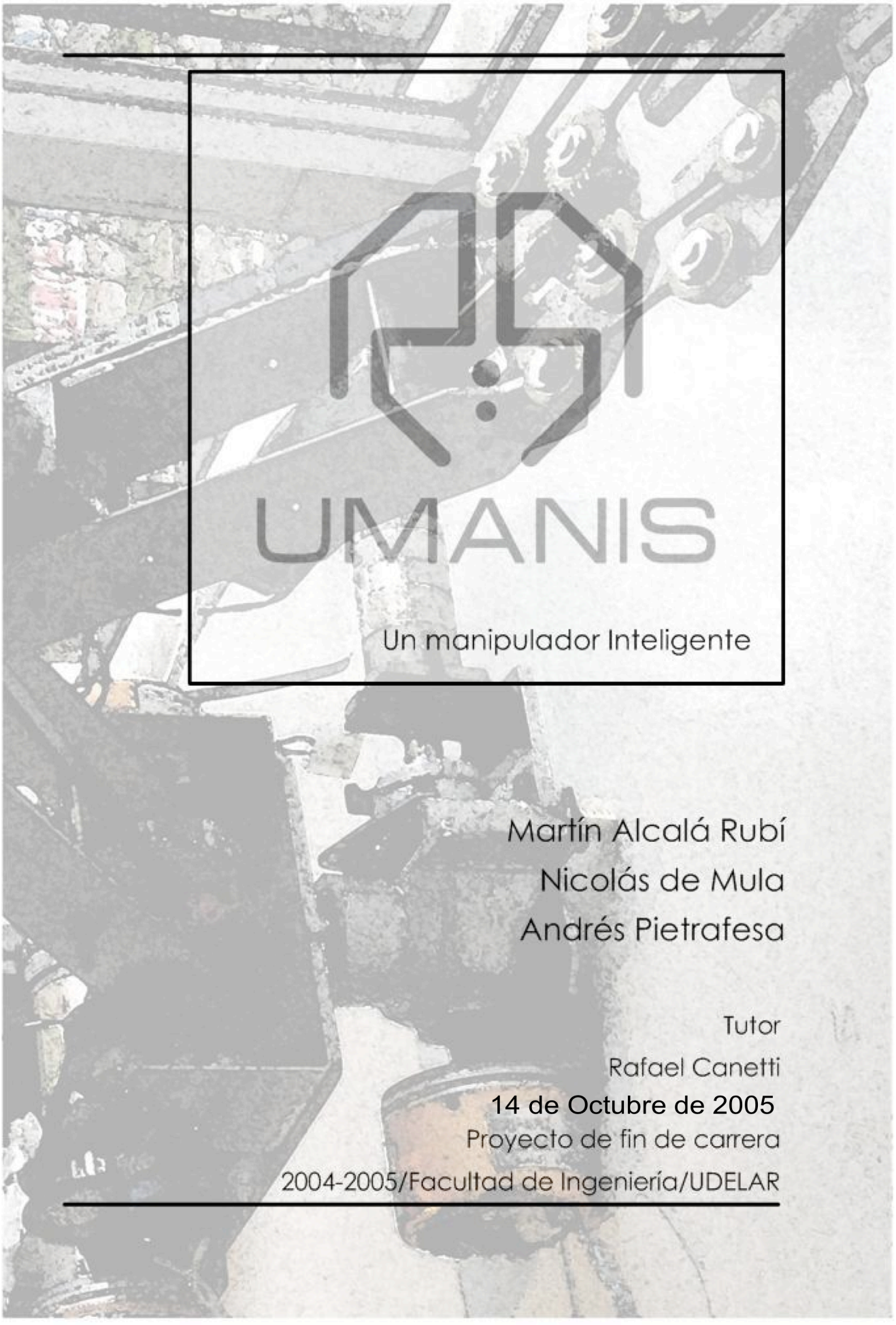


UMANIS

Un manipulador Inteligente

Martín Alcalá Rubí
Nicolás de Mula
Andrés Pietrafesa

Tutor
Rafael Canetti

14 de Octubre de 2005

Proyecto de fin de carrera
2004-2005/Facultad de Ingeniería/UDELAR

Esta documentación fue elaborada en Microsoft Word 2000.
Fueron usadas las fuentes Century Gothic y Sylfaen.

Versión del 27 de Octubre de 2005.

Contenido

Agradecimientos

Prólogo

I Introducción
- I.1 Generalidades
- I.2 ¿Qué es Sulla?
- I.3 Diseño general de Umanis

II Diseño mecánico
- II.1 Introducción
- II.2 Nomenclatura
- II.3 Estructura
- II.4 Transmisiones mecánicas
- II.5 Finales de carrera y posición "home"
- II.6 Motores
- II.7 Encoders
- II.8 El Sistema de Coordenadas de Umanis

III Hardware Eléctrico
- III.1 Generalidades
- III.2 Alimentación
- III.3 Placa de Control
  - III.3.1 Conectores
  - III.3.2 Entorno del Rabbit
  - III.3.3 Interfaz de Programación
  - III.3.4 Uso de Batería
  - III.3.5 Protección
  - III.3.6 Control de Motores
  - III.3.7 Oscilador
  - III.3.7 Control de Pinza
  - III.3.8 Gestión de alarmas
- III.4 Placa de Potencia
  - III.4.1 Descripción General
  - III.4.2 Driver (Puente H)
  - III.4.3 Capacitores de protección
  - III.4.4 Elevador de tensión
  - III.4.5 Censado de corriente y generación de alarma por exceso de consumo.
  - III.4.6 Interruptores de home y final de carrera

IV Control de movimiento
- IV.1 Introducción:
- IV.2 Cinemática
- IV.3 Dinmica
- IV.4 Dinámica del actuador
- IV.5 Controlador

V Desarrollo de Software
- V.1 Generalidades
- V.2 Introducción, Dynamic C
- V.3 Arquitectura de Software

V.3.1 ¿Cómo funciona?
V.4 El planeador de Trayectorias de Umanis
V.4.1 ¿Qué es ? y ¿Qué hace?
V.4.2 Análisis Matemático
V.4.3 ¿Cómo funciona?
V.5 El Sistema Operativo
V.5.1 Introducción
V.5.2 UC/OS-II: El Sistema Operativo elegido para Umanis
V.5.3 Configuración de tareas y diseño de SO
V.6 Descripción de Bibliotecas
V.7 Umanis como Front-End

VI El Proyecto
VI.1 Pruebas realizadas
VI.2 Cosas por hacer
VI.3 Conclusiones

Referencias

Anexos
Contenido del CD
LM629 Precision Motion Controller Datasheet
LMD18200 H-Bridge Datasheet

Agradecimientos

Antes que nada, queremos dedicar esta sección a agradecer a las siguientes personas.

En primer lugar a Dios,

Andrés Alcalá Rubí, Leticia Alcalá Rubí, Luciano Alcalá Rubí, Oscar Alcalá Rubí, Washington Alcalá Rubí (hijo), Washington Alcalá Rubí (padre), Rosalía Amado (Pecha), Raul Arbiza, Daniel Barnech, Marcia Barnech, Leonardo Berro, Gabriel Botti, Darío Buschiazzo, Mercedes Brian, Agustín Caldevilla, Rafael Canetti, Malu Carriquiry, Pablo Cestau, Eduardo Cota, Guillermo Dalla Vechia, Carlos de Mula, Federico de Mula, Ignacio de Mula, Eduardo Da Costa, Bill Gates, Josefina Ferrás, Rafaela Fiorelli, Juan Garat, Ana María García, Gabriel Gómez, Patricia Gramuglia, Rafael Grompone, Eliana Katz, Jean Labrosse, Federico Lecumberry, María Misa, Federico Morales, Ximena Olmedo, Osho, Guillermo Pedezert, Julio Pérez, Nacho Pérez, Hector Pietrafesa, Federico Porto, Agustin Quaglioti, Adrian Rancagno, Lucía Rodríguez, Pablo Rolando, Anika Rotzinger, Antonio Saenz, José Lisandro Sena, Claro Taroco. CCC del Uruguay.

Todos sus aportes, por pequeños que les hayan parecido fueron enormes para nosotros. Gracias por su tiempo, su dedicación y su apoyo!!.

Prólogo

Varios años han pasado desde que el hombre concibió por primera vez la idea de lo automático, de lo autónomo, de la creación de un algo que cobrara vida por si mismo, de una tecnología que permitiera a partir de cierta cantidad de objetos inanimados la creación de una ficción de vida propia, de inteligencia artificial. Este anhelo acompañó a la humanidad desde hace siglos, comenzando con fábulas, historias imaginarias de un futuro posible, cuentos de ciencia ficción. A medida que maduraron los procesos tecnológicos, y que los aventureros del conocimiento (AKA científicos) desarrollaron vorazmente el entendimiento de la física y matemática, estas fábulas comenzaron a cobrar cierto sentido, un sentido cada vez más real y alcanzable; de este proceso surgió entre tantas disciplinas, la robótica.

¿Qué es la robótica?, hay muchas definiciones para este concepto, variadas maneras de expresarlo, más o menos académicamente, podemos recurrir a la etimología de la palabra Robot en el idioma Polaco, pero creo que esa información la podemos encontrar en muchos lados, y tal vez lo que elijamos transmitir en este documento no sea esa visión, preferimos ilustrar sobre lo que la robótica inspira y significa para nosotros, y por qué tres personas luego de estudiar muchos años deciden dedicar su proyecto de grado a este apasionante tema. Para nosotros la robótica significa desafío, significa creatividad y entusiasmo por unir dos disciplinas que tal vez puedan resultar yuxtapuestas para algunas personas: la Ingeniería y el Arte. Comencemos por el término Ingeniería considerado en su sentido más simplista y amplio: dícese de aquella disciplina en que personas utilizan su *ingenio* para resolver un problema tecnológico. Y ¿por qué Arte?, porque en el mundo de la robótica, todo es incierto, difícilmente existen verdades absolutas o esquemas rígidos para construir un sistema. El diseño y la implementación están a merced de la creatividad de los diseñadores.

De modo que de eso pretende tomar partida este Proyecto de fin de Carrera, y esta documentación. Pretende narrar la creación que surge de tres personas que decidieron culminar su Carrera soñando, que decidieron que su obra última como *estudiantes de grado* (porque el verdadero investigador es amante del conocimiento, lo que lo convierte en el estudiante eterno), sería una aventura hacia lo desconocido, atravesando las fronteras del conocimiento, y lo creado.

I. Introducción

I.1 Generalidades

Umanis es un Proyecto de fin de carrera de la Facultad de Ingeniería de la UDELAR, el cual consiste en desarrollar un manipulador robótico que es integrado a una robota móvil llamada Sulla, desarrollada también en el Instituto de Ingeniería Eléctrica anteriormente por otro grupo de estudiantes.

La iniciativa de crear este manipulador robótico fue propuesto a nosotros por el Prof. Ing. Rafael Canetti, con la finalidad de continuar la creación de una plataforma de estudio práctico de robótica e inteligencia artificial. El espíritu es esta iniciativa es generar los medios para desarrollar el estudio e investigación de estas áreas aquí en nuestro país, disponiendo de un sistema real, que permita realizar pruebas empíricas y no basarse solamente en simulaciones informáticas.

Como punto de partida se nos dieron las siguientes especificaciones para el manipulador robótico:

- Debe tener un alcance de 40 cm.
- Debe ser capaz de levantar una carga de 500 gr. a 40 cm. de distancia.
- Debe poseer una precisión de posicionamiento en el espacio comprendida en un cubo de 3 mm. la lado.
- Debe ser capaz de manipular objetos.
- Debe ser desarrollado para cumplir objetivos generales, no resolver problemas particulares.
- Debe estar integrado con la robota móvil Sulla.

Debido a la última especificación, nos parece importante dedicar una breve sección a introducir a Sulla.

I.2 ¿Qué es Sulla?

Sulla [25] es una robota móvil con el aspecto que se ve en la siguiente figura

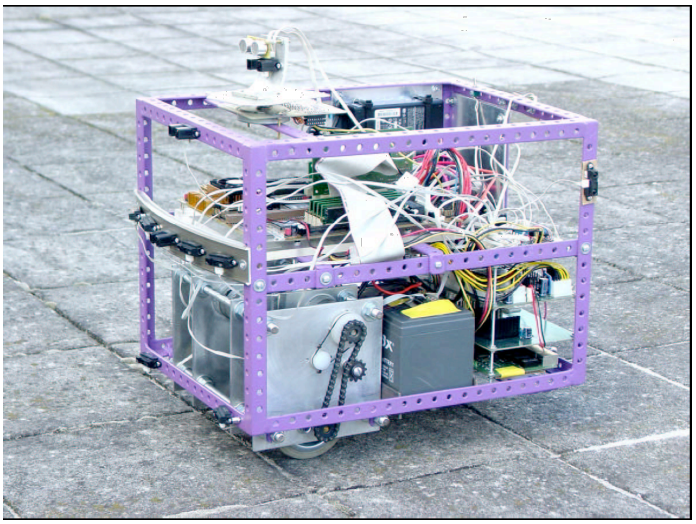


Fig. I.1 – Sulla, la robota móvil [25]

Sus dimensiones son, aproximadamente, 45 cm. de largo, 32cm de ancho y 40 cm. de alto. En su cuerpo está incluido el sistema móvil, las baterías para alimentación autónoma y su sistema de carga, sensores, 1 microcontrolador, 1 PC y el espacio para

agregar otros dispositivos como ser cámaras o en nuestro caso específico, un manipulador.

Sulla tiene 3 ruedas, una gira libremente y las otras dos se controlan en forma independiente. Es capaz de girar o moverse en trayectorias curvas moviendo con distintas velocidades las ruedas de cada lado.

La información del entorno la obtiene de 17 sensores infrarrojos de distancia, 2 sensores ultrasónicas de distancia, un sensor de luz, sensores de temperatura y algunos sensores de contacto.

El hardware computacional se realiza en torno de una red Ethernet estándar. Esto permite agregar nuevos módulos de procesamiento que tengan esta interfaz. En la primera etapa de Sulla, la red es de dos nodos: un microcontrolador (Rabbit 3000) para manejo de eventos en tiempo real y un PC estándar para el procesamiento de alto nivel.

La razón para hacer esta división es poder poner toda la programación de tiempo real fuera del PC, y de esta forma poder usar en el PC un Sistema Operativo como GNU/Linux [24], que no es de tiempo real, pero facilita la reutilización de código y disponibilidad de herramientas a la vez que reduce los conocimientos necesarios para programar.

El esquema de organización de la programación global, es en módulos software que funcionan en forma independiente, donde toda interacción entre ellos ocurre por mensajes UDP [5]. Este modelo de comunicación, junto al protocolo establecido para la comunicación, toma el nombre de Sinapsis [25].

Sinapsis permite la comunicación en forma homogénea entre módulos ejecutados en distintas computadoras o en la misma, independiente del Sistema Operativo o del lenguaje de programación usado. Cualquier lenguaje de programación o ambiente de trabajo que sea capaz de enviar y recibir mensajes UDP podrá ser usado en Sinapsis.

Sinapsis está orientado a formar una estructura de módulos que ínter operan para coordinar acciones según información de los sensores, información del pasado y objetivos variados con distintas prioridades. El protocolo permite identificar y referirse a módulos específicos, establecer comunicación para pedir información que otro módulo tiene disponible o dar órdenes.

Por información detallada sobre esta plataforma, referirse a la documentación del Proyecto de fin de carrera Sulla, una robota móvil.

I.3 Diseño general de Umanis

Ahora que vimos un pantallazo general de Sulla, la plataforma sobre la cual está montada Umanis, estamos en condiciones de dar una introducción al diseño general de Umanis.

Como una imagen, en ocasiones, vale mas que mil palabras, comencemos por mostrar el hardware mecánico de Umanis.

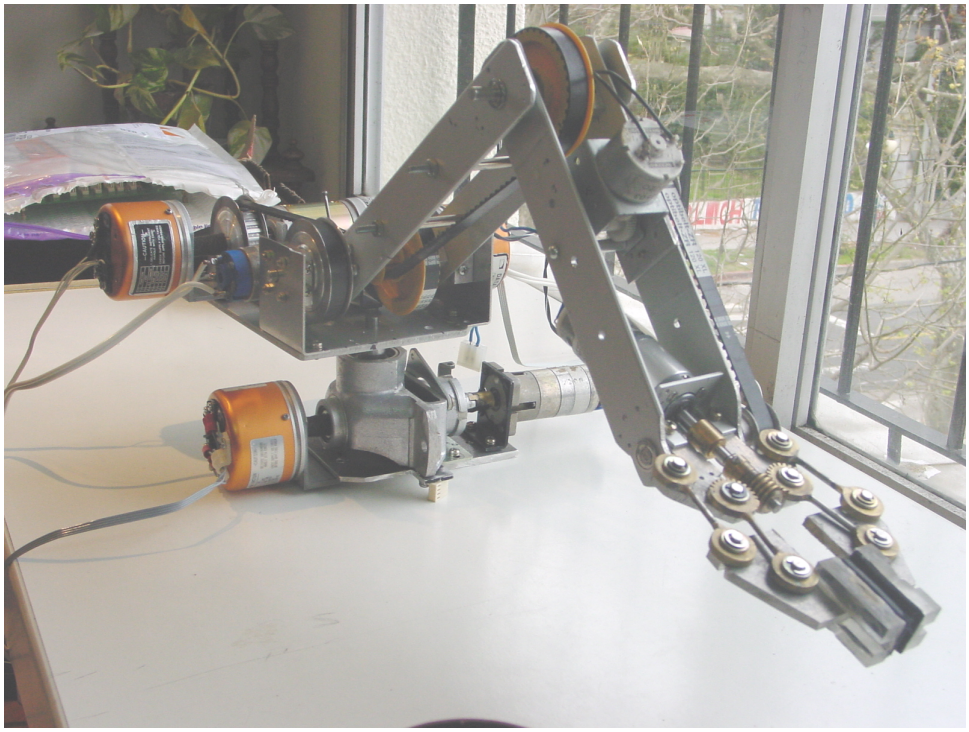


Fig. I.2 – Manipulador mecánico de Umanis

Como se puede apreciar, es un manipulador robótico con un total de 5 grados de libertad, o como se conoce en la jerga de la robótica, un manipulador de 5 DOF (acrónimo de *Degrees of Freedom*). Estos son:

- "Cintura" (o giro de hombro)
- "Hombro"
- "Codo"
- "Muñeca"
- Pinza.

Esta nomenclatura de los grados de libertad será desarrollada con la precisión adecuada en los capítulos de diseño mecánico. Todos estos grados de libertad o articulaciones, a excepción de la pinza, son de movimiento giratorio. En caso de la pinza, esta puede ser cerrada o abierta realizando un control de toque. Todos los actuadores (motores) de las articulaciones son motores eléctricos de corriente continua.

Umanis también está compuesto por dos placas de Electrónica, una para manejar las señales de potencia que serán inyectadas en los motores, y otra que posee la "inteligencia" manipulador, es decir microprocesadores dedicados, además de electrónica de manejo de alarmas. A esta placa la llamaremos *placa de Electrónica de señal*.

Como vimos en la sección referida a Sulla, los módulos que sean agregados a la robota, deberán ser capaces de comunicarse vía paquetes UDP, utilizando un protocolo llamado Sinapsis, por lo que Umanis posee un microcontrolador Rabbit 3000 que dentro de otras cosas, es capaz de comunicarse a través de Ethernet en una red de datos TCP/IP.

Como veremos detalladamente en secciones posteriores, el control de los distintos motores del manipulador es efectuado por 3 microcontroladores y un sistema electrónico de control de torque para la Pinza. Estos 3 microcontroladores son:

- Un Rabbit 3000 (*RCM 3200 RabbitCore*, manufacturado por *Z·World*)
- Dos LM629 (manufacturado por *National Semiconductor*)

Siendo el microprocesador Rabbit 3000 el principal actor, entre ellos son capaces de entre otras cosas: controlar los motores, generar trayectorias para el manipulador conjunto, mantener la sincronicidad entre el comando de motores y ejecutar directivas que arriban a la interfaz Ethernet.

El software de Umanis está pensado para ejecutar y encolar comandos provenientes de una entidad externa a través del protocolo Sinapsis. En este caso, dicha entidad es Sulla. Esta funcionalidad permitiría que Umanis fuera comandado utilizando como medio de comunicación una red de datos estándar, en particular permitiría que el manipulador fuera comandado a través de *Internet*.

Con este concepto, y como se desarrollará en un capítulo posterior, podemos ver a Umanis como un Front-End de aplicaciones mecánicas, es decir, que dado un comando expresado en la sintaxis del protocolo Sinapsis (desarrollado en Sulla), Umanis será capaz de desarrollar una aplicación mecánica.

Culminamos esta sección con una imagen de Umanis montado sobre Sulla.

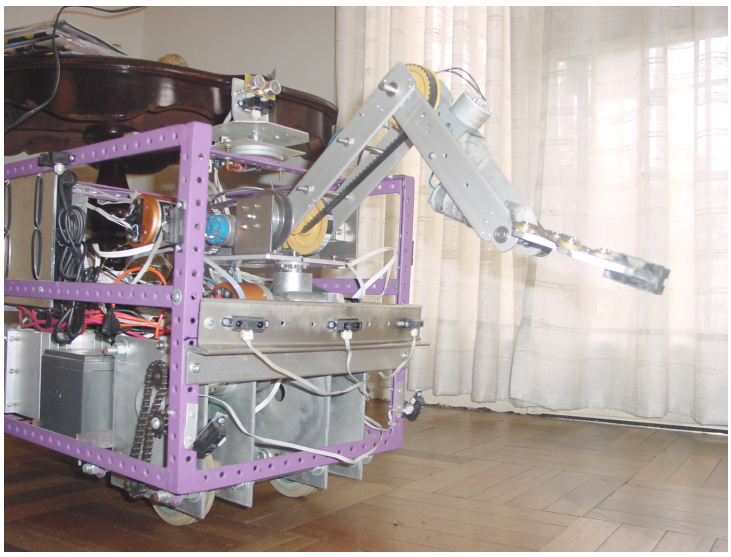


Fig. I.4 – Umanis & Sulla

II. Diseño mecánico

II.1 Introducción

A continuación se detallan la partes mecánicas y los aspectos constructivos de nuestro proyecto, se intenta dejar claro como está implementada cada una de las articulaciones (5 en total), qué elementos intervienen y como interactúan. También se incluye en esta sección una caracterización y modelado de los motores utilizados así como de los encoders ópticos (dispositivos de realimentación).

En esta área es tal vez donde encontramos mayores inconvenientes y problemas, por un lado es difícil encontrar en plaza los elementos necesarios y por otro al comprarlos en el exterior los precios son mayores a lo que hay que sumarle el envió más los impuestos de importación y además generalmente los trámites de compra son largos y poco predecibles.

En algunos casos la forma que encontramos para resolver estos temas fue comprar componentes usados, sobre todo en el caso de los motores y lo encoders ópticos. Un lugar que fue una fuente importante de partes fue un desarmadero donde las cosas eran vendidas prácticamente como chatarra.

Al diseñar y construir el manipulador nunca se perdieron de vista los objetivos y restricciones, sobretodo en cuanto a la fuerza y precisión necesarias para el brazo, se buscó utilizar materiales livianos y fuertes (como por ejemplo el aluminio), los encoders se buscaron para cumplir con la precisión requerida y los motores para soportar la carga máxima exigida y que además su consumo de corriente fuera razonable y sus dimensiones reducidas.

II.2 Nomenclatura

Manipulador es todo el conjunto de partes mecánicas que conforman nuestro proyecto.
Para hacer referencia a las distintas articulaciones se utilizan los nombres correspondientes de las partes análogas en el cuerpo humano, esto es:

- La **cintura** corresponde a un eje vertical, solidario a la robota, sobre el cual gira el manipulador completo.
- El **brazo** y el **antebrazo** son los segmentos que van desde el hombro al codo y desde el codo a la muñeca respectivamente.
- El **hombro** es el eje horizontal, solidario a la base giratoria, que sube y baja el brazo.
- El **codo** es el eje horizontal, solidario al brazo, que mueve el antebrazo respecto al brazo.
- La **muñeca**, eje horizontal solidario al antebrazo que mueve la mano respecto al antebrazo
- Y la **mano** corresponde a la pinza o elemento actuador.

II.3 Estructura

A continuación se describe la estructura general del manipulador, como están distribuidos y conectados los distintos elementos y se intenta dejar claro las características básicas del movimiento. En la figura se ve un dibujo del brazo completo con todas las articulaciones.

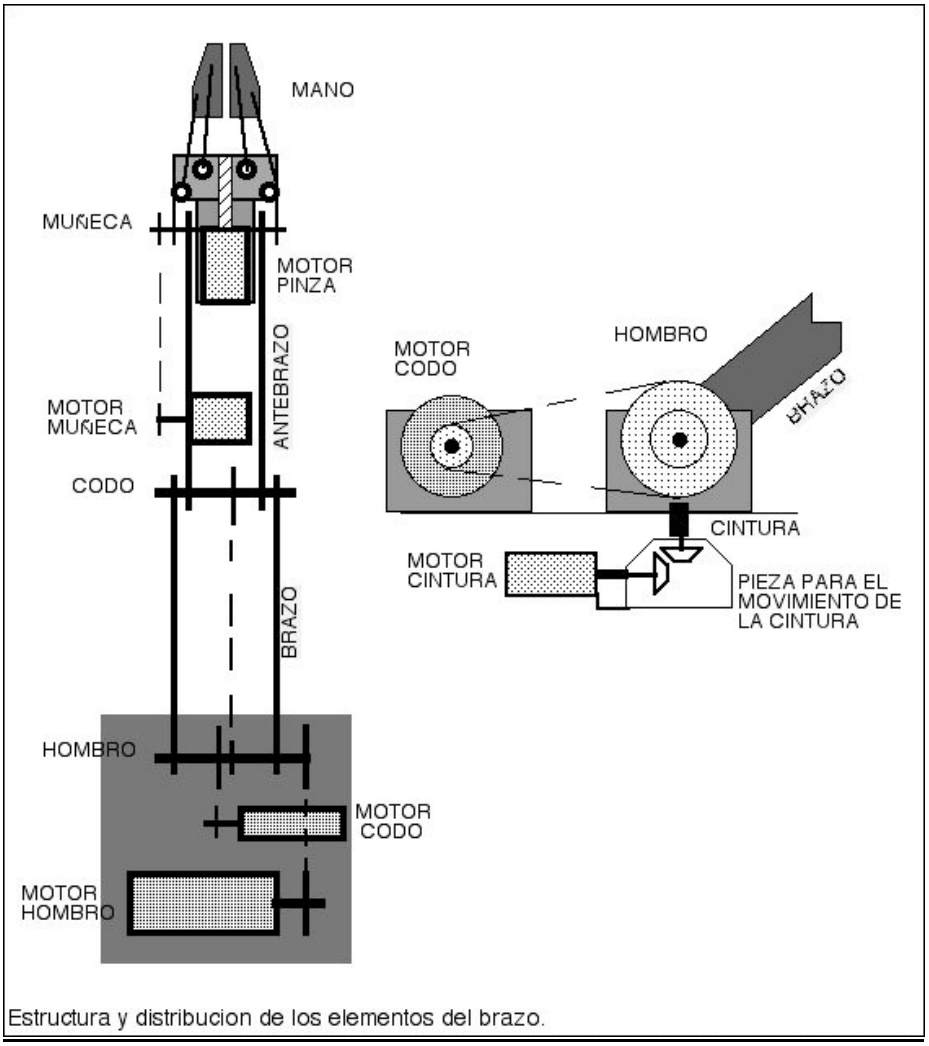


Fig. II.1 – Esquemático de la estructura general del manipulador

La base que soporta el manipulador es de aluminio de 4.5 mm de espesor, 16 cm de largo y 12 cm. de ancho. En esta base están fijos los motores del hombro y el codo (con sus respectivos encoders) y también el eje correspondiente al hombro.
Esta base se inserta (mediante un eje ranurado) en una pieza modular fija a la robota, cuya función se explica mas adelante.

Tanto el brazo como el antebrazo están compuestos por dos planchuelas de aluminio de 21 cm. de largo (18 cm. entre eje y eje), 5 cm. de ancho y 3 mm de espesor. Las planchuelas están ubicadas en forma paralela y distan entre si 5.8 cm. para el caso de brazo y 4.5 cm. para el antebrazo.

Los ejes del hombro, codo y muñeca son de 6.3 mm de diámetro y el de la cintura tiene un diámetro de 8 mm.

El eje que corresponde a la cintura está fijo a la base giratoria (y perpendicular a la misma) para luego introducirse en la pieza modular que es la que le da movimiento.
El eje del hombro está montado sobre rulemanes solidarios a la base (y sostenidos por planchuelas en forma de L) y está fijo a las planchuelas que conforman el brazo.

El eje del codo está sobre rulemanes solidarios al brazo y está fijo a las planchuelas del antebrazo.

El eje de la muñeca gira sobre rulemanes en las planchuelas del antebrazo y está fijo a la pinza y el motor de la misma.

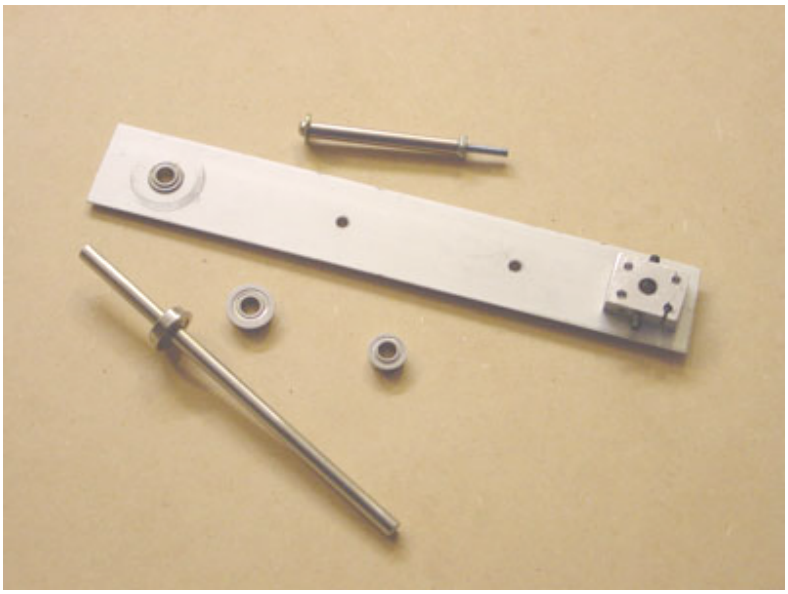


Fig. II.2 – Planchuelas de brazo y antebrazo, ejes y rulemanes

II.4 Transmisiones mecánicas

Para todas las articulaciones se utilizaron transmisiones mecánicas para transferir la fuerza generada por los motores hacia el eje de giro de la articulación correspondiente, estas se implementaron de diferentes maneras las cuales pasamos a describir [22].

- **Cintura:** El giro del manipulador se logra mediante una pieza modular donde va montada la base del mismo y que mediante engranajes cónicos se logra tener dos ejes solidarios perpendiculares entre si, el eje vertical corresponde a la cintura y en el eje horizontal se tiene el motor encargado del giro.
- **Hombro:** La transmisión entre el motor del hombro y el eje se implementa con una correa dentada del tipo XL, los dos engranajes son iguales entre si (no hay reducción en esta articulación), su cantidad de dientes es 36y son de aluminio.
- **Codo:** Para concentrar el peso del brazo en la base se ubicó el motor encargado de mover el codo en la misma, la transmisión desde este motor hacia el eje del codo propiamente dicho tiene dos etapas, los engranajes son todos de nylon y las correas son del tipo XL

Cada etapa tiene una reducción 1:3. en el motor hay un engranaje de 12 dientes, desde ahí con una primera correa se llega hasta un engranaje de 36 dientes ubicado en el eje del hombro pero que gira libremente sobre él. Solidario a este engranaje hay otro de 12 dientes que mediante una segunda correa llega hasta un cuarto engranaje de 36 dientes ubicado en, y fijo al, eje del codo. De esta manera para esta articulación se tiene una reducción de 1:9.

Para un mayor comprensión de esta transmisión se muestra la siguiente figura.

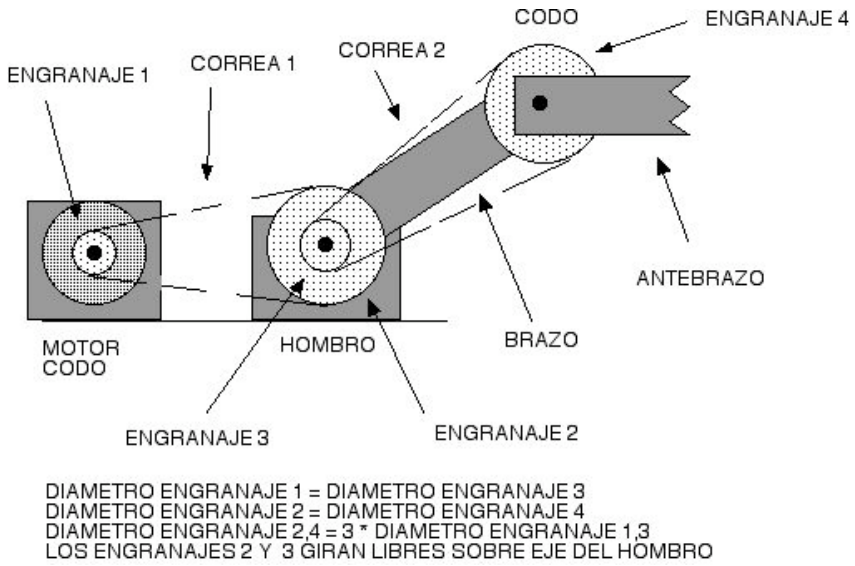


Figura II.3 – Esquemático de transmisiones

- **Muñeca:** El motor encargado de mover la muñeca se ubicó sobre el antebrazo (fijo en el espacio comprendido entre las planchuelas), en un punto intermedio entre el codo y la muñeca (a 13 cm. de la muñeca). La transmisión es mediante una correa dentada del tipo XL y los engranajes son iguales entre si de 12 dientes y el material es aluminio.

- **Mano:** Para el cierre y la apertura de la pinza se utiliza un tornillo sinfín, fijo al eje del motor (ubicado entre las dos planchuelas del antebrazo), que acciona el mecanismo.

En la figura se pueden ver los elementos de las transmisiones, se muestran una correa de ejemplo y los engranajes, tanto de nylon como de aluminio.

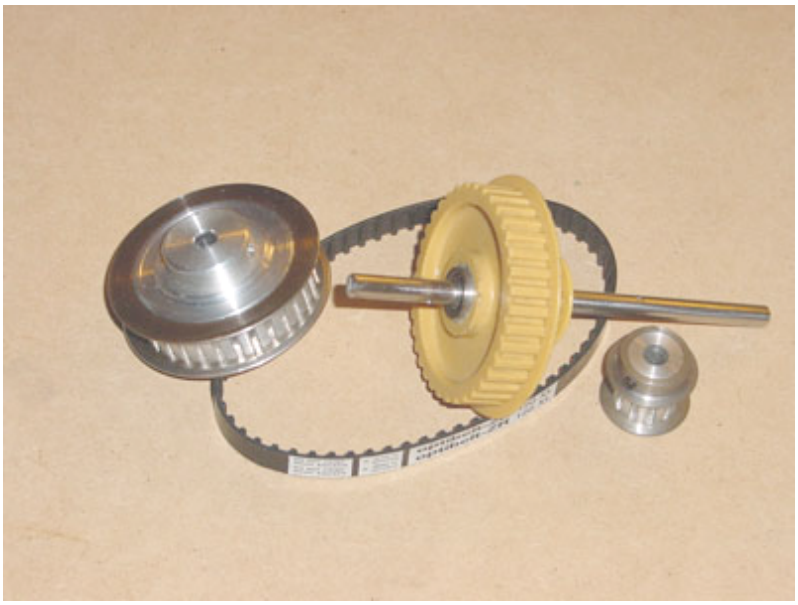


Fig. II.4 – Elementos de transmisión, engranajes, ejes y correas

Mecanismo de la pinza (mano):
Para esta articulación se buscó que la apertura y cierre se realice en forma paralela. Para esto se armó un mecanismo como el que se ve en la figura (Fig. II.5).
Todos los discos que se pueden apreciar giran libremente sobre sus centros. Los discos dentados son girados simultáneamente por el tornillo sinfín (fijo al eje del motor) accionando los paralelogramos correspondientes a cada "dedo" generando el efecto deseado. La apertura máxima posible de la pinza es de 6 cm.

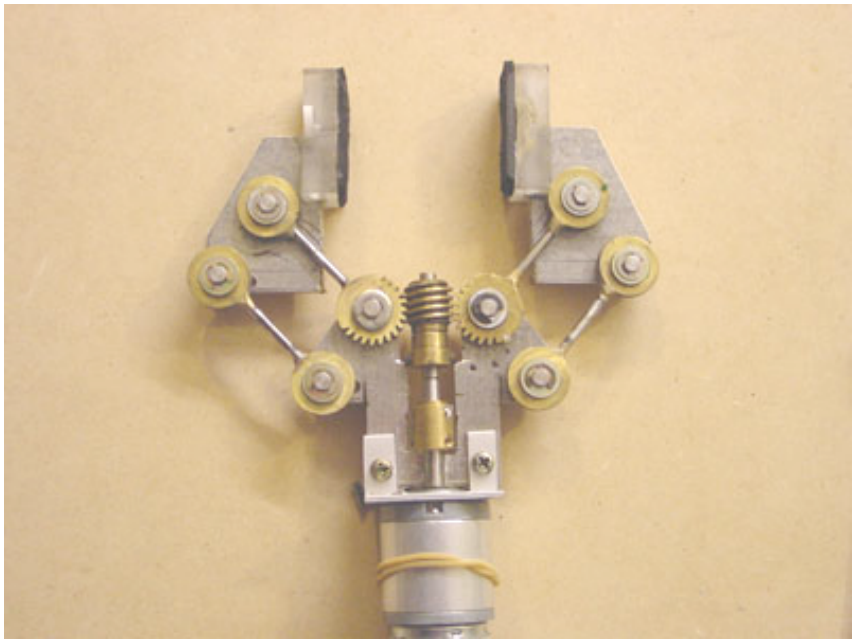


Fig. II.5 – Mano o pinza

Para una mayor comprensión de las explicaciones anteriores agregamos la siguiente foto donde se puede observar la parte mecánica completa del manipulador.

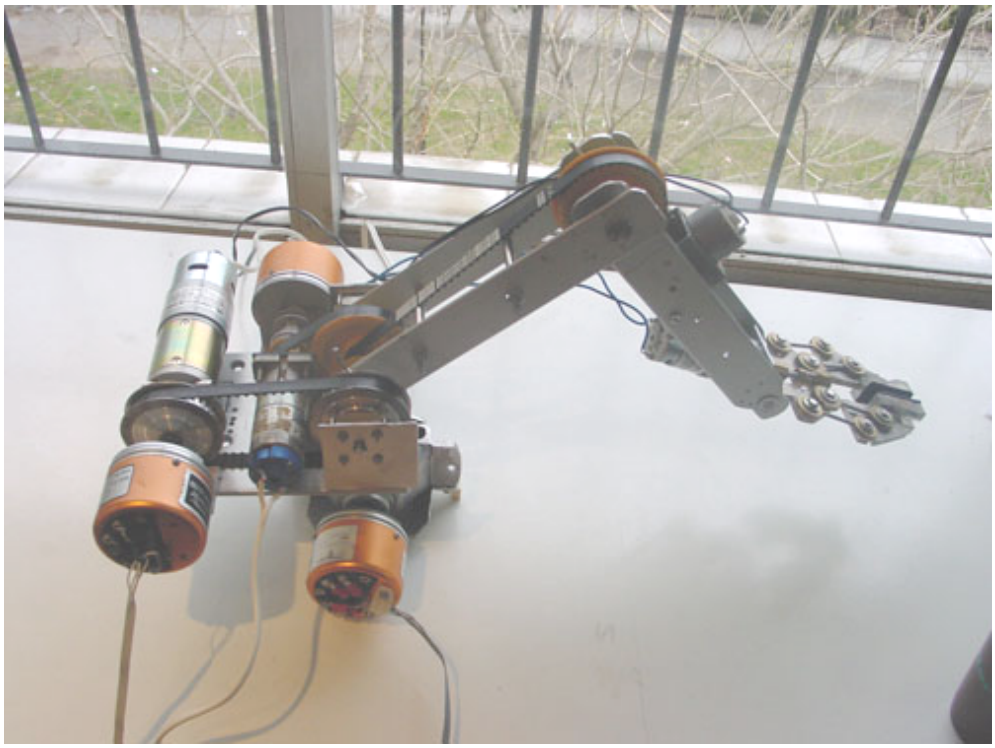


Fig. II.6 – Manipulador completo

II.5 Finales de carrera y posición "home"

En todas las articulaciones se incluyeron interruptores para detectar cuando se llega a un punto límite a partir del cual se podría producir un ruptura de alguna parte del brazo o la robota (finales de carrera).

Estos interruptores activan un sistema electrónico (ubicado en la placa de control) el cual al producirse un evento límite en cualquier articulación inmediatamente deshabilita todos los motores para luego localizar donde se produjo la situación indeseada.

Estos interruptores son botones o llaves con un pulsador, del tipo que se pueden encontrar en los Mouse de computadora, se ubicaron para que una parte mecánica accione el pulsador cuando se llega a la situación límite.

Estos finales de carrera limitan el movimiento del brazo de la siguiente manera.
La cintura tiene un ángulo de giro total de 120 grados, este movimiento está limitado por la propia robota.

El motor del hombro tiene un ángulo de giro de 140 grados (85 hacia arriba y 55 hacia abajo) y también está limitado por la robota en la parte superior y por la propia base del manipulador en la parte inferior

El ángulo de giro del codo está limitado por el giro del antebrazo que para cierto ángulo colapsa con el brazo y es de 210 grados.
Para la muñeca se tienen 190 grados de giro completo y es limitado por la intersección de la mano con el antebrazo.

La mano tiene una apertura máxima de 6 centímetros y está limitada por la integridad del mecanismo de la misma.

De una manera similar se implementa un sistema de detección de la posición de reposo o "home" del manipulador, se utilizan interruptores idénticos a los utilizados para los finales de carrera y se ubican para que el brazo tenga como posición por defecto una que no interfiera con el movimiento de la robota ni con sus sensores, además se busca que en esta posición el torque efectuado por cada uno de los motores sea el mínimo posible.

Estos interruptores también se utilizan para ubicar una posición conocida en caso de que por el lado del procesador se pierda rastro de la ubicación de las distintas articulaciones del manipulador.
De esta manera se tiene la siguiente posición de Home:

- Cintura: En un punto medio entre los dos finales de carrera, o sea, el manipulador queda orientado hacia adelante de la robota.
- Hombro: El brazo queda orientado hacia arriba con 70 grados sobre la horizontal.
- Codo: El antebrazo queda orientado hacia abajo formando un ángulo de 45 grados con el brazo
- Muñeca: La muñeca queda en forma vertical
- Mano: La posición de reposo para la pinza es cerrada.

De esta manera la posición de reposo del manipulador queda con el brazo contraído con el codo hacia arriba y el antebrazo y la mano orientados hacia el piso, sin interferir con el cinturón de sensores de Sulla.

II.6 Motores

En todos los casos para las distintas articulaciones se utilizan motores de corriente continua de voltaje nominal 12 V, salvo el motor encargado de mover la muñeca que es de 24 V.

Se consideró la posibilidad de utilizar motores paso a paso para alguna de las articulaciones, la ventaja de este tipo de motores es que es prácticamente innecesario tener un lazo de realimentación para controlarlos, alcanza con dar una cierta secuencia de pulsos en sus terminales y su eje girara un numero determinado y fijo de grados, de esta manera se tiene certeza de la posición del eje.

Por otro lado la desventaja y la razón por la cual no fueron utilizados por nosotros es que no son capaces de generar grandes torques, para lograr pares del orden de los necesarios el tamaño del motor crece considerablemente por lo que se opto por dejarlos de lado.

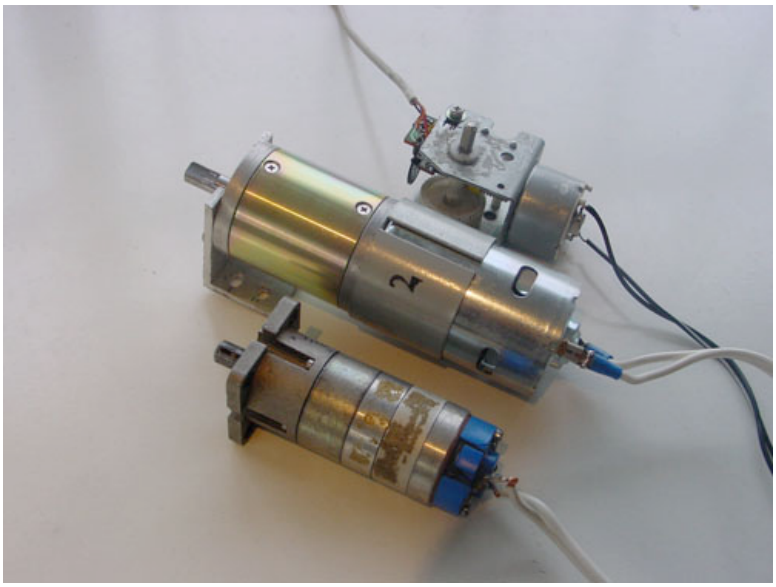


Fig. II.7 – Tres motores del manipulador. Desde abajo hacia arriba: codo, hombro y muñeca.

Todos los motores tienen reducción incorporada, sólo para el motor del hombro y la pinza tenemos información de su valor (estos son de 1/624 y 1/200 respectivamente) dado que los demás motores fueron comprados usados y por lo tanto sin ningún dato sobre el mismo (excepto el voltaje nominal).

Para el motor de la muñeca, al tener expuesta su reducción, se pudo estimar la misma (mediante la utilización de dos encoders, uno antes y otro después de la reducción), el valor encontrado fue de 325.

Para el cálculo del torque que debe realizar cada uno de los motores los dos elementos que más repercuten son la mano (incluyendo el motor y el mecanismo de la misma) y la propia masa a ser levantada, también, aunque en mucho menor medida influye el motor de la muñeca que esta posicionado en el antebrazo.

Los demás elementos, o bien están ubicados en la base (por lo que no aportan al torque) o son muy livianos, como ser los engranajes, las correas dentadas, los rulemanes y los ejes, de todas maneras se tendrán en cuenta con un factor de corrección.

En el siguiente análisis se considera el peor caso de exigencia hacia los motores que es cuando el manipulador está totalmente extendido en forma horizontal, recordamos que la extensión del brazo desde el hombro hasta la mano es 46 cm.

Claramente el motor mas exigido en cuanto a torque es el motor encargado del movimiento del hombro, el cálculo del torque es el siguiente:

- Torque aportado por la carga: La carga máxima a ser levantada es de 500g y se encuentra a 46 cm. del hombro, entonces el torque en este caso es de 2.254 Nm o sea 23 Kg.cm.
- Torque aportado por la mano: El peso de la mano completa es de 450g y su centro de masa se encuentra sobre el eje de la muñeca que dista 36 cm. del hombro. El torque aportado en este caso es 1.588 Nm o sea 16.2 Kg.cm.

El motor de la muñeca pesa 120g y su centro de masa se encuentra a 23 cm. del hombro (con el manipulador extendido) o sea que el torque es: 0.27 Nm, que equivale a 2.76 Kg.cm.

Entonces el torque máximo total exigido a este motor es de:

$$(23 + 16.2 + 2.76)\text{Kg.cm} = 42\ \text{Kg.cm}.$$

Para esta articulación se utilizó un motor modelo *"IG-42GM 01"* con reductor 1/624 fabricado por *Shayang Ye Industrial Co*. El punto de trabajo nominal de estos motores es de 30 Kg.cm a la velocidad nominal de 10.8 rpm, o sea que para esta articulación se tiene una restricción en la velocidad de trabajo en el caso de mayor torque, como se puede ver en la siguiente gráfica torque-velocidad el valor máximo de velocidad para este punto de trabajo es de 9.5 rpm. Esta gráfica se obtuvo a partir de puntos de trabajo adquiridos con carga conocida y relevando la velocidad del eje.

