UNIVERSIDAD AUTÓNOMA QUERÉTARO
Facultad de Ingeniería

"SIMULADOR DE MANIPULADORES ROBÓTICOS DESACOPLADOS 6R."

T E S I S

Que como parte de los requisitos para obtener el grado de Maestro en Ciencias Línea Terminal Instrumentación y Control Automático

Presenta:
Erika Martínez Ramírez

Santiago de Querétaro, Qro., Junio del 2001

UNIVERSIDAD AUTÓNOMA QUERÉTARO
Facultad de Ingeniería

"SIMULADOR DE MANIPULADORES ROBÓTICOS DESACOPLADOS 6R."

T E S I S

Que como parte de los requisitos para obtener el grado de
Maestro en Ciencias
Línea Terminal Instrumentación y Control Automático

Presenta:
Erika Martínez Ramírez

Dirigido por:
Dr. Carlos Santiago López Cajún

SINODALES

Dr. Carlos Santiago López Cajún Presidente	Firma
MC. Victor Manuel Hernández Guzmán Secretario	Firma
Dr. Eduardo Castillo Castañeda Vocal	Firma
Dr. Marco Tulio Mata Jiménez Suplente	Firma
Dr. José Emilio Vargas Soto Suplente	Firma

Ing. Jorge Martínez Carrillo
Director de la Facultad

Dr. Sergio Quesada Aldana
Director de Investigación y Posgrado

Centro Universitario, Querétaro, Qro.
Junio del 2001.

RESUMEN

En este trabajo se presenta un programa que simula los movimientos de un manipulador desacoplado de sexto grado de libertad del tipo 6R. Para la realización de este simulador se estableció el modelo geométrico del manipulador, a partid del cual se formularon las ecuaciones para obtener la cinemática directa e inversa de posición y velocidad. También se incluyó un módulo para la planeación de trayectorias. Un manipulador desacoplado es aquel en el que los últimos tres ejes se intersecan en un mismo punto, permitiendo separar el problema de la cinemática inversa en cinemática inversa de orientación y cinemática inversa de posición. Este programa presenta una interfase para que el usuario pueda interactuar con la simulación sin necesidad de hacer modificaciones a los programas fuentes. El usuario, a través de la interfase, puede rotar cada uno de los eslabones para conocer la posición del órgano terminal (cinemática directa). De igual modo, se puede indicar una trayectoria por medio de una ecuación paramétrica o especificando puntos en el espacio y el programa realiza los cálculos necesarios para rotar cada uno de los eslabones y posicionarlo en la configuración deseada (cinemática inversa). Es posible indicar la velocidad del órgano terminal y obtener la velocidad de las articulaciones, o por el contrario, indicar la velocidad de las articulaciones y obtener la velocidad del órgano terminal. Además, éste último puede tener una orientación fija en cada punto de la trayectoria o pueda tener una orientación variable que dependerá de los marcos de dirección que se especifique en la trayectoria a realizar. Esto es útil al considerar tareas de soldadura, transportación de piezas, manejo de reactivos, etc. Para evitar colisiones, se realiza el estudio de los espacios de trabajo del manipulador. Con esta animación se puede previsualizar las acciones del robot en el seguimiento de trayectorias antes de su ejecución real y con ello prevenir posibles colisiones. Además, el programa permite generar los datos de código para la ejecución real del robot. La simulación se realiza en Max Script del software 3D Studio Max 2.5. Esta simulación se evaluó utilizando el robot alemán marca CLOOS modelo ROMAT 56 que se encuentra en el Laboratorio de Mecatrónica de la Universidad Autónoma de Querétaro

(Palabras clave: robótica, simulación, cinemática inversa, manipulador desacoplado, planeación de trayectorias)

SUMMARY

In this work a program that simulates the movements of a decoupled sixth-degree-of-freedom manipulator of the 6R type. The geometric model of the manipulator was established, then, the equations for direct and inverse kinematics of both position and velocity were derived. The trajectory planning and the workspace evaluation of a six joints decoupled manipulator were also. A decoupled manipulator is one in which the last three axes intersect at the same point. This allows us to separate the problem of the inverse kinematics in inverse position kinematics and inverse orientation kinematics. This program displays an interface so that the user can interact with the simulation without modifying source programs. The user, through the interface, can rotate each of the links to determine the end-effector position (direct kinematics). A trajectory can be specified by a parametric equation or via points in the workspace. The calculations necessary to rotate each link to move the robot in the desired configuration are made by the program (inverse kinematics). The end-effector velocity can be indicated and the joints velocity obtained, or on the contrary, to indicate the joints velocities and obtain the end-effector velocity. In addition, the end-effector might have a fixed direction at each point of the trajectory or a variable direction that will depend on the trayectory direction frames. This is useful for welding tasks, piece transportation, materials handling, etc. For collision avoidance, the study of the manipulator's workspace is also done. With this animation it is possible to preview the robot movements in the path tracking before its real execution in order to prevent possible collisions. In addition, the program allows generation of the necessary data for the real execution of the robot. The simulation was done in Max Script of 3d Studio Max 2.5. This simulation was evaluated using the German robot CLOOS, model ROMAT 56, located in the Laboratory of Mecatronics of the University of Queretaro.

(Key words: robotics, simulation, inverse kinematics, decoupled manipulator, path planning)

A mi madre **Margarita Ramírez Villagómez**

A mi familia por todo su **cariño**, **comprensión** y **apoyo** a lo largo de mi vida.

AGRADECIMIENTOS

Al Dr. Gilberto Herrera Ruiz por su apoyo incondicional y su valiosa aportación a mi desarrollo profesional.

Al Dr. Carlos Santiago López Cajún por el gran apoyo en la realización de esta tesis.

Al MC. Victor Manuel Hernández Guzmán y al Dr. Arturo Zavala Río por su paciencia, dedicación y sugerencias.

A los sinodales Dr. Eduardo Castillo Castañeda, Dr. Marco Tulio Mata Jiménez y Dr. José Emilio Vargas Soto por sus comentarios y constructivas sugerencias.

A la Coordinación de la Maestría en Ciencias línea Instrumentación y Control Automático dirigida por el Dr. Gilberto Herrera Ruiz por todas las facilidades otorgadas para la realización de esta tesis.

Al Ing. Héctor Guízar Castañeda por su apoyo incondicional.

Un agradecimiento muy especial al Ing. Rufino García Mendoza por su tiempo, paciencia, comprensión y apoyo incondicional.

Índice General

Índice de Figuras

Introducción

En la actualidad hablar sobre el tema de manipuladores robóticos es de lo más común. La gran mayoría de las industrias e instituciones educativas emplean en sus áreas de trabajo algún tipo de manipulador. Estos manipuladores son de gran utilidad en una diversidad de operaciones, que van desde tareas educativas y de aprendizaje hasta las tareas industriales como la carga y descarga de materiales, el ensamble de componentes y la soldadura de punto, por mencionar algunas. El uso de estos manipuladores en la industria se manifiesta en un aumento de productividad y una disminución de costos operativos. A causa de esto, son cada vez más industrias que están incluyendo en sus líneas de producción algún tipo de robot que pueda ser operado por medio de una computadora. El robot recibe una serie de instrucciones para ser ejecutadas en tiempo real a través de una computadora o un control remoto. El proceso de especificar y evaluar estas instrucciones requiere, en algunos casos, detener la operación en el que se encuentra trabajando el manipulador provocando a la empresa tiempos muertos en la línea de operación y elevando los costos. Por esta razón y por la necesidad de evaluar las posibles colisiones, surge el interés de contar con un programa que reciba estas instrucciones y determine, gráficamente, la posición en cada instante de los eslabones del robot. Esta visualización se realiza previo a una ejecución real y permite obtener un mejor desempeño del robot.

Este trabajo presenta un programa para facilitar el proceso de introducir y evaluar las instrucciones que se le indican al manipulador utilizando la simulación en tres dimensiones (3D). En el programa, el usuario puede verificar la tarea que realizará el robot y evaluar que el trayecto se realice en forma suave, es decir, sin movimientos abruptos. Además, el programa es útil para visualizar que los movimientos del robot estén libres de colisiones, ya sea con alguno de sus componentes o con su entorno. Además, genera las instrucciones de los movimientos de las articulaciones una vez que han sido evaluadas, por lo tanto, el tiempo de programación del robot real disminuye considerablemente.

Para la programación de este simulador se utilizaron como referencia las publicaciones de (Ángeles, 1997) y (Spong y Vidyasagar, 1989), entre otros. Este programa se realizó para generar animaciones de manipuladores desacoplados con seis articulaciones (6R); esto es, los ejes de rotación de las últimas tres articulaciones se intersecan en un mismo punto. El programa permite cambiar la configuración del robot y los parámetros Denavit-Hartenberg, siempre y cuando cumpla con la característica de desacoplamiento y sea de seis articulaciones.

Los conceptos que se emplean para la programación de este simulador son el modelo geométrico del manipulador, del cual se obtiene su cinemática directa e inversa para la posición, así como la cinemática de velocidad y la planeación de trayectorias. El programa de simulación permite al usuario, a través de la interfase, girar cada uno de los eslabones o articulaciones para conocer la posición y orientación del órgano terminal (*cinemática directa*). En el caso contrario,

a partir de una posición y orientación dadas del órgano terminal es posible obtener los ángulos de rotación de cada una de las articulaciones (*cinemática inversa*). La cinemática inversa de un manipulador desacoplado 6R produce, en general, ocho posibles soluciones para que el órgano terminal del robot llegue a un determinado punto cartesiano. La elección de alguna de éstas depende de la evaluación del espacio de trabajo, el cual se puede visualizar con la ayuda del simulador. La velocidad con la que se moverá cada eslabón del robot o la velocidad que tendrá el órgano terminal se especifica en la *cinemática de velocidad.* El módulo de la planeación de trayectorias se compone de trayectorias continuas y trayectorias punto a punto. El cálculo de las *trayectorias continuas* se realiza indicando una ecuación paramétrica para la posición y una determinada orientación para el órgano terminal. La planeación de trayectoria punto a punto se realiza mediante *interpolación polinomial o cicloidal* si se habla de dos puntos, o *funciones splines* si son más puntos. El programa presenta las diferentes configuraciones del manipulador para el seguimiento de trayectorias y deja a elección del usuario aquella que se implementará en el robot. El órgano terminal puede tener una orientación fija en el seguimiento de la trayectoria o puede tener una orientación variable que dependerá de los marcos de dirección que se especifiquen en cada punto de la trayectoria. Esto es de gran utilidad si se consideran tareas como la de soldadura que depende de la dirección de la herramienta para soldar. Por último, este programa genera un archivo de texto con los ángulos de rotación de los eslabones en cada instante de la trayectoria para su ejecución real.

El programa de simulación se realizó en Max Script del *software* 3D Studio Max ver.2.5. Las gráficas se programaron en Matlab ver. 5.2. El simulador permite utilizar los parámetros y la configuración de cualquier robot desacoplado de 6R. El simulador se evaluó utilizando el robot alemán CLOOS modelo Romat 56 que se encuentra en el Laboratorio de Mecatrónica de la Universidad Autónoma de Querétaro.

Capítulo 1

Estado del conocimiento

La robótica es un campo relativamente nuevo de la tecnología moderna que traspasa los límites de la ingeniería tradicional. La palabra robot tiene diferentes significados para cada persona por lo cual se considera la definición empleada por el Instituto de Robótica de América que es más específica: "un robot es un manipulador multifuncional reprogramable diseñado para mover materiales, piezas, herramientas, o dispositivos especializados, por medio de variables programadas para la realización de varias tareas ". Otra definición incluye los manipuladores mecánicos, las máquinas de control numérico, las máquinas andantes, y los humanoides de la ciencia ficción. Aunque la mayoría de la gente percibe a los robots antropomórficamente, los robots industriales de hoy son solamente algo humanoide en aspecto. De hecho, la mayoría de los robots industriales son manipuladores mecánicos.

Un manipulador puede programarse para realizar tareas en forma repetitiva en el área educativa o industrial. Esta programación se especifica directamente en la computadora que controla al manipulador y de esta manera el robot realiza los movimientos de cada eslabón para posicionarse en una configuración deseada. Además, se puede programar una secuencia de posiciones para que el robot realice una tarea específica. Algunas de las tareas pueden ser la carga y descarga de materiales, el ensamble de componentes y la soldadura de punto. Una vez que se programan las instrucciones de acción al manipulador, éste las ejecuta en tiempo real. La necesidad de evaluar las configuraciones del manipulador en el seguimiento de una trayectoria despierta el interés de crear un programa de simulación. Más aún, la visualización gráfica de los movimientos del robot previamente a su ejecución real permite prever posibles colisiones.

Una manera de evaluar el seguimiento de trayectorias es a través de la obtención de la cinemática directa e inversa del manipulador, la cinemática de velocidad y la planeación de trayectorias. El *software* en el que se realiza este programa permite definir instrucciones de código para programar dichos cálculos y así generar la simulación del manipulador. Un modelador de objetos en 3D admite la creación de ambientes de manufactura, con base en la descripción de objetos geométricos simples y su agrupación en objetos complejos.

1.1 Desarrollo histórico

Aunque la palabra robot entra en el vocabulario a principios de siglo, el desarrollo real de los manipuladores robóticos ocurrió a partir de los años cuarentas. La aparición de los robots industriales, inicialmente, era debido a la necesidad de manejar materiales peligrosos, la exploración

del espacio y la de alcanzar la automatización flexible. Cerca de los años cuarentas se comenzó el desarrollo de manipuladores maestro-esclavo para manejar material radiactivo. El manipulador maestro es dirigido por un operador, mientras que el manipulador esclavo reproduce el movimiento generado por el maestro en una estación remota. A finales de los años cincuentas se desarrolló el manipulador mecánico programable, que más adelante se perfeccionó. La característica importante de éste es la incorporación de un controlador - o microprocesador - con sensores de retroalimentación para mejorar la capacidad multifuncional programable, el cual hizo posible el control por computadora del manipulador.

La robótica es un campo interdisciplinario que va desde el diseño de componentes eléctricos y mecánicos hasta la creación de tecnología de sensores, paquetes computacionales e inteligencia artificial. La incorporación del microcontrolador involucra un desarrollo tecnológico en la programación de tareas repetitivas y tareas multidisciplinarias que engloban diferentes ramas de la ciencia y la ingeniería, tales como ciencias computacionales, electrónica y control.

Mientras que los manipuladores se encuentran todavía en desarrollo, encontramos ejemplos de sistemas mecano-robóticos en operación conocidos como manipuladores industriales de seis ejes, simuladores de vuelo de seis grados de libertad, máquinas andantes y manos mecánicas (Ángeles, 1997)

1.1.1 Antecedentes

El área de modelación y simulación de manipuladores es relativamente amplio, en donde podemos apreciar los proyectos realizados por diversas instituciones o compañías.

El grupo de Ingeniería en Sistemas de Control del Departamento de Ingeniería Eléctrica en la Universidad de Hagen, Alemania, realiza el proyecto de "Modelación y Simulación de Sistemas Robóticos". Este proyecto está enfocado al análisis de la modelación matemática, cinemática y dinámica. Se aplican todas estas técnicas para desarrollar ambientes de animación en 3D en el Lenguaje de Modelación de Realidad Virtual (VRML) para un manipulador industrial (PUMA 562) y la herramienta de simulación DYNAST para la simulación de sistemas dinámicos.(Bischoff, 2000)

En la Universidad Carlos III de Madrid, en el área de ingeniería de sistemas y automática dentro de las líneas de trabajo se encuentra "Robótica industrial". En la investigación sobresale el desarrollo de un simulador dinámico de robots para el ensamblado de estructuras en el espacio. (Ingeniería de sistemas y automática, 2000)

En el Instituto Tecnológico de Massachusetts (MIT) en Estados Unidos se realiza un proyecto que establece un ambiente gráfico de simulación de robots interactivo y con las propiedades físicas del diseño. Este programa se realiza en Java2 y Java 3D. La página web presenta demos de gráficas de robots de 2 y 3 grados de libertad.(Cheng, 2000)

En la Universidad del Estado de Iowa, se ha desarrollado un paquete para la simulación robótica y programación fuera de línea como se muestra en la figura 1. Este paquete se desarrolló para utilizarse en estaciones de trabajo Silicon Graphics, el cual tiene la capacidad de utilizar muchos tipos de dispositivos de Realidad Virtual (VR) comerciales. Este *software* incluye una librería para manipuladores industriales de cuatro, cinco y seis grados de libertad, con la cinemática inversa y traslación que convierte los comandos de simulación gráfica en el lenguaje del dispositivo de control.(Vanderploeg, 2000)

Figura1. Simulación robótica y programación fuera de línea. Ejemplo de simuladores comerciales de la Universidad del Estado de Iowa.

El Simulador de soldadura LT-3200 es una unidad de alta tecnología controlada por computadora. Permite al operador simular y experimentar la mayoría de las condiciones que se presentan en el proceso actual de soldadura (Simulador de soldadura, 2000).

En el Instituto Tecnológico y de Estudio Superiores de Monterrey campus Monterrey, el grupo de robótica inteligente, trabaja en el proyecto REDII cuyo objetivo es la generación de metodologías y herramientas para la creación de ambientes virtuales de manufactura y la simulación interactiva de tareas. Este grupo, que inició en 1990, realiza investigaciones en robots móviles, ambientes virtuales y manipulación remota de robots. Además de la planeación de trayectorias, movimientos, secuencias de ensamblado y la ejecución de tareas de captores y actuadores.

Las celdas de manufactura y cualquier dispositivo automatizado, se describen y simulan dentro de una computadora, para reproducir sus condiciones de programación y de operación. Los métodos utilizados se ilustran mediante animación gráfica, que consiste en el despliegue de los objetos en ambiente 3D, incluyendo el movimiento de luces y cámaras. El proyecto requiere interactuar entre diversos ambientes de desarrollo, tales como Java 3D y VRML, puesto que cada uno de ellos enfatiza algún aspecto importante para el sistema. La caracterísitica importante de Java 3D es la facilidad y generalidad de creación, mientras que VRML enfatiza la facilidad de navegación e interacción (Gordillo, 2000).

En la Facultad de Ingeniería de la Universidad Autónoma de Querétaro se realiza el "Estudio y simulación en 3D de la cinemática de un robot manipulador de 6 grados de libertad" aplicado al robot Mitsubishi tipo MoveMaster EX de 5 grados de libertad (GDL) y en el órgano terminal del manipulador se le agrega otro eslabón para tener los 6 GDL. El programa para la simulación se desarrolló en el lenguaje Turbo C++, donde la presentación gráfica de cada eslabón está señalada por sus vértices (Arteaga, R. 2000).

El sistema de visualización robótica (*Robotic Visualization System* RVS) es un *software* que se desarrolló en la Universidad McGill en lenguaje C y funciona en estaciones de trabajo Silicon-Graphics. Este *software* es un simulador de manipuladores robóticos (Ángeles, 2000).

Existen también paquetes diseñados para trabajar gráficos en 3D como el *software* VRML. Éste es una herramienta poderosa de lenguaje texto para crear "realidad virtual" en internet. El VRML incluye objetos 3D, fuente de iluminación y animación para generar gráficos interactivos de fácil acceso vía internet (Telerobotics, 2000). El *software* SoftImage 3D Extreme FX es un programa poderoso de animación que opera en estaciones de trabajo Silicon-Graphics y entre sus multiples aplicaciones se utiliza para la creación de caricaturas y en los spots de televisión (3D World. 2000). El *software* 3D Studio Max permite crear simulaciones en tres dimensiones, realizar animaciones a través de instrucciones de código, trabajar en ambiente Windows y puede instalarse en una computadora pentium.

El sotfware que se utiliza para el desarrollo de este programa de simulación es el 3D Studio Max ver. 2.5.

1.2 Los manipuladores seriales

Un manipulador, en general, es cualquier sistema mecánico que ayuda al hombre a realizar una tarea manipulable (como mover algo con la mano). Aunque los manipuladores han existido desde que el hombre creó la primera herramienta, no es sino hasta finales de la segunda guerra mundial que se desarrollaron los primeros manipuladores capaces de imitar el movimiento del brazo humano. Esto contribuyó al desarrollo de los primeros manipuladores de seis grados de libertad. Un manipulador robótico se distingue de los anteriores por la capacidad de ser programados y realizar tareas en forma repetitiva. La capacidad de los robots industriales se ha aprovechado tanto en la programación del manipulador con *software*, como en la realización de tareas de acuerdo a una planeación de trayectorias. Los manipuladores seriales se asocian a la cadena cinemática de tipo *simple*, es decir, cada uno de los eslabones está acoplado a lo más con otros dos eslabones. Una cadena cinemática simple puede ser cerrada o abierta. Una cadena es cerrada si cada uno de sus eslabones está acoplado a otros dos eslabones. Una cadena es abierta si contiene exactamente dos eslabones acoplados a cada eslabón, mientras que el último esta acoplado con un sólo eslabón.(Ángeles, 1997)

Para el seguimiento de una trayectoria es necesario conocer los ángulos de rotación que sitúan cada eslabón del manipulador para que el órgano terminal se encuentre en la posición deseada. Una manera de obtener estas variables de las articulaciones es por medio de la cinemática inversa.(Ángeles, 1997) y (Spong y Vidyasagar, 1989)

El problema general de la cinemática inversa para manipuladores con seis articulaciones es complicado; sin embargo, cuando se intersecan los últimos tres ejes de rotación en un punto es posible desacoplar la cinemática inversa en dos problemas simples, cinemática inversa de posición y cinemática inversa de orientación. El punto de intersección de los tres ejes de rotación recibe el nombre de *centro de la muñeca C*. Visto de otra forma, para un manipulador de seis grados de libertad con una muñeca esférica, el problema de la cinemática inversa puede separarse, primero resolviendo el problema de posición del punto de intersección de los ejes de la muñeca, llamado centro de la muñeca (wrist center), y después, resolver el problema de orientación del órgano terminal del manipulador. A esta separación del problema se le llama desacoplamiento

cinemático.(Spong y Vidyasagar, 1989)

1.3 Módulos del simulador

En la Universidad Autónoma de Querétaro, en el laboratorio del Posgrado de Instrumentación y Control Automático de la Facultad de Ingeniería, se adquirió recientemente un robot alemán CLOOS tipo Romat 56. Este robot es de seis grados de libertad, similar al PUMA, sirvió para evaluar este programa de simulación.

El presente trabajo describe la cinemática directa e inversa, cinemática de velocidad, planeación de trayectorias y la simulación en tres dimensiones para manipuladores robóticos desacoplados de 6R. Además, genera el código programable para su ejecución real. Este *software* se probará con el robot CLOOS y con un robot de configuración arbitraria. El programa será de utilidad para probar los algoritmos de control en la simulación antes de aplicarlos en una tarea específica en el robot real.

Inicialmente se obtienen los parámetros del manipulador considerando la nomenclatura de Denavit-Hartenberg (D-H). Se toman las medidas reales del manipulador y se va creando cada uno de los eslabones en 3D Studio Max, a través de instrucciones de código programable, para visualizar al robot en tres dimenciones. Los cálculos que se obtuvieron de los modelos cinemáticos directo e inverso se programan en MaxScript. Al obtener la posición de las articulaciones para alcanzar una posición deseada es posible planear una secuencia de puntos o trayectoria que seguirá el órgano terminal del manipulador. Una vez que se evalúa gráficamente que las posiciones del robot se encuentran libres de colisiones se convertirán los ángulos de rotación en código programable para su ejecución real.

Este trabajo consta de varios módulos, los cuales se exhiben en la figura 2:

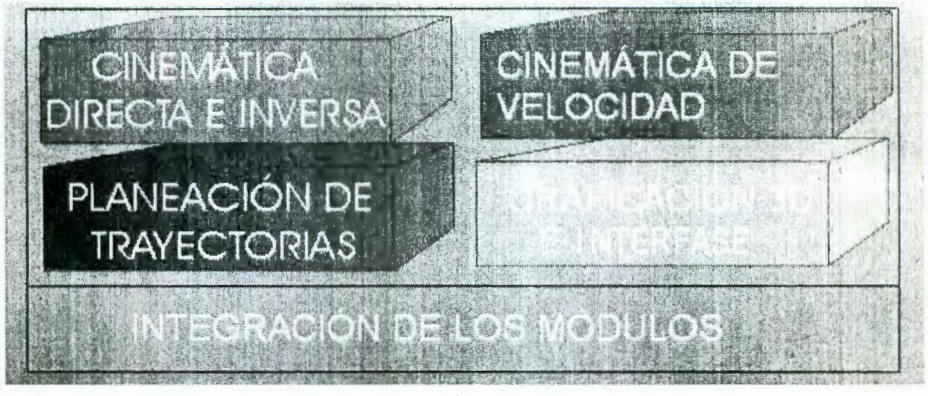

Figura 2. Módulos del simulador.

1.3.1 Cinemática directa e inversa

En la cinemática directa se obtiene el conjunto de ecuaciones que permiten conocer la posición y orientación del órgano terminal del robot, medidas en el espacio cartesiano, si se conocen los desplazamientos angulares en cada articulación del robot. La herramienta fundamental utilizada en esta parte es la nomenclatura de Denavit-Hartenberg (Spong y Vidyasagar, 1989). Por el

contrario, en la cinemática inversa se conoce la posición y orientación del órgano terminal del robot, a partir de las cuales se desea identificar las variables de articulación o desplazamientos angulares para cada eslabón del robot.

Cinemática directa. Dada las variables de articulación se determina la posición y orientación del órgano terminal. La figura 3 muestra un ejemplo de cinemática directa con un mecanismo de dos eslabones, donde se asignan valores a los ángulos de rotación α y β posicionando al órgano terminal en un cierto punto P en el espacio cartesiano.

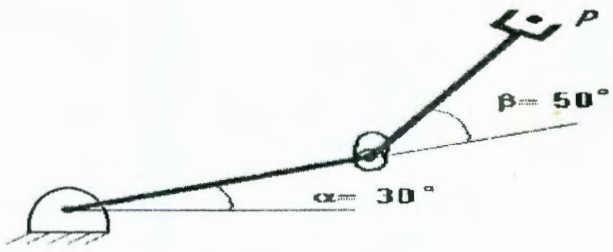

Figura 3. Representación de la cinemática directa.

Cinemática inversa. Dada la posición y orientación del órgano terminal encontrar los valores de las variables de articulación. La figura 4 muestra la cinemática inversa con un mecanismo de dos eslabones, donde se especifica un punto P para posicionar el órgano terminal y se calcula los ángulos de rotación α y β de cada articulación. Como se aprecia en esta figura, se tiene dos posibles soluciones para que el órgano terminal alcance el punto P.

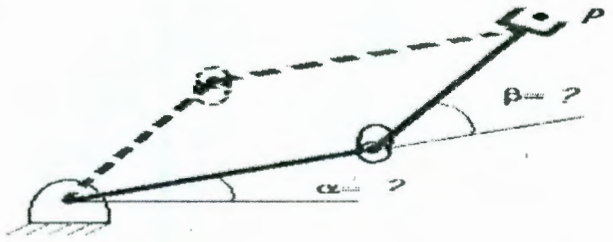

Figura 4. Representación de la cinemática inversa.

Una vez que se ha obtenido los cálculos de la cinemática se programan en MaxScript.

1.3.2 Cinemática de velocidad

La obtención de la cinemática de velocidad se realiza en base a referencia (Ángeles, 1997). La velocidad se establece para el órgano terminal y se calcula la velocidad de cada una de las articulaciones o por el contrario, se especifica la velocidad de las articulaciones y se obtiene la velocidad del órgano terminal.

1.3.3 Planeación de trayectorias

Los movimientos de un sistema mecano-robótico deben ser, como regla, lo más suave posible; es decir, sin cambios abruptos en la posición, velocidad y aceleración. Los movimientos suaves se

pueden planear con técnicas simples, sin embargo, esto no garantiza que esté excento de algún movimiento abrupto. La planeación del movimiento del robot en un ambiente no-estructurado va más allá de este estudio, involucrando áreas de reconocimiento de patrones e inteligencia artificial. Por esta razón, la planeación del movimiento del robot se dedicará solamente a ambientes estructurados. La planeación de trayectorias se separa en trayectorias continuas y en trayectorias punto a punto. En las trayectorias continuas se utilizó una ecuación paramétrica. Las trayectorias punto a punto se realizan de diferentes métodos. En este trabajo se considera la *Interpolación 4-5-6-7* para dos puntos de trayectoria porque minimiza los movimientos abruptos y las *Splines cúbicas* porque permiten especificar varios puntos de trayectorias.

1.3.4 Simulación e interfase

En este proyecto se esta utilizando el *software* 3D Studio Max ver. 2.5 (Elliott, Miller et Al., 1998) para construir el simulador porque facilita la creación de imágenes y la programación de animaciones. Para crear una figura se debe especificar su forma geométrica y sus dimensiones.

Cuando se desee que el órgano terminal del robot siga una trayectoria (coordenadas cartesianas) se utilizará el modelo cinemático inverso para determinar la trayectoria correspondiente a ser seguida por cada articulación (ángulos de giro de cada eje). Con esto se podrá realizar una animación del movimiento del robot permitiendo determinar si dicha trayectoria produce el movimiento deseado o si es segura en el sentido de que no haya colisiones del robot con el entorno que lo rodea. Es importante contar con este simulador para la evaluación visual del espacio de trabajo que permitirá detectar colisiones del robot con el entorno o con sí mismo, o si los ejes de giro sobrepasan el intevalo permitido mecánicamente por el robot. Una vez verificado esto se generará el código necesario para ordenar al robot que realice dichos movimientos en tiempo real, o bien, para almacenar dicho movimiento en un archivo de video de modo que pueda ser analizado posteriormente.

La simulación se presenta en forma de persianas. Cada una de éstas realizan una función en específico. Algunas de estas funciones son: cinemática directa, cinemática inversa, trayectorias continuas, trayectorias punto a punto, crear robot arbitrario, generar archivo de texto con las instrucciones para el robot y cambiar los parámetros de Denavit-Hartenberg.

Capítulo 2

Modelos y métodos utilizados

2.1 Modelo cinemático

En esta sección se presentan los modelos de la cinemática directa e inversa para manipuladores desacoplados de seis articulaciones.

2.1.1 Cinemática directa

La cinemática directa determina la posición y orientación del órgano terminal del robot en base a los ángulos de rotación de las articulaciones. Se utiliza la notación Denavit-Hartenberg (D-H) para describir la arquitectura del manipulador. Una vez que se tienen los parámetros D-H, obtenemos las matrices homogéneas que representan la rotación de cada uno de los eslabones, y de esto, obtenemos la matriz de transformación que presenta la posición y orientación del órgano terminal (end-effector EE). Para el caso de articulaciones, las variables son angulares y para acoplamientos prismáticos las variables son desplazamientos como se expone en la figura 5.

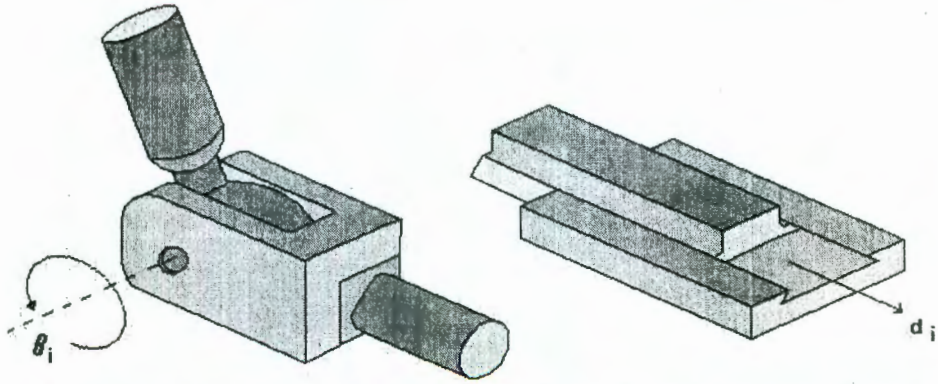

Figura 5. Presentación de un par o acoplamiento rotativo (R) y un par o acoplamiento prismático (P)

Para obtener el modelo esquemático generalizado del manipulador se emplea la notación Denavit-Hartenberg (Ángeles, 1997) y (Spong y Vidyasagar, 1989), cuyos pasos pueden resumirse como:

1. Localizar los ejes Z de cada articulación.
2. Establecer los ejes X_1 y Y_1 del marco base.
3. Localizar el origen O_i donde la normal común a Z_i y Z_{i-1} intersecte a Z_i.
4. Establecer X_i como la perpendicular común a Z_{i-1}y Z_i, en dirección del primero al último.
5. Establecer Y_i siguiendo la regla de la mano derecha.
6. Establecer el marco del EE.
7. Crear la tabla de parámetros de los eslabones a_i, b_i, α_i y θ_i donde:

a_i = se define como la distancia entre Z_i y Z_{i+1}, la cual es positiva.
b_i = se denota como la coordenada Z_i de la intersección O_i' de Z_i con X_{i+1}.
α_i = se define como el ángulo entre Z_i y Z_{i+1} y se mide sobre la dirección positiva de X_{i-1}.
θ_i = se define como el ángulo entre X_iy X_{i+1} y se mide sobre la dirección positiva de Z_i.

El origen del último marco está situado en el EE llamado *punto de operación, P.* Éste se utiliza para definir la posición y orientación en la realización de una tarea específica. La transformación de un marco de referencia a otro se puede representar por matrices de rotación y traslación. Después de que se tienen los parámetros D-H, obtenemos la transformación homogénea A_i, donde i corresponde a cada uno de los eslabones.

La matriz de transformación homogénea T_i representa una rotación α grados o radianes sobre el eje X, seguido por una traslación de a unidades a lo largo del eje X. Después por una traslación de b unidades a lo largo del eje Z y por último, una rotación de θ grados o radianes sobre el eje Z, es decir,

$$T_i = Rot_{z,\theta_i} Trans_{z,b_i} Trans_{x,a_i} Rot_{x,\alpha_i}$$

$$= \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & b_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & \sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & b_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

esta matriz de transformación, de igual manera, se puede representar como,

$$= \begin{bmatrix} \mathbf{Q}_i & \mathbf{a}_i \\ 0 & 1 \end{bmatrix}$$

donde (Ángeles, 1997),

$$\mathbf{Q}_i \equiv \begin{bmatrix} \cos\theta_i & -\lambda_i\sin\theta_i & \mu_i\sin\theta_i \\ \sin\theta_i & \lambda_i\cos\theta_i & -\mu_i\cos\theta_i \\ 0 & \mu_i & \lambda_i \end{bmatrix} \qquad [\mathbf{a}_i]_i = \begin{bmatrix} a_i\cos\theta_i \\ a_i\sin\theta_i \\ b_i \end{bmatrix}$$

y

$$\lambda_i \equiv \cos\alpha_i, \qquad \mu_i \equiv \sin\alpha_i$$

conociendo esto, se especifican las matrices de transformación para cada articulación desde T_1, T_2 ... T_n, donde los subíndices $1, 2, ..., n$ dependen del número de articulaciones que tenga el manipulador. Una vez obtenidas las matrices de transformación para cada articulación, la multiplicación de éstas conducirá a una matriz de tranformación final,

$$\mathbf{T}_1\mathbf{T}_2\mathbf{T}_3\mathbf{T}_4\mathbf{T}_5\mathbf{T}_6 = \mathbf{T}$$

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ r_{41} & r_{42} & r_{43} & r_{44} \end{bmatrix}$$

donde la matriz $\mathbf{Q}_1^n$ de tres por tres representa la matriz de orientación del punto de operación, mientras que el último vector de tres por uno $\mathbf{a}_1^n$, representa el vector de posición del punto de operación en T,

$$\mathbf{a}_1^n = \begin{bmatrix} r_{14} \\ r_{24} \\ r_{34} \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

y

$$\mathbf{Q}_1^n = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

De esta matriz de transformación $\mathbf{Q}_1^n$ la igualamos a la matriz de *roll-yaw-pitch* para obtener la orientación del órgano terminal ,

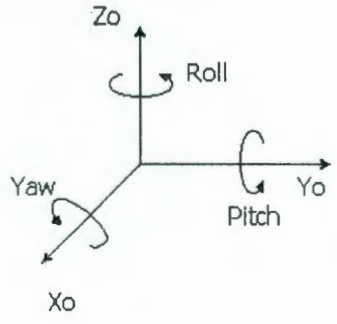

Figura 6. Ángulos '*roll*', '*yaw* 'y '*pitch*'.

Una matriz de rotación R puede describirse como un producto suscesivo de rotaciones sobre el eje coordenado principal x_0, y_0 y z_0. Estas rotaciones definen los ángulos *roll-yaw -pitch*, los cuales también se denotan por (ϕ, θ, ψ) como se muestra en la figura 6.

$$
\begin{aligned}
R &= R_{z,\phi} R_{y,\theta} R_{x,\psi} \\
&= \begin{bmatrix} c_\phi & -s_\phi & 1 \\ s_\phi & c_\phi & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\psi & -s_\psi \\ 0 & s_\psi & c_\psi \end{bmatrix} \\
&= \begin{bmatrix} c_\phi c_\theta & -s_\phi c_\psi + c_\phi s_\theta s_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ s_\phi c_\theta & c_\phi c_\psi + s_\phi s_\theta s_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix}
\end{aligned}
$$

Igualando la matriz roll-yaw-pitch con la matriz Q_1^n se tiene

$$
\mathbf{Q}_1^n = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} c_\phi c_\theta & -s_\phi c_\psi + c_\phi s_\theta s_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ s_\phi c_\theta & c_\phi c_\psi + s_\phi s_\theta s_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix}
$$

1) Calcular el valor para θ

$$
r_{31} = -s\theta => s\theta = -r_{31}
$$

$$
c\theta = \pm\sqrt{1 - r_{31}^2}
$$

$$
\theta = arc\tan\left(\pm\sqrt{1 - r_{31}^2}, -r_{31}\right)
$$

2) Obtener el valor de ϕ

$$
\begin{aligned}
r_{11} &= c\phi c\theta \\
r_{12} &= s\phi c\theta \\
\frac{r_{21}}{r_{11}} &= \tan\phi
\end{aligned}
$$

$$
\begin{aligned}
\phi = arc\tan(r_{11}, r_{21}) \quad & \text{si} \quad c\theta > 0 \\
\phi = arc\tan(-r_{11}, -r_{21}) \quad & \text{si} \quad c\theta < 0
\end{aligned}
$$

3) Calcular el valor de ψ

$$\begin{aligned} r_{32} &= c\theta s\psi \\ r_{33} &= c\theta c\psi \\ \frac{r_{32}}{r_{33}} &= \tan\psi \end{aligned}$$

$$\begin{aligned} \psi = arc\tan(r_{33}, r_{32}) \quad & \text{si} \quad c\theta > 0 \\ \psi = arc\tan(-r_{33}, -r_{32}) \quad & \text{si} \quad c\theta < 0 \end{aligned}$$

Una vez que se obtuvo la matriz de transformación $\mathbf{Q}_1^6$ es posible obtener las invariantes naturales, vector de dirección $\mathbf{e}$ y el ángulo de rotación ϕ (Ángeles, 1997). Primero se obtiene el vector $\mathbf{q}$ como,

$$\text{vect}(\mathbf{Q}) = \mathbf{q} = \frac{\mathbf{1}}{\mathbf{2}} \begin{bmatrix} q_{32} - q_{23} \\ q_{13} - q_{31} \\ q_{21} - q_{12} \end{bmatrix}$$

se calcula la traza de $\mathbf{Q}$ como,

$$\text{tr}(\mathbf{Q}) = q_{11} + q_{22} + q_{33}$$

con estos valores se calcula el ángulo de rotación ϕ ,

$$\phi = \frac{\text{tr}(\mathbf{Q}) - 1}{2}$$

a partir del ángulo ϕ y el vector $\mathbf{q}$ se calcula el vector de dirección $\mathbf{e}$,

$$\mathbf{e} = \frac{\mathbf{q}}{\sin\boldsymbol{\phi}}$$

2.1.2 Cinemática inversa

La cinemática inversa, parte de un punto en las coordenadas cartesianas con una cierta orientación del órgano terminal del manipulador para así obtener las variables de las articulaciones, ya sean variables angulares o de desplazamiento, que conducirán al manipulador a una configuración deseada.

En el momento en que los últimos tres ejes de las articulaciones se intersecan en un mismo punto, como se muestra en la figura 7, se dice que la cinemática inversa se puede desacoplar en dos problemas más simples, el problema de la cinemática inversa de posición y el problema de la cinemática inversa de orientación.

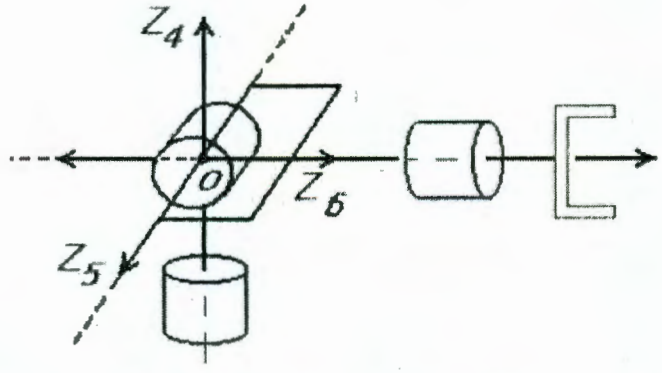

Figura 7. Representación de la intersección de los últimos tres ejes de rotación en un mismo punto.

Una vez que se especifica un punto **p** y la matriz de orientación se obtiene el centro de la muñeca **c**. Para la solución de la cinemática inversa existen varios métodos; uno de ellos es la interpretación geométrica (Spong y Vidyasagar, 1989), cuyo método obtiene la cinemática inversa para un manipulador específico (Martínez, E. et Al. 2000). Otro método (Ángeles, 1997), cuyo algoritmo permite calcular la cinemática inversa de manipuladores seriales desacoplados de seis articulaciones. Este último método se presenta a continuación.

El problema de posicionamiento

El vector $[p]_1$ representa las coordenadas cartesianas del punto de operación del órgano terminal en el marco base, y el vector $[c]_1$ representa las coordenadas cartesianas del centro de la muñeca del manipulador en el marco base.

$$[\mathbf{p}]_1 = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad [\mathbf{c}]_1 = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

para obtener los valores x_c, y_c y z_c se sigue la siguiente ecuación:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} x - (q_{11}a_6 + q_{12}b_5\mu_6 + q_{12}b_6\lambda_6) \\ y - (q_{21}a_6 + q_{22}b_5\mu_6 + q_{22}b_6\lambda_6) \\ z - (q_{31}a_6 + q_{32}b_5\mu_6 + q_{32}b_6\lambda_6) \end{bmatrix}$$

donde q_{ij} son los componentes de la matriz de orientación, que pueden obtenerse por medio de la matriz *roll-yaw-pitch*, ángulos Euler-Lagrange (Spong y Vidyasagar, 1989) o por la matriz formada por los vectores tangente, normal y binormal a cada punto de la trayectoria (Ángeles, 1997).

Una vez que se obtiene la posición del centro de la muñeca con la solución de la cinemática inversa, se calculan los coeficientes de la ecuación: $Ac_1 + Bs_1 + Cc_3 + Ds_3 + E = 0$

$$
\begin{aligned}
A &= 2a_1x_c \\
B &= 2a_1y_c \\
C &= 2a_2a_3 - 2b_2b_4\mu_2\mu_3 \\
D &= 2a_3b_2\mu_2 + 2a_2b_4\mu_3 \\
E &= a_2^2 + a_3^2 + b_2^2 + b_3^2 + b_4^2 - a_1^2 - x_c^2 - y_c^2 - (z_c - b_1)^2 \\
&\quad + 2b_2b_3\lambda_2 + 2b_2b_4\lambda_2\lambda_3 + 2b_3b_4\lambda_3
\end{aligned}
$$

y también, de la ecuación $Fc_1 + Gs_1 + Hc_3 + Is_3 + J = 0$

$$
\begin{aligned}
F &= y_c\mu_1 \\
G &= -x_c\mu_1 \\
H &= -b_4\mu_2\mu_3 \\
I &= a_3\mu_2 \\
J &= b_2 + b_3\lambda_2 + b_4\lambda_2\lambda_3 - (z_c - b_1)\lambda_1
\end{aligned}
$$

de aquí es posible calcular el $\cos\theta_1$ y $\sin\theta_1$ siempre y cuando $\triangle_1 = AG - FB = -2a_1\mu_1(x_c^2 + y_c^2)$ sea diferente de 0.

$$
\begin{aligned}
c_1 &= \frac{-G(Cc_3 + Ds_3 + E) + B(Hc_3 + Is_3 + J)}{\triangle_1} \\
s_1 &= \frac{F(Cc_3 + Ds_3 + E) + A(Hc_3 + Is_3 + J)}{\triangle_1}
\end{aligned}
$$

antes de calcular θ_1 es necesario conocer θ_3 por lo tanto si $\triangle_1 \neq 0$ se calcula los coeficientes de $R\tau_3^4 + S\tau_3^3 + T\tau_3^2 + U\tau_3 + V = 0$, donde

$$
\begin{aligned}
R &= 4a_1^2(J - H)^2 + \mu_1^2(E - C)^2 - 4\rho^2a_1^2\mu_1^2 \\
S &= 4[4a_1^2I(J - H) + \mu_1^2D(E - C)] \\
T &= 2[4a_1^2(J^2 - H^2 + 2I^2) + \mu_1^2(E^2 - C^2 + 2D^2) - 4\rho^2a_1^2\mu_1^2] \\
U &= 4\left[4a_1^2I(H + J) + \mu_1^2D(C + E)\right] \\
V &= 4a_1^2(J + H)^2 + \mu_1^2(E + C)^2 - 4\rho^2a_1^2\mu_1^2
\end{aligned}
$$

y $\rho^2 = x_c^2 + y_c^2$.

