.8.3 con una tarjeta de bus CAN por usb, ya que este paquete sí compila.

- Otras versiones más actualizadas de la librería de bus CAN en http://eris.liralab.it/wiki/Esd_Can_Bus.

b) Librería m5api: controla los módulos PowerCube (ok en Ubuntu 8.04.1)

Copiar la librería en /opt:
. $ sudo cp /(CD1)/.../M5DLL_SUSE_10.0.tar /opt

Desempaquetarla:
. $ cd /opt
. $ sudo tar xvf M5DLL_SUSE_10.0.tar

Crear el directorio lib en /opt/m5api
. $ cd /opt/m5api
. $ sudo mkdir lib

Compilar librerías:
. $ cd Device
. $ sudo make
. $ cd M5apiw32
. $ sudo make m5api
. $ cd ../Util
. $ sudo make

Copiar las librerías en lib
. $ cd /opt/m5api/lib
*. $ sudo cp *.a /opt/modular/lib/*

c) Librería Roboop (ok en Ubuntu 8.04.1)

Copiar la librería en /opt, compilarla y copiarala en lib:
. $ sudo cp /(CD1)/.../roboop.tar /opt
. $ cd /opt
. $ sudo tar xvf roboop.tar
. $ cd roboop
. $ sudo make -f makefile.gcc
. $ sudo cp libnewmat.a /opt/modular/lib
. $ sudo cp libroboop.a /opt/modular/lib

Nota: el paquete boost (se puede obtener vía Synaptics) debe estar presente el el sistema para poder compilar.

d) Librería RAPID (ok en Ubuntu 8.04.1)

Copiar el directorio Vcolide201.lin en /opt, compilarla y copiarla en lib:
. $ sudo /(CD1)/.../VColide201.lin /opt
. $ cd /opt/VColide201.lin/RAPID
. $ sudo make clean
. $ sudo make
. $ sudo cp librapid.a /opt/modular/lib/

d) Librería Bhand (ok en Ubuntu 8.04.1)

Copiar, compilar e instalar
. $ cd /opt/modular/libbhand
. $ sudo make
. $ sudo cp libbhand.a /opt/modular/lib/

No compila correctamente a menos que se edite el fichero BHand.h y se inserte la siguiente línea:

DWORD _stdcall _ComThreadFunction(void pHand);*

justo antes de:

//Class Bhand declaration

5.6.3. Instalación de la librería OpenJAUS

Copiar los siguientes archivos del CD1:
. $ sudo cp jdk-6u1-linux-i586-bin /opt
. $ sudo cp CIMAR.tar /opt

Instalar el entorno de desarrollo java:
. $ cd /opt
. $ sudo chmod +x jdk-6u1-linux-i586.bin

. $ sudo ./jdk-6u1-linux-i586.bin
(aceptar el contrato de licencia)
. $ cd /usr/bin
. $ sudo ln -sf /opt/jdk1.6.0_01/bin/java java
. $ sudo ln -sf /opt/jdk1.6.0_01/bin/javac javac
. $ sudo ln -sf /opt/jdk1.6.0_01/bin/javadoc javadoc
. $ sudo ln -sf /opt/jdk1.6.0_01/bin/javah javah
. $ sudo ln -sf /opt/jdk1.6.0_01/bin/javap javap

Instalar OpenJAUS:
. $ cd /opt
. $ sudo tar xvf CIMAR.tar
. $ cd /opt/CIMAR/Core

. $ sudo make distclean
. $ modificar el fichero jaus.h para que desactivar la elección de arquitectura y solo descomentar la de i386
. $ introducirse en libjausC, libcimar y libnodeManager para ejecutar ./configure

. $ cd /opt/CIMAR/Core/libjausC
. $ sudo vi Makefile
en la línea DEFAULT_INCLUDES añadir esto:
-I/opt/CIMAR/Core/libcimar/include)
. $ cd /opt/CIMAR/Core/libnodeManager
. $ sudo vi Makefile
en la línea DEFAULT_INCLUDES añadir esto:
-I/opt/CIMAR/Core/libjausC/include
-I/opt/CIMAR/Core/libcimar/include)
. $ cd /opt/CIMAR/Core
. $ sudo make
. $ sudo make install
. $ introducirse en libcimar, libjausC y libnodeManager para ejecutar ./Install-CIMAR.sh en cada uno de ellos
. $ no hacer make clean!