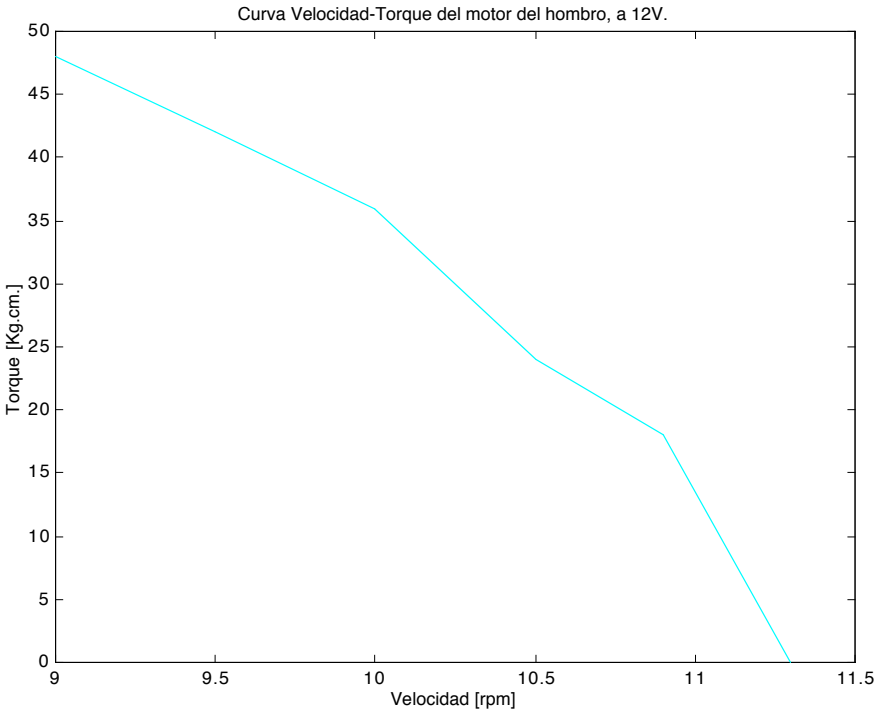


Fig. II.8 – Curva Velocidad – Torque del motor del hombro a 12V.

El segundo motor en cuanto a la exigencia de torque es el del codo, para esta articulación el cálculo es el siguiente.

El torque aportado por la carga es 14 Kg.cm (500g a 28 cm.), el aportado por la mano es 8.1 Kg.cm (450 g a 18 cm.) y el del motor de la muñeca es de 0.6 Kg.cm (120 g a 5 cm.); por lo que el torque total es de 14 + 8.1 + 0.6 = 22.7 Kg.cm.

A este motor hay que sumarle el hecho de que entre el motor y el eje del codo hay una reducción de 1:9, por lo que el torque exigido al motor se debe dividir entre 9 (esto no es del todo correcto pero sirve como aproximación, como se verá mas adelante este motor no se ve comprometido o sea que un posible error en esta aproximación no repercute en el funcionamiento). Tomando en cuenta este factor se tiene que el motor debe ser capaz de entregar un par mecánico de 2.52 Kg.cm.

Para este motor no se tenía la hoja de datos (el motor fue comprado en un desarmadero), en la "chapa" del motor se lee: "Powertron". Se realizaron los mismos ensayos que para el motor del hombro y se obtuvo la siguiente gráfica torque-velocidad. Como se puede ver para el rango de velocidades utilizadas por nosotros este motor no presenta problemas para entregar la fuerza necesaria, máxime teniendo en cuenta que la grafica es para un voltaje reducido de 5 V, cuando el voltaje nominal del motor es de 12 V.

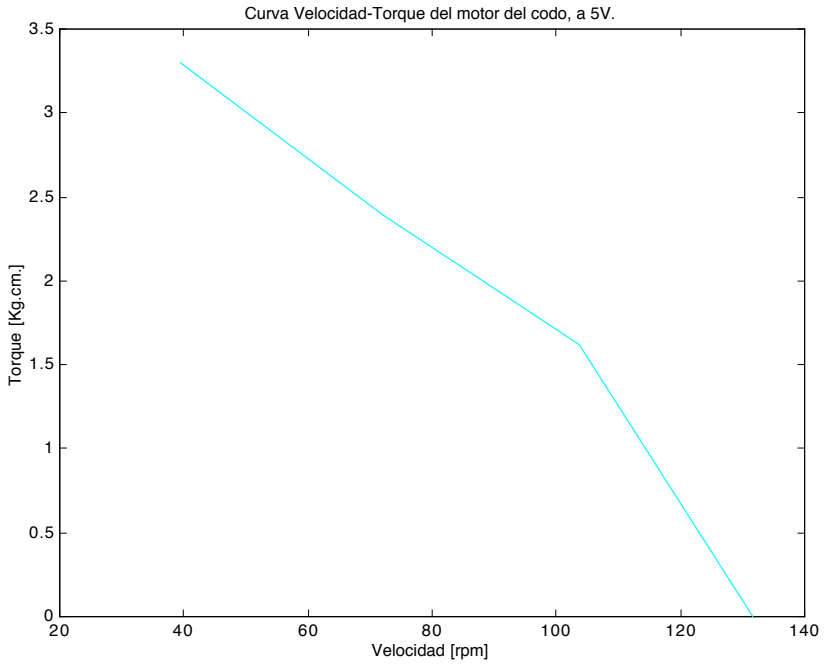


Fig. II.9 – Curva Velocidad – Torque del motor del codo a 5V.

Para el motor de la muñeca el único elemento que interviene en el cálculo del torque es la masa (el centro de masa de la pinza se encuentra sobre el eje de la muñeca), entonces el torque es 5 Kg. cm. (500 g a 10 cm.), este es un motor pequeño pero con una gran reducción, el valor calculado para la misma es de 325 (este motor fue comprado usado por lo que no tenemos datos sobre el mismo), por lo que es capaz de entregar la fuerza suficiente, recordamos que este motor es alimentado con un máximo de 20 V, su voltaje nominal es de 24 V.

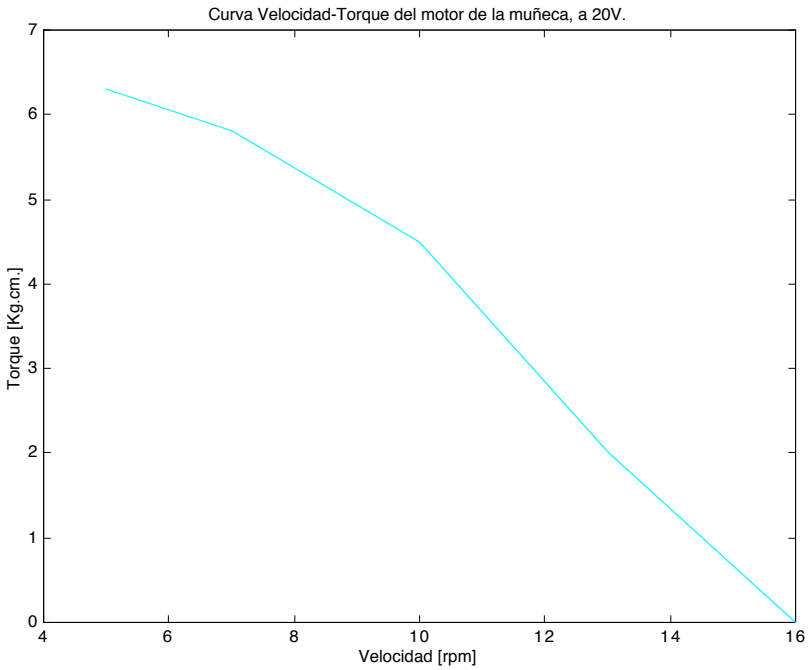


Fig. II.10 – Curva Velocidad-Torque del motor de la muñeca a 20V.

En todos los casos vemos que aun con un factor de corrección de, por ejemplo, el 10% sobre el cálculo del torque debido a la influencia de los elementos mas livianos, los motores no presentan mayor dificultad para realizar la fuerza necesaria a las velocidades de trabajo (en todos los casos).

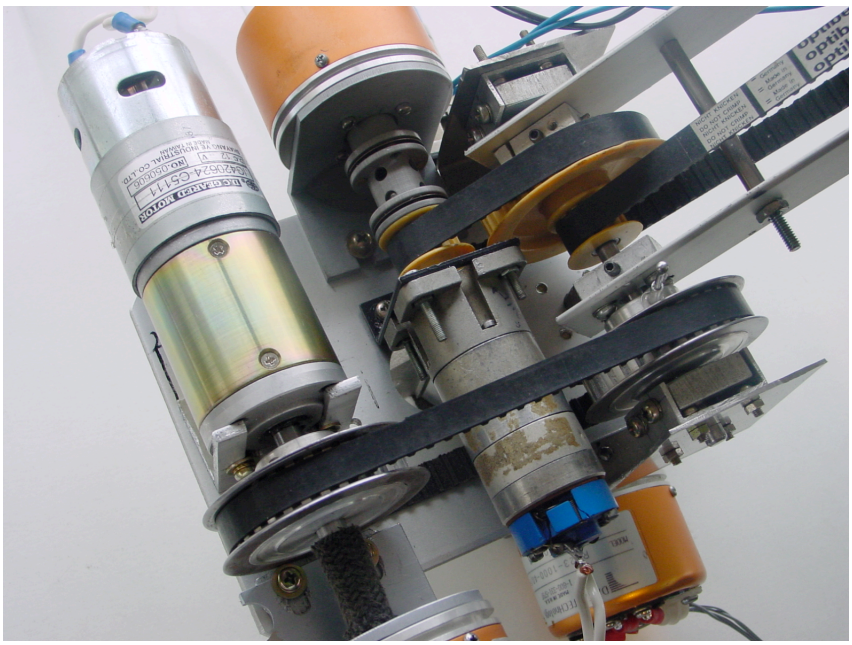


Fig. II.11 – Motores, encoders y transmisiones.

II.7 Encoders

Como dispositivos de realimentación para tener información sobre la posición de los ejes se utilizan encoders ópticos incrementales (*shaft encoders*).

El principio de funcionamiento de estos es el siguiente: en un encoder simple se tiene un disco ranurado solidario al eje de giro y un par emisor/receptor de luz, de tal manera que el disco gire en el espacio intermedio de los dos. A medida que el disco gira las ranuras producen que el receptor reciba alternadamente luz y sombra generando así a su salida un onda cuadrada.

Suponiendo que el eje gira siempre en el mismo sentido, se puede saber, contando la cantidad de pulsos a la salida y conociendo la cantidad de ranuras del disco, la cantidad de grados que giró el eje.

El tipo de encoders utilizado presenta una variación que aporta de dos maneras al problema de control. En lugar de existir un solo par emisor/receptor se introduce un segundo par ubicado exactamente a una distancia del primero la mitad de la distancia entre dos ranuras del disco.

De esta manera a medida que el disco gira se tienen dos señales cuadradas con un desfasaje entre sí, este desfasaje permite (como primer aporte) determinar el sentido de giro del eje, de esta manera:

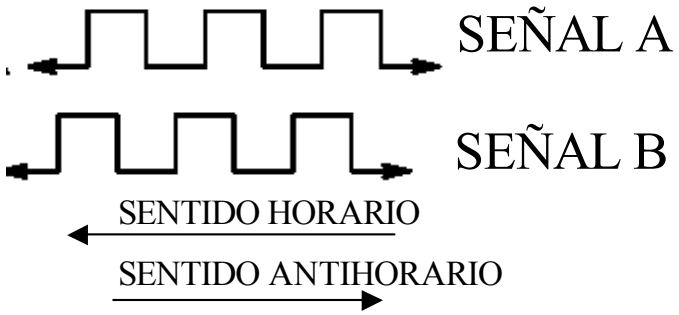


Fig II.12 – Señales en cuadratura

Analizando los cambios de estados del par AB se tiene el sentido de giro, es decir, suponiendo que se está en el estado AB = 11 y se pasa al estado 01 entonces se esta girando en sentido horario, si por el contrario se pasa al estado 10 se esta girando en sentido antihorario.

Como segundo aporte, al tener estas señales en cuadratura, se incrementa la resolución del dispositivo, por cada ranura del disco se tienen cuatro estados de la salida, si por ejemplo se tiene un disco con 10 ranuras se tendrán 40 estados por revolución del disco lo que equivale a una precisión angular de 9 grados, se incrementa la resolución en factor de 2 con respecto a si se utilizara solo la señal A o B (en este caso se tendrían 2 cambios de estado por cada ranura , flanco de subida y de bajada).

Los encoders que utilizamos en Umanis son los siguientes:

- Los encoders de la cintura y el hombro son de la marca "*DATA TECHnology, Inc.*", el producto es "*RS23 Encoder*". Tienen 1000 ranuras, y están acoplados solidariamente al eje del motor y además estas dos articulaciones no tienen

reducción, o sea que se tiene un total de 4000 estados por revolución de la articulación en ambos casos.
Estos encoders permiten entonces una precisión de 360/4000 = 0.09 grados. Que equivale en el extremo del manipulador a un arco de 0.72 mm.

- El encoder del codo es un encoder de la misma marca pero de 200 ranuras, este encoder está fijo al eje del motor, entonces, teniendo en cuenta que el codo presenta una reducción de 1:9, la situación es equivalente a tener un encoder de 1800 ranuras directamente en la articulación, o sea 7200 estados por revolución.
  Con estos valores se tiene una precisión de 360/7200 = 0.05 grados. Que equivale en el extremo del manipulador a un arco de 0.245 mm.

- Para la articulación de la muñeca se utiliza un encoder extraído de un Mouse de computadora que tiene 40 ranuras, en este caso el disco se ubicó solidario al eje del motor pero antes de su reducción, por lo que la precisión en la articulación se multiplica por un factor igual al de la reducción del motor (325). De esta manera tenemos la situación equivalente a un encoder de 13000 ranuras directo en el eje de la articulación, o sea 52000 estados por revolución. La precisión es de 360/52000 = 0,0069 grados, un arco de 0.012 mm. en el extremo.

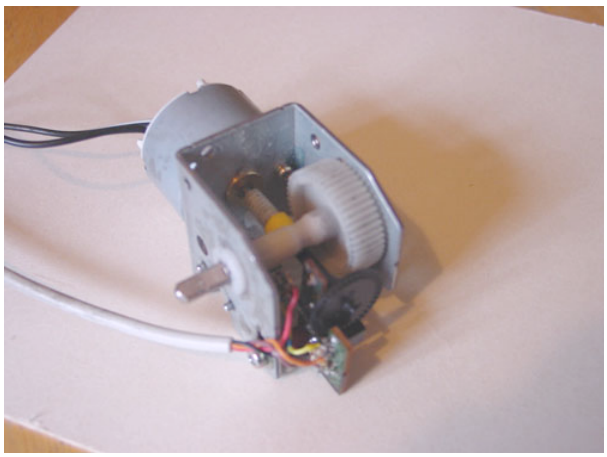


Fig. II.13 - Encoder de la muñeca, se ubicó el disco ranurado en el eje del motor, antes de la reducción

Fig. II.14 - Dos de los encoders "DATA TECHnology"

II.8 El Sistema de Coordenadas de Umanis

El sistema de Coordenadas de Umanis está compuesto por ejes de revolución de coordenadas cilíndricas, haciendo la salvedad de la coordenada utilizada para la pinza, la cual hace referencia a un "desplazamiento" lineal.

Se incluye en el sistema de coordenadas un sistema cartesiano ubicado en la base del manipulador a los efectos de poder diseñar de forma más directa las trayectorias lineales del extremo del manipulador.

En la figura siguiente se muestra esquemáticamente el manipulador y sus coordenadas naturales correspondientes.

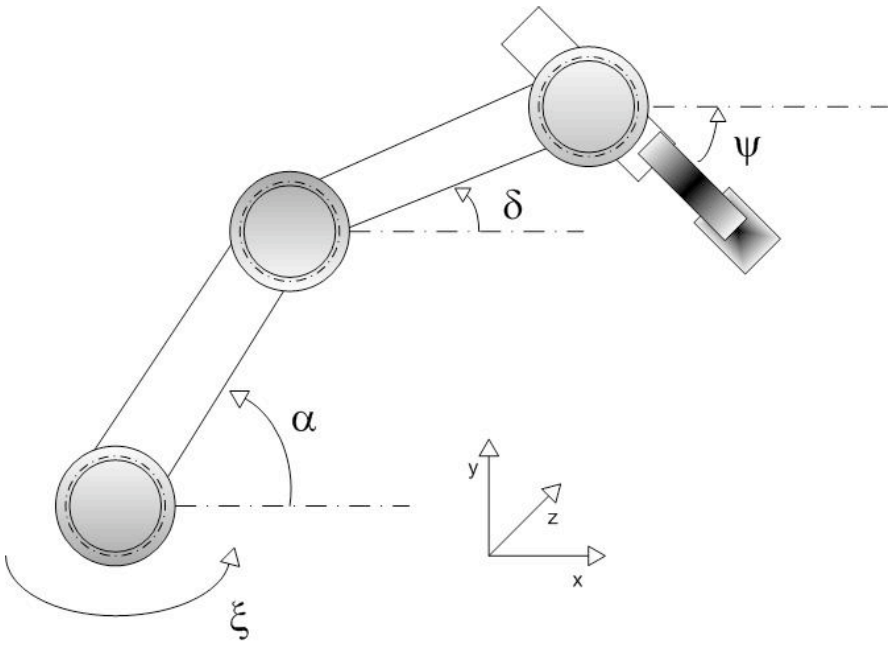


Fig. II.15 – Diagrama de sistema de coordenadas de Umanis

Como se puede apreciar, el manipulador posee 4 grados de libertad, sin tener en cuenta el cierre/apertura de la pinza. Esta perspectiva es perpendicular al plano de trabajo del manipulador, o sea, podríamos decir con seda=0. Como se puede observar, los restantes ángulos están referidos a la horizontal, por lo que no son independientes entre sí. Es decir, por ejemplo, si altero alfa, con esta definición, delta será alterado. Se tomó esta decisión con respecto a la referencia de las coordenadas ya que permite pensar el generador de trayectorias de manera más simple, dejando para el Software (de aquí en adelante SW) de más bajo nivel la conversión de ángulos relativos a ángulos absolutos.

Esto también se vio motivado por otros efectos mecánicos a ser "traducidos" por el SW, más precisamente lo que nosotros llamamos, el efecto "Coder". Este efecto consiste básicamente en una variación del ángulo relativo entre el segundo segmento y el primero (brazo y antebrazo), cuando el primero es girado. Más explícitamente, se observa que si se altera alfa en noventa grados, el ángulo relativo mencionado decrecerá 1/3 del desplazamiento de alfa, o sea 30 grados. Esto se debe a las características constructivas de las transmisiones entre el codo y el hombro. Naturalmente, este efecto es propagado a las articulaciones superiores. Todos estos inconvenientes, más la desreferenciación de las coordenadas, fue solucionado y tenido en cuenta en un módulo de SW llamado traductor.lib.

Esta convención será seguida en todos los módulos de SW y cálculos realizados en esta documentación.

De modo que la disposición de los ángulos será la siguiente:

- Alfa: referente al hombro.
- Delta: Referente al codo.
- Psi: Referente a la muñeca.
- Seda: Referente al giro del hombro o cintura.

III Hardware Eléctrico

III.1 Generalidades

La finalidad de nuestro hardware eléctrico es el implementar el control de los motores DC de forma de poder lograr que el comportamiento del manipulador se ajuste a los objetivos de este proyecto. Para cumplir con este fin se debe lograr que los motores tomen valores específicos de posición, velocidad y/o torque motor en momentos del tiempo también determinados. La forma en que se determina la configuración (1) del manipulador en cada momento no es motivo de análisis en este apartado, pero si lo es el funcionamiento del sistema que a partir de una configuración actual y real recibe la orden de alcanzar una configuración dada en un tiempo posterior T.

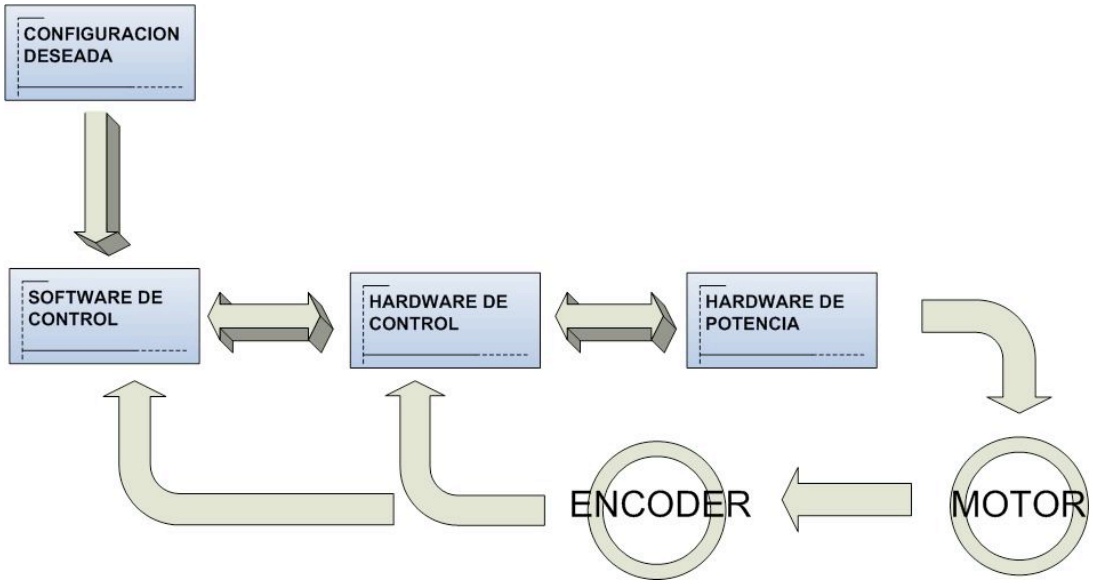


Fig. III.1 – Diagrama de bloques.

Se fabricaron dos circuitos impresos, el correspondiente a la etapa de potencia y el correspondiente a la etapa de control, además de la utilización del módulo de Z-World RCM3200 [20] que contiene el microprocesador Rabbit 3000 [19]. La complejidad de la placa de control hizo que buscáramos la alternativa de mandarla construir en el exterior, mas precisamente en EEUU, ya que las alternativas que teníamos para la construcción de la misma en nuestro país no nos resultaron satisfactorias.

(1)La configuración del manipulador es el conjunto de posiciones instantáneas de cada uno de los motores, que determinan la posición del extremo del manipulador y su orientación.

III.2 Alimentación

La alimentación de las placas se obtiene de las baterías existentes en SULLA. De las mismas se obtiene los niveles de tensión necesarios para el correcto funcionamiento de todo el sistema.

De la BAT1 existente en SULLA se alimenta la placa de potencia con 12v; que son utilizados por los drivers para alimentar los motores.
De la BAT2, en SULLA se alimenta una fuente que entrega los niveles de tensión necesarios para alimentar toda la electrónica allí presente. Nosotros solo necesitamos lo niveles de 5v, 3.3v y -5v para alimentar correctamente los circuitos integrados utilizados para el manejo del manipulador.

Como se puede apreciar se diseñó toda la electrónica teniendo en cuenta los elementos ya existentes en SULLA de forma de poder aprovechar el trabajo ya realizado y asegurar la compatibilidad y la autonomía que son objetivos fundamentales de este proyecto.

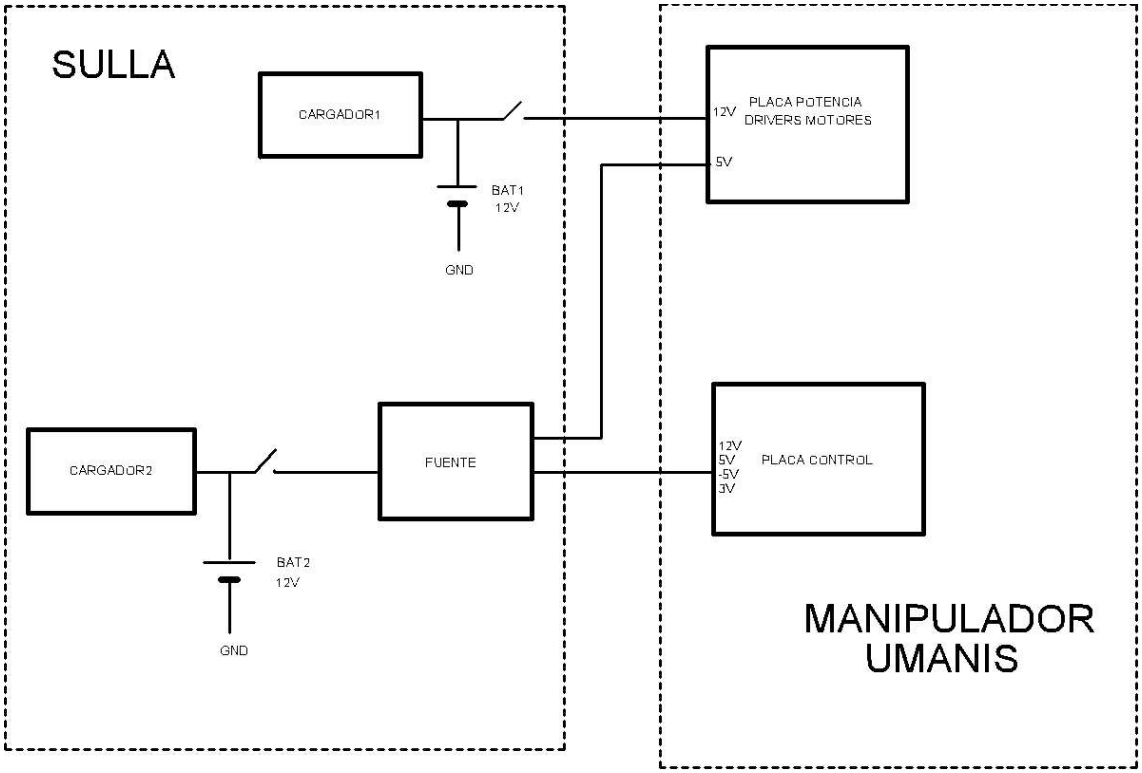


Fig. III.2 Diagrama de las conexiones de alimentación

III.3 Placa de Control

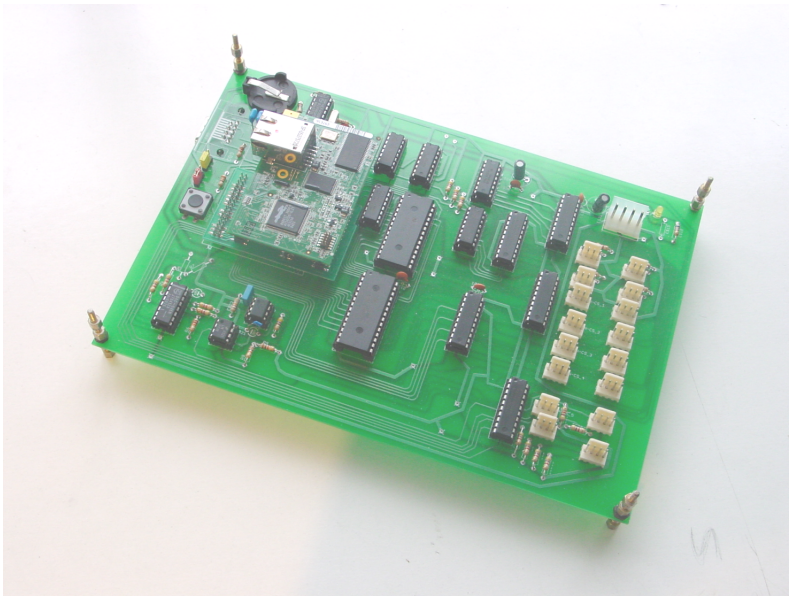


Fig. III.3 – Placa de Control o Electrónica de señal

La placa de control está compuesta por bloques que en su conjunto determinan, a partir de las órdenes recibidas del módulo RCM3200, el comportamiento del manipulador. A través de los conectores J1 y J2 se intercambian un conjunto de señales lógicas entre la placa de control y el módulo RCM3200, al mismo tiempo que las señales generadas en todo el bloque de control se derivan a la placa de potencia para manejar los motores DC.

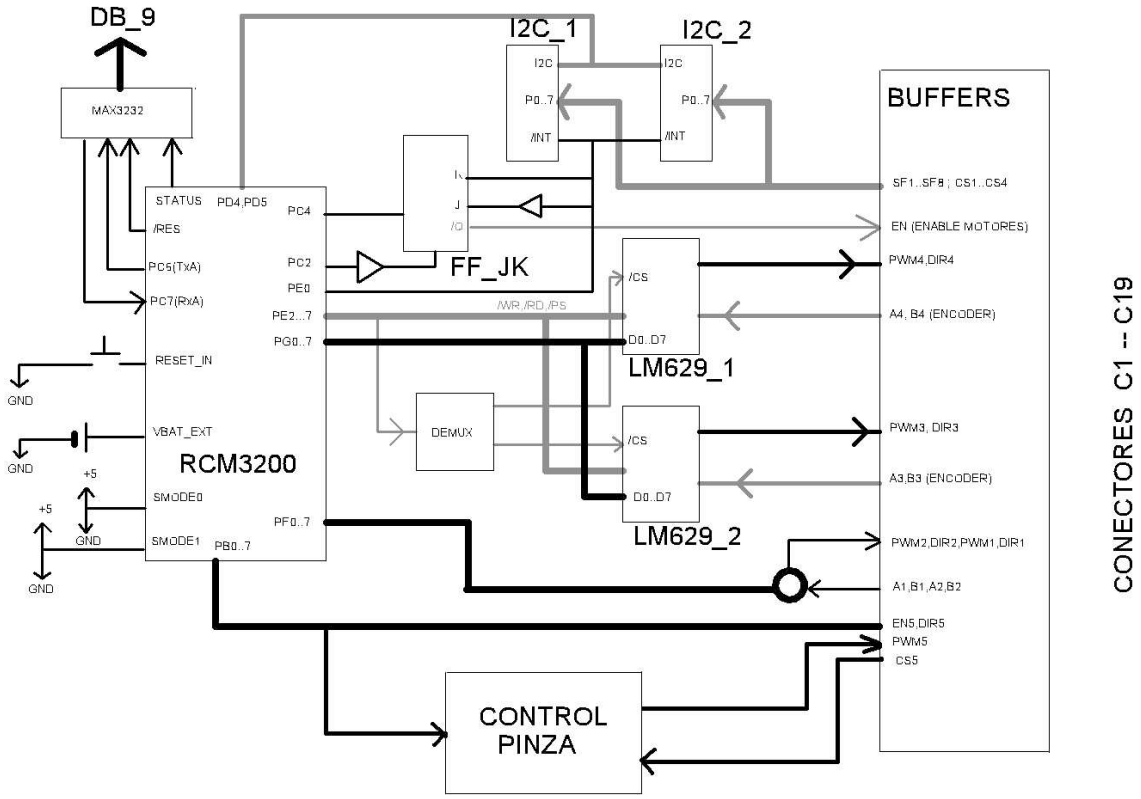


Fig. III.4 – Esquemático de la placa de control

Módulo RCM3200

Fig. III.5 – Módulo RCM 3200 [20]

Las alternativas que se manejaron al momento de elegir el dispositivo que fuera capaz de cumplir con las funcionalidades que requieren cierta inteligencia fueron varias pero solo dos fueron alternativas verdaderamente factibles para nuestros objetivos. Por un lado la posibilidad de realizar el sistema con varios PIC de forma que cada uno de ellos se ocupara de una función específica, por ejemplo el control de un motor, y que a la vez existiera una comunicación entre ellos. Por otro lado la utilización de un módulo de Z-world RCM3200.

Al comparar las alternativas notamos claramente que el módulo de RCM3200 era la mejor alternativa por tener una velocidad de procesamiento más alta, mayor

capacidad de memoria, un puerto Ethernet y un entorno de desarrollo interesante. Además de personas allegadas con cierta experiencia en el manejo del mismo, motivo nada despreciable en el momento que se tomó la decisión.

El módulo necesita ser alimentado con 3,3v y tiene la posibilidad de implementar una interfáz de programación además de un pulsador de reset y una batería externa para mantener la información en la memoria mientras no está alimentado. Tiene dos conectores J1 y J2 desde los cuales y hacia los cuales fluyen las señales de interés para las distintas funciones a implementar. El voltaje máximo que soportan las señales digitales en estos conectores es de 5,5v, lo que nos permitió utilizar chips que utilizan lógica TTL con voltaje de alimentación de 5v.

Dentro de esta placa de control se pueden distinguir dos circuitos integrados LM629 que realizan cada uno el control de un motor [3], un bloque que se encarga de recibir del manipulador posibles alarmas sobre algún comportamiento anormal, un bloque para realizar el control de torque del motor que se encarga de la apertura y cierre de la pinza, elementos que implementan funcionalidades que propician el correcto funcionamiento del módulo RCM3200, así como cinco buffers que tienen como función la protección de los elementos de la placa contra posibles sobre corrientes.

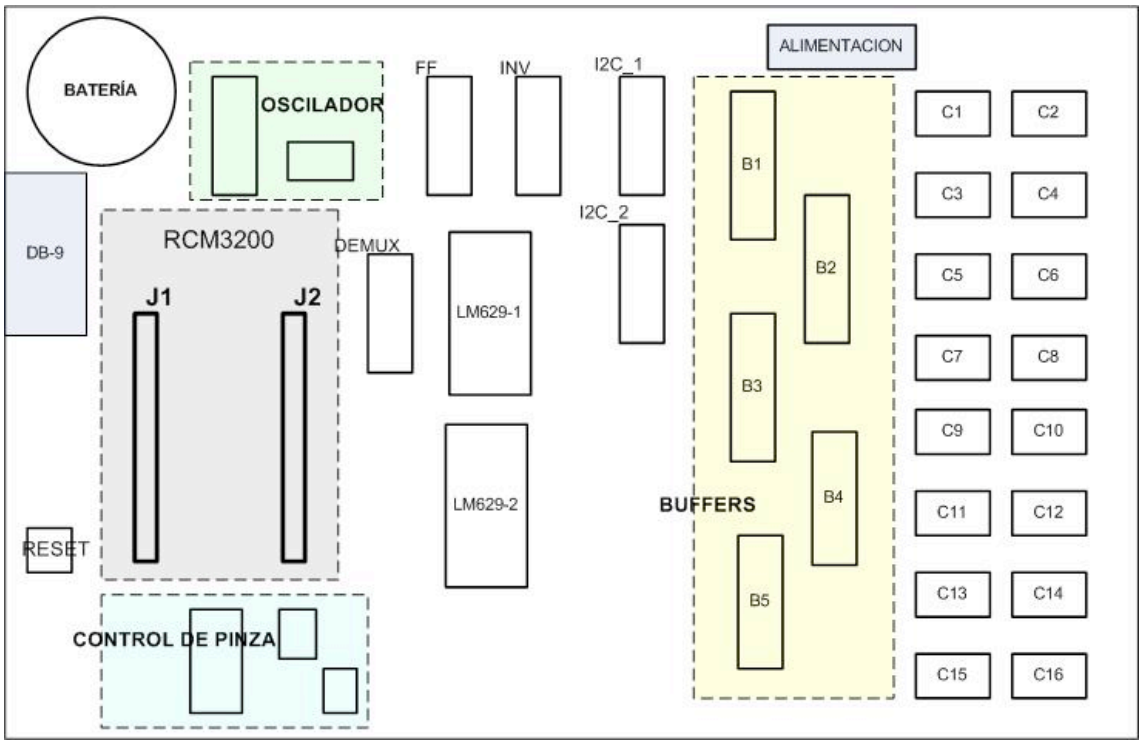


Fig. III.6 – Diagrama de la placa de Control

III.3.1 Conectores

Los conectores C1 al C16 son utilizados para intercambiar señales con la placa de potencia, además de alimentar y leer señales desde los encoders. También permiten la conexión de los switchs de final de carrera y de home del manipulador (véase figura III.5).
La siguiente tabla especifica cuales son las señales que manejan los mismos:

C1

SF1	SF2	GND	GND

C2

SF3	SF4	GND	GND

C3			
SF5	SF6	GND	GND

C4			
SF7	SF8	GND	GND

C5			
PWM_4	DIR_4	EN	CS_4

C6			
A_4	B_4	+5V	GND

C7			
PWM_3	DIR_3	EN	CS_3

C8			
A_3	B_3	+5V	GND

C9			
PWM_2	DIR_2	EN	CS_2

C10			
A_2	B_2	+5V	GND

C11			
PWM_1	DIR_1	EN	CS_1

C12			
A_1	B_1	+5V	GND

C14			
PWM_5	DIR_5	BK_5	CS_5

C15			
S_1	S_2	+5V	GND

C1			
H1	H2	GND	GND

C2			
H3	H4	GND	GND

III.3.2 Entorno del rabbit

Se implementó sobre la placa de control, la interfaz de programación, en selector de modo de inicio, un pulsador que genera una señal de reset sobre el microprocesador y la batería que se conecta a la señal VBAT_EXT del conector J1.

Esto se realizó a partir de las especificaciones que se encuentran tanto en el manual de usuario del rabbit3000(microprocesador) así como en el de la placa RCM3200.

III.3.3 Interfaz de Programación

El circuito que se usó para la interfaz de programación se encuentra en el esquemático proporcionado por el fabricante y nuestra función en este caso fue limitarnos a intentar reproducir con la mayor fidelidad el circuito electrónico que se presenta en el esquema antes mencionado.

Utiliza un circuito integrado *MAX3232* para realizar la comunicación con el puerto serie de la PC. El mismo realiza una conversión eléctrica entre niveles CMOS que maneja el microprocesador con los niveles RS-232 que maneja el puerto serie de la PC.

Los pines SMODE0 y SMODE1 son utilizados para indicarle al microprocesador el modo en el que se encuentra y son manejados a través de jumpers. El modo programación es uno de los modos de "boostrap" definidos en el manual de usuario del *Rabbit3000* y el mismo se produce cuando después del reset del microprocesador tiene sus entradas SMODE = 11 (Suponemos SMODE = (SMODE1 , SMODE0)) , en este caso el puerto serie A (PC6 y PC7) es seleccionado para la operación de "boostrap" . En estas condiciones el microprocesador se encuentra listo para recibir de la PC toda la información que permite la programación del *Rabbit3000* y el "debugging" correspondiente. En el caso de encontrarse SMODE = 00, luego del reset, las instrucciones son ejecutadas a partir

de la dirección 0x0000 de la memoria. Estos dos modos son los que nos permite separar entre modo programación y modo normal que son los que van a ser utilizados durante el uso del manipulador tanto para reprogramar el microprocesador como para el uso normal del mismo. Los otros modos de "boostrap" no serán utilizados.

Cabe mencionar que la comunicación con la PC se hace a través del conector *DB-9* (macho) instalado en la placa, conectando al mismo un cable hembra-hembra directamente a un puerto serie de la PC.

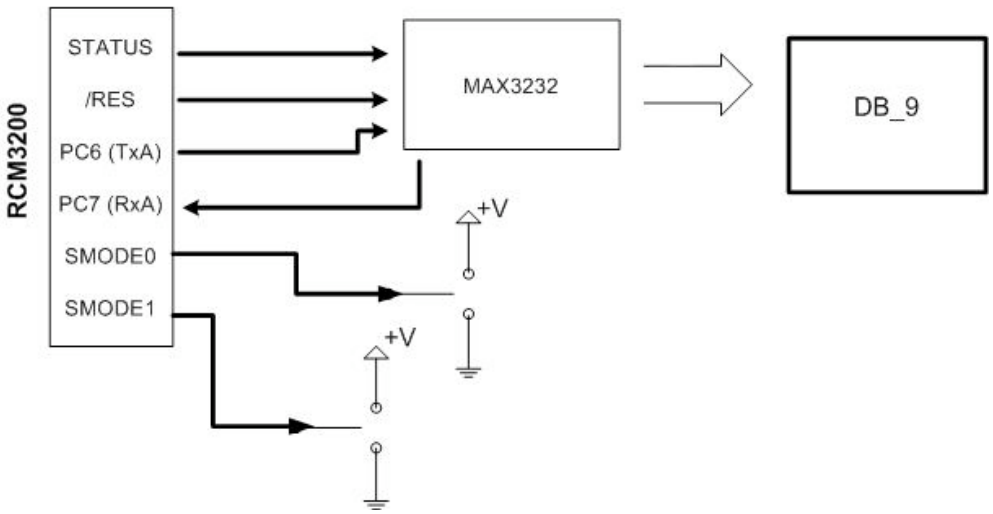


Fig. III.7 – Diagrama de bloques del cable de programación del Rabbit 3000

III.3.4 Uso de Batería

Se utiliza una batería de litio de 3v cuyas especificaciones coinciden con las exigidas en el manual de la placa RCM3200. La misma es conectada a la señal VBAT_EXT y permite mantener la información en la memoria RAM mientras la placa no se encuentra alimentada, además de tener conectada en serie una resistencia para limitar la corriente

Un pulsador conectado al pin RESET_IN que se encuentra en el conector J1, permite ejecutar la función de reset sobre el Rabbit3000.

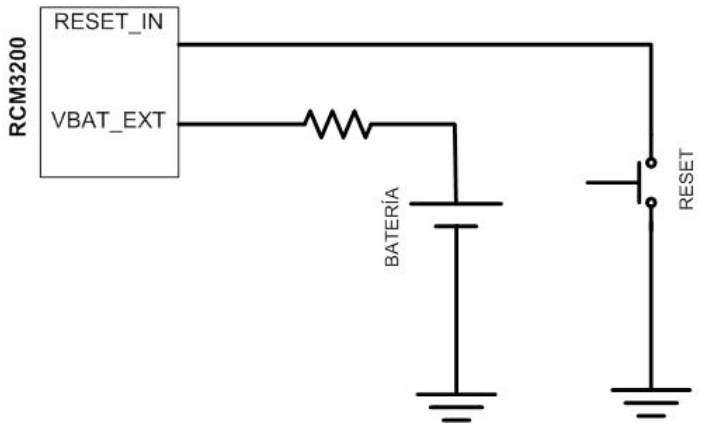


Fig. III.8 – Esquemático de conexión de batería.

Todas estas funciones fueron realizadas, al igual que en el caso del cable de programación, tratando de seguir fielmente a las recomendaciones de los manuales correspondientes

III.3.5 Protección

Utilizamos como protección buffers no inversores 74HCT244. Éstos son buffers CMOS-TTL compatibles y funcionan siempre habilitados, su función es únicamente proteger al rabbit3000 y al resto de componentes, de posibles excesos de corrientes. Los pines del RCM3200 son capaces de manejar hasta 5,5v de tensión y 6mA de corriente. Se logra con el buffer manejar corrientes de hasta 35mA, las cuales no son necesarios en condiciones normales, pero si tal vez frente algún imprevisto. Lo mismo si ante algún problema en la placa de potencia, llegaran 12v a través de los conectores, la sobretensión sería absorbida por el buffer y no propagada al resto de los componentes de la placa.

III.3.6 Control de Motores

De los cinco motores a controlar, uno se controla en torque y cuatro son controlados en posición. De estos últimos dos de ellos son controlados directamente por RCM3200 y dos son controlados por circuitos integrados dedicados al control de motores servo. Estos integrados, (LM629) permiten el control de lazo cerrado de motores servo tanto en velocidad como en posición [4]. El mismo posee un generador de trayectoria, un filtro PID, una interfaz con un host y un bloque de lectura de encoder óptico incremental en cuadratura. Utilizando el rabbit3000 como host, a través de un bus de datos implementado en el puerto paralelo G del microprocesador y de varios pines de control estamos en condiciones de delegar a los LM629 el control de un motor servo cada uno.
La elección de utilizar circuitos integrados dedicados para el control de dos motores surge del estudio realizado en la etapa previa. Se realizó un estudio y prueba de los integrados para generar una alternativa al control, al de controlar todos los motores por parte del microprocesador rabbit3000. El resultado de ésta fue altamente satisfactoria dando además los LM629 una fácil implementación de comunicación con un host que en nuestro caso coincide con el microprocesador. Para aprovechar el tiempo invertido en el mencionado estudio y teniendo en cuenta que el microprocesador trae librerías que permite el control de dos motores servo, nos inclinamos por utilizar las librerías ya existentes para controlar dos motores e implementar la comunicación con los LM629, para que estos últimos se encarguen de los otros dos motores que llevan un control en posición.

Esto nos permite liberar un poco al microprocesador para realizar las otras tareas que tiene asignadas.

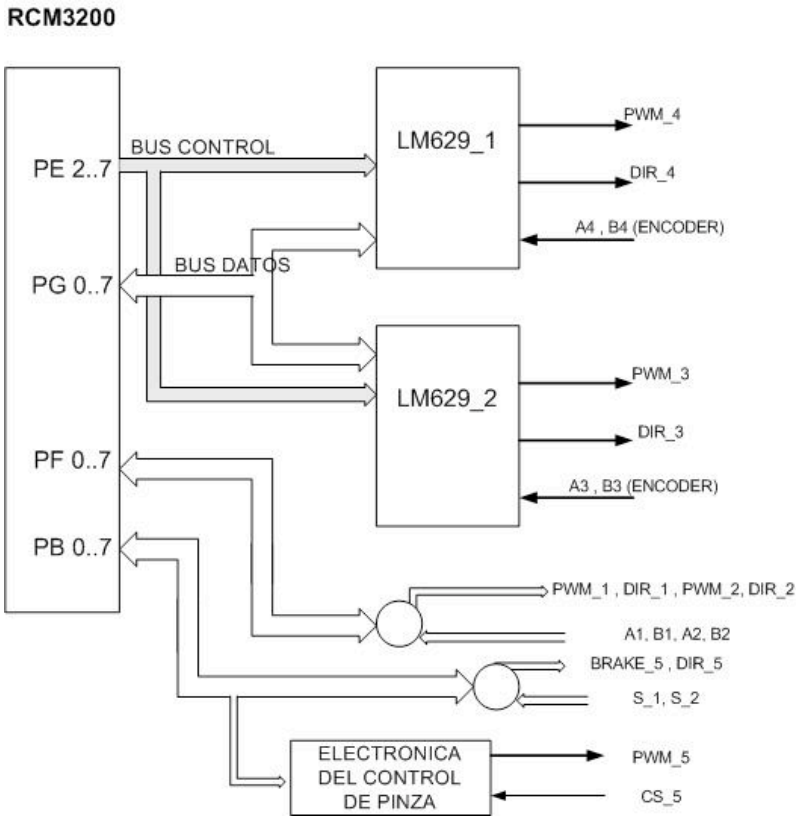


Fig. III.9 – Diagrama de conexiones

Motores controlados por RCM3200

La placa RCM3200 tiene un bloque de hardware dedicado al control de hasta 2 motores servo. Esto implica señales de salida de la misma que generan PWM que se traducen en un nivel de voltaje sobre el motor y entradas que permite la lectura de un encoder óptico incremental en cuadratura por motor. El puerto paralelo F fue el elegido por el fabricante para esta función y nosotros decidimos utilizar esta facilidad que nos provee la placa RCM3200 para nuestros fines.

A continuación un esquema de las conexiones realizadas:

RCM3200 (Puerto F)	*CONECTORES*
PF6	**PWM_1**
PF7	**DIR_1**
PF1	**A_1**
PF0	**B_1**
PF4	**PWM_2**
PF5	**DIR_2**
PF3	**A_2**
PF2	**B_2**

Motores controlados por LM629

Como ya hemos mencionado los LM629 son integrados dedicados al control de movimiento de motores servo. La interfaz de software con el host es simple ya que provee un conjunto de comandos de alto nivel [1]. El mismo genera una salida directa para manejar el puente H que en nuestro caso están implementados por los drivers LMD18200. Las conexiones entre el controlador LM629 y el driver utilizado están expuestas en las propias hojas de aplicación del controlador y no nos generó ninguna dificultad. Los controladores requieren de un oscilador de entre 1MHz a 8MHz por lo que se tuvo que construir un bloque de hardware que cumpliera con los requerimientos detallados en la hoja de datos.

Para realizar la comunicación entre la placa RCM3200 y los controladores se utilizó una arquitectura de buses. Por un lado se encuentra el bus de datos que corresponde al puerto **G** directamente conectado a los pines que tienen asignados los controladores para este fin y también un bus con las señales de control. Para el caso de la señal que selecciona el controlador con el que se va a establecer la comunicación (CS) se introdujo en el circuito un demultiplexor de forma de tener la seguridad de que no sean seleccionados por error los dos controladores generando un problema de contención en el bus de datos.
A continuación un esquema sobre las señales intercambiadas:

RCM3200 (Puerto E)	*LM629A*
PE6	**/CS_1**
PE7	**/CS_2**
PE5	**/RD**
PE4	**/WT**
PE3	**/PS**
PE1	**/RST**

III.3.7 Oscilador

En la placa dedicada al control fue necesario incluir un bloque de circuito que implemente un oscilador para ser usado como entrada de reloj tanto en los LM629 como en el flip-flop que maneja el sistema de alarmas.

Se diseñó un oscilador a cristal de 7 Mhz (la máxima frecuencia aceptada por los LM629 es de 8 Mhz. y la del Flip-Flop es de 25 Mhz.) [16].

Se implementa con un par de inversores y el circuito es relativamente simple, fue extraído de las hojas de datos del propio chip inversor (HEF4069 de Philips).

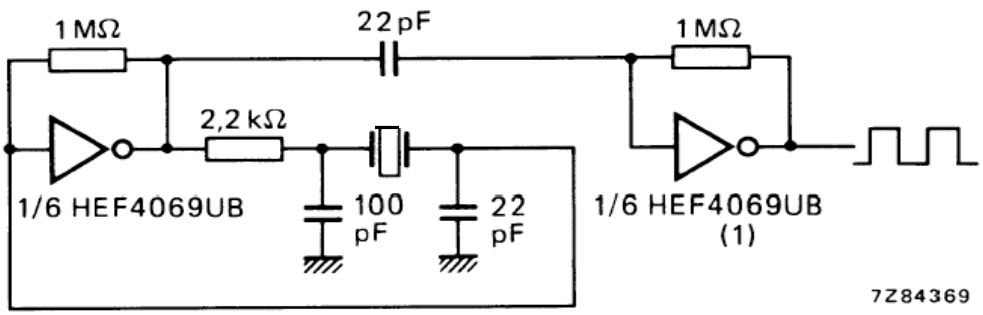


Fig. III.10 – Esquemático del Oscilador []

El segundo inversor es el encargado de adaptar los niveles de la salida para que sea CMOS compatible.

