62
2 es.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ANALISIS CINEMATICO Y PROGRAMA DE SIMULACION PARA UN MANIPULADOR ANTROPOMORFICO TELEOPERADO

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO MECANICO ELECTRICISTA
AREA MECANICA

PRESENTA

HUGO FIGUEROA ROSAS

DIRECTOR DE TESIS

M. I. VICTOR JAVIER GONZALEZ VILLELA

CIUDAD UNIVERSITARIA *1998*

A mis padres, a mis hermanos y a mis amigos

Un agradecimiento especial al M. I. Víctor Javier González Villela
y al Departamento de Mecatrónica de la Facultad de Ingeniería
por la confianza y el apoyo brindados en la realización de este trabajo

CONTENIDO

ILUSTRACIONES

CAPÍTULO 1

Introducción

En este trabajo se presentan los resultados obtenidos a partir del desarrollo del módulo de visualización y procesamiento de información, perteneciente al proyecto integral de teleoperación denominado Manipulador Antropomórfico Teleoperado (MAT), el cual se lleva a cabo dentro del Departamento de Mecatrónica de la Facultad de Ingeniería, y consiste a grandes rasgos en un manipulador semejante a una mano humana controlado mediante un elemento sensor colocado sobre la mano de un usuario.

La justificación de un proyecto como este, está dada por el hecho de que muchas veces los brazos mecánicos industriales poseen en su parte terminal algún tipo de herramienta o dispositivo de sujeción. Sin embargo, gran parte de éstos son muy simples y generalmente se fabrican de manera que puedan realizar una tarea muy específica; lo cual trae como consecuencia que a veces manifiesten una cierta "torpeza". Ésta se ve reflejada en algunas circunstancias tales como: la falta de destreza en la manipulación; el número limitado de tipos de sujeción que pueden lograr y que provoca que las herramientas tengan que sustituirse por otra adecuada; y la imposibilidad para llevar a cabo movimientos muy finos con un control preciso de la fuerza aplicada.

Las manos articuladas o provistas de estructuras semejantes a dedos, ofrecen soluciones a algunos de los problemas mencionados, ya que su uso dota al robot que las posee, con la capacidad de llevar a cabo tareas de una manera versátil y con cierta destreza. La habilidad que este tipo de manos tiene para reconfigurarse y adoptar formas que permiten realizar distintos tipos de sujeción, reduce la posibilidad de tener que cambiar las herramientas de sujeción. De hecho las manos articuladas representan un concepto antropomórfico en lo que se refiere a las tareas de manipulación con destreza, ya que por ejemplo en el humano, el brazo se utiliza para posicionar la mano en una cierto lugar del espacio para después emplear tanto la muñeca como los dedos para interactuar de forma detallada con el ambiente.

A pesar de las ventajas que este tipo de manipuladores ofrece, en el terreno práctico hay que saber valorar si realmente resultan ser la mejor solución, ya que si bien se gana en el terreno de la manipulación, también hay que pagar un precio elevado por lo que toca al sistema de control, ya que se vuelve más complejo. Generalmente las necesidades específicas de una cierta aplicación son las que van determinar si vale la pena adoptar esta clase de sistemas de manipulación.

Desde hace mucho tiempo se han tratado de fabricar diferentes tipos de manos mecánicas; sólo que en aquel entonces la motivación de su desarrollo estaba

más bien centrado en la elaboración de prótesis. Ahora los motivos están más allá de querer fabricar miembros de reemplazo, ya que además eso, han surgido áreas de aplicación bastante amplias, como lo puede ser por ejemplo la *teleoperación*. Entre los intentos más recientes que se han hecho para diseñar dispositivos antropomórficos de sujeción se hallan por ejemplo: la mano mecánica Salisbury, también conocida como Stanford/JPL (ver Foto 1.1); la mano Jameson (JH-2) desarrollada en el Centro Espacial Johnson para aplicaciones en el espacio exterior (ver Foto 1.2); y la mano UTAH/MIT (ver Foto 1.3).

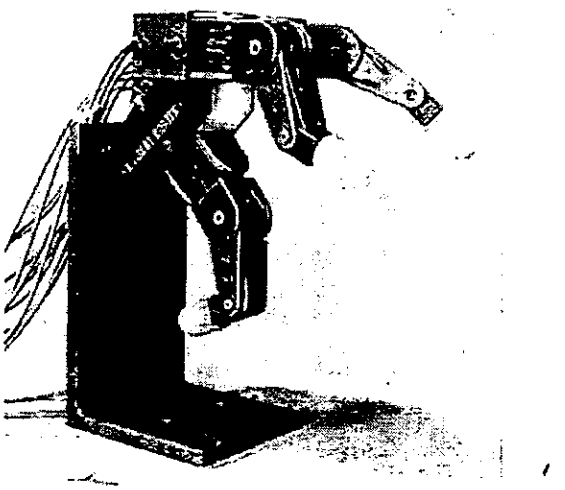

Foto 1.1 Mano mecánica Salisbury (Stanford / JPL).

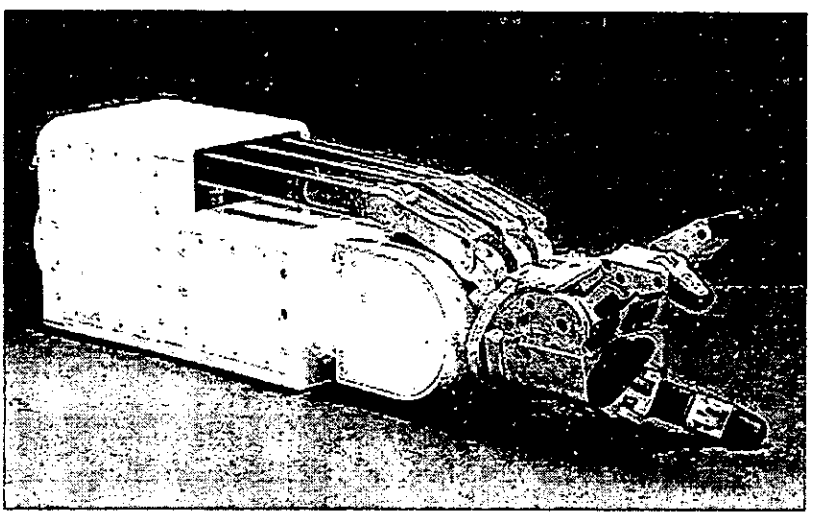

Foto 1.2 Mano mecánica Jameson (JH-2).

Como se dijo, la teleoperación ha motivado en buena parte el interés en la investigación y desarrollo de este tipo de sistemas mecánicos. Esto se debe a que con la teleoperación se puede tener acceso a ambientes de trabajo tan desfavorables como pueden ser: lugares con radioactividad, debajo de la superficie marina, en el espacio exterior, debajo de la superficie terrestre, y así sucesivamente. Últimamente se han visto las ventajas que su uso podría tener en aplicaciones quirúrgicas, en donde el cirujano puede realizar una operación estando lejos del lugar en donde ésta se está llevando a efecto.

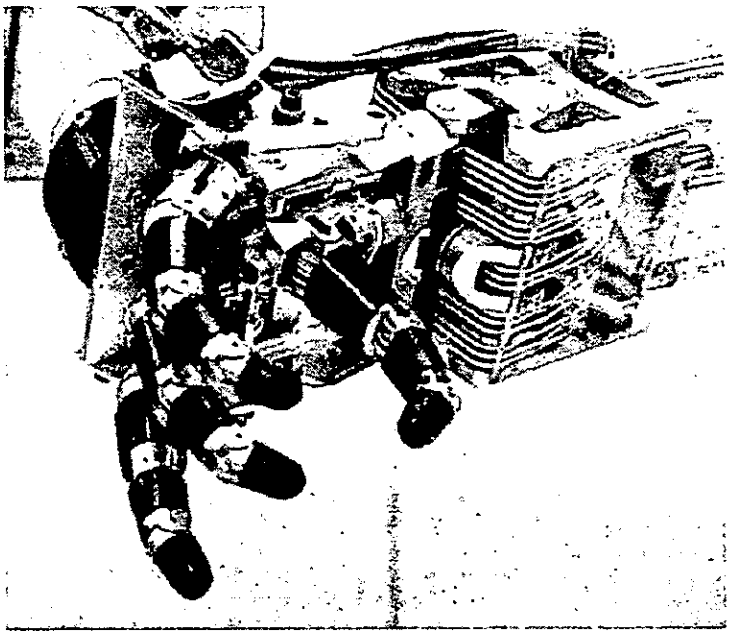

Foto 1.3 Mano mecánica Utah / MIT.

1.1 Proyecto Manipulador Antropomórfico Teleoperado (MAT)

Como se ha mencionado, el proyecto Manipulador Antropomórfico Teleoperado (MAT) tiene como propósito crear un manipulador similar a una mano humana, que sea capaz de controlarse mediante un sistema de operación intuitivo colocado en la mano del operador. Las partes del proyecto pueden agruparse en cuatro módulos principales: un sistema de sensado de los movimientos de los dedos de la mano humana; un sistema de visualización, simulación y procesamiento de datos; un sistema de control y actuación; y el mecanismo del manipulador.

Vale la pena resaltar que el principal objetivo del proyecto, es el de poner en funcionamiento todos los sistemas que intervienen, a pesar de que por el momento se hayan hecho algunas simplificaciones o no se hayan tomando en cuenta todos los aspectos que afectan lo que debería ser el funcionamiento del sistema integral. Partiendo de esta primera experiencia es posible introducir mejoras en cada uno de los sistemas, de manera que el conjunto se torne más preciso y eficiente.

1.1.1 Sistema de medición de la posición de los dedos

Como su nombre lo indica, tiene a su cargo la medición de las posiciones angulares relativas en las articulaciones de los dedos de la mano del usuario. El dispositivo está compuesto por un guante flexible sobre el que se montan una serie de pares diodo emisor - fototransistor infrarrojos. Estos pares se colocan exactamente sobre las articulaciones, de manera que cada uno de los elementos del sensor queda en lados opuestos de la articulación. Cuando en ésta ocurre algún desplazamiento, se produce un desalineamiento de los componentes del par, lo que se traduce en una variación en la señal de salida del fototransistor, la cual se aprovecha como medida del ángulo.

Además de esto, el sistema se encarga de organizar y dar formato a la salida de información para que pueda ser procesada tanto por el sistema de visualización, como por el sistema de control.

1.1.2 Sistema de visualización y procesamiento de información

La finalidad de este sistema (tratado en el presente trabajo) es la de, mediante un programa de cómputo, proporcionar al usuario información visual a modo de despliegue gráfico espacial, acerca de la posición de los dedos de su mano según el dispositivo sensor, estableciendo con este último una comunicación directa en tiempo real. Del mismo modo, este sistema es capaz de servir como plataforma para la planeación y simulación de nuevos movimientos del manipulador, haciendo uso de métodos para la generación de trayectorias a partir de unos cuantos puntos cercanos a ésta, y de rutinas que permiten transmitir la nueva información de posición al sistema de control y actuación, a fin de reproducir en el manipulador los movimientos proyectados mediante la computadora.

1.1.3 Sistema de control y actuación

Esta parte del proyecto es la que se encarga de dar movimiento a cada una de los elementos que conforman al manipulador, procurando mediante un esquema de control analógico, que se alcancen las posiciones de referencia establecidas ya sea por el sistema de medición o por el sistema de visualización y procesamiento de información.

La potencia requerida para llevar a cabo los desplazamientos del manipulador se obtiene de una serie de motores eléctricos de corriente directa, cada uno de los cuales se ocupa de un solo grado de libertad. A fin de contar a la salida del sistema de actuación con un par que permita ejercer una fuerza suficiente, se acopla una caja de reducción de velocidad a la flecha de los motores.

1.1.4 Sistema mecánico de manipulación

Este sistema es el que permite interactuar físicamente con el ambiente de trabajo y, como se ha mencionado, el aspecto del mecanismo es muy similar al de una mano humana, de ahí que se le haya dado el nombre de manipulador antropomórfico. La razón por la cual se decidió que el manipulador fuera de esta clase, está ligada a la

explicación dada anteriormente, es decir, al hecho de que la estructura de la mano permite realizar gran parte de los posibles tipos de sujeción, como por ejemplo: plana, cilíndrica, esférica, etc..

Entre las características más importantes de este manipulador, se tiene por ejemplo que se trata de la imitación de una mano derecha que únicamente cuenta con cuatro dedos: tres de los cuales cumplen con las funciones de los dedos índice, medio y anular; y uno adicional en una posición ligeramente opuesta con respecto a los demás, que adquiere las funciones de un dedo pulgar. Por lo que toca a la forma como se produce el movimiento del manipulador, cabe señalar que esto se logra mediante la acción de tendones conectados a los motores eléctricos mencionados en la parte de actuación y control.

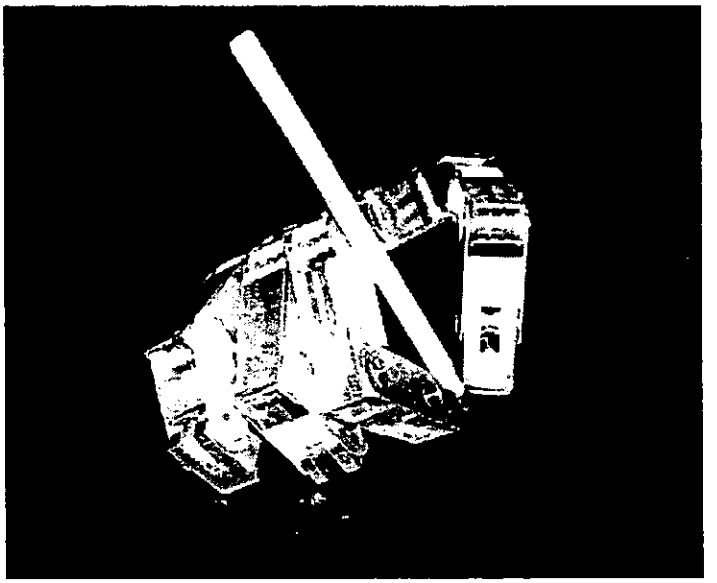

Foto 1.4 Manipulador Antropomórfico Teleoperado (MAT).

1.2 Objetivos del trabajo

A pesar de que en los apartados anteriores se hace mención de los objetivos globales del proyecto, así como de una breve explicación de las funciones de cada uno de los módulos integrantes, no está de más establecer con precisión los objetivos particulares de este trabajo, correspondiente a la parte del sistema de visualización y procesamiento de información.

El primer objetivo, es el de llevar a cabo el análisis cinemático de un manipulador antropomórfico con las restricciones y características señaladas en las especificaciones dadas en la concepción del mecanismo, para lo cual se hará uso de nuevas técnicas de análisis, sobre todo en la parte que corresponde a la cinemática directa. De la misma manera, se pretende a través de algunos otros elementos teóricos, dejar las herramientas necesarias para llevar a cabo la incorporación a los modelos de control de los aspectos dinámicos del movimiento del manipulador.

El segundo objetivo principal, es el de elaborar un programa de cómputo en el que se pueda hacer la simulación espacial de los movimientos del manipulador, a

partir del análisis cinemático que se realice, dando al usuario la posibilidad de verificar su posición de acuerdo con el sistema de medición, en caso de que no tuviera contacto visual con dicho manipulador.

Además, se pretende que dicho programa sirva como una especie de plataforma para el estudio de los movimientos realizados por el usuario, grabando y reproduciendo secuencias. Asimismo, el programa debe incorporar una herramienta para la planeación de nuevos movimientos de los dedos del manipulador, que puedan ser reproducidos en un instante posterior.

1.3 Descripción del contenido

Este trabajo está organizado en seis capítulos (incluyendo éste), tres apéndices y en una sección de información bibliográfica. En el Capítulo 2 se hace referencia a los principios generales para la descripción del movimiento de un cuerpo rígido. Comienza estableciendo las bases analíticas y matemáticas para poder representar los movimientos principales de un cuerpo rígido, tal como son rotaciones y traslaciones, haciendo uso extensivo del álgebra de vectores y matrices. También se presentan los medios para poder realizar el análisis cinemático de mecanismos de cadena abierta, haciendo hincapié en el método del producto de exponenciales. Esto se debe a las ventajas que dicho método ofrece en comparación con otros cuyo uso en aplicaciones de robótica se ha vuelto casi tradicional. Finalmente, el capítulo concluye con la presentación de la teoría que permite determinar las velocidades de los elementos que conforman a un manipulador, poniendo especial interés en la obtención del Jacobiano, el cual tiene aplicación en el desarrollo de etapas futuras del proyecto.

El Capítulo 3 recoge todos los puntos vistos en el capítulo anterior y los aplica en el análisis cinemático directo del manipulador antropomórfico. Esto es, se hace la descripción en cuanto a posición y orientación del manipulador en términos de los ángulos relativos entre sus elementos constituyentes, teniendo en cuenta todas las características físicas y de funcionamiento del sistema mecánico. Del mismo modo, se lleva a cabo la obtención de los Jacobianos tanto para los dedos índice, medio y anular, como para el dedo pulgar.

En el Capítulo 4 se tratan algunos de los aspectos que intervienen en la planeación de trayectorias para el movimiento de los dedos del manipulador. Por un lado se hace el análisis de cinemática inversa para el caso especial de los dedos, y por otro, se estudia la generación de la propia trayectoria con base en las llamadas curvas B-spline de tercer grado.

Todos los elementos vistos en los demás capítulos se conjuntan en el Capítulo 5, traduciéndose en la creación de un programa de cómputo que es capaz de interactuar con las demás partes que integran al proyecto, es decir, tanto con el sistema de medición, como con el sistema de control y actuación. Se hace la descripción en cuanto a funcionamiento y operación de cada uno de los módulos que intervienen, incluyendo los aspectos relacionados con la comunicación entre los diferentes sistemas.

Finalmente en el Capítulo 6 se muestran las conclusiones obtenidas a partir del trabajo desarrollado, así como las expectativas y posibilidades de desarrollo

futuro. En Apéndice A se incluye una breve explicación del método de Denavit-Hartenberg para la descripción cinemática de un mecanismo. Tanto el Apéndice B como el Apéndice C muestran el código fuente de los programas a los que hace referencia el Capítulo 5.

CAPÍTULO 2

Principios generales del movimiento de un cuerpo rígido

Antes de hablar del movimiento de los cuerpos rígidos, es preciso definir lo que para los propósitos de este documento se entenderá por un cuerpo rígido. Un *cuerpo rígido* es idealmente un conjunto de partículas en las que éstas siempre guardan la misma distancia entre sí, sin importar los movimientos o las fuerzas a las que esté sometido dicho cuerpo. Gran parte de los mecanismos que se emplean, se modelan a partir de lo que se conoce como una cadena cinemática, la cual está integrada por una serie de eslabones o elementos que en la mayoría de las ocasiones pueden considerarse rígidos. Es por esto que la comprensión del movimiento de estos últimos es de gran importancia para el estudio y análisis de los sistemas mecánicos.

2.1 Transformaciones de cuerpo rígido

De la definición dada anteriormente puede decirse entonces que el movimiento de un cuerpo rígido es el desplazamiento continuo de las partículas que lo integran, de tal modo que la distancia entre éstas permanece constante en todo momento. Esto puede expresarse matemáticamente del modo siguiente: Si $p_1(t)$ y $p_2(t)$ son dos puntos que representan la localización de dos partículas de un cuerpo en un tiempo dado t, entonces debe cumplirse que

$$\|p_2(t_1) - p_1(t_1)\| = \|p_2(t_2) - p_1(t_2)\| = C,$$

donde t_2 representa un tiempo posterior a t_1 y C es un valor constante. Así se garantiza (aunque no totalmente, como más adelante se verá) que el objeto en movimiento permanece en todo momento libre de deformaciones.

El movimiento de un cuerpo rígido puede ser representado a través de una función que describa el movimiento de cada una de las partículas que lo conforman. De esta manera puede definirse una función vectorial de variable vectorial, a la que se denominará *transformación*, que tenga la forma $g : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ y que precisamente transforme las coordenadas de las partículas de un cuerpo rígido, de las posiciones iniciales que guardaban, a sus posiciones finales después de haber concluido el desplazamiento. Debido a que las distancias entre las partículas de un cuerpo no pueden ser alteradas, las transformaciones que pretendan describir su movimiento

deben cumplir con una condición: que la distancia entre puntos se conserve después de su aplicación. A pesar de que la anterior es una condición necesaria, no es suficiente, ya que matemáticamente podrían presentarse reflexiones internas de las partículas, con lo que la distancia entre éstas se mantendría, pero no su orientación. Es por ello que para eliminar esta posibilidad las transformaciones también deben conservar el producto vectorial.

En resumen, $g : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ es una *transformación de cuerpo rígido* (es decir, es capaz de describir el movimiento de un cuerpo rígido) si

1. La distancia se conserva: $\|g(p_2) - g(p_1)\| = \|p_2 - p_1\|$ para todos los puntos p_1 y $p_2 \in \mathbb{R}^3$.
2. El producto vectorial se conserva: $g(v_1 \times v_2) = g(v_1) \times g(v_2)$ para todos los vectores v_1 y $v_2 \in \mathbb{R}^3$.

Con base en estas propiedades, es posible demostrar que el producto interno también se conserva al aplicar la transformación, con lo que

$$v_1^T v_2 = g(v_1)^T g(v_2).$$

En este punto vale la pena hacer una aclaración en cuanto a la definición de transformación de cuerpo rígido. En el primer caso, las coordenadas del punto están dadas por un vector de posición, mientras que en el segundo, se trata de vectores libres. Aunque operativamente no existe diferencia, conceptualmente se trata de cosas diferentes, ya que un vector de posición es aquel que parte del origen del sistema de referencia hacia el punto de interés, y un vector libre es el resultado de la diferencia de dos vectores de posición.

El hecho de que la distancia entre puntos se mantenga, al igual que el producto vectorial, no implica que no puedan existir desplazamientos relativos entre partículas; ciertamente el único movimiento que estaría permitido sería la rotación alrededor de un punto. Es por esto que para conservar una referencia de la configuración inicial del cuerpo es posible acoplar a un punto del mismo un sistema coordenado de referencia. De este modo puede realizarse el seguimiento de los movimientos del cuerpo a través del movimiento de este sistema de referencia con respecto a otro sistema fijo. A este último también se le conoce como *sistema de referencia inercial.*

Dadas las condiciones con las que cumple una transformación de cuerpo rígido, si se desea realizar la transformación sobre los vectores que conforman un sistema de referencia, entonces se obtendrá como resultado un sistema coordenado del mismo tipo, que en la mayoría de los casos es un sistema de referencia derecho[1]. Con esto, si la configuración inicial del cuerpo está dada por medio de tres vectores que formen un sistema de referencia acoplados a un punto del cuerpo, la nueva configuración del cuerpo producto de la transformación estará determinada

[1] Un sistema de referencia cartesiano derecho consta de tres vectores ortonormales **i**, **j** y **k**, en el que **k** es resultado del producto vectorial de **i** y **j**: **i** × **j** = **k**.

por los vectores transformados del sistema original de referencia, acoplados al punto original también transformado.

2.2 Rotaciones en tres dimensiones

Antes de revisar el movimiento general de un objeto, que consiste en una combinación de movimientos, tanto traslacionales como rotacionales, se comenzará con el estudio del movimiento rotacional de los cuerpos. Como ya se mencionó en la sección anterior, la orientación de un objeto puede determinarse mediante la posición relativa que guarda una sistema de referencia acoplado al objeto respecto de un sistema inercial. Sea A el sistema de referencia inercial, B el sistema de referencia acoplado al objeto, y $\mathbf{x}_{ab}$, $\mathbf{y}_{ab}$, $\mathbf{z}_{ab} \in \mathbb{R}^3$ los vectores unitarios que representan a los ejes del sistema B referidos al sistema A (ver Figura 2.1).

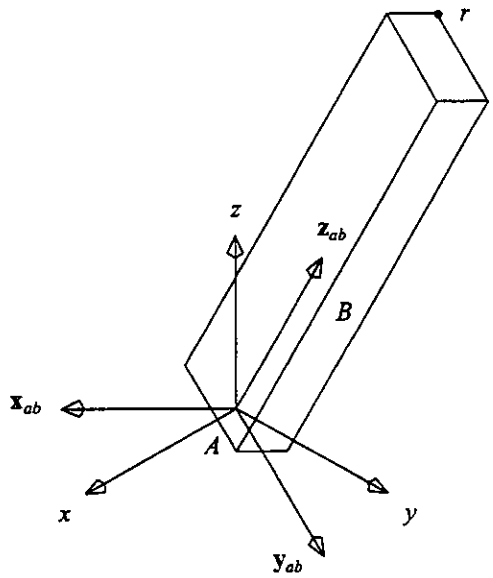

Figura 2.1 Sistema de referencia acoplado a un cuerpo rígido.

Arreglando dichos vectores a manera de columnas se obtiene una matriz de 3 × 3 como la siguiente:

$$R_{ab} = \begin{bmatrix} \mathbf{x}_{ab} & \mathbf{y}_{ab} & \mathbf{z}_{ab} \end{bmatrix}. \tag{2.1}$$

Este tipo de matrices reciben el nombre de *matrices de rotación*, y cada rotación del objeto relativa al sistema inercial corresponde a una matriz de esta forma.

2.2.1 Matrices de rotación

Debido a la manera como se construye una matriz de rotación, ésta posee características muy particulares. El hecho de que sus columnas sean vectores ortonormales implica en primer lugar que

$$RR^T = R^T R = I\,, \tag{2.2}$$

y en segundo lugar que

$$\det R = 1\,.$$

De lo anterior y bajo el contexto de las estructuras algebraicas, las matrices de rotación constituyen un subgrupo de las matrices cuadradas, en las que la operación binaria es la multiplicación entre matrices. Este subgrupo tiene la peculiaridad de que el elemento identidad es la matriz identidad, y el elemento inverso es la misma matriz pero transpuesta. Este último tipo de matrices en las que

$$R^T = R^{-1}\,,$$

reciben el nombre de *matrices ortogonales*.

Las matrices de rotación sirven como una operación de transformación, ya que convierten las coordenadas de un punto en un sistema de referencia, a sus coordenadas en otro sistema. Como ejemplo se puede considerar al punto r, en donde $r_b = (x_b, y_b, z_b)$ son las coordenadas de r relativas al sistema de referencia B. Dado que x_b, y_b, z_b son las proyecciones del punto r sobre los ejes del sistema B, y las proyecciones de estos últimos respecto del sistema de referencia A son $\mathbf{x}_{ab}$, $\mathbf{y}_{ab}$, $\mathbf{z}_{ab}$, las coordenadas de r relativas a A pueden calcularse de la siguiente manera:

$$r_a = \mathbf{x}_{ab} x_b + \mathbf{y}_{ab} y_b + \mathbf{z}_{ab} z_b\,,$$

que de una forma más elegante puede escribirse como

$$r_a = \begin{bmatrix} \mathbf{x}_{ab} & \mathbf{y}_{ab} & \mathbf{z}_{ab} \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = R_{ab} r_b\,.$$

De esto puede decirse que R_{ab} es una transformación que rota las coordenadas de un punto del sistema B al sistema de referencia A.

De la misma forma en que se tenía un sistema de referencia B relativo al sistema A, también puede existir un sistema C relativo al sistema B. La manera como se relacionan las coordenadas del sistema C con las del sistema A es a través de la multiplicación de las matrices R_{ab} y R_{bc}:

$$R_{ac} = R_{ab} R_{bc},$$

(2.3)

en donde R_{ac} se encarga de rotar las coordenadas de un punto del sistema C al sistema de referencia A, primero rotando de C a B, y después rotando de B a A.

Antes de mencionar algunas propiedades operativas de las matrices de rotación, es preciso revisar algunas características algebraicas del producto vectorial. Como se sabe, el producto vectorial (o cruz) está definido como

$$v \times w = \begin{bmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{bmatrix}.$$

Al tratarse ésta de una transformación lineal sobre el vector w, puede asociarse la expresión $(v\times)$ con una matriz de transformación de la siguiente forma:

$$(v)^{\wedge} = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix},$$

(2.4)

de modo que

$$v \times w = (v)^{\wedge} w,$$

(2.5)

o bien, de manera simplificada

$$v \times w = \hat{v} w.$$

Cabe mencionar que las matrices del tipo $\hat{v}$, son *matrices antisimétricas*, es decir

$$\hat{v}^T = -\hat{v}.$$

Las propiedades operativas de las matrices de rotación relacionadas con el producto vectorial son:

$$R(v \times w) = (Rv) \times (Rw),$$

(2.6)

$$R(v)^{\wedge} R^T = (Rv)^{\wedge}.$$

(2.7)

De estas propiedades, la segunda tiene una importancia especial, sobre todo en la deducción de las expresiones relacionadas con la velocidad de un cuerpo rígido.

2.2.2 Coordenadas exponenciales para el movimiento rotacional

Sin duda, los pares cinemáticos (o tipos de unión) de los que más hacen uso la mayoría de los robots industriales, son tanto el prismático como el rotacional. Como ya se vio, el movimiento rotacional puede representarse por medio de una matriz; sin embargo, sería conveniente que ésta pudiera determinarse como una función del eje en torno al cual se rota, y del ángulo que indica la magnitud de la rotación. Para esto, puede considerarse un punto r acoplado a un cuerpo rígido que está rotando en torno a un eje representado por el vector unitario ω (ver Figura 2.2).

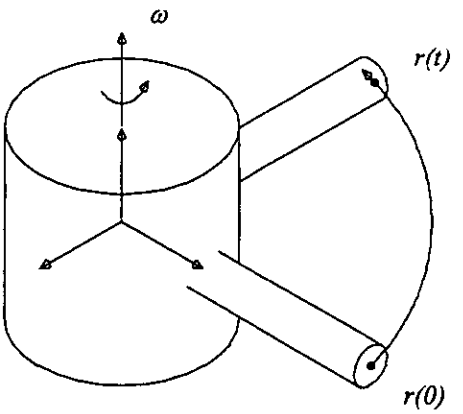

Figura 2.2 Punto rotando en torno a un eje ω.

Si la velocidad de rotación del objeto es constante y unitaria, la velocidad del punto r puede escribirse como

$$\dot{r}(t) = \omega \times r(t) = \hat{\omega} r(t) . \tag{2.8}$$

Dicha expresión es una ecuación diferencial lineal invariante con el tiempo, que puede integrarse empleando por ejemplo el método de la transformada de Laplace:

$$sr(s) - r(0) = \hat{\omega} r(s) ,$$

$$(sI - \hat{\omega}) r(s) = r(0) ,$$

$$r(s) = (sI - \hat{\omega})^{-1} r(0).$$

Recordando que para

$$f(s) = \frac{c}{s-a},$$

la antitransformada correspondiente está dada por

$$f(t) = ce^{at},$$

y haciendo la analogía con el caso matricial se tiene finalmente que

$$r(t) = e^{\hat{\omega}t} r(0), \tag{2.9}$$

donde $r(0)$ es la posición de r cuando $t = 0$ y $e^{\hat{\omega}t}$ es la función exponencial, que partiendo de su representación en forma de serie infinita para números reales, se define en forma matricial como

$$e^{\hat{\omega}t} = I + \hat{\omega}t + \frac{(\hat{\omega}t)^2}{2!} + \frac{(\hat{\omega}t)^3}{3!} + \cdots.$$

De aquí puede verse que si se rota en torno al eje ω con velocidad unitaria durante θ unidades tiempo, entonces la rotación total estará dada por

$$R(\omega, \theta) = e^{\hat{\omega}\theta}. \tag{2.10}$$

Como se acaba de ver, si se tiene una matriz $\hat{\omega}$ tal que $\|\omega\| = 1$, y un número real θ, el exponencial de $\hat{\omega}\theta$ se escribirá como

$$\exp(\hat{\omega}\theta) = e^{\hat{\omega}\theta} = I + \theta\hat{\omega} + \frac{\theta^2}{2!}\hat{\omega}^2 + \frac{\theta^3}{3!}\hat{\omega}^3 + \cdots. \tag{2.11}$$

Esta expresión es una serie infinita cuyo uso no es práctico. Para solucionar esta situación, se emplean las siguientes propiedades de las potencias de $\hat{m}$:

$$\hat{m}^2 = mm^T - \|m\|^2 I, \tag{2.12}$$

$$\hat{m}^3 = -\|m\|^2 \hat{m}, \tag{2.13}$$

que se emplean a su vez para calcular potencias superiores. Aplicando estas propiedades y haciendo $m = \omega\theta$, se tiene que

$$e^{\hat{\omega}\theta} = I + \left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \cdots\right)\hat{\omega} + \left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \frac{\theta^6}{6!} - \cdots\right)\hat{\omega}^2,$$

y recordando la expansión en series de las funciones $\cos(x)$ y $\text{sen}(x)$ finalmente queda

$$\boxed{e^{\hat{\omega}\theta} = I + (\text{sen}\,\theta)\hat{\omega} + (1 - \cos\theta)\hat{\omega}^2}, \tag{2.14}$$

expresión conocida como la *fórmula de Rodrigues*.

Geométricamente, la matriz antisimetrica $\hat{\omega}$ corresponde al eje de rotación, y la transformación exponencial se encarga de rotar una cantidad θ de radianes en torno al eje especificado.

Por otro lado si se cuenta con una matriz de rotación $R=[r_{ij}]$, los parámetros del eje de rotación y del ángulo de rotación pueden obtenerse del siguiente modo:

$$\theta = ang\cos\left(\frac{tr(R) - 1}{2}\right), \tag{2.15}$$

$$\omega = \frac{1}{2\,\text{sen}\,\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}, \tag{2.16}$$

donde $\text{tr}(R)$ es la traza de R, es decir, $r_{11} + r_{22} + r_{33}$. De aquí puede verse que a cada R corresponden dos parejas de valores del eje y del ángulo, ya que tanto ω y θ, como $-\omega$ y $2\pi - \theta$, generan la misma matriz de rotación.

Las componentes del vector $\omega\theta$ dadas por las ecuaciones (2.15) y (2.16) reciben el nombre de *coordenadas exponenciales* de R.

2.3 Movimiento de un cuerpo rígido en tres dimensiones

Hasta el momento sólo se ha revisado el movimiento rotacional de los cuerpos rígidos, sin embargo, como ya se dijo, el movimiento general de los cuerpos tiene una componente rotacional así como una componente traslacional. Esta última puede representarse escogiendo un punto del objeto para después seguir el comportamiento de sus coordenadas respecto a un sistema de referencia fijo. Al igual que en el caso de rotación, esto puede hacerse mediante el empleo de un sistema de coordenadas acoplado al cuerpo. La forma como se lleva a cabo la representación del movimiento general de un cuerpo teniendo acoplado un sistema de referencia, tiene que ver en primer lugar con la posición del origen del sistema del

objeto, y en segundo lugar con la orientación de éste con respecto al sistema inercial.

Imaginando un sistema coordenado B acoplado a un cuerpo, referido a un sistema inercial A (ver Figura 2.3), sea p_{ab} el vector de posición (en coordenadas del sistema A) del sistema B, y R_{ab} la matriz de representa la rotación del sistema B relativa al sistema A.

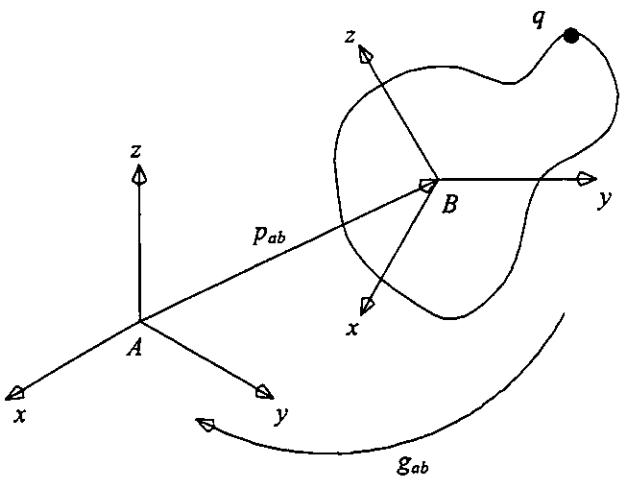

Figura 2.3 Movimiento de un cuerpo rígido.

Tanto p_{ab} como R_{ab} sirven por un lado, para hacer la descripción de la configuración que guarda el sistema B con respecto al A; y por otro lado, para llevar a cabo la transformación de coordenadas de un sistema al otro. Si q_a y q_b son respectivamente las coordenadas de una partícula del cuerpo relativas a los sistemas A y B; dadas las coordenadas de q_b, pueden hallarse las de q_a de la siguiente manera:

$$q_a = p_{ab} + R_{ab} q_b \,. \tag{2.17}$$

Haciendo que $g_{ab} = (p_{ab}, R_{ab})$ especifique la configuración del sistema B con respecto al A, y que $g_{ab}(q_b)$ exprese la acción de una transformación de cuerpo rígido sobre un punto (como se hizo en el apartado 2.1), se tendría entonces que

$$q_a = g_{ab}(q_b) = p_{ab} + R_{ab} q_b \,.$$

Para el caso de los vectores, dado que son la diferencia de dos puntos, se tendría que al aplicar sobre ellos una transformación de cuerpo rígido, únicamente tendría efecto la parte correspondiente a la rotación, por lo que

$$g(v) = Rv\,.$$

2.3.1 Representación en coordenadas homogéneas del movimiento de un cuerpo rígido en tres dimensiones

La representación matricial de las transformaciones de cuerpo rígido, ha resultado ser una herramienta muy práctica, sobre todo por su facilidad para implementarse en aplicaciones computacionales. A pesar de esto, hasta este momento la única matriz de transformación con la que se cuenta es con la de rotación, con la cual no es posible incorporar el efecto traslacional del movimiento. Para solucionar esto, lo que se hace es incrementar en uno la dimensión del espacio de trabajo, es decir, se pasa de $\mathbb{R}^3$ a $\mathbb{R}^4$. Esto se logra agregando una cuarta coordenada a los vectores de posición, que para los fines que se persiguen (que es la descripción cinemática de mecanismos) será un 1. De esta manera la imagen en $\mathbb{R}^4$ de un punto r será

$$\bar{r} = \begin{bmatrix} r_1 & r_2 & r_3 & 1 \end{bmatrix}^T .$$

Este tipo de representación recibe el nombre de *coordenadas homogéneas*, que en el caso de los vectores libres tiene la forma

$$\bar{v} = \begin{bmatrix} v_1 & v_2 & v_3 & 0 \end{bmatrix}^T .$$

Con la definición de las coordenadas homogéneas para un punto, la transformación denotada por la expresión (2.17) tendría una representación matricial del siguiente tipo:

$$\bar{q}_a = \begin{bmatrix} q_a \\ 1 \end{bmatrix} = \begin{bmatrix} R_{ab} & p_{ab} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} q_b \\ 1 \end{bmatrix} = \bar{g}_{ab}\bar{q}_b\,. \tag{2.18}$$

La matriz $\bar{g}_{ab}$ de 4 × 4 recibe el nombre de *representación homogénea de* g_{ab}. En general, si $g = (p,R)$, entonces

$$\bar{g} = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \tag{2.19}$$

Como puede apreciarse, esta matriz de transformación homogénea consta de cuatro submatrices. Las dos superiores son respectivamente la matriz de rotación y el vector de posición, mientras que las dos inferiores tienen una utilidad en cuanto a propiedades de representación gráfica, ya que la primera está relacionada con una transformación de perspectiva, y la segunda, que en este caso es un 1, es un factor de escala. El hecho de que no exista transformación de perspectiva, y de que el

factor de escala sea unitario, no afecta las características de transformación de cuerpo rígido de la matriz.

Dadas las propiedades de la submatriz de rotación, puede determinarse casi de un modo directo la matriz inversa $\overline{g}^{-1}$, siendo ésta

$$\overline{g}^{-1} = \begin{bmatrix} R^T & -R^T p \\ 0 & 1 \end{bmatrix}. \tag{2.20}$$

De igual modo que en las matrices de rotación, la composición de nuevas matrices de transformación se logra mediante el producto matricial. Así pues, si g_{bc} es la configuración de un sistema de referencia C relativo a un sistema B, y g_{ab} es la configuración del sistema B con respecto al A, la configuración de C respecto de A estará dada por

$$\overline{g}_{ac} = \overline{g}_{ab}\overline{g}_{bc} = \begin{bmatrix} R_{ab}R_{bc} & R_{ab}p_{bc} + p_{ab} \\ 0 & 1 \end{bmatrix}. \tag{2.21}$$

Con el fin de hacer más simple la notación, de aquí en adelante se utilizará g en lugar de $\overline{g}$.

Debido a la forma como se definieron las coordenadas homogéneas tanto para puntos como para vectores, la acción de la transformación de cuerpo rígido en su representación homogénea, tiene exactamente los mismos efectos que cuando no se había realizado el cambio de notación, es decir, cuando se aplica sobre un punto, se obtienen las coordenadas del punto alteradas por las componentes traslacional y rotacional, mientras que al aplicarla sobre un vector, se obtienen las coordenadas del vector afectadas únicamente por la componente rotacional.

2.3.2 Coordenadas exponenciales para el movimiento de un cuerpo rígido

Como se vio anteriormente, el movimiento rotacional puede describirse mediante una transformación de cuerpo rígido que involucra una operación exponencial matricial. El concepto de la transformación exponencial puede extenderse de modo que permita la descripción del movimiento general de los cuerpos rígidos.

Sea un par cinemático rotacional (ver Figura 2.4a) en el que ω es un vector unitario que representa al eje de rotación, y en el que q es un punto sobre el eje.

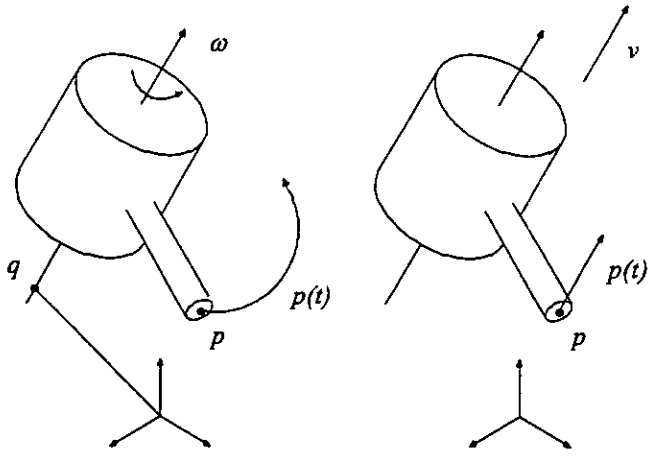

Figura 2.4 (a) Par rotacional. (b) Par prismático.

Suponiendo que el objeto rota con velocidad unitaria, la velocidad del punto $p(t)$ será entonces

$$\dot{p}(t) = \omega \times \big(p(t) - q\big).$$

(2.22)

Esta expresión puede transformarse a coordenadas homogéneas definiendo la matriz $\hat{\xi}$ de 4 × 4 tal que

$$\hat{\xi} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix},$$

(2.23)

en la que $v = -\omega \times q$, de modo que la expresión (2.22) puede reescribirse como

$$\begin{bmatrix} \dot{p} \\ 0 \end{bmatrix} = \begin{bmatrix} \hat{\omega} & -\omega \times q \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ 1 \end{bmatrix} = \hat{\xi} \begin{bmatrix} p \\ 1 \end{bmatrix} \Rightarrow \dot{\overline{p}} = \hat{\xi}\overline{p}\,.$$

(2.24)

Siguiendo el mismo procedimiento que se empleó para el caso rotacional, la solución a la expresión anterior estaría dada por

$$\overline{p}(t) = e^{\hat{\xi}t}\,\overline{p}(0),$$

(2.25)

donde $e^{\hat{\xi}t}$ es el exponencial de la matriz $\hat{\xi}t$, definido como

$$e^{\hat{\xi}t} = I + \hat{\xi}t + \frac{\left(\hat{\xi}t\right)^2}{2!} + \frac{\left(\hat{\xi}t\right)^3}{3!} + \cdots. \tag{2.26}$$

El exponencial de $\hat{\xi}t$ es la transformación de las coordenadas de un punto de su posición inicial a su posición final después de haber rotado t radianes (ya que se escogió una velocidad de rotación unitaria). Al igual que en el caso rotacional, el movimiento traslacional puede describirse a través del exponencial de una matriz de 4×4.

Por otro lado, la velocidad de un punto acoplado a un par prismático (ver Figura 2.4b) estaría dada por

$$\dot{p}(t) = v, \tag{2.27}$$

cuya solución sería una expresión similar a la (2.25), con la diferencia de que

$$\hat{\xi} = \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix}, \tag{2.28}$$

y de que t, en lugar de representar la cantidad de radianes rotados, representaría el monto total de la traslación o desplazamiento.

Haciendo una analogía con las matrices antisimétricas generadas por la operación definida en (2.4), puede definirse un elemento $\xi \in \mathbb{R}^6$ tal que

$$\xi = \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{2.29}$$

y una transformación $(\,)^{\wedge} : \mathbb{R}^6 \rightarrow (\mathbb{R}^4 \times \mathbb{R}^4)$ de modo que

$$\hat{\xi} = \begin{bmatrix} v \\ \omega \end{bmatrix}^{\wedge} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}, \tag{2.30}$$

así como su respectiva transformación inversa definida a través del operador $(\,)^{\vee}$, de tal forma que

$$\left(\hat{\xi}\right)^{\vee} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}^{\vee} = \begin{bmatrix} v \\ \omega \end{bmatrix} = \xi,$$

(2.31)

donde $\xi := (v,\omega)$ son las *coordenadas de giro* de $\hat{\xi}$. Con estas definiciones, es posible demostrar que dado un elemento $\xi \in \mathbb{R}^6$ y un número $\theta \in \mathbb{R}$, el exponencial de $\hat{\xi}\theta$ tiene la estructura de una transformación de cuerpo rígido.

Si por ejemplo, como ya se vio, $\hat{\xi}$ está dada por la expresión (2.23) y $\omega = 0$, es fácil ver que

$$\hat{\xi}^2 = \hat{\xi}^3 = \hat{\xi}^4 = \cdots = 0,$$

por lo que empleando directamente la expresión (2.26) se tiene que

$$e^{\hat{\xi}\theta} = I + \hat{\xi}\theta,$$

obteniendo finalmente:

$$\boxed{e^{\hat{\xi}\theta} = \begin{bmatrix} I & v\theta \\ 0 & 1 \end{bmatrix} \qquad \omega = 0.}$$

(2.32)

Si por el contrario, $\omega \neq 0$ de forma que $\|\omega\| = 1$, puede definirse una transformación de cuerpo rígido g como

$$g = \begin{bmatrix} I & \omega \times v \\ 0 & 1 \end{bmatrix},$$

(2.33)

la cual, empleada junto con la propiedad denotada por la expresión (2.20), sirve para obtener un elemento

$$\hat{\xi}' = g^{-1}\hat{\xi}g = \begin{bmatrix} \hat{\omega} & (\omega^T v)\omega \\ 0 & 0 \end{bmatrix}.$$

(2.34)

Dicho elemento se emplea en la siguiente igualdad:

$$e^{\hat{\xi}\theta} = e^{g(\hat{\xi}\cdot\theta)g^{-1}} = ge^{\hat{\xi}\cdot\theta}g^{-1},$$

(2.35)

en la que se asegura que basta con calcular el exponencial de $\hat{\xi}'\theta$ para obtener el resultado deseado. Con este cambio se aprovecha el hecho de que $\omega \times \omega = 0$, con lo que

$$\left(\hat{\xi}'\right)^2 = \begin{bmatrix} \hat{\omega}^2 & 0 \\ 0 & 0 \end{bmatrix}, \quad \left(\hat{\xi}'\right)^3 = \begin{bmatrix} \hat{\omega}^3 & 0 \\ 0 & 0 \end{bmatrix}, \quad \cdots.$$

Esta característica permite aprovechar los cálculos hechos para la obtención de la expresión (2.14), teniendo entonces

$$e^{\hat{\xi}'\theta} = \begin{bmatrix} e^{\hat{\omega}\theta} & (\omega^T v)\theta\omega \\ 0 & 0 \end{bmatrix},$$

de lo cual, utilizando la igualdad (2.35) se tiene finalmente

$$\boxed{e^{\hat{\xi}\theta} = \begin{bmatrix} e^{\hat{\omega}\theta} & (I - e^{\hat{\omega}\theta})(\omega \times v) + (\omega^T v)\theta\omega \\ 0 & 1 \end{bmatrix} \qquad \omega \neq 0.} \tag{2.36}$$

Como puede verse, las expresiones (2.32) y (2.36) tienen la estructura de una transformación de cuerpo rígido. A pesar de esto, su significado es diferente al que se venía manejando hasta el momento, ya que en lugar de representar la transformación de coordenadas de un sistema de referencia a otro, representa la transformación de las coordenadas de un punto desde su posición inicial en $p(0)$, hasta su posición final después de haber sido aplicado el movimiento:

$$p(\theta) = e^{\hat{\xi}\theta} p(0)$$

Vale la pena señalar que tanto $p(\theta)$ como $p(0)$ se hallan referidos al mismo sistema de referencia (al sistema de referencia fijo o inercial). De forma similar, si $g_{ab}(0)$ es la configuración inicial de un cuerpo rígido relativa a un sistema de referencia A, entonces la configuración final del cuerpo (también referida a A) después de aplicar un desplazamiento estará dada por

$$g_{ab}(\theta) = e^{\hat{\xi}\theta} g_{ab}(0).$$

(2.37)

De lo anterior puede decirse que a cada transformación de cuerpo rígido g corresponde al exponencial de algún producto $\hat{\xi}\theta$. A las componentes del vector $\xi\theta \in \mathbb{R}^6$ se les conoce con el nombre de *coordenadas exponenciales de la transformación de cuerpo rígido* g.

2.3.3 El movimiento helicoidal

En la sección anterior se vio que una transformación de cuerpo rígido puede parametrizarse a través de las coordenadas de giro. Estas coordenadas tienen características geométricas que ayudan a comprender el movimiento de los sistemas mecánicos. Para esto, considérese un movimiento de cuerpo rígido consistente, primeramente, de una rotación de θ radianes en torno a un eje en el espacio, y en segundo lugar, de una traslación de monto d a lo largo del mismo eje (ver Figura 2.5a). Este tipo de movimiento recibe el nombre de *movimiento helicoidal*, que recuerda el movimiento de un tornillo, ya que éste también gira y se traslada sobre el mismo eje. Tomando esta analogía, también se define el parámetro h, que vendría a ser lo que en un tornillo se conoce como el paso, pero que para lo propósitos del capítulo será la relación que existe de la traslación a la rotación, es decir $h = d/\theta$ (con $\theta \neq 0$). Por lo tanto, el movimiento traslacional neto después de una rotación de θ radianes estará dado por $h\theta$.

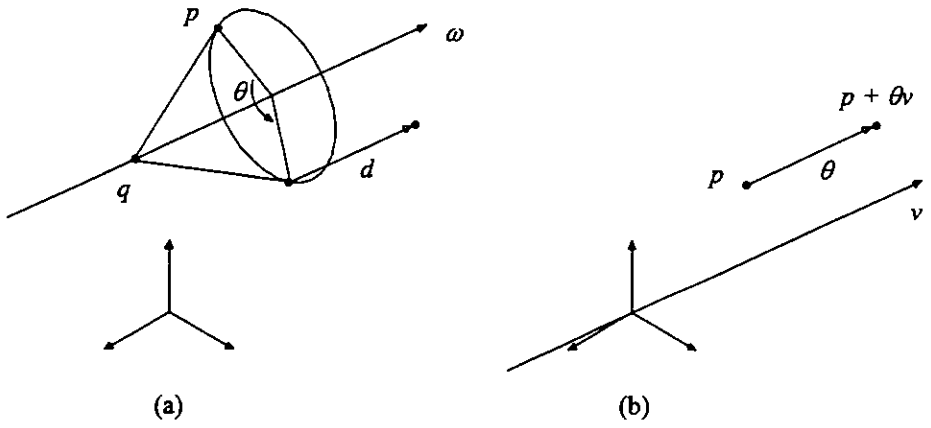

Figura 2.5 (a) Movimiento helicoidal general. (b) Traslación pura.

El eje se representa mediante la ecuación vectorial de una recta, para lo cual se requiere de un punto $q \in \mathbb{R}^3$ que esté sobre la recta, y de un vector $\omega \in \mathbb{R}^3$ que indique la dirección, por lo cual el eje quedará denotado por

$$l = \{q + \lambda\omega : \lambda \in \mathrm{R}\}. \tag{2.38}$$

Para el caso en el que la rotación valga cero, el eje estará dado por la línea que pasa a través del origen en la dirección del vector v (ver Figura 2.5b). Por

convención, el paso de este tipo de movimiento helicoidal vale ∞, y su magnitud será el monto de la traslación en la dirección de v.

Con los elementos anteriormente vistos, puede decirse que el movimiento helicoidal consta de un *eje* l, un *paso* h, y una *magnitud* M. Este movimiento representa una rotación de valor $\theta = M$ en torno a un eje l, seguida de una traslación de valor $h\theta$ paralela al eje l. Si $h = 0$, entonces se trata de una rotación pura, y si por el contrario $h = \infty$, se tratará de una traslación pura a lo largo del eje especificado, siendo M la distancia desplazada.

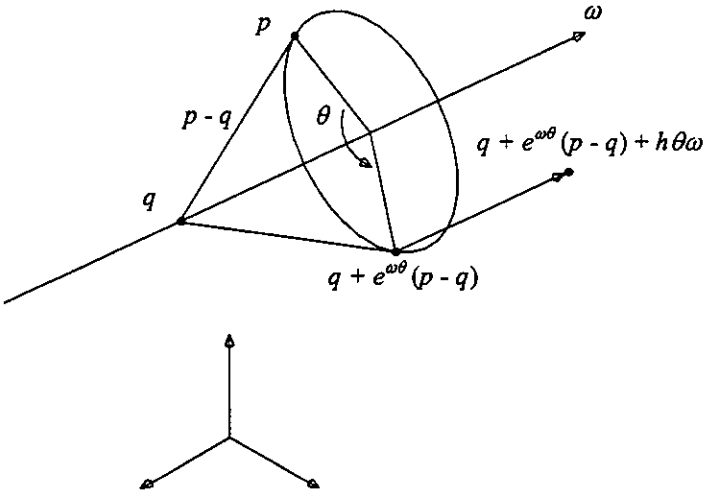

Figura 2.6 Movimiento helicoidal general.

Debido a que el movimiento helicoidal puede considerarse como un tipo de movimiento de cuerpo rígido, también puede ser asociado con una transformación. Para esto considérese un punto p (ver Figura 2.6), cuya posición final estará dada por

$$gp = q + e^{\hat{\omega}\theta}(p - q) + h\theta\omega,$$

que en coordenadas homogéneas tendría la forma de

$$g\begin{bmatrix} p \\ 1 \end{bmatrix} = \begin{bmatrix} e^{\hat{\omega}\theta} & (I - e^{\hat{\omega}\theta})q + h\theta\omega \\ 0 & 1 \end{bmatrix}\begin{bmatrix} p \\ 1 \end{bmatrix}.$$

Dado que relación debe ser válida para todo $p \in \mathbb{R}^3$, el movimiento rígido generado por la hélice (o movimiento helicoidal) será

$$g = \begin{bmatrix} e^{\hat{\omega}\theta} & \left(I - e^{\hat{\omega}\theta}\right)q + h\theta\omega \\ 0 & 1 \end{bmatrix}.$$

(2.39)

Como se pudo comprobar en la sección anterior, esta expresión transforma los puntos acoplados a un cuerpo rígido, de sus coordenadas iniciales a sus coordenadas finales después de aplicado el movimiento, sin olvidar que todos los puntos están referidos al mismo sistema de coordenadas (al sistema inercial).

Haciendo una comparación, puede verse que el desplazamiento de cuerpo rígido de la ecuación (2.39), tiene la misma forma que el exponencial de $\hat{\xi}\theta$ de la ecuación (2.36). De hecho, si en ésta se hace que

$$v = -\omega \times q + h\omega,$$

(2.40)

entonces $\xi := (v,\omega)$ produce el movimiento helicoidal de la ecuación (2.39) (suponiendo $\|\omega\| = 1$ y $\theta \neq 0$). En el caso de rotación pura ($h = 0$), las coordenadas de giro asociadas con el movimiento helicoidal generado serian $\xi := (-\omega \times q, \omega)$.

Si por otro lado se presenta el caso de traslación pura, la transformación asociada al movimiento helicoidal sería

$$g = \begin{bmatrix} I & \theta v \\ 0 & 1 \end{bmatrix},$$

(2.41)

que es precisamente el resultado que se obtendría con $\exp(\hat{\xi}\theta)$, si $\xi := (v,0)$. De esto puede verse que el movimiento helicoidal y las coordenadas de giro se encuentran relacionados casi de una forma "natural", e inclusive pueden obtenerse los parámetros de un movimiento helicoidal a partir de las coordenadas de giro. Así pues, considerando que la magnitud de ω puede ser diferente de la unidad, se tiene que el *paso* está dado por

$$h = \frac{\omega^T v}{\|\omega\|^2},$$

(2.42)

el *eje* por

$$l = \begin{cases} \left\{ \frac{\omega \times v}{\|\omega\|^2} + \lambda\omega : \lambda \in \mathrm{R} \right\}, & \text{si } \omega \neq 0 \\ \left\{ 0 + \lambda\omega : \lambda \in \mathrm{R} \right\}, & \text{si } \omega = 0 \end{cases},$$

(2.43)

y finalmente la *magnitud* por

$$M = \begin{cases} \|\omega\| , \text{ si } \omega \neq 0 \\ \|v\| , \text{ si } \omega = 0 \end{cases} . \tag{2.44}$$

De este último parámetro se puede ver que si la magnitud del movimiento helicoidal es unitaria, entonces $g = \exp(\hat{\xi}\theta)$ permite representar el movimiento de cuerpo rígido originado por un par cinemático, ya sea prismático o rotacional, donde θ sería el monto de la traslación o de la rotación según el caso.

La significación geométrica del movimiento helicoidal está contenida en el teorema de Chasles, el cual menciona que *cualquier movimiento de cuerpo rígido puede realizarse mediante una rotación en torno a un eje, combinada con una traslación paralela a dicho eje.*

Retomando algo de lo revisado en la sección anterior, ahora se puede interpretar de manera más clara el efecto de la transformación g de la expresión (2.37). Si un sistema coordenado de referencia B está acoplado a un cuerpo rígido que está siendo objeto de un movimiento helicoidal, la configuración instantánea del sistema B con respecto al A será

$$g_{ab}(\theta) = e^{\hat{\xi}\theta} g_{ab}(0) ,$$

en donde la multiplicación por $g_{ab}(0)$ transforma las coordenadas de un punto en el sistema B a las coordenadas del A, y posteriormente el exponencial transforma dicho punto a sus coordenadas finales (referidas a A).