. $ cd /opt/CIMAR/Core/nodeManager
. $ sudo ./NM
(si da error 'log4j' puede ser porque los enlaces simbólicos de java tienen algún error)

INFO: openjaus.nodemanager.NodeManager – NodeManager starting
OpenJAUS Node Manager Version: 3.2 (03/22/2006)
JAUS Address: 1.9.1
INFO openjaus.nodemanager.NodeManager - NodeManager: Network Interface: eth0
INFO openjaus.nodemanager.NodeManager - Node IP Address: /192.168.0.9
...

INFO openjaus.nodemanager.SendThread - SendThread starting
INFO openjaus.nodemanager.ReceiveThread - ReceiveThread running
...

5.6.4.- Instalación del software RESCUER MODULAR (en Ubuntu 8.04.1

Copiar el directorio MODULAR del CD1 en la capeta de trabajo (/opt, /root, /home/modular):

. $ sudo cp /(CD1)/.../MODULAR /opt
. $ cd /opt/MODULAR
. $ sudo make clean
. $ sudo make

OBS1: si no se copian antes las librerías creadas en /opt/modules/lib en /opt/MODULAR/lib no se puede compilar.

OBS2: a pesar de eso hay errores de compilación: comentando la línea 124 de utils.h (short sign(const Real x), evitamos errores al compilar.

Iniciar el NodeManager (el programa que gestiona la comunicación y sincronización entre nodos):

. $ cd /opt/CIMAR/Core/nodeManager
. $ sudo ./NM

Desde un terminal diferente iniciar el programa de control del robot Modular:

. $ cd /opt/modular
. $ sudo ./bin/modular -d4

- El parámetro -d4 indica el nivel de depuración. Esta opción fueza que los mensajes de la función cDebug se escriban en /var/log/MODULAR/modular.log)

- La ejecución del programa modular debe detectarla el NodeManager. Desde el terminal de NodeManager puede teclearse 't' para mostrar los componentes activos.

Compartiendo directorios con nfs o ssh:

Para trabajar remotamente en el PC MODULAR se recomienda instalar los servidores nfs o ssh para compartir los directorios de código fuente y trabajar con ellos desde otro lugar.

Para nfs:

- Instalar los paquetes nfs-common, nfs-kernel-server, portmap
- En /etc/exports añadir los directorios a ser compartidos:
  /home/robotnik 192.168.1.0/24 (rw)

- En /etc/hosts.allow añadir la ip del pc remoto:
ALL: 192.168.1.106

Scrips de inicio:

1) autologin

Este script se encuentra en el CD1 de instalación y sirve para presentarse automáticamente en un terminal:

```
#!/bin/sh
exec 0</dev/$1 1>/dev/$1 2>&1
cat /etc/issue
shift
exec $*
```

Copiar este fichero como /sbin/autologin y darle permisos de ejecución (chmod +x autologin).

2) inittab

Este script configura el init y se encuentra en /etc/inittab. Ha de modificarse sustituyendo las siguientes líneas:

```
2:23:respawn: /sbin/getty 38400 tty2
3:23:respawn: /sbin/getty 38400 tty3
4:23:respawn: /sbin/getty 38400 tty4
```

por:

```
2:23:respawn: /sbin/autologin tty2 login -f user
3:23:respawn: /sbin/autologin tty3 login -f user
4:23:respawn: /sbin/autologin tty3 login -f root
```

Donde user es cualquier usuario del sistema. Con este cambio los terminales 2, 3 y 4 se presentarán automáticamente cuando se inicie el PC.

3) .bashrc

Este script está en el directorio home de cada usuario. Hay que editarlo para cada usuario del script anterior (/home/user/.bashrc), añadiendo el siguiente código al final de los ficheros:

```
echo “ROBOTNIK MODULAR ROBOT ARM”
Terminal1='tty'

case $Terminal in
        “/dev/tty2”)    /opt/CIMAR/NM;;          # Comienza NODE
MANAGER en el terminal 2
```

```
        "/dev/tty3")    cd /opt/MODULAR; echo "Enlazando modular...";
sleep 5;
        ./bin/modular ;;                    # Comienza modular en el
terminal 3
        Esac
```

6.1.- Conclusiones finales

Este proyecto me ha llevado a conocer una serie de cosas que me creo que me van a servir mucho a lo largo de mi carrera profesional.

Una de las grandes ventajas de esta época informática es la posibilidad de visualizar videos, hay algunos que muestran el estado de la robótica, esta imagen de abajo muestra el robot ASIMO subiendo unas escaleras, el problema es que por un error de cálculo, cae rodando por las escaleras. Esto demuestra que aún hay mucho camino que avanzar en esto de la robótica.

