23
2ej

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

PLANEACION DE TRAYECTORIAS PARA MANIPULADORES Y ROBOTS MOVILES

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N :

JUAN ALFONSO MARTINEZ PADILLA
LUIS ADRIAN LETEPICHIA FLORES
JORGE ALFONSO HERNANDEZ SANTIS

Director: Dr. Francisco Cervantes Pérez

MEXICO, D. F. 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO.

I INTRODUCCION

1.1 MOTIVACION

En la actualidad la robótica es un campo de investigación que está muy avanzado en paises industrializados como Estados Unidos, Alemania y Japón, al grado de que algunas fábricas como las de Japón estan totalmente automatizadas, operando casi exclusivamente con robots y con apenas unas cuantas personas que supervisen el sistema. El avance de la investigación en robótica y su aplicación, hace que se aumente la productividad de la industria, elevando la eficiencia de sus procesos de produccion , puesto que los robots debido a sus caracteristicas pueden realizar grandes series de trabajos siempre con la misma precision y con mayor rapidez que un operador humano, se pueden adaptar para diversos tipos de trabajo, pueden trabajar las 24 horas del dia, soportar ambientes de trabajo hostiles y necesitan poco mantenimiento. En adición a todas estas ventajas de tipo operativo, los robots liberan al hombre de realizar tareas tediosas y peligrosas, permitèndole dedicarse a labores más creativas.

En nuestro pais, el estado de la robótica es incipiente, a diferencia de los paises desarrollados , debido principalmente a que, si bien, las investigaciones en torno al tema datan ya de algún tiempo y hay gente reconocida que actualmente se dedica a la robótica, no ha sido suficiente como para tener un impulso significativo que permita tener una tecnologia robótica nacional.

Actualmente los robots que se encuentran operando en las fábricas del pais, son importados y por lo tánto de un costo muy elevado, requiriendo tener personal de mantenimiento entrenado en el extranjero. Por lo anterior, una de las principales motivaciones para la realización de este trabajo es resaltar la importancia de la investigación en robótica, despertando este interés en las personas relacionadas con la aplicación de esta tecnologia , para que se propongán, aprovenen y realizen proyectos multidisciplinarios en robótica perceptual. Por otra parte también se intenta que este trabajo pueda servir de base para otros proyectos, o pueda ser aplicado en educación, como auxiliar en un curso de robótica a nivel universitario, o su aplicación directa a la industria, como en la planeacion de trayectorias para definir nuevas tareas en una planta de ensamblaje.

Una de las caracteristicas más importantes del campo de la robótica es su naturaleza multidisciplinaria, en donde se conjuntan varias áreas como son Electrónica, Mecánica, Control, Planeación, Economia, Sociologia y en la actualidad la Inteligencia Artificial. Es importante señalar que el avance en la investigación de alguna de estas areas beneficia a las

demas, a veces de manera significativa . El tema central de esta tesis cae dentro de la Planeación y más especificamente, dentro de la Planeación de Trayectorias mediante algoritmos implementados en la computadora.

1.2 OBJETIVOS

El objetivo fundamental de esta tesis es desarrollar, y probar a través de simulaciones en computadora, algoritmos de planeación de trayectorias para robots con capacidades sensoriales (principalmente visión). Este objetivo fundamental esta constituido por cuatro objetivos especificos, los cuales se discuten enseguida.

a) En primer lugar se busca desarrollar un sistema de planeación y ejecución de trayectorias para un robot manipulador fijo. Entendemos como robot a un manipulador multifuncional y reprogramable, diseñado para mover materiales, piezas, herramientas o dispositivos especiales, mediante movimientos programables y variables que permitan llevar a cabo tareas diversas (Ref. 2 pag. 14). El ambiente del robot esta configurado a base de objetos tridimensionales de forma y tamaño predefinidos. Dependiendo de la tarea que se le asigne al sistema, por ejemplo poner un objeto sobre otro, este debe ser capaz de planear una trayectoria óptima de movimiento de la pinza del robot para que ésta tome los objetos y los traslade a las posiciones especificadas. Se supone que para propositos de planeación el robot toma datos del medio ambiente a traves de sensores (camara de T. V.) y este sensor solo se simula dentro del sistema.

b) Se pretende diseñar un sistema gráfico tridimensional que permita visualizar en la pantalla de la computadora el robot y los movimientos definidos por el sistema de planeación para manipular los objetos. El sistema gráfico debe permitir el trazado de la trayectoria elaborada por el planificador y que sigue la pinza del robot sin dibujar a este.

c) El tercer objetivo especifico, se relaciona con un robot móvil, y entendiendose por Robot Móvil un aparato electromecánico que puede moverse por medio de ruedas, u otro sistema de locomoción, en un ambiente controlado y que además puede o no tener brazos manipuladores. Partiendo de esto, lo que se busca es desarrollar un sistema que permita planear y ejecutar trayectorias de movimiento para el robot (no para sus brazos) en un ambiente estructurado en forma de habitaciones dentro de las cuales se encuentran colocados una serie de objetos con diversas configuraciones y localizaciones. Se supone que para propositos de planeación el robot accesa datos del

medio ambiente a traves de sensores (cámara de T.V.), y al igual que en el caso del robot fijo, los datos de este sensor, solo son simulados dentro del sistema.

d) También se pretende diseñar una interface gráfica que nos permita visualizar en la pantalla de la computadora la trayectoria planeada para el recorrido del robot móvil, la cual estara integrada en un solo programa junto con el del sistema de planeación.

1.3 ORGANIZACION DEL TRABAJO ESCRITO

Esta tèsis esta organizada en cinco capítulos y dos apèndices. El capítulo dos, "Revisión Bibliográfica", es un compendio de los principales aspectos de la robótica: Diseño de Manipuladores, Robots basados en Sensores, Lenguajes de Programación y Aplicaciones. En la parte de diseño de manipuladores se habla de la Cinemática, Dinámica y Control de Robots, exponiendo la teoria matemática para describir las posiciones, velocidades y aceleraciones, asi como también el control de las partes de un manipulador; en la parte de Robots basados en Sensores se hase enfasis en los diferentes tipos de sensores para robots que existen en la actualidad; mientras que en la parte de Lenguajes de Programación se explican los principales tipos de lenguajes de Programación para robos que existen; y por último, en la parte de Aplicaciones se presentan algunas de las principales aplicaciones de los robots en la industria.

En el capítulo tres "Planeación de Trayectorias para un Manipulador", primero se hace una descripción general de distintos sistemas de representación y planeación de trayectorias para un robot manipulador que se han desarrollado hasta la fecha, exponiendo sus ventajas y desventajas. Posteriormente, se explica la representación utilizada y el algoritmo diseñado en esta tesis para la planeación de trayectoria de un robot manipulador, describiendose el funcionamiento del sistema. Como parte de este capítulo también se describe la tècnica de graficación tridimensional diseñada con el fin de visualizar en la pantalla de la computadora tanto al robot y sus movimientos (planeados por el sistema de planeación) como el ambiente en que trabaja.

El capítulo cuatro "Planeación de Trayectorias para un Robot Móvil", describe las características del dominio de robot móvil y explica brevemente los tipos de control que existen para este tipo de robots. Después se hace un estudio de diferentes tipos de representaciones usadas en la planeación de trayectorias. En la última parte de este capítulo se describe el sistema de planeación de trayectorias que se diseño en esta tèsis, explicando el algoritmo de planeación, la representación usada, y los módulos que constituyen el sistema.

En ambos capítulos, tres y cuatro, se incluye una sección de "Simulaciones y resultados" que contienen varios ejemplos gráficos de funcionamiento de los sistemas, tanto del robot manipulador como del robot móvil.

El capítulo cinco, "Discusión y conclusiones", se analizan los aspectos importantes del trabajo presentado en esta tèsis: una evaluación de los algoritmos de Planeación de Trayectorias implementados; los objetivos logrados; las aplicaciones que pudieran tener en las areas de educación e industria; y la definición de posibles etapas futuras, tanto para mejorar el sistema como para ampliar la gama de aplicaciones.

Los apèndices contienen los listados del código fuente de los programas para cada uno de los sistemas (apèndice A), y un manual de usuario, el que contiene los requerimientos y el proceso a seguir para correr cada uno de los sistemas desarrollados (apèndice B).

2. REVISION BIBLIOGRAFICA

En este capitulo se revisan varios aspectos de la robótica de carácter general. Esta revisión no pretende ser exhaustiva, únicamente tratamos de presentar el material suficiente para un entendimiento de lo que es la Robótica actualmente y hacia donde se dirige la investigación en este campo. Por lo tanto, la sección 2.1 trata aspectos del diseño y control de manipuladores, como son la Cinemática, Dinámica y Control. La sección 2.2 incluye los diferentes tipos de sensores que existen para robots, mientras que en la 2.3 se describen los lenguajes de programación para manipuladores. Finalmente, en la sección 2.4, se dan las aplicaciones principales de la Robótica.

2.1 DISEÑO Y CONTROL DE MANIPULADORES.

Se puede considerar que un robot esta configurado básicamente por tres partes fundamentales (ver fig. 2.1):

a) El manipulador,
b) El controlador, y
c) Mecanismos sensoriales.

El manipulador es la parte mecánica del robot y se compone de la siguiente manera :

a1) Varios eslabones (rígidos en primera aproximación) relacionados entre si mediante junturas (o uniones) que permiten un movimiento relativo; ya sea una rotación alrededor de un eje, para junturas rotacionales R, o una traslación para las junturas prismáticas P .

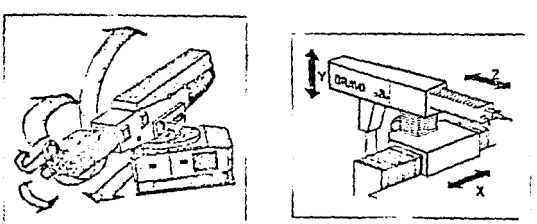


fig. 2.1. Configuración de un robot.

a2) Una "mano" (efector final) cuya función es sujetar, orientar y operar sobre las piezas, objetos o herramientas a manipular.

a3) Mecanismos de movimiento, tales como dispositivos neumáticos e hidráulicos, motores de paso, motores eléctricos de corriente continua, etc. Estos proporcionan energía mecánica que se trasmite directamente a través de elementos auxiliares como engranajes, correas dentadas, etc.

Por su parte, el sistema de control normalmente es llevado a cabo por un computador, que debe definir y manejar el movimiento del mecanismo a través de la coordinación en espacio y tiempo de los actuadores que controlan los diferentes grados de libertad (generalmente seis) del manipulador.

En lo que se refiere a los mecanismos sensoriales, son dispositivos que permiten la interacción del robot con su entorno. Esta parte solo existe en los robots inteligentes, donde el sistema de control tiene capacidad de toma de decisiones; esto es, son capaces de llevar a cabo procesos de Coordinación Sensora-motora.

El diseño de manipuladores no sólo involocra la definición de la forma y tamaño del manipulador, sino que también se refiere al desarrollo teórico de los criterios necesarios para el control del mismo en condiciones estáticas (Cinemática) y de movimiento (Dinámica).

2.1.1 CONTROL

Los robots industriales son manipuladores mecánicos controlados por computadora y generalmente son utilizados en aplicaciones industriales. El número de articulaciones de robots comercialmente disponibles varía de tres a siete. Tipicamente tienen seis articulaciones, dando 6 grados de libertad, con un "gripper", el cuál es referenciado a una mano o efector final. Cada articulación del robot requiere de un control posicional con un sistema de control, el cuál puede ser de varios tipos.

En esta sección se presenta una descripción general de los sistemas de control utilizados en los robots, desde los más sencillos, en los que un solo microprocesador controla al robot, hasta los más avanzados, que emplean varios microprocesadores que conforman un sistema de proceso distribuido.

Con relación al control de un movimiento global del robot, existe una clasificación con respecto a la forma de definir la tarea y, por ende, los movimientos parciales que el sistema debe verificar que se realicen en forma controlada. Esta clasificación es:

1. Robots con control punto a punto sin realimentación
2. Robots con control punto a punto con realimentación
3. Robots con control continuo
4. Robots con capacidad de generación de trayectorias
5. Sistemas de Control Distribuido
6. Sistemas de Control para Robots Inteligentes

1.- ROBOTS CON CONTROL PUNTO A PUNTO SIN REALIMENTACION

La caracteristica de este tipo de robots es que deben realizar un "aprendizaje" de la trayectoria que deben seguir antes de ejecutarla. Para ello, el operador, utilizando ya sea un teclado, un "joystick", un brazo maestro o trasladando directamente el efector final, "enseña" al manipulador la trayectoria que debe seguir, mientras que el manipulador va almacenando en su memoria todos los puntos de dicha trayectoria. Después en la fase de ejecución el robot solo tiene que seguir la trayectoria que ya tiene almacenada en memoria en forma de coordenadas de puntos.

El control es de malla abierta (ver fig. 2.2), esto es, no hay información sobre la situación actual de cada elemento del brazo manipulador durante la ejecución del movimiento. Esto se logra utilizando topes mecánicos o finales de carrera, o utilizando motores de paso cuya posición final se conoce de antemano de acuerdo con el número de impulsos aplicados.

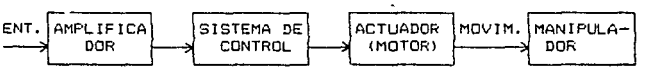


Figura 2.2. La entrada procesada por el amplificador y sistema de control se aplica al actuador, produciendo un movimiento conocido.

Cuando se usan motores de paso, hay una cierta limitación, y esta es que no puede haber posicionamiento del eje a cualquier ángulo y es debido a que la rotación de un paso del motor equivale a un giro de un ángulo fijo del eje.

Dos de los robots que usan este tipo de control son el TEACHMOVER y el ARMDROID. Para una explicación de las caracteristicas de estos robots ver la referencia 2.

2.- ROBOTS CON CONTROL PUNTO A PUNTO CON REALIMENTACION

El sistema de control para este tipo de robots funciona en malla cerrada, lo que quiere decir que siempre se tiene una información precisa de los parámetros del actuador como es la posición, velocidad, aceleración, etc. Esta información

que constituye la señal de salida, es comparada con la señal de entrada (posición, velocidad o aceleración deseada), obteniendose una señal de error, la cual es utilizada por el controlador para seleccionar la señal que debe enviarse al actuador, tratando de que la salida llegue al valor deseado, y de esta forma controlar el elemento motriz del robot manipulador (ver fig. 2.3).