2.4 Velocidad de un cuerpo rígido

A partir de los conceptos involucrados con las transformaciones de cuerpo rígido, pueden derivarse las expresiones que permiten describir el comportamiento instantáneo de la velocidad de un objeto en movimiento.

2.4.1 Velocidad angular

Antes de llevar a cabo la determinación de la velocidad de un objeto sujeto a la acción de un movimiento general de cuerpo rígido, se considerará por el momento únicamente el caso rotacional. Sea $R_{ab}(t)$ una matriz de rotación que representa la trayectoria de un sistema de referencia B acoplado a un objeto, cuyo origen coincide con el del sistema A, al cual se le dará el nombre de *sistema inercial*, mientras que el sistema B será llamado *sistema local*[2]. Cualquier punto q acoplado al cuerpo seguirá una trayectoria en coordenadas inerciales dada por

$$q_a(t) = R_{ab}(t) q_b ,$$

[2] Con el fin de conservar una notación internacional, en la que al sistema inercial se le refiere como *spatial frame*, y al sistema local como *body frame*, los superíndices de identificación serán "s" y "b" para el sistema inercial y para el sistema local respectivamente.

en donde q_b está fijo con respecto al sistema local de referencia. De aquí, la velocidad en coordenadas inerciales será

$$v_{q_a}(t) = \frac{d}{dt} q_a(t) = \dot{R}_{ab}(t) q_b .$$

(2.45)

En esta expresión, el término $\dot{R}_{ab}(t)$ transforma directamente las coordenadas de un punto en el sistema local, en la velocidad inercial (o referida al sistema inercial) de dicho punto. Es posible obtener una representación más compacta utilizando algunas características de la matriz $\dot{R}_{ab}$. Por lo pronto, la ecuación (2.45) puede reescribirse como

$$v_{q_a}(t) = \frac{d}{dt} q_a(t) = \dot{R}_{ab}(t) R_{ab}^{-1}(t) R_{ab}(t) q_b ,$$

(2.46)

en donde puede demostrarse que el término $\dot{R}_{ab}(t) R_{ab}^{-1}(t)$ es una matriz antisimétrica del tipo denotado por la expresión (2.4), y que por tanto es susceptible de convertirse en un vector de $\mathbb{R}^3$.

De hecho es posible definir la *velocidad angular instantánea inercial* denotada por $\omega_{ab}^s \in \mathbb{R}^3$ como

$$\hat{\omega}_{ab}^s = \dot{R}_{ab} R_{ab}^{-1} .$$

(2.47)

Así, el vector ω_{ab}^s corresponde a la velocidad angular instantánea del objeto, visto desde el sistema inercial de referencia. De forma similar se define la *velocidad angular instantánea local* denotada por $\omega_{ab}^b \in \mathbb{R}^3$ como

$$\hat{\omega}_{ab}^b = R_{ab}^{-1} \dot{R}_{ab} .$$

(2.48)

Aquí, la velocidad angular local describe la velocidad angular del objeto, vista desde el sistema de referencia instantáneo local. De las ecuaciones (2.47) y (2.48) se establece una relación entre ambas velocidades angulares:

$$\hat{\omega}_{ab}^b = R_{ab}^{-1} \hat{\omega}_{ab}^s R_{ab} ,$$

(2.49)

o también

$$\omega_{ab}^{b} = R_{ab}^{-1}\omega_{ab}^{s}.$$

(2.50)

Regresando a la ecuación (2.46), la velocidad de un punto puede expresarse en términos de su velocidad angular instantánea. De este modo, sustituyendo la expresión (2.47) en la (2.46) se tiene

$$v_{q_a}(t) = \hat{\omega}_{ab}^{s} R_{ab}(t) q_b = \omega_{ab}^{s}(t) \times q_a(t).$$

(2.51)

Si por el contrario, se desea expresar la velocidad del punto en coordenadas locales, empleando las ecuaciones (2.49) y (2.50) finalmente resulta

$$v_{q_b}(t) = R_{ab}^{T}(t) v_{q_a}(t) = \omega_{ab}^{b}(t) \times q_b.$$

(2.52)

2.4.2 Velocidad de un cuerpo rígido

Considerando el caso general en el que $g_{ab}(t)$ representa la trayectoria de un sistema de referencia acoplado a un objeto relativo a un sistema de referencia fijo, se tiene que, como ocurrió en el caso rotacional, $\dot{g}_{ab}(t)$ por sí mismo no resulta muy útil, sino que más bien los términos $\dot{g}_{ab}g_{ab}^{-1}$ y $g_{ab}^{-1}\dot{g}_{ab}$ son los que guardan una significación más cercana a los parámetros del movimiento de un cuerpo rígido. Si

$$g_{ab}(t) = \begin{bmatrix} R_{ab}(t) & p_{ab}(t) \\ 0 & 1 \end{bmatrix},$$

entonces se tiene

$$\dot{g}_{ab}g_{ab}^{-1} = \begin{bmatrix} \dot{R}_{ab} & \dot{p}_{ab} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_{ab}^{T} & -R_{ab}^{T}p_{ab} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \dot{R}_{ab}R_{ab}^{T} & -\dot{R}_{ab}R_{ab}^{T}p_{ab} + \dot{p}_{ab} \\ 0 & 0 \end{bmatrix},$$

que tiene la forma de coordenadas de giro transformadas, es decir, tiene la estructura

$$\hat{\xi} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}.$$

A partir de esto y de la analogía con la velocidad angular, se define la *velocidad inercial* $\hat{V}_{ab}^{s} \in (\mathbb{R}^4 \times \mathbb{R}^4)$ como

$$\hat{V}_{ab}^{s} = \dot{g}_{ab}g_{ab}^{-1},$$

(2.53)

o bien como

$$V_{ab}^{s} = \begin{bmatrix} v_{ab}^{s} \\ \omega_{ab}^{s} \end{bmatrix} = \begin{bmatrix} -\dot{R}_{ab}R_{ab}^{T}p_{ab} + \dot{p}_{ab} \\ \left(\dot{R}_{ab}R_{ab}^{T}\right)^{\vee} \end{bmatrix},$$

(2.54)

que tiene la forma de las coordenadas de giro, y que representa la velocidad de un punto visto desde el sistema inercial. Las coordenadas de un punto q acoplado a un cuerpo estarían dadas en coordenadas inerciales por

$$q_a(t) = g_{ab}(t)q_b .$$

Derivando esta expresión con respecto al tiempo se tiene

$$v_{q_a} = \dot{q}_a = \dot{g}_{ab}q_b = \dot{g}_{ab}g_{ab}^{-1}q_a ,$$

y por tanto

$$v_{q_a} = \hat{V}_{ab}^{s}q_a = \omega_{ab}^{s} \times q_a + v_{ab}^{s} .$$

(2.55)

Las componentes de la velocidad inercial tienen la siguiente interpretación: ω_{ab}^{s} es la velocidad angular instantánea del cuerpo visto desde el sistema inercial, y v_{ab}^{s} es la velocidad de un punto del cuerpo (posiblemente imaginario), el cual está pasando por el origen del sistema inercial en ese momento, es decir, sería la velocidad instantánea de un punto del objeto que se mediría, si se estuviera parado en el origen del sistema inercial, y estuviera pasando dicho punto por ese mismo lugar. Hay que señalar que esta última componente no es la velocidad del origen del sistema de referencia local.

De igual modo que en el caso rotacional, la velocidad de un cuerpo rígido puede especificarse con respecto al sistema (instantáneo) de coordenadas locales. Así pues se define la *velocidad local* como

$$\hat{V}_{ab}^{b} = g_{ab}^{-1}\dot{g}_{ab} = \begin{bmatrix} R_{ab}^{T}\dot{R}_{ab} & R_{ab}^{T}\dot{p}_{ab} \\ 0 & 0 \end{bmatrix},$$

(2.56)

o bien, con la estructura de coordenadas de giro como

$$V_{ab}^{b} = \begin{bmatrix} v_{ab}^{b} \\ \omega_{ab}^{b} \end{bmatrix} = \begin{bmatrix} R_{ab}^{T}\dot{p}_{ab} \\ \left(R_{ab}^{T}\dot{R}_{ab}\right)^{\vee} \end{bmatrix}.$$

(2.57)

La velocidad de un punto en el sistema local de coordenadas está dada por

$$v_{q_b} = g_{ab}^{-1} v_{q_a} = g_{ab}^{-1} \dot{g}_{ab} q_b = \hat{V}_{ab}^b(t) q_b .$$

De aquí puede verse que la acción de $\hat{V}_{ab}^b$ es la de tomar las coordenadas de un punto en coordenadas locales, y regresar la velocidad de ese punto en coordenadas también locales:

$$v_{q_b} = \hat{V}_{ab}^b q_b = \omega_{ab}^b \times q_b + v_{ab}^b . \tag{2.58}$$

La interpretación que se le da a las componentes de la velocidad local es mucho más intuitiva, en comparación con la interpretación de la velocidad inercial, ya que v_{ab}^b es la velocidad del sistema local referida al sistema inercial, pero vista en ese instante desde el sistema local; y ω_{ab}^b es la velocidad angular del sistema de referencia, también vista desde el sistema local instantáneo. En este caso hay que notar que la velocidad local no es la velocidad del objeto referida al sistema local, pues ésta, en todo momento vale cero.

Tanto la velocidad inercial como la velocidad local se pueden relacionar mediante una transformación. Para llevar a cabo esto, puede apreciarse que

$$\hat{V}_{ab}^s = \dot{g}_{ab} g_{ab}^{-1} = \left(g_{ab} g_{ab}^{-1}\right) \dot{g}_{ab} g_{ab}^{-1} = g_{ab} \left(g_{ab}^{-1} \dot{g}_{ab}\right) g_{ab}^{-1} = g_{ab} \hat{V}_{ab}^b g_{ab}^{-1} . \tag{2.59}$$

De forma alternativa puede escribirse

$$\omega_{ab}^s = R_{ab} \omega_{ab}^b ,$$

$$v_{ab}^s = -\omega_{ab}^s \times p_{ab} + \dot{p}_{ab} = p_{ab} \times \left(R_{ab} \omega_{ab}^b\right) + R_{ab} v_{ab}^b ,$$

de manera que finalmente se obtiene

$$V_{ab}^s = \begin{bmatrix} v_{ab}^s \\ \omega_{ab}^s \end{bmatrix} = \begin{bmatrix} R_{ab} & \hat{p}_{ab} R_{ab} \\ 0 & R_{ab} \end{bmatrix} \begin{bmatrix} v_{ab}^b \\ \omega_{ab}^b \end{bmatrix} . \tag{2.60}$$

Esta matriz de 6 × 6 que transforma las coordenadas de giro de un sistema de referencia a otro, recibe el nombre de *transformación adjunta asociada a* g. De manera formal puede establecerse que, dada una transformación de cuerpo rígido g, existe la transformación $\mathrm{Ad}_g : \mathbb{R}^6 \to \mathbb{R}^6$ tal que

$$\boxed{\mathrm{Ad}_g = \begin{bmatrix} R & \hat{p}R \\ 0 & R \end{bmatrix}}, \tag{2.61}$$

para cual existe inversa (antitransformada), dada por

$$\mathrm{Ad}_g^{-1} = \begin{bmatrix} R^T & -(R^T p)^{\wedge} R^T \\ 0 & R^T \end{bmatrix} = \begin{bmatrix} R^T & -R^T \hat{p} \\ 0 & R^T \end{bmatrix} = \mathrm{Ad}_{g^{-1}}. \tag{2.62}$$

Así, para el caso de la velocidad de un cuerpo rígido, la transformada adjunta "mapea" las coordenadas de giro de la velociad local, en las coordenadas de giro de la velocidad inercial:

$$V^s = \mathrm{Ad}_g V^b.$$

2.4.3 Velocidad del movimiento helicoidal

Como ya se ha visto, el movimiento helicoidal es capaz de describir completamente el desplazamiento general de un cuerpo. De la misma manera, se vio que este tipo de movimiento se encuentra relacionado de una forma íntima con las llamadas coordenadas de giro, de modo que sería conveniente establecer las expresiones para la velocidad de un objeto en términos de dichas coordenadas. Considerando el caso general en donde

$$g_{ab}(\theta) = e^{\hat{\xi}\theta} g_{ab}(0),$$

representa la configuración de un sistema de referencia B relativo a un sistema A; si $\hat{\xi}$ es constante, entonces

$$\frac{d}{dt}\left(e^{\hat{\xi}\theta}\right) = \hat{\xi}\dot{\theta}e^{\hat{\xi}\theta},$$

y la velocidad inercial estará dada por

$$\hat{V}_{ab}^s = \dot{g}_{ab}(\theta) g_{ab}^{-1}(\theta) = \left(\hat{\xi}\dot{\theta}e^{\hat{\xi}\theta} g_{ab}(0)\right)\left(g_{ab}^{-1}(0)e^{-\hat{\xi}\theta}\right),$$

por lo que la velocidad generada por el movimiento helicoidal es

$$\hat{V}_{ab}^s = \hat{\xi}\dot{\theta}.$$

De modo similar puede calcularse la velocidad local, siendo ésta

$$\hat{V}_{ab}^{b} = g_{ab}^{-1}(\theta)\dot{g}_{ab}(\theta) = \left(g_{ab}^{-1}(0)e^{-\hat{\xi}\theta}\right)\left(\hat{\xi}\dot{\theta}e^{\hat{\xi}\theta}g_{ab}(0)\right) = \left(g_{ab}^{-1}(0)\hat{\xi}g_{ab}(0)\right)\dot{\theta}$$

que aprovechando el cálculo hecho para la expresión (2.59) finalmente se tiene

$$\hat{V}_{ab}^{b} = \left(\mathrm{Ad}_{g_{ab}^{-1}(0)}\xi\right)^{\wedge}\dot{\theta}$$

Como en el caso de otras transformaciones, para el caso de la velocidad también existe una regla de composición, y puede demostrarse que para la velocidad inercial, en coordenadas de giro, es valido que

$$V_{ac}^{s} = V_{ab}^{s} + \mathrm{Ad}_{g_{ab}}V_{bc}^{s}. \tag{2.63}$$

Por otro lado, para la velocidad local se tiene que

$$V_{ac}^{b} = \mathrm{Ad}_{g_{bc}^{-1}}V_{ab}^{b} + V_{bc}^{b}. \tag{2.64}$$

Estas dos últimas ecuaciones sirven para transformar la velocidad de un cuerpo rígido entre diferentes sistemas de referencia. En muchas ocasiones, cuando se cuenta con tres sistemas de referencia, dos de estos están fijos el uno con respecto al otro, con lo que las relaciones de velocidad pueden simplificarse. Por ejemplo, si A y B son dos sistemas que permanecen fijos, uno con respecto al otro, la velocidad inercial del sistema C cumple con

$$V_{ac}^{s} = \mathrm{Ad}_{g_{ab}}V_{bc}^{s}, \tag{2.65}$$

mientras que para la velocidad local

$$V_{ac}^{b} = V_{bc}^{b}. \tag{2.66}$$

Las ecuaciones (2.63) y (2.64) también son aplicables a las coordenadas de giro que describen algún tipo de movimiento helicoidal, como aquellos que describen el comportamiento de los pares cinemáticos. Si ξ son las coordenadas de giro de un movimiento helicoidal, y se desea afectar este movimiento mediante otra transformación de cuerpo rígido g, las nuevas coordenadas de giro, empleando la ecuación (2.65), estarán dadas por

$$\xi' = \mathrm{Ad}_{g}\xi, \tag{2.67}$$

o bien por

$$\hat{\xi}' = g\hat{\xi}g^{-1}. \tag{2.68}$$

La importancia de estas igualdades radica en el hecho de que permiten saber cómo se van moviendo por ejemplo, los ejes de los eslabones de un robot.

2.5 Cinemática de un manipulador

La cinemática de un manipulador permite describir las relaciones entre las alteraciones en las variables de movimiento, y los desplazamientos que sufren los cuerpos rígidos que integran a dicho manipulador como producto de estas modificaciones. Como ya se ha mencionado, una buena parte de los robots se modelan a partir de una cadena cinemática abierta, cuyos eslabones se conectan por medio de uniones o pares cinemáticos, también llamados pares inferiores. Estos últimos, de acuerdo al tipo de movimiento que permiten, pueden clasificarse en diversos tipos, como por ejemplo: *prismático*, de *revolución*, *helicoidal*, *esférico*, *plano*, *cilíndrico*, etc. A pesar de esta variedad de mecanismos, los más empleados son sin duda tanto los pares prismáticos como los de revolución.

La cinemática de un mecanismo puede manejarse de dos modos diferentes: de modo directo o de modo inverso. En la cinemática directa, el objetivo es determinar la posición del elemento final del mecanismo a partir de los valores dados a las variables de movimiento, es decir, dados los desplazamientos relativos, ya sea angulares o lineales entre los eslabones, se pretende conocer la posición o velocidad del último elemento de la cadena (aquel que funciona como herramienta). Por otro lado, en la cinemática inversa, el objetivo es determinar el valor de los desplazamientos relativos que tiene que haber entre los eslabones, para lograr que la herramienta alcance una cierta posición y/u orientación.

A continuación se hará la descripción de un método que permite representar la cinemática directa de gran parte de los mecanismos y robots comerciales. Para el caso de la cinemática inversa, por tratarse de un método muy particular aplicable casi exclusivamente al tipo de mecanismo propuesto para la construcción del manipulador antropomórfico; se verá por conveniencia en el capítulo correspondiente a la generación de trayectorias de los dedos.

2.5.1 El producto de exponenciales

Como se ha descrito, la cinemática directa pretende determinar la configuración del eslabón final (o herramienta) a partir de la posición relativa entre los eslabones que conforman la cadena cinemática. Para llevar a cabo el análisis cinemático de un mecanismo vale la pena establecer una cierta notación: los elementos de unión (variables de movimiento) se numeran de 1 hasta n comenzando desde la base. De la misma manera se numeran los eslabones de modo que la unión i conecta los

eslabones i - 1 e i. Se considera que la base del manipulador es el eslabón 0, mientras que el n sería el último eslabón (que funge como herramienta).

Por lo que toca a las variables de movimiento y a la forma como éstas se miden, se tiene que, como se ha mencionado en varias ocasiones, los pares cinemáticos más utilizados son el prismático y el de revolución. En el caso de un par prismático, la variable de movimiento es un desplazamiento lineal a lo largo del eje, donde la dirección positiva del movimiento está dada por la correspondiente dirección positiva del vector que representa al eje. Por otro lado, para el caso de un par de revolución se tiene que la variable de movimiento está dada por un ángulo cuyo rango de valores está comprendido entre 0 y 2π, y cuya medición se realiza empleando la regla de la mano derecha[3].

Otro elemento que también es necesario para el análisis, es la designación de sistemas de referencia. Una ventaja del método que se va a emplear, es que únicamente se requiere de dos sistemas de referencia: uno fijo en la base del manipulador al que se denomina *sistema base* (S), y otro acoplado al último eslabón, que recibe el nombre de *sistema de la herramienta* (T). De lo anterior puede decirse que el sistema base funciona como el sistema inercial, mientras que el sistema de la herramienta lo hace como un sistema local.

Ya con estos elementos, lo que se pretende es determinar la configuración final de la herramienta (o del sistema T) a partir de los valores que se den a las diferentes variables de movimiento del manipulador. Esto se traduce matemáticamente en una transformación g_{st}, que es una función matricial de todas las variables de movimiento, que de acuerdo con los valores de éstas, permite transformar las coordenadas relativas a la herramienta, a coordenadas referidas al sistema inercial.

Tradicionalmente esta transformación se construye a partir de los movimientos rígidos derivados de cada una de las variables de movimiento. De esta manera, si se define una transformación $g_{l_{i-1}l_i}(\theta_i)$ entre eslabones adyacentes, puede establecerse la cinemática directa de un manipulador de cadena abierta a partir de la siguiente expresión:

$$g_{st}(\theta) = g_{sl_1}(\theta_1) g_{l_1 l_2}(\theta_2) \cdots g_{l_{n-1} l_n}(\theta_n) g_{l_n t} . \tag{2.69}$$

Una descripción más geométrica de la cinemática puede obtenerse a partir del hecho de que las coordenadas de giro están asociadas con un movimiento helicoidal capaz de describir el desplazamiento relativo entre eslabones. Si ξ corresponde a unas ciertas coordenadas de giro, el movimiento de cuerpo rígido asociado con el eje descrito por dichas coordenadas estará dado por

$$g_{ab}(\theta) = e^{\hat{\xi}\theta} g_{ab}(0) .$$

[3] Esta regla práctica consiste primeramente en colocar el dedo pulgar de manera que con respecto al resto de los dedos quede en una posición perpendicular. Después de esto se alinea el dedo pulgar con el eje de rotación de modo que se apunte en la dirección positiva del eje, tras de lo cual se flexionan los demás dedos. La dirección indicada por los dedos al flexionarse corresponde a la dirección positiva del ángulo en cuestión.

En esta expresión no está de más recordar que θ corresponde al monto del desplazamiento lineal si ξ describe el comportamiento de un par prismático; y si se trata de un par rotacional, entonces θ representa el ángulo de rotación.

Para la derivación de la expresión para describir la cinemática directa, considérese como ejemplo, a un manipulador de dos eslabones con dos grados de libertad, en donde la uniones están representadas por pares cinemáticos de revolución (ver Figura 2.7).

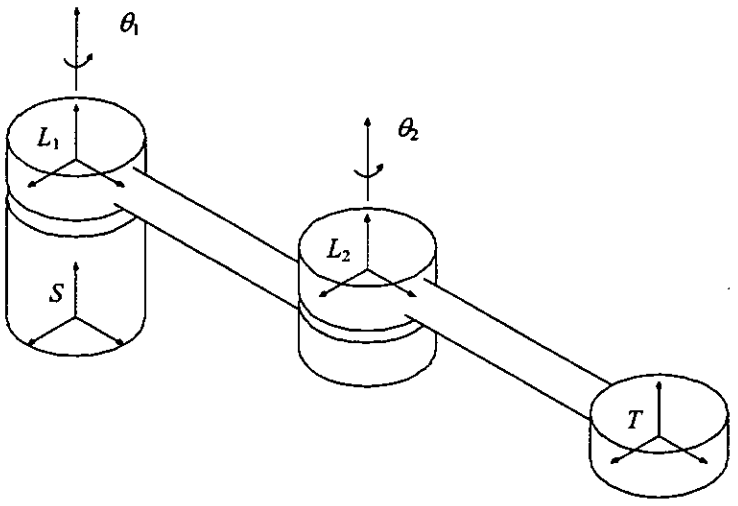

Figura 2.7 Manipulador de dos grados de libertad.

Suponiendo que la primera articulación permanece inmóvil y que la configuración del sistema de referencia de la herramienta es únicamente función de θ_2, entonces puede escribirse

$$g_{st}(\theta_2) = e^{\hat{\xi}_2\theta_2} g_{st}(0),$$

en donde ξ_2 son las coordenadas de giro correspondientes a la rotación en torno al eje de la segunda articulación. Una vez hecho esto, se fija θ_2 y se procede a variar θ_1. Por composición, la configuración del eslabón final está dada por

$$g_{st}(\theta_1,\theta_2) = e^{\hat{\xi}_1\theta_1} g_{st}(\theta_2) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} g_{st}(0),$$

en donde ξ_1 está asociado con la primera articulación. Es importante notar que las coordenadas de giro tanto para ξ_1 como para ξ_2, son obtenidas cuando θ_1 y θ_2 son iguales a cero. La expresión anterior se obtiene moviendo primero el eslabón más distante y después el más cercano, lo cual permite representar el movimiento de las articulaciones como si fueran hechas en torno a ejes constantes. A pesar de esto, el

resultado no depende del orden en que se efectúen los movimientos. Para demostrar esto, ahora se moverá primero θ_1 y después θ_2. De esta manera se tiene en primer lugar

$$g_{st}(\theta_1) = e^{\hat{\xi}_1\theta_1} g_{st}(0).$$

Como este movimiento afecta la posición de ξ_2, la segunda rotación se llevará a cabo en torno de un nuevo eje:

$$\xi_2' = \mathrm{Ad}_{e^{\hat{\xi}_1\theta_1}} \xi_2 .$$

Empleando la propiedad denotada por la expresión (2.35) se tiene que la transformación

$$e^{\hat{\xi}_2'\theta_2} = e^{\hat{\xi}_1\theta_1}\left(e^{\hat{\xi}_2\theta_2}\right)e^{-\hat{\xi}_1\theta_1}$$

describe el movimiento en torno al nuevo eje, por lo que

$$g_{st}(\theta_1,\theta_2) = e^{\hat{\xi}_2'\theta_2} e^{\hat{\xi}_1\theta_1} g_{st}(0) = e^{\hat{\xi}_1\theta_1}\left(e^{\hat{\xi}_2\theta_2}\right)e^{-\hat{\xi}_1\theta_1} e^{\hat{\xi}_1\theta_1} g_{st}(0),$$

obteniéndose finalmente

$$g_{st}(\theta_1,\theta_2) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} g_{st}(0),$$

que es el mismo resultado que se obtuvo al operar los movimientos de manera inversa.

De este modo puede hacerse una generalización para la cinemática directa de un mecanismo de cadena abierta con n grados de libertad, en donde se tienen dos sistemas de referencia, uno acoplado a la base del manipulador (S) y otro acoplado al último eslabón (T). Para esto tiene que definirse primero lo que es la *configuración de referencia* de un manipulador. Ésta corresponde a la configuración del manipulador cuando todas la variables de movimiento valen cero, y está asociada con $g_{st}(0)$, que es la transformación de cuerpo rígido entre el sistema T y el sistema S cuando el mecanismo se halla en esta configuración.

Así pues, bajo la configuración de referencia, se determinan las coordenadas de giro ξ_i para cada una de las articulaciones o elementos de unión, recordando que para un par de revolución

$$\xi_i = \begin{bmatrix} -\omega_i \times q_i \\ \omega_i \end{bmatrix}, \tag{2.70}$$

en donde ω_i es un vector unitario en la dirección del eje de giro, y q_i es un punto cualquiera sobre dicho eje. Y para un par prismático se tiene que

$$\xi_i = \begin{bmatrix} v_i \\ 0 \end{bmatrix},$$

(2.71)

donde v_i es un vector unitario que apunta en la dirección de la traslación.

De este modo, combinando los movimientos individuales de cada articulación se tiene que la cinemática directa de un manipulador de cadena abierta está dada por

$$\boxed{g_{st}(\theta) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} \cdots e^{\hat{\xi}_n\theta_n} g_{st}(0)},$$

(2.72)

expresión que recibe el nombre *producto de exponenciales*. De acuerdo con la notación establecida, los elementos ξ_i deben numerarse secuencialmente desde la base hasta el último eslabón.

Un aspecto muy importante dentro de este enfoque, es la elección de la configuración de referencia, así como del sistema base, ya que mucho depende de esto la simplicidad o complejidad que pueda tener el modelo cinemático. Por ejemplo, una manera de lograr una simplificación, es elegir al sistema base coincidente con el sistema de la herramienta, ya que de esta forma, el último término de la expresión (2.72) se convierte en una matriz identidad. Si esto no es posible, por lo menos hay que tratar de elegir los ejes de movimiento paralelos a los ejes del sistema de referencia inercial.

2.5.2 Jacobiano de un manipulador

Además de conocer la configuración final del manipulador en función de las posiciones relativas entre los eslabones, muchas veces también se desea conocer la velocidad de estos últimos a partir de las velocidades en cada uno de los elementos de unión. De acuerdo con lo que se ha visto hasta el momento, sea $g_{st}(\theta(t))$ la transformación de cuerpo rígido que permite conocer la configuración de la herramienta acoplada al extremo del manipulador. Empleando la expresión (2.53), la velocidad inercial instantánea del último eslabón estará dada por

$$\hat{V}_{st}^{s} = \dot{g}_{st}(\theta) g_{st}^{-1}(\theta).$$

que al aplicarle la regla de la cadena se obtiene

$$\hat{V}_{st}^{s} = \sum_{i=1}^{n} \left(\frac{\partial g_{st}}{\partial \theta} \dot{\theta}_i \right) g_{st}^{-1}(\theta) = \sum_{i=1}^{n} \left(\frac{\partial g_{st}}{\partial \theta} g_{st}^{-1}(\theta) \right) \dot{\theta}_i .$$

(2.73)

De esta expresión puede verse que la velocidad de la herramienta guarda una relación lineal con las velocidades en cada una de las articulaciones. En coordenadas de giro, la ecuación (2.73) tendría la siguiente forma

$$\hat{V}_{st}^{s} = J_{st}^{s}(\theta)\dot{\theta},$$

en donde

$$J_{st}^{s}(\theta) = \left[\left(\frac{\partial g_{st}}{\partial \theta_1} g_{st}^{-1} \right)^{\vee} \quad \cdots \quad \left(\frac{\partial g_{st}}{\partial \theta_n} g_{st}^{-1} \right)^{\vee} \right]. \tag{2.74}$$

La matriz $J_{st}^{s}(\theta) \in \mathbb{R}^{6\times n}$ recibe el nombre de *Jacobiano inercial del manipulador*, que en la configuración denotada por los valores de θ, transforma las velocidades de las articulaciones en la correspondiente velocidad de la herramienta.

Este mismo concepto puede obtenerse a partir del uso del producto de exponenciales. Si la cinemática directa de un manipulador se representa por medio de

$$g_{st}(\theta) = e^{\hat{\xi}_1\theta_1} \cdots e^{\hat{\xi}_n\theta_n} g_{st}(0),$$

en donde ξ_i son las coordenadas de giro para cada articulación, y cuya inversa está dada por

$$g_{st}^{-1}(\theta) = g_{st}^{-1}(0) e^{-\hat{\xi}_n\theta_n} \cdots e^{-\hat{\xi}_1\theta_1}, \tag{2.75}$$

entonces

$$\left(\frac{\partial g_{st}}{\partial \theta_i} \right) g_{st}^{-1} = e^{\hat{\xi}_1\theta_1} \cdots e^{\hat{\xi}_{i-1}\theta_{i-1}} \frac{\partial}{\partial \theta_i} \left(e^{\hat{\xi}_i\theta_i} \right) e^{\hat{\xi}_{i+1}\theta_{i+1}} \cdots e^{\hat{\xi}_n\theta_n} g_{st}(0) g_{st}^{-1},$$

$$\left(\frac{\partial g_{st}}{\partial \theta_i} \right) g_{st}^{-1} = e^{\hat{\xi}_1\theta_1} \cdots e^{\hat{\xi}_{i-1}\theta_{i-1}} \left(\hat{\xi}_i \right) e^{\hat{\xi}_i\theta_i} \cdots e^{\hat{\xi}_n\theta_n} g_{st}(0) g_{st}^{-1},$$

$$\left(\frac{\partial g_{st}}{\partial \theta_i} \right) g_{st}^{-1} = e^{\hat{\xi}_1\theta_1} \cdots e^{\hat{\xi}_{i-1}\theta_{i-1}} \left(\hat{\xi}_i \right) e^{-\hat{\xi}_{i-1}\theta_{i-1}} \cdots e^{-\hat{\xi}_1\theta_1},$$

que convirtiendo a coordenadas de giro se tiene

$$\left(\frac{\partial g_{st}}{\partial \theta_i} g_{st}^{-1}\right)^{\vee} = \mathrm{Ad}_{\left(e^{\xi_1 \theta_1} \ldots e^{\xi_{i-1} \theta_{i-1}}\right)} \xi_i .$$

(2.76)

Por lo tanto el Jacobiano inercial del manipulador estará dado por

$$\boxed{J_{st}^{s}(\theta) = \left[\xi_1 \quad \xi_2' \quad \cdots \quad \xi_n'\right]},$$

(2.77)

$$\boxed{\xi_i' = \mathrm{Ad}_{\left(e^{\xi_1 \theta_1} \ldots e^{\xi_{i-1} \theta_{i-1}}\right)} \xi_i}.$$

(2.78)

De la expresión (2.78) puede verse que la *i*-ésima columna del Jacobiano depende únicamente de $(\theta_1,\ldots,\theta_{i-1})$, es decir, la aportación de la velocidad de la *i*-ésima articulación a la velocidad de la herramienta es independiente de la configuración de eslabones subsecuentes en la cadena. Además, las columnas también corresponden a las coordenadas de giro de cada una las articulaciones del manipulador, con la particularidad de que éstas se hayan calculadas no para la configuración de referencia, sino para la configuración actual. Esto implica que la construcción del Jacobiano inercial del manipulador puede realizarse de una manera casi directa.

Como ha ocurrido con otros conceptos, aquí también es posible calcular un *Jacobiano local del manipulador*, cuya definición está dada por

$$V_{st}^{b} = J_{st}^{b}(\theta)\dot{\theta} .$$

A partir de la definición de la velocidad local dada en la expresión (2.56), puede llevarse a cabo un procedimiento similar al mostrado anteriormente, obteniendo

$$\boxed{J_{st}^{b}(\theta) = \left[\xi_1^{\circ} \quad \cdots \quad \xi_{n-1}^{\circ} \quad \xi_n^{\circ}\right]},$$

(2.79)

$$\boxed{\xi_i^{\circ} = \mathrm{Ad}_{\left(e^{\xi_i \theta_i} \ldots e^{\xi_n \theta_n} g_{st}(0)\right)} \xi_i}.$$

(2.80)

Las columnas del Jacobiano corresponden a las coordenadas de giro de las articulaciones escritas con respecto al sistema de referencia de la herramienta. Como era de esperarse, también puede establecerse una relación entre el Jacobiano inercial y el local, la cual se realiza mediante una transformación adjunta:

$$J_{st}^{s}(\theta) = \mathrm{Ad}_{g_{st}(\theta)} J_{st}^{b}(\theta) .$$

(2.81)

Ambos Jacobianos puede emplearse para calcular la velocidad de un punto acoplado al eslabón final. Si q^b representa a un punto de la herramienta referido al

sistema de ésta, entonces la velocidad de dicho punto expresada también en coordenadas de la herramienta estará dada por

$$v_q^b = \hat{V}_{st}^b q^b = \left(J_{st}^b(\theta)\dot{\theta}\right)^{\wedge} q^b .$$

De modo similar, si ahora se representa dicho punto en coordenadas inerciales, se obtendrá entonces

$$v_q^s = \hat{V}_{st}^s q^s = \left(J_{st}^s(\theta)\dot{\theta}\right)^{\wedge} q^s .$$

Si se desea conocer la velocidad del origen de la herramienta, habrá que tener presente que $q^b = 0$, sin embargo $q^s = g_{st}(\theta)q^b = p(\theta)$, que es la componente traslacional de la transformación que describe la cinemática directa del manipulador. Así pues, utilizando la representación homogénea se tiene

$$v_q^s = \begin{bmatrix} \dot{p}(\theta) \\ 0 \end{bmatrix} = R_{st}\hat{V}_{st}^b \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \hat{V}_{st}^s \begin{bmatrix} p(\theta) \\ 1 \end{bmatrix}.$$

A pesar de que, por los alcances de este trabajo, solamente se hace mención de su utilidad para el cálculo de las velocidades de los elementos que componen a un mecanismo, el Jacobiano tiene otras aplicaciones muy importantes relacionadas con el control de los manipuladores, ya que por ejemplo se emplea para:

1. El análisis dinámico, es decir, para la obtención de los torques que tienen que aplicarse en las articulaciones para lograr una determinada fuerza de sujeción, o un determinado desplazamiento de los eslabones tomando en cuenta las fuerzas que actúan sobre ellos.
2. La detección de configuraciones singulares del mecanismo, en las cuales el manipulador no es capaz de realizar movimientos o aplicar fuerzas en ciertas direcciones.

2.5.3 Comparación del producto de exponenciales con otros métodos para el cálculo de la cinemática de un manipulador

Hasta el momento se ha visto la utilidad del producto de exponenciales como un medio efectivo para realizar la descripción cinemática de los mecanismos, sin embargo, éste no es el procedimiento más empleado para estos fines, de hecho existen otros métodos que se utilizan con mucho más frecuencia; uno de éstos es el de los parámetros de Denavit-Hartenberg (D-H). Como en el caso del producto de exponenciales, el método de D-H también hace uso de transformaciones homogéneas que tienen una representación matricial.

La obtención de los parámetros para la construcción de las matrices de transformación se hace siguiendo un conjunto de reglas, las cuales especifican la posición y la orientación de los sistemas de referencia que van acoplados a cada

uno de los eslabones del mecanismo. Como se indica en la ecuación (2.69), la cinemática del mecanismo queda descrita mediante el producto de las matrices de transformación ente eslabones:

$$g_{st}(\theta) = g_{sl_1}(\theta_1) g_{l_1 l_2}(\theta_2) \cdots g_{l_{n-1} l_n}(\theta_n) g_{l_n t} .$$

Cada una de las transformaciones tiene la forma

$$g_{l_{i-1} l_i} = \begin{bmatrix} \cos\phi_i & -\operatorname{sen}\phi_i \cos\alpha_i & \operatorname{sen}\phi_i \operatorname{sen}\alpha_i & a_i \cos\phi_i \\ \operatorname{sen}\phi_i & \cos\phi \cos\alpha_i & -\cos\phi_i \operatorname{sen}\alpha_i & a_i \operatorname{sen}\phi_i \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

en donde los escalares a_i, α_i, d_i, y ϕ_i son los parámetros para el i-ésimo eslabón. El modo como se escogen los sistemas de referencia para cada eslabón y la interpretación geométrica de los parámetros se explican de forma más detallada en el Apéndice A.

Como puede apreciarse, los parámetros de D-H son sólo cuatro en comparación con los seis (las coordenadas de giro) que se requieren para la descripción cinemática por medio del producto de exponenciales. Esto se debe a que ocurren una serie de cancelaciones dada la forma como se eligen los sistemas de referencia. A pesar de esto, puede establecerse una relación entre las coordenadas de giro y los parámetros de D-H, que no es del todo directa, ya que a diferencia de las coordenadas de giro, que están todas referenciadas al sistema coordenado inercial, los parámetros de D-H se obtienen estableciendo relaciones entre eslabones adyacentes. Para establecer la relación entre los parámetros de los dos métodos, primero se obtienen las coordenadas de giro de cada eslabón con respecto al sistema coordenado del eslabón anterior; después se premultiplica por la transformación adjunta correspondiente a la matriz que transforma las coordenadas relativas al eslabón anterior al sistema inercial, obteniendo esta última bajo la configuración de referencia:

$$\xi_i = \mathrm{Ad}_{g_{sl_{i-1}}(0)} \xi_{i-1,i} .$$

(2.82)

En esta expresión, ξ_i se halla referenciada al sistema inercial. Desde luego, si se pretende pasar de los parámetros de D-H al producto de exponenciales, es mucho más sencillo determinar desde el principio las coordenadas de giro correspondientes a cada tipo de articulación.

Haciendo una comparación entre ambos métodos, el producto de exponenciales resulta más conveniente principalmente por dos razones:

1. A diferencia del método de D-H, el uso del producto de exponenciales requiere únicamente de dos sistemas de referencia, el de la base (o inercial) y el de la herramienta.
2. La obtención de los parámetros es más sencilla dado que su interpretación geométrica es más directa.

La dificultad que implica el uso de seis parámetros en lugar de cuatro es bastante relativa, ya que tres de esos seis números corresponden a las componentes de un vector libre que ilustra la dirección del eje de desplazamiento lineal o rotacional según el caso; y los otros tres, corresponden al resultado del producto vectorial negativo entre este último vector y las coordenadas de un punto sobre dicho eje.

CAPÍTULO 3

Análisis cinemático del manipulador antropomórfico

Algo que se espera de un manipulador, sin importar su tipo, es que se puedan controlar sus movimientos de la manera más precisa posible. Para esto es necesario conocer la configuración de cada uno de los eslabones de la cadena cinemática en función de la variación de sus posiciones relativas. Aquello que permite establecer la relación entre las variables de movimiento y la configuración final del manipulador, es el análisis cinemático, del cual se describieron sus fundamentos básicos en el capítulo anterior. De esta manera, con base en las herramientas analíticas vistas y dadas las ventajas que ofrece en comparación con otros métodos, el producto de exponenciales es el método que se eligió para la descripción de la cinemática directa del Manipulador Antropomórfico Teleoperado (MAT).

3.1 Determinación de los parámetros cinemáticos del MAT

Como se ha mencionado en varias ocasiones, una de las principales virtudes del producto de exponenciales consiste en que la interpretación geométrica de los parámetros de los que hace uso es bastante directa y accesible, sin embargo, antes de comenzar con la determinación de los parámetros que sirven para la descripción cinemática directa del manipulador, es necesario realizar antes algunas precisiones en cuanto a sus características estructurales.

3.1.1 Características estructurales del MAT

Al hablar de parámetros geométricos, necesariamente se tiene que hacer referencia a las dimensiones de los elementos que conforman a un manipulador, o en dado caso, a las proporciones que puedan existir entre dichos elementos. De estas medidas o relaciones dependen muchas de las variables que están involucradas con algunos aspectos del movimiento, como por ejemplo: el área de trabajo, la velocidad de los eslabones, etc..

En el primer capítulo se habló brevemente de las características físicas que se establecieron para el análisis y la construcción del manipulador. De éstas, una de las más significativas era que se trataba de una mano derecha caracterizada por la ausencia del dedo meñique, de modo que el manipulador únicamente contaría con cuatro dedos: tres que corresponden a los dedos índice, medio y anular; y uno más

que tiene las funciones de un dedo pulgar. Tomando en cuenta esto último es que se llevará a cabo el modelado cinemático, es decir, los grados de libertad del dedo meñique son los únicos que se eliminan de entrada. Esta aclaración se hace por el hecho de que hasta el momento no se ha tomado una decisión en cuanto al número definitivo de grados de libertad con los que contará el mecanismo, de modo que el análisis que aquí se haga, tomará en cuenta a la mayoría de éstos.

Otro aspecto estructural que se tomó en cuenta, tiene que ver con el hecho de que los metacarpos de los dedos índice, medio y anular, pueden considerarse agrupados en una sola estructura que viene a ser la palma de la mano. De este modo cada uno de los dedos puede manejarse como un manipulador separado, con la particularidad de que todos comparten el primer eslabón (ver Figura 3.1).

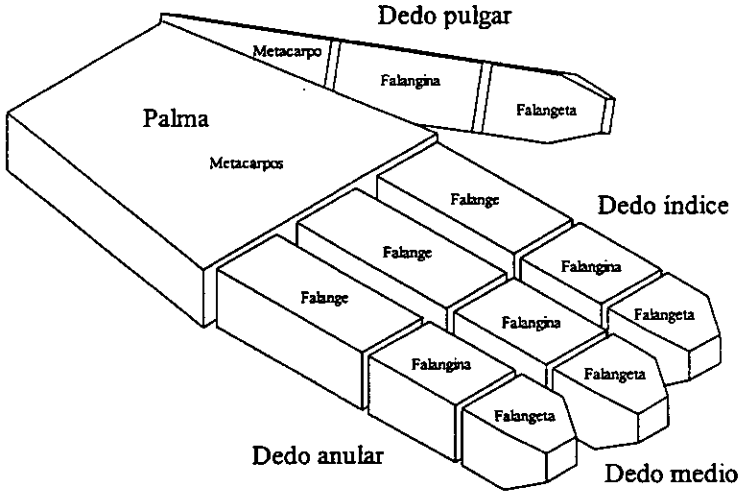

Figura 3.1 Esquema del manipulador antropomórfico.

Además de la anterior, existe otra consideración que tiene que ver con la independencia de los grados de libertad; obviamente si el movimiento de una articulación está relacionado con el de otra, no se puede hablar de varios grados de libertad, sino de uno solo. La consideración a la que se hace referencia, va únicamente para los dedos índice, medio y anular. Si se analiza el movimiento de éstos en una mano humana, puede notarse que normalmente el movimiento de la falangeta está asociado con el de la falangina, por lo que es posible eliminar como tal, el grado de libertad de esta última y dejarlo en función del de la primera. Así pues, de acuerdo con un estudio funcional de los dedos[4], puede decirse que el ángulo relativo entre la falangina y la falange, es aproximadamente el mismo que existe entre la falangeta y la falangina.

Del mismo estudio funcional, también se pudo apreciar que la característica de oponibilidad del dedo pulgar se debe a una orientación muy especial de sus ejes

[4] Tema tratado en la parte del proyecto encargada de la construcción del manipulador.

de movimiento. Más adelante se hará una explicación más detallada del cómo se orientan dichos ejes.

Las medidas más importantes para el análisis cinemático son las longitudes de cada uno de los elementos que integran al manipulador. En este caso, al tratarse de un manipulador antropomórfico, las medidas de los elementos del robot corresponden a las de la palma y los dedos de una mano humana. Debido a que existe una gran variación en cuanto al tamaño que una mano puede tener, se decidió que el análisis se haría con base en las relaciones geométricas que existen entre los elementos. De esta manera fue que se tomó como medida unitaria, la longitud de la falange del dedo índice (ver Tabla 3.1), de modo que si se desea dar un tamaño determinado al manipulador, únicamente se tiene que aplicar el factor de escala apropiado. Además cabe señalar que estas relaciones corresponden de manera aproximada a las que posee la media de la especie[5].

Tabla 3.1 Relaciones geométricas entre las longitudes de los dedos.

Dedo	Metacarpo	Falange	Falangina	Falangeta
Indice	1.7070	1.0000	0.5855	0.4879
Medio	1.7325	1.0980	0.6830	0.5120
Anular	1.7070	1.0700	0.6343	0.5120
Pulgar	0.9246	-	0.7628	0.6010

Aunque para los objetivos de este trabajo no tienen mayor repercusión, también se incluyen las medidas del ancho y la altura de los dedos, las cuales se proporcionan considerando las mismas condiciones que se establecieron para la Tabla 3.1 (ver Tabla 3.2). Si por otro lado se tratara de un análisis de interferencia entre dedos, entonces en ese caso habría que considerar todas y cada una de las medidas del manipulador.

Tabla 3.2 Relaciones geométricas del ancho y la altura de los dedos.

Ancho de los dedos	Altura de los dedos
0.4392	0.3659

3.1.2 Configuración de referencia.

Como se estableció en la sección 2.5.1, antes de emplear el método del producto de exponenciales, se tiene que definir la configuración de referencia del manipulador. En ésta, es donde prácticamente se establecen todas las condiciones físicas y de movimiento. Recordando un poco, esta configuración corresponde a aquélla en la que todas las variables de movimiento valen cero, lo cual implica que hay que definir todos los vectores que indique la orientación de los ejes de desplazamiento, ya sea

[5] Tema tratado la parte del proyecto encargada de la construcción del manipulador.

rotacional o traslacional, así como su localización en el espacio. Para el caso que se trata en este trabajo, se tiene que todos los pares cinemáticos que intervienen en el manipulador antropomórfico son de revolución, es decir, únicamente existen desplazamientos angulares entre los elementos que lo conforman.

Una vez que se ha hecho esta precisión, es necesario seguir los tres pasos siguientes:

1. Definir la localización de los sistemas de referencia tanto inercial, como de la herramienta.
2. Determinar los vectores que indiquen el eje de rotación y la dirección del movimiento.
3. Establecer puntos referidos al sistema inercial, que se encuentren sobre cada uno de los ejes de las articulaciones.

En el capítulo anterior se había hablado de que únicamente se requería de dos sistemas de referencia para la definición cinemática de un mecanismo de cadena abierta: el inercial y el de la herramienta; sin embargo existen algunos factores que hacen que esto no sea posible. El hecho de que cada dedo pueda considerarse como un manipulador separado, implica que por lo menos tiene que haber un sistema de referencia (el de la herramienta) por cada uno de ellos. Esto bastaría si únicamente se quisiera conocer las posiciones finales de los extremos de los dedos, no obstante, dado que uno de los objetivos de este trabajo es crear un sistema de despliegue visual, necesariamente tiene que mostrarse la posición de todos los eslabones que componen al manipulador. Estos eslabones tienen que representarse gráficamente mediante algún tipo de objeto, cuyo trazado es mucho más sencillo si se utilizan coordenadas relativas en lugar de coordenadas absolutas. Por esta razón resulta conveniente colocar sistemas de referencia en cada uno de los eslabones de los dedos. Cabe hacer la aclaración de que esto se hace por razones de representación gráfica, y que no afecta la manera como se calcula la cinemática directa del manipulador.

Comenzando con los pasos para el establecimiento de los parámetros cinemáticos, primero se tiene que definir la posición y la orientación del sistema inercial de referencia. Como se había explicado anteriormente, de la elección de éste depende una buena parte de la complejidad del modelo matemático. Se mencionó que había que tratar en la medida de lo posible, de que coincidiera el sistema inercial con el de la herramienta. En este caso, al tratarse de "varios manipuladores", la localización del sistema inercial no ofrece mayor ventaja en la simplificación del análisis. También se había mencionado que si no era posible hacer coincidir los sistemas de referencia, por lo menos debía intentarse que los ejes del sistema inercial fueran paralelos a la mayor parte de los ejes de movimiento. La ventaja de esto radica en que únicamente se tiene una componente en el vector que funciona como eje de la articulación. Para esta ocasión en particular, y tomando en cuenta la Figura 3.1, se puede ver que una localización y una orientación del sistema inercial, como los mostrados en la Figura 3.2, son los más convenientes, ya que, aunque todavía no se especifica claramente, existe la noción de que en esta posición, la mayoría de los ejes de rotación de los elementos son paralelos, a excepción de los del dedo pulgar, dadas sus características especiales.

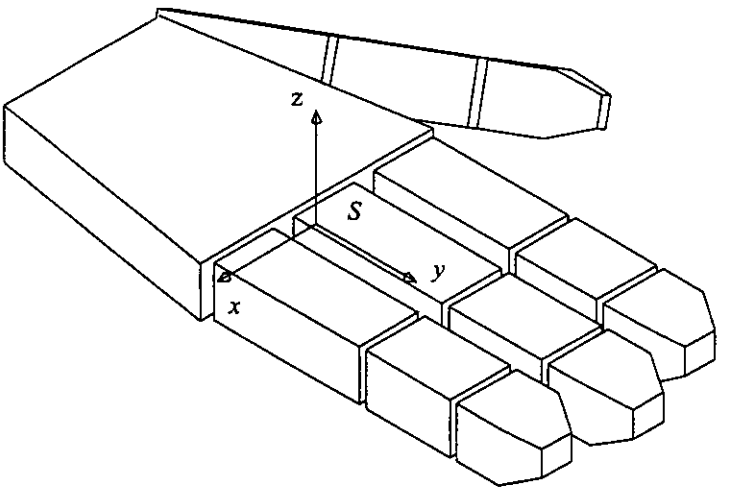

Figura 3.2 Configuración de referencia y localización del sistema inercial.

La localización del sistema inercial mostrado en la Figura 3.2 corresponde a la posición del centro del extremo proximal de la falange del dedo medio. Aunque, como se vio, la ubicación del sistema inercial no es un factor crítico en el desarrollo de la cinemática directa, se eligió de esta manera por razones de representación gráfica, ya que en este lugar, el origen se halla centrado con respecto al manipulador, por lo que es más sencillo controlar el trazado de la imagen ante los cambios de vista.

3.1.3 Parámetros cinemáticos de la palma y los dedos índice, medio y anular

Una vez establecida la posición del sistema inercial, se procede a la identificación de los sistemas locales de coordenadas de cada uno de los elementos que conforman a la palma y a los dedos índice, medio y anular. Del mismo modo, se establecen los vectores de los ejes de las articulaciones, así como los puntos en el espacio que se hallen sobre las líneas de dichos ejes. Para evitar una aglomeración de información sobre los esquemas, lo que se hace es mostrar únicamente al miembro de interés, pero guardando la misma disposición con respecto al sistema inercial que se estableció en la Figura 3.2.

En primer lugar, se tiene la estructura de la palma, de la cual se puede decir que posee en su extremo proximal un par cinemático esférico. Este tipo de unión permite realizar rotaciones arbitrarias, y suele implementarse tanto mecánica como matemáticamente, como una combinación de tres pares de revolución, de tal forma que sus ejes de rotación se intersecan en un punto. En este caso se colocan los tres ejes de movimiento paralelos a los ejes del sistema inercial de coordenadas (ver Figura 3.3). La dirección positiva está representada por los vectores ω_i que, como puede apreciarse, tienen la misma dirección positiva que la de los ejes del sistema

inercial. Por lo que toca a la elección de un punto que se halle sobre cada eje de rotación, en lugar de escoger tres diferentes puntos, se escoge uno solo, el cual está dado por la intersección de los tres ejes. Finalmente, para el caso del sistema local de referencia, éste se coloca en la intersección de los tres ejes de movimiento.

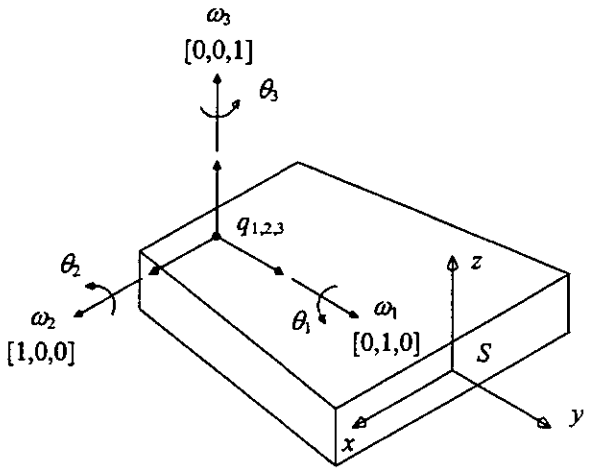

Figura 3.3 Configuración de referencia y parámetros cinemáticos de la palma.

Cuando se habla de pares cinemáticos esféricos, los movimientos que se realizan en torno a los diferentes ejes suelen recibir nombres especiales, como por ejemplo, tomando como base la Figura 3.3, la rotación en torno al eje y recibe el nombre de *balanceo* (*rolling*); en torno al eje x toma el nombre de *inclinación* (*pitch*); y por último, una rotación en torno al eje z se le conoce como *guiñada* (*yaw*).

En este momento vale la pena recordar que cuando se llevan a cabo rotaciones en torno a los ejes de movimiento, el efecto final no depende del orden en que se apliquen los movimientos en las articulaciones, sin embargo, hay que tener presente que si un eslabón se mueve, inmediatamente se altera la posición de los eslabones subsecuentes en la cadena cinemática. Se hace esta aclaración por el hecho de que, como se mencionó anteriormente, al manipulador antropomórfico se le puede considerar como una serie de pequeños manipuladores que comparten el primer eslabón, de modo que cualquier movimiento de la palma afecta la posición de todos los dedos.

Prosiguiendo con la identificación de los parámetros para el modelado cinemático, toca el turno al dedo índice, cuya configuración de referencia y parámetros se muestran en la Figura 3.4. En esta ocasión, al igual que con la palma, los ejes de movimiento son paralelos y tienen la misma dirección positiva que los ejes del sistema inercial. Por lo que toca a los puntos que tienen que colocarse en algún lugar de los ejes de rotación, se decidió ubicarlos en el centro de la sección del extremo proximal de cada uno de los elementos, lugar en el que también se colocaron los sistemas locales de referencia. Para el caso de la falange, cabe

señalar que este lugar coincide con el punto de intersección de los dos ejes de rotación.

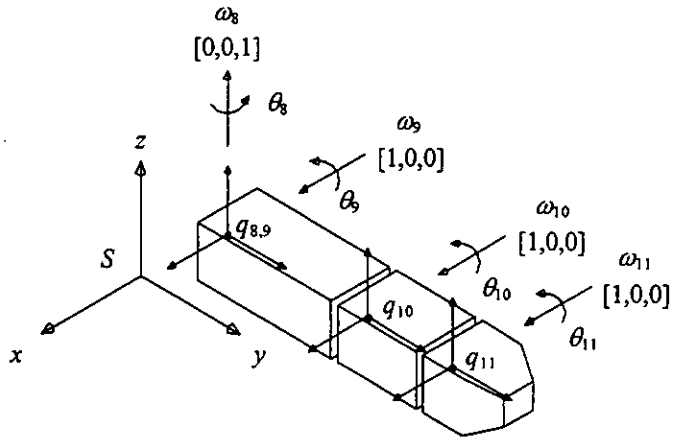

Figura 3.4 Configuración de referencia y parámetros cinemáticos del dedo índice.

En lo referente a los dedos medio y anular, puede decirse que la orientación y localización de los vectores de los ejes; de los sistemas de referencia locales; y de los puntos base[6], se dan exactamente de la misma manera como ocurre con el dedo índice (ver Figuras 3.5 y 3.6), la única diferencia que existe de un caso con respecto a los otros dos, es que varía la localización general del miembro.

El hecho de que se halla considerado dentro de las restricciones planteadas al comienzo del capítulo, que los dedos índice, medio y anular, poseen únicamente tres grados de libertad en lugar de cuatro, no quita que se tenga que especificar la ubicación de todos y cada uno de los ejes de rotación que intervienen en el manipulador.

[6] Entendiendo por punto base, aquel punto que se localiza en algún lugar de la línea del eje de movimiento.

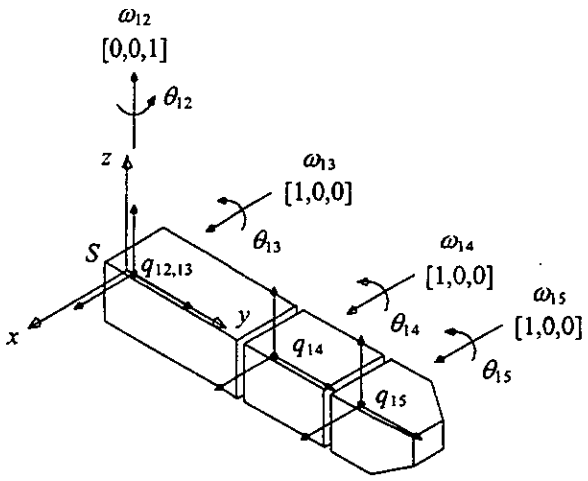

Figura 3.5 Configuración de referencia y parámetros cinemáticos del dedo medio.