Fig. 6.1.- ASIMO antes del aparatoso accidente

Vista general sobre el estado actual de la robótica

Los robots son utilizados en una diversidad de aplicaciones, desde robots tortugas en los salones de clases, robots soldadores en la industria automotriz, hasta brazos teleoperados en el transbordador espacial.

Cada robot lleva consigo su problemática propia y sus soluciones afines; no obstante que mucha gente considera que la automatización de procesos a través de robots está en sus inicios, es un hecho innegable que la introducción de la tecnología robótica en la industria, ya ha causado un gran impacto. En este sentido la industria Automotriz desempeña un papel preponderante.

Es necesario hacer mención de los problemas de tipo social, económicos e incluso político, que puede generar una mala orientación de robotización de la industria. Se hace indispensable que la planificación de los recursos humanos, tecnológicos y financieros se realice de una manera inteligente.

. Para muchos la idea de tener un robot agricultor es ciencia ficción, pero la realidad es muy diferente; o al menos así parece ser para el Instituto de Investigación Australiano, el cual ha invertido una gran cantidad

de dinero y tiempo en el desarrollo de este tipo de robots. Entre sus proyectos se encuentra una máquina que esquila a las ovejas. La trayectoria del cortador sobre el cuerpo de las ovejas se planea con un modelo geométrico de la oveja.

Para compensar el tamaño entre la oveja real y el modelo, se tiene un conjunto de sensores que registran la información de la respiración del animal como de su mismo tamaño, ésta es mandada a una computadora que realiza las compensaciones necesarias y modifica la trayectoria del cortador en tiempo real.

Debido a la escasez de trabajadores en los mataderos de cerdos, se desarrolla otro proyecto, que consiste en hacer un sistema automatizado de un obrador, el prototipo requiere un alto nivel de coordinación entre una cámara de vídeo y el efector final que realiza en menos de 30 segundos ocho cortes al cuerpo del cerdo.

Por su parte en Francia se hacen aplicaciones de tipo experimental para incluir a los robots en la siembra, y poda de los viñedos, como en la pizca de la manzana.

La exploración espacial posee problemas especiales para el uso de robots. El medio ambiente es hostil para el ser humano, quien requiere un equipo de protección muy costoso tanto en la Tierra como en el Espacio. Muchos científicos han hecho la sugerencia de que es necesario el uso de Robots para continuar con los avances en la exploración espacial; pero como todavía no se llega a un grado de automatización tan precisa para ésta aplicación, el ser humano aún no ha podido ser reemplazado por estos. Por su parte, son los teleoperadores los que han encontrado aplicación en los transbordadores espaciales.

Espacio

En Marzo de 1982 el transbordador Columbia fue el primero en utilizar este tipo de robots, aunque el ser humano participa en la realización del control de lazo cerrado.

Algunas investigaciones están encaminadas al diseño, construcción y control de vehículos autónomos, los cuales llevarán a bordo complejos laboratorios y cámaras muy sofisticadas para la exploración de otros planetas.

En Noviembre de 1970 los Rusos consiguieron el alunizaje del Lunokhod 1, el cual poseía cámaras de televisión, sensores y un pequeño laboratorio, era controlado remotamente desde la tierra.

En Julio de 1976, los Norteamericanos aterrizaron en Marte el Viking 1, llevaba abordo un brazo robotizado, el cual recogía muestras de piedra, tierra y otros elementos las cuales eran analizados en el laboratorio que fue acondicionado en el interior del robot. Por supuesto también contaba con un equipo muy sofisticado de cámaras de vídeo.

Vehículos submarinos

Dos eventos durante el verano de 1985 provocaron el incremento por el interés de los vehículos submarinos. En el primero - Un avión de la Air Indian se estrelló en el Océano Atlántico cerca de las costas de Irlanda - un vehículo submarino guiado remotamente, normalmente utilizado para el tendido de cable, fue utilizado para encontrar y recobrar la caja negra del avión. El segundo fue el descubrimiento del Titanic en el fondo de un cañón, donde había permanecido después del choque con un iceberg en 1912, cuatro kilómetros abajo de la superficie. Un vehículo submarino fue utilizado para encontrar, explorar y filmar el hallazgo.

En la actualidad muchos de estos vehículos submarinos se utilizan en la inspección y mantenimiento de tuberías que conducen petróleo, gas o aceite en las plataformas oceánicas; en el tendido e inspección del cableado para comunicaciones, para investigaciones geológicas y geofísicas en el suelo marino.