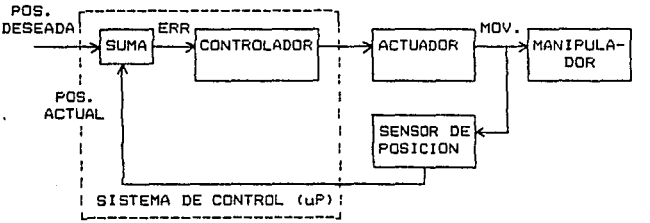


Figura 2.3. La posición actual es comparada constantemente con la posición deseada. El controlador, en base al error calculado, define la acción del actuador, corrigiendose la posición del manipulador hacia la deseada.

Los robots GIZAMAT I y GIZAMAT II, diseñados por IKERLAN, Centro de Investigación de Euzkadi, son dos ejemplos del tipo de robots punto a punto con realimentación (ver la ref. 2).

3.- ROBOTS CON CONTROL CONTINUO

Para algunas aplicaciones industriales, como pintura y soldadura continua, hay que reproducir una trayectoria completa, por lo que no es suficiente tener los puntos inicial, intermedios y final, como pasaba con los robots controlados punto a punto.

La programación de este tipo de robots, se realiza casi manualmente; primero, se debe colocar la pistola de pintura o la pinza de soldadura en el extremo del manipulador; después, el operario lleva la herramienta por el camino adecuado y con la velocidad conveniente. Durante este proceso, automáticamente el sistema de control va registrando los datos emitidos por los sensores de posición de todos los ejes y los almacena en memoria, a un ritmo que puede llegar, para cada eje, a las 80 tomas de datos por segundo.

Una vez programado, el robot dispone de la información en coordenadas generalizadas sobre los movimientos efectuados por cada eje, siendo capaz de reproducir fielmente la trayectoria descrita.

En la referencia 2 se explica la aplicación del sistema de control continuo de trayectoria para el robot GIZAMAT I.

4.- ROBOTS CON CAPACIDAD DE GENERACION DE TRAYECTORIAS

La diferencia fundamental entre este tipo de robots y los anteriores reside en el software, la potencia de procesamiento y el cálculo de la unidad de control.

Esta nueva generación de robots combina las ventajas propias de la programación punto a punto, y la posibilidad de definir las posiciones de forma precisa, con una mínima ocupación de memoria, así como las ventajas que se derivan del control continuo de trayectoria.

Estos robots pueden gobernarse directamente mediante coordenadas cartesianas $X = (x,y,z,\alpha,\beta,\tau)$, así cuando se programan, se puede hacer que la pinza suba o baje perpendicularmente, o se desplace hacia adelante o hacia atras en linea recta. La unidad de cálculo se encarga de efectuar de manera automática, la transformación $X \dashrightarrow Q$, para obtener los valores de las entradas de los servomecanismos que consiguen el movimiento solicitado. Además, basta con preparar unos cuantos puntos importantes de la trayectoria deseada, puesto que en la fase de repetición el sistema de control calculará los puntos restantes de la trayectoria para que enlacen con los programados, ya sea mediante una curva continua o lineas rectas.

Las ventajas que presenta este tipo de robots, son: a) En la fase de "aprendizaje", el sistema realiza la coordinación de los ejes para posicionar y orientar el efector final en el punto deseado, idependientemente de la estructura del robot. b) El operador no actua sobre las articulaciones del brazo manipulador, sino sobre los selectores referidos a las coordenadas cartesianas, cilindricas, etc., del sistema ligado a la pinza. c) El operador no programa la totalidad de la trayectoria deseada, sino sus puntos más importantes. d) En la fase de repetición, el sistema de control genera las trayectorias de los puntos programados, pudiendo recorrerlas con la velocidad y aceleración deseadas.

Estas dos caracteristicas plantean dos problemas importantes. Por un lado, la definición y generación de las trayectorias, referidas a un sistema coordenado fijo definido en la base del robot. Por otra parte, es necesario realizar la transformación de coordenadas en linea.

En la referencia 2 se describe la aplicación del sistema de control con capacidad de generación de trayectorias al robot GIZAMAT I.

5.- SISTEMAS DE CONTROL DISTRIBUIDO

Los complejos cálculos matemáticos que requieren los robots industriales para funcionar en tiempo real, ademas de la gran velocidad de trabajo, han hecho que los sistemas de control sólo se pudieran implementar con minicomputadoras. Sin embargo, la aparición de los microprocesadores de 16 y 32 bits, con sus familias de elementos (i.e. coprocesadores matemáticos), permiten actualmente soportar los rápidos y potentes cálculos. Esto se logra a base de varios microprocesadores, lo que representa una reducción de costos, asi como una modularidad muy eficaz.

A cada sistema microprocesador se le asigna un conjunto de tareas concretas y unos recursos especificos de software. Existen configuraciones en las que todos los microprocesadores comparten un unico sistema de "buses" y una memoria común; sin embargo, la arquitectura basada en niveles jerárquicos es la más empleada en la práctica, pudiéndose distinguir hasta tres niveles en estos sistemas (ver fig. 2.4):

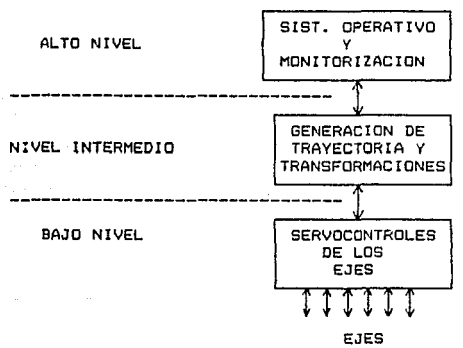


Figura 2.4. Niveles jerárquicos. Cada uno puede configurarse con uno o varios microprocesadores.

1. Alto nivel. Conformado por microprocesadores que se encargan del control operativo de todo el sistema, así como del monitoreo.

2. Nivel intermedio. Se compone de uno o varios microprocesadores, encargados de la generación de las trayectorias y de las transformaciones de coordenadas.

3. Bajo nivel. Está compuesto también por uno o varios microprocesadores, que se encargan de los servocontroles de los ejes del manipulador.

6.- SISTEMAS DE CONTROL PARA ROBOTS INTELIGENTES

En la actualidad, muchos centros de investigación de carácter industrial y universitario realizan diseños de nuevos modelos de robots, cuyas características son que se pueden programar con lenguajes naturales, adaptarse al entorno usando sensores avanzados y tomar decisiones, de forma automática, mediante el uso de la Inteligencia Artificial.

Si bien, este nuevo tipo de robots está en fase de investigación y su aplicación práctica no es realista, sus requerimientos de hardware y software puede originar que su construcción espere algunos años, hasta que nuevos modelos de microprocesadores puedan soportar las exigencias precisas de potencia y velocidad de cálculo.

Un sistema desarrollado por UNIMATION INC. (ref. 2), que incorpora visión y adaptación al entorno, es el SRI, que precisa de dos minicomputadoras para el proceso de la información y cuyo esquema general se muestra en la fig. 2.5.

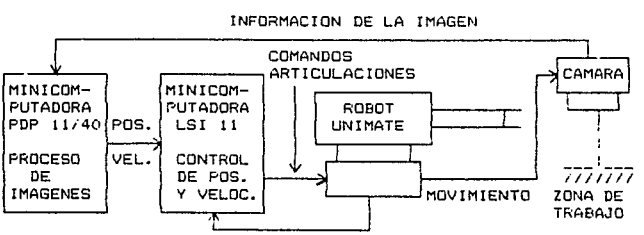


Figura 2.5. El sistema SRI, de control visual del robot, precisa de dos minicomputadoras.

La nueva generación de robots estará dotada de una coordinación entre Percepción y Acción, por medio de un sistema de entendimiento de información sensorial (visión, tacto, etc.).

La inteligencia artificial, con la que irán provistos los nuevos robots, estará formada por un sistema de software capaz de reconocer formas, comprender lenguajes naturales, tomar decisiones y tener la posibilidad de elaboración de planes de actuación para adaptarse a las variaciones del entorno.

Los grandes recursos que, para el procesamiento de información necesitan estos nuevos robots, hacen esperar que su implantación ocurra dentro de varios años.

2.1.2 CINEMATICA

La cinemàtica, en relaciòn a un robot manipulador, se refiere al estudio de su configuraciòn espacial en funciòn del tiempo, sin prestar atenciòn a las fuerzas y momentos que causan su movimiento. En particular, la atenciòn se centra en determinar la relaciòn entre junturas y variables espaciales, asi como en la definiciòn de la posiciòn y orientaciòn del efector final del manipulador.

Los problemas cinemàticos son dos :

1.- El problema cinemàtico directo. Encontrar la posiciòn y orientaciòn del efector final del manipulador con respecto a un sistema coordenado de referencia, dado un vector de estado q=(q1.q2....,qn), formado por las variables de las junturas, y los paràmetros geomètricos de los eslabones. El subindice n representa los grados de libertad del manipulador.

2.- El problema cinemàtico inverso. Calcular el valor adecuado del vector de estado q para posicionar el efector final del manipulador en la posiciòn y con la orientaciòn deseadas, dadas la posiciòn y orientaciòn actuales del manipulador con respecto al sistema coordenado de referencia y los diferentes paràmetros geomètricos de los eslabones.

2.1.2.1 TRANSFORMACIONES HOMOGENEAS.

Las transformaciones homogèneas son la herramienta teòrica que nos permite expresar formalmente, la posiciòn y orientaciòn de la mano (herramienta o efecto final) del manipulador y de los objetos en su entorno, con respecto a un sistema de ejes coordenados especifico.

Estos problemas se ven desde un punto de vista matricial, lo cuàl es ventajoso ya que el planteamiento se hace usando el producto de una matriz de transformaciòn por un vector que representa un punto de algùn objeto.

Un robot se diseña para manipular piezas y realizar diversas operaciones, en una determinada àrea de trabajo (o regiòn accesible). Estas manipulaciones se reducen a giros y traslaciones de sus eslabones y junturas en el espacio tridimensional; ademàs, si el robot està dotado de mecanismos sensoriales (por ejemplo visiòn), debe ser capàz de reconocer una misma pieza a diferentes distancias y en distintas posiciones.

TIPOS DE TRANSFORMACION.

Dado que el movimiento de un manipulador se realiza con rotaciones y traslaciones y, que para resolver las ecuaciones cinemàticas se se harà uso de transformaciones inversas, consideraremos las siguientes transformaciones homogèneas :

- Traslaciòn.
- Rotaciòn.
- Inversa.

TRANSFORMACION DE TRASLACION.

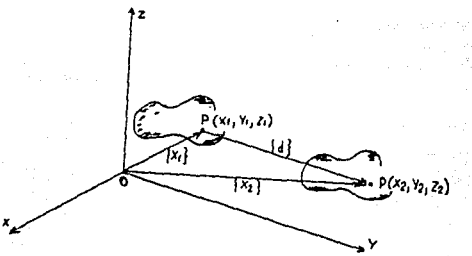


fig. 2.6 Traslaciòn de un cuerpo rigido.

Sea un cuerpo rigido en el espacio tridimensional (ver fig. 2.6); un punto del mismo se define en coordenadas homogèneas, a través de un vector V1 :

$$V1 = \begin{bmatrix} x1 \\ y1 \\ z1 \end{bmatrix}$$

Una traslaciòn aplicada al cuerpo rigido la definiremos por medio de una matriz de traslacion, de tal manera, que el punto de posiciòn V1 del cuerpo, pasarà a ocupar la posiciòn V2 definida por :

$$V2 = \begin{bmatrix} x2 \\ y2 \\ z2 \\ 1 \end{bmatrix} = tras(dx,dy,dz)\ V1 = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ z1 \\ 1 \end{bmatrix}$$

TRANSFORMACION DE ROTACION.

Para el caso de tres dimensiones tenemos una transformaciòn de rotaciòn con respecto a cada uno de los ejes :

Rotación de un ángulo θ alrededor del eje x.

$$rot(\theta,x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -sen(\theta) & 0 \\ 0 & sen(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación de un ángulo Ø alrededor del eje y.

$$rot(\emptyset\ ,y) = \begin{bmatrix} \cos(\emptyset) & 0 & sen(\emptyset) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(\emptyset) & 0 & \cos(\emptyset) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación de un ángulo Γ alrededor de z.

$$rot(\Gamma,z) = \begin{bmatrix} \cos(\Gamma) & -sen(\Gamma) & 0 & 0 \\ sen(\Gamma) & \cos(\Gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

TRANSFORMACION INVERSA.

En general, para una matriz de transformaciòn del tipo

$$[T] = \begin{bmatrix} nx & ox & ax & px \\ ny & oy & ay & py \\ nz & oz & as & pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

su matriz inversa està dada por

$$T^{-1} = \begin{bmatrix} nx & ox & ax & -p\cdot n \\ ny & oy & ay & -p\cdot o \\ nz & oz & az & -p\cdot a \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

donde

$$p\cdot n = px\ nx + py\ ny + pz\ nz$$
$$p\cdot o = px\ ox + py\ oy + pz\ oz$$
$$p\cdot a = px\ ax + py\ ay + pz\ az$$

2.1.2.2 APLICACION DE LAS TRANFORMACIONES HOMOGENEAS.

Las transformaciones homogèneas se utilizan para describir la posiciòn y orientaciòn entre eslabones del manipulador.

La notaciòn de Denavit y Hartenberg(1955), refiere a la matriz de transformaciòn homogenea como la matriz [A], la cual describe la relaciòn entre el sistema coordenado de referencia asignado a un eslabòn y el del adyacente. Esta matriz se obtiene a partir de definir los movimientos de traslaciòn y orientaciòn relativas que ocurren entre ambos sistemas.

Para un manipulador con seis eslabones unidos por junturas se tiene que la matriz de transformaciòn T est' dada por :

$$T6 = [A1]\ [A2].....[A6]$$

donde:

[A1] representa la posiciòn y orientaciòn del primer eslabòn con respecto a la base(referencia global), y

[Ai] con i=2,...,6 la del eslabòn (i) con respecto al eslabòn (i-1).

Esto es, la posiciòn y orientacion del segundo eslabòn con respecto al sistema de referencia global se define por :

$$[T2] = [A1]\ [A2]$$

En este momento denominaremos [Ti] a los productos de matrices [Ai], la cùal nos representa la posiciòn y orientaciòn del ùltimo eslabòn(mano de sujeciòn) con respecto a la referencia global.