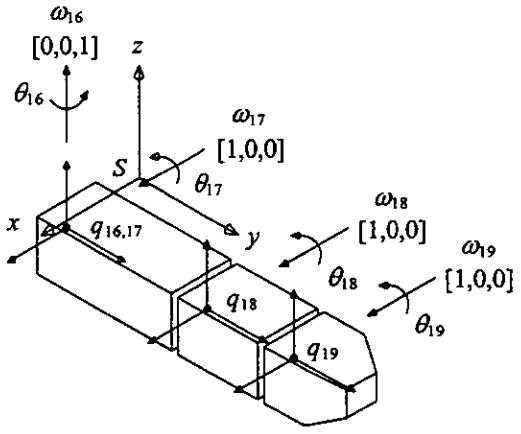

Figura 3.6 Configuración de referencia y parámetros cinemáticos del dedo anular.

De esta manera, lo que hace falta es detallar exactamente las posiciones de los puntos base, así como la de los sistemas de referencia locales, que de hecho, dada la forma como se colocaron, resultan ser prácticamente la mismas. En la Tabla 3.3 se especifica tanto la ubicación de los sistemas coordenados locales, como los puntos base q_i que corresponden a esa localización. La coordenadas están dadas tomando en cuenta las mismas condiciones que se establecieron para las Tablas 3.1

y 3.2, es decir, la unidad de medida corresponde a la longitud de la falange del dedo índice.

Tabla 3.3 Coordenadas de los sistemas locales de referencia (palma, dedo índice, dedo medio y dedo anular).

	q_i	x_i	y_i	z_i
Palma	q_1,q_2,q_3	0	-1.7325	0
Índice				
Falange	q_8,q_9	-0.5867	-0.0255	0
Falangina	q_{10}	-0.5867	0.9745	0
Falangeta	q_{11}	-0.5867	1.5600	0
Medio				
Falange	q_{12},q_{13}	0	0	0
Falangina	q_{14}	0	1.0980	0
Falangeta	q_{15}	0	1.7810	0
Anular				
Falange	q_{16},q_{17}	0.5867	-0.0255	0
Falangina	q_{18}	0.5867	1.0445	0
Falangeta	q_{19}	0.5867	1.6788	0

Así como las posiciones de los puntos base y de los sistemas coordenados locales son importantes, también lo son los límites de movimiento de cada uno de los miembros elementos del manipulador. A pesar de que en la cinemática directa esta condición no es tan crítica, en la cinemática inversa tiene un impacto bastante considerable, ya que puede ayudar a simplificar el número de soluciones posibles. De esta manera es que física y matemáticamente se establece que el movimiento de los dedos sea hacia el "interior" de la palma. Esto se pone de manifiesto en la Tabla 3.4.

Tabla 3.4 Límites superiores e inferiores para el movimiento de los elementos que conforman al manipulador.

	ω_i	Lím. Inferior	Lím. superior
Palma	ω_1	-180	180
	ω_2	-90	90
	ω_3	-90	90
Dedos			
Falange	ω_8, ω_{12}, ω_{16}	-20	20
	ω_9, ω_{13}, ω_{17}	-105	20
Falangina	ω_{10}, ω_{14}, ω_{18}	-90	0
Falangeta	ω_{11}, ω_{15}, ω_{19}	-90	0

3.1.4 Parámetros cinemáticos del dedo pulgar

El dedo pulgar tiene la característica de ser oponible, lo cual se debe a su posición y orientación especiales. Por esta razón, lo primero que tiene que revisarse antes de comenzar con el análisis de sus parámetros, es la orientación de los sistemas locales de referencia de los elementos que lo integran.

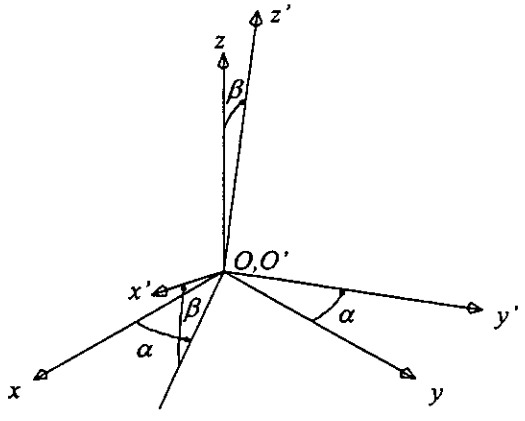

Figura 3.7 Orientación de los sistemas de referencia del dedo pulgar.

En la Figura 3.7 puede apreciarse un sistema de referencia O que es paralelo al sistema de referencia inercial. Tomando como base este sistema, es posible realizar las rotaciones correspondientes, que hagan que el pulgar tome su orientación final (sistema O'). La orientación puede sufrir variaciones de acuerdo con el diseño del mecanismo que más convenga, sin embargo, debe mantenerse una restricción que se deriva del estudio funcional de la mano: el eje y' debe estar contenido en el plano x-y del sistema inercial.

Las componentes de los vectores que representan a los ejes de los sistemas locales del pulgar pueden calcularse de la siguiente manera:

$$x' = \begin{bmatrix} \cos\alpha\cos\beta & \operatorname{sen}\alpha\cos\beta & \operatorname{sen}\beta \end{bmatrix}^T, \tag{3.1}$$

$$y' = \begin{bmatrix} -\operatorname{sen}\alpha & \cos\alpha & 0 \end{bmatrix}^T, \tag{3.2}$$

$$z' = \begin{bmatrix} -\cos\alpha\operatorname{sen}\beta & -\operatorname{sen}\alpha\operatorname{sen}\beta & \cos\beta \end{bmatrix}^T. \tag{3.3}$$

Con estas expresiones automáticamente se obtienen los vectores de los ejes de rotación en las articulaciones, ya que si se recuerda de los otros dedos, los ejes de movimiento resultaban ser paralelos a los ejes de los sistemas coordenados locales.

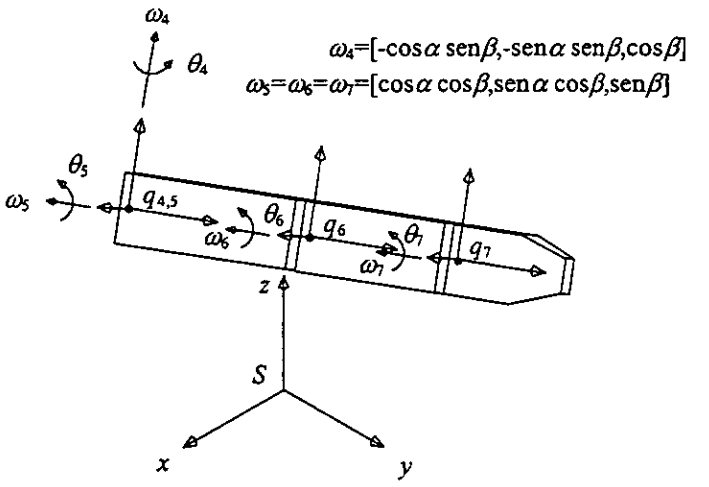

Figura 3.8 Configuración de referencia y parámetros cinemáticos del dedo pulgar.

En la Figura 3.8 se muestra la orientación de los ejes de movimiento, así como la de los sistemas locales. Haciendo referencia a lo mencionado en el párrafo anterior, el vector ω_4 tiene exactamente las mismas componentes que el vector z', que están denotadas por la expresión (3.3), mientras que los vectores ω_5, ω_6, y ω_7 tienen las componentes del vector x' de la expresión (3.1).

Por lo que toca a los puntos base, el único que se puede especificar de manera precisa, es el correspondiente a la unión del pulgar con la palma, ya que los restantes, que también se colocan en el centro del extremo proximal del elemento, dependen de la orientación que se le dé al dedo pulgar. A pesar de esto, la localización de estos puntos puede proporcionarse como una función de los ángulos que intervienen en la orientación de los sistemas locales del pulgar. En la Tabla 3.5, se establece la ubicación de dichos puntos:

Tabla 3.5 Coordenadas de los sistemas locales de referencia (dedo pulgar).

	q_i	x_i	y_i	z_i
Metacarpo	q_4,q_5	-0.5867	-1.7325	0
Falangina	q_6	-0.5867-(l_1)sα	-1.7325+(l_1)cα	0
Falangeta	q_7	-0.5867-(l_1+l_2)sα	-1.7325+(l_1+l_2)cα	0

De la tabla, l_1 y l_2 representan respectivamente las longitudes del metacarpo y de la falangina (ver Tabla 3.1) y, cα y sα corresponden a la notación abreviada de cos(α) y sen(α) también de manera respectiva.

Del mismo modo que para la palma y los dedos índice, medio y anular, para el pulgar también se establecen los límites para el movimientos de cada uno de los eslabones (ver Tabla 3.6).

Tabla 3.6 Límites superiores e inferiores para el movimiento de los elementos del pulgar.

	ω_i	Lím. Inferior	Lím. Superior
Pulgar			
Metacarpo	ω_4	0	90
	ω_5	-90	20
Falangina	ω_6	-90	0
Falangeta	ω_7	-90	0

3.2 Cinemática directa de los dedos índice, medio y anular

En el capítulo anterior se vio que una de las maneras como se llevaba a cabo la descripción del comportamiento cinemático de un mecanismo de cadena abierta, era mediante el producto de exponenciales:

$$g_{st}(\theta) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} \cdots e^{\hat{\xi}_n\theta_n} g_{st}(0).$$

En esta expresión, cada uno de los términos es una matriz de transformación que permite representar el efecto que tiene cada tipo de articulación sobre el movimiento de toda la cadena cinemática. En este caso, dadas las características del manipulador antropomórfico, únicamente se hará uso de aquellas matrices que describan el efecto de pares cinemáticos de revolución.

También se habló de que un par cinemático podía parametrizarse a través de las llamadas coordenadas de giro: ξ. De la expresión (2.70) puede verse que para el caso de un par cinemático de revolución, las coordenadas de giro tenían la forma siguiente:

$$\xi = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} -\omega \times q \\ \omega \end{bmatrix},$$

en donde ω es un vector unitario en la dirección del eje de giro, y q es un punto cualquiera sobre dicho eje. No está de más decir que tanto ω como q se obtienen cuando el manipulador se halla en su configuración de referencia.

De esta manera las matrices de transformación, que no son más que el exponencial matricial de las coordenadas de giro transformadas multiplicadas por el ángulo de giro, pueden calcularse mediante las expresiones (2.36) y (2.40). Al tratarse de un movimiento que se efectúa exclusivamente alrededor de un eje, puede hacerse una serie de simplificaciones, de modo que finalmente se obtiene

$$e^{\hat{\xi}\theta} = \begin{bmatrix} e^{\hat{\omega}\theta} & \left(I - e^{\hat{\omega}\theta}\right)(\omega \times v) \\ 0 & 1 \end{bmatrix},$$

(3.4)

en donde el término $e^{\hat{\omega}\theta}$ representa la componente puramente rotacional del desplazamiento, que puede calcularse utilizando la expresión (2.14).

Con estas expresiones y una vez obtenidos todos los parámetros, que básicamente son los vectores de los ejes de rotación y los puntos base, es posible generar el modelo que permite la descripción cinemática del manipulador.

3.2.1 Movimiento en torno a ejes principales

En los apartados anteriores se pudo ver que muchos de los ejes, en torno a los cuales se lleva a cabo el movimiento de los eslabones, son paralelos a los ejes del sistema inercial. Por esta razón resulta conveniente establecer expresiones generales para las matrices de transformación que describan precisamente esta clase de movimientos. Como el sistema inercial que se eligió para el manipulador se trata de un sistema coordenado cartesiano derecho, y las direcciones de sus ejes están dadas por los vectores unitarios **i**:[1,0,0], **j**:[0,1,0] y **k**:[0,0,1], entonces serán éstos los vectores que se tomarán como base para el cálculo de las matrices de transformación.

Sea ω = **i**:[1,0,0] el vector del eje en torno al cual se llevará a cabo el movimiento, y q:[x, y, z] un punto sobre el eje. En primer lugar es necesario determinar las coordenadas de giro correspondientes, para cuya obtención se tiene

$$v = -\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T \times \begin{bmatrix} x & y & z \end{bmatrix}^T = \begin{bmatrix} 0 & z & -y \end{bmatrix}^T,$$

por lo que

$$\xi = \begin{bmatrix} 0 & z & -y & 1 & 0 & 0 \end{bmatrix}^T.$$

(3.5)

Con estas coordenadas y tomando como guía a la expresión (2.14), se calcula el término $e^{\hat{\omega}\theta}$. Definiendo $c\theta = \cos(\theta)$, $s\theta = \text{sen}(\theta)$ y $v\theta = (1-\cos(\theta))$ se tiene

$$e^{\hat{\omega}\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta & -s\theta \\ 0 & s\theta & c\theta \end{bmatrix}.$$

(3.6)

Realizando las operaciones indicadas por la ecuación (3.4), finalmente se obtiene

$$e^{\hat{\xi}\theta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta & -s\theta & y\cdot v\theta + z\cdot s\theta \\ 0 & s\theta & c\theta & -y\cdot s\theta + z\cdot v\theta \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \omega = \mathbf{i}. \tag{3.7}$$

Siguiendo el mismo procedimiento, se tiene que cuando ω = **j**:[0,1,0], las coordenadas de giro tienen la siguiente forma:

$$\xi = \begin{bmatrix} -z & 0 & x & 0 & 1 & 0 \end{bmatrix}^T, \tag{3.8}$$

mientras que la submatriz de rotación queda definida por

$$e^{\hat{\omega}\theta} = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix}, \tag{3.9}$$

de modo que al final se tiene

$$e^{\hat{\xi}\theta} = \begin{bmatrix} c\theta & 0 & s\theta & x\cdot v\theta - z\cdot s\theta \\ 0 & 1 & 0 & 0 \\ -s\theta & 0 & c\theta & x\cdot s\theta + z\cdot v\theta \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \omega = \mathbf{j}. \tag{3.10}$$

Cuando $\omega = \mathbf{k}$:[0,0,1], las coordenadas de giro están dadas por

$$\xi = \begin{bmatrix} y & -x & 0 & 0 & 0 & 1 \end{bmatrix}^T, \tag{3.11}$$

la submatriz de rotación por

$$e^{\hat{\omega}\theta} = \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.12}$$

y la matriz de transformación por

$$e^{\hat{\xi}\theta} = \begin{bmatrix} c\theta & -s\theta & 0 & x\cdot v\theta + y\cdot s\theta \\ s\theta & c\theta & 0 & -x\cdot s\theta + y\cdot v\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \omega = \mathbf{k}.$$

(3.13)

3.2.2 Expresiones para la cinemática directa de los dedos índice, medio y anular

Una vez obtenidas las expresiones generales para las matrices de transformación, es posible "armar" el modelo cinemático para cada uno de los dedos índice, medio y anular. Éstos, como se dijo con anterioridad, pueden considerarse como manipuladores separados que comparten el primer eslabón. Por esta razón, primero es necesario establecer las expresiones para el movimiento de la palma.

Respetando los datos contenidos tanto en la Figura 3.3 como en la Tabla 3.3 y haciendo uso de las expresiones (3.7), (3.10) y (3.13), se tiene que para las variables de movimiento θ_1, θ_2 y θ_3, las matrices de transformación están dadas respectivamente por

$$e^{\hat{\xi}_1\theta_1} = \begin{bmatrix} c\theta_1 & 0 & s\theta_1 & x_1\cdot v\theta_1 - z_1\cdot s\theta_1 \\ 0 & 1 & 0 & 0 \\ -s\theta_1 & 0 & c\theta_1 & x_1\cdot s\theta_1 + z_1\cdot v\theta_1 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

(3.14)

$$e^{\hat{\xi}_2\theta_2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta_2 & -s\theta_2 & y_2\cdot v\theta_2 + z_2\cdot s\theta_2 \\ 0 & s\theta_2 & c\theta_2 & -y_2\cdot s\theta_2 + z_2\cdot v\theta_2 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

(3.15)

$$e^{\hat{\xi}_3\theta_3} = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & x_3\cdot v\theta_3 + y_3\cdot s\theta_3 \\ s\theta_3 & c\theta_3 & 0 & -x_3\cdot s\theta_3 + y_3\cdot v\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

(3.16)

En donde x_i, y_i y z_i son las coordenadas de los puntos base q_i. Cabe señalar que estas tres matrices deben incluirse en el modelo de todos los dedos, si es que se quiere analizar su movimiento incluyendo los efectos de los desplazamientos de la palma. En caso contrario pueden omitirse, pero teniendo en cuenta que si esto sucede, entonces el sistema inercial estaría ubicado sobre la palma.

3.2.2.1 Dedo índice

Del mismo modo como se hizo con la palma, y tomando como base a la Figura 3.4 y a la Tabla 3.3, es posible obtener las matrices de transformación que describen el movimiento del dedo índice, por lo cual se tiene

$$e^{\hat{\xi}_8\theta_8} = \begin{bmatrix} c\theta_8 & -s\theta_8 & 0 & x_8 \cdot v\theta_8 + y_8 \cdot s\theta_8 \\ s\theta_8 & c\theta_8 & 0 & -x_8 \cdot s\theta_8 + y_8 \cdot v\theta_8 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.17}$$

$$e^{\hat{\xi}_9\theta_9} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta_9 & -s\theta_9 & y_9 \cdot v\theta_9 + z_9 \cdot s\theta_9 \\ 0 & s\theta_9 & c\theta_9 & -y_9 \cdot s\theta_9 + z_9 \cdot v\theta_9 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.18}$$

$$e^{\hat{\xi}_{10}\theta_{10}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta_{10} & -s\theta_{10} & y_{10} \cdot v\theta_{10} + z_{10} \cdot s\theta_{10} \\ 0 & s\theta_{10} & c\theta_{10} & -y_{10} \cdot s\theta_{10} + z_{10} \cdot v\theta_{10} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.19}$$

$$e^{\hat{\xi}_{11}\theta_{11}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta_{11} & -s\theta_{11} & y_{11} \cdot v\theta_{11} + z_{11} \cdot s\theta_{11} \\ 0 & s\theta_{11} & c\theta_{11} & -y_{11} \cdot s\theta_{11} + z_{11} \cdot v\theta_{11} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.20}$$

De la expresión (2.72) se puede apreciar que además de las matrices que corresponden al efecto de las articulaciones, existe otra matriz $g_{st}(0)$, que es aquélla que se encarga de transformar las coordenadas de los puntos referidos al sistema de la herramienta, a coordenadas del sistema inercial cuando el manipulador se halla en su configuración de referencia. Dado que los ejes del sistema local de la falangeta son paralelos a los del sistema inercial, no existe rotación relativa entre los sistemas. Por lo cual sólo basta con incluir la parte correspondiente a la traslación, de modo que

$$g_{st_i}(0) = \begin{bmatrix} 1 & 0 & 0 & x_{11} \\ 0 & 1 & 0 & y_{11} \\ 0 & 0 & 1 & z_{11} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.21}$$

Finalmente, conjuntando los resultados y siguiendo el modelo del producto de exponenciales, se tiene que la cinemática directa del dedo índice queda completamente descrita por la siguiente expresión:

$$\boxed{g_{st_i}(\theta) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} e^{\hat{\xi}_3\theta_3} e^{\hat{\xi}_8\theta_8} e^{\hat{\xi}_9\theta_9} e^{\hat{\xi}_{10}\theta_{10}} e^{\hat{\xi}_{11}\theta_{11}} g_{st_i}(0)}. \tag{3.22}$$

No se muestra el desarrollo de esta expresión, ya que es más útil contar con el método de construcción de cada una de las matrices; las operaciones pueden realizarse a través de un medio más eficiente para el cálculo, como puede ser una computadora.

3.2.2.2 Dedo medio

Para determinar las matrices de transformación para el movimiento del dedo medio, se sigue exactamente el mismo procedimiento que se empleó para el dedo índice. Con base en la Figura 3.5 y en la Tabla 3.3 se tiene

$$e^{\hat{\xi}_{12}\theta_{12}} = \begin{bmatrix} \mathrm{c}\theta_{12} & -\mathrm{s}\theta_{12} & 0 & x_{12}\cdot \mathrm{v}\theta_{12} + y_{12}\cdot \mathrm{s}\theta_{12} \\ \mathrm{s}\theta_{12} & \mathrm{c}\theta_{12} & 0 & -x_{12}\cdot \mathrm{s}\theta_{12} + y_{12}\cdot \mathrm{v}\theta_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.23}$$

$$e^{\hat{\xi}_{13}\theta_{13}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathrm{c}\theta_{13} & -\mathrm{s}\theta_{13} & y_{13}\cdot \mathrm{v}\theta_{13} + z_{13}\cdot \mathrm{s}\theta_{13} \\ 0 & \mathrm{s}\theta_{13} & \mathrm{c}\theta_{13} & -y_{13}\cdot \mathrm{s}\theta_{13} + z_{13}\cdot \mathrm{v}\theta_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.24}$$

$$e^{\hat{\xi}_{14}\theta_{14}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathrm{c}\theta_{14} & -\mathrm{s}\theta_{14} & y_{14}\cdot \mathrm{v}\theta_{14} + z_{14}\cdot \mathrm{s}\theta_{14} \\ 0 & \mathrm{s}\theta_{14} & \mathrm{c}\theta_{14} & -y_{14}\cdot \mathrm{s}\theta_{14} + z_{14}\cdot \mathrm{v}\theta_{14} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.25}$$

$$e^{\hat{\xi}_{15}\theta_{15}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta_{15} & -s\theta_{15} & y_{15}\cdot v\theta_{15} + z_{15}\cdot s\theta_{15} \\ 0 & s\theta_{15} & c\theta_{15} & -y_{15}\cdot s\theta_{15} + z_{15}\cdot v\theta_{15} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.26}$$

En este caso la matriz $g_{st}(0)$ se refiere al sistema local de la falangeta del dedo medio:

$$g_{st_m}(0) = \begin{bmatrix} 1 & 0 & 0 & x_{15} \\ 0 & 1 & 0 & y_{15} \\ 0 & 0 & 1 & z_{15} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.27}$$

Por lo que la cinemática directa del dedo medio queda descrita por

$$\boxed{g_{st_m}(\theta) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} e^{\hat{\xi}_3\theta_3} e^{\hat{\xi}_{12}\theta_{12}} e^{\hat{\xi}_{13}\theta_{13}} e^{\hat{\xi}_{14}\theta_{14}} e^{\hat{\xi}_{15}\theta_{15}} g_{st_m}(0)}. \tag{3.28}$$

3.2.2.3 Dedo anular

Siguiendo el método utilizado para los dedos índice y medio, y ahora tomando en cuenta la Figura 3.6 y la Tabla 3.3 se tiene que las matrices de transformación para el dedo anular están dadas por

$$e^{\hat{\xi}_{16}\theta_{16}} = \begin{bmatrix} c\theta_{16} & -s\theta_{16} & 0 & x_{16}\cdot v\theta_{16} + y_{16}\cdot s\theta_{16} \\ s\theta_{16} & c\theta_{16} & 0 & -x_{16}\cdot s\theta_{16} + y_{16}\cdot v\theta_{16} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.29}$$

$$e^{\hat{\xi}_{17}\theta_{17}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta_{17} & -s\theta_{17} & y_{17}\cdot v\theta_{17} + z_{17}\cdot s\theta_{17} \\ 0 & s\theta_{17} & c\theta_{17} & -y_{17}\cdot s\theta_{17} + z_{17}\cdot v\theta_{17} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.30}$$

$$e^{\hat{\xi}_{18}\theta_{18}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta_{18} & -s\theta_{18} & y_{18}\cdot v\theta_{18} + z_{18}\cdot s\theta_{18} \\ 0 & s\theta_{18} & c\theta_{18} & -y_{18}\cdot s\theta_{18} + z_{18}\cdot v\theta_{18} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.31}$$

$$e^{\hat{\xi}_{19}\theta_{19}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta_{19} & -s\theta_{19} & y_{19}\cdot v\theta_{19} + z_{19}\cdot s\theta_{19} \\ 0 & s\theta_{19} & c\theta_{19} & -y_{19}\cdot s\theta_{19} + z_{19}\cdot v\theta_{19} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.32}$$

La matriz $g_{st}(0)$ al igual que en los dos casos anteriores tiene la forma siguiente:

$$g_{st_a}(0) = \begin{bmatrix} 1 & 0 & 0 & x_{19} \\ 0 & 1 & 0 & y_{19} \\ 0 & 0 & 1 & z_{19} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.33}$$

Por lo que la cinemática directa del dedo anular está dada por

$$\boxed{g_{st_a}(\theta) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} e^{\hat{\xi}_3\theta_3} e^{\hat{\xi}_{16}\theta_{16}} e^{\hat{\xi}_{17}\theta_{17}} e^{\hat{\xi}_{18}\theta_{18}} e^{\hat{\xi}_{19}\theta_{19}} g_{st_a}(0)}. \tag{3.34}$$

3.3 Cinemática directa del dedo pulgar

Las expresiones de las que se hace uso para la determinación de la cinemática directa del dedo pulgar son las mismas que se emplearon para el resto de los dedos. El hecho de que el pulgar se maneje por separado, se debe en primer lugar a que la orientación del dedo pulgar es diferente de la de los otros dedos, y en segundo, sirve para ilustrar el método de cómo se establecen las matrices de transformación cuando se tiene un miembro con una orientación arbitraria.

Como se vio, primero se obtienen las coordenadas de giro que parametrizan a la articulación. En esta ocasión, al tratarse de pares cinemáticos de revolución, se trata de establecer una expresión general para dicho tipo de coordenadas. Sea ω:[ω_x, ω_y, ω_z] el vector *unitario* del eje en torno al cual se lleva a cabo el movimiento, y q:[x, y, z] un punto sobre el eje. De acuerdo con la ecuación (2.70), las coordenadas de giro estarán dadas por

$$\xi = \begin{bmatrix} \omega_z y - \omega_y z \\ \omega_x z - \omega_z x \\ \omega_y x - \omega_x y \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}. \tag{3.35}$$

El término $e^{\hat{\omega}\theta}$ acorde con lo establecido por la Fórmula de Rodrigues, tendría en su expresión más general la siguiente forma:

$$e^{\hat{\omega}\theta} = \begin{bmatrix} \omega_x^2 \mathrm{v}\theta + \mathrm{c}\theta & \omega_x\omega_y \mathrm{v}\theta - \omega_z \mathrm{s}\theta & \omega_x\omega_z \mathrm{v}\theta + \omega_y \mathrm{s}\theta \\ \omega_x\omega_y \mathrm{v}\theta + \omega_z \mathrm{s}\theta & \omega_y^2 \mathrm{v}\theta + \mathrm{c}\theta & \omega_y\omega_z \mathrm{v}\theta - \omega_x \mathrm{s}\theta \\ \omega_x\omega_z \mathrm{v}\theta - \omega_y \mathrm{s}\theta & \omega_y\omega_z \mathrm{v}\theta + \omega_x \mathrm{s}\theta & \omega_z^2 \mathrm{v}\theta + \mathrm{c}\theta \end{bmatrix}. \tag{3.36}$$

Haciendo las operaciones indicadas en la expresión (3.4) se tiene que la matriz de transformación para un par de revolución con orientación arbitraria de su eje estaría dada por

$$e^{\hat{\xi}\theta} = \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix},$$

en donde

$$\begin{aligned}
e_{11} &= \omega_x^2 \cdot \mathrm{v}\theta + \mathrm{c}\theta \\
e_{12} &= \omega_x \cdot \omega_y \cdot \mathrm{v}\theta - \omega_z \cdot \mathrm{s}\theta \\
e_{13} &= \omega_x \cdot \omega_z \cdot \mathrm{v}\theta + \omega_y \cdot \mathrm{s}\theta \\
e_{14} &= x \cdot \mathrm{v}\theta \cdot (1 - \omega_x^2) + y \cdot (\mathrm{s}\theta \cdot \omega_z - \omega_x \cdot \omega_y \cdot \mathrm{v}\theta) + z \cdot (-\mathrm{s}\theta \cdot \omega_y - \omega_x \cdot \omega_z \cdot \mathrm{v}\theta)
\end{aligned}$$

$$\begin{aligned}
e_{21} &= \omega_x \cdot \omega_y \cdot \mathrm{v}\theta + \omega_z \cdot \mathrm{s}\theta \\
e_{22} &= \omega_y^2 \cdot \mathrm{v}\theta + \mathrm{c}\theta \\
e_{23} &= \omega_y \cdot \omega_z \cdot \mathrm{v}\theta - \omega_x \cdot \mathrm{s}\theta \\
e_{24} &= x \cdot (-\mathrm{s}\theta \cdot \omega_z - \omega_x \cdot \omega_y \cdot \mathrm{v}\theta) + y \cdot \mathrm{v}\theta \cdot (1 - \omega_y^2) + z \cdot (\mathrm{s}\theta \cdot \omega_x - \omega_y \cdot \omega_z \cdot \mathrm{v}\theta)
\end{aligned}$$

$$\begin{aligned}
e_{31} &= \omega_x \cdot \omega_z \cdot \mathrm{v}\theta - \omega_y \cdot \mathrm{s}\theta \\
e_{32} &= \omega_y \cdot \omega_z \cdot \mathrm{v}\theta + \omega_x \cdot \mathrm{s}\theta \\
e_{33} &= \omega_z^2 \cdot \mathrm{v}\theta + \mathrm{c}\theta \\
e_{34} &= x \cdot (\mathrm{s}\theta \cdot \omega_y - \omega_x \cdot \omega_z \cdot \mathrm{v}\theta) + y \cdot (-\mathrm{s}\theta \cdot \omega_x - \omega_y \cdot \omega_z \cdot \mathrm{v}\theta) + z \cdot \mathrm{v}\theta \cdot (1 - \omega_z^2)
\end{aligned}$$

$$e_{41}=0$$
$$e_{42}=0$$
$$e_{43}=0$$
$$e_{44}=1$$

(3.37)

Por lo que toca a la matriz $g_{st}(0)$, como los ejes de los sistemas locales del pulgar no son paralelos con respecto a los del sistema inercial; hay que introducir una submatriz de rotación que transforme las coordenadas referidas al sistema local de la falangeta del pulgar, a coordenadas del sistema base[7]. Esta submatriz, como se vio en el capítulo anterior, se integra con los vectores del sistema local en coordenadas del sistema inercial acomodados en columnas. Como en este caso la orientación de los ejes del sistema local tiene que cumplir con características especiales, las cuales están denotadas por las expresiones (3.1), (3.2) y (3.3), se pondrá a la matriz $g_{st}(0)$ en función de los mismos parámetros que se emplearon para determinar la orientación de los ejes del pulgar:

$$g_{st_p}(0)=\begin{bmatrix} c\alpha\cdot c\beta & -s\alpha & -c\alpha\cdot s\beta & x_7 \\ s\alpha\cdot c\beta & c\alpha & -s\alpha\cdot s\beta & y_7 \\ s\beta & 0 & c\beta & z_7 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

(3.38)

en donde las coordenadas del punto base, representadas en la cuarta columna de la matriz, se calculan de acuerdo con las expresiones dadas en la Tabla 3.4.

Así pues, eligiendo los parámetros correspondientes a cada articulación del dedo pulgar en lo que toca a ejes de rotación y puntos base, y apoyándose en las expresiones anteriores, es posible construir el modelo que describe su comportamiento cinemático directo:

$$\boxed{g_{st_p}(\theta)=e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}e^{\hat{\xi}_4\theta_4}e^{\hat{\xi}_5\theta_5}e^{\hat{\xi}_6\theta_6}e^{\hat{\xi}_7\theta_7}g_{st_p}(0)}$$

(3.39)

3.4 Jacobiano de los dedos índice, medio y anular

Como se vio en el capítulo anterior, el Jacobiano permite calcular las velocidades cartesianas de los eslabones de una cadena cinemática a partir de las velocidades relativas en las articulaciones. Además de esto, su obtención representa por un lado, el primer paso hacia lo que es el análisis dinámico del mecanismo, y por otro, el medio para conocer la capacidad que posee un robot para llevar a cabo tareas de manipulación con cierta destreza.

[7] Al sistema de referencia inercial también se le conoce como sistema base.

En las expresiones (2.77) y (2.78) se puede apreciar que la estructura del Jacobiano inercial de un manipulador de cadena abierta es la siguiente

$$J_{st}^{s}(\theta)=\left[\xi_1 \quad \xi_2' \quad \cdots \quad \xi_n'\right],$$

$$\xi_i'=\mathrm{Ad}_{\left(e^{\hat{\xi}_1\theta_1}\cdots e^{\hat{\xi}_{i-1}\theta_{i-1}}\right)}\xi_i .$$

Se había dicho que sus columnas correspondían a las coordenadas de giro de las articulaciones que intervenían, con la particularidad de que dichas coordenadas iban siendo afectadas por el movimiento de articulaciones anteriores.

Cuando se analiza un manipulador, especialmente uno que posee estructuras semejantes a dedos (no necesariamente antropomórfico), y se quiere determinar la capacidad que tiene para realizar ciertos movimientos, se pone particular interés en los elementos que integran a los dedos y los desplazamientos de éstos con respecto a la estructura que los sostiene, que para el caso que en este trabajo se trata, sería el movimiento de los dedos con respecto a la palma. Lo que se pretende decir con esto es que si se desea obtener el Jacobiano de los dedos, únicamente basta con incluir en éste, el efecto de sus propios grados de libertad, excluyendo los de la palma. De hecho si se observan las matrices de transformación que intervienen en el modelo cinemático de los dedos, puede notarse que si la variable de movimiento vale cero, entonces la matriz de transformación se convierte en una matriz identidad, con lo que no se afectan las demás partes del modelo; en otras palabras, es como si no se incluyeran en éste. Por lo que el no incluir las matrices correspondientes a las articulaciones de la palma, equivale a que sus variables tomaran un valor de cero.

Aprovechando esto y suponiendo que en la muñeca no ocurren desplazamientos, es posible eliminar del modelo cinemático las matrices de transformación correspondientes a dichos movimientos, con lo que se obtiene una simplificación importante. De este modo el Jacobiano de la falangeta del dedo índice estaría dado por

$$\boxed{J_{st_i}^{s}(\theta)=\left[\xi_8 \quad \xi_9' \quad \xi_{10}' \quad \xi_{11}'\right]}, \tag{3.40}$$

$$\boxed{\xi_i'=\mathrm{Ad}_{\left(e^{\hat{\xi}_8\theta_8}\cdots e^{\hat{\xi}_{i-1}\theta_{i-1}}\right)}\xi_i}. \tag{3.41}$$

Esta última expresión puede desarrollarse con el fin de hacer más transparente su cálculo, por lo que se tiene

$$\xi_i'=\begin{bmatrix} R & \hat{p}R \\ 0 & R \end{bmatrix}\begin{bmatrix} v_i \\ \omega_i \end{bmatrix}=\begin{bmatrix} Rv_i+\hat{p}R\omega_i \\ R\omega_i \end{bmatrix}=\begin{bmatrix} Rv_i+p\times\left(R\omega_i\right) \\ R\omega_i \end{bmatrix}=\begin{bmatrix} -R\omega_i\times\left(Rq_i+p\right) \\ R\omega_i \end{bmatrix}. \tag{3.42}$$

Llevando a cabo las operaciones indicadas, da como resultado que las columnas del Jacobiano están dadas por

$$\xi_8' = \begin{bmatrix} y_8 \\ -x_8 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \tag{3.43}$$

$$\xi_9' = \begin{bmatrix} -z_5 s\theta_8 \\ z_5 c\theta_8 \\ -y_8 c\theta_8 + x_8 s\theta_8 + y_8 - y_9 \\ c\theta_8 \\ s\theta_8 \\ 0 \end{bmatrix}, \tag{3.44}$$

$$\xi_{10}' = \begin{bmatrix} s\theta_8\left[(z_9 - z_{10})c\theta_9 + (y_9 - y_{10})s\theta_9 - z_9\right] \\ -c\theta_8\left[(z_9 - z_{10})c\theta_9 + (y_9 - y_{10})s\theta_9 - z_9\right] \\ -y_8 c\theta_8 + x_8 s\theta_8 + (y_9 - y_{10})c\theta_9 + (z_{10} - z_9)s\theta_9 + y_8 - y_9 \\ c\theta_8 \\ s\theta_8 \\ 0 \end{bmatrix}, \tag{3.45}$$

$$\xi_{11}' = \begin{bmatrix} s\theta_8\left\{\begin{matrix} c\theta_9\left[(z_{10} - z_{11})c\theta_{10} + (y_{10} - y_{11})s\theta_{10} + z_9 - z_{10}\right] + \\ + s\theta_9\left[(y_{10} - y_{11})c\theta_{10} + (z_{11} - z_{10})s\theta_{10} + y_9 - y_{10}\right] - z_9 \end{matrix}\right\} \\ -c\theta_8\left\{\begin{matrix} c\theta_9\left[(z_{10} - z_{11})c\theta_{10} + (y_{10} - y_{11})s\theta_{10} + z_9 - z_{10}\right] + \\ + s\theta_9\left[(y_{10} - y_{11})c\theta_{10} + (z_{11} - z_{10})s\theta_{10} + y_9 - y_{10}\right] - z_9 \end{matrix}\right\} \\ \left\{\begin{matrix} -y_8 c\theta_8 + x_8 s\theta_8 + c\theta_9\left[(y_{10} - y_{11})c\theta_{10} + (z_{11} - z_{10})s\theta_{10} + y_9 - y_{10}\right] + \\ -s\theta_9\left[(z_{10} - z_{11})c\theta_{10} + (y_{10} - y_{11})s\theta_{10} + z_9 - z_{10}\right] + y_8 - y_9 \end{matrix}\right\} \\ c\theta_8 \\ s\theta_8 \\ 0 \end{bmatrix}. \tag{3.46}$$

Recordando cómo se calcula la velocidad cartesiana de un punto de la falangeta en coordenadas del sistema inercial se tiene

$$v_q^s = \hat{V}_{st}^s q^s = \left(J_{st}^s(\theta)\dot{\theta}\right)^{\wedge} q^s,$$

en donde $\dot{\theta}$ es el vector columna que contiene las velocidades angulares de las articulaciones; y q^s que es un punto unido al extremo distal del dedo en coordenadas del sistema inercial.

Por lo que toca a los Jacobianos de los dedos medio y anular, basta con sustituir en las expresiones (3.43), (3.44), (3.45) y (3.46) los puntos q_8, q_9, q_{10} y q_{11}, y los ángulos θ_8, θ_9, θ_{10}, y θ_{11}; por los puntos y los ángulos equivalentes, teniendo en cuenta que $q_i : [x_i, y_i, z_i]$. De esta manera, el Jacobiano del dedo medio estaría dado por

$$\boxed{J_{st_m}^s(\theta) = \left[\xi_{12} \quad \xi_{13}' \quad \xi_{14}' \quad \xi_{15}'\right]}, \tag{3.47}$$

$$\boxed{\xi_i' = \mathrm{Ad}_{\left(e^{\hat{\xi}_{12}\theta_{12}} \ldots e^{\hat{\xi}_{i-1}\theta_{i-1}}\right)}\xi_i}, \tag{3.48}$$

y el del dedo anular por

$$\boxed{J_{st_a}^s(\theta) = \left[\xi_{16} \quad \xi_{17}' \quad \xi_{18}' \quad \xi_{19}'\right]}, \tag{3.49}$$

$$\boxed{\xi_i' = \mathrm{Ad}_{\left(e^{\hat{\xi}_{16}\theta_{16}} \ldots e^{\hat{\xi}_{i-1}\theta_{i-1}}\right)}\xi_i}. \tag{3.50}$$

3.5 Jacobiano del dedo pulgar

Con el pulgar se puede hacer exactamente la misma suposición que se hizo para el resto de los dedos, es decir, el movimiento de la muñeca puede descartarse y dejar únicamente los grados de libertad que corresponden al dedo. Así se tendría que la expresión del Jacobiano queda definida por

$$\boxed{J_{st_p}^s(\theta) = \left[\xi_4 \quad \xi_5' \quad \xi_6' \quad \xi_7'\right]}, \tag{3.51}$$

$$\boxed{\xi_i' = \mathrm{Ad}_{\left(e^{\hat{\xi}_4\theta_4} \ldots e^{\hat{\xi}_{i-1}\theta_{i-1}}\right)}\xi_i}. \tag{3.52}$$

Dada la complejidad en el desarrollo de sus términos, lo que se hará es proporcionar las expresiones que llevan a su obtención. Para ello se requiere hacer antes algunos cálculos útiles. Así pues se tiene que

$$e^{\hat{\xi}_4\theta_4} = \begin{bmatrix} R_4 & p_4 \\ 0 & 1 \end{bmatrix},$$

$$e^{\hat{\xi}_4\theta_4} e^{\hat{\xi}_5\theta_5} = \begin{bmatrix} R_4R_5 & R_4p_5 + p_4 \\ 0 & 1 \end{bmatrix},$$

$$e^{\hat{\xi}_4\theta_4} e^{\hat{\xi}_5\theta_5} e^{\hat{\xi}_6\theta_6} = \begin{bmatrix} R_4R_5R_6 & R_4R_5p_6 + R_4p_5 + p_4 \\ 0 & 1 \end{bmatrix}.$$

Con estos resultados y siguiendo lo establecido en (3.41) y (3.42) se tiene finalmente

$$\xi_4 = \begin{bmatrix} -\omega_4 \times q_4 \\ \omega_4 \end{bmatrix}, \tag{3.53}$$

$$\xi_5' = \begin{bmatrix} -R_4\omega_5 \times (R_4q_5 + p_4) \\ R_4\omega_5 \end{bmatrix}, \tag{3.54}$$

$$\xi_6' = \begin{bmatrix} -R_4R_5\omega_6 \times (R_4R_5q_6 + R_4p_5 + p_4) \\ R_4R_5\omega_6 \end{bmatrix}, \tag{3.55}$$

$$\xi_7' = \begin{bmatrix} -R_4R_5R_6\omega_7 \times (R_4R_5R_6q_7 + R_4R_5p_6 + R_4p_5 + p_4) \\ R_4R_5R_6\omega_7 \end{bmatrix}. \tag{3.56}$$

Antes de concluir, cabe aclarar que las expresiones de los Jacobianos corresponden en cada caso al de la falangeta de los dedos. Si se quisiera analizar el movimiento de los otros elementos constitutivos de los dedos, que además es muy factible dado que toda la estructura del dedo es un posible punto de contacto con un objeto de manipulación, bastaría con eliminar la última columna si se tratara de la falangina, y con eliminar las dos últimas si se tratara de las falanges o del metacarpo en el caso del pulgar.

CAPÍTULO 4

Planeación de trayectorias para el movimiento de los dedos

Uno de los problemas más importantes en robótica, es el de controlar los movimientos de un manipulador de modo que siga una trayectoria preestablecida o que sea capaz de generarla a partir de las condiciones que le impone el medio exterior. Aunque esto puede descomponerse en una serie de problemas más pequeños, existen dos que van de acuerdo con los intereses del proyecto aquí presentado. El primero está relacionado con la generación de la trayectoria en sí, y el segundo tiene que ver con la cinemática inversa del mecanismo.

Para generar una trayectoria, tienen que tomarse en cuenta varios factores que restringen la capacidad de movimiento. Entre estos factores se puede mencionar la evasión de obstáculos; el hecho de querer alcanzar puntos de interés para la realización de tareas específicas; y las mismas limitaciones del volumen de trabajo del manipulador. Esto último tiene que ver con la cinemática inversa, ya que es ésta la que indica si un determinado punto del espacio puede ser alcanzado; y si se puede, entonces también establece la forma como se alcanza. Recordando de los capítulos anteriores, la cinemática inversa permite conocer los ángulos que tiene que haber entre los eslabones para que algún elemento del manipulador, en este caso la herramienta, alcance la posición especificada por la trayectoria.

Se podría pensar que la planeación de una trayectoria sería más conveniente y apreciable en un brazo, más que en un elemento de manipulación, como lo podría ser una mano mecánica. Sin embargo, el hecho de contar con un manipulador de tipo antropomórfico, en el que se incorporan una serie de pequeños manipuladores, trae consigo el que éstos tengan que actuar de forma coordinada a fin de realizar tareas más específicas. Aunque en este caso aún no se ha desarrollado el modelo que corresponde a dicha coordinación, es evidente que se requiere de una base que permita generar los movimientos necesarios.

Otras restricciones que pueden incorporarse a la generación de trayectorias tienen que ver con condiciones de velocidad y aceleración en determinados puntos. En este trabajo no se incorpora la parte dinámica debido a que el interés principal en esta primera etapa de construcción y pruebas estaba enfocada en lograr el movimiento del manipulador, es decir, en controlar únicamente en base a posición. Además, la parte de control se diseñó para cubrir de manera general algunos aspectos dinámicos. Así que por el momento basta con generar el vector de ángulos en las articulaciones para alcanzar las posiciones especificadas.

4.1 Cinemática inversa de los dedos

Tal como se señaló, uno de los aspectos que intervienen para que un robot pueda seguir una trayectoria determinada, es su cinemática inversa. En el caso del manipulador antropomórfico, como se pretende que en un futuro los dedos puedan actuar de manera coordinada para llevar a cabo tareas con cierta habilidad, el análisis de cinemática inversa estará centrado en la obtención de los valores de los ángulos únicamente de los dedos, es decir, se considerará que el sistema inercial estará localizado sobre la palma.

4.1.1 Cinemática inversa de los dedos índice, medio y anular

En el capítulo anterior se hizo mención de algunas consideraciones estructurales y de movimiento del manipulador. Entre éstas, se encuentra aquélla que dice que para los dedos índice, medio y anular, el desplazamiento de la falangeta está directamente relacionado con el de la falangina, de modo que puede considerarse que el ángulo relativo entre la falange y la falangina es prácticamente el mismo que existe entre la falangina y la falangeta. Tomando en cuenta esto último, es que se establece el método de obtención de los diferentes ángulos del dedo.

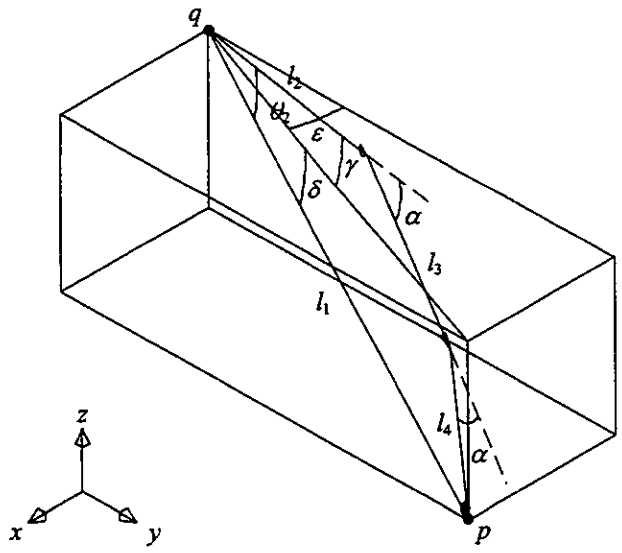

Figura 4.1 Parámetros para la cinemática inversa de los dedos índice, medio y anular.

Sean q:$[q_x, q_y, q_z]$ el punto donde se une el dedo con la palma, y p:$[p_x, p_y, p_z]$ el punto al cual se pretende llegar con el extremo del dedo (ver Figura 4.1). Del mismo modo, sean respectivamente l_2, l_3 y l_4 las longitudes de falange, falangina y falangeta. La distancia entre los puntos p y q puede obtenerse mediante la magnitud del vector que los une, por lo que

$$l_1 = \|p - q\|.$$

Una vez hecho esto, se puede acomodar la figura formada por los segmentos l_1, l_2, l_3 y l_4 de una manera conveniente de tal forma que la obtención de los ángulos relativos entre eslabones sea más sencilla. En la Figura 4.2 se muestran los segmentos como si se tratara de un mecanismo de cuatro barras, en donde el segmento l_1 actúa como la barra fija. Además de las condiciones con las que tiene que cumplir un mecanismo normal de cuatro barras, en este caso tiene que cumplirse con otra restricción: que los ángulos entre las barras l_2 y l_3, y entre las barras l_3 y l_4 sean iguales (ángulo denotado por α).

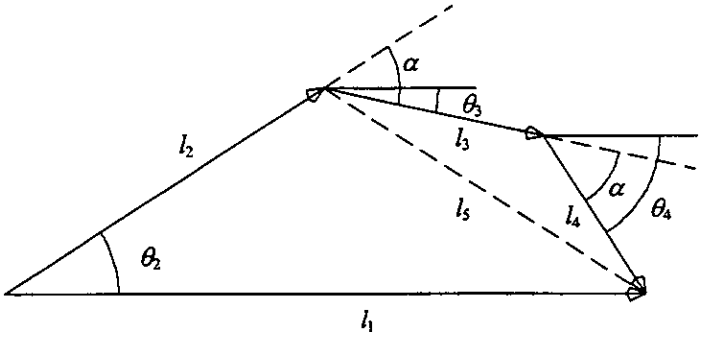

Figura 4.2 Mecanismo de cuatro barras formado por los elementos de los dedos.

Para llevar a cabo el análisis del mecanismo se puede aprovechar la notación de números complejos. La configuración del mecanismo puede expresarse mediante una suma vectorial que tendría la siguiente forma:

$$l_2 e^{i\theta_2} + l_3 e^{i\theta_3} + l_4 e^{i\theta_4} = l_1 e^{i\theta_1}, \tag{4.1}$$

en donde θ_i representa el ángulo que forma cada uno de los segmentos con la horizontal (siendo la rotación positiva aquélla que va en contra de las manecillas del reloj). Recordando que $e^{i\theta} = cis\theta = \cos\theta + i\,\mathrm{sen}\theta$, se tiene

$$l_2 cis\theta_2 + l_3 cis\theta_3 + l_4 cis\theta_4 = l_1 cis\theta_1.$$

Separando la parte real de la parte imaginaria y definiendo $c\theta = \cos\theta$, $s\theta = \mathrm{sen}\theta$ se tiene

$$l_2 c\theta_2 + l_3 c\theta_3 + l_4 c\theta_4 = l_1 c\theta_1,$$
$$l_2 s\theta_2 + l_3 s\theta_3 + l_4 s\theta_4 = l_1 s\theta_1.$$

Si se considera que el segmento l_1 se halla completamente horizontal, entonces el ángulo $\theta_1 = 0$, por lo que

$$l_2 c\theta_2 + l_3 c\theta_3 + l_4 c\theta_4 = l_1, \tag{4.2}$$
$$l_2 s\theta_2 + l_3 s\theta_3 + l_4 s\theta_4 = 0. \tag{4.3}$$

Observando la figura se aprecia que los ángulos θ_3 y θ_4 pueden expresarse en términos de θ_2 y de α, de modo que

$$\theta_3 = \theta_2 - \alpha, \tag{4.4}$$
$$\theta_4 = \theta_2 - 2\alpha, \tag{4.5}$$

que al sustituirse en (4.2) y (4.3) se tiene

$$l_2 c\theta_2 + l_3 c(\theta_2 - \alpha) + l_4 c(\theta_2 - 2\alpha) = l_1,$$
$$l_2 s\theta_2 + l_3 s(\theta_2 - \alpha) + l_4 s(\theta_2 - 2\alpha) = 0.$$

Empleando las identidades trigonométricas para el coseno y el seno de una diferencia de ángulos[8], las ecuaciones anteriores pueden expresarse como

$$l_2 c\theta_2 + l_3\left(c\theta_2 c\alpha + s\theta_2 s\alpha\right) + l_4\left(c\theta_2 c(2\alpha) + s\theta_2 s(2\alpha)\right) = l_1, \tag{4.6}$$
$$l_2 s\theta_2 + l_3\left(s\theta_2 c\alpha - c\theta_2 s\alpha\right) + l_4\left(s\theta_2 c(2\alpha) - c\theta_2 s(2\alpha)\right) = 0. \tag{4.7}$$

Para simplificar aun más la notación, se hace una sustitución de términos de tal forma que

$$[\mathrm{I}] = \left(c\theta_2 c\alpha + s\theta_2 s\alpha\right),$$
$$[\mathrm{II}] = \left(c\theta_2 c(2\alpha) + s\theta_2 s(2\alpha)\right),$$
$$[\mathrm{III}] = \left(s\theta_2 c\alpha - c\theta_2 s\alpha\right),$$
$$[\mathrm{IV}] = \left(s\theta_2 c(2\alpha) - c\theta_2 s(2\alpha)\right).$$

[8] $\cos(a - b) = \cos(a)\cos(b) + \mathrm{sen}(a)\,\mathrm{sen}(b)$,
$\mathrm{sen}(a - b) = \mathrm{sen}(a)\cos(b) - \cos(a)\,\mathrm{sen}(b)$.

Realizando las sustituciones propuestas, y elevando al cuadrado las ecuaciones (4.6) y (4.7) se obtiene

$$l_2^2 \mathrm{c}^2\theta_2 + l_3^2[\mathrm{I}]^2 + l_4^2[\mathrm{II}]^2 + 2l_2l_3\mathrm{c}\theta_2[\mathrm{I}] + 2l_2l_4\mathrm{c}\theta_2[\mathrm{II}] + 2l_3l_4[\mathrm{I}][\mathrm{II}] = l_1^2,$$
$$l_2^2 \mathrm{s}^2\theta_2 + l_3^2[\mathrm{III}]^2 + l_4^2[\mathrm{IV}]^2 + 2l_2l_3\mathrm{s}\theta_2[\mathrm{III}] + 2l_2l_4\mathrm{s}\theta_2[\mathrm{IV}] + 2l_3l_4[\mathrm{III}][\mathrm{IV}] = 0.$$

Expresiones que al sumarse dan como resultado

$$l_2^2\left(\mathrm{c}^2\theta_2 + \mathrm{s}^2\theta_2\right) + l_3^2\left([\mathrm{I}]^2 + [\mathrm{III}]^2\right) + l_4^2\left([\mathrm{II}]^2 + [\mathrm{IV}]^2\right) +$$
$$+ 2l_2l_3\left(\mathrm{c}\theta_2[\mathrm{I}] + \mathrm{s}\theta_2[\mathrm{III}]\right) + 2l_2l_4\left(\mathrm{c}\theta_2[\mathrm{II}] + \mathrm{s}\theta_2[\mathrm{IV}]\right) + 2l_3l_4\left([\mathrm{I}][\mathrm{II}] + [\mathrm{III}][\mathrm{IV}]\right) = l_1^2 \tag{4.8}$$

Llevando a cabo el desarrollo de los términos puede notarse que

$$\left(\mathrm{c}^2\theta_2 + \mathrm{s}^2\theta_2\right) = 1$$
$$\left([\mathrm{I}]^2 + [\mathrm{III}]^2\right) = 1$$
$$\left([\mathrm{II}]^2 + [\mathrm{IV}]^2\right) = 1$$
$$\left(\mathrm{c}\theta_2[\mathrm{I}] + \mathrm{s}\theta_2[\mathrm{III}]\right) = \mathrm{c}\alpha$$
$$\left(\mathrm{c}\theta_2[\mathrm{II}] + \mathrm{s}\theta_2[\mathrm{IV}]\right) = \mathrm{c}(2\alpha) = 2\mathrm{c}^2\alpha - 1$$
$$\left([\mathrm{I}][\mathrm{II}] + [\mathrm{III}][\mathrm{IV}]\right) = \mathrm{c}\alpha$$

que al sustituirse en (4.8) da

$$l_2^2 + l_3^2 + l_4^2 - l_1^2 + 2l_2l_3\mathrm{c}\alpha + 2l_2l_4\left(2\mathrm{c}^2\alpha - 1\right) + 2l_3l_4\mathrm{c}\alpha = 0.$$

Los términos de la ecuación anterior pueden agruparse de una manera conveniente, de manera que se obtenga la forma de una ecuación de segundo grado:

$$\left(4l_2l_4\right)\mathrm{c}^2\alpha + \left(2l_3\left(l_2 + l_4\right)\right)\mathrm{c}\alpha + \left(l_2^2 + l_3^2 + l_4^2 - l_1^2 - 2l_2l_4\right) = 0. \tag{4.9}$$

Aprovechando la expresión general para resolver ecuaciones de segundo grado, se tiene que el ángulo α estaría dado por

$$\boxed{\alpha_{1,2} = ang\cos\left(\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}\right)}, \tag{4.10}$$

en donde

$$\boxed{A = 4l_2l_4}, \tag{4.11}$$

$$\boxed{B = 2l_3\left(l_2 + l_4\right)}, \tag{4.12}$$

$$\boxed{C = l_2^2 + l_3^2 + l_4^2 - l_1^2 - 2l_2l_4}. \tag{4.13}$$

Una vez calculado α, se procede a la obtención del ángulo θ_2. De la Figura 4.2 se observa que puede formarse un triángulo con los segmentos l_2, l_3 y l_5, en donde la magnitud al cuadrado de este último está dada, de acuerdo con la ley de los cosenos, por

$$l_5^2 = l_3^2 + l_4^2 - 2l_3l_4\mathrm{c}(180 - \alpha),$$

pudiéndose reescribir como

$$l_5^2 = l_3^2 + l_4^2 + 2l_3l_4\mathrm{c}\alpha. \tag{4.14}$$

De la misma manera puede formarse otro triángulo con los segmentos l_1, l_2 y l_5, de modo que al utilizar nuevamente la ley de los cosenos se tiene

$$l_5^2 = l_1^2 + l_2^2 - 2l_1l_2\mathrm{c}\theta_2. \tag{4.15}$$

Igualando (4.14) y (4.15), y resolviendo para θ_2 finalmente resulta

$$\boxed{\theta_2 = ang\cos\left(\frac{l_1^2 + l_2^2 - l_3^2 - l_4^2 - 2l_3l_4\mathrm{c}\alpha}{2l_1l_2}\right)}. \tag{4.16}$$

Dado que ahora el sistema inercial está fijado a la palma y por consiguiente los ángulos de la falange tienen que medirse a partir del plano y-z y del plano x-y, el ángulo θ_2 no es directamente el ángulo que se busca. De la Figura 4.1 puede verse que el ángulo que realmente interesa, es aquel que forma el segmento l_2 con el plano horizontal, es decir, el ángulo γ. Para poder calcularlo se necesita obtener primero el ángulo δ, que es precisamente el ángulo que forma dicho plano horizontal con el segmento l_1 y que está dado por

$$\delta = ang\,\mathrm{sen}\left(\frac{p_y - q_y}{l_1}\right), \tag{4.17}$$

De esta manera se tiene que

$$\gamma = \theta_2 + \delta, \tag{4.18}$$

en donde el valor de δ puede ser positivo o negativo, dependiendo del punto al que se pretenda llegar.

Además de estos ángulos también se requiere determinar el ángulo que existe entre el plano que contiene al dedo, y el plano y-z. Este ángulo, denotado por ε puede obtenerse mediante la siguiente expresión:

$$\varepsilon = ang\,\mathrm{sen}\left(\frac{q_x - p_x}{l_1}\right), \tag{4.19}$$

en la cual puede notarse que el orden de los términos en el numerador está invertido. Esto es para que el signo del ángulo esté de acuerdo con el rango de valores establecido en la configuración de referencia.