La tendencia hacia el estudio e investigación de este tipo de robots se incrementará a medida que la industria se interese aún más en la utilización de los robots, sobra mencionar los beneficios que se obtendrían si se consigue una tecnología segura para la exploración del suelo marino y la explotación del mismo.

Educación

Los robots están apareciendo en los salones de clases de tres distintas formas. Primero, los programas educacionales utilizan la simulación de control de robots como un medio de enseñanza. Un ejemplo palpable es la utilización del lenguaje de programación del robot Karel, el cual es un subconjunto de Pascal; este es utilizado por la introducción a la enseñanza de la programación.

El segundo y de uso más común es el uso del robot tortuga en conjunción con el lenguaje LOGO para enseñar ciencias computacionales. LOGO fue creado con la intención de proporcionar al estudiante un medio natural y divertido en el aprendizaje de las matemáticas.

En tercer lugar está el uso de los robots en los salones de clases. Una serie de manipuladores de bajo costo, robots móviles, y sistemas completos han sido desarrollados para su utilización en los laboratorios educacionales. Debido a su bajo costo muchos de estos sistemas no poseen una fiabilidad en su sistema mecánico, tienen poca exactitud, no existen los sensores y en su mayoría carecen de software.

El mercado de la robótica y las perspectivas futuras

Las ventas anuales para robots industriales han ido creciendo en Estados Unidos a razón del 25% de acuerdo a estadísticas del año 1981 a 1992. El incremento de ésta tasa se debe a factores muy diversos. En primer lugar, hay más personas en la industria que tienen conocimiento de la

tecnología y de su potencial para sus aplicaciones de utilidad. En segundo lugar, la tecnología de la robótica mejorará en los próximos años de manera que hará a los robots más amistosos con el usuario, más fáciles de interconectar con otro hardware y más sencillos de instalar.

En tercer lugar, que crece el mercado, son previsibles economías de escala en la producción de robots para proporcionar una reducción en el precio unitario, lo que haría los proyectos de aplicaciones de robots más fáciles de justificar. En cuarto lugar se espera que el mercado de la robótica sufra una expansión más allá de las grandes empresas, que ha sido el cliente tradicional para ésta tecnología, y llegue a las empresas de tamaño mediano, pequeño y por que no; las microempresas. Estas circunstancias darán un notable incremento en las bases de clientes para los robots.

La robótica es una tecnología con futuro y también para el futuro. Si continúan las tendencias actuales, y si algunos de los estudios de investigación en el laboratorio actualmente en curso se convierten finalmente en una tecnología factible, los robots del futuro serán unidades móviles con uno o más brazos, capacidades de sensores múltiples y con la misma potencia de procesamiento de datos y de cálculo que las grandes computadoras actuales. Serán capaces de responder a órdenes dadas con voz humana. Así mismo serán capaces de recibir instrucciones generales y traducirlas, con el uso de la inteligencia artificial en un conjunto específico de acciones requeridas para llevarlas a cabo. Podrán ver, oír, palpar, aplicar una fuerza media con precisión a un objeto y desplazarse por sus propios medios.

En resumen, los futuros robots tendrían muchos de los atributos de los seres humanos. Es difícil pensar que los robots llegarán a sustituir a los seres humanos en el sentido de la obra de Carel Kapek, Robots Universales de Rossum. Por el contrario, la robótica es una tecnología que solo puede destinarse al beneficio de la humanidad. Sin embargo, como otras tecnologías, hay peligros potenciales implicados y deben establecerse salvaguardas para no permitir su uso pernicioso.

El paso del presente al futuro exigirá mucho trabajo de ingeniería mecánica, ingeniería electrónica, informática, ingeniería industrial, tecnología de materiales, ingenierías de sistemas de fabricación y ciencias sociales.

Locomoción

Uno de los grandes retos en el área de la robótica, es el introducir locomoción **capaz de moverse por cualquier tipo de entorno**, por muy escarpado o difícil de practicar que sea. Esto tiene especial interés en la exploración de otros planetas, en los que no se sabe qué tipo de terreno nos podemos encontrar.

Hasta 1993, se diseñaban robots para cada tipo de terreno. En 1994 apareció el primer robot basado en los paradigmas de la robótica modular reconfigurable :Polybot. ¿Por qué no trabajar en una nueva línea de

investigación en la que no se diseñen robots específicos para cada terreno, sino que el propio robot se adapte al terreno, modificando su forma y su manera de de desplazarse?