Así, [T6] contiene toda la información sobre la posición y orientación de la mano del manipulador. La mano de sujeción de un brazo manipulador (con seis grados de libertad) puede ser ubicada y orientada en cualquier posición dentro de su zona de trabajo con el uso de la matriz [T6].

El contenido de la matriz [T6] es el siguiente :

$$[T6] = \begin{bmatrix} nx & sx & ax & px \\ ny & sy & ay & py \\ nz & sz & az & pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

En la fig. 2.7 se ve el significado físico de los elementos de la matriz [T6].

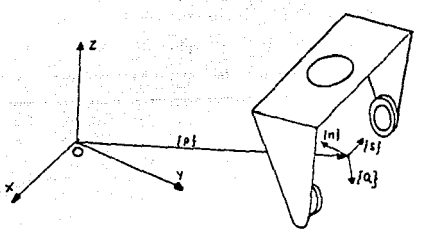


fig. 2.7. Significado de la matriz [T6].

Las tres primeras columnas son vectores unitarios que nos indican la orientación y la cuarta columna es un vector que define la posición.

El origen del sistema de referencia unido a la mano se define por medio del vector {p}.

En el caso de la orientación, el vector correspondiente al eje z se llamará vector de aproximación {a}; el que corresponde al eje y se conoce con el nombre de vector orientación {s}, su dirección es la que une los "dedos" de la mano; mientras que al eje x le corresponde un vector llamado normal {n}, de tal forma que se verifica que sean ortogonales. Esto es : {n} = {s} x {a}

2.1.2.3 ESPECIFICACION DE [T6] EN TERMINOS DE MATRICES [Ai]

Ya hemos comentado, que la matriz homogenea [Ti] que define la posición y orientación del eslabón (i) con respecto al sistema global se puede obtener por el producto de las sucesivas matrices de transformación. En lo que se refiere a la posición de la mano con respecto al sistema de referencia del eslabón (i), está definida por medio de una $^{(i-1)}$[T6], siendo :

$$^{(i-1)}[T6] = [Ai]\,[A(i+1)]\,.....[A6]$$

Cuando todas la junturas son de revolución (ver fig.2.8), las matrices [Ai] que se mencionan en esta ecuación, tienen la siguiente forma :

$$[Ai] = \begin{bmatrix} \cos\theta i & -\text{sen}\theta i\,\cos\alpha i & \text{sen}\theta i\,\text{sen}\alpha i & ai\,\cos\theta i \\ \text{sen}\theta i & \cos\theta i\,\cos\alpha i & -\cos\theta i\,\text{sen}\alpha i & ai\,\text{sen}\theta i \\ 0 & \text{sen}\alpha i & \cos\alpha i & di \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

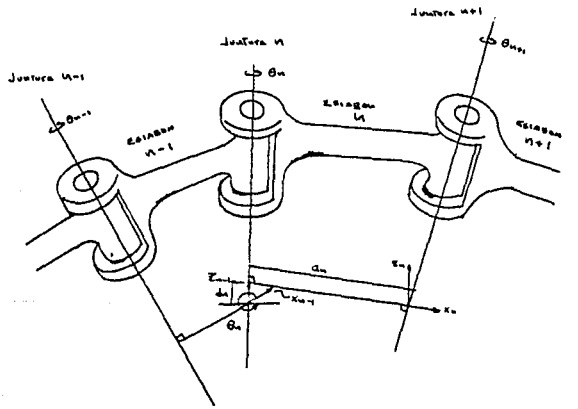


fig. 2.8. Eslabones y junturas de revolución con sus paràmetros asociados.

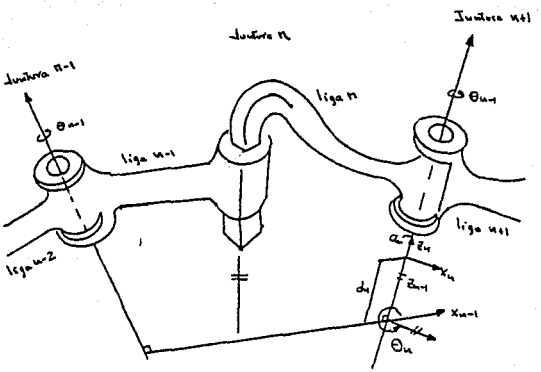


fig 2.9 Eslabones y junturas (con una prismàtica) con sus parà-metros asociados.

En caso de que exista una juntura prismàtica (ver fig. 2.9) la variable es la distancia di, siendo su direcciòn la del movimiento relativo permitido. Para este caso, la longitud ai no tiene sentido, por lo cùal se hace igual a cero. Entonces, la matriz [Ai] està dada por :

$$[Ai] = \begin{bmatrix} \cos\theta i & -\text{sen}\theta i\ \cos\alpha i & \text{sen}\theta i\ \text{sen}\alpha i & 0 \\ \text{sen}\theta i & \cos\theta i\ \cos\alpha i & -\cos\theta i\ \text{sen}\alpha i & 0 \\ 0 & \text{sen}\alpha i & \cos\alpha i & di \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Observaciòn : en la matriz [Ai] para las junturas rotacionales la variable es el valor de θi, mientras que, en las junturas prismàticas es di.

La obtenciòn de estas matrices depende del sistema de referencia que se asigne a los eslabones del manipulador.Para asignar sistemas de referencia a los eslabones del manipulador se siguen cuatro reglas fundamentales:

a) El eje z(i-1)se sitúa en la dirección del movimiento relativo permitido en la juntura (i).

b) El eje xi es perpendicular al zi y al z(i-1).

c) El eje yi se define por la siguiente expresión :

$$\{yi\} = \{zi\} \times \{xi\}$$

siendo {zi} y {xi} vectores unitarios en la dirección zi y xi respectivamente.

d) El origen del sistema de referencia del eslabón (i) se localiza en la intersección de la normal común entre los ejes de las junturas (i) e (i+1) y el eje de la juntura (i+1). Si tenemos el caso de que los ejes se corten, el origen se coloca en el punto de intersección.

Si seguimos estas reglas, somos libres de elegir la posición del sistema de referencia global en cualquier punto de la juntura 1, teniendo en cuenta que el eje z0 debe coincidir con la dirección de la primera juntura.

En resumen, la representación, por medio del criterio de Denavit y Hartenberg, de un eslabón del manipulador depende de cuatro valores que describen completamente las junturas :

1) θi es el ángulo de la juntura medido entre los ejes x(i-1) y xi alrededor del eje z(i-1).

2) di es la distancia entre el origen del sistema de referencia del eslabón (i-1) y el punto de intersección del eje z(i-1) con el zi, medida en la dirección del eje z(i-1).

3) ai es la longitud del eslabón (i), medida desde la intersección del eje z(i-1) con el eje xi hasta el origen del sistema de referencia del eslabón (i). Se puede definir, también, como la distancia mínima entre los ejes z(i-1) y zi.

4) αi es el ángulo de torsión del eslabón, medido entre los ejes z(i-1) y zi.

Recordar que, si tenemos una juntura R, el parámetro correspondiente al grado de libertad es θi, mientras que, en una juntura P, es di.

Debemos señalar, además, que el origen del sistema de referencia del eslabòn 6 se elige coincidente con el del 5. En caso de que se haya colocado una herramienta o mano de sujeción cuyo origen y ejes no coincidan con el sistema coordenado del eslabòn 6, esta mano o herramienta se puede tener en cuenta mediante una transformación homogenea constante respecto del eslabòn 6 (matriz [G] anteriormente). Si el manipulador se encuentra situado, con respecto a un sistema de referencia (x,y,z), mediante una matriz de transformación [Z], la posición y orientación del extremo de la mano o herramienta, con respecto a (x,y,z) es (ver fig. 2.10) :

$$[M] = [Z]\,[T6]\,[E]$$

o lo que es lo mismo :

$$[T6] = Z^{-1}\,[M]\,E^{-1}$$

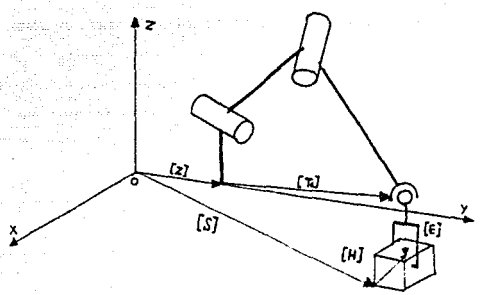


fig. 2.10. Manipulador con respecto a un sistema de referencia.

2.1.2.4 ECUACIONES CINEMATICAS PARA EL MANIPULADOR STANFORD.

La fig. 2.11 muestra al manipulador Stanford con sistemas coordenados asociados a sus eslabones, de acuerdo al procedimiento mencionado en la parte anterior.

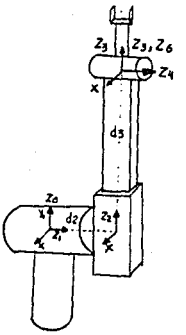


fig. 2.11. Asignaciòn de sistemas coordenados a los eslabones del manipulador Stanford.

Usaremos las siguientes abreviaturas para el seno y el coseno del ángulo θ :

$$\text{sen}(\theta i) = Si \quad ; \quad \cos(\theta i) = Ci$$
$$\text{sen}(\theta i+\theta j) = Sij \quad ; \quad \cos(\theta i+\theta j) = Cij$$

Para determinar las matrices [Ai], utilizaremos los datos de la tabla I, la cual contiene los paràmetros y variables relacionados con los eslabones del manipulador, de acuerdo con los sistemas coordenados asignados.

Eslabón	Variable	α	a	d	cosα	senα
1	θ1	-90°	0	0	0	-1
2	θ2	90°	0	d2	0	1
3	d3	0°	0	d3	1	0
4	θ4	-90°	0	0	0	-1
5	θ5	90°	0	0	0	1
6	θ6	0°	0	0	1	0

tabla 1

Sustituyendo los datos de la tabla 1 en las ecuaciones para calcular [Ai] para junturas R y P, obtenemos las [Ai] para el manipulador Stanford :

$$A1 = \begin{bmatrix} C1 & 0 & -S1 & 0 \\ S1 & 0 & C1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A2 = \begin{bmatrix} C2 & 0 & S2 & 0 \\ S2 & 0 & -C2 & 0 \\ 0 & 1 & 0 & d2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A4 = \begin{bmatrix} C4 & 0 & -S4 & 0 \\ S4 & 0 & C4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A5 = \begin{bmatrix} C5 & 0 & S5 & 0 \\ S5 & 0 & -C5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A6 = \begin{bmatrix} C6 & -S6 & 0 & 0 \\ S6 & C6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Los productos de matrices [Ai] para este manipulador, iniciando en el eslabòn 6, son :

$$^{5}[T6] = \begin{bmatrix} C6 & -S6 & 0 & 0 \\ S6 & C6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{4}[T6] = \begin{bmatrix} C5\ S6 & -C5\ S6 & S5 & 0 \\ S5\ C6 & -S5\ S6 & -C5 & 0 \\ S6 & C6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{3}[T6] = \begin{bmatrix} C4\ C5\ C6-S4\ S6 & -C4\ C5\ C6-S4\ C6 & C4\ S5 & 0 \\ S4\ C5\ C6+C4\ S6 & -S4\ C5\ S6+C4\ C6 & S4\ S5 & 0 \\ -S5\ C6 & S5\ S6 & C5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{2}[T6] = \begin{bmatrix} C4\ C5\ C6-S4\ S6 & -C4\ C5\ C6-S4\ C6 & C4\ S5 & 0 \\ S4\ C5\ C6+C4\ S6 & -S4\ C5\ S6+C4\ C6 & S4\ S5 & 0 \\ -S5\ C6 & S5\ S6 & C5 & d3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{1}[T6] = \begin{bmatrix} C2(C4C5C6-S4S6)-S2S5C6 & -C2(C4C5S6+S4C6)+S2S5S6 & C2C4S5+S2C5 & S2d3 \\ S2(C4C5C6-S4S6)+C2S5C6 & -S2(C4C5S6+S4C6)-C2S5S6 & S2C4S5-C2C5 & -C2d3 \\ S4C5C6+C4S6 & -S4C5S6+C4C6 & S4S5 & d2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Usando la siguiente matriz :

$$[T6] = \begin{bmatrix} nx & sx & ax & px \\ ny & sy & ay & py \\ nz & sz & az & pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

e igualando tèrmino a tèrmino, obtenemos las ecuaciones cinemàticas para el manipulador de Stanford :

```
nx = C1[C2(C4C5C6-S4S6)-S2S5C6]-S1(S4C5C6+C4S6)
ny = S1[C2(C4C5C6-S4S6)-S2S5C6]+C1(S4C5C6+C4S6)
nz = -S2(C4C5C6-S4S6)-C2S5C6
sx = C1[-C2(C4C5S6+S4C6)+S2S5S6]-S1(-S4C5S6+C4C6)
sy = S1[-C2(C4C5S6+S4C6)+S2S5S6]+C1(-S4C5S6+C4C6)
sz = S2(C4C5S6+S4C6)+C2S5S6
ax = C1(C2C4S5+S2C5)-C1S4S5
ay = S1(C2C4C5+S2C5)+C1S4S5
az = -S2C4S5+C2C5
px = C1S2d3-S1d2
py = S1S2d3+C1d2
pz = C2d3
```

2.1.2.5 SOLUCION DE LAS ECUACIONES CINEMATICAS.

La solución de las ecuaciones cinemàticas consiste en encontrar las posiciones de cada eslabòn del manipulador cuando se conoce la matriz [T6]. Esto es, se tienen que encontrar los valores de las variables de movimiento; que para el caso del manipulador Stanford son θ1,θ2,d3,θ4,θ5,θ6.

Para el caso de robots con menos o igual a seis grados de libertad se puede encontrar la solución sin que sea necesario recurrir a procedimientos iterativos de resolución de sistemas de ecuaciones no lineales. Por el contrario, para los robots con mas de seis grados de libertad, en los que hay que recurrir a criterios de optimización para obtener la posición de los eslabones, resulta imprescindible usar procedimientos iterativos.

Para el manipulador Stanford, la solución se obtiene a continuación (ver Paul, 1982).