Ahora, considerando dos identidades trigonométricas tangente de la mitad del ángulo,

$$
c_3 \equiv \frac{1 - \tau_3^2}{1 + \tau_3^2}, s_3 \equiv \frac{2\tau_3^2}{1 + \tau_3^2},
$$

donde $\tau_3 \equiv \tan\left(\frac{\theta_3}{2}\right)$, y se calcula el valor para $\theta_3 = arc\tan 2(s_3, c_3)$.

Una vez que se obtiene los 4 valores de θ_3, cada uno de éstos se sustituyen en las ecuaciones de c_1 y s_1 anteriores, los cuales producen 4 valores para θ_i.

El valor de $\triangle_1 = 0$ si y sólo si, cualquiera de a_i, $\mu_i = 0$ ó $x_c^2 + y_c^2 = 0$

- Los valores de a_i y μ_i dependen de la arquitectura (en manipuladores industriales por lo regular ocurre que alguna de estas sea igual a cero pero no al mismo tiempo)

- Si $a_i = 0$ y $\mu_i = 0$ entonces la configuración del manipulador no es factible de posicionarse en cualquier punto en el espacio.

- Los valores $x_c + y_c$ depende de la posición. Si $x_c^2 + y_c^2 = 0$ significa que el punto C cae sobre el eje Z_1, tal configuración se le conoce como *"primera singularidad"*. En este caso, cualquier movimiento de la primera articulación con las otras dos articulaciones fijas, no cambia la localización de C,como se exhibe en la figura 8.

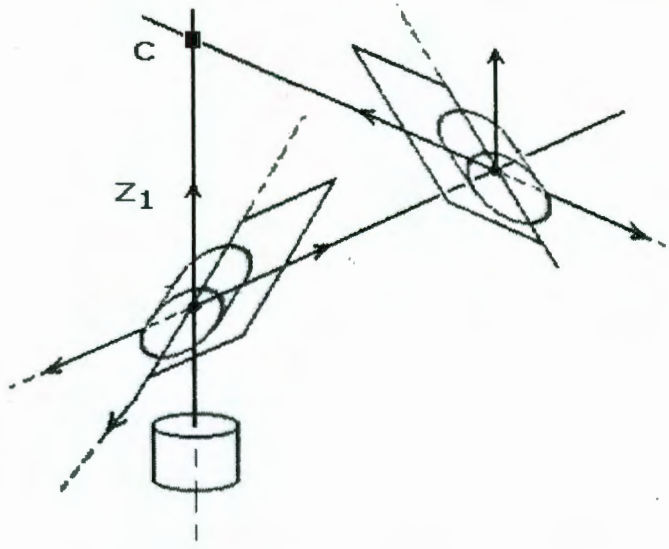

Figura 8. Primera singularidad de un manipulador.

Ahora, se considera la situación en el que $\triangle_1 = 0$, en donde pueden ocurrir dos casos:

+Caso 1. Para los valores $a_1 \neq 0$ y $\mu_1 = 0$,

Se tiene que $A, B \neq 0$, y $F = G = 0$. Por lo tanto,

$$(J - H)\tau_3^2 + 2I\tau_3 + (J + H) = 0$$

la cual produce dos valores para τ_3,

$$(\tau_3)_{1,2} = \frac{-I \pm \sqrt{I^2 - J^2 + H^2}}{J - H}$$

es conveniente verificar que $(I^2 - J^2 + H^2) \geq 0$ para evitar tener raíces imaginarias, ya que ésto indica que el punto no es físicamente realizable.

Para el cálculo de θ_3 se considera el valor τ_3 , donde

$$c_3 = \frac{1 - \tau_3^2}{1 + \tau_3^2}, s_3 = \frac{2\tau_3}{1 + \tau_3^2}$$

Por lo tanto, θ_3 es igual a,

$$\theta_3 = \text{atan2}\ (s_3, c_3)$$

Después de obtener los dos valores para θ_3 se resuelve para θ_1 derivando

$$(E' - A)\,\tau_1^2 + 2B\tau_1 + (E' + A) = 0$$

donde,

$$E' = Cc_3 + Ds_3 + E, \tau_1 \equiv \tan\left(\frac{\theta_1}{2}\right)$$

cuyas raíces

$$(\tau_1)_{1,2} = \frac{-B \pm \sqrt{B^2 - E'^2 + A^2}}{E' - A}$$

para calcular τ_1 se verifica que $(B^2 - E'^2 + A^2) \geq 0$ de lo contrario, la configuración no será realizable para este punto.

Se resuelven los ángulos de rotación para θ_1, de igual forma que para θ_3.

+Caso 2. Para los valores de $a_1 = 0$ y $\mu_1 \neq 0$,

Se tiene que $A = B = 0$, y $F, G \neq 0$. Se resuelve para

$$(E - C)\tau_3^2 + 2D\tau_3 + (E + C) = 0$$

que produce dos valores en τ_3

$$(\tau_3)_{1,2} = \frac{-D \pm \sqrt{D^2 - E^2 + C^2}}{E - C}$$

y se obtienen los valores para θ_3 de igual manera como se indicó anteriormente en el caso 1.

Para calcular θ_1 se resuelve para

$$(J' - F)\tau_1^2 + 2G\tau_1 + (J' + F) = 0$$

donde

$$J' = Hc_3 + Is_3 + J, \tau_1 \equiv \tan\left(\frac{\theta_1}{2}\right)$$

cuyas raíces son,

$$(\tau_1)_{1,2} = \frac{-G \pm \sqrt{G^2 - J'^2 + F^2}}{J' - F}$$

Los valores para θ_1 se calculan de igual manera como se indicó anteriormente en el caso 1.

Después de calcular los valores para cada θ_1 y θ_3 puede calcularse el valor de θ_2 .

$$A_{11} = a_2 + a_3 \cos\theta_3 + b_4\mu_3 \sin\theta_3$$
$$A_{12} \equiv -a_3\lambda_2 \sin\theta_3 + b_3\mu_2 + b_4\lambda_2\mu_3 \cos\theta_3 + b_4\mu_2\lambda_3$$

Si $A_{11} \neq 0$ y $A_{12} \neq 0$ el ángulo θ_2 es,

$$\cos\theta_2 = \frac{1}{\triangle_2}\{A_{11}(x_c\cos\theta_1 + y_c\sin\theta_1 - a_1) - A_{12}[-x_c\lambda_1\sin\theta_1 + y_c\lambda_1\cos\theta_1 + (z_c - b_1)\mu_1]\}$$
$$\sin\theta_2 = \frac{1}{\triangle_2}\{A_{12}(x_c\cos\theta_1 + y_c\sin\theta_1 - a_1) - A_{11}[-x_c\lambda_1\sin\theta_1 + y_c\lambda_1\cos\theta_1 + (z_c - b_1)\mu_1]\}$$

donde $\triangle_2 \equiv A_{11}^2 + A_{12}^2 \equiv a_2^2 + (\cos^2\theta_3 + \lambda_2^2\sin^2\theta_3)a_3^2 + b_3^2\mu_2^2 + 2a_2a_3\cos\theta_3 - 2a_3b_3\lambda_2\mu_2\sin\theta_3$

Si $\triangle_2 = 0$, entonces se presenta la segunda singularidad, esto es, el punto C cae en el eje Z_2 de la segunda articulación.

El problema de orientación

Para el cálculo de la orientación se considera los valores de θ_1, θ_2 y θ_3 para conocer las matrices de rotación, $\mathbf{Q}_1$, $\mathbf{Q}_2$ y $\mathbf{Q}_3$.

Si se conoce que la matriz $\mathbf{Q}_1^6 = \mathbf{Q}$, donde $\mathbf{Q}$ es la matriz de orientación del órgano terminal, es posible,

$$\mathbf{Q}_1\mathbf{Q}_2\mathbf{Q}_3\mathbf{Q}_4\mathbf{Q}_5\mathbf{Q}_6 = \mathbf{Q}$$

Despejando las matrices $\mathbf{Q}$ para θ_4, θ_5 y θ_6 se tiene

$$\mathbf{Q}_4\mathbf{Q}_5\mathbf{Q}_6 = \mathbf{Q}_3^T\mathbf{Q}_2^T\mathbf{Q}_1^T\mathbf{Q}$$

con $\mathbf{R}$ definida como

$$\mathbf{R} = \mathbf{Q}_3^T\mathbf{Q}_2^T\mathbf{Q}_1^T\mathbf{Q}$$

Más aún, sean las entradas de $\mathbf{R}$ en el cuarto marco coordenado dadas como

$$[\mathbf{R}]_4 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Para obtener θ_4 se considera,

$$\tau_4 = \frac{\xi\mu_4 \pm \sqrt{(\xi^2 + \eta^2)\mu_4^2 - (\lambda_5 - \zeta\lambda_4)^2}}{\lambda_5 - \zeta\lambda_4 - \eta\mu_4}$$

y se calcula el valor de θ_4 como se indicó en el caso 1 de la cinemática inversa de posición. Ahora, igualando términos de $\mathbf{Q}_5 = \mathbf{Q}_4^T \mathbf{R} \mathbf{Q}_6^T$ se tiene para θ_5,

$$\mu_5 s_5 = (\mu_6 r_{12} + \lambda_6 r_{13}) c_4 + (\mu_6 r_{22} + \lambda_6 r_{23}) s_4$$
$$-\mu_5 c_5 = -\lambda_4 (\mu_6 r_{12} + \lambda_6 r_{13}) s_4 + \lambda_4 (\mu_6 r_{22} + \lambda_6 r_{23}) c_4 + \mu_4 (\mu_6 r_{32} + \lambda_6 r_{33})$$

produce un sólo valor de θ_5 para cada valor de θ_4.

Finalmente, igualando términos de $\mathbf{Q}_6 = \mathbf{Q}_5^T \mathbf{Q}_4^T \mathbf{R}$ se obtiene para θ_6,

$$c_6 = w_1 c_5 + w_2 s_5$$
$$s_6 = -w_1 \lambda_5 s_5 + w_2 \lambda_5 c_5 + w_3 \mu_5$$

y produce un valor único de θ_6 para cada articulación de valores (θ_4, θ_5).

2.2 Cinemática de velocidad

El estudio de la cinemática de velocidad es de gran utilidad, ya que con esto, se puede especificar una velocidad en las articulaciones y obtener la velocidad con la que se desplazaría el órgano terminal (cinemática directa de velocidad). O por el contrario, especificar la velocidad del órgano terminal y evaluar la velocidad de las articulaciones (cinemática inversa de velocidad).

Si se considera $x = f(\theta)$, donde

$$x = \begin{bmatrix} x \\ y \\ z \\ \psi \\ \theta \\ \phi \end{bmatrix} = f \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{bmatrix}$$

la velocidad en el órgano terminal se obtiene del producto de la matriz jacobiana por las velocidades de las articulaciones.

$$\dot{x} = \frac{\partial f}{\partial \theta}(\dot{\theta})$$

donde el jacobiano J es equivalente a $\frac{\partial f}{\partial \theta}$, por lo tanto,

$$\dot{x} = J\dot{\theta}$$

donde

$$\mathbf{J} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix}$$

igual a,

$$\mathbf{J} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \\ \mathbf{e}_1 \times \mathbf{r}_1 & \mathbf{e}_2 \times \mathbf{r}_2 & \cdots & \mathbf{e}_n \times \mathbf{r}_n \end{bmatrix}$$

y el vector r_i se define a partir de la articulación O_i al punto P, en dirección del primero al último, es decir,

$$r_i \equiv \mathbf{a}_i + \mathbf{a}_{i+1} + \cdots + \mathbf{a}_n$$

Ahora bien, los manipuladores desacoplados permiten solucionar la cinemática de velocidad de manera más simple. Para estos manipuladores es conveniente trabajar con la velocidad del centro de la muñeca C que con el punto de operación P. Así, se tiene

$$\mathbf{t}_c = \mathbf{J}\dot{\boldsymbol{\theta}}$$

donde $\mathbf{t}_c$ está definido como

$$\mathbf{t}_c = \begin{bmatrix} \omega \\ \dot{c} \end{bmatrix}$$

y puede obtenerse de $\mathbf{t}_p \equiv [\omega^T, \dot{\mathbf{p}}^T]^T$ utilizando la siguiente ecuación

$$\mathbf{t}_c \equiv \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{P} - \mathbf{C} & \mathbf{1} \end{bmatrix} \mathbf{t}_p$$

con $\mathbf{C}$ y $\mathbf{P}$ definidos como las matrices producto-cruz de los vectores de posición $\mathbf{c}$ y $\mathbf{p}$, respectivamente.

Si C es el punto de intersección de los últimos tres ejes, los movimientos de éstos no afectan su velocidad, y se puede escribir

$$\dot{\mathbf{c}} = \dot{\theta}_1 \mathbf{e}_1 \times \mathbf{r}_1 + \dot{\theta}_2 \mathbf{e}_2 \times \mathbf{r}_2 + \dot{\theta}_3 \mathbf{e}_3 \times \mathbf{r}_3$$

en el caso de un manipulador desacoplado, el vector $\mathbf{r}_i$ está definido como la dirección de O_i a C. Por otro lado, se tiene

$$\mathbf{J} \equiv \begin{bmatrix} \mathbf{J}_{11} & \mathbf{J}_{11} \\ \mathbf{J}_{21} & \mathbf{O} \end{bmatrix}$$

donde $\mathbf{O}$ denota la matriz zero de 3×3. Los bloques 3×3 de $\mathbf{J}_{11}$, $\mathbf{J}_{12}$ y $\mathbf{J}_{21}$ se dan a continuación, para *manipuladores con articulaciones únicamente*,

$$\mathbf{J}_{11} = [\ \mathbf{e}_1 \quad \mathbf{e}_2 \quad \mathbf{e}_3\ ]$$
$$\mathbf{J}_{12} = [\ \mathbf{e}_4 \quad \mathbf{e}_5 \quad \mathbf{e}_6\ ]$$
$$\mathbf{J}_{21} = [\ \mathbf{e}_1 \times \mathbf{r}_1 \quad \mathbf{e}_2 \times \mathbf{r}_2 \quad \mathbf{e}_3 \times \mathbf{r}_3\ ]$$

Más aún, el vector $\dot{\theta}$ está *particionado* de acuerdo:

$$\dot{\theta} \equiv \begin{bmatrix} \dot{\theta}_a \\ \dot{\theta}_\omega \end{bmatrix}$$

donde

$$\dot{\theta}_a \equiv \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \quad , \quad \dot{\theta}_\omega \equiv \begin{bmatrix} \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{bmatrix}$$

De ahora en adelante, las tres componentes de $\dot{\theta}_a$ se referirán como la *velocidad del brazo*, mientras que $\dot{\theta}_\omega$ se llamará la *velocidad de la muñeca*. Por lo tanto, para manipuladores desacoplados se tiene

$$\begin{aligned} \mathbf{J}_{11}\dot{\boldsymbol{\theta}}_a + \mathbf{J}_{12}\dot{\boldsymbol{\theta}}_\omega &= \omega \\ \mathbf{J}_{21}\dot{\boldsymbol{\theta}}_a &= \dot{\mathbf{c}} \end{aligned}$$

al despejar se obtiene,

$$\begin{aligned} \mathbf{J}_{21}\dot{\boldsymbol{\theta}}_a &= \dot{\mathbf{c}} \\ \mathbf{J}_{12}\dot{\boldsymbol{\theta}}_\omega &= \omega - \mathbf{J}_{11}\dot{\boldsymbol{\theta}}_a \end{aligned}$$

de las relaciones cinemáticas anteriores es posible obtener,

$$\begin{aligned} \dot{\boldsymbol{\theta}}_a &= \mathbf{J}_{21}^{-1}\dot{\mathbf{c}} \\ \dot{\boldsymbol{\theta}}_\omega &= \mathbf{J}_{12}^{-1}(\omega - \mathbf{J}_{11}\dot{\boldsymbol{\theta}}_a) \end{aligned}$$

de esta manera, se obtiene la cinemática inversa de velocidad.

En el cálculo de las derivadas de las articulaciones para un manipulador desacoplado, se asume que $\mathbf{J}_{12}$ y $\mathbf{J}_{21}$ son no-singulares. Si el último es singular entonces ninguna de las derivadas de las articulaciones puede resolverse, aún si el primero es no-singular. Sin embargo, si $\mathbf{J}_{21}$ es no-singular, entonces la ecuación $[\mathbf{J}_{11}\dot{\boldsymbol{\theta}}_a + \mathbf{J}_{12}\dot{\boldsymbol{\theta}}_\omega = \omega]$ puede resolverse para las derivadas del brazo aún si $\mathbf{J}_{12}$ es singular.

Ahora se analiza $\mathbf{J}_{21}$, cuya singularidad determina si es posible cualquier solución para la derivada de la articulación. Primero, a partir de la ecuación

$$[\mathbf{J}_{21} = [\ \mathbf{e}_1 \times \mathbf{r}_1 \quad \mathbf{e}_2 \times \mathbf{r}_2 \quad \mathbf{e}_3 \times \mathbf{r}_3\ ]]$$

donde las columnas de $\mathbf{J}_{21}$ son tres vectores $\mathbf{e}_1 \times \mathbf{r}_1$, $\mathbf{e}_2 \times \mathbf{r}_2$ y $\mathbf{e}_3 \times \mathbf{r}_3$. $\mathbf{J}_{21}$ es singular si cualquiera de estos tres vectores se hace coplanar o al menos uno de ellos desaparece, a esto se le llama *singularidad del brazo.*

Con respecto a la *singularidad de la muñeca*, se estudia con respecto al sub-Jacobiano $\mathbf{J}_{12}$ de la ecuación $\mathbf{J}_{12} = [\ \mathbf{e}_4 \quad \mathbf{e}_5 \quad \mathbf{e}_6\ ]$. Este sub-Jacobiano desaparece cuando la muñeca se configura de forma tal que sus tres ejes son coplanares, es decir $\mathbf{e}_4 \times \mathbf{e}_5 \cdot \mathbf{e}_6 = 0$.

2.3 Planeación de trayectorias

Con la planeación de trayectorias se pretende obtener rutas lo más suaves posibles, es decir, deben evitarse los cambios abruptos en la posición, velocidad y aceleración. De hecho, los movimientos abruptos requieren cantidades ilimitadas de energía para que se implementen, en la que los motores no pueden proporcionarla a causa de sus limitaciones físicas. Por otro lado, debe evitarse que el robot colisione con algún objeto. Este trabajo se enfoca a la planeación de movimientos del robot en ambientes estructurados.

Dos tareas típicas para las técnicas de planeación de trayectoria son,

+ Trayectorias punto a punto (*pick-and-place operations*).
+ Trayectorias continuas (*continuous paths*).

2.3.1 Trayectorias punto a punto

En las trayectorias punto a punto se especifica un *punto inicial* y un *punto final* con orientaciones respectivas para cada una. Para sintetizar la trayectoria de las articulaciones, se debe mapear el punto inicial y final de la pieza de trabajo, dentro del espacio de las articulaciones. Se consideran las siguientes condiciones (Ángeles, 1997),

$$\begin{array}{lll} \mathbf{p}(0) = \mathbf{p}_I & \dot{\mathbf{p}}(0) = \mathbf{0} & \ddot{\mathbf{p}}(0) = \mathbf{0} \\ \mathbf{Q}(0) = \mathbf{Q}_I & \boldsymbol{\omega}(0) = \mathbf{0} & \dot{\boldsymbol{\omega}}(0) = \mathbf{0} \\ \mathbf{p}(\mathrm{T}) = \mathbf{p}_F & \dot{\mathbf{p}}(\mathrm{T}) = \mathbf{0} & \ddot{\mathbf{p}}(\mathrm{T}) = \mathbf{0} \\ \mathbf{Q}(\mathrm{T}) = \mathbf{Q}_F & \boldsymbol{\omega}(\mathrm{T}) = \mathbf{0} & \dot{\boldsymbol{\omega}}(\mathrm{T}) = \mathbf{0} \end{array}$$

En ausencia de singularidades, las condiciones de velocidad cero y aceleración cero implican que,

$$\begin{array}{lll} \boldsymbol{\theta}(0) = \boldsymbol{\theta}_I & \dot{\boldsymbol{\theta}}(0) = \mathbf{0} & \ddot{\boldsymbol{\theta}}(0) = \mathbf{0} \\ \boldsymbol{\theta}(\mathrm{T}) = \boldsymbol{\theta}_F & \dot{\boldsymbol{\theta}}(\mathrm{T}) = \mathbf{0} & \ddot{\boldsymbol{\theta}}(\mathrm{T}) = \mathbf{0} \end{array}$$

Existen varios métodos para obtener la planeación de trayectorias:

-Interpolación polinomial 3-4-5
-Interpolación polinomial 4-5-6-7
-Movimiento cicloidal
-Trayectoria vía puntos
-Splines cúbicas

En este trabajo se presenta la interpolación polinomial 4-5-6-7, ya que éste elimina los cambios abruptos (*jerk*) al inicio y al término de la interpolación. También se utilizan las splines cúbicas, porque permiten especificar varios puntos para la secuencia de una trayectoria.

Interpolación polinomial 4-5-6-7

En la interpolación polinomial 4-5-6-7 se agregan dos condiciones

$$s'''(0) = 0, \quad s'''(1) = 0$$

que hacen que desaparezca la tercera derivada al inicio y al final del trayecto de la ecuación de interpolación,

$$s(\tau) = a\tau^7 + b\tau^6 + c\tau^5 + d\tau^4 + e\tau^3 + f\tau^2 + g\tau + h$$

La tercera derivada, también conocida como la articulación tirón-bruzco (*jerk*), requiere de una energía adicional la cual no es posible que la realice el robot. Las derivadas de la ecuación polinomial son,

$$\begin{aligned} s'(\tau) &= 7a\tau^6 + 6b\tau^5 + 5c\tau^4 + 4d\tau^3 + 3e\tau^2 + 2f\tau + g \\ s''(\tau) &= 42a\tau^5 + 30b\tau^4 + 20c\tau^3 + 12d\tau^2 + 6e\tau + 2f \\ s'''(\tau) &= 210a\tau^4 + 120b\tau^3 + 60c\tau^2 + 24d\tau + 6e \end{aligned}$$

Las tres condiciones iniciales conducen a

$$e = f = g = h = 0$$

Más aún, conducen a cuatro ecuaciones lineales con cuatro incognitas,

$$\begin{aligned} a + b + c + d &= 1 \\ 7a + 6b + 5c + 4d &= 0 \\ 42a + 30b + 20c + 12d &= 0 \\ 210a + 120b + 60c + 24d &= 0 \end{aligned}$$

y de aquí se obtiene la solución

$$a = -20, \qquad b = 70 \qquad c = -84 \qquad d = 35$$

siendo así el polinomio deseado como

$$s(\tau) = -20\tau^7 + 70\tau^6 - 84\tau^5 + 35\tau^4$$

es cual es un *polinomial 4-5-6-7*. El polinomial 4-5-6-7 es similar al polinomial 3-4-5 (Ángeles, 1997), excepto que la tercera derivada de el primero desaparece en los extremos de el intervalo de interés. Donde

$$\tau = \frac{t}{T}$$

Teniendo así un *polinomio normal* que permite representar cada una de las variables de las articulaciones $\boldsymbol{\theta}_j$ a través de su intevalo de movimiento, tal que

$$\theta_J(t) = \theta_j^I + (\theta_j^F - \theta_j^I)s(\tau)$$

donde θ_j^I y θ_j^F se dan como valor inicial y final de las variables de la j-ésima articulación.

$$\theta(t) = \theta_I + (\theta_F - \theta_I)s(\tau)$$

y por lo tanto,

$$\dot{\theta}(t) = (\theta_F - \theta_I)s'(\tau)\dot{\tau}(t) = (\theta_F - \theta_I)\frac{1}{T}s'(\tau)$$

de igual modo,

$$\ddot{\theta}(t) = \frac{1}{T^2}(\theta_F - \theta_I)s''(\tau)$$

y

$$\dddot{\theta}(t) = \frac{1}{T^3}(\theta_F - \theta_I)s'''(\tau)$$

Un movimiento alternativo que produce velocidad y aceleración cero en los extremos de un intervalo finito es el *movimiento cicloidal* (vease anexo 2).

Splines cúbicas

Cuando el número de puntos que se especifican en un trayectoria se incrementan, es recomendable utilizar las splines cúbicas, como una alternativa de polinomios de alto-orden. La característica atractiva de las splines cúbicas es que pueden definir un conjunto de polinomios de bajo-orden unidos a un número de puntos de soporte. Las matrices que provienen de un problema de interpolación asociadas con una función spline, son tal que su número de condiciones es sólo superficial dependiendo del número de puntos de soporte. Las splines cúbicas ofrecen la posibilidad de interpolación sobre un número ilimitado virtualmente de puntos sin producir problemas en las condicionantes numéricas. Las splines cúbicas son especialmente útiles para la planeación de rutas en la robótica.

Una función spline cúbica $s(x)$, que conecta N puntos $P_k(x_k, y)$, para $k = 1, 2, ..., N$, es una función definida por $N-1$ polinomios cúbicos uniendo los puntos P_k, tal que $s(x_k) = y(k)$. Más aún, la función spline así definida es dos veces diferenciable en cualquier parte en $x_1 \leq x \leq x_N$. Por lo tanto, las splines cúbicas se dicen ser funciones C^2, es decir, tener derivadas continuas al segundo orden.

Sean $P_k(x_k, y_k)$ y $P_{k+1}(x_{k+1}, y_{k+1})$ los puntos de soporte consecutivos. El polinomio cúbico k-ésimo, $s_k(x)$ entre estos puntos esta dado por

$$s_k(x) = A_k(x - x_k)^3 + B_k(x - x_k)^2 + C_k(x - x_k) + D_k$$

si $x_k \leq x \leq x_{k+1}$. Así, para la spline $s(x)$, se determinan los $4(N-1)$ coeficientes A_k, B_k, C_k, y D_k en $k = 1, ..., N-1$, como

$$A_k = \frac{1}{6\triangle x_k}\left(s''_{k+1} - s''_k\right)$$

$$B_k = \frac{1}{2}s''_k$$

$$C_k = \frac{\triangle s_k}{\triangle x_k} - \frac{1}{6}\triangle x_k(s''_{k+1} + 2s''_k)$$

$$D_k = s_k$$

donde se tiene las abreviaciones $s_k \equiv s(x_k)$, $s'_k \equiv s'(x_k)$ y $s'_k \equiv s'(x_k)$. Estos coeficientes se sustituyen en la primera y segunda derivada de $s_k(x)$ que son,

$$s'_k(x) = 3A_k(x - x_k)^2 + 2B_k(x - x_k) + C_k$$
$$s''_k(x) = 6A_k(x - x_k) + 2B_k$$

también, se definen $\triangle x_k \equiv x_{k+1} - x_k$ y $\triangle s_k \equiv s_{k+1} - s_k$.

Además, el vector N-dimensional $\mathbf{s}$ cuya k-ésima componente es s_k, con el vector s'' definido como,

$$\mathbf{s} = [s_1, ..., s_N]^T, \quad \mathbf{s''} = [s''_1, ..., s''_N]^T$$

La relación entre $\mathbf{s}$ y $\mathbf{s''}$ se puede reescribir de la siguiente forma,

$$\mathbf{As''} = \mathbf{6Cs}$$

donde $\mathbf{A}$ y $\mathbf{C}$ son matrices $(N-2) \times N$ definidas como:

$$\mathbf{A} = \begin{bmatrix} \alpha_1 & 2\alpha_{1,2} & \alpha_2 & 0 & \cdots & 0 & 0 \\ 0 & \alpha_2 & 2\alpha_{2,3} & \alpha_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \alpha_{N'''} & 2\alpha_{N''',N''} & \alpha_{N''} & 0 \\ 0 & 0 & 0 & \cdots & \alpha_{N''} & 2\alpha_{N'',N'} & \alpha_{N'} \end{bmatrix}$$

y

$$\mathbf{C} = \begin{bmatrix} \beta_1 & -\beta_{1,2} & \beta_2 & 0 & \cdots & 0 & 0 \\ 0 & \beta_2 & -\beta_{2,3} & \beta_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \beta_{N'''} & -\beta_{N''',N''} & \beta_{N''} & 0 \\ 0 & 0 & 0 & \cdots & \beta_{N''} & -\beta_{N'',N'} & \beta_{N'} \end{bmatrix}$$

para $i, j, k = 1, ..., N-1$,

$$\alpha_k \equiv \triangle x_k, \qquad \alpha_{i,j} \equiv \alpha_i + \alpha_j$$
$$\beta_k \equiv 1/\alpha_k, \qquad \beta_{i,j} \equiv \beta_i + \beta_j$$
$$N' \equiv N - 1, \qquad N'' \equiv N - 2, \qquad N''' = N - 3$$

Por otro lado, si se está interesado en funciones periódicas, las cuales son el caso cuando los movimientos de *tomar-y-colocar*, se imponen las condiciones $s_1 = s_N$, $s'_1 = s'_N$ y $s''_1 = s''_N$ de tal manera que producen *splines cúbicas periódicas.* (Ángeles, 1997)

2.3.2 Trayectorias continuas

Las trayectorias continuas se especifican utilizando una ecuación paramétrica que proporcionan los puntos P en los ejes X, Y y Z. La matriz de orientación para el órgano terminal se puede especificar por la matriz *roll-yaw-pitch*, la matriz de los ángulos Euler-Lagrange o la matriz formada por los vectores tangente, normal y binormal. Esta última matriz se obtiene especificando un vector tangente a la trayectoria. El vector binormal se compone del producto cruz de dos vectores sobre la superficie de la trayectoria. Por último, el vector normal se obtiene del producto cruz del vector tangente con el vector normal.

Capítulo 3

Operación del simulador

La estructura del programa que se desarrolló se muestra en los siguientes diagramas de bloques y diagramas de flujo.

3.1 Diagramas de bloques

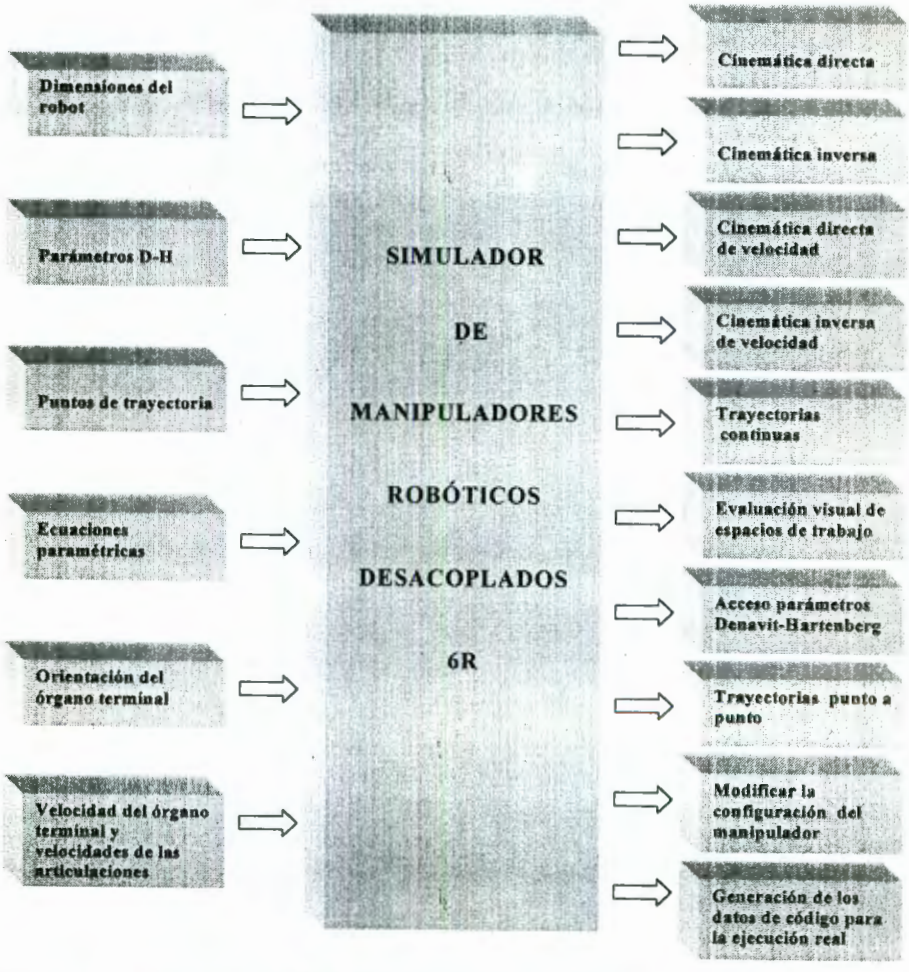

Figura 9. Diagrama global del programa de simulación.

El diagrama global del simulador, figura 9, muestra todas las funciones que realiza el simulador. La interfase del simulador se presenta en forma de persianas y cada una de estas realizan funciones distintas. Esta interfase facilita el acceso y permite la interacción con el usuario. El simulador calcula la cinemática directa, cinemática inversa, cinemática directa de velocidad, cinemática inversa de velocidad, planeación de trayectorias continuas y planeación de trayectorias punto a punto. El programa calcula los movimientos del robot en cada instante de la trayectoria permitiendo al usuario visualizar las posibles colisiones que se presenten con alguno de sus componentes y/o con su entorno. Una vez seleccionada la solución de posicionamiento del robot en el seguimiento de la trayectoria, genera un archivo con los ángulos de rotación de cada una de las articulaciones.

A continuación se presenta una serie de figuras que muestran los módulos que componen el simulador. Cada uno de estos módulos se detalla en la siguiente sección mediante diagramas de flujo.

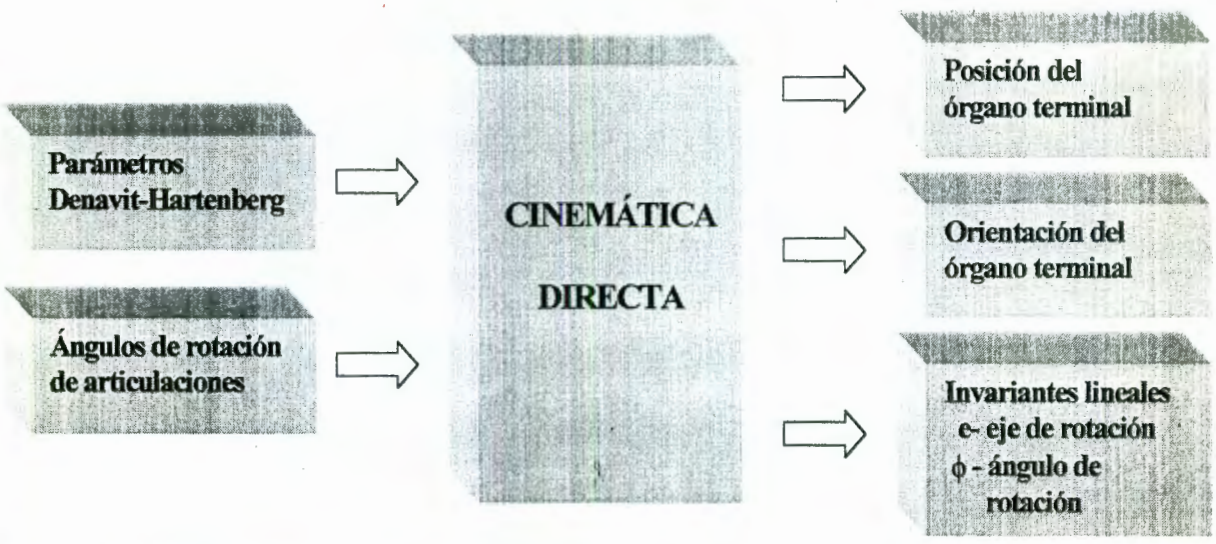

Figura 10. Módulo para la cinemática directa.

El módulo de la cinemática directa, figura 10, obtiene el vector de posición y la matriz de orientación del órgano terminal de cualquier ángulo de rotación de cada una de las articulaciones. Además en este módulo se obtienen los invariantes naturales; esto es, el vector **e** que indica la dirección del eje de rotación y ϕ que es el ángulo de rotación, como se mencionó en la sección 2.1.

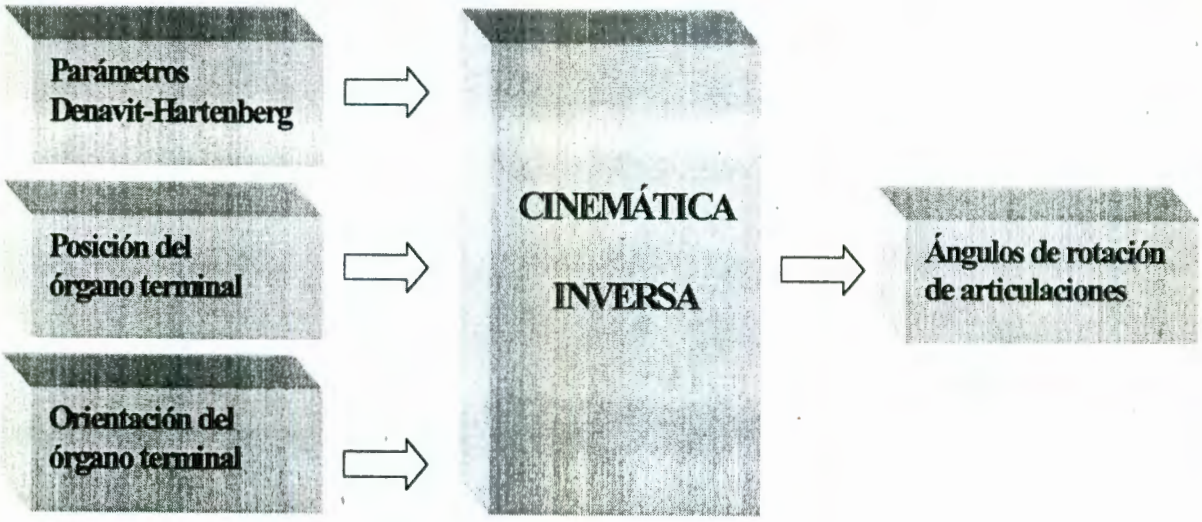

Figura 11. Módulo para la cinemática inversa.

En el módulo de la cinemática inversa, figura 11, se especifica un vector de posición x, y y z y una matriz de orientación. Con estos datos se obtienen las diferentes soluciones para los ángulos de rotación de cada una de las articulaciones. En la simulación se grafican estas soluciones y se pueden visualizar aquellas en las que existen colisiones.

Figura 12. Módulo para la cinemática directa de velocidad.

La cinemática directa de velocidad, figura 12, es útil cuando se desea especificar que las articulaciones se muevan a una cierta velocidad. La velocidad con la que se mueva el extremo del manipulador dependerá de la velocidad de las articulaciones.

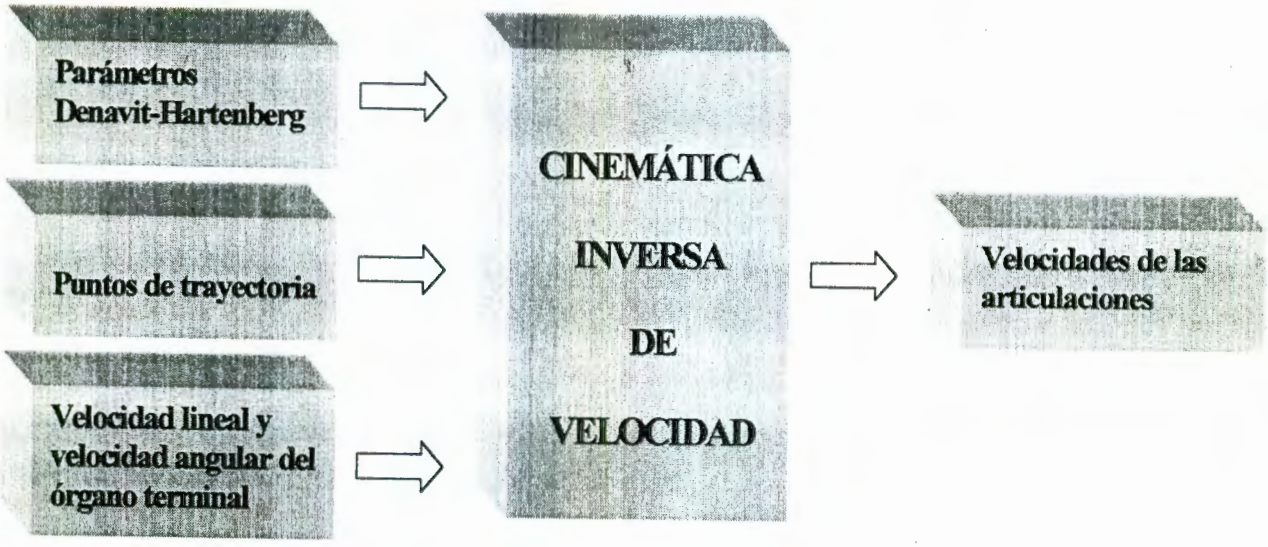

Figura 13. Módulo para la cinemática inversa de velocidad.

En el caso contrario, si se desea que el órgano terminal se desplace a una cierta velocidad en el seguimiento de una trayectoria, se obtienen las velocidades de cada articulación mediante la cinemática inversa de velocidad, figura 13, para producir la velocidad del órgano terminal deseada.

Figura 14. Módulo para la planeación de trayectorias continuas.

El simulador permite planear la trayectoria previamente a su ejecución real, en el cual se visualizan las configuraciones que puede adquirir el manipulador. La trayectorias que pueden obtenerse con este simulador son para trayectorias continuas como se muestra figura 14, y trayectorias punto a punto.

Figura 15. Módulo para la planeación de trayectorias punto a punto.

La matriz de orientación representa la dirección con la que el robot alcanzará un punto. Esta orientación puede ser constante o variable en cada punto de la trayectoria. La trayectoria punto a punto, figura 15, se calculó utilizando polinomios de interpolación 4-5-6-7 y las funciones splines cúbicas. Las splines cúbicas se emplean cuando se desea conectar una secuencia de puntos en forma suave. El uso de la spline favorece el desarrollo de los cálculos cuando se presentan dos o más puntos en un recorrido. Con la interpolación polinomial 4-5-6-7 se evitan los movimientos de *jerk* al inicio y al término del trayecto.

3.2 Diagramas de flujo

En las siguientes figuras se presentan los diagramas de flujo de las tareas que puede realizar el simulador.

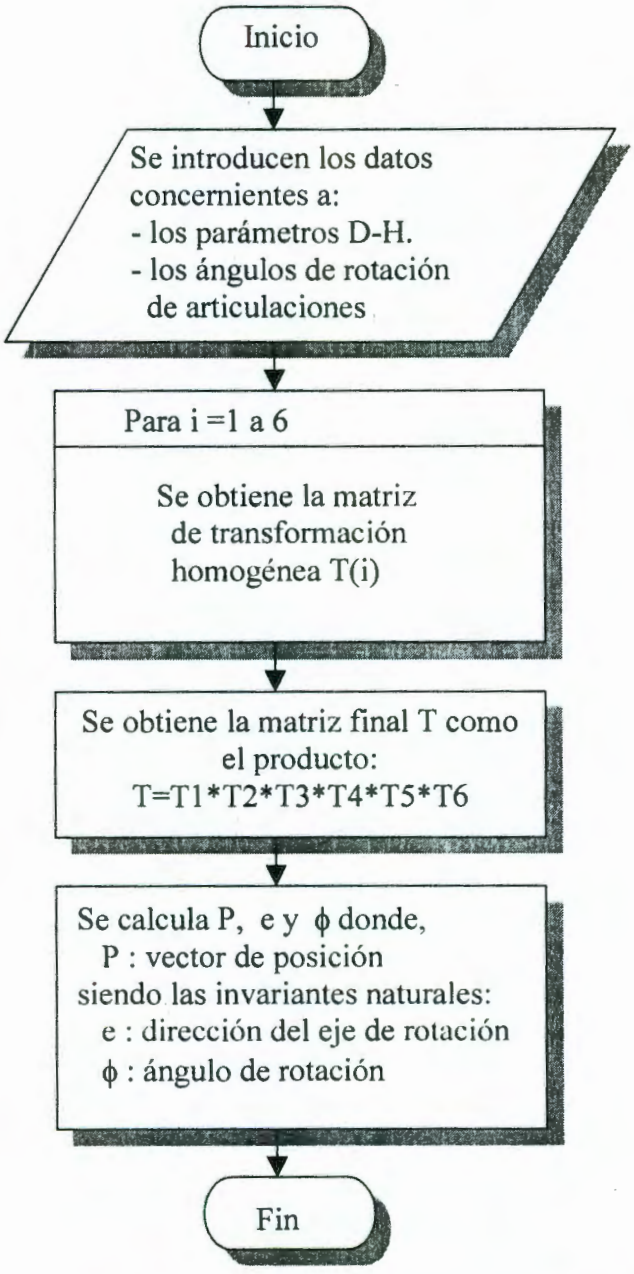

Figura 16. Diagrama de flujo de la cinemática directa.

En el diagrama de flujo de la figura 16 se presenta la secuencia de pasos requeridos para obtener la cinemática directa. El programa grafica los movimientos de cada eslabón e imprime el vector de posición, la matriz de orientación y los invariantes naturales. El problema de la cinemática inversa se separa en cinemática inversa de posición y cinemática inversa de orientación. Las ecuaciones y fórmulas para los cálculos de estos sub-módulos están expresados en la sección 2.1.