III.3.7 Control de Pinza

Para la apertura y cierre de la pinza se optó por un control de torque sobre el motor correspondiente, implementado indirectamente por un control de corriente.

Se optó por este método porque no creíamos necesario tener un control preciso sobre cuan abierta estaba la mano, lo cual complicaba el sistema bastante, dado que era necesaria la inclusión de un encoder en el eje del motor o un mecanismo similar para ver su posición. Con el método elegido no es necesario ningún hardware mecánico extra, el problema de control es resuelto completamente con electrónica.

Con este control, cada vez que es necesario abrir la mano, esta se abre lo máximo posible (este máximo es detectado con el sensor de final de carrera) y al cerrarla se controla su torque (corriente) dependiendo del objeto y su fragilidad.
El circuito es el siguiente:

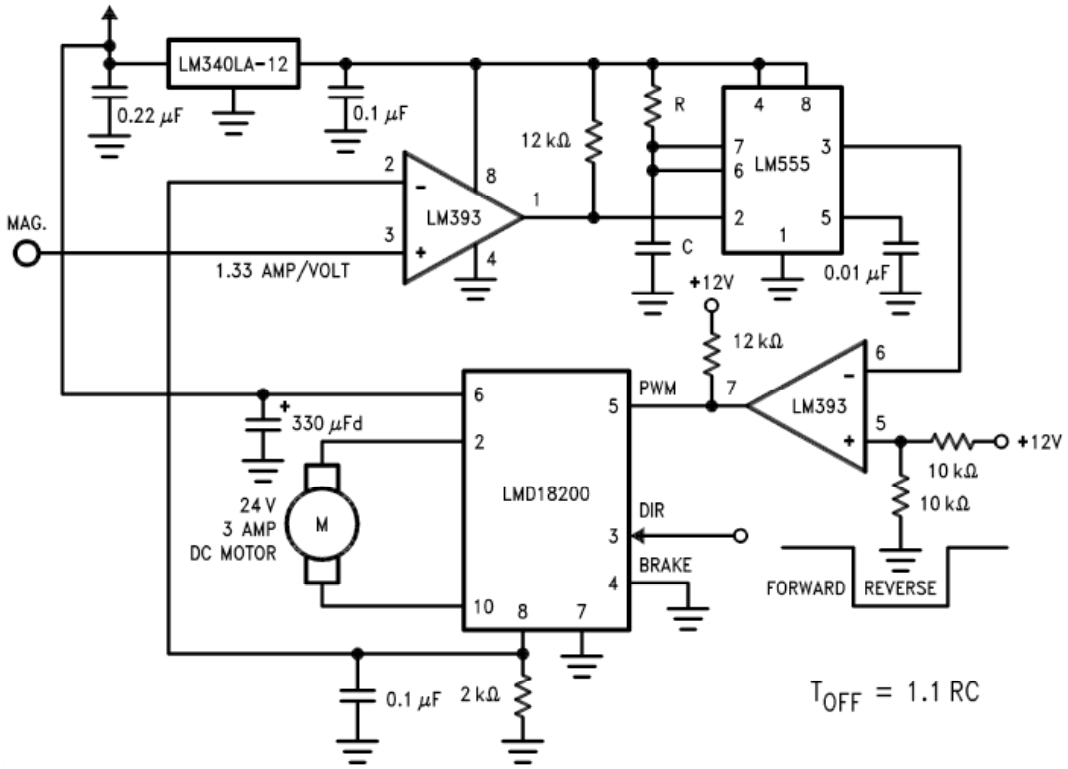


Fig. III.11 – Esquemático del circuito de Control de Pinza [2]

Aprovechando una función implementada en el LMD18200 la cual entrega una corriente igual a 377 micro Ampere por cada Ampere que circula por el motor, se tiene, conectando en ese lugar una resistencia a tierra, un voltaje proporcional a la corriente por el motor.

Este valor analógico de voltaje es comparado (el chip comparador es el LM393) con un valor de magnitud (MAG. en la figura) determinado por el rabbit a través de un conversor analógico digital.

III.3.8 Gestión de alarmas

Se implementó sobre la placa de control un sistema que permitiera a partir de alguna situación anormal que sucediera sobre el manipulador, identificar el problema surgido y en la medida de las posibilidades que el propio sistema encuentre una solución o directamente bloquear todo el movimiento del manipulador y enviar un mensaje de alarma al usuario a través del puerto Ethernet.

El módulo está compuesto por un FF y por dos expansores de bus I2C, a saber PCF8574A [26] que se conectan directamente al módulo RCM3200.

Se toman como situaciones anormales el caso de que ya sea por que el manipulador haya tomado un objeto más pesado de lo que figura en los objetivos de este proyecto así como si se topara con algún objeto extraño, la generación de un esfuerzo extra sobre los motores, así como también el caso de que se llegara a topar el manipulador con algunos de los límites impuestos en los recorridos de cada articulación.

En el caso de que la situación anormal sea el haber llegado al tope de algún recorrido, los pulsadores se activarían bajando a nivel lógico cero una de las entradas de los expansores, este haría dispararse la señal /int del expansor que generaría a través del FF que la señal de habilitación de los motores subiera a uno, deshabilitando los mismos a la vez que genera una interrupción externa en el microprocesador. El sistema depende a partir de ese momento de la rutina de atención a la interrupción para volver a las condiciones normales de funcionamiento. Similar para el caso de que se hubiera generado un exceso de corriente consumida por alguno de los motores, la forma en la que se genera esta señal se detalla en el apartado que corresponde a la placa de potencia.

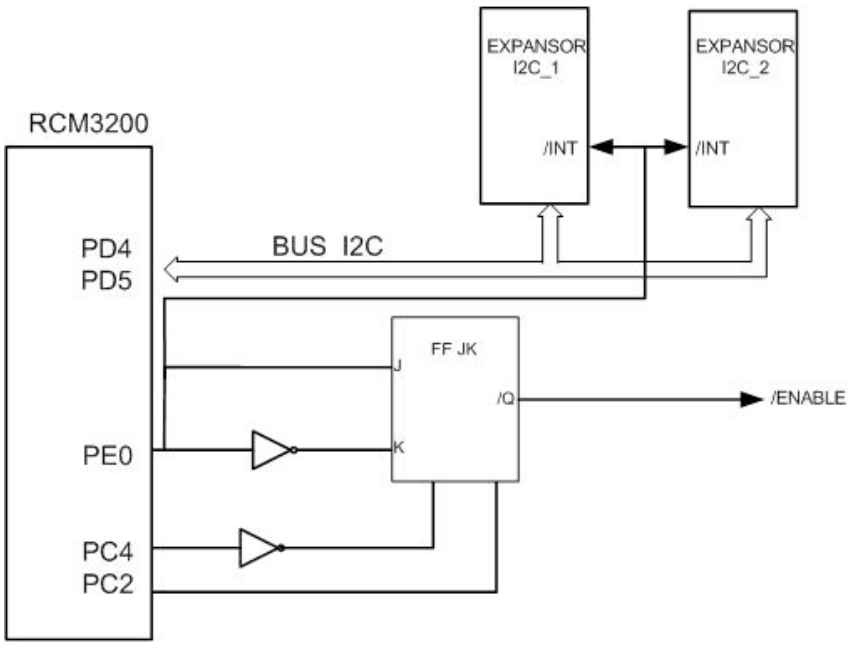


Fig. III.12 – Esquemático de circuito de gestión de alarmas.

III.4 Placa de Potencia

III.4.1 Descripción General

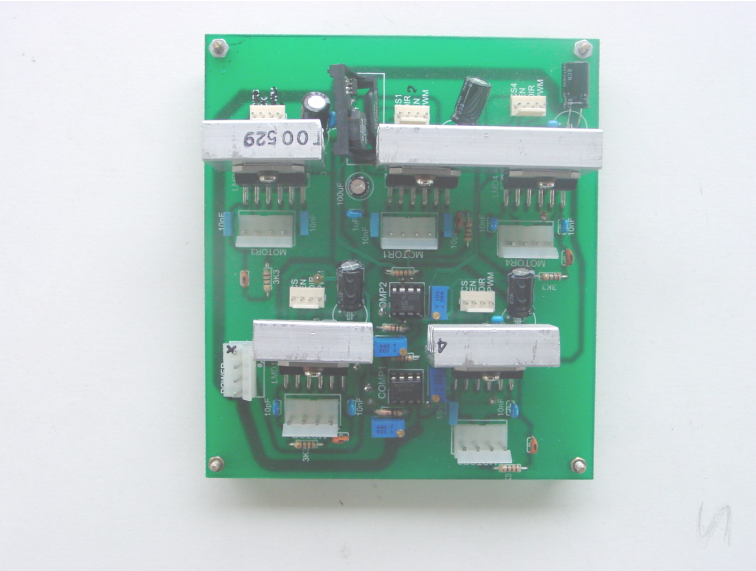


Fig. III.13 – Placa de Potencia

Le llamamos placa de potencia a la placa donde se encuentra ubicado el conjunto de componentes electrónicos, que transforma las señales de control de tipo lógico en un nivel de potencia suficiente para alimentar los motores. A la misma llega la información necesaria para comandar los drivers y éstos entregan un nivel de tensión directamente a cada motor.

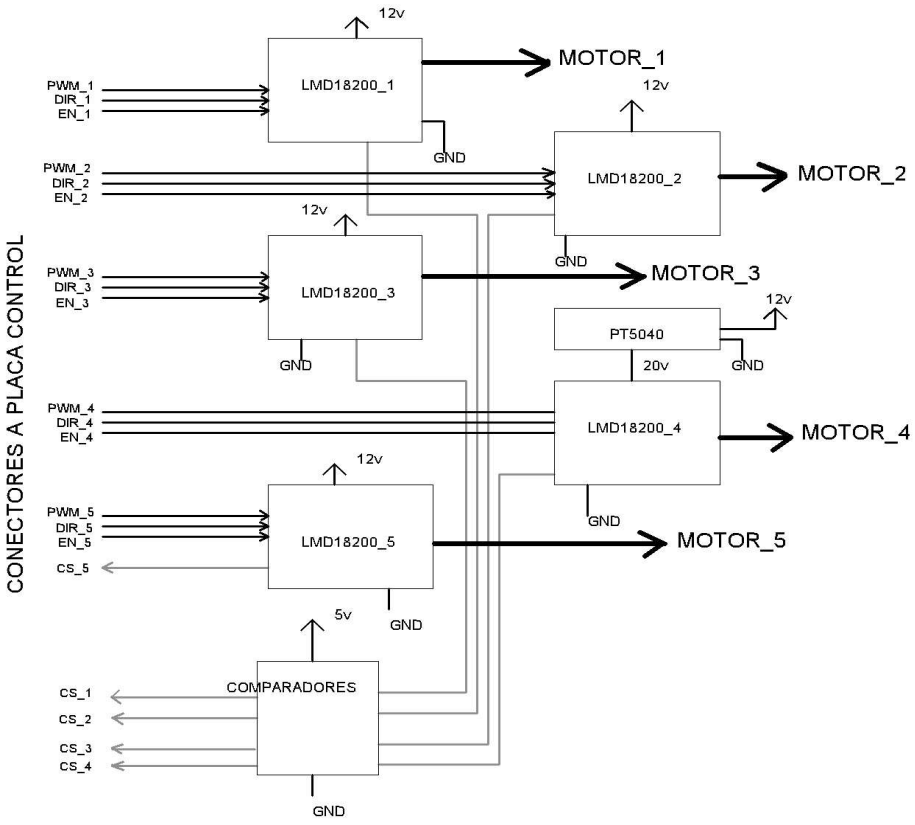


Fig. III.14 – Esquemático de la placa de potencia

Se pueden identificar en la placa cinco módulos o bloques, uno correspondiente a cada motor. Tres de ellos son similares, ya que los tres alimentan cada uno a un motor DC de 12v e intercambian las mismas señales con el bloque de control. Los otros dos difieren levemente ya que uno de ellos alimenta un motor DC de 24v por lo que el bloque tiene incorporado un elevador de tensión y el otro alimenta el motor que se encarga de la apertura y cierre de la pinza por lo que las señales que intercambia con el bloque de control difieren con las que intercambian los otros bloques.

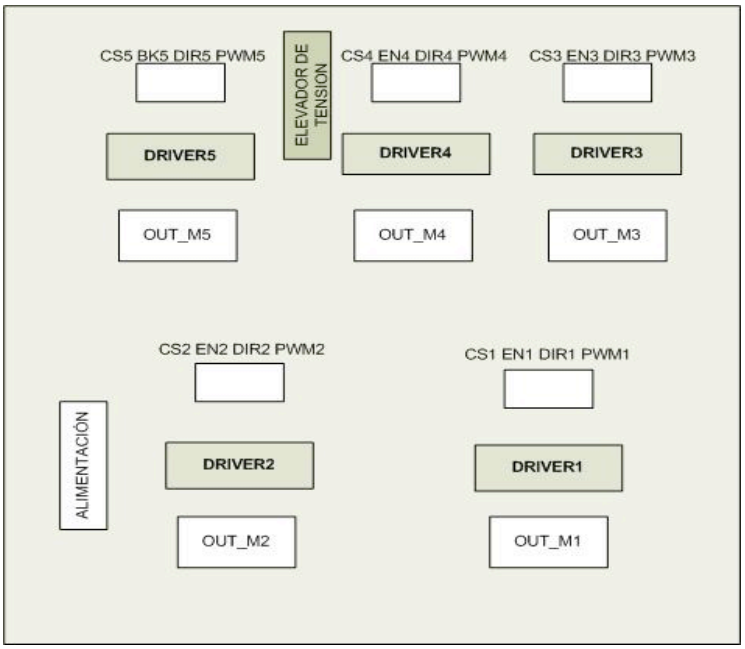


Fig. III.15 – Diagrama de bloques de Placa de Potencia

III.4.2 Driver (Puente H)

Cada bloque o módulo de los que se mencionaron tiene incorporado un circuito integrado LMD18200. El LMD18200 es un circuito integrado con capacidad de manejar tanto motores DC como motores de paso. Combina tecnologías bipolar y CMOS dentro de un bloque lógico interno con dispositivos de potencia DMOS que conforman un puente H.
Puede comandar corrientes de 3A en forma continua así como picos de hasta 6A. También implementa una salida analógica que entrega 337uA por Amper consumido por el motor.(2)

El dispositivo consta de tres entradas lógicas (PWM, DIR y BRAKE) que tienen como característica el ser TTL-CMOS compatibles. A partir de éstas, la lógica del dispositivo comanda las llaves del puente H para que se genere sobre el motor el nivel de voltaje conveniente.

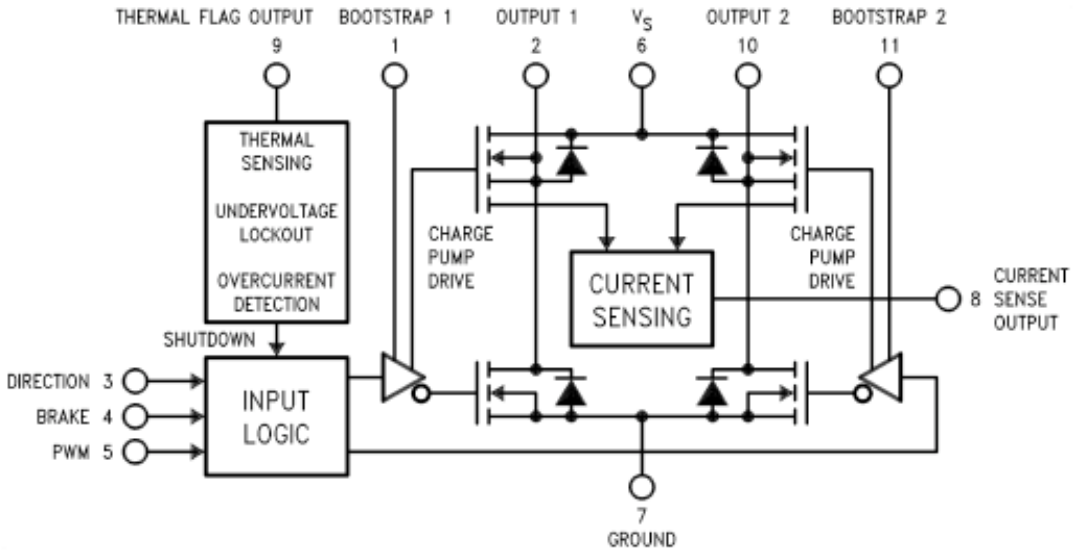


Fig. III.16 – Diagrama de bloques del integrado LMD 18200 [14]

III.4.3 Capacitores de protección

Durante la apertura de las llaves del puente H, los cambios bruscos de corriente producen transitorios de voltaje que pueden llegar a valores muy altos. Para proteger al sistema de estos transitorios se instaló en el circuito entre la alimentación y tierra del driver un capacitor capaz de soportar dichos transitorios. Típicamente los capacitores cerámicos tienen una muy buena respuesta a altas frecuencias que es lo necesario en estos casos.

En paralelo al capacitor cerámico se conectó un capacitor de gran capacidad que permite absorber las corrientes inversas producidas en las cargas de tipo inductivas. El valor de los capacitores utilizados corresponde con las recomendaciones existentes en la hoja de datos del integrado.

Se colocaron disipadores en los drivers para evitar posibles sobrecalentamientos de los mismos. Para las corrientes manejadas por los mismos, no sería necesario ya que el consumo de los motores utilizados no excederían los valores dados por los fabricantes; de todas formas se instalaron disipadores como un elemento más de protección sobre los drivers.

(2)Por otros datos técnicos consultar su hoja de datos.

III.4.4 Elevador de tensión

Al tener un motor de 24v, el alimentarlo con un voltaje máximo de 12v, nos limita fuertemente las posibilidades del mismo para lograr los objetivos planteados tanto de torque como de velocidad.

Se propuso como solución la utilización de un elevador de tensión PT5049.
El mismo es un circuito que toma tensiones en un rango de 5v a 14v y entrega un voltaje de 20v con una corriente máxima de 0.5Amp. Lo que dado las características del motor y sus requerimientos representa una solución satisfactoria.

En nuestro caso alimentamos el elevador de voltaje con 12v lo que representa una mayor eficiencia del integrado, comparado con la posibilidad de alimentar el mismo con 5v, además de mantener el criterio de diseño que mantiene separada la alimentación de los motores de la electrónica.

III.4.5 Sensado de corriente y generación de alarma por exceso de consumo.

A la señal *current sense* entregada por el dispositivo *LMD18200* se le conecta entre ésta y tierra una resistencia de valor conocido, con esto tenemos un nivel de voltaje sobre la misma proporcional a la corriente que está circulando por el motor. El factor de proporcionalidad depende básicamente de la resistencia y de la relación de corrientes que te genera el propio integrado (337uA/A). Esto fué utilizado para dos fines; por un lado para controlar que la corriente que consuman los motores no extralimiten un umbral que será definido a partir de los ensayos efectuados sobre los motores, generando una alarma que corte la alimentación sobre los mismos, desactivando la señal de enable que está conectada directamente a la entrada de Brake de los drivers. Por otro lado en el caso del motor que maneja la apertura y cierre de la pinza para realizar un control sobre el torque que genera el motor.

Para evitar sobre corrientes en los motores, los mismos fueron ensayados y se relevó la corriente que consume en configuraciones que requieren poco esfuerzo y en otras que requieren el máximo esfuerzo de acuerdo a los requerimientos. La idea es tomar esa corriente máxima y a partir de la misma dimensionar un umbral de funcionamiento del motor bajo condiciones normales. En el caso que la corriente consumida por el motor excediera el umbral, se considera que el sistema no se encuentra bajo condiciones normales y se corta la alimentación sobre los motores.

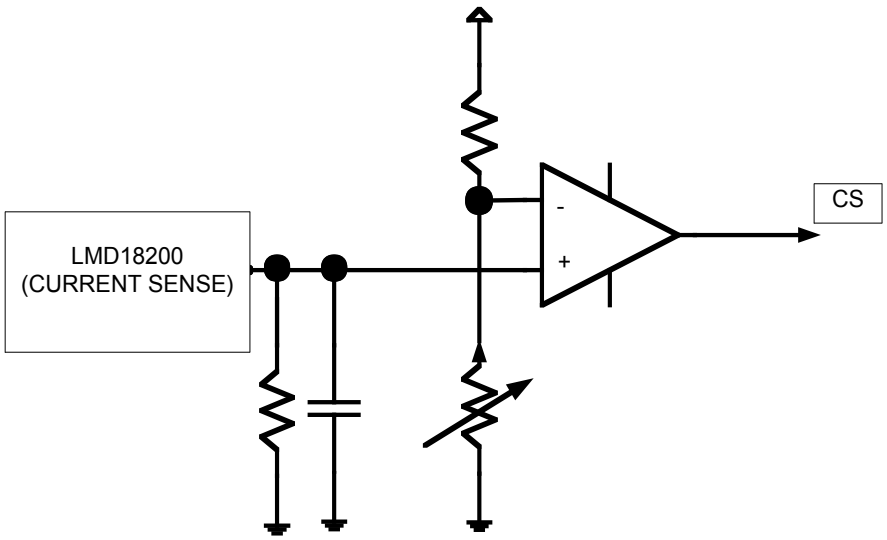


Fig. III.17 – Esquemático del comparador

Se puede apreciar en el diagrama que los potenciómetros multivueltas nos permiten ajustar el valor del umbral a partir de los valores obtenidos de los ensayos. En el momento que la señal *current sense* supera el umbral se dispara la alarma a través del comparador *LM239* que informa al microprocesador a través de una señal lógica la situación problemática, al mismo tiempo que corta la alimentación sobre los motores.

Para el caso de el control de torque es la propia señal analógica *current sense* la que se envía hacia la placa de control llevando la información sobre la corriente que consume el motor a cada instante.

III.4.6 Interruptores de home y final de carrera

Los interruptores van todos conectados de forma similar a los pines de entrada del expansor en el caso de los final de carrera y directamente al puerto A del RCM3200 en el caso de los interruptores de home.

Tienen en todos los casos uno de sus extremos conectado a una resistencia de *Pull-Up* y el otro extremo a tierra como se ve en la figura.

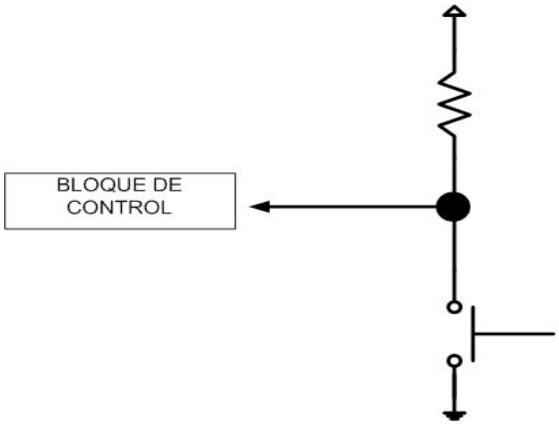


Fig. III.18 – Diagrama de interruptores

La señal de interés está normalmente conectada a nivel lógico alto a través de la resistencia y en el caso que se activara algún pulsador, ésta pasa a estar en nivel lógico bajo. Las señales identificadas como final de carrera son 8 y van desde SF1 a SF8, mientras que las señales que identifican la posición de home van desde H1 a H4.

IV Control de movimiento

IV.1 Introducción:

El manipulador robótico es básicamente un sistema posicionador con múltiples grados de libertad. La dinámica del robot es un sistema con varias entradas y varias salidas y es no lineal. El principal problema para el control de movimiento es la complejidad de la dinámica, resultante de la no linealidad y los múltiples grados de libertad.

Dado que el realizar un estudio profundo de la dinámica del manipulador es una tarea que excede el alcance de nuestro proyecto, no haremos un análisis exhaustivo de este tema. La idea de esta sección es introducir los fundamentos teóricos mas relevantes en el análisis de esta disciplina, para luego utilizarlos en el diseño del sistema de control.

La indeterminación paramétrica viene del hecho que se tiene un conocimiento impreciso de los parámetros de inercia, mientras que la indeterminación dinámica viene de la flexibilidad de las articulaciones, la fricción de las distintas partes, el posible ruido en los sensores de posición y el desconocimientos de la dinámica del ambiente.

Consideramos el manipulador como un sistema de 5 tramos interconectados por articulaciones o junturas que conforman una cadena cinemática. En nuestro caso todas las junturas son de revolución.

IV.2 Cinemática

Definimos las variables de las articulaciones q1, q2, q3, q4 como los ángulos entre los distintos tramos, de esta manera qi es el ángulo entre el tramo i e i-1. En nuestro sistema de coordenadas (ver sección de sistema de coordenadas), dichas variables se traducirían en:

- $q1 = \xi$
- $q2 = \alpha + 90°$
- $q3 = 180° - (\alpha - \delta)$
- $q4 = 180° - (\delta - \psi)$

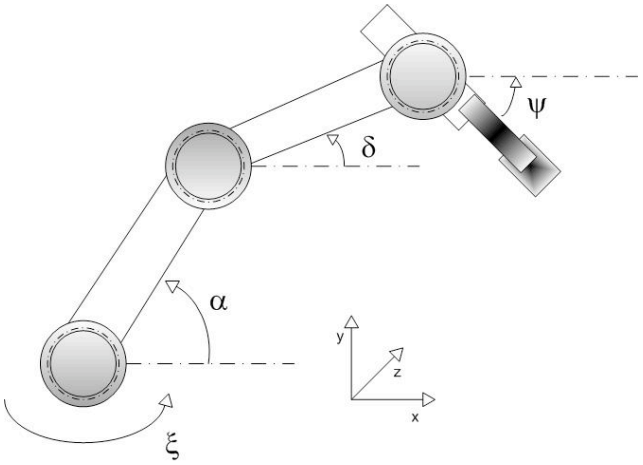


Fig. IV.1 – Esquemático del manipulador

Un vector $q = (q1, q2, q3, q4)^T$ es llamado **configuración**. El conjunto de todas las posibles configuraciones es llamado **Espacio de Configuraciones** (C) .

El **Espacio de actuador** es el espacio de todas las posiciones y orientaciones del actuador o pinza.

Tomamos como origen de coordenadas la base de nuestro manipulador y como punto a describir el extremo del mismo.

El mapeo de **Cinemática directa** es una función del espacio de configuraciones al espacio del actuador en términos de la posición de las articulaciones. El mapeo de **Cinemática inversa** da la configuración de las articulaciones como función de la posición y orientación del actuador.

El mapeo de cinemática directa es de muchos a uno, o sea, varias configuraciones distintas pueden dar como resultado la misma posición (y orientación) del actuador, además, cada configuración da como resultado una única posición (y orientación) del actuador, por el contrario la cinemática inversa tiene, generalmente múltiples soluciones.

El mapeo de cinemática directa se puede definir como una función del espacio de configuración en $\mathcal{R}^6$.

$$X_0 = \begin{pmatrix} x(q) \\ o(q) \end{pmatrix}$$

donde $x(q) \in \mathcal{R}^3$ da la posición cartesiana del actuador y $o(q)=(\phi(q), \theta(q), \gamma(q))$ representa la orientación del actuador.

La **Cinemática de velocidad** se refiere a la relación entre la velocidad de las articulaciones y la velocidad del actuador.

Si se utiliza el mapeo de cinemática directa visto anteriormente, entonces las velocidades están dadas por :

$$V = \begin{pmatrix} v \\ \omega \end{pmatrix} = J_0(q)\dot{q}$$

donde $J_0(q)$ es una matriz $6x4$ llamada la **Jacobiana del manipulador**. Los vectores ν y ω representan la velocidad lineal y angular respectivamente del actuador.

IV.3 Dinámica

La dinámica del manipulador será descripta forma Lagrangeana. En esta visión las variables de los ángulos de las articulaciones ofician como coordenadas generalizadas [21] [17].

La energía cinética del manipulador es dada por una forma cuadrática, simétrica y definida positiva.

$$K = \frac{1}{2}\sum_{i,j=1}^{4} d_{ij}(q)\dot{q}_i\dot{q}_j = \frac{1}{2}\dot{q}^T D(q)\dot{q}$$

donde $D(q)$ es la matriz de inercia.

La energía potencial del manipulador es dada por una función diferencial continua de $\mathcal{P} : C \rightarrow \mathcal{R}$. En el caso de nuestro manipulador la energía potencial es debida solamente a la gravedad.
Entonces la dinámica del sistema se escribe por las ecuaciones Lagrangeanas.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_k} - \frac{\partial L}{\partial q_k} = \tau_k, k = 1,..,4$$

donde $L=K - \mathcal{P}$ es el Lagrangeano y $\tau_1,..,\tau_4$ representan las fuerzas generalizadas de entrada. En coordenadas locales la ecuación Lagrangeana puede ser escrita así:

$$\sum_{j=1}^{4} d_{kj}(q)\ddot{q}_j + \sum_{i,j=1}^{4} \Gamma_{ijk}(q)\dot{q}_i\dot{q}_j + \phi_k(q) = \tau_k, k = 1,..,4$$

donde

$$\Gamma_{ijk} = \frac{1}{2}\left\{\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{kj}}{\partial q_j} - \frac{\partial d_{ij}}{\partial q_k}\right\}$$

$$\phi_k = \frac{\partial P}{\partial q_k}$$

En forma matricial, la ecuación Lagrangeana puede ser escrita de la siguiente manera:

$$D(q)\ddot{q} + C(q,\dot{q}) + g(q) = \tau$$

donde g(q) representa las fuerzas gravitacionales y C las fuerzas no conservativas.

La matriz D representa la matriz de inercia de las articulaciones, además en este caso en que se tienen motores con grandes reducciones, es importante incluir las inercias de los motores. Estas inercias son especificadas por una matriz diagonal 4 x 4.

$$I = diag(I_1 r_1^2,..,I_4^2),$$

donde I_i y r_i son la inercia del motor y el factor de reducción respectivamente de la juntura *i*. Definiendo la matriz $\mathcal{M}(q) = \mathcal{D}(q) + I$ modificamos la dinámica con este termino adicional:

$$M(q)\ddot{q} + C(q,\dot{q}) + g(q) = \tau$$

IV.4 Dinámica del actuador

Como las articulaciones son accionadas por motores de corriente continua (DC) de imán permanente, haremos un breve análisis de la dinámica de estos actuadores.

$$L\frac{di}{dt} + Ri = V - K_b\dot{q}$$

donde *i, V* son vectores representando las corrientes de armadura y voltaje respectivamente y L,R,K_b son matrices, las cuales representan inductancias de

armadura, resistencia de armadura y la constante de *fem* inversa (multiplicado por el factor de reducción).
Como el torque del motor en una articulación (τ) y la corriente de armadura (i) están relacionados por: $\tau = K_m\, i$, donde K_m es la constante del motor, podemos describir el sistema completo de la siguiente manera:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = K_m i$$
$$L\frac{di}{dt} + Ri = V - K_b \dot{q}$$

IV.5 Controlador

A la luz de la complejidad de la dinámica de un sistema robótico de varias articulaciones, es remarcable descubrir que leyes relativamente simples de control pueden ser aplicadas en la mayoría de los casos [23].

Cabe destacar que estamos en presencia de un sistema regido por vínculos holónomos, es decir que a partir de la posición instantánea de cada una de las articulaciones puede ser determinada la configuración exacta del manipulador. Esto es una diferencia con el caso de Sulla, en el cual el vínculo que existe entre la robota y las ruedas es un vínculo no-holónomo o integrable. Esto se debe a que se puede hallar un vínculo entre sus velocidades, pero no una relación instantánea entre la posición de la robota y la posición angular de las ruedas.

Luego de estudiar diversos papers sobre control de manipuladores robóticos [7][17][21][22][23], llegamos a la conclusión de que en general, estos sistemas pueden llegar a controlarse satisfactoriamente con esquemas de control como el implementado por nosotros, utilizando bloques con controladores PID [13] para cada actuador (motor) [7].

Utilizando un control realimentado PID individualmente para cada uno de los actuadores y mediante un sistema de inteligencia superior que los coordine y supervise, se pueden lograr resultados más que satisfactorios en control de movimiento del manipulador [17]. Otro punto a favor de la utilización de este tipo de controlador es la relativa fácil implementación del mismo, y que es una solución muy estudiada y ampliamente implementada en el mercado ya sea en versiones de hardware o software. Nosotros hemos optado por una solución mixta entre controladores implementados en hardware, como ser los microcontroladores *LM629*, y controladores de software, como ser las librerías de control del *Rabbit 3000*.

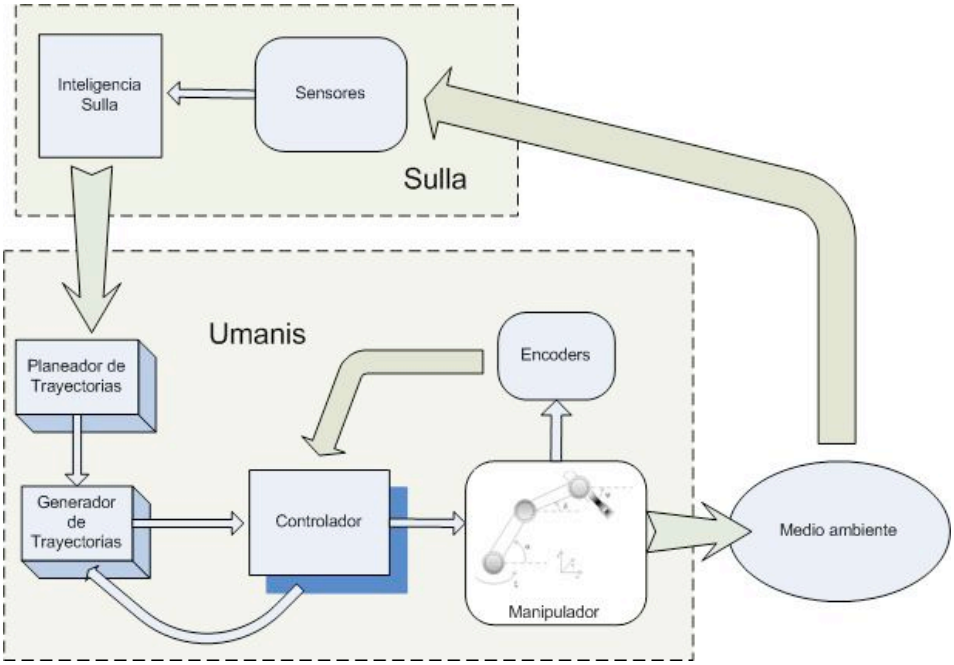


Fig. IV.2 - Diagrama de bloques del sistema de control.

Como se puede apreciar en la figura, de lo general a lo particular encontramos tres elementos: Umanis, Sulla y medio ambiente. La encargada de estimular el sistema de control de Umanis, es la inteligencia de Sulla. Por medio de Sinapsis serán enviados comandos los cuales serán recepcionados por el planeador de trayectorias. El planeador de trayectorias invoca al generador de trayectorias el cual inyecta (mediante la cooperación del sistema operativo) con una cadencia determinada elementos de una matriz de trayectorias en el bloque de Controlador.

El bloque Controlador es realimentado por los datos de los encoders y comanda el manipulador a partir de los datos de referencia y la posición real obtenida. Como ya hemos mencionado, el algoritmo de control dentro de este bloque es un PID sintonizado a cada actuador. Este bloque realiza sus rutinas a una cadencia significativamente mayor a la cadencia con la que recibe datos del bloque de generación de trayectorias.

Una capacidad nada trivial que posee este sistema de control, es que el propio bloque Controlador es capaz de dialogar con el bloque Generador de Trayectorias para informarlo de que es necesario crear una trayectoria de ajuste en tiempo real, por lo que, en el caso de que la configuración del sistema esté tomando valores alejados de los deseados para una trayectoria original este será capaz de tomar una acción correctiva que lo devuelva a la configuración deseada.

La estrategia de control adoptada para generar las trayectorias a partir de cada dato inyectado fue controlar a los motores en posición. De modo que la posición angular de cada motor sea especificada para un intervalo de tiempo dado. La suma de los aportes de cada actuador determina en el manipulador una configuración dada, con cierta coherencia temporal. Si elegimos una cadencia correcta de inyección de datos, más allá de que estemos controlando posiciones discretas, se llega a un comportamiento suave. Aquí se presenta un compromiso entre la suavidad del movimiento y el tiempo de respuesta de los actuadores.

V Desarrollo de Software

V.1 Generalidades

En este capítulo se describen los detalles de programación de Umanis. El SW de Umanis fue escrito en el lenguaje de programación Dynamic C 7.32, el cual fue desarrollado por Rabbit Semiconductors, para programar su línea de microprocesadores. Este lenguaje es un derivado del lenguaje de programación C. En nuestro caso, al igual que en Sulla, utilizamos la placa RCM3200 de Zworld, la cual contiene un microprocesador R3000. Su arquitectura es muy similar a la del clásico Z80. Esta placa no sólo incluye el microprocesador, sino que también incluye una serie de periféricos de especial interés para nosotros, como ser: decodificadores de encoders, generadores de señal PWM, varios puertos serie, relojes de tiempo real, etc. Como características particulares del RCM3200 podemos destacar lo siguiente:

- Posee Interfaz Ethernet
- 512K memoria flash + 512K memoria RAM
- 52 I/O digitales
- Reloj de 44.2 MHz

V.2 Introducción, Dynamic C

El sistema de desarrollo de Rabbit permite, mediante una interfaz serial, compilar programas y cargarlos en memoria. También posee herramientas útiles de debugging como ser el llamado Watchdog, que permite conocer valores de variables en tiempo de ejecución, y el valor de los registros del micro. Otra herramienta que resulta interesante es la capacidad de utilizar como stdio (conocido en lenguaje C como entrada/salida en pantalla) la pantalla del PC en donde se está desarrollando el SW. También vale la pena mencionar, que Dynamic C permite compilar programas sin necesidad de tener conectado el micro a la PC (targetless compile), esto es de suma importancia, ya que permite a varios desarrolladores trabajar en paralelo corrigiendo errores de compilación sin la necesidad de tener el Hardware en ese momento.

Pero sin duda, la mayor ventaja de utilizar este entorno de programación, y por ende el RCM3200, es la cantidad de código que existe implementado en forma de bibliotecas, pensado para ser utilizado en sistemas embebidos, y además de eso, existen infinidad de ejemplos en cada área. Como ejemplo de esto podemos mencionar las librerías de uC/OS-II ucos2.lib, las librerías de control de motores servo.lib, las librerías de TCP/IP dcrtcp.lib, las librerías de manejo de bus I2C i2c.lib, etc. Esto permite ahorrar considerablemente tiempo de desarrollo.

Por información detallada del entorno de programación y la placa utilizada, se recomienda la lectura de los siguientes manuales:

- Rabbit 3000 Microprocessor User's Manual [19]
- RabbitCore RCM3200 User's Manual [20]
- Dynamic C User's Manual [9]
- Dynamic C TCP/IP User's Manual [8]

Todos estos manuales pueden ser descargados gratuitamente en la página de Zworld (www.zworld.com).

Podemos encontrar que en este lenguaje es permitida la modularización del SW en varios archivos llamados librerías o bibliotecas (extensión .lib), al igual que en el código C, pero a diferencia del mismo, sólo podrá haber un archivo con extensión ".c" en el cual se encontrará el "main" del programa principal. En nuestro caso ese archivo se llama Umanis-main.c. Otra diferencia a destacar con el Lenguaje de programación C

tradicional es que no existen los archivos de header, con extensión ".h", sino que se encuentra tanto el prototipo de las funciones como la implementación de las mismas dentro de los archivos de librería.

La manera de indicarle al compilador que utilice el código de una librería en un archivo externo es la siguiente:

```
#use "biblioteca.lib"
```

Usualmente esta directiva es vista como una mala práctica en la programación de Lenguaje C, recomendándose la directiva "#include", dando el argumento de que la directiva "#use" no es portable, pero en Dynamic C sólo está permitida esta última.

Para que Dynamic C efectivamente sea capaz de utilizar el código externo, debe seguirse un protocolo de comentado de secciones en las librerías, y se debe agregar la ruta a las mismas en el archivo "lib.dir". Estas indicaciones están explayadas ampliamente en la sección 4.21 de Dynamic C User's Manual.

V.3 Arquitectura de Software

Ahora que discutimos como se estructura el SW en Dynamic C, estamos en condiciones de analizar la arquitectura del SW de Umanis. Todo comienza con el archivo "Umanis-main.c", el cual inicializa el Sistema Operativo y realiza el anidamiento de librerías. Para ilustrar la arquitectura y el front-end de Umanis, se encuentra la siguiente figura.

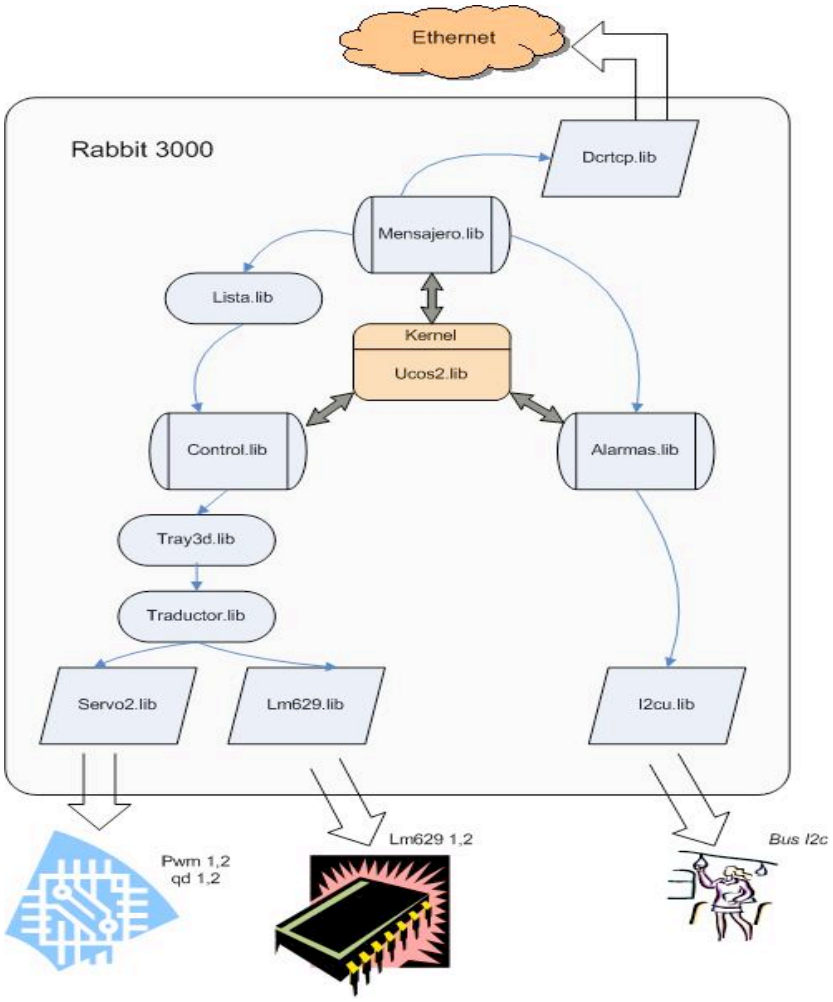


Fig. V.1 – Arquitectura de Software

Comencemos por los rasgos generales. En el centro del diagrama encontramos al Kernel o Sistema Operativo. Este es responsable de generar y administrar tres Tareas:

- Una tarea de Control, implementada en Control.lib.
- Una tarea de Mensajería, implementada en Mensajero.lib.
- Una tarea de gestión de Alarmas, implementada en Alarmas.lib

Como se puede apreciar, tanto el Kernel como las tareas, tienen íconos característicos que las diferencian de los demás elementos, llámese, las librerías de interfaz (representadas por un paralelogramo) o las librerías internas (representadas por una figura ovalada). Otro elemento que podemos encontrar en este diagrama son las flechas que atraviesan la interfaz, y se dirigen hacia los "periféricos" externos al microprocesador con los que querremos comunicarnos. Estos son:

- Una interfaz Ethernet, mediante la cual recibiremos comandos mediante paquetes UDP, en este caso de Sulla. Cabe destacar que podrían ser paquetes UDP generados por otras entidades, siempre y cuando cumplan con la especificación de la estructura Sinapsis.
- Generadores de PWM y decodificadores de Encoders, estos son periféricos de la placa RC3200 encargados de generar dos ondas PWM independientes y leer los dos encoders correspondientes.
- Interfaz con 2 LM629, recordemos que 2 motores se controlan con los periféricos del RCM3200 y dos se controlan con los microprocesadores LM629.
- Bus I2c, el cual es utilizado como un expansor de entrada/salida para hacer la gestión de las alarmas.

Luego tenemos los módulos de SW que se encuentran dentro de etiquetas ovaladas, los cuales generan sub-capas entre los módulos principales y las interfaces.

V.3.1 ¿Cómo funciona?

Para explicar como funciona este sistema de SW, hagamos una recorrida mental de todos los procesos que suceden desde que es ejecutado un comando de acción, hasta llegar a la consecución de su objetivo.

Luego que el sistema es inicializado, se pone en espera a recibir un comando vía un paquete UDP. Una vez que el módulo mensajero.lib recibe un comando válido, lo decodifica, y procede a encolarlo en el módulo lista.lib. Si la tarea de control, implementada en control.lib se encuentra en estado IDLE (desocupada), va a buscar el próximo comando a la lista. Cabe destacar que en la lista, los comandos se encolan en formato FIFO (el primero en entrar, es el primero en salir). Supongamos que este comando es una trayectoria que involucra el movimiento de todas las articulaciones del manipulador, entonces, lo primero que deberá hacer el módulo de control, es generar la trayectoria implicada en el comando, de esa tarea se encargará la librería tray3d.lib (por más información sobre cómo es generada la trayectoria, ver la sección de Generador de Trayectorias).

Inmediatamente esa información deberá ser traducida en parámetros entendibles por las interfaces de bajo nivel que ejecutan el movimiento, y de esa "traducción", es la que se encarga el módulo traductor.lib. Luego que sucedió todo esto ingresamos el vector resultante en el loop de control, el cual se encargará de ejecutar los comandos de control de posición de manera sincrónica para todos los motores involucrados, e inyectarlos con la cadencia adecuada.

Mediante la ayuda del SO, podemos, a la misma vez a que todo esto sucede, estar consultando mediante la tarea de gestión de alarmas llamada alarmas.lib que no se

haya dado ningún comportamiento anormal que implique el daño del manipulador, y también continuar a la espera de la recepción de nuevos comandos.

Esta es una explicación somera del funcionamiento del SW, que pretende ilustrar en líneas generales el flujo de la información.

Es importante tener presente, que dado que Umanis es capaz de recibir comandos vía paquetes UDP, en particular podría ser conectado a Internet para ser comandado a distancia, si a esto le sumamos la posibilidad de ubicar una cámara Web en el lugar, se nos abren las grandes posibilidades del comando a distancia o la automatización remota.

En caso de que el lector desee profundizar en la implementación de estos procesos, se agrega como apéndice a esta documentación el código completo de Umanis (disponible en CD adjunto a la versión de biblioteca).

V.4 El planeador de Trayectorias de Umanis

V.4.1 ¿Qué es ? y ¿Qué hace?

El planeador de trayectorias de Umanis es el módulo de Software (más precisamente la librería tray3d.lib) encargado de generar vectores de trayectorias expresados en las coordenadas naturales del sistema para recorrer una trayectoria dada. Éste es un módulo crítico ya que será el que le permitirá al manipulador realizar movimientos coordinados.

La filosofía que adoptamos para la creación de este módulo fue la siguiente: Atacar el problema de lo general a lo particular; por lo que el generador de trayectorias es capaz de generar trayectorias en línea recta entre dos puntos cualesquiera del espacio, siempre y cuándo estos pertenezcan al espacio de trabajo del manipulador (esto depende del punto de montaje en Sulla). Estas trayectorias pueden tener una cantidad de muestras arbitraria. Luego con la composición de pequeñas trayectorias en línea recta, se pueden generar trayectorias más complejas.

Para ilustrar de mejor manera el funcionamiento de éste módulo se hizo una simulación en Matlab de la trayectoria seguida por el eje de la muñeca del manipulador, pero enfatizamos que el generador es capaz de crear trayectorias para todos los grados de libertad de Umanis (por más información ver "El Sistema de coordenadas de Umanis").