En los manipuladores, [T6] es conocida y es igual al producto de matrices [Ai]

$$[T6] = [A1]\,[A2]\,[A3]\,[A4]\,[A5]\,[A6]$$

Si premultiplicamos en forma sucesiva por matrices inversas [Ai] se obtiene

$$ {}^{-1}A1\ [T6] = {}^{1}[T6] $$

$$ {}^{-1}A2\ {}^{-1}A1\ [T6] = {}^{2}[T6] $$

$$ {}^{-1}A3\ {}^{-1}A2\ {}^{-1}A1\ [T6] = {}^{3}[T6] $$

$$ {}^{-1}A4\ {}^{-1}A3\ {}^{-1}A2\ {}^{-1}A1\ [T6] = {}^{4}[T6] $$

$$ {}^{-1}A5\ {}^{-1}A4\ {}^{-1}A3\ {}^{-1}A2\ {}^{-1}A1\ [T6] = {}^{5}[T6] $$

Si premultiplicamos la ec. de $[T6]$ por $A1^{-1}$

$$ {}^{-1}A1\ [T6] = [A2]\ [A3]\ [A4]\ [A5]\ [A6] $$

desarrollando obtenemos

$$ {}^{-1}A1\ [T6] = \begin{bmatrix} C1 & S1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -S1 & C1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} nx & sx & ax & px \\ ny & sy & ay & py \\ nz & sz & az & pz \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{-1}A1\ [T6] = \begin{bmatrix} f11(n) & f11(s) & f11(a) & f11(p) \\ f12(n) & f12(s) & f12(a) & f12(p) \\ f13(n) & f13(s) & f13(a) & f13(p) \\ 0 & 0 & 0 & 1 \end{bmatrix} . $$

donde

$$ f11 = C1\ x + C1\ y $$
$$ f12 = -z $$
$$ f13 = -S1\ x + C1\ y $$

Si usamos la ec. de ${}^{1}[T6]$ obtenida anteriormente e igualando

$$ {}^{-1}A1\ [T6] = {}^{1}[T6] $$

se obtiene

$$ f13(p) = d2 $$

o lo que es lo mismo

$$-S1\ px + C1\ py = d2$$

Para resolver ecuaciones de esta forma, creamos sustituciones trigonomètricas de la siguiente forma :

$$px = r\ cos\phi$$
$$py = r\ sen\phi$$

donde

$$r = +\sqrt{px^2 + py^2}$$
$$\phi = angtan(py/px)$$

Sustituyendo px y py en la ec. obtenida y haciendo operaciones obtenemos :

$$sen\phi\ cos\theta1 - cos\phi\ sen\theta1 = d2/r$$

con

$$0 < d2/r <= 1$$

La ecuaciòn anterior se reduce a

$$sen(\phi - \theta1) = d2/r$$

con

$$0 < \phi - \theta1 < \pi$$

ademàs, tenemos que

$$cos(\phi - \theta1) = \pm\sqrt{1 - (d2/r)^2}$$

Donde el signo - corresponde a un manipulador de brazo izquierdo y el signo + a uno de brazo derecho.

Finalmente

$$\theta1 = angtan(py/px) - angtan(d2/+\sqrt{r^2 - d2^2})$$

Si igualamos el elemento 1,4 y el 2,4 de la ec. de [Tó] con f11(p) y f12(p) se tiene

$$S2\ d3 = C1\ px + S1\ py$$

$$-C2\ d3 = -pz$$

dividiendo estas ecuaciones se obtiene

$$\theta 2 = \text{angtan}((C1\ px + S1\ py)/pz)$$

Evaluando los elementos de la ecuación

$$A2^{-1}\ A1^{-1}\ [T6] = {}^{2}[T6]$$

se obtiene

$$A2^{-1}\ A1^{-1}\ [T6] = \begin{bmatrix} f21(n) & f21(s) & f21(a) & 0 \\ f22(n) & f22(s) & f22(a) & 0 \\ f23(n) & f23(s) & f23(a) & f23(p) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T6] = \begin{bmatrix} C4\ C5\ C6-S4\ S6 & -C4\ C5\ C6-S4\ C6 & C4\ S5 & 0 \\ S4\ C5\ C6+C4\ S6 & -S4\ C5\ S6+C4\ C6 & S4\ S5 & 0 \\ -S5\ C6 & S5\ S6 & C5 & d3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

donde

$$f21 = C2(C1\ x + S1\ y) - S2\ z$$
$$f22 = -S1\ x + C1\ y$$
$$f23 = S2(C1\ x + S1\ y) + C2\ z$$

igualando el elemento 3,4 de ${}^{2}[T6]$ con f23(p)

$$d3 = S2(C1\ px + S1\ py) + C2\ pz$$

Evaluando los elementos de la ecuación

$$A4^{-1}\ A3^{-1}\ A2^{-1}\ A1^{-1}\ [T6] = {}^{4}[T6]$$

se obtiene

$$A4^{-1}\ A3^{-1}\ A2^{-1}\ A1^{-1}\ [T6] = \begin{bmatrix} f41(n) & f41(s) & f41(a) & 0 \\ f42(n) & f42(s) & f42(a) & 0 \\ f43(n) & f43(s) & f43(a) & f43(p) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$
{}^{4}[T6] = \begin{bmatrix} C5\,S6 & -C5\,S6 & S5 & 0 \\ S5\,C6 & -S5\,S6 & -C5 & 0 \\ S6 & C6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

donde

```
f41 = C4[C2(C1 x +S1 y) - S2 z] + S4[-S1 x + C1 y]
f42 = -S2(C1 x + S1 y) - C2 z
f43 = -S4[C2(C1 x + S1 y) - S2 z] + C4[-S1 x + C1 y]
```

El elemento f43(a) nos da

```
-S4[C2(C1 ax + S1 ay) - S2 az] + C4[-S1 ax + C1 ay] = 0
```

resolviendo obtenemos

```
θ4 = angtan((-S1 ax + C1 ay)/(C2(C1 ax + S1 ay) - S2 az))
```

Para los elementos 1,3 y 2,3 se tiene

```
 S5 = C4[C2(C1 x +S1 y) - S2 z] + S4[-S1 x + C1 y]
-C5 = -S2(C1 x + S1 y) - C2 z
```

resolviendo se obtiene

```
θ5 = angtan((C4[C2(C1 ax + S1 ay) - S2 az]
     + S4[-S1 ax + C1 ay])
       /(S2(C1 ax + S1 ay) + C2 az))
```

Para calcular el grado de libertad θ6 se evaluan los elementos de la siguiente ecuación :

$$
{}^{-1}_{A5}{}^{-1}_{A4}{}^{-1}_{A3}{}^{-1}_{A2}{}^{-1}_{A1}[T6] = {}^{5}[T6]
$$

$$
\begin{bmatrix} f51(n) & f51(s) & 0 & 0 \\ f52(n) & f52(s) & 0 & 0 \\ f53(n) & f53(s) & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C6 & -S6 & 0 & 1 \\ S6 & C6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

donde

$$f51 = C5(C4[C2(C1\ x + S1\ y) - S2\ z] + S4[-S1\ x + C1\ y])$$
$$+ S5(-S2(C1\ x + S1\ y) - C2\ z)$$
$$f52 = -S4[C2(C1\ x + S1\ y) - S2\ z] + C4[-S1\ x + C1\ y]$$
$$f53 = S5(C4[C1\ x + S1\ y) - S2\ z] + S4[-S1\ x + C1\ y])$$
$$+ C5(S2(C1\ x + S1\ y) + C2\ z)$$

igualando los elementos 1,2 y 2,2 obtenemos :

$$S6 = -C5(C4[C2(C1\ ox + S1\ oy) - S2\ oz] + S4[-S1\ ox + C1\ oy])$$
$$+ S5(-S2(C1\ ox + S1\ oy) - C2\ oz)$$

$$C6 = -S4[C2(C1\ ox + S1\ oy) - S2\ oz] + C4[-S1\ ox + C1\ oy]$$

Con lo cúal se obtiene :

$$\theta 6 = \text{angtan}(S6/C6)$$

2.1.3 DINAMICA

La finalidad del anàlisis dinàmico, es llegar a conocer la naturaleza y magnitud de las fuerzas y movimientos en los elementos del sistema mecànico (incluido el elemento fijo o soporte); los modelos dinàmicos, se expresan por un conjunto de ecuaciones diferenciales lineales o no que relacionan fuerzas y momentos con posiciones, velocidades y aceleraciones.

En los estudios de la dinàmica de los robots existen dos problemas fundamentales, el directo y el inverso.

El problema dinàmico directo se utiliza en aplicacones del diseño del manipulador y consiste en encontrar las velocidades y aceleraciones teniendo como entrada los valores de las fuerzas y momentos de cada una de las junturas del robot.

El problema dinàmico inverso se usa en aplicaciones de control para manufactura en tiempo real y consiste en determinar las fuerzas y momentos a aplicar a los motores del manipulador del manipulador, teniendo como entrada los valores de las aceleraciones y velocidades para cada segmento de trayectoria a moverse.

Para la soluciòn de ambos problemas, se consideran los siguientes pasos:

1- Elecciòn del modelo matemàtico mas adecuado al sistema real. La elecciòn deberà considerar condiciones tanto de precisiòn como econòmicas.

2- Podemos modelar a los elementos mecánicos del manipulador como deformables o indeformables haciendo uso de la dinámica del sóiido deformable para el primer caso o la del sólido rígido para el segundo. Este paso, consiste en realizar una simplificaciòn del modelo matemàtico tratando que los mètodos de anàlisis usados para su soluciòn se puedan resolver en tiempo real, el cuàl es un factor importante para el control del robot.

3- Planteamiento de las ecuaciones Dinàmicas utilizando uno de los siguientes mètodos:

- Newton
- D'Alambert
- Lagrange
- Trabajos virtuales
- Hamilton

4- Resoluciòn de las ecuaciones que seràn en general ecuaciones diferenciales fuertemente no lineales, por lo que se hace impresindible el uso de computadoras.

El método que ha sido utilizado para el planteamiento de las ecuaciones din'amicas de un robot es el de Lagrange.

Para mostrar el uso de esta metodologia utilizaremos el ejemplo dado en la fig. 2.12.

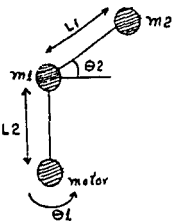


fig. 2.12. Manipulador con un grado de libertad.

El único grado de libertad es θ1 por lo que el vector de estado q(t) esta dado por:

$$q(t) = \begin{pmatrix} \theta_1(t) \end{pmatrix}$$

Los parámetros geométricos del manipulador son:

L1, L2 - longitudes de los eslabones

θ2 - ángulo del eslabón L2 con respecto a la horizontal.

m1, m2 - masa de los elementos de las junturas.

$K=(1/2)*I*w^2$ donde $I=[L2*\cos(\theta 2)]^2*m2$

$w^2 = \dot{\theta}1^2$

de donde

$$k(\theta 1, \dot{\theta}1) = (1/2)*m2*[L2*\cos\theta 2]^2*\dot{\theta}1^2$$

La energía potencial $V(q) = V(\theta 1)$ esta dada por:

$$V(\theta 1) = m1*g*L1 + m2*g*[L1+L2*\mathrm{sen}(\theta 2)]$$

Por lo tanto, el lagrangiano es $L = K-V$

donde

$$L = (1/2)*m2*[L2*\cos(\theta 2)]^2*\dot{\theta}1^2 - m1*g*L1 - m2*g*[L1+L2\,\mathrm{sen}(\theta 2)]$$

en seguida se calcula la

$$\frac{\partial L}{\partial qi} = \frac{\partial L}{\partial \theta 1}$$

y esto es igual a:

$$\frac{\partial L}{\partial \theta 1} = 2*(1/2)*m2*[L2*\cos(\theta 2)]^2*\dot{\theta}1 =$$

$$\frac{\partial L}{\partial \theta 1} = m2*[L2*\cos(\theta 2)]^2*\dot{\theta}1$$

se requiere

$$\frac{\partial L}{\partial qi} = \frac{\partial L}{\partial \theta 1}$$ La cuál esta dada por :

$$\frac{\partial L}{\partial \theta 1} = 0$$

Se necesita finalmente:

$$\frac{d}{dt}\left[\frac{\partial L}{\partial \dot{\theta}1}\right]$$

la cual es:

$$\frac{d}{dt}\left[\frac{\partial L}{\partial \dot{\theta}1}\right] = m2*[L2*\cos(\theta 2)]^2*\ddot{\theta}1$$

aplicando esta en la ecuación de lagrange la cual es:

$$\frac{d}{dt}\left[\frac{\partial L}{\partial \dot{\theta}1}\right] - \frac{\partial L}{\partial \theta 1} = \tau$$

se obtiene finàlmente el modelo dinàmico del brazo:

$$m2*[L2*\cos(\theta 2)]^2*\ddot{\theta}1 = \tau$$

2.2 ROBOTS CON CAPACIDADES SENSORIALES.

Un sistema de robot industrial consta, además del manipulador y del controlador, de uno o varios elementos que le adaptan al trabajo a realizar y que le relacionan con el ambiente que los rodea. Estos dispositivos son los elementos terminales acoplables al efector final del manipulador, y los sensores que informan sobre las circunstancias más interesantes que envuelven el ambiente de trabajo.

Para lograr un correcto funcionamiento del robot es necesario primero conocer la posición que guardan cada una de sus articulaciones (o junturas cinemáticas) y, en ciertas aplicaciones de mayor sofisticación, detectar el entorno y y sus variaciones. Este es el papel de los sensores: aquellos que informan el estado de las articulaciones son llamados propioceptivos e incluyen potenciómetros, sincros(resolvers), regletas y círculos codificados, tacómetros, y otros.

Por otro lado, tenemos los sensores exteroceptivos que informan sobre el entorno y sus variaciones. Básicamente estos son ya sea de tacto (fuerza o par), usando galgas extensométricas; o ya sea de visión, usando televisión, laser o infrarrojos. Sólo algunos robots comercializados poseen percepción visual o de tacto (ver Angulo Usategui, 1984).

La actuación de los sensores permite al sistema de robot trabajar en lazo cerrado, estando informado el controlador de la situación real del entorno, por lo que puede definir los planes de acción pertinentes.

2.2.1 TIPOS DE SENSORES Y CARACTERISTICAS.

En general, los sensores pueden ser usados para las siguientes funciones :

+ Obtener información acerca del espacio de trabajo y de los objetos que se van a manipular.

+ Corregir errores de posición y orientación del brazo y mano del robot respectivamente.