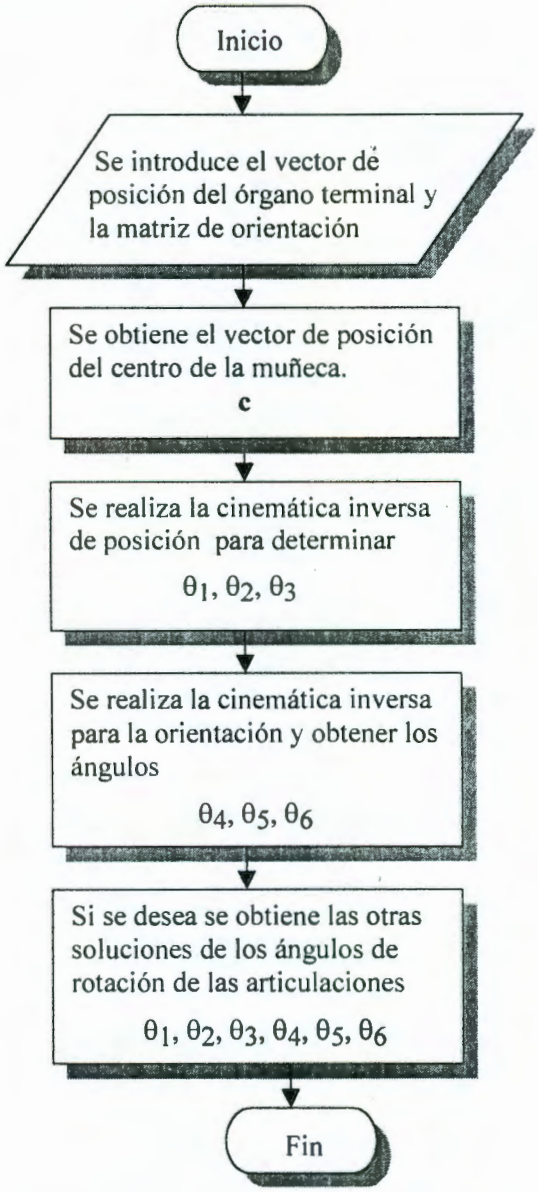

Figura 17. Diagrama de flujo de la cinemática inversa.

El cálculo de la cinemática inversa, figura 17, se desarrolló utilizando el algoritmo de la sección 2.1.2. Con este algoritmo se obtiene los ángulos de rotación para manipuladores desacoplados de seis articulaciones.

En el cálculo de la cinemática inversa, el programa de simulación grafica las diferentes soluciones para posicionar al robot en un mismo punto. El programa permite modificar el vector de posición y la matriz de orientación para calcular nuevamente los ángulos de las articulaciones.

El diagrama de flujo de la cinemática inversa de posición, figura 18, señala tres formas de calcular los valores para los ángulos θ_1 y θ_3.

Forma 1. Se obtienen los coeficientes de la ecuación $R\tau_3^4 + S\tau_3^3 + T\tau_3^2 + U\tau_3 + V = 0$. Se calculan las raíces de este polinomio y se obtiene θ_3 para despues calcular θ_1.

Forma 2. Se calculan los coeficientes para la ecuación $(J - H)\tau_3^2 + 2I\tau_3 + (J + H) = 0$ para calcular θ_3 y los coeficientes de $(E' - A)\tau_1^2 + 2B\tau_1 + (E' + A) = 0$ para calcular θ_1.

Forma 3. Se calculan los coeficientes para la ecuación $(E-C)\tau_3^2+2D\tau_3+(E+C)=0$ para calcular θ_3 y los coeficientes de $(J'-F)\tau_1^2+2G\tau_1+(J'+F)=0$ para calcular θ_1.

La figura 19 muestra el diagrama de flujo para la cinemática inversa de orientación.

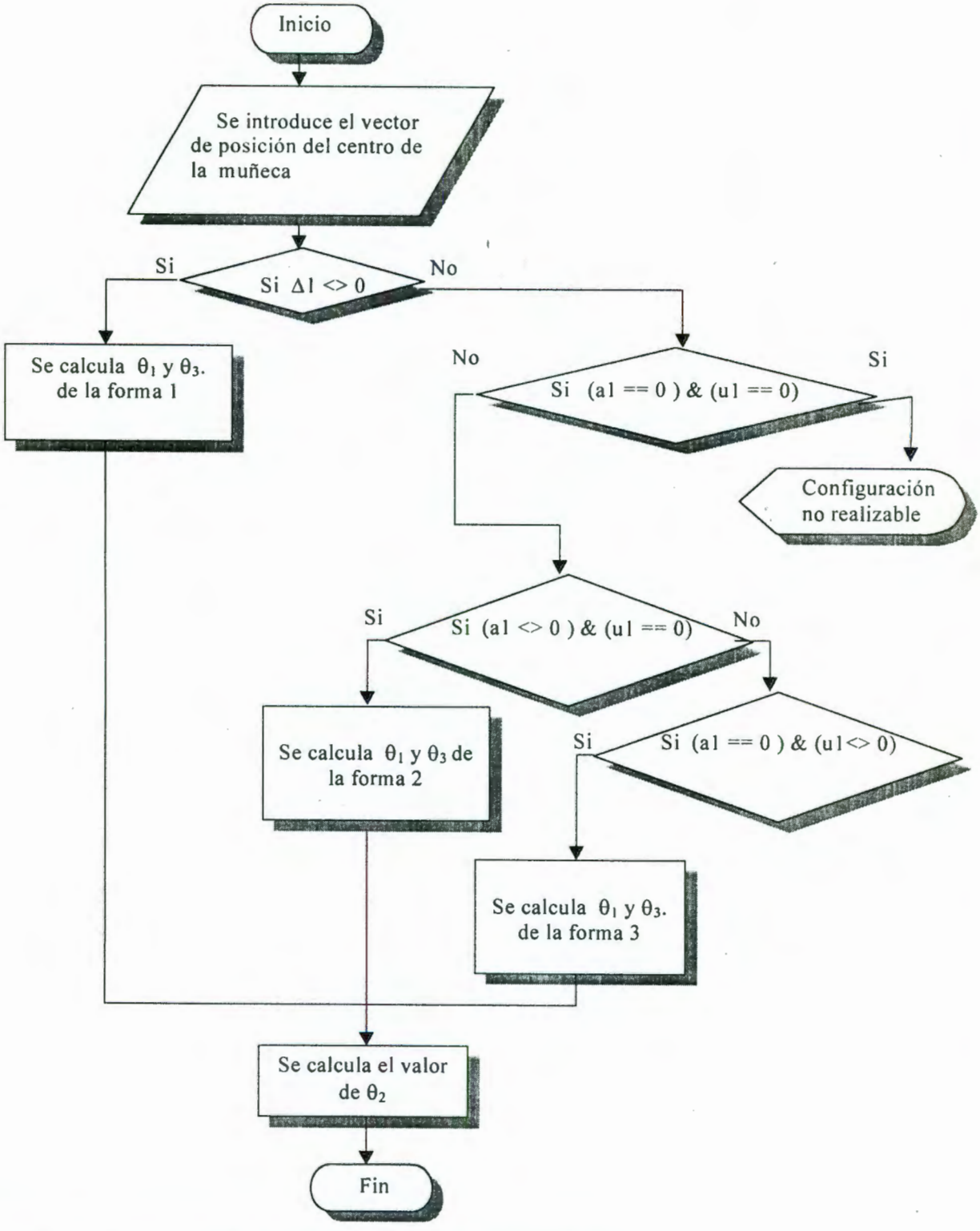

Figura 18. Diagrama de flujo de la cinemática inversa de posición.

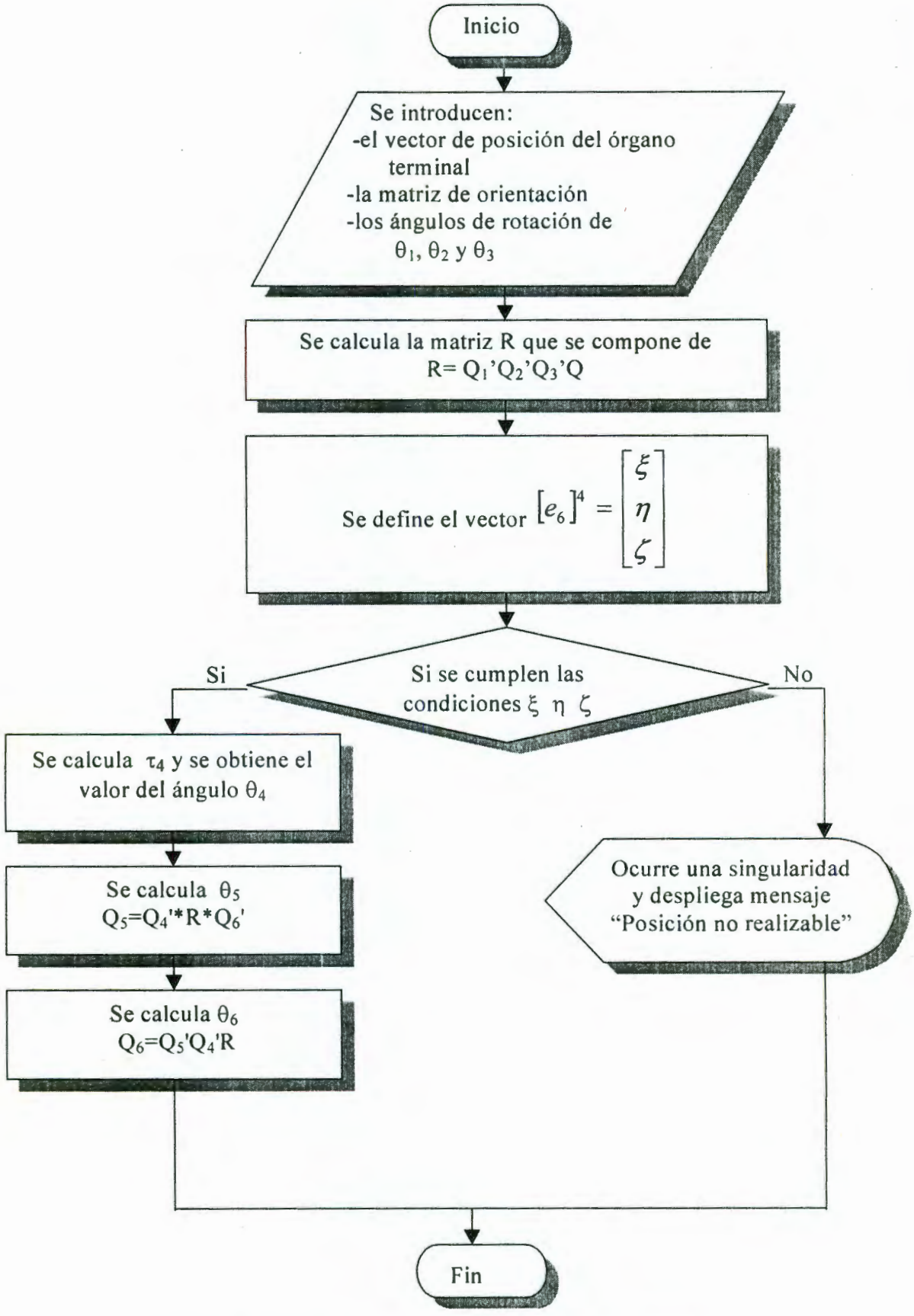

Figura 19. Diagrama de flujo de la cinemática inversa de orientación.

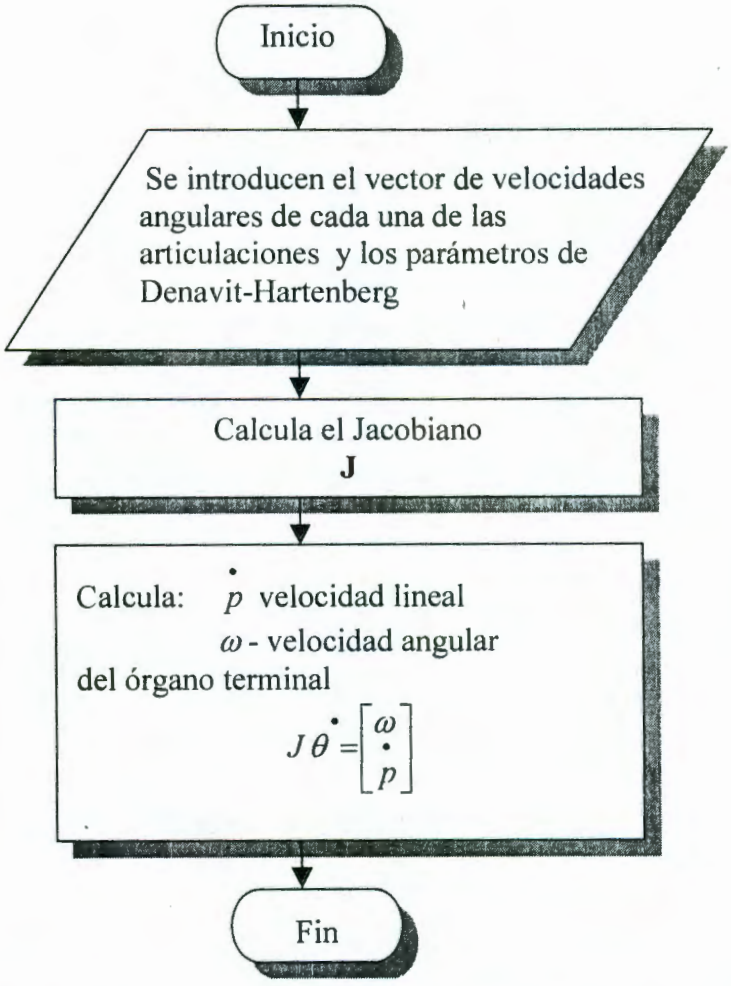

Figura 20. Diagrama de flujo de la cinemática directa de velocidad.

La figura 20 presenta el diagrama de flujo de la cinemática directa de velocidad. El cálculo del jacobiano y la obtención de la velocidad del órgano terminal se describen en la sección 2.2 de este trabajo. La figura 21 presenta el diagrama de flujo de la cinemática inversa de velocidad, cuyos cálculos se detallan en la misma sección.

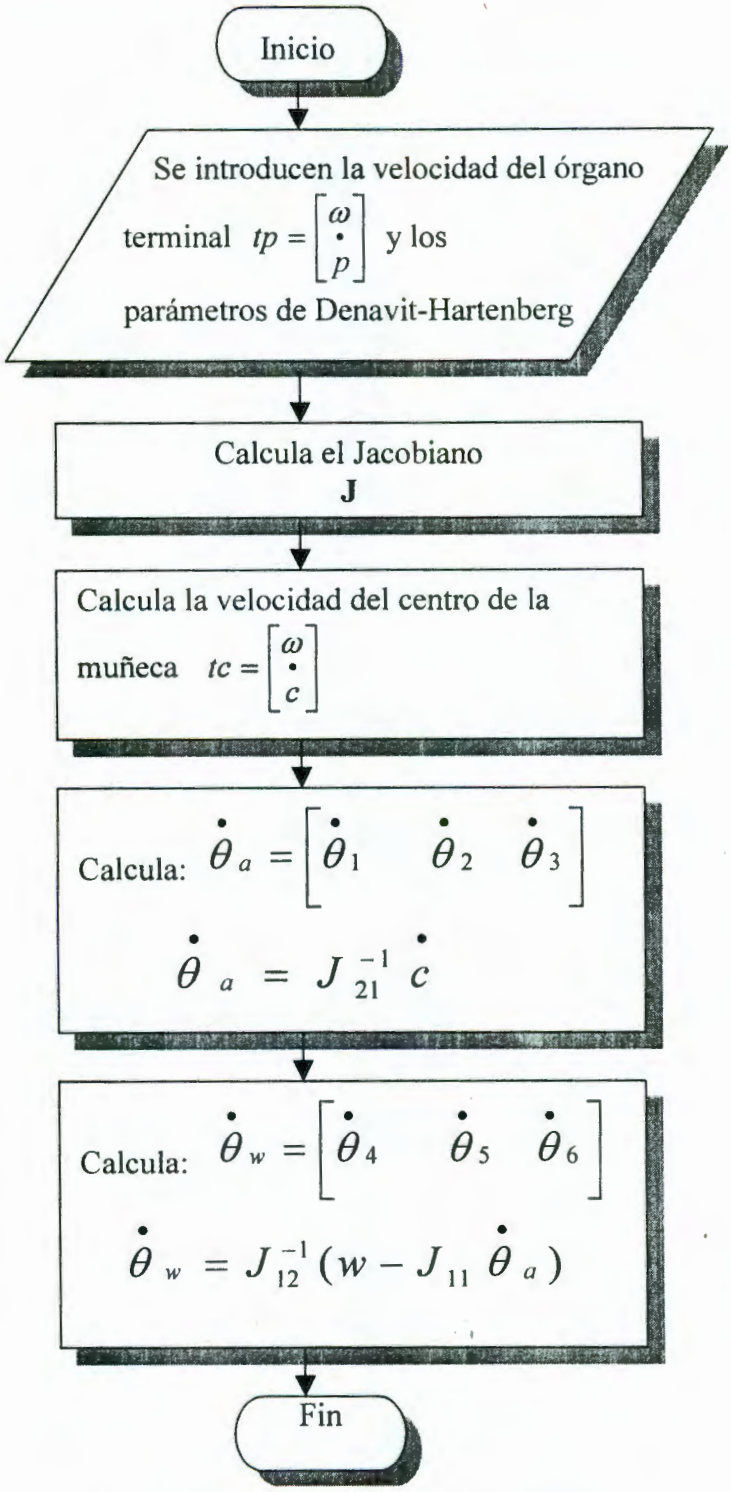

Figura 21. Diagrama de flujo de la cinemática inversa de velocidad.

El diagrama de la figura 21 presenta las instrucciones requeridas para determinar la velocidad que deberán tener las articulaciones para que el órgano terminal se mueva a una velocidad específica.

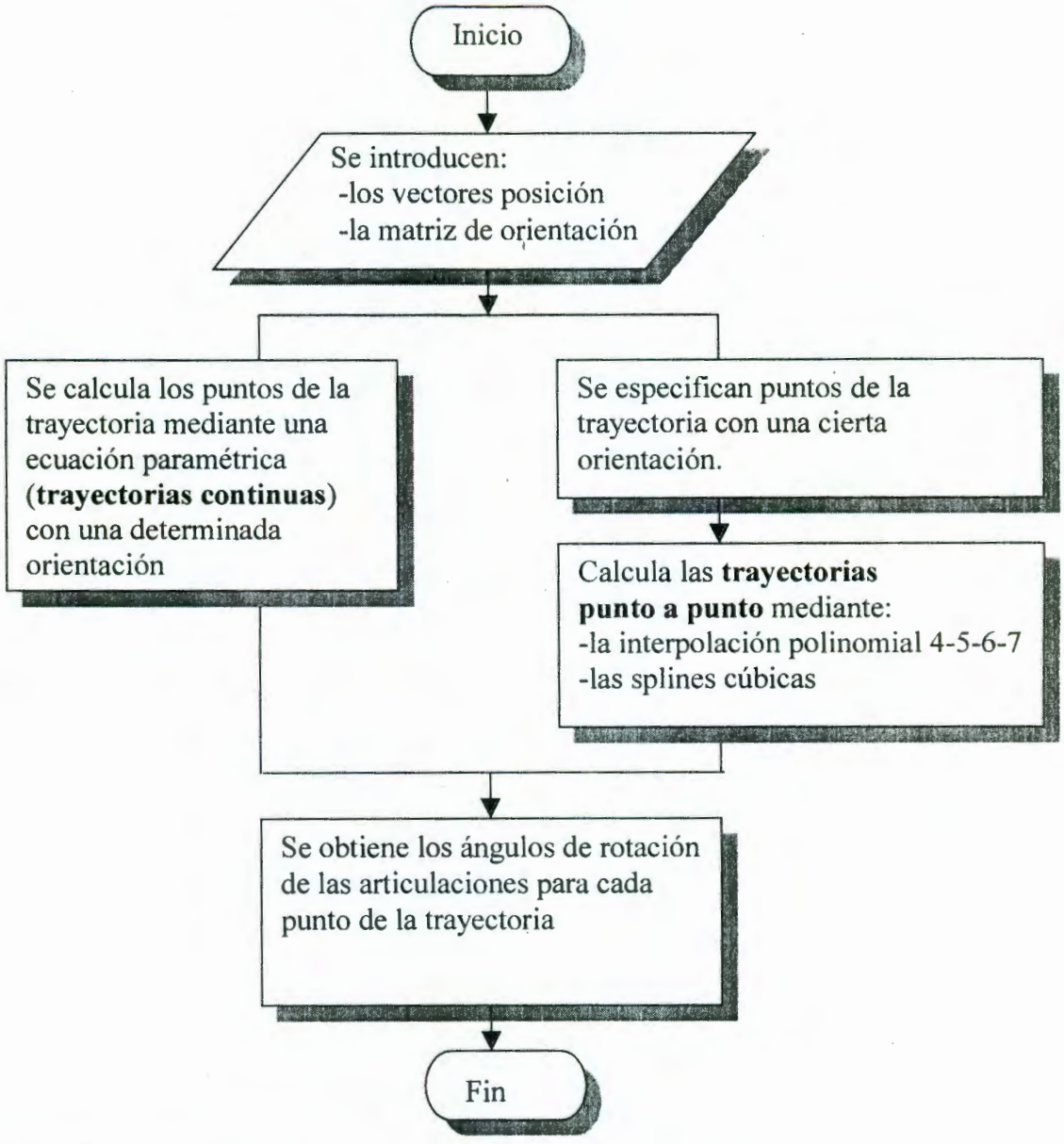

Figura 22. Diagrama de flujo de la planeación de trayectorias.

Los sub-módulos de la planeación de trayectorias de la figura 22, trayectorias continuas y trayectorias punto a punto, requieren expresarse en diagramas de flujo por separado. Cada uno de estos diagramas muestran los pasos para desarrollar cada tipo de recorrido.

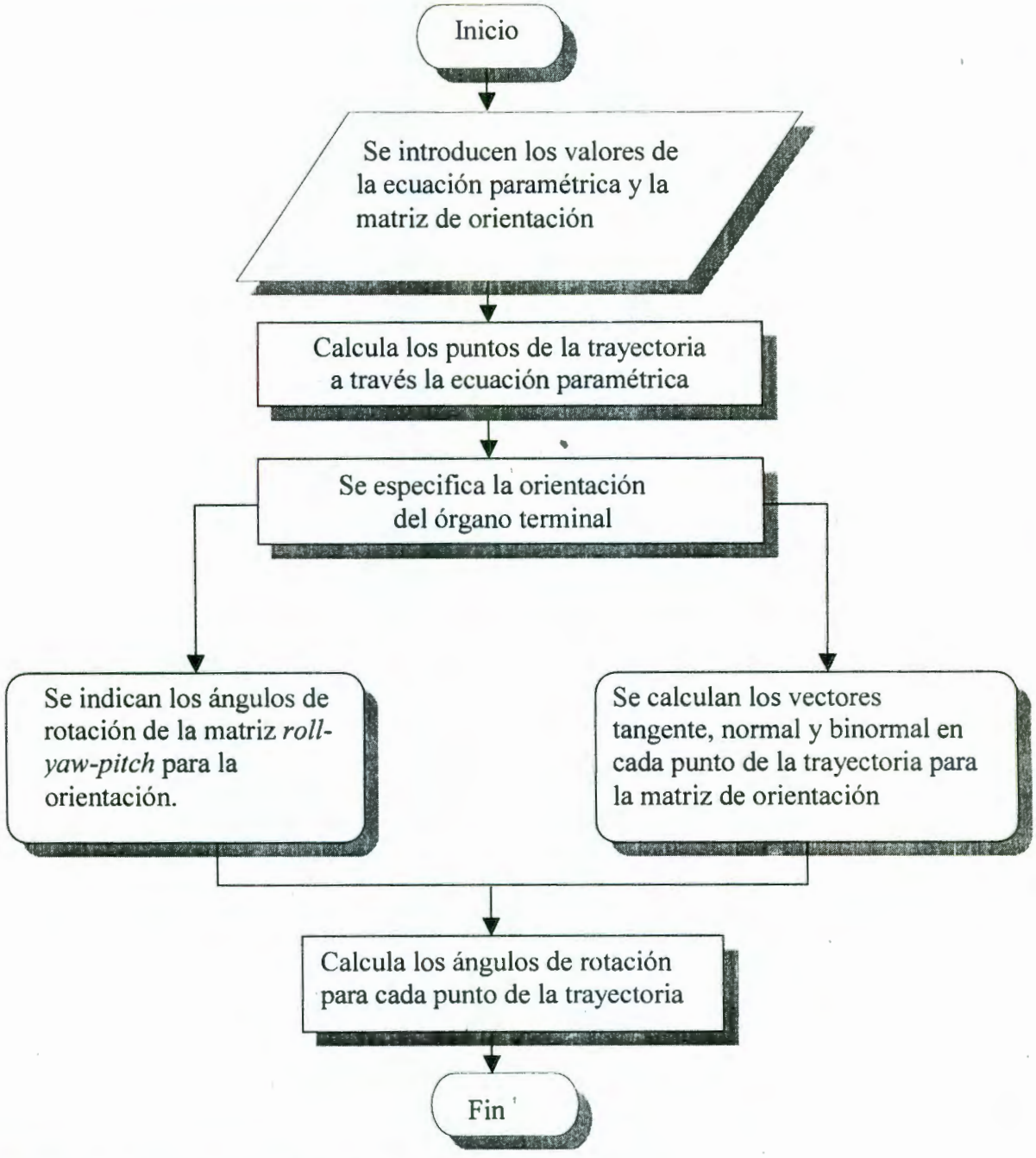

Figura 23. Diagrama de flujo de las trayectorias continuas.

El programa de simulación presenta el módulo de las trayectorias continuas, figura 23. Este tipo de planeación de trayectorias es útil para obtener los vectores de posición en cada punto del recorrido mediante una ecuación paramétrica y permite variar estos parámetros arbitrariamente. La orientación que tendrá el órgano terminal se determina indicando los ángulos de rotación para la matriz *roll-yaw-pitch* o también, obteniendo los vectores tangente, normal y binormal a cada punto de la trayectoria.

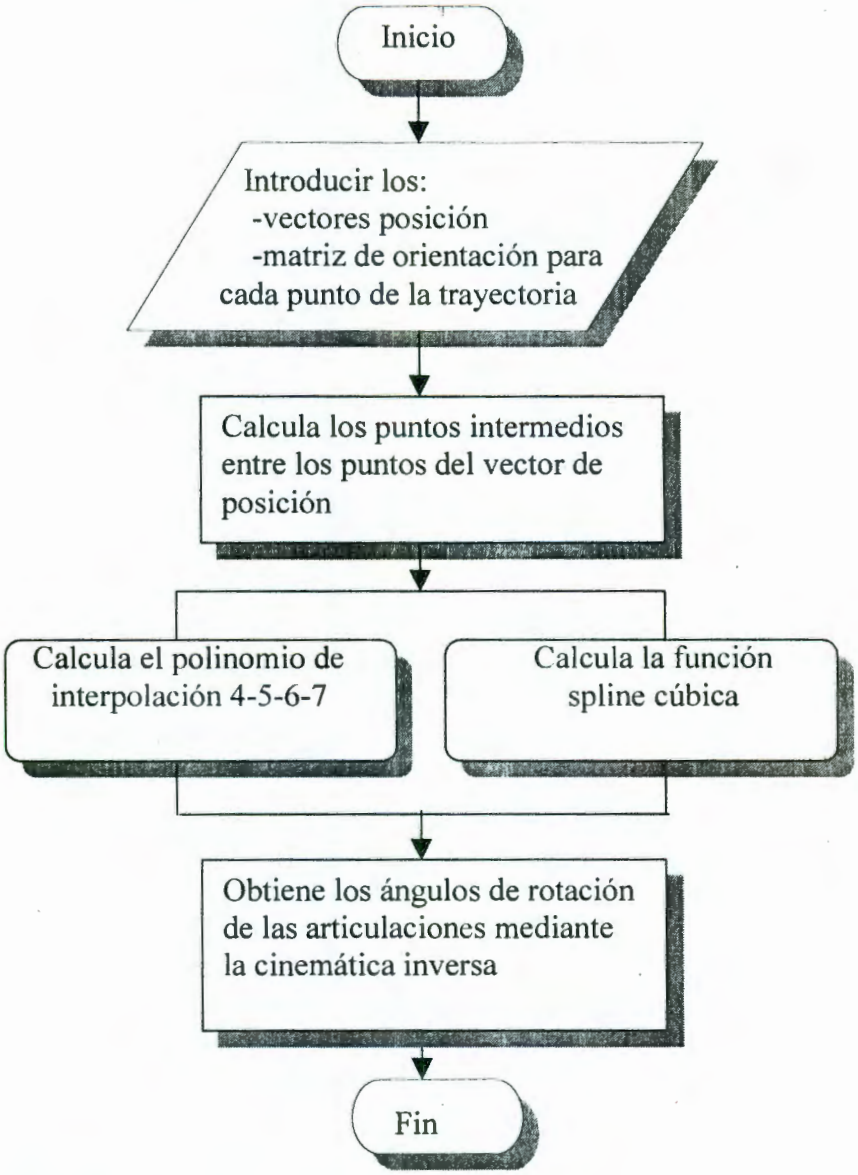

Figura 24. Diagrama de flujo de las trayectorias punto a punto.

Para el cálculo de las trayectorias punto a punto el simulador elige cualquiera de estos dos métodos: polinomio de interpolación 4-5-6-7 o la función spline cúbica. Los diagramas de las figuras 25 y 26 muestran estos métodos respectivamente. La descripción detallada se presenta en la sección 2.3.1.

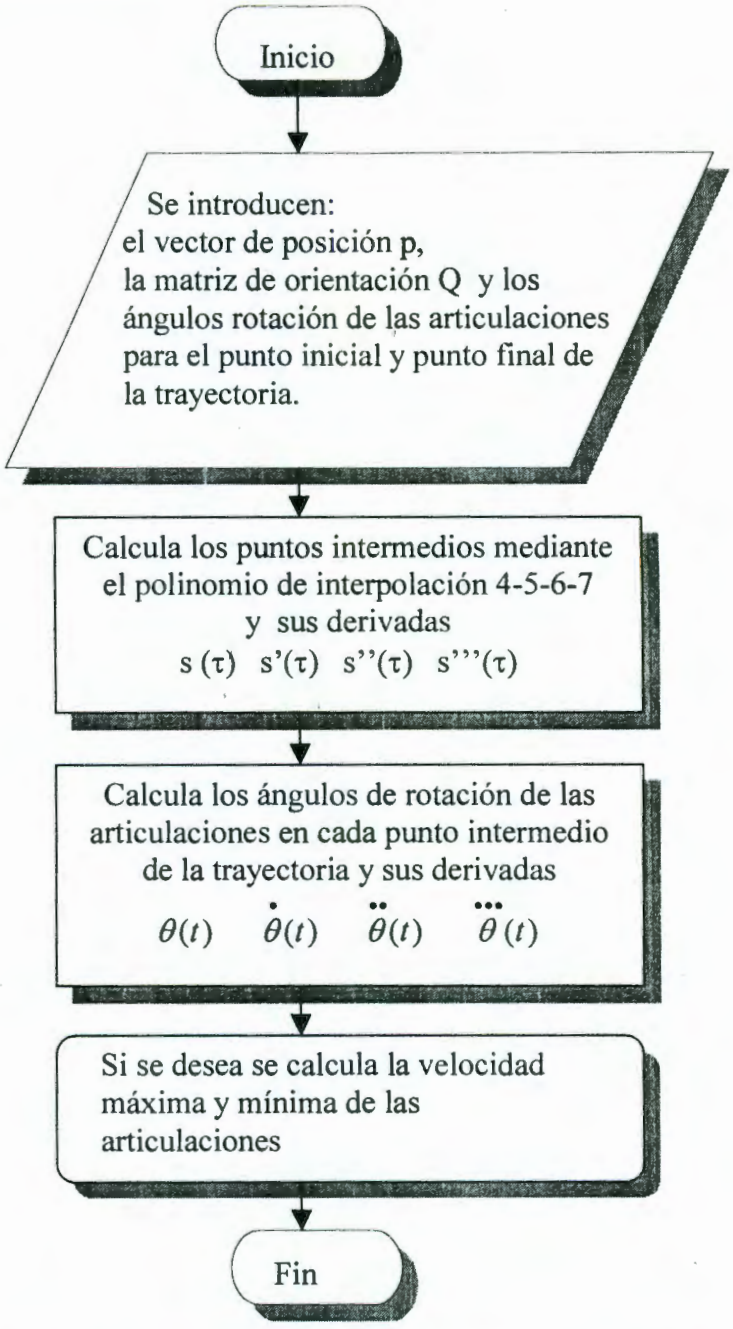

Figura 25. Diagrama de flujo de la interpolación 4-5-6-7.

La interpolación 4-5-6-7 se utiliza en este programa para calcular los ángulos de rotación de las articulaciones que conectan dos puntos. Se programa el polinomio de interpolación y sus derivadas. Al satisfacer las condiciones la tercera derivada elimina los movimientos abruptos al inicio y al témino de la trayectoria. En la sección 4.5.1 presenta las gráficas de las ecuaciones de interpolación y sus derivadas. Las splines cúbicas producen varias ecuaciones de conexión de un punto a otro de la trayectoria. El algoritmo para el cálculo de esta splines cúbicas se presenta en la figura 26.

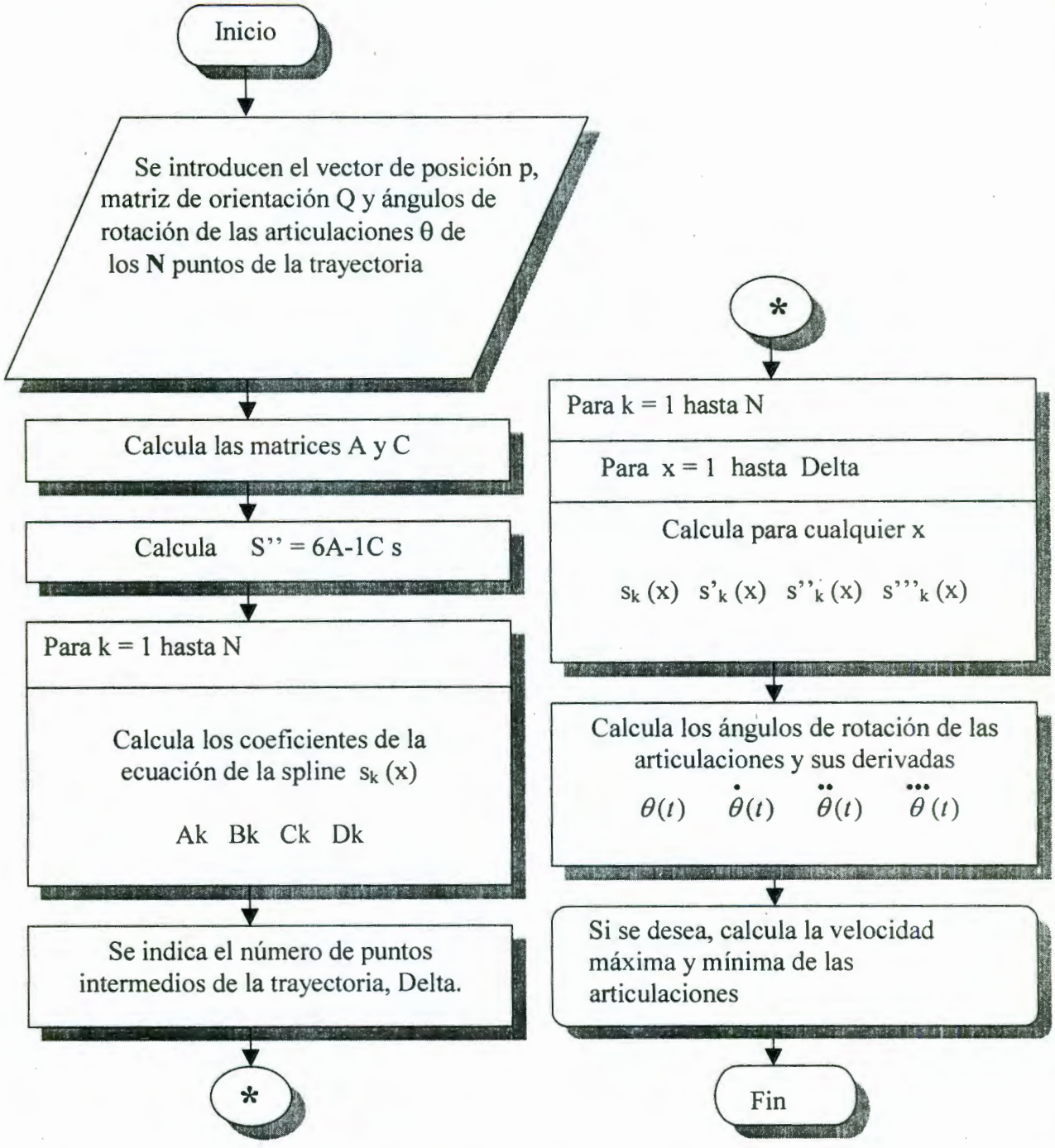

Figura 26. Diagrama de flujo de las splines cúbicas

Cada uno de los algoritmos y métodos utilizados en el programa de simulación se expresaron en estos diagramas de bloques y diagramas de flujo. A través de los cuales se visualiza cada función del programa que se describe a detalle en la sección 2.3.1, facilitando el entendimiento y desarrollo de los módulos del programa de simulación.

Capítulo 4

Aplicación: Simulación por computadora

Este programa está diseñado de manera tal que simula los movimientos de cualquier manipulador desacoplado de seis articulaciones. En la interfase, el usuario puede crear un robot con parámetros específicos a través de una serie de pasos. Este programa de simulación se evaluó utilizando las características de configuración y los parámetros de Denavit-Hartenberg del robot CLOOS modelo Romat 56, que se encuentra en el laboratorio de Mecatrónica de la Universidad Autónoma de Querétaro. A continuación se presentan las características del robot y los cálculos que se requirieron para la cinemática de posición, la cinemática de velocidad y la planeación de trayectorias.

4.1 Características del manipulador CLOOS modelo Romat 56

El robot CLOOS tiene una superficie de 360 x 480 mm con un peso aproximado de 110 kg y una carga nominal de 5kg máximo.

El intervalo de trabajo de cada uno de los eslabones es de: (López, G. 2001)

Eslabón	Intervalo (grados)	Alcance (grados)
1	328	−163.0 a 165.0
2	260	−41.2 a 218.8
3	280	−52.2 a 209.4
4	333	−166.5 a 166.5
5	197	−98.5 a 98.5
6	370	−185.0 a 185.0

Los intervalos que se indican en el programa de simulación limitan los movimientos para cada eslabón a sus alcances físicos. Antes de presentar la solución de la cinemática directa e inversa, cinemática de velocidad y planeación de trayectorias se muestra la información de como opera el programa de simulación.

4.2 Simulación

En este tema de tesis se construyó un simulador gráfico para el seguimiento de trayectorias para visualizar los movimientos del robot antes de ejecutarlos. Esto es de gran utilidad pues así se pueden prever posibles colisiones. Actualmente existe *software* que facilita el desarrollo de gráficas en tres dimensiones y que puede usarse para determinar, en simulación, la posición de cada eslabón conforme el robot sigue una trayectoria. Una vez que se haya verificado, en simulación, que el manipulador sigue la trayectoria de manera satisfactoria puede generarse el código necesario para que el robot realice los movimientos en tiempo real.

En este proyecto se utilizó el *software* 3D Studio Max ver. 2.5 para construir el simulador porque facilita la creación de imágenes y la programación de animaciones. El *software* descrito permite observar diferentes vistas del robot: 1) vista superior, 2) vista frontal, 3) vista lateral izquierda , 4) vista lateral derecha y 5) perspectiva. La figura 27 muestra las diferentes vistas que presenta el programa y las persianas de interacción. Así como también, la vista en perspectiva el robot en la posición inicial.

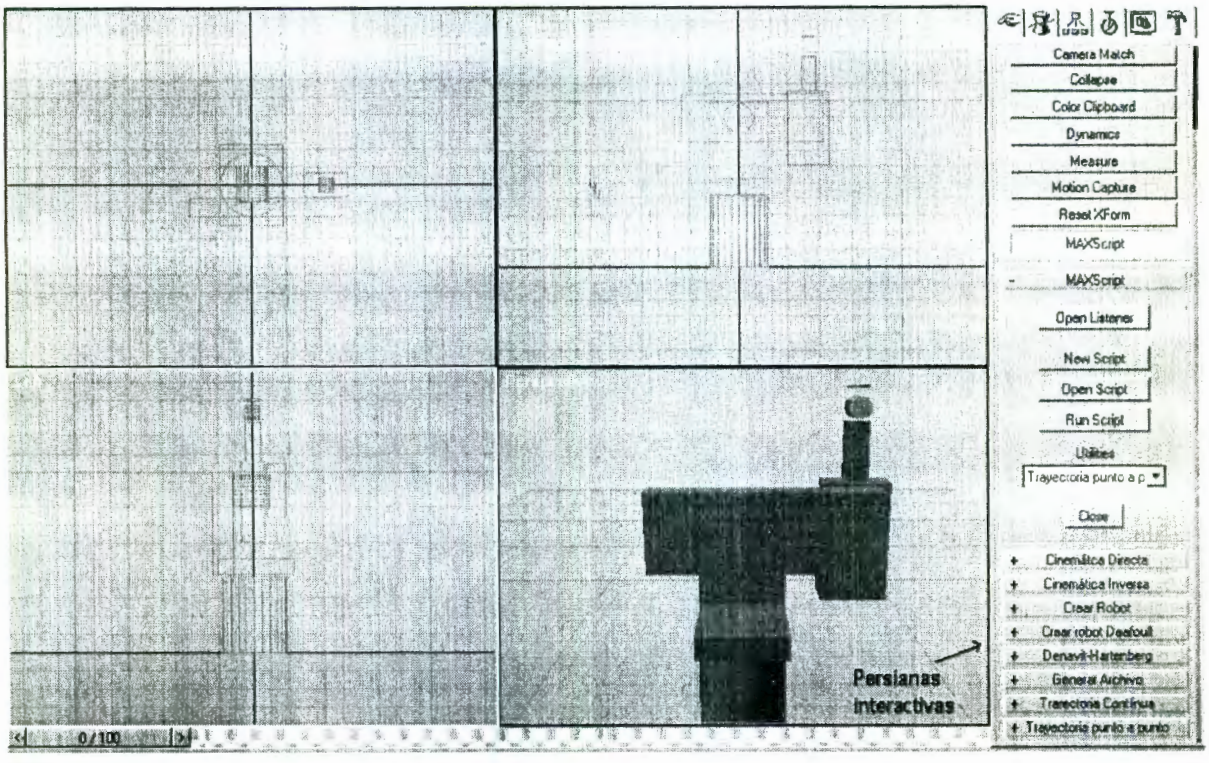

Figura 27. Pantalla general del simulador.

En la figura 28 muestra la persiana "Cinemática directa" en estado "abierta" para que el usuario pueda interactuar con el programa. La figura 29 muestra el robot en la que los eslabones están rotados a diferentes ángulos. Cuando se desee que el órgano terminal del robot siga una trayectoria (coordenadas cartesianas), se utilizará el modelo cinemático inverso para determinar la trayectoria correspondiente a ser seguida por cada articulación (ángulos de giro de cada eje). Con esto se podrá realizar una animación del movimiento del robot, pudiendo determinar si dicha trayectoria produce el movimiento deseado o si es segura en el sentido de que no haya colisiones

del robot con el entorno que lo rodea.

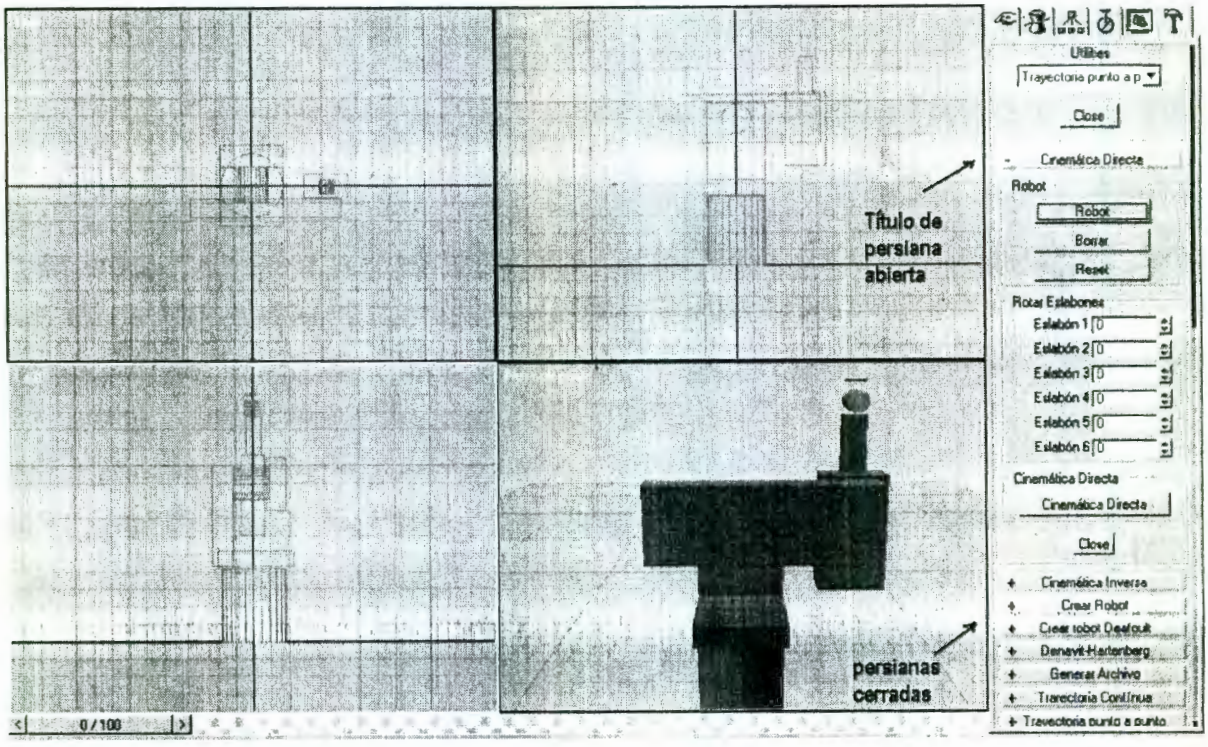

Figura 28. Presentación de la persiana de la cinemática directa.