Mark Yim se puede considerar como el padre de esta disciplina. Actualmente está en el PARC, trabajando en la tercera generación de módulos para Polybot.

El diseño de robots ápodos se puede abordar desde esta perspectiva. En vez de diseñar un robot ápodo específico, es mejor diseñar un módulo sencillo que se pueda unir formando cadenas. Este mismo módulo también servirá para construir otros robots diferentes.

La mayoría de estos sistemas están enfocados para ser capaces de desplazarse por cualquier superficie. La inspiración viene de la naturaleza, los robots capaces de moverse por cualquier superficie están inspirados en insectos sobre todo, ya que suelen ser capaces de adaptarse a superficies bastante difíciles.

Biomedicina

Un uso que puede ser el futuro de las manos robóticas es en la introducción en el sector de la medicina. Un equipo de investigadores de las Universidades de Harvard y Yale han desarrollado una mano robótica delicada y simple capaz de agarrar una serie de objetos suavemente, y que ajusta automáticamente sus dedos para proporcionar una buena sujeción. La nueva mano también se podría utilizar como brazo prostético.

“Cuando empiezas a utilizar robots en el entorno humano, el hecho de poder adaptarse es muy ventajoso,” afirma Charlie Kemp, profesor asistente en Georgia Tech y que se dedica al diseño de robots para la asistencia en el hogar. “No siempre sabes dónde están las cosas. No es cuestión de abrirse camino y acabar rompiendo cualquier cosa. Tienes que hacer que la mecánica se adapte.”

Hay otros investigadores que ya han creado manos robóticas delicadas, tales como el Sensopac de Intel, u Obrero, creado por MIT, aunque normalmente están cargadas con diversos tipos de complejos sensores y motores, y necesitan utilizar potentes algoritmos para tener en cuenta cada movimiento de los dedos o articulaciones. Por el contrario, la nueva mano robótica sólo posee unos pocos sensores y un único motor, aunque puede recoger una gran variedad de objetos con la flexibilidad de una mano humana.

Aaron Dollar, profesor asistente en la Universidad de Yale y director del estudio, señala que cuando alcanzamos un objeto normalmente no lo agarramos con fuerza, sino que nuestros dedos están relajados para así evitar que el objeto se caiga. Al hacer que la mano del robot sea flexible se consigue recoger objetos incluso con errores de cálculo mínimos. Los sensores integrados también permiten a esta nueva mano sentir el objeto y ajustar el

agarre. Es muy similar, afirma Dollar, a si tomas una taza de café con los ojos cerrados: primero calculas el agarre necesario antes de tomarla con la mano.

"Básicamente no sabemos nada acerca del objeto antes de agarrarlo, alargamos la mano hacia adelante a través de un sencillo algoritmo, para tener un mejor conocimiento de la localización del objeto," afirma Dollar. La licencia de uso de la mano acaba de ser otorgada aBarret Technology, una compañía con sede en Cambridge, Massachusetts, que se dedica a la venta de manos robóticas para tareas de investigación.

"Han logrado demostrar que la mano se adapta, por lo que no tienes que posicionarla de forma precisa como ocurre con otros tipos de manos," afirma Kemp en relación al trabajo de Dollar. "El robot no tiene que saber exactamente dónde está el objeto, y en los entornos humanos esto es algo que resulta de mucho valor." Añade que un agarre rígido requiere un control y una ejecución de mayor precisión—y, consecuentemente, una mayor potencia de proceso informático.

Un futuro lejano

Durante mi investigación, me encontre diferentes, articulos, y uno de ellos me pareció muy curioso, este hablaba sobre un futuro de la robótica un poco catastrófico, y me parece interesante introducirlo, como nota curiosa.

De tanto en tanto, un grupo de científicos especialistas en robótica e inteligencia artificial se reúne en algún sitio para analizar el estado de la tecnología que ellos mismos están desarrollando y determinar las posibles consecuencias que podrían tener sus inventos en el futuro de la humanidad. No siempre tenemos noticias de los resultados de estas reuniones, y con frecuencia no tienen siquiera un carácter "oficial". Pero hace unos días ha tenido lugar una reunión de este tipo en la Asilomar Conference Grounds, desarrollada en California (EE.UU). Entre los asistentes a la misma se encontraban personalidades como Eric Horvitz, Raymond Kurzweil, Paul Berg o Tom Mitchell, quienes debatieron sobre el futuro de la robótica. Las conclusiones a la que han llegado son -como mínimo- inquietantes.