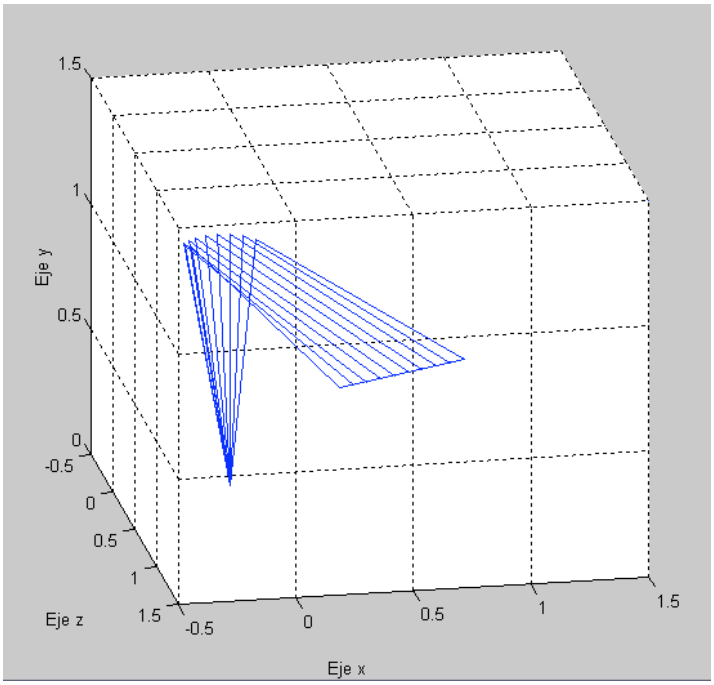


Fig. V.2 – Simulación 3D de trayectoria en línea recta

En esta figura el manipulador sigue una trayectoria en línea recta entre dos puntos del espacio con una resolución de 10 muestras. Cabe destacar que cuánto mayor sea la resolución de la trayectoria, ésta más se parecerá a una ideal, porque tengamos en cuenta que para un sistema de coordenadas polares una trayectoria en línea recta es algo bastante anti-natural. Para mejor apreciación de la figura se dejaron fijados en pantalla las configuraciones del manipulador en cada muestra. Los andamientos de los ángulos respectivos son los siguientes:

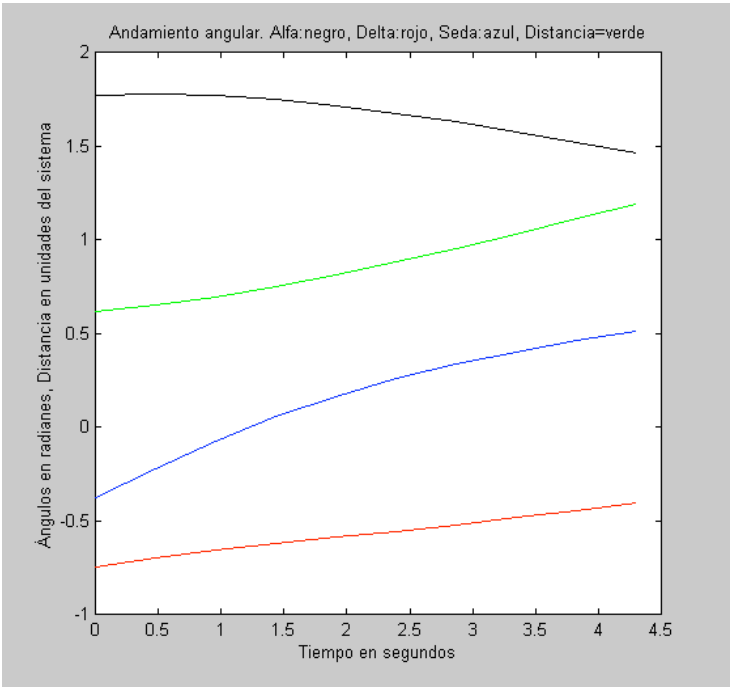


Fig. V.3 – Andamientos angulares simulados

Como decíamos anteriormente, se pueden componer las trayectorias para generar circuitos más complejos. Por ejemplo:

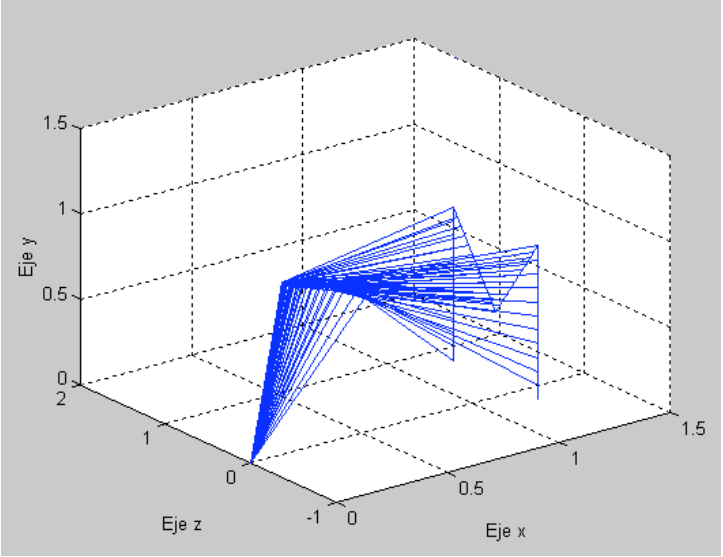


Fig. V.4 – Trayectoria compuesta para generar la letra "M"

En esta figura el manipulador "escribe" la letra M en un plano paralelo al frente de trabajo.

Cabe hacerse la siguiente pregunta: ¿Con qué fidelidad seguirá nuestro manipulador esta trayectoria?. Bueno, la respuesta excede al alcance del generador de trayectorias. Éste es sólo capaz de dar la configuración que debería tener el manipulador a nivel de coordenadas en un instante dado de tiempo. La respuesta a dicha pregunta dependerá de la eficacia del algoritmo de control de cada motor y de la capacidad del sistema para manejar movimientos simultáneos, por lo que involucrará al sistema operativo de Umanis y su capacidad de gestión.

V.4.2 Análisis Matemático

Dados dos puntos en el espacio de trabajo del manipulador expresados en coordenadas cartesianas, se genera una cantidad arbitraria de puntos equidistantes. La consigna es realizar la llamada *"cinemática inversa"* para hallar la configuración del manipulador que tenga como consecuencia el posicionamiento del eje de la muñeca en los puntos dados y que además sea *"continua"*, porque como veremos la gran dificultad de la cinemática inversa es la multiplicidad de soluciones encontradas para un mismo punto dado en el espacio.

Para dar un ejemplo sencillo de esto planteemos el problema en dos dimensiones, por lo que hallaremos 2-uplas (α,δ) que generen una posición (a,b) en el extremo del manipulador.

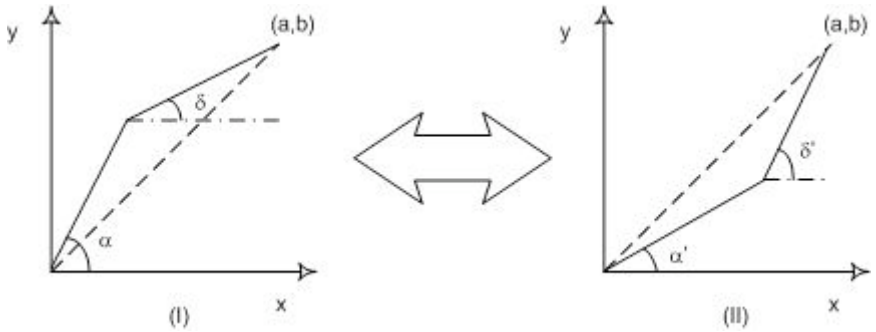


Fig. V.5 – Dificultades de la cinemática inversa

Como se puede apreciar en la figura, para este caso simple, dada una posición final del actuador (a,b), encontramos dos configuraciones inversas (α,δ) y (α',δ') que cumplen con la especificación. A lo que viene el comentario de la "*continuidad*", es a que, para una misma trayectoria no podemos adoptar configuraciones de tipo (I) para un punto, y configuraciones del tipo (II) para el siguiente.

Para solucionar este problema se adoptaron soluciones del tipo de configuración (I) para todas las trayectorias, o como las llamamos nosotros, configuración "*grúa*" (en la jerga de la robótica es llamada "*elbow-up configuration*"). Esta elección es más que conveniente para el aspecto mecánico, ya que si se mantiene la configuración "*grúa*" para todos los puntos de la trayectoria el torque resistivo en la articulación siempre tendrá el mismo sentido, por lo que se evita el juego en dicha articulación.

Dicho esto, pasemos a la solución general del problema en tres dimensiones. Dado un punto A=(x,y,z), perteneciente al espacio de trabajo del manipulador, encontremos la configuración en coordenadas naturales del sistema determinada, por la 3-upla (α,δ,ξ), tal que posicione el eje de la muñeca en el punto A.

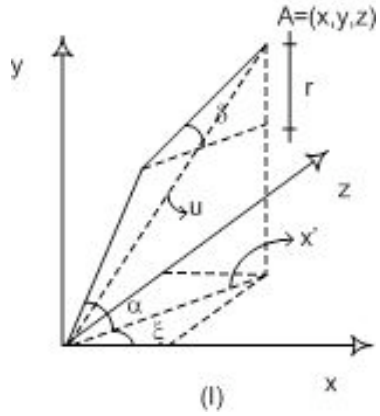


Fig. V.6 – Caso tridimensional

Consideramos los dos segmentos de igual tamaño y de largo *a*.

Entonces:

$$\xi = \tan^{-1}\left(\frac{z}{x}\right)$$

$$u = \sqrt{x'^2 + y^2}$$

Ahora calculamos α utilizando el segmento auxiliar x':

$$\alpha = \cos^{-1}\left(\frac{\frac{u}{2}}{a}\right) + \tan^{-1}\left(\frac{y}{x'}\right)$$

Calculemos δ utilizando el segmento auxiliar *r*:

$$\partial = sen^{-1}\left(\frac{r}{a}\right)$$

Hallemos el valor de los segmentos auxiliares *x'* y *r*:

$$x' = \sqrt{x^2 + z^2} \qquad r = y - a.sen(\alpha)$$

Así llegamos a las expresiones finales:

$$\alpha = \cos^{-1}\left[\frac{\sqrt{(\sqrt{x^2 + y^2}) + y^2}}{2a}\right] + \tan^{-1}\left(\frac{y}{\sqrt{x^2 + z^2}}\right)$$

$$\partial = sen^{-1}\left(\frac{y - a.sen(\alpha)}{a}\right)$$

V.4.3 ¿Cómo funciona?

Como sabemos, tanto Umanis como Sulla, tienen la finalidad de oficiar como plataformas de estudio de robótica y inteligencia artificial, por lo que resulta importante para nosotros dar información que vaya más allá de lo descriptivo, creemos que es importante dar detalles sobre cómo implementamos cada funcionalidad nivel técnico, para que esta documentación resulte rica ante los ojos de los futuros desarrolladores de robótica de nuestra Universidad. Se podría decir que queremos que Umanis llegue a ser un Robot "Open Source". Como fue indicado anteriormente, todo el código de nuestro proyecto oficia como apéndice de esta documentación. Pero también creemos que no sólo eso es suficiente, a veces es necesario que el propio desarrollador comente la idea general o finalidad del sistema de SW, por lo que en esta sección daremos una recorrida por los detalles de implementación de este módulo.

Primero, digamos que podemos considerar una trayectoria de nuestro sistema a un conjunto de "configuraciones" del mismo expresadas en sus coordenadas naturales. Entonces nos encontraremos ante la necesidad de crear una Estructura de datos que ampare estas configuraciones. Por ejemplo, nosotros decidimos implementarlo de la siguiente manera:

```
typedef struct {
        float alfa;
        float delta;
        float seda;
        float psi;
        int pinza;
```

```
} corn;
```

De modo que fue creado el tipo de datos corn (a saber, "cor", acrónimo de coordenada y "n" hace referencia a natural, por ser las coordenadas del sistema). Además, si examinamos el punto de partida de nuestro generador: dos puntos en el espacio, también nos vimos en la necesidad de una estructura de datos que ampare el sistema de coordenadas cartesianas.

```
typedef struct {
        float x;
        float y;
        float z;
} cor;
```

En este momento nos encontramos en condiciones de redefinir nuestro problema en términos más cercanos al desarrollo. Entonces la nueva consigna sería: Dados dos puntos en el espacio definidos por variables del tipo de datos cor, encontrar la trayectoria en línea recta con un número arbitrario de muestras (puntos de esa trayectoria), expresada en datos de tipo corn (coordenadas naturales del sistema).

Este problema fue resuelto en varias etapas. Primero definimos estructuras de datos para las trayectorias, a saber : tray_xyz y tray_nat. Estas estructuras no son más que arrays de los tipos de datos definidos anteriormente de tamaño TRAY_MAX_CANT_ELEM (esta constante está definida en la librería de constantes consantes.lib y puede ser manipulado a gusto por el futuro desarrollador).

Luego creamos una función que es capaz de generar la trayectoria en línea recta entre dos puntos en coordenadas cartesianas. El encabezado es el siguiente:

```
void tray3d_xyz(tray_xyz* traye, cor inicio, cor final, int muestra);
```

Como se puede observar, además de los puntos inicial y final se indican la cantidad de muestras que serán generadas y, un detalle que no es para nada menor, es que recibe un puntero a una estructura de tipo tray_xyz. Esto significa que el valor de traye en pasado por referencia. Recordemos que el tipo de datos tray_xyz es a su vez un array de otro tipo de datos, por lo que es de esperarse que su tamaño sea considerable. ¿A qué viene este análisis?, a lo siguiente, el compilador de C en particular Dynamic C, si recibe un parámetro por valor en vez de por referencia, hace una copia del valor de la variable en memoria para trabajar con él, por lo que estaríamos duplicando la memoria usada, la cuál es un recurso muy valioso en Microprocesadores dedicados. De modo que el desarrollador antes de llamar a esta función debe crear una variable de tipo tray_xyz, sobre la cual operará esta función (nota: recordemos que habrá que "desreferenciarla" con el símbolo & para introducirla como parámetro de la función).

De modo que en este momento nos encontraríamos en poder de una trayectoria expresada en coordenadas cartesianas, y mediante el uso de la función tray3d_nat, será expresada en coordenadas naturales, por lo que habremos alcanzado nuestro objetivo.

```
void tray3d_nat(tray_xyz* traye, tray_nat* traye_n)
```

Nuevamente se aprecia en el encabezado de esta función que toma punteros y opera sobre los mismos, ya que devuelve void [18].

V.5 El Sistema Operativo

V.5.1 Introducción

Antes de introducirnos en los detalles y la implementación del Sistema Operativo de Umanis comencemos por definir lo que un sistema operativo es. Existen variadas definiciones, éstas son algunas de ellas:

- Un sistema operativo (SO) es un conjunto de programas o software destinado a permitir la comunicación del usuario con un ordenador y gestionar sus recursos de manera cómoda y eficiente. Comienza a trabajar cuando se enciende el ordenador, y gestiona el hardware de la máquina desde los niveles más básicos.
  Ref.:
  es.wikipedia.org/wiki/Sistema_operativo
- Conjunto de software que controla los distintos recursos del ordenador. En inglés: «operating-system».
  Ref.:
  www.linux-es.org/Faq/Files/Html/FAQ_Linux_V2.0.2-145.html
- Es un programa que se ejecuta ininterrumpidamente en un segundo plano de la computadora y que permite básicamente trabajar con el sistema.
  Ref.:
  www.ututo.org/slw/HTML/suse/go01.html
- Software que controla la ejecución de programas de cómputo y que proporciona el control de entrada-salida, la administración de los datos, la asignación de áreas de almacenamiento y los servicios relacionados a la utilización de la computadora.
  Ref.:
  sgp.cna.gob.mx/Publico/Diccionarios/Glosario.htm

De modo que podríamos decir esquemáticamente que un Sistema Operativo (de aquí en adelante SO) es un Software que de alguna manera gerencia recursos de un computador y los administra entre aplicaciones. Este concepto, puede ser heredado hacia el tipo de computador existente en Umanis, un microprocesador dedicado, o como se le llama en la jerga, un sistema embebido. Esta denominación se debe a que el usuario del sistema en cuestión (en este caso un manipulador robótico) podría no llegar a percibir explícitamente la presencia de este microcontrolador, este existe para la consecución de un objetivo de orden mayor, como en este caso, dar inteligencia y controlar al manipulador.

Dicho esto, adentrémonos un poco más en el mundo de los SO, comentemos someramente algunos de sus sabores y características.

- SO Multitarea (MultiTask):
  Son capaces de generar la percepción en el usuario de ejecutar varias tareas simultáneamente. A su vez existen subtipos de los mismos:
  - Cooperative Multitasking:
    Las tareas ceden voluntariamente el control del microprocesador, implementado una máquina de estados en un loop infinito. En el caso de Dynamic C, este tipo de multitarea puede ser incluso implementada en ausencia de un SO (funciones costate).
  - Preemptive Multitasking:
    Las tareas hacen uso de los recursos del microprocesador organizándose en base a prioridades y/o repartiendo el

tiempo en ejecución en intervalos dados. Existen dos maneras de implementar esto en Dynamic C, una es usando UC/OS-II (será descrito posteriormente) y otra es usando declaraciones de tipo slice.

- SO de Tiempo Real (RTOS).
  Son capaces de garantizar con cierta precisión los tiempos máximos en los que el sistema responde a eventos.

V.5.2 UC/OS-II: El Sistema Operativo elegido para Umanis

Luego de investigar las diferentes opciones, decidimos utilizar uC/OS-II. Este sistema operativo fue concebido especialmente para trabajar con sistemas embebidos, y posee ciertas características especialmente útiles para sistemas de control robóticos. A saber:

- Permite implementar Preemptive Multitasking.
- Es un SO de Tiempo Real.

Estas dos características son fundamentales, porque pensemos que además de necesitar la capacidad de ejecutar varias tareas simultáneamente (controlar motores, hacer polling de alarmas, manejar las comunicaciones TCP-IP), lo que desde el punto funcional se podría hacer perfectamente en un entorno cooperativo, debemos ser capaces de manejar un sistema de prioridades bien definido, ya que el punto fundamental de nuestro sistema es controlar de manera eficiente los distintos grados de libertad de nuestro manipulador, y para ello está diseñado el Preemptive Multitasking. Por otro lado debemos ser capaces de garantizar con la mayor precisión posible los intervalos de ejecución de nuestros algoritmos de control, por lo que es necesaria la característica de tiempo real.

Mucho puede ser hecho con este SO, nosotros utilizamos sus prestaciones de manera básica para poder implementar la automatización requerida. El SO uC/OS-II dispone de colas, mensajes, banderas y semáforos que permiten enviar mensajes entre tareas y compartir recursos. Nosotros sólo usamos, al igual que Sulla, las capacidades de definir prioridades entre tareas y manejo de tiempo entre las mismas (OSTimeDly()).

Estas funcionalidades nos permiten, dentro de ciertas salvedades, programar cada tarea como si se dispusiera del microprocesador solamente para ella. Es decir, por ejemplo, esto nos permitió desarrollar de manera totalmente independiente las rutinas de control de las rutinas de comunicación y alarmas.

De modo que estamos en condiciones de decir que con este SO, seremos capaces de crear "tareas" que nos den la percepción desde un entorno macro de que el sistema es capaz de realizar varias acciones simultáneamente.

Sin entrar en detalles exhaustivos, ya que nos alejaríamos demasiado del foco de esta documentación, investiguemos un poco más cómo es creada esta ficción. Pensamos que es importante dar nociones generales de cómo es este proceso ya que el entendimiento de esta arquitectura, facilitará el escalamiento de este sistema por futuros desarrolladores que utilicen esta plataforma. Esta manera de pensar las tareas, permite de fácil manera agregar nuevas tareas o funcionalidades al sistema.

En la figura siguiente se muestra un diagrama de estados. Este representa los cinco posibles estados en los que se puede encontrar una tarea, y los métodos que permiten cambiar de estado. Notemos rápidamente que en este escenario debemos permitir el manejo de un arma fundamental para los microprocesadores, que es la atención a interrupciones, o sea la capacidad de interrumpir el proceso actual en ejecución y

atender a una tarea frente a una estimulación asíncrona. Esto no es para nada trivial, ya que el SO en sí mismo maneja interrupciones para mantenerse funcional.

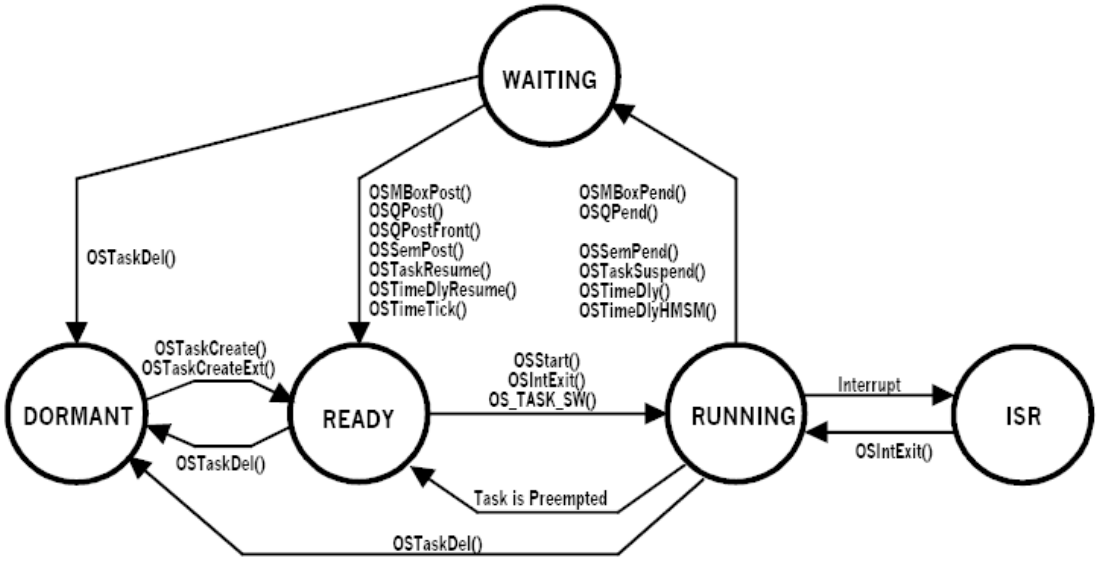


Fig. V.7 – Diagrama de estados de tareas en UC/OS-II [15]

Entonces, solamente una tarea estará en el estado "running", y esta será la que haya estado en estado "ready" y tenga el mayor nivel de prioridad.

También existen métodos especiales para violar este esquema, ya que existen instancias dónde sería especialmente riesgoso entregar el control del microprocesador a otra tarea, como por ejemplo cuando se está escribiendo un registro en un microprocesador aledaño, o cuando se atiende a una alarma del sistema. Para ese fin existen las instrucciones de tipo OS_ENTER_CRITICAL() y OS_EXIT_CRITICAL().

A modo de finalización citemos al creador de este sistema operativo, a saber Jean J. Labrosse, el cual describe en profundidad todos los detalles y prestaciones de este SO en su libro "MicroC/OS-II The Real Time Kernel" [15]. Este texto es muy recomendado ya que no sólo describe este SW sino que también habla de la historia de los SO en general y de los diferentes percances en la historia de la creación de los mismos.

V.5.3 Configuración de tareas y diseño de SO

Es tiempo de entrar en el detalle de la implementación del sistema de tareas de Umanis. Básicamente podremos definir tres tareas que deberán ejecutarse "simultáneamente" por el microprocesador, estás serán:

- Una tarea de control de motores (task_control)
- Una tarea de gestión de alarmas (task_alarmas)
- Una tarea de mensajería (task_mensajero)

Confiando en que la descripción funcional de dichas tareas haya sido suficientemente desarrollada en las secciones correspondientes, desarrollemos los aspectos concernientes al SO de dichas tareas.

Como es de esperarse, la tarea que será de mayor prioridad para nosotros es la tarea de control de motores, ya que esta determina la exactitud y sincronicidad de los

movimientos del manipulador. Luego con prioridad dos, actuará la tarea de gestión del sistema de alarmas, la cual será capaz de interrumpir momentáneamente la ejecución del SO haciendo uso de las instrucciones de orden crítico (descriptas anteriormente). Por último, pero no menos importante, con prioridad tres encontramos a la tarea de mensajería, la cuál es la encargada de comunicar a Umanis con el mundo exterior así como entregar mensajes de comandos al sistema de control. En la figura siguiente se puede apreciar un esquema de esta distribución.

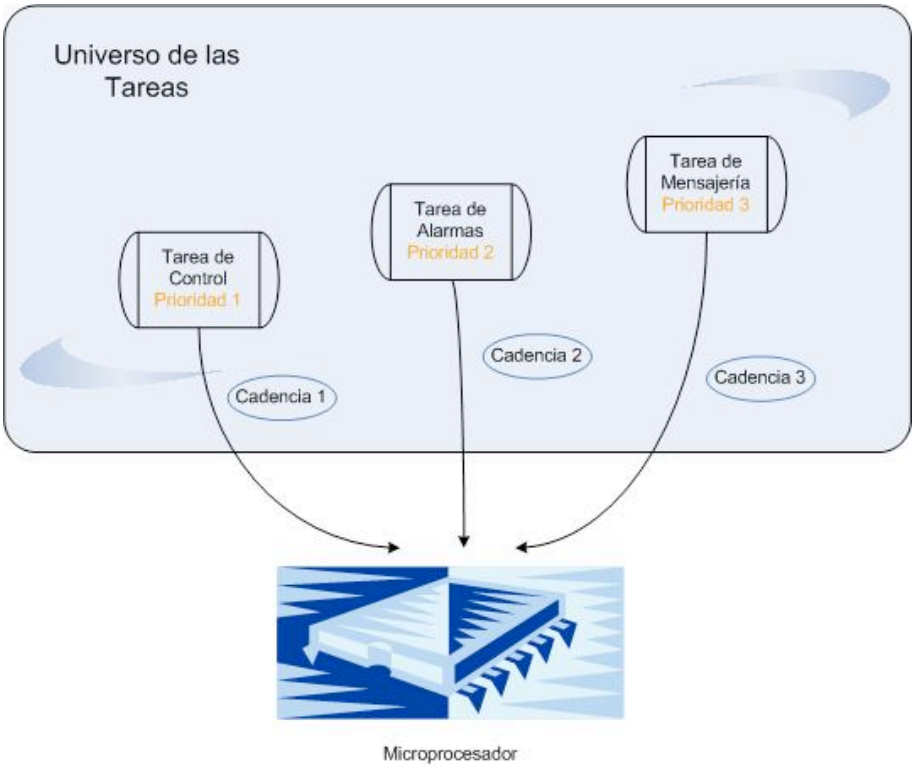


Fig. V.8 – Esquema de tareas de Sistema Operativo

Como podemos observar, en este paradigma, existen básicamente dos elementos en la obtención del microprocesador. El primero, es la prioridad de la tarea, y el segundo, , es la cadencia (configurable) en la que la tarea estará pronta para ser ejecutada. Esto es genial desde el punto de vista que podemos perfectamente agregar otra tarea, con una prioridad perfectamente configurable, y con la cadencia que queramos. Además de esto, por ejemplo, basta con cambiar el parámetro de cadencia de la tarea de control en el SO para alterar la frecuencia con la que se ejecuta nuestro algoritmo de control (delay de espera), y por lo tanto los comandos a los motores!!. Este esquema es cabalmente escalable, y nos permite agregar funcionalidades a gusto sin perjudicar de ninguna manera nuestra tarea de mayor interés, que es la tarea de control, ya que esta es la de mayor prioridad.

V.6 Descripción de Bibliotecas

Esta sección está dedicada a desarrollar desde un aspecto más técnico el sistema de librerías de Umanis.

Para comenzar damos un esquemático donde se detallan tanto las bibliotecas como las tareas. Esto se hace desde el punto de vista la tarea que desarrollan en un esquema de capas.

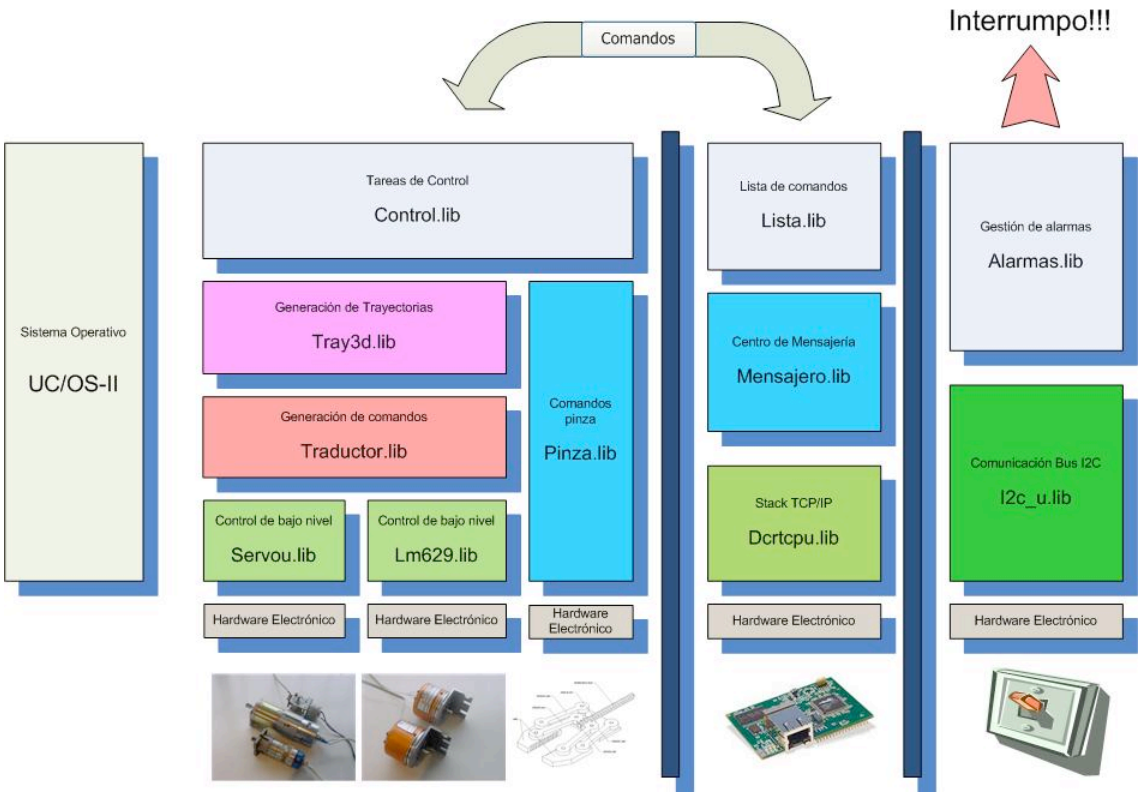


Fig. V.9 – Sistema de librerías de Umanis

Comencemos con las librerías referentes a las tareas de Control:

- Servou.lib:
  - Control PID de bajo nivel para motores de hombro y codo.
  - Creada a partir de la librería servo.lib (proveniente del set de librerías de RCM3200).
  - Fue utilizada para controlar motores de CC en posición. Las funciones utilizadas reciben dentro de otros parámetros la posición absoluta a alcanzar con el eje del motor expresado en cuentas del encoder correspondiente.
- Lm629.lib:
  - Implementa el protocolo de comunicación y comando para los microprocesadores LM629. Esta librería se creó en base a las especificaciones del fabricante.
- Pinza.lib:
  - Escribe bits en el puerto destinado al control de la pinza. Como ya fue mencionado anteriormente el control de la pinza es un control de torque y se implementa integramente en HW.
- Traductor.lib:

    - Librería encargada de transformar los ángulos para cada articulación expresados en radianes a comandos factibles para cada controlador. También hace correciones sobre referencias a ángulos relativo y corrige el efecto "coder".
- Tray3d.lib:
    - Biblioteca encargada de generar trayectorias en línea recta en el espacio a partir de un punto inicial, un punto final y una cantidad de muestras. Su salida es vector de coordenadas naturales al manipulador que representa la configuración del mismo para cada instante de tiempo en puntos equiespaciados en la trayectoria deseada.
- Control.lib:
    - Librería que oficia como comandante o estímulo de las tareas de más bajo nivel. Aquí encontramos las primitivas básicas y rutinas de calibración entro otras. También es la responsable ejecutar los comandos encolados.

Prosigamos con las bibilotecas referentes a la mensajería:

- Dcrtcpu.lib:
    - Librería desarrollada a partir de dcrtcp.lib proveniente del sistema de librerías de Dynamic C para RCM3200. Encargada de crear el stack de protocolos TCP/IP.
- Mensajero.lib:
    - Librería destinada a recibir y decodificar los paquetes provenientes en protocolo Sinapsis. Luego de decodificados los comandos, éstos serán encolados.
- Lista.lib:
    - Encargada del manejo de la cola de comandos.

Bibliotecas referentes a la gestión de alarmas:

- I2c_u.lib:
    - Derivada de i2c.lib. Encargada de la comunicación mediante el bus I2C.
- Alarmas.lib:
    - Gestión de alarmas.

Otros ficheros relevantes:

- Umanis-main.lib:
    - Main del complejo. Entre otros crea tareas e inicializa el Sistema Operativo.
- Const-ini.lib:
    - Librería que contiene la declaración de constantes de interés a utilizar en todos los módulos. Por ej.: cantidad de elementos de una trayectoria, cantidad de tareas, tamaño de stacks, constantes de conversión y corrección, etc.

V.7 Umanis como Font-End

Estamos en condiciones de expresar a Umanis como un sistema complejo, formado por diversos subsistemas que cumplen una función en conjunto. Pero sobre todo queremos unificar el concepto de nuestra plataforma como un *Front-end*. Esto implica que Umanis puede ser pensado en una manera abstracta como un *Front-end* que ofrece servicios aplicación mecánicos al mundo exterior. Esta manera de pensar Umanis nos llevó a diseñar una interfaz de comunicación con servicios puntuales a ser brindados. Pero antes de eso debió ser decidido el esquema de comunicación con los módulos de software del mundo exterior, lo que será discutido en la siguiente sub-sección de esta documentación.

Paradigma de comunicación

Como sabemos los esquemas de comunicación o interacción entre módulos computacionales pueden ser representados básicamente en tres esquemas conocidos:

- Maestro-Esclavo (MS)
- Cliente Servidor (CS)
- Peer to Peer (P2P)

En esta sección discutiremos la brevemente cada uno de estos paradigmas y justificaremos la elección de paradigma de interacción para Umanis.

Comencemos con el esquema de Maestro-Esclavo, en esta esquematización el único módulo que tiene la opción de iniciar diálogos, es el maestro, de modo que los esclavos deberán esperar pasivamente a ser interrogados por el Maestro, ya sea para solicitarles algún tipo de información o para incitarlos a realizar una acción determinada. Nosotros consideramos que esta no sería una elección adecuada para nuestra plataforma ya que nos implicaría una interrogación constante por parte del módulo al cual sirva Umanis (en esta caso a Sulla), lo que disminuiría el valor agregado a la plataforma, y de alguna manera repercutiría sobre el concepto de autonomía de nuestro sistema, a nuestro entender.

Prosigamos con el paradigma de Cliente-Servidor, en este esquema el módulo Servidor espera a ser interrogado por el/los clientes para brindarle sus prestaciones. Este es un esquema más rico a nuestro entender, ya que en él surge el concepto de conexión, o sea, los Clientes inician conexiones hacia el servidor, lo que implica un flujo bilateral de información. De modo que luego de introducir este concepto, se podría definir el paradigma Cliente-Servidor de la siguiente manera: el servidor de aplicación espera pasivamente a que los clientes inicien conexiones con él, siendo los clientes los únicos poseedores de esta capacidad (el servidor no tiene la capacidad de iniciar conexiones). Este paradigma es el más usado en el mundo de la informática, es el que usar por ejemplo los servidores de aplicación web que comúnmente encontramos en Internet.

El esquema que resta por analizar es el de Peer to Peer, es una nueva tendencia en los sistemas computacionales de uso cotidiano, como pueden ser las populares aplicaciones tipo Kazaa, Napster o eMule. En este esquema se establece una igualdad entre las facultades de los integrantes de la comunicación, de modo que ambas partes tienen la posibilidad de iniciar conexiones al mundo exterior. Se podría llegar a clasificar como un paradigma anárquico de interconexión.

¿Cuál fue nuestra elección?, nuestra elección fue el paradigma de Cliente-Servidor. Creemos que es el que mejor se adapta a nuestra plataforma. De modo que estaríamos definiendo a Umanis como un Servidor de Aplicación que interactuará con

diversos clientes (en esta caso Sulla). Este desarrollo nos deja en condiciones de introducir el Front-end de aplicación de nuestra plataforma servidora.

Especificación del Front-End

En el Front-End de Umanis están definidas las prestaciones que brindará nuestra plataforma. Cabe declarar que el diseño de la misma y su implementación es abierta, lo que permite, y lo que permitiría cambios y Upgrades.

Umanis posee una interfaz Ethernet, sobre la cual se implementó un servidor UDP escuchando en el puerto 9930. La IP configurada a dicha interfaz es 192.168.1.40, la cual pertenece a la misa clase "C" en la que se encuentran tanto la interfaz de red del Rabbit de Sulla, como la interfaz de red del Linux de Sulla. A efectos de debugging también fue implementado en Umanis un servicio que responde consultas ICMP (ping), para verificar que el servidor esté efectivamente levantado y que la interfaz funciona correctamente.

Cabe destacar que Umanis hace visible en su front-end las primitivas de bajo nivel, con el fin de permitir a futuros desarrolladores implementar nuevas aplicaciones complejas desde plataformas de más alto nivel, reutilizando las primitivas básicas, con lo que se conseguiría un buen desempeño, ya que estas fueron implementadas aprovechando las características de tiempo real obtenidas utilizando el microprocesador dedicado.

La forma de comandar a Umanis es enviándole mensajes con la sintaxis adecuada en el *paylod* de paquetes UDP. La sintaxis es la siguiente:

<tipo de comando>,<parámetro 1>,<parámetro 2> ,<parámetro 3>

- Tipo de comando: cuatro caracteres
- Parámetro 1,2,3: número real.

Observación importante: los parámetros deben ser separados por el caracter coma "," y para los tipos de comandos en los que no apliquen todos los parámetros, los sobrantes deberán ser rellenados con ceros. Un ejemplo de sintaxis de comandos sería el siguiente:

tray,22.8,5,1.5

Este *payload* UDP enviado a la IP 192.168.1.40 y puerto 9930 haría que el manipulador ejecute una trayectoria en línea recta hasta el punto x=22.8cm, y=5cm y z=1.5cm.El origen de coordenadas se encuentra en el centro del eje inferior del hombro. Por más información referente al posicionamiento, ver la sección sistema de coordenadas de Umanis.

Los servicios de acción están inicialmente discriminados por miembro del manipulador, y luego tomando el manipulador como un conjunto. Esto hace las veces de "arquitectura de capas", ya que a partir de las primitivas de cada miembro se construyen servicios más elaborados que implican la coordinación de más de un miembro, así como la consecución de objetivos de mayor complejidad.

A continuación haremos una descripción de los mismos.

Pinza:

- "pinz,<dirección>,<torque>,0"
  Cierra o abre la pinza con un torque determinado.
  - *dirección*: 0 cierra, 1 abre.
  - *torque*: 1,2,3. Torque mínimo, medio y máximo respectivamente.

Muñeca:

- "mune,<grados>,0,0"
  Mueve la muñeca una cantidad de grados dada.
  - *grados*: Un número positivo corresponde a sentido antihorario y uno negativo a horario.

Codo:

- "codo,<grados>,0,0"
  Mueve el codo una cantidad de grados dada.
  - *grados*: Un número positivo corresponde a sentido antihorario y uno negativo a horario.

Hombro:

- "homb,<grados>,0,0"
  Mueve el hombro una cantidad dada de grados.
  - *grados*: Un número positivo corresponde a sentido antihorario y uno negativo a horario.

Cintura

- "cint,<grados>,0,0"
  Mueve la cintura una cantidad de grados dada.
  - *grados*: Un número positivo corresponde a sentido antihorario y uno negativo a horario.

Servicios conjuntos (o de alto nivel):

- "cali,0,0,0"
  Implementa la calibración del manipulador. Cada articulación será llevada a una posición conocida para poder efectuar en posterioridad el seguimiento de trayectorias. **Importante**: luego de efectuada la calibración no se deben ejecutar primitivas de articulación si se quiere que el generador de trayectorias en línea recta funcione adecuadamente. Las única salvedad a esto son las articulaciones de la pinza y la muñeca.
- "tray,<x>,<y>,<z>"
  Ejecuta una trayectoria en línea recta para el centro del eje de la muñeca con posición final x,y,z, sin mover el motor de la muñeca.

- "tr_p,<x>,<y>,<z>"
  Ejecuta una trayectoria en línea recta para el centro del eje de la muñeca con posición final x,y,z, manteniendo la dirección original de la muñeca fija (por ej, horizontal).

- "halt,0,0,0"

  Deshabilita motores y cierra los servicios de Umanis hasta el próximo *Reset* (comando de apagado).

Está implementado en el Linux de Sulla un cliente que envía *payloads* genéricos a Umanis utilizando la estructura de mensajería de Sinapsis, este programa se puede encontrar en:

/home/sulla/umanis/cliente_umanis

Luego de ejecutar el programa pueden ser enviados todos los comandos anteriormente mencionados simplemente escribiendo el *payload* en pantalla.

VI El Proyecto

VI.1 Pruebas realizadas

Para verificar el funcionamiento del manipulador en su conjunto se realizaron varias pruebas, algunas de ellas dedicadas exclusivamente a comprobar que se cumplía cada una de las restricciones (de presición y de fuerza especialmente).

Pruebas referentes a la fuerza

Se ensayo cada motor individualmente para verificar que podía levantar la carga calculada para el peor caso, estos ensayos arrojaron como resultado las gráficas torque-velocidad que se muestran en la sección de la construcción mecánica.
Cabe resaltar que estos ensayos se realizaron con niveles continuos de voltaje (utilizando una fuente externa) y no con la PWM generada por nuestras placas.
Al probar los motores con nuestras placas y con el manipulador armado completamente se encontró una limitación en la carga que era capaz de levantar el motor de la muñeca, la limitación derivaba de que a partir de aproximadamente los 350g el elevador de voltaje PT5049 que se encuentra en la placa de potencia, era incapaz de entregar la corriente suficiente al motor (recordamos que se utilizo este elevador porque el motor era de voltaje nominal 24 V y solo teníamos acceso a 12 V con las baterias de Sulla). La corriente máxima que es capaz de entregar este integrado es de 500 mA parte de los cuales son consumidos por el driver LMD18200 y la corriente disponible para el motor no era suficiente.
Las consecuencias de este detalle es que a partir de los 350g las trayectorias que podría llegar a seguir el manipulador deben ser con la muñeca siempre en la misma posición, esto no parece ser algo muy relevante dado que todos los demás motores no presentan ningún problema para mover esta carga, por lo que el movimiento seria posible de todas maneras.

Pruebas referentes a la precisión

En este punto es donde se ve mas claramente la dificultad de implementar los algoritmos de control óptimos para el movimiento preciso del brazo en todas las posiciones.
Las pruebas arrojaron claramente que se obtenía mayor precisión para trayectorias en que el torque exigido a cada motor no variaba demasiado y además no era ninguno de los extremos (ni mucha, ni poca carga), esto se debe a que el ajuste de controlador se efectuó para valores medios de carga, no para las posiciones en que el torque era muy reducido o muy grande.
Se realizaron pruebas en las que se buscaba seguir una trayectoria y se veía la precisión durante toda la trayectoria (una manera bastante gráfica de realizar esto fue haciendo al manipulador dibujar distintas rectas y viendo los resultados, incluso midiendo el largo de esas rectas para ver si eran correctas), los resultados para trayectorias razonables fueron aceptables.
Otra forma de ensayar la precisión del manipulador fue viendo la precisión en la posición final y no necesariamente durante la trayectoria, por ejemplo hacer al manipulador recorrer varias trayectorias al azar y luego volver a la posición original, en este caso los resultados fueron bastante mejores dado que a diferencia del caso del análisis de la trayectoria completa al buscar una posición final los errores tienden a cancelarse para obtener un punto final mas preciso.

Pruebas referentes al generador de trayectorias

Para estas pruebas simplemente se le pedían diferentes trayectorias al manipulador y se comprobaba el buen funcionamiento.
En el caso de las rectas en un plano horizontal el método de escribir fue muy adecuado y para otro tipo de rectas que salían de este plano se utilizo un método visual (en estas pruebas no se buscaba ver la presicion del manipulador sino comprobar la generación correcta de las trayectorias a nivel de software).

Pruebas de la pinza

Para las pruebas del control de torque de la pinza se utilizo primero un método manual, comprobando con el tacto si la pinza presionaba con mayor o menor fuerza según se lo indicábamos.
Otra forma de prueba fue presionando elementos deformables (un vaso de plástico, por ejemplo) y comprobando cuanto se deformaba con cada nivel de torque. Con estos dos ensayos fue bastante claro que esta función se implementaba correctamente.
Se realizo otra prueba un tanto pintoresca en esta área, se busco que el manipulador primero sostuviera un huevo de gallina sin romperlo (con nivel de torque mínimo) y luego subiendo el torque se lograra romperlo. La primera parte se logro con éxito (el huevo no se rompió) pero luego no fue posible romperlo al subir el torque. Creemos que esto no se debió a que el torque no fuera suficiente sino a que la goma que utilizamos para los "dedos" envolvía el huevo de manera que este no se rompía (es fácilmente comprobable lo difícil que es romper un huevo por ejemplo envolviéndolo con la mano, sin clavarle los dedos), también creemos que si no se utilizara esa goma sino un material mas duro (el acrílico de los dedos, por ejemplo) si se hubiera podido cumplir con las dos etapas de la prueba, pero esto no se realizo.

Pruebas de integración con Sulla

Se utilizaron los sensores del "cinturón" de Sulla para detectar que cual de tres posiciones posibles y bien determinadas se encontraba un objeto para luego ordenar al manipulador que se dirigía a esa posición y cierre la pinza, esta prueba se realizo con éxito.

VI.2 Cosas por hacer

Son varias las cosas que se pueden seguir aportando a este proyecto, es un plataforma que permite diversas aplicaciones que tal vez necesiten algunas mejoras al manipulador tal cual esta actualmente.

En lo que refiere a la construcción mecánica, las principales mejoras corresponden a la articulación de la cintura, por un lado el motor que acciona esta articulación presenta mucho "juego" lo cual repercute considerablemente en el control de movimiento, una opción (tal vez la mas adecuada) es instalar otro motor de similares dimensiones pero en mejores condiciones sobre todo en lo referente al juego interno, de optar por cambiar este motor también se podría aprovechar la oportunidad e instalar un motor con mas reducción, por un lado se bajaría la velocidad nominal de este motor (la cual es demasiado alta en este momento) y por otro se aumentaría la fuerza que también sería un buen aporte debido al gran momento de inercia que debe soportar esta articulación.
Por otro lado, otra mejora que puede ayudar mucho es un mecanismo (que constaría de algún tipo de rulemán especial) para que todo el peso del manipulador no repose sobre el eje de la cintura sino sobre otra superficie mas amplia. Consideramos que el

actual montaje de la cintura es un tanto "peligroso" para la integridad de la pieza modular que le da movimiento dado que esta pieza probablemente no fue diseñada para soportar tanto peso (ni tanta palanca) en ese punto.

Las demás articulaciones no presentan demasiados problemas, tal vez en el motor del hombro, cuando cambia de lado la tensión de la correa (o sea, hacia atrás de la posición con el hombro a 90 grados, con el brazo hacia arriba) se nota un juego debido a este cambio de tensión, esto se solucionaría tensando un poco mas la correa (trasladando el motor del hombro unos mm. hacia atrás). De todas maneras este detalle no es demasiado importante dado que son pocas las trayectorias en que el hombro alcanza esa posición, en la mayoría de los movimientos la tensión de la correa es siempre hacia el mismo lado.

El área donde mas aportes son posibles es, sin duda, el área de control, hacer algoritmos de control robustos que se adapten a todas las particularidades y variaciones de la planta excedía en gran medida el alcance de nuestro proyecto, pero se deja la plataforma para implementarlos con todas las facilidades en las áreas ajenas al control. Como aclaramos en otras secciones en este proyecto se implemento una estrategia de control relativamente simple que daba resultados satisfactorios pero lógicamente son muchas las posibilidades para implementar algo mas adecuado y mas versátil.

Por el lado de Software posibles mejoras pueden aplicarse a las rutinas de atención de alarmas, por ejemplo se podría implementar algoritmos que detecten cual fue el problema que desencadeno la alarma para en un futuro modificar algún detalle (podría ser de software, de hardware mecánico o de hardware eléctrico) de tal manera que tales problemas no vuelvan a ocurrir.
También se podrían implementar generadores de trayectoria para movimientos que no fueran en línea recta o bien que no fueran con el codo en "posición grúa".
También se podría (a nivel de software) mejorar la interacción con Sulla, por ejemplo un problema a solucionar son las situaciones en las que el brazo interfiere con algunos de los sensores de Sulla, en estos casos la robota estaría interpretando este objeto como algo externo por lo cual reaccionaria ante su presencia. Esto no parece ser demasiado difícil de implementar, simplemente se deberían hallar las posiciones del manipulador que interfieren con los distintos sensores y para estas posiciones anular dicho sensor para que Sulla no reaccione.

En casi todas las áreas hay posibilidad de implementar cosas nuevas, y las alternativas son muchas, de solo pensar la cantidad de tareas que se podrían hacer con un robot móvil con un manipulador incluido se abren una gran cantidad de posibilidades.