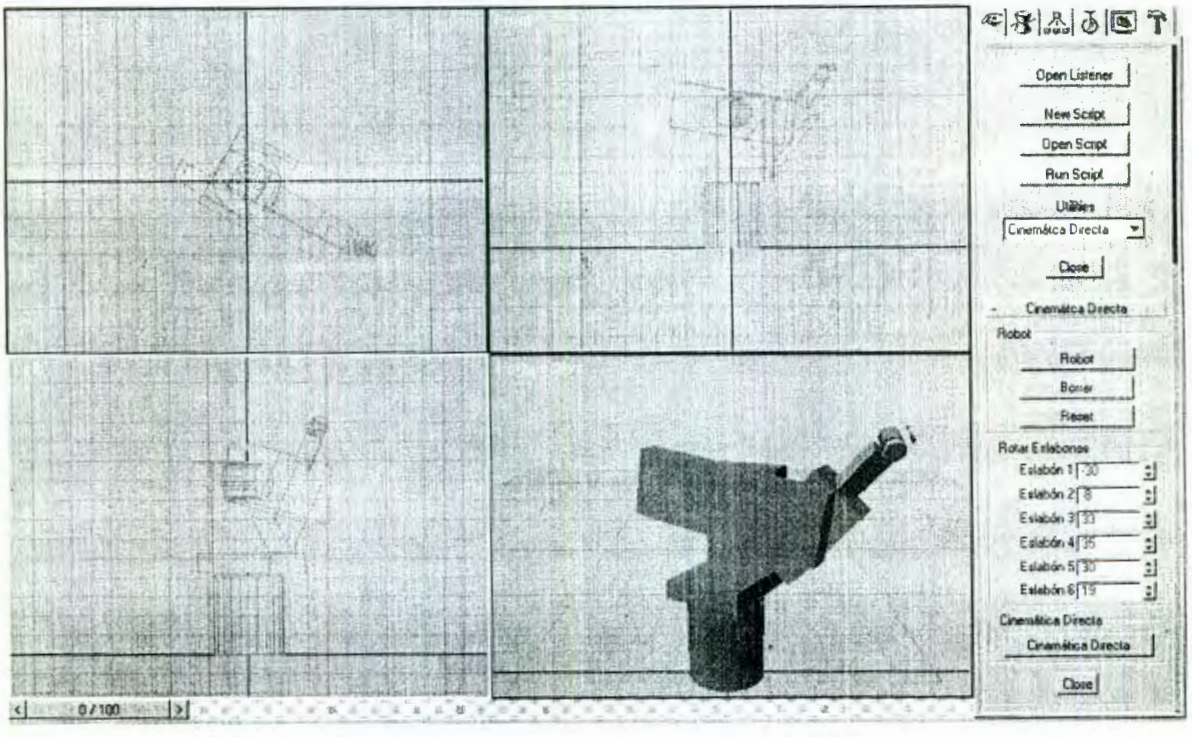

Figura 29. Imagen del robot con todos sus eslabones girados a diferentes ángulos.

Para obtener la imagen de un robot se debe especificar los parámetros y dimensiones de cada eslabón en la persiana de "Crear Robot". Esto se consigue definiendo el pivote o eje de giro

de cada eslabón y la vinculación entre cada uno de ellos. Este proceso se repite para todos los eslabones permitiendo transmitir el movimiento a todos los eslabones (a manera de herencia). Este *software* incluye una base de tiempo indispensable para la creación de animaciones. De este modo, si se desea animar un movimiento coordinado del robot, se debe especificar: 1) la configuración del robot, 2) los parámetros de Denavit-Hartenberg, 3) los puntos y orientaciones de la trayectoria, 4) la planeación de trayectorias continuas o punto a punto, y 5) el tiempo en que realizará la animación. El programa de simulación consta de diversas persianas con direrentes funciones como, cinemática directa, cinemática inversa, planeación de trayectorias continuas y planeación de trayectorias punto a punto.

4.3 Cinemática directa

El modelo esquemático del manipulador se obtiene para el cálculo de la cinemática y la planeación de trayectorias. (vease anexo 2)

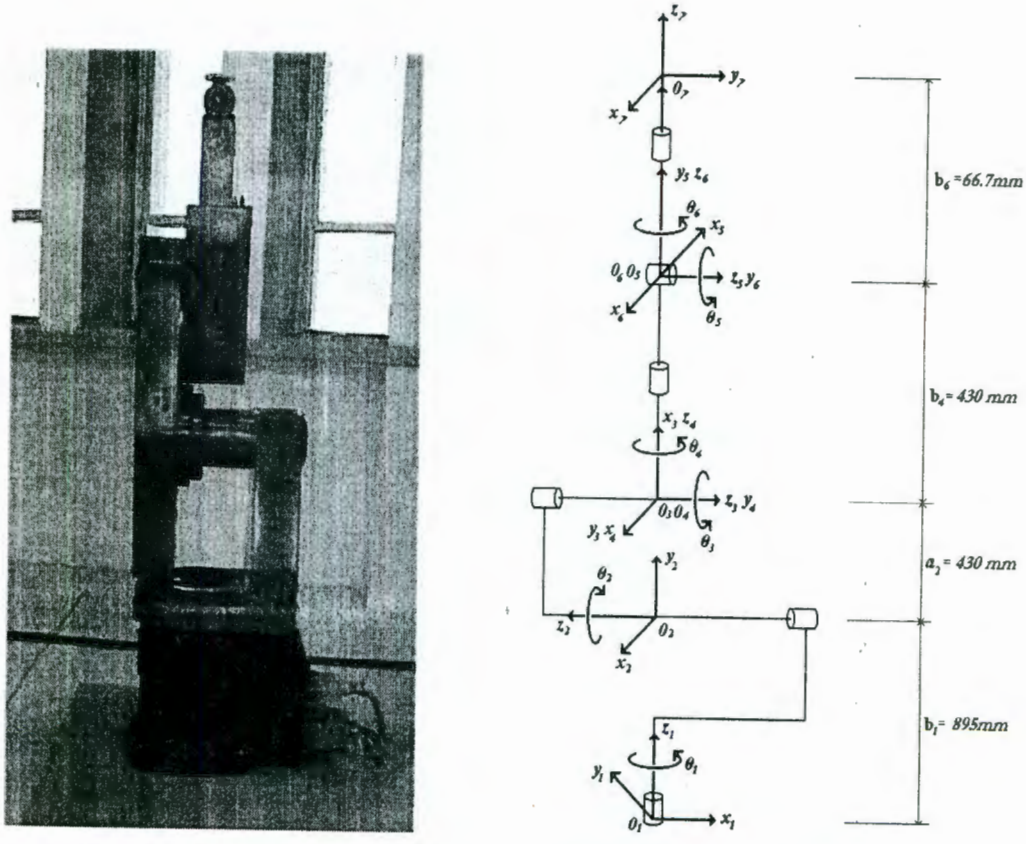

Figura 30. Modelo esquemático del robot CLOOS tipo Romat 56.

Los parámetros de Denavit-Hartenberg para el robot CLOOS son:

Eslabón	a_i(mm)	b_i(mm)	α_i	θ_i
1	0	895	90°	θ_1
2	430	0	180°	θ_2
3	0	0	90°	θ_3
4	0	430	90°	θ_4
5	0	0	90°	θ_5
6	0	66.7	0	θ_6

Las matrices de transformación, $\mathbf{T}_i$, para $i = 1, ..., 6$ son:

$$\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ \sin\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 1 & 0 & b_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}_2 = \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 & a_2\cos\theta_2 \\ \sin\theta_2 & -\cos\theta_2 & 0 & a_2\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_3 = \begin{bmatrix} \cos\theta_3 & 0 & \sin\theta_3 & 0 \\ \sin\theta_3 & 0 & -\cos\theta_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}_4 = \begin{bmatrix} \cos\theta_4 & 0 & \sin\theta_4 & 0 \\ \sin\theta_4 & 0 & -\cos\theta_4 & 0 \\ 0 & 1 & 0 & b_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_5 = \begin{bmatrix} \cos\theta_5 & 0 & \sin\theta_5 & 0 \\ \sin\theta_5 & 0 & -\cos\theta_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}_6 = \begin{bmatrix} \cos\theta_6 & \sin\theta_6 & 0 & 0 \\ \sin\theta_6 & -\cos\theta_6 & 0 & 0 \\ 0 & 0 & 1 & b_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La multiplicación de estas matrices produce la matriz de transformación final que contiene la posición y orientación del órgano terminal con respecto a la base. La matriz de transformación $\mathbf{T}_1^6 = \mathbf{T}_1\mathbf{T}_2\mathbf{T}_3\mathbf{T}_4\mathbf{T}_5\mathbf{T}_6$ es la siguiente:

$$T_1^6 = \begin{bmatrix} q_{11} & q_{21} & q_{31} & \mathbf{a}_1 \\ q_{21} & q_{22} & q_{32} & \mathbf{a}_2 \\ q_{31} & q_{32} & q_{33} & \mathbf{a}_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

en donde la matriz de orientacion Q es

$$\mathbf{Q} = \begin{bmatrix} q_{11} & q_{21} & q_{31} \\ q_{21} & q_{22} & q_{32} \\ q_{31} & q_{32} & q_{33} \end{bmatrix}$$

donde,

$$q_{11} = c_1c_2c_3c_4c_5c_6 + s_2s_3c_1c_4c_5c_6 - s_1s_4c_5c_6 + s_3s_5c_1c_2c_6 - s_2s_5c_1c_3c_6 + s_4s_6c_1c_2c_3 + +s_2s_3s_4s_6c_1 + s_1s_6c$$

$$q_{21} = s_1c_2c_3c_4c_5c_6 + s_1s_2s_3c_4c_5c_6 + s_4c_1c_5c_6 + s_1s_3s_5c_2c_6 - s_1s_2s_5c_3c_6 + s_1s_4s_6c_2c_3 + +s_1s_2s_3s_4s_6 - s_6c_1c$$

$$q_{31} = s_2c_3c_4c_5c_6 - s_3c_2c_4c_5c_6 + s_2s_3s_5c_6 + s_5c_2c_3c_6 + s_2s_4s_6c_3 - s_3s_4s_6c_2$$

$$q_{12} = -s_6c_1c_2c_3c_4c_5 - s_2s_3s_6c_1c_4c_5 + s_1s_4s_6c_5 - s_3s_5s_6c_1c_2 + s_2s_5s_6c_1c_3 + s_4c_1c_2c_3c_6 + s_2s_3s_4c_1c_6 + s_1c_4c$$

$$q_{22} = -s_1s_6c_2c_3c_4c_5 - s_1s_2s_3s_6c_4c_5 - s_4s_6c_1c_5 - s_1s_3s_5s_6c_2 + s_1s_2s_5s_6c_3 + s_1s_4c_2c_3c_6 + s_1s_2s_3s_4c_6 - c_1c_4$$

$$q_{32} = -s_2s_6c_3c_4c_5 + s_3s_6c_2c_4c_5 - s_2s_3s_5s_6 - s_5s_6c_2c_3 + s_2s_4c_3c_6 - s_3s_4c_2c_6$$

$$q_{13} = s_5c_1c_2c_3c_4 + s_2s_3s_5c_1c_4 - s_1s_4s_5 - s_3c_1c_2c_5 + s_2c_1c_3c_5$$

$$q_{23} = s_1s_5c_2c_3c_4 + s_1s_2s_3s_5c_4 + s_4s_5c_1 - s_1s_3c_2c_5 + s_1s_2c_3c_5$$

$$q_{33} = s_2s_5c_3c_4 - s_3s_5c_2c_4 - s_2s_3c_5 - c_2c_3c_5$$

El vector resultante $\mathbf{a}_i$ es:

$$\mathbf{a}_1 = b_6s_5c_1c_2c_3c_4 + b_6s_2s_3s_5c_1c_4 - b_6s_1s_4s_5 - b_6s_3c_1c_2c_5 + b_6s_2c_1c_3c_5 + b_4s_3c_1c_2 - b_4s_2c_1c_3 + a_2c_1c_2$$

$$\mathbf{a}_2 = b_6s_1s_5c_2c_3c_4 + b_6s_1s_2s_3s_5c_4 + b_6s_4s_5c_1 - b_6s_1s_3c_2c_5 + b_6s_1s_2c_3c_5 + b_4s_1s_3c_2 - b_4s_1s_2c_3 + a_2s_1c_2$$

$$\mathbf{a}_3 = b_6s_2s_5c_3c_4 - b_6s_3s_5c_2c_4 - b_6s_2s_3c_5 - b_6c_2c_3c_5 + b_4s_2s_3 + b_4c_2c_3 + a_2s_2$$

4.4 Cinemática inversa

La cinemática inversa se calculó utilizando el algoritmo de la sección 2.1.2. La cinemática inversa obtiene los ángulos de rotación para una configuración inicial. Como se mencionó en la obtención de los parámetros de D-H, sección 2.1.1, el ángulo formado entre los ejes X_i y X_{i+1} medido sobre la dirección positiva de Z_i es el ángulo θ_i. Para especificar la configuración inicial del manipulador en donde todos los ejes X_i estén en la misma dirección, se rota cada eslabón hasta conseguirlo. El robot CLOOS requirió de las rotaciones que se muestran en las figuras 30 a la 36 para posicionarlo en una configuración inicial.

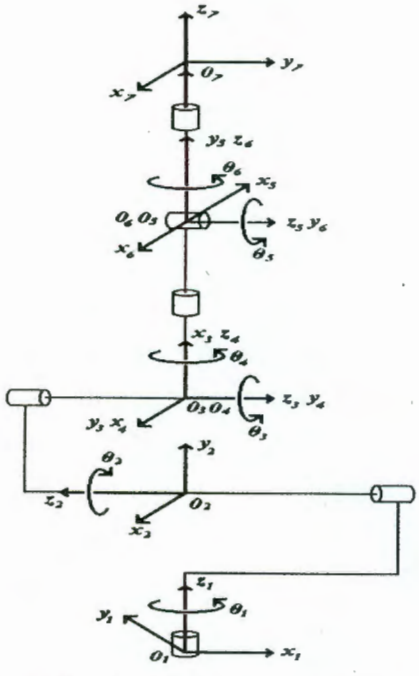

Figura 31. Modelo esquemático 1.

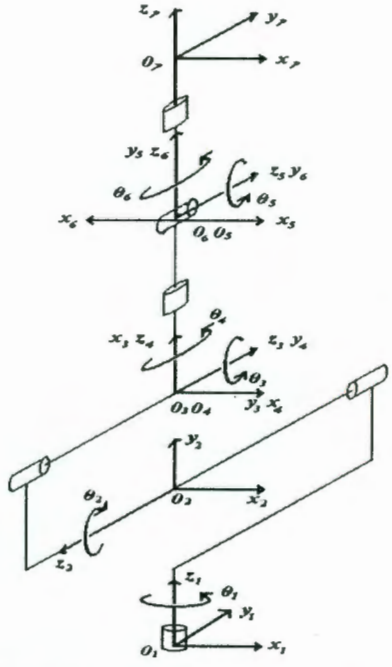

Figura 32. Modelo esquemático 2. Rotación del ángulo $\theta_1 = -90$.

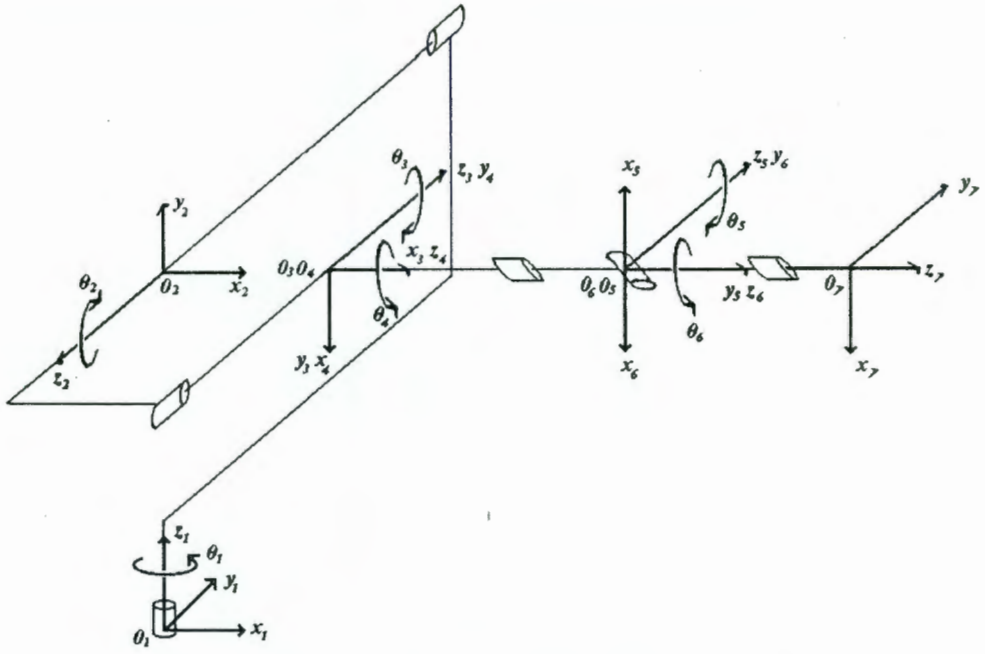

Figura 33. Modelo esquemático 3. Rotación del ángulo $\theta_2 = -90$.

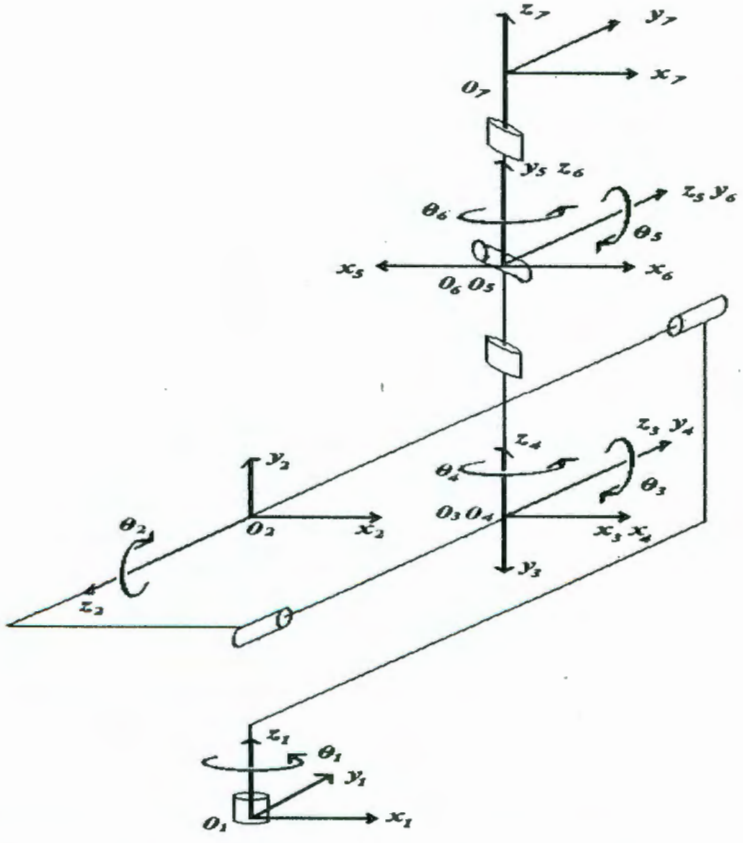

Figura 34. Modelo esquemático 4. Rotación del ángulo $\theta_3 = -90$.

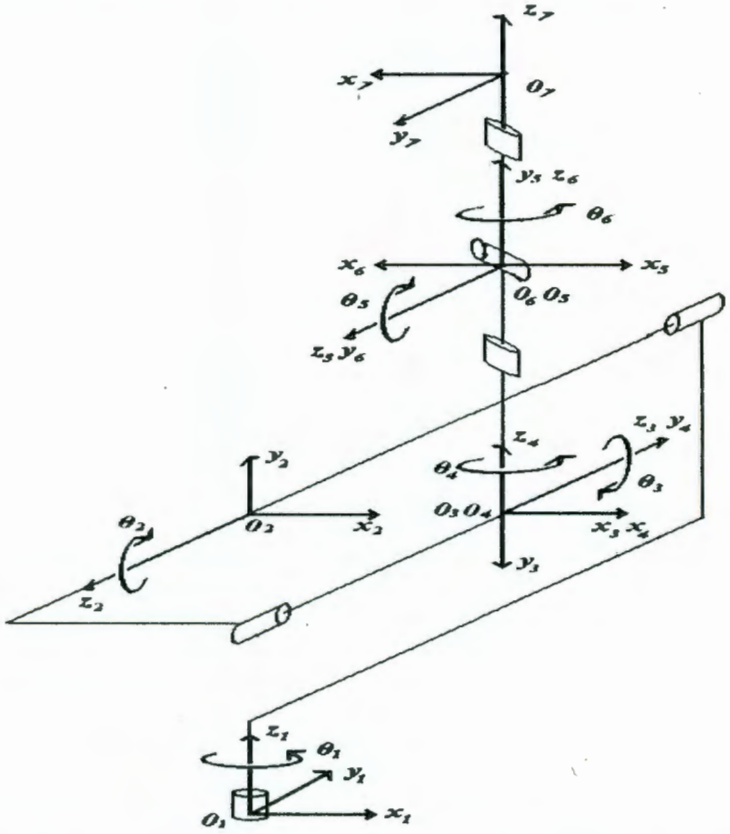

Figura 35. Modelo esquemático 5. Rotación del ángulo $\theta_4 = 180$.

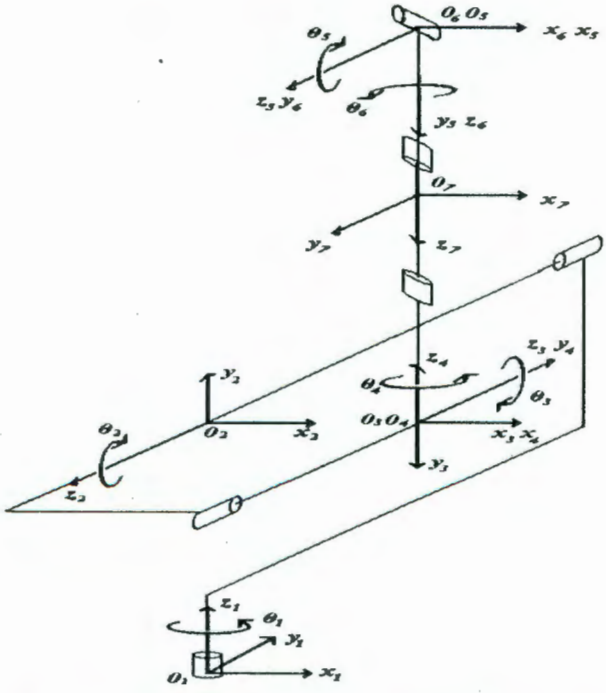

Figura 36. Modelo esquemático 6. Rotación del ángulo $\theta_5 = -180$.

El modelo esquemático 6 de la figura 36, tiene todos los ejes $X's$ en la misma dirección pero esta configuración no es realizable porque colisionaría el órgano terminal con el eslabón 5. Como consecuencia, se elige como configuración inicial el modelo esquemático 5 de la figura 35 con un desfase en el ángulo θ_5 de -180 grados. Por lo tanto, para el cálculo del ángulo θ_5 se le resta esta cantidad.

Es importante comentar que el robot puede adquirir una configuración inicial deseada. Esta configuracion puede tener los ejes $X's$ en diferentes direcciones. El ángulo de rotación formado por estos ejes se indica en la persiana "Anexos crear robot" del programa de simulación, que serán consideradas en los cálculos de la cinemática inversa.

El cálculo de la cinemática inversa utiliza el algoritmo presentado en la sección 2.1.2. El posicionamiento del robot para alcanzar un punto con una cierta orientación puede tener 8 posibles soluciones. La figura 37 muestra la interfase del programa para obtener la cinemática inversa para un punto en el espacio con una determinada orientación.

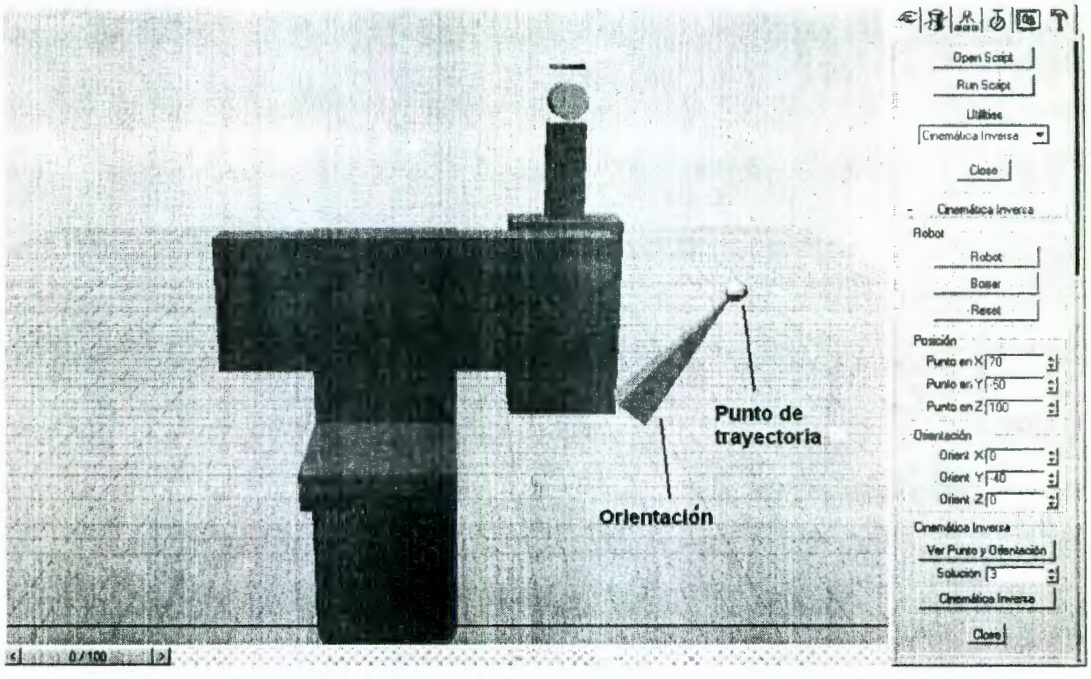

Figura 37. Interfase para el cálculo de la cinemática inversa.

Cada figura, puntos de las trayectorias, las orientaciones para cada punto y las persianas de interacción, así como, los cálculos de la cinemática de posición, de velocidad y la planeación de trayectorias se desarrolló mediante instrucciones de código. Estas instrucciones se programaron en MaxScript del 3D Studio Max ver.2.5; por tanto se tiene control sobre la graficación, rotación y animación de los objetos o eslabones para la creación del programa de simulación. La figura 38 muestra la primera solución para la cinemática inversa.

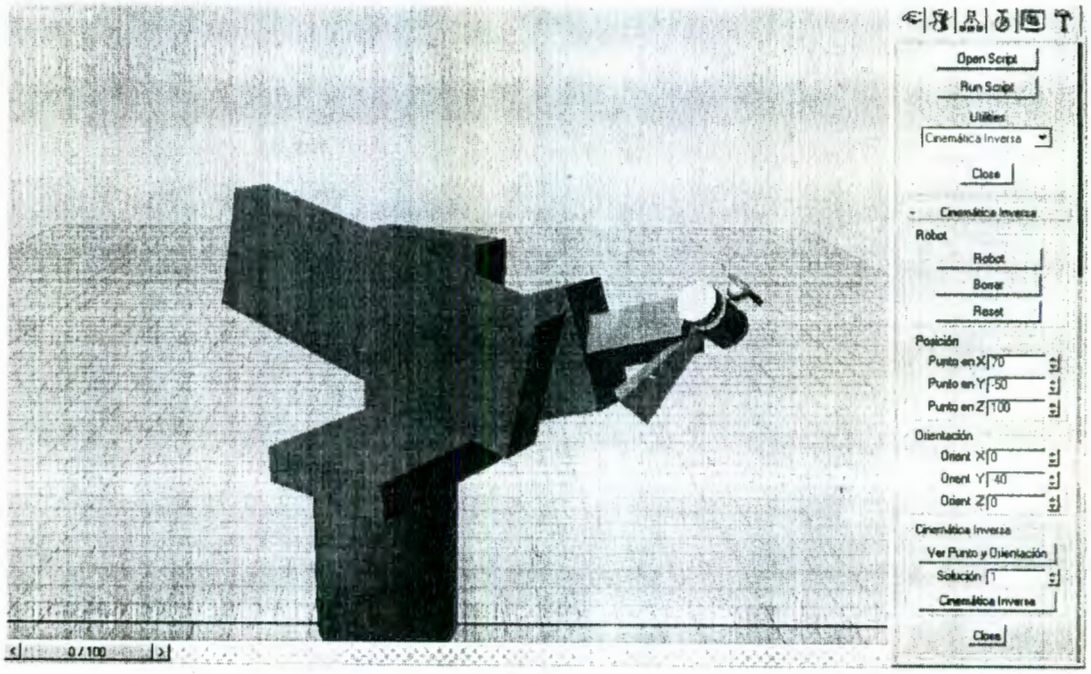

Figura 38. Cinemática inversa para un punto (primera solución).

En la figura 39 se muestran los movimientos de cada eslabón para posicionar el órgano terminal en un cierto punto en el espacio con una cierta orientación. Esta figura muestra la segunda solución para la cinemática inversa.

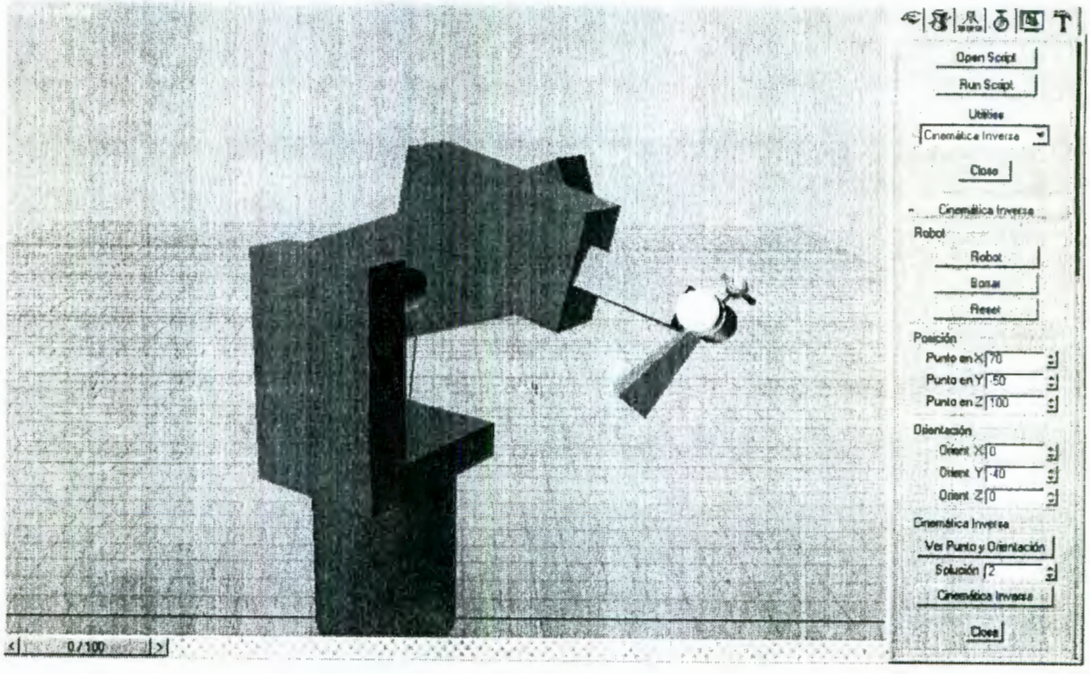

Figura 39. Cinemática inversa para un punto (segunda solución).

Este simulador es de gran utilidad para visualizar las diferentes opciones de posicionamiento para que el manipulador alcance un punto.

4.5 Cinemática de velocidad

Para la cinemática de velocidad de manipuladores desacoplados el programa los separa en cinemática directa de velocidad y cinemática inversa de velocidad. El jacobiano se utiliza para obtener estos cálculos. El procedimiento que se sigue está descrito en la sección 2.2 para la obtención del jacobiano y se aplica al robot CLOOS desacoplado,

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_{11} & \mathbf{J}_{12} \\ \mathbf{J}_{21} & 0 \end{bmatrix}$$

donde cada uno de sus componentes $\mathbf{J}_{ij}$ es,

$$\begin{aligned} \mathbf{J}_{11} &= [\ \mathbf{e}_1 \quad \mathbf{e}_2 \quad \mathbf{e}_3\ ] \\ \mathbf{J}_{12} &= [\ \mathbf{e}_4 \quad \mathbf{e}_5 \quad \mathbf{e}_6\ ] \\ \mathbf{J}_{21} &= [\ \mathbf{e}_1 \times \mathbf{r}_1 \quad \mathbf{e}_2 \times \mathbf{r}_2 \quad \mathbf{e}_3 \times \mathbf{r}_3\ ] \end{aligned}$$

Los vectores e_i son:

$$\mathbf{e}_1 = \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix} \quad \mathbf{e}_2 = \begin{bmatrix} -s_1 \\ c_1 \\ 0 \end{bmatrix} \quad \mathbf{e}_3 = \begin{bmatrix} c_1c_2s_3 - c_1s_2c_3 \\ s_1c_2s_3 - s_1s_2c_3 \\ s_2s_3 + c_2c_3 \end{bmatrix}$$

$$\mathbf{e}_4 = \begin{bmatrix} c_1c_2c_3s_4 + c_1s_2s_3s_4 + s_1c_4 \\ s_1c_2c_3s_4 + s_1s_2s_3s_4 - c_1c_4 \\ s_2c_3s_4 - c_2s_3s_4 \end{bmatrix}$$

$$\mathbf{e}_5 = \begin{bmatrix} c_1c_2c_3c_4s_5 + c_1s_2s_3c_4s_5 - s_1s_4s_5 - c_1c_2s_3c_5 + c_1s_2c_3c_5 \\ s_1c_2c_3c_4s_5 + s_1s_2s_3c_4s_5 + c_1s_4s_5 - s_1c_2s_3c_5 + s_1s_2c_3c_5 \\ s_2c_3c_4s_5 - c_2s_3c_4s_5 + s_2s_3c_5 - c_2c_3c_5 \end{bmatrix}$$

$$\mathbf{e}_6 = \begin{bmatrix} c_1c_2c_3c_4s_5 + c_1s_2s_3c_4s_5 - s_1s_4s_5 - c_1c_2s_3c_5 + c_1s_2c_3c_5 \\ s_1c_2c_3c_4s_5 + s_1s_2s_3c_4s_5 + c_1s_4s_5 - s_1c_2s_3c_5 + s_1s_2c_3c_5 \\ s_2c_3c_4s_5 - c_2s_3c_4s_5 + s_2s_3c_5 - c_2c_3c_5 \end{bmatrix}$$

El vector r_i se forma por el origen O_i hacia el centro de la muñeca c, como se muestra en la figura 40. Una vez calculados los vectores r_1, r_2 y r_3, se calcula el producto cruz $\mathbf{e}_i \times r_i$.

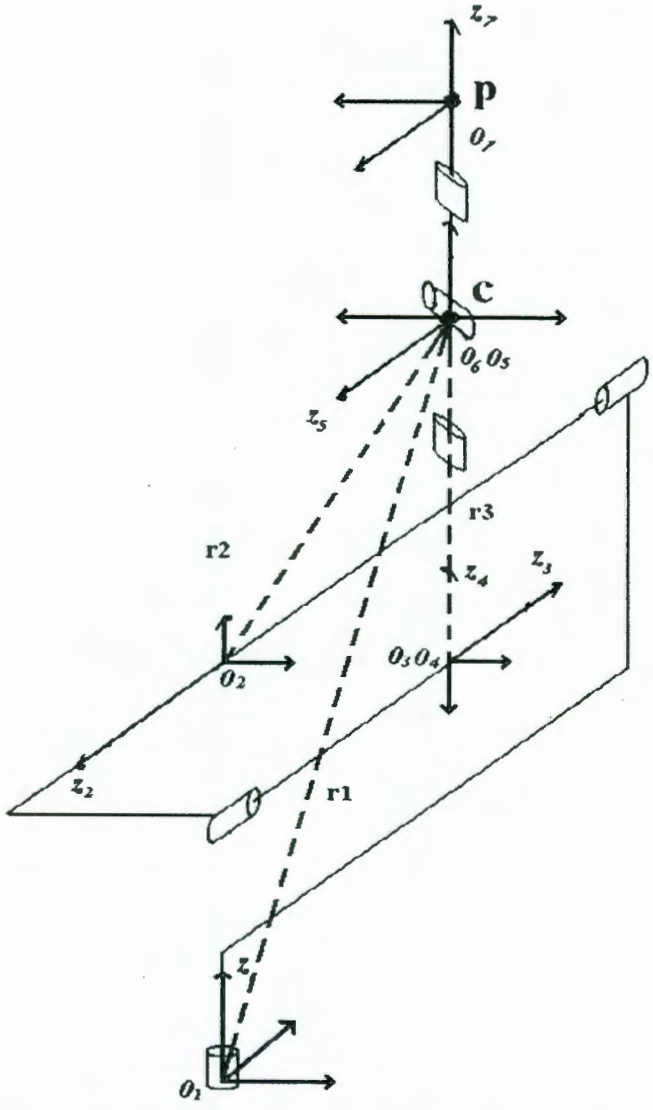

Figura 40. Representación de los vectores r_i de un manipulador desacoplado.

La singularidad se presenta cuando el volumen formado por los vectores $\mathbf{e}_4$, $\mathbf{e}_5$ y $\mathbf{e}_6$ se convierte en un plano, es decir, $\mathbf{e}_4 \times \mathbf{e}_5 \cdot \mathbf{e}_6 = 0$.

La figura 41 muestra la gráfica de la posición y velocidad del órgano terminal

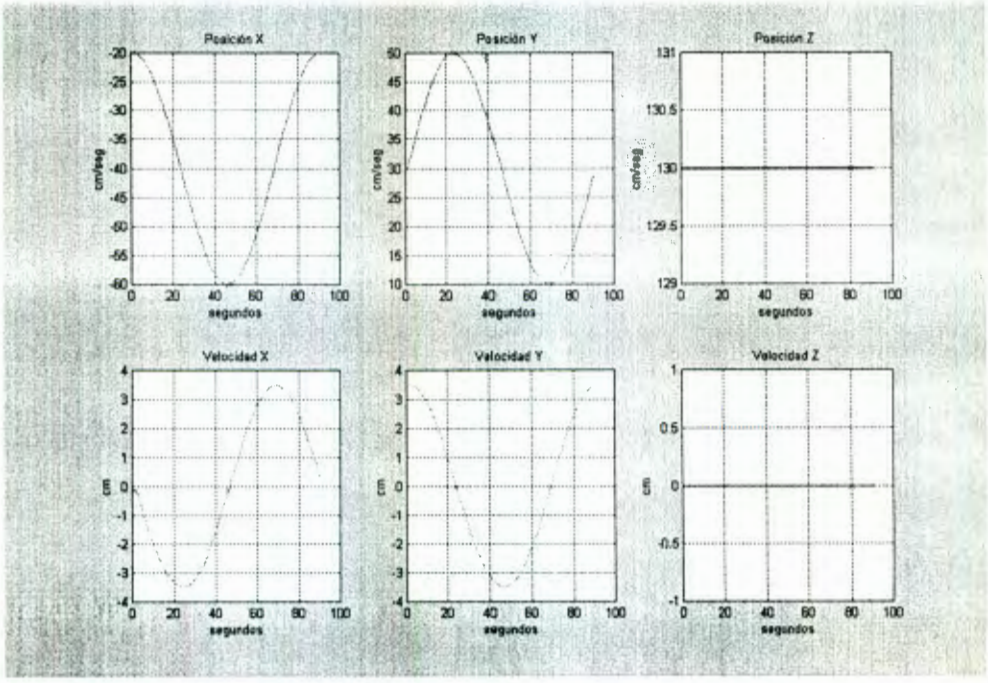

Figura 41. Gráfica de la posición y velocidad del órgano terminal.

La forma de la trayectoria es una circunferencia para la cual se grafican los ángulos de rotación de cada articulación. La gráfica de la posición del eje X tiene la forma de una señal cosenoidal y su derivada o velocidad se asemeja a una senoidal. La posición en el eje Y es de la forma de una senoidal, por lo tanto la velocidad es similar a una cosenoidal. El intervalo de tiempo entre un punto y otro de la trayectoria es de 2 segundos, por esta razón, las gráficas de posición y velocidad se muestran casi continuas. La figura 41 representa la cinemática directa de velocidad.

La figura 42 muestra la posición y velocidad de las articulaciones de los ángulos θ_1, θ_2 y θ_3. Para obtener estas gráficas se calculó la cinemática inversa de velocidad.

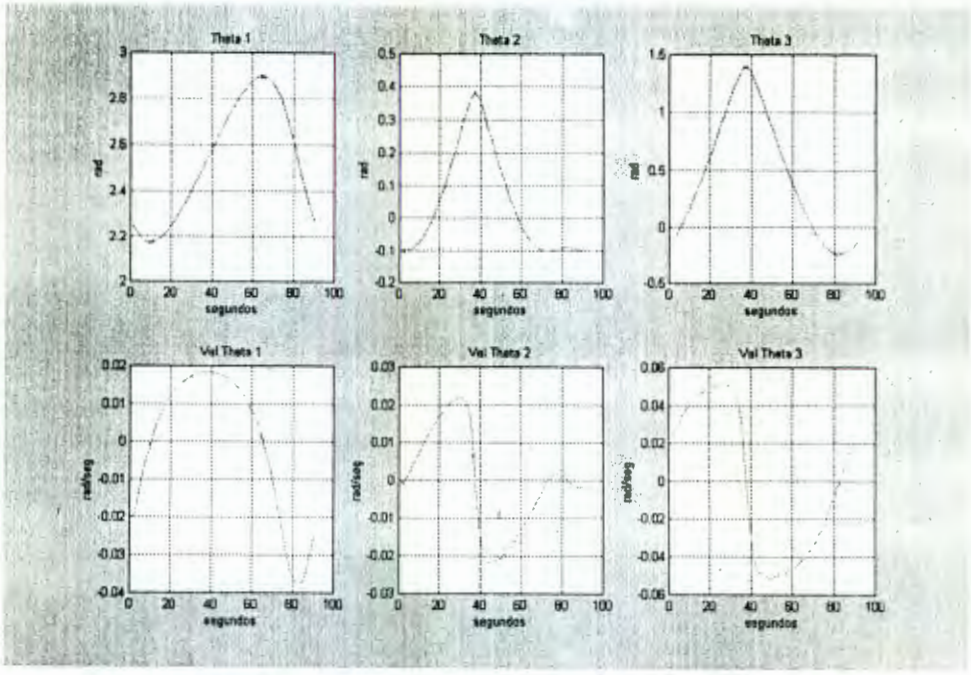

Figura 42. Gráfica de la posición y velocidad de los ángulos θ_1, θ_2 y θ_3.

La figura 43 muestra la posición y velocidad de los ángulos θ_4, θ_5 y θ_6.

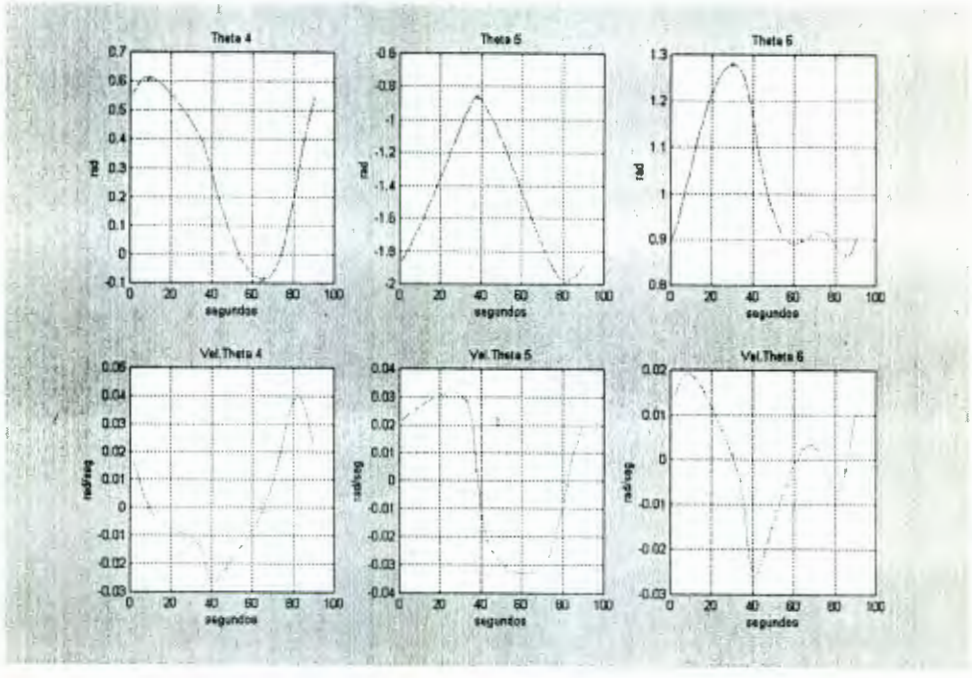

Figura 43. Gráfica de la posición y velocidad de los ángulos θ_4, θ_5 y θ_6.

La velocidad indica el perfil de movimiento de un punto a otro para cada una de las articulaciones. Las gráficas anteriores presentan la posición y velocidad para cada una de las articulaciones.

4.6 Planeación de trayectorias

El problema de planeación de trayectorias puede dividirse en:

* Trayectorias punto a punto.
* Trayectorias continuas.

4.6.1 Trayectorias punto a punto

Para el cálculo de las trayectorias punto a punto se utilizaron las interpolaciones 4-5-6-7 y las splines cúbicas.