Todos hemos visto cómo en los últimos años la robótica ha conseguido logros de todo tipo. Sabemos que agencias como DARPA han desarrollado modelos capaces de cortarte en pedacitos y biodigerirte para obtener energía, o empresas que desarrollan asistentes robóticas capaces de trabajar dentro de un quirófano codo a codo con los cirujanos humanos. La electrónica y la inteligencia artificial han avanzado tanto, y han tenido una difusión tan grande, que incluso un estudiante o aficionado puede, sin gastarse una fortuna, construir un robot más que interesante en su propia casa.

Esta situación hace que los especialistas en robótica, conocedores de los entresijos de las tecnologías implicadas en el desarrollo de sus criaturas, se muestren preocupados por el rumbo que están tomando las cosas. Dejando bien en claro que la robótica es una ciencia capaz de solucionar muchos de los problemas que enfrenta la humanidad (como la limpieza de residuos nucleares

o la ejecución de tareas repetitivas y muy precisas) y que los robots pueden ser grandes aliados de los humanos a la hora de (por ejemplo) explorar el Sistema Solar, los científicos creen que el rumbo que están tomando algunos desarrollos debería ser reconsiderado.

Disponemos de cerebros electrónicos que poseen la inteligencia equivalente a una cucaracha. Dicho de esta forma no parece algo muy impresionante, pero si miramos hacia atrás, hace un puñado de años no teníamos nada. Si los desarrollos de este tipo siguen una curva semejante a la Ley de Moore (o incluso una mucho menos pronunciada), puede que en pocas décadas tengamos robots tan inteligentes como nosotros mismos. Una situación como la planteada en “2001: Una odisea espacial”, donde un ordenador loco pone en peligro la vida los tripulantes de una nave espacial operando los sistemas robóticas de abordo ya no es algo tan remoto. “El desarrollo de la inteligencia artificial debería ser controlado de forma activa antes de tener máquinas superinteligentes correteando (y eventualmente disparando) por allí.”, dijo Eric Horvitz, investigador de Microsoft y presidente de la asociación Association for the Advancement of Artificial Intelligence(AAAI).

Otros, como el cofundador de Sun Microsystems, William Joy o el famoso científico -tomador profesional de pastillas- y creador del concepto de “singularidad” Raymond Kurzweil, dicen que estamos asistiendo a las primeras señales de lo que ellos llaman “una explosión de la inteligencia artificial”. Insisten en que cada día se dan pequeños pasos que no llaman demasiado la atención, pero que visto con una perspectiva adecuada, la IA avanza a una velocidad asombrosa. Al hombre le ha tomado millones de años desarrollar su capacidad intelectual, pero a los robots solo le llevará un siglo o dos superarlo. A pesar de lo incipiente de esta tecnología, ya tenemos máquinas capaces de buscar un tomacorriente para alimentarse, de pilotear un avión comercial o de tomar decisiones bursátiles mejor que un humano. Y muchas veces, por no decir casi todas, ponemos en mano de estos cerebros electrónicos el control de “juguetes” tan peligrosos como las armas nucleares. No es bueno ser tan terriblemente pesimista como para creer que terminaremos pisoteados por máquinas asesinas al estilo de Terminator, pero hay muchas otras formas en que la robótica, sin un control adecuado, puede convertirse en un problema para la humanidad.

La perdida de empleos, por ejemplo, es una forma sencilla en que la robótica puede generar malestar (y hasta grandes disturbios) entre los humanos. Las ventajas de los robots industriales frente a un trabajador humano son innegables, y difícilmente nadie quiera desactivarlos y volver a ensuciarse las manos armando automóviles u ordenadores como hace 30 años. Pero el grupo de científicos asegura que debemos buscar la forma de que cada trabajador reemplazado por una de estas máquinas reciba la capacitación necesaria como para poder seguir ganándose el pan de cada día.

Puede que los pronósticos de la AAAI puedan parecer pesimistas. Es fácil pensar que estos científicos se están adelantando demasiado, y que seguramente encontraremos la solución a estos problemas cuando se

presenten. Pero la experiencia dice que nunca es bueno subestimar a un problema, y mucho menos cuando quienes alertan sobre los peligros son justamente los especialistas que han desarrollado la tecnología en cuestión. Si queremos no queremos terminar odiando a los robots, deberíamos comenzar ya mismo a planificar su futuro.