Como puede verse de la expresión (2.10), existen dos soluciones para el ángulo α , sin embargo, como el movimiento del dedo se halla restringido por los límites establecidos en el capítulo anterior, puede descartarse la solución que no se encuentre dentro de dichos límites, de modo que automáticamente queda condicionada la solución de θ_2. Por lo que toca al resto de los ángulos no existe mayor problema en cuanto a multiplicidad de soluciones.

4.1.2 Cinemática inversa del dedo pulgar

El modo como se aborda el problema de la cinemática inversa del dedo pulgar es muy similar al que se empleó para resolver el de los dedos índice, medio y anular. Sin embargo existe un detalle que hace que la solución cambie ligeramente. A diferencia de lo que ocurre con el resto de los dedos, el movimiento de la falangeta del pulgar puede considerarse independiente del de la falangina, con lo que se tiene un grado de libertad adicional.

En la Figura 4.3 se muestra un esquema parecido al que se utilizó para los demás dedos, con la diferencia de que el ángulo entre el metacarpo y la falangina (β), y el ángulo entre la falangina y la falangeta (α) en este caso no son iguales. Otra cuestión que vale la pena señalar, es que el sistema de referencia que se muestra, no es paralelo al sistema inercial, sino que se trata de un sistema cuyos ejes son paralelos a los ejes de movimiento del pulgar, razón por la cual, las coordenadas que se manejan para el análisis son relativas a dicho sistema.

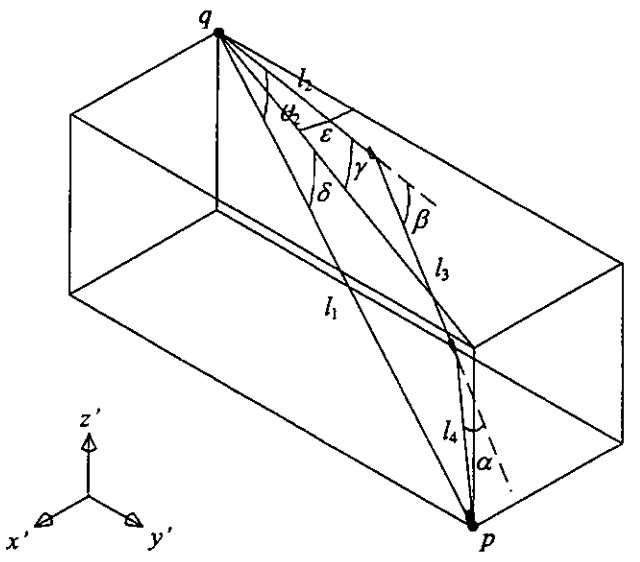

Figura 4.3 Parámetros para la cinemática inversa del pulgar.

De esta manera sean q:[$q_{x'}$, $q_{y'}$, $q_{z'}$] el punto donde se une el pulgar con la palma, y p:[$p_{x'}$, $p_{y'}$, $p_{z'}$] el punto al cual se pretende llegar con el extremo de la falangeta. Así mismo sean l_2, l_3 y l_4 las longitudes de metacarpo, falangina y falangeta respectivamente. Para poder seguir con el procedimiento, es necesario obtener de nueva cuenta la distancia entre los puntos p y q, por lo que

$$l_1 = \|p - q\|.$$

Una vez más puede acomodarse el plano que contiene a los elementos del dedo de un modo que permita ver con claridad la disposición de éstos. En la Figura 4.4 se muestra la geometría originada por los diferentes segmentos, y en la cual puede hacerse la semejanza con un mecanismo de cuatro barras. Como en el caso anterior, θ_i representa el ángulo que forma cada uno de los elementos con la horizontal, siendo la medición positiva aquélla que se hace en el sentido contrario a las manecillas del reloj.

Así pues, considerando que l_1 representa la barra fija, puede hacerse la descripción de las configuraciones del supuesto mecanismo mediante una suma vectorial. Aprovechando nuevamente la notación para números complejos, se tendría

$$l_2 e^{i\theta_2} + l_3 e^{i\theta_3} + l_4 e^{i\theta_4} = l_1 e^{i\theta_1}.$$

Separando la parte imaginaria de la parte real quedaría

$$l_2 c\theta_2 + l_3 c\theta_3 + l_4 c\theta_4 = l_1 c\theta_1,$$
$$l_2 s\theta_2 + l_3 s\theta_3 + l_4 s\theta_4 = l_1 s\theta_1.$$

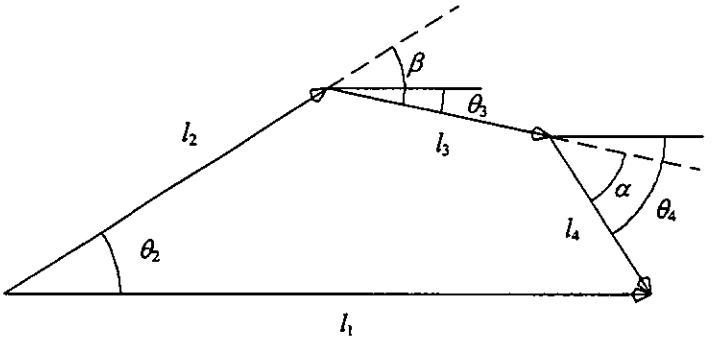

Figura 4.4 Mecanismo de cuatro barras formado por los elementos del pulgar.

Acomodando los términos y suponiendo que el segmento l_1 se encuentra completamente horizontal ($\theta_1 = 0$), se tendría entonces que

$$-l_1 + l_2 c\theta_2 + l_4 c\theta_4 = -l_3 c\theta_3 , \tag{4.20}$$

$$l_2 s\theta_2 + l_4 s\theta_4 = -l_3 s\theta_3 . \tag{4.21}$$

Elevando ambas expresiones al cuadrado se obtiene

$$l_1^2 + l_2^2 c^2\theta_2 + l_4^2 c^2\theta_4 - 2l_1 l_2 c\theta_2 - 2l_1 l_4 c\theta_4 + 2l_2 l_4 c\theta_2 c\theta_4 = l_3^2 c\theta_3 ,$$
$$l_2^2 s^2\theta_2 + l_4^2 s^2\theta_4 + 2l_2 l_4 s\theta_2 s\theta_4 = l_3^2 s\theta_3 ,$$

que al sumarse resulta

$$l_1^2 + l_2^2 + l_4^2 - 2l_1 l_2 c\theta_2 - 2l_1 l_4 c\theta_4 + 2l_2 l_4 \left(c\theta_2 c\theta_4 + s\theta_2 s\theta_4\right) = l_3^2 .$$

Agrupando términos y dividiendo entre $2l_2 l_4$ se tiene

$$\frac{l_1^2 + l_2^2 + l_4^2 - l_3^2}{2l_2 l_4} - \frac{l_1}{l_4} c\theta_2 - \frac{l_1}{l_2} c\theta_4 + \left(c\theta_2 c\theta_4 + s\theta_2 s\theta_4\right) = 0 ,$$

que puede reescribirse como

$$k_1 - k_2 c\theta_2 - k_3 c\theta_4 + \left(c\theta_2 c\theta_4 + s\theta_2 s\theta_4\right) = 0,$$

(4.22)

en donde

$$\boxed{k_1 = \frac{l_1^2 + l_2^2 + l_4^2 - l_3^2}{2 l_2 l_4}},$$

(4.23)

$$\boxed{k_2 = \frac{l_1}{l_4}},$$

(4.24)

$$\boxed{k_3 = \frac{l_1}{l_2}}.$$

(4.25)

Si en la ecuación (4.22) se sustituyen el coseno y el seno de θ_2 por las siguientes identidades trigonométricas:

$$\text{sen}\theta = \frac{2\tan\left(\theta/2\right)}{1+\tan^2\left(\theta/2\right)}, \qquad \cos\theta = \frac{1-\tan^2\left(\theta/2\right)}{1+\tan^2\left(\theta/2\right)},$$

y además se realiza una manipulación inteligente de los términos, resulta que se obtiene una expresión del tipo

$$A \cdot \tan^2\left(\theta_2/2\right) + B \cdot \tan\left(\theta_2/2\right) + C = 0,$$

(4.26)

en la cual puede aprovecharse la expresión general para resolver ecuaciones de segundo grado, de modo que θ_2 estaría dado por

$$\boxed{\theta_{2_{1,2}} = 2 \cdot ang\tan\left(\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}\right)},$$

(4.27)

en donde

$$\boxed{A = k_1 + k_2 - \left(1 + k_3\right) c\theta_4},$$

(4.28)

$$\boxed{B = 2 s\theta_4},$$

(4.29)

$$\boxed{C = k_1 - k_2 + (1 - k_3)\mathrm{c}\theta_4}. \tag{4.30}$$

Si ahora en la ecuaciones (4.20) y (4.21) en lugar de aislar los términos que contienen θ_3, se aíslan los que tienen θ_2, resulta entonces que

$$-l_1 + l_3\mathrm{c}\theta_3 + l_4\mathrm{c}\theta_4 = -l_2\mathrm{c}\theta_2, \tag{4.31}$$

$$l_3\mathrm{s}\theta_3 + l_4\mathrm{s}\theta_4 = -l_2\mathrm{s}\theta_2. \tag{4.32}$$

Siguiendo un procedimiento completamente similar al seguido para la obtención de θ_2, se tiene que finalmente θ_3 está dado por

$$\boxed{\theta_{3_{1,2}} = 2 \cdot angtan\left(\frac{-E \pm \sqrt{E^2 - 4DF}}{2D}\right)}, \tag{4.33}$$

en donde

$$\boxed{D = k_4 + k_5 - (1 + k_6)\mathrm{c}\theta_4}, \tag{4.31}$$

$$\boxed{E = 2\mathrm{s}\theta_4}, \tag{4.31}$$

$$\boxed{F = k_4 - k_5 + (1 - k_6)\mathrm{c}\theta_4}. \tag{4.31}$$

A su vez las constantes k_4, k_5, y k_6 están dadas por

$$\boxed{k_4 = \frac{l_1^2 + l_3^2 + l_4^2 - l_2^2}{2l_3l_4}}, \tag{4.31}$$

$$\boxed{k_5 = \frac{l_1}{l_4}}, \tag{4.32}$$

$$\boxed{k_6 = \frac{l_1}{l_3}}. \tag{4.33}$$

Como lo que interesa obtener para el caso de la falangina y de la falangeta, son los ángulos relativos, es decir α y β, se tiene entonces que

$$\boxed{\alpha = \theta_3 - \theta_4}, \tag{4.34}$$

$$\boxed{\beta = \theta_2 - \theta_3}. \tag{4.35}$$

En el caso del metacarpo, los ángulos se calculan de la misma forma como se hizo para la falange de los otros dedos, por lo que si

$$\boxed{\delta = ang\,\mathrm{sen}\left(\frac{p_{y'} - q_{y'}}{l_1}\right)} \tag{4.36}$$

entonces

$$\boxed{\gamma = \theta_2 + \delta}, \tag{4.37}$$

que es el ángulo que formaría el metacarpo con el plano x'-y'. Así mismo el ángulo que forma el plano que contiene a todo el pulgar con el plano y'-z' estaría dado por

$$\boxed{\varepsilon = ang\,\mathrm{sen}\left(\frac{q_{x'} - p_{x'}}{l_1}\right)}. \tag{4.38}$$

Como puede verse, en el pulgar también se presenta la posibilidad de varias soluciones, sin embargo, éstas se ven restringidas dados los límites que se establecieron para el movimiento de los elementos.

Además de lo anterior, en el caso de los otros dedos bastaba con fijar una posición en el espacio, para que pudieran calcularse todos los ángulos que permitieran al dedo alcanzarla. Para el caso del pulgar, como se ha mencionado, la situación se ve afectada por el hecho de que se cuenta con un grado extra de libertad, por lo que en esta ocasión además de proporcionar la posición que tiene que ser alcanzada, es necesario establecer de antemano la orientación de uno de los segmentos. La orientación que resulta de mayor interés, es la de la falangeta, ya que con ésta es con la que generalmente se hace contacto con algún objeto de manipulación. De hecho, esto fue lo que se consideró para llevar a cabo los cálculos, ya que si se pone atención en éstos, para poder encontrar una solución completa se requiere conocer el valor de $\theta 4$, que corresponde precisamente a la orientación de la falangeta.

Cabe señalar que el método que se empleó para la obtención de los ángulos se emplea con frecuencia para la resolución de mecanismos de cuatro barras, y es conocido como el *método de Freudenstein*.

4.2 Generación de trayectorias

Como se dijo al comienzo del capítulo, entre los problemas asociados al hecho de que un manipulador se mueva describiendo una determinada trayectoria, se encontraba el de generar propiamente dicha curva. El problema de la generación de trayectorias, es generalmente un problema de interpolación, ya que en muchas ocasiones se desea que la curva se obtenga a partir de un cierto número de puntos, llamados *puntos de control*, los cuales se pretende que pertenezcan a ella, o que por lo menos sean cercanos.

En el caso de que se requiera una interpolación exacta de los puntos de control, puede recurrirse a una función de tipo polinomial de grado igual al número de puntos menos uno. A pesar de que con este método se obtiene una sola curva que pasa por todos los puntos, en ocasiones resulta inconveniente cuando éstos son demasiados, ya que, como se mencionó, el grado del polinomio depende del número de puntos. Además de esto, el hecho de que la curva se construya a partir de un polinomio cuyo grado es demasiado alto, hace que se introduzca una especie de ruido en forma de oscilaciones provocando que la curva no sea lo suficientemente suave.

Una manera de evitar estos problemas, es generar la trayectoria mediante una curva segmentada. Esto ofrece la ventaja de que cada uno de dichos segmentos se obtiene a partir de un conjunto de funciones polinómicas que por lo general son de tercer grado (aunque pueden ser de grados superiores), y cuyo manejo es mucho más simple, sobre todo cuando se trata de un gran número de puntos de control. No obstante las ventajas que posee el generar una curva por segmentos, la mayoría de los métodos que así lo hacen tienen el inconveniente de que no interpolan exactamente los puntos de control. Es precisamente por esto que se mencionaba al comienzo de esta sección, que estos últimos podían o no pertenecer a la trayectoria generada.

Para este proyecto se decidió emplear un método de generación de curvas por segmentos debido a las razones expuestas anteriormente, es decir, por la dificultad que implica el hecho de que el grado del polinomio que genera la curva sea dependiente del número de puntos de control, y por las irregularidades que podrían presentarse en la curva de utilizarse polinomios de orden muy superior. De esta manera, aun cuando se sabe de antemano que los puntos no pueden interpolarse, por lo menos se garantiza que con el uso de este tipo de procedimientos puede obtenerse trayectorias suaves. Existen varios métodos que permiten crear trayectorias segmentadas, entre los cuales se encuentra el de las curvas B-spline.

4.2.1 Curvas B-spline uniformes

El método de curvas B-spline (ver Figura 4.5) permite generar trayectorias segmentadas en el espacio tridimensional a partir de una serie de puntos de control. Como se dijo, hay otros métodos que son similares, sin embargo, éste en particular ofrece una ventaja sobre los demás, y es el hecho de que los puntos de control ejercen una influencia muy localizada sobre el trazado de la curva, es decir, si se modifican las coordenadas de alguno de los puntos de control, solamente se afecta

un número reducido de segmentos de dicha curva. Este aspecto quedará mucho más claro a medida que se avance en el desarrollo del capítulo.

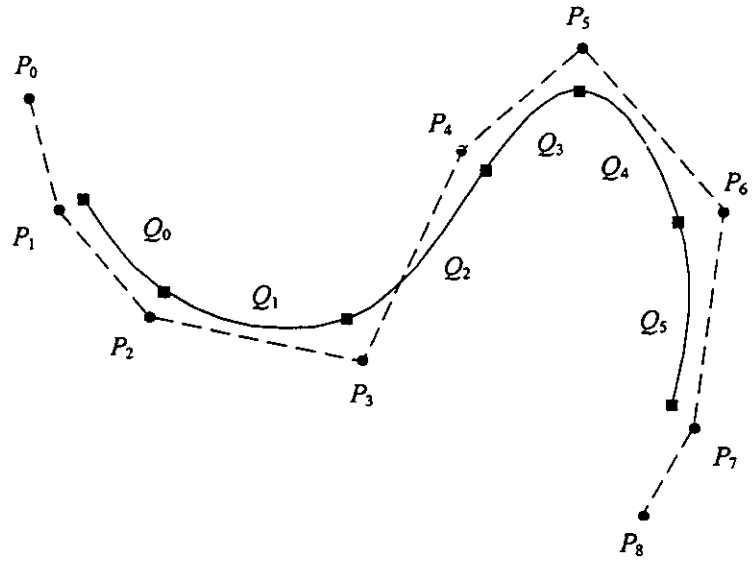

Figura 4.5 Curva B-spline cúbica de seis segmentos definida por nueve puntos de control.

La manera como matemáticamente se define un segmento de una curva B-spline es la siguiente:

$$Q_k(u) = \frac{1}{6}\begin{bmatrix} P_{k_x} & P_{(k+1)_x} & P_{(k+2)_x} & P_{(k+3)_x} \\ P_{k_y} & P_{(k+1)_y} & P_{(k+2)_y} & P_{(k+3)_y} \\ P_{k_z} & P_{(k+1)_z} & P_{(k+2)_z} & P_{(k+3)_z} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix}, \tag{4.39}$$

en donde el primer término matricial corresponde a las coordenadas en forma de columna de cuatro puntos de control sucesivos. El segundo y tercer términos matriciales, así como el escalar, corresponden a las llamadas funciones base, las cuales permiten llevar a cabo la interpolación de los puntos que integran a la curva. De una manera explícita, las funciones base estarían dadas por

$$\begin{aligned} B_1(u) &= \tfrac{1}{6}u^3 \\ B_2(u) &= \tfrac{1}{6}\left(-3u^3 + 3u^2 + 3u + 1\right) \\ B_3(u) &= \tfrac{1}{6}\left(3u^3 - 6u^2 + 4\right) \\ B_4(u) &= \tfrac{1}{6}\left(-u^3 + 3u^2 - 3u + 1\right) \end{aligned}, \tag{4.40}$$

en donde u es un parámetro local que varía desde 0 hasta 1. Lo que indica la palabra local en este caso, es que la variación de este parámetro sirve únicamente para la generación de un segmento de la curva, y no para la curva completa.

Como puede apreciarse, las funciones base no son más que polinomios de tercer grado (aunque podrían ser de grado superior) que son válidos para el rango de valores establecido para u (ver Figura 4.6), y que de hecho ellos mismos se constituyen en segmentos de una curva de tercer grado uniforme (ver Figura 4.7), que tiene como característica el ser diferente de cero a lo largo de todo del rango excepto en los extremos.

Además de esto, vale la pena aclarar que cada una de las funciones en (4.40), se ocupa de la interpolación de un sólo punto del conjunto de cuatro que se tienen para la generación de un segmento, es decir, la función $B_1(u)$ se encarga de escalar al cuarto punto, $B_2(u)$ se encarga del tercer punto, $B_3(u)$ del segundo punto y finalmente $B_4(u)$ del primer punto de control.

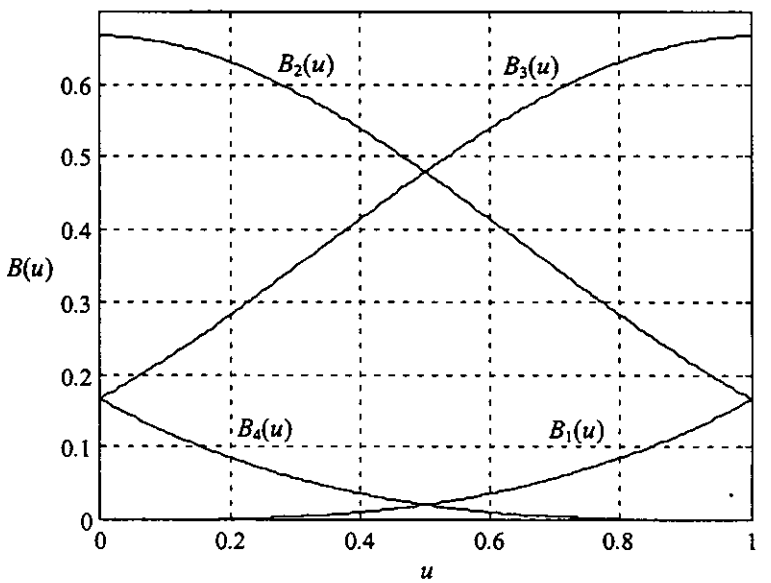

Figura 4.6 Funciones base $B_i(u)$ valuadas en el rango $0 \leq u \leq 1$.

De la expresión (4.39) puede notarse que cada uno de los segmentos está definido por cuatro funciones base y por cuatro puntos de control. Esta es la razón por la que el número de funciones base compuestas y de puntos de control, supera por tres unidades al número de segmentos de curva, como ocurre en la Figura 4.5, en donde se tiene una curva de seis segmentos generada a partir de nueve puntos de control.

De la misma manera, puede verse que a medida que en la expresión (4.39) se van recorriendo los puntos de control, cada uno de éstos va interviniendo en la interpolación de cuatro segmentos de curva. Esto trae como consecuencia que si por alguna razón se modifican las coordenadas de un punto de control, únicamente se verán afectados cuatro segmentos. Es precisamente esta característica, la que se

constituye como una de las principales ventajas de este método, y que al comienzo de la sección se refería como la influencia localizada que los puntos de control ejercen sobre la trayectoria. Además de esto, el hecho de que los puntos de control sean compartidos por los segmentos de la curva, hace que la continuidad entre éstos se mantenga.

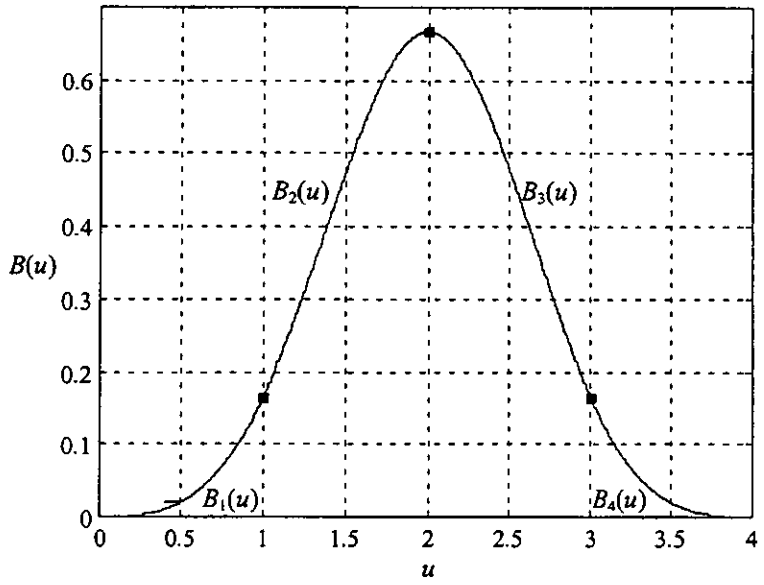

Figura 4.7 B-spline cúbica uniforme.

La manera como se va dando el proceso de interpolación está representada en la Figura 4.8, la cual plantea la generación de una curva B-spline de tres segmentos a partir de seis puntos de control. Como se mencionó anteriormente, se requiere por cada punto de control de una función polinómica compuesta (como la mostrada en la Figura 4.7). Éstas se van colocando a intervalos iguales al rango de valores de u. Al punto que en la gráfica marca el comienzo o el fin de un intervalo se conoce como *nodo*. Además, si como en este caso, existe la misma separación entre nodos, se dirá entonces que se trata de una *curva B-spline uniforme*.

Aunque en las Figuras (4.7) y (4.8) aparece u como si se tratara de un parámetro global, el hecho es que las funciones son válidas para el intervalo en las que se definieron, es decir, entre cero y uno; sólo que se ponen de esta manera para ejemplificar su funcionamiento de una forma más clara.

Regresando a la Figura 4.8, puede observarse que al colocar las funciones compuestas a un cierto intervalo, ocurre un traslape de las mismas. Si se lleva a cabo la suma de dichas funciones, puede notarse que se obtiene un valor de uno para el rango entre 3 y 6, lo cual representa tres intervalos de u, que corresponden a los tres segmentos de curva que pueden generarse con el número propuesto de puntos de control. Una consecuencia que se deriva de lo anterior, es que cada uno de los segmentos se halla contenido en la parte "interior" del polígono definido por los puntos de control que intervienen en su generación.

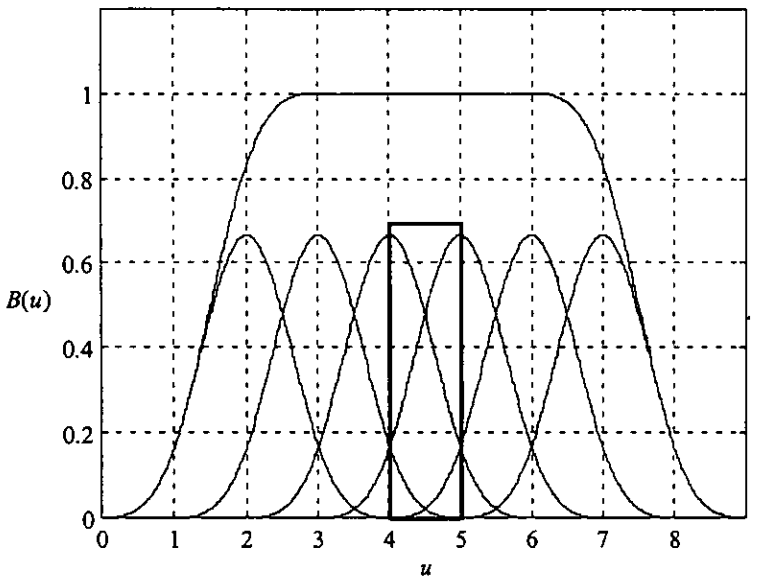

Figura 4.8 B-splines que se emplearían en la construcción de una curva de tres segmentos.

Considerando un sólo segmento, como se ha visto, se tiene que éste queda definido por un intervalo de valores de u en el que la suma de las funciones base da como resultado la unidad. En la figura, la sección sombreada corresponde precisamente a este intervalo. En él puede comprobarse el comportamiento de las funciones base a medida que u va cambiando. De esta manera puede decirse que en general, para valores de u que no corresponden a los de los nodos, cuatro funciones base se encuentran activas y suman uno. Cuando u toma el valor de alguno de los nodos, entonces una de las funciones base se "apaga", mientras que otra se "enciende" inmediatamente.

Haciendo referencia a lo que se había mencionado anteriormente, las curvas B-spline no interpolan de forma "natural" a ningún punto de control. A pesar de esto, puede hacerse que una curva de este tipo los interpole si al momento de realizar los cálculos se utiliza varias veces el punto de control de interés.

De manera intuitiva se puede pensar que un punto de control ejerce una influencia mayor si éste se repite. Como se vio, un segmento se construye haciendo que una serie de funciones base escalen a los puntos de control. Si uno de éstos se repite, entonces será utilizado más de una vez en la evaluación del segmento, haciendo que la curva sea atraída hacia dicho punto.

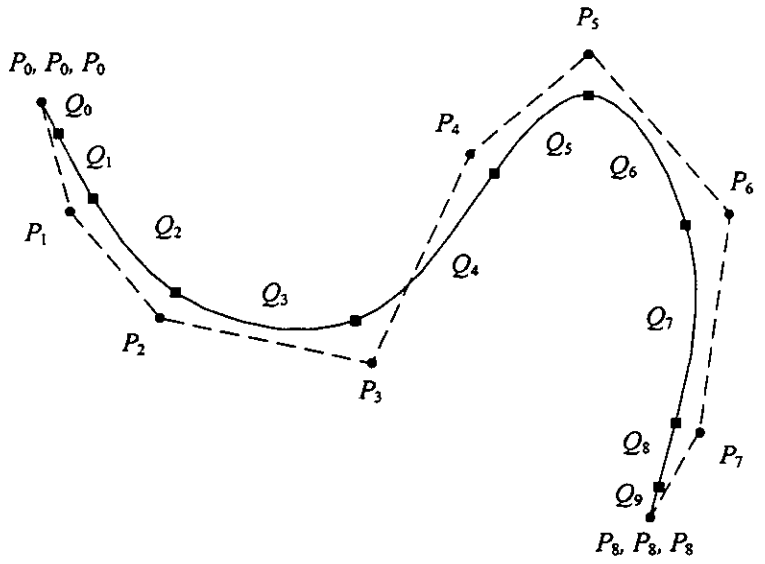

Figura 4.9 Efecto de la multiplicidad de los puntos de control extremos.

Este procedimiento puede emplearse para interpolar puntos tanto intermedios como extremos. Sin embargo, el hecho de emplear un punto de control en varias ocasiones, puede acarrear problemas de continuidad, por lo que si de todos modos se usa, es preferible aplicarlo a los puntos extremos. Por ejemplo en la Figura 4.9, los puntos de control de los extremos son empleados tres veces cada uno en la evaluación de la curva, con lo cual se logra que esta última interpole exactamente a dichos puntos.

Además de las curvas B-spline uniformes, existen las que no son uniformes, y también pueden ser empleadas para interpolar puntos de control. Su no uniformidad consiste en que el espaciamiento de los nodos puede ser diferente requiriendo de algoritmos especiales para poder generar las funciones base. Debido a que el método de curvas B-spline uniformes cumple bastante bien con los objetivos primarios de este trabajo, únicamente se dejará en mención el método de las B-spline no uniformes.

CAPÍTULO 5

Programación del simulador e integración de los sistemas

Como se mencionó al comienzo de este trabajo, uno de los objetivos del mismo, además de llevar a cabo un análisis de la cinemática tanto directa como inversa de un manipulador antropomórfico, era el de crear un programa de cómputo en el que se pudiera llevar a cabo la simulación espacial del mecanismo; esto con el fin de proporcionar al usuario del sistema de manipulación teleoperada, un medio para retroalimentar la información en cuanto a posición, para el caso en que dicho usuario no tuviera contacto visual con el manipulador. Asimismo, otro de los objetivos era que dicho programa sirviera tanto de plataforma para el estudio de los movimientos realizados por el usuario, como de herramienta para realizar la planeación de movimientos para su posterior ejecución por parte del manipulador.

De esta manera en el programa de aplicación propuesto se pretende integrar los diferentes aspectos vistos en los capítulos anteriores, tales como son el análisis cinemático directo, que permite conocer la forma como se va moviendo el mecanismo de acuerdo con las variables de entrada, que en este caso es la posición angular relativa entre los elementos; así como el método para generar trayectorias a partir de unos cuantos puntos de control, que junto con el análisis cinemático inverso, hacen posible que los elementos del manipulador sean capaces de seguir un camino predeterminado.

Además de lo anterior, el programa debe incorporar algunas capacidades que le permitan comunicarse con los demás sistemas que integran al proyecto, es decir, debe poder recibir e interpretar la información proveniente del sistema de medición de la posición de la mano del usuario, así como procesar la información necesaria para después comunicarse con el sistema de actuación y control, proporcionándole así la información de referencia que requiere.

Para poder llevar a cabo la programación, necesariamente se requiere de un lenguaje. La mayoría de éstos son capaces de manejar los elementos que se requieren para llevar a cabo los cálculos para el manejo del espacio tridimensional, es decir, elementos como matrices y vectores. Del mismo modo, la mayoría de dichos lenguajes también soporta el empleo de gráficos (requeridos para realizar la visualización del manipulador). A pesar de esto, el lenguaje que finalmente se eligió para realizar la programación fue C++. Las razones por las cuales se decidió programar en este lenguaje estuvieron sujetas en primer término a la experiencia personal en su uso; y en segundo lugar, a las ventajas que dicho lenguaje ofrece al

ser uno de los que está orientado a objetos (cosa que facilita mucho la programación), además de la velocidad de ejecución que lo hace atractivo para desarrollar aplicaciones de ingeniería.

5.1 Programa de simulación para WIN32

Dentro del manejo del C++ existen dos posibilidades en cuanto al ambiente de ejecución; la primera corresponde a lo que se conoce como ambiente MS-DOS (16 bits de procesamiento), y la segunda está relacionada con la utilización de un entorno gráfico como lo es Windows 95 (procesamiento de 32 bits), también llamado WIN32. A fin de contar con un programa cuya capacidad para interactuar con el usuaria fuera amplia; y de aprovechar la capacidad de cómputo de los nuevos equipos que soportan procesamiento de 32 bits, se decidió en primera instancia crear una versión para ejecutarse en Windows 95. En el Apéndice B se muestra el código fuente completo de todos los archivos que integran el proyecto MAT-I.IDE y que generan el archivo ejecutable MAT-I.EXE[9].

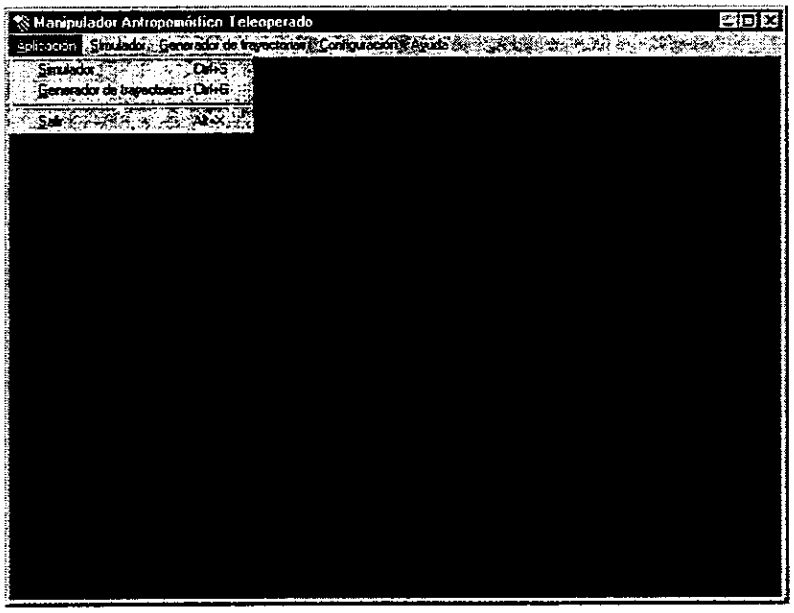

Figura 5.1 Pantalla del programa de simulación (versión WIN32).

Al iniciar la sesión, lo primero que aparece es la ventana general que administra las funciones del programa (ver Figura 5.1). En el menú de la barra superior pueden apreciarse diferentes opciones, las cuales se irán explicando conforme se vaya avanzando en el capítulo. Por lo pronto la primera de ellas: *Aplicación*, tiene por objeto el activar las dos funciones principales del programa, que en el submenú de aparición súbita están representadas por *Simulador* y *Generador de trayectorias*. Al seleccionar *Simulador*, se activa la ventana de visualización y simulación, mientras que al optar por *Generador de Trayectorias*, se activa la ventana de trazado de curvas que sirven como trayectorias para el movimiento de

[9] El programa fue compilado utilizando Borland C++ 5.

los dedos. Esta última ventana también permite simular el desplazamiento de los dedos a lo largo de la trayectoria generada.

5.1.1 Visualización y simulación

Cuando se elige la opción de *Simulador* en el submenú de *Aplicación*, aparece la ventana mostrada en la Figura 5.2, en la cual puede apreciarse una representación esquemática tridimensional del manipulador antropomórfico teleoperado. Del mismo modo, al activarse la ventana de visualización, automáticamente quedan habilitadas las opciones del submenú de *Simulador* en la barra principal. Estas opciones, que son *Ejecutar* y *Pausa/Continuar*, permiten controlar los momentos de ejecución o pausa de la simulación.

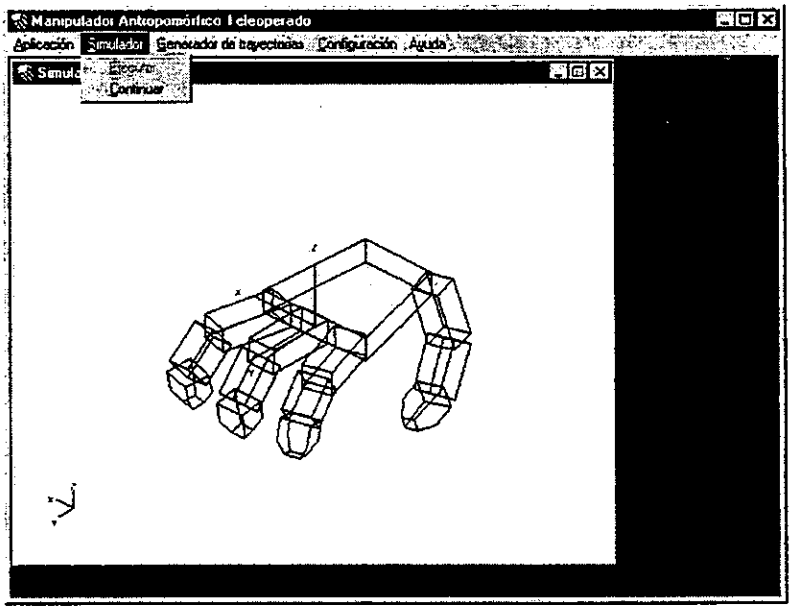

Figura 5.2 Ventana de visualización y simulación.

Para que la representación del manipulador pueda llevarse a cabo, necesariamente tiene que conocerse la configuración del manipulador como función de ciertos valores de entrada, que en este caso son las posiciones angulares relativas entre los elementos que conforman a los dedos. Precisamente a lo que esto se refiere es a la cinemática directa del manipulador, la cual ya fue revisada en el Capítulo 3.

Internamente, lo que hace la rutina que se encarga de la obtención de la cinemática directa, es tomar los valores de los ángulos relativos en las articulaciones de los dedos de la mano. Una vez obtenidos éstos, se procede a calcular las matrices de transformación para cada uno de los elementos que constituyen al manipulador, según se indica en las ecuaciones (3.22), (3.28), (3.34) y (3.39). Posteriormente, estas matrices se multiplican por las coordenadas de los vértices de los bloques que sirven para representar al mecanismo, dando como resultado las nuevas coordenadas que corresponden a la configuración actual del manipulador.

Ya que se tienen las nuevas posiciones, se manda llamar a la rutina encargada de su representación gráfica.

Tabla 5.1 Comandos adicionales para el manejo de la ventana de visualización y simulación.

Tecla	Función
F9	Inicialización de los ángulos de visión
↑	Incrementa el ángulo de elevación
↓	Decrementa el ángulo de elevación
←	Incrementa el ángulo de acimut
→	Decrementa el ángulo de acimut
F5	Rotación antihoraria del plano de proyección
F6	Rotación horaria del plano de proyección
TAB	Intercala la selección de los elementos de la mano
Q	Palma: movimiento de supinación[10] Dedos: movimiento de aducción[11]
A	Palma: movimiento de pronación[12] Dedos: movimiento de abducción[13]
W	Palma: movimiento de inclinación positiva Dedos: extensión de la falange Pulgar: extensión del metacarpo
S	Palma: movimiento de inclinación negativa Dedos: flexión de la falange Pulgar: flexión del metacarpo
E	Palma: movimiento de aducción Dedos: extensión de falangina y falangeta Pulgar: extensión de la falangina
D	Palma: movimiento de abducción Dedos: flexión de falangina y falangeta Pulgar: flexión de la falangina
R	Pulgar: extensión de la falangeta
F	Pulgar: flexión de la falangeta
ESC	Inicializa los ángulos de todos los elementos de la mano

Haciendo referencia a lo que es la representación gráfica del manipulador, además de las funciones que aparecen en el menú, se tienen algunos comandos adicionales (ver Tabla 5.1). Entre éstos, hay algunos que permiten modificar el punto de observación, lo cual es muy útil, ya que en ocasiones es difícil interpretar correctamente la posición del manipulador. También se cuenta con algunos otros comandos que hacen posible el movimiento individual de sus elementos.

Se ha mencionado que la configuración del manipulador es función de los ángulos relativos en las articulaciones de los dedos, sin embargo, estos ángulos

[10] Supinación.- Movimiento del antebrazo que hace girar la mano de tal forma que la palma queda hacia arriba.
[11] Aducción.- Movimiento por el cual se acerca una parte del cuerpo al plano medio de éste.
[12] Pronación.- Movimiento de rotación de la mano de modo que la palma queda hacia abajo.
[13] Abducción.- Movimiento por el cual se aleja un miembro del plano medio del cuerpo.

tienen que ser proporcionados de algún modo. Para llevar a cabo esto, se tienen dos posibilidades: en la primera, los ángulos de referencia se obtienen mediante una función matemática incorporada en el programa; y en la segunda, los ángulos se adquieren por vía externa, siendo precisamente los datos transmitidos por el sistema de medición de la posición de los dedos de la mano del usuario. De hecho una de las características de esta ventana consiste en que puede mantener comunicación bidireccional, ya que por un lado, es capaz de recibir la información proveniente del sistema sensor, y por otro, puede transmitir la información de ángulos generados a partir de funciones internas hacia el sistema de actuación y control. Se dice que esta ventana funciona como un elemento de visualización cuando ahí se representa la información del sistema de medición; y se dice que actúa como un elemento de simulación cuando dentro del programa se generan y representan las nuevas posiciones de los dedos de la mano para su posterior transmisión al sistema de actuación y control.

5.1.2 Comunicación con los sistemas de medición y control

Como se dijo en el primer capítulo, el proyecto MAT consta de cuatro partes principales: el sistema de medición de la posición de los dedos de la mano del usuario; el sistema de visualización y procesamiento de información; el sistema de control y actuación; y el sistema mecánico de manipulación. A pesar de que todas estas partes están relacionadas entre sí por formar parte de un sistema integral, el sistema de visualización y procesamiento de información (o simplemente programa de simulación), que es la parte tratada en este trabajo, únicamente se comunica directamente tanto con el sistema de medición, como con el sistema de control.

El sistema de medición tiene como señales de salida las posiciones angulares relativas de cada uno de los elementos que conforman a los dedos de la mano del usuario. Estas señales, de alguna manera tienen que ser "tomadas" por el programa de simulación. Del mismo modo, el sistema de control y actuación tiene como señales de entrada las posiciones de referencia que en un principio tienen que ser proporcionadas por el sistema de medición. Dado que el simulador también puede emplearse para generar y planear nuevos movimientos del manipulador, el programa tiene que ser capaz de transmitir las posiciones generadas al sistema de control.

Cuando el usuario tiene contacto visual con el manipulador, y lo que se quiere es llevar a cabo alguna tarea que no requiera algún procesamiento adicional que implique la intervención de la computadora, entonces el sistema de medición puede conectarse directamente al sistema de control. Si por el contrario, se desea hacer el análisis de los movimientos hechos por el usuario o se requiere dar un tipo de desplazamiento determinado al manipulador, entonces puede conectarse la computadora a manera de intermediario entre ambos sistemas, siendo el programa de simulación el encargado de administrar los recursos de ésta.

Lo anterior trae como consecuencia que tenga que haber una compatibilidad en lo se refiere tanto a la manera de transmitir los datos, como al protocolo de comunicación. Por lo que toca al primer punto, se decidió utilizar comunicación serial dado que todos los sistemas involucrados cuentan con elementos físicos de procesamiento que son capaces de establecer comunicación de este tipo. Por ejemplo, en el caso de los sistemas de medición y, de control y actuación, se tienen

microcontroladores HC11 que cuentan con puerto serial configurable. Por lo que refiere a la computadora, el puerto serial es un elemento que todos los equipos actuales poseen de fábrica. Cabe señalar que la comunicación física entre los elementos, se logra a través de un cable que cumple con los requerimientos establecidos para el tipo de interfaz RS-232.

Además de esto, la transmisión serial, aunque más lenta que otros modos de comunicación, es más sencilla en cuanto a la implementación tanto física como computacional. De igual manera, la comunicación serial resulta lo suficientemente satisfactoria como para cumplir los objetivos planteados en esta primera etapa del proyecto.

5.1.2.1 Configuración del puerto serial

La configuración del puerto serial en la computadora puede realizarse mediante la opción *Configuración* del menú principal, la cual por el momento sólo cuenta con la función *Configuración del puerto serie*. Al elegir ésta, aparece un cuadro de diálogo (ver Figura 5.3) que permite establecer los parámetros principales para llevar a cabo la comunicación.

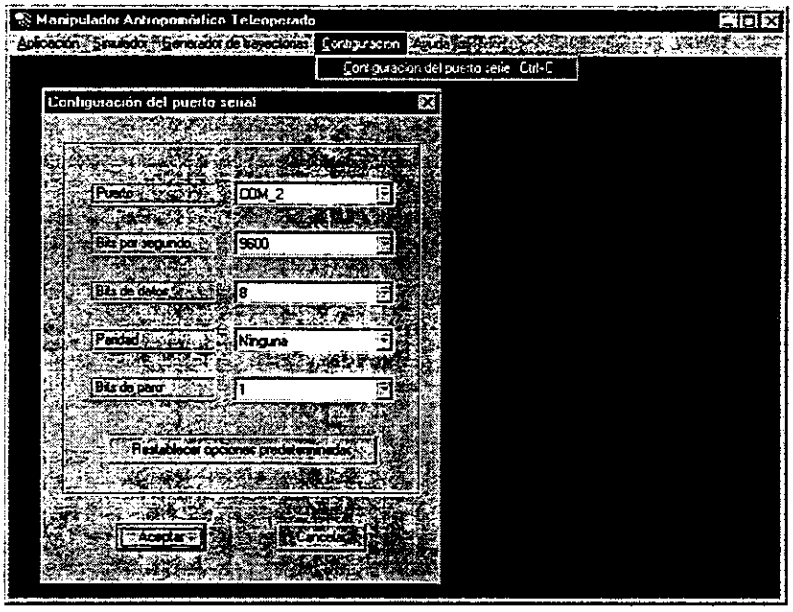

Figura 5.3 Cuadro de diálogo para la configuración del puerto serial.

A pesar de que existen diversas posibilidades para la configuración del puerto, por lo general se emplean las opciones establecidas como predeterminadas en el cuadro de diálogo, siendo éstas las que se muestran en la figura, es decir, 9600 bauds, 8 bits de datos, sin paridad y un bit de paro.

5.1.2.2 Protocolo de comunicación

Una vez que se cuenta con los medios para entablar una comunicación serial, es preciso proponer un protocolo para la transmisión de los datos. El protocolo que se utilizó para tal fin es semejante al empleado por los sistemas MIDI, y consiste básicamente en establecer bloques de información bien delimitados. Cada uno de estos bloques está formado por una cadena de bytes, la cual se caracteriza por el hecho de que el primer byte, así como el último, son bytes de los llamados de control. Por otro lado, el resto de los bytes sirve para transmitir la información sobre la posición de los elementos móviles de la mano.

La manera como se pueden distinguir los bytes de control de los de datos, es mediante la estructura de cada uno de ellos. Como se sabe, un byte está compuesto de 8 bits, lo cual permite representar en forma binaria hasta 256 números (de 0 a 255). Así pues, tomando en cuenta lo anterior y el tipo de protocolo empleado en las aplicaciones MIDI, se establece que el bit más significativo de los bytes de control esté puesto en 1, a diferencia de lo que ocurre con los bytes de datos, en los que se pide que el bit más significativo esté puesto en 0.

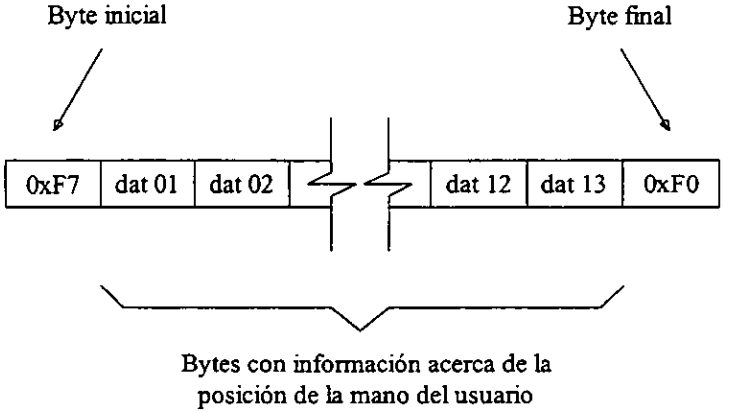

Figura 5.4 Configuración de la cadena de información.

De esta manera se tiene que el byte inicial de la cadena de información generalmente está dado por el número hexadecimal F7, mientras que el final está representado por el F0 (ver Figura 5.4). Por lo que toca a los bytes de datos ocurre que, al restringir el valor del bit más significativo, se produce una disminución del número de valores diferentes que pueden tomar, pasando de 256 a sólo 128 (de 0 a 127). A pesar de que con esto se pierde algo de resolución en el sistema de medición, el hecho de establecer una distinción tan determinante entre los bytes de datos y los de control, hace que la asignación de canales y el procesamiento de la información sean mucho más seguros.

Como se dijo, además de los bytes de control se tienen también algunos de datos, los cuales dentro de la cadena de información ascienden al número de trece. Cada uno de estos bytes corresponde a un canal por donde se transmiten las

posiciones de los trece posibles grados de libertad de la mano (sin contar los movimientos de la muñeca). Dado que por el momento los movimientos de aducción y abducción de los dedos índice, medio y anular no están contemplados en el dispositivo sensor del sistema de medición, únicamente se hace uso de diez canales, dejando los otros tres para su posterior utilización.

5.1.2.3 Manejo de la información

Una vez que existe un medio y un protocolo para realizar la comunicación, lo único que hace falta es establecer una manera para interpretar los datos. De la sección anterior se sabe que por lo que toca a los bytes de datos, que en este caso representan ángulos, el número de posibilidades está restringido a 128, partiendo desde 0 hasta 127. Estos posibles valores no corresponden directamente a los ángulos que existen en el dispositivo sensor, ya que algunos límites de movimiento son diferentes dependiendo del dedo que se esté midiendo. Es por esta razón que debe hacerse una interpolación, a fin de obtener el valor en grados del ángulo que realmente está siendo medido. La expresión empleada para esto tiene la forma siguiente:

$$y = \frac{y_2 - y_1}{x_2 - x_1}\left(x - x_1\right) + y_1 , \tag{5.1}$$

en donde y_2 y y_1 son respectivamente los ángulos máximo y mínimo permisibles para el movimiento del elemento en cuestión (ver Tablas 3.4 y 3.6); x_2 y x_1 son los valores máximo y mínimo que de manera respectiva se han registrado en el puerto hasta el momento; x es el valor actual recibido en el puerto; y finalmente y es el ángulo en grados que corresponde al valor de x y que representa el ángulo existente en la mano del usuario. Para el caso en el que, en lugar de recibir información del sistema de medición, se envían datos al sistema de control y actuación, los papeles de las variables de la expresión (5.1) se invierten, de modo que al final x se convierte en el valor del ángulo en grados que debe existir en un elemento determinado de la mano, y y se transforma en un valor entre 0 y 127 que corresponde a dicho ángulo, el cual será enviando por puerto hacia el sistema de control y actuación.

Como puede verse, la ecuación (5.1) corresponde a lo que es una interpolación lineal. Esto ocurre de esta manera porque las expresiones matemáticas que modelan el comportamiento de los dispositivos sensores, se comportan en una gran porción de la región de operación de una manera muy lineal. Si esto no ocurriera de este modo, entonces habría que emplear algún otro tipo de expresión que se ajustara más cabalmente a la conducta real del sensor para llevar a cabo la interpolación de los valores registrados en el puerto.

Se mencionó que en la interpolación intervienen los valores máximo y mínimo registrados en el puerto hasta el momento de la lectura. Dado que los datos proporcionados por el sistema de medición no siempre abarcan el rango total en cuanto al número de valores posibles y no siempre son los mismos límites superior e inferior, es preciso llevar a cabo un reajuste de éstos. La manera como funciona el

reajuste de valores es la siguiente: en un principio, antes de recibir cualquier dato en el puerto serial, los límites se hallan invertidos, es decir, el límite superior se establece en el lugar del límite inferior y viceversa. La programación está hecha de tal modo que si un valor de los que se reciben en el puerto excede al límite superior, o es menor que el inferior, y además se halla activo el modo de calibración, entonces ocurre que dicho valor se convierte en el nuevo límite, ya sea superior o inferior. Así pues, al estar los límites invertidos, y al funcionar el programa del modo como que se explicó, se lleva a cabo una modificación de los límites que garantiza el máximo aprovechamiento del rango disponible. Dado que el sistema de medición realiza un tratamiento previo de la información que hace que el rango de valores que transmite se halle entre 0 y 127, se tiene la certeza de que éstos jamás excederán los límites máximos. También vale la pena destacar que el ajuste de límites se realiza por canal de manera independiente.

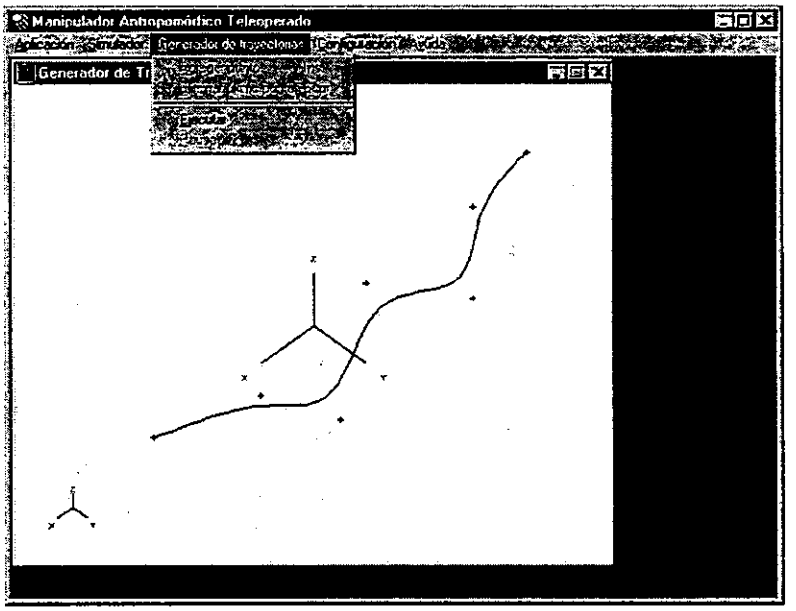

Figura 5.5 Ventana de generación de trayectorias.

5.1.3 Generación de trayectorias

Siguiendo con la explicación del programa MAT-I.EXE, toca el turno a la segunda opción del submenú de *Aplicación*. Al seleccionar de éste: *Generador de Trayectorias*, además de aparecer la ventana mostrada en la Figura 5.5, quedan activadas las funciones de la opción del menú principal de la barra superior *Generador de trayectorias*. Las funciones que ahí pueden apreciarse son: *Agregar punto*, *Eliminar último punto*, *Ejecutar* y *Pausa/Continuar*.

De acuerdo con lo dicho al comienzo de la sección, la ventana de generación de trayectorias tiene el propósito de, a partir de una serie de puntos proporcionados por el usuario, crear la curva que sirva como trayectoria para el movimiento de los dedos de la mano mecánica, a fin de lograr en un futuro la consecución de tareas coordinadas de manipulación.

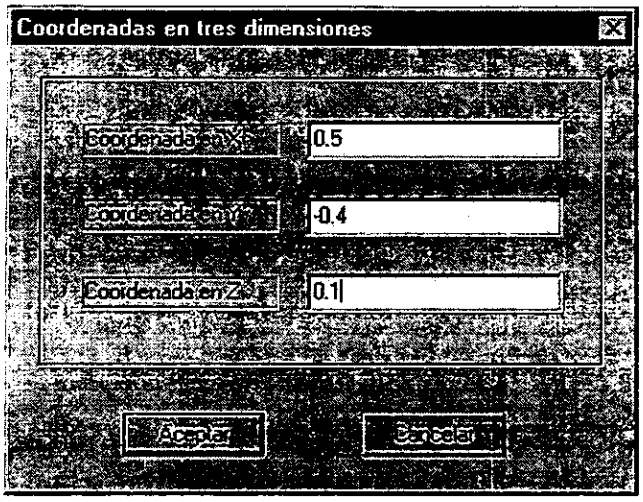

Figura 5.6 Cuadro de diálogo para la adquisición de las coordenadas de nuevos puntos de control.

Figura 5.7 Cuadro de diálogo para la eliminación de puntos de control.

El método que se emplea para generar las trayectorias es el que se basa en la utilización de las curvas B-spline de tercer grado, el cual ya fue revisado en el Capítulo 4. Este método requiere de un conjunto de puntos llamados puntos de control, que crean una especie de polígono que dan la idea básica acerca de la forma final de la curva. La manera como se introducen al programa las coordenadas de aquellos puntos que funcionan como puntos de control, es por medio de la función *Agregar punto*. Al activar ésta, aparece el cuadro de diálogo mostrado en la Figura 5.6. Si por alguna razón se desea borrar las coordenadas del último punto introducido, basta con seleccionar la función *Eliminar último punto* (ver Figura 5.7), la cual permite borrar la secuencia de puntos introducidos desde el último hasta el primero, siempre y cuando no se activado el trazado de la trayectoria.

Una vez que se han introducido todos los puntos de interés, puede iniciarse el cálculo y el trazado de la curva, lo cual se hace presionando el botón derecho del ratón. Según se explicó en el Capítulo 4, las curvas B-spline no interpolan exactamente los puntos de control, sin embargo, mediante la multiplicidad de éstos podía conseguirse que en zonas donde la continuidad y la suavidad de la curva no fuera crítica, se pudiera alcanzar exactamente el punto repetido. De hecho los

puntos en donde esto puede realizarse sin afectar las características anteriores, son precisamente los puntos de partida y de llegada, con lo que puede tenerse control de los lugares donde comienza y termina el movimiento del manipulador. Por esta razón dentro del programa, tanto al primer punto, como al último, automáticamente se les asigna una multiplicidad de tres, de modo que al momento de generar y trazar la trayectoria, éstos son interpolados con precisión.

Del mismo modo que la ventana de visualización y simulación, la ventana de generación de trayectorias posee algunos comandos extra que permiten modificar las condiciones de visualización (ver Tabla 5.2), así como también algunas funciones asociadas a los botones del ratón, como por ejemplo la introducción de puntos mediante el puntero del ratón, o la activación del cálculo y el trazado de la curva.

Tabla 5.2 Comandos adicionales para el manejo de la ventana de generación de trayectorias.