Intepolación 4-5-6-7

Para el cálculo de la conexión de un punto a otro se utilizó la interpolación 4-5-6-7. La descripción del método se especificó en la sección 2.3.1. En la figura 44 se presenta la gráfica con la ecuación de interpolación y sus derivas. En esta gráfica se muestra que al inicio y al final de la trayectoria la tercera derivada desaparece, es decir, los movimientos abruptos, *jerk*, se eliminan.

El polinomio de interpolación 4-5-6-7 recibe como parámetros los siguientes datos:

$$P_{INICIAL} = [43, 0, 139.5] \qquad P_{FINAL} = [-37, -52, 60]$$

$$\mathbf{Q}_{INICIAL} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{Q}_{FINAL} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$\theta INICIAL = \begin{bmatrix} -179.8372 \\ 90.0002 \\ 0.0017 \\ 0 \\ 90.0014 \\ 179.8372 \end{bmatrix} \qquad \theta FINAL = \begin{bmatrix} 54.4774 \\ 161.3671 \\ 13.9318 \\ 180 \\ 177.4353 \\ 144.4774 \end{bmatrix}$$

con estos datos se calcula la interpolación generando la gráfica de la figura 44. En esta figura, la señal (1) representa la ecuación de interpolación, la señal (2) su derivada o velocidad, la señal (3) indica la aceleración y la señal (4) indica la tercera derivada o *jerk*. En la figura 45 se grafican los valores del ángulo θ_1 para conectar un punto con otro. La velocidad y aceleración son las señales (2) y (3) respectivamente. La tercera derivada que es la señal (4) muestra que los movimientos abruptos desaparecen al inicio y al final de la trayectoria.

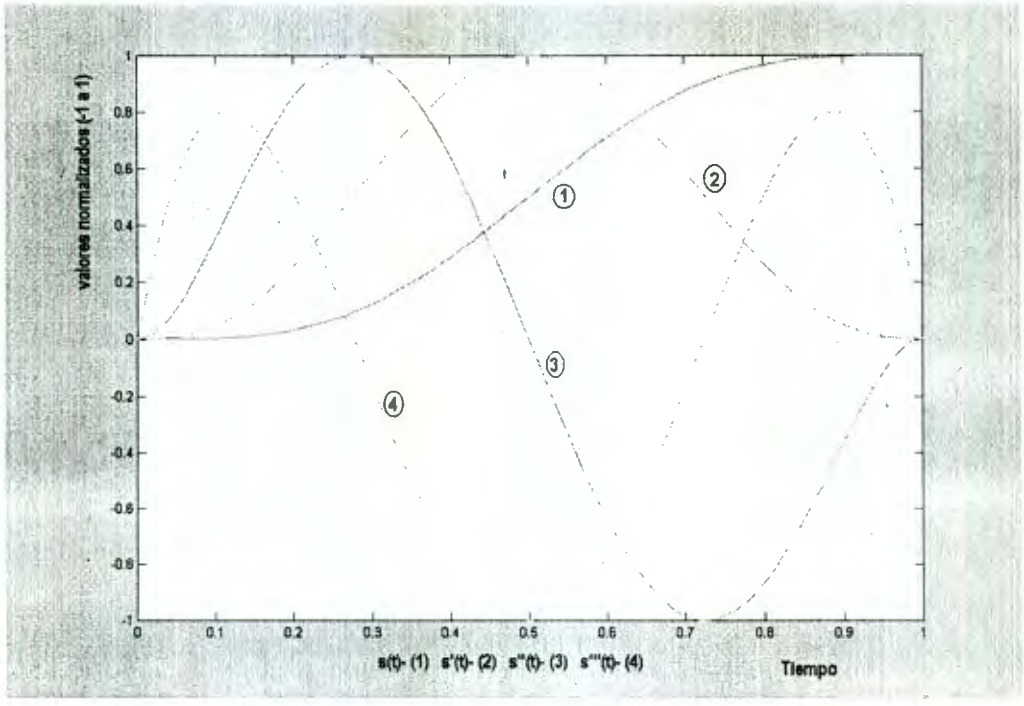

Figura 44. Gráfica de interpolación 4-5-6-7.

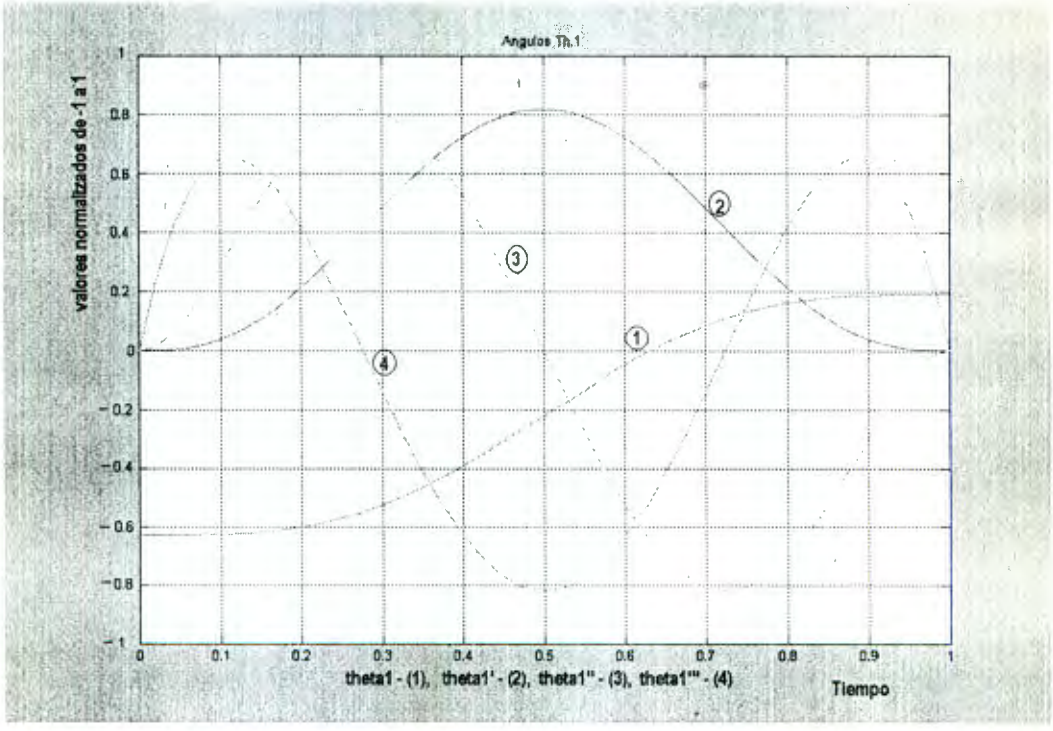

Figura 45. Gráfica del ángulo θ_1 y sus derivadas.

Splines cúbicas

Para el cálculo de la función splines cúbicas se indican una serie de puntos cartesianos con una cierta orientación. Los puntos de la trayectoria se muestran en la figura 46.

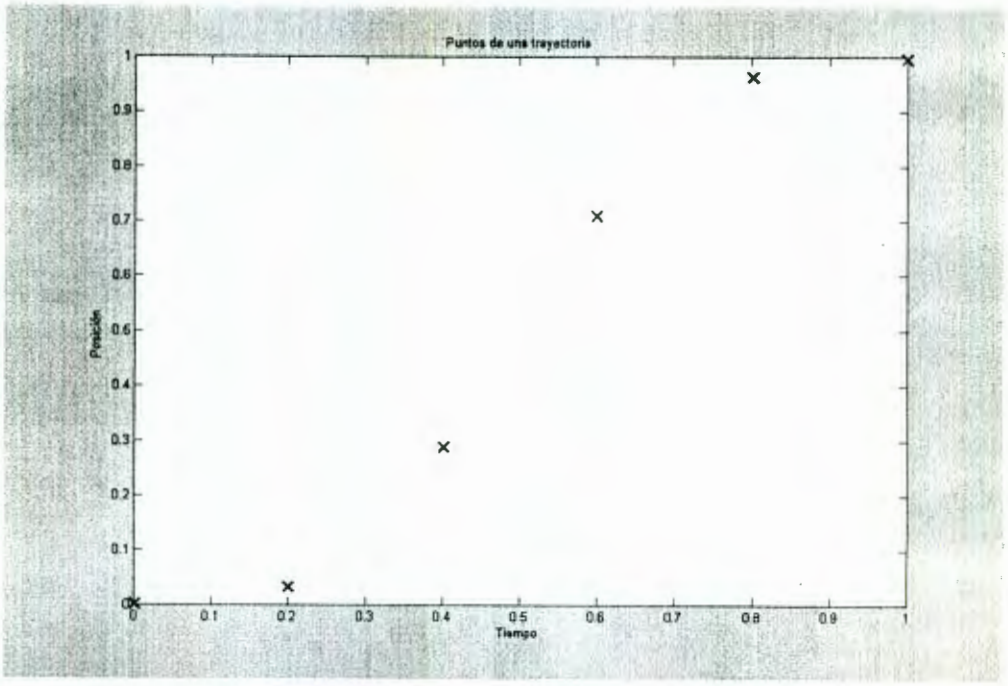

Figura 46. Puntos que definen trayectoria.

La figura 47 muestra la conexión de estos puntos utilizando la función spline cúbica.

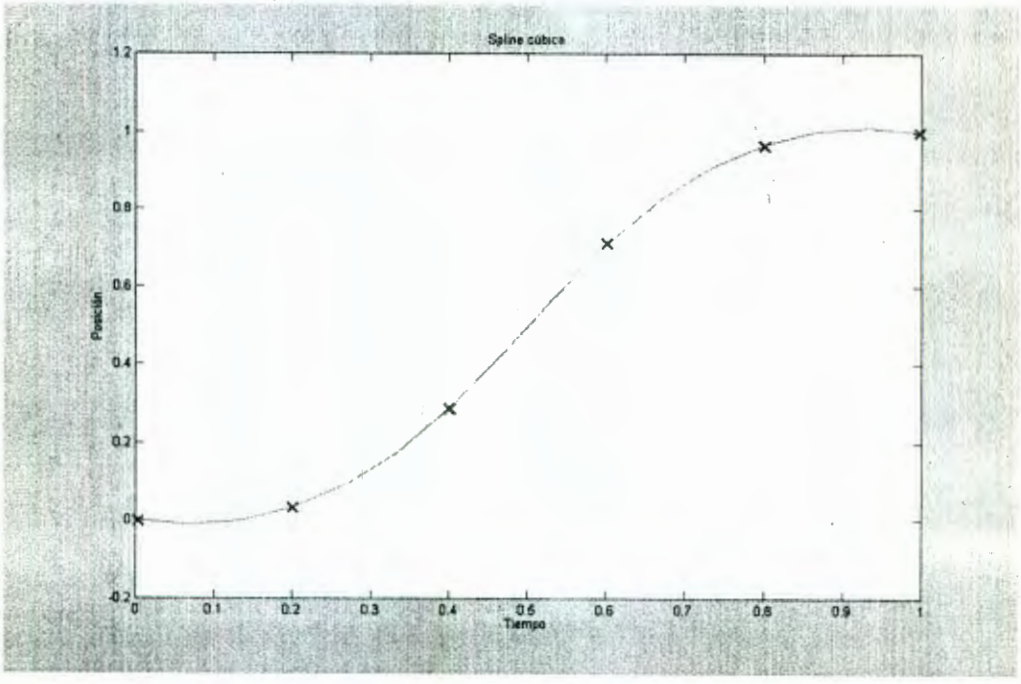

Figura 47. Función splines cúbicas.

La figura 48 muestra el módulo de planeación de trayectorias punto a punto.

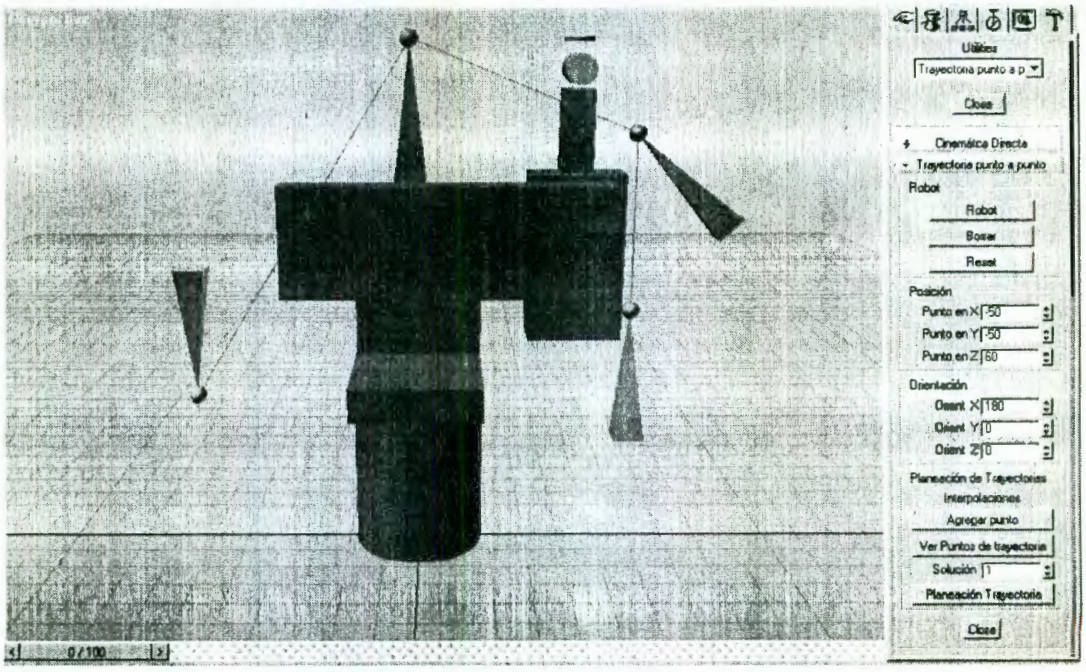

Figura 48. Trayectorias punto a punto con el robot en posición inicial.

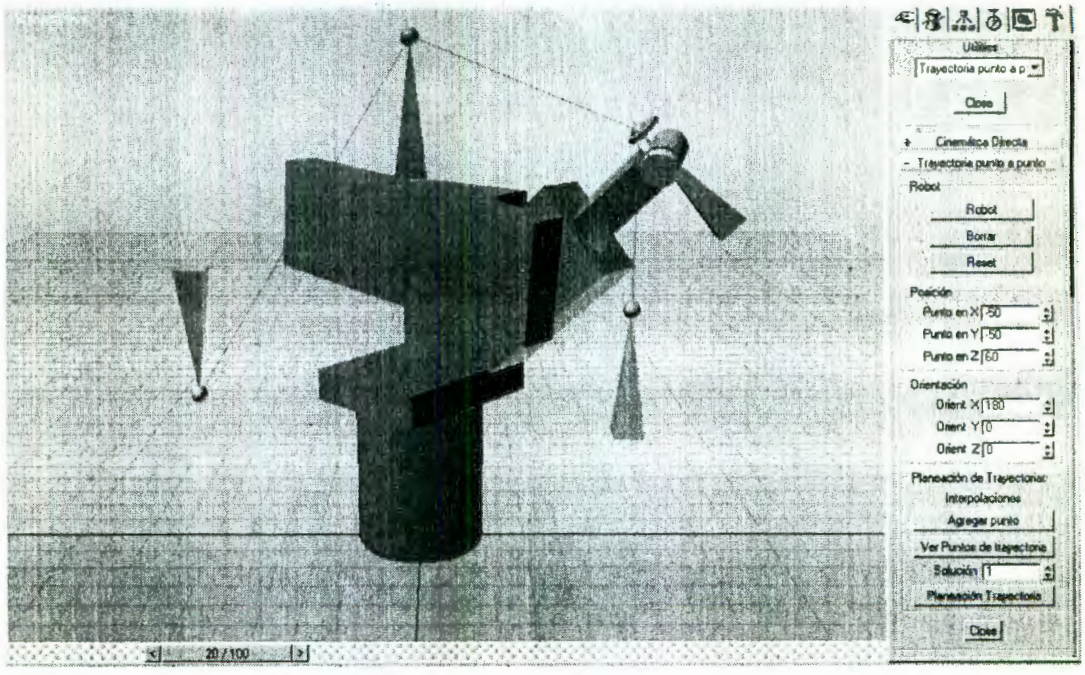

Figura 49. Trayectorias punto a punto con animación.

El programa calcula el seguimiento de la trayectoria para cualquier número de puntos de conexión. La versatilidad de este módulo es que permite especificar varios puntos y su respectiva orientación, ver figura 49. El número de puntos es arbitrario. El usuario comprueba que el robot se mueva a cada punto del recorrido con una cierta orientación.

4.6.2 Trayectorias continuas

En esta sección se presentan los ángulos de rotación de los eslabones que se obtuvieron al calcular la cinemática inversa en cada uno de los puntos de una circunferencia. La figura 50 muestra los puntos x, y y z de la trayectoria de la circunferencia.

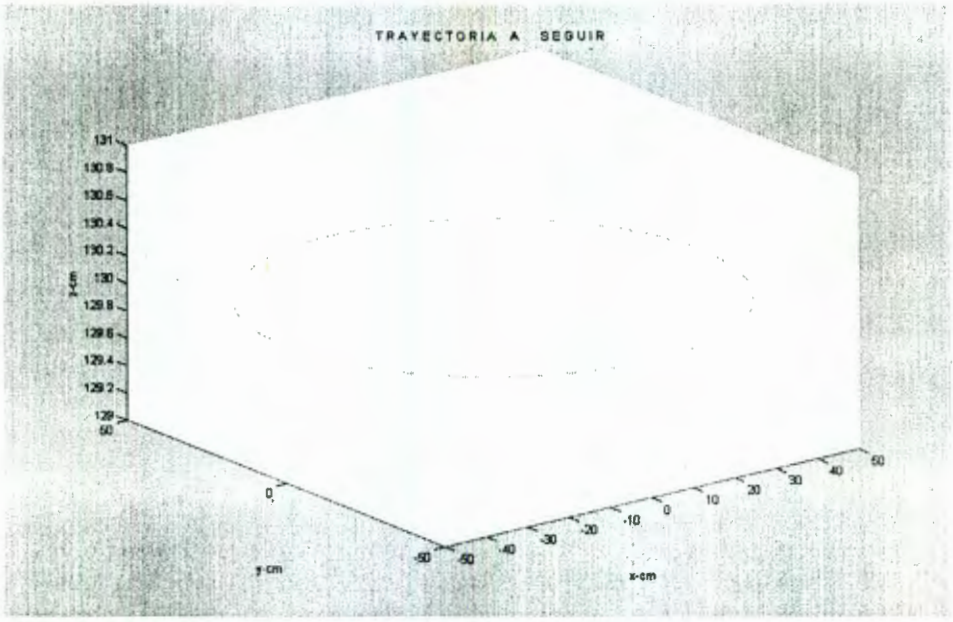

Figura 50. Gráfica de los puntos de la trayectoria de la circunferencia.

Los componentes x, y y z de la circunferencia se obtuvieron mediante las siguientes ecuaciones,

$$\begin{aligned} for\ i &= ini : paso : rango \\ k &= k + 1; \\ x(k) &= (ax * cos(i * (pi/180))) + hx; \\ y(k) &= (ax * sin(i * (pi/180))) + ky; \\ z(k) &= lz; \end{aligned}$$

donde:

$$\begin{aligned} i &= \text{tiempo} \\ ax &= \text{radio de la circunferencia} \\ hx &= \text{posición del centro de la circunferencia en el eje x} \\ ky &= \text{posición del centro de la circunferencia en el eje y} \\ lz &= \text{altura del centro de la circunferencia en el eje z} \\ paso &= \text{cantidad del incremento en el tiempo} \end{aligned}$$

la matriz *roll-yaw-pitch* se utilizó para especificar la orientación del órgano terminal, con los valores para $\psi = 15$, $\theta = 45$ y $\phi = 0$. Una vez que se especifican el radio y posición de la

circunferencia en el espacio cartesiano se obtienen los puntos de la trayectoria. Con la obtención de los puntos de la trayectoria y la matriz de orientación se calcula los ángulos de rotación de las articulaciones (cinemática inversa). Para ejemplificar, en la figura 51 se muestra la trayectoria, indicando la orientación con una figura triangular y el punto de la trayectoria con una esfera.

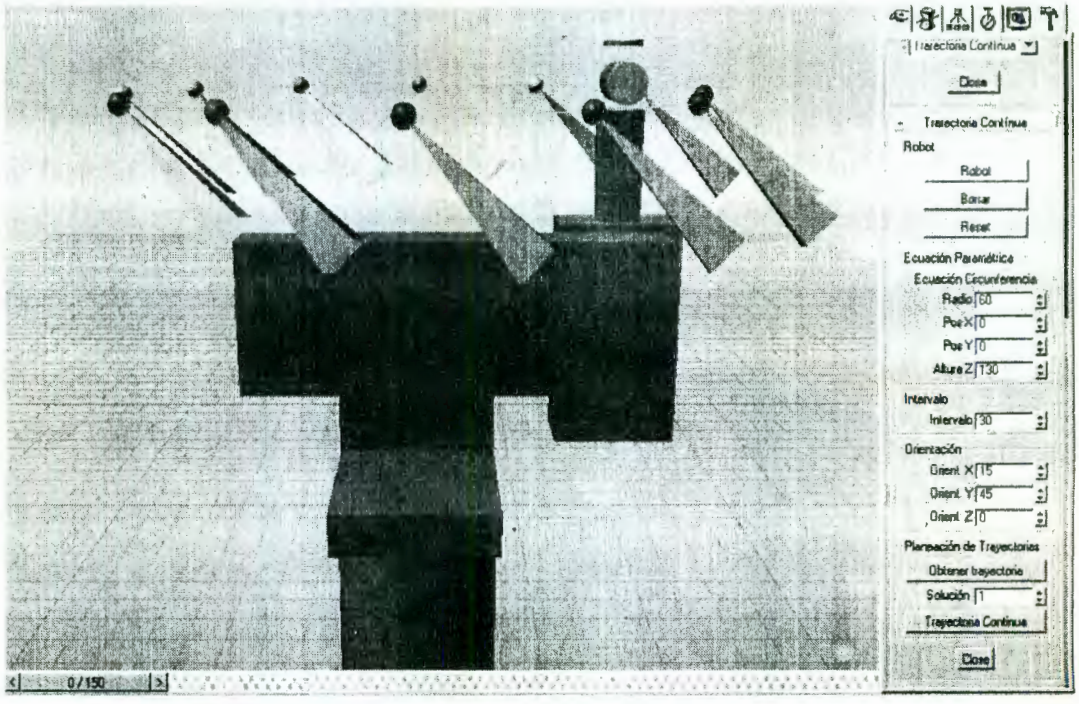

Figura 51. Representación de una trayectoria circular utilizando el simulador.

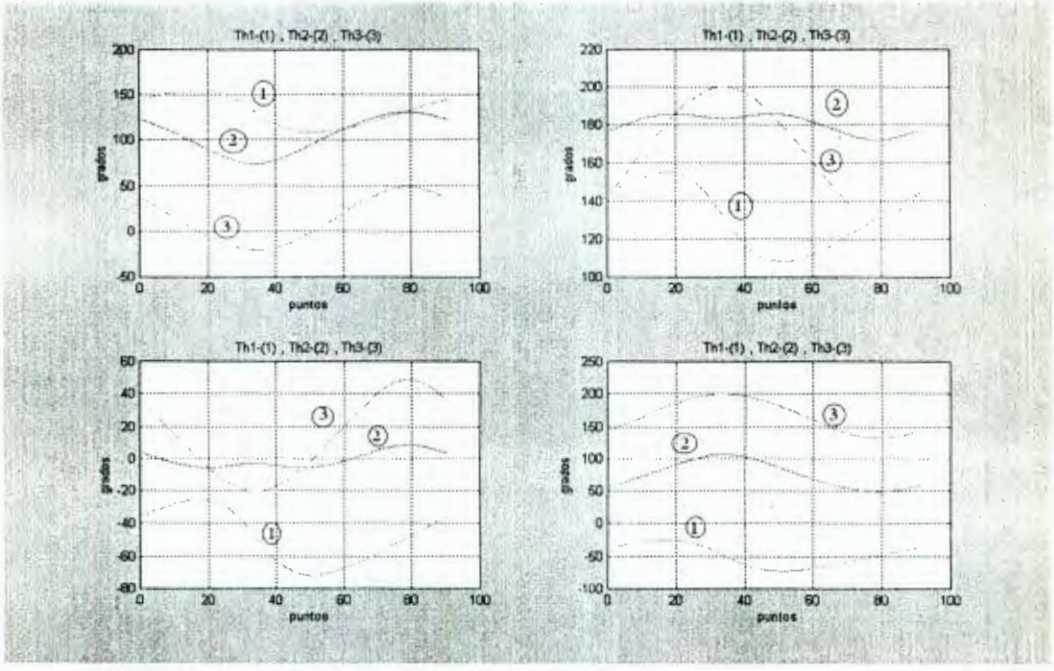

Figura 52. Gráfica de los ángulos de rotación θ_1, θ_2 y θ_3.

En la figura 52 presenta los ángulos de rotación θ_1, θ_2 y θ_3 de cada solución para obtener señales suaves que indiquen la inexistencia de cambios abruptos.

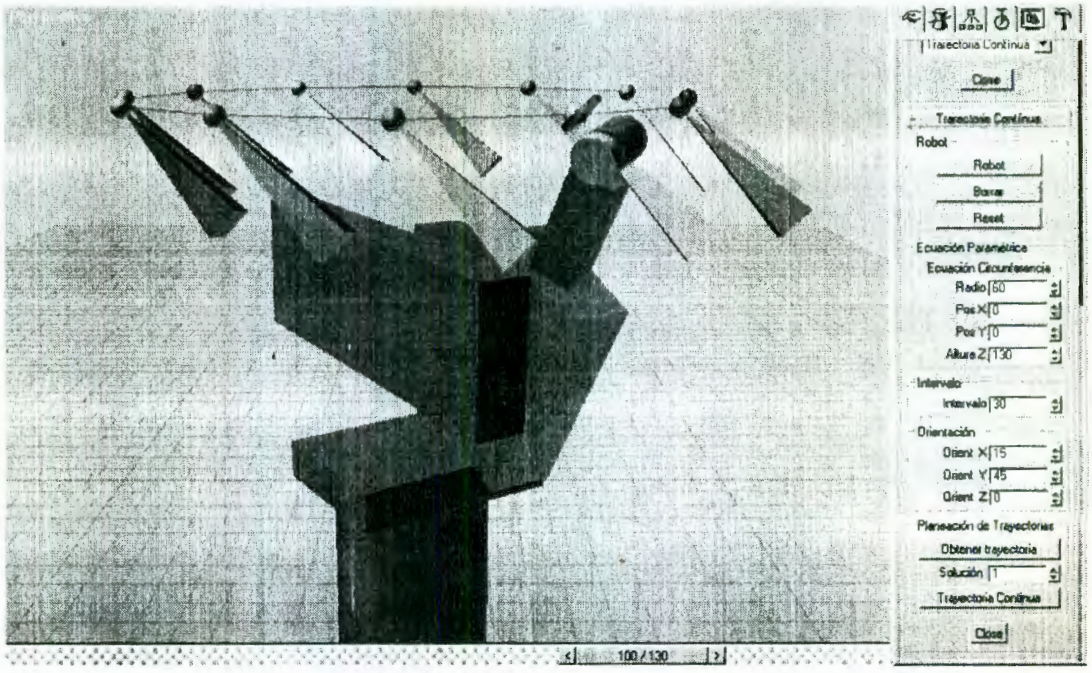

Figura 53. Primera solución de la cinemática inversa para la trayectoria.

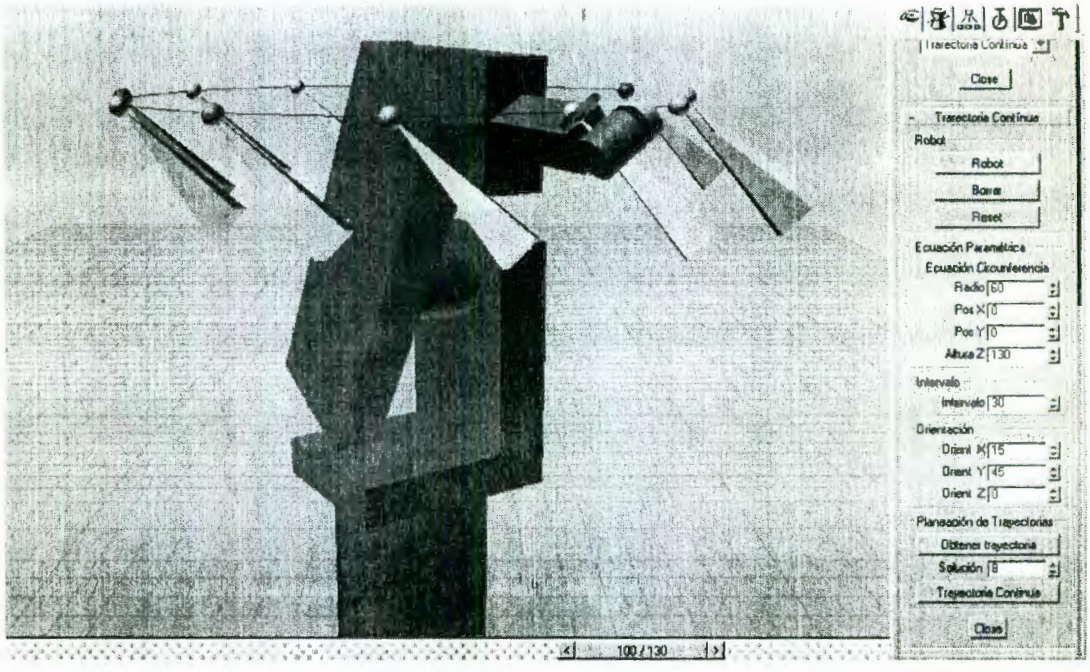

Figura 54. Octava solución de la cinemática inversa para la trayectoria.

En las figuras 53 y 54 se muestran la primera y la octava solución de la cinemática inversa. De esta manera el usuario puede evaluar visualmente los movimientos de los eslabones en un recorrido y elegir aquella que esté libre de colisiones.

Una vez que se elija una solución se generará el código necesario para la realización de dichos movimientos en tiempo real, o bien, para almacenar dicho movimiento en un archivo de video de modo que pueda ser analizado posteriormente. El *software* tiene la facilidad de generar en formato AVI un video con la simulación seleccionada. El usuario puede visualizar el video sin la necesidad de estar trabajando en 3D Studio Max.

Figura 55. Simulación en video.

Las figura 56 muestra el seguimiento del contorno de una barra de metal en simulación. Mientras que la figura 57 ilustra la ejecución real del seguimiento del contorno de la barra metálica. La figura 58 muestra el seguimiento de la cara superior de una pieza metálica en simulación y por último, en la figura 59 se visualiza la ejecución real del robot siguiendo el movimiento que se simuló en la figura anterior.

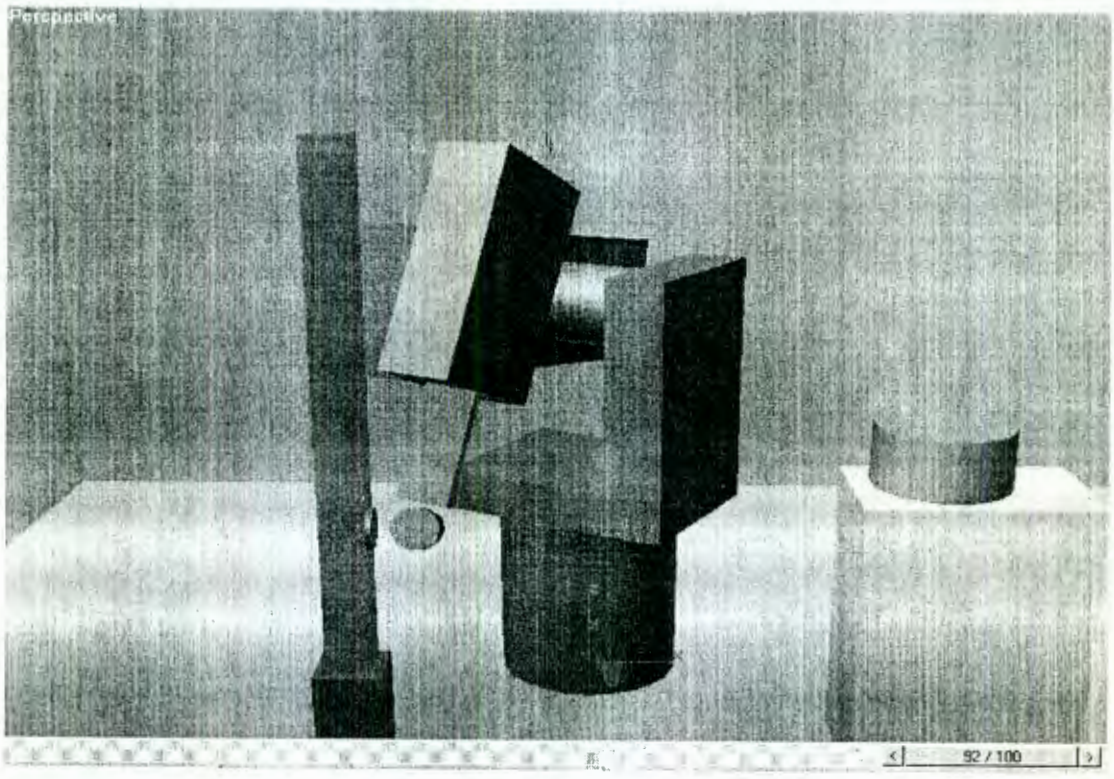

Figura 56. Simulación del robot en el desarrollo de una tarea.

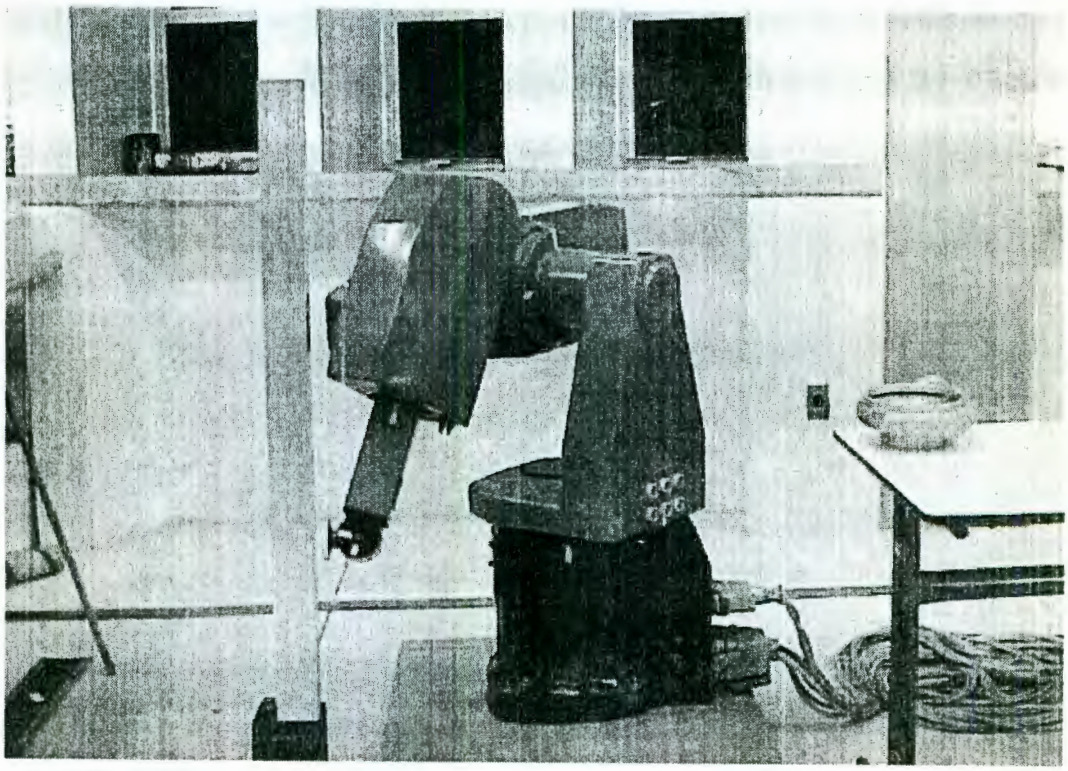

Figura 57. Ejecución real en el desarrollo de una tarea.

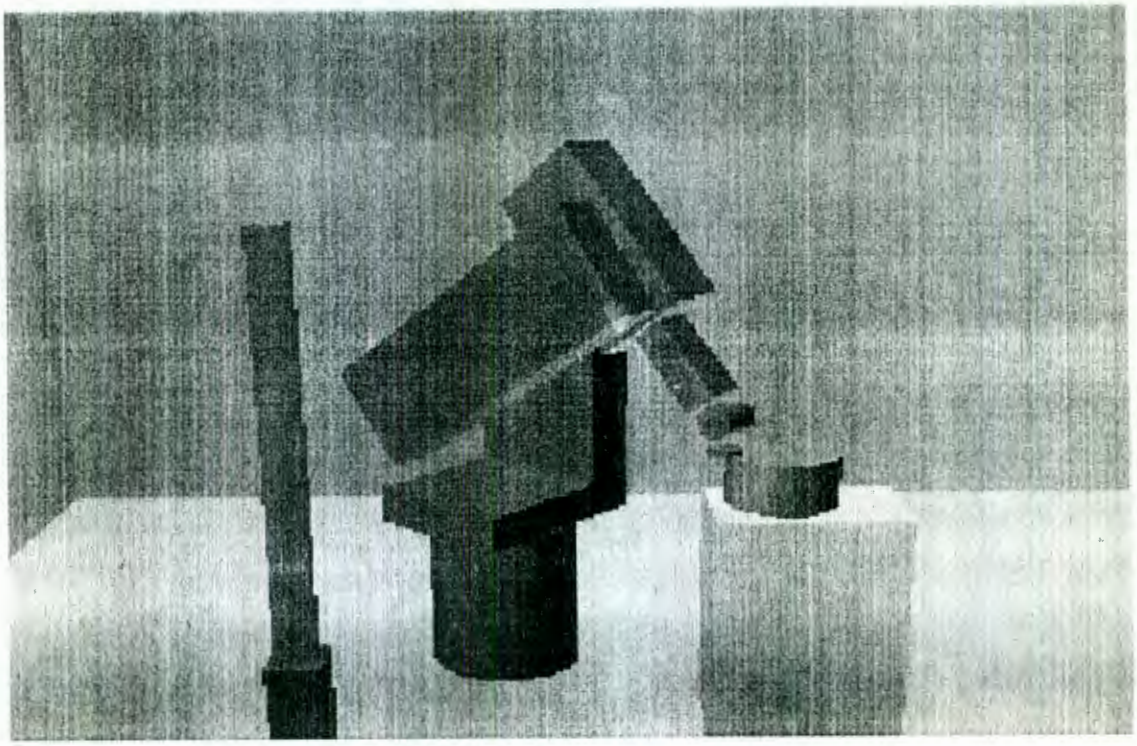

Figura 58. Simulación del seguimiento de una superficie de una pieza

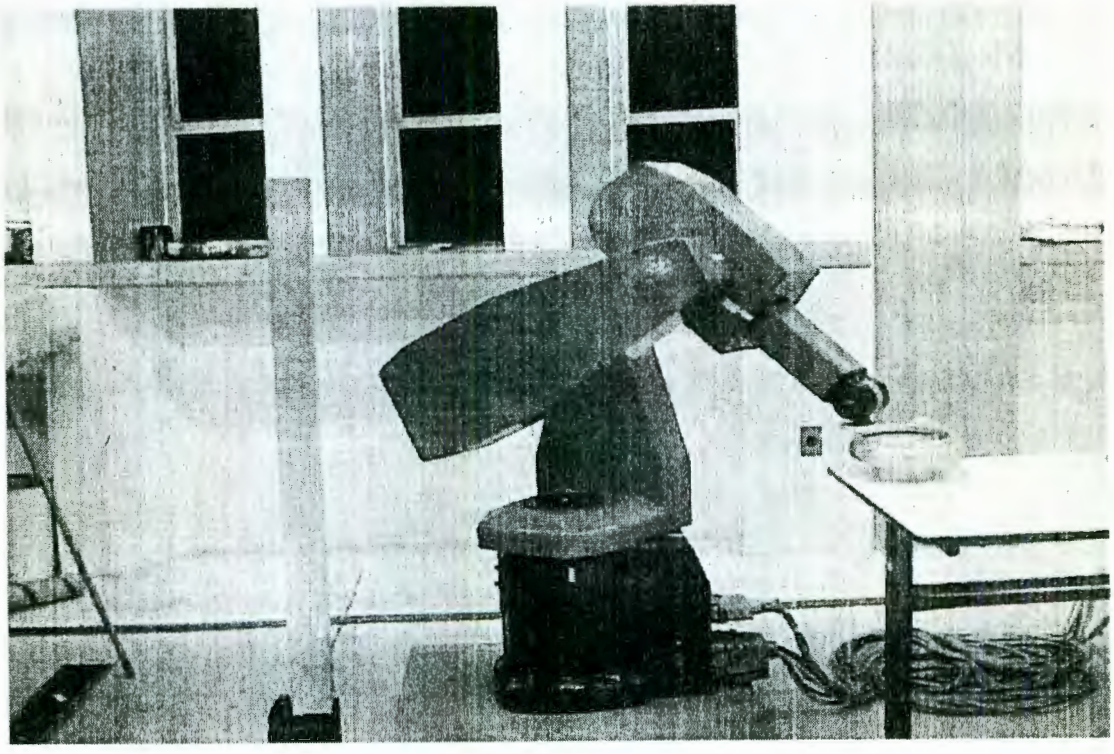

Figura 59. Ejecución real del seguimiento de una superficie de una pieza

Los ángulos obtenidos de la cinemática inversa, en el programa de simulación, se comprobaron satisfactoriamente con los movimientos efectuados en el robot real.

Conclusiones

El uso de un simulador favorece el proceso de programación de instrucciones en el robot real, ya que se pueden evaluar dichas instrucciones fuera de línea asegurando un mejor desempeño del robot.

El programa para la simulación de manipuladores robóticos desacoplados 6R se elaboró en 3D Studio Max ver. 2.5. Éste último permite crear un ambiente en tres dimensiones por medio de instrucciones de código. El programa permite ingresar los datos de configuracón de cualquier manipulador desacoplado y simular sus movimientos. Además, da la opción de generar un archivo de texto con los ángulos de rotación de la solución seleccionada en el desarrollo de una tarea. Para obtener los ángulos de las articulaciones que posicionen un robot en un punto, en el espacio cartesiano con una determinada orientación, se utilizó las características y parámetros del manipulador robótico CLOOS modelo Romat 56. Además, las diferentes posibles soluciones se calcularon. Se realizó el estudio y programación de la cinemática de velocidad obteniendo para ello el Jacobiano. Para obtener una velocidad específica en el órgano terminal del manipulador se calculan las velocidades de las articulaciones, cinemática inversa de velocidad. Por el contrario, si se parte de las velocidades de las articulaciones se calcula la velocidad del órgano terminal del manipulador, cinemática directa de velocidad. Además, se programó la planeación de trayectorias continuas y punto a punto para que el manipulador se desplace con movimientos suaves. En el módulo de la trayectoria punto a punto, el usuario puede agregar una secuencia de puntos cartesianos con sus respectivas orientaciones para que el programa simule el seguimiento de estos puntos.

Uno de los aspectos importantes de mencionar es la facilidad del *software* 3D Studio Max para programar animaciones. Este *software* tiene la opción de crear objetos, animaciones, persianas de interfase, etc., por medio de instrucciones de código. Por lo tanto, mediante la programación se tiene control y manejo de la simulación. La ventaja de éste paquete es que puede trabajar en computadoras personales y no necesariamente en una estación de trabajo Silicon-Graphics, pero requiere de suficiente memoria en RAM. La computadora en que se desarrolló el programa de simulación tiene 150 MHZ en memoria RAM, sin embargo se recomienda que sea de mayor capacidad.

Para el cálculo de la cinemática inversa, se especificó una configuración inicial para que el manipulador rote cada uno de sus eslabones. Se mostró cómo se obtuvo la configuración incial para el robot CLOOS a partir de los parámetros de Denavit-Hartenberg. La ubicación en una misma dirección de los ejes $X's$ de cada articulación del robot se presenta en los modelos esquemáticos presentados en la sección 4.4. El programa de simulación grafica la animación de los movimientos del robot en el seguimiento de una trayectoria. La simulación se muestra de forma tal que el usuario puede visualizar y evaluar las distintas soluciones para que el manipulador

lleve a cabo una tarea específica. El programa de simulación necesitó del estudio de los modelos cinemáticos, cinemática de velocidad y planeación de trayectorias.

Otra caracterísitica del simulador es la capacidad de ingresar un robot de configuración arbitraria. Siempre que éste robot esté desacoplado y con seis articulaciones, el programa efectuará los cálculos correctos para uno de sus módulos. Con la ayuda de este simulador se podrá seleccionar visualmente la solución que obtenga los movimientos del manipulador libre de colisiones y generar un archivo con los ángulos de rotación.

Referencias

Ángeles J., 1997. Fundamentals of Robotic Mechanical Systems, Theory, Methods, and Algorithms, Springer-Verlag, New York.

Ángeles J., 1982. Spatial Kinematic Chains, Analysis-Synthesis. Optimization Springer-Verlag, New York.

Ángeles J., 2000. Universidad de McGill.. Canadá. http://www.cim.mcgill.ca/~rvs/

Arteaga Coronel, R. 2000. Tésis "Estudio y Simulación en 3D de la Cinemática de un Robot Manipulador de 6GDL" Universidad Autónoma de Querétaro. En prensa.

Bicalho, A. and Feltman, S. 2000. Mastering.MAXScript and the SDK for 3D Studio MAX. Sybex, Inc. USA.