+ Detectar obstáculos y minimizar sus efectos.

El mundo de los sensores, normalmente se refiere a sensores externos, los cuáles pueden ser de dos clases :

- Sensores de contacto, que permiten determinar fuerza/momento y toque/presión cuando está en contacto físico con el objeto.

- Sensores de no contacto, que sirven para sensar imágenes, rango, y presencia de objetos sin tener contacto físico.

Los sensores de contacto sensan(miden) fuerza/momento y toque/presión cuando están en contacto físico con el objeto.

Los sensores de no contacto sensan imágenes, rango, y la presencia de objetos sin tener contacto físico.

2.2.1.1 SENSORES DE CONTACTO.

Este tipo de sensores incluyen sensores tactiles, los cuales usados para obtener información del contacto entre los dedos de la mano del manipulador y los objetos situados en el espacio de trabajo. Normalmente son montados en la superficie interior de cada dedo o efector final, son mucho más ligeros que la mano y sensibles a fuerzas muy pequeñas. La información que se puede obtener es la identificación de objetos con su posición y orientación, así como de obstáculos no previstos.

Estos sensores se clasifican en : binarios y análogos.

Los binarios son dispositivos de contacto semejantes a swtiches. La salida de estos sensores puede ser fácilmente incorporada a la computadora para controlar el manipulador. Un simple sensor de contacto binario consite de dos microswitches, cada uno colocado en las partes interiores de cada dedo, los cuales cierran sus contactos, generalmente, como consecuencia de la presión, aportando una información muy importante acerca de las proporciones de las piezas. Se fabrican también membranas de Mylar con una matriz interna de puntos de detección por contacto y que, mediante un decodificador, informan sobre el area de presión del objeto que se está en ese momento sujetando.

Los sensores de contacto análogos son dispositivos cuya salida es proporcional a una fuerza local; normalmente se montan en el interior de los dedos para medir fuerzas de sujeción y para extraer información acerca del objeto sujetado entre los dedos.

Dentro de los sensores de contacto podemos incluir a los presductores, que son, en general, células digitales de presión. Pueden emplearse conjuntos de presductores, en forma de matriz, para situar la posición de una pieza.

2.2.1.2 SENSORES DE NO CONTACTO.

Aquí podemos tener :

++ Sensores de visión.
++ Sensores acústicos (sonar).

SENSORES DE VISION.

Para que el robot se adapte a su entorno y lleve a cabo sus tareas, debe tener acceso una información completa, la cual puede estar contenida en imàgenes del espacio que lo rodea.

La importancia de la percepción visual está propiciando intensas investigaciones; aunque en la actualidad los equipos de visión para la robótica son caros e imprecisos, son de dos dimensiones y aplicables a una tarea específica.

Un sistema de visión ideal optimizará :

- Costo.- que sea bajo.

- Flexibilidad.- poder cambiar fàcilmente el programa de trabajo (con ayuda de la inteligencia artificial). Por ahora son difíciles los algoritmos de tratamiento de información visual lo cual ocasiona que se desarrollen para trabajos específicos.

- Precisión.- que la calidad del trabajo sea buena. La precisión se ve limitada debido a la alteración de los parámetros de las imágenes de acuerdo con las condiciones ambientales (luz, movimientos, sombras, ruidos, etc.).

- Velocidad.- el hecho de trabajar en tiempo real ocasiona tener que procesar las escenas rápidamente, para actuar de acuerdo con su información. Esto exige que se disminuya la cantidad de información afectando la confiabilidad. Hasta ahora, los procesos de tratamiento de imágenes y algoritmos empleados son muy complejos, lo que lleva a tener tiempos de respuesta muy largos y potentes computadoras con una gran capacidad de memoria.

La fig. 2.13 muestra cómo la información visual cierra el lazo de realimentación más efectivo para el sistema de control.

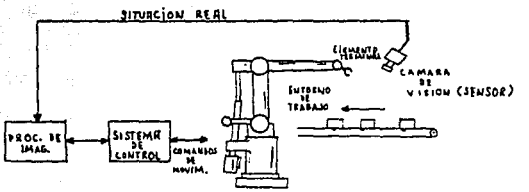


fig. 2.13. Sensor visual realimentando al sistema de control.

En resumen, los procesos realizados por un robot con capacidades sensoriales son :

* Destección (sensa) de imàgenes.
* Procesamiento de imágenes.
* Intercomunicación con el sistema de control.
* El sistema de control toma decisiones en función de la información sensorial y manda ordenes al manipulador sobre las acciones que debe ejecutar.

Los algoritmos de visión nos informan sobre la identificación de objetos, su posición y características pertinentes.

En la mayoria de los sistemas robòticos que cuentan con procesamiento de información visual se tienen las siguientes fases fundamentales :

+ PERCEPCION.
+ PREPROCESO DE LA INFORMACION.
+ SEGMENTACION.
+ RECONOCIMIENTO.
+ INTERPRETACION.

PERCEPCION.

Existen muchos sensores para adquirir informació de una imagen, pero las cámaras de TV de estado sólido son las màs usadas.

Tambièn presentan interès los conjuntos lineales de dispositivos de acoplo de carga (CCD). Estos conjuntos consisten de una serie longitudinal de celdas CCD, cada una de las cuàles nos da un voltaje proporcional a la iluminación recibida. Estas señales (voltajes) se almacenan en un registro de corrimiento para su posterior procesamiento.

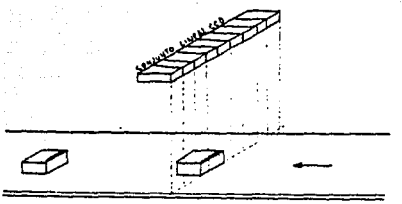


fig. 2.14. Conjunto lineal CCD.

En la fig. 2.14 se observa la actuación de un conjunto lineal CCD.

Ahora bien, el dispositivo lineal CCD sòlo proporciona información de la linea sobre la que trabaja.

--> La conversión de escenas tridimensionales del mundo real en imágenes bidimensionales ocasiona que se planteen problemas serios en la visión robótica. Con esto, surgen problemas de ocultaciones, sombras, distorsiones por perspectiva, etc.

PREPROCESO.

Una vez conocidas las características de la cámara de T.V. así como la forma de conseguir la digitalización de la imagen y la calibración automática. La siguiente fase del tratamiento de la información obtenida consiste en un "preproceso". Básicamente, consiste en la transformación de la imagen captada por la cámara, con una determinada gama de niveles de gris, en una imagen binaria mediante una comparación de los niveles de gris con un umbral fijo o variable.

Esta fase se dedica, principalmente, al realce de la imagen y a la reducción del ruido. Por lo general, consume demasiado tiempo, por lo que, en las aplicaciones de la robótica se tratan de minimizar. La fig. 2.15 muestra la forma de la representación de las imágenes.

Observar que la coordenada z se destina a reflejar la intensidad luminosa en función de las coordenadas (x,y).

La matriz de elementos fotosensibles constituye el conjunto de pixels que se posicionan sobre el plano (x,y). La coordenada z representa el nivel luminoso que la cámara cuantifica en varios escalones para simular el estado de la imagen (ver fig. 2.16).

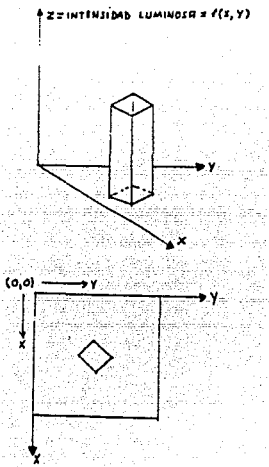


fig. 2.15. Forma de representar las imàgenes.

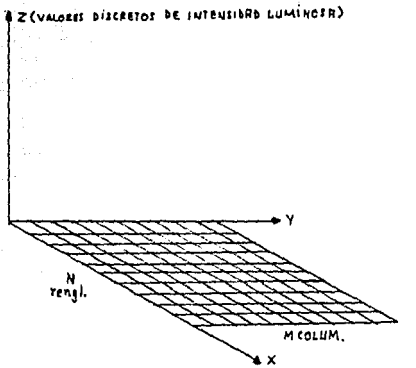


fig. 2.16. Conjunto de pixels en el plano x,y.

SEGMENTACION.

En esta fase, se obtienen y determinan los objetos de interés de una escena.

Los algoritmos más empleados en la segmentación siguen dos principios distintos :

a) Discontinuidad o "detección de borde".
b) Similaridad o "crecimiento por región".

Un método de segmentación consiste en definir un "umbral", para distinguir el fondo de los objetos. En la fig. 2.17, todos los niveles de gris inferiores al umbral U son fondo; los superiores, se consideran pertenecientes al objeto.

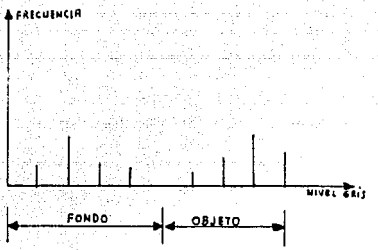


fig. 2.17. Mètodo de segmentaciòn.

Otro procedimiento para la segmentación emplea "plantillas", que son formaciones diseñadas para detectar cierta propiedad invariable en una región dad. La imagen se rastra con la plantilla renglón a renglón, de modo que su centro pase, sucesivamente, por cada uno de los pixels, cuantificando su valor binario (1 ó 0). Además de plantillas para el análisis puntual de cada pixel, existen, tambièn, plantillas de linea y de gradiente.

RECONOCIMIENTO e INTERPRETACION.

Una vez detectado el borde de una figura en la fase de segmentación, la aplicación del "código cadena" permite obtener información útil para la ubicación y reconocimiento de los objetos. Esta información proporciona el área, el perímetro, el centro de gravedad, la distancia entre un extremo y otro del objeto, etc.

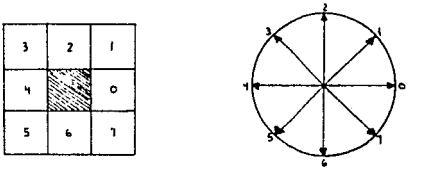


fig. 2.18. Còdigo de cadena de ocho direcciones.

El código cadena es una estructura de datos para almacenar contornos de objetos en memoria. Consiste en un listado de las coordenadas del punto inicial y de los valores

que indican las direcciones siguientes del contorno. En la fig. 2.18, se presenta un código de cadena de ocho direcciones.

Una imagen de borde binaria se codifica en tres pasos :

1.- Rastrear la imagen hasta encontrar un punto desde el cual iniciar la cadena.

2.- Codificar en cadena todos los bordes que no formen bordes cerrados.

3.- Codificar en cadena los bucles cerrados.

En el primer paso, se explora cada pixel de la imagen y se determina cuántos pixels de borde son vecinos, pudiéndose dar los siguientes casos :

a) Pixel sin vecinos: punto aislado.
b) Pixel con un vecino: punto final.
c) Pixel con dos vecinos: punto medio (ignorarlo).
d) Pixel con más de dos vecinos: punto nudo o bifurcación.

Una vez establecido y aplicado el código cadena de una imagen, una serie de algoritmos matemáticos determinan con rapidez la longitud de la cadena, su altura y anchura, los momentos, el centroide, etc.

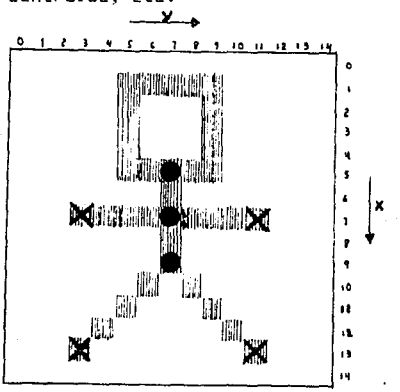


fig. 2.19 Aplicación del código cadena a una escena.

La fig. 2.19, es un ejemplo de aplicación del código cadena a una escena. con puntos finales, medios y nudos.

SENSORES ACUSTICOS (SONAR).

Las características especiales de propagación de las ondas ultrasonoras, de mas de 15000 Hz, unidas a la existencia en el mercado de conjuntos emisores-receptores de bajo precio, con un funcionamiento fácil y seguro, ha determinado que se empleen en la robótica, para la detección de objetos y cálculo de distancias.

Cuando las ondas ultrasonoras que circulan por un medio chocan con un medio diferente, una parte de ellas se refleja a su origen. Teniendo en cuenta la velocidad de propagación y el tiempo que transcurre hasta el regreso de las ondas al emisor, se puede calcular, fácilmente, la distancia entre el foco de ultrasonidos y el objeto reflector, así como el espesor de los materiales.

En la fig. 2.20, se muestra un diagrama de bloques de un sistema ultrasónico capáz de incorporarse a robots móviles.

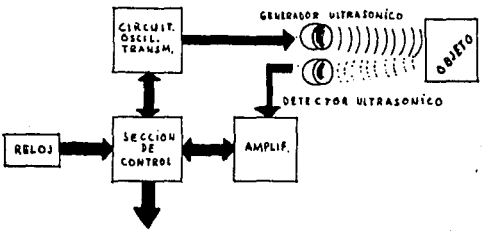


fig. 2.20. Sistema ultrasónico.

Para evitar interferencias, los generadores emiten, secuencialmente, varias ondas de diferentes frecuencias; por ejemplo, de 48 a 64 MHz, realizándose la detección en el mismo orden, y consiguiendo, al mismo tiempo, una gran exactitud. La posibilidad de montar los sistemas ultrasónicos en plataformas giratorias supone la exploración de todo el entorno. Utilizando dos o más sistemas colocados en diferentes puntos y combinando sus resultados, se puede obtener un mapa general del entorno, con los obstáculos que rodean al robot.

2.3 LENGUAJES DE PROGRAMACION PARA ROBOTS.

El lenguaje, es el medio que utiliza el hombre para comunicarse con el robot, de este modo, el rendimiento y productividad de este radica en una buena programabilidad, fundamentada en el empleo de un lenguaje adecuado.

Debido a que los lenguajes clàsicos de programaciòn (BASIC, FORTRAN, etc) carecen de recursos para aproximarse a las caracteristicas y trabajos que efectuàn las màquinas, los fabricantes y centros de investigaciòn trabajan con intensidad en el diseño de lenguajes especiales.

Aunque en la actualidad se han desarrollado lenguajes de programaciòn para robots, estos han tenido las siguientes caracter isticas:

- Cada lenguaje se ha diseñado tomando, como base, a algùn robot concreto del mercado.