Tecla	Función
F9	Inicialización de los ángulos de visión
↑	Incrementa el ángulo de elevación
↓	Decrementa el ángulo de elevación
←	Incrementa el ángulo de acimut
→	Decrementa el ángulo de acimut
F5	Rotación antihoraria del plano de proyección
F6	Rotación horaria del plano de proyección
B. DER. RATÓN	Introduce la coordenada actual del puntero del ratón como punto sobre el plano *x-y*
B. IZQ. RATÓN	Suspende la introducción de puntos y traza la curva correspondiente

Además de las características anteriormente mencionadas, la ventana tiene la capacidad de correr una simulación de un dedo tratando de seguir la curva recién trazada. El control de la simulación se realiza mediante las opciones *Ejecutar* y *Pausa/Continuar* del submenú de *Generación de trayectorias*. Un ejemplo de este tipo de simulación puede verse en la Figura 5.8.

La manera como funciona el programa es la siguiente: ya que han sido introducidos los puntos de control, se genera la curva utilizando la expresión (4.39), en donde cada uno de los puntos que la conforman es almacenado en memoria. En el momento que se activa la ejecución de la simulación, se van tomando cada uno de los puntos de la curva, para los cuales se resuelve el problema de cinemática inversa, dando como resultado los ángulos relativos que deben existir entre los elementos constitutivos del dedo, a fin de alcanzar la posición requerida. La expresión a emplear, depende del dedo para el cual se pretenda resolver el problema, ya que puede ser para cualquiera de los dedos índice, medio y anular, o para el dedo pulgar. En el Capítulo 4 puede verse de una forma más detallada el proceso para determinar la cinemática inversa de los dedos del manipulador.

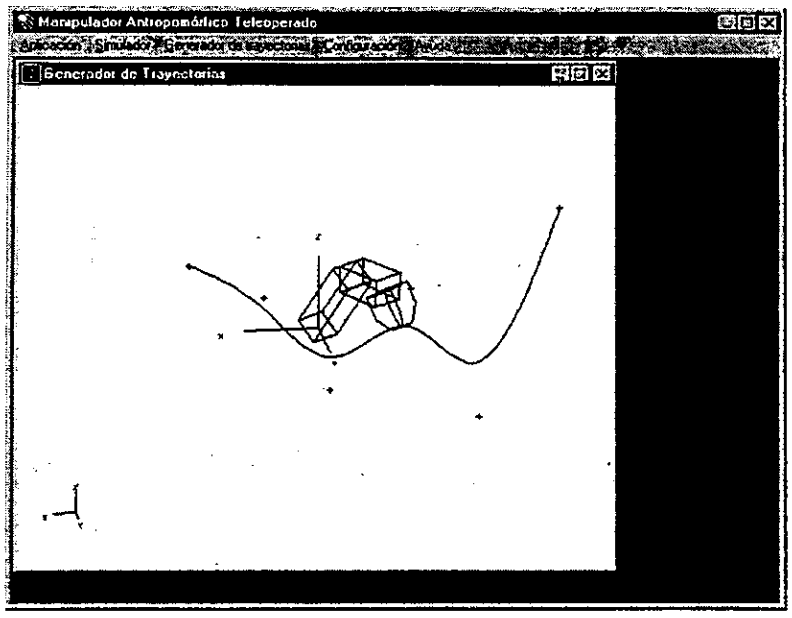

Figura 5.8 Simulación de un dedo siguiendo una trayectoria.

La información que se genera acerca de los ángulos relativos en los dedos, como en el caso de la ventana de simulación y visualización, puede ser enviada a través del puerto serial al sistema control y actuación a fin de reproducir los movimientos en el sistema mecánico de manipulación. Finalmente, para concluir la ejecución del programa basta con seleccionar la opción *Salir* del submenú de *Aplicación*, o presionar Alt-X.

5.2 Programa de simulación para MS-DOS

Además del programa diseñado para operar bajo el ambiente gráfico de 32 bits de Windows 95, se decidió crear un programa similar que se ejecutara bajo el ambiente de 16 bits de MS-DOS. La razón por la cual se tomo esta decisión, estuvo ligada al hecho de que existieron algunas dificultades en el primer programa, sobre todo en la parte de visualización.

El ambiente WIN32 tiene como una de sus características el tratar de emular a los sistema multitarea. Esto quiere decir que el tiempo de procesamiento se reparte entre varias actividades, algunas de las cuales no son tan evidentes como para que el usuario pueda percatarse de ellas. Esta repartición del tiempo de proceso trae a su vez como consecuencia que la ejecución de un programa muchas veces no sea continua. Este aspecto de la operación del Windows 95 fue la principal causa de que hayan existido problemas en la parte de visualización.

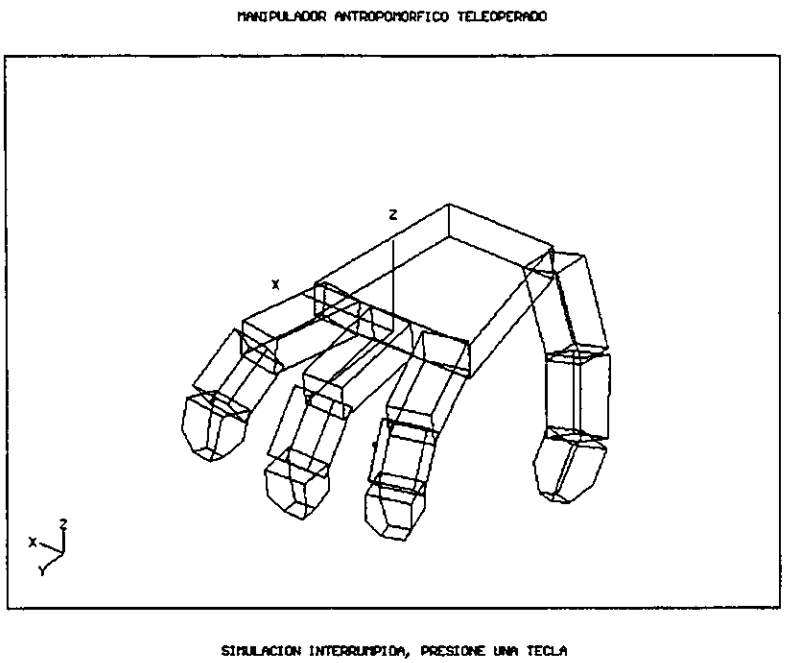

Figura 5.9 Pantalla del programa de simulación (versión MS-DOS).

Al acoplar un osciloscopio a las terminales del puerto de la computadora, podía apreciarse que entre lectura y lectura de los datos enviados por el sistema de medición había un tiempo de aproximadamente 50 (ms). Además de este retraso, debe considerarse el tiempo de lectura, que para una transmisión a 9600 bauds es de 15.6 (ms); así como el tiempo de procesamiento de la computadora, que en este caso no es tan significativo, ya que dependiendo de la velocidad del equipo llega a ser de 3.5 (ms). Lo anterior aunado al tiempo de lectura y procesamiento del microcontrolador del sistema sensor, que llega a ser de aproximadamente 11.6 (ms), da como resultado que el tiempo entre despliegues de la mano sea de cerca de 80.7 (ms), que a su vez da una frecuencia de muestreo de 12.4 (Hz). Esta frecuencia está por debajo del mínimo propuesto para aplicaciones que pretendan ejecutarse en tiempo real, que es de 14 (Hz). A todo esto hay que agregar que el sistema de medición no espera a que haya sido recibida toda la información enviada, sino que continuamente manda los datos registrados en el dispositivo sensor. Como la computadora tarda más tiempo en procesar un bloque de información, de lo que se tarda el sistema de medición en enviar nueva información, necesariamente se tiene una pérdida de información.

Todo lo anterior se ve reflejado en la pantalla de visualización como un defasamiento entre el movimiento de la mano del usuario y la imagen desplegada, además de una discretización perceptible de los movimientos.

Cuando un programa se ejecuta bajo el ambiente de MS-DOS, lo que ocurre es que el procesador se dedica casi por completo a atender las necesidades del programa. Fue que aprovechando esta circunstancia, se decidió crear una versión para DOS del programa de visualización, empleando nuevamente como lenguaje de programación C++. El listado completo del código fuente del programa DOSMAT-I.EXE se presenta en el Apéndice C.

La pantalla que aparece en el momento de correr el programa es muy similar a la de la versión para WIN32 (ver Figura 5.9). De hecho, los algoritmos para el cálculo de la cinemática directa, el tipo de comunicación incluyendo, y el protocolo, son exactamente los mismos.

El funcionamiento del programa también es similar y se da de la siguiente manera: los datos provenientes del sistema de medición se leen a través del puerto serial de la computadora. Inmediatamente después de iniciar la operación del programa no se cuenta con limites para establecer el rango de operación, por lo cual se debe comenzar con el proceso de calibración. Mediante este proceso los límites para los movimientos, así como la resolución de los mismos, se van ajustando a medida que se van recibiendo datos. La estabilización en el comportamiento de la imagen del manipulador es una señal de que los límites ya no están sufriendo modificaciones, por lo que puede desactivarse la función de calibración.

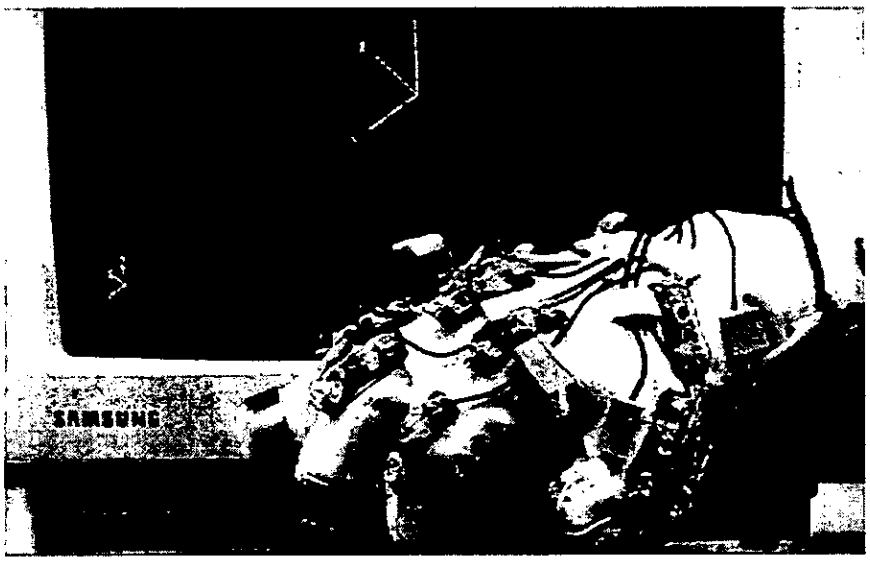

Foto 5.1 Simulador acoplado al guante sensor.

En ocasiones el sistema de medición puede tener algunas variaciones en sus lecturas, por lo que también puede presentarse una desajuste de los límites. Si esto ocurre, basta con activar nuevamente la rutina de calibración. De hecho, la calibración puede llevarse a efecto las veces que sea necesaria.

Además de la calibración, el programa cuenta con una opción que permite almacenar secuencias de movimientos realizados por el usuario, las cuales pueden ser reproducidas en algún momento posterior. Este tipo de funciones no sólo habilita al programa como un medio de entrenamiento para la operación del manipulador, sino que también puede servir como plataforma para el estudio de los movimientos de los dedos de la mano, de manera que en un futuro puedan incorporarse como funciones teóricas que intervengan en los algoritmos de manipulación coordinada.

Al igual que en el otro programa, éste posee funciones de visualización y control que hacen más flexible la operación del programa. En la Tabla 5.3 puede consultarse el resumen de estos comandos.

Tabla 5.3 Comandos adicionales para el manejo de la pantalla de visualización.

Tecla	Función
F11	Inicialización de los ángulos de visión
I	Incrementa el ángulo de elevación
K	Decrementa el ángulo de elevación
J	Incrementa el ángulo de acimut
L	Decrementa el ángulo de acimut
F5	Rotación antihoraria del plano de proyección
F6	Rotación horaria del plano de proyección
TAB	Intercala la selección de los elementos de la mano
Q	Palma: movimiento de supinación Dedos: movimiento de aducción
A	Palma: movimiento de pronación Dedos: movimiento de abducción
W	Palma: movimiento de inclinación positiva Dedos: extensión de la falange Pulgar: extensión del metacarpo
S	Palma: movimiento de inclinación negativa Dedos: flexión de la falange Pulgar: flexión del metacarpo
E	Palma: movimiento de aducción Dedos: extensión de falangina y falangeta Pulgar: extensión de la falangina
D	Palma: movimiento de abducción Dedos: flexión de falangina y falangeta Pulgar: flexión de la falangina
R	Pulgar: extensión de la falangeta
F	Pulgar: flexión de la falangeta
F12	Inicializa los ángulos de todos los elementos de la mano
C	Inicia o interrumpe la secuencia de calibración
G	Inicia o interrumpe el almacenamiento de la secuencia de movimientos del usuario
P	Reproduce la secuencia almacenada
ESC	Termina la ejecución del programa

Por lo que toca a la mejora en cuanto a la frecuencia de muestreo, el tiempo de retraso que existía en el caso del programa para Windows se eliminó por completo. En cambio el tiempo de procesamiento aumentó al doble, por lo que ahora es de aproximadamente 7 (ms). Aun con el incremento en el tiempo de proceso, el tiempo total entre despliegues de la pantalla se redujo a cerca de 34.2 (ms), con lo que la frecuencia de muestreo queda finalmente en 29.3 (Hz), que en un valor bastante aceptable, tomando en cuenta la etapa en la que se encuentra el proyecto. Como resultado de esta mejora puede apreciarse en la Foto 5.1 al sistema sensor actuando conjuntamente con el programa de visualización. A pesar del éxito logrado en las pruebas, la frecuencia de muestreo puede ser todavía mejorada, sobre todo si

se trabaja en lo referente a la velocidad de transmisión, y en la parte del procesamiento del sistema de medición.

CAPÍTULO 6

Conclusiones

De acuerdo con los objetivos planteados desde el comienzo de este trabajo, y haciendo referencia a los resultados obtenidos durante el desarrollo de las diferentes partes que integran a este módulo, pueden plantearse una serie de conclusiones muy concretas.

De las conclusiones más importantes que pueden destacarse de este trabajo, está en primer lugar el que se demostró la utilización, la efectividad, y la sencillez para implementar un modelo teórico que permite describir el comportamiento cinemático de un sistema mecánico mediante el uso del producto de exponenciales; convirtiéndose en una opción bastante atractiva en comparación con otros métodos similares, tales como el de Denavit-Hartenberg.

Otro aspecto importante es que se sentaron los fundamentos para la creación de una plataforma para el estudio y simulación no sólo del proyecto del manipulador antropomórfico teleoperado, sino para cualquier tipo de sistema mecánico. Además, vale la pena mencionar que las herramientas analíticas vistas en el presente trabajo para llevar a cabo la descripción cinemática del manipulador, pueden tener aplicación en casos que pudieran tener a simple vista poca relación. De hecho actualmente se trabaja en un proyecto que incluye un programa de simulación para el movimiento de un robot guiado automáticamente, el cual hace uso extensivo de la teoría empleada en el manipulador antropomórfico.

Un punto fundamental que está ligado al aspecto académico, es que con el desarrollo del proyecto en su conjunto, se establece un precedente y una serie de líneas de investigación relacionadas con los elementos que intervienen en los sistemas mecatrónicos, que favorecen el crecimiento de esta disciplina dentro de la Facultad de Ingeniería. Esto sin contar la formación de personal especializado en este tipo de proyectos, que por cierto cada vez son más frecuentes.

Como se dijo desde un principio, el objetivo primordial del proyecto era el de poner en funcionamiento cada uno de los sistemas involucrados, a fin de detectar todos aquellos aspectos en donde pudiera caber la posibilidad de implementar algún tipo de mejora. Como era de esperarse, el sistema de visualización y procesamiento no quedó exento de esta situación, por lo que a continuación se mencionan una serie de elementos que pueden contribuir en las siguientes etapas de desarrollo al aumento de las capacidades y la eficiencia del sistema.

En primer término y desde el punto de vista de análisis teórico, es posible determinar el modelo dinámico de cada uno de los dedos con la ayuda de los Jacobianos obtenidos. Con esto, y con la integración de un control digital, las

señales de salida corresponderían a los torques que habría que aplicar en cada una de las articulaciones. Por otro lado, la profundización en los aspectos teóricos llevaría eventualmente al campo de la manipulación coordinada de los dedos, con lo que se podría dotar a manipuladores futuros con un cierto grado de destreza y habilidad.

Otra situación que podría ser interesante, es el empleo de paquetería tal como MATLAB y SIMULINK que ofrecen un lenguaje de programación, que en comparación con otros que se utilizan con frecuencia en aplicaciones de ingeniería, es mucho más simple y práctico. Además, este tipo de paquetes tiene la ventaja de que está enfocado a casos de simulación haciendo uso extensivo de matrices y vectores, ya que cuenta con funciones integradas para la operación con este tipo de elementos. Resulta particularmente útil para la aplicación tratada en este trabajo, el hecho de que dentro de las funciones que operan con matrices, se halla la función exponencial matricial.

Tocando otros aspectos relacionados con la programación, habría que ahondar en los conceptos del lenguaje utilizado, de modo que el código fuera mucho menos redundante y por ende más eficiente. Esto a su vez daría como resultado que, en lugar de tener dos programas, se tuviera sólo uno que incorporara todas las funciones que por el momento se hallan separadas. Además podría caber la posibilidad de explorar otro tipo de plataformas de desarrollo, como por ejemplo UNIX y sus variaciones, a fin de aprovechar el poder de cálculo de equipos tales como las llamadas estaciones de trabajo que son mucho más poderosas que una PC.

Otra de las cosas en las que se puede profundizar para incrementar la eficiencia, es en la generación de trayectorias. Actualmente el programa utiliza curvas B-spline uniformes, con lo que no es posible interpolar exactamente las posiciones de los puntos de control. Sin embargo, como se mencionaba en el capítulo correspondiente, existe un tipo de curva similar llamada curva B-spline no uniforme, que también es capaz de generar trayectorias segmentadas y que permite interpolar todos los puntos de control.

Para efectos de visualización, también pueden realizarse mejoras substanciales, ya que existen lenguajes enfocados al desarrollo de aplicaciones de realidad virtual o de juegos tridimensionales, que permiten crear ambientes bastante realista con una velocidad de respuesta que se ajustan perfectamente a aplicaciones que requieren de una operación en tiempo real.

Además de las conclusiones directas que se obtienen a partir del trabajo desarrollado, pueden establecerse otra serie de conclusiones que más bien están relacionadas con la manera de concebir los problemas y con la filosofía que interviene en el diseño de las soluciones de cualquier clase de proyecto. En primer lugar, las experiencias recogidas durante el tiempo de desarrollo, indica que en la mayoría de las ocasiones, no es posible prever todos los aspectos que intervienen en un problema determinado, por lo que su solución va acompañada de un proceso de prueba y error.

Por otro lado, está muy claro con proyectos como el desarrollado, que el ingeniero debe ser capaz de actuar en un ambiente multidisciplinario, ya que actualmente los problemas llegan a ser tan complejos, que requieren para su solución, de la intervención de todo tipo de sistemas, la ingeniería mecatrónica es un

claro ejemplo de esto, ya que dentro de su concepción integra la participación de sistemas electrónicos, mecánicos y computacionales.

APÉNDICE A

Parámetros de Denavit-Hartenberg (D-H)

J, Denavit y R. S. Hartenberg propusieron en 1955 un método que permite determinar la posición y la orientación de los sistemas de referencia que van acoplados a cada uno de los eslabones que conforman la cadena cinemática. De esta manera, el enfoque de D-H se manifiesta por medio de una serie de transformaciones homogéneas en forma de matrices de 4 × 4, que representan al sistema coordenado de cada eslabón (colocado en la unión entre elementos), referenciado al sistema coordenado del eslabón anterior. Por lo tanto, a través de transformaciones sucesivas se pueden expresar las coordenadas relativas al sistema de la herramienta en coordenadas del sistema de referencia inercial.

Se escoge como sistema de referencia para cada eslabón un sistema coordenado cartesiano ortonormal. Para su definición, se dirá que el manipulador o cadena cinemática constará de n+1 eslabones con n articulaciones o conexiones, siendo esto válido si se considera que únicamente se cuenta con un grado de libertad por cada articulación. La base del manipulador se toma como el eslabón cero, al cual se acopla el primer eslabón a través de la primera articulación.

De acuerdo con lo anterior, el primer paso consiste en asignar a la base un sistema cartesiano derecho $x_0y_0z_0$, de modo que el eje z_0 represente al eje de movimiento de la primera articulación. El origen de este sistema de coordenadas puede colocarse en cualquier parte de la base del manipulador, siempre y cuando se halle sobre el eje de movimiento.

Para el siguiente eslabón i, los ejes coordenados se escogen siguiendo los siguientes pasos:

1. El eje z_i se coloca coincidiendo con el eje de movimiento del eslabón i+1. Si se trata de un par cinemático de revolución, entonces será un eje de rotación; y si se trata de un par prismático, entonces será un eje de desplazamiento lineal. La dirección positiva del eje se toma de manera que la rotación o el desplazamiento lineal sean positivos (la rotación es positiva si ésta va en contra del giro de las manecillas del reloj).
2. Si los ejes z_i y z_{i-1} se intersecan, el eje x_i tiene una dirección determinada por el producto cruz $\pm(\mathbf{k}_{i-1} \times \mathbf{k}_i)$, donde $\mathbf{k}_{i-1}$ y $\mathbf{k}_i$ son los vectores unitarios en la dirección positiva de los correspondientes ejes z_{i-1} y z_i. Si $\mathbf{k}_{i-1}$ y $\mathbf{k}_i$ son paralelos, entonces la dirección del eje x_i es a lo largo de la normal común entre los vectores $\mathbf{k}_{i-1}$ y $\mathbf{k}_i$.

3. El eje y_i se determina por medio de la regla de la mano derecha, de modo que se complete el sistema de referencia, es decir, $\mathbf{j}_i = \mathbf{k}_i \times \mathbf{i}_i$ donde $\mathbf{i}_i$ y $\mathbf{j}_i$ son los vectores unitarios en las direcciones x_i y y_i respectivamente.
4. El origen del sistema coordenado $x_iy_iz_i$ para el eslabón i, se coloca en la intersección de los ejes z_{i-1} y z_i, o en la intersección del eje z_i con la normal común entre los ejes z_{i-1} y z_i.

Cabe destacar que los pasos anteriores constituyen solamente una guía sistemática, ya que los sistemas de referencia locales pueden definirse de maneras ligeramente diferentes.

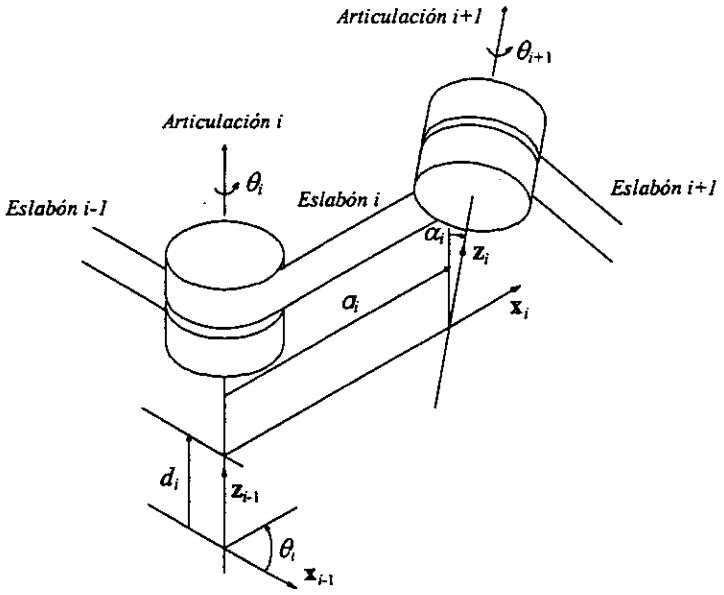

Figura A.1 Parámetros de los sistemas de referencia locales.

Para establecer las matrices de transformación que permitan expresar las coordenadas de un punto o vector en el sistema $x_{i-1}y_{i-1}z_{i-1}$ en términos del sistema $x_iy_iz_i$, es necesario definir cuatro parámetros geométricos asociados a cada eslabón. Estos parámetros pueden describir completamente cualquier par, ya sea de revolución o prismático, siendo estos parámetros: la distancia a_i, el ángulo α_i, la distancia d_i, y el ángulo ϕ_i, los cuales se determinan de la siguiente manera (Fig. A.1):

1. a_i : Es la distancia medida sobre el eje x_i que hay de la intersección de éste con el eje z_{i-1}, al origen del i-ésimo sistema de referencia, es decir, se trata de la distancia mínima (perpendicular) entre z_{i-1} y z_i.
2. α_i : Es el ángulo de rotación medido sobre el eje x_i desde el eje z_{i-1} (o su proyección paralela) hasta el eje z_i, empleando la regla de la mano derecha.
3. d_i : Es la distancia medida sobre el eje z_{i-1} que hay de la intersección de éste con el eje x_i, al origen del (i-1)-ésimo sistema de referencia. Si los ejes z_{i-1} y x_i no se intersecan, entonces d_i es la distancia perpendicular entre los ejes x_i y x_{i-1}.

4. ϕ_i : Es el ángulo de rotación medido sobre el eje z_{i-1} desde el eje x_{i-1} hasta el eje x_i (o su proyección).

Una vez que los parámetros y los sistemas de referencia de cada eslabón han sido determinados, se puede establecer un matriz de transformación que relacione al sistema de referencia i con el sistema (i-1), de modo que un punto o vector expresado en coordenadas del primero, puede ser expresado en términos del último aplicando la siguiente serie de transformaciones:

1. Rotar en torno al eje z_{i-1} un ángulo ϕ_i, a fin de alinear el eje x_{i-1} con el eje x_i.
2. Trasladar una distancia d_i a lo largo del eje z_{i-1}, de modo que los ejes x_{i-1} y x_i coincidan.
3. Trasladar a lo largo del eje x_i una distancia a_i, haciendo coincidir tanto los orígenes de los sistemas, como los ejes x.
4. Rotar en torno al eje x_i un ángulo α_i hasta lograr la coincidencia de ambos sistemas de referencia.

Cada una de estas transformaciones puede representarse mediante una matriz homogénea básica, la cual puede ser de traslación o de rotación. Mediante el producto de estas matrices básicas se obtiene una matriz de transformación compuesta, a la que se denominará como $g_{i-1,i}$ y que tiene la estructura siguiente:

$$g_{i_{i-1}i_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi_i & -\operatorname{sen}\phi_i & 0 & 0 \\ \operatorname{sen}\phi_i & \cos\phi_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\operatorname{sen}\alpha_i & 0 \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{A.1}$$

Al realizar las operaciones queda la siguiente matriz de transformación:

$$g_{i_{i-1}i_i} = \begin{bmatrix} \cos\phi_i & -\operatorname{sen}\phi_i\cos\alpha_i & \operatorname{sen}\phi_i\operatorname{sen}\alpha_i & a_i\cos\phi_i \\ \operatorname{sen}\phi_i & \cos\phi\cos\alpha_i & -\cos\phi_i\operatorname{sen}\alpha_i & a_i\operatorname{sen}\phi_i \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{A.2}$$

donde la variable de movimiento es, para el caso de un par de revolución, el ángulo ϕ_i, mientras que para un par prismático es la distancia a_i.

La manera como se relacionan las coordenadas de un punto p en un sistema de referencia con respecto a otro adyacente es:

$$p_{i-1} = g_{i_{i-1}i_i} p_i, \tag{A.3}$$

por lo que si se desea referir un punto en coordenadas del último eslabón o herramienta a coordenadas del sistema inercial, habría que incluir el efecto de cada una de las matrices de transformación, de modo que se tendría una expresión como esta:

$$p_s = g_{sl_1}(\theta_1) g_{l_1 l_2}(\theta_2) \cdots g_{l_{n-1} l_n}(\theta_n) g_{l_n t} p_t^{\cdot}, \tag{A.4}$$

en la que s se representa al sistema inercial, y t al sistema de la herramienta.

APÉNDICE B

Programa de simulación para WIN32 (Plataforma de 32 bits)

A continuación se presenta el código fuente de los archivos que integran el proyecto MAT-I.IDE que se encarga de realizar la simulación gráfica de los movimientos así como de la generación de trayectorias para los dedos del manipulador antropomórfico teleoperado (MAT-I). Este programa debe compilarse para correrse desde WIN 95.

B.1 Código fuente del archivo de definición MAT-I.DEF

```
EXETYPE WINDOWS
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE
```

B.2 Código fuente del archivo de encabezado MAT-I.RH

```
#define IDE_XPOS            101
#define IDE_YPOS            102
#define IDE_ZPOS            103

#define CM_GET_POINT        104
#define CM_DEL_POINT        105

#define MANO_ICON           1
#define TG_ICON             2
#define FI_ICON             3

#define MAIN_MENU           100

#define CM_SIMULATOR        200
#define CM_TRAJ_GEN         201
#define CM_SERIAL_CONFIG    202

#define CM_PROGRAM_RUN      203
#define CM_PROGRAM_PAUSE    204
#define CM_HELP_ABOUT       205

#define CM_ANIM_RUN         206
#define CM_ANIM_PAUSE       207

#define ABOUT_BOX           300

#define IDCB_PORT           501
#define IDCB_BAUDS          502
#define IDCB_DATABITS       503
#define IDCB_PARITY         504
#define IDCB_STOPBITS       505

#define IDB_RESTORE         601
#define IDB_DONE            602
```

B.3 Código fuente del archivo de recursos MAT-I.RC

```
#include <owl\window.rh>
#include <owl\except.rc>
#include "mat-i.rh"

SERIAL_CONFIG_BOX DIALOG 50, 50, 220, 240
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
WS_CAPTION | WS_SYSMENU
CAPTION "Configuración del puerto serial"
FONT 8, "MS Sans Serif"
{
 CONTROL "", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 10, 15,
200, 180
 CONTROL " Puerto", -1, "static", SS_LEFT | SS_SUNKEN | WS_CHILD | WS_VISIBLE,
25, 35, 70, 11
 CONTROL "", IDCB_PORT, "combobox", CBS_DROPDOWNLIST | WS_CHILD |
WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 105, 34, 90, 65
 CONTROL " Bits por segundo", -1, "static", SS_LEFT | SS_SUNKEN | WS_CHILD |
WS_VISIBLE, 25, 60, 70, 11
 CONTROL "", IDCB_BAUDS, "combobox", CBS_DROPDOWNLIST | WS_CHILD |
WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 105, 59, 90, 65
```

```
 CONTROL " Bits de datos", -1, "static", SS_LEFT | SS_SUNKEN | WS_CHILD |
WS_VISIBLE, 25, 85, 70, 11
 CONTROL "", IDCB_DATABITS, "combobox", CBS_DROPDOWNLIST | WS_CHILD |
WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 105, 84, 90, 65
 CONTROL " Paridad", -1, "static", SS_LEFT | SS_SUNKEN | WS_CHILD |
WS_VISIBLE, 25, 110, 70, 11
 CONTROL "", IDCB_PARITY, "combobox", CBS_DROPDOWNLIST | WS_CHILD |
WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 105, 109, 90, 65
 CONTROL " Bits de paro", -1, "static", SS_LEFT | SS_SUNKEN | WS_CHILD |
WS_VISIBLE, 25, 135, 70, 11
 CONTROL "", IDCB_STOPBITS, "combobox", CBS_DROPDOWNLIST | WS_CHILD |
WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 105, 134, 90, 65
 CONTROL "Restablecer opciones predeterminadas", IDB_RESTORE, "button",
BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 35,
165, 150, 14
 CONTROL "Aceptar", IDB_DONE, "button", BS_DEFPUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 40, 210, 50, 14
 CONTROL "Cancelar", IDCANCEL, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 129, 210, 50, 14
}

3D_POINTS_BOX DIALOG 50, 50, 220, 145
EXSTYLE WS_EX_DLGMODALFRAME | WS_EX_RIGHT
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
WS_CAPTION | WS_SYSMENU
CAPTION "Coordenadas en tres dimensiones"
FONT 8, "MS Sans Serif"
{
 CONTROL "", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 10, 10,
200, 95
 CONTROL "Coordenada en X :", -1, "static", SS_LEFT | SS_SUNKEN | WS_CHILD |
WS_VISIBLE, 25, 25, 70, 11
 CONTROL "", IDE_XPOS, "edit", ES_RIGHT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 105, 24, 90, 13
 CONTROL "Coordenada en Y :", -1, "static", SS_LEFT | SS_SUNKEN | WS_CHILD |
WS_VISIBLE, 25, 50, 70, 11
 CONTROL "", IDE_YPOS, "edit", ES_RIGHT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 105, 49, 90, 13
 CONTROL "Coordenada en Z :", -1, "static", SS_LEFT | SS_SUNKEN | WS_CHILD |
WS_VISIBLE, 25, 75, 70, 11
 CONTROL "", IDE_ZPOS, "edit", ES_RIGHT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 105, 74, 90, 13
 CONTROL "Aceptar", IDB_DONE, "BUTTON", BS_DEFPUSHBUTTON | BS_CENTER
| WS_CHILD | WS_VISIBLE | WS_TABSTOP, 40, 120, 50, 14
 CONTROL "Cancelar", IDCANCEL, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 125, 120, 50, 14
}

ABOUT_BOX DIALOG 50, 50, 220, 90
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
WS_CAPTION
CAPTION "Acerca de Manipulador Antropomórfico Teleoperado - I "
FONT 8, "MS Sans Serif"
{
 CONTROL "Manipulador Antropomórfico Teleoperado - I\nHugo Figueroa Rosas\n1998",
-1, "STATIC", SS_CENTER | SS_SUNKEN | WS_CHILD | WS_VISIBLE | WS_GROUP,
55, 18, 145, 28
 CONTROL MANO_ICON, -1, "STATIC", SS_ICON | SS_REALSIZEIMAGE |
SS_SUNKEN | WS_CHILD | WS_VISIBLE, 19, 21, 23, 21
 CONTROL "Aceptar", IDOK, "BUTTON", BS_PUSHBUTTON | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 85, 60, 50, 14
}

MAIN_MENU MENU
{
 POPUP "&Aplicación"
 {
  MENUITEM "&Simulador\tCtrl+S", CM_SIMULATOR
  MENUITEM "&Generador de trayectorias\tCtrl+G", CM_TRAJ_GEN
  MENUITEM SEPARATOR
  MENUITEM "&Salir\tAlt+X", CM_EXIT
 }

 MENUITEM SEPARATOR
 POPUP "&Simulador"
 {
  MENUITEM "&Ejecutar", CM_PROGRAM_RUN
  MENUITEM "&Pausa", CM_PROGRAM_PAUSE
 }

 MENUITEM SEPARATOR
 POPUP "&Generador de trayectorias"
 {
  MENUITEM "&Agregar punto\tIns", CM_GET_POINT
  MENUITEM "&Eliminar último punto\tSupr", CM_DEL_POINT
  MENUITEM SEPARATOR
  MENUITEM "&Ejecutar", CM_ANIM_RUN
  MENUITEM "&Pausa", CM_ANIM_PAUSE
 }

 MENUITEM SEPARATOR
 POPUP "&Configuración"
 {
  MENUITEM "&Configuración del puerto serie\tCtrl+C", CM_SERIAL_CONFIG
 }

 MENUITEM SEPARATOR
 POPUP "A&yuda"
 {
  MENUITEM "&Acerca de ...", CM_HELP_ABOUT
 }

}

MAIN_MENU ACCELERATORS
{
 "S", CM_SIMULATOR, VIRTKEY, CONTROL
 "G", CM_TRAJ_GEN, VIRTKEY, CONTROL
 "X", CM_EXIT, VIRTKEY, ALT
 VK_INSERT, CM_GET_POINT, VIRTKEY
 VK_DELETE, CM_DEL_POINT, VIRTKEY
 "C", CM_SERIAL_CONFIG, VIRTKEY, CONTROL
}

MANO_ICON ICON
{
'00 00 01 00 01 00 20 20 10 00 00 00 00 00 E8 02'
'00 00 16 00 00 00 28 00 00 00 20 00 00 00 40 00'
'00 00 01 00 04 00 00 00 00 00 80 02 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 80 00 00 80 00 00 00 80 80 00 80 00'
'00 00 80 00 80 00 80 80 00 00 C0 C0 C0 00 80 80'
'80 00 00 00 FF 00 00 FF 00 00 00 FF FF 00 FF 00'
'00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 05 55'
'55 55 55 55 55 00 00 00 00 00 00 00 00 00 05 99'
'99 99 99 99 95 50 77 70 00 00 00 00 00 00 05 99'
'99 99 99 99 99 07 77 F0 00 00 77 70 00 00 05 99'
'99 99 99 99 90 07 77 00 00 07 7F 70 00 00 05 99'
'99 99 99 99 07 70 0F 05 00 77 77 F0 00 00 05 99'
'99 99 99 90 77 77 F0 95 07 0F 7F 00 00 00 05 99'
'99 99 99 07 7F 77 09 90 77 70 F0 00 00 00 05 99'
'99 99 99 00 77 77 09 0F 7F 7F 00 00 00 00 05 99'
'99 99 90 77 00 F0 90 77 77 70 00 07 77 00 05 99'
'99 99 07 F7 77 09 07 07 7F 05 50 7F 77 00 05 99'
'99 90 77 77 77 00 77 70 70 99 07 07 7F 00 05 99'
'99 00 07 77 F0 07 77 77 00 90 77 70 F7 00 05 99'
'90 07 07 77 07 0F 7F 77 09 07 77 77 00 00 05 59'
'07 77 00 F7 00 77 77 F0 90 07 F7 7F 00 00 00 50'
'07 77 07 00 00 77 77 00 07 70 77 70 00 00 00 00'
'77 70 77 77 77 07 F0 70 77 77 0F 05 50 00 00 07'
'77 07 77 77 7F 70 00 70 7F 77 F0 99 55 00 00 77'
'70 77 7F 7F 77 77 70 07 77 77 09 90 05 50 00 77'
'07 77 77 77 7F 7F 7F 07 77 F0 99 07 77 50 00 77'
'07 7F 7F 77 77 77 F7 F0 77 09 90 77 7F 00 00 70'
'77 77 77 7F 7F 7F 7F F0 F0 99 07 77 F0 00 00 07'
'7F 7F 77 77 77 F7 F7 F0 09 00 07 7F 70 50 00 77'
'77 77 7F 7F 7F 7F FF 70 00 77 70 07 09 50 00 7F'
'7F 77 77 77 F7 F7 F7 00 07 77 77 F0 09 50 00 00'
'77 7F 7F 7F 7F FF 70 77 07 F7 7F 00 99 50 00 00'
'07 77 77 F7 F7 F7 07 77 70 7F 70 99 99 50 00 00'
'00 7F 7F 7F FF 00 7F 77 F0 F0 09 99 99 50 00 00'
'00 00 F7 F7 F0 77 77 7F 00 09 99 99 99 50 00 00'
'00 00 00 00 07 77 7F 70 09 99 99 99 99 50 00 00'
'00 00 00 00 00 00 00 00 55 55 55 55 55 50 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF'
'FF FF 80 03 1F FF 80 00 0F 1F 80 00 0E 0F 80 00'
'1C 0F 80 00 08 0F 80 00 00 1F 80 00 00 3F 80 00'
'00 63 80 00 00 C1 80 00 00 01 80 00 00 01 80 00'
'00 01 80 00 00 03 80 00 00 07 C0 00 00 0F C0 00'
'00 07 C0 00 00 03 80 00 00 01 80 00 00 01 80 00'
'00 01 80 00 00 01 80 00 00 01 80 00 00 01 80 00'
'00 01 C0 00 00 01 E0 00 00 01 F0 00 00 01 F8 00'
'00 01 FE 00 00 01 FF F0 00 01 FF FF FF FF'
}

FI_ICON ICON
{
'00 00 01 00 01 00 20 20 10 00 00 00 00 00 E8 02'
'00 00 16 00 00 00 28 00 00 00 20 00 00 00 40 00'
'00 00 01 00 04 00 00 00 00 00 80 02 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 80 00 00 80 00 00 00 80 80 00 80 00'
'00 00 80 00 80 00 80 80 00 00 C0 C0 C0 00 80 80'
'80 00 00 00 FF 00 00 FF 00 00 00 FF FF 00 FF 00'
'00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF'
'FF FF FF FF FF FF FF FF FF FF FF FF FF 00 0F FF'
'FF FF FF FF FF FF FF FF FF FF FF FF FF F0 0F FF'
'FF FF FF FF FF FF FF FF FF FF FF FF FF F0 0F 99'
'99 99 99 FF FF FF FF FF F9 99 99 99 99 F0 0F 99'
'99 99 99 FF FF FF FF FF F9 99 99 99 99 F0 0F 99'
'99 99 99 FF FF FF FF FF F9 99 99 99 99 F0 0F 99'
```

```
'99 99 99 FF FF FF FF FF F9 99 99 99 99 F0 0F F9'
'99 99 99 FF FF FF FF FF F9 99 99 99 9F F0 0F FF'
'FF F9 99 99 99 99 99 9F F9 99 9F FF FF F0 0F FF'
'99 FF FF F9 99 99 99 9F FF FF FF 99 FF F0 0F FF'
'99 FF 99 FF FF FF FF FF FF 99 FF 99 FF F0 0F FF'
'99 FF 99 FF F9 9F F9 9F FF 99 FF 99 FF F0 0F 99'
'99 99 99 99 99 99 99 9F F9 99 99 99 99 F0 0F 99'
'99 99 99 FF FF FF FF FF F9 99 99 99 99 F0 0F 99'
'99 99 99 FF FF FF FF FF F9 99 99 99 99 F0 0F 99'
'99 99 99 99 99 99 99 9F F9 99 99 99 99 F0 0F 99'
'99 99 99 99 99 99 99 9F F9 99 99 99 99 F0 0F 99'
'99 99 99 99 99 99 99 9F F9 99 99 99 99 F0 0F 99'
'99 99 99 99 99 99 99 9F F9 99 99 99 99 F0 0F 99'
'99 99 99 99 99 99 99 9F F9 99 99 99 99 F0 0F FF'
'FF FF FF FF FF FF FF FF FF FF FF FF FF F0 0F FF'
'FF FF FF FF FF FF FF FF FF FF FF FF FF F0 00 FF'
'FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF'
'FF FF FF FF FF FF C0 00 00 03 80 00 00 01 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 80 00 00 01 C0 00'
'00 03 FF FF FF FF FF FF FF FF FF FF FF FF'
}

TG_ICON ICON
{
'00 00 01 00 01 00 20 20 10 00 00 00 00 00 E8 02'
'00 00 16 00 00 00 28 00 00 00 20 00 00 00 40 00'
'00 00 01 00 04 00 00 00 00 00 80 02 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 80 00 00 80 00 00 00 80 80 00 80 00'
'00 00 80 00 80 00 80 80 00 00 C0 C0 C0 00 80 80'
'80 00 00 00 FF 00 00 FF 00 00 00 FF FF 00 FF 00'
'00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00 00 0F'
'FF FF FF FF FF FF FF FF FF FF FF FF F0 00 00 FF'
'FF FF FF FF FF FF FF FF FF FF FF FF FF 00 0F F0'
'00 00 00 00 00 00 00 00 00 00 00 00 0F F0 FF 00'
'00 00 00 00 00 00 00 00 0A AA A0 00 00 FF FF 00'
'00 00 00 00 00 00 00 00 00 AA AA 00 00 00 FF FF 00'
'00 00 00 00 00 00 00 00 0A AA 00 0B BB 00 FF FF 00'
'00 00 00 0B BB 00 00 AA A0 00 0B BB 00 FF FF 00'
'00 00 00 0B BB 00 0A AA 00 00 0B BB 00 FF FF 00'
'00 00 00 0B BB 00 AA A0 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 AA 00 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 0A AA 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 0A A0 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 0A A0 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 AA A0 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 AA 00 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 AA 00 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 AA 00 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 AA 00 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 0A AA 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 0A A0 00 00 00 00 00 FF FF 0B'
'BB 00 00 00 AA A0 0B BB 00 00 00 00 00 FF FF 0B'
'BB 00 00 00 0A AA 00 0B BB 00 00 00 00 FF FF 0B'
'BB 00 00 00 AA A0 00 0B BB 00 00 00 00 FF FF 00'
'00 00 AA AA 00 00 00 00 00 00 00 00 00 FF FF 00'
'00 AA AA A0 00 00 00 00 00 00 00 00 00 FF FF 00'
'AA AA A0 00 00 00 00 00 00 00 00 00 00 FF FF 0A'
'AA A0 00 00 00 00 00 00 00 00 00 00 00 FF FF 0A'
'A0 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 00'
'00 00 00 00 00 00 00 00 00 00 00 00 FF 0F F0'
'00 00 00 00 00 00 00 00 00 00 00 00 0F F0 00 FF'
'FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00 0F'
'FF FF FF FF FF FF FF FF FF FF FF FF F0 00 E0 00'
'00 07 C0 00 00 03 80 00 00 01 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 80 00 00 01 C0 00 00 03 E0 00 00 07'
}
```