Bischoff A. PRT. 2000. Modelación y Simulación de Sistemas Robóticos. Universidad de Hagen. Alemania. http://prt.fernuni-hagen.de/pro/richodl/simulation.html

Craig, J. 1989. Introduction to Robotics. Mechanics and Control, Second Edition, Addison-Wesley Publishing Company.

Elliott, S. Miller P. et Al. 1998. 3D Studio MAX 2, (edición especial) Prentice Hall.

Foley, J. and Van Dam A, 1982, Fundamentals of Interative Computer Graphics Addison-Wesley Pub.

Ingeniería de Sistemas y Automática. 2000. Universidad Carlos III de Madrid. España. http://www.uc3m.es/uc3m/gral/IV/ivltin04.html

López, G. 2001. Validación del modelo de la cinemático directo y modelo de pares gravitacionales de un robot de Industrial, En prensa.

Martínez, E. et Al, 2000. Estudio de la cinemática de un robot industrial. Simposio "Segunda semana del quehacer científico y tecnológico en Querétaro 2000" Querétaro,Qro.

OpenCAD International Inc., 2000. Training Pak CD. http://opencad.complete-support.com

Vanderploeg. M. 2000. Robot Simulation and Off-line Programming,. ISU Visualization Laboratory.Universidad del Estado de Iowa. http://www.icemt.iastate.edu/research/manufacturing/robot/index.html

Gordillo, J. 2000.Proyecto REDII. ITESM campus Monterrey. México. http://www-cia.mty.itesm.mx/~gordillo/REDII

Simulador de Soldadura LT-3200.2000. http://www.telatrain.com/spahish/techlenco.htm

Simulación de Robot.2000. Instituto Tecnológico de Massachusetts MIT. Estados Unidos.
http://icmit.mit.edu/~cheng/robot/simulation.html

Spong, M.W., and Vidyasagar, M. 1989. Robot Dynamics and Control John Wiley & Sons.

Telerobotics. 2000.
http://www.dsl.whoi.edu/DSL/sayers/VRMLplot

Tsai L. 1999. Robot Analysis. The mechanics of serial and parallel manipulators, Wiley-Interscience.

3D WORLD. 3D Studio Max.
Año 3. Número 30, España.

Anexo 1 Movimiento Cicloidal

Un movimiento alternativo que produce velocidad y aceleración cero en los extremos de un intervalo finito es el *movimiento cicloidal.* En forma normal, este movimiento está dado por

$$s(\tau) = \tau - \frac{1}{2\pi}\sin 2\pi\tau$$

siendo su derivada como

$$\begin{aligned} s'(\tau) &= 1 - \cos 2\pi\tau \\ s''(\tau) &= 2\pi \sin 2\pi\tau \\ s'''(\tau) &= 4\pi^2 \sin 2\pi\tau \end{aligned}$$

El movimiento cicloidal y sus primeras tres derivadas en el tiempo, normalizado entre el intervalo (-1,1) se muestra en la figura A1. Mientras este movimiento tiene velocidad y aceleración cero y los extremos del intervalo $0 \leq \tau \leq 1$, su movimiento *jerk* no es cero en esos puntos y por lo tanto, presenta discontinuidades en los extremos del intervalo.

Cuando se implemente el movimiento cicloidad en la planeación de trayectorias punto a punto, se tiene para la j-ésima articulación,

$$\begin{aligned} \theta_J(t) &= \theta_j^I + (\theta_j^F - \theta_j^I)s(\tau) \\ \dot{\theta}(t) &= \frac{(\theta_F - \theta_I)}{T}s'(\tau) \\ \ddot{\theta}(t) &= \frac{(\theta_F - \theta_I)}{T^2}s''(\tau) \end{aligned}$$

la solución para los ángulos de las articulaciones se resuelve de forma similar en la interpolación 4-5-6-7.

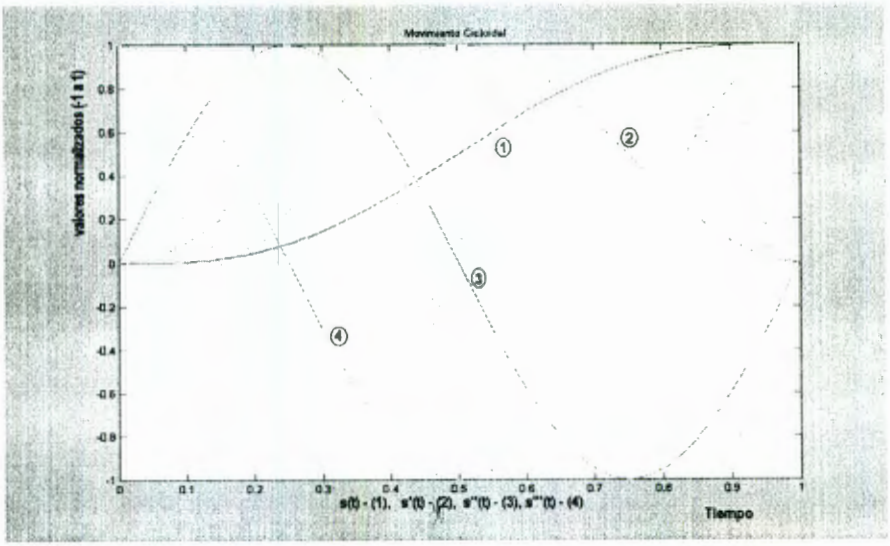

Figura A1. Movimiento Cicloidal

Anexo 2 "Estudio de la cinemática de un robot industrial"

ESTUDIO DE LA CINEMATICA DE UN ROBOT INDUSTRIAL

E. Martínez Ramírez, V. M. Hernández Guzmán, A. Zavala Río, C. G. López Guevara y C. S. López Cajún
Universidad Autónoma Querétaro, Facultad de Ingeniería
Centro Universitario, Cerro de las Campanas, Querétaro, Qro.
Tel. y Fax: (4) 2 15 43 40, e-mail: mare_eri@hotmail.com

Resumen

El presente trabajo describe la cinemática directa e inversa de un manipulador robótico de seis grados de libertad. Este trabajo forma parte del proyecto financiado por CONCYTEQ titulado *Desarrollo de una estación robotizada de soldadura.* En este proyecto se busca el desarrollo de un controlador para robots de 6 grados de libertad. El avance del proyecto que se presenta se refiere a los modelos cinemáticos directo e inverso de un robot alemán marca CLOOS modelo Romat 56, así mismo se presenta el desarrollo de un simulador de los movimientos del robot.

1. Introducción

El diseño de leyes de control para robots manipuladores de *n* grados de libertad requiere el conocimiento previo de su modelo dinámico. A su vez, es necesario un algoritmo de generación de trayectorias el cual requiere el conocimiento de los modelos cinemáticos del robot. Estos son pues indispensables en el desarrollo de un sistema de control para robots manipuladores. El presente trabajo se enfoca en los resultados obtenidos del estudio de la cinemática de un robot industrial de 6 grados de libertad, producido por *CLOOS* (empresa alemana), modelo Romat 56 (véase figura 1.1). Sus uniones son rotativas. El movimiento de sus eslabones es producido por medio de motores de corriente directa acoplados con juegos de engranes. Su estructura es semejante al de un robot tipo PUMA (*Programmable Universal Manipulator for Assembly*), pero no es idéntico. Su peso es de de 110 kg. Su altura máxima (todo extendido verticalmente) es de 2.1 mts. En su extremidad final se anexa la herramienta con la cual va a realizar la tarea programada, y cuyo peso no puede exceder de 5 kg. El robot cuenta con 6 servomotores de corriente directa para el control de cada eslabón, estos tienen instalados en la flecha encoderes ópticos de 1024 pulsos por revolución, a su vez cada motor cuenta con un freno mecánico el cual se controla con una señal digital de 24 Volts; cada motor se controla a través de un servoamplificador cuya función es amplificar la señal de control de la computadora de +/- 10 Volts a una señal de corriente, los servoamplificadores estan programados en lazo de corriente.

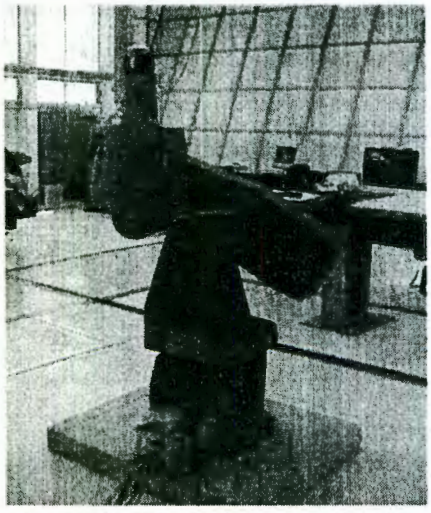

Figura 1.1 Manipulador tipo PUMA.

2. Modelo Cinemático Directo

En esta parte se obtiene el conjunto de ecuaciones que permiten conocer la posición y la orientación del extremo del robot, medidas en el espacio cartesiano, si se conocen los desplazamientos angulares en cada unión del robot. La herramienta fundamental utilizada en esta parte es la representación de Denavit-Hartenberg (Spong y Vidyasagar, 1989).

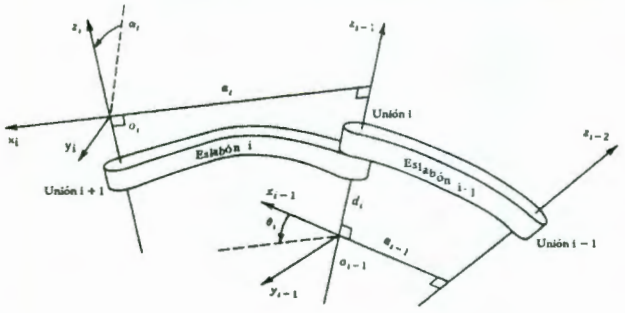

Figura 2.1 Parámetros de los eslabones en la convención de Denavit-Hartenberg.

De acuerdo a esta convención el eslabón i puede ser caracterizado usando cuatro parámetros representados por a_i , d_i, α_i y θ_i (ver figura 2.1) y se le fija el marco de coordenadas cartesianas O_i que se mueve junto con él. La posición del punto O_i y la orientación de los ejes x_i, y_i, z_i pueden ser medidas respecto al punto O_{i-1} y a la orientación de los ejes x_{i-1}, y_{i-1}, z_{i-1} mediante las siguientes operaciones sobre el marco de coordenadas O_i (en este orden):

1) una rotación por un ángulo α_i (el sentido positivo se determina mediante la regla de la mano derecha) sobre x_i, 2) una traslación de longitud a_i a lo largo del eje x_i, 3) una traslación de longitud d_i a lo largo del eje z_i y 4) una rotación por un ángulo θ_i sobre z_i. Este conjunto de operaciones básicas es representada matricialmente como (Spong y Vidyasagar, 1989):

$$T^{i}_{i-1} = \begin{bmatrix} \cos\theta_i & -\operatorname{sen}\theta_i \cos\alpha_i & \operatorname{sen}\theta_i \operatorname{sen}\alpha_i & a_i\cos\theta_i \\ \operatorname{sen}\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i \operatorname{sen}\alpha_i & a_i \operatorname{sen}\theta_i \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R^{i}_{i-1} & d^{i}_{i-1} \\ 0_3 & 1 \end{bmatrix} \tag{2.1}$$

donde 0_3 representa un vector renglón de tres elementos, todos ellos de valor cero, 1 representa un escalar igual a la unidad, $d^{i}_{i-1} \in \Re^3$ representa la posición de O_i medida desde O_{i-1} respecto a los ejes x_{i-1}, y_{i-1}, z_{i-1}, $R^{i}_{i-1} \in \Re^{3\times3}$ es una matriz de rotación cuya primera columna representa la proyección de un vector unitario en la dirección de x_i, sobre los ejes x_{i-1}, y_{i-1}, z_{i-1}, la segunda columna representa la proyección de un vector unitario en la dirección de y_i, sobre los ejes x_{i-1}, y_{i-1}, z_{i-1} y la tercera columna representa la proyección de un vector unitario en la dirección de z_i, sobre los ejes x_{i-1}, y_{i-1}, z_{i-1}. Nótese que si la unión i es rotativa entonces el único parámetro del eslabón i que no es constante es θ_i y si dicha unión es prismática entonces el único parámetro que no es constante es d_i. Cuando se trata de un robot con n grados de libertad (n uniones) es posible definir n+1 marcos de coordenadas y el modelo cinemático directo se obtiene como el siguiente producto matricial (Spong y Vidyasagar, 1989):

$$T^n{}_0 = T^1{}_0 T^2{}_1 \ldots T^{n-1}{}_{n-2} T^n{}_{n-1} \qquad (2.2)$$

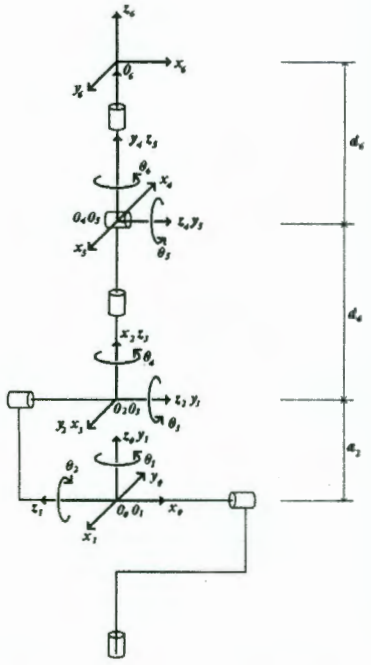

Figura 2.2 Asignación de marcos de coordenadas.

La matriz $T^n{}_0$ también se puede particionar como se hizo con $T^i{}_{i-1}$ en (2.1) pudiéndose definir un vector $d^n{}_0$ y una matriz $R^n{}_0$ que, de manera similar, representan la posición y la orientación del marco de coordenadas n respecto al marco de coordenadas 0, fijo en el laboratorio. El procedimiento para obtener la cinemática directa basado en la convención de Denavit-Hartenberg es resumido en (Spong y Vidyasagar, 1989). Siguiendo este procedimiento los diferentes marcos de coordenadas se asignan como se muestra en la figura 2.2, donde los cilindros representan las uniones (rotativas) del robot. En la tabla No. 1 se presentan los parámetros de Denavit-Hartenberg de cada eslabón. Usando (2.1) y la tabla No. 1 se obtienen las siguientes matrices[1]:

$$T_0^1 = \begin{bmatrix} c1 & 0 & s1 & 0 \\ s1 & 0 & -c1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_1^2 = \begin{bmatrix} c2 & s2 & 0 & a_2c2 \\ s2 & -c2 & 0 & a_2s2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_2^3 = \begin{bmatrix} c3 & 0 & s3 & 0 \\ s3 & 0 & -c3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^4 = \begin{bmatrix} c4 & 0 & s4 & 0 \\ s4 & 0 & -c4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_4^5 = \begin{bmatrix} c5 & 0 & s5 & 0 \\ s5 & 0 & -c5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_5^6 = \begin{bmatrix} c6 & -s6 & 0 & 0 \\ s6 & c6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.3)$$

Tabla No. 1

Eslabón	a_i (mm)	d_i (mm)	α_i	θ_i
1	0	0	90°	θ_1
2	*430*	0	180°	θ_2
3	0	0	90°	θ_3
4	0	430	90°	θ_4
5	0	0	90°	θ_5
6	0	66.7	0	θ_6

Finalmente, usando (2.2) y (2.3) se obtiene:

$$T_0^6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R_0^6 & d_0^6 \\ 0_3 & 1 \end{bmatrix} \qquad (2.4)$$

[1] En lo que sigue se usa ci y si para representar $\cos\theta_i$ y $\text{sen}\theta_i$, respectivamente.

donde:

r_{11}= c1c2c3c4c5c6+s2s3c1c4c5c6-s1s4c5c6+s3s5c1c2c6-s2s5c1c3c6+ s4s6c1c2c3+ +s2s3s4s6c1+ s1s6c4

r_{21}= s1c2c3c4c5c6+s1s2s3c4c5c6+s4c1c5c6+s1s3s5c2c6-s1s2s5c3c6+ s1s4s6c2c3+ +s1s2s3s4s6-s6c1c4

r_{31}= s2c3c4c5c6-s3c2c4c5c6+s2s3s5c6+s5c2c3c6+s2s4s6c3-s3s4s6c2

r_{41}=0

r_{12}= -s6c1c2c3c4c5-s2s3s6c1c4c5+s1s4s6c5-s3s5s6c1c2+s2s5s6c1c3+s4c1c2c3c6+ +s2s3s4c1c6+s1c4c6

r_{22}= -s1s6c2c3c4c5-s1s2s3s6c4c5-s4s6c1c5-s1s3s5s6c2+s1s2s5s6c3+s1s4c2c3c6+ +s1s2s3s4c6-c1c4c6

r_{32}= -s2s6c3c4c5+s3s6c2c4c5-s2s3s5s6-s5s6c2c3+s2s4c3c6-s3s4c2c6

r_{42}=0

r_{13}=s5c1c2c3c4+s2s3s5c1c4-s1s4s5-s3c1c2c5+s2c1c3c5

r_{23}=s1s5c2c3c4+s1s2s3s5c4+s4s5c1-s1s3c2c5+s1s2c3c5

r_{33}=s2s5c3c4-s3s5c2c4-s2s3c5-c2c3c5

r_{43}=0

r_{14}=d_6s5c1c2c3c4+d_6s2s3s5c1c4-d_6s1s4s5-d_6s3c1c2c5+d_6s2c1c3c5+d_4s3c1c2- -d_4s2c1c3+a_2c1c2

r_{24}=d_6s1s5c2c3c4+d_6s1s2s3s5c4+d_6s4s5c1 - d_6s1s3c2c5+d_6s1s2c3c5+d_4s1s3c2- -d_4s1s2c3+a_2s1c2

r_{34}=d_6s2s5c3c4-d_6s3s5c2c4-d_6s2s3c5-d_6c2c3c5+d_4s2s3+d_4c2c3+ a_2s2

r_{44}=1

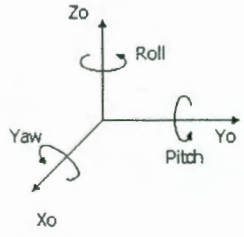

Figura 2.3 Angulos *roll* (ϕ), *pitch* (θ) y *yaw* (ψ)

Una manera de definir la orientación del marco de referencia O_6 medido respecto al marco O_0, es usando los ángulos de rotación *roll* (ϕ), *pitch* (θ) y *yaw* (ψ) mostrados en la figura 2.3. La importancia de estos tres ángulos radica en que la orientación de los ejes x_6, y_6, z_6 medida respecto a los ejes x_0, y_0, z_0 puede encontrarse mediante tres rotaciones sucesivas sobre estos últimos ejes, es decir, la matriz R^6_0 puede obtenerse a través de las siguientes rotaciones: 1) un ángulo ψ sobre x_0, 2) un ángulo θ sobre y_0 y 3) un ángulo ϕ sobre z_0, matricialmente (Spong y Vidyasagar, 1989):

$$R_0^6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} c\phi c\theta & -s\phi c\psi + c\phi s\theta s\psi & s\phi s\psi + c\phi s\theta c\psi \\ s\phi c\theta & c\phi c\psi + s\phi s\theta s\psi & -c\phi s\psi + s\phi s\theta c\psi \\ -s\theta & c\theta s\psi & c\theta c\psi \end{bmatrix} \quad (2.5)$$

El ángulo θ se puede calcular como;

$$s\theta = -r_{31} \quad , \quad c\theta = \pm\sqrt{1-r_{31}^2} \quad , \quad \theta = arctan(\pm\sqrt{1-r_{31}^2}, -r_{31}) \quad (2.6)$$

mientras que el cálculo de los ángulos ϕ y ψ se puede separar en dos casos:

1) Si cθ es diferente de cero:

$$\phi = arctan(r_{11}, r_{21}), \quad \psi = arctan(r_{33}, r_{32}) \quad c\theta > 0$$
$$\phi = arctan(-r_{11}, -r_{21}), \quad \psi = arctan(-r_{33}, -r_{32}) \quad c\theta < 0 \qquad (2.7)$$

2) Si cθ es igual a cero pueden ocurrir dos situaciones:

a) Si sθ =1:

$$\phi = \psi + arctan(r_{22}, -r_{12}) \qquad (2.8)$$

b) Si sθ =-1:

$$\phi = -\psi + arctan(r_{22}, -r_{12}) \qquad (2.9)$$

En los incisos 2a) y 2b) el ángulo ϕ queda en función de ψ, por lo que existen muchas combinaciones de estos dos ángulos que satisfacen las ecuaciones (2.8) y (2.9), así que se deberá proponer un valor de ψ para luego obtener el valor correspondiente de ϕ. Esta situación ocurre cuando $\theta = \pm 90^{\circ}$, debido a que en este caso los ángulos ϕ y ψ representan rotaciones sobre el mismo eje. Finalmente, el modelo cinemático directo del robot está dado del siguiente modo: si p_x, p_y, p_z representan las coordenadas x,y,z de O_6 medidas respecto al marco de coordenadas O_0 entonces:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} r_{14} \\ r_{24} \\ r_{34} \end{bmatrix}$$

mientras que los ángulos ϕ, θ y ψ dados en (2.6), (2.7), (2.8) y (2.9) definen la orientación de O_6 respecto a la de O_0.

3. Modelo Cinemático Inverso

En la sección anterior, se presentó el Modelo Cinemático Directo (MCD) obtenido de nuestro robot tipo Puma. Con este modelo es posible obtener la posición y orientación del órgano final, $x = (p_x, p_y, p_z, \psi, \theta, \phi)$,[2] dada una combinación de valores de las variables de unión, $\Theta = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$, *i.e.* $x = f_d(\Theta)$. Surge, entonces, una pregunta: ¿Es posible obtener un modelo con el cual se obtenga la combinación de valores de las variables de unión necesaria (o suficiente) para llevar al manipulador a una configuración dada, *i.e.* $\Theta = f_i(x)$? Un tal modelo recibe el nombre de *Modelo Cinemático Inverso* (MCI). La respuesta no es afirmativa para cualquier tipo de manipulador con estructura arbitraria, pero lo es para el nuestro. En efecto, nuestro robot tipo Puma cumple con una característica importante: su estructura es tal que los ejes de rotación de sus últimas tres uniones se intersectan en un punto común. Ésta es una condición suficiente que asegura la obtención de una *expresión cerrada* del MCI de un brazo manipulador de 6 grados de libertad (gdl) (Asada y Slotine, 1986; Spong y Vidyasagar, 1989; Canudas de Wit, *etal.*, 1996; Angeles, 1997). Esto se cumple para toda configuración dada dentro del *espacio de*

[2] A menos que se especifique explícitamente otra cosa, en esta sección, al hablar de *posición*, ésta será referida al sistema de coordenadas 0, es decir que se considerará en coordenadas expresadas con respecto al marco de referencia base (aquél que queda fijo independientemente de la configuración del manipulador); del mismo modo, al hablar de *orientación*, ésta también será referida al sistema de coordenadas 0.

trabajo del robot, *i.e.* dentro de los límites geométricos que impone su estructura. El conjunto de uniones cuyos ejes de rotación se intersectan en un punto común es denominado *muñeca* del manipulador, y el punto de intersección es denotado *centro de la muñeca*. La posición del centro de la muñeca está determinada por las variables de unión correspondientes a las primeras tres uniones y es invariante con respecto al resto. Por otro lado, las variables de unión correspondientes a la muñeca determinan la orientación del órgano final. Es decir que el posicionamiento del centro de la muñeca y la orientación del órgano final están desacoplados, o en otras palabras, un cambio en la orientación del órgano final deja fija la posición del centro de la muñeca y viceversa. Debido a esto, los manipuladores que cumplen con esta característica son llamados por algunos autores *manipuladores desacoplados* (Angeles, 1997).

Mientras que la expresión obtenida como MCD del manipulador es única, dando así una configuración única para cada combinación dada de valores de variables de unión, el MCI da, en general, un número múltiple de soluciones (posibles combinaciones de valores de variables de unión) para una configuración dada. La multiplicidad de soluciones puede incluso ser infinita en el caso de los *robots redundantes*, es decir, de aquéllos con más de 6 gdl. En el caso de manipuladores de 6 gdl o menos, la multiplicidad de soluciones es en general finita. Para ejemplificar este caso, considérese el manipulador planar de 3 gdl mostrado en la figura 3.1, donde puede apreciarse que dos configuraciones distintas del robot llevan al centro de su muñeca (tercera unión) a la misma posición dejando a su órgano final (tercer eslabón) con la misma orientación.

Figura 3.1 Manipulador planar de 3 gdl.

Existen diversas fuentes literarias que describen procesos analíticos para la obtención del MCI de manipuladores desacoplados. En (Angeles, 1997), por ejemplo, se presenta un análisis general, *i.e.* para cualquier tipo de manipulador desacoplado, en el que se obtienen expresiones generales de las variables de unión en términos de las variables de configuración deseada (o dada). En (Asada y Slotine, 1986) y (Canudas de Wit, *etal.*, 1996), se analizan manipuladores desacoplados con estructuras específicas con el fin de ejemplificar el proceso de obtención del MCI. En (Spong y Vidyasagar, 1989), una metodología analítica es expuesta. Ésta es explicada paso a paso y es aplicable a cualquier tipo de manipulador desacoplado. Más aún, una metodología alternativa suplementaria basada en la geometría del robot es presentada. Ésta permite la visualización conceptual (geométrica) de la relación obtenida para cada variable de unión en términos de la configuración deseada. Debido al balance de claridad y generalidad expuestos en (Spong y Vidyasagar, 1989), dichas metodologías fueron usadas para la obtención del MCI de nuestro robot (los resultados de una corroboraron los obtenidos con la otra). Escencialmente, estos métodos se basan en el hecho de que, dados un vector de posición del (origen del sistema de coordenadas asociado al) órgano final, $d = (p_x, p_y, p_z)$, y una matriz de orientación del órgano final, $R \in \Re^{3\times3}$,[3] la posición del centro de la muñeca, $d_c = (p_{cx}, p_{cy}, p_{cz})$, queda determinada como: $d_c = d - d_6 Rk$, donde $k = (0, 0, 1)^T$ —denota

[3] Nótese que d y R son conocidos y constantes (representan, respectivamente, la posición y orientación que se quiere dar al órgano final).

la dirección del eje z_6 referido a su propio marco de referencia (relativo al órgano final)— y d_6 es el parámetro del MCD correspondiente a la distancia, sobre el eje z_5, del origen o_5 al o_6 (relativos a los sistemas de coordenadas correspondientes); es decir que el centro de la muñeca está ubicado d_6 unidades (longitudinales) , medidas desde o_6, en dirección del eje z_6 —colineal al eje z_5— referido al marco de referencia 0, *Rk* (y en su sentido negativo, *i.e.* $-Rk$), *i.e.* justo en el punto de intersección de los ejes z_6, z_5 y z_4. Entonces, ${d_0}^4(\theta_1,\theta_2,\theta_3) = d_c$, donde ${d_0}^4$ proviene de la matriz de transformación homogénea ${T_0}^4$ (primeros tres elementos de la cuarta columna), denota la ubicación del origen o_4 (común a o_5) y depende (exclusivamente) de θ_1, θ_2 y θ_3, es un sistema de tres ecuaciones con tres incógnitas de donde se pueden obtener las expresiones de θ_1, θ_2 y θ_3 en términos de los elementos de *d* y *R*. Por otro lado, ${R_0}^6(\Theta) = {R_0}^3(\theta_1,\theta_2,\theta_3){R_3}^6(\theta_4,\theta_5,\theta_6) = R$, de donde se obtiene ${R_3}^6(\theta_4,\theta_5,\theta_6) = [\![{R_0}^3(\theta_1,\theta_2,\theta_3)]\!]^{-1}R = [\![{R_0}^3(\theta_1,\theta_2,\theta_3)]\!]^{T}R$, donde θ_1, θ_2 y θ_3 son previamente calculados como se explica arriba. De esta relación se obtienen, finalmente, las expresiones de θ_4, θ_5 y θ_6 en términos los elementos (conocidos) de $[\![{R_0}^3]\!]^T R$. Así pues, siguiendo esta metodología se llega a las siguientes expresiones para el caso de nuestro manipulador:

$$\theta_1 = \arctan(p_{cx}, p_{cy})$$

$$\theta_3 = \arctan(\pm\sqrt{1-D^2}\ , D)$$

$$\theta_2 = \arctan(r\ , p_{cz}) - \arctan(a_2 + d_4 s_3\ , d_4 c_3)$$

$$\theta_5 = \arctan(-E_{33}\ , \pm\sqrt{1-{E_{33}}^2}\ )$$

$$\theta_4 = \arctan(\sigma_{s5}E_{13}\ , \sigma_{s5}E_{23})$$

$$\theta_6 = \arctan(\sigma_{s5}E_{31}\ , -\sigma_{s5}E_{32})$$

donde: la función *arctan*(*x*,*y*) da como resultado el ángulo (único) cuyo coseno es igual a $\frac{x}{\sqrt{x^2+y^2}}$ y cuyo seno es igual a $\frac{y}{\sqrt{x^2+y^2}}$, $\forall(x,y)\neq(0,0)$; $D = \frac{R^2 - {d_4}^2 - {a_2}^2}{2a_2 d_4}$; $R = \sqrt{{p_{cx}}^2 + {p_{cy}}^2 + {p_{cz}}^2}$;[4] $r = \sqrt{{p_{cx}}^2 + {p_{cy}}^2}$; a_2 y d_4 son parámetros del MCD; s_3 y c_3 denotan,

[4] Recuérdese que p_{cx} , p_{cy} y p_{cz} representan las coordenadas del centro de la muñeca, tal como se definió arriba: $d_c = (p_{cx}, p_{cy}, p_{cz})$, y que además éstas están determinadas por: $d_c = d - d_6 Rk$.

respectivamente, el $sen\theta_3$ y $cos\theta_3$; $\sigma_{s5} = sign(sen\theta_5)$,[5] y E_{ij} representa al elemento en el renglón i y la columna j de la matriz $[\![R_0^3]\!]^T R$. La figura 3.2 muestra una representación gráfica de las relaciones encontradas para θ_1, θ_2 y θ_3. Cabe mencionar que ($\theta_1-\pi$, $\pi-\theta_2$, $\pi-\theta_3$) también es solución de $d_0^4(\theta_1,\theta_2,\theta_3) = d_c$.[6] Por otro lado, nótese, de las expresiones obtenidas, que

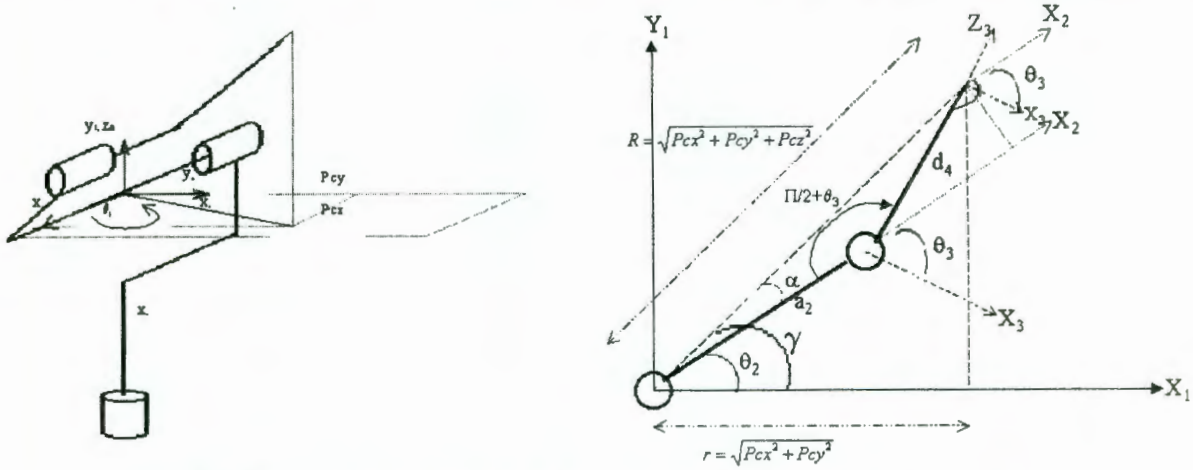

Figura 3.2 Representación gráfica de las relaciones encontradas para θ_1, θ_2 y θ_3.

(θ_1 , θ_2 , θ_3) acepta dos posibles soluciones dependiendo del signo tomado en la expresión correspondiente a θ_3. Es decir que existen cuatro posibles soluciones para la posición del centro de la muñeca de nuestro manipulador. Simultáneamente, (θ_4 , θ_5 , θ_6) (*i.e.* la orientación del órgano final) acepta también dos posibles soluciones dependiendo del signo tomado en la expresión correspondiente a θ_5. Es decir que nuestro modelo ofrece ocho posibles combinaciones de (θ_1 , θ_2 , θ_3 , θ_4 , θ_5 , θ_6) para cada configuración dada dentro del espacio de trabajo del manipulador. Finalmente, existen algunos casos especiales en las relaciones encontradas para θ_4 y θ_6 que por falta de espacio no se mencionarán en éste texto.

4. Simulador para Verificar Trayectorias

En este proyecto se esta construyendo un simulador gráfico de seguimiento de trayectorias para visualizar los movimientos del robot antes de ejecutarlos. Esto es de gran utilidad pues así se pueden prever posibles colisiones. Actualmente existe software que facilita el desarrollo de gráficas en tres dimensiones y que puede ser usado para determinar, en simulación, la posición de cada eslabón conforme el robot sigue una trayectoria, para lo cual son necesarios los modelos cinemáticos directo e inverso. Una vez que se haya verificado, en simulación, que la trayectoria es seguida de manera satisfactoria puede generarse el código necesario para que el robot realice los movimientos en tiempo real.

En este proyecto se esta utilizando el software 3D Studio MAX versión 2.5 (Elliott, Miller et Al., 1998) para construir el simulador porque facilita la creación de imágenes y la

---

[5] La función $sign(x)$ está definida como 1 si $x > 0$, -1 si $x < 0$ y 0 si $x = 0$.

[6] Por falta de espacio, esto no será mostrado gráficamente.

programación de animaciones. Para crear una figura se debe especificar su forma geométrica y sus dimensiones. De este modo se forma una imagen de cada eslabón. Para obtener la imagen completa del robot se debe especificar como se unen los eslabones entre sí. Esto se consigue definiendo el pivote o eje de giro de cada eslabón y la vinculación entre cada uno de ellos. Para esto último se especifica cual es el eslabón base y cual es el eslabón conectado inmediatamente a él. Este proceso se repite para todos los eslabones. Esto permite trasmitir el movimiento a todos los eslabones (a manera de herencia). Este software incluye una base de tiempo indispensable para las animaciones requeridas. De este modo, si se desea animar un movimiento coordinado del robot, se debe especificar: 1) los ejes que se desea mover, 2) el ángulo de giro de cada eje, 3) el instante en el que cada eje debe iniciar dicho giro y 4) el instante en el que cada eje debe terminar dicho giro.

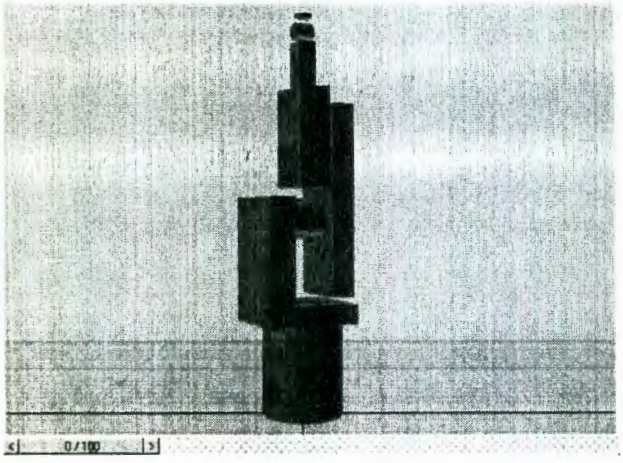

Figura 4.1 Imagen del robot en posición vertical (perspectiva del usuario).

Este software permite observar diferentes vistas del robot :1) vista superior, 2) vista frontal, 3) vista lateral izquierda , 4) vista lateral derecha y 5) perspectiva del usuario. Esta última es la única que presenta imágenes en tres dimensiones, pues las demás sólo lo hacen en dos dimensiones. En la figura 4.1 se presenta la vista en perspectiva del usuario del robot cuando éste se encuentra colocado en posición vertical. En la figura 4.2 se muestra una imagen del robot en la que los eslabones se encuentran rotados a diferentes ángulos. Cuando se desee que el extremo del robot siga una trayectoria (coordenadas cartesianas) se utilizará el modelo cinemático inverso para determinar la trayectoria correspondiente a ser seguida por cada unión (ángulos de giro de cada eje). Con esto se podrá realizar una animación del movimiento del robot pudiendo determinar si dicha trayectoria produce el movimiento deseado o si es segura en el sentido de que no haya colisiones del robot con el entorno que lo rodea. Para esto es importante contar con un módulo de evaluación del espacio de trabajo que permitirá detectar colisiones del robot con el entorno o con sí mismo o si los ejes de giro sobrepasan el rango permitido mecánicamente por el robot. Una vez verificado esto se generará el código necesario para ordenar al robot que realice dichos movimientos en tiempo real o bien para almacenar dicho movimiento en un archivo de video de modo que pueda ser analizado posteriormente.

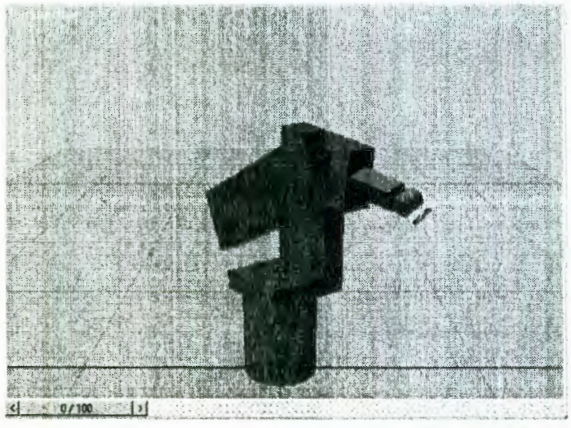

Figura 4.2 Imagen del robot con todos sus eslabones girados a diferentes ángulos (perspectiva del usuario).

5. Conclusiones

Se han obtenido los modelos cinemáticos directo e inverso para un robot industrial semejante al PUMA y se han identificado las diferentes posibles soluciones de ambos modelos. Como paso siguiente se deberá validar experimentalmente dichos modelos. Actualmente se está construyendo un simulador gráfico que será usado para verificar mediante animaciones las trayectorias diseñadas para el robot. Esto permitirá estudiar los movimientos del robot antes de que ser ejecutados en tiempo real con el fin de detectar posibles colisiones con el entorno que lo rodea y estudiar la calidad de los mismos. También se tiene contemplado obtener del modelo dinámico del robot que será utilizado, por un lado, para conseguir una animación más real en el simulador y, por otro lado, para estudiar el comportamiento dinámico del robot cuando se utilicen diferentes algoritmos de control. En este sentido se tiene previsto utilizar, en una primera etapa, controladores PID diseñados independientemente para los motores colocados en cada una de las uniones. Sin embargo, es de gran interés hacer un estudio comparativo entre diferentes técnicas de control de robots reportadas en la literatura, estudio que se piensa hacer primero en simulación y posteriormente en tiempo real directamente sobre el robot industrial. El objetivo final de este trabajo es diseñar una celda de soldadura robotizada por lo que también se esta trabajando el la parte electrónica de potencia e interfaces.

Referencias

Angeles, J. (1997). *Fundamentals of Robotic Mechanical Systems*. Springer, Nueva York.

Asada, H. y J.J.E. Slotine (1986). *Robot Analysis and Control*. John Wiley & Sons, E.E.U.U.

Canudas de Wit, C., B. Siciliano y G. Bastin (eds.) (1996) *Theory of Robot Control*. Springer, Londres.

Elliott, S., P. Miller et Al. (1998). *3D Studio MAX 2*. Ed. Prentice Hall, Edición Especial, España.

Spong, M.W. y M. Vidyasagar (1989). *Robot Dynamics and Control*. John Wiley & Sons, E.E.U.U.