- Algunos de ellos se han diseñado para alguna aplicaciòn especifica.

Un hecho importante que marca la evoluciòn de los robots se basa en la mejora de sus recursos de software. En la actualidad, la evoluciòn de los lenguajes nos va llevando hacia la programaciòn textual en la cuàl se permite la depuraciòn y creaciòn de programas de trabajo fuera de linea.lo cual se justifica por el hecho del alto costo de los manipuladores, el tiempo empleado en la creaciòn de programas y la posibilidad de que el dispositivo a programar este operando conjuntamente con otras manipuladores.

El interès de los fabricantes en desarrollar sistemas de percepciòn del entorno cada vez mas avanzados, haciendo uso de tècnicas de inteligencia artificial para la valoraciòn del mundo exterior y la determinaciòn de los planes de accion alternativos nos esta llevando al desarrollo de sistemas vàlidos para cualquier robot y aplicaciòn.

A continuaciòn, vamos a tratar de una forma explicita los distintos niveles de programaciòn que existen en la actualidad asi como sus caracterìsticas importantes, para finalmente hacer una serie de conclusiones respecto a los lenguajes mas idòneos para el sistema controlador de trayectorias desarrollado por nosotros.

2.3.1 PROGRAMACION GESTUAL O DIRECTA.

En este tipo de programaciòn, el mismo brazo manipulador interviene en el trazado del camino y en las acciones a desarrollar en la tarea de aplicacion deseadas. Esta caracteristica determina la programaciòn on-line.

La programación Gestual se subdivide en dos clases:

- Programaciòn por aprendizaje directo.

- Programaciòn mediante un dispositivo de enseñanza.

En el aprendizaje directo, el punto final del brazo se traslada con ayuda de un dispositivo especial colocado en su muñeca, o utilizando un brazo maestro sobre el que se efectuan los desplazamientos los cuales al ser memorizados pueden ser repetidos por el manipulador.

La tècnica de aprendizaje directo se utiliza extensamente en talleres de pintura. El operario conduce la muñeca del manipulador o del brazo maestro, determinando los tramos a recorrer y aquellos en los que la pistola debe expulsar una cierta cantidad de pintura. Con este tipo de programaciòn, los operarios sin conocimiento de software, pero con experiencia en el trabajo a desarrollar pueden programar los robots eficientemente.

La programaciòn mediante un dispositivo de enseñanza consiste en determinar las acciones y movimientos del brazo manipulador a traves del elemento especial para este cometido. El dispositivo de enseñanza, suele estar construido por botones, teclas, o joysticks.

2.3.2 PROGRAMACION TEXTUAL.

En este caso, el programa queda constituido por un texto de instruccciones o sentencias cuyo desarrollo no requiere de la instrucciòn del robot, es decir se efectua off-line. Con este tipo de programaciòn el operador no define pràcticamente las acciones del brazo manipulador , sino que se calculan en el programa mediante el empleo de las instrucciones textuales adecuadas.

La programaciòn textual tambièn se subdivide en dos grupos:

- Programaciòn textual explìcita.

- Programación textual especificativa.

Programación textual explìcita.

En este caso, el programa consta de una serie de ordenes o instrucciones concretas que van definiendo con rigor las operaciones necesarias para llevar a cabo la aplicaciòn. Se puede decir que la programaciòn explìcita, engloba a los lenguajes que definen al movimiento punto a punto, similares a los de programaciòn gestual, pero bajo la forma de un lenguaje formal. Con este tipo de programaciòn las situaciones anormales como colisiones y su tratamiento son responsabilidad del programador.

La programación textual especificativa, consiste de una programación del tipo no procesal, en la que el usuario describe las especificaciones de los productos mediante su modelado, al igual que las tareas que hay que realizar sobre ellos. El sistema de información para la programación textual especificativa debe disponer de un modelo del universo,o un mundo donde se desenvuelve el robot, este modelo sera una base de datos mas o menos compleja, segun la clase de aplicación,pero que requiere siempre de computadoras potentes para el procesado de gran cantidad de información. El trabajo de programación consistirá, en la descripción de las tareas a realizar, lo que supone poder llevar a cabo trabajos complicados.

Actualmente los modelos del universo que se utilizan son del tipo geométrico, asimismo,este tipo de programación se subdivide en programación textual especificativa , en la que el lenguaje trabaja con objetos ellos, y en la que la programación se realiza off-line haciendo posible la conexion con sistemas CAM. Si la programación se orienta hasia los objetivos, se tendrá un tipo de programación textual especificativa en la que se define al producto final.

2.3.3 LENGUAJES DE PROGRAMACION.

2.3.3.1 WAVE.

CARACTERIST. GRALES. : el movimiento es preplaneado y sólo se hacen muy pequeñas desviaciones durante su ejecución.

LUGAR DE DESARROLLO : Universidad de Stanford.

TIPO DE LENGUAJE : Propósito general.

IMPLEMENTACION : Se ensambla en una PDP-10 y se ejecuta su archivo ejecutable en una PDP-6.

ESTRUCTURA DE WAVE : Algoritmos para planeación de trayectorias, compensación dinámica, computo intensivo, asi como funciones de alto nivel que especifican posición, rotación, y coordenadas de juntura.

El siguiente programa escrito en WAVE sirve para tomar una punta y colocarla dentro de un agujero:

TRANS PIN... : localidad de la pinza
TRANS HOLE... : localidad del agujero

ASSIGN TRIES 2 : número de agarres a ser atendidos

```
MOVE  PIN        : muevete a la pinza, primero sobre el eje +Z,
                   y despues un punto abajo de la punta, con la
                   direccion  -Z.
PICKUP           :
CLOSE1           : agarra la punta
SKIPE 2          : la siguiente instrucciòn,si el error
                   2 ocurre:
ERROR 2          : cierra la pinza...
JUMP OK          : si no ocurre,ir a OK
OPEN 5           : error si ocurre,abre los dedos
CHANGE,Z,-1,
NIL,0,0          : muevete hasia abajo una pulgada
SOJG  TRIES,
PICKUP           : decrementa la prueba,si no es negativa salta
                   a PICKUP.

WAIT NO PIN      : imprime NO PIN y espera por el operador
JUMP  PICKUP     : prueba otra vez cuando los tipos
                   de operadores procedan.
OK               :
MOVE HOLE        : muevete abajo del agujero.
STOP FV NIL      : detente en 50 oz.
CHANGE Z,-1,
NIL,0,0          : prueba para ir hasia abajo una pulgada
SKIPE 23         : error 23, falla al detenerse
JUMP NO HOLE     : error si no ocurre(...)
FREE  2,X,Y      : procede  con  la  inserciòn,por  la
                   complementaciòn a traves de los ejes x,y
SPIN  2,X,Y      : tambien complementado con torques  a
                   traves de x,y

STOP FV,NIL      : detente en 50 oz
CHANGE Z,-2,
NIL,0,0          : efectùa la inserciòn

NO HOLE          :
WAIT NO HOLE     : falla
```

2.3.3.2 AL

CARACTERIST .GRALES. : Lenguaje estructurado de programaciòn explicita basado en Pascal concurrente, AL provee construcciones para el control de mùltiples brazos en movimientos cooperativos.

LUGAR DE DESARROLLO : Laboratorio de Inteligencia Artificial de la Universidad de Stanford.

IMPLEMENTACION : Actualmente los programas AL son compilados en una PDP-10 . El còdigo resultante se descarga en una PDP-

	11/45 en la que se ejecuta en tiempo real.
USOS :	Control de dos PUMAS y de dos brazos STANFORD al mismo tiempo.

Un programa en AL para el problema de colocar en el agujero:

```
BEGIN "inserta el clavo dentro del agujero"
FRAME ..._..,..._pinza,..._...,...,..;
{  los ejes coordenados representan la posicion actual de las
caracterizticas de los objetos }
....<---FRAME(nilrot,VECTOR(20,30,0)*pulgadas);
....<---FRAME(nilrot,VECTOR(25,35,0)*pulgadas);
{ agarrando la posicion relativa a .....}
peg_grasp <---FRAME(ROT(XHAT,180*grados),3*zhat*pulgadas)
tries <---2;
grasped(--FALSE;
{  el tope en el agujero esta definido a tener  una  relaciòn
fija a el ...}
AFFIX tope_agujero a ..._agujero* peg*agarre
WHILE NOT agarre AND i<prueba DO
BEGIN "atiende el agarre"
CLOSE ...TO 0*inches;
{inicia el movimiento a el ..,note la matriz destino}
WHILE NOT agarre AND i<tries DO
BEGIN"atiende el agarre"
CLOSE bhand TO 0*inches;
IF ...<peg_diameter/2+1*inches;
MOVE barm TO -i*inches
```

2.3.3.3 VAL

CARACTERIST. GRALES. :	Es un lenguaje diseñado para la generaciones que envuelven operaciones que envuelven posiciones de robot predefinidas.
IMPLEMENTACION :	Es un intèrprete.
CAPACIDADES :	Interpolaciòn de juntura, movimiento cartesiano, especificaciòn de matrices, coordenadas, variables enteras aritmèticas y procedimientos.
USO :	Fue inicialmente usado con el manipulador PUMA (programable universal machine for assembly) pero los controladores de VAL estàn actualmente disponibles para muchos de los robots UNIMATE y SERIE 4000.

USUARIO : Ingeniero manufacturador.

A continuación vemos un programa escrito en VAL:

```
SETI  TRIES=2
REMARK          si la mano se cierra a menos de 100 mm, ir a
                sentencia etiquetado con un 20.
10 GRASP 100,20
REMARK          en otro caso continuar al estado 30
GO TO 30
REMARK          abre los dedos,desplaza hacia abajo a través
                del eje z y prueba otra vez
20    OPENI 500
DRAW 0,0,-200
SET I  TRIES=TRIES-1

IF TRIES GE 0 THEN 10
TYPE NO PIN
STOP
REMARK          mueve 300 mm sobre el agujero siguiendo una
                linea recta
30 APPROS HOLE,300
REMARK          si la señal es activada durante el siguiente
                movimiento
REACTI 3,ENDIT
APPROS HOLE,200

REMARK
MOVES  HOLE
```

2.3.3.4 PAL.

CARACTERIST. GRALES. : Consiste de una secuencia de ecuaciones coordenadas homogènea de los objetos y los robots del efector final.
Los programas PAL manipulan matrices coordenadas bàsicas que definen la posición de las características claves del robot.

CAPACIDADES : PAL define matrices coordenadas bàsicas, que definen la posición claves del robot. Se permite la redefinición de matrices ,lo que dota de gran flexibilidad en el posicionamiento.

Sus extensiones tratan con información sensorial, la cual sirve para definir el valor actual de algunas matrices coordenadas de las ecuaciones coordenadas.

2.3.3.5 AML

CARACTERIST. GRALES. : Dar un sistema de medio ambiente donde diversas interfaces para manipuladores puedan ser construidas.

Proveer un lenguaje poderoso con simples subconjuntos para uso por programadores con amplia experiencia.

IMPLEMENTACION : Un intèrprete implementa el lenguaje base y define las operaciones primitivas como las reglas para manipular vectores y otros objetos agregados que son requeridos para describir un medio ambiente de robots.

CAPACIDADES : Soporta planeaciòn de trayectorias, sujeto a restricciones de posiciòn, velocidad y movimiento, asimismo, soporta movimiento cartesiano.

Soporta operaciones en vectores, rotaciones y ejes coordenados.

USO : AML se utiliza para el control de ensamble de los robots 7555.

RESTRICCIONES : No soporta movimiento general.

Ejemplo de programa escrito en AML:

```
PICKUP : SUBR(PART_DATA,TRIES)
MOVE(GRIPPER,DIAMETER(part_data)+0.2)
IF TRIES LT 1 THEN RETURN ('no part')
IF TRIES LT 1 THEN RETURN('no part')
DMOVE(3,-1,0);
IF GRASP(DIAMETER(PART_DATA)+0.2)
IF TRIES LT 1 THEN RETURN ('no part')
DMOVE(3,-1,0)
IF GRASP(DIAMETER(part_data))='no part'
THEN TRY_PICKUP(part_data,tries-1)
END;
GRASP:SUBR(DIAMETER,F);
FMONS:NEW APPLY(smonitor,pinch-force(f));
MOVE(pinza,0,fmons);
RETURN  (IF Qposition(gripper) le diameter/2 then 'no part'
 else 'part');
END;
INSERT:SUBR(parte_datos,cavidad);
FMONS:NEW APPLY(smonitor,tip_fuerza(landing_fuerza);
MOVE(<1,2,3>,agujero +<0,0,0.25));
DMOVE(3,-1,0,fmons);
```

2.3.3.6 AUTOPASS

CARACTERIST. GRALES. :	Sistema de programaciòn automàtica para montaje automàtico controlado por computadora. Utiliza instrucciones muy comunes en el lenguaje inglès. prevee colisiones y genera acciones a partir de las situaciones reales.
LUGAR DE DESARROLLO :	IBM Corporation
USOS :	Ensamblaje de piezas
IMPLEMENTACION :	Hace uso de un computador de varios Megabytes, de capacidad de memoria.
ESTRUCTURA :	Realiza todos sus càlculos sobre una base de datos, que define a los objetos como poliedros de un màximo de 20 000 caras, ademàs, està escrito en PL/1 y es intèrprete y compilable.