VI.3 Conclusiones

Hemos aprendido muchísimas cosas en la realización de este proyecto. Más allá del conocimiento técnico que hemos adquirido, que fue muy rico y abarcó las más diversas áreas de la Ingeniería, hemos aprendido mucho del trabajo en grupo, de la organización, de cómo lidiar con los inconvenientes más inesperados, a adoptar planes de contingencia, a trabajar en base a objetivos y a hacernos entender entre nosotros de manera productiva.

Construir un robot, es una tarea sumamente amplia, que implica el dominio de muchas áreas de la ingeniería eléctrica, como ser: la electrónica de señal, la electrónica de potencia, microprocesadores, diseño y producción de circuitos impresos, desarrollo de software, redes de datos, etc. También implicó lidiar con áreas no necesariamente relacionadas con la ingeniería eléctrica, como por ejemplo: diseño mecánico, sistemas de transmisión, creación de piezas, e infinidad de detalles que pueden sonar

irrelevantes a los ojos de la ingeniería, como hacer cables, fabricar arandelas y actividades del estilo que nos consumieron muchas horas de trabajo.

Desarrollamos nuestros conocimientos en muchas áreas, y por sobre todo logramos el feliz resultado de llegar a una implementación funcional, es decir algo que funciona. Creamos desde los cimientos un sistema integrado, y logramos que funcionara como un conjunto.

Como aplicación final, y en este afán de sacar conclusiones positivas, logramos básicamente dos objetivos sumamente importantes:

- Hacer que el manipulador escriba con un marcador sobre pizarrón en posición horizontal (demostrando la precisión de las trayectorias en línea recta en el espacio).
- Lograr que el manipulador tome un objeto ubicado en una posición predefinida y lo deposite en el plano de tierra. El valor agregado de esta prueba es la integración con el sistema sensorial de Sulla, ya que existen dos posiciones predefinidas y Umanis debe interrogar a los sensores de Sulla para saber en donde se encuentra el objeto en cuestión.

Realmente, sabemos que hay muchas cosas por mejorar, pero estamos sumamente orgullosos de nuestro proyecto de fin de carrera.

Referencias [12]

1. National Semiconductors. ***AN-693: Application Note 693 LM628 Programming Guide***. January 1993.

2. National Semiconductor. ***AN-694: Application Note 694 A DMOS 3A, 55V, H-Bridge: The LMD18200***. December 1999.

3. National Semiconductors. ***AN-706: Application Note 706 LM628/629 User Guide***. January 1993.

4. National Semiconductors. ***AN-868: Interfacing the HPC and LM629 for Motion Control***. January 1993.

5. Hall Brian. ***Beej's Guide to Network Programming Using Internet Sockets*** [online]. Consultado en Abril de 2005. Disponible en << http://beej.us/guide/bgnet/output/htmlsingle/bgnet.html>>.

6. Tanenbaum, Andrew S. ***Computer Networks***. 3a. Edición, Prentice-Hall 1996.

7. Berns K., Vogt H., Asfour T., Dillmann R. ***Design and Control Architecture of an Anthropomorphic Robot***. Forschungszentrum Informatik Karlsruhe. D-76131 Karlsruhe, Germany.

8. Z·World. ***Dynamic C TCP/IP User's Manual***. Part Number 040313·019-0100-H. 2002

9. Z·World. ***Dynamic C User's Manual***. Part Number 040313·019-0125-B. 2002

10. Philips Semiconductors. ***HEF4069 Datasheet.*** January 1995.

11. Boone Kevin. ***How to write a techincal report***. Version 1.03. January 2003.

12. ISO 690-2:1987. ***Information and documentation -- Bibliographic references --*** [online]. Consultado el 17 de Setiembre de 2005. Diponible en <<http://www.collectionscanada.ca/iso/tc46sc9/standard/690-2e.htm >>

13. OGATA K. ***Ingeniería de Control Moderna***. 3ra Edición. PRENTICE HALL ISBN 970-17-0048-1.

14. National Semiconductors. ***LMD18200 Datasheet.*** April 2005.

15. Labrosse Jean J. ***µC/OS-II, The Real Time Kernel***. 2nd Edition. March 2002.

16. Sedra, Adel S., Smith, Kenneth C. ***Microelectronic Circuits***. 4th Edition. Oxford University Press. 1998.

17. Papadoupolous E., Poulakakis J. ***Planning and Model-Based Control for Mobile Manipulators***. Department of Mechanical Engineering. National Technical University of Athens, 15780 Zografou, Greece.

18. Marshall A. D. ***Programming in C*** [online]. Consultado en Agosto de 2004. Disponible en <<http://www.cs.cf.ac.uk/Dave/C/CE.html>>.

19. Rabbit Semiconductor. ***Rabbit 3000 Microprocessor User's Manual***. Part Number 019-0108·040731-O. 2002.

20. Z·World. ***RabbitCore RCM3200 User's Manual***. Part Number 019-0108·031205-F. 2003.

21. Kumar V. ***Robot Geometry and Kinematics*** [online]. Consultado en Junio de 2004. Disponible en <<http://www.seas.upenn.edu/~meam100/handouts/robotkin.pdf>>.

22. Sandin P. ***Robot Mechanisms and Mechanical Devices***. Mc Graw-Hill Companies. 2003

23. Spong Mark W. ***Robust and Adaptative Motion Control of Manipulators***. Coordinated Science Laboratory. University of Illinois at Urbana-Champaign.

24. Welsh Matt, Kalle Dalheimer Matthias, Dawson Terry, Kaufman Lar. ***Running LINUX***. *4TH Edition. O'Reilly*. December 2002.

25. Grompone Rafael, Quagliotti Agustín, Rolando Pablo. ***Sulla, una robota móvil***. 2003.

26. Philips Semiconductors. ***The I2C-Bus Specification***. Version 2.1. January 2000.

Anexos

Contenido del CD.

Requerimientos: Estos son los requerimientos básicos para poder ver en forma correcta los archivos que se pueden encontrar en este CD. En algunos casos con versiones anteriores de alguno de los programas o con un PC con menor capacidad de lo mencionado se podrá visualizar el material en forma correcta, lo aquí mencionado es simplemente una guía.

- Procesador de 333 MHz.
- 64 K de memoria RAM
- Resolución del monitor 800 x 600

- *Acrobat Reader 5.0* (para archivos .pdf)
- *Eagle* (Cadsoft) 4.0 o superior (para archivos .brd)
- *Bloc de notas* o cualquier editor de textos (para archivos .lib y .c)

Contenido:

Archivo / *Carpeta*	
Documentación.pdf	Versión digital de la documentación final, incluye correcciones del tribunal.
Paper.pdf	Articulo (en forma de resumen) con el formato de la IEEE.
Hojas de datos	Carpeta que contiene las hojas de datos de los distintos componentes electrónica utilizados.
Control.brd	Esquemático de la placa de control, a ser visualizado con el programa Eagle de Cadsoft.
Potencia.brd	Esquemático de la placa de potencia, a ser visualizado con el programa Eagle de Cadsoft
Codigo	Carpeta con todo el código del software de Umanis

National Semiconductor

January 2003

LM628/LM629
Precision Motion Controller

General Description

The LM628/LM629 are dedicated motion-control processors designed for use with a variety of DC and brushless DC servo motors, and other servomechanisms which provide a quadrature incremental position feedback signal. The parts perform the intensive, real-time computational tasks required for high performance digital motion control. The host control software interface is facilitated by a high-level command set. The LM628 has an 8-bit output which can drive either an 8-bit or a 12-bit DAC. The components required to build a servo system are reduced to the DC motor/actuator, an incremental encoder, a DAC, a power amplifier, and the LM628. An LM629-based system is similar, except that it provides an 8-bit PWM output for directly driving H-switches. The parts are fabricated in NMOS and packaged in a 28-pin dual in-line package or a 24-pin surface mount package (LM629 only). Both 6 MHz and 8 MHz maximum frequency versions are available with the suffixes -6 and -8, respectively, used to designate the versions. They incorporate an SDA core processor and cells designed by SDA.

Features

- 32-bit position, velocity, and acceleration registers
- Programmable digital PID filter with 16-bit coefficients
- Programmable derivative sampling interval
- 8- or 12-bit DAC output data (LM628)
- 8-bit sign-magnitude PWM output data (LM629)
- Internal trapezoidal velocity profile generator
- Velocity, target position, and filter parameters may be changed during motion
- Position and velocity modes of operation
- Real-time programmable host interrupts
- 8-bit parallel asynchronous host interface
- Quadrature incremental encoder interface with index pulse input
- Available in a 28-pin dual in-line package or a 24-pin surface mount package (LM629 only)

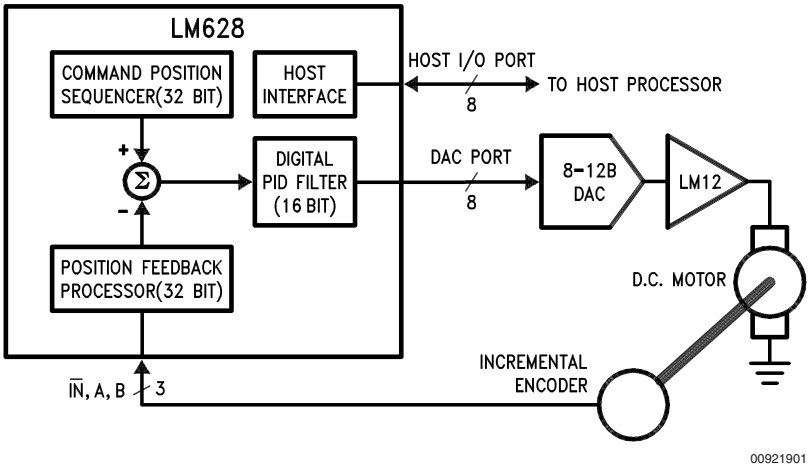
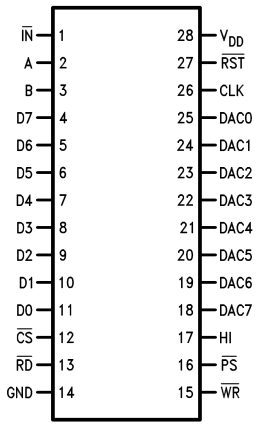


FIGURE 1. Block Diagram

TRISTATE&® is a registered trademark of National Semiconductor Corporation.

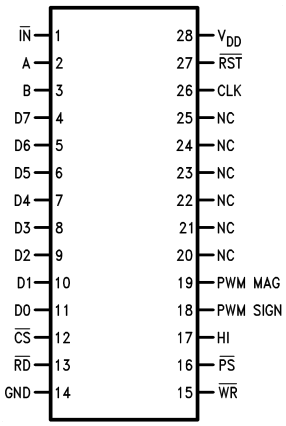
Connection Diagrams

LM628N



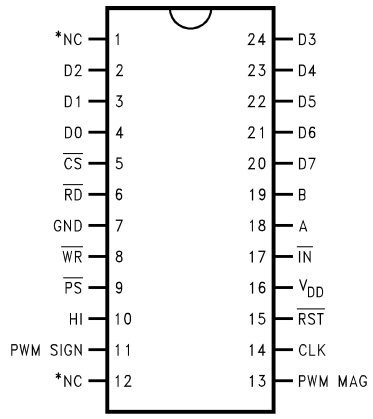
00921902

LM629N



00921903

LM629M



00921921

*Do not connect.

Order Number LM629M-6, LM629M-8, LM628N-6, LM628N-8, LM629N-6 or LM629N-8
See NS Package Number M24B or N28B

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Voltage at Any Pin with	
Respect to GND	−0.3V to +7.0V
Ambient Storage Temperature	−65˚C to +150˚C
Lead Temperature	
28-pin Dual In-Line	
Package (Soldering, 4 sec.)	260˚C
24-pin Surface Mount	
Package (Soldering, 10 sec.)	300˚C
Maximum Power Dissipation	
($T_A \leq 85$˚C, (Note 2)	605 mW
ESD Tolerance	
(C_{ZAP} = 120 pF, R_{ZAP} = 1.5k)	1000V

Operating Ratings

Temperature Range	$-40˚C < T_A < +85˚C$
Clock Frequency:	
LM628N-6, LM629N-6,	
LM629M-6	$1.0\ MHz < f_{CLK} < 6.0\ MHz$
LM628N-8, LM629N-8,	
LM629M-8	$1.0\ MHz < f_{CLK} < 8.0\ MHz$
V_{DD} Range	$4.5V < V_{DD} < 5.5V$

DC Electrical Characteristics

(V_{DD} and T_A per Operating Ratings; f_{CLK} = 6 MHz)

Symbol	Parameter	Conditions	Tested Limits		Units
			Min	Max	
I_{DD}	Supply Current	Outputs Open		110	mA
INPUT VOLTAGES					
V_{IH}	Logic 1 Input Voltage		2.0		V
V_{IL}	Logic 0 Input Voltage			0.8	V
I_{IN}	Input Currents	$0 \leq V_{IN} \leq V_{DD}$	−10	10	µA
OUTPUT VOLTAGES					
V_{OH}	Logic 1	I_{OH} = −1.6 mA	2.4		V
V_{OL}	Logic 0	I_{OL} = 1.6 mA		0.4	V
I_{OUT}	TRI-STATE® Output Leakage Current	$0 \leq V_{OUT} \leq V_{DD}$	−10	10	µA

AC Electrical Characteristics

(V_{DD} and T_A per Operating Ratings; f_{CLK} = 6 MHz; C_{LOAD} = 50 pF; Input Test Signal $t_r = t_f$ = 10 ns)

Timing Interval	T#	Tested Limits		Units
		Min	Max	
ENCODER AND INDEX TIMING (See *Figure 2*)				
Motor-Phase Pulse Width	T1	$\frac{16}{f_{CLK}}$		µs
Dwell-Time per State	T2	$\frac{8}{f_{CLK}}$		µs
Index Pulse Setup and Hold (Relative to A and B Low)	T3	0		µs
CLOCK AND RESET TIMING (See *Figure 3*)				
Clock Pulse Width				
LM628N-6, LM629N-6, LM629M-6	T4	78		ns
LM628N-8, LM629N-8, LM629M-8	T4	57		ns
Clock Period				
LM628N-6, LM629N-6, LM629M-6	T5	166		ns
LM628N-8, LM629N-8, LM629M-8	T5	125		ns
Reset Pulse Width	T6	$\frac{8}{f_{CLK}}$		µs

AC Electrical Characteristics (Continued)

(V_{DD} and T_A per Operating Ratings; f_{CLK} = 6 MHz; C_{LOAD} = 50 pF; Input Test Signal $t_r = t_f$ = 10 ns)

Timing Interval	T#	Tested Limits		Units
		Min	Max	
STATUS BYTE READ TIMING (See *Figure 4*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Read Data Access Time	T10		180	ns
Read Data Hold Time	T11	0		ns
$\overline{RD}$ High to Hi-Z Time	T12		180	ns
COMMAND BYTE WRITE TIMING (See *Figure 5*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Busy Bit Delay	T13		(Note 3)	ns
$\overline{WR}$ Pulse Width	T14	100		ns
Write Data Setup Time	T15	50		ns
Write Data Hold Time	T16	120		ns
DATA WORD READ TIMING (See *Figure 6*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Read Data Access Time	T10		180	ns
Read Data Hold Time	T11	0		ns
$\overline{RD}$ High to Hi-Z Time	T12		180	ns
Busy Bit Delay	T13		(Note 3)	ns
Read Recovery Time	T17	120		ns
DATA WORD WRITE TIMING (See *Figure 7*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Busy Bit Delay	T13		(Note 3)	ns
$\overline{WR}$ Pulse Width	T14	100		ns
Write Data Setup Time	T15	50		ns
Write Data Hold Time	T16	120		ns
Write Recovery Time	T18	120		ns

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond the above Operating Ratings.

Note 2: When operating at ambient temperatures above 70˚C, the device must be protected against excessive junction temperatures. Mounting the package on a printed circuit board having an area greater than three square inches and surrounding the leads and body with wide copper traces and large, uninterrupted areas of copper, such as a ground plane, suffices. The 28-pin DIP (N) and the 24-pin surface mount package (M) are molded plastic packages with solid copper lead frames. Most of the heat generated at the die flows from the die, through the copper lead frame, and into copper traces on the printed circuit board. The copper traces act as a heat sink. Double-sided or multi-layer boards provide heat transfer characteristics superior to those of single-sided boards.

Note 3: In order to read the busy bit, the status byte must first be read. The time required to read the busy bit far exceeds the time the chip requires to set the busy bit. It is, therefore, impossible to test actual busy bit delay. The busy bit is guaranteed to be valid as soon as the user is able to read it.

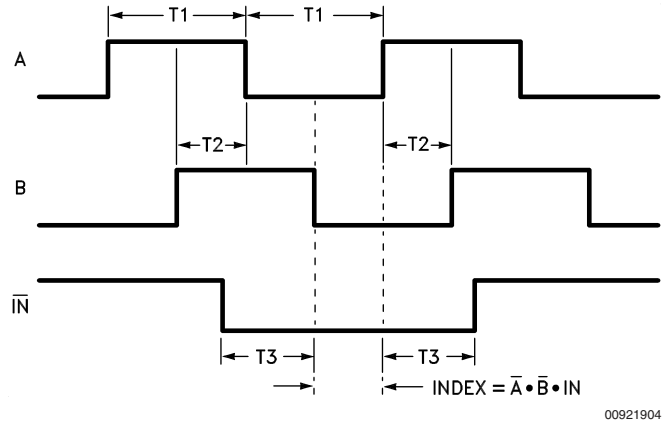


FIGURE 2. Quadrature Encoder Input Timing

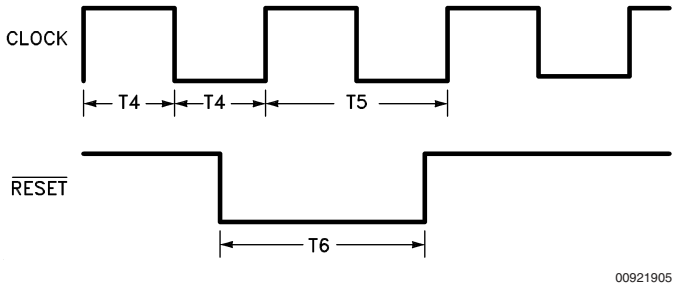


FIGURE 3. Clock and Reset Timing

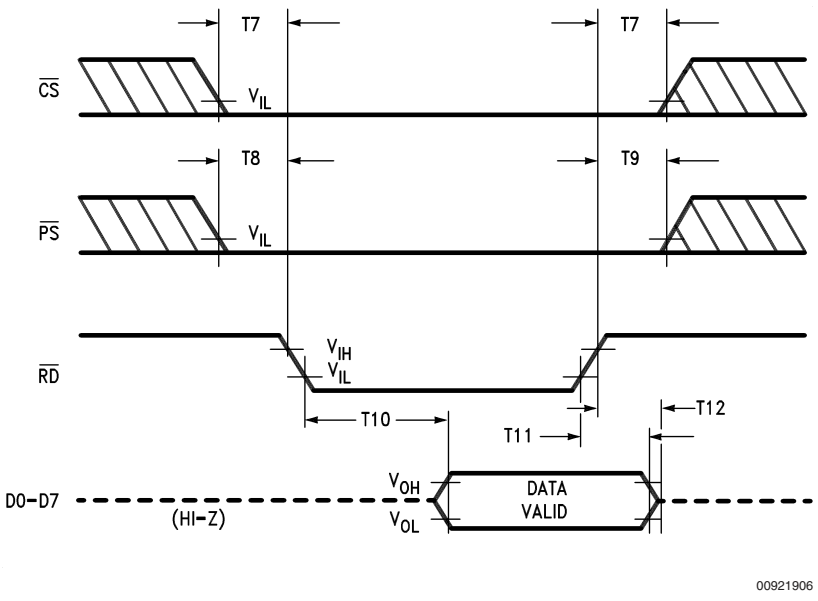


FIGURE 4. Status Byte Read Timing

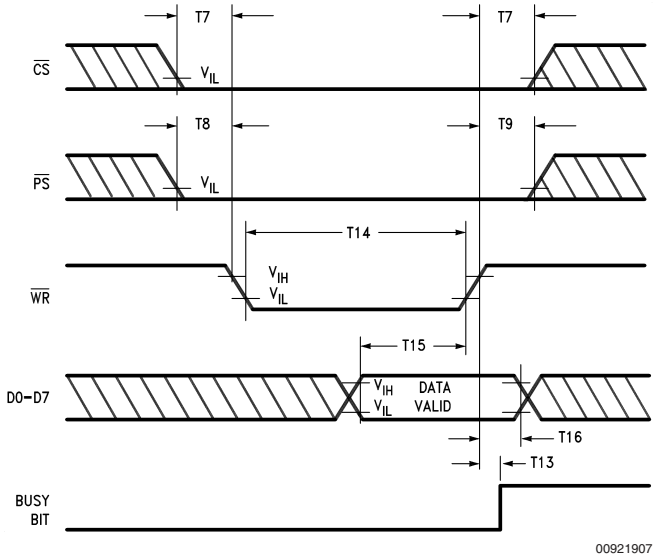


FIGURE 5. Command Byte Write Timing

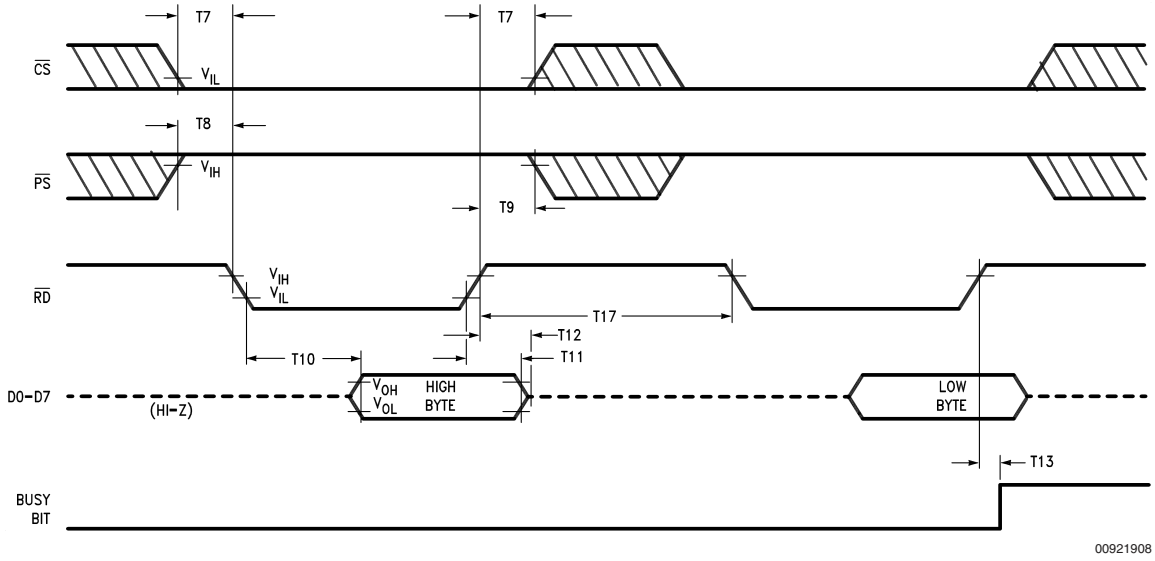


FIGURE 6. Data Word Read Timing

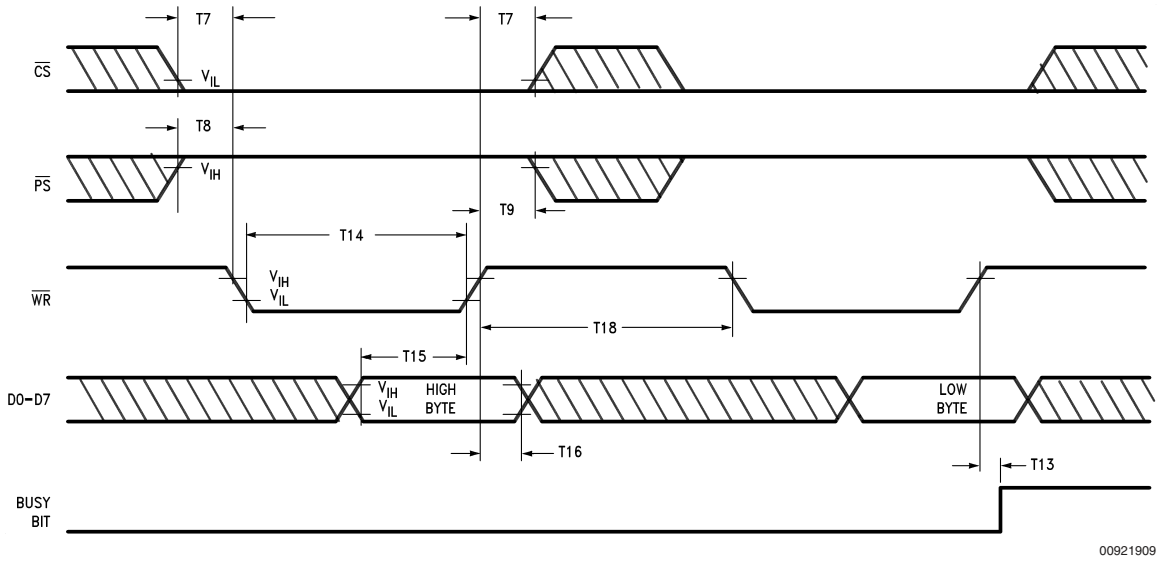


FIGURE 7. Data Word Write Timing

Pinout Description

(See Connection Diagrams) Pin numbers for the 24-pin surface mount package are indicated in parentheses.

Pin 1 (17), Index ($\overline{IN}$) Input: Receives optional index pulse from the encoder. Must be tied high if not used. The index position is read when Pins 1, 2, and 3 are low.

Pins 2 and 3 (18 and 19), Encoder Signal (A, B) Inputs: Receive the two-phase quadrature signals provided by the incremental encoder. When the motor is rotating in the positive ("forward") direction, the signal at Pin 2 leads the signal at Pin 3 by 90 degrees. Note that the signals at Pins 2 and 3 must remain at each encoder state (See *Figure 9*) for a minimum of 8 clock periods in order to be recognized. Because of a four-to-one resolution advantage gained by the method of decoding the quadrature encoder signals, this corresponds to a maximum encoder-state capture rate of 1.0 MHz (f_{CLK} = 8.0 MHz) or 750 kHz (f_{CLK} = 6.0 MHz). For other clock frequencies the encoder signals must also remain at each state a minimum of 8 clock periods.

Pins 4 to 11 (20 to 24 and 2 to 4), Host I/O Port (D0 to D7): Bi-directional data port which connects to host computer/processor. Used for writing commands and data to the LM628, and for reading the status byte and data from the LM628, as controlled by $\overline{CS}$ (Pin 12), $\overline{PS}$ (Pin 16), $\overline{RD}$ (Pin 13), and $\overline{WR}$ (Pin 15).

Pin 12 (5), Chip Select ($\overline{CS}$) Input: Used to select the LM628 for writing and reading operations.

Pin 13 (6), Read ($\overline{RD}$) Input: Used to read status and data.

Pin 14 (7), Ground (GND): Power-supply return pin.

Pin 15 (8), Write ($\overline{WR}$) Input: Used to write commands and data.

Pin 16 (9), Port Select ($\overline{PS}$) Input: Used to select command or data port. Selects command port when low, data port when high. The following modes are controlled by Pin 16:

1. Commands are written to the command port (Pin 16 low),
2. Status byte is read from command port (Pin 16 low), and
3. Data is written and read via the data port (Pin 16 high).

Pin 17 (10), Host Interrupt (HI) Output: This active-high signal alerts the host (via a host interrupt service routine) that an interrupt condition has occurred.

Pins 18 to 25, DAC Port (DAC0 to DAC7): Output port which is used in three different modes:

1. LM628 (8-bit output mode): Outputs latched data to the DAC. The MSB is Pin 18 and the LSB is Pin 25.
2. LM628 (12-bit output mode): Outputs two, multiplexed 6-bit words. The less-significant word is output first. The MSB is on Pin 18 and the LSB is on Pin 23. Pin 24 is used to demultiplex the words; Pin 24 is low for the less-significant word. The positive-going edge of the signal on Pin 25 is used to strobe the output data. *Figure 8* shows the timing of the multiplexed signals.
3. LM629 (sign/magnitude outputs): Outputs a PWM sign signal on Pin 18 (11 for surface mount), and a PWM magnitude signal on Pin 19 (13 for surface mount). Pins 20 to 25 are not used in the LM629. *Figure 11* shows the PWM output signal format.

Pin 26 (14), Clock (CLK) Input: Receives system clock.

Pin 27 (15), Reset ($\overline{RST}$) Input: Active-low, positive-edge triggered, resets the LM628 to the internal conditions shown below. Note that the reset pulse must be logic low for a minimum of 8 clock periods. Reset does the following:

1. Filter coefficient and trajectory parameters are zeroed.
2. Sets position error threshold to maximum value (7FFF hex), and effectively executes command LPEI.
3. The SBPA/SBPR interrupt is masked (disabled).
4. The five other interrupts are unmasked (enabled).
5. Initializes current position to zero, or "home" position.
6. Sets derivative sampling interval to $2048/f_{CLK}$ or 256 µs for an 8.0 MHz clock.
7. DAC port outputs 800 hex to "zero" a 12-bit DAC and then reverts to 80 hex to "zero" an 8-bit DAC.

Immediately after releasing the reset pin from the LM628, the status port should read "00". If the reset is successfully completed, the status word will change to hex "84" or "C4"

Pinout Description (Continued)

within 1.5 ms. If the status word has not changed from hex "00" to "84" or "C4" within 1.5 ms, perform another reset and repeat the above steps. To be certain that the reset was properly performed, execute a **RSTI** command. If the chip has reset properly, the status byte will change from hex "84" or "C4" to hex "80" or "C0". If this does not occur, perform another reset and repeat the above steps.

Pin 28 (16), Supply Voltage (V_{DD}): Power supply voltage (+5V).

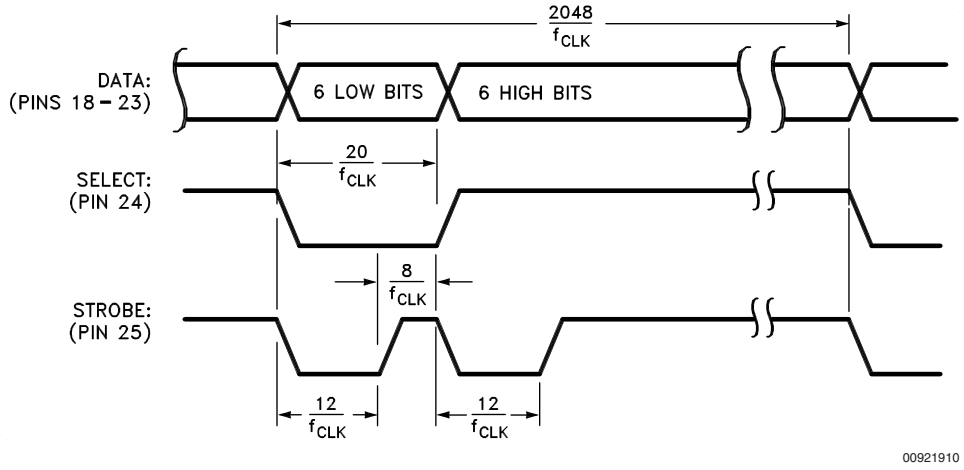


FIGURE 8. 12-Bit Multiplexed Output Timing

Theory of Operation

INTRODUCTION

The typical system block diagram (See *Figure 1*) illustrates a servo system built using the LM628. The host processor communicates with the LM628 through an I/O port to facilitate programming a trapezoidal velocity profile and a digital compensation filter. The DAC output interfaces to an external digital-to-analog converter to produce the signal that is power amplified and applied to the motor. An incremental encoder provides feedback for closing the position servo loop. The trapezoidal velocity profile generator calculates the required trajectory for either position or velocity mode of operation. In operation, the LM628 subtracts the actual position (feedback position) from the desired position (profile generator position), and the resulting position error is processed by the digital filter to drive the motor to the desired position. *Table 1* provides a brief summary of specifications offered by the LM628/LM629:

POSITION FEEDBACK INTERFACE

The LM628 interfaces to a motor via an incremental encoder. Three inputs are provided: two quadrature signal inputs, and an index pulse input. The quadrature signals are used to keep track of the absolute position of the motor. Each time a logic transition occurs at one of the quadrature inputs, the LM628 internal position register is incremented or decremented accordingly. This provides four times the resolution over the number of lines provided by the encoder. See *Figure 9*. Each of the encoder signal inputs is synchronized with the LM628 clock.

The optional index pulse output provided by some encoders assumes the logic-low state once per revolution. If the LM628 is so programmed by the user, it will record the absolute motor position in a dedicated register (the index register) at the time when all three encoder inputs are logic low.

If the encoder does not provide an index output, the LM628 index input can also be used to record the home position of the motor. In this case, typically, the motor will close a switch which is arranged to cause a logic-low level at the index input, and the LM628 will record motor position in the index register and alert (interrupt) the host processor. Permanently grounding the index input will cause the LM628 to malfunction.

TABLE 1. System Specifications Summary

Position Range	–1,073,741,824 to 1,073,741,823 counts
Velocity Range	0 to 1,073,741,823/2^{16} counts/sample; ie, 0 to 16,383 counts/sample, with a resolution of 1/2^{16} counts/sample
Acceleration Range	0 to 1,073,741,823/2^{16} counts/sample/sample; ie, 0 to 16,383 counts/sample/sample, with a resolution of 1/2^{16} counts/sample/sample
Motor Drive Output	LM628: 8-bit parallel output to DAC, or 12-bit multiplexed output to DAC LM629: 8-bit PWM sign/magnitude signals
Operating Modes	Position and Velocity
Feedback Device	Incremental Encoder (quadrature signals; support for index pulse)

Theory of Operation (Continued)

TABLE 1. System Specifications Summary (Continued)

Control Algorithm	Proportional Integral Derivative (PID) (plus programmable integration limit)
Sample Intervals	Derivative Term: Programmable from $2048/f_{CLK}$ to $(2048 * 256)/f_{CLK}$ in steps of $2048/f_{CLK}$ (256 to 65,536 µs for an 8.0 MHz clock). Proportional and Integral: $2048/f_{CLK}$

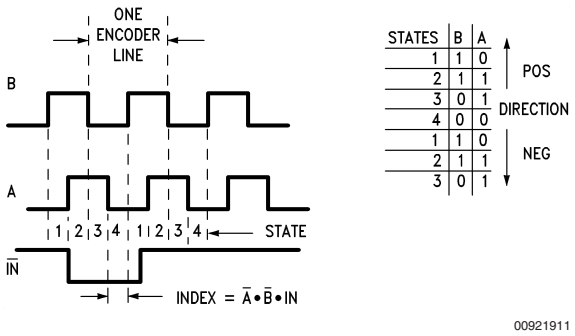


FIGURE 9. Quadrature Encoder Signals

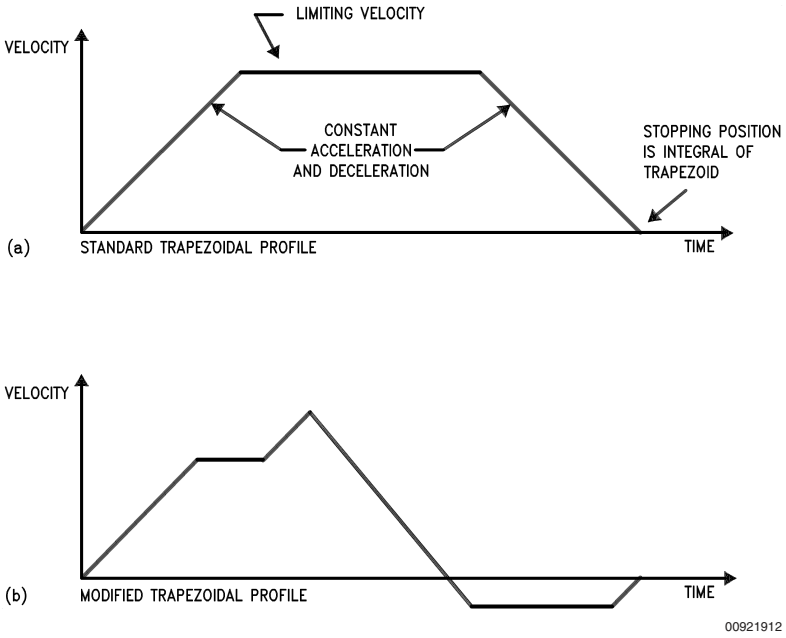


FIGURE 10. Typical Velocity Profiles

VELOCITY PROFILE (TRAJECTORY) GENERATION

The trapezoidal velocity profile generator computes the desired position of the motor versus time. In the position mode of operation, the host processor specifies acceleration, maximum velocity, and final position. The LM628 uses this information to affect the move by accelerating as specified until the maximum velocity is reached or until deceleration must begin to stop at the specified final position. The deceleration rate is equal to the acceleration rate. At any time during the move the maximum velocity and/or the target position may be changed, and the motor will accelerate or decelerate accordingly. *Figure 10* illustrates two typical trapezoidal velocity profiles. *Figure 10*(a) shows a simple trapezoid, while *Figure 10*(b) is an example of what the trajectory looks like when velocity and position are changed at different times during the move.

When operating in the velocity mode, the motor accelerates to the specified velocity at the specified acceleration rate and maintains the specified velocity until commanded to stop. The velocity is maintained by advancing the desired position at a constant rate. If there are disturbances to the motion during velocity mode operation, the long-time average velocity remains constant. If the motor is unable to maintain the specified velocity (which could be caused by a locked rotor,

Theory of Operation (Continued)

for example), the desired position will continue to be increased, resulting in a very large position error. If this condition goes undetected, and the impeding force on the motor is subsequently released, the motor could reach a very high velocity in order to catch up to the desired position (which is still advancing as specified). This condition is easily detected; see commands LPEI and LPES.

All trajectory parameters are 32-bit values. Position is a signed quantity. Acceleration and velocity are specified as 16-bit, positive-only integers having 16-bit fractions. The integer portion of velocity specifies how many counts per sampling interval the motor will traverse. The fractional portion designates an additional fractional count per sampling interval. Although the position resolution of the LM628 is limited to integer counts, the fractional counts provide increased average velocity resolution. Acceleration is treated in the same manner. Each sampling interval the commanded acceleration value is added to the current desired velocity to generate a new desired velocity (unless the command velocity has been reached).

One determines the trajectory parameters for a desired move as follows. If, for example, one has a 500-line shaft encoder, desires that the motor accelerate at one revolution per second per second until it is moving at 600 rpm, and then decelerate to a stop at a position exactly 100 revolutions from the start, one would calculate the trajectory parameters as follows:

let P = target position (units = encoder counts)
let R = encoder lines * 4 (system resolution)
then R = 500 * 4 = 2000
and P = 2000 * desired number of revolutions
P = 2000 * 100 revs = 200,000 counts (value to load)
P (coding) = 00030D40 (hex code written to LM628)
let V = velocity (units = counts/sample)
let T = sample time (seconds) = 341 µs (with 6 MHz clock)
let C = conversion factor = 1 minute/60 seconds
then V = R * T * C * desired rpm
and V = 2000 * 341E−6 * 1/60 * 600 rpm
V = 6.82 counts/sample
V (scaled) = 6.82 * 65,536 = 446,955.52
V (rounded) = 446,956 (value to load)
V (coding) = 0006D1EC (hex code written to LM628)
let A = acceleration (units = counts/sample/sample)
A = R * T * T * desired acceleration (rev/sec/sec)
then A = 2000 * 341E−6 * 341E-6 * 1 rev/sec/sec
and A = 2.33E−4 counts/sample/sample
A (scaled) = 2.33E−4 * 65,536 = 15.24
A (rounded) = 15 (value to load)
A (coding) = 0000000F (hex code written to LM628)

The above position, velocity, and acceleration values must be converted to binary codes to be loaded into the LM628. The values shown for velocity and acceleration must be multiplied by 65,536 (as shown) to adjust for the required integer/fraction format of the input data. Note that after scaling the velocity and acceleration values, literal fractional data cannot be loaded; the data must be rounded and converted to binary. The factor of four increase in system resolution is due to the method used to decode the quadrature encoder signals, see *Figure 9*.

PID COMPENSATION FILTER

The LM628 uses a digital Proportional Integral Derivative (PID) filter to compensate the control loop. The motor is held at the desired position by applying a restoring force to the motor that is proportional to the position error, plus the integral of the error, plus the derivative of the error. The following discrete-time equation illustrates the control performed by the LM628:

$$u(n) = kp*e(n) + ki \sum_{N=0}^{n} e(n) + kd[e(n') - e(n' - 1)]: \quad \text{(Eq. 1)} \tag{1}$$

where u(n) is the motor control signal output at sample time n, e(n) is the position error at sample time n, n' indicates sampling at the derivative sampling rate, and kp, ki, and kd are the discrete-time filter parameters loaded by the users.

The first term, the proportional term, provides a restoring force porportional to the position error, just as does a spring obeying Hooke's law. The second term, the integration term, provides a restoring force that grows with time, and thus ensures that the static position error is zero. If there is a constant torque loading, the motor will still be able to achieve zero position error.

The third term, the derivative term, provides a force proportional to the rate of change of position error. It acts just like viscous damping in a damped spring and mass system (like a shock absorber in an automobile). The sampling interval associated with the derivative term is user-selectable; this capability enables the LM628 to control a wider range of inertial loads (system mechanical time constants) by providing a better approximation of the continuous derivative. In general, longer sampling intervals are useful for low-velocity operations.

In operation, the filter algorithm receives a 16-bit error signal from the loop summing-junction. The error signal is saturated at 16 bits to ensure predictable behavior. In addition to being multiplied by filter coefficient kp, the error signal is added to an accumulation of previous errors (to form the integral signal) and, at a rate determined by the chosen *derivative* sampling interval, the previous error is subtracted from it (to form the derivative signal). All filter multiplications are 16-bit operations; only the bottom 16 bits of the product are used.

The integral signal is maintained to 24 bits, but only the top 16 bits are used. This scaling technique results in a more usable (less sensitive) range of coefficient ki values. The 16 bits are right-shifted eight positions and multiplied by filter coefficient ki to form the term which contributes to the motor control output. The absolute magnitude of this product is compared to coefficient il, and the lesser, appropriately signed magnitude then contributes to the motor control signal.

The derivative signal is multiplied by coefficient kd each *derivative* sampling interval. This product contributes to the motor control output *every* sample interval, independent of the user-chosen *derivative* sampling interval.

The kp, limited ki, and kd product terms are summed to form a 16-bit quantity. Depending on the output mode (wordsize), either the top 8 or top 12 bits become the motor control output signal.

Theory of Operation (Continued)

LM628 READING AND WRITING OPERATIONS

The host processor writes commands to the LM628 via the host I/O port when Port Select ($\overline{PS}$) input (Pin 16) is logic low. The desired command code is applied to the parallel port line and the Write ($\overline{WR}$) input (Pin 15) is strobed. The command byte is latched into the LM628 on the rising edge of the $\overline{WR}$ input. When writing command bytes it is necessary to first read the status byte and check the state of a flag called the "busy bit" (Bit 0). If the busy bit is logic high, no command write may take place. The busy bit is never high longer than 100 µs, and typically falls within 15 µs to 25 µs.

The host processor reads the LM628 status byte in a similar manner: by strobing the Read ($\overline{RD}$) input (Pin 13) when $\overline{PS}$ (Pin 16) is low; status information remains valid as long as $\overline{RD}$ is low.

Writing and reading data to/from the LM628 (as opposed to writing commands and reading status) are done with $\overline{PS}$ (Pin 16) logic high. These writes and reads are always an integral number (from one to seven) of two-byte words, with the first byte of each word being the more significant. Each byte requires a write ($\overline{WR}$) or read ($\overline{RD}$) strobe. When transferring data words (byte-pairs), it is necessary to first read the status byte and check the state of the busy bit. When the busy bit is logic low, the user may then sequentially transfer both bytes comprising a data word, but the busy bit must again be checked and found to be low before attempting to transfer the next byte pair (when transferring multiple words). Data transfers are accomplished via LM628-internal interrupts (which are not nested); the busy bit informs the host processor when the LM628 may not be interrupted for data transfer (or a command byte). If a command is written when the busy bit is high, the command will be ignored.

The busy bit goes high immediately after writing a command byte, or reading or writing a second byte of data (See *Figure 5* thru *Figure 7*).

MOTOR OUTPUTS

The LM628 DAC output port can be configured to provide either a latched eight-bit parallel output or a multiplexed 12-bit output. The 8-bit output can be directly connected to a flow-through (non-input-latching) D/A converter; the 12-bit output can be easily demultiplexed using an external 6-bit latch and an input-latching 12-bit D/A converter. The DAC output data is offset-binary coded; the 8-bit code for zero is 80 hex and the 12-bit code for zero is 800 hex. Values less than these cause a negative torque to be applied to the motor and, conversely, larger values cause positive motor torque. The LM628, when configured for 12-bit output, provides signals which control the demultiplexing process. See for details.

The LM629 provides 8-bit, sign and magnitude PWM output signals for directly driving switch-mode motor-drive amplifiers. *Figure 11* shows the format of the PWM magnitude output signal.

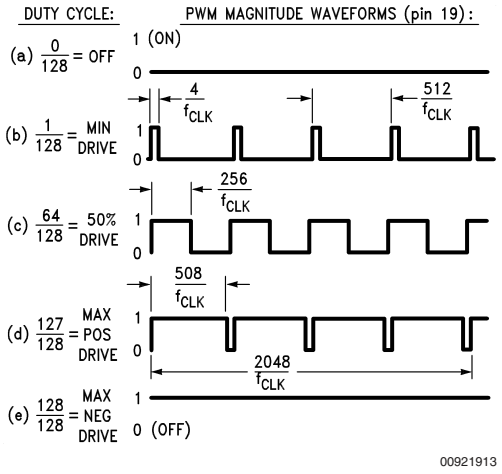


FIGURE 11. PWM Output Signal Format (Sign output (pin 18) not shown)

TABLE 2. LM628 User Command Set

Command	Type	Description	Hex	Data Bytes	Note
RESET	Initialize	Reset LM628	00	0	1
PORT8	Initialize	Select 8-Bit Output	05	0	2
PORT12	Initialize	Select 12-Bit Output	06	0	2
DFH	Initialize	Define Home	02	0	1
SIP	Interrupt	Set Index Position	03	0	1
LPEI	Interrupt	Interrupt on Error	1B	2	1
LPES	Interrupt	Stop on Error	1A	2	1
SBPA	Interrupt	Set Breakpoint, Absolute	20	4	1

Theory of Operation (Continued)

TABLE 2. LM628 User Command Set (Continued)

Command	Type	Description	Hex	Data Bytes	Note
SBPR	Interrupt	Set Breakpoint, Relative	21	4	1
MSKI	Interrupt	Mask Interrupts	1C	2	1
RSTI	Interrupt	Reset Interrupts	1D	2	1
LFIL	Filter	Load Filter Parameters	1E	2 to 10	1
UDF	Filter	Update Filter	04	0	1
LTRJ	Trajectory	Load Trajectory	1F	2 to 14	1
STT	Trajectory	Start Motion	01	0	3
RDSTAT	Report	Read Status Byte	None	1	1, 4
RDSIGS	Report	Read Signals Register	0C	2	1
RDIP	Report	Read Index Position	09	4	1
RDDP	Report	Read Desired Position	08	4	1
RDRP	Report	Read Real Position	0A	4	1
RDDV	Report	Read Desired Velocity	07	4	1
RDRV	Report	Read Real Velocity	0B	2	1
RDSUM	Report	Read Integration Sum	0D	2	1

Note 4: Commands may be executed "On the Fly" during motion.

Note 5: Commands not applicable to execution during motion.

Note 6: Command may be executed during motion if acceleration parameter was not changed.

Note 7: Command needs no code because the command port status-byte read is totally supported by hardware.

User Command Set

GENERAL

The following paragraphs describe the user command set of the LM628. Some of the commands can be issued alone and some require a supporting data structure. As examples, the command STT (STarT motion) does not require additional data; command LFIL (Load FILter parameters) requires additional data (derivative-term sampling interval and/or filter parameters).

Commands are categorized by function: initialization, interrupt control, filter control, trajectory control, and data reporting. The commands are listed in *Table 2* and described in the following paragraphs. Along with each command name is its command-byte code, the number of accompanying data bytes that are to be written (or read), and a comment as to whether the command is executable during motion.

Initialization Commands

The following four LM628 user commands are used primarily to initialize the system for use.

RESET COMMAND: RESET THE LM628

Command Code: 00 Hex
Data Bytes: None
Executable During Motion: Yes

This command (and the hardware reset input, Pin 27) results in setting the following data items to zero: filter coefficients and their input buffers, trajectory parameters and their input buffers, and the motor control output. A zero motor control output is a half-scale, offset-binary code: (80 hex for the 8-bit output mode; 800 hex for 12-bit mode). During reset, the DAC port outputs 800 hex to "zero" a 12-bit DAC and reverts to 80 hex to "zero" an 8-bit DAC. The command also clears five of the six interrupt masks (only the SBPA/SBPR interrupt is masked), sets the output port size to 8 bits, and defines the current absolute position as home. Reset, which may be executed at any time, will be completed in less than 1.5 ms. Also see commands PORT8 and PORT12.