El Gobierno del Estado de Querétaro
a través de su Consejo de Ciencia y Tecnología

Otorga la presente

CONSTANCIA

Al **Lic. Erika Martínez Ramírez**
M.C. Víctor Manuel Hernández Guzmán
Dr. Arturo Zavala Río
Ing. Carlos Guillermo López Guevara
Dr. Carlos Santiago López Cajún

Por haber participado en el simposio La Investigación y el Desarrollo Tecnológico en Querétaro 2000 con el artículo de investigación "Estudio de la cinemática de un robot industrial"

Santiago de Querétaro, agosto del 2000

2a semana EN QUERETARO

HACER CIENTIFICO Y TECNOLOGICO 2000

DR. ALEJANDRO LOZANO GUZMAN
DIRECTOR GENERAL DEL CONCYTEQ

HECHOS POR QUERÉTARO

Anexo 3 Programas fuentes del simulador

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- Programa     : Simulador de Manipuladores Robóticos Desacoplados 6R
-- Archivo      : Simulado.ms
-- Revisión     : 1.0
-- Autor        : Erika Martínez
-- Compañía     :
-- Fecha        : 01-may-2001
-- Descripción : Cinemática Directa, Cinemática Inversa,Parametros Denavit-
Hartenberg
--                      Planeción de Trayectorias
--                      Utiliza los elementos de interfaz, Panel de Utilidades,
para
--                      crear una persiana, que contiene los elementos para
obtener
--                      el cálculo de la cinemática inversa y visualizar los
movimientos
--                      en el robot.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---DECLARACIÓN DE VARIABLES GLOBALES
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
 vt1=#(0);vt2=#(0);vt3=#(0);vt41=#(0);vt51=#(0);vt61=#(0);
 vt42=#(0);vt52=#(0);vt62=#(0)
 nTraPtos=0
---DECLARACIÓN DE VARIABLES GLOBALES

rgoValor=1000; rgoGrados=360;
oriXEE=0;oriYEE=0;oriZEE=0;
x1eslab=0;x2eslab=0;x3eslab=0;
x4eslab=0;x5eslab=0;x6eslab=0;
es1=0; es2=0; es3=0; es4=0; es5=0; es6=0; --Eslabones
--Para cada eslabón arbitrario se definen los siguientes parametros:
vForEs=#(0,0,0,0,0,0);  --Forma del eslabón
vRP1=#(0,0,0,0,0,0);    --1ra. rotación del eje Pivote
vVRP1=#(0,0,0,0,0,0);   --1er. valor para la rotación del eje Pivote
vRP2=#(0,0,0,0,0,0);    --2do. rotación del eje Pivote
vVRP2=#(0,0,0,0,0,0);   --2do. valor para la rotación del eje Pivote
vRP3=#(0,0,0,0,0,0);    --3er. rotación del eje Pivote
vVRP3=#(0,0,0,0,0,0);   --3er. valor para la rotación del eje Pivote
vRadCil=#(0,0,0,0,0,0); --Valor del radio del cilindro
vAltCil=#(0,0,0,0,0,0); --Valor de la altura del cilindro
vlenCub=#(0,0,0,0,0,0); --Valor de length del cubo
vwidCub=#(0,0,0,0,0,0); --Valor de width del cubo
vheiCub=#(0,0,0,0,0,0); --Valor de height del cubo
vXpp=#(0,0,0,0,0,0);    --Valor de posición X del pivote
vYpp=#(0,0,0,0,0,0);    --Valor de posición Y del pivote
vZpp=#(0,0,0,0,0,0);    --Valor de posición Z del pivote
vXpe=#(0,0,0,0,0,0);    --Valor de posición X del eslabón
vYpe=#(0,0,0,0,0,0);    --Valor de posición Y del eslabón
vZpe=#(0,0,0,0,0,0);    --Valor de posición Z del eslabón
vNE=#(0,0,0,0,0,0);           --Numero de eslabón creado
a=#(0,0,0,0,0,0);
b=#(0,0,0,0,0,0);
```

```
alfa=#(0,0,0,0,0,0);


------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- Declaración de Funciones
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- Rutine       : elimE12 val x y
-- Function       : Pone en 0 los valores menores de exponente a 1e-12
-- Parameters   : val-> valor, vector o matriz
--                        x-> número de renglones
--                        y-> número de columnas
-- Return value : val-> valor, vector o matriz
------------------------------------------------------------------------------
function elimE12 val x y =
(
      for i=1 to x do
         for j=1 to y do
        if abs(val[i][j])<1e-12 then
              val[i][j]=0;
      return val;
)

------------------------------------------------------------------------------
-- Rutine       : multMat a b n
-- Function       : Multiplicación de una matriz 'a' por otra matriz 'b' de
orden 'n'
-- Parameters   : a-> matriz cuadrada de orden 'n'
--                        b-> matriz cuadrada de orden 'n'
--                        n-> orden de la matriz
-- Return value : mfin-> matriz resultante de la multiplicación
------------------------------------------------------------------------------
fn multMat a b n=
(
    aV=0;i=0;j=0;k=0;
      mfin=#(#(0,0,0,0),#(0,0,0,0),#(0,0,0,0),#(0,0,0,0));
      for i=1 to n do
            for j=1 to n do
            (
                  aV=0;
                  for k = 1 to n do
                        aV=aV+a[i][k]*b[k][j];
                  mfin[i][j]=aV;
            )
      return mfin;
)

------------------------------------------------------------------------------
-- Rutine       : multMat_N1 a b n
-- Function       : Multiplicacion de  una matriz [n][n] por un vector [n][1]
-- Parameters   : a-> matriz cuadrada de orden 'n'
--                        b-> vector [n][1]
--                        n-> orden de la matriz y dimensión del vector
-- Return value : mfin-> vector resultante de la multiplicación
```

```
-------------------------------------------------------------------------------
fn multMat_N1 a b n=
(
    aV=0;i=0;j=0; k=1;
      mfin=#(#(0),#(0),#(0),#(0));
            for i=1 to n do
            (
                  aV=0;
                  for j = 1 to n do
                        aV=aV+a[i][j]*b[j][k];
                  mfin[i][k]=aV;
            )
      return mfin;
)

-------------------------------------------------------------------------------
-- Rutine        : multVecxEsc a b n
-- Function       : Multiplicacion de un vector de [n][1] por un escalar
-- Parameters     : a-> vector [n][1]
--                          b-> escalar
--                          n-> dimensión del vector
-- Return value : mfin-> escalar resultante la multiplicación
-------------------------------------------------------------------------------
fn multVecxEsc a b n=
(
    aV=0; i=0;  k=1;
      mfin=#(#(0),#(0),#(0),#(0));
            for i=1 to n do
            (
                  mfin[i][k]=a[i][k]*b;
            )
      return mfin;
)

-------------------------------------------------------------------------------
-- Rutine        : restVecxVec a b n
-- Function       : Resta de un vector de [n][1] con otro de vector [n][1]
-- Parameters     : a-> vector [n][1]
--                          b-> vector [n][1]
--                          n-> dimensión del vector
-- Return value : mfin-> vector resultante de la resta
-------------------------------------------------------------------------------
fn restVecxVec a b n=
(
    aV=0; i=0;  k=1;
      mfin=#(#(0),#(0),#(0),#(0));
            for i=1 to n do
            (
                  mfin[i][k]=a[i][k]-b[i][k];
            )
      return mfin;
)

-------------------------------------------------------------------------------
-- Rutine        : trasMat a n
-- Function       : Trasponer una matriz de grado [n][n]
-- Parameters     : a-> matriz cuadrada de orden 'n'
```

```
--                          n-> orden de la matriz
-- Return value : mfin-> matriz resultante
-------------------------------------------------------------------------------
fn trasMat a n=
(
    aV=0; i=0; j=1;
      mfin=#(#(0,0,0,0),#(0,0,0,0),#(0,0,0,0),#(0,0,0,0));
      for i=1  to n do
            for j=1 to n do
            (
                  mfin[j][i]=a[i][j];
            )
      return mfin;
)

-------------------------------------------------------------------------------
-- Rutine       : InvKinet vd vExOrien
-- Function      : Obtiene la cinemática inversa para un punto en el espacio
cartesiano
--                          con cierta orientación
-- Parameters   : vd-> vector de posición X, Y y Z  de la forma #(X,Y,Z)
--                          vExOrien-> vector de orientación en el eje oX, oY y
oZ,forma #(oX,oY,oZ)
-- Return value : matriz con las diferentes soluciones para la cinematica
inversa
--                          vAng=#(th1,th2,th3,th41,th42,th51,th52,th61,th62);
-------------------------------------------------------------------------------
function InvKinet vd vExOrien =
(
-- DECLARACION DE VARIABLES
oYX=vexOrien[1];
oPY=vexOrien[2];
oRZ=vexOrien[3]+180;

af=#(#(0,0,0),#(0,0,0),#(0,0,0));
mRYP=#(#(0,0,0),#(0,0,0),#(0,0,0));
pcx=0;  pcy=0;  pcz=0;
dc=#(#(0),#(0),#(0));

-- INICIO DEL PROGRAMA

-- DECLARACION DE LA MATRIZ DE ROTACION " ROLL, YAW, PITCH " PARA DETERMINAR LA
ORIENTACION DEL EE.
-- considerando los siguientes angulos de rotación en cada eje.

m11=cos(oRz)*cos(oPy); m12=-sin(oRz)*cos(oyx)+cos(orz)*sin(oPy)*sin(oyx);
m13=sin(orz)*sin(oyx)+cos(orz)*sin(oPy)*cos(oyx);
m21=sin(oRz)*cos(oPy); m22=cos(oRz)*cos(oyx)+sin(orz)*sin(oPy)*sin(oyx);  m23=-
cos(orz)*sin(oyx)+sin(orz)*sin(oPy)*cos(oyx);
m31=-sin(oPy);          m32=cos(oPy)*sin(oyx);
        m33=cos(opy)*cos(oyx);

a1i=#(m11,m12,m13,0);  a2i=#(m21,m22,m23,0);   a3i=#(m31,m32,m33,0);
a4i=#(0,0,0,1);
ai=#(a1i,a2i,a3i,a4i);
--ai=elimE12 ai 4 4;
-- OBTIENE LA MATRIZ DE ORIENTACION ROLL, YAW, PITCH Y LA ALMACENA EN "mRYP"
```

```
mRYP=elimE12 ai 4 4;
 mRYP=ai;
vr11= mRYP[1][1];       vr12= mRYP[1][2]; vr13= mRYP[1][3];
vr21= mRYP[2][1];       vr22= mRYP[2][2]; vr23= mRYP[2][3];
vr31= mRYP[3][1];       vr32= mRYP[3][2]; vr33= mRYP[3][3];

-- COORDENADAS DEL PUNTO DEL EE EN EL EJE X,Y,Z  d = (px,py,pz)
--vd[3]=vd[3];
vdX=vd[1];
vdY=vd[2];
vdZ=vd[3];
d=#(#(vdX),#(vdY),#(vdZ));


-- VALORES DE LOS PARAMETROS DE D-H. (la distancia de d1=0 si el origen O1 y 02
estan juntos
a2= 43; d1=89.5; d4=43; d6=7;

-- OBTENEMOS EL VALOR DE LA POSICION DE LA MUÑECA A PARTIR DE DONDE SE ORIENTARÁ
EL EE.
-- dc = d - d6*R*K     donde R = mRYP

vk=#(#(0),#(0),#(1));
dc = restVecxVec d (multVecxEsc (multMat_N1 mRYP vk 3) (d6) 3) 3;

--  OBTENEMOS EL VALOR DE LOS ANGULOS THETA-1 , THETA-2 , THETA-3
pcx=dc[1][1];   pcy=dc[2][1];  pcz=dc[3][1];

--pcx=vd[1]-mRYP[1][3]*d6;
--pcy=vd[2]-mRYP[2][3]*d6;
--pcz=vd[3]-mRYP[3][3]*d6;


d=#(#(vd[1]),#(vd[2]),#(vd[3]));
th1=#(0,0,0,0);
th1[1] = atan2 pcy pcx;
th1[2] = 180+atan2 pcy pcx;
th1[3] = 180+atan2 pcy pcx;
th1[4] = atan2 pcy pcx;
--th1 = atan2 pcy pcx;

--th1=elimE12 th1 1 4
  rR=sqrt(pcx^2+pcy^2+pcz^2);
  dD=(rR^2-d4^2-a2^2)/(2*a2*d4);

--if vExOrien == 3  then  th3 = (atan2 dD (sqrt(1-dD^2)))  else th3 = 180-(atan2
dD (sqrt(1-dD^2)));
th3=#(0,0,0,0);
th3[1] = (atan2 dD (sqrt(1-dD^2)));
th3[2] = (atan2 dD (sqrt(1-dD^2)));
th3[3] = 180-(atan2 dD (sqrt(1-dD^2)));
th3[4] = 180-(atan2 dD (sqrt(1-dD^2)));
--th3=elimE12 th3 1 4
  r=sqrt(pcx^2+pcy^2);
--  if th3 == 0 then th3=0 else th3=th3;
  th2=#(#(0,0,0,0));
--for i=1 to 4 do
```

```
--    th2[1][i] = atan2 pcz r - atan2 (d4*cos (th3[1][i])) (a2+d4*sin
(th3[1][i]))
--    th2 = atan2 pcz r - atan2 (d4*cos th3) (a2+d4*sin th3)
--th2=elimE12 th2 1 4

  th11=#(#(0,0,0,0));
  th21=#(#(0,0,0,0));
  th31=#(#(0,0,0,0));

-------------------------------------
-- Calculo de theta 2
   a=#(0,0,0,0,0,0);
a[1]=0;
a[2]=43;
a[3]=0;
a[4]=0;
a[5]=0;
a[6]=0;
b=#(0,0,0,0,0,0);
b[1]=0--89.5;
b[2]=0;
b[3]=0;
b[4]=43;
b[5]=0;
b[6]=7;
al=#(0,0,0,0,0,0);
al[1]=90--pi/2;
al[2]=180--pi;
al[3]=90--pi/2;
al[4]=90--pi/2;
al[5]=90--pi/2;
al[6]=0; --pi/180;

ct=#(0,0,0,0,0,0);
st=#(0,0,0,0,0,0);
l=#(0,0,0,0,0,0);
u=#(0,0,0,0,0,0);
for i=1 to 6 do
 (
   l[i]=cos(al[i]);
   if abs(l[i])<1e-12 then
      l[i]=0;
   u[i]=sin(al[i]);
   if abs(u[i])<1e-12 then
     u[i]=0;

)

cont=4;
for i=1 to cont do
( --th11[i]=th1[i]*pi/180;
       --th31[i]=th3[i]*pi/180;
   -- Definiciónde las constantes AA11, AA12 y delta2
   A11=a[2]+a[3]*cos(th3[i])+b[4]*u[3]*sin(th3[i]);
   A12=-
a[3]*l[2]*sin(th3[i])+b[3]*u[2]+b[4]*l[2]*u[3]*cos(th3[i])+b[4]*u[2]*l[3];
   delta2=A11^2+A12^2;
```

```
   --% Verificacion de la Segunda singularidad
   if delta2==0 then
   (
      sss='\n\n  ERROR 8. Error de Segunda singularidad\n';
      --fprintf['\n Elija otro centro\n');
      out1=0;
   )
   --Calculo de theta2
   pec1= pcx*cos(th1[i])+pcy*sin(th1[i])-a[1];
   pec2= -pcx*l[1]*sin(th1[i])+pcy*l[1]*cos(th1[i])+(pcz-b[1])*u[1];
   pec3= A11*pec1 - A12*pec2;
   pec4= A12*pec1 + A11*pec2;
   th2[i]=atan2 (pec4/delta2) (pec3/delta2);  --theta 2
   --th2[i]=th21[i]*180/pi;
) --end     %Fin de ciclo for (cálculo de theta2)


-------------------------------------
--   OBTENEMOS LA MATRIZ DE ROTACION "R 0-3"
d1=0;       -- en el caso de que los orígenes 01 y 02 esten en el mismo punto
--mR03=
th41=#(0,0,0,0);
th51=#(0,0,0,0);
th61=#(0,0,0,0);
th42=#(0,0,0,0);
th52=#(0,0,0,0);
th62=#(0,0,0,0);

for i=1 to 4 do
(
ma1=#(cos th1[i]*cos(th2[i]-th3[i]), -sin th1[i], -cos th1[i]*sin(th2[i]-
th3[i]));
ma2=#(sin th1[i]*cos(th2[i]-th3[i]),cos th1[i], -sin th1[i]* sin(th2[i]-
th3[i]));
ma3=#(sin(th2[i]-th3[i]),0,cos(th2[i]-th3[i]));
mR03=#(ma1,ma2,ma3);

-- OBTNEMOS       6          3  -1
--              R 3  =    ( R o )     *  R     donde R es la matriz de orientacion
Roll,Yaw,Pitch
E11=cos th1[i]* cos (th2[i]-th3[i])*vr11+sin th1[i]*cos(th2[i]-
th3[i])*vr21+sin(th2[i]-th3[i])*vr31;
E12=cos th1[i]* cos (th2[i]-th3[i])*vr12+sin th1[i]*cos(th2[i]-
th3[i])*vr22+sin(th2[i]-th3[i])*vr32;
E13=cos th1[i]* cos (th2[i]-th3[i])*vr13+sin th1[i]*cos(th2[i]-
th3[i])*vr23+sin(th2[i]-th3[i])*vr33;

E21=-sin th1[i]*vr11 + cos th1[i]*vr21;
E22=-sin th1[i]*vr12 + cos th1[i]*vr22;
E23=-sin th1[i]*vr13 + cos th1[i]*vr23;

E31=-cos th1[i]*sin(th2[i]-th3[i])*vr11 - sin th1[i]*sin(th2[i]-th3[i])*vr21 +
cos(th2[i]-th3[i])*vr31;
E32=-cos th1[i]*sin(th2[i]-th3[i])*vr12 - sin th1[i]*sin(th2[i]-th3[i])*vr22 +
cos(th2[i]-th3[i])*vr32;
```

```
E33=-cos th1[i]*sin(th2[i]-th3[i])*vr13 - sin th1[i]*sin(th2[i]-th3[i])*vr23 +
cos(th2[i]-th3[i])*vr33;

-- OBTIENE ANGULOS THETA-4 , THETA-5 , THETA-6

--th5 = 180 - atan2 (sqrt(1-E33^2)) (-E33);
th51[i] = -atan2 (sqrt(1-E33^2)) (-E33);
if sin th51[i] > 0 then sigma = 1 else sigma = -1;
th41[i] = atan2 (sigma*E23) (sigma*E13);
th61[i] = atan2 (-sigma*E32) (sigma*E31);

th52[i] = atan2 (sqrt(1-E33^2)) (-E33);
if sin th52[i] > 0 then sigma = 1 else sigma = -1;
th42[i] = atan2 (sigma*E23) (sigma*E13);
th62[i] = atan2 (-sigma*E32) (sigma*E31);

)
vAng=#(th1,th2,th3,th41,th42,th51,th52,th61,th62);
return vAng;
)
-------------------------------------------------------------------------------
-- Rutine       : rotar t1 t2 t3 t41 t51 t61 t42 t52 t62 vPos nP
-- Function       : Rota los eslabones del robot de acuerdo al número de
solución indicada
-- Parameters    : t1,t2,t3-> soluciones de posición para los eslabones 1, 2 y 3
respectivamente.
--t41,t51,t61-> primera orientación para los eslabones 4, 5 y 6 respectivamente.
--t42,t52,t62-> segunda orientación para los eslabones 4, 5 y 6 respectivamente.
--vPos-> punto P en el espacio cartesiano
--nP-> número de solución
-- Return value : ninguno
-------------------------------------------------------------------------------
function rotar t1 t2 t3 t41 t51 t61 t42 t52 t62 vPos nP =
(
      kk=0;
      if nP>4 then (k=nP-4; kk=1;) else (k=nP)
      d1=89.5;
      in coordsys local rotate $oEsBas 0 z_axis
      in coordsys local rotate $oEs1  (t1[k]) z_axis
      in coordsys local rotate $oEs2  (-(t2[k])) x_axis
      in coordsys local rotate $oEs3  (t3[k]) x_axis
      if kk==0 then
      (
          in coordsys local rotate $oEs4  (t41[k]) z_axis
          in coordsys local rotate $oEs5  ( 180-(t51[k] )) z_axis
            in coordsys local rotate $oEs6  (-(t61[k])) z_axis
      )else
      (
          in coordsys local rotate $oEs4  (t42[k]) z_axis
          in coordsys local rotate $oEs5  (180-(t52[k])) z_axis
            in coordsys local rotate $oEs6  (-(t62[k])) z_axis
      )
)

-------------------------------------------------------------------------------
-- Rutine       : rob a
-- Function       : Dibuja robot
```

```
-- Parameters    : a-> instrucción de entrada con valor de 1
-- Return value : ninguno
--------------------------------------------------------------------------------
function rob a =
(
      --Base del robot
    d1=89.5; a2=43; d4=43; d6=7;
      Base=cylinder radius:18 height:42 name:"oEsBas" wireColor:[229,70,7]
pivot:[0,0,0] pos:[0,0,0];

      --Primer Eslabón
      Es1_1=box length:36 width:43 height:9 name:"oEs1_1" wireColor:[229,70,7]
      $oEs1_1.rotation.z_rotation=90;
      $oEs1_1.pos=[0,0,a2]; --[.3,0,47]; --pivot:[-3.2,0,4.5] pos:[.3,0,47];

      Es1_2=box length:36 width:12 height:56.5 name:"oEs1_2"
wireColor:[229,70,7]
      $oEs1_2.rotation.z_rotation=90;
      $oEs1_2.pos=[0,(18.7),a2]; -- pivot:[0,0,4.5] pos:[19.0,0,47];

      Es1_3=cylinder radius:9 height:20.2 name:"oEs1_3" wireColor:[229,70,7]; --
pivot:[0,0,0] pos:[-7.5,0,89.5];
      $oEs1_3.rotation.x_rotation=90
      $oEs1_3.pos=[0,12.7,d1];

    group $oEs1_* name:"oEs1";
      $oEs1.pivot=[0,0,0];
      $oEs1.pos=[0,-1.5,0];
    $oEs1.parent=$oEsBas

      --Segundo Eslabón
      mEs2=box length:29.7 width:10.8 height:87 name:"oEs2" wireColor:[229,70,7]
      $oEs2.dir=[1,0,0];
      $oEs2.pivot=[35.3,0,0];
      $oEs2.pos=[0,(-12.9),d1];
    $oEs2.parent=$oEs1

      --Tercer Eslabón
      mEs3=box length:25.5 width:13.5 height:45 name:"oEs3" wireColor:[229,70,7]
      $oEs3.rotation.z_rotation=90;

      $oEs3.pivot=[0,0,28.7];
    $oEs3.pos=[a2,0,d1]
      $oEs3.parent=$oEs2

      --Cuarto Eslabón
      mEs4=box length:8.5 width:8.5 height:21.7 name:"oEs4" wireColor:[229,70,7]
pivot:[0,0,0] pos:[(a2),0,(d1+16.3)]
      $oEs4.parent=$oEs3

      --Quinto Eslabón
      mEs5=cylinder radius:4 height:8.5 name:"oEs5" wireColor:[229,70,7]
      $oEs5.dir=[0,1,0];
      $oEs5.pivot=[0,4,0];
      $oEs5.pos=[a2,0,(d1+d4)];
      $oEs5.parent=$oEs4;
```

```
      mEs51=cylinder radius:4 height:2.5 name:"oEs51" wireColor:[149,197,178]
      $oEs51.dir=[0,1,0];
      $oEs51.pivot=[0,4,0];
      $oEs51.pos=[a2,-4,(d1+d4)];
      $oEs51.parent=$oEs4;

      --Sexto Eslabón
      mEs6=cylinder radius:4 height:1 name:"oEs6" wireColor:[229,70,7]
pivot:[0,0,0] pos:[a2,0,(d1+d4+d6)]
      $oEs6.parent=$oEs5
)

-------------------------------------------------------------------------------
-- Rutine       : delrob a
-- Function       : Borra robot
-- Parameters   : a-> instrucción de entrada con valor de 1
-- Return value : ninguno
-------------------------------------------------------------------------------
function delRob a =
(
      delete $oEs1;
      delete $oEs2;
      delete $oEs3;
      delete $oEs4;
      delete $oEs5;
      delete $oEs51;
      delete $oEs6;
      delete $oEsbas;
)

-------------------------------------------------------------------------------
-- Rutine       : delOri a
-- Function       : Borra orientación y punto de la pantalla
-- Parameters   : a-> instrucción de entrada con valor de 1
-- Return value : ninguno
-------------------------------------------------------------------------------
function delOri a =
(
      delete $oOri;
      delete $oPos;
)
-------------------------------------------------------------------------------
-- Rutine       : delOriPT n
-- Function       : Borra orientaciones y puntos de la pantalla en Planeacion
Trayectorias
-- Parameters   : n-> número de puntos de la trayectoria
--                      vPirOrient-> vector de orientaciones de la trayectoria
--                      ss->linea de conexion entre puntos
-- Return value : ninguno
-------------------------------------------------------------------------------
function delOriPT vPirOrient ss n =
(
      for i=1 to n do
      (
            delete vPirOrient[i];
            delete $oPos;
      )
```

```
   deletespline ss (1)
)

-------------------------------------------------------------------------------
-- Rutine        : DirKinet a1 a2 a3 a4 a5 a6
-- Function        : Obtiene la Cinemática Directa
-- Parameters    : a1,a2,a3,a4,a5,a6-> ángulos de rotación de los eslabones
-- Return value : matriz T06 de 4x4
-------------------------------------------------------------------------------
function DirKinet at1 at2 at3 at4 at5 at6 =
(
      a=#(0,0,0,0,0,0);
      a[1]=0;
      a[2]=43;
      a[3]=0;
      a[4]=0;
      a[5]=0;
      a[6]=0;
      b=#(0,0,0,0,0,0);
      b[1]=89.5;
      b[2]=0;
      b[3]=0;
      b[4]=43;
      b[5]=0;
      b[6]=7;
      al=#(0,0,0,0,0,0);
      al[1]=90;
      al[2]=180;
      al[3]=90;
      al[4]=90;
      al[5]=90;
      al[6]=0;

      th=#(at1,at2,at3,at4,at5,at6);
      ct=#(0,0,0,0,0,0);
      st=#(0,0,0,0,0,0);
      l=#(0,0,0,0,0,0);
      u=#(0,0,0,0,0,0);
      tT1=#(#(0,0,0,0),#(0,0,0,0),#(0,0,0,0),#(0,0,0,0));
      tT2=#(#(0,0,0,0),#(0,0,0,0),#(0,0,0,0),#(0,0,0,0));
      tT3=#(#(0,0,0,0),#(0,0,0,0),#(0,0,0,0),#(0,0,0,0));
      tT4=#(#(0,0,0,0),#(0,0,0,0),#(0,0,0,0),#(0,0,0,0));
      tT5=#(#(0,0,0,0),#(0,0,0,0),#(0,0,0,0),#(0,0,0,0));
      tT6=#(#(0,0,0,0),#(0,0,0,0),#(0,0,0,0),#(0,0,0,0));
      for i=1 to 6 do
      (
         ct[i]=cos(th[i]);
         if abs(ct[i])<1e-7 then
            ct[i]=0;
         st[i]=sin(th[i]);
         if abs(st[i])<1e-7 then
            st[i]=0;
         l[i]=cos(al[i]);
         if abs(l[i])<1e-7 then
            l[i]=0;
         u[i]=sin(al[i]);
         if abs(u[i])<1e-7 then
```

```
      u[i]=0;
     )
     for i=1 to 6 do
     (
      --%Obtención de la matriz T de 4x4
        t1=#(ct[i],(-st[i]*l[i]),( st[i]*u[i]),( a[i]*ct[i]));
        t2=#(st[i],( ct[i]*l[i]),(-ct[i]*u[i]),( a[i]*st[i]));
        t3=#(0,     u[i],              l[i],       b[i]);
        t4=#(0,0,0,1);

        q1=#(ct[i],-st[i]*l[i], st[i]*u[i]);
        q2=#(st[i], ct[i]*l[i],-ct[i]*u[i]);
        q3=#(0,     u[i],              l[i]   );
     --Obtención del vector a de 3x1
        aa1=a[i]*ct[i];
        aa2=a[i]*st[i];
        aa3=b[i];

           if i== 1 then (
                 tT1=#(t1, t2, t3, t4);
                 tT1=elimE12 tT1 4 4 )
           if i== 2 then (
                tT2=#(t1, t2, t3, t4);
                 tT2=elimE12 tT2 4 4   )
           if i== 3 then (
                 tT3=#(t1, t2, t3, t4);
                 tT3=elimE12 tT3 4 4       )
           if i== 4 then (
                 tT4=#(t1, t2, t3, t4);
                 tT4=elimE12 tT4 4 4       )
           if i== 5 then (
                 tT5=#(t1, t2, t3, t4);
                 tT5=elimE12 tT5 4 4       )
           if i== 6 then (
                 tT6=#(t1, t2, t3, t4);
                 tT6=elimE12 tT6 4 4       )
     )

     T12=multmat tT1 tT2 4;
     T13=multmat T12 tT3 4;
     T14=multmat T13 tT4 4;
     T15=multmat T14 tT5 4;
     T16=multmat T15 tT6 4;

     T06= multmat (multmat(multmat (multmat (multmat tT1 tT2 4) tT3 4) tT4 4)
tT5 4) tT6 4
     pto=#(T06[1][4],T06[2][4],T06[3][4])

     return T06;
)

-------------------------------------------------------------------------------
-- Rutine       : Anima t1 t2 t3 t41 t51 t61 t42 t52 t62 k n
-- Function       : Simula los movimientos del robot en el seguimiento de n
puntos
--                        de una trayectoria especificada
```

```
-- Parameters    : t1,t2,t3,t41,t51,t61,t42,t52,t62-> ángulos de rotación de los
eslabones
--                          k-> número de solucción
--                          n-> número de puntos
-- Return value : ninguno
-------------------------------------------------------------------------------
function Anima t1 t2 t3 t41 t51 t61 t42 t52 t62 k n =
(
      kk=0;
      if k>4 then (k=k-4; kk=1;)
      d1=89.5;
      --rob 1
      --    mEsfera=sphere name:"EE" position:[vPos[i,1],vPos[i,2],(vPos[i,3])]
radius:2;
      --for i=1 to nP do
      --(
      --    mEsfera=sphere name:"EE"
position:[vPos[i][1],vPos[i][2],(vPos[i][3])] radius:2
      --)

animate on
(
for i=1 to n do
(
      if i==1 then
      (
          at time (i*10) in coordsys local rotate $oEsBas 0 z_axis
          at time (i*10) in coordsys local rotate $oEs1  (t1[i][k]) z_axis
          at time (i*10) in coordsys local rotate $oEs2  (-t2[i][k]) _axis
          at time (i*10) in coordsys local rotate $oEs3  (t3[i][k]) x_axis
            if kk==0 then
            (
             at time (i*10) in coordsys local rotate $oEs4  (t41[i][k]) z_axis
             at time (i*10) in coordsys local rotate $oEs5  -(180+t51[i][k])
z_axis
                          at time (i*10) in coordsys local rotate $oEs6  (-
t61[i][k]) z_axis
                        )else
                        (  at time (i*10) in coordsys local rotate $oEs4
(t42[i][k]) z_axis
                          at time (i*10) in coordsys local rotate $oEs5  (180-
t52[i][k]) z_axis
                          at time (i*10) in coordsys local rotate $oEs6  (-
t62[i][k]) z_axis
                        )
                )
                else
                (
                at time (i*10) in coordsys local rotate $oEsBas 0 z_axis
                at time (i*10) in coordsys local rotate $oEs1  ((t1[i][k]-
t1[i-1][k])) z_axis
                      at time (i*10) in coordsys local rotate $oEs2  (-
(t2[i][k]-t2[i-1][k])) x_axis
                      at time (i*10) in coordsys local rotate $oEs3  ((t3[i][k]-
t3[i-1][k])) x_axis
                        if kk==0 then
                        (
```

```
                                at time (i*10) in coordsys local rotate $oEs4
((t41[i][k]-t41[i-1][k])) z_axis
                                at time (i*10) in coordsys local rotate $oEs5  ( -
((t51[i][k]-t51[i-1][k] ))) z_axis
                                  at time (i*10) in coordsys local rotate $oEs6  (-
(t61[i][k]-t61[i-1][k])) z_axis
                            )else
                            (
                                at time (i*10) in coordsys local rotate $oEs4
((t42[i][k]-t42[i-1][k])) z_axis
                                at time (i*10) in coordsys local rotate $oEs5  ( -
((t52[i][k]-t52[i-1][k] ))) z_axis
                                  at time (i*10) in coordsys local rotate $oEs6  (-
(t62[i][k]-t62[i-1][k])) z_axis
                            )
                    )
            )
      )
)
-------------------------------------------------------------------------------
-- Rutine       : RotaPivote eslab eje grados
-- Function      : Rota el eje (X,Y o Z) Pivote de un eslabón (grados) grados
-- Parameters    : eslab-> eslabón
--                       eje-> eje (X,Y o Z)
--                       grados-> grados
-- Return value : eslab-> el eslabón con el pivote modificado
-- Utility       : "Crear Robot" "Robot Defoult"
-------------------------------------------------------------------------------
function  RotaPivote eslab eje grados =
(
      if eje == 1 then(
            in coordsys local
            eslab.rotation.x_rotation =grados )
      if eje == 2 then(
            in coordsys local
            eslab.rotation.y_rotation =grados )
      if eje == 3 then(
            in coordsys local
            eslab.rotation.z_rotation =grados )
      return eslab
)

-------------------------------------------------------------------------------
-- Rutine       : RobArbit a
-- Function      : Crea un robot de una configuración arbitraria.
--                       Permite que el usuario especifique parametros de la
configuración del robot.
-- Parameters    : a -> instrucción de entrada con valor de 1
-- Return value : ninguno
-- Utility       : "Crear Robot" "Robot Defoult"
-------------------------------------------------------------------------------
function RobArbit a =(

      for i=1 to 6 do
      (
            if vForEs[i]==1 then
                  esb=cylinder radius:vRadCil[i] height:vAltCil[i]
```

```
            else
                  esb=box length:vlenCub[i] width:vwidCub[i] height:vheiCub[i]
            esb= RotaPivote esb vRP1[i] vVRP1[i]
            esb= RotaPivote esb vRP2[i] vVRP2[i]
            esb= RotaPivote esb vRP3[i] vVRP3[i]
            esb.pivot=[vXpp[i],vYpp[i],vZpp[i]];
            esb.pos=[vXpe[i],vYpe[i],vZpe[i]];

            if i==1 then      (     es1=esb;
                  es1.name="oEs1"; )
            if i==2 then      (     es2=esb;
                  es2.name="oEs2"; )
            if i==3 then      (     es3=esb;
                  es3.name="oEs3"; )
            if i==4 then      (     es4=esb;
                  es4.name="oEs4"; )
            if i==5 then      (     es5=esb;
                  es5.name="oEs5"; )
            if i==6 then      (     es6=esb;
                  es6.name="oEs6"; )
      )
      $oEs6.parent=$oEs5;
      $oEs5.parent=$oEs4;
      $oEs4.parent=$oEs3;
      $oEs3.parent=$oEs2;
      $oEs2.parent=$oEs1;
)
-------------------------------------------------------------------------------
-- Rutine       : robotDefault a
-- Function     : Crea un robot de una configuración arbitraria con valores
Default.
--                      Se considera los valores del robot Cloos
-- Parameters   : a -> instrucción de entrada con valor de 1
-- Return value : ninguno
-- Utility      : "Robot Default"
-------------------------------------------------------------------------------

function robotDefault a=
(     d1=89.5
      a2=43
      d4=43
      d6=7

      vForEs=#(1,2,2,2,1,1) --Forma del eslabón
      vRP1=#(0,1,1,0,1,3)                              --1ra. rotación del eje
Pivote
      vVRP1=#(0,90,-90,0,90,180)                       --1er. valor para la
rotación del eje Pivote
      vRP2=#(0,0,0,0,0,0)                              --2do. rotación del eje
Pivote
      vVRP2=#(0,0,0,0,0,0)                       --2do. valor para la rotación
del eje Pivote
      vRP3=#(0,0,0,0,0,0)                              --3er. rotación del eje
Pivote
      vVRP3=#(0,0,0,0,0,0)                       --3er. valor para la rotación
del eje Pivote
      vRadCil=#(18,0,0,0,4,4)                    --Valor del radio del cilindro
```

```
      vAltCil=#(d1,0,0,0,8.5,1)                          --Valor de la altura del
cilindro
      vlenCub=#(0,29.7,16,8.5,0,0)                 --Valor de length del cubo
      vwidCub=#(0,a2,25,8.5,-4.25,0)                     --Valor de width del
cubo
      vheiCub=#(0,10.8,13.5,21.7,0,0)                    --Valor de height del
cubo
      vXpp=#(0,(-a2/2),0,0,0,0)                          --Valor de posición X
del pivote
      vYpp=#(0,0,0,0,-4.25,0)                        --Valor de posición Y del
pivote
      vZpp=#(0,0,-8,0,0,0)                           --Valor de posición Z del
pivote
      vXpe=#(0,0,a2,a2,a2,a2)                        --Valor de posición X del
eslabón
      vYpe=#(0,0,0,0,0,0)                            --Valor de posición Y del
eslabón
      vZpe=#(0,d1,d1,(d1+16),(d1+d4),(d1+d4+d6))                              --
Valor de posición Z del eslabón
      vNE=#(1,2,3,4,5,6)                             --Numero de eslabón creado
)

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Crea un Panel de Utilidades que despliega una persiana de iteracción
-- C I N E M A T I C A    D I R E C T A
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
utility CinDir "Cinemátca Directa"
      (
      local pot,  oldr1=0,oldr2=0,oldr3=0,oldr4=0,oldr5=0,oldr6=0, robExist=0
      local mT06

-------------------------------------------------------------------------------
-- Rutine       : "Robot", "Rotar Eslabones"
-- Function     : Elementos de la Persiana
-- Parameters   :
-- Return value : ninguno
-------------------------------------------------------------------------------
            group "Robot"
            (
                  button brob "Robot" width:100 height:20
                  button bdelrob "Borrar" width:100 height:20
                  button breset  "Reset" width:100 height:20
            )
            group "Rotar Eslabones"
            (
                  spinner r1 "Eslabón 1" range:[-30,130,0] type:#integer
                  spinner r2 "Eslabón 2" range:[-30,130,0] type:#integer
                  spinner r3 "Eslabón 3" range:[-30,130,0] type:#integer
                  spinner r4 "Eslabón 4" range:[-30,130,0] type:#integer
                  spinner r5 "Eslabón 5" range:[-30,130,0] type:#integer
                  spinner r6 "Eslabón 6" range:[-30,130,0] type:#integer
            )
            group "Cinemática Directa"
            (
                  button bCD "Cinemática Directa" width:130 height:20
```

```
)
-------------------------------------------------------------------------------
-- Rutine       : Ordenes para los diferentes elementos de las persianas
-- Function     : Elementos de la Persiana
-- Parameters   :
-- Return value : ninguno
-------------------------------------------------------------------------------

            on brob pressed do
            (  if robExist== 0 then (rob 1
                 --r1.value=0; r2.value=0; r3.value=0; r4.value=0; r5.value=0;
r6.value=0;
                 --oldr1=0,oldr2=0;oldr3=0;oldr4=0;oldr5=0;oldr6=0;
                 in coordsys local rotate $oEs1  r1.value z_axis
                in coordsys local rotate  $oEs2  -r2.value x_axis
                in coordsys local rotate  $oEs3  r3.value x_axis
            in coordsys local rotate $oEs4  r4.value z_axis
                in coordsys local rotate  $oEs5  r5.value z_axis
            in coordsys local rotate $oEs6  r6.value z_axis
                 robExist=1; )
            )
            on bdelrob pressed do
            (  if robExist==1 then (  delrob 1
                 r1.value=0; r2.value=0; r3.value=0; r4.value=0; r5.value=0;
r6.value=0;
                 oldr1=0;oldr2=0;oldr3=0;oldr4=0;oldr5=0;oldr6=0; robExist=0; )
            )
            on breset pressed do
            (
                 if robExist==1 then (
                 in coordsys local rotate $oEs1  -r1.value z_axis
                in coordsys local rotate $oEs2  r2.value x_axis
                in coordsys local rotate $oEs3  -r3.value x_axis
            in coordsys local rotate $oEs4  -r4.value z_axis
                in coordsys local rotate $oEs5  -r5.value z_axis
            in coordsys local rotate $oEs6  r6.value z_axis
                 r1.value=0; r2.value=0; r3.value=0; r4.value=0; r5.value=0;
r6.value=0;
                 oldr1=0;oldr2=0;oldr3=0;oldr4=0;oldr5=0;oldr6=0; )
            )
            on r1 changed value do
            (    if robExist==1 then (
                 in coordsys local rotate $oEs1  (r1.value-oldr1) z_axis
                 oldr1=r1.value; )
            )
            on r2 changed value do
            (    if robExist==1 then (
                 in coordsys local rotate $oEs2  (-r2.value+oldr2) x_axis
            oldr2=r2.value;)
            )
            on r3 changed value do
            (    if robExist==1 then (
                 in coordsys local rotate $oEs3  (r3.value-oldr3) x_axis
                 oldr3=r3.value;)
            )
            on r4 changed value do
            (    if robExist==1 then (
```

```
            in coordsys local rotate $oEs4  (r4.value-oldr4) z_axis
            oldr4=r4.value;)
        )
        on r5 changed value do
        (     if robExist==1 then (
            in coordsys local rotate $oEs5  (r5.value-oldr5) z_axis
           oldr5=r5.value;)
        )
        on r6 changed value do
        (     if robExist==1 then (
            in coordsys local rotate $oEs6  (-r6.value+oldr6) z_axis
            oldr6=r6.value;)
        )
        on bCD pressed do
        (
          mT06=DirKinet r1.value r2.value r3.value r4.value (r5.value+180)
r6.value
            format "\n Matriz de Orientación"
            --print mT06
            format "\n% \t% \t%" mT06[1][1] mT06[1][2] mT06[1][3]
            format "\n% \t% \t%" mT06[2][1] mT06[2][2] mT06[2][3]
            format "\n% \t% \t%" mT06[3][1] mT06[3][2] mT06[3][3]
            format "\n\n Vector de Posición"
            format "\nPosición en X: %" mT06[1][4]
            format "\nPosición en Y: %" mT06[2][4]
            format "\nPosición en Z: %" mT06[3][4]
            format "\n"
            -- Especificar valores de angulos si se desea generar archivo
            vt1=r1; vt2=r2; vt3=r3; vt4=r4; vt5=r5; vt6=r6;
            nTraPtos=1;
        )
    )--fin de Cinemática Directa


------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- Crea un Panel de Utilidades que despliega una persiana de iteracción
-- C I N E M A T I C A    I N V E R S A
------------------------------------------------------------------------------
------------------------------------------------------------------------------
    utility CinInv "Cinemática Inversa"
    (
        -- Declaración de Variables
        local pot, robExist=0, oriExist=0, robRotar=0
        local eOri,ePos
        local oldX=0, oldY=0, oldZ=0, iX=50, iY=-50, iZ=80
        local pPto, pOri, pAng, pPto1
        local b1=89.5,a2=43,b4=43,b6=7, err=0
        local erX,erY,erZ,h2,h

------------------------------------------------------------------------------
-- Rutine       : "Robot", "Posición", "Orientación" y "Cinemática Inversa"
-- Function      : Elementos de la Persiana
-- Parameters    :
-- Return value : ninguno
------------------------------------------------------------------------------
        group "Robot"
```

```
            (
                  button brob "Robot" width:100 height:20
                  button bdelrob "Borrar" width:100 height:20
                  button breset  "Reset" width:100 height:20
            )
            group "Posición"
          (
                  spinner rX "Punto en X" range:[-100,100,iX] type:#integer
                  spinner rY "Punto en Y" range:[-100,100,iY] type:#integer
                  spinner rZ "Punto en Z" range:[-300,300,iZ] type:#integer
            )
            group "Orientación"
         (
                  spinner orX "Orient  X" range:[-rgoGrados,rgoGrados,oldX]
type:#integer
                  spinner orY "Orient  Y" range:[-rgoGrados,rgoGrados,oldY]
type:#integer
                  spinner orZ "Orient  Z" range:[-rgoGrados,rgoGrados,oldZ]
type:#integer
            )
            group "Cinemática Inversa"
         (
                  button bOri  "Ver Punto y Orientación" width:130 height:20
                  spinner oSol "Solución " range:[1,8,1] type:#integer
                  button bCI "Cinemática Inversa" width:130 height:20
            )
-------------------------------------------------------------------------------
-- Rutine       : Ordenes para los diferentes elementos de las persianas
-- Function     : Elementos de la Persiana
-- Parameters   :
-- Return value : ninguno
-------------------------------------------------------------------------------
            on brob pressed do
            (  if robExist==0 then rob 1
               robExist=1; robRotar=0;
            )
            on bdelrob pressed do
            (  if robExist==1 then  delrob 1
               if oriExist==1 then  delori 1
               robExist=0; oriExist=0; robRotar=0;
            )
            on breset pressed do
            (   if robExist==1 then (
                  delrob 1
                  rob 1
                  robRotar=0;)
            )
            on bOri pressed do
            (   if OriExist ==1 then delori 1
                  eOri= pyramid width:7 depth:7 height:30 name:"oOri"
pivot:[0,0,30]
                $oOri.rotation.z_rotation=180
                  $oOri.pos=[rX.value, rY.value, rZ.value];
                  in coordsys local rotate $oOri (orZ.value) z_axis
                  in coordsys local rotate $oOri (orY.value) y_axis
                  in coordsys local rotate $oOri (orX.value) x_axis
```

```
                Pos= sphere radius:2 name:"oPos"
pos:[rX.value,rY.value,rZ.value];
                OriExist=1;
            )
            on bCI pressed do
            (   --Evalua si el punto es alcanzable por el manipulador
                err=0
                h=sqrt(rX.value^2+rY.value^2);
                h2=sqrt(rX.value^2+rY.value^2+(rZ.value-b1)^2);
                if ( ( rZ.value <= (b1+a2+b4+b6)) and (rZ.value >= (b1-
(a2+b4+b6)) )) then erZ=0 else erZ=1
                if (abs(rY.value) <= (a2+b4+b6)) then erY=0 else erY=1
                if (abs(rX.value) <= (a2+b4+b6) ) then erX=0 else erX=1

                if erY==0 and erX==0 and erZ==0 then
                (     if (rZ.value < b1 and h<=(a2+b4+b6)) then err =0 else
                      if (rZ.value >= b1 and h2<=(a2+b4+b6)) then err =0 else
                      (err=1;
                      messagebox "El punto esta fuera del espacio de trabajo
del robot \n Elija otro punto"
                      )

                )
                if err==0 then
                      if erX==1 or erY==1 or erZ==1 then
                      (err=1;
                            messagebox "El punto esta fuera del espacio de
trabajo del robot \n Elija otro punto"
                      )

                if err==0  then(
                pPto=#(rX.value, rY.value, rZ.value );
                pPto1=#(rX.value, rY.value, (rZ.value-89.5) );
              pOri=#(orX.value, orY.value, orZ.value );

                ang=InvKinet pPto1 pOri
                t1=ang[1];              t2=ang[2];              t3=ang[3];
                t41=ang[4];             t42=ang[5];             t51=ang[6];
                t52=ang[7];             t61=ang[8];             t62=ang[9];
                if robExist == 0 then   ( rob 1; robExist=1;)
                if robRotar == 1 then if robExist == 1 then (delrob 1; rob 1;)

                rotar t1 t2 t3 t41 t51 t61 t42 t52 t62 pPto oSol.value
                robRotar=1;
                err=0;
                -- Asignar valores de los angulos a variables globales para si
se quiere generar archivo
                vt1=t1; vt2=t2; vt3=t3;
                vt41=t41; vt51=t51; vt61=t61;
                vt42=t42; vt52=t52; vt62=t62;
                nTraPtos=1;
          )


      )--fin de Cinemática Inversa
)
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Crea un Panel de Utilidades que despliega una persiana de iteracción
-- P A R A M E T R O S  D E  D E N A V I T - H A R T E N B E R G
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
utility ParamDH "Denavit-Hartenberg"
      (
            -- Declaración de Variables
            local b1=89.5,a2=43,b4=43,b6=7
            local a,b,alfa;
-------------------------------------------------------------------------------
-- Rutine       : "Valores a", "Valores b", "Valores Alfa" y "Par. Denavit-
Hartenberg"
-- Function     : Elementos de la Persiana
-- Parameters   :
-- Return value : ninguno
-------------------------------------------------------------------------------
            group "Valores de  ' a '"
            (     spinner dha1 "Eslabón 1" range:[-rgoValor,rgoValor,0]
type:#float
                  spinner dha2 "Eslabón 2" range:[-rgoValor,rgoValor,a2]
type:#float
                  spinner dha3 "Eslabón 3" range:[-rgoValor,rgoValor,0]
type:#float
                  spinner dha4 "Eslabón 4" range:[-rgoValor,rgoValor,0]
type:#float
                  spinner dha5 "Eslabón 5" range:[-rgoValor,rgoValor,0]
type:#float
                  spinner dha6 "Eslabón 6" range:[-rgoValor,rgoValor,0]
type:#float
            )
            group "Valores de  ' b '"
            (     spinner dhb1 "Eslabón 1" range:[-rgoValor,rgoValor,b1]
type:#float
                  spinner dhb2 "Eslabón 2" range:[-rgoValor,rgoValor,0]
type:#float
                  spinner dhb3 "Eslabón 3" range:[-rgoValor,rgoValor,0]
type:#float
                  spinner dhb4 "Eslabón 4" range:[-rgoValor,rgoValor,b4]
type:#float
                  spinner dhb5 "Eslabón 5" range:[-rgoValor,rgoValor,0]
type:#float
                  spinner dhb6 "Eslabón 6" range:[-rgoValor,rgoValor,b6]
type:#float
            )
            group "Valores de  ' alpha '"
            (     label dat "valores en grados"
                  spinner dhf1 "Eslabón 1" range:[-rgoGrados,rgoGrados,90]
type:#float
                  spinner dhf2 "Eslabón 2" range:[-rgoGrados,rgoGrados,180]
type:#float
                  spinner dhf3 "Eslabón 3" range:[-rgoGrados,rgoGrados,90]
type:#float
                  spinner dhf4 "Eslabón 4" range:[-rgoGrados,rgoGrados,90]
type:#float
```

```
                spinner dhf5 "Eslabón 5" range:[-rgoGrados,rgoGrados,90]
type:#float
                spinner dhf6 "Eslabón 6" range:[-rgoGrados,rgoGrados,0]
type:#float
            )
            group "Par. Denavit-Hartenberg"
        (   button bDH  "Grabar" width:130 height:40
            )

      ------------------------------------------------------------------------
-- Rutine       : Ordenes para los diferentes elementos de las persianas
-- Function     : Elementos de la Persiana
-- Parameters   :
-- Return value : ninguno
------------------------------------------------------------------------------
            on bDH pressed do
            (   --Asignar los valores a los parámetros de Denavit-Hartenberg
                  a=#(0,0,0,0,0,0);
                  b=#(0,0,0,0,0,0);
                  alfa=#(0,0,0,0,0,0);
                  a[1]=dha1.value;  a[2]=dha2.value;  a[3]=dha3.value;
                  a[4]=dha4.value;  a[5]=dha5.value;  a[6]=dha6.value;
                  b[1]=dhb1.value;  b[2]=dhb2.value;  b[3]=dhb3.value;
                  b[4]=dhb4.value;  b[5]=dhb5.value;  b[6]=dhb6.value;
                  alfa[1]=dhf1.value;     alfa[2]=dha2.value;
      alfa[3]=dha3.value;
                  alfa[4]=dhf4.value;     alfa[5]=dha5.value;
      alfa[6]=dha6.value;