B.4 Código fuente del archivo de programa MAT-I.CPP

```
/*************************************************************************/
//
// MANIPULADOR ANTROPOMORFICO TELEOPERADO - I ( MAT - I )
//
// MAT-I.CPP
//
// Programa de simulacion y generacion de trayectorias de un manipulador
// antropomófico
//
// Plataforma: WIN32 (32 bits)
//
// Hugo Figueroa Rosas  1998.
//
/*************************************************************************/

/*************************************************************************/
// Archivos de encabezado
/*************************************************************************/

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <owl\button.h>
#include <owl\combobox.h>
#include <owl\edit.h>
#include <owl\pch.h>

#include "list.h"
#include "mati.rh"

/*************************************************************************/
// Definicion de funciones trigonometricas simplificadas
/*************************************************************************/

#define S(x) sin((x)*PI/180)          //funcion seno con argumento en grados
#define C(x) cos((x)*PI/180)          //funcion coseno con argumento en grados
#define AS(x) (asin(x)*180/PI)        //funcion arco seno en grados
#define AC(x) (acos(x)*180/PI)        //funcion arco coseno en grados
#define AT(x) (atan(x)*180/PI)        //funcion arco tangente en grados
#define AT2(y,x) (atan2(y,x)*180/PI) //funcion arco tangente en grados (y/x)

/*************************************************************************/
// Definicion de constantes matematicas empleadas
/*************************************************************************/

const double PI = 3.1415926535898;  //pi

const int ORD = 4;     //tamano de las matrices cuadradas y vectores

const double DANG = 5;   //delta de angulo

/*************************************************************************/
// Constantes para instancias falso/verdadero
/*************************************************************************/

const bool T = true;     //valor verdadero
const bool F = false;    //valor falso

/*************************************************************************/
// Constantes para el trazo de los dedos
/*************************************************************************/

const double FESC = 1;           //factor de escala para los dedos
const double WW = FESC*0.4392;   //ancho de los dedos
const double HH = FESC*0.3659;   //alto de los dedos

const double D = FESC*5;     //distancia de la camara al plano de proyeccion
const double U = FESC*1;     //longitud de los ejes del origen
const double R = FESC*0.5;   //longitud de los ejes de los sistemas de referencia

/*************************************************************************/
// Constantes de la palma
/*************************************************************************/

const double XPL = FESC*0;          //coordenada en x del origen de la palma
const double YPL = FESC*-1.7325;    //coordenada en y del origen de la palma
const double ZPL = FESC*0;          //coordenada en z del origen de la palma

const double LPAL = FESC*1.7325;   //longitud de la palma

const double LSPL1 = 180;   //lim sup grado 1
const double LIPL1 = -180;  //lim inf grado 1
const double LSPL2 = 90;    //lim sup grado 2
const double LIPL2 = -90 ;  //lim inf grado 2
const double LSPL3 = 90;    //lim sup grado 3
const double LIPL3 = -90;   //lim inf grado 3
```

```
/*****************************************************************************/
// Constantes del dedo pulgar
/*****************************************************************************/

const double XP = FESC*-0.5867;  //coordenada en x del origen del pulgar
const double YP = FESC*-1.7325;  //coordenada en y del origen del pulgar
const double ZP = FESC*0;        //coordenada en z del origen del pulgar

const double L1P = FESC*0.9246;  //longitud de la falange proximal del pulgar
const double L2P = FESC*0.7628;  //longitud de la falange media del pulgar
const double L3P = FESC*0.601;   //longitud de la falange distal del pulgar

const double LSP1 = 90;    //lim sup grado 1
const double LIP1 = 0;     //lim inf grado 1
const double LSP2 = 20;    //lim sup grado 2
const double LIP2 = -90;   //lim inf grado 2
const double LSP3 = 0;     //lim sup grado 3
const double LIP3 = -90;   //lim inf grado 3
const double LSP4 = 0;     //lim sup grado 4
const double LIP4 = -90;   //lim inf grado 4

const double A = 30;   //angulo de orientacion del sistema del pulgar (eje z')
const double B = 35;   //angulo de orientacion del sistema del pulgar (eje y')

/*****************************************************************************/
// Constantes del dedo indice
/*****************************************************************************/

const double XI = FESC*-0.5867;  //coordenada en x del origen del indice
const double YI = FESC*-0.0255;  //coordenada en y del origen del indice
const double ZI = FESC*0;        //coordenada en z del origen del indice

const double L1I = FESC*1;       //longitud de la falange proximal del indice
const double L2I = FESC*0.5855;  //longitud de la falange media del indice
const double L3I = FESC*0.4879;  //longitud de la falange distal del indice

const double LSD1 = 20;    //lim sup grado 1 para el resto de los dedos
const double LID1 = -20;   //lim inf grado 1
const double LSD2 = 20;    //lim sup grado 2
const double LID2 = -105;  //lim inf grado 2
const double LSD3 = 0;     //lim sup grado 3 y 4
const double LID3 = -90;   //lim inf grado 3 y 4

/*****************************************************************************/
// Constantes del dedo medio
/*****************************************************************************/

const double XM = FESC*0;      //coordenada en x del origen del medio
const double YM = FESC*0;      //coordenada en y del origen del medio
const double ZM = FESC*0;      //coordenada en z del origen del medio

const double L1M = FESC*1.098;  //longitud de la falange proximal del medio
const double L2M = FESC*0.683;  //longitud de la falange media del medio
const double L3M = FESC*0.512;  //longitud de la falange distal del medio

/*****************************************************************************/
// Constantes del dedo anular
/*****************************************************************************/

const double XA = FESC*0.5867;  //coordenada en x del origen del anular
const double YA = FESC*-0.0255;  //coordenada en y del origen del anular
const double ZA = FESC*0;        //coordenada en z del origen del anular

const double L1A = FESC*1.07;    //longitud de la falange proximal del medio
const double L2A = FESC*0.6343;  //longitud de la falange media del medio
const double L3A = FESC*0.512;   //longitud de la falange distal del medio

/*****************************************************************************/
// Constantes de temporizacion
/*****************************************************************************/

const uint TIME1 = 20;  //intervalo de retraso en milisegundos
const uint TIME2 = 20;

/*****************************************************************************/
// Definicion de una matriz de 4x4 para el manejo de rotaciones en 3D
/*****************************************************************************/

class m4
{
public:
  double m[ORD][ORD];
};

/*****************************************************************************/
// Definicion de un vector de 4 elementos para el manejo de puntos
// y vectores en 3D
/*****************************************************************************/

class v4
{
public:
  double v[ORD];
};

/*****************************************************************************/
// Definicion de un vector de 8 elementos v4 para el manejo de coordenadas
// de las aristas de paralelepipedos y prismas trapezoidales
/*****************************************************************************/

class v48
{
public:
  v4 v[8];
};

/*****************************************************************************/
// Definicion de un vector de 12 elementos v4 para el manejo de coordenadas
// de las aristas de prismas de seccion variable
/*****************************************************************************/

class v412
{
public:
  v4 v[12];
};

/*****************************************************************************/
// Definicion de un vector de 19 elementos para el manejo de los angulos
// en cada uno de los eslabones
/*****************************************************************************/

class v19
{
public:
  double v[19];
};

/*****************************************************************************/
// Definicion de un vector de 13 elementos para el manejo de la entrada
// y salida de informacion por puerto serie
/*****************************************************************************/

class v13
{
public:
  double v[13];
};

/*****************************************************************************/
// Definicion colores
/*****************************************************************************/

const TColor CBLACK    = TColor(0,0,0);
const TColor CBLUE     = TColor(0,0,255);
const TColor CGREEN    = TColor(0,255,0);
const TColor CRED      = TColor(255,0,0);
const TColor CYELLOW   = TColor(255,255,0);
const TColor CMAGENTA  = TColor(255,0,255);
const TColor CCYAN     = TColor(0,255,255);
const TColor CWHITE    = TColor(255,255,255);

const TColor CDBLUE    = TColor(0,0,128);
const TColor CDGREEN   = TColor(0,128,0);
const TColor CDRED     = TColor(128,0,0);
const TColor CDYELLOW  = TColor(128,128,0);
const TColor CDMAGENTA = TColor(128,0,128);
const TColor CDCYAN    = TColor(0,128,128);
const TColor CGRAY     = TColor(128,128,128);

/*****************************************************************************/
// Variables para el control de ventanas
/*****************************************************************************/

enum EstadoSimulador {esEnabled,esDisabled} estsim;
enum EstadoGenTray {egtEnabled,egtDisabled} estgt;
enum EstadoPuerto {epEnabled,epDisabled} estprt;

/*****************************************************************************/
// Variables de comunicacion serial
/*****************************************************************************/

HANDLE hCom;    //Maneja la instancia del puerto serial
DCB dcb = {0};  //Estructura que establece las caracteristicas del puerto
DCB dcbant = {0};  //Estructura que restablece las caracteristicas del puerto

/*****************************************************************************/
/*****************************************************************************/
// DECLARACION DE LA CLASE << TSerialConfigDialog >>
/*****************************************************************************/
/*****************************************************************************/

const char far* puerto;
unsigned long baudios;
unsigned char bitsdatos,paridad,bitsparo;
```

```
static int port=1,     //COM_2
       bauds=6,        //9600 bauds
       databits=4,     //8 bits de datos
       parity=0,       //Sin paridad
       stopbits=0;     //Un bit de paro

class TSerialConfigDialog : public TDialog
{
private:

  TComboBox *Port,*Bauds,*DataBits,*Parity,*StopBits;

public:

  TSerialConfigDialog(TWindow* parent,TModule* module=0);
  ~TSerialConfigDialog();
  virtual void SetupWindow();

  void CmDone();
  void CmRestoreOptions();

DECLARE_RESPONSE_TABLE(TSerialConfigDialog);
};

DEFINE_RESPONSE_TABLE1(TSerialConfigDialog,TDialog)
  EV_BN_CLICKED(IDB_DONE,CmDone),
  EV_BN_CLICKED(IDB_RESTORE,CmRestoreOptions),
END_RESPONSE_TABLE;

/*************************************************************************/
// FUNCIONES PROPIAS DE CLASE
/*************************************************************************/

/*************************************************************************/
// Funcion: TSerialCofigDialog()
// Constructor de clase
/*************************************************************************/

TSerialConfigDialog::TSerialConfigDialog(TWindow* parent,TModule* module)
  : TDialog(parent,"SERIAL_CONFIG_BOX",module)
{
  Port = new TComboBox(this,IDCB_PORT);
  Bauds = new TComboBox(this,IDCB_BAUDS);
  DataBits = new TComboBox(this,IDCB_DATABITS);
  Parity = new TComboBox(this,IDCB_PARITY);
  StopBits = new TComboBox(this,IDCB_STOPBITS);
}

/*************************************************************************/
// Funcion: ~TSerialConfigDialog()
// Destructor de clase
/*************************************************************************/

TSerialConfigDialog::~TSerialConfigDialog()
{

}

/*************************************************************************/
// Funcion: SetupWindow()
// Inicializa el elemento visual
/*************************************************************************/

void TSerialConfigDialog::SetupWindow()
{
  int i;

  static char* s1[]={"COM_1","COM_2",NULL};
  static char* s2[]={"110","300","600","1200","2400","4800",
                     "9600","14400","19200","38400","56000",
                     "57600","115200","128000","256000",NULL};
  static char* s3[]={"4","5","6","7","8",NULL};
  static char* s4[]={"Ninguna","Impar","Par","Marca","Espacio",NULL};
  static char* s5[]={"1","1.5","2",NULL};

  TDialog::SetupWindow();

  if(Port)
  {
    for(i=0;s1[i];++i)
    Port->AddString(s1[i]);
    Port->SetSelIndex(port);
  }

  if(Bauds)
  {
    for(i=0;s2[i];++i)
      Bauds->AddString(s2[i]);
    Bauds->SetSelIndex(bauds);
  }

  if(DataBits)
  {
    for(i=0;s3[i];++i)
      DataBits->AddString(s3[i]);
    DataBits->SetSelIndex(databits);
  }

  if(Parity)
  {
    for(i=0;s4[i];++i)
       Parity->AddString(s4[i]);
    Parity->SetSelIndex(parity);
  }

  if(StopBits)
  {
    for(i=0;s5[i];++i)
      StopBits->AddString(s5[i]);
    StopBits->SetSelIndex(stopbits);
  }
}

/*************************************************************************/
// FUNCIONES DE LA TABLA DE RESPUESTA
/*************************************************************************/

/*************************************************************************/
// Función: CmRestoreOptions()
// Responde al mensaje IDB_RESTORE
/*************************************************************************/

void TSerialConfigDialog::CmRestoreOptions()
{
  if(Port&&Bauds&&DataBits&&Parity&&StopBits)
  {
    Port->SetSelIndex(1);     //COM_2
    Bauds->SetSelIndex(6);    //9600 bauds
    DataBits->SetSelIndex(4); //8 bits
    Parity->SetSelIndex(0);   //Sin paridad
    StopBits->SetSelIndex(0); //1 bit de paro
  }
}

/*************************************************************************/
// Funcion: CmDone()
// Responde al mensaje IDB_DONE
/*************************************************************************/

void TSerialConfigDialog::CmDone()
{
  const char far *PortTable[]={"COM1","COM2"};
  unsigned long BaudsTable[]={CBR_110,CBR_300,CBR_600,CBR_1200,CBR_2400,
                              CBR_4800,CBR_9600,CBR_14400,CBR_19200,
                              CBR_38400,CBR_56000,CBR_57600,CBR_115200,
                              CBR_128000,CBR_256000};
  unsigned char DataBitsTable[]={4,5,6,7,8};
  unsigned char ParityTable[]={NOPARITY,ODDPARITY,EVENPARITY,
                               MARKPARITY,SPACEPARITY};
  unsigned char StopBitsTable[]={ONESTOPBIT,ONE5STOPBITS,TWOSTOPBITS};

  if(Port)
    port=Port->GetSelIndex();

  if(Bauds)
    bauds=Bauds->GetSelIndex();

  if(DataBits)
    databits=DataBits->GetSelIndex();

  if(Parity)
    parity=Parity->GetSelIndex();

  if(StopBits)
    stopbits=StopBits->GetSelIndex();

  if(Port&&Bauds&&DataBits&&Parity&&StopBits)
  {
    puerto=PortTable[port];
    baudios=BaudsTable[bauds];
    bitsdatos=DataBitsTable[databits];
    paridad=ParityTable[parity];
    bitsparo=StopBitsTable[stopbits];
  }

  Destroy(IDB_DONE);
}

/*************************************************************************/
/*************************************************************************/
// DECLARACION DE LA CLASE << THandMDIClient >>
/*************************************************************************/
/*************************************************************************/

class THandMDIClient : public TMDIClient
```

```
{
public:

  THandMDIClient();

  virtual bool CanClose();
  void CmSerialConfig();

DECLARE_RESPONSE_TABLE(THandMDIClient);
};

DEFINE_RESPONSE_TABLE1(THandMDIClient,TMDIClient)
  EV_COMMAND(CM_SERIAL_CONFIG,CmSerialConfig),
END_RESPONSE_TABLE;

/*************************************************************************/
// FUNCIONES PROPIAS DE LA CLASE
/*************************************************************************/

/*************************************************************************/
// Funcion: THandMDIClient()
// Constructor de clase
/*************************************************************************/

THandMDIClient::THandMDIClient()
  : TMDIClient()
{
  estprt=epDisabled;
}

/*************************************************************************/
// FUNCIONES DE LA TABLA DE RESPUESTA
/*************************************************************************/

/*************************************************************************/
// Funcion: CanClose()
// Responde ante la solicitud de cierre de la ventana
/*************************************************************************/

bool THandMDIClient::CanClose()
{
  bool bResult;
  bResult=MessageBox("¿Desea cerrar la aplicación?",
             "Manipulador Antropomórfico Teleoperado",
             MB_YESNO|MB_ICONQUESTION)==IDYES;
  if(bResult)
  {
    if(estprt==epEnabled)
      if(!SetCommState(hCom,&dcbant))
        MessageBox("Falló SetCommState","Mensaje",
                MB_OK|MB_ICONSTOP);
    CloseHandle(hCom);
  }

  return bResult;
}

/*************************************************************************/
// Funcion: CmSerialConfig()
// Responde al mensaje CM_SERIAL_CONFIG
/*************************************************************************/

void THandMDIClient::CmSerialConfig()
{
  COMMTIMEOUTS to;

  if(TSerialConfigDialog(this).Execute()==IDB_DONE)
  {
    estprt = epEnabled;

    CloseHandle(hCom);
    hCom = CreateFile(puerto,
              GENERIC_READ | GENERIC_WRITE,
              0,
              NULL,
              OPEN_EXISTING,
              FILE_FLAG_OVERLAPPED,
              NULL);

    if(hCom == INVALID_HANDLE_VALUE)
    {
      MessageBox("Falló CreateFile","Mensaje",MB_OK|MB_ICONSTOP);
      estprt=epDisabled;
    }
    else
    {
      if(!GetCommTimeouts(hCom,&to))
        MessageBox("Falló GetCommTimeOuts","Mensaje",
                MB_OK|MB_ICONSTOP);
      else
      {
        to.ReadIntervalTimeout=20;
        to.ReadTotalTimeoutMultiplier=10;
        to.ReadTotalTimeoutConstant=100;
        to.WriteTotalTimeoutMultiplier=10;
        to.WriteTotalTimeoutConstant=100;

        if(!SetCommTimeouts(hCom,&to))
          MessageBox("Falló SetCommTimeOuts","Mensaje",
                MB_OK|MB_ICONSTOP);
      }

      dcb.DCBlength = sizeof(dcb);
      dcbant.DCBlength = sizeof(dcbant);

      if(!GetCommState(hCom,&dcbant))
        MessageBox("Falló GetCommState","Mensaje",MB_OK|MB_ICONSTOP);
      else
      {
        dcb=dcbant;
        dcb.BaudRate = baudios;
        dcb.ByteSize = bitsdatos;
        dcb.Parity = paridad;
        dcb.StopBits = bitsparo;
        dcb.EvtChar=247;

        if(!SetCommState(hCom,&dcb))
          MessageBox("Falló SetCommState","Mensaje",
                MB_OK|MB_ICONSTOP);
      }
    }
  }
}

/*************************************************************************/
/*************************************************************************/
// DECLARACION DE LA CLASE << THandChildSim >>
/*************************************************************************/
/*************************************************************************/

class THandChildSim : public TMDIChild
{
protected:

  typedef std::list<v4>        ptList;
  typedef std::list<v4>::iterator ptIter;

  enum Estado   {eActivo,eInactivo,ePausa};
  Estado estado;  //Estado de ejecucion

  enum Miembro {mNada,mPalma,mPulgar,mIndice,mMedio,mAnular};
  Miembro miembro;  //identificador de miembro

  bool flag2;    //bandera que activa la busqueda del valor minimo
  bool flag3;    //bandera que activa la busqueda del valor maximo
  bool flag4;    //bandera que activa la secuencia de calibracion
  bool flag5;    //bandera que activa la secuencia de grabado
  bool flag6;    //bandera que indica ejecucion de secuencia

  uint Timer;    //ID del temporizador del programa

  TMemoryDC *memDC;

  double THETA,PHI,RHO;  //angulos del plano de proyección

  int WI,HE;       //ancho y altura de la ventana
  int MARGEN;      //margen alrededor de area de trazado
  int XL,XR,YB,YT;  //limites izq, der, inf, y sup para el área de trazo
  int XDIF,YDIF;    //anchura y altura de la pantalla
  double XMIN,XMAX,YMIN,YMAX;  //limites teoricos deseados por el usuario
  double SCALX,SCALY;          //factores de escala para el trazado

  int UCS_XL,UCS_XR,UCS_YT,UCS_YB;  //limites para el ucs_icon
  int UCS_XDIF,UCS_YDIF;
  double UCS_XMIN,UCS_XMAX,UCS_YMIN,UCS_YMAX;
  double UCS_SCALX,UCS_SCALY;

  m4 mview,g[13],gs[13],gs0[17],e[19];
  v4  Q[17],I,J,K,IP,JP,KP,NULO;
  v19 ang,anginf,angsup;
  v48 bb[9];
  v412 tt[4];

  TColor CBackground,CHand,CSelection,COrigin,CUcsOrigin;

public:

  THandChildSim(TMDIClient& parent);
  ~THandChildSim();
  void CleanupWindow();
  void SetupWindow();
  void Paint(TDC& dc,bool erase,TRect& rect);

  bool CanClose();
  void CmPrgPause();
  void CmPrgPauseEnabler(TCommandEnabler& tce);
```

```
  void CmPrgRun();
  void CmPrgRunEnabler(TCommandEnabler& tce);
  void EvChar(uint key,uint repeatCount,uint flags);
  bool EvEraseBkgnd(HDC);
  void EvKeyDown(uint key,uint repeatCount,uint flags);
  void EvTimer(uint id);

  void initpuntos(void);
  m4 mgs0(const v4& p);
  m4 mgsp0(const v4& p);
  void initmatgs0(void);
  m4 ee1(double a,const v4& p);
  m4 ee2(double a,const v4& p);
  m4 ee3(double a,const v4& p);
  m4 ee4(double a,const v4& w,const v4& p);
  void mg(const v19& a);
  m4 m_rot(double a,const v4& b,double x,double y,double z);

  m4 m_esc(double esc,const m4& a);
  m4 m_transp(const m4& a);
  m4 mm_mult(const m4& a,const m4& b);
  v4 mv_mult(const m4& a,const v4& b);
  m4 m_inv(const m4& a);

  v4 p_in(double x,double y,double z,double k = 1);
  v4 v_in(double x,double y,double z,double k = 0);
  v4 v_sum(const v4& a,const v4& b,bool oper = T);
  v4 v_esc(double esc,const v4& a);
  double dot(const v4& a,const v4& b);
  v4 cross(const v4& a,const v4& b);
  double mag2(const v4& a);
  double mag(const v4& a);
  v4 v_unit(const v4& a);

  v48 v48_in(double a,double b,double h,double l);
  v412 v412_in(double w1,double h1,double w2,double h2,double l,double r);

  m4 mmview(double theta,double& phi,double rho);
  TPoint vscreen(double d,const v4& a,const m4& m);
  void ucs_icon(TDC& dc,TColor color,double l);
  void origen(TDC& dc,TColor color,double l);
  void sistref(TDC& dc,TColor color,double l,const m4& rot);
  void barra(TDC& dc,v48& p3d,const m4& rot);
  void tip(TDC& dc,v412& p3d,const m4& rot);
  void palma(TDC& dc,TColor color);
  void pulgar(TDC& dc,TColor color);
  void indice(TDC& dc,TColor color);
  void medio(TDC& dc,TColor color);
  void anular(TDC& dc,TColor color);
  void mano(TDC& dc,Miembro flag);
  void pt2d(TDC& dc,TColor color,TPoint& point,int l = 3);
  void pt3d(TDC& dc,TColor color,const v4& p,int l = 3);

  double interpol(double x1,double x2,double y1,double y2,double x);

  bool WriteBuffer(char* Buffer,unsigned long dwToWrite);
  void ReadBuffer();

  v4 cin_inv_dedo(double l2,double l3,double l4,const v4& p,const v4& q);
  v4 cin_inv_pulg(double l2,double l3,double l4,const v4& p,const v4& q,
        double a4);

DECLARE_RESPONSE_TABLE(THandChildSim);
};

DEFINE_RESPONSE_TABLE1(THandChildSim,TMDIChild)
  EV_COMMAND(CM_PROGRAM_PAUSE,CmPrgPause),
  EV_COMMAND_ENABLE(CM_PROGRAM_PAUSE,CmPrgPauseEnabler),
  EV_COMMAND(CM_PROGRAM_RUN,CmPrgRun),
  EV_COMMAND_ENABLE(CM_PROGRAM_RUN,CmPrgRunEnabler),
  EV_WM_CHAR,
  EV_WM_ERASEBKGND,
  EV_WM_KEYDOWN,
  EV_WM_TIMER,
END_RESPONSE_TABLE;

/*****************************************************************/
// FUNCIONES PROPIAS DE CLASE
/*****************************************************************/

/*****************************************************************/
// Funcion: THandChildSim()
// Constructor de clase
/*****************************************************************/

THandChildSim::THandChildSim(TMDIClient& parent)
  : TMDIChild(parent)
{
  WI=600;
  HE=400;
  MARGEN=10;

  Attr.Style &= ~WS_THICKFRAME;
  Attr.Style &= ~WS_MAXIMIZEBOX;
  Attr.W=WI+5;
  Attr.H=HE+25;
  SetCaption("Simulador");

  estado=eInactivo;

  XL=MARGEN;
  XR=WI-MARGEN;
  YB=HE-MARGEN;
  YT=MARGEN;

  XDIF=XR-XL;
  YDIF=YB-YT;

  XMIN=-3;
  XMAX=3;
  YMIN=-3;
  YMAX=3;

  XMIN=((XMIN*XDIF)/YDIF);
  XMAX=((XMAX*XDIF)/YDIF);

  SCALX=((double)XDIF/(double)(XMAX-XMIN));
  SCALY=((double)YDIF/(double)(YMAX-YMIN));

  UCS_XL=XL;
  UCS_XR=XL+(int)(YDIF*0.2);
  UCS_YT=YT+(int)(YDIF*0.8);
  UCS_YB=YB;
  UCS_XMIN=-1;
  UCS_XMAX=1;
  UCS_YMIN=-1;
  UCS_YMAX=1;

  UCS_XDIF=UCS_XR-UCS_XL;
  UCS_YDIF=UCS_YB-UCS_YT;

  UCS_SCALX=((double)UCS_XDIF/(double)(UCS_XMAX-UCS_XMIN));
  UCS_SCALY=((double)UCS_YDIF/(double)(UCS_YMAX-UCS_YMIN));

  initpuntos();
  initmatgs0();

  for(int i=0;19>i;i++)
    ang.v[i]=0;

  THETA=45;
  PHI=55;
  RHO=0;

  mview=mmview(THETA,PHI,RHO);

  CBackground=CBLACK;
  CHand=CGREEN;
  CSelection=CCYAN;
  COrigin=CYELLOW;
  CUcsOrigin=CWHITE;

  miembro=mNada;
  flag2=F;
  flag3=F;
  flag4=F;
  flag5=F;
  flag6=F;
}

/*****************************************************************/
// Funcion: ~THandChildSim()
// Destructor de clase
/*****************************************************************/

THandChildSim::~THandChildSim()
{

}

/*****************************************************************/
// Funcion: CleanupWindow()
/*****************************************************************/

void THandChildSim::CleanupWindow()
{
  KillTimer(Timer);
}

/*****************************************************************/
// Funcion: SetupWindow()
// Inicializa el elemento visual
/*****************************************************************/

void THandChildSim::SetupWindow()
{
```

```
  TMDIChild::SetupWindow();

  Timer=0;
  if(SetTimer(1,TIME1))
    Timer=1;

  TClientDC dc(HWindow);
  memDC = new TMemoryDC(dc);
  memDC->SelectObject(TBitmap(dc,W1,HE));
}

/*************************************************************************/
// Funcion: Paint(TDC& dc,bool erase,TRect& rect)
// Trazado de la pantalla
/*************************************************************************/

void THandChildSim::Paint(TDC& dc,bool,TRect&)
{
   memDC->FillRect(GetClientRect(),TBrush(CBackground));
  ucs_icon(*memDC,CUcsOrigin,0.5*U);
  origen(*memDC,COrigin,U);
  mano(*memDC,miembro);

  pt3d(*memDC,CRED,p_in(XM-1,YM-1,ZM-1));
  pt3d(*memDC,CRED,p_in(XM-1,YM-1,ZM+1));
  pt3d(*memDC,CRED,p_in(XM-1,YM+1,ZM-1));
  pt3d(*memDC,CRED,p_in(XM-1,YM+1,ZM+1));
  pt3d(*memDC,CRED,p_in(XM+1,YM-1,ZM-1));
  pt3d(*memDC,CRED,p_in(XM+1,YM-1,ZM+1));
  pt3d(*memDC,CRED,p_in(XM+1,YM+1,ZM-1));
  pt3d(*memDC,CRED,p_in(XM+1,YM+1,ZM+1));

  dc.BitBlt(GetClientRect(),*memDC,TPoint(0,0));
}

/*************************************************************************/
// FUNCIONES DE LA TABLA DE RESPUESTA
/*************************************************************************/

/*************************************************************************/
// Funcion: CanClose()
// Responde ante la solicitud de cierre de la ventana
/*************************************************************************/

bool THandChildSim::CanClose()
{
  bool bResult;
  bResult=MessageBox("¿Desea cerrar la ventana del simulador?",
             "Manipulador Antropomórfico Teleoperado",
              MB_YESNO|MB_ICONQUESTION)==IDYES;
  if(bResult)
    estsim=esDisabled;
  return bResult;
}

/*************************************************************************/
// Funcion: CmPrgPause()
// Responde al mensaje CM_PROGRAM_PAUSE
/*************************************************************************/

void THandChildSim::CmPrgPause()
{
  if(estado==eActivo)
    estado=ePausa;
  else
    if(estado==ePausa)
      estado=eActivo;
  Invalidate();
}

/*************************************************************************/
// Funcion: CmPrgPauseEnabler(TCommandEnabler& tce)
// Responde al mensaje CM_PROGRAM_PAUSE
/*************************************************************************/

void THandChildSim::CmPrgPauseEnabler(TCommandEnabler& tce)
{
  if(estado==eInactivo)
    tce.Enable(F);
  else
    if(estado==ePausa)
      tce.SetText("&Continuar");
    else
      tce.SetText("&Pausa");
}

/*************************************************************************/
// Funcion: CmPrgRun()
// Responde al mensaje CM_PROGRAM_RUN
/*************************************************************************/

void THandChildSim::CmPrgRun()
{
  estado=eActivo;
  miembro=mNada;
  for(int i=0;19>i;i++)
    ang.v[i]=0;
  Invalidate();
}

/*************************************************************************/
// Funcion: CmPrgRunEnabler(TCommandEnabler& tce)
// Responde al mensaje CM_PROGRAM_RUN
/*************************************************************************/

void THandChildSim::CmPrgRunEnabler(TCommandEnabler& tce)
{
  if(estado==eActivo||estado==ePausa)
    tce.Enable(F);
}

/*************************************************************************/
// Funcion: EvChar(uint key,uint repeatCount,uint flags)
// Responde al mensaje EV_WM_CHAR
/*************************************************************************/

void THandChildSim::EvChar(uint key,uint,uint)
{
  uint key2=toupper(key);

  if(estado==eActivo||estado==ePausa)
  {
    switch(miembro)
    {
      case mPalma:

        if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
          key2!='D')
          break;

        if((key2=='Q')&&(ang.v[0]<LSPL1))
          ang.v[0]+=DANG;
        if((key2=='A')&&(ang.v[0]>LIPL1))
          ang.v[0]-=DANG;

        if((key2=='W')&&(ang.v[1]<LSPL2))
          ang.v[1]+=DANG;
        if((key2=='S')&&(ang.v[1]>LIPL2))
          ang.v[1]-=DANG;

        if((key2=='E')&&(ang.v[2]<LSPL3))
          ang.v[2]+=DANG;
        if((key2=='D')&&(ang.v[2]>LIPL3))
          ang.v[2]-=DANG;

        Invalidate();
        break;

      case mPulgar:

        if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
          key2!='D'&&key2!='R'&&key2!='F')
          break;

        if((key2=='Q')&&(ang.v[3]<LSP1))
          ang.v[3]+=DANG;
        if((key2=='A')&&(ang.v[3]>LIP1))
          ang.v[3]-=DANG;

        if((key2=='W')&&(ang.v[4]<LSP2))
          ang.v[4]+=DANG;
        if((key2=='S')&&(ang.v[4]>LIP2))
          ang.v[4]-=DANG;

        if((key2=='E')&&(ang.v[5]<LSP3))
          ang.v[5]+=DANG;
        if((key2=='D')&&(ang.v[5]>LIP3))
          ang.v[5]-=DANG;

        if((key2=='R')&&(ang.v[6]<LSP4))
          ang.v[6]+=DANG;
        if((key2=='F')&&(ang.v[6]>LIP4))
          ang.v[6]-=DANG;

        Invalidate();
        break;

      case mIndice:

        if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
          key2!='D')
          break;

        if((key2=='Q')&&(ang.v[7]<LSD1))
          ang.v[7]+=DANG;
        if((key2=='A')&&(ang.v[7]>LID1))
```

```
        ang.v[7]-=DANG;

      if((key2=='W')&&(ang.v[8]<LSD2))
        ang.v[8]+=DANG;
      if((key2=='S')&&(ang.v[8]>LID2))
        ang.v[8]-=DANG;

      if((key2=='E')&&(ang.v[9]<LSD3))
        ang.v[9]+=DANG,ang.v[10]+=DANG;
      if((key2=='D')&&(ang.v[9]>LID3))
        ang.v[9]-=DANG,ang.v[10]-=DANG;

      Invalidate();
      break;

    case mMedio:

      if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
        key2!='D')
        break;

      if((key2=='Q')&&(ang.v[11]<LSD1))
        ang.v[11]+=DANG;
      if((key2=='A')&&(ang.v[11]>LID1))
        ang.v[11]-=DANG;

      if((key2=='W')&&(ang.v[12]<LSD2))
        ang.v[12]+=DANG;
      if((key2=='S')&&(ang.v[12]>LID2))
        ang.v[12]-=DANG;

      if((key2=='E')&&(ang.v[13]<LSD3))
        ang.v[13]+=DANG,ang.v[14]+=DANG;
      if((key2=='D')&&(ang.v[13]>LID3))
        ang.v[13]-=DANG,ang.v[14]-=DANG;

      Invalidate();
      break;

    case mAnular:

      if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
        key2!='D')
        break;

      if((key2=='Q')&&(ang.v[15]<LSD1))
        ang.v[15]+=DANG;
      if((key2=='A')&&(ang.v[15]>LID1))
        ang.v[15]-=DANG;

      if((key2=='W')&&(ang.v[16]<LSD2))
        ang.v[16]+=DANG;
      if((key2=='S')&&(ang.v[16]>LID2))
        ang.v[16]-=DANG;

      if((key2=='E')&&(ang.v[17]<LSD3))
        ang.v[17]+=DANG,ang.v[18]+=DANG;
      if((key2=='D')&&(ang.v[17]>LID3))
        ang.v[17]-=DANG,ang.v[18]-=DANG;

      Invalidate();
      break;
  }
 }
}

/*************************************************************************/
// Funcion: EvEraseBkgnd(HDC)
// Responde al mensaje EV_WM_ERASEBKGND
/*************************************************************************/

bool THandChildSim::EvEraseBkgnd(HDC)
{
 return TRUE;
}

/*************************************************************************/
// Funcion: EvKeyDown(uint key,uint repeatCount,uint flags)
// Responde al mensaje EV_WM_KEYDOWN
/*************************************************************************/

void THandChildSim::EvKeyDown(uint key,uint,uint)
{
 if(estado==eActivo||estado==ePausa)
 {
  switch(key)
  {
    case VK_TAB:
      if(miembro==mAnular)
        miembro=mPalma;
      else
        miembro++;
      Invalidate();
      break;

    case VK_RIGHT:
      if(!(PHI==0)||(PHI==180)||(PHI==-180))
      {
        THETA+=DANG;
        mview=mmview(THETA,PHI,RHO);
        Invalidate();
        break;
      }
      else
        break;

    case VK_LEFT:
      if(!(PHI==0)||(PHI==180)||(PHI==-180))
      {
        THETA-=DANG;
        mview=mmview(THETA,PHI,RHO);
        Invalidate();
        break;
      }
      else
        break;

    case VK_DOWN:
      PHI+=DANG;
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;

    case VK_UP:
      PHI-=DANG;
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;

    case VK_F5:
      RHO+=DANG;
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;

    case VK_F6:
      RHO-=DANG;
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;

    case VK_F9:
      THETA=45;
      PHI=55;
      RHO=0;
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;

    case VK_ESCAPE:
      miembro=mNada;
      for(int i=0;19>i;i++)
        ang.v[i]=0;
      Invalidate();
      break;
  }
 }
}

/*************************************************************************/
// Funcion: EvTimer(uint timerId)
// Responde al mensaje EV_WM_TIMER
/*************************************************************************/
/*
void THandChildSim::EvTimer(uint)
{
 OVERLAPPED OlpStat = {0};
 uint32 dwCommEvent;
 uint32 dwOvRes;
 uint32 dwRes;
 uint32 dwTimeOut = 10;
 bool bWaitingOnStat = F;

 if(!SetCommMask(hCom,EV_RXFLAG))
 {
   MessageBox("Falló SetCommMask","Mensaje",MB_OK|MB_ICONSTOP);
 }

 //CloseHandle(OlpStat.hEvent);

 OlpStat.hEvent = CreateEvent(NULL,T,F,NULL);
 if(OlpStat.hEvent==NULL)
 {
   MessageBox("Falló CreateEvent (Status)","Mensaje",MB_OK|MB_ICONSTOP);
 }
```

```
  if((estado==eActivo)&&(estprt==epEnabled))
  {
    if(!bWaitingOnStat)
    {
      if(!WaitCommEvent(hCom,&dwCommEvent,&OlpStat))
      {
        if(GetLastError()==ERROR_IO_PENDING)
          bWaitingOnStat=T;
      }
      else
      {
        ReadBuffer();
        PurgeComm(hCom,PURGE_RXCLEAR);
      }
    }

    if(bWaitingOnStat)
    {
      dwRes=WaitForSingleObject(OlpStat.hEvent,dwTimeOut);
      switch(dwRes)
      {
        case WAIT_OBJECT_0:
          if(GetOverlappedResult(hCom,&OlpStat,&dwOvRes,F))
          {
            ReadBuffer();
            PurgeComm(hCom,PURGE_RXCLEAR);
            //bWaitingOnStat=F;
          }
          break;

        default:
          //bWaitingOnStat=F;
          //CloseHandle(OlpStat.hEvent);
          break;
      }
    }

    CloseHandle(OlpStat.hEvent);
  }
}

/*************************************************************************/
// Funcion: EvTimer(uint timerId)
// Responde al mensaje EV_WM_TIMER
/*************************************************************************/

void THandChildSim::EvTimer(uint)
{
  static double t=0;
  double x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4;
  v4 ww;

  char s[101];
  uint8 y[19];

  if(estado==eActivo)
  {
    x1=C(1*t)/2;
    y1=S(1*t)/2;
//  z1=S(1*t)/2;

//  x2=C(1*t)/2;
    y2=-C(1*t)/2;
    z2=S(1*t)/2;

//  x3=C(1*t)/2;
    y3=-C(1*t+20)/2;
    z3=S(1*t+20)/2;

//  x4=C(1*t)/2;
    y4=-C(1*t+40)/2;
    z4=S(1*t+40)/2,

//  ww=cin_inv_pulg(L1P,L2P,L3P,NULO,v_in(x1,y1+1,0),-135);
    ww=cin_inv_dedo(L1P,L2P,L3P,NULO,v_in(x1-.5,y1+1.2,-.5));
    ang.v[3]=ww.v[0];
    ang.v[4]=ww.v[1];
    ang.v[5]=ww.v[2];
    ang.v[6]=ww.v[2];

    ww=cin_inv_dedo(L1I,L2I,L3I,NULO,v_in(-.4,y2+1.2,z2-.5));
    ang.v[7]=ww.v[0];
    ang.v[8]=ww.v[1];
    ang.v[9]=ww.v[2];
    ang.v[10]=ww.v[2];

    ww=cin_inv_dedo(L1M,L2M,L3M,NULO,v_in(0,y3+1.2,z3-.5));
    ang.v[11]=ww.v[0];
    ang.v[12]=ww.v[1];
    ang.v[13]=ww.v[2];
    ang.v[14]=ww.v[2];

    ww=cin_inv_dedo(L1A,L2A,L3A,NULO,v_in(.4,y4+1.2,z4-.5));
    ang.v[15]=ww.v[0];
    ang.v[16]=ww.v[1];
    ang.v[17]=ww.v[2];
    ang.v[18]=ww.v[2];

    if(estprt == epEnabled)
    {
      for(int i=3;19>i;i++)
        y[i]=(uint8)interpol(anginf.v[i],angsup.v[i],127,0,ang.v[i]);

      sprintf(s,"%c%c%c%c%c%c%c%c%c%c%c%c%c%c",0xf7,y[3],y[4],y[5],y[6],
          y[7],y[8],y[9],y[11],y[12],y[13],y[15],y[16],y[17],0xf0);

      WriteBuffer(s,strlen(s));
    }

    Invalidate();
    t+=36;
  }
}

/*************************************************************************/
// FUNCIONES DEL MODELO MATEMATICO
/*************************************************************************/

/*************************************************************************/
// Funcion: initpuntos(void)
// Inicializa puntos clave de la mano
/*************************************************************************/

void THandChildSim::initpuntos(void)
{
// eje x para el origen del sistema inercial
  I=v_in(1,0,0);
// eje y para el origen del sistema inercial
  J=v_in(0,1,0);
// eje z para el origen del sistema inercial
  K=v_in(0,0,1);

// palma: origen
  Q[0]=p_in(XPL,YPL,ZPL);

// eje x para el origen del pulgar
  IP=v_unit(v_in(C(B)*C(A),C(B)*S(A),S(B)));
// eje y para el origen del pulgar
  JP=v_unit(v_in(-S(A),C(A),0));
// eje z para el origen del pulgar
  KP=v_unit(v_in(-S(B)*C(A),-S(B)*S(A),C(B)));

// vector nulo
  NULO=v_in(0,0,0);

// pulgar: origen falange
  Q[1]=p_in(XP,YP,ZP);
// pulgar: origen falangina
  Q[2]=p_in(XP+L1P*JP.v[0],YP+L1P*JP.v[1],ZP+L1P*JP.v[2]);
// pulgar: origen falangeta
  Q[3]=p_in(XP+(L1P+L2P)*JP.v[0],YP+(L1P+L2P)*JP.v[1],
        ZP+(L1P+L2P)*JP.v[2]);
// pulgar: extremo falangeta
  Q[4]=p_in(XP+(L1P+L2P+L3P)*JP.v[0],YP+(L1P+L2P+L3P)*JP.v[1],
        ZP+(L1P+L2P+L3P)*JP.v[2]);

// indice: origen falange
  Q[5]=p_in(XI,YI,ZI);
// indice: origen falangina
  Q[6]=p_in(XI,YI+L1I,ZI);
// indice: origen falangeta
  Q[7]=p_in(XI,YI+L1I+L2I,ZI);
// indice: extremo falangeta
  Q[8]=p_in(XI,YI+L1I+L2I+L3I,ZI);

// medio: origen falange
  Q[9]=p_in(XM,YM,ZM);
// medio: origen falangina
  Q[10]=p_in(XM,YM+L1M,ZM);
// medio: origen falangeta
  Q[11]=p_in(XM,YM+L1M+L2M,ZM);
// medio: extremo falangeta
  Q[12]=p_in(XM,YM+L1M+L2M+L3M,ZM);

// anular: origen falange
  Q[13]=p_in(XA,YA,ZA);
// anular: origen falangina
  Q[14]=p_in(XA,YA+L1A,ZA);
// anular: origen falangeta
  Q[15]=p_in(XA,YA+L1A+L2A,ZA);
// anular: extremo falangeta
  Q[16]=p_in(XA,YA+L1A+L2A+L3A,ZA);

// palma
  bb[0]=v48_in(LPAL/1.5,LPAL,HH,LPAL);
```

```
// pulgar: falange
  bb[1]=v48_in(WW,WW,HH,L1P);
// pulgar: falangina
  bb[2]=v48_in(WW,WW,HH,L2P);
// pulgar: falangeta
  tt[0]=v412_in(WW,HH,WW/2,HH/2,L3P,0.6);

// indice: falange
  bb[3]=v48_in(WW,WW,HH,L1I);
// indice: falangina
  bb[4]=v48_in(WW,WW,HH,L2I);
// indice: falangeta
  tt[1]=v412_in(WW,HH,WW/2,HH/2,L3I,0.6);

// medio: falange
  bb[5]=v48_in(WW,WW,HH,L1M);
// medio: falangina
  bb[6]=v48_in(WW,WW,HH,L2M);
// medio: falangeta
  tt[2]=v412_in(WW,HH,WW/2,HH/2,L3M,0.6);

// anular: falange
  bb[7]=v48_in(WW,WW,HH,L1A);
// anular: falangina
  bb[8]=v48_in(WW,WW,HH,L2A);
// anular: falangeta
  tt[3]=v412_in(WW,HH,WW/2,HH/2,L3A,0.6);

  anginf.v[0]=LIPL1;angsup.v[0]=LSPL1;
  anginf.v[1]=LIPL2;angsup.v[1]=LSPL2;
  anginf.v[2]=LIPL3;angsup.v[2]=LSPL3;

  anginf.v[3]=LIP1;angsup.v[3]=LSP1;
  anginf.v[4]=LIP2;angsup.v[4]=LSP2;
  anginf.v[5]=LIP3;angsup.v[5]=LSP3;
  anginf.v[6]=LIP4;angsup.v[6]=LSP4;

  anginf.v[7]=LID1;angsup.v[7]=LSD1;
  anginf.v[8]=LID2;angsup.v[8]=LSD2;
  anginf.v[9]=LID3;angsup.v[9]=LSD3;
  anginf.v[10]=LID3;angsup.v[10]=LSD3;

  anginf.v[11]=LID1;angsup.v[11]=LSD1;
  anginf.v[12]=LID2;angsup.v[12]=LSD2;
  anginf.v[13]=LID3;angsup.v[13]=LSD3;
  anginf.v[14]=LID3;angsup.v[14]=LSD3;

  anginf.v[15]=LID1;angsup.v[15]=LSD1;
  anginf.v[16]=LID2;angsup.v[16]=LSD2;
  anginf.v[17]=LID3;angsup.v[17]=LSD3;
  anginf.v[18]=LID3;angsup.v[18]=LSD3;
}

/*************************************************************************/
// Funcion: mgs0(const v4& p)
// Define la matriz gs0
//
// p: punto base del eslabon
// Posicion inicial de la palma
// <Q[0]> palma
// Posicion inicial de la falange proximal
// <Q[5]> indice, <Q[9]> medio, <Q[13]> anular
// Posicion inicial de la falange media
// <Q[6]> indice, <Q[10]> medio, <Q[14]> anular
// Posicion inicial de la falange distal
// <Q[7]> indice, <Q[11]> medio, <Q[15]> anular
// Posicion inicial del extremo de la falange distal
// <Q[8]> indice, <Q[12]> medio, <Q[16]> anular
/*************************************************************************/

m4 THandChildSim::mgs0(const v4& p)
{
  double x,y,z;
  x=p.v[0];y=p.v[1];z=p.v[2];

  m4 m={{{1,0,0,x},
         {0,1,0,y},
         {0,0,1,z},
         {0.0.0,1}}};

  return m;
}

/*************************************************************************/
// Funcion: mgsp0(const v4& p)
// Define la matriz gsp0
// Posicion inicial de los eslabones del pulgar
//
// p: punto base del eslabon
// <Q[1]> f. prox., <Q[2]> f. med., <Q[3]> f. dist., <Q[4]> ext. dist.
/*************************************************************************/

m4 THandChildSim::mgsp0(const v4& p)
{
  double x,y,z;
  x=p.v[0];y=p.v[1];z=p.v[2];

  m4 m={{{IP.v[0],JP.v[0],KP.v[0],x},
         {IP.v[1],JP.v[1],KP.v[1],y},
         {IP.v[2],JP.v[2],KP.v[2],z},
         {  0,    0,     0,1}}};

  return m;
}

/*************************************************************************/
// Funcion: initmatgs0(void)
// Inicializa las matrices gs0 (posicion inicial)
/*************************************************************************/

void THandChildSim::initmatgs0(void)
{
  gs0[0]=mgs0(Q[0]);    //palma

  gs0[1]=mgsp0(Q[1]);   //pulgar(falange proximal)
  gs0[2]=mgsp0(Q[2]);   //pulgar(falange media)
  gs0[3]=mgsp0(Q[3]);   //pulgar(falange distal)

  gs0[4]=mgs0(Q[5]);    //indice(falange proximal)
  gs0[5]=mgs0(Q[6]);    //indice(falange media)
  gs0[6]=mgs0(Q[7]);    //indice(falange distal)

  gs0[7]=mgs0(Q[9]);    //medio(falange proximal)
  gs0[8]=mgs0(Q[10]);   //medio(falange media)
  gs0[9]=mgs0(Q[11]);   //medio(falange distal)

  gs0[10]=mgs0(Q[13]);  //anular(falange proximal)
  gs0[11]=mgs0(Q[14]);  //anular(falange media)
  gs0[12]=mgs0(Q[15]);  //anular(falange distal)

  gs0[13]=mgsp0(Q[4]);  //pulgar(extremo falange distal)
  gs0[14]=mgs0(Q[8]);   //indice(extremo falange distal)
  gs0[15]=mgs0(Q[12]);  //medio(extremo falange distal)
  gs0[16]=mgs0(Q[16]);  //anular(extremo falange distal)
}

/*************************************************************************/
// Funcion: ee1(double a,const v4& p)
// Calcula la matriz de transformacion No. 1 (ROLL)
// Rotacion en el eje "y"
//
// a: angulo de rotacion en grados
// p: punto base del eslabon
// <Q[0]> palma
/*************************************************************************/

m4 THandChildSim::ee1(double a,const v4& p)
{
  double s,c,v,x,z;
  s=S(a);c=C(a);v=1-c;
  x=p.v[0];z=p.v[2];

  m4 m={{{ c,0,s,x*v-z*s},
         { 0,1,0,    0},
         {-s,0,c,x*s+z*v},
         { 0,0,0,    1}}};

  return m;
}

/*************************************************************************/
// Funcion: ee2(double a,const v4& p)
// Calcula la matriz de transformacion No. 2 (PITCH)
// Rotacion en el eje "x"
//
// a: angulo de rotacion en grados
// p: punto base del eslabon
// <Q[0]> palma
// <Q[5]> prox. indice, <Q[9]> prox. medio,  <Q[13]> prox. anular
// <Q[6]> med.indice,   <Q[10]> med. medio,  <Q[14]> med. anular
// <Q[7]> dist. indice, <Q[11]> dist. medio, <Q[15]> dist. anular
/*************************************************************************/

m4 THandChildSim::ee2(double a,const v4& p)
{
  double s,c,v,y,z;
  s=S(a);c=C(a);v=1-c;
  y=p.v[1];z=p.v[2];

  m4 m={{{1,0, 0,    0},
         {0,c,-s, y*v+z*s},
         {0,s, c,-y*s+z*v},
         {0,0, 0,    1}}};

  return m;
}
```

```
/*************************************************************************/
// Funcion: ee3(double a,const v4& p)
// Calcula la matriz de transformacion No. 3 (YAW)
// Rotacion en el eje "z"
//
// a: angulo de rotacion en grados
// p: punto base del eslabon
// <Q[0]> palma, <Q[5]> indice, <Q[9]> medio, <Q[13]> anular
/*************************************************************************/

m4 THandChildSim::ee3(double a,const v4& p)
{
  double s,c,v,x,y;
  s=S(a);c=C(a);v=1-c;
  x=p.v[0];y=p.v[1];

  m4 m={{{c,-s,0, x*v+y*s},
         {s, c,0,-x*s+y*v},
         {0, 0,1,     0},
         {0, 0,0,     1}}};

  return m;
}

/*************************************************************************/
// Funcion: ee4(double a,const v4& w,const v4& p)
// Calcula la matriz de transformacion No. 4
// Rotacion en el eje "z" y "x'" de cualquier eslabon del pulgar
//
// a: angulo de rotacion en grados
// w: vector que representa el eje de rotacion
// <IP> eje x', <KP> eje z'
// p:   punto base del eslabon
// <Q[i]> f. proximal, <Q[2]> f. media, <Q[3]> f. distal
/*************************************************************************/

m4 THandChildSim::ee4(double a,const v4& w,const v4& p)
{
  double s,c,v,w1,w2,w3,x,y,z;
  s=S(a);c=C(a);v=1-c;
  w1=w.v[0];w2=w.v[1];w3=w.v[2];
  x=p.v[0];y=p.v[1];z=p.v[2];

  m4 m={{{w1*w1*v+c,
          w1*w2*v-w3*s,
          w1*w3*v+w2*s,
          x*v*(1-w1*w1)+y*(s*w3-w1*w2*v)+z*(-s*w2-w1*w3*v)},

         {w1*w2*v+w3*s,
          w2*w2*v+c,
          w2*w3*v-w1*s,
          x*(-s*w3-w1*w2*v)+y*v*(1-w2*w2)+z*(s*w1-w2*w3*v)},

         {w1*w3*v-w2*s,
          w2*w3*v+w1*s,
          w3*w3*v+c,
          x*(s*w2-w1*w3*v)+y*(-s*w1-w2*w3*v)+z**v*(1-w3*w3)},

         {0,
          0,
          0,
          1}}};
  return m;
}

/*************************************************************************/
// Funcion: mg(const v19& a)
// Calcula las matrices g
//
// a: vector de todos los angulos de las articulaciones
/*************************************************************************/

void THandChildSim::mg(const v19& a)
{
//palma
  e[0]=ee1(a.v[0],Q[0]);
  e[1]=ee2(a.v[1],Q[0]);
  e[2]=ee3(a.v[2],Q[0]);
//pulgar
  e[3]=ee4(a.v[3],KP,Q[1]);
  e[4]=ee4(a.v[4],IP,Q[1]);
  e[5]=ee4(a.v[5],IP,Q[2]);
  e[6]=ee4(a.v[6],IP,Q[3]);
//indice
  e[7]=ee3(a.v[7],Q[5]);
  e[8]=ee2(a.v[8],Q[5]);
  e[9]=ee2(a.v[9],Q[6]);
  e[10]=ee2(a.v[10],Q[7]);
//medio
  e[11]=ee3(a.v[11],Q[9]);
  e[12]=ee2(a.v[12],Q[9]);
  e[13]=ee2(a.v[13],Q[10]);
  e[14]=ee2(a.v[14],Q[11]);
//anular
  e[15]=ee3(a.v[15],Q[13]);
  e[16]=ee2(a.v[16],Q[13]);
  e[17]=ee2(a.v[17],Q[14]);
  e[18]=ee2(a.v[18],Q[15]);

//palma
  gs[0]=mm_mult(mm_mult(e[0],e[1]),e[2]);
//pulgar
  gs[1]=mm_mult(mm_mult(gs[0],e[3]),e[4]);
  gs[2]=mm_mult(gs[1],e[5]);
  gs[3]=mm_mult(gs[2],e[6]);
//;ndice
  gs[4]=mm_mult(mm_mult(gs[0],e[7]),e[8]);
  gs[5]=mm_mult(gs[4],e[9]);
  gs[6]=mm_mult(gs[5],e[10]);
//medio
  gs[7]=mm_mult(mm_mult(gs[0],e[11]),e[12]);
  gs[8]=mm_mult(gs[7],e[13]);
  gs[9]=mm_mult(gs[8],e[14]);
//anular
  gs[10]=mm_mult(mm_mult(gs[0],e[15]),e[16]);
  gs[11]=mm_mult(gs[10],e[17]);
  gs[12]=mm_mult(gs[11],e[18]);

//palma
  g[0]=mm_mult(gs[0],gs0[0]);
//pulgar
  g[1]=mm_mult(gs[1],gs0[1]);
  g[2]=mm_mult(gs[2],gs0[2]);
  g[3]=mm_mult(gs[3],gs0[3]);
//indice
  g[4]=mm_mult(gs[4],gs0[4]);
  g[5]=mm_mult(gs[5],gs0[5]);
  g[6]=mm_mult(gs[6],gs0[6]);
//medio
  g[7]=mm_mult(gs[7],gs0[7]);
  g[8]=mm_mult(gs[8],gs0[8]);
  g[9]=mm_mult(gs[9],gs0[9]);
//anular
  g[10]=mm_mult(gs[10],gs0[10]);
  g[11]=mm_mult(gs[11],gs0[11]);
  g[12]=mm_mult(gs[12],gs0[12]);
}

/*************************************************************************/
// Funcion: m_rot(double a,const v4& b,double x,double y,double z)
// Calcula matrices de rotacion
//
// a: angulo de rotacion
// b: vector que representa al eje de rotacion
// x: coordenada x del punto de anclaje del sist. de ref.
// y: coordenada y del punto de anclaje del sist. de ref.
// z: coordenada z del punto de anclaje del sist. de ref.
/*************************************************************************/

m4 THandChildSim::m_rot(double a,const v4& b,double x,double y,double z)
{
  v4 w;
  double s,c,v;
  s=S(a);c=C(a);v=1-c;
  w=v_unit(b);

  m4 m={{{w.v[0]*w.v[0]*v+c,
          w.v[0]*w.v[1]*v-w.v[2]*s,
          w.v[0]*w.v[2]*v+w.v[1]*s,
          x},

         {w.v[0]*w.v[1]*v+w.v[2]*s,
          w.v[1]*w.v[1]*v+c,
          w.v[1]*w.v[2]*v-w.v[0]*s,
          y},

         {w.v[0]*w.v[2]*v-w.v[1]*s,
          w.v[1]*w.v[2]*v+w.v[0]*s,
          w.v[2]*w.v[2]*v+c,
          z},

         {0,
          0,
          0,
          1}}};

  return m;
}

/*************************************************************************/
// FUNCIONES DE PARA OPERAR CON MATRICES
/*************************************************************************/

/*************************************************************************/
// Funcion: m_esc(double esc,const m4& a)
```

```
// Multiplica un escalar por una matriz
//
// esc: escalar por el cual se multiplica la matriz
// a: matriz objeto de la operacion
/*************************************************************************/

m4 THandChildSim::m_esc(double esc,const m4& a)
{
  int i,j;
  m4 m;
  for(i=0;ORD>i;i++)
    for(j=0;ORD>j;j++)
      m.m[i][j]=esc*a.m[i][j];
  return m;
}

/*************************************************************************/
// Funcion: m_transp(const m4& a)
// Obtiene la transpuesta de una matriz
//
// a: matriz objeto de la operacion
/*************************************************************************/

m4 THandChildSim::m_transp(const m4& a)
{
  int i,j;
  m4 m;
  for(i=0;ORD>i;i++)
    for(j=0;ORD>j;j++)
      m.m[j][i]=a.m[i][j];
  return m;
}

/*************************************************************************/
// Funcion: mm_mult(const m4& a,const m4& b)
// Multiplica una matriz por otra (matrices cuadradas de tamano ORD)
//
// a: matriz que funciona como primer factor
// b: matriz que funciona como segundo factor
/*************************************************************************/

m4 THandChildSim::mm_mult(const m4& a,const m4& b)
{
  int i,j,k;
  m4 m;
  for(i=0;ORD>i;i++)
    for(j=0;ORD>j;j++)
    {
      m.m[i][j]=0;
      for(k=0;ORD>k;k++)
        m.m[i][j]=m.m[i][j]+a.m[i][k]*b.m[k][j];
    }
  return m;
}

/*************************************************************************/
// Funcion: mv_mult(const m4& a,const v4& b)
// Multiplica matrices por vectores
//
// a: matriz que funciona como primer factor
// b: vector que funciona como segundo factor
/*************************************************************************/

v4 THandChildSim::mv_mult(const m4& a,const v4& b)
{
  int i,k;
  v4 v;