PORT8 COMMAND: SET OUTPUT PORT SIZE TO 8 BITS

Command Code: 05 Hex
Data Bytes: None
Executable During Motion: Not Applicable

The default output port size of the LM628 is 8 bits; so the PORT8 command need not be executed when using an 8-bit DAC. This command must not be executed when using a 12-bit converter; it will result in erratic, unpredictable motor behavior. The 8-bit output port size is the required selection when using the LM629, the PWM-output version of the LM628.

PORT12 COMMAND: SET OUTPUT PORT SIZE TO 12 BITS

Command Code: 06 Hex
Data Bytes: None
Executable During Motion: Not Applicable

When a 12-bit DAC is used, command PORT12 should be issued very early in the initialization process. Because use of this command is determined by system hardware, there is only one foreseen reason to execute it later: if the RESET command is issued (because an 8-bit output would then be selected as the default) command PORT12 should be immediately executed. This command must not be issued when using an 8-bit converter or the LM629, the PWM-output version of the LM628.

Initialization Commands (Continued)

DFH COMMAND: DEFINE HOME

Command Code: 02 Hex
Data Bytes: None
Executable During Motion: Yes

This command declares the current position as “home”, or absolute position 0 (Zero). If DFH is executed during motion it will not affect the stopping position of the on-going move unless command STT is also executed.

Interrupt Control Commands

The following seven LM628 user commands are associated with conditions which can be used to interrupt the host computer. In order for any of the potential interrupt conditions to actually interrupt the host via Pin 17, the corresponding bit in the interrupt mask data associated with command MSKI must have been set to logic high (the non-masked state).

The identity of all interrupts is made known to the host via reading and parsing the status byte. Even if all interrupts are masked off via command MSKI, the state of each condition is still reflected in the status byte. This feature facilitates polling the LM628 for status information, as opposed to interrupt driven operation.

SIP COMMAND: SET INDEX POSITION

Command Code: 03 Hex
Data Bytes: None
Executable During Motion: Yes

After this command is executed, the absolute position which corresponds to the occurrence of the next index pulse input will be recorded in the index register, and bit 3 of the status byte will be set to logic high. The position is recorded when both encoder-phase inputs and the index pulse input are logic low. This register can then be read by the user (see description for command RDIP) to facilitate aligning the definition of home position (see description of command DFH) with an index pulse. The user can also arrange to have the LM628 interrupt the host to signify that an index pulse has occurred. See the descriptions for commands MSKI and RSTI.

LPEI COMMAND: LOAD POSITION ERROR FOR INTERRUPT

Command Code: 1B Hex
Data Bytes: Two
Data Range: 0000 to 7FFF Hex
Executable During Motion: Yes

An excessive position error (the output of the loop summing junction) can indicate a serious system problem; e.g., a stalled rotor. Instruction LPEI allows the user to input a threshold for position error detection. Error detection occurs when the absolute magnitude of the position error exceeds the threshold, which results in bit 5 of the status byte being set to logic high. If it is desired to also stop (turn off) the motor upon detecting excessive position error, see command LPES, below. The first byte of threshold data written with command LPEI is the more significant. The user can have the LM628 interrupt the host to signify that an excessive position error has occurred. See the descriptions for commands MSKI and RSTI.

LPES COMMAND: LOAD POSITION ERROR FOR STOPPING

Command Code: 1A Hex
Data Bytes: Two
Data Range: 0000 to 7FFF Hex
Executable During Motion: Yes

Instruction LPES is essentially the same as command LPEI above, but adds the feature of turning off the motor upon detecting excessive position error. The motor drive is not actually switched off, it is set to half-scale, the offset-binary code for zero. As with command LPEI, bit 5 of the status byte is also set to logic high. The first byte of threshold data written with command LPES is the more significant. The user can have the LM628 interrupt the host to signify that an excessive position error has occurred. See the descriptions for commands MSKI and RSTI.

SBPA COMMAND:

Command Code: 20 Hex
Data Bytes: Four
Data Range: C0000000 to 3FFFFFFF Hex
Executable During Motion: Yes

This command enables the user to set a breakpoint in terms of absolute position. Bit 6 of the status byte is set to logic high when the breakpoint position is reached. This condition is useful for signaling trajectory and/or filter parameter updates. The user can also arrange to have the LM628 interrupt the host to signify that a breakpoint position has been reached. See the descriptions for commands MSKI and RSTI.

SBPR COMMAND:

Command Code: 21 Hex
Data Bytes: Four
Data Range: See Text
Executable During Motion: Yes

This command enables the user to set a breakpoint in terms of relative position. As with command SBPA, bit 6 of the status byte is set to logic high when the breakpoint position (relative to the current commanded target position) is reached. The relative breakpoint input value must be such that when this value is added to the target position the result remains within the absolute position range of the system (C0000000 to 3FFFFFFF hex). This condition is useful for signaling trajectory and/or filter parameter updates. The user can also arrange to have the LM628 interrupt the host to signify that a breakpoint position has been reached. See the descriptions for commands MSKI and RSTI.

MSKI COMMAND: MASK INTERRUPTS

Command Code: 1C Hex
Data Bytes: Two
Data Range: See Text
Executable During Motion: Yes

The MSKI command lets the user determine which potential interrupt condition(s) will interrupt the host. Bits 1 through 6 of the status byte are indicators of the six conditions which are candidates for host interrupt(s). When interrupted, the host then reads the status byte to learn which condition(s) occurred. Note that the MSKI command is immediately followed by two data bytes. Bits 1 through 6 of the second (less significant) byte written determine the masked/unmasked status of each potential interrupt. Any zero(s) in this 6-bit

Interrupt Control Commands

(Continued)

field will mask the corresponding interrupt(s); any one(s) enable the interrupt(s). Other bits comprising the two bytes have no effect. The mask controls only the host interrupt process; reading the status byte will still reflect the actual conditions independent of the mask byte. See *Table 3*.

TABLE 3. Mask and Reset Bit Allocations for Interrupts

Bit Position	Function
Bits 15 thru 7	Not Used
Bit 6	Breakpoint Interrupt
Bit 5	Position-Error Interrupt
Bit 4	Wrap-Around Interrupt
Bit 3	Index-Pulse Interrupt
Bit 2	Trajectory-Complete Interrupt
Bit 1	Command-Error Interrupt
Bit 0	Not Used

RSTI COMMAND: RESET INTERRUPTS

Command Code: 1D Hex
Data Bytes: Two
Data Range: See Text
Executable During Motion: Yes

When one of the potential interrupt conditions of *Table 3* occurs, command RSTI is used to reset the corresponding interrupt flag bit in the status byte. The host may reset one or all flag bits. Resetting them one at a time allows the host to service them one at a time according to a priority programmed by the user. As in the MSKI command, bits 1 through 6 of the second (less significant) byte correspond to the potential interrupt conditions shown in *Table 3*. Also see description of RDSTAT command. Any zero(s) in this 6-bit field reset the corresponding interrupt(s). The remaining bits have no effect.

Filter Control Commands

The following two LM628 user commands are used for setting the derivative-term sampling interval, for adjusting the filter parameters as required to tune the system, and to control the timing of these system changes.

LFIL COMMAND: LOAD FILTER PARAMETERS

Command Code: 1E Hex
Data Bytes: Two to Ten
Data Ranges…
Filter Control Word: See Text
Filter Coefficients: 0000 to 7FFF Hex (Pos Only)
Integration Limit: 0000 to 7FFF Hex (Pos Only)
Executable During Motion: Yes

The filter parameters (coefficients) which are written to the LM628 to control loop compensation are: kp, ki, kd, and il (integration limit). The integration limit (il) constrains the contribution of the integration term

$$\left[ki * \sum_{N=0}^{n} e(n) \right]$$

(see Eq. 1) to values equal to or less than a user-defined maximum value; this capability minimizes integral or reset "wind-up" (an overshooting effect of the integral action). The positive-only input value is compared to the absolute magnitude of the integration term; when the magnitude of integration term value exceeds il, the il value (with appropriate sign) is substituted for the integration term value.

The derivative-term sampling interval is also programmable via this command. After writing the command code, the first two data bytes that are written specify the derivative-term sampling interval and which of the four filter parameters is/are to be written via any forthcoming data bytes. The first byte written is the more significant. Thus the two data bytes constitute a filter control word that informs the LM628 as to the nature and number of any following data bytes. See *Table 4*.

TABLE 4. Filter Control word Bit Allocation

Bit Position	Function
Bit 15	Derivative Sampling Interval Bit 7
Bit 14	Derivative Sampling Interval Bit 6
Bit 13	Derivative Sampling Interval Bit 5
Bit 12	Derivative Sampling Interval Bit 4
Bit 11	Derivative Sampling Interval Bit 3
Bit 10	Derivative Sampling Interval Bit 2
Bit 9	Derivative Sampling Interval Bit 1
Bit 8	Derivative Sampling Interval Bit 0
Bit 7	Not Used
Bit 6	Not Used
Bit 5	Not Used
Bit 4	Not Used
Bit 3	Loading kp Data
Bit 2	Loading ki Data
Bit 1	Loading kd Data
Bit 0	Loading il Data

Bits 8 through 15 select the derivative-term sampling interval. See *Table 5*. The user must locally save and restore these bits during successive writes of the filter control word.

Bits 4 through 7 of the filter control word are not used.

Bits 0 to 3 inform the LM628 as to whether any or all of the filter parameters are about to be written. The user may choose to update any or all (or none) of the filter parameters. Those chosen for updating are so indicated by logic one(s) in the corresponding bit position(s) of the filter control word.

The data bytes specified by and immediately following the filter control word are written in pairs to comprise 16-bit words. The order of sending the data words to the LM628 corresponds to the descending order shown in the above description of the filter control word; i.e., beginning with kp, then ki, kd and il. The first byte of each word is the more-significant byte. Prior to writing a word (byte pair) it is necessary to check the busy bit in the status byte for readiness. The required data is written to the primary buffers of a double-buffered scheme by the above described operations;

Filter Control Commands (Continued)

it is not transferred to the secondary (working) registers until the UDF command is executed. This fact can be used advantageously; the user can input numerous data ahead of their actual use. This simple pipeline effect can relieve potential host computer data communications bottlenecks, and facilitates easier synchronization of multiple-axis controls.

UDF COMMAND: UPDATE FILTER

Command Code:	04 Hex
Data Bytes:	None
Executable During Motion:	Yes

The UDF command is used to update the filter parameters, the specifics of which have been programmed via the LFIL command. Any or all parameters (derivative-term sampling interval, kp, ki, kd, and/or il) may be changed by the appropriate command(s), but command UDF must be executed to affect the change in filter tuning. Filter updating is synchronized with the calculations to eliminate erratic or spurious behavior.

Trajectory Control Commands

The following two LM628 user commands are used for setting the trajectory control parameters (position, velocity, acceleration), mode of operation (position or velocity), and direction (velocity mode only) as required to describe a desired motion or to select the mode of a manually directed stop, and to control the timing of these system changes.

LTRJ COMMAND: LOAD TRAJECTORY PARAMETERS

Command Code:	1F Hex
Data Bytes:	Two to Fourteen
Data Ranges…	
Trajectory Control Word:	See Text
Position:	C0000000 to 3FFFFFFF Hex
Velocity:	00000000 to 3FFFFFFF Hex (Pos Only)
Acceleration:	00000000 to 3FFFFFFF Hex (Pos Only)
Executable During Motion:	Conditionally, See Text

TABLE 5. Derivative-Term Sampling Interval Selection Codes

	Bit Position								Selected Derivative
	15	14	13	12	11	10	9	8	Sampling Interval
	0	0	0	0	0	0	0	0	256 µs
	0	0	0	0	0	0	0	1	512 µs
	0	0	0	0	0	0	1	0	768 µs
	0	0	0	0	0	0	1	1	1024 µs, etc…
thru	1	1	1	1	1	1	1	1	65,536 µs

Note 8: Sampling intervals shown are when using an 8.0 MHz clock. The 256 corresponds to 2048/8 MHz; sample intervals must be scaled for other clock frequencies.

The trajectory control parameters which are written to the LM628 to control motion are: acceleration, velocity, and position. In addition, indications as to whether these three parameters are to be considered as absolute or relative inputs, selection of velocity mode and direction, and manual stopping mode selection and execution are programmable via this command. After writing the command code, the first two data bytes that are written specify which parameter(s) is/are being changed. The first byte written is the more significant. Thus the two data bytes constitute a trajectory control word that informs the LM628 as to the nature and number of any following data bytes. See *Table 6*.

TABLE 6. Trajectory Control Word Bit Allocation

Bit Position	Function
Bit 15	Not Used
Bit 14	Not Used
Bit 13	Not Used
Bit 12	Forward Direction (Velocity Mode Only)
Bit 11	Velocity Mode
Bit 10	Stop Smoothly (Decelerate as Programmed)
Bit 9	Stop Abruptly (Maximum Deceleration)
Bit 8	Turn Off Motor (Output Zero Drive)
Bit 7	Not Used
Bit 6	Not Used
Bit 5	Acceleration Will Be Loaded
Bit 4	Acceleration Data Is Relative
Bit 3	Velocity Will Be Loaded
Bit 2	Velocity Data Is Relative
Bit 1	Position Will Be Loaded
Bit 0	Position Data Is Relative

Bit 12 determines the motor direction when in the velocity mode. A logic one indicates forward direction. This bit has no effect when in position mode.

Bit 11 determines whether the LM628 operates in velocity mode (Bit 11 logic one) or position mode (Bit 11 logic zero).

Bits 8 through 10 are used to select the method of *manually stopping* the motor. These bits are *not* provided for one to merely specify the desired *mode* of stopping, in position mode operations, normal stopping is always smooth and

Trajectory Control Commands

(Continued)

occurs automatically at the end of the specified trajectory. Under exceptional circumstances it may be desired to manually intervene with the trajectory generation process to affect a premature stop. In velocity mode operations, however, the normal means of stopping *is* via bits 8 through 10 (usually bit 10). Bit 8 is set to logic one to stop the motor by turning off motor drive output (outputting the appropriate offset-binary code to apply zero drive to the motor); bit 9 is set to one to stop the motor abruptly (at maximum available acceleration, by setting the target position equal to the current position); and bit 10 is set to one to stop the motor smoothly by using the current user-programmed acceleration value. Bits 8 through 10 are to be used *exclusively;* only one bit should be a logic one at any time.

Bits 0 through 5 inform the LM628 as to whether any or all of the trajectory controlling parameters are about to be written, and whether the data should be interpreted as absolute or relative. The user may choose to update any or all (or none) of the trajectory parameters. Those chosen for updating are so indicated by logic one(s) in the corresponding bit position(s). Any parameter may be changed while the motor is in motion; however, if acceleration is changed then the next STT command must not be issued until the LM628 has completed the current move or has been manually stopped.

The data bytes specified by and immediately following the trajectory control word are written in pairs which comprise 16-bit words. Each data item (parameter) requires two 16-bit words; the word and byte order is most-to-least significant. The order of sending the parameters to the LM628 corresponds to the descending order shown in the above description of the trajectory control word; i.e., beginning with acceleration, then velocity, and finally position.

Acceleration and velocity are 32 bits, positive only, but range only from 0 (00000000 hex) to $[2^{30}]-1$ (3FFFFFFF hex). The bottom 16 bits of both acceleration and velocity are scaled as fractional data; therefore, the least-significant integer data bit for these parameters is bit 16 (where the bits are numbered 0 through 31). To determine the coding for a given velocity, for example, one multiplies the desired velocity (in counts per sample interval) times 65,536 and converts the result to binary. The units of acceleration are counts per sample per sample. The value loaded for acceleration must not exceed the value loaded for velocity. Position is a signed, 32-bit integer, but ranges only from $-[2^{30}]$ (C0000000 hex) to $[2^{30}]-1$ (3FFFFFFF Hex).

The required data is written to the primary buffers of a double-buffered scheme by the above described operations; it is not transferred to the secondary (working) registers until the STT command is executed. This fact can be used advantageously; the user can input numerous data ahead of their actual use. This simple pipeline effect can relieve potential host computer data communications bottlenecks, and facilitates easier synchronization of multiple-axis controls.

STT COMMAND: START MOTION CONTROL

Command Code:	01 Hex
Data Bytes:	None
Executable During Motion:	Yes, if acceleration has not been changed

The STT command is used to execute the desired trajectory, the specifics of which have been programmed via the LTRJ command. Synchronization of multi-axis control (to within one sample interval) can be arranged by loading the required trajectory parameters for each (and every) axis and then simultaneously issuing a single STT command to all axes. This command may be executed at any time, unless the acceleration value has been changed and a trajectory has not been completed or the motor has not been manually stopped. If STT is issued during motion and acceleration has been changed, a command error interrupt will be generated and the command will be ignored.

Data Reporting Commands

The following seven LM628 user commands are used to obtain data from various registers in the LM628. Status, position, and velocity information are reported. With the exception of RDSTAT, the data is read from the LM628 data port after first writing the corresponding command to the command port.

RDSTAT COMMAND: READ STATUS BYTE

Command Code:	None
Byte Read:	One
Data Range:	See Text
Executable During Motion:	Yes

The RDSTAT command is really not a command, but is listed with the other commands because it is used very frequently to control communications with the host computer. There is no identification code; it is directly supported by the hardware and may be executed at any time. The single-byte status read is selected by placing $\overline{CS}$, $\overline{PS}$ and $\overline{RD}$ at logic zero. See *Table 7*.

TABLE 7. Status Byte Bit Allocation

Bit Position	Function
Bit 7	Motor Off
Bit 6	Breakpoint Reached [Interrupt]
Bit 5	Excessive Position Error [Interrupt]
Bit 4	Wraparound Occurred [Interrupt]
Bit 3	Index Pulse Observed [Interrupt]
Bit 2	Trajectory Complete [Interrupt]
Bit 1	Command Error [Interrupt]
Bit 0	Busy Bit

Bit 7, the motor-off flag, is set to logic one when the motor drive output is off (at the half-scale, offset-binary code for zero). The motor is turned off by any of the following conditions: power-up reset, command RESET, excessive position error (if command LPES had been executed), or when command LTRJ is used to manually stop the motor via turning the motor off. Note that when bit 7 is set in conjunction with command LTRJ for producing a manual, motor-off stop, the actual setting of bit 7 does not occur until command STT is issued to affect the stop. Bit 7 is cleared by command STT, except as described in the previous sentence.

Bit 6, the breakpoint-reached interrupt flag, is set to logic one when the position breakpoint loaded via command SBPA or SBPR has been exceeded. The flag is functional independent of the host interrupt mask status. Bit 6 is cleared via command RSTI.

Bit 5, the excessive-position-error interrupt flag, is set to logic one when a position-error interrupt condition exists. This occurs when the error threshold loaded via command

Data Reporting Commands

(Continued)

LPEI or LPES has been exceeded. The flag is functional independent of the host interrupt mask status. Bit 5 is cleared via command RSTI.

Bit 4, the wraparound interrupt flag, is set to logic one when a numerical "wraparound" has occurred. To "wraparound" means to exceed the position address space of the LM628, which could occur during velocity mode operation. If a wraparound has occurred, then position information will be in error and this interrupt helps the user to ensure position data integrity. The flag is functional independent of the host interrupt mask status. Bit 4 is cleared via command RSTI.

Bit 3, the index-pulse acquired interrupt flag, is set to logic one when an index pulse has occurred (if command SIP had been executed) and indicates that the index position register has been updated. The flag is functional independent of the host interrupt mask status. Bit 3 is cleared by command RSTI.

Bit 2, the trajectory complete interrupt flag, is set to logic one when the trajectory programmed by the LTRJ command and initiated by the STT command has been completed. Because of overshoot or a limiting condition (such as commanding the velocity to be higher than the motor can achieve), the motor may not yet be at the final commanded position. This bit is the logical OR of bits 7 and 10 of the Signals Register, see command RDSIGS below. The flag functions independently of the host interrupt mask status. Bit 2 is cleared via command RSTI.

Bit 1, the command-error interrupt flag, is set to logic one when the user attempts to read data when a write was appropriate (or vice versa). The flag is functional independent of the host interrupt mask status. Bit 1 is cleared via command RSTI.

Bit 0, the busy flag, is frequently tested by the user (via the host computer program) to determine the busy/ready status prior to writing and reading any data. Such writes and reads may be executed only when bit 0 is logic zero (not busy). Any command or data writes when the busy bit is high will be ignored. Any data reads when the busy bit is high will read the current contents of the I/O port buffers, not the data expected by the host. Such reads or writes (with the busy bit high) will not generate a command-error interrupt.

RDSIGS COMMAND: READ SIGNALS REGISTER

Command Code:	0C Hex
Bytes Read:	Two
Data Range:	See Text
Executable During Motion:	Yes

The LM628 internal "signals" register may be read using this command. The first byte read is the more significant. The less significant byte of this register (with the exception of bit 0) duplicates the status byte. See *Table 8*.

TABLE 8. Signals Register Bit Allocation

Bit Position	Function
Bit 15	Host Interrupt
Bit 14	Acceleration Loaded (But Not Updated)
Bit 13	UDF Executed (But Filter Not yet Updated)
Bit 12	Forward Direction
Bit 11	Velocity Mode
Bit 10	On Target
Bit 9	Turn Off upon Excessive Position Error
Bit 8	Eight-Bit Output Mode
Bit 7	Motor Off
Bit 6	Breakpoint Reached [Interrupt]
Bit 5	Excessive Position Error [Interrupt]
Bit 4	Wraparound Occurred [Interrupt]
Bit 3	Index Pulse Acquired [Interrupt]
Bit 2	Trajectory Complete [Interrupt]
Bit 1	Command Error [Interrupt]
Bit 0	Acquire Next Index (SIP Executed)

Bit 15, the host interrupt flag, is set to logic one when the host interrupt output (Pin 17) is logic one. Pin 17 is set to logic one when any of the six host interrupt conditions occur (if the corresponding interrupt has not been masked). Bit 15 (and Pin 17) are cleared via command RSTI.

Bit 14, the acceleration-loaded flag, is set to logic one when acceleration data is written to the LM628. Bit 14 is cleared by the STT command.

Bit 13, the UDF-executed flag, is set to logic one when the UDF command is executed. Because bit 13 is cleared at the end of the sampling interval in which it has been set, this signal is very short-lived and probably not very profitable for monitoring.

Bit 12, the forward direction flag, is meaningful only when the LM628 is in velocity mode. The bit is set to logic one to indicate that the desired direction of motion is "forward"; zero indicates "reverse" direction. Bit 12 is set and cleared via command LTRJ. The actual setting and clearing of bit 12 does not occur until command STT is executed.

Bit 11, the velocity mode flag, is set to logic one to indicate that the user has selected (via command LTRJ) velocity mode. Bit 11 is cleared when position mode is selected (via command LTRJ). The actual setting and clearing of bit 11 does not occur until command STT is executed.

Bit 10, the on-target flag, is set to logic one when the trajectory generator has completed its functions for the last-issued STT command. Bit 10 is cleared by the next STT command.

Bit 9, the turn-off on-error flag, is set to logic one when command LPES is executed. Bit 9 is cleared by command LPEI.

Bit 8, the 8-bit output flag, is set to logic one when the LM628 is reset, or when command PORT8 is executed. Bit 8 is cleared by command PORT12.

Bits 0 through 7 replicate the status byte (see), with the exception of bit 0. Bit 0, the acquire next index flag, is set to logic one when command SIP is executed; it then remains set until the next index pulse occurs.

RDIP COMMAND: READ INDEX POSITION

Command Code:	09 Hex
Bytes Read:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command reads the position recorded in the index register. Reading the index register can be part of a system error checking scheme. Whenever the SIP command is executed, the new index position minus the old index position,

Data Reporting Commands

(Continued)

divided by the incremental encoder resolution (encoder lines times four), should always be an integral number. The RDIP command facilitates acquiring these data for host-based calculations. The command can also be used to identify/verify home or some other special position. The bytes are read in most-to-least significant order.

RDDP COMMAND: READ DESIRED POSITION

Command Code:	08 Hex
Bytes Read:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command reads the instantaneous desired (current *temporal*) position output of the profile generator. This is the "setpoint" input to the position-loop summing junction. The bytes are read in most-to-least significant order.

RDRP COMMAND: READ REAL POSITION

Command Code:	0A Hex
Bytes Read:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command reads the current actual position of the motor. This is the feedback input to the loop summing junction. The bytes are read in most-to-least significant order.

RDDV COMMAND: READ DESIRED VELOCITY

Command Code:	07 Hex
Bytes Read:	Four
Data Range:	C0000001 to 3FFFFFFF
Executable During Motion:	Yes

This command reads the integer and fractional portions of the instantaneous desired (current *temporal*) velocity, as used to generate the desired position profile. The bytes are read in most-to-least significant order. The value read is properly scaled for numerical comparison with the user-supplied (commanded) velocity; however, because the two least-significant bytes represent *fractional* velocity, only the two most-significant bytes are appropriate for comparison with the data obtained via command RDRV (see below). Also note that, although the velocity *input* data is constrained to positive numbers (see command LTRJ), the data returned by command RDDV represents a *signed* quantity where negative numbers represent operation in the reverse direction.

RDRV COMMAND: READ REAL VELOCITY

Command Code:	0B Hex
Bytes Read:	Two
Data Range:	C000 to 3FFF Hex, See Text
Executable During Motion:	Yes

This command reads the *integer* portion of the instantaneous actual velocity of the motor. The internally maintained fractional portion of velocity is not reported because the reported data is derived by reading the incremental encoder, which produces only integer data. For comparison with the result obtained by executing command RDDV (or the user-supplied input value), the value returned by command RDRV must be multiplied by 2^{16} (shifted left 16 bit positions). Also, as with command RDDV above, data returned by command RDRV is a *signed* quantity, with negative values representing reverse-direction motion.

RDSUM COMMAND: READ INTEGRATION-TERM SUMMATION VALUE

Command Code:	0D Hex
Bytes Read:	Two
Data Range:	00000 Hex to ± the Current Value of the Integration Limit
Executable During Motion:	Yes

This command reads the value to which the integration term has accumulated. The ability to read this value may be helpful in initially or adaptively tuning the system.

Typical Applications

PROGRAMMING LM628 HOST HANDSHAKING (INTERRUPTS)

A few words regarding the LM628 host handshaking will be helpful to the system programmer. As indicated in various portions of the above text, the LM628 handshakes with the host computer in two ways: via the host interrupt output (Pin 17), or via polling the status byte for "interrupt" conditions. When the hardwired interrupt is used, the status byte is also read and parsed to determine which of six possible conditions caused the interrupt.

When using the hardwired interrupt it is very important that the host interrupt service routine does not interfere with a command sequence which might have been in progress when the interrupt occurred. If the host interrupt service routine were to issue a command to the LM628 while it is in the middle of an ongoing command sequence, the ongoing command will be aborted (which could be detrimental to the application).

Two approaches exist for avoiding this problem. If one is using hardwired interrupts, they should be disabled at the host prior to issuing any LM628 command sequence, and re-enabled after each command sequence. The second approach is to avoid hardwired interrupts and poll the LM628 status byte for "interrupt" status. The status byte always reflects the interrupt-condition status, independent of whether or not the interrupts have been masked.

TYPICAL HOST COMPUTER/PROCESSOR INTERFACE

The LM628 is interfaced with the host computer/processor via an 8-bit parallel bus. *Figure 12* shows such an interface and a minimum system configuration.

As shown in *Figure 12*, the LM628 interfaces with the host data, address and control lines. The address lines are decoded to generate the LM628 $\overline{CS}$ input; the host address LSB directly drives the LM628 $\overline{PS}$ input. *Figure 12* also shows an 8-bit DAC and an LM12 Power Op Amp interfaced to the LM628.

LM628 AND HIGH PERFORMANCE CONTROLLER (HPC)INTERFACE

Figure 13 shows the LM628 interfaced to a National HPC High Performance Controller. The delay and logic associated with the $\overline{WR}$ line is used to effectively increase the write-data hold time of the HPC (as seen at the LM628) by causing the $\overline{WR}$ pulse to rise early. Note that the HPC CK2 output

Typical Applications (Continued)

provides the clock for the LM628. The 74LS245 is used to decrease the read-data hold time, which is necessary when interfacing to fast host busses.

INTERFACING A 12-BIT DAC

Figure 14 illustrates use of a 12-bit DAC with the LM628. The 74LS378 hex gated-D flip-flop and an inverter demultiplex the 12-bit output. DAC offset must be adjusted to minimize DAC linearity and monotonicity errors. Two methods exist for making this adjustment. If the DAC1210 has been socketed, remove it and temporarily connect a 15 kΩ resistor between Pins 11 and 13 of the DAC socket (Pins 2 and 6 of the LF356) and adjust the 25 kΩ potentiometer for 0V at Pin 6 of the LF356.

If the DAC is not removable, the second method of adjustment requires that the DAC1210 inputs be presented an all-zeros code. This can be arranged by commanding the appropriate move via the LM628, but with no feedback from the system encoder. When the all-zeros code is present, adjust the pot for 0V at Pin 6 of the LF356.

A MONOLITHIC LINEAR DRIVE USING LM12 POWER OP AMP

Figure 15 shows a motor-drive amplifier built using the LM12 Power Operational Amplifier. This circuit is very simple and can deliver up to 8A at 30V (using the LM12L/LM12CL). Resistors R1 and R2 should be chosen to set the gain to provide maximum output voltage consistent with maximum input voltage. This example provides a gain of 2.2, which allows for amplifier output saturation at ±22V with a ±10V input, assuming power supply voltages of ±30V. The amplifier gain should not be higher than necessary because the system is non-linear when saturated, and because gain should be controlled by the LM628. The LM12 can also be configured as a current driver, see 1987 Linear Databook, Vol. 1, p. 2–280.

TYPICAL PWM MOTOR DRIVE INTERFACES

Figure 16 shows an LM18298 dual full-bridge driver interfaced to the LM629 PWM outputs to provide a switch-mode power amplifier for driving small brush/commutator motors.

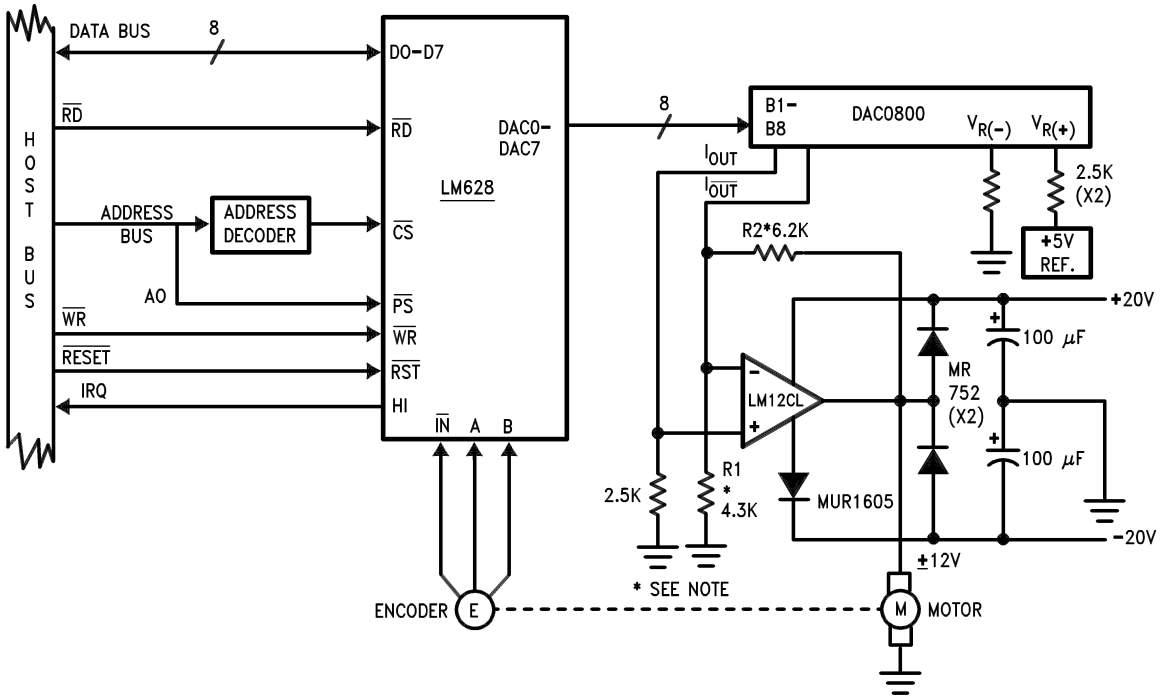
Incremental Encoder Interface

The incremental (position feedback) encoder interface consists of three lines: Phase A (Pin 2), Phase B (Pin 3), and Index (Pin 1). The index pulse output is not available on some encoders. The LM628 will work with both encoder types, but commands SIP and RDIP will not be meaningful without an index pulse (or alternative input for this input … be sure to tie Pin 1 high if not used).

Some consideration is merited relative to use in high Gaussian-noise environments. If noise is added to the encoder inputs (either or both inputs) and is such that it is not sustained until the next encoder transition, the LM628 decoder logic will reject it. Noise that mimics quadrature counts or persists through encoder transitions must be eliminated by appropriate EMI design.

Simple digital “filtering” schemes merely reduce susceptibility to noise (there will always be noise pulses longer than the filter can eliminate). Further, any noise filtering scheme reduces decoder bandwidth. In the LM628 it was decided (since simple filtering does not eliminate the noise problem) to not include a noise filter in favor of offering maximum possible decoder bandwidth. Attempting to drive encoder signals too long a distance with simple TTL lines can also be a source of “noise” in the form of signal degradation (poor risetime and/or ringing). This can also cause a system to lose positional integrity. Probably the most effective countermeasure to noise induction can be had by using balanced-line drivers and receivers on the encoder inputs. *Figure 17* shows circuitry using the DS26LS31 and DS26LS32.

Typical Applications (Continued)



Note:

$$A_V = \frac{R1 + R2}{R1} \approx 2.4$$

$$\frac{R1 \times R2}{R1 + R2} \triangleq 2.5k$$

FIGURE 12. Host Interface and Minimum System Configuration

Typical Applications (Continued)

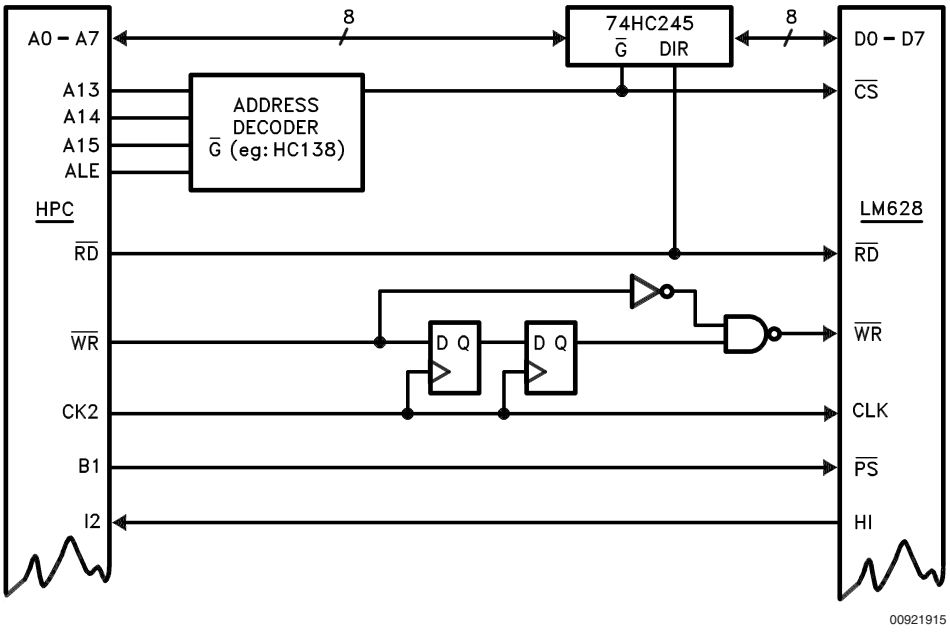
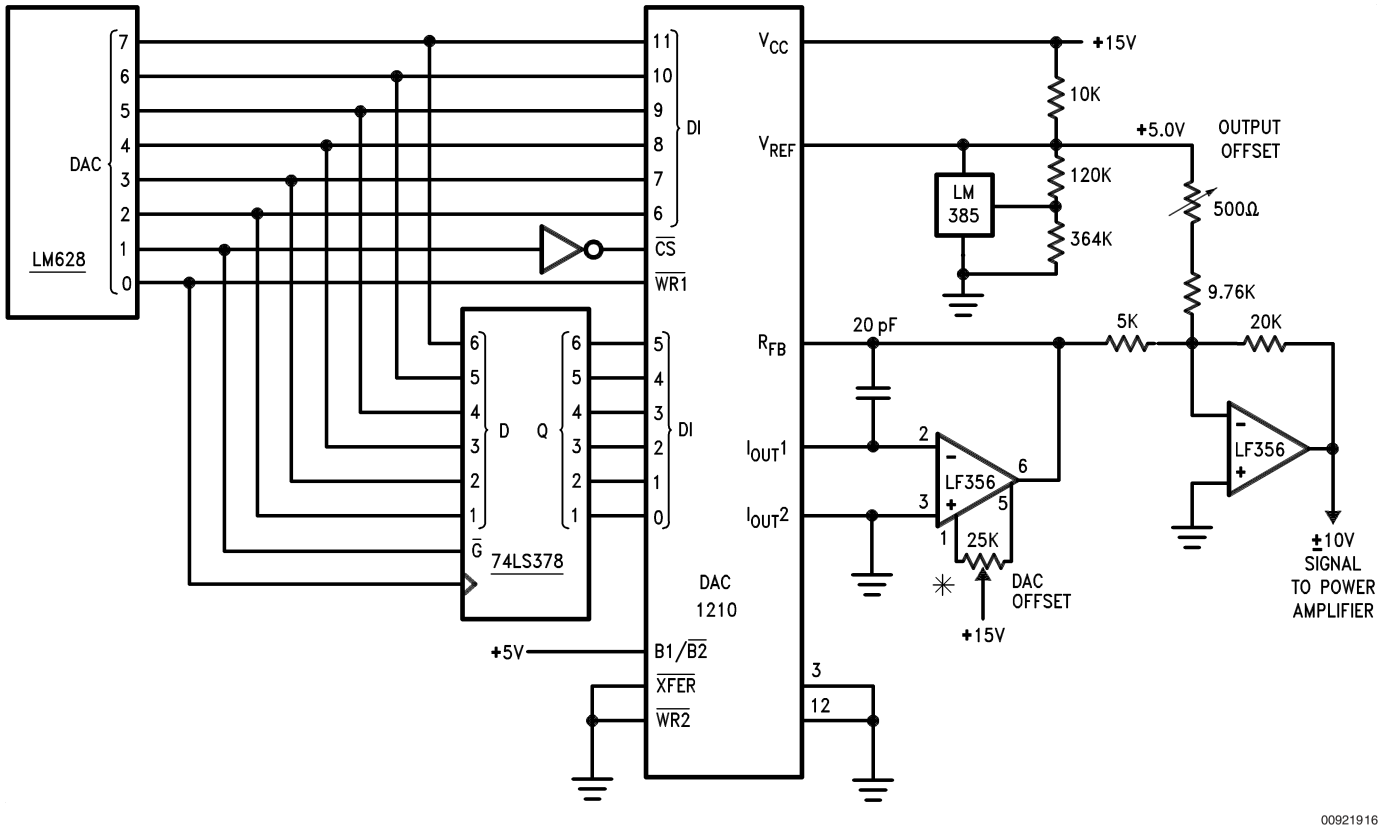


FIGURE 13. LM628 and HPC Interface

Typical Applications (Continued)



*DAC offset must be adjusted to minimize DAC linearity and monotonicity errors. See text.

FIGURE 14. Interfacing a 12-Bit DAC and LM628

Typical Applications (Continued)

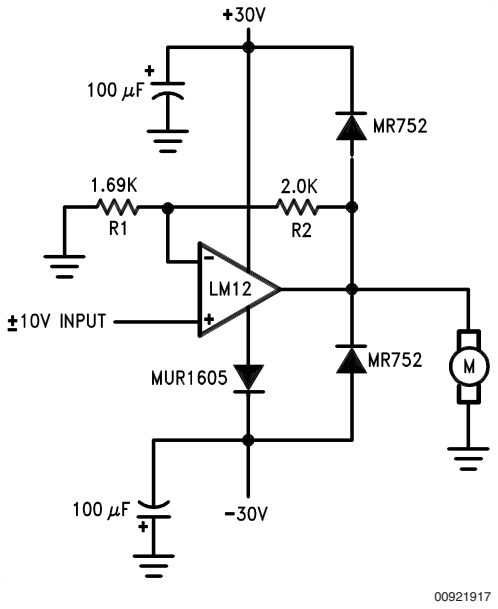


FIGURE 15. Driving a Motor with the LM12 Power Op Amp

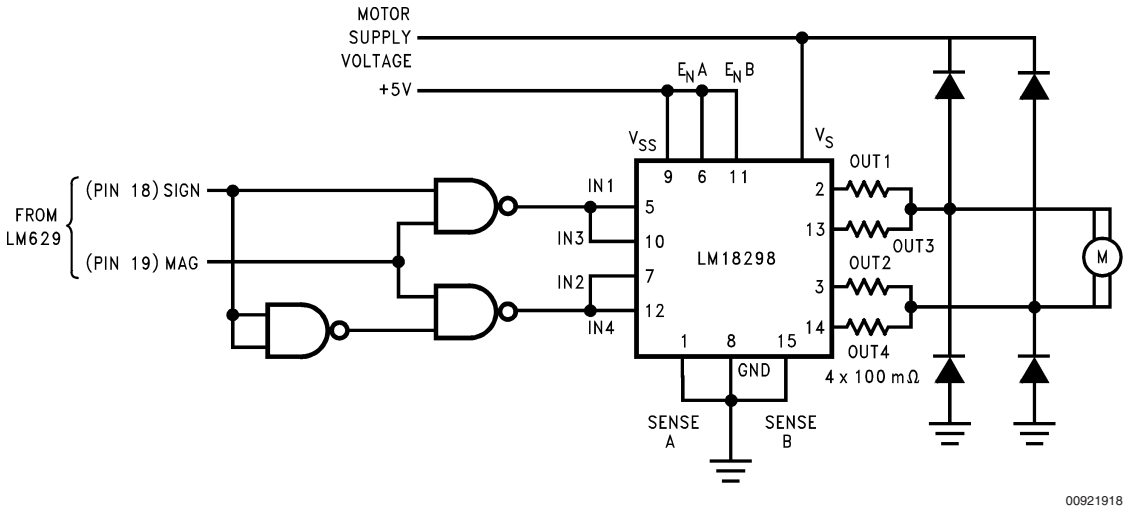


FIGURE 16. PWM Drive for Brush/Commutator Motors

Typical Applications (Continued)

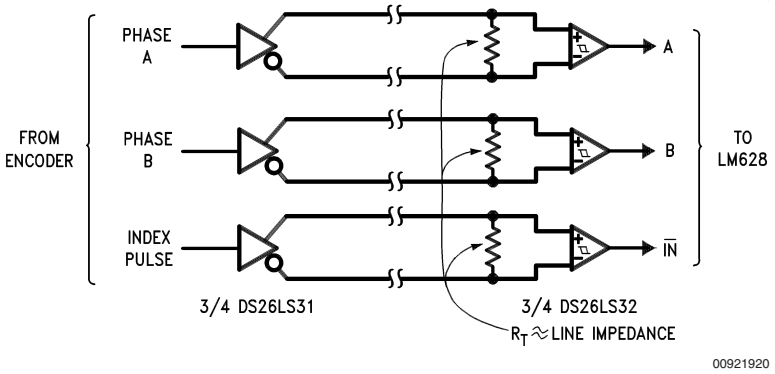
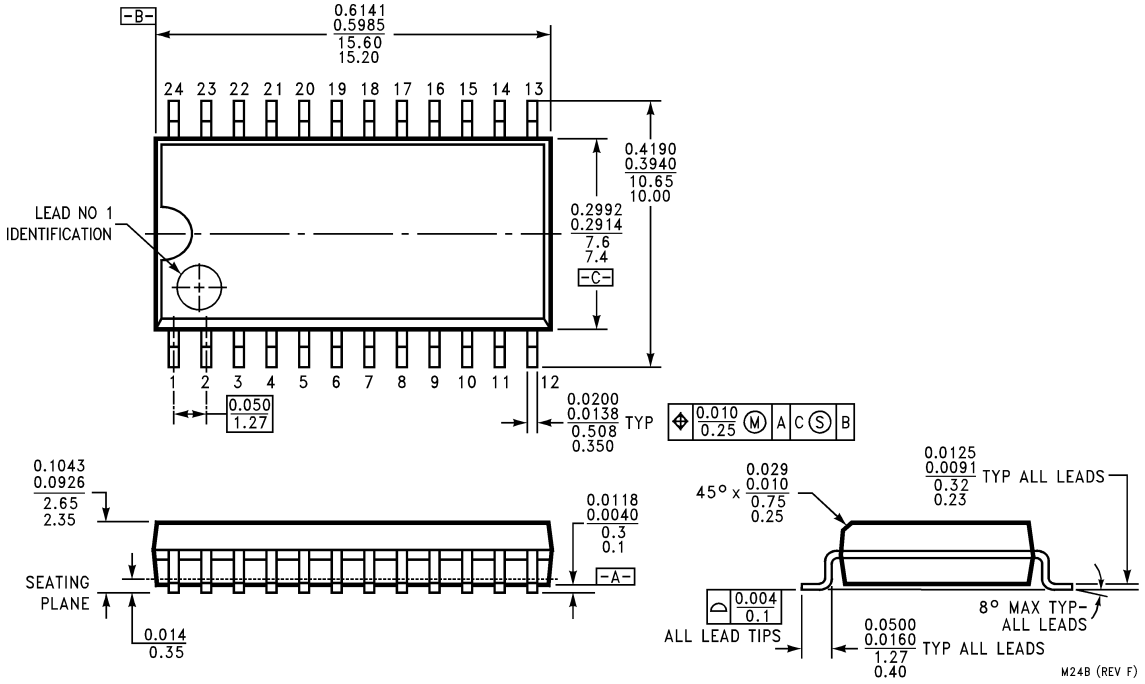
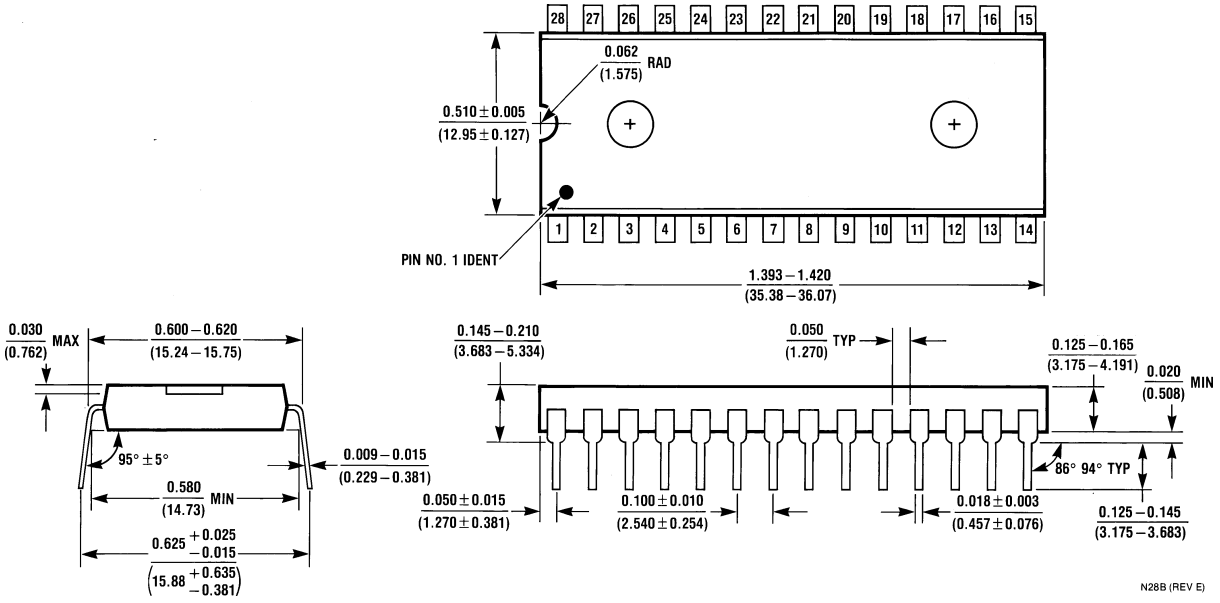


FIGURE 17. Typical Balanced-Line Encoder Input Circuit

Physical Dimensions inches (millimeters) unless otherwise noted



24-Lead Small Outline Package (M)
Order Number LM629M-6 or LM629M-8
NS Package Number M24B



28 Lead Molded Dual-In-Line Package (N)
Order Number LM628N-6, LM628N-8, LM629N-6 or LM629N-8
NS Package Number N28B

Notes

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor
Americas Customer
Support Center
Email: new.feedback@nsc.com
Tel: 1-800-272-9959

www.national.com

National Semiconductor
Europe Customer Support Center
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor
Asia Pacific Customer
Support Center
Fax: 65-6250 4466
Email: ap.support@nsc.com
Tel: 65-6254 4466

National Semiconductor
Japan Customer Support Center
Fax: 81-3-5639-7507
Email: nsj.crc@jksmtp.nsc.com
Tel: 81-3-5639-7560

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.