                  -- Desplegar información en pantalla
                  format "\nParámetros de Denavit-Hartenberg"
                  format "\nvalores de a:"
                  format "\n% \t% \t% \t% \t% \t%" a[1] a[2] a[3] a[4] a[5] a[6]
                  format "\nvalores de b:"
                  format "\n% \t% \t% \t% \t% \t%" b[1] b[2] b[3] b[4] b[5] b[6]
                  format "\nvalores de alpha:"
                  format "\n% \t% \t% \t% \t% \t%" alfa[1] alfa[2] alfa[3]
alfa[4] alfa[5] alfa[6]
                  format "\n"
            )
      )--fin de Parámetros de Denavit-Hartenberg


------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- Crea un Panel de Utilidades que despliega una persiana de iteracción
-- P L A N E A C I O N  D E  T R A Y E C T O R I A S
-- INTERPOLACIONES
------------------------------------------------------------------------------
------------------------------------------------------------------------------
utility TrajPlan "Trayectoria punto a punto"
      (
            -- Declaración de Variables
            local pot, robExist=0, oriExist=0, robRotar=0
            local eOri,ePos
            local oldX=0, oldY=0, oldZ=0, iX=50, iY=-50, iZ=80
            local pPto, pOri, pAng, pPto1
```

```
            local b1=89.5,a2=43,b4=43,b6=7, err=0
            local erX,erY,erZ,h2,h
            local a,b,alfa;
------------------------------------------------------------------------------
-- Rutine       : "Robot" "Posición" "Orientación" "Planeación de Trayectoria"
-- Function     : Elementos de la Persiana
-- Parameters   :
-- Return value : ninguno
------------------------------------------------------------------------------
      group "Robot"
            (
                  button brob "Robot" width:100 height:20
                  button bdelrob "Borrar" width:100 height:20
                  button breset  "Reset" width:100 height:20
            )
            group "Posición"
          (
                  spinner rX "Punto en X" range:[-100,100,iX] type:#integer
                  spinner rY "Punto en Y" range:[-100,100,iY] type:#integer
                  spinner rZ "Punto en Z" range:[-300,300,iZ] type:#integer
            )
            group "Orientación"
        (
                  spinner orX "Orient  X" range:[-rgoGrados,rgoGrados,oldX]
type:#integer
                  spinner orY "Orient  Y" range:[-rgoGrados,rgoGrados,oldY]
type:#integer
                  spinner orZ "Orient  Z" range:[-rgoGrados,rgoGrados,oldZ]
type:#integer
            )
            group "Planeación de Trayectorias"
            (
                  label lInter "Interpolaciones"
                  button bAddPto "Agregar punto" width:135 height:20
                  button bVerPtos  "Ver Puntos de trayectoria" width:135
height:20
                  spinner oSolPT "Solución " range:[1,8,1] type:#integer
                  button bPT "Planeación Trayectoria" width:135 height:20
            )
------------------------------------------------------------------------------
-- Rutine       : Ordenes para los diferentes elementos de las persianas
-- Function     : Elementos de la Persiana
-- Parameters   :
-- Return value : ninguno
------------------------------------------------------------------------------
            local vPtos=#(0), vOri=#(0),n=0,PtosExist=0, nVPtos=0,i;
            local ss

            local eOriPto=#(0);
            local agrePtos=0
            on bAddPto pressed do
            (   n=n+1; i=n;
                  if n<=15 then (
                        vPtos[i]=#(rX.value,  rY.value,  rZ.value);
                        vOri[i]=#(orX.value,  orY.value,  orZ.value);
                  )
                  print vPtos
```

```
            print vOri
            agrePtos=1;
        )
        on bVerPtos pressed do
        (   if n>1 then
            (   if agrePtos==1 and PtosExist== 1 then
                (   delOriPT eOriPto ss nVPtos
                    PtosExist=0;
                )
                if PtosExist ==0 or agrePtos==1  then(
                ss=splineshape()
                addnewspline ss
                for i=1 to n do
                (
                    eOriPto[i]= pyramid width:7 depth:7 height:30
pivot:[0,0,30]
                eOriPto[i].rotation.z_rotation=180
                    eOriPto[i].pos=[vPtos[i][1], vPtos[i][2],
vPtos[i][3]];
                    in coordsys local rotate eOriPto[i] (vOri[i][3])
z_axis
                    in coordsys local rotate eOriPto[i] (vOri[i][2])
y_axis
                    in coordsys local rotate eOriPto[i] (vOri[i][1])
x_axis
                    Pos= sphere radius:2 name:"oPos" pos:[vPtos[i][1],
vPtos[i][2], vPtos[i][3]];
                    addknot ss 1 #corner #curve [vPtos[i][1],
vPtos[i][2], vPtos[i][3]]
                )
                PtosExist=1;
                nVPtos=n;
                --close ss 1
                updateshape ss
                agrePtos=0;

                )
            )else messagebox "Agregar al menos dos puntos para la
planeación de trayectorias"
        )

        on bPT pressed do
        (   if n>0 then
            (
                for i= 1 to n do
                (   pPto1=#(vPtos[i][1],vPtos[i][2],(vPtos[i][3]-89.5)
);
                pOri=#(orX.value, orY.value, orZ.value );

                ang=InvKinet pPto1 vOri[i]
                vt1[i]=ang[1];                  vt2[i]=ang[2];
        vt3[i]=ang[3];
                vt41[i]=ang[4];                 vt42[i]=ang[5];
        vt51[i]=ang[6];
                vt52[i]=ang[7];                 vt61[i]=ang[8];
        vt62[i]=ang[9];
                if robExist == 0 then   ( rob 1; robExist=1;)
```

```
                        if robRotar == 1 then if robExist == 1 then
(delrob 1; rob 1;)
                    )
                    Anima vt1 vt2 vt3 vt41 vt51 vt61 vt42 vt52 vt62
oSolPT.value nVPtos
                    -- Asignar valores de los angulos a variables globales
para si se quiere generar archivo
                    nTraPtos=nVPtos;

            )
        )
        on brob pressed do
        (  if robExist==0 then rob 1
           robExist=1; robRotar=0;
        )
        on bdelrob pressed do
        (  if robExist==1 then  delrob 1
           if ptosExist==1 then  deloriPT  eOriPto ss n
           robExist=0; ptosExist=0; robRotar=0;
        )
        on breset pressed do
        (   if robExist==1 then (
              delrob 1
              rob 1
              robRotar=0;)
        )

)--fin de Trayectoria Spline


-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Crea un Panel de Utilidades que despliega una persiana de iteracción
-- P L A N E A C I O N  D E  T R A Y E C T O R I A S
-- TRAYECTORIAS CONTINUAS (ECUACIÓN PARAMÉTRICA)
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
utility TrajCont "Trarectoria Contínua"
      (
            -- Declaración de Variables
            local pot, robExist=0, oriExist=0, robRotar=0
            local eOri,ePos
            local oldX=0, oldY=0, oldZ=0, iX=50, iY=-50, iZ=80
            local pPto, pOri, pAng, pPto1
            local b1=89.5,a2=43,b4=43,b6=7, err=0
            local erX,erY,erZ,h2,h
            local a,b,alfa;
-------------------------------------------------------------------------------
-- Rutine       : "Robot" "Ecuación Paramétrica" "Intervalo" "Orientación"
--                      "Planeación de Trayectorias"
-- Function      : Elementos de la Persiana
-- Parameters    :
-- Return value : ninguno
-------------------------------------------------------------------------------
            group "Robot"
            (
                  button brob "Robot" width:100 height:20
```

```
            button bdelrob "Borrar" width:100 height:20
            button breset  "Reset" width:100 height:20
        )
        group "Ecuación Paramétrica"
      ( label lep "Ecuación Circunferencia"
            spinner oRad "Radio" range:[0,rgoValor,50] type:#integer
            spinner oRPX "Pos X" range:[-rgoValor,rgoValor,0]
type:#integer
            spinner oRPY "Pos Y" range:[-rgoValor,rgoValor,0]
type:#integer
            spinner oRPZ "Altura Z" range:[-rgoValor,rgoValor,129.5]
type:#integer

        )
        group "Intervalo"
        (     spinner oInter "Intervalo" range:[0,rgoValor,30] type:#integer
        )
        group "Orientación"
    (
            spinner orX "Orient  X" range:[-rgoGrados,rgoGrados,oldX]
type:#integer
            spinner orY "Orient  Y" range:[-rgoGrados,rgoGrados,oldY]
type:#integer
            spinner orZ "Orient  Z" range:[-rgoGrados,rgoGrados,oldZ]
type:#integer
        )
        group "Planeación de Trayectorias"
        (
            button bVerPtos  "Obtener trayectoria" width:135 height:20
            spinner oSolTC "Solución " range:[1,8,1] type:#integer
            button bTC "Trayectoria Continua" width:135 height:20
        )

 ------------------------------------------------------------------------------
-- Rutine       : Ordenes para los diferentes elementos de las persianas
-- Function     : Elementos de la Persiana
-- Parameters   :
-- Return value : ninguno
------------------------------------------------------------------------------

        local vPtos=#(0), vOri=#(0),n=0,PtosExist=0, nVPtos=0,i;
        local ss
        --local vt1=#(0),vt2=#(0),vt3=#(0),vt41=#(0),vt51=#(0),vt61=#(0),
vt42=#(0),vt52=#(0),vt62=#(0)
        local eOriPto=#(0);
        local agrePtos=0
           local n, k, f, i, nSol
        on bAddPto pressed do
        (   n=n+1; i=n;
            if n<=15 then (
                  vPtos[i]=#(rX.value,  rY.value,  rZ.value);
                  vOri[i]=#(orX.value,  orY.value,  orZ.value);
            )
            print vPtos
            print vOri
            agrePtos=1;
        )
```

```
local rango=360, k=0,kk=0;
local vPtosTC=#(0),vOriTC=#(0),vP
on bVerPtos pressed do
(           k=0; kk=0;
            for i= 1 to rango do
            (     k=k+oInter.value;
                  -- Obtención de los puntos de la trayectoria
                  if k <= rango then
                  (     kk=kk+1;
                        -- ecuación parametrica
                        x=(oRad.value*cos(k)) + oRPX.value;

                        y=(oRad.value*sin(k)) + oRPY.value;
                        z=oRPZ.value;
                        vPtosTC[kk]=#(x,y,z);
                  vOriTC[kk]=#(orX.value, orY.value, orZ.value );
                  )
            )
            nVPtos =kk;
            if PtosExist== 1 then
            (     delOriPT eOriPto ss nVPtos
                  PtosExist=0;
            )
            if PtosExist ==0  then
            (
                  ss=splineshape()
                  addnewspline ss
                  for i=1 to kk do
                  (
                        eOriPto[i]= pyramid width:7 depth:7
height:30 pivot:[0,0,30]
                  eOriPto[i].rotation.z_rotation=180
                        eOriPto[i].pos=[vPtosTC[i][1],
vPtosTC[i][2], vPtosTC[i][3]];
                        in coordsys local rotate eOriPto[i]
(vOriTC[i][3]) z_axis
                        in coordsys local rotate eOriPto[i]
(vOriTC[i][2]) y_axis
                        in coordsys local rotate eOriPto[i]
(vOriTC[i][1]) x_axis
                        Pos= sphere radius:2 name:"oPos"
pos:[vPtosTC[i][1], vPtosTC[i][2], vPtosTC[i][3]];
                        addknot ss 1 #corner #curve [vPtosTC[i][1],
vPtosTC[i][2], vPtosTC[i][3]]
                  )
            PtosExist=1;
            updateshape ss
            agrePtos=0;
            --print nVPtos
            )
)
on bTC pressed do
(     kk=0; print nVPtos
      for kk=1 to nVPtos do
      (
            vP=#(vPtosTC[kk][1],vPtosTC[kk][2],vPtosTC[kk][3]-89.5);
            ang=InvKinet vP vOriTC[kk]
```

```
                        vt1[kk]=ang[1];                    vt2[kk]=ang[2];
            vt3[kk]=ang[3];
                        vt41[kk]=ang[4];                vt42[kk]=ang[5];
      vt51[kk]=ang[6];
                        vt52[kk]=ang[7];                vt61[kk]=ang[8];
      vt62[kk]=ang[9];
                        if robExist == 0 then   ( rob 1; robExist=1;)
                        if robRotar == 1 then if robExist == 1 then (delrob 1;
rob 1;)
                  )
                  Anima vt1 vt2 vt3 vt41 vt51 vt61 vt42 vt52 vt62 oSolTC.value
nVPtos
                  -- Asignar valores de los angulos a variables globales para si
se quiere generar archivo
                  nTraPtos=nVPtos
            )
            on brob pressed do
            (  if robExist==0 then rob 1
               robExist=1; robRotar=0;
            )
            on bdelrob pressed do
            (  if robExist==1 then  delrob 1
               if ptosExist==1 then  deloriPT  eOriPto ss nVPtos
               robExist=0; ptosExist=0; robRotar=0;
            )
            on breset pressed do
            (   if robExist==1 then (
                  delrob 1
                  rob 1
                  robRotar=0;)
            )

)--fin de Trayectoria Contínua
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Crea un Panel de Utilidades que despliega una persiana de iteracción
-- G E N E R A R    A R C H I V O
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Utility genArchivo "Generar Archivo"
(
 -----------------------------------------------------------------------------
-- Rutine       : "Archivo"
-- Function       : Elementos de la Persiana
-- Parameters   :
-- Return value : ninguno
-------------------------------------------------------------------------------
      group "Archivo"
      (
            edittext eNom "Nom. Archivo:" fieldWidth:70 text:"nombre.txt"
            spinner rNPrn "Trayectoria" range:[1,8,1] type:#integer
            button bsave "Generar" width:100 height:20
      )
 -----------------------------------------------------------------------------
-- Rutine       : Ordenes para los diferentes elementos de las persianas
-- Function       : Elementos de la Persiana
-- Parameters   :
```

```
-- Return value : ninguno
-------------------------------------------------------------------------------
      on bsave pressed do
      (
            --n=5;
            print "putnos"
            print nTraPtos
            nSol=rNPrn.value
            createfile eNom.text
            f=openfile eNom.text mode:"a"
            --print "hola" to:f
            if nSol>4 then k=nSol-4; else k=nSol;
            format "\nTheta 1\t Theta 2\t Theta 3\t Theta 4\t Theta 5\t Theta 6"
to:f
            for i=1 to nTraPtos do
            (
                  if i==1 then
                  (
                    if nSol>4 then
                        format "\n% \t% \t% \t% \t% \t%" vt1[i][k] vt2[i][k]
vt3[i][k] vt41[i][k] vt51[i][k] vt61[i][k] to:f
                    else
                        format "\n% \t% \t% \t% \t% \t%" vt1[i][k] vt2[i][k]
vt3[i][k] vt42[i][k] vt52[i][k] vt62[i][k] to:f
                  )else
                  (
                    if nSol>4 then
                        format "\n% \t% \t% \t% \t% \t%" (vt1[i][k]-vt1[i-1][k])
(vt2[i][k]-vt2[i-1][k]) (vt3[i][k]-vt3[i-1][k]) (vt41[i][k]-vt41[i-1][k])
(vt51[i][k]-vt51[i][k]) (vt61[i][k]-vt61[i-1][k]) to:f
                    else
                        format "\n% \t% \t% \t% \t% \t%" (vt1[i][k]-vt1[i-1][k])
(vt2[i][k]-vt2[i-1][k]) (vt3[i][k]-vt3[i-1][k]) (vt42[i][k]-vt42[i-1][k])
(vt52[i][k]-vt52[i][k]) (vt62[i][k]-vt62[i-1][k]) to:f
                  )
            )
            format "" to:f
            Print " Archivo generado "
            print f
            close f
      )
) --Fin de Generar Archivo
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Crea un Panel de Utilidades que despliega una persiana de iteracción
-- C R E A R    A R C H I V O
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

utility crearRob "Crear Robot"
(
      local vEslabones=#("Eslabón 1","Eslabón 2","Eslabón 3","Eslabón
4","Eslabón 5","Eslabón 6")
      local VerExist=0, ForEs=1
 -------------------------------------------------------------------------------
-- Rutine       : "1.Número de eslabón" "2. Forma de eslabón" "3.
-- Function      : Elementos de la Persiana
```

```
-- Parameters   :
-- Return value : ninguno
-------------------------------------------------------------------------------

      group "1. Número de eslabón"
      (
            dropdownlist rNumEs "Esabón" items:vEslabones
      )
      group "2. Forma del eslabón"
      (
            radiobuttons rForEs labels:#("Cilindro", "Cubo")
            button bVerCilCu "Crear"
      )
      group "3. Pivote orientación"
      (  label lRot1 "Primera Rotación "
            label lPivOr1 "Eje                              Grados"
            dropdownlist dpr1 "" items:#("X","Y","Z") width:50 across:2
            spinner spr1 "" fieldwidth:50  range:[-rgoGrados,rgoGrados,0]
type:#integer
         label lRot2 "Primera Rotación "
            label lPivOr2 "Eje                              Grados"
            dropdownlist dpr2 "" items:#("X","Y","Z") width:50 across:2
            spinner spr2 "" fieldwidth:50  range:[-rgoGrados,rgoGrados,0]
type:#integer
         label lRot3 "Primera Rotación "
            label lPivOr3 "Eje                              Grados"
            dropdownlist dpr3 "" items:#("X","Y","Z") width:50 across:2
            spinner spr3 "" fieldwidth:50  range:[-rgoGrados,rgoGrados,0]
type:#integer
            button bRotPiv "Rotar Pivote" width:130 height:20
      )
      group "4. Modificar forma de eslabón"
      (     label lmCil "Modificar Cilindro"
            spinner spRad "Radio" fieldwidth:50  range:[-rgoValor,rgoValor,20]
type:#integer align:#center
            spinner spAlt "Altura" fieldwidth:50  range:[-rgoValor,rgoValor,30]
type:#integer align:#center
            label lmCub "Modificar Cubo"
            spinner spLar "Largo  " fieldwidth:50  range:[-rgoValor,rgoValor,40]
type:#integer align:#center
            spinner spAnc "Ancho " fieldwidth:50  range:[-rgoValor,rgoValor,40]
type:#integer align:#center
            spinner spAlt1 "Altura  " fieldwidth:50  range:[-
rgoValor,rgoValor,40] type:#integer align:#center
            button bModEs "Modificar Eslabón" width:130 height:20
      )
      group "5. Posición del pivote"
      (     label lXYZ "X            Y            Z   "
            spinner sppX "" fieldwidth:37  range:[-rgoValor,rgoValor,0]
type:#integer across:3
            spinner sppY "" fieldwidth:37 range:[-rgoValor,rgoValor,0]
type:#integer
            spinner sppZ "" fieldwidth:37 range:[-rgoValor,rgoValor,0]
type:#integer
            button bMovPiv "Mover Pivote" width:130 height:20
      )
      group "7. Posición del eslabón"
```

```
      (     label lmesPE "( Depende de la posición" align:#left
            label lmesPE2 "del pivote )" align:#left
            label lXYZEs "X            Y            Z   "
            spinner speX "" fieldwidth:37  range:[-rgoValor,rgoValor,0]
type:#integer across:3
            spinner speY "" fieldwidth:37 range:[-rgoValor,rgoValor,0]
type:#integer
            spinner speZ "" fieldwidth:37 range:[-rgoValor,rgoValor,0]
type:#integer
            button bMovEs "Mover Eslabón" width:130 height:20
      )
      group "8. Guardar Eslabón"
      (     button bGuardarEslab "Guardar Eslabón" width:130 height:20
            button bNuevoEslab "Nuevo Eslabón" width:130 height:20
      )
      group "9. Crear Eslabón"
      (     button bCrearRobot "Crear robot" width:130 height:20
      )
      group "10. Offset entre eslabones"
      (     label lmoff "Ángulo entre los ejes X's"--formado entre los ejes "
align:#left
            label lmoff2 "(de los Parámetros D-H)"
            spinner spe1 "Elabón 1" range:[-rgoGrados,rgoGrados,0] type:#integer
            spinner spe2 "Elabón 2" range:[-rgoGrados,rgoGrados,0] type:#integer
            spinner spe3 "Elabón 3" range:[-rgoGrados,rgoGrados,0] type:#integer
            spinner spe4 "Elabón 4" range:[-rgoGrados,rgoGrados,0] type:#integer
            spinner spe5 "Elabón 5" range:[-rgoGrados,rgoGrados,0] type:#integer
            spinner spe6 "Elabón 6" range:[-rgoGrados,rgoGrados,0] type:#integer

            button bGuardarFF "Guardar" width:130 height:20
      )
      group "11. Rotación del EE a la base"
      (     spinner spXEE "Rotación en X"   range:[-rgoGrados,rgoGrados,0]
type:#integer align:#center
            spinner spYEE "Rotación en Y"   range:[-rgoGrados,rgoGrados,0]
type:#integer align:#center
            spinner spZEE "Rotación en Z"   range:[-rgoGrados,rgoGrados,0]
type:#integer align:#center
            button bGuardarEE "Guardar" width:130 height:20
      )

      local oldFor=0
      on rForEs changed  state do   -- Forma del eslabón
      (     format "Forma %\n" rForEs.state
      )

      on bVerCilCu pressed do             -- Dibujar Eslabón o círculo
      (
            if VerExist==1 then (delete eslab; VerExist=0)
            if VerExist== 0 then (
                  if rForEs.state == 1 then
                  (--if rNumEs== "Eslabón 1"
                        eslab=cylinder radius:20 height:30
                        oldFor=1
                  )
                  else
                  (     eslab =box length:40 width:40 height:40
```

```
                    oldFor=2
                )
                VerExist=1
                select eslab
            )
        )
        on bRotPiv pressed do                -- Rotar el punto pivote
        (
            if VerExist==0 then
                messagebox "No existe eslabón, seleccionar cilindro o cubo"
            else
            (
                eslab=RotaPivote eslab dpr1.selection spr1.value
                eslab=RotaPivote eslab dpr2.selection spr2.value
                eslab=RotaPivote eslab dpr3.selection spr3.value

            )
        )
        on bMovPiv pressed do                -- Mover el punto pivote
        (
            eslab.pivot=[sppX.value,sppY.value,sppZ.value];
        )
        on bModEs pressed do                 -- Modificar la forma del eslabón
        (   if verExist==1 then
            if rForEs.state==1 then (
                eslab.radius=spRad.value;
                eslab.height=spAlt.value;
            )else (
                eslab.length=spLar.value;
                eslab.width=spAnc.value;
                eslab.height=spAlt1.value;
            )
        )
        on bMovEs pressed do                 -- Mover el eslabón
        (
            eslab.pos=[speX.value,speY.value,speZ.value];
        )
        local si
        on bGuardarEslab pressed do          -- Guardar eslabón
        (   if vNE[rNumEs.selection]>0 then
                if queryBox "Esalbón ya existe. Desea sobreescribir?"  then
si=1; else si=0;
            if si==1 or vNE[rNumEs.selection]==0 then
            (
                --print vNE[rNumEs.selection]
                vForEs[rNumEs.selection]=rForEs.state;
                vRP1[rNumEs.selection]=dpr1.selection;
                vRP2[rNumEs.selection]=dpr2.selection;
                vRP3[rNumEs.selection]=dpr3.selection;
                vVRP1[rNumEs.selection]=spr1.value;
                vVRP2[rNumEs.selection]=spr2.value;
                vVRP3[rNumEs.selection]=spr3.value;
                vRadCil[rNumEs.selection]=spRad.value;
                vAltCil[rNumEs.selection]=spAlt.value;
                vlenCub[rNumEs.selection]=splar.value;
                vwidCub[rNumEs.selection]=spanc.value;
                vheiCub[rNumEs.selection]=spAlt1.value;
```

```
            vXpp[rNumEs.selection]=sppX.value;
            vYpp[rNumEs.selection]=sppY.value;
            vZpp[rNumEs.selection]=sppZ.value;
            vXpe[rNumEs.selection]=speX.value;
            vYpe[rNumEs.selection]=speY.value;
            vZpe[rNumEs.selection]=speZ.value;
            vNE[rNumEs.selection]=rNumEs.selection;
            format "Grabó configuración del eslabón : %" rNumEs.selection
            print ""
      )
)
on bNuevoEslab pressed do              -- Nuevo eslabón
(           --vForEs[rNumEs.selection]=rForEs.state;
            dpr1.selection=1;
            dpr2.selection=1;
            dpr3.selection=1;
            spr1.value=0;
            spr2.value=0;
            spr3.value=0;
            spRad.value=20;
            spAlt.value=30;
            splar.value=40;
            spanc.value=40;
            spAlt1.value=40;
            sppX.value=0;
            sppY.value=0;
            sppZ.value=0;
            speX.value=0;
            speY.value=0;
            speZ.value=0;
)
on bCrearRobot pressed do              -- Crear robot
(     local i=0, n=6, err=0;
      for i=1 to n do
            if vNE[i]==0 then
                  err=i
      print err
      print vNE
      if err>0 then (format "El eslabón % no se ha creado" err;
            print "";)
      else (
            print "Creando Robot Arbitrario"
            RobArbit 1;
      )
      print rForEs.state
)

on bGuardarFF presssed do
(     x1eslab=spe1.value;
      x2eslab=spe2.value;
      x3eslab=spe3.value;
      x4eslab=spe4.value;
      x5eslab=spe5.value;
      x6eslab=spe6.value;
)
on bGuardarEE presssed do
(     oriXEE= spYEE.value;
```

```
            oriYEE= spXEE.value;
            oriZEE= spZEE.value;
      )

)


utility robDefa "Crear robot Deafoult"
(
      button robDefa "Robot Defoult"

      on robDefa pressed do
      (
            robotDefault 1
            robArbit 1
      )
)
```

106

118

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Programa     : Calculo de la cinemática inversa de manipuladores 6R
-- Archivo      : kinet.ms
-- Revisión     : 1.0
-- Autor        : Erika Martínez
-- Compañía     :
-- Fecha        : 01-may-2001
-- Descripción : Cinemática Inversa de posición y
-- y cinemática inversa de orientación
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

function  kinet xc yc zc Q = (

--Solución de la cinemática inversa de posición para los ejemplos
--4.4.2 y 4.4.3 del libro de Angeles.
--aa=40;
--xc=0;
--yc=2*aa;
--zc=-aa;

--aa=40;
--xc=0;
--yc=aa;
--zc=0;
-------------------------------------------------------------------------------
--CINEMATICA INVERSA DE POSICION
--Parámetros Denavit-Hartenberg
parDH;

--Cinemática Inversa de Posicionamiento

-- Coeficientes de  Ac1 + Bs1 + Cc3 + Ds3 + E = 0
AA=2*a[1]*xc;
B=2*a[1]*yc;
C=(2*a(2)*a(3)) - (2*b(2)*b(4)*u[2]*u[3]);
D=(2*a(3)*b(2)*u[2]) + (2*a(2)*b(4)*u[3]);
E=(a(2))^2 + (a(3))^2 + (b(2))^2 + (b(3))^2 + (b(4))^2 - (a[1])^2 - (xc)^2 -
(yc)^2 - (zc-b(1))^2 + (2*b(2)*b(3)*l[2]) + (2*b(2)*b(4)*l[2]*l[3]) +
(2*b(3)*b(4)*l[3]);

-- Coeficientes Fc1 + Gs1 + Hc3 + Is3 + J = 0
F=yc*u[1];
G=-xc*u[1];
H=-b(4)*u[2]*u[3];
I=a(3)*u[2];
J=b(2) + (b(3)*l[2]) + (b(4)*l[2]*l[3]) - ((zc-b(1))*l[1]);

-- Cálculo de Delta 1
delta1=AA*G-F*B;
cpim=0;
-- Primer caso de Delta 1 <> 0
if delta1~=0 then (
   -- Coeficientes Rr3^4 + Sr3^3 + Tr3^2 + Ur3 + V = 0
   ro=xc^2+yc^2;
   R=4*a[1]^2*(J-H)^2+u[1]^2*(E-C)^2-4*ro*a[1]^2*u[1]^2;
```

```
    S=4*(4*a[1]^2*I*(J-H)+u[1]^2*D*(E-C));
    T=2*(4*a[1]^2*(J^2-H^2+2*I^2)+u[1]^2*(E^2-C^2+2*D^2)-4*ro*a[1]^2*u[1]^2);
    U=4*(4*a[1]^2*I*(H+J)+u[1]^2*D*(C+E));
    V=4*a[1]^2*(J+H)^2+u[1]^2*(E+C)^2-4*ro*a[1]^2*u[1]^2;

    -- Solución a una ecuación de cuarto orden
    -- Rr3^4 + Sr3^3 + Tr3^2 + Ur3 + V = 0
    pol4=[R S T U V];
    rtem=roots(pol4);

    cpo=0;
     nRaizPol4=size(rtem);
    nRPol4=nRaizPol4[1];

    for i=1:nRPol4 do (
          tau3(cpo+1)=rtem[i]; --Solo se almacenan raices reales
          cpo=cpo+1;
    )
    if nRPol4==0 then (
       print('\n ERROR 1. El sistema no es físicamente realizable\n');
       print('\n No hay valores reales para r3');
    )


    k=0;
    cont=nRPol4;
    for i=1:cont do (
       --                  Calculo de theta 3
       cos3[i]=(1-tau3[i]^2)/(1+tau3[i]^2);
       sen3[i]=(2*tau3[i])/(1+tau3[i]^2);
       th3[i]=atan2(sen3[i],cos3[i]);
       pec1=C*cos(th3[i])+D*sin(th3[i])+E;
       pec2=H*cos(th3[i])+I*sin(th3[i])+J;
       pec3=-G*pec1+B*pec2;
       pec4= F*pec1-AA*pec2;
       --                  Calculo de theta 1
       th1[i]=atan2(pec4/delta1,pec3/delta1);
    )
    contpo=cont;
)

if a[1]==0 & u[1]==0 then (
   print('\n ERROR 2. Robot incapacitados para trabajar en tres dimensiones\n');
   print('\n Seleccione otro manipulador\n');
)

-- delta 1 es igual a 0
if xc^2+yc^2==0 then (
   print('\n\n ERROR 3. Primera singularidad\n');
   cont=0;
)

if a[1]==0 | u[1]==0 then (
   --  zeros(size(tau3));
   sig=-1;
   cont=0;
   -- Primer Caso u1=0, ai<>0
```

```
 if a[1]~=0 & u[1]==0 then (
    --   Solucion de (J-H)r3^2 + 2Ir3 + (J-H) = 0
    --   Verifica si el radical es menor que cero
    radical = I^2-J^2+H^2;
    if (radical < 0) then (
       print('\n ERROR 4. Error para la solución de theta \n');
       cont=0; )
    else (
       for m=1:2 do (
          --  cambio de signo
          sig=-1*sig;
          --Obtención de theta3
          tau3(m)=(-I+sig*sqrt(radical))/(J-H);
          cos3(m)=(1-tau3(m)^2)/(1+tau3(m)^2);
          sen3(m)=(2*tau3(m))/(1+tau3(m)^2)   ;
          theta3(m)=atan2(sen3(m),cos3(m));

          --  (Ea-AA)r1^2 + 2Br1 + (Ea+AA) = 0
          --  donde Ea = Cc3 + Ds3 + E
          for n=1:2 do (
             Ea(n)=C*cos(theta3(m))+D*sin(theta3(m))+E;
             --  cambio de signo
             sig=-1*sig;
             radical2(n) = B^2-Ea(n)^2+AA^2
             --   Radical negativo
             if (radical2(n) > 0)  then (
                --Obtener theta 1
                tau1(n)=(-B+sig*sqrt(radical2(n)))/(Ea(n)-AA); --tau1
                cos1(n)=(1-tau1(n)^2)/(1+tau1(n)^2);
                sen1(n)=(2*tau1(n))/(1+tau1(n)^2);
                th1(n+(m-1)*m)=atan2(sen1(n),cos1(n));
                th3(n+(m-1)*m)=theta3(m);
                cont=cont+1;
             )
          )
       )
       if ((radical2(1) < 0) & (radical2(2) < 0)) then (
          print('\n ERROR 5. Error en la solución de theta 1');
          cont=0;
       )
    )
    contpo=cont;
)

--  Caso 2 u1<>0, ai=0
if a[1]==0 & u[1]~=0 then (
   --    (E-C)r3^2 + 2Dr3 + (E+C) = 0
   radical = D^2-E^2+C^2; )
   if (radical < 0) then (
      print('\n ERROR 6. Error en la solución de theta 3');
      cont=0; )
   else (
      for m=1:2 do (
         --  cambio de signo
         sig=-1*sig;
         --Obtención de theta3
         denEC=E-C;
```

```
            if (denEC==0) then
               if m==1 then
                  theta3(m)=0;
               else
                  theta3(m)=pi;

            else (
               tau3(m)=(-D+sig*sqrt(radical))/(E-C);
               cos3(m)=(1-tau3(m)^2)/(1+tau3(m)^2);
               sen3(m)=(2*tau3(m))/(1+tau3(m)^2);
               theta3(m)=atan2(sen3(m),cos3(m));
            )
            for n=1:2 do (
               Ja(n)=H*cos(theta3(m))+I*sin(theta3(m))+J;
               sig=-1*sig;
               radical2(n) = G^2-Ja(n)^2+F^2;
               if (radical2(n) > 0) then (
                  --Obtener theta 1
                  tau1(n)=(-G+sig*sqrt(radical2(n)))/(Ja(n)-F);
                  cos1(n)=(1-tau1(n)^2)/(1+tau1(n)^2);
                  sen1(n)=(2*tau1(n))/(1+tau1(n)^2);
                  th1(n+(m-1)*m)=atan2(sen1(n),cos1(n));
                  th3(n+(m-1)*m)=theta3(m);
                  cont=cont+1;
               )
            )
         )
         if ((radical2(1) < 0) & (radical2(2) < 0)) then (
            print('\nERROR 7. Error en solución de theta 1');
            cont=0;
         )
      )
      contpo=cont;
   )
)

-- Calculo de theta 2

cont=contpo;
for i=1:cont do (
   A11=a(2)+a(3)*cos(th3[i])+b(4)*u[3]*sin(th3[i]);
   A12=-
a(3)*l[2]*sin(th3[i])+b(3)*u[2]+b(4)*l[2]*u[3]*cos(th3[i])+b(4)*u[2]*l[3];
   delta2=A11^2+A12^2;

   -- Segunda singularidad
   if delta2==0 then
      print('\n\n  ERROR 8. Error de Segunda singularidad\n');
           out1=0;
   )

   --theta2
   pec1= xc*cos(th1[i])+yc*sin(th1[i])-a[1];
   pec2= -xc*l[1]*sin(th1[i])+yc*l[1]*cos(th1[i])+(zc-b(1))*u[1];
   pec3= A11*pec1 - A12*pec2;
   pec4= A12*pec1 + A11*pec2;
   th2[i]=atan2(pec4/delta2,pec3/delta2);  --theta 2
```

```
)     --Fin de ciclo for (cálculo de theta2)

out1=0;
if cpim==4 | contpo==0  then (
   print('\n ERROR 9. El sistema no es físicamente realizable\n');
   out1=0;
)
   t1=th1;
   t2=th2;
   t3=th3;


--  cálculo de los ángulos de orientación del robot
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------

col=size(t3,2);
for i=1:col do (
   -- Cálculo de Q1,Q2,Q3
   ct1=cos(t1[i]);
   if abs(ct1)<1e-15  then    ct1=0;

   st1=sin(t1[i]);
   if abs(st1)<1e-15  then    st1=0;

   ct2=cos(t2[i]);
   if abs(ct2)<1e-15  then    ct2=0;

   st2=sin(t2[i]);
   if abs(st2)<1e-15  then    st2=0;

   ct3=cos(t3[i]);
   if abs(ct3)<1e-15  then    ct3=0;

   st3=sin(t3[i]);
   if abs(st3)<1e-15  then    st3=0;

   Q1=[ct1,-l[1]*st1,u[1]*st1;st1,l[1]*ct1,-u[1]*ct1;0,u[1],l[1]];
   Q2=[ct2,-l[2]*st2,u[2]*st2;st2,l[2]*ct2,-u[2]*ct2;0,u[2],l[2]];
   Q3=[ct3,-l[3]*st3,u[3]*st3;st3,l[3]*ct3,-u[3]*ct3;0,u[3],l[3]];

   -- Cálculo de R=Q3'*Q2'*Q1'*Q
   R=(Q3')*(Q2')*(Q1')*Q;

   -- Definición de valores xi, eta y zeta de la ecuación
   -- [(l[5]-eta*u[4]-zeta*l[4]]*tau[4]^2 - [2*xi*u[4]]*tau[4] +[(l[5]-eta*u[4]-
zeta*l[4]] = 0
   xi = R[1][3];          eta = R[2][3];              zeta = R[3][3];

   --verificacion del denominador
   denom = l[5]-zeta*l[4]-eta*u[4];
   if (denom == 0)
      -- Caso particular del Robot del laboratorio
      -- Matriz de R (0 - 3) = Rot z,theta1 * Rot y,theta2 * Rot -y,theta3
      R1 = [cos(t1[i])*cos(t2[i]-t3[i]), -sin(t1[i]), -cos(t1[i])*sin(t2[i]-
t3[i])];
```

```
      R2 = [sin(t1[i])*cos(t2[i]-t3[i]),  cos(t1[i]), -sin(t1[i])*sin(t2[i]-
t3[i])];
      R3 = [sin(t2[i]-t3[i]),                0,         cos(t2[i]-t3[i])];
      R03 = [R1;R2;R3];

      -- Matriz de R (3 - 6) = Rot z,theta4 * Rot y,theta5 * Rot z,theta6
      -- R36 = R03'*Q
      R36 = R03'*Q;


      -- cálculo de theta 5
      theta51[i] = atan2 ( sqrt(1-R36(3,3)^2), -R36(3,3));       --theta51
      theta52[i] = atan2 (-sqrt(1-R36(3,3)^2), -R36(3,3));       --theta52
      if (sin(theta51[i]) > 0) then (
         -- cálculo de theta 4
         theta41[i] = atan2(R36(2,3)*sin(theta51[i]),R36(1,3)*sin(theta51[i]));
      --theta41
         -- cálculo de theta 6
         theta61[i] = atan2(-R36(3,2)*sin(theta51[i]),R36(3,1)*sin(theta51[i]));
--theta61
      else
         -- cálculo de theta 4
         theta41[i] = atan2(-R36(2,3)*sin(theta51[i]),-
R36(1,3))*sin(theta51[i]); --theta42
         -- cálculo de theta 6
         theta61[i] = atan2(R36(3,2)*sin(theta51[i]),-R36(3,1)*sin(theta51[i]));
--theta62
      )
      if (sin(theta52[i]) > 0)  then   (
         -- cálculo de theta 4
         theta42[i] = atan2(R36(2,3)*sin(theta52[i]),R36(1,3)*sin(theta52[i]));
      --theta41
         -- cálculo de theta 6
         theta62[i] = atan2(-R36(3,2)*sin(theta52[i]),R36(3,1)*sin(theta52[i]));
--theta61
    )
      else
    (
         -- cálculo de theta 4
         theta42[i] = atan2 (-R36(2,3)*sin(theta52[i]) (-
R36(1,3))*sin(theta52[i]);  --theta42
         -- cálculo de theta 6
         theta62[i] = atan2(R36(3,2)*sin(theta52[i]),-R36(3,1)*sin(theta52[i]));
--theta62
      )
   else (
      --verificacion de condiciones de xi, eta zeta
      as1=round(xi^2+eta^2+zeta^2);
      as2=(xi^2+eta^2)*u[4]^2-(l[5]-zeta*l[4])^2;
      if (round(xi^2+eta^2+zeta^2) ==1) & ((xi^2+eta^2)*u[4]^2-(l[5]-
zeta*l[4])^2>= 0) then (
         -- Cálculo de
         -- (xi^2 + eta^2)*u[4]^2 - [ l[5] - zeta*l[4]]^2
         raiz = ( (xi^2+eta^2)*(u[4]^2) - ( l[5]-zeta*l[4] )^2 );
         if (raiz < 0 )
            print('\nERROR 10. Error posición no alcanzable\n');
            break;
```

```
        else
           raiz=sqrt(raiz);
        )

        --              Theta 4
        t41[i]=(xi*u[4]+raiz)/denom;
        t42[i]=(xi*u[4]-raiz)/denom;
        cos41[i]=(1-t41[i]^2)/(1+t41[i]^2);
        sen41[i]=(2*t41[i])/(1+t41[i]^2);
        theta41[i]=atan2(sen41[i],cos41[i]);
        cos42[i]=(1-t42[i]^2)/(1+t42[i]^2);
        sen42[i]=(2*t42[i])/(1+t42[i]^2);
        theta42[i]=atan2(sen42[i],cos42[i]);

        --              Theta 5
        --              Q5=Q4'*R*Q6'
        tem1 = R[1][2]*u[6] + R[1][3]*l[6];
        tem2 = R[2][2]*u[6] + R[2][3]*l[6];
        tem3 = R[3][2]*u[6] + R[3][3]*l[6];

        --                    Theta51
        tem511 = [cos(theta41[i])*tem1 + sin(theta41[i])*tem2]/u[5];
        tem512 = [-l[4]*sin(theta41[i])*tem1 + l[4]*cos(theta41[i])*tem2 +
u[4]*tem3 ] / (-u[5]);
        theta51[i]=atan2 tem511 tem512 ;

        -- Cálculo de theta52
        tem521 = [cos(theta42[i])*tem1 + sin(theta42[i])*tem2]/u[5];
        tem522 = [-l[4]*sin(theta42[i])*tem1 + l[4]*cos(theta42[i])*tem2 +
u[4]*tem3 ] / (-u[5]);
        theta52[i]=atan2 tem521 tem522 ;

        --              Theta 6
        --             Q6=Q5'Q4'R

        r1=[R(1,1); R(2,1); R(3,1)];
        -- Cálculo de theta61
        -- Cálculo de Q4 con theta41
        Q4=[cos(theta41[i]), -l[4]*sin(theta41[i]), u[4]*sin(theta41[i]);
sin(theta41[i]), l[4]*cos(theta41[i]), -u[4]*cos(theta41[i]); 0,u[4],l[4]];

        -- Cálculo de w=Q4'*r1
        w = Q4'*r1;
        --              Theta 61
        tem611=w(1,1)*cos(theta51[i])+w(2,1)*sin(theta51[i]);
        tem612=-
w(1,1)*l[5]*sin(theta51[i])+w(2,1)*l[5]*cos(theta51[i])+w(3,1)*u[5];
        theta61[i] = atan2(tem612,tem611);

        --                    Theta62
        Q4=[cos(theta42[i]), -l[4]*sin(theta42[i]), u[4]*sin(theta42[i]);
sin(theta42[i]), l[4]*cos(theta42[i]), -u[4]*cos(theta42[i]); 0,u[4],l[4]];

        -- Cálculo de w=Q4'*r1
        w = Q4'*r1;

        tem621=w(1,1)*cos(theta52[i])+w(2,1)*sin(theta52[i]);
```

```
         tem622=-
w(1,1)*l[5]*sin(theta52[i])+w(2,1)*l[5]*cos(theta52[i])+w(3,1)*u[5];
         theta62[i] = atan2(tem622,tem621);
      else (
         --print('\nERROR 11. Configuración no realizable \n');
         --break;
               -- Robot Cloos Interpretación geométrica (Spong)
      -- Matriz de R (0 - 3) = Rot z,theta1 * Rot y,theta2 * Rot -y,theta3
      R1 = [cos(t1[i])*cos(t2[i]-t3[i]), -sin(t1[i]), -cos(t1[i])*sin(t2[i]-
t3[i])];
      R2 = [sin(t1[i])*cos(t2[i]-t3[i]),  cos(t1[i]), -sin(t1[i])*sin(t2[i]-
t3[i])];
      R3 = [sin(t2[i]-t3[i]),                0,         cos(t2[i]-t3[i])];
      R03 = [R1;R2;R3];

      -- Matriz de R (3 - 6) = Rot z,theta4 * Rot y,theta5 * Rot z,theta6
      -- R36 = R03'*Q
      R36 = R03'*Q;


      -- cálculo de theta 5
      theta51[i] = atan2 ( sqrt(1-R36(3,3)^2), -R36(3,3));       --theta51
      theta52[i] = atan2 (-sqrt(1-R36(3,3)^2), -R36(3,3));       --theta52
      if (sin(theta51[i]) > 0) then (
                  theta41[i] =
atan2(R36(2,3)*sin(theta51[i]),R36(1,3)*sin(theta51[i]));  --theta41
                  theta61[i] = atan2(-
R36(3,2)*sin(theta51[i]),R36(3,1)*sin(theta51[i]));  --theta61
)
      else
 (                 theta41[i] = atan2(-R36(2,3)*sin(theta51[i]),-
R36(1,3))*sin(theta51[i]); --theta42
                  theta61[i] = atan2(R36(3,2)*sin(theta51[i]),-
R36(3,1)*sin(theta51[i]));  --theta62
      )
      if (sin(theta52[i]) > 0)   then (
                  theta42[i] =
atan2(R36(2,3)*sin(theta52[i]),R36(1,3)*sin(theta52[i]));
                  theta62[i] = atan2(-
R36(3,2)*sin(theta52[i]),R36(3,1)*sin(theta52[i]));   )
      else (
         --              theta 4
         theta42[i] = atan2(-R36(2,3)*sin(theta52[i]),-
R36(1,3))*sin(theta52[i]);
         --             theta 6
         theta62[i] = atan2(R36(3,2)*sin(theta52[i]),-R36(3,1)*sin(theta52[i]));
      )


      )
   )
)     --fin de ciclo 2 (for -> fin)
if col==0
   print('\nERROR 12. El sistema no es físicamente realizable\n');
   break;
)
t1; t2; t3;
t41=theta41;
```

```
t42=theta42;
t51=theta51;
t52=theta52;
t61=theta61;
t62=theta62;
```