  for(i=0;ORD>i;i++)
  {
    v.v[i]=0;
    for(k=0;ORD>k;k++)
      v.v[i]=v.v[i]+a.m[i][k]*b.v[k];
  }
  return v;
}

/*************************************************************************/
// Funcion: m_inv(const m4& a)
// Obtiene la inversa de una matriz tipo g (caso especial)
//
// a: matriz objeto de la inversion
/*************************************************************************/

m4 THandChildSim::m_inv(const m4& a)
{
  int i,j,k;
  m4 m=a;
  for(i=0;(ORD-1)>i;i++)
    for(j=0;(ORD-1)>j;j++)
      m.m[j][i]=a.m[i][j];
  for(i=0;(ORD-1)>i;i++)
  {
    m.m[i][ORD-1]=0;
    for(k=0;(ORD-1)>k;k++)
      m.m[i][ORD-1]=m.m[i][ORD-1]-m.m[i][k]*a.m[k][ORD-1];
  }
  return m;
}

/*************************************************************************/
// FUNCIONES PARA OPERAR CON VECTORES
/*************************************************************************/

/*************************************************************************/
// Funcion: p_in(double x,double y,double z,double k = 1)
// Ingresa datos en un punto
//
// x: componente x del punto
// y: componente y del punto
// z: componente z del punto
// k: cuarto elemento comodin
/*************************************************************************/

v4 THandChildSim::p_in(double x,double y,double z,double k)
{
  v4 v={{x,y,z,k}};
  return v;
}

/*************************************************************************/
// Funcion: v_in(double x,double y,double z,double k = 0)
// Ingresa datos en un vector
//
// x: componente x del vector
// y: componente y del vector
// z: componente z del vector
// k: cuarto elemento comodin
/*************************************************************************/

v4 THandChildSim::v_in(double x,double y,double z,double k)
{
  v4 v={{x,y,z,k}};
  return v;
}

/*************************************************************************/
// Funcion: v_sum(const v4& a,const v4& b,bool oper = T)
// Suma dos vectores en forma algebraica
//
// a: vector que funciona como primer sumando
// b: vector que funciona como segundo sumando
// oper: tipo de operacion, <T> suma, <F> resta
/*************************************************************************/

v4 THandChildSim::v_sum(const v4& a,const v4& b,bool oper)
{
  int i;
  v4 v;
  if(oper)
    for(i=0;(ORD-1)>i;i++)
      v.v[i]=a.v[i]+b.v[i];
  else
    for(i=0;(ORD-1)>i;i++)
      v.v[i]=a.v[i]-b.v[i];
  v.v[ORD-1]=0;
  return v;
}

/*************************************************************************/
// Funcion: v_esc(double esc,const v4& a)
// Calcula el producto de un escalar con un vector
//
// esc: escalar por el cual se multiplica el vector
// a: vector objeto de la operacion
/*************************************************************************/

v4 THandChildSim::v_esc(double esc,const v4& a)
{
  int i;
  v4 v;
  for(i=0;(ORD-1)>i;i++)
    v.v[i]=esc*a.v[i];
  v.v[ORD-1]=0;
  return v;
}

/*************************************************************************/
// Funcion: dot(const v4& a,const v4& b)
// Calcula el producto punto entre dos vectores
//
// a: vector que funciona como primer operando
// b: vector que funciona como segundo operando
/*************************************************************************/

double THandChildSim::dot(const v4& a,const v4& b)
{
```

```
  int i;
  double prod=0;
  for(i=0;(ORD-1)>i;i++)
    prod=prod+a.v[i]*b.v[i];
  return prod;
}

/*************************************************************************/
// Funcion: cross(const v4& a,const v4& b)
// Calcula el producto cruz entre dos vectores
//
// a: vector que funciona como primer operando
// b: vector que funciona como segundo operando
/*************************************************************************/

v4 THandChildSim::cross(const v4& a,const v4& b)
{
  v4 v={{a.v[1]*b.v[2]-a.v[2]*b.v[1],
         a.v[2]*b.v[0]-a.v[0]*b.v[2],
         a.v[0]*b.v[1]-a.v[1]*b.v[0],
         0}};
  return v;
}

/*************************************************************************/
// Funcion: mag2(const v4& a)
// Obtiene el cuadrado de la magnitud de un vector
//
// a: vector objeto de la operacion
/*************************************************************************/

double THandChildSim::mag2(const v4& a)
{
  return dot(a,a);
}

/*************************************************************************/
// Funcion: mag(const v4& a)
// Obtiene la magnitud de un vector
//
// a: vector objeto de la operacion
/*************************************************************************/

double THandChildSim::mag(const v4& a)
{
  return sqrt(mag2(a));
}

/*************************************************************************/
// Funcion: v_unit(const v4& a)
// Normaliza un vector
//
// a: vector objeto de la normalizacion
/*************************************************************************/

v4 THandChildSim::v_unit(const v4& a)
{
  return v_esc((1/mag(a)),a);
}

/*************************************************************************/
// FUNCIONES PARA OPERAR CON ESTRUCTURAS DE DATOS ESPECIALES
/*************************************************************************/

/*************************************************************************/
// Funcion: v48_in(double a,double b,double h,double l)
// Ingresa datos en un elemanto v48 (aristas de barras y prismas trapezoidales)
//
// a: ancho en un extremo de la barra
// b: ancho en el otro extremo de la barra
// h: altura de la barra
// l: longitud de la barra
/*************************************************************************/

v48 THandChildSim::v48_in(double a,double b,double h,double l)
{
  v48 v={{{{ a/2,0,-h/2,1}},
          {{-a/2,0,-b/2,1}},
          {{ a/2,0, h/2,1}},
          {{-a/2,0, b/2,1}},
          {{ b/2,l,-h/2,1}},
          {{-b/2,l,-h/2,1}},
          {{ b/2,l, h/2,1}},
          {{-b/2,l, h/2,1}}}};
  return v;
}

/*************************************************************************/
// Funcion: v412_in(double w1,double h1,double w2,double h2,double l,double r)
// Ingresa datos en un elemanto v412 (aristas prismas de seccion variable)
//
// w1: ancho en la primera seccion de la barra
// h1: altura en la primera seccion de la barra
// w2: ancho en el otro extremo de la barra
// h2: altura en el otro extremo de la barra
// l: longitud de la barra
// r: relacion de longitudes de la primera seccion a la longitud total
/*************************************************************************/

v412 THandChildSim::v412_in(double w1,double h1,double w2,double h2,
                            double l,double r)
{
  double l1=r*l;
  v412 v={{{{ w1/2, 0,-h1/2,1}},
          {{-w1/2, 0,-h1/2,1}},
          {{ w1/2, 0, h1/2,1}},
          {{-w1/2, 0, h1/2,1}},
          {{ w1/2,l1,-h1/2,1}},
          {{-w1/2,l1,-h1/2,1}},
          {{ w1/2,l1, h1/2,1}},
          {{-w1/2,l1, h1/2,1}},
          {{ w2/2, l,-h2/2,1}},
          {{-w2/2, l,-h2/2,1}},
          {{ w2/2, l, h2/2,1}},
          {{-w2/2, l, h2/2,1}}}};
  return v;
}

/*************************************************************************/
// FUNCIONES GRAFICAS
/*************************************************************************/

/*************************************************************************/
// Funcion: mmview(double theta,double& phi,double rho)
// Calcula la matriz de pasa de coord. en 3d a coord. graficas
//
// theta: azimut
// phi:   elevacion
// rho:   giro de vista
/*************************************************************************/

m4 THandChildSim::mmview(double theta,double& phi,double rho)
{
  double c=C(rho),s=S(rho);
  static v4 u;
  v4 v,n,vp;

  if(phi==-180)
    phi=180;
  if(phi>180)
    phi=phi-360;

  n=v_in(S(phi)*C(theta),S(phi)*S(theta),C(phi));
  vp=v_in(0,0,((phi>0)&&(phi<180))? 1 : -1);

  if((n.v[0]==0)&&(n.v[1]==0))
    v=cross(n,u);
  else
  {
    v=v_unit(v_sum(vp,v_esc(dot(vp,n),n),F));
    u=cross(v,n);
  }

  m4 m1={{{ c,s,0,0},
          {-s,c,0,0},
          { 0,0,1,0},
          { 0,0,0,1}}};

  m4 m2={{{u.v[0],u.v[1],u.v[2],0},
          {v.v[0],v.v[1],v.v[2],0},
          {n.v[0],n.v[1],n.v[2],0},
          {   0,   0,   0,1}}};

  m4 m3=mm_mult(m1,m2);

  return m3;
}

/*************************************************************************/
// Funcion: vscreen(double d,const v4& a,const m4& m)
// Transforma las cood. en 3d a coord. de la pantalla
//
// d: distancia de la camara al plano de proyeccion de la imagen
// a: vector que representa un punto en el espacio
// m: matriz de pasa de coord. en 3d a coord. graf.
/*************************************************************************/

TPoint THandChildSim::vscreen(double /*d*/,const v4& a,const m4& m)
{
  v4 v;
  v=mv_mult(m,a);
  TPoint p((int)(XL+(v.v[0]-XMIN)*SCALX),(int)(YB-(v.v[1]-YMIN)*SCALY));
//  TPoint p((int)((((v.v[0]*d)/fabs(v.v[2]))-XMIN)*SCALX),
//           (int)(YDIF-(((v.v[1]*d)/fabs(v.v[2]))-YMIN)*SCALY));
  return p;
}
```

```
/*************************************************************************/
// Funcion: ucs_icon(TDC& dc,TColor color,double l)
// Crea el icono del sistema universal de coordenadas
//
// dc: contexto de dispositivo
// color: color del origen
// l: longitud de los ejes
/*************************************************************************/

void THandChildSim::ucs_icon(TDC& dc,TColor color,double l)
{
  int i;
  v4 v;
  TPoint p2d[7];
  v4 p3d[]={{{0,0,0,1}},      //origen
            {{l,0,0,1}},      //eje X
            {{0,l,0,1}},      //eje Y
            {{0,0,l,1}},      //eje Z
            {{1.3*l,0,0,1}},  //etiqueta de eje "X"
            {{0,1.3*l,0,1}},  //etiqueta de eje "Y"
            {{0,0,1.3*l,1}}}; //etiqueta de eje "Z"

  for(i=0;7>i;i++)
  {
    v=mv_mult(mview,p3d[i]);
    p2d[i].x=(int)(UCS_XL+(v.v[0]-UCS_XMIN)*UCS_SCALX);
    p2d[i].y=(int)(UCS_YB-(v.v[1]-UCS_YMIN)*UCS_SCALY);
  }

  dc.SelectObject(TPen(color));
  dc.MoveTo(p2d[0]);dc.LineTo(p2d[1]);
  dc.MoveTo(p2d[0]);dc.LineTo(p2d[2]);
  dc.MoveTo(p2d[0]);dc.LineTo(p2d[3]);
  dc.SelectObject(TFont("Arial",7));
  dc.SetTextColor(color);
  dc.SetBkColor(CBackground);
  dc.SetTextAlign(TA_CENTER);
  dc.TextOut(p2d[4],"X");
  dc.TextOut(p2d[5],"Y");
  dc.TextOut(p2d[6],"Z");
}

/*************************************************************************/
// Funcion: origen(TDC& dc,TColor color,double l)
// Crea el origen en la pantalla
//
// dc: contexto de dispositivo
// color: color del origen
// l: longitud de los ejes
/*************************************************************************/

void THandChildSim::origen(TDC& dc,TColor color,double l)
{
  int i;
  TPoint p2d[7];
  v4 p3d[]={{{0,0,0,1}},      //origen
            {{l,0,0,1}},      //eje X
            {{0,l,0,1}},      //eje Y
            {{0,0,l,1}},      //eje Z
            {{1.3*l,0,0,1}},  //etiqueta de eje "X"
            {{0,1.3*l,0,1}},  //etiqueta de eje "Y"
            {{0,0,1.3*l,1}}}; //etiqueta de eje "Z"

  for(i=0;7>i;i++)
    p2d[i]=vscreen(D,p3d[i],mview);

  dc.SelectObject(TPen(color));
  dc.MoveTo(p2d[0]);dc.LineTo(p2d[1]);
  dc.MoveTo(p2d[0]);dc.LineTo(p2d[2]);
  dc.MoveTo(p2d[0]);dc.LineTo(p2d[3]);
  dc.SelectObject(TFont("Arial",8));
  dc.SetTextColor(color);
  dc.SetBkColor(CBackground);
  dc.SetTextAlign(TA_CENTER);
  dc.TextOut(p2d[4],"X");
  dc.TextOut(p2d[5],"Y");
  dc.TextOut(p2d[6],"Z");
}

/*************************************************************************/
// Funcion: sistref(TDC& dc,TColor color,double l,const m4& rot)
// Crea sistemas de referencia en la pantalla
//
// dc: contexto de dispositivo
// color: color del origen
// l: longitud de los ejes
// rot: matriz de rotacion asociado a dicho sistema
/*************************************************************************/

void THandChildSim::sistref(TDC& dc,TColor color,double l,const m4& rot)
{
  int i;
  TPoint p2d[4];
  v4 p3d[]={{{0,0,0,1}},
            {{l,0,0,1}},
            {{0,l,0,1}},
            {{0,0,l,1}}};

  for(i=0;4>i;i++)
    p2d[i]=vscreen(D,mv_mult(rot,p3d[i]),mview);

  dc.SelectObject(TPen(color));
  dc.MoveTo(p2d[0]);dc.LineTo(p2d[1]);
  dc.MoveTo(p2d[0]);dc.LineTo(p2d[2]);
  dc.MoveTo(p2d[0]);dc.LineTo(p2d[3]);
}

/*************************************************************************/
// Funcion: barra(TDC& dc,v48& p3d,const m4& rot)
// Crea una barra en el espacio
//
// dc: contexto de dispositivo
// p3d: coordenadas de las aristas de la barra
// rot: matriz de rotacion asociada a la barra
/*************************************************************************/

void THandChildSim::barra(TDC& dc,v48& p3d,const m4& rot)
{
  int i;
  TPoint p2d[8];

  for(i=0;8>i;i++)
    p2d[i]=vscreen(D,mv_mult(rot,p3d.v[i]),mview);

  dc.MoveTo(p2d[0]);
  dc.LineTo(p2d[4]);
  dc.LineTo(p2d[6]);
  dc.LineTo(p2d[2]);
  dc.LineTo(p2d[3]);
  dc.LineTo(p2d[7]);
  dc.LineTo(p2d[5]);
  dc.LineTo(p2d[1]);
  dc.LineTo(p2d[0]);
  dc.LineTo(p2d[2]);
  dc.MoveTo(p2d[1]);
  dc.LineTo(p2d[3]);
  dc.MoveTo(p2d[4]);
  dc.LineTo(p2d[5]);
  dc.MoveTo(p2d[6]);
  dc.LineTo(p2d[7]);
}

/*************************************************************************/
// Funcion: tip(TDC& dc,v412& p3d,const m4& rot)
// Crea la punta de los dedos
//
// dc: contexto de dispositivo
// p3d: coordenadas de las aristas de la barra
// rot: matriz de rotacion asociada a la barra
/*************************************************************************/

void THandChildSim::tip(TDC& dc,v412& p3d,const m4& rot)
{
  int i;
  TPoint p2d[12];

  for(i=0;12>i;i++)
    p2d[i]=vscreen(D,mv_mult(rot,p3d.v[i]),mview);

  dc.MoveTo(p2d[0]);
  dc.LineTo(p2d[4]);
  dc.LineTo(p2d[8]);
  dc.LineTo(p2d[10]);
  dc.LineTo(p2d[6]);
  dc.LineTo(p2d[2]);
  dc.LineTo(p2d[3]);
  dc.LineTo(p2d[7]);
  dc.LineTo(p2d[11]);
  dc.LineTo(p2d[9]);
  dc.LineTo(p2d[5]);
  dc.LineTo(p2d[1]);
  dc.LineTo(p2d[0]);
  dc.LineTo(p2d[2]);
  dc.MoveTo(p2d[1]);
  dc.LineTo(p2d[3]);
  dc.MoveTo(p2d[8]);
  dc.LineTo(p2d[9]);
  dc.MoveTo(p2d[10]);
  dc.LineTo(p2d[11]);

//  dc.MoveTo(p2d[4]);
//  dc.LineTo(p2d[5]);
//  dc.LineTo(p2d[7]);
//  dc.LineTo(p2d[6]);
//  dc.LineTo(p2d[4]);
```

```
}

/*************************************************************************/
// Funcion: palma(TDC& dc,TColor color)
// Dibuja la palma
//
// dc: contexto de dispositivo
// color: color de la palma
/*************************************************************************/

void THandChildSim::palma(TDC& dc,TColor color)
{
  dc.SelectObject(TPen(color));
  barra(dc,bb[0],g[0]);

//  sistref(dc,COrigin,R,g[0]);
}

/*************************************************************************/
// Funcion: pulgar(TDC& dc,TColor color)
// Dibuja el dedo pulgar
//
// dc: contexto de dispositivo
// color: color del pulgar
/*************************************************************************/

void THandChildSim::pulgar(TDC& dc,TColor color)
{
  dc.SelectObject(TPen(color));
  barra(dc,bb[1],g[1]);
  barra(dc,bb[2],g[2]);
  tip(dc,tt[0],g[3]);

//  sistref(dc,COrigin,R,g[1]);
//  sistref(dc,COrigin,R,g[2]);
//  sistref(dc,COrigin,R,g[3]);
}

/*************************************************************************/
// Funcion: indice(TDC& dc,TColor color)
// Dibuja el dedo indice
//
// dc: contexto de dispositivo
// color: color del indice
/*************************************************************************/

void THandChildSim::indice(TDC& dc,TColor color)
{
  dc.SelectObject(TPen(color));
  barra(dc,bb[3],g[4]);
  barra(dc,bb[4],g[5]);
  tip(dc,tt[1],g[6]);

//  sistref(dc,COrigin,R,g[4]);
//  sistref(dc,COrigin,R,g[5]);
//  sistref(dc,COrigin,R,g[6]);
}

/*************************************************************************/
// Funcion: medio(TDC& dc,TColor color)
// Dibuja el dedo medio
//
// dc: contexto de dispositivo
// color: color del dedo medio
/*************************************************************************/

void THandChildSim::medio(TDC& dc,TColor color)
{
  dc.SelectObject(TPen(color));
  barra(dc,bb[5],g[7]);
  barra(dc,bb[6],g[8]);
  tip(dc,tt[2],g[9]);

//  sistref(dc,COrigin,R,g[7]);
//  sistref(dc,COrigin,R,g[8]);
//  sistref(dc,COrigin,R,g[9]);
}

/*************************************************************************/
// Funcion: anular(TDC& dc,TColor color)
// Dibuja el dedo anular
//
// dc: contexto de dispositivo
// color: color del anular
/*************************************************************************/

void THandChildSim::anular(TDC& dc,TColor color)
{
  dc.SelectObject(TPen(color));
  barra(dc,bb[7],g[10]);
  barra(dc,bb[8],g[11]);
  tip(dc,tt[3],g[12]);

//  sistref(dc,COrigin,R,g[10]);
//  sistref(dc,COrigin,R,g[11]);
//  sistref(dc,COrigin,R,g[12]);
}

/*************************************************************************/
// Funcion: mano(TDC& dc,Miembro flag)
// Dibuja la mano
//
// dc: contexto de dispositivo
// flag: bandera que identifica el miembro en movimiento
// <1> palma, <2> pulgar, <3> indice, <4> medio, <5> anular
/*************************************************************************/

void THandChildSim::mano(TDC& dc,Miembro flag)
{
  TColor coldef,colesp,colpal,colind,colmed,colanu,colpul;

  coldef=CHand;
  colesp=CSelection;
  colpal=coldef;
  colind=coldef;
  colmed=coldef;
  colanu=coldef;
  colpul=coldef;

  switch (flag)
  {
    case mPalma:  colpal=colesp; break;
    case mPulgar:  colpul=colesp; break;
    case mIndice:  colind=colesp; break;
    case mMedio:  colmed=colesp; break;
    case mAnular:  colanu=colesp; break;
    default:  break;
  }

  rng(ang);      //Aqui se calculan todas las matrices

  palma(dc,colpal);
  pulgar(dc,colpul);
  indice(dc,colind);
  medio(dc,colmed);
  anular(dc,colanu);
}

/*************************************************************************/
// Funcion: pt2d(TDC& dc,TColor color,TPoint& point,int l)
// Crea una marca en el punto señalado
//
// dc: contexto de dispositivo
// color: color de la marca
// p: punto en la pantalla
// l: longitud de los brazos de la marca
/*************************************************************************/

void THandChildSim::pt2d(TDC& dc,TColor color,TPoint& p,int l)
{
  dc.SelectObject(TPen(color));
  dc.MoveTo((int)(p.x-l),(int)p.y);
  dc.LineTo((int)(p.x+1+l),(int)p.y);
  dc.MoveTo((int)p.x,(int)(p.y-l));
  dc.LineTo((int)p.x,(int)(p.y+1+l));
}

/*************************************************************************/
// Funcion: pt3d(TDC& dc,TColor color,const v4& p,int l)
// Dibuja un punto en el espacio
//
// dc: contexto de dispositivo
// color: color del punto dibujado
// p: coordenada en 3d del punto
/*************************************************************************/

void THandChildSim::pt3d(TDC& dc,TColor color,const v4& p,int l)
{
  TPoint p2d;
  dc.SelectObject(TPen(color));
  p2d=vscreen(D,p,mview);
  pt2d(dc,color,p2d,l);
}

/*************************************************************************/
// FUNCIONES MISCELANEAS
/*************************************************************************/

/*************************************************************************/
// Funcion: interpol(double x1,double x2,double y1,double y2,double x)
//
// x1: Punto 1 sobre las abscisas
// x2: Punto 2 sobre las abscisas
// y1: Punto 1 sobre las ordenadas
// y2: Punto 2 sobre las ordenadas
// x: Punto actual
```

```
/*************************************************************************/

double THandChildSim::interpol(double x1,double x2,double y1,
                               double y2,double x)
{
  double y,k;
  k=(x2-x1);
  if(k==0)
    k=1;
  y=((y2-y1)/k)*(x-x1)+y1;
  return y;
}

/*************************************************************************/
// FUNCIONES PARA EL USO DEL PUERTO SERIAL
/*************************************************************************/

/*************************************************************************/
// Funcion: WriteBuffer()
// Escribe el contenido del buffer al puerto serie
//
// Buffer: datos que se van e enviar
// dwToWrite: cantidad de bytes que se van a enviar
/*************************************************************************/

bool THandChildSim::WriteBuffer(char* Buffer,unsigned long dwToWrite)
{
  OVERLAPPED OlpWrite = {0};
  unsigned long dwWritten;
  unsigned long dwRes;
  bool Result;

  OlpWrite.hEvent = CreateEvent(NULL,T,F,NULL);
  if(OlpWrite.hEvent==NULL)
  {
    MessageBox("Falló CreateEvent","Mensaje",MB_OK|MB_ICONSTOP);
    return Result=F;
  }

  if(!WriteFile(hCom,Buffer,dwToWrite,&dwWritten,&OlpWrite))
  {
    if(GetLastError()!=ERROR_IO_PENDING)
      Result=F;
    else
    {
      dwRes = WaitForSingleObject(OlpWrite.hEvent,INFINITE);
      switch(dwRes)
      {
        case WAIT_OBJECT_0:
          if(!GetOverlappedResult(hCom,&OlpWrite,&dwWritten,F))
            Result=F;
          else
            Result=T;
          break;

        default:
          Result=F;
          break;
      }
    }
  }
  else
    Result=T;

  CloseHandle(OlpWrite.hEvent);
  return Result;
}

/*************************************************************************/
// Funcion: ReadBuffer()
// Lee valores en el puerto serial y calcula los nuevos angulos
/*************************************************************************/

void THandChildSim::ReadBuffer()
{
  OVERLAPPED OlpRead = {0};
  uint32 dwToRead = 15;
  uint32 dwRead;
  uint32 dwRes;
  uint32 dwReadTimeOut = 20;
  uint8 chData[15];
  bool bWaitingOnRead = F;
  bool bResult = F;

  OlpRead.hEvent = CreateEvent(NULL,T,F,NULL);
  if(OlpRead.hEvent==NULL)
  {
    MessageBox("Falló CreateEvent (Read)","Mensaje",MB_OK|MB_ICONSTOP);
    //Abort;
  }

  if(!bWaitingOnRead)
  {
    if(!ReadFile(hCom,&chData,dwToRead,&dwRead,&OlpRead))
    {
      if(GetLastError()==ERROR_IO_PENDING)
        bWaitingOnRead=T;
    }
    else
    {
      bResult=T;
    }
  }

  if(bWaitingOnRead)
  {
    dwRes = WaitForSingleObject(OlpRead.hEvent,dwReadTimeOut);
    switch(dwRes)
    {
      case WAIT_OBJECT_0:
        if(GetOverlappedResult(hCom,&OlpRead,&dwRead,F))
        {
          bResult=T;
        }
        break;

      default:
        break;
    }
  }

  if(bResult&&(chData[0]==247)&&(chData[14]==240))
  {
    ang.v[3]=interpol(0,127,-45,45,chData[5]);
    ang.v[4]=interpol(0,127,-90,0,chData[4]);
    ang.v[5]=interpol(0,127,-90,0,chData[6]);
    ang.v[6]=interpol(0,127,-90,0,chData[0]);

    ang.v[8]=interpol(0,127,-90,0,chData[7]);
    ang.v[9]=interpol(0,127,-90,0,chData[1]);
    ang.v[10]=ang.v[9];

    ang.v[12]=interpol(0,127,-90,0,chData[8]);
    ang.v[13]=interpol(0,127,-90,0,chData[2]);
    ang.v[14]=ang.v[13];

    ang.v[16]=interpol(0,127,-90,0,chData[9]);
    ang.v[17]=interpol(0,127,-90,0,chData[3]);
    ang.v[18]=ang.v[17];

    Invalidate();
  }

  CloseHandle(OlpRead.hEvent);
}

/*************************************************************************/
// FUNCIONES PARA LA GENERACION DE TRAYECTORIAS
/*************************************************************************/

/*************************************************************************/
// Funcion: cin_inv_dedo(double l2,double l3,double l4,const v4& p,const v4& q)
// Calcula la cinematica inversa de un dedo (excepto pulgar)
//
// l2: longitud de la falange
// l3: longitud de la falangina
// l4: longitud de la falangeta
// p: punto base del dedo
// q: punto al que se desea llegar
/*************************************************************************/

v4 THandChildSim::cin_inv_dedo(double l2,double l3,double l4,const v4& p,
                               const v4& q)
{
  double a,b,c,disc,l1,l5_2;
  double ang1,ang2,ang3,cang3;
  double alf,calf,bet,dist;
  v4 v1;

  v1=v_sum(q,p,F);
  l1=mag(v1);
  dist=mag(v_in(v1.v[0],v1.v[1],0));

  a=4*l2*l4;
  b=2*l3*(l2+l4);
  c=(l2*l2)+(l3*l3)+(l4*l4)-(l1*l1+2*l2*l4);
  disc=b*b-4*a*c;

  if(disc>=0)
  {
    cang3=(-b+sqrt(disc))/(2*a);
    if(1>=fabs(cang3))
    {
      ang3=AC(cang3);
      l5_2=(l3*l3)+(l4*l4)+(2*l3*l4*cang3);
      calf=((l1*l1)+(l2*l2)-l5_2)/(2*l1*l2);
```

```
    alf=AC(calf);
   }
   else
    ang3=0;
 }
 else
   ang3=0;

 if(dist==0)
   ang1=0;
 else
   ang1=AT2(v1.v[1],v1.v[0])-90;

 bet=AT2(v1.v[2],dist);

 ang2=alf+bet;

 if((ang1<-90)||(ang1>90))
 {
   ang1=ang1+180;
   if(v1.v[2]>=0)
     ang2=alf+(180-fabs(bet));
   if(v1.v[2]<0)
     ang2=alf-(180-fabs(bet));
 }

 v1=v_in(ang1,ang2,-ang3);

 return v1;
}

/*************************************************************************/
// Funcion: cin_inv_pulg(double l2,double l3,double l4,const v4& p,const v4& q,double a4)
// Calcula la cinematica inversa del pulgar
//
// l2: longitud del metacarpo
// l3: longitud de la falangina
// l4: longitud de la falangeta
// p: punto base del dedo
// q: punto al que se desea llegar
/*************************************************************************/

v4 THandChildSim::cin_inv_pulg(double l2,double l3,double l4,const v4& p,
                               const v4& q,double a4)
{
  double l1,k1,k2,k3,k4,k5,k6;
  double dist,disc;
  double a,b,c,d,e,f;
  double bet,a2,a3,ca4,sa4,ang1,ang2,ang3,ang4;
  v4 v1,v2;

  v1=v_sum(q,p,F);
  l1=mag(v1);
  v2=v_in(v1.v[0],v1.v[1],0);
  dist=mag(v2);

  k1=(l1*l1+l2*l2+l4*l4-l3*l3)/(2*l2*l4);
  k2=(l1/l4);
  k3=(l1/l2);
  k4=(l1*l1+l3*l3+l4*l4-l2*l2)/(2*l3*l4);
  k5=k2;
  k6=(l1/l3);

  ca4=C(a4);
  sa4=S(a4);

  a=k1+k2-(1+k3)*ca4;
  b=2*sa4;
  c=k1-k2+(1-k3)*ca4;
  d=k4+k5-(1+k6)*ca4;
  e=b;
  f=k4-k5+(1-k6)*ca4;

  disc=b*b-4*a*c;
  if(disc>=0)
  {
    a2=2*AT((-b+sqrt(disc))/(2*a));
    disc=e*e-4*d*f;

    if(disc>=0)
      a3=2*AT((-e-sqrt(disc))/(2*d));
    else
      a2=a3=a4=0;
  }
  else
    a2=a3=a4=0;

  ang3=a3-a2;
  ang4=a4-a3;

  if(dist==0)
    ang1=0;
  else
    ang1=AT2(v1.v[1],v1.v[0])-90;

  bet=AT2(v1.v[2],dist);
  ang2=a2+bet;

  if((ang1<-90)||(ang1>90))
  {
    ang1=ang1+180;
    if(v1.v[2]>=0)
      ang2=a2+(180-fabs(bet));
    if(v1.v[2]<0)
      ang2=a2-(180-fabs(bet));
  }

  v4 v={{ang1,ang2,ang3,ang4}};

  return v;
}

/*************************************************************************/
/*************************************************************************/
// DECLARACION DE LA CLASE << T3DPointsDialog >>
/*************************************************************************/
/*************************************************************************/

double XPos,YPos,ZPos;   //variables globales que guardan las coordenadas del punto actual

class T3DPointsDialog : public TDialog
{
private:

  int MaxLen;
  double xpos,ypos,zpos;
  TEdit *XCoord,*YCoord,*ZCoord;

public:

  T3DPointsDialog(TWindow* parent,TModule* module=0);
  ~T3DPointsDialog();
  virtual void SetupWindow();

  void CmDone();

DECLARE_RESPONSE_TABLE(T3DPointsDialog);
};

DEFINE_RESPONSE_TABLE1(T3DPointsDialog,TDialog)
  EV_BN_CLICKED(IDB_DONE,CmDone),
END_RESPONSE_TABLE;

/*************************************************************************/
// FUNCIONES PROPIAS DE CLASE
/*************************************************************************/

/*************************************************************************/
// Funcion: T3DPointsDialog()
// Constructor de clase
/*************************************************************************/

T3DPointsDialog::T3DPointsDialog(TWindow* parent,TModule* module)
  : TDialog(parent,"3D_POINTS_BOX",module)
{
  MaxLen=20;

  xpos=0;
  ypos=0;
  zpos=0;

  XCoord = new TEdit(this,IDE_XPOS,MaxLen);
  YCoord = new TEdit(this,IDE_YPOS,MaxLen);
  ZCoord = new TEdit(this,IDE_ZPOS,MaxLen);
}

/*************************************************************************/
// Funcion: ~T3DPointsDialog()
// Destructor de clase
/*************************************************************************/

T3DPointsDialog::~T3DPointsDialog()
{

}

/*************************************************************************/
// Funcion: SetupWindow()
// Inicializa el elemento visual
/*************************************************************************/

void T3DPointsDialog::SetupWindow()
{
  TDialog::SetupWindow();

  if(XCoord&&YCoord&&ZCoord)
```

```
  {
    XCoord->SetWindowText("0");
    YCoord->SetWindowText("0");
    ZCoord->SetWindowText("0");
  }
}

/*************************************************************************/
// FUNCIONES DE LA TABLA DE RESPUESTA
/*************************************************************************/

/*************************************************************************/
// Funcion: CmDone()
// Responde al mensaje IDB_DONE
/*************************************************************************/

void T3DPointsDialog::CmDone()
{
  if(XCoord)
  {
    char* str;
    int size=XCoord->GetWindowTextLength()+1;
    if((size>1)&&(0!=(str=new char[size])))
    {
      XCoord->GetWindowText(str,size);
      xpos=atof(str);
    }
    else
    {
      XCoord->SetWindowText("0");
      xpos=0;
    }
    delete str;
  }

  if(YCoord)
  {
    char* str;
    int size=YCoord->GetWindowTextLength()+1;
    if((size>1)&&(0!=(str=new char[size])))
    {
      YCoord->GetWindowText(str,size);
      ypos=atof(str);
    }
    else
    {
      YCoord->SetWindowText("0");
      ypos=0;
    }
    delete str;
  }

  if(ZCoord)
  {
    char* str;
    int size=ZCoord->GetWindowTextLength()+1;
    if((size>1)&&(0!=(str=new char[size])))
    {
      ZCoord->GetWindowText(str,size);
      zpos=atof(str);
    }
    else
    {
      ZCoord->SetWindowText("0");
      zpos=0;
    }
    delete str;
  }

  if(XCoord&&YCoord&&ZCoord)
  {
    XPos=xpos;
    YPos=ypos;
    ZPos=zpos;
  }

  Destroy(IDB_DONE);
}

/*************************************************************************/
/*************************************************************************/
// DECLARACION DE LA CLASE << THandChildTrj >>
/*************************************************************************/
/*************************************************************************/

class THandChildTrj : public THandChildSim
{
protected:

  bool first,last;  //indicador para el registro de puntos
  ptList pt;
  ptList trjpt;

public:

  THandChildTrj(TMDIClient& parent);
  ~THandChildTrj();
  void CleanupWindow();
  void SetupWindow();
  void Paint(TDC& dc,bool erase,TRect& rect);

  bool CanClose();
  void CmGetPoint();
  void CmGetPointEnabler(TCommandEnabler& tce);
  void CmDelPoint();
  void CmDelPointEnabler(TCommandEnabler& tce);
  void CmAnimPause();
  void CmAnimPauseEnabler(TCommandEnabler& tce);
  void CmAnimRun();
  void CmAnimRunEnabler(TCommandEnabler& tce);
  bool EvEraseBkgnd(HDC);
  void EvKeyDown(uint key,uint repeatCount,uint flags);
  void EvLButtonDown(uint modkeys,TPoint& point);
//  void EvMButtonDown(uint modKeys, TPoint& point);
  void EvRButtonDown(uint modkeys,TPoint& point);
  void EvTimer(uint id);

  void trajgen();

  v4 sv_xy(TPoint& p);
  TPoint vs_xy(v4& v);
  void drawControlPoints2D(TDC& dc,TColor);
  void drawControlPoints3D(TDC& dc,TColor);
  void drawCurve2D(TDC& dc,TColor color);
  void drawCurve3D(TDC& dc,TColor color);

DECLARE_RESPONSE_TABLE(THandChildTrj);
};

DEFINE_RESPONSE_TABLE1(THandChildTrj,TMDIChild)
  EV_COMMAND(CM_GET_POINT,CmGetPoint),
  EV_COMMAND_ENABLE(CM_GET_POINT,CmGetPointEnabler),
  EV_COMMAND(CM_DEL_POINT,CmDelPoint),
  EV_COMMAND_ENABLE(CM_DEL_POINT,CmDelPointEnabler),
  EV_COMMAND(CM_ANIM_PAUSE,CmAnimPause),
  EV_COMMAND_ENABLE(CM_ANIM_PAUSE,CmAnimPauseEnabler),
  EV_COMMAND(CM_ANIM_RUN,CmAnimRun),
  EV_COMMAND_ENABLE(CM_ANIM_RUN,CmAnimRunEnabler),
  EV_WM_ERASEBKGND,
  EV_WM_KEYDOWN,
  EV_WM_LBUTTONDOWN,
  EV_WM_MBUTTONDOWN,
  EV_WM_RBUTTONDOWN,
  EV_WM_TIMER,
END_RESPONSE_TABLE;

/*************************************************************************/
// FUNCIONES PROPIAS DE CLASE
/*************************************************************************/

/*************************************************************************/
// Funcion: THandChildTrj()
// Constructor de clase
/*************************************************************************/

THandChildTrj::THandChildTrj(TMDIClient& parent)
  : THandChildSim(parent)
{
  WI=600;
  HE=400;
  MARGEN=10;

  Attr.Style &= ~WS_THICKFRAME;
  Attr.Style &= ~WS_MAXIMIZEBOX;
  Attr.W=WI+5;
  Attr.H=HE+25;
  SetCaption("Generador de Trayectorias");

  estado=eInactivo;

  XL=MARGEN;
  XR=WI-MARGEN;
  YB=HE-MARGEN;
  YT=MARGEN;

  XDIF=XR-XL;
  YDIF=YB-YT;

  XMIN=-3;
  XMAX=3;
  YMIN=-3;
  YMAX=3;

  XMIN=((XMIN*XDIF)/YDIF);
  XMAX=((XMAX*XDIF)/YDIF);
```

```
  SCALX=(double)(XDIF/(XMAX-XMIN));
  SCALY=(double)(YDIF/(YMAX-YMIN));

  UCS_XL=XL;
  UCS_XR=XL+(int)(YDIF*0.2);
  UCS_YT=YT+(int)(YDIF*0.8);
  UCS_YB=YB;
  UCS_XMIN=-1;
  UCS_XMAX=1;
  UCS_YMIN=-1;
  UCS_YMAX=1;

  UCS_XDIF=UCS_XR-UCS_XL;
  UCS_YDIF=UCS_YB-UCS_YT;

  UCS_SCALX=(double)(UCS_XDIF/(UCS_XMAX-UCS_XMIN));
  UCS_SCALY=(double)(UCS_YDIF/(UCS_YMAX-UCS_YMIN));

  initpuntos();
  initmatgs0();

  for(int i=0;19>i;i++)
    ang.v[i]=0;

  THETA=45;
  PHI=55;
  RHO=0;

  mview=mmview(THETA,PHI,RHO);

  CBackground=CBLACK;
  CSelection=CCYAN;
  COrigin=CYELLOW;
  CUcsOrigin=CWHITE;

  first=T;
  last=F;
}

/*************************************************************************/
// Funcion: ~THandChildTrj()
// Destructor de clase
/*************************************************************************/

THandChildTrj::~THandChildTrj()
{
  //delete memDC;
}

/*************************************************************************/
// Funcion: CleanupWindow()
/*************************************************************************/

void THandChildTrj::CleanupWindow()
{
  KillTimer(Timer);
}

/*************************************************************************/
// Funcion: SetupWindow()
/*************************************************************************/

void THandChildTrj::SetupWindow()
{
  TMDIChild::SetupWindow();

  Timer=0;
  if(SetTimer(1,TIME2))
    Timer=1;

  TClientDC dc(HWindow);
  memDC = new TMemoryDC(dc);
  memDC->SelectObject(TBitmap(dc,WI,HE));
}

/*************************************************************************/
// Funcion: Paint(TDC& dc,bool erase,TRect& rect)
// Trazado de la pantalla
/*************************************************************************/

void THandChildTrj::Paint(TDC& dc,bool,TRect&)
{
  memDC->FillRect(GetClientRect(),TBrush(CBackground));
  ucs_icon(*memDC,CUcsOrigin,0.5*U);
  origen(*memDC,COrigin,U);

  drawControlPoints2D(*memDC,CMAGENTA);
  drawCurve2D(*memDC,CDGREEN);

  drawControlPoints3D(*memDC,CRED);
  drawCurve3D(*memDC,CGREEN);

  if((estado==eActivo)||(estado==ePausa))
  {
    mg(ang);
    medio(*memDC,CSelection);
  }

  dc.BitBlt(GetClientRect(),*memDC,TPoint(0,0));
}

/*************************************************************************/
// FUNCIONES DE LA TABLA DE RESPUESTA
/*************************************************************************/

/*************************************************************************/
// Funcion: CanClose()
// Responde ante la solicitud de cierre de la ventana
/*************************************************************************/

bool THandChildTrj::CanClose()
{
  bool result;
  result=MessageBox("¿Desea cerrar la ventana del generador de trayectorias?",
            "Manipulador Antropomórfico Teleoperado",
             MB_YESNO|MB_ICONQUESTION)==IDYES;
  if(result)
    estgt=egtDisabled;
  return result;
}

/*************************************************************************/
// Funcion: EvEraseBkgnd(HDC)
// Responde al mensaje EV_WM_ERASEBKGND
/*************************************************************************/

bool THandChildTrj::EvEraseBkgnd(HDC)
{
  return TRUE;
}

/*************************************************************************/
// Funcion: CmGetPoint()
// Responde al mensaje CM_GET_POINT
/*************************************************************************/

void THandChildTrj::CmGetPoint()
{
  v4 v;

  if(T3DPointsDialog(this).Execute()==IDB_DONE)
  {
    v=p_in(XPos,YPos,ZPos);
    if(first)
    {
      for(int i=0;3>i;i++)
        pt.push_back(v);
      first=F;
      Invalidate();
    }
    else if(!last)
    {
      pt.push_back(v);
      Invalidate();
    }
  }
}

/*************************************************************************/
// Funcion: CmGetPointEnabler(TCommandEnabler& tce)
// Responde al mensaje CM_GET_POINT
/*************************************************************************/

void THandChildTrj::CmGetPointEnabler(TCommandEnabler& tce)
{
  if(last)
    tce.Enable(F);
}

/*************************************************************************/
// Funcion: CmDelPoint()
// Responde al mensaje CM_DEL_POINT
/*************************************************************************/

void THandChildTrj::CmDelPoint()
{
  bool result;
  char s[151];
  v4 v;
  ptIter index1,index2;

  index1=pt.end();
  index1--;
  v=*index1;

  sprintf(s,"¿Desea eliminar el ultimo punto?\n\n\tX: %f\n\tY: %f\n\tZ: %f",
```

```
      v.v[0],v.v[1],v.v[2]);

  result=MessageBox(s,"Manipulador Antropomórfico Teleoperado",
            MB_YESNO|MB_ICONQUESTION)==IDYES;

  index2=index1;
  for(int i=0;2>i;i++)
    index2--;

  if(result&&!last)
  {
    if(index2==pt.begin())
    {
      for(int i=0;3>i;i++)
      {
        pt.erase(index1);
        index1--;
      }
      first=T;
      Invalidate();
    }
    else
    {
      pt.erase(index1);
      Invalidate();
    }
  }
}

/*************************************************************************/
// Funcion: CmDelPointEnabler(TCommandEnabler& tce)
// Responde al mensaje CM_DEL_POINT
/*************************************************************************/

void THandChildTrj::CmDelPointEnabler(TCommandEnabler& tce)
{
  if(first||last)
    tce.Enable(F);
  else
    tce.Enable(T);
}

/*************************************************************************/
// Funcion: CmAnimPause()
// Responde al mensaje CM_ANIM_PAUSE
/*************************************************************************/

void THandChildTrj::CmAnimPause()
{
  if(estado==eActivo)
    estado=ePausa;
  else
    if(estado==ePausa)
      estado=eActivo;
  Invalidate();
}

/*************************************************************************/
// Funcion: CmAnimPauseEnabler(TCommandEnabler& tce)
// Responde al mensaje CM_ANIM_PAUSE
/*************************************************************************/

void THandChildTrj::CmAnimPauseEnabler(TCommandEnabler& tce)
{
  if(estado==eInactivo)
    tce.Enable(F);
  else
    if(estado==ePausa)
      tce.SetText("&Continuar");
    else
      tce.SetText("&Pausa");
}

/*************************************************************************/
// Funcion: CmAnimRun()
// Responde al mensaje CM_ANIM_RUN
/*************************************************************************/

void THandChildTrj::CmAnimRun()
{
  estado=eActivo;
  miembro=mNada;
  for(int i=0,19>i;i++)
    ang.v[i]=0;
  Invalidate();
}

/*************************************************************************/
// Funcion: CmPrgRunEnabler(TCommandEnabler& tce)
// Responde al mensaje CM_ANIM_RUN
/*************************************************************************/

void THandChildTrj::CmAnimRunEnabler(TCommandEnabler& tce)
{
  if(estado==eActivo||estado==ePausa)
    tce.Enable(F);
}

/*************************************************************************/
// Funcion: EvKeyDown(uint key,uint repeatCount,uint flags)
// Responde al mensaje EV_WM_KEYDOWN
/*************************************************************************/

void THandChildTrj::EvKeyDown(uint key,uint,uint)
{
  switch(key)
  {
    case VK_RIGHT:
      if(!(PHI==0)||(PHI==180)||(PHI==-180))
      {
        THETA+=DANG;
        mview=mmview(THETA,PHI,RHO);
        Invalidate();
        break;
      }
      else
        break;
    case VK_LEFT:
      if(!(PHI==0)||(PHI==180)||(PHI==-180))
      {
        THETA-=DANG;
        mview=mmview(THETA,PHI,RHO);
        Invalidate();
        break;
      }
      else
        break;
    case VK_DOWN:
      PHI+=DANG,
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;
    case VK_UP:
      PHI-=DANG;
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;
    case VK_F5:
      RHO+=DANG;
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;
    case VK_F6:
      RHO-=DANG;
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;

    case VK_F9:
      THETA=45;
      PHI=55;
      RHO=0;
      mview=mmview(THETA,PHI,RHO);
      Invalidate();
      break;
  }
}

/*************************************************************************/
// Funcion: EvLButtonDown(uint modkeys,TPoint& point)
// Responde al mensaje EV_WM_LBUTTONDOWN
/*************************************************************************/

void THandChildTrj::EvLButtonDown(uint /*modkeys*/,TPoint& point)
{
  TRect rect;
  GetClientRect(rect);

  if(rect.Contains(point))
  {
    if(first)
    {
      for(int i=0,3>i;i++)
        pt.push_back(sv_xy(point));

      first=F;
      Invalidate();
    }
    else if(!last)
    {
      pt.push_back(sv_xy(point));
      Invalidate();
    }
  }
}
```

```
/*************************************************************************/
// Funcion: EvMButtonDown(uint modkeys,TPoint& point)
// Responde al mensaje EV_WM_MBUTTONDOWN
/*************************************************************************/
/*
void THandChildTrj::EvMButtonDown(uint,TPoint& point)
{
  ptIter index;
  TRect rect;
  GetClientRect(rect);

  if(rect.Contains(point))
  {
    if(!last)
    {
      index=pt.end();
      index--;
      pt.erase(index);
      Invalidate();
    }
  }
}

/*************************************************************************/
// Funcion: EvRButtonDown(uint modkeys,TPoint& point)
// Responde al mensaje EV_WM_RBUTTONDOWN
/*************************************************************************/

void THandChildTrj::EvRButtonDown(uint,TPoint& point)
{
  ptIter index;
  TRect rect;
  GetClientRect(rect);

  if(rect.Contains(point))
  {
    if(!first&&!last)
    {
      for(int i=0;2>i;i++)
      {
        index=pt.end();
        index--;
        pt.push_back(*index);
      }
      last=T;
      trajgen();
      Invalidate();
    }
    else
      Invalidate();
  }
}

/*************************************************************************/
// Funcion: EvTimer(uint timerId)
// Responde al mensaje EV_WM_TIMER
/*************************************************************************/

void THandChildTrj::EvTimer(uint)
{
  v4 w;
  static ptIter index1=trjpt.begin();

  if(estado==cActivo)
  {
    if(index1==trjpt.end())
      index1=trjpt.begin();

    w=cin_inv_dedo(L1M,L2M,L3M,NULO,*index1);
    ang.v[11]=w.v[0];
    ang.v[12]=w.v[1];
    ang.v[13]=w.v[2];
    ang.v[14]=w.v[2];

    Invalidate();
    index1++;
  }
}

/*************************************************************************/
// FUNCIONES PARA LA GENERACION DE TRAYECTORIAS EN 2D Y 3D
/*************************************************************************/

/*************************************************************************/
// Funcion: trajgen()
// Genera una trayectoria en 2 dimensiones
/*************************************************************************/

void THandChildTrj::trajgen()
{
  double u,k;
  ptIter index,index1,index2;
  v4 v,p[4];

  k=0.166666666667;
  m4 m2={{{-1, 3,-3,1},
          { 3,-6, 0,4},
          {-3, 3, 3,1},
          { 1, 0, 0,0}}};

  index1=pt.begin();
  for(int i=0;3>i;i++)
    index1++;

  for(index=index1;index!=pt.end();index++)
  {
    index2=index;
    for(int i=3;i>=0;i--)
    {
      p[i]=*index2;
      index2--;
    }

    m4 m1={{{p[0].v[0],p[1].v[0],p[2].v[0],p[3].v[0]},
            {p[0].v[1],p[1].v[1],p[2].v[1],p[3].v[1]},
            {p[0].v[2],p[1].v[2],p[2].v[2],p[3].v[2]},
            {   0,        0,        0,        0}}};

    for(u=0.0;1.0>=u;u+=0.1)
    {
      v=p_in(u*u*u,u*u,u,1);
      v=v_esc(k,mv_mult(m1,mv_mult(m2,v)));
      trjpt.push_back(v);
    }
  }
}

/*************************************************************************/
// FUNCIONES GRAFICAS
/*************************************************************************/

/*************************************************************************/
// Funcion: sv_xy(TPoint& p)
// Transforma coord. graficas en coord. del sistema en el plano xy
//
// p: punto de la pantalla
/*************************************************************************/

v4 THandChildTrj::sv_xy(TPoint& p)
{
  return p_in((double)((p.x-XL)/SCALX+XMIN),(double)((YB-p.y)/SCALY+YMIN),0);
}

/*************************************************************************/
// Funcion: vs_xy(v4& v)
// Transforma coord. del sistema en el plano xy en coord. graficas
//
// v: punto del sistema
/*************************************************************************/

TPoint THandChildTrj::vs_xy(v4& v)
{
  TPoint p((int)(XL+(v.v[0]-XMIN)*SCALX),(int)(YB-(v.v[1]-YMIN)*SCALY));
  return p;
}

/*************************************************************************/
// Funcion: drawControlPoints2D(TDC& dc,TColor color)
// Dibuja los puntos en la pantalla
//
// dc: contexto de dispositivo
// color: color de los puntos
/*************************************************************************/

void THandChildTrj::drawControlPoints2D(TDC& dc,TColor color)
{
  int i;
  ptIter index,index1,index2;

  dc.SelectObject(TPen(color));

  index1=pt.begin();
  for(i=0;2>i;i++)
    index1++;

  if(!last)
  {
    for(index=index1;index!=pt.end();index++)
      pt2d(dc,color,vs_xy(*index),2);
  }
  else
  {
    index2=pt.end();
    for(i=0;2>i;i++)
      index2--;
    for(index=index1;index!=index2;index++)
```

```
      pt2d(dc,color,vs_xy(*index),2);
    }
  }

/*************************************************************************/
// Funcion: drawControlPoints3D(TDC& dc,TColor color)
// Dibuja los puntos en la pantalla
//
// dc: contexto de dispositivo
// color: color de los puntos
/*************************************************************************/

void THandChildTrj::drawControlPoints3D(TDC& dc,TColor color)
{
  int i;
  pdIter index,index1,index2;

  dc.SelectObject(TPen(color));

  index1=pt.begin();
  for(i=0;2>i;i++)
    index1++;

  if(!last)
  {
    for(index=index1;index!=pt.end();index++)
      pt3d(dc,color,*index,2);
  }
  else
  {
    index2=pt.end();
    for(i=0;2>i;i++)
      index2--;
    for(index=index1;index!=index2;index++)
      pt3d(dc,color,*index,2);
  }
}

/*************************************************************************/
// Funcion: drawCurve2D(TDC& dc,Tcolor color)
// Dibuja la curva generada
//
// dc: contexto de dispositivo
// color: color de la curva
/*************************************************************************/

void THandChildTrj::drawCurve2D(TDC& dc,TColor color)
{
  bool id = F;
  pdIter index;

  if(last)                                          /
  {
    dc.SelectObject(TPen(color));
    for(index=trjpt.begin();index!=trjpt.end();index++)
    {
      if(id==F)
      {
        dc.MoveTo(vs_xy(*index));
        id=T;
      }
      else
      {
        dc.LineTo(vs_xy(*index));
      }
    }
  }
}

/*************************************************************************/
// Funcion: drawCurve3D(TDC& dc,Tcolor color)
// Dibuja la curva generada
//
// dc: contexto de dispositivo
// color: color de la curva
/*************************************************************************/

void THandChildTrj::drawCurve3D(TDC& dc,TColor color)
{
  bool id = F;
  pdIter index;

  if(last)
  {
    dc.SelectObject(TPen(color));
    for(index=trjpt.begin();index!=trjpt.end();index++)
    {
      if(id==F)
      {
        dc.MoveTo(vscreen(D,*index,mview));
        id=T;
      }
      else
      {
        dc.LineTo(vscreen(D,*index,mview));
      }
    }
  }
}

/*************************************************************************/
/*************************************************************************/
// DECLARACION DE LA CLASE << THandApp >>
/*************************************************************************/
/*************************************************************************/

class THandApp : public TApplication
{
public:

  THandMDIClient* MdiClient;

  THandApp();
  void InitMainWindow();

  void CmAboutBox();
  void CmSimulator();
  void CmSimulatorEnabler(TCommandEnabler& tce);
  void CmTrajGen();
  void CmTrajGenEnabler(TCommandEnabler& tce);

DECLARE_RESPONSE_TABLE(THandApp);
};

DEFINE_RESPONSE_TABLE1(THandApp,TApplication)
  EV_COMMAND(CM_HELP_ABOUT,CmAboutBox),
  EV_COMMAND(CM_SIMULATOR,CmSimulator),
  EV_COMMAND_ENABLE(CM_SIMULATOR,CmSimulatorEnabler),
  EV_COMMAND(CM_TRAJ_GEN,CmTrajGen),
  EV_COMMAND_ENABLE(CM_TRAJ_GEN,CmTrajGenEnabler),
END_RESPONSE_TABLE;

/*************************************************************************/
// Funcion: THandApp()
// Constructor de clase
/*************************************************************************/

THandApp::THandApp()
  : TApplication()
{
    nCmdShow=SW_SHOWDEFAULT;
}

/*************************************************************************/
// Funcion: InitMainWindow()
// Inicializa parametros de la aplicacion
/*************************************************************************/

void THandApp::InitMainWindow()
{
  MdiClient = new THandMDIClient;
  TDecoratedMDIFrame* mdif = new TDecoratedMDIFrame(
                  "Manipulador Antropomórfico Teleoperado",
                  MAIN_MENU,
                  *MdiClient,
                  T);

  mdif->Attr.W=750;
  mdif->Attr.H=550;
  mdif->Attr.Style=WS_THICKFRAME|WS_OVERLAPPED|WS_CAPTION|
                  WS_SYSMENU|WS_MINIMIZEBOX|WS_MAXIMIZEBOX;
  mdif->Attr.AccelTable = MAIN_MENU;
  mdif->SetIcon(this,MANO_ICON);
  mdif->SetIconSm(this,MANO_ICON);

  SetMainWindow(mdif);

  estsim=esDisabled;
  estgt=egtDisabled;
}

/*************************************************************************/
// Funcion: CmAboutBox()
// Responde al mensaje CM_HELP_ABOUT
/*************************************************************************/

void THandApp::CmAboutBox()
{
  sndPlaySound("sound.wav",SND_ASYNC|SND_NODEFAULT);
  TDialog(GetMainWindow(),ABOUT_BOX).Execute();
}

/*************************************************************************/
// Funcion: CmSimulator()
// Responde al mensaje CM_SIMULATOR
/*************************************************************************/
```

```
void THandApp::CmSimulator()
{
  if(estsim==esDisabled)
  {
    THandChildSim* child = new THandChildSim(*MdiClient);
    child->SetIcon(this,MANO_ICON);
    child->SetIconSm(this,MANO_ICON);
    child->Create();

    estsim=esEnabled;
  }
}

/*************************************************************************/
// Funcion: CmSimulatorEnabler(TCommandEnabler& tce)
// Responde al mensaje CM_SIMULATOR
/*************************************************************************/

void THandApp::CmSimulatorEnabler(TCommandEnabler& tce)
{
  if(estsim==esEnabled)
    tce.Enable(F);
}

/*************************************************************************/
// Funcion: CmTrajGen()
// Responde al mensaje CM_TRAJ_GEN
/*************************************************************************/

void THandApp::CmTrajGen()
{
  if(estgt==egtDisabled)
  {
    THandChildTrj* child = new THandChildTrj(*MdiClient);
    child->SetIcon(this,TG_ICON);
    child->SetIconSm(this,TG_ICON);
    child->Create();

    estgt=egtEnabled;
  }
}

/*************************************************************************/
// Funcion: CmTrajGenEnabler(TCommandEnabler& tce)
// Responde al mensaje CM_TRAJ_GEN
/*************************************************************************/

void THandApp::CmTrajGenEnabler(TCommandEnabler& tce)
{
  if(estgt==egtEnabled)
    tce.Enable(F);
}

/*************************************************************************/
/*************************************************************************/
// Programa principal
/*************************************************************************/
/*************************************************************************/

int OwlMain(int,char *[])
{
  int retVal = THandApp().Run();
  return retVal;
}
```