National Semiconductor

April 2005

LMD18200
3A, 55V H-Bridge

General Description

The LMD18200 is a 3A H-Bridge designed for motion control applications. The device is built using a multi-technology process which combines bipolar and CMOS control circuitry with DMOS power devices on the same monolithic structure. Ideal for driving DC and stepper motors; the LMD18200 accommodates peak output currents up to 6A. An innovative circuit which facilitates low-loss sensing of the output current has been implemented.

Features

- Delivers up to 3A continuous output
- Operates at supply voltages up to 55V
- Low R_{DS}(ON) typically 0.3Ω per switch
- TTL and CMOS compatible inputs
- No "shoot-through" current
- Thermal warning flag output at 145˚C
- Thermal shutdown (outputs off) at 170˚C
- Internal clamp diodes
- Shorted load protection
- Internal charge pump with external bootstrap capability

Applications

- DC and stepper motor drives
- Position and velocity servomechanisms
- Factory automation robots
- Numerically controlled machinery
- Computer printers and plotters

Functional Diagram

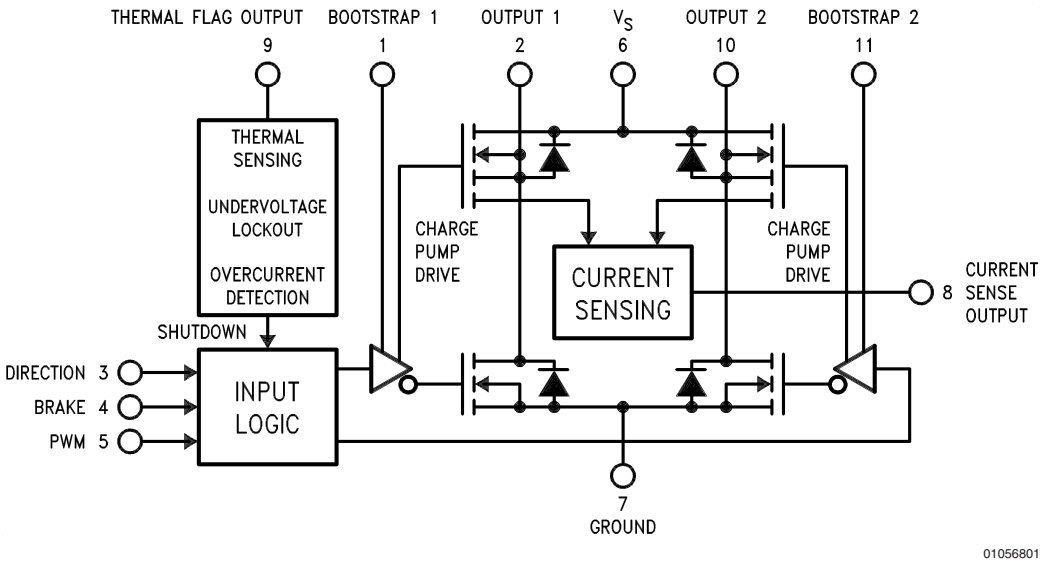
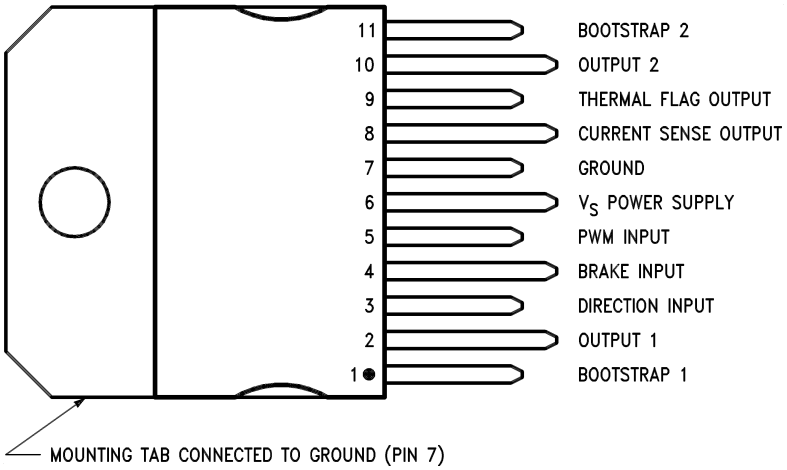


FIGURE 1. Functional Block Diagram of LMD18200

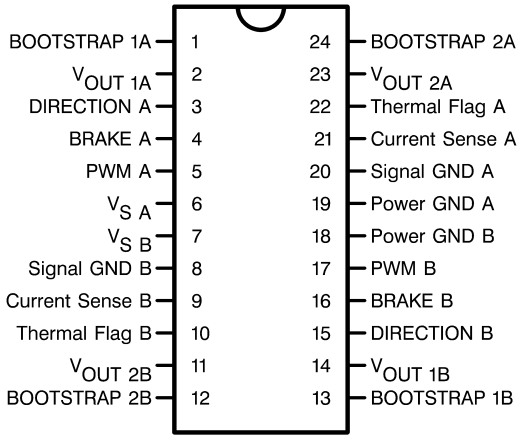
© 2005 National Semiconductor Corporation

Connection Diagrams and Ordering Information



01056802

11-Lead TO-220 Package
Top View
Order Number LMD18200T
See NS Package TA11B



01056825

24-Lead Dual-in-Line Package
Top View
Order Number LMD18200-2D-QV
5962-9232501VXA
LMD18200-2D/883
5962-9232501MXA
See NS Package DA24B

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Total Supply Voltage (V_S, Pin 6)	60V
Voltage at Pins 3, 4, 5, 8 and 9	12V
Voltage at Bootstrap Pins (Pins 1 and 11)	V_{OUT} +16V
Peak Output Current (200 ms)	6A
Continuous Output Current (Note 2)	3A
Power Dissipation (Note 3)	25W
Power Dissipation (T_A = 25˚C, Free Air)	3W
Junction Temperature, $T_{J(max)}$	150˚C
ESD Susceptibility (Note 4)	1500V
Storage Temperature, T_{STG}	−40˚C to +150˚C
Lead Temperature (Soldering, 10 sec.)	300˚C

Operating Ratings (Note 1)

Junction Temperature, T_J	−40˚C to +125˚C
V_S Supply Voltage	+12V to +55V

Electrical Characteristics (Note 5)

The following specifications apply for V_S = 42V, unless otherwise specified. **Boldface** limits apply over the entire operating temperature range, −40˚C ≤ T_J ≤ +125˚C, all other limits are for T_A = T_J = 25˚C.

Symbol	Parameter	Conditions	Typ	Limit	Units
R_{DS}(ON)	Switch ON Resistance	Output Current = 3A (Note 6)	0.33	0.4/**0.6**	Ω (max)
R_{DS}(ON)	Switch ON Resistance	Output Current = 6A (Note 6)	0.33	0.4/**0.6**	Ω (max)
V_{CLAMP}	Clamp Diode Forward Drop	Clamp Current = 3A (Note 6)	1.2	1.5	V (max)
V_{IL}	Logic Low Input Voltage	Pins 3, 4, 5		**−0.1**	V (min)
				0.8	V (max)
I_{IL}	Logic Low Input Current	V_{IN} = −0.1V, Pins = 3, 4, 5		**−10**	µA (max)
V_{IH}	Logic High Input Voltage	Pins 3, 4, 5		**2**	V (min)
				12	V (max)
I_{IH}	Logic High Input Current	V_{IN} = 12V, Pins = 3, 4, 5		**10**	µA (max)
	Current Sense Output	I_{OUT} = 1A (Note 8)	377	325/**300**	µA (min)
				425/**450**	µA (max)
	Current Sense Linearity	1A ≤ I_{OUT} ≤ 3A (Note 7)	±6	±9	%
	Undervoltage Lockout	Outputs turn OFF		9	V (min)
				11	V (max)
T_{JW}	Warning Flag Temperature	Pin 9 ≤ 0.8V, I_L = 2 mA	145		˚C
V_F(ON)	Flag Output Saturation Voltage	T_J = T_{JW}, I_L = 2 mA	0.15		V
I_F(OFF)	Flag Output Leakage	V_F = 12V	0.2	10	µA (max)
T_{JSD}	Shutdown Temperature	Outputs Turn OFF	170		˚C
I_S	Quiescent Supply Current	All Logic Inputs Low	13	25	mA (max)
t_{Don}	Output Turn-On Delay Time	Sourcing Outputs, I_{OUT} = 3A	300		ns
		Sinking Outputs, I_{OUT} = 3A	300		ns
t_{on}	Output Turn-On Switching Time	Bootstrap Capacitor = 10 nF			
		Sourcing Outputs, I_{OUT} = 3A	100		ns
		Sinking Outputs, I_{OUT} = 3A	80		ns
t_{Doff}	Output Turn-Off Delay Times	Sourcing Outputs, I_{OUT} = 3A	200		ns
		Sinking Outputs, I_{OUT} = 3A	200		ns
t_{off}	Output Turn-Off Switching Times	Bootstrap Capacitor = 10 nF			
		Sourcing Outputs, I_{OUT} = 3A	75		ns
		Sinking Outputs, I_{OUT} = 3A	70		ns
t_{pw}	Minimum Input Pulse Width	Pins 3, 4 and 5	1		µs
t_{cpr}	Charge Pump Rise Time	No Bootstrap Capacitor	20		µs

Electrical Characteristics Notes

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its rated operating conditions.

Note 2: See Application Information for details regarding current limiting.

Note 3: The maximum power dissipation must be derated at elevated temperatures and is a function of $T_{J(max)}$, θ_{JA}, and T_A. The maximum allowable power dissipation at any temperature is $P_{D(max)} = (T_{J(max)} - T_A)/\theta_{JA}$, or the number given in the Absolute Ratings, whichever is lower. The typical thermal resistance from junction to case (θ_{JC}) is 1.0˚C/W and from junction to ambient (θ_{JA}) is 30˚C/W. For guaranteed operation $T_{J(max)}$ = 125˚C.

Note 4: Human-body model, 100 pF discharged through a 1.5 kΩ resistor. Except Bootstrap pins (pins 1 and 11) which are protected to 1000V of ESD.

Note 5: All limits are 100% production tested at 25˚C. Temperature extreme limits are guaranteed via correlation using accepted SQC (Statistical Quality Control) methods. All limits are used to calculate AOQL, (Average Outgoing Quality Level).

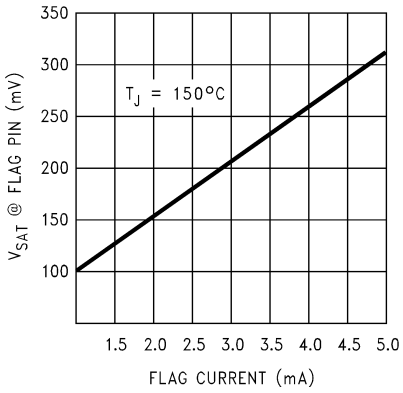
Note 6: Output currents are pulsed (t_W < 2 ms, Duty Cycle < 5%).

Note 7: Regulation is calculated relative to the current sense output value with a 1A load.

Note 8: Selections for tighter tolerance are available. Contact factory.

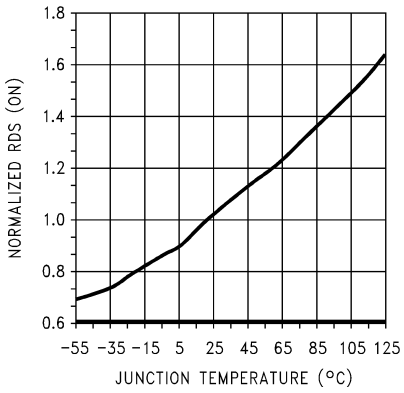
Typical Performance Characteristics

V_{SAT} vs Flag Current



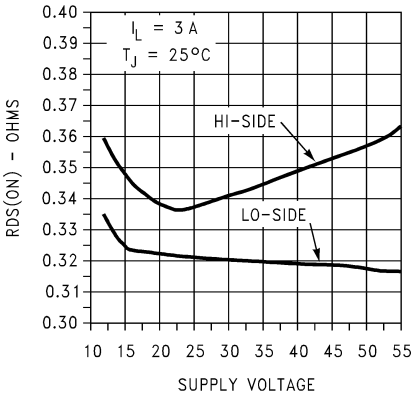
01056816

R_{DS}(ON) vs Temperature



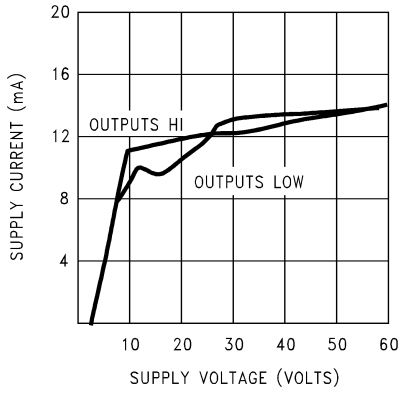
01056817

R_{DS}(ON) vs Supply Voltage



01056818

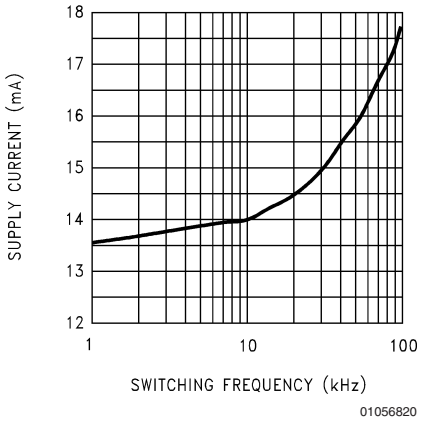
Supply Current vs Supply Voltage



01056819

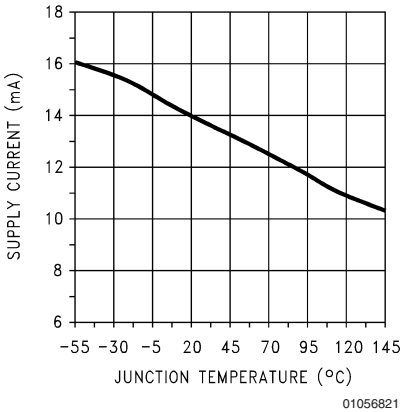
Typical Performance Characteristics (Continued)

Supply Current vs Frequency (V_S = 42V)



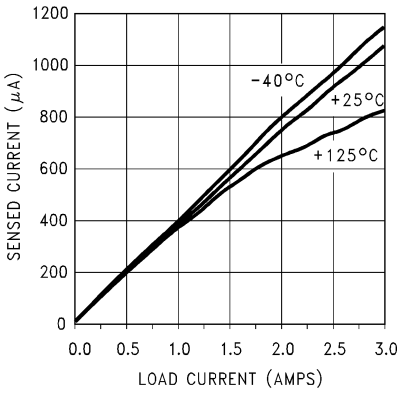
01056820

Supply Current vs Temperature (V_S = 42V)



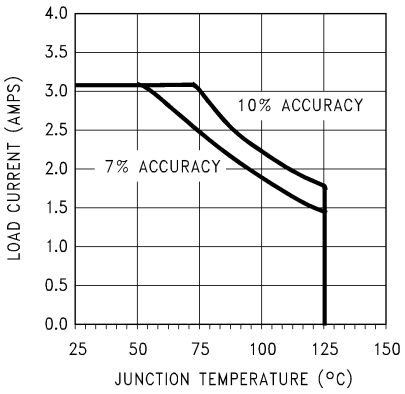
01056821

Current Sense Output vs Load Current



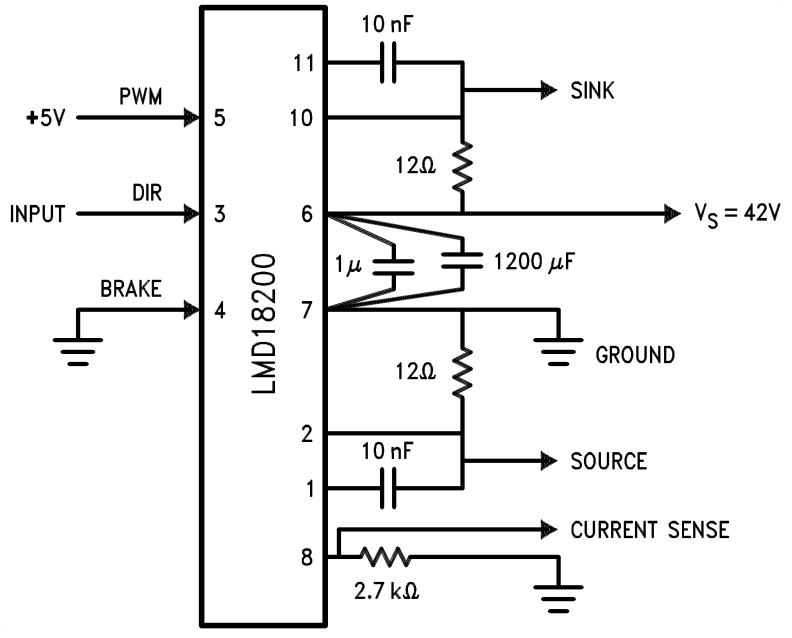
01056822

Current Sense Operating Region



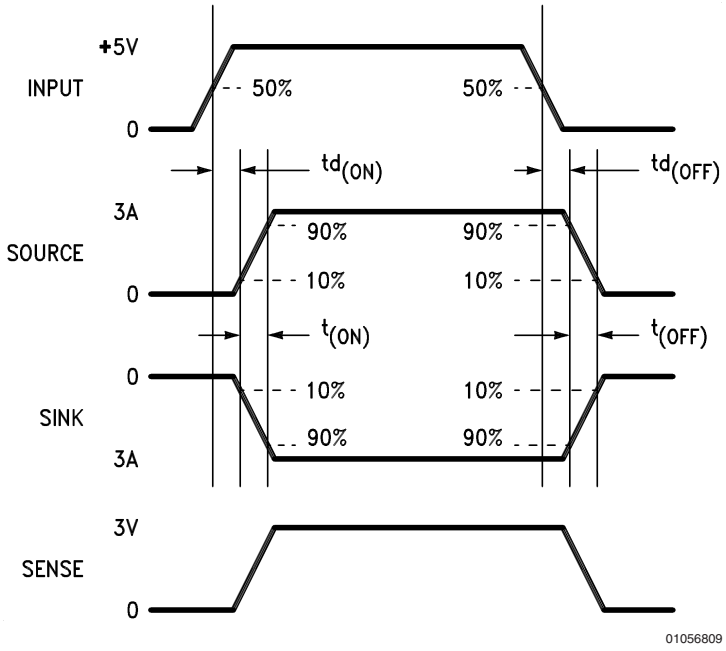
01056823

Test Circuit



01056808

Switching Time Definitions



Pinout Description

(See Connection Diagram)

Pin 1, BOOTSTRAP 1 Input: Bootstrap capacitor pin for half H-bridge number 1. The recommended capacitor (10 nF) is connected between pins 1 and 2.

Pin 2, OUTPUT 1: Half H-bridge number 1 output.

Pin 3, DIRECTION Input: See *Table 1*. This input controls the direction of current flow between OUTPUT 1 and OUTPUT 2 (pins 2 and 10) and, therefore, the direction of rotation of a motor load.

Pin 4, BRAKE Input: See *Table 1*. This input is used to brake a motor by effectively shorting its terminals. When braking is desired, this input is taken to a logic high level and it is also necessary to apply logic high to PWM input, pin 5. The drivers that short the motor are determined by the logic level at the DIRECTION input (Pin 3): with Pin 3 logic high, both current sourcing output transistors are ON; with Pin 3 logic low, both current sinking output transistors are ON. All output transistors can be turned OFF by applying a logic high to Pin 4 and a logic low to PWM input Pin 5; in this case only a small bias current (approximately –1.5 mA) exists at each output pin.

Pin 5, PWM Input: See *Table 1*. How this input (and DIRECTION input, Pin 3) is used is determined by the format of the PWM Signal.

Pin 6, V_S Power Supply

Pin 7, GROUND Connection: This pin is the ground return, and is internally connected to the mounting tab.

Pin 8, CURRENT SENSE Output: This pin provides the sourcing current sensing output signal, which is typically 377 µA/A.

Pin 9, THERMAL FLAG Output: This pin provides the thermal warning flag output signal. Pin 9 becomes active-low at 145˚C (junction temperature). However the chip will not shut itself down until 170˚C is reached at the junction.

Pin 10, OUTPUT 2: Half H-bridge number 2 output.

Pin 11, BOOTSTRAP 2 Input: Bootstrap capacitor pin for Half H-bridge number 2. The recommended capacitor (10 nF) is connected between pins 10 and 11.

TABLE 1. Logic Truth Table

PWM	Dir	Brake	Active Output Drivers
H	H	L	Source 1, Sink 2
H	L	L	Sink 1, Source 2
L	X	L	Source 1, Source 2
H	H	H	Source 1, Source 2
H	L	H	Sink 1, Sink 2
L	X	H	NONE

Application Information

TYPES OF PWM SIGNALS

The LMD18200 readily interfaces with different forms of PWM signals. Use of the part with two of the more popular forms of PWM is described in the following paragraphs.

Simple, locked anti-phase PWM consists of a single, variable duty-cycle signal in which is encoded both direction and amplitude information (see *Figure 2*). A 50% duty-cycle PWM signal represents zero drive, since the net value of voltage (integrated over one period) delivered to the load is zero. For the LMD18200, the PWM signal drives the direction input (pin 3) and the PWM input (pin 5) is tied to logic high.

Application Information (Continued)

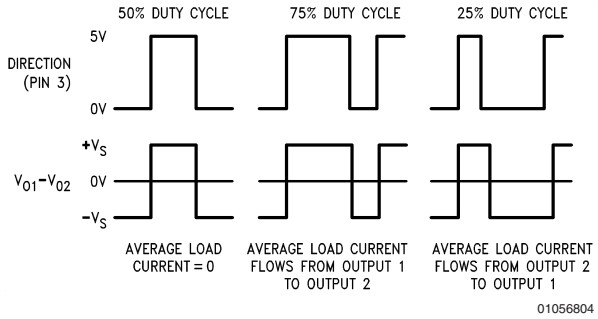


FIGURE 2. Locked Anti-Phase PWM Control

Sign/magnitude PWM consists of separate direction (sign) and amplitude (magnitude) signals (see *Figure 3*). The (absolute) magnitude signal is duty-cycle modulated, and the absence of a pulse signal (a continuous logic low level) represents zero drive. Current delivered to the load is proportional to pulse width. For the LMD18200, the DIRECTION input (pin 3) is driven by the sign signal and the PWM input (pin 5) is driven by the magnitude signal.

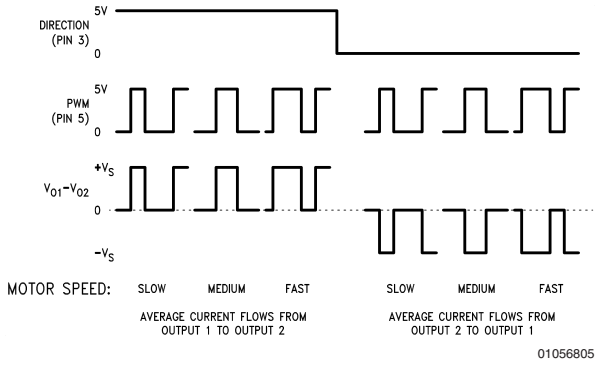


FIGURE 3. Sign/Magnitude PWM Control

SIGNAL TRANSITION REQUIREMENTS

To ensure proper internal logic performance, it is good practice to avoid aligning the falling and rising edges of input signals. A delay of at least 1 µsec should be incorporated between transitions of the Direction, Brake, and/or PWM input signals. A conservative approach is be sure there is at least 500ns delay between the end of the first transition and the beginning of the second transition. See *Figure 4*.

Application Information (Continued)

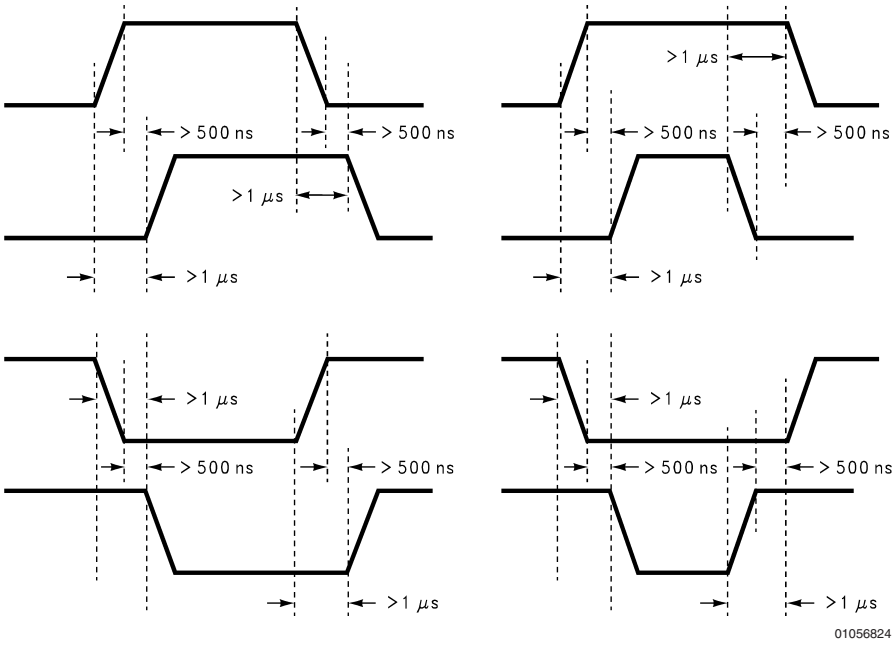


FIGURE 4. Transitions in Brake, Direction, or PWM Must Be Separated By At Least 1 µsec

USING THE CURRENT SENSE OUTPUT

The CURRENT SENSE output (pin 8) has a sensitivity of 377 µA per ampere of output current. For optimal accuracy and linearity of this signal, the value of voltage generating resistor between pin 8 and ground should be chosen to limit the maximum voltage developed at pin 8 to 5V, or less. The maximum voltage compliance is 12V.

It should be noted that the recirculating currents (free wheeling currents) are ignored by the current sense circuitry. Therefore, only the currents in the upper sourcing outputs are sensed.

USING THE THERMAL WARNING FLAG

The THERMAL FLAG output (pin 9) is an open collector transistor. This permits a wired OR connection of thermal warning flag outputs from multiple LMD18200's, and allows the user to set the logic high level of the output signal swing to match system requirements. This output typically drives the interrupt input of a system controller. The interrupt service routine would then be designed to take appropriate steps, such as reducing load currents or initiating an orderly system shutdown. The maximum voltage compliance on the flag pin is 12V.

SUPPLY BYPASSING

During switching transitions the levels of fast current changes experienced may cause troublesome voltage transients across system stray inductance.

It is normally necessary to bypass the supply rail with a high quality capacitor(s) connected as close as possible to the V_S Power Supply (Pin 6) and GROUND (Pin 7). A 1 µF high-frequency ceramic capacitor is recommended. Care should be taken to limit the transients on the supply pin below the Absolute Maximum Rating of the device. When operating the chip at supply voltages above 40V a voltage suppressor (transorb) such as P6KE62A is recommended from supply to ground. Typically the ceramic capacitor can be eliminated in the presence of the voltage suppressor. Note that when driving high load currents a greater amount of supply bypass capacitance (in general at least 100 µF per Amp of load current) is required to absorb the recirculating currents of the inductive loads.

CURRENT LIMITING

Current limiting protection circuitry has been incorporated into the design of the LMD18200. With any power device it is important to consider the effects of the substantial surge currents through the device that may occur as a result of shorted loads. The protection circuitry monitors this increase in current (the threshold is set to approximately 10 Amps) and shuts off the power device as quickly as possible in the event of an overload condition. In a typical motor driving application the most common overload faults are caused by shorted motor windings and locked rotors. Under these conditions the inductance of the motor (as well as any series inductance in the V_{CC} supply line) serves to reduce the magnitude of a current surge to a safe level for the LMD18200. Once the device is shut down, the control circuitry will periodically try to turn the power device back on. This feature allows the immediate return to normal operation in the event that the fault condition has been removed. While the fault remains however, the device will cycle in and out of thermal shutdown. This can create voltage transients on the V_{CC} supply line and therefore proper supply bypassing techniques are required.

The most severe condition for any power device is a direct, hard-wired ("screwdriver") long term short from an output to ground. This condition can generate a surge of current through the power device on the order of 15 Amps and require the die and package to dissipate up to 500 Watts of power for the short time required for the protection circuitry

Application Information (Continued)

to shut off the power device. This energy can be destructive, particularly at higher operating voltages (>30V) so some precautions are in order. Proper heat sink design is essential and it is normally necessary to heat sink the V_{CC} supply pin (pin 6) with 1 square inch of copper on the PCB.

INTERNAL CHARGE PUMP AND USE OF BOOTSTRAP CAPACITORS

To turn on the high-side (sourcing) DMOS power devices, the gate of each device must be driven approximately 8V more positive than the supply voltage. To achieve this an internal charge pump is used to provide the gate drive voltage. As shown in *Figure 5*, an internal capacitor is alternately switched to ground and charged to about 14V, then switched to V supply thereby providing a gate drive voltage greater than V supply. This switching action is controlled by a continuously running internal 300 kHz oscillator. The rise time of this drive voltage is typically 20 µs which is suitable for operating frequencies up to 1 kHz.

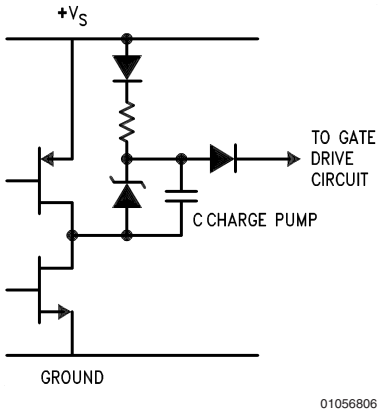


FIGURE 5. Internal Charge Pump Circuitry

For higher switching frequencies, the LMD18200 provides for the use of external bootstrap capacitors. The bootstrap principle is in essence a second charge pump whereby a large value capacitor is used which has enough energy to quickly charge the parasitic gate input capacitance of the power device resulting in much faster rise times. The switching action is accomplished by the power switches themselves *Figure 6*. External 10 nF capacitors, connected from the outputs to the bootstrap pins of each high-side switch provide typically less than 100 ns rise times allowing switching frequencies up to 500 kHz.

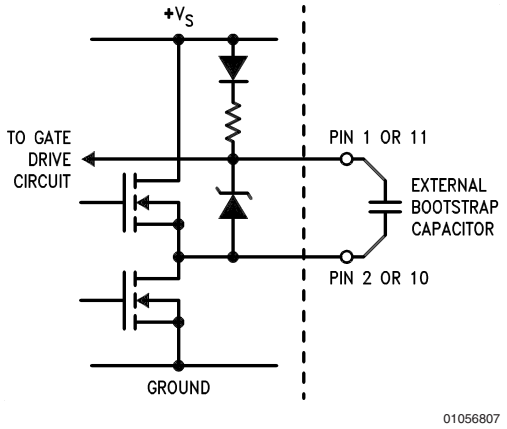


FIGURE 6. Bootstrap Circuitry

INTERNAL PROTECTION DIODES

A major consideration when switching current through inductive loads is protection of the switching power devices from the large voltage transients that occur. Each of the four switches in the LMD18200 have a built-in protection diode to clamp transient voltages exceeding the positive supply or ground to a safe diode voltage drop across the switch.

The reverse recovery characteristics of these diodes, once the transient has subsided, is important. These diodes must come out of conduction quickly and the power switches must be able to conduct the additional reverse recovery current of the diodes. The reverse recovery time of the diodes protecting the sourcing power devices is typically only 70 ns with a reverse recovery current of 1A when tested with a full 6A of forward current through the diode. For the sinking devices the recovery time is typically 100 ns with 4A of reverse current under the same conditions.

Typical Applications

FIXED OFF-TIME CONTROL

This circuit controls the current through the motor by applying an average voltage equal to zero to the motor terminals for a fixed period of time, whenever the current through the motor exceeds the commanded current. This action causes the motor current to vary slightly about an externally controlled average level. The duration of the Off-period is adjusted by the resistor and capacitor combination of the LM555. In this circuit the Sign/Magnitude mode of operation is implemented (see Types of PWM Signals).

Typical Applications (Continued)

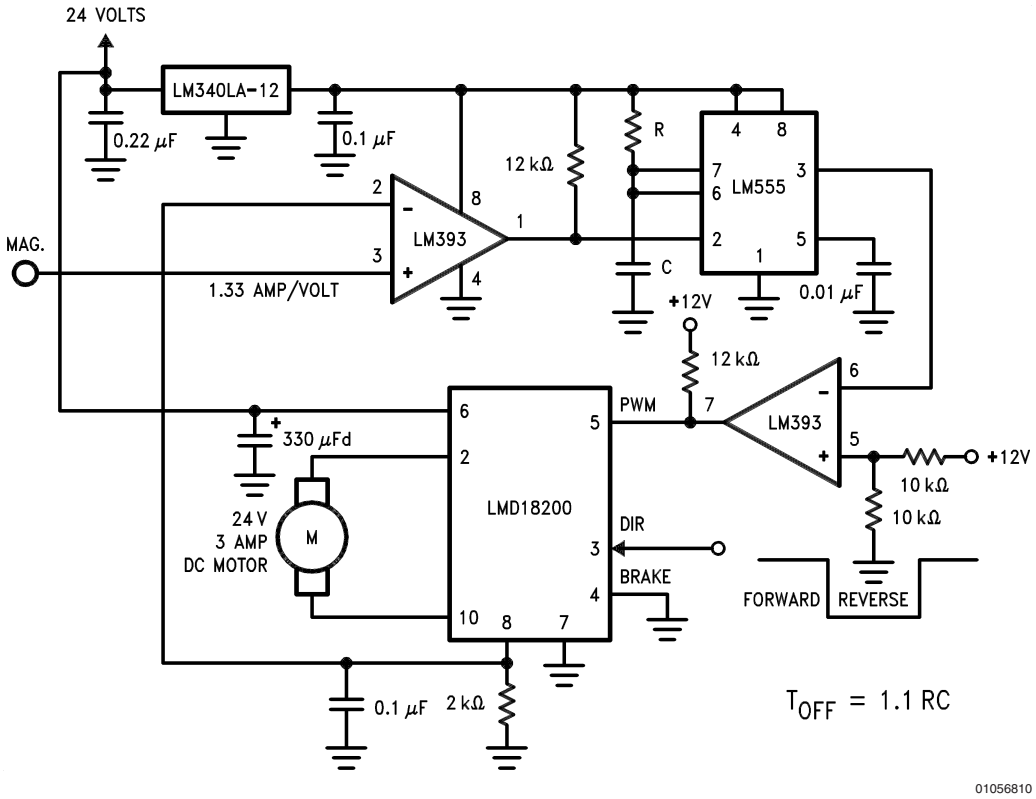


FIGURE 7. Fixed Off-Time Control

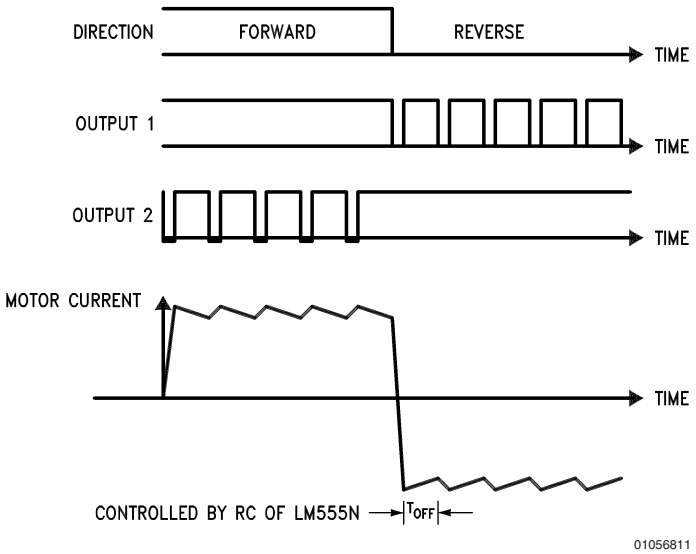


FIGURE 8. Switching Waveforms

TORQUE REGULATION

Locked Anti-Phase Control of a brushed DC motor. Current sense output of the LMD18200 provides load sensing. The LM3524D is a general purpose PWM controller. The relationship of peak motor current to adjustment voltage is shown in *Figure 10*.

Typical Applications (Continued)

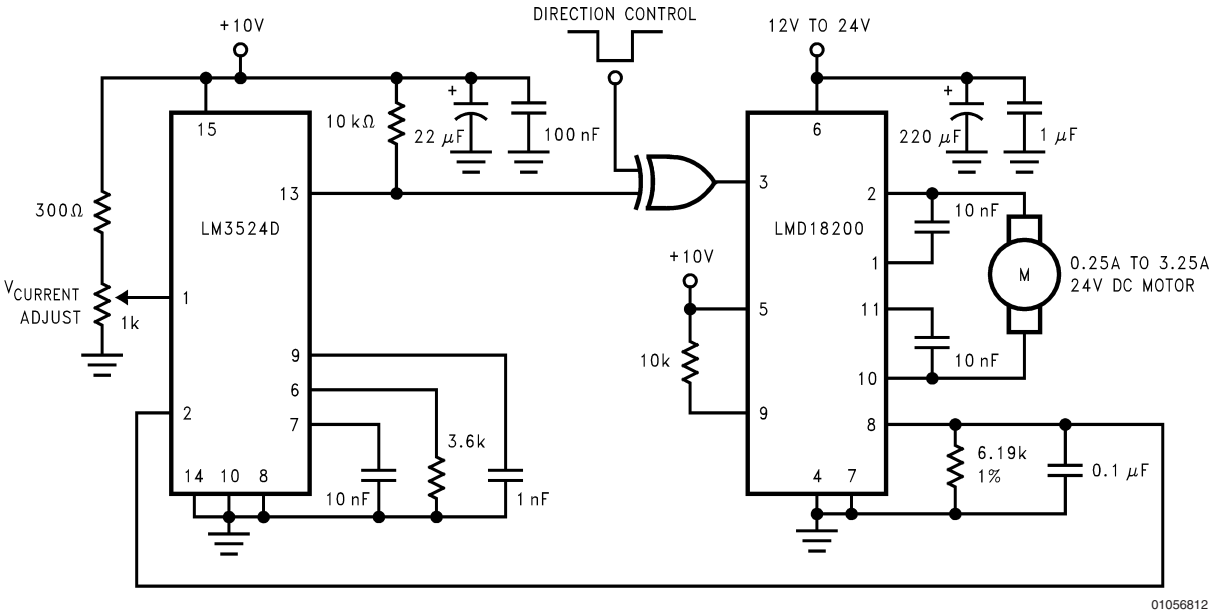


FIGURE 9. Locked Anti-Phase Control Regulates Torque

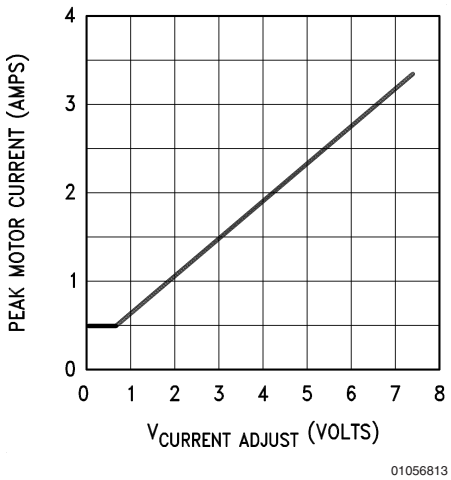
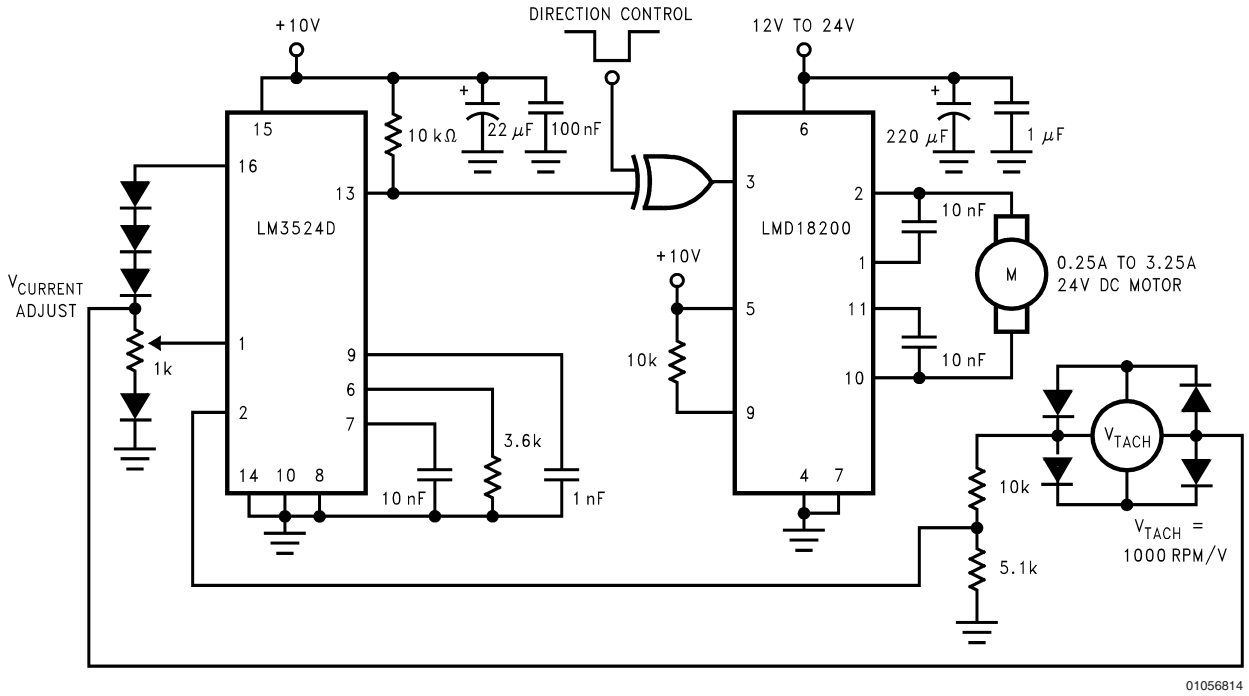


FIGURE 10. Peak Motor Current vs Adjustment Voltage

VELOCITY REGULATION

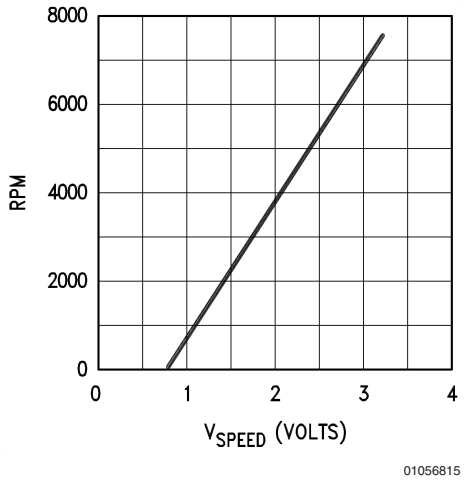
Utilizes tachometer output from the motor to sense motor speed for a locked anti-phase control loop. The relationship of motor speed to the speed adjustment control voltage is shown in *Figure 12*.

Typical Applications (Continued)



01056814

FIGURE 11. Regulate Velocity with Tachometer Feedback

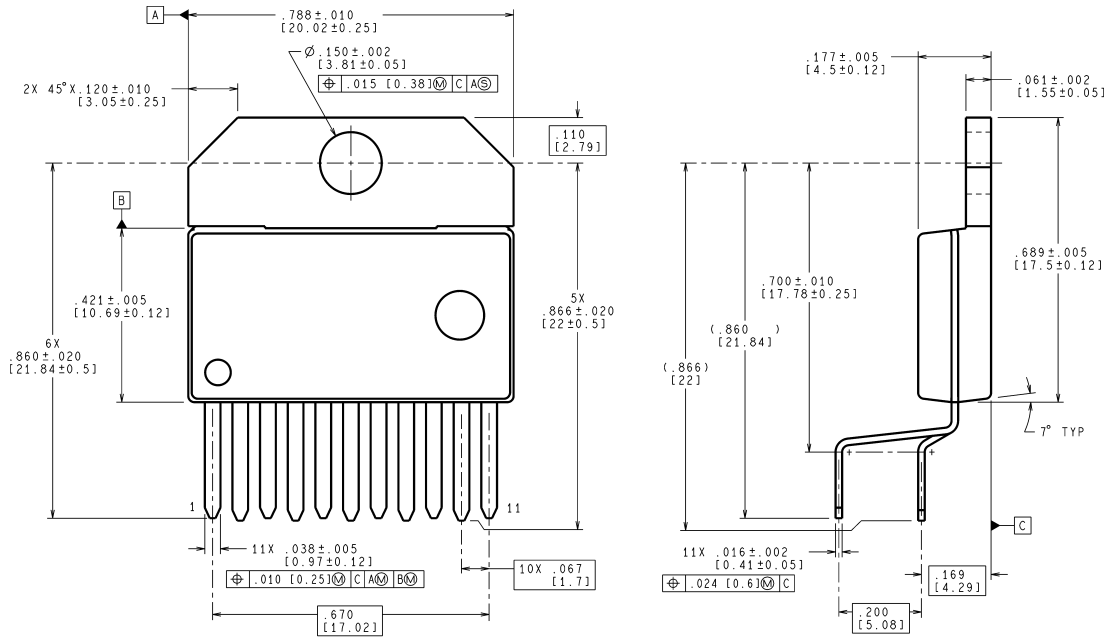


01056815

FIGURE 12. Motor Speed vs Control Voltage

Physical Dimensions inches (millimeters)

unless otherwise noted

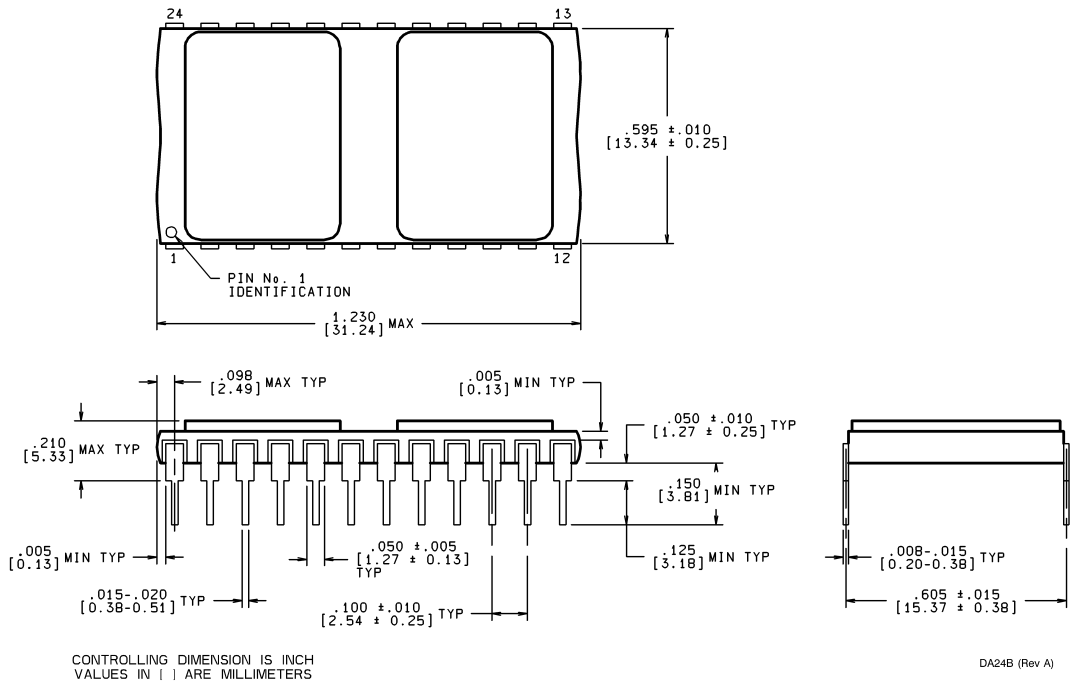


CONTROLLING DIMENSION IS INCH
VALUES IN [ ] ARE MILLIMETERS

TA11B (Rev B)

11-Lead TO-220 Power Package (T)
Order Number LMD18200T
NS Package Number TA11B

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



24-Lead Dual-in-Line Package
Order Number LMD18200-2D-QV
5962-9232501VXA
LMD18200-2D/883
5962-9232501MXA
NS Package Number DA24B

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.