El siguiente programa, coloca la parte inferior del cuerpo C1 alineada con la parte superior del cuerpo C2. Asimismo, alinea los orificios A1 y A2 de C1,con los correspondientes de C2.

```
PLACE C1
SUCH THAT C1 BOT CONTACTS C2TOP
AND B1A1 IS ALIGNED WITH C2A1
AND B1A2 IS ALIGNED WITH C2A2
```

2.3.4 TABLAS COMPARATIVAS DE LOS LENGUAJES DE PROGRAMACION

LENG. PROGR.	LENGUAJE	NUM. BRAZOS	SENSORES DE VISION
FUNKY	progr. punto a punto		tacto
T3	ensamblador	1	limitado a switches
ANORAD	NC-program.	1	
EMINLY	ensamblador	2	tacto
RCL	fortran	1	
SIGLA	ensamblador	1-4	
RPL	fortran	1	tacto, posic. visiòn orientac.
VAL	ensamblador	1	visiòn
HELP	pascal	1-4	
MAPLE	pl/1	1	fuerza reconocim. torque
HELP	Pascal	1-4	
MAPLE	pl/1	1	fuerza proximidad
MCL	apt	1	tacto visiòn
PAL	transformaciones	1	
AUTOPASS	pl/1	1	sensores

LENGUAJE	MANUFACTURADOR	CONF. BRAZO	NO. EJES	ROBOT
T3	Cincinnati Milacron	rrrrrr	6	T3
RPL VAL	Unimation	rrrrrr	6	Puma
AL PAL	Sheinmann	rrprrr	6	Standfor
AUTOPASS FUNKY MAPLE	IBM	rpppprrr	7	IBM
RCL	Bendix	rpprrr	6	Pals
HELP	Allegro	ppprrr	6	Allegro
ANORAD	Anorad	pppr	4	Anomatic
SIGLA	Olivetti	pppp	4	Sigla

r=juntura rotacional p=juntura prismàtica

ESTRUCTURAS DE CONTROL	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
DECLARACIONES		X	X	X	X		X	X
IF THEN	X	X	X	X	X		X	X
IF THEN ELSE	X	X	X	X	X	X	X	X
WHILE DO	X	X	X	X	X	X	X	X
DO UNTIL	X	X		X		X	X	
CASE	X			X		X	X	
FOR	X	X		X		X	X	
BEGIN END	X	X		2	X			
COBEGIN COEND	X							
SUBRUTINA	X	X	X		X	X	X	X

TIPOS DE DATOS GEOMETRICOS								
	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
MATRIZ	X			X	X	X		X
ANGULOS DE JUNTURA		X		X				X
VECTOR	X	X		X	X			
TRANSFORMACION	X			X	X			X
ROTACION	X			X	X			

MODULOS DE SOPORTE

	AL	AML	HELPS	JARS	MCL	RAIL	RPL	VAL
Editor de textos	1	X	3	3		X	X	X
Sistema de archivos	1	X	3	3		X	X	X
Interprete	X	X	X			X	1	X
Compilador	X			X	X		X	
Simulador	X	2			X			
Macros	X		X		X			
Sent. inclución	X	X						
Logica de errores	X							
Func. auxiliares	X	X						

1= AL usa las caracts. de soporte del S.O. de la PDP-10

2= Un simulador ha sido desarrollado en el centro de investigación IBM Watson.

3= JARS y HELP usan las caract. del sistema de soporte del S.O. de la R.T-11.

HABILIDAD PARA EL CONTROL DE MULTIPLES BRAZOS

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Brazo Unico		X		X		X	X	X
Brazos Màltiples	X	1	X		X			

1=no comercialmente disponible

MODOS DE CONTROL

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
POSICION	X	X	X		X	X	X	X
RETROALIM VISUAL	3			3		3	3	3,5

DESPLIEGUE Y ESPECIFICACION DE ROTACION

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
MATRIZ DE ROTACION	1			3				
ANGULOS EULER	2	X		X		X		X
ROLL,PITCH,YAW	3		3		3			

1=Despliega rotaciones con una matriz de rotaciòn
2=AL acepta dirèctamente la especificaciòn de una orientaciòn por 3 àngulos de euler
3=la orientaciòn AL puede tambièn ser especificada por los ángulos roll,pitch,yaw

Conclusiones acerca de los lenguajes de programación analizados.

En el aspecto de claridad y sencillez, la programación gestual, es la mas eficaz, pero necesita de la creación de programas propiamente dichos. Los lenguajes a nivel de movimientos elementales como VAL disponen de bastantes comandos para definir acciones muy parecidas, que fueron surgiendo segun las necesidades y por tanto que obscurecen su comprensión y entendimiento. Si bien un lenguaje que tenga una aplicación sencilla de aplicación tenemos al MCL, el cual esta dedicado a las màquinas herramientas APT.

El lenguaje PAL es adecuado para el personal familiarizado en transformaciones. Debido a su sintaxis, muy parecida al lenguaje inglès, el AUTOPASS es uno de los lenguajes mas fàciles de utilizar. Los lenguajes AL, MAPLE, MCL, tienen comandos para el control de la sensibilidad del tacto de los dedos. Los lenguajes PAL y HELP, carecen de capacidad de adaptación sensorial. Los lenguajes estructurados de programación explicita, (AL, PAL, MAPLE), tienen estructuras de datos de tipo complejo.

2.4 APLICACIONES

Las posibles aplicaciones de los robots a la industria son tantas y con tan variadas tecnologias que su comentario y descripción detallada requeriria un volumen dedicado especialmente al tema. Por lo tanto sólo se enumerarán y describirán brevemente las aplicaciones más usuales como son:

- Soldadura por puntos
- Soldadura por arco
- Manejo del vidrio
- Forja
- Prensa
- Fundición
- Alimentación de màquinas herramienta
- Pintura
- Ensamblaje

2.4.1 Soldadura por puntos

Esta es una de las aplicaciones más tipicas de los robots industriales y también una de las primeras que se emplearon, principalmente en la industria del automovil. En esta aplicación el robot, fijado a una estación de trabajo, maneja una pistola de soldadura por puntos para soldar diversas partes de la carroceria de un automovil (marcos de puertas, aberturas de ventanas, paneles, etc.), colocando un punto de soldadura a una velocidad media de uno cada 1.5 segs. Una carroceria de coche moderno puede tener de 1400 1800 puntos de soldadura colocados por robots.

La soldadura por puntos es un trabajo que se adapta muy bien al robot, puesto que la pistola de soldadura puede pesar de cinco a treinta y cinco kilos y el manejo de esta por una persona puede ser muy fatigante. Es muy probable que un soldador humano cansado no coloque los puntos de soldadura con la misma seguridad y precisión que un robot, y como las cadenas de montaje de carrocerias casi siempre funcionan con turnos seguidos, el rendimiento económico no seria apropiado.

2.4.2 Soldadura por arco

Este tipo de aplicación es mucho más utilizado en la industria que la soldadura por puntos y, por lo tanto, la aplicación potencial para los robots es también superior. Sin embargo, este trabajo exige que el robot sea más sofisticado que para la soldadura por puntos debido a que el robot debe poder mover la herramienta de soldadura sobre una linea continua a una velocidad constante. Esto no es necesario para un robot de soldadura por puntos, ya que solo necesita saltar de un punto a otro y no tiene que mantener un control tan estricto sobre el recorrido de la herramienta.

La soldadura por arco es un trabajo pesado para una persona debido a los humos que se desprenden, fragmentos de metal fundido y luz ultravioleta que puede dañar la vista. El robot supera en porcentaje de tiempo de soldadura a una persona que constantemente debe detenerse para examinar su trabajo. Con la tecnologia actual, la soldadura de arco mediante robots se restringe generalmente a series de gran volumen, donde las partes encajan perfectamente. Desafortunadamente esto sólo es una pequeña parte de todo el proceso de soldadura por arco. En la mayoria de los casos, no se mantienen las tolerancias estrechas y la calidad del trabajo depende de la habilidad del operario.

2.4.3 Manejo del vidrio

Uno de los trabajos en donde la aplicación de los robots resulta ideal esta en las industrias que trabajan con vidrio caliente para fabricar artefactos como son los tubos de imagen de televisión. Los embudos y las pantallas de televisión se hacen a partir del vidrio fundido.

La aplicación resulta ideal para robots porque estos pueden estar descargando moldes y colocando piezas en hornos de templado a una temperatura extrema, las veinticuatro horas al dia, siete dias a la semana, dando por resultado un rendimiento muy elevado. Lo anterior supone grandes exigencias en el diseño de los robots, pues estos deben ser muy confiables, debido a que cuando una fundición ha comenzado la operación no puede interrumpirse. Después de cada parada son necesarias muchas horas para hacer que funcione de nuevo la producción. Un robot que no pueda proporcionar una eficacia de al menos 99.7% es inadecuado para el trabajo, porque la perdidas en tiempo son demasiado caras.

2.4.4 Forja

La caracteristica de los robots que justifica su utilización en forja es la capacidad de trabajar correctamente en ambientes hostiles al ser humano, durante largos periodos de tiempo. Cuando se ocupan operarios para el trabajo de forja, es necesario dotarles de un manipulador que les permita manejar las piezas pesadas y a elevada temperatura. Esto no es necesario si se utilizan robots que tienen la capacidad de soportar grandes cargas y equipados con una mano de sujeción o garra diseñada especialmente. Hay también sistemas más complejos en los que los dedos de la mano de sujeción se mueven manteniéndose paralelos entre si, dando al robot una capacidad mayor de manipulación.

2.4.5 Prensa

La prensa es una tarea donde, de la misma forma que el trabajo de forja, la utilización de robots supone eliminar un trabajo peligroso y en un ambiente hostil al ser humano. Las chapas de material que se utilizan en prensa, vienen frecuentemente con rebabas que pueden cortar al operario. Pueden producirse también roturas de partes, que salen proyectadas o puede haber accidentes cuando se intenta liberar piezas atascadas (aunque hay mecanismos que bloquean la máquina cuando hay algún incidente o se intenta introducir las manos).

Aunque la aplicación de los robots para los trabajos de prensa puede parecer sencilla, existen algunos problemas como es el modo de alimentación, debido a que si los robots no tienen capacidad de visión, las piezas deben suministrarse de forma ordenada y siempre en la misma posición.

2.4.6 Fundición

Una de las industrias en donde son muy utilizados los robots es en las instalaciones de fundición inyectada y, de forma especial, en la industria del automovil. La tarea del robot es la descarga de piezas fundidas, asi como el corte de metal sobrante y la eliminación de rebabas.

Hay varias ventajas que resultan de utilizar robots en una instalación de fundición inyectada. En primer lugar, la productividad aumenta entre un 30% y un 40%, además los gastos de mantenimiento se reducen y, también se ha observado una disminución del ausentismo laboral del personal de estas instalaciones. Esto se debe muy posiblemente a que, mientras en una instalación tradicional, las condiciones de trabajo son muy nocivas, en una instalación automatizada los robots se encargan de hacer el trabajo más peligroso.

2.4.7 Alimentación de máquinas herramienta

Para este trabajo, la aplicación de robots es evidente, encontrando su principal interes en la posibilidad de atender a varias máquinas herramienta a la vez. Una máquina herramienta es un dispositivo que realiza un cierto proceso a un tipo de material; un ejemplo de este tipo de máquinas son los tornos mecánicos. Esta aplicación se debe a que los procesos de mecanizado duran con frecuencia varios minutos, por lo que en este tiempo, el robot puede hacer otras cosas como es la carga y descarga de otras máquinas. Si el número de máquinas es reducido (dos o tres), el robot puede estar rodeado de ellas y atenderlas, pero si hay un número mayor de máquinas es preciso que el robot tenga siete u ocho grados de libertad y se desplace para atender a todas las máquinas.

En ocasiones cuando se utilizan brazos de seis grados de libertad (independientemente de los de desplazamiento de todo el robot), la alimentación y la descarga de las máquinas plantea problemas de interferencia. Para resolver esto, se debe recurrir a robots con más grados de libertad.

2.4.8 Pintura

La pintura industrial, principalmente en la industria del automovil y de los aparatos electrodomésticos, es uno de los mejores trabajos para los robots y una de las aplicaciones donde los robots liberan al hombre de un trabajo en condiciones muy incómodas. Estas se deben a los gases y aerosoles que se producen en la pintura con pistola, asi como al ruido producido por la instalación de aire comprimido que se utiliza como propulsor.

En la pintura a pistola el robot debe poder seguir un recorrido continuo, pero debido al barrido de la pistola, la presición no es imprescindible. En los sistemas más comunes, un pintor experto dirige literalmente el brazo del robot de un punto a otro con la pistola en funcionamiento. Si el robot aspersor conducido manualmente ha hecho un buen trabajo, el programa se almacena y luego puede repetir la misma acción. Estos robots se utilizan actualmente en todo el mundo, siempre que la pintura o la selladora se aplique en series de producción largas o medias.

2.4.9 Ensamblaje

El ensamblado de piezas para construir un cierto conjunto es una de las tareas para los robots que reviste más dificultades. Esto se debe a que, cuando las piezas son muy pequeñas y de geometria complicada, la manipulación de estas por el robot requiere una correcta coordinación entre los movimientos del robot, los de las piezas que se desean ensamblar y los diferentes canales de alimentación de componentes.

Un caso muy frecuente de montaje es cuando hay que añadir una serie de elementos a un bastidor o pieza base. Estos elementos deben ser colocados en diferentes partes del bastidor, por lo que es conveniente situar a este en una plataforma que permita girarlo y trasladarlo según una secuencia determinada. A veces las necesidades de montaje requieren tal precisión, que el tipo de robot a emplear viene condicionado por esta misma exigencia.

3. PLANEACION DE TRAYECTORIAS PARA UN MANIPULADOR

3.1 INTRODUCCION.

El movimiento de un manipulador durante la ejecución de una tarea consiste en el desplazamiento de la mano de sujeción o de la herramienta, desde un punto A hasta un punto B. En este desplazamiento, las posiciones extremas son conocidas (A y B) mientras que la trayectoria intermedia para el traslado no.

La determinación de la trayectoria antes de ejecutar el movimiento es muy importante, en especial en operaciones de coger, mover y colocar un objeto; puesto que de no hacerlo podría haber colisiones con otros objetos que se encuentran en el mismo espacio de trabajo.

El propósito de este capítulo es describir el Sistema de Planeación de Trayectorias Tridimensionales (SPTT) para un manipulador fijo, el cuál utiliza una representacion gráfica tridimensional del manipulador Stanford y realiza la planeacion de trayectorias para un manipulador en un ambiente de trabajo poco estructurado, Para esto, hemos dividido el sistema SPTT en dos subsistemas: Sistema Controlador de Trayectorias (SCT), encargado de la planeación de trayectorias para el traslado de objetos tridimensionales en un medio ambiente poco estructurado desde su posición inicial a su posición final; y el Sistema Graficador de Trayectorias (SGT) diseñado para desplegar la representación tridimensional en pantalla de la computadora del manipulador Stanford, y de la manipulación de los objetos haciendo uso de la trayectoria generada por el subsistema controlador de trayectorias (SCT). A continuación se explica lo que entendemos por planeación de trayectorias para manipuladores, se describe el Sistema Controlador de Trayectorias (SCT), se explica el funcionamiento del Sistema Graficador de Trayectorias (SGT) y, finalmente, se presenta la integración de ambos sistemas en una sección de simulaciones.