APÉNDICE C

Programa de simulación para MS-DOS (Plataforma de 16 bits)

A continuación se presenta el código fuente del programa que se encarga de realizar la simulación gráfica de los movimientos del manipulador antropomórfico teleoperado (MAT-I). Este programa debe compilarse para correrse desde MS-DOS.

C.1 Código fuente del archivo de programa DOSMAT-I.CPP

```
/*****************************************************************/
//
// MANIPULADOR ANTROPOMORFICO TELEOPERADO - I ( MAT - I )
//
// DOSMAT-I.CPP
//
// Programa de simulacion para conectarse al guante sensor
//
// Plataforma: DOS (16 bits)
//
// Hugo Figueroa Rosas  1998.
//
/*****************************************************************/

/*****************************************************************/
// Archivos de encabezado
/*****************************************************************/

#include <bios.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <graphics.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*****************************************************************/
// Definicion de funciones trigonometricas simplificadas
/*****************************************************************/

#define S(x) sin((x)*PI/180)          //funcion seno con argumento en grados
#define C(x) cos((x)*PI/180)          //funcion coseno con argumento en grados
#define AS(x) (asin(x)*180/PI)        //funcion arco seno en grados
#define AC(x) (acos(x)*180/PI)        //funcion arco coseno en grados
#define AT(x) (atan(x)*180/PI)        //funcion arco tangente en grados
#define AT2(y,x) (atan2(y,x)*180/PI)  //funcion arco tangente en grados (y/x)

/*****************************************************************/
// Definicion de constantes matematicas empleadas
/*****************************************************************/

const double PI = 3.1415926535898;  //pi

const int ORD = 4;     //tamano de las matrices cuadradas y vectores

const double DANG = 5;  //delta de angulo

/*****************************************************************/
// Constantes para instancias falso/verdadero
/*****************************************************************/

const bool T = true;    //valor verdadero
const bool F = false;   //valor falso

/*****************************************************************/
// Constantes para el trazo de los dedos
/*****************************************************************/

const double FESC = 1;                //factor de escala para los dedos
const double WW = FESC*0.4392;  //ancho de los dedos
const double HH = FESC*0.3659;  //alto de los dedos

const double D = FESC*5;        //distancia de la camara al plano de proyeccion
const double U = FESC*1;        //longitud de los ejes del origen
const double R = FESC*0.5;      //longitud de los ejes de los sistemas de referencia

/*****************************************************************/
// Constantes de la palma
/*****************************************************************/

const double XPL = FESC*0;          //coordenada en x del origen de la palma
const double YPL = FESC*-1.7325;  //coordenada en y del origen de la palma
const double ZPL = FESC*0;          //coordenada en z del origen de la palma

const double LPAL = FESC*1.7325;  //longitud de la palma

const double LSPL1 = 180;  //lim sup grado 1
const double LIPL1 = -180;  //lim inf grado 1
const double LSPL2 = 90;    //lim sup grado 2
const double LIPL2 = -90;    //lim inf grado 2
const double LSPL3 = 90;    //lim sup grado 3
const double LIPL3 = -90;    //lim inf grado 3

/*****************************************************************/
// Constantes del dedo pulgar
/*****************************************************************/

const double XP = FESC*-0.5867;  //coordenada en x del origen del pulgar
const double YP = FESC*-1.7325;  //coordenada en y del origen del pulgar
const double ZP = FESC*0;          //coordenada en z del origen del pulgar

const double L1P = FESC*0.9246;  //longitud de la falange proximal del pulgar
const double L2P = FESC*0.7628;  //longitud de la falange media del pulgar
const double L3P = FESC*0.601;   //longitud de la falange distal del pulgar

const double LSP1 = 90;    //lim sup grado 1
const double LIP1 = 0;     //lim inf grado 1
const double LSP2 = 20;    //lim sup grado 2
const double LIP2 = -90;   //lim inf grado 2
const double LSP3 = 0;     //lim sup grado 3
```

```
const double LIP3 = -90;    //lim inf grado 3
const double LSP4 = 0;      //lim sup grado 4
const double LIP4 = -90;    //lim inf grado 4

const double A = 30;    //angulo de orientacion del sistema del pulgar (eje z')
const double B = 35;    //angulo de orientacion del sistema del pulgar (eje y')

/*****************************************************************/
// Constantes del dedo indice
/*****************************************************************/

const double XI = FESC*-0.5867;  //coordenada en x del origen del indice
const double YI = FESC*-0.0255;  //coordenada en y del origen del indice
const double ZI = FESC*0;        //coordenada en z del origen del indice

const double L1I = FESC*1;       //longitud de la falange proximal del indice
const double L2I = FESC*0.5855;  //longitud de la falange media del indice
const double L3I = FESC*0.4879;  //longitud de la falange distal del indice

const double LSD1 = 20;    //lim sup grado 1 para el resto de los dedos
const double LID1 = -20;   //lim inf grado 1
const double LSD2 = 20;    //lim sup grado 2
const double LID2 = -105;  //lim inf grado 2
const double LSD3 = 0;     //lim sup grado 3 y 4
const double LID3 = -90;   //lim inf grado 3 y 4

/*****************************************************************/
// Constantes del dedo medio
/*****************************************************************/

const double XM = FESC*0;       //coordenada en x del origen del medio
const double YM = FESC*0;       //coordenada en y del origen del medio
const double ZM = FESC*0;       //coordenada en z del origen del medio

const double L1M = FESC*1.098;  //longitud de la falange proximal del medio
const double L2M = FESC*0.683;  //longitud de la falange media del medio
const double L3M = FESC*0.512;  //longitud de la falange distal del medio

/*****************************************************************/
// Constantes del dedo anular
/*****************************************************************/

const double XA = FESC*0.5867;  //coordenada en x del origen del anular
const double YA = FESC*-0.0255; //coordenada en y del origen del anular
const double ZA = FESC*0;       //coordenada en z del origen del anular

const double L1A = FESC*1.07;   //longitud de la falange proximal del medio
const double L2A = FESC*0.6343; //longitud de la falange media del medio
const double L3A = FESC*0.512;  //longitud de la falange distal del medio

/*****************************************************************/
// Constantes de temporizacion
/*****************************************************************/

const int TIME = 10;  //intervalo de retraso para la ejecucion de secuencias

const int CLIP = 1;

/*****************************************************************/
// Clase para el manejo de puntos en dos dimensiones
/*****************************************************************/

class TPoint
{
public:
  int x;
  int y;
};

/*****************************************************************/
// Definicion de una matriz de 4x4 para el manejo de rotaciones en 3D
/*****************************************************************/

class m4
{
public:
  double m[ORD][ORD];
};

/*****************************************************************/
// Definicion de un vector de 4 elementos para el manejo de puntos
// y vectores en 3D
/*****************************************************************/

class v4
{
public:
  double v[ORD];
};

/*****************************************************************/
// Definicion de un vector de 8 elementos v4 para el manejo de coordenadas
// de las aristas de paralelepipedos y prismas trapezoidales
/*****************************************************************/

class v48
{
public:
  v4 v[8];
};

/*****************************************************************/
// Definicion de un vector de 12 elementos v4 para el manejo de coordenadas
// de las aristas de prismas de seccion variable
/*****************************************************************/

class v412
{
public:
  v4 v[12];
};

/*****************************************************************/
// Definicion de un vector de 19 elementos para el manejo de los angulos
// en cada uno de los eslabones
/*****************************************************************/

class v19
{
public:
  double v[19];
};

/*****************************************************************/
// Definicion de un vector de 13 elementos para el manejo de la entrada
// y salida de informacion por puerto serie
/*****************************************************************/

class v13
{
public:
  unsigned int v[13];
};

/*****************************************************************/
/*****************************************************************/
// DECLARACION DE LA CLASE << THandSim >>
/*****************************************************************/
/*****************************************************************/

class THandSim
{
protected:

  enum Estado   {eActivo,eInactivo,ePausa} estado;  //Estado de ejecucion

  enum Miembro {mNada,mPalma,mPulgar,mIndice,mMedio,mAnular};
  Miembro miembro;  //identificador de miembro

  union REGS rin, rout;

  bool flag1;    //bandera que activa la secuencia de calibracion
  bool flag2;    //bandera que activa la secuencia de grabado
  bool flag3;    //bandera que indica ejecucion de secuencia

  double THETA,PHI,RHO;  //angulos del plano de proyección

  int WI,HE;       //ancho y altura de la ventana
  int MARGEN;      //margen alrededor de area de trazado
  int XI,XR,YB,YT; //limites izq, der, inf, y sup para el área de trazo
  int XDIF,YDIF;   //anchura y altura de la pantalla
  double XMIN,XMAX,YMIN,YMAX;  //limites teoricos deseados por el usuario
  double SCALX,SCALY;          //factores de escala para el trazado

  int UCS_XI,UCS_XR,UCS_YT,UCS_YB;  //limites para el ucs_icon
  int UCS_XDIF,UCS_YDIF;
  double UCS_XMIN,UCS_XMAX,UCS_YMIN,UCS_YMAX;
  double UCS_SCALX,UCS_SCALY;

  m4 mview,g[13],gs[13],gs0[17],e[19];
  v4  Q[17],I,J,K,IP,JP,KP,NULO;
  v19 ang,anginf,angsup;
  v48 bb[9];
  v412 tt[4];

  v13 canal,cmin,cmax;
  int CBackground,CHand,CSelection,COrigin,CUcsOrigin,CFrame;

public:

  THandSim();
  ~THandSim();
  void SetupWindow();
  void CloseWindow();
  void Paint();
```

```
void PrgPause();
void PrgRun();
void EvChar(char key);

void initpuntos(void);
m4 mgs0(const v4& p);
m4 mgsp0(const v4& p);
void initmatgs0(void);
m4 ee1(double a,const v4& p);
m4 ee2(double a,const v4& p);
m4 ee3(double a,const v4& p);
m4 ee4(double a,const v4& w,const v4& p);
void mg(const v19& a);
m4 m_rot(double a,const v4& b,double x,double y,double z);

m4 m_esc(double esc,const m4& a);
m4 m_transp(const m4& a);
m4 mm_mult(const m4& a,const m4& b);
v4 mv_mult(const m4& a,const v4& b);
m4 m_inv(const m4& a);

v4 p_in(double x,double y,double z,double k = 1);
v4 v_in(double x,double y,double z,double k = 0);
v4 v_sum(const v4& a,const v4& b,char oper);
v4 v_esc(double esc,const v4& a);
double dot(const v4& a,const v4& b);
v4 cross(const v4& a,const v4& b);
double mag2(const v4& a);
double mag(const v4& a);
v4 v_unit(const v4& a);

v48 v48_in(double a,double b,double h,double l);
v412 v412_in(double w1,double h1,double w2,double h2,double l,double r);

m4 mmview(double theta,double& phi,double rho);
TPoint vscreen(double d,const v4& a,const m4& m);
void ln(TPoint& p1,TPoint& p2);
void frame(int color);
void ucs_icon(int color,double l);
void origen(int color,double l);
void sistref(int color,double l,const m4& rot);
void barra(v48& p3d,const m4& rot);
void tip(v412& p3d,const m4& rot);
void palma(int color);
void pulgar(int color);
void indice(int color);
void medio(int color);
void anular(int color);
void mano(Miembro flag);
void pt2d(int color,TPoint& point,int l = 3);
void pt3d(int color,const v4& p,int l = 3);
void cal_icon(int color,bool flag);
void rec_icon(int color,bool flag);
void play_icon(int color,bool flag);

void init_rs(void);
int test_rs(void);
char leer(void);
void escribir(char c);
v13 get13(void);

double interpol(double x1,double x2,double y1,double y2,double x);
v13 minimo(v13 datos,v13 minimos);
v13 maximo(v13 datos,v13 maximos);

void salvar(v19 buf);
void cargar(double tiempo);
};

/*************************************************************************/
// FUNCIONES PROPIAS DE CLASE
/*************************************************************************/

/*************************************************************************/
// Funcion: THandSim()
// Constructor de clase
/*************************************************************************/

THandSim::THandSim()
{
  SetupWindow();

  WI=getmaxx()+1;
  HE=getmaxy()+1;
  MARGEN=40;

  estado=elnactivo;

  XL=MARGEN;
  XR=WI-MARGEN;
  YB=HE-MARGEN;
  YT=MARGEN;

  XDIF=XR-XL;
  YDIF=YB-YT;

  YMIN=-2.5;
  YMAX=-YMIN;
  XMIN=YMIN;
  XMAX=YMAX;

  XMIN=((XMIN*XDIF)/YDIF);
  XMAX=((XMAX*XDIF)/YDIF);

  SCALX=((double)XDIF/(double)(XMAX-XMIN));
  SCALY=((double)YDIF/(double)(YMAX-YMIN));

  UCS_XL=XL;
  UCS_XR=XL+(int)(YDIF*0.2);
  UCS_YT=YT+(int)(YDIF*0.8);
  UCS_YB=YB;
  UCS_XMIN=-1;
  UCS_XMAX=1;
  UCS_YMIN=-1;
  UCS_YMAX=1;

  UCS_XDIF=UCS_XR-UCS_XL;
  UCS_YDIF=UCS_YB-UCS_YT;

  UCS_SCALX=((double)UCS_XDIF/(double)(UCS_XMAX-UCS_XMIN));
  UCS_SCALY=((double)UCS_YDIF/(double)(UCS_YMAX-UCS_YMIN));

  init_rs();
  initpuntos();
  initmatgs0();

  int i;

  for(i=0;13>i;i++)
  {
    cmin.v[i]=127;
    cmax.v[i]=0;
  }

  for(i=0;19>i;i++)
    ang.v[i]=0;

  THETA=45;
  PHI=55;
  RHO=0;

  mview=mmview(THETA,PHI,RHO);

  CBackground=BLACK;
  CHand=GREEN;
  CSelection=CYAN;
  COrigin=YELLOW;
  CUcsOrigin=WHITE;
  CFrame=GREEN;

  miembro=mNada;
  flag1=F;
  flag2=F;
  flag3=F;
}

/*************************************************************************/
// Funcion: ~THandSim()
// Destructor de clase
/*************************************************************************/

THandSim::~THandSim()
{

}

/*************************************************************************/
// Funcion: SetupWindow()
// Inicializa el elemento visual
/*************************************************************************/

void THandSim::SetupWindow()
{
  //int gdriver=DETECT;
  //int gmode;
  int gdriver=VGA;
  int gmode=1;  //VGAMED 640*350 16c 2p

  initgraph(&gdriver,&gmode,"");
  settextjustify(CENTER_TEXT,CENTER_TEXT);
  settextstyle(SMALL_FONT,HORIZ_DIR,4);

}

/*************************************************************************/
// Funcion: CloseWindow()
```

```
// Destruye el elemento visual
/*************************************************************************/

void THandSim::CloseWindow()
{
  setcolor(CFrame);
  outtextxy(WI/2,HE-10,"SIMULACION INTERRUMPIDA, PRESIONE UNA TECLA");
  flushall();
  getch();
  closegraph();
  exit(0);
}


/*************************************************************************/
// Funcion: Paint()
// Trazado de la pantalla
/*************************************************************************/

void THandSim::Paint()
{
  static int page = 0;

  if(page==0)
    page=1;
  else
    page=0;

  setactivepage(page);

  clearviewport();
  frame(CFrame);
  ucs_icon(CUcsOrigin,0.5*U);
  origen(COrigin,U);
  mano(miembro);

  pt3d(RED,p_in(XM-1,YM-1,ZM-1));
  pt3d(RED,p_in(XM-1,YM-1,ZM+1));
  pt3d(RED,p_in(XM-1,YM+1,ZM-1));
  pt3d(RED,p_in(XM-1,YM+1,ZM+1));
  pt3d(RED,p_in(XM+1,YM-1,ZM-1));
  pt3d(RED,p_in(XM+1,YM-1,ZM+1));
  pt3d(RED,p_in(XM+1,YM+1,ZM-1));
  pt3d(RED,p_in(XM+1,YM+1,ZM+1));

  cal_icon(YELLOW,flag1);
  rec_icon(RED,flag2);
  play_icon(CYAN,flag3);

  setvisualpage(page);
}

/*************************************************************************/
// FUNCIONES DE LA TABLA DE RESPUESTA
/*************************************************************************/

/*************************************************************************/
// Funcion: PrgPause()
/*************************************************************************/

void THandSim::PrgPause()
{
  if(estado==eActivo)
    estado=ePausa;
  else
    if(estado==ePausa)
      estado=eActivo;
  Paint();
}

/*************************************************************************/
// Funcion: PrgRun()
// Ejecuta el programa
/*************************************************************************/

void THandSim::PrgRun()
{
  unsigned char t1;
  char t2;

  estado=eActivo;
  miembro=mNada;

  Paint();

  for(;;)
  {
    t2=lee_kb();
    if(t2!=NULL)
      EvChar(t2);

    if(test_rs())
    {
      t1=leer();

      if(t1==0xf0)
      {
        canal=get13();

        if(flag1)
        {
          cmin=minimo(canal,cmin);
          cmax=maximo(canal,cmax);
        }

        ang.v[3]=interpol(cmin.v[5],cmax.v[5],-45,45,canal.v[5]);
        ang.v[4]=interpol(cmin.v[4],cmax.v[4],-90,0,canal.v[4]);
        ang.v[5]=interpol(cmin.v[6],cmax.v[6],-90,0,canal.v[6]);
        ang.v[6]=interpol(cmin.v[0],cmax.v[0],-90,0,canal.v[0]);

        ang.v[8]=interpol(cmin.v[7],cmax.v[7],-90,0,canal.v[7]);
        ang.v[9]=interpol(cmin.v[1],cmax.v[1],-90,0,canal.v[1]);
        ang.v[10]=ang.v[9];

        ang.v[12]=interpol(cmin.v[8],cmax.v[8],-90,0,canal.v[8]);
        ang.v[13]=interpol(cmin.v[2],cmax.v[2],-90,0,canal.v[2]);
        ang.v[14]=ang.v[13];

        ang.v[16]=interpol(cmin.v[9],cmax.v[9],-90,0,canal.v[9]);
        ang.v[17]=interpol(cmin.v[3],cmax.v[3],-90,0,canal.v[3]);
        ang.v[18]=ang.v[17];

        if(flag2)
          salvar(ang);

        Paint();
      }
    }
  }
}

/*************************************************************************/
// Funcion: EvChar(unsigned char key)
/*************************************************************************/

void THandSim::EvChar(char key)
{
  char key2=toupper(key);

  if(estado==eActivo||estado==ePausa)
  {
    switch(key)
    {
      case 27: //VK_ESCAPE
        CloseWindow();
        break;

      case 9: //VK_TAB
        if(miembro==mAnular)
          miembro=mPalma;
        else
          miembro++;
        Paint();
        break;

      //case -99: // ALT + VK_RIGHT
      case 'l':
      case 'L':
        if(!(PHI==0)||(PHI==180)||(PHI==-180))
        {
          THETA+=DANG;
          mview=mmview(THETA,PHI,RHO);
          Paint();
          break;
        }
        else
          break;

      //case -101: //ALT + VK_LEFT
      case 'j':
      case 'J':
        if(!(PHI==0)||(PHI==180)||(PHI==-180))
        {
          THETA-=DANG;
          mview=mmview(THETA,PHI,RHO);
          Paint();
          break;
        }
        else
          break;

      //case -96: //ALT + VK_DOWN
      case 'k':
      case 'K':
        PHI+=DANG;
        mview=mmview(THETA,PHI,RHO);
```

```
    Paint();
    break;

  //case -104: //ALT + VK_UP:
  case 'i':
  case 'I':
    PHI-=DANG;
    mview=mmview(THETA,PHI,RHO);
    Paint();
    break;

  case 63: //VK_F5
    RHO+=DANG;
    mview=mmview(THETA,PHI,RHO);
    Paint();
    break;

  case 64: //VK_F6
    RHO-=DANG;
    mview=mmview(THETA,PHI,RHO);
    Paint();
    break;

  case -123: //VK_F11
    THETA=45;
    PHI=55;
    RHO=0;
    mview=mmview(THETA,PHI,RHO);
    Paint();
    break;

  case -122: //VK_F12
    miembro=mNada;
    for(int i=0;19>i;i++)
      ang.v[i]=0;
    Paint();
    break;
}

switch(key2)
{
  case 'C':
    if(!flag1)
    {
      for(int i=0;13>i;i++)
      {
        cmin.v[i]=127;
        cmax.v[i]=0;
      }
      flag1=T;
    }
    else
      flag1=F;
    break;

  case 'G':
    if(!flag2)
      flag2=T;
    else
      flag2=F;
    break;

  case 'P':
    if(!flag2)
    {
      flag3=T;
      cargar(TIME);
      flag3=F;
    }
    break;
}

switch(miembro)
{
  case mPalma:

    if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
      key2!='D')
      break;

    if((key2=='Q')&&(ang.v[0]<LSPL1))
      ang.v[0]+=DANG;
    if((key2=='A')&&(ang.v[0]>LIPL1))
      ang.v[0]-=DANG;

    if((key2=='W')&&(ang.v[1]<LSPL2))
      ang.v[1]+=DANG;
    if((key2=='S')&&(ang.v[1]>LIPL2))
      ang.v[1]-=DANG;

    if((key2=='E')&&(ang.v[2]<LSPL3))
      ang.v[2]+=DANG;
    if((key2=='D')&&(ang.v[2]>LIPL3))
      ang.v[2]-=DANG;

    Paint();
    break;

  case mPulgar:

    if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
      key2!='D'&&key2!='R'&&key2!='F')
      break;

    if((key2=='Q')&&(ang.v[3]<LSP1))
      ang.v[3]+=DANG;
    if((key2=='A')&&(ang.v[3]>LIP1))
      ang.v[3]-=DANG;

    if((key2=='W')&&(ang.v[4]<LSP2))
      ang.v[4]+=DANG;
    if((key2=='S')&&(ang.v[4]>LIP2))
      ang.v[4]-=DANG;

    if((key2=='E')&&(ang.v[5]<LSP3))
      ang.v[5]+=DANG;
    if((key2=='D')&&(ang.v[5]>LIP3))
      ang.v[5]-=DANG;

    if((key2=='R')&&(ang.v[6]<LSP4))
      ang.v[6]+=DANG;
    if((key2=='F')&&(ang.v[6]>LIP4))
      ang.v[6]-=DANG;

    Paint();
    break;

  case mIndice:

    if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
      key2!='D')
      break;

    if((key2=='Q')&&(ang.v[7]<LSD1))
      ang.v[7]+=DANG;
    if((key2=='A')&&(ang.v[7]>LID1))
      ang.v[7]-=DANG;

    if((key2=='W')&&(ang.v[8]<LSD2))
      ang.v[8]+=DANG;
    if((key2=='S')&&(ang.v[8]>LID2))
      ang.v[8]-=DANG;

    if((key2=='E')&&(ang.v[9]<LSD3))
      ang.v[9]+=DANG,ang.v[10]+=DANG;
    if((key2=='D')&&(ang.v[9]>LID3))
      ang.v[9]-=DANG,ang.v[10]-=DANG;

    Paint();
    break;

  case mMedio:

    if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
      key2!='D')
      break;

    if((key2=='Q')&&(ang.v[11]<LSD1))
      ang.v[11]+=DANG;
    if((key2=='A')&&(ang.v[11]>LID1))
      ang.v[11]-=DANG;

    if((key2=='W')&&(ang.v[12]<LSD2))
      ang.v[12]+=DANG;
    if((key2=='S')&&(ang.v[12]>LID2))
      ang.v[12]-=DANG;

    if((key2=='E')&&(ang.v[13]<LSD3))
      ang.v[13]+=DANG,ang.v[14]+=DANG;
    if((key2=='D')&&(ang.v[13]>LID3))
      ang.v[13]-=DANG,ang.v[14]-=DANG;

    Paint();
    break;

  case mAnular:

    if(key2!='Q'&&key2!='A'&&key2!='W'&&key2!='S'&&key2!='E'&&
      key2!='D')
      break;

    if((key2=='Q')&&(ang.v[15]<LSD1))
      ang.v[15]+=DANG;
    if((key2=='A')&&(ang.v[15]>LID1))
      ang.v[15]-=DANG;
```

```
        if((key2=='W')&&(ang.v[16]<LSD2))
          ang.v[16]+=DANG;
        if((key2=='S')&&(ang.v[16]>LID2))
          ang.v[16]-=DANG;

        if((key2=='E')&&(ang.v[17]<LSD3))
          ang.v[17]+=DANG,ang.v[18]+=DANG;
        if((key2=='D')&&(ang.v[17]>LID3))
          ang.v[17]-=DANG,ang.v[18]-=DANG;

        Paint();
        break;
    }
  }
}

/*************************************************************************/
// FUNCIONES DEL MODELO MATEMATICO
/*************************************************************************/

/*************************************************************************/
// Funcion: initpuntos(void)
// Inicializa puntos clave de la mano
/*************************************************************************/

void THandSim::initpuntos(void)
{
// vector nulo
  NULO=v_in(0,0,0);

// eje x para el origen del sistema inercial
  I=v_in(1,0,0);
// eje y para el origen del sistema inercial
  J=v_in(0,1,0);
// eje z para el origen del sistema inercial
  K=v_in(0,0,1);

// palma: origen
  Q[0]=p_in(XPL,YPL,ZPL);

// eje x para el origen del pulgar
  IP=v_unit(v_in(C(B)*C(A),C(B)*S(A),S(B)));
// eje y para el origen del pulgar
  JP=v_unit(v_in(-S(A),C(A),0));
// eje z para el origen del pulgar
  KP=v_unit(v_in(-S(B)*C(A),-S(B)*S(A),C(B)));

// pulgar: origen falange
  Q[1]=p_in(XP,YP,ZP);
// pulgar: origen falangina
  Q[2]=p_in(XP+L1P*JP.v[0],YP+L1P*JP.v[1],ZP+L1P*JP.v[2]);
// pulgar: origen falangeta
  Q[3]=p_in(XP+(L1P+L2P)*JP.v[0],YP+(L1P+L2P)*JP.v[1],
        ZP+(L1P+L2P)*JP.v[2]);
// pulgar: extremo falangeta
  Q[4]=p_in(XP+(L1P+L2P+L3P)*JP.v[0],YP+(L1P+L2P+L3P)*JP.v[1],
        ZP+(L1P+L2P+L3P)*JP.v[2]);

// indice: origen falange
  Q[5]=p_in(XI,YI,ZI);
// indice: origen falangina
  Q[6]=p_in(XI,YI+L1I,ZI);
// indice: origen falangeta
  Q[7]=p_in(XI,YI+L1I+L2I,ZI);
// indice: extremo falangeta
  Q[8]=p_in(XI,YI+L1I+L2I+L3I,ZI);

// medio: origen falange
  Q[9]=p_in(XM,YM,ZM);
// medio: origen falangina
  Q[10]=p_in(XM,YM+L1M,ZM);
// medio: origen falangeta
  Q[11]=p_in(XM,YM+L1M+L2M,ZM);
// medio: extremo falangeta
  Q[12]=p_in(XM,YM+L1M+L2M+L3M,ZM);

// anular: origen falange
  Q[13]=p_in(XA,YA,ZA);
// anular: origen falangina
  Q[14]=p_in(XA,YA+L1A,ZA);
// anular: origen falangeta
  Q[15]=p_in(XA,YA+L1A+L2A,ZA);
// anular: extremo falangeta
  Q[16]=p_in(XA,YA+L1A+L2A+L3A,ZA);

// palma
  bb[0]=v48_in(LPAL/1.5,LPAL,HH,LPAL);

// pulgar: falange
  bb[1]=v48_in(WW,WW,HH,L1P);
// pulgar: falangina
  bb[2]=v48_in(WW,WW,HH,L2P);
// pulgar: falangeta
  tt[0]=v412_in(WW,HH,WW/2,HH/2,L3P,0.6);

// indice: falange
  bb[3]=v48_in(WW,WW,HH,L1I);
// indice: falangina
  bb[4]=v48_in(WW,WW,HH,L2I);
// indice: falangeta
  tt[1]=v412_in(WW,HH,WW/2,HH/2,L3I,0.6);

// medio: falange
  bb[5]=v48_in(WW,WW,HH,L1M);
// medio: falangina
  bb[6]=v48_in(WW,WW,HH,L2M);
// medio: falangeta
  tt[2]=v412_in(WW,HH,WW/2,HH/2,L3M,0.6);

// anular: falange
  bb[7]=v48_in(WW,WW,HH,L1A);
// anular: falangina
  bb[8]=v48_in(WW,WW,HH,L2A);
// anular: falangeta
  tt[3]=v412_in(WW,HH,WW/2,HH/2,L3A,0.6);
}

/*************************************************************************/
// Funcion: mgs0(const v4& p)
// Define la matriz gs0
//
// p: punto base del eslabon
// Posicion inicial de la palma
// <Q[0]> palma
// Posicion inicial de la falange proximal
// <Q[5]> indice, <Q[9]> medio, <Q[13]> anular
// Posicion inicial de la falange media
// <Q[6]> indice, <Q[10]> medio, <Q[14]> anular
// Posicion inicial de la falange distal
// <Q[7]> indice, <Q[11]> medio, <Q[15]> anular
// Posicion inicial del extremo de la falange distal
// <Q[8]> indice, <Q[12]> medio, <Q[16]> anular
/*************************************************************************/

m4 THandSim::mgs0(const v4& p)
{
  double x,y,z;
  x=p.v[0];y=p.v[1];z=p.v[2];

  m4 m={{{1,0,0,x},
         {0,1,0,y},
         {0,0,1,z},
         {0,0,0,1}}};

  return m;
}

/*************************************************************************/
// Funcion: mgsp0(const v4& p)
// Define la matriz gsp0
// Posicion inicial de los eslabones del pulgar
//
// p: punto base del eslabon
// <Q[1]> f. prox., <Q[2]> f. med., <Q[3]> f. dist., <Q[4]> ext. dist.
/*************************************************************************/

m4 THandSim::mgsp0(const v4& p)
{
  double x,y,z;
  x=p.v[0];y=p.v[1];z=p.v[2];

  m4 m={{{IP.v[0],JP.v[0],KP.v[0],x},
         {IP.v[1],JP.v[1],KP.v[1],y},
         {IP.v[2],JP.v[2],KP.v[2],z},
         {  0,    0,    0,    1}}};

  return m;
}

/*************************************************************************/
// Funcion: initmatgs0(void)
// Inicializa las matrices gs0 (posicion inicial)
/*************************************************************************/

void THandSim::initmatgs0(void)
{
  gs0[0]=mgs0(Q[0]);    //palma

  gs0[1]=mgsp0(Q[1]);   //pulgar(falange proximal)
  gs0[2]=mgsp0(Q[2]);   //pulgar(falange media)
  gs0[3]=mgsp0(Q[3]);   //pulgar(falange distal)

  gs0[4]=mgs0(Q[5]);    //indice(falange proximal)
  gs0[5]=mgs0(Q[6]);    //indice(falange media)
  gs0[6]=mgs0(Q[7]);    //indice(falange distal)
```

```
  gs0[7]=mgs0(Q[9]);     //medio(falange proximal)
  gs0[8]=mgs0(Q[10]);    //medio(falange media)
  gs0[9]=mgs0(Q[11]);    //medio(falange distal)

  gs0[10]=mgs0(Q[13]);   //anular(falange proximal)
  gs0[11]=mgs0(Q[14]);   //anular(falange media)
  gs0[12]=mgs0(Q[15]);   //anular(falange distal)

  gs0[13]=mgsp0(Q[4]);   //pulgar(extremo falange distal)
  gs0[14]=mgs0(Q[8]);    //indice(extremo falange distal)
  gs0[15]=mgs0(Q[12]);   //medio(extremo falange distal)
  gs0[16]=mgs0(Q[16]);   //anular(extremo falange distal)
}

/*************************************************************************/
// Funcion: ee1(double a,const v4& p)
// Calcula la matriz de transformacion No. 1 (ROLL)
// Rotacion en el eje "y"
//
// a: angulo de rotacion en grados
// p: punto base del eslabon
// <Q[0]> palma
/*************************************************************************/

m4 THandSim::ee1(double a,const v4& p)
{
  double s,c,v,x,z;
  s=S(a);c=C(a);v=1-c;
  x=p.v[0];z=p.v[2];

  m4 m={{{ c,0,s,x*v-z*s},
         { 0,1,0,    0},
         {-s,0,c,x*s+z*v},
         { 0,0,0,    1}}};

  return m;
}

/*************************************************************************/
// Funcion: ee2(double a,const v4& p)
// Calcula la matriz de transformacion No. 2 (PITCH)
// Rotacion en el eje "x"
//
// a: angulo de rotacion en grados
// p: punto base del eslabon
// <Q[0]> palma
// <Q[5]> prox. indice, <Q[9]> prox. medio,  <Q[13]> prox. anular
// <Q[6]> med.indice,   <Q[10]> med. medio,  <Q[14]> med. anular
// <Q[7]> dist. indice, <Q[11]> dist. medio, <Q[15]> dist. anular
/*************************************************************************/

m4 THandSim::ee2(double a,const v4& p)
{
  double s,c,v,y,z;
  s=S(a);c=C(a);v=1-c;
  y=p.v[1];z=p.v[2];

  m4 m={{{1,0, 0,    0},
         {0,c,-s, y*v+z*s},
         {0,s, c,-y*s+z*v},
         {0,0, 0,    1}}};

  return m;
}

/*************************************************************************/
// Funcion: ee3(double a,const v4& p)
// Calcula la matriz de transformacion No. 3 (YAW)
// Rotacion en el eje "z"
//
// a: angulo de rotacion en grados
// p: punto base del eslabon
// <Q[0]> palma, <Q[5]> indice, <Q[9]> medio, <Q[13]> anular
/*************************************************************************/

m4 THandSim::ee3(double a,const v4& p)
{
  double s,c,v,x,y;
  s=S(a);c=C(a);v=1-c;
  x=p.v[0];y=p.v[1];

  m4 m={{{c,-s,0, x*v+y*s},
         {s, c,0,-x*s+y*v},
         {0, 0,1,    0},
         {0, 0,0,    1}}};

  return m;
}

/*************************************************************************/
// Funcion: ee4(double a,const v4& w,const v4& p)
// Calcula la matriz de transformacion No. 4
// Rotacion en el eje "z" y "x" de cualquier eslabon del pulgar
//
// a: angulo de rotacion en grados
// w: vector que representa el eje de rotacion
// <IP> eje x', <KP> eje z'
// p:  punto base del eslabon
// <Q[1]> f. proximal, <Q[2]> f. media, <Q[3]> f. distal
/*************************************************************************/

m4 THandSim::ee4(double a,const v4& w,const v4& p)
{
  double s,c,v,w1,w2,w3,x,y,z;
  s=S(a);c=C(a);v=1-c;
  w1=w.v[0];w2=w.v[1];w3=w.v[2];
  x=p.v[0];y=p.v[1];z=p.v[2];

  m4 m={{{w1*w1*v+c,
          w1*w2*v-w3*s,
          w1*w3*v+w2*s,
          x*v*(1-w1*w1)+y*(s*w3-w1*w2*v)+z*(-s*w2-w1*w3*v)},

         {w1*w2*v+w3*s,
          w2*w2*v+c,
          w2*w3*v-w1*s,
          x*(-s*w3-w1*w2*v)+y*v*(1-w2*w2)+z*(s*w1-w2*w3*v)},

         {w1*w3*v-w2*s,
          w2*w3*v+w1*s,
          w3*w3*v+c,
          x*(s*w2-w1*w3*v)+y*(-s*w1-w2*w3*v)+z*v*(1-w3*w3)},

         {0,
          0,
          0,
          1}}};
  return m;
}

/*************************************************************************/
// Funcion: mg(const v19& a)
// Calcula las matrices g
//
// a: vector de todos los angulos de las articulaciones
/*************************************************************************/

void THandSim::mg(const v19& a)
{
//palma
  e[0]=ee1(a.v[0],Q[0]);
  e[1]=ee2(a.v[1],Q[0]);
  e[2]=ee3(a.v[2],Q[0]);
//pulgar
  e[3]=ee4(a.v[3],KP,Q[1]);
  e[4]=ee4(a.v[4],IP,Q[1]);
  e[5]=ee4(a.v[5],IP,Q[2]);
  e[6]=ee4(a.v[6],IP,Q[3]);
//indice
  e[7]=ee3(a.v[7],Q[5]);
  e[8]=ee2(a.v[8],Q[5]);
  e[9]=ee2(a.v[9],Q[6]);
  e[10]=ee2(a.v[10],Q[7]);
//medio
  e[11]=ee3(a.v[11],Q[9]);
  e[12]=ee2(a.v[12],Q[9]);
  e[13]=ee2(a.v[13],Q[10]);
  e[14]=ee2(a.v[14],Q[11]);
//anular
  e[15]=ee3(a.v[15],Q[13]);
  e[16]=ee2(a.v[16],Q[13]);
  e[17]=ee2(a.v[17],Q[14]);
  e[18]=ee2(a.v[18],Q[15]);

//palma
  gs[0]=mm_mult(mm_mult(e[0],e[1]),e[2]);
//pulgar
  gs[1]=mm_mult(mm_mult(gs[0],e[3]),e[4]);
  gs[2]=mm_mult(gs[1],e[5]);
  gs[3]=mm_mult(gs[2],e[6]);
//indice
  gs[4]=mm_mult(mm_mult(gs[0],e[7]),e[8]);
  gs[5]=mm_mult(gs[4],e[9]);
  gs[6]=mm_mult(gs[5],e[10]);
//medio
  gs[7]=mm_mult(mm_mult(gs[0],e[11]),e[12]);
  gs[8]=mm_mult(gs[7],e[13]);
  gs[9]=mm_mult(gs[8],e[14]);
//anular
  gs[10]=mm_mult(mm_mult(gs[0],e[15]),e[16]);
  gs[11]=mm_mult(gs[10],e[17]);
  gs[12]=mm_mult(gs[11],e[18]);

//palma
  g[0]=mm_mult(gs[0],gs0[0]);
//pulgar
```

```
  g[1]=mm_mult(gs[1],gs0[1]);
  g[2]=mm_mult(gs[2],gs0[2]);
  g[3]=mm_mult(gs[3],gs0[3]);
//indice
  g[4]=mm_mult(gs[4],gs0[4]);
  g[5]=mm_mult(gs[5],gs0[5]);
  g[6]=mm_mult(gs[6],gs0[6]);
//medio
  g[7]=mm_mult(gs[7],gs0[7]);
  g[8]=mm_mult(gs[8],gs0[8]);
  g[9]=mm_mult(gs[9],gs0[9]);
//anular
  g[10]=mm_mult(gs[10],gs0[10]);
  g[11]=mm_mult(gs[11],gs0[11]);
  g[12]=mm_mult(gs[12],gs0[12]);
}

/*************************************************************************/
// Funcion: m_rot(double a,const v4& b,double x,double y,double z)
// Calcula matrices de rotacion
//
// a: angulo de rotacion
// b: vector que representa al eje de rotacion
// x: coordenada x del punto de anclaje del sist. de ref.
// y: coordenada y del punto de anclaje del sist. de ref.
// z: coordenada z del punto de anclaje del sist. de ref.
/*************************************************************************/

m4 THandSim::m_rot(double a,const v4& b,double x,double y,double z)
{
  v4 w;
  double s,c,v;
  s=S(a);c=C(a);v=1-c;
  w=v_unit(b);

  m4 m={{{w.v[0]*w.v[0]*v+c,
          w.v[0]*w.v[1]*v-w.v[2]*s,
          w.v[0]*w.v[2]*v+w.v[1]*s,
          x},

         {w.v[0]*w.v[1]*v+w.v[2]*s,
          w.v[1]*w.v[1]*v+c,
          w.v[1]*w.v[2]*v-w.v[0]*s,
          y},

        {w.v[0]*w.v[2]*v-w.v[1]*s,
         w.v[1]*w.v[2]*v+w.v[0]*s,
         w.v[2]*w.v[2]*v+c,
         z},

        {0,
         0,
         0,
         1}}};

  return m;
}

/*************************************************************************/
// FUNCIONES DE PARA OPERAR CON MATRICES
/*************************************************************************/

/*************************************************************************/
// Funcion: m_esc(double esc,const m4& a)
// Multiplica un escalar por una matriz
//
// esc: escalar por el cual se multiplica la matriz
// a: matriz objeto de la operacion
/*************************************************************************/

m4 THandSim::m_esc(double esc,const m4& a)
{
  int i,j;
  m4 m;
  for(i=0;ORD>i;i++)
    for(j=0;ORD>j;j++)
      m.m[i][j]=esc*a.m[i][j];
  return m;
}

/*************************************************************************/
// Funcion: m_transp(const m4& a)
// Obtiene la transpuesta de una matriz
//
// a: matriz objeto de la operacion
/*************************************************************************/

m4 THandSim::m_transp(const m4& a)
{
  int i,j;
  m4 m;
  for(i=0;ORD>i;i++)
    for(j=0;ORD>j;j++)
      m.m[j][i]=a.m[i][j];
  return m;
}

/*************************************************************************/
// Funcion: mm_mult(const m4& a,const m4& b)
// Multiplica una matriz por otra (matrices cuadradas de tamano ORD)
//
// a: matriz que funciona como primer factor
// b: matriz que funciona como segundo factor
/*************************************************************************/

m4 THandSim::mm_mult(const m4& a,const m4& b)
{
  int i,j,k;
  m4 m;
  for(i=0;ORD>i;i++)
    for(j=0;ORD>j;j++)
    {
      m.m[i][j]=0;
      for(k=0;ORD>k;k++)
        m.m[i][j]=m.m[i][j]+a.m[i][k]*b.m[k][j];
    }
  return m;
}

/*************************************************************************/
// Funcion: mv_mult(const m4& a,const v4& b)
// Multiplica matrices por vectores
//
// a: matriz que funciona como primer factor
// b: vector que funciona como segundo factor
/*************************************************************************/

v4 THandSim::mv_mult(const m4& a,const v4& b)
{
  int i,k;
  v4 v;
  for(i=0;ORD>i;i++)
  {
    v.v[i]=0;
    for(k=0;ORD>k;k++)
      v.v[i]=v.v[i]+a.m[i][k]*b.v[k];
  }
  return v;
}

/*************************************************************************/
// Funcion: m_inv(const m4& a)
// Obtiene la inversa de una matriz tipo g (caso especial)
//
// a: matriz objeto de la inversion
/*************************************************************************/

m4 THandSim::m_inv(const m4& a)
{
  int i,j,k;
  m4 m=a;
  for(i=0;(ORD-1)>i;i++)
    for(j=0;(ORD-1)>j;j++)
      m.m[j][i]=a.m[i][j];
  for(i=0;(ORD-1)>i;i++)
  {
    m.m[i][ORD-1]=0;
    for(k=0;(ORD-1)>k;k++)
      m.m[i][ORD-1]=m.m[i][ORD-1]-m.m[i][k]*a.m[k][ORD-1];
  }
  return m;
}

/*************************************************************************/
// FUNCIONES PARA OPERAR CON VECTORES
/*************************************************************************/

/*************************************************************************/
// Funcion: p_in(double x,double y,double z,double k = 1)
// Ingresa datos en un punto
//
// x: componente x del punto
// y: componente y del punto
// z: componente z del punto
// k: cuarto elemento comodin
/*************************************************************************/

v4 THandSim::p_in(double x,double y,double z,double k)
{
  v4 v={{x,y,z,k}};
  return v;
}

/*************************************************************************/
// Funcion: v_in(double x,double y,double z,double k = 0)
// Ingresa datos en un vector
```

```
//
// x: componente x del vector
// y: componente y del vector
// z: componente z del vector
// k: cuarto elemento comodin
/*************************************************************************/

v4 THandSim::v_in(double x,double y,double z,double k)
{
  v4 v={{x,y,z,k}};
  return v;
}

/*************************************************************************/
// Funcion: v_sum(const v4& a,const v4& b,char oper)
// Suma dos vectores en forma algebraica
//
// a: vector que funciona como primer sumando
// b: vector que funciona como segundo sumando
// oper: tipo de operacion, <'+'> suma, <'-'> resta
/*************************************************************************/

v4 THandSim::v_sum(const v4& a,const v4& b,char oper)
{
  int i;
  v4 v;
  if(oper=='+')       //'+'=43
    for(i=0;(ORD-1)>i;i++)
      v.v[i]=a.v[i]+b.v[i];
  if(oper=='-')       //'-'=45
    for(i=0;(ORD-1)>i;i++)
      v.v[i]=a.v[i]-b.v[i];
  v.v[(ORD-1)]=0;
  return v;
}

/*************************************************************************/
// Funcion: v_esc(double esc,const v4& a)
// Calcula el producto de un escalar con un vector
//
// esc: escalar por el cual se multiplica el vector
// a: vector objeto de la operacion
/*************************************************************************/

v4 THandSim::v_esc(double esc,const v4& a)
{
  int i;
  v4 v;
  for(i=0;(ORD-1)>i;i++)
    v.v[i]=esc*a.v[i];
  v.v[ORD-1]=0.
  return v;
}

/*************************************************************************/
// Funcion: dot(const v4& a,const v4& b)
// Calcula el producto punto entre dos vectores
//
// a: vector que funciona como primer operando
// b: vector que funciona como segundo operando
/*************************************************************************/

double THandSim::dot(const v4& a,const v4& b)
{
  int i;
  double prod=0;
  for(i=0;(ORD-1)>i;i++)
    prod=prod+a.v[i]*b.v[i];
  return prod;
}

/*************************************************************************/
// Funcion: cross(const v4& a,const v4& b)
// Calcula el producto cruz entre dos vectores
//
// a: vector que funciona como primer operando
// b: vector que funciona como segundo operando
/*************************************************************************/

v4 THandSim::cross(const v4& a,const v4& b)
{
  v4 v={{a.v[1]*b.v[2]-a.v[2]*b.v[1],
         a.v[2]*b.v[0]-a.v[0]*b.v[2],
         a.v[0]*b.v[1]-a.v[1]*b.v[0],
         0}};
  return v;
}

/*************************************************************************/
// Funcion: mag2(const v4& a)
// Obtiene el cuadrado de la magnitud de un vector
//
// a: vector objeto de la operacion
/*************************************************************************/

double THandSim::mag2(const v4& a)
{
  return dot(a,a);
}

/*************************************************************************/
// Funcion: mag(const v4& a)
// Obtiene la magnitud de un vector
//
// a: vector objeto de la operacion
/*************************************************************************/

double THandSim::mag(const v4& a)
{
  return sqrt(mag2(a));
}

/*************************************************************************/
// Funcion: v_unit(const v4& a)
// Normaliza un vector
//
// a: vector objeto de la normalizacion
/*************************************************************************/

v4 THandSim::v_unit(const v4& a)
{
  return v_esc((1/mag(a)),a);
}

/*************************************************************************/
// FUNCIONES PARA OPERAR CON ESTRUCTURAS DE DATOS ESPECIALES
/*************************************************************************/

/*************************************************************************/
// Funcion: v48_in(double a,double b,double h,double l)
// Ingresa datos en un elemanto v48 (aristas de barras y prismas trapezoidales)
//
// a: ancho en un extremo de la barra
// b: ancho en el otro extremo de la barra
// h: altura de la barra
// l: longitud de la barra
/*************************************************************************/

v48 THandSim::v48_in(double a,double b,double h,double l)
{
  v48 v={{{{ a/2,0,-h/2,1}},
          {{-a/2,0,-h/2,1}},
          {{ a/2,0, h/2,1}},
          {{-a/2,0, h/2,1}},
          {{ b/2,l,-h/2,1}},
          {{-b/2,l,-h/2,1}},
          {{ b/2,l, h/2,1}},
          {{-b/2,l, h/2,1}}}};
  return v;
}

/*************************************************************************/
// Funcion: v412_in(double w1,double h1,double w2,double h2,double l,double r)
// Ingresa datos en un elemanto v412 (aristas prismas de seccion variable)
//
// w1: ancho en la primera seccion de la barra
// h1: altura en la primera seccion de la barra
// w2: ancho en el otro extremo de la barra
// h2: altura en el otro extremo de la barra
// l: longitud de la barra
// r: relacion de longiudes de la primera seccion a la longitud total
/*************************************************************************/

v412 THandSim::v412_in(double w1,double h1,double w2,double h2,
                       double l,double r)
{
  double l1=r*l;
  v412 v={{{{ w1/2, 0,-h1/2,1}},
           {{-w1/2, 0,-h1/2,1}},
           {{ w1/2, 0, h1/2,1}},
           {{-w1/2, 0, h1/2,1}},
           {{ w1/2,l1,-h1/2,1}},
           {{-w1/2,l1,-h1/2,1}},
           {{ w1/2,l1, h1/2,1}},
           {{-w1/2,l1, h1/2,1}},
           {{ w2/2, l,-h2/2,1}},
           {{-w2/2, l,-h2/2,1}},
           {{ w2/2, l, h2/2,1}},
           {{-w2/2, l, h2/2,1}}}};
  return v;
}

/*************************************************************************/
// FUNCIONES GRAFICAS
/*************************************************************************/
```

```
/*************************************************************************/
// Funcion: mmview(double theta,double& phi,double rho)
// Calcula la matriz de pasa de coord. en 3d a coord. graficas
//
// theta: azimut
// phi:   elevacion
// rho:   giro de vista
/*************************************************************************/

m4 THandSim::mmview(double theta,double& phi,double rho)
{
  double c=C(rho),s=S(rho);
  static v4 u;
  v4 v,n,vp;

  if(phi==-180)
    phi=180;
  if(phi>180)
    phi=phi-360;

  n=v_in(S(phi)*C(theta),S(phi)*S(theta),C(phi));
  vp=v_in(0,0,((phi>0)&&(phi<180))? 1 : -1);

  if((n.v[0]==0)&&(n.v[1]==0))
    v=cross(n,u);
  else
  {
    v=v_unit(v_sum(vp,v_esc(dot(vp,n),n),'-'));
    u=cross(v,n);
  }

  m4 m1={{{ c,s,0,0},
          {-s,c,0,0},
          { 0,0,1,0},
          { 0,0,0,1}}};

  m4 m2={{{u.v[0],u.v[1],u.v[2],0},
          {v.v[0],v.v[1],v.v[2],0},
          {n.v[0],n.v[1],n.v[2],0},
          {   0,   0,   0,1}}};


  m4 m3=mm_mult(m1,m2);

  return m3;
}

/*************************************************************************/
// Funcion: vscreen(double d,const v4& a,const m4& m)
// Transforma las cood. en 3d a coord. de la pantalla
//
// d: distancia de la camara al plano de proyeccion de la imagen
// a: vector que representa un punto en el espacio
// m: matriz de pasa de coord. en 3d a coord. graf.
/*************************************************************************/

TPoint THandSim::vscreen(double /*d*/,const v4& a,const m4& m)
{
  v4 v;
  TPoint p;
  v=mv_mult(m,a);
  p.x=(int)(XL+(v.v[0]-XMIN)*SCALX);
  p.y=(int)(YB-(v.v[1]-YMIN)*SCALY);
//  p.x=(int)((((v.v[0]*d)/fabs(v.v[2]))-XMIN)*SCALX);
//  p.y=(int)(YDIF-(((v.v[1]*d)/fabs(v.v[2]))-YMIN)*SCALY);
  return p;
}

/*************************************************************************/
// Funcion: ln(TPoint& p1,TPoint& p2)
// Dibuja una linea dados los dos extremos
//
// p1: punto inicial
// p2: punto final
/*************************************************************************/

void THandSim::ln(TPoint& p1,TPoint& p2)
{
  line(p1.x,p1.y,p2.x,p2.y);
}

/*************************************************************************/
// Funcion: frame(int color)
// Dibuja el marco de presentacion
/*************************************************************************/

void THandSim::frame(int color)
{
  setcolor(color);
  outtextxy(WI/2,10,"MANIPULADOR ANTROPOMORFICO TELEOPERADO");
  rectangle(XL,YT,XR,YB);
}

/*************************************************************************/
// Funcion: ucs_icon(int color,double l)
// Crea el icono del sistema universal de coordenadas
//
// color: color del origen
// l: longitud de los ejes
/*************************************************************************/

void THandSim::ucs_icon(int color,double l)
{
  int i;
  v4 v;
  TPoint p2d[7];
  v4 p3d[]={{{0,0,0,1}},      //origen
            {{l,0,0,1}},      //eje X
            {{0,l,0,1}},      //eje Y
            {{0,0,l,1}},      //eje Z
            {{1.3*l,0,0,1}},  //etiqueta de eje "X"
            {{0,1.3*l,0,1}},  //etiqueta de eje "Y"
            {{0,0,1.3*l,1}}};  //etiqueta de eje "Z"

  for(i=0;7>i;i++)
  {
    v=mv_mult(mview,p3d[i]);
    p2d[i].x=(int)(UCS_XL+(v.v[0]-UCS_XMIN)*UCS_SCALX);
    p2d[i].y=(int)(UCS_YB-(v.v[1]-UCS_YMIN)*UCS_SCALY);
  }

  setcolor(color);
  ln(p2d[0],p2d[1]);
  ln(p2d[0],p2d[2]);
  ln(p2d[0],p2d[3]);

  outtextxy(p2d[4].x,p2d[4].y,"X");
  outtextxy(p2d[5].x,p2d[5].y,"Y");
  outtextxy(p2d[6].x,p2d[6].y,"Z");
}

/*************************************************************************/
// Funcion: origen(int color,double l)
// Crea el origen en la pantalla
//
// color: color del origen
// l: longitud de los ejes
/*************************************************************************/

void THandSim::origen(int color,double l)
{
  int i;
  TPoint p2d[7];
  v4 p3d[]={{{0,0,0,1}},      //origen
            {{l,0,0,1}},      //eje X
            {{0,l,0,1}},      //eje Y
            {{0,0,l,1}},      //eje Z
            {{1.3*l,0,0,1}},  //etiqueta de eje "X"
            {{0,1.3*l,0,1}},  //etiqueta de eje "Y"
            {{0,0,1.3*l,1}}};  //etiqueta de eje "Z"

  for(i=0;7>i;i++)
    p2d[i]=vscreen(D,p3d[i],mview);

  setcolor(color);
  ln(p2d[0],p2d[1]);
  ln(p2d[0],p2d[2]);
  ln(p2d[0],p2d[3]);

  outtextxy(p2d[4].x,p2d[4].y,"X");
  outtextxy(p2d[5].x,p2d[5].y,"Y");
  outtextxy(p2d[6].x,p2d[6].y,"Z");
}

/*************************************************************************/
// Funcion: sistref(int color,double l,const m4& rot)
// Crea sistemas de referencia en la pantalla
//
// color: color del origen
// l: longitud de los ejes
// rot: matriz de rotacion asociado a dicho sistema
/*************************************************************************/

void THandSim::sistref(int color,double l,const m4& rot)
{
  int i;
  TPoint p2d[4];
  v4 p3d[]={{{0,0,0,1}},
            {{l,0,0,1}},
            {{0,l,0,1}},
            {{0,0,l,1}}};

  for(i=0;4>i;i++)
     p2d[i]=vscreen(D.mv_mult(rot,p3d[i]),mview);

  setcolor(color);
```

```
  ln(p2d[0],p2d[1]);
  ln(p2d[0],p2d[2]);
  ln(p2d[0],p2d[3]);
}

/*************************************************************************/
// Funcion: barra(v48& p3d,const m4& rot)
// Crea una barra en el espacio
//
// p3d: coordenadas de las aristas de la barra
// rot: matriz de rotacion asociada a la barra
/*************************************************************************/

void THandSim::barra(v48& p3d,const m4& rot)
{
  int i;
  TPoint p2d[8];

  for(i=0;8>i;i++)
     p2d[i]=vscreen(D,mv_mult(rot,p3d.v[i]),mview);

  ln(p2d[0],p2d[4]);
  ln(p2d[4],p2d[6]);
  ln(p2d[6],p2d[2]);
  ln(p2d[2],p2d[3]);
  ln(p2d[3],p2d[7]);
  ln(p2d[7],p2d[5]);
  ln(p2d[5],p2d[1]);
  ln(p2d[1],p2d[0]);
  ln(p2d[0],p2d[2]);
  ln(p2d[1],p2d[3]);
  ln(p2d[4],p2d[5]);
  ln(p2d[6],p2d[7]);
}

/*************************************************************************/
// Funcion: tip(v412& p3d,const m4& rot)
// Crea la punta de los dedos
//
// p3d: coordenadas de las aristas de la barra
// rot: matriz de rotacion asociada a la barra
/*************************************************************************/

void THandSim::tip(v412& p3d,const m4& rot)
{
  int i;
  TPoint p2d[12];

  for(i=0;12>i;i++)
     p2d[i]=vscreen(D,mv_mult(rot,p3d.v[i]),mview);

  ln(p2d[0],p2d[4]);
  ln(p2d[4],p2d[8]);
  ln(p2d[8],p2d[10]);
  ln(p2d[10],p2d[6]);
  ln(p2d[6],p2d[2]);
  ln(p2d[2],p2d[3]);
  ln(p2d[3],p2d[7]);
  ln(p2d[7],p2d[11]);
  ln(p2d[11],p2d[9]);
  ln(p2d[9],p2d[5]);
  ln(p2d[5],p2d[1]);
  ln(p2d[1],p2d[0]);
  ln(p2d[0],p2d[2]);
  ln(p2d[1],p2d[3]);
  ln(p2d[8],p2d[9]);
  ln(p2d[10],p2d[11]);

// ln(p2d[4],p2d[5]);
//  ln(p2d[5],p2d[7]);
//  ln(p2d[7],p2d[6]);
//  ln(p2d[6],p2d[4]);
}

/*************************************************************************/
// Funcion: palma(int color)
// Dibuja la palma
//
// color: color de la palma
/*************************************************************************/

void THandSim::palma(int color)
{
  setcolor(color);
  barra(bb[0],g[0]);

//  sistref(COrigin,R,g[0]);
}

/*************************************************************************/
// Funcion: pulgar(int color)
// Dibuja el dedo pulgar
//
// color: color del pulgar
/*************************************************************************/

void THandSim::pulgar(int color)
{
  setcolor(color);
  barra(bb[1],g[1]);
  barra(bb[2],g[2]);
  tip(tt[0],g[3]);

//  sistref(COrigin,R,g[1]);
//  sistref(COrigin,R,g[2]);
//  sistref(COrigin,R,g[3]);
}

/*************************************************************************/
// Funcion: indice(int color)
// Dibuja el dedo indice
//
// color: color del indice
/*************************************************************************/

void THandSim::indice(int color)
{
  setcolor(color);
  barra(bb[3],g[4]);
  barra(bb[4],g[5]);
  tip(tt[1],g[6]);

//  sistref(COrigin,R,g[4]);
//  sistref(COrigin,R,g[5]);
//  sistref(COrigin,R,g[6]);
}

/*************************************************************************/
// Funcion: medio(int color)
// Dibuja el dedo medio
//
// color: color del dedo medio
/*************************************************************************/

void THandSim::medio(int color)
{
  setcolor(color);
  barra(bb[5],g[7]);
  barra(bb[6],g[8]);
  tip(tt[2],g[9]);

//  sistref(COrigin,R,g[7]);
//  sistref(COrigin,R,g[8]);
//  sistref(COrigin,R,g[9]);
}

/*************************************************************************/
// Funcion: anular(int color)
// Dibuja el dedo anular
//
// color: color del anular
/*************************************************************************/

void THandSim::anular(int color)
{
  setcolor(color);
  barra(bb[7],g[10]);
  barra(bb[8],g[11]);
  tip(tt[3],g[12]);

//  sistref(COrigin,R,g[10]);
//  sistref(COrigin,R,g[11]);
//  sistref(COrigin,R,g[12]);
}

/*************************************************************************/
// Funcion: mano(Miembro flag)
// Dibuja la mano
//
// flag: bandera que identifica el miembro en movimiento
// <1> palma, <2> pulgar, <3> indice, <4> medio, <5> anular
/*************************************************************************/

void THandSim::mano(Miembro flag)
{
  int coldef,colesp,colpal,colind,colmed,colanu,colpul;

  coldef=CHand;
  colesp=CSelection;
  colpal=coldef;
  colind=coldef;
  colmed=coldef;
  colanu=coldef;
  colpul=coldef;

  switch (flag)
```

```
  {
    case mPalma:   colpal=colesp; break;
    case mPulgar:  colpul=colesp; break;
    case mIndice:  colind=colesp; break;
    case mMedio:   colmed=colesp; break;
    case mAnular:  colanu=colesp; break;
    default:  break;
  }

  mg(ang);       //Aqui se calculan todas las matrices

  palma(colpal);
  pulgar(colpul);
  indice(colind);
  medio(colmed);
  anular(colanu);
}

/*************************************************************************/
// Funcion: pt2d(int color,TPoint& point,int l)
// Crea una marca en el punto señalado
//
// color: color de la marca
// p: punto en la pantalla
// l: longitud de los brazos de la marca
/*************************************************************************/

void THandSim::pt2d(int color,TPoint& p,int l)
{
  setcolor(color);
  line((int)(p.x-l),(int)p.y,(int)(p.x+1+l),(int)p.y);
  line((int)p.x,(int)(p.y-l),(int)p.x,(int)(p.y+1+l));
}

/*************************************************************************/
// Funcion: pt3d(int color,const v4& p,int l)
// Dibuja un punto en el espacio
//
// color: color del punto dibujado
// p: coordenada en 3d del punto
/*************************************************************************/

void THandSim::pt3d(int color,const v4& p,int l)
{
  TPoint p2d;
  setcolor(color);
  p2d=vscreen(D,p,mview);
  pt2d(color,p2d,l);
}

/*************************************************************************/
// Funcion: rec_icon(int color,bool flag)
// Dibuja el icono de "CAL" (calibracion)
//
// color: color del icono
// flag: bandera de activacion
/*************************************************************************/

void THandSim::cal_icon(int color,bool flag)
{
  if(flag==T)
  {
    int x,y;
    x=XR-75;
    y=10;
    setcolor(color);
    setfillstyle(SOLID_FILL,color);
    fillellipse(x,y,3,3);
    setcolor(CFrame);
    outtextxy(x,y+10,"CAL");
  }
}

/*************************************************************************/
// Funcion: rec_icon(int color,bool flag)
//  Dibuja el icono de "REC" (guardar)
//
//  color: color del icono
// flag: bandera de activacion
/*************************************************************************/

void THandSim::rec_icon(int color,bool flag)
{
  if(flag==T)
  {
    int x,y;
    x=XR-45;
    y=10;
    setcolor(color);
    setfillstyle(SOLID_FILL,color);
    fillellipse(x,y,3,3);
    setcolor(CFrame);
    outtextxy(x,y+10,"REC");
  }
}

/*************************************************************************/
// Funcion: play_icon(int color,bool flag)
//  Dibuja el icono de "PLAY" (cargar)
//
// color: color del icono
// flag: bandera de activacion
/*************************************************************************/

void THandSim::play_icon(int color,bool flag)
{
  if(flag==T)
  {
    int x,y;
    x=XR-15;
    y=10;
    setcolor(color);
    setfillstyle(SOLID_FILL,color);
    fillellipse(x,y,3,3);
    setcolor(CFrame);
    outtextxy(x,y+10,"PLAY");
  }
}

/*************************************************************************/
// FUNCIONES DE COMUNICACION SERIAL
/*************************************************************************/

const int SET = 0;          //configuracion de puerto
const int SEND = 1;         //envia caracter por el puerto
const int READ = 2;         //recive un caracter en el puerto
const int STAT = 3;         //verifica el estado del puerto

const int COM1 = 0;         //puerto serie COM 1
const int COM2 = 1;         //puerto serie COM 2

const int DB_7 = 0x02;      //7 bits de datos
const int DB_8 = 0x03;      //8 bits de datos

const int SB_1 = 0x00;      //1 bit de paro
const int SB_2 = 0x04;      //2 bits de paro

const int NO_PAR = 0x00;    //sin paridad
const int ODD_PAR = 0x08;   //paridad non
const int EVEN_PAR = 0x18;  //paridad par

const int B_110 = 0x00;     //110 bauds
const int B_150 = 0x20;     //150 bauds
const int B_300 = 0x40;     //300 bauds
const int B_600 = 0x60;     //600 bauds
const int B_1200 = 0x80;    //1200 bauds
const int B_2400 = 0xa0;    //2400 bauds
const int B_4800 = 0xc0;    //4800 bauds
const int B_9600 = 0xe0;    //9600 bauds

const int DATA_READY = 0x100;  //hay un caracter en el puerto

/*************************************************************************/
// Funcion: init_rs(void)
// Inicializa el puerto serial
/*************************************************************************/

void THandSim::init_rs(void)
{
  bioscom(SET,DB_8|SB_1|NO_PAR|B_9600,COM2);
}

/*************************************************************************/
// Funcion: test_rs(void)
// Prueba si hay informacion disponible en el puerto serial
/*************************************************************************/

int THandSim::test_rs(void)
{
  return(bioscom(STAT,0,COM2)&DATA_READY);
}

/*************************************************************************/
// Funcion: leer(void)
// Lee un caracter del puerto serial
/*************************************************************************/

char THandSim::leer(void)
{
  return(bioscom(READ,0,COM2)&0x00ff);
}

/*************************************************************************/
// Funcion: escribir(char c)
// Escribe un caracter al puerto serial
//
```

```
// c: caracter que va a ser mandado por el puerto
/*************************************************************************/

void THandSim::escribir(char c)
{
  bioscom(SEND,c,COM2);
}

/*************************************************************************/
// Funcion: get13(void)
// Captura una secuencia de 13 bytes del puerto serial
/*************************************************************************/

v13 THandSim::get13(void)
{
  int i=0;
  v13 v;

  for(;13>i;)
  {
    if(test_rs())
    {
      v.v[i]=leer();
      i++;
    }
  }
  return v;
}

/*************************************************************************/
// FUNCIONES MISCELANEAS
/*************************************************************************/

/*************************************************************************/
// Funcion: interpol(double x1,double x2,double y1,double y2,double x)
// Realiza la interpolación lineal de puntos
//
// x1: Punto 1 sobre las abscisas
// x2: Punto 2 sobre las abscisas
// y1: Punto 1 sobre las ordenadas
// y2: Punto 2 sobre las ordenadas
// x: Punto actual
/*************************************************************************/

double THandSim::interpol(double x1,double x2,double y1,
                          double y2,double x)
{
  double y,k;
  k=(x2-x1);
  if(k==0)
    k=1;
  y=((y2-y1)/k)*(x-x1)+y1;
  return y;
}

/*************************************************************************/
// Funcion: minimo(v16 datos,v16 minimos)
// Obtiene el minimo de los valores que son leidos en el puerto
//
// datos: vector de datos actuales recibidos en el puerto
// minimos: vector de datos minimos recibidos en el puerto
/*************************************************************************/

v13 THandSim::minimo(v13 datos,v13 minimos)
{
  for(int i=0;13>i;i++)
    if(datos.v[i]<minimos.v[i])
      minimos.v[i]=datos.v[i];
  return minimos;
}

/*************************************************************************/
// Funcion: maximo(v13 datos,v13 maximos)
// Obtiene el maximo de los valores que son leidos en el puerto
//
// datos: vector de datos actuales recibidos en el puerto
// maximos: vector de datos maximos recibidos en el puerto
/*************************************************************************/

v13 THandSim::maximo(v13 datos,v13 maximos)
{
   for(int i=0;13>i;i++)
     if(datos.v[i]>maximos.v[i])
       maximos.v[i]=datos.v[i];
  return maximos;
}

/*************************************************************************/
// FUNCIONES DE MANEJO DE ARCHIVOS
/*************************************************************************/

/*************************************************************************/
// Funcion: salvar(v19 buf)
// Graba la secuencia de angulos en el archivo: "Serie.sec"
//
// buf: guarda el valor de todos los ángulos
/*************************************************************************/

void THandSim::salvar(v19 buf)
{
  FILE *outfile;

  if((outfile = fopen("Serie.sec","a+b"))==NULL)
  {
    printf("\nError en escritura");
  }

  fwrite(&buf,sizeof(v19),1,outfile);
  fclose(outfile);
}

/*************************************************************************/
// Funcion: cargar(double tiempo)
// Carga la secuencia almacenada en el archivo: "Serie.sec"
//
// tiempo: retraso en milisegundo entre imagenes
/*************************************************************************/

void THandSim::cargar(double tiempo)
{
  FILE *infile;
  v19 buf;

  if((infile = fopen("Serie.sec","rb"))==NULL)
  {
    printf("\nError en lectura!");
  }

  fseek(infile, SEEK_SET, 0);

  while(!feof(infile))
  {
    fread(&buf,sizeof(v19),1,infile);
    ang=buf;
    Paint();
    delay(tiempo);
  }

  fclose(infile);
}

/*************************************************************************/
// Funcion: lee_kb()
// Lee un caracter del teclado
/*************************************************************************/

union REGS in,out;

char lee_kb(void)
{
  in.h.ah=0x06;
  in.h.dl=0xFF;
  int86(0x21,&in,&out);
  return(out.h.al);
}

/*************************************************************************/
// Programa principal
/*************************************************************************/

main()
{
  THandSim *Hand;

  Hand = new THandSim();

  Hand->PrgRun();
}
```

BIBLIOGRAFÍA

[1] Ali, M. S., Kyriakopoulos, K. J. & Stephanou, H. E., *The Anthrobot-2 Dextrous Hand*, IEEE International Conference on Robotics and Automation, Vol. 3, 1993.

[2] Arnush, C., *Aprendiendo Borlamd C++ 5 en 21 días*, Prentice-Hall, México, 1996.

[3] Beer, F. P. & Johnston, E. R., *Mecánica vectorial para ingenieros: Dinámica*, 5ª Ed., McGraw-Hill, México, 1990.

[4] Craig, J. J., *Introduction to Robotics, Mechanics and Control*, 2nd Ed., Addison-Wesley, USA, 1989.

[5] Denver, A., *Serial Communication in Win32*, Microsoft Windows Developer Support, December, 1995 (http//www.microsoft.com).

[6] Foley, J. D., van Dam, A., Feiner, S. K. & Hughes, J. F., *Computer Graphics: Principles and Practice* , 2nd Ed., Addison-Wesley, USA, 1990.

[7] Fu, K. S., González, R. C. & Lee, C. S. G., *Robotics: Control, Sensing, Vision and Intelligence*, McGraw-Hill, USA, 1987.

[8] Hibbeler, R. C., *Mecánica para ingenieros: Dinámica*, 2ª Ed., CECSA, México, 1989.

[9] Hodgman, C. D. (Editor), *Mathematical Tables from HandBook of Chemistry and Physics*, 11th Ed., Chemical Rubber, USA, 1963.

[10] Koivo, A. J., *Fundamentals for Control of Robotic Manipulators*, John Wiley & Sons, USA, 1989.

[11] Koren, Y., *Robotics for Engineering*, McGraw-Hill, USA, 1985.

[12] Mabie, H. H. y Ocvirk, F. W., *Mecanismos y dinámica de maquinaria*, Limusa, México, 1990.

[13] Mason, M. T. & Salisbury, J. K., *Robot Hands and the Mechanics of Manipulation*, MIT Press, USA, 1985.

[14] Murray, R. M., Li, Z., & Sastry, S. S., *A Mathematical Introduction to Robotic Manipulation*, CRC Press, USA, 1994.

[15] Nof, S. Y. (Editor), *Handbook of Industrial Robotics*, John Wiley & Sons, USA, 1985.

[16] Rumsey, F. (Editor), *Midi Systems and Control*, Focal Press, UK, 1994.

[17] Schildt, H., *C Manual de referencia*, 2ª Ed., McGraw-Hill, México, 1993.

[18] Solar, E. y Speziale, L., *Apuntes de álgebra lineal*, 2ª Ed., Limusa, México, 1991.

[19] Venkataraman, S. T. & Iberall, T. (Editors), *Dextrous Robot Hands*, Springer-Verlag, USA, 1990.

[20] Vukobratovic, M. & Kircanski, N., *Real-Time Dynamics of Manipulation Robots (Scientific Fundamentals of Robotics, Vol. 4)*, Springer-Verlag, Berlin, 1985.

[21] Watt, A., *3D Computer Graphics*, 2nd Ed., Addison-Wesley, UK, 1989.