For the most current product information visit us at www.national.com.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

BANNED SUBSTANCE COMPLIANCE

National Semiconductor manufactures products and uses packing materials that meet the provisions of the Customer Products Stewardship Specification (CSP-9-111C2) and the Banned Substances and Materials of Interest Specification (CSP-9-111S2) and contain no "Banned Substances" as defined in CSP-9-111S2.

National Semiconductor
Americas Customer
Support Center
Email: new.feedback@nsc.com
Tel: 1-800-272-9959

www.national.com

National Semiconductor
Europe Customer Support Center
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor
Asia Pacific Customer
Support Center
Email: ap.support@nsc.com

National Semiconductor
Japan Customer Support Center
Fax: 81-3-5639-7507
Email: jpn.feedback@nsc.com
Tel: 81-3-5639-7560

UMANIS

UMANIS, UN MANIPULADOR INTELIGENTE

Martín Alcalá Rubí, Nicolás de Mula, Andrés Pietrafesa

I. Introducción

UMANIS es un proyecto de fin de carrera del Instituto de Ingeniería Eléctrica de la Universidad de la República. Se desarrolló durante los años 2004 y 2005 por los estudiantes Martín Alcalá Rubí, Nicolás de Mula y Andrés Pietrafesa en el marco de una iniciativa de desarrollo e investigación de robótica e inteligencia artificial impulsada por el Ingeniero Rafael Canetti (tutor de este proyecto).

El proyecto consistío básicamente en el diseño y construcción de un manipulador robótico autónomo a ser integrado y montado en una robota ya existente (Sulla), realizada en el año 2003 por otros estudiantes de la misma institución[1].

Como punto de partida se plantearon algunas restricciones para el brazo, sobre todo en cuanto a su alcance (40 cm.), su fuerza (capaz de levantar 500g) y su presición (cubo de 3mm de lado), además de las restricciones impuestas por la robota (como ser consumo de energía y dimensiones del manipulador). También es requisito de este proyecto la correcta comunicación entre el manipulador (Umanis) y la robota (Sulla).

[1] Estos estudiantes son: Rafael Grompone, Agustín Quaglioti y Pablo Rolando.

II. Sobre Sulla

Sulla es una robota movil con las siguientes caracteristicas: Sus dimensiones son, aproximadamente, 45 cm. de largo, 32cm de ancho y 40 cm. de alto. En su cuerpo está incluido el sistema móvil, las baterías para alimentación autónoma y su sistema de carga, sensores, 1 microcontrolador, 1 PC y el espacio para agregar otros dispositivos como ser cámaras o en nuestro caso específico, un manipulador.

Sulla tiene 3 ruedas, una gira libremente y las otras dos se controlan en forma independiente. Es capaz de girar o moverse en trayectorias curvas moviendo con distintas velocidades las ruedas de cada lado.

La información del entorno la obtiene de 17 sensores infrarrojos de distancia, 2 sensores ultrasónicas de distancia, un sensor de luz, sensores de temperatura y algunos sensors de contacto.

El hardware computacional se realiza en torno de una red Ethernet estándar. Esto permite agregar nuevos módulos de procesamiento que tengan esta interfaz. En la primera etapa de Sulla, la red es de dos nodos: un microcontrolador (Rabbit 3000) para manejo de eventos en tiempo real y un PC estándar para el procesamiento de alto nivel.

La razón para hacer esta división es poder poner toda la programación de tiempo real fuera del PC, y de esta forma poder usar en el PC un Sistema Operativo como GNU/Linux, que no es de tiempo real, pero facilita la reutilización de código y disponibilidad de herramientas a la vez que reduce los conocimientos necesarios para programar.

El esquema de organización de la programación global, es en módulos software que funcionan en forma independiente, donde toda interacción entre ellos ocurre por mensajes UDP. Este modelo de comunicación, junto al protocolo establecido para la comunicación, toma el nombre de Sinapsis.

Sinapsis permite la comunicación en forma homogénea entre módulos ejecutados en distintas computadoras o en la misma, independiente del Sistema Operativo o del

lenguaje de programación usado. Cualquier lenguaje de programación o ambiente de trabajo que sea capaz de enviar y recibir mensajes UDP podrá ser usado en Sinapsis.

Sinapsis está orientado a formar una estructura de módulos que ínter operan para coordinar acciones según información de los sensores, información del pasado y objetivos variados con distintas prioridades. El protocolo permite identificar y referirse a módulos específicos, establecer comunicación para pedir información que otro módulo tiene disponible o dar órdenes.

III. DISEÑO GENERAL DE *UMANIS*

Se podría decir que *Umanis* tiene tres partes fundamentales y bien diferenciadas:

A. Diseño y construcción mecánica

El manipulador contiene un total de 5 grados de libertad, estos son: cintura, hombro, codo, muñeca y mano o pinza. Se utilizaron los nombres correspondientes a las partes del cuerpo humano para una mayor comprensión.

Todos estos grados de libertad o articulaciones, a excepción de la pinza, son de movimiento giratorio. En caso de la pinza, esta puede ser cerrada o abierta realizando un control de torque. Todos los actuadores (motores) de las articulaciones son motores eléctricos de corriente continua.

B. Hardware eléctrico

Umanis también está compuesto por dos placas de Electrónica, una para manejar las señales de potencia que serán inyectadas en los motores, y otra que posee la "inteligencia" manipulador, es decir microprocesadores dedicados, además de electrónica de manejo de alarmas. A esta placa la llamaremos placa de Electrónica de señal.

Los módulos que sean agregados a la robota, deberán ser capaces de comunicarse vía paquetes UDP, utilizando un protocolo llamado Sinapsis, por lo que *Umanis* posee un microcontrolador Rabbit 3000 que dentro de otras cosas, es capaz de comunicarse a través de Ethernet en una red de datos TCP/IP.

Como veremos en secciones posteriores, el control de los distintos motores del manipulador es efectuado por 3 microcontroladores y un sistema electrónico de control de torque para la Pinza. Estos 3 microcontroladores son:

- Un Rabbit 3000 (RCM 3200 RabbitCore, manufacturado por Z·World)
- Dos LM629 [2] (manufacturado por National Semiconductor).

Siendo el microprocesador Rabbit 3000 el principal actor, entre ellos son capaces de entre otras cosas: controlar los motores, generar trayectorias para el manipulador conjunto, mantener la sincronicidad entre el comando de motores y ejecutar directivas que arriban a la interfaz Ethernet.

C. Desarrollo de Software

El software de Umanis está pensado para ejecutar y encolar comandos provenientes de una entidad externa a través del protocolo Sinapsis. En este caso, dicha entidad es Sulla. Esta funcionalidad permitiría que Umanis fuera comandado utilizando como medio de comunicación una red de datos estándar, en particular permitiría que el manipulador fuera comandado a través de Internet.

La mayor parte de los módulos de software se desarrollan en el Rabbit antes mencionado y algunas partes se desarrollaron en la PC incluida en Sulla.

IV. DISEÑO Y CONSTRUCCIÓN MECÁNICA

Al diseñar y construir el manipulador nunca se perdieron de vista los objetivos y restricciones, sobretodo en cuanto a la fuerza y precisión necesarias para el brazo, se busco utilizar materiales livianos y fuertes (como por ejemplo el aluminio), los encoders se buscaron para cumplir con la precisión requerida y los motores para soportar la carga máxima exigida y que además su consumo de corriente fuera razonable y sus dimensiones reducidas.

A. Nomenclatura

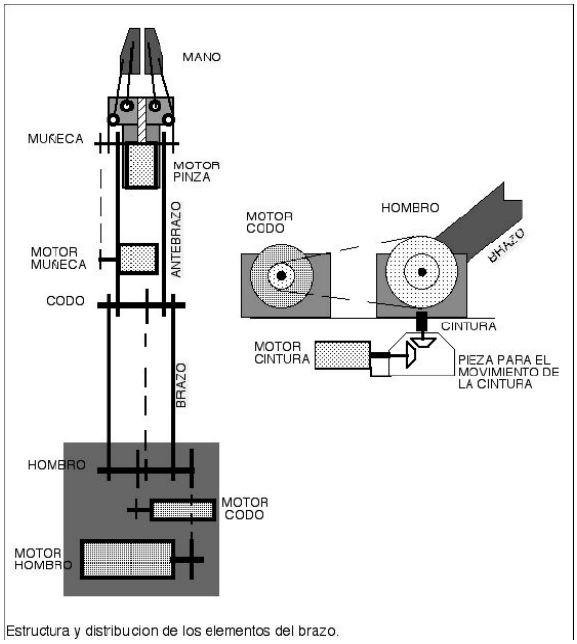
Manipulador es todo el conjunto de partes mecánicas que conforman nuestro proyecto. Para hacer referencia a las distintas articulaciones se utilizan los nombres correspondientes de las partes análogas en el cuerpo humano, esto es:

1) La *cintura* corresponde a un eje vertical, solidario a la robota, sobre el cual gira el manipulador completo.
2) El *brazo* y el antebrazo son los segmentos que van desde el hombro al codo y desde el codo a la muñeca respectivamente.
3) El *hombro* es el eje horizontal, solidario a la base giratoria, que sube y baja el brazo.
4) El *codo* es el eje horizontal, solidario al brazo, que mueve el antebrazo respecto al brazo.
5) La *muñeca,* eje horizontal solidario al antebrazo que mueve la mano respecto al antebrazo
6) Y la *mano* corresponde a la pinza o elemento actuador.

B. Estructura del manipulador

[2] El LM629 es un microprocesador dedicado al control de movimiento de motores de corriente continua.

En la imagen se puede ver un dibujo de la distribución de los elementos del brazo.



Estructura y distribucion de los elementos del brazo.

La base es una planchuela rectangular de aluminio que sostiene el motor del hombro y del codo además del resto de la estructura del manipulador.
Tanto el brazo como el antebrazo están compuestos por dos planchuelas de aluminio ubicadas en forma paralela, el largo de cada segmento es de 18 cm. entre eje y eje.

El eje que corresponde a la cintura esta fijo a la base giratoria (y perpendicular a la misma) para luego introducirse en la pieza modular que es la que le da movimiento. El eje del hombro esta montado sobre rulemanes solidarios a la base (y sostenidos a la misma por planchuelas en forma de L) y esta fijo a las planchuelas que conforman el brazo. El eje del codo esta sobre rulemanes solidarios al brazo y esta fijo a las planchuelas del antebrazo.
El eje de la muñeca gira sobre rulemanes en las planchuelas del antebrazo y esta fijo a la pinza y el motor de la misma.

C. Articulaciones

Para cada una de las articulaciones se busco un diseño que aportara de alguna manera el problema de control, sobre todo en lo referente al problema de la fuerza, tratando de ubicar los motores (claramente lo elementos mas pesados en el manipulador) en una posición que ayudara a disminuir el torque exigido a alguno de los actuadores.
Las transmisiones mecánicas se implementaron en su mayoría con correas de goma dentadas del tipo XL (se eligió este tipo de correas por su poco peso y porque son muy confiables en cuanto a la resistencia y el juego con los engranajes), los engranajes en algunos casos son de aluminio y en otro son de nylon.

1) Cintura

Para la cintura se utiliza una pieza muy adecuada que implementa, mediante dos engranajes cónicos, la posibilidad de tener dos ejes solidarios perpendiculares entre si, de esta manera sobre uno de los ejes se sostiene la base con el manipulador completo y en el otro esta fijo el motor, de esta manera se logra que la palanca que genera el manipulador no sea soportada por el eje del motor.

2) Hombro

El motor del hombro es el motor mas pesado de los cinco utilizados, se aprovecho este hecho y se ubico el motor en la parte de atrás de la base para que hiciera las veces de contrapeso para contrarrestar el peso del manipulador y acercar el centro de masa al eje de la cintura
La transmisión se implemento con correas dentadas de goma del tipo XL y engranajes de aluminio.

3) Codo

En el codo se busco sacar el motor de la estructura del brazo para que no sumara mucho torque para el motor del hombro, se diseño una transmisión mediante la cual un sistema complejo que consta de dos etapas ubica al motor en la base, lo cual ayudo mucho a disminuir el torque exigido al motor del hombro.
La primera etapa va desde el motor del codo (ubicado en la base) hasta el eje del hombro, ahí hay un punto intermedio con dos engranajes montados sobre rulemanes en el eje mencionado, desde ahí hay una segunda etapa hasta el eje del codo; cada etapa presenta una reducción 1:3 dada por la cantidad de dientes de los engranajes.
Las dos etapas de la transmisión se implementan con correas dentadas de goma del tipo XL y engranajes de nylon.

4) Muñeca

Se ubico el motor alejado de la muñeca, sobre el antebrazo, en un punto intermedio entre el codo y la muñeca, la transmisión se realizo con engranajes de aluminio y correas de goma dentadas.
De esta manera, al atrasar el motor, este aporta considerablemente menos torque a los motores tanto del codo como del hombro

5) Pinza

Para la pinza se opto por un mecanismo de cierre un tanto particular que produce un movimiento de la pinza con sus ¨dedos¨ siempre en forma paralela y además permite una ubicación del motor de esta articulación en una posición adecuada para el problema del torque y también que mantiene la simetría del brazo.

D. *Elementos principales*

Para poder accionar cada una de las articulaciones son necesarios los llamados *elementos actuadores* y para poder cerrar el lazo de control y tener la información de la ubicación de cada parte son necesarios lo *dispositivos de realimentación*.

1) *Elementos Actuadores*

Los elementos que hacen efectivo el movimiento son motores de corriente continua de imán permanente con caja de reducción incorporada, para su elección se tuvieron en cuenta las restricciones de fuerza de nuestro proyecto y también que su voltaje nominal y su consumo de corriente fueran los adecuados para las restricciones impuestas por Sulla.
En todos los casos, con excepción al motor de la muñeca, se utilizaron motores de voltaje nominal de 12 V, el motor con mas exigencia en cuanto a torque (y por consecuencia en cuanto a corriente) es el del hombro, el torque en el peor caso calculado para este motor fue de 42 Kg.cm., se busco un motor que para estos valores de torque no superara 1 A de corriente consumida.

Con estos elementos actuadores fue posible cumplir con las restricciones de torque requeridas (500 g) para el peor caso, que corresponde al manipulador totalmente extendido (el largo total del manipulador es de 46 cm.).

Existe una restricción para el motor de la muñeca, este no es capaz de mover mas de 350 g (aprox.), pero esta limitación no es impuesta por el motor sino por parte de la electrónica que lo maneja (mas precisamente un elevador de voltaje).
De todas maneras esta limitación no restringe las posibilidades del brazo, simplemente para levantar mas de 350g. el movimiento del manipulador deberá ser con la muñeca siempre en la misma posición.

2) *Dispositivos de realimentación*

Para sensar la posición de los diferentes ejes se utilizaron *encoders ópticos incrementales (Shaft encoders)* con salidas en cuadratura, estos elementos utilizan un disco ranurado solidario al eje de giro y dos pares emisor-receptor de luz de manera que a medida que el disco gira se generan dos ondas cuadras en cuadratura que hacen posible (contando la cantidad de cambios de estado en el par de señales de salida) saber la posición del eje con una buena presición además de saber el sentido de giro del eje

Para los encoders utilizados se logro una precisión teórica superior a la especificada, aproximadamente se logra una presicion de un cubo de 1mm de lado.

E. *OTROS ELEMENTOS*

Además de los elementos propios de la estructura y los motores y shaft encoders también se instalaron en el manipulador dispositivos de alarma, también llamados interruptores de fin de carrera.
Estos se implementaron mediante interruptores o switchs ubicados en lugares críticos y que son accionados cuando una parte mecánica llega a un límite que pueda comprometer la integridad del propio brazo o de la robota, Estos interruptores llegan al microprocesador en forma de interrupciones externas y son interpretados como alarmas en las que el movimiento debe ser inmediatamente interrumpido.

V. HARDWARE ELÉCTRICO

La finalidad de nuestro hardware eléctrico es el implementar el control de los motores DC de forma de poder lograr que el manipulador se comporte de forma que se ajuste a los objetivos de este proyecto. Para cumplir con este fin se debe lograr que los motores tomen valores específicos de posición, velocidad y/o torque motor en momentos del tiempo también determinados. La forma en que se determina la configuración [3] del manipulador en cada momento no es motivo de análisis en este apartado, pero si lo es el funcionamiento del sistema que a partir de una configuración actual y real recibe la orden de alcanzar una configuración [3] dada en un tiempo posterior T.

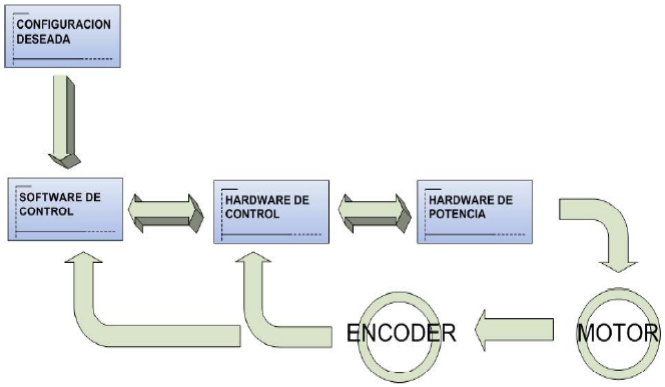


Fig. III.1 – Diagrama de bloques.

Se fabricaron dos circuitos impresos, el correspondiente a la etapa de potencia y el correspondiente a la etapa de control, además de la utilización del módulo de Z-World RCM3200 que contiene el microprocesador Rabbit 3000 . La complejidad de la placa de control hizo que buscáramos la alternativa de mandarla construir en el exterior, mas precisamente en EEUU, ya que las alternativas que

[3] La configuración del manipulador es el conjunto de posiciones y velocidades instantáneas de cada uno de los motores, que determinan la posición del extremo del manipulador y su orientación.

teníamos para la construcción de la misma en nuestro país no nos resultaron satisfactorias.

Las alternativas que se manejaron al momento de elegir en dispositivo que fuera capaz de cumplir con las funcionalidades que requieren cierta inteligencia fueron varias pero solo dos fueron alternativas verdaderamente factibles para nuestros objetivos. Por un lado la posibilidad de realizar el sistema con varios PIC de forma que cada uno de ellos se ocupara de una función específica, por ejemplo el control de un motor, y que a la ves existiera una comunicación entre ellos. Por otro lado la utilización de un módulo de Z-world RCM3200.

Al comparar las alternativas notamos claramente que el modulo de RCM3200 era la mejor alternativa por tener una velocidad de procesamiento más alta, mayor capacidad de memoria, un puerto Ethernet y un entorno de desarrollo interesante. Además de personas allegadas con cierta experiencia en el manejo del mismo, motivo nada despreciable en el momento que se tomó la decisión.

A. Placa de control

La placa de control esta compuesta por bloques que en su conjunto determinan, a partir de las órdenes recibidas del módulo RCM3200, el comportamiento del manipulador. A través de los conectores J1 y J2 se intercambian un conjunto de señales lógicas entre la placa de control y el módulo *RCM3200*, al mismo tiempo que las señales generadas en todo el bloque de control se derivan a la placa de potencia para manejar los motores DC.
Dentro de esta placa de control se pueden distinguir dos circuitos integrados *LM629* que realizan cada uno el control de un motor, un bloque que se encarga de recibir del manipulador posibles alarmas sobre algún comportamiento anormal, un bloque para realizar el control de torque del motor que se encarga de la apertura y cierre de la pinza, elementos que implementan funcionalidades que propician el correcto funcionamiento del módulo *RCM3200*, así como cinco buffers que tienen como función la protección de los elementos de la placa contra posibles sobre corrientes.

También esta incluida en la placa de control la electrónica que implementa la interfaz de programación par el *Rabbit,* el elemento principal de esta interfaz es el chip *MAX3232* que realiza una conversión eléctrica entre niveles CMOS que maneja el microprocesador con los niveles RS-232 que maneja el puerto serie de la PC.
Cabe mencionar que la comunicación con la PC se hace a través del conector DB-9 (macho) instalado en la placa, conectando al mismo un cable hembra-hembra directamente a un puerto serie de la PC.

Otros elementos que incluye la placa de control son:

- Batería que permite mantener la información en la memoria RAM mientras la placa no se encuentra alimentada.
- Botón de reset para reiniciar el *Rabbit*.
- Jumpers SMODE0 y SMODE1 que seleccionan modo de programación o modo *bootstrap*.
- Un circuito oscilador de 7 MHz. implementado con un cristal que sirve para generar la señal de reloj que luego es utilizada tanto por ambos *LM629* como por un *Flip-Flop JK* utilizado en el bloque de las alarmas.

1) Bloque de alarmas

Como se menciono en la sección anterior, existen a lo largo del manipulador interruptores que indican cuando alguna articulación alcanza un punto critico.
Estas señales ingresan a la placa de potencia a unos *Expansores I2C* los cuales detectan un cambio en cualquiera de sus entradas y lo comunican al *Rabbit* a través de una interrupción, además deshabilitan los motores directamente desde hardware (utilizando un circuito implementado con un Flip Flop JK).

El microprocesador, en su rutina de atención a interrupción, es capaz de consultar a los expansores cual fue el interruptor accionado (mediante una comunicación serie), para luego actuar de acuerdo a cada caso.

2) Bloque de control de torque (Pinza)

Para controlar la corriente que circula por el motor de la pinza (y por consecuencia el torque), se utilizo una particularidad de los Drivers LMD18200 (descriptos mas adelante) que entregan por uno de sus pines una corriente proporcional a la corriente entregada al motor.
De esta manera, conectando dicho pin con una resistencia a tierra se obtiene un valor analógico de voltaje proporcional a la corriente por el motor. Este valor luego sera comparado con un valor generado por el microprocesador (*Rabbit*) a través de un conversor digital-analógico.

La salida de esa comparación es utilizada cono disparador en un timer LM555 que es el que genera la señal PWM a ser entregada al motor.

B. Placa de potencia

Le llamamos placa de potencia a la placa donde se encuentra ubicado el conjunto de componentes electrónicos, que transforma las señales de control de tipo lógico en un nivel de potencia suficiente para alimentar los motores (mas precisamente las señales entregadas a los motores son señales PWM –modulación por ancho de pulso-). A la misma llega la información necesaria para comandar los *drivers* y éstos entregan un nivel de tensión directamente a cada motor. Se pueden identificar en la placa cinco módulos o bloques uno correspondiente a cada motor. Tres de ellos son similares, ya que los tres alimentan cada uno a un motor DC de 12v e intercambian las mismas señales con el bloque

de control. Los otros dos difieren levemente ya que uno de ellos alimenta un motor DC de 24v por lo que el bloque tiene incorporado un elevador de tensión y el otro alimenta el motor que se encarga de la apertura y cierre de la pinza por lo que las señales que intercambia con el bloque de control difieren con las que intercambian los otros bloques.

1) Drivers (Puentes H)

Cada bloque o módulo de los que se mencionaron tiene incorporado un circuito integrado *LMD18200*. El *LMD18200* es un circuito integrado con capacidad de manejar tanto motores DC como motores de paso. Combina tecnologías bipolar y CMOS dentro de un bloque lógico interno con dispositivos de potencia DMOS que conforman un puente H. Puede comandar corrientes de 3A en forma continua así como picos de hasta 6A. También implementa una salida analógica que entrega 337uA por Amper consumido por el motor.[4]

2) Generación de alarma por exceso de corriente

A la señal *current sense* (salida de corriente proporcional a la corriente por el motor) entregada por el dispositivo *LMD18200* se le conecta entre ésta y tierra una resistencia de valor conocido, con esto tenemos un nivel de voltaje sobre la misma proporcional a la corriente que está circulando por el motor.
Esto fue utilizado para dos fines; por un lado para controlar que la corriente que consuman los motores no extralimiten un umbral predeterminado, y por otro lado en el caso del motor que maneja la apertura y cierre de la pinza para realizar un control sobre el torque que genera el motor.

VI. CONTROL DE MOVIMIENTO

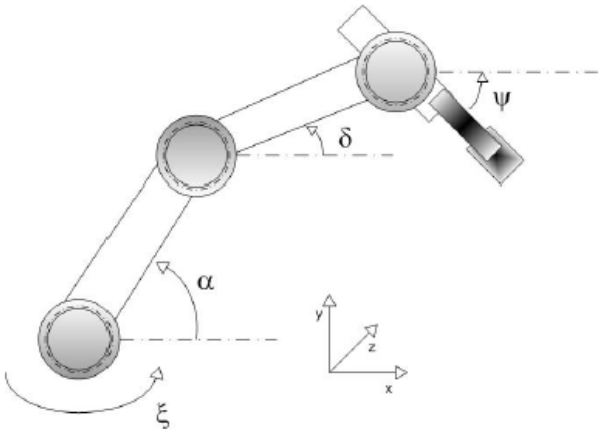
A. Introducción

El manipulador robótico es básicamente un sistema posicionador con múltiples grados de libertad. La dinámica del robot es un sistema con varias entradas y varias salidas y no lineal. El principal problema para el control de movimiento es la complejidad de la dinámica resultante de la no linealidad y los múltiples grados de libertad ambos parametritos y dinámicos.

La indeterminación paramétrica viene del hecho que se tiene un conocimiento impreciso de los parámetros de inercia, mientras que la indeterminación dinámica viene de la flexibilidad de las articulaciones, la fricción de las distintas partes, el posible ruido en los sensores de posición y el desconocimientos de la dinámica del ambiente.

[4] Por mas información consultar la hoja de datos del chip

Consideramos el manipulador como un sistema de 5 tramos interconectados por articulaciones o junturas que conforman una cadena cinemática. En nuestro caso todas las junturas son de revolución.



B. Cinemática

Definimos las variables de las articulaciones q1, q2, q3, q4 como los ángulos entre los distintos tramos, de esta manera qi es el ángulo entre el tramo i e i-1. En nuestro sistema de coordenadas (ver sección de sistema de coordenadas), dichas variables se traducirían en:

- $q1 = \xi$
- $q2 = \alpha + 90º$
- $q3 = 180º - (\alpha - \delta)$
- $q4 = 180º - (\delta - \psi)$

Un vector $q = (q1, q2, q3, q4)T$ es llamado configuración. El conjunto de todas las posibles configuraciones es llamado Espacio de Configuraciones (C) .
El Espacio de actuador es el espacio de todas las posiciones y orientaciones del actuador o pinza.

El mapeo de Cinemática directa es una función del espacio de configuraciones al espacio del actuador en términos de la posición de las articulaciones. El mapeo de Cinemática inversa da la configuración de las articulaciones como función de la posición y orientación del actuador.

El mapeo de cinemática directa se puede definir como una función del espacio de configuración en R^6.

$$X_0 = \begin{pmatrix} x(q) \\ o(q) \end{pmatrix}$$

donde $x(q) \in R^3$ da la posición cartesiana del actuador y $o(q)=(\varphi(q), \theta(q), \gamma(q))$ representa la orientación del actuador.

La Cinemática de velocidad se refiere a la relación entre la velocidad de las articulaciones y la velocidad del actuador. Si se utiliza el mapeo de cinemática directa visto anteriormente, entonces la velocidades están dadas por :

$$V = \begin{pmatrix} v \\ \omega \end{pmatrix} = J_0(q)\dot{q}$$

donde Jo(q) es una matriz 6x4 llamada la Jacobiana del manipulador. Los vectores ν y ω representan la velocidad lineal y angular respectivamente del actuador.

C. *Dinámica*

La dinámica del manipulador se describirá por el la forma Lagrangeana. En esta visión las variables de los ángulos de las articulaciones sirven como coordenadas generalizadas.

Escribiendo las dinámica del sistema por las ecuaciones *Lagrangeanas:*

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_k} - \frac{\partial L}{\partial q_k} = \tau_k, k = 1,..,4$$

donde L=K - P es el Lagrangeano y τ1,.., τ4 representan las fuerzas generalizadas de entrada.

Planteando el problema en forma matricial, se obtiene:

$$D(q)\ddot{q} + C(q,\dot{q}) + g(q) = \tau$$

donde g(q) representa las fuerzas gravitacionales y C las fuerzas no conservativas. La matriz D representa la matriz de inercia de las articulaciones, además en este caso en que se tienen motores con grandes reducciones, es importante incluir las inercias de los motores. Estas inercias son especificadas por una matriz diagonal 4 x 4.

$$I = diag(I_1 r_1^2,..,I_4^2),$$

donde Ii y ri son la inercia del motor y el factor de reducción respectivamente de la juntura i. Definiendo la matriz M(q) = D(q) + I modificamos la dinámica con este termino adicional:

$$M(q)\ddot{q} + C(q,\dot{q}) + g(q) = \tau$$

Al introducir la dinámica de los elementos actuadores o motores, se obtiene:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = K_m i$$
$$L\frac{di}{dt} + Ri = V - K_b\dot{q}$$

donde i, V son vectores representando las corrientes de armadura y voltaje respectivamente y L,R,Kb son matrices, las cuales representan inductancias de armadura, resistencia de armadura y la constante de fem inversa (multiplicado por el factor de reducción).

D. *Controlador*

A la luz de la complejidad de la dinámica de un sistema robótico de varias articulaciones, es remarcable descubrir que leyes relativamente simples de control pueden ser aplicadas en la mayoría de los casos.
Luego de estudiar y analizar el problema, llegamos a la conclusión de que en general, estos sistemas pueden llegar a controlarse satisfactoriamente con esquemas de control como el implementado por nosotros, utilizando bloques con controladores PID para cada actuador (motor).

Utilizando un control realimentado PID individualmente para cada uno de los actuadores y mediante un sistema de inteligencia superior que los coordine y supervise que se pueden lograr resultados más que satisfactorios en control de movimiento del manipulador. Otro punto a favor de la utilización de este tipo de controlador es la relativa fácil implementación del mismo, y que es una solución muy estudiada y ampliamente implementada en el mercado ya sea en versiones de hardware o software. Nosotros hemos optado por una solución mixta entre controladores implementados en hardware, como ser los microcontroladores *LM629*, y controladores de software, como ser las librerías de control del *Rabbit 3000*.

VII. DESARROLLO DE SOFTWARE

El SW de *Umanis* fue escrito en el lenguaje de programación *Dynamic C 7.32*, el cual fue desarrollado por *Rabbit Semiconductors*, para programar su línea de microprocesadores. Este lenguaje es un derivado del lenguaje de programación C. En nuestro caso, al igual que en *Sulla*, utilizamos la placa *RCM3200* de *Zworld*, la cual contiene un microprocesador *R3000*. Su arquitectura es muy similar a la del clásico *Z80*. Esta placa
no sólo incluye el microprocesador, sino que también incluye una serie de periféricos de especial interés para nosotros, como ser: decodificadores de encoders, generadores de señal PWM, varios puertos serie, relojes de tiempo real, etc.

A. *Dynamic C*

La mayor ventaja de utilizar este entorno de programación, y por ende el RCM3200, es la cantidad de código que existe implementado en forma de librerías, pensado para ser utilizado en sistemas embebidos, y además de eso, existen infinidad de examples en cada área. Como ejemplo de esto podemos mencionar las librerías de uC/OS-II ucos2.lib, las librerías de control de motores servo.lib, las librerías de TCP/IP dcrtcp.lib, las librerías de manejo de bus I2C

i2c.lib, etc. Esto permite ahorrar considerablemente tiempo de desarrollo.

Por información detallada del entorno de programación y la placa utilizada, se recomienda la lectura de los siguientes manuales [5]:
• Rabbit 3000 Microprocessor User's Manual
• RabbitCore RCM3200 User's Manual
• Dynamic C User's Manual
• Dynamic C TCP/IP User's Manual

Podemos encontrar que en este lenguaje es permitida la modularización del SW en varios archivos llamados librerías o bibliotecas (extensión .lib), al igual que en el código C, pero a diferencia del mismo, sólo podrá haber un archivo con extensión ".c" en el cual se encontrará el "main" del programa principal. En nuestro caso ese archivo se llama Umanis-main.c. Otra diferencia a destacar con el Lenguaje de programación C tradicional ese que no existen los archivos de header, con extensión ".h", sino que se encuentra tanto el prototipo de las funciones como la implementación de las mismas dentro de los archivos de librería.

B. Arquitectura de Software

Todo comienza con el archivo "Umanis-main.c", el cual inicializa el Sistema Operativo y realiza el anidamiento de librerías. Para ilustrar la arquitectura y el front-end de Umanis, se encuentra la siguiente figura.

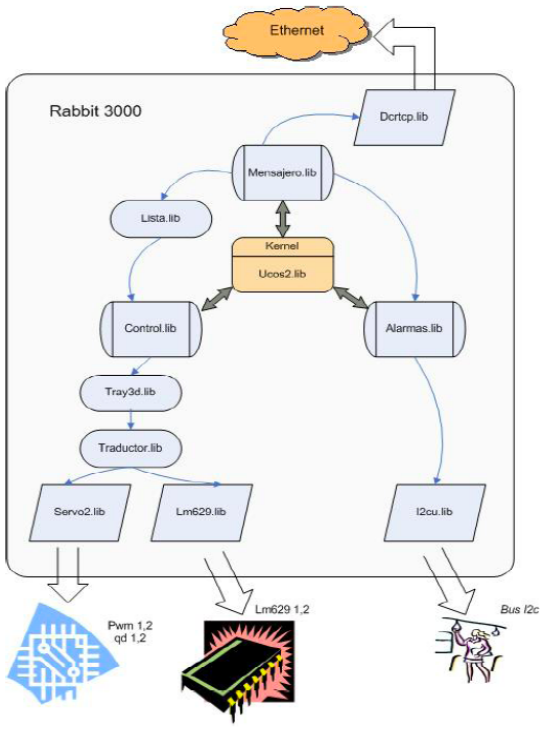


Fig. V.1 – Arquitectura de Software

[5] Todos estos manuales pueden ser descargados gratuitamente en la página de Zworld (www.zworld.com)

Comencemos por los rasgos generales. En el centro del diagrama encontramos al Kernel o Sistema Operativo. Este es responsable de generar y administrar tres Tareas:
• Una tarea de Control, implementada en Control.lib.
• Una tarea de Mensajería, implementada en Mensajero.lib.
• Una tarea de gestión de Alarmas, implementada en Alarmas.lib

1) Funcionamiento:

Luego que el sistema es inicializado, se pone en espera a recibir un comando vía un paquete UDP. Una vez que el módulo mensajero.lib recibe un comando válido [6], lo decodifica, y procede a encolarlo en el módulo lista.lib. Si la tarea de control, implementada en control.lib se encuentra en estado IDLE (desocupada), va a buscar el próximo comando a la lista. Cabe destacar que en la lista, los comandos se encolan en formato FIFO (el primero en entrar, es el primero en salir). Supongamos que este comando es una trayectoria que involucra el movimiento de todas las articulaciones del manipulador, entonces, lo primero que deberá hacer el módulo de control, es generar la trayectoria implicada en el comando, de esa tarea se encargará la librería tray3d.lib (por más información sobre cómo es generada la trayectoria, ver la sección de Generador de Trayectorias).

Inmediatamente esa información deberá ser traducida en parámetros entendibles por las interfaces de bajo nivel que ejecutan el movimiento, y de esa "traducción", es la que se encarga el módulo traductor.lib. Luego que sucedió todo esto ingresamos el vector resultante en el loop de control, el cual se encargará de ejecutar los comandos de control de posición de manera sincrónica para todos los motores involucrados, e inyectarlos con la cadencia adecuada.

Mediante la ayuda del SO, podemos, a la misma vez a que todo esto sucede, estar consultando mediante la tarea de gestión de alarmas llamada alarmas.lib que no se haya dado ningún comportamiento anormal que implique el daño del manipulador, y también continuar a la espera de la recepción de nuevos comandos.

C. El planeador de trayectorias de Umanis

1) Introducción

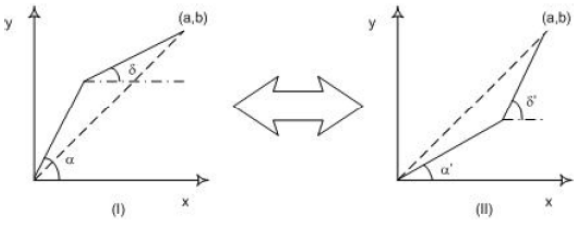
El planeador de trayectorias de Umanis es el módulo de Software (más precisamente la librería tray3d.lib) encargado de generar vectores de trayectorias expresados en las coordenadas naturales del sistema para recorrer una trayectoria dada. Éste es un módulo crítico ya que será el que le permitirá al manipulador realizar movimientos coordinados.

La filosofía que adoptamos para la creación de este módulo fue la siguiente: Atacar el problema de lo general a lo

particular; por lo que el generador de trayectorias es capaz de generar trayectorias en línea recta entre dos puntos cuales quiera del espacio, siempre y cuándo estos pertenezcan al espacio de trabajo del manipulador.
Estas trayectorias pueden tener una cantidad de muestras arbitraria. Luego con la composición de pequeñas trayectorias en línea recta, se pueden generar trayectorias más complejas.

2) Análisis Matemático

Dados dos puntos en el espacio de trabajo del manipulador expresados en coordenadas cartesianas, se genera una cantidad arbitraria de puntos equidistantes.
La consigna es realizar la llamada "cinemática inversa" para hallar la configuración del manipulador que tenga como consecuencia el posicionamiento del eje de la muñeca en los puntos dados y que además sea "continua", porque como veremos la gran dificultad de la cinemática inversa es la multiplicidad de soluciones encontradas para un mismo punto dado en el espacio.



Para solucionar este problema se adoptaron soluciones del tipo de configuración (I) para todas las trayectorias, o como las llamamos nosotros, configuración "grúa" (en la jerga de la robótica es llamada "elbow-up configuration"). Esta elección es más que conveniente para el aspecto mecánico, ya que si se mantiene la configuración "grúa" para todos los puntos de la trayectoria el torque resistivo en la articulación siempre tendrá el mismo sentido, por lo que se evita el juego en dicha articulación.

Dicho esto, pasemos a la solución general del problema en tres dimensiones. Dado un punto A=(x,y,z), perteneciente al espacio de trabajo del manipulador, encontremos la configuración en coordenadas naturales del sistema determinada, por la 3-upla (α,δ,ξ), tal que posicione el eje de la muñeca en el punto A.

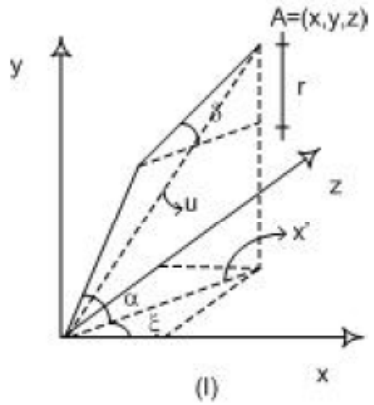


Fig. V.6 – Caso tridimensional

Consideramos los dos segmentos de igual tamaño y de largo a. entonces se llega a las siguientes ecuaciones:

$$\alpha = \cos^{-1}\left[\frac{\sqrt{(\sqrt{x^2+y^2})+y^2}}{2a}\right] + \tan^{-1}\left(\frac{y}{\sqrt{x^2+z^2}}\right)$$

$$\partial = sen^{-1}\left(\frac{y - a.sen(\alpha)}{a}\right)$$

$$\xi = \tan^{-1}\left(\frac{z}{x}\right)$$

3) Funcionamiento

Primero, digamos que podemos considerar una trayectoria de nuestro sistema a un conjunto de "configuraciones" del mismo expresadas en sus coordenadas naturales.
Entonces nos encontraremos ante la necesidad de crear una estructura de datos que ampare estas configuraciones.
Entonces fue creado el tipo de datos corn (a saber, "*cor*", acrónimo de coordenada y "n" hace referencia a natural, por ser las coordenadas del sistema), el cual contiene 5 valores correspondientes a cada una de las articulaciones.
Además, si examinamos el punto de partida de nuestro generador: dos puntos en el espacio, también nos vimos en la necesidad de una estructura de datos que ampare el sistema de coordenadas cartesianas, se creo el tipo de datos *cor* que contiene 3 valores correspondientes a las 3 coordenadas (x,y,z).

En este momento nos encontramos en condiciones de redefinir nuestro problema en términos más cercanos al desarrollo. Entonces la nueva consigna sería: Dados dos puntos en el espacio definidos por variables del tipo de datos *cor*, encontrar la trayectoria en línea recta con un número arbitrario de muestras (puntos de esa trayectoria), expresada en datos de tipo *corn* (coordenadas naturales del sistema).

Este problema fue resuelto en varias etapas. Primero definimos estructuras de datos para las trayectorias, a saber : *tray_xyz y tray_nat*. Estas estructuras no son más que arrays de los tipos de datos definidos anteriormente de tamaño *TRAY_MAX_CANT_ELEM* (esta constante está definida en la librería de constantes *consantes.lib* y puede ser manipulado a gusto por el futuro desarrollador).
Luego creamos una función que es capaz de generar la trayectoria en línea recta entre dos puntos en coordenadas cartesianas. El encabezado es el siguiente: *void tray3d_xyz(tray_xyz* traye, cor inicio, cor final, int muestra);*

De modo que en este momento nos encontraríamos en poder de una trayectoria en expresada en coordenadas cartesianas, y mediante el uso de la función *tray3d_nat*,

será expresada en coordenadas naturales, por lo que habremos alcanzado nuestro objetivo. *void tray3d_nat(tray_xyz* traye, tray_nat* traye_n).*

D. Sistema Operativo

Luego de investigar las diferentes opciones, decidimos utilizar uC/OS-II. Este sistema operativo fue concebido especialmente para trabajar con sistemas embebidos, y posee ciertas características especialmente útiles para sistemas de control robóticos. A saber:

- Permite implementar Preemptive Multitasking.
- Es un SO de Tiempo Real.

Estas dos características son fundamentales, porque pensemos que además de necesitar la capacidad de ejecutar varias tareas simultáneamente (controlar motores, hacer polling de alarmas, manejar las comunicaciones TCP-IP), lo que desde el punto funcional se podría hacer perfectamente en un entorno cooperativo, debemos ser capaces de manejar un sistema de prioridades bien definido, ya que el punto fundamental de nuestro sistema es controlar de manera eficiente los distintos grados de libertad de nuestro manipulador, y para ello está diseñado el Preemptive Multitasking. Por otro lado debemos ser capaces de garantizar con la mayor precision posible los intervalos de ejecución de nuestros algoritmos de control, por lo que es necesaria la característica de tiempo real.

El SO uC/OS-II dispone de colas, mensajes, banderas y semáforos que permiten enviar mensajes entre tareas y compartir recursos. Nosotros sólo usamos, al igual que *Sulla*, las capacidades de definir prioridades entre tareas, manejo de tiempo entre las mismas (OSTimeDly()) y un tipo de semáforos, llamados de mutua exclusión (mutex). Esto implica que la tarea que desee acceder a cierto recurso compartido, deberá esperar la “luz verde” para poder acceder a dicho recurso.

Estas funcionalidades nos permiten, dentro de ciertas salvedades, programar cada tarea como si se dispusiera del microprocesador solamente para ella. Es decir, por ejemplo, esto nos permitió desarrollar de manera totalmente independiente las rutinas de control de las rutinas de comunicación y alarmas.

De modo que estamos en condiciones de decir que con este SO, seremos capaces de crear “tareas” que nos den la percepción desde un entorno macro de que el sistema es capaz de realizar varias acciones simultáneamente.

Básicamente podremos definir tres tareas que deberán ejecutarse “simultáneamente” por el microprocesador, estás serán:

- Una tarea de control de motores (*task_control*)
- Una tarea de gestión de alarmas (*task_alarmas*)
- Una tarea de mensajería (*task_mensajero*)

Como es de esperarse, la tarea que será de mayor prioridad para nosotros es la tarea de control de motores, ya que esta determina la exactitud y sincronicidad de los movimientos del manipulador. Luego con prioridad dos, actuará la tarea de gestión del sistema de alarmas, la cual será capaz de interrumpir momentáneamente la ejecución del SO haciendo uso de las instrucciones de orden crítico (descriptas anteriormente). Por último, pero no menos importante, con prioridad tres encontramos a la tarea de mensajería, la cuál es la encargada de comunicar a *Umanis* con el mundo exterior así como entregar mensajes de comandos al sistema de control.

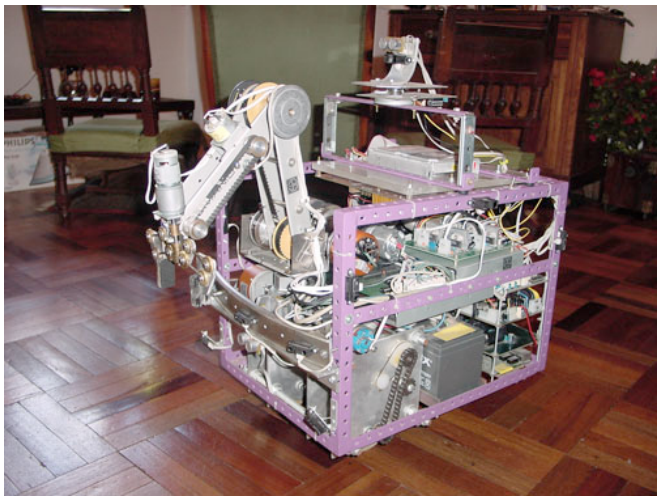
Como podemos observar, en este paradigma, existen básicamente dos elementos en la obtención del microprocesador. El primero, es la prioridad de la tarea, y el segundo, y lo que le da la genialidad y versatilidad a este sistema, es la cadencia en la que la tarea estará pronta para ser ejecutada. Esto es genial desde el punto de vista que podemos perfectamente agregar otra tarea, con una prioridad perfectamente configurable, y con la cadencia que queramos. Además de esto, por ejemplo, basta con cambiar el parámetro de cadencia de la tarea de control en el SO para alterar la frecuencia con la que se ejecuta nuestro algoritmo de control, y por lo tanto los comandos a los motores!!. Este esquema es cabalmente escalable, y nos permite agregar funcionalidades a gusto sin perjudicar de ninguna manera nuestra tarea de mayor interés, que es la tarea de control, ya que esta es la de mayor prioridad.

VIII. CONCLUSIONES

Hemos aprendido muchísimas cosas en la realización de este proyecto. Más allá del conocimiento técnico que hemos adquirido, que fue muy rico y abarcó las más diversas áreas de la Ingeniería, hemos aprendido mucho del trabajo en grupo, de la organización, de cómo lidiar con los inconvenientes más inesperados, a adoptar planes de contingencia, a trabajar en base a objetivos y a hacernos entender entre nosotros de manera productiva.

Construir un robot, es una tarea sumamente amplia, que implica el dominio de muchas áreas de la ingeniería eléctrica, como ser: la electrónica de señal, la electrónica de potencia, microprocesadores, diseño y producción de circuitos impresos, desarrollo de software, redes de datos, etc. También implicó lidiar con áreas no necesariamente relacionadas con la ingeniería eléctrica, como ser: diseño mecánico, sistemas de transmisión, creación de piezas, e infinidad de detalles que pueden sonar irrelevantes a los ojos de la ingeniería, como hacer cables, fabricar arandelas y actividades del estilo que nos consumieron muchas horas de trabajo.

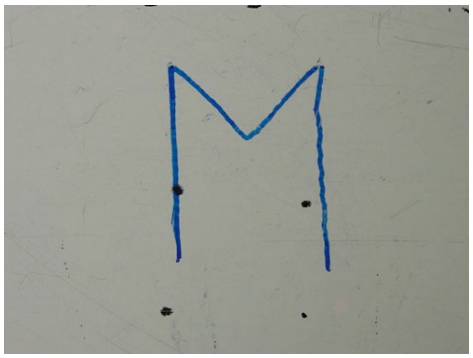
Desarrollamos nuestros conocimientos en muchas áreas, y por sobre todo logramos el feliz resultado de llegar a una implementación funcional, es decir algo que funciona. Creamos desde los cimientos un sistema integrado, y logramos que funcionara como un conjunto.



A. Logros y pruebas realizadas.

Entre las cosas que logramos hacer con Umanis se encuentran:

Seguir trayectorias en linea recta con relativamente buena presición, como caso particular logramos escribir en un pizarron ubicado en un plano horizontal distintas letras, en la siguiente figura se puede ver la letra M dibujada por el manipulador.



También cumplimos el objetivo de levantar 500g, la única limitación es que para este peso (mas específicamente para cargas mayores a 350 g) no es posible mover el motor de la muñeca, (esta limitación es impuesta por un elevador de voltaje incluido en la placa de potencia que no es capaz de entregar la corriente suficiente), de todas maneras el movimiento es posible dejando la mencionada articulación con una posición fija.

Sobre la integración con Sulla, una de las pruebas que realizamos fue ubicar un objeto en una de varias posiciones predeterminadas para luego utilizando los sensores de Sulla determinar en cual de esas posiciones se encuentra el objeto para luego agarrarlo con el manipulador y llevarlo a una posición final.

Lógicamente las posibilidades son muchas, nosotros hemos probado simplemente alguna de las cosas que nos parecian mas ilustrativas para apreciar las capacidades del conjunto robot-manipulador, pero la cantidad de opciones para posibles aplicaciones de un robot móvil autónomo con un manipulador incorporado son virtualmente ilimitadas.

La plataforma esta construida y activa, ahora se abren muchas puertas para nuevas investigaciones y nuevos proyectos para desarrollar este tema tan apasionante como la robótica.