3.2 PLANEACION DE TRAYECTORIAS PARA MANIPULADORES.

La planificación de trayectorias para robots fijos involucra las siguientes etapas:

A) - Representación del medio ambiente

B) - Control de la trayectoria:

- Generación de trayectoria
- Evitación de obstáculos

Estas etapas se encuentran interrelacionadas entre si de tal manera que es necesario tener una representación del medio ambiente circundante, de un conjunto de algoritmos que permitan detectar obstáculos que se encuentran obstruyendo el traslado continuo del robot desde su posición inicial a su posición final, asi como la generación de trayectorias para poder realizar el traslado eficiente.

A continuación daremos una subdivisiòn de la etapa de control de trayectoria tabulados en la siguiente tabla.

GENERACION DE TRAYECTORIA CON DETECCCION DE OBSTACULOS EN LINEA	GENERACION DE TRAYECTORIA CON DETECCION DE OBSTACULOS.	SEGUIMIENTO DE TRAYECTORIA SIN DETECCION DE OBSTACULOS

tabla 3.1

3.2.1 GENERACION DE TRAYECTORIAS CON DETECCION DE OBSTACULOS EN LINEA

Comprende la generación de trayectorias que va desde una posición inicial de traslado Po hasta una posición final PF, la cuál se realiza en linea con el traslado, tomando en consideracion el medio ambiente y permitiendo su replaneaciòn en caso necesario. Este tipo de control se utiliza en robots móviles en los que se implementa una cámara de T.V para detectar los objetos circundantes en la posición actual del robot (fig # 3.1). En el capitulo 4 se da una explicación mas detallada de los distintos mètodos para la generación de trayectorias en robots móviles.

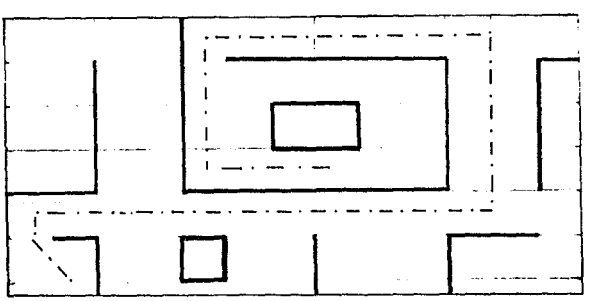


fig # 3.1.

3.2.2 GENERACION DE TRAYECTORIAS CON DETECCION DE OBSTACULOS

Consiste en la generación de trayectorias para el traslado de objetos en un medio ambiente no estructurado, para su ejecución posterior; como ejemplo, tenemos un manipulador fijado a una mesa de trabajo con un sensor (una cámara de T.V colocada en la parte superior de la mesa de trabajo) y en la que es posible planear el traslado utilizando la información proporcionada por la cámara de las posiciones actuales de los objetos. El sistema SPTT cae dentro de esta clasificación (fig # 3.2).

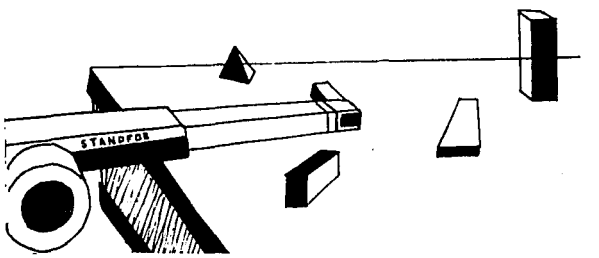


fig # 3.2.

3.2.3 SEGUIMIENTO DE TRAYECTORIAS SIN DETECCION DE OBSTACULOS

Consiste en el seguimiento de trayectorias para el traslado de objetos, sin considerar al medio ambiente. Como ejemplo tenemos los robots que no poseen capacidades sensoriales ni de algoritmos que procesan este tipo de información y que por tanto, efectuan el traslado de objetos a "ciegas"; actualmente, existen fàbricas, como la de la Industria Automotriz, en las que se almacena en memoria de computadora la trayectoria deseada utilizando un "Joystick", ejecutandose esta trayectoria eficientemente pero considerando que ningun objeto o persona entrarà dentro de su región de trabajo (fig # 3.3).

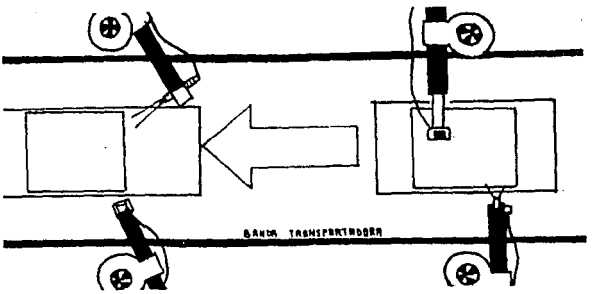


fig # 3.3.

La planificación de trayectorias constituye un tema abierto de gran interès dentro de las aplicaciones del campo de la robótica. Este interès deriva tanto del problema mismo de la generación de trayectorias para el traslado de objetos, como de la interrelación que existe con problemas tales como la programación automàtica de robots o bien la generación de estrategias de navegación para robots móviles. La complejidad que se han encontrado hasta la fecha para la solución de este problema es exponencial respecto al numero de grados de libertad del movimiento. Esto ha sido la causa de que hasta la fecha solo se hayan abordado particularizaciones del problema de planeación de trayectorias.

La Planeación de trayectorias se divide para su estudio en representación del medio ambiente y generación de trayectorias. Dentro de los esfuerzos para abordar el problema de la representación del medio ambiente, estan los trabajos en que se aborda el problema en el espacio de estados en el que el móvil se ve reducido a un punto (Udupa, 1983; Lozano Pèrez,1984). Estos autores (Udupa, 1983; Lozano Pèrez, 1984), plantean la construcción del espacio de estados para la representación del móvil y de los obstáculos; mientras Udupa(1983), considera esta conversión punto a punto, otros como Lozano Pèrez(1984), efectuan una serie de hipótesis restrictivas para su posible representación analítica.

El algoritmo diseñado por Udupa(1983), utiliza para el manipulador Stanford dos regiones de trabajo; la primera para las dos junturas de translación y la segunda para las de agarre. Además de ello, utiliza una función recursiva que halla una trayectoria lineal segura para trasladar objetos eludiendo obstáculos; finalmente, utiliza otra función recursiva y genera otra trayectoria lineal para el pos. final.

Lozano Pèrez(1984), desarrollo un algoritmo para un manipulador cartesiano, en el que utilizando el espacio de objetos poliédricos "A" que contiene la configuracion actual de los objetos dentro de la mesa de trabajo, efectua un compensación de las areas de los objetos que impiden el traslado del objeto N a su posición final, transformándolo en un espacio "B" en el que se obtiene la trayectoria lineal óptima para el libre traslado; para la configuración del espacio libre, utiliza estructuras de arbol en las que los nodos representan celdas y cuyas ligas son junturas entre estas, efectuando una busqueda en este arbol para evaluar el traslado.

Dentro de los mètodos para le generación geomètrica de las trayectorias, comunmente se han utilizado segmentos lineales dispuestos de tal forma que permiten el traslado, utilizando para la posición inicial un segmento 1 con un angulo de inclinación 1, continuando la trayectoria con otro segmento 2 con otro angulo de inclinacion 2, y asi sucesivamente hasta llegar al punto final deseado (fig # 3.4).

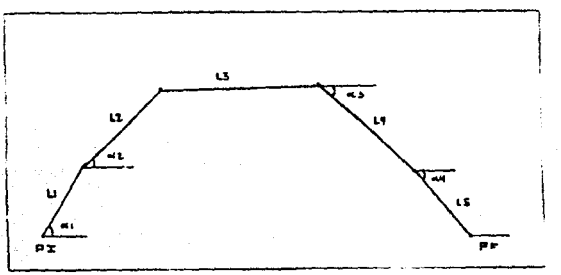


fig # 3.4.

3.3 CICLO ACCION-PERCEPCION.

El ciclo acciòn-percepciòn (Nlisser, 1976), representa, para cualquier tipo de sistema retroalimentado, sea vivo o no, el esquema de representación con el cuál es posible efectuar una acción determinada en función de cambios en el medio ambiente con el que se interactua.

Para el caso, el sistema SPTT, la acción consiste en el traslado de posición de la pinza del manipulador, con o sin el objeto, basado en la información visual de la cámara de filmación utilizada para captar la información del medio ambiente en el que el manipulador interactua. Este ciclo se subdivide en las siguientes fases (fig # 3.5).

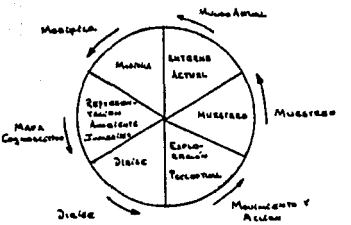


fig # 3.5.

MUESTREAR.-El muestreo es la fase consistente en la toma de datos D1, D2, D3,...DN de un Mundo Actual, dando como resultado una representación interna del entorno actual; para la realización del muestreo, se puede utilizar una cámara de T.V fijada a la parte superior de la mesa de trabajo donde está ubicado el manipulador.

MODIFICAR.- A continuación, se realiza una modificación, en base al entorno actual del MAPA COGNOSCITIVO que consiste en la representación interna del ambiente inmediato; para esta representación,se utilizaron las coordenadas de los vèrtices, representadas como coordenadas homogèneas.

DIRIGIR.- Teniendo ya una representación actualizada del ambiente inmediato, es posible realizar la fase de DIRECCION (desarrollada en el sistema SCT consistente en la generacion de trayectorias eludiendo obstáculos), que derivara en la acción mas adecuada (MOVIMIENTO Y ACCION de las junturas del manipulador), para interactuar con la situación ambiental existente. Este proceso se mantiene en forma ciclica hasta la consecución de una meta especifica (o una tarea predefinida), manteniendose de esta manera las restricciones del manipulador con su medio ambiente dentro del contexto del Ciclo Acción Percepción. Se continua nuevamente con la fase de muestreo con la finalidad de continuar con el CICLO ACCION-PERCEPCION.

3.4 DESCRIPCION DEL SISTEMA PLANIFICADOR DE TRAYECTORIAS TRIDIMENSIONALES (SPTT).

El sistema global SPTT tiene como entradas la definición de las caracteristicas del medio ambiente de los objetos tridimensionales ubicados en posiciones seleccionadas arbitrarias y las posiciones donde seran trasladados los objetos a su posición final, y como salida, el despliegue gráfico en computadora del robot manipulador Standford ejecutando en tiempo real el traslado de los objetos almacenados a sus posiciones finales indicadas (fig # 3.6).

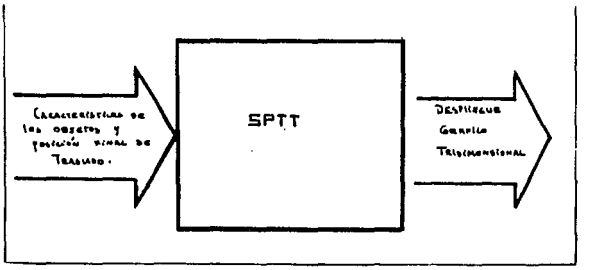


fig # 3.6.

Para la creación del sistema SPTT, se tomó en consideración la metodologia de división de sistemas en subsistemas y considerando que el sistema que necesitabamos desarrollar tenia una división muy precisa: el sistema de simulación gráfica tridimensional por computadora y el sistema generador de trayectorias, se opto por dividir el sistema SPTT en dos subsistemas denominados Sistema Controlador de Trayectorias (SCT) y Sistema Graficador de Trayectorias (SGT), (ver fig # 3.7).

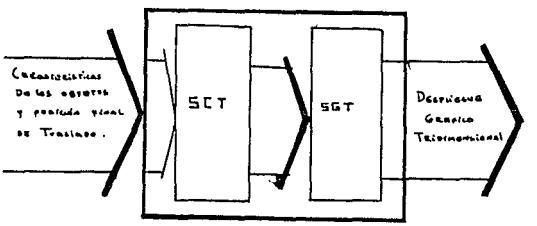


fig # 3.7.

3.5 DESCRIPCION DEL SISTEMA CONTROLADOR DE TRAYECTORIAS (SCT)

Nuestro próposito es desarrollar un modelo que permita el traslado de algun objeto desde su posición inicial a una final, interactuando con el medio ambiente y eludiendo los posibles obstáculos fijos que se le presenten. A continuación, se describen las caracteristicas del sistema Controlador de Trayectorias (SCT).

3.5.1 CARACTERISTICAS DEL SISTEMA SCT.

El sistema SCT, es un planificador de trayectorias para interactuar con un medio ambiente poco estructurado. Este sistema, define matemáticamente el conjunto de posiciones P1,P2,P3,.....PN que representan la trayectoria a seguir por el manipulador durante el traslado de su efector final ("mano"), sujetando o no un objeto, desde su posición actual P0 a la posición deseada PF, en función de las caracteristicas paramètricas actuales de su medio ambiente.

El sistema se desarrollo utilizando la idea de Lozano Perez, de utilizar un modelo fisico analitico del medio ambiente, pero considerando un algoritmo propio para la estimación de los parámetros de los obstáculos que en ese momento impiden efectuar el traslado, ver mas abajo la explicación de esta etapa denominada REGIONALIZADORA. Esto permite efectuar un traslado con evitación de obstáculos segura y eficiente lo cual da al sistema un alto margen de seguridad para la industria.

Asimismo, el sistema considera la idea de generación de trayectorias geométricas basadas en ecuaciones algebraicas, para nuestro caso particular, segmentos de recta y parábolas, que pueden ser modificables o adicionables, con el objeto de tener diversas posibilidades de trayectoria al trasladar los objetos en situaciones en las que por impedimentos del medio ambiente, un tipo de trayectoria especifica no puede ser utilizado y es necesario generar otras trayectorias.

En adición, el sistema se implementó en el lenguaje de programación PASCAL, utilizando las normas de programación estructurada, una representación modular de los algoritmos permitiendo su fácil manipulación, o interpretación.

El sistema es altamente modificable, permitiendo insertar nuevos bloques con distintas caracteristicas geométricas de trayectoria dentro del programa de manera sencilla.

3.5.2 ORGANIZACION SCT.

El sistema consiste de las siguientes etapas:

ETAPA DE LECTURA
ETAPA DE REGIONALIZACION
ETAPA DE GENERACION DE TRAYECTORIA

A continuación damos una representación gráfica de estas (fig # 3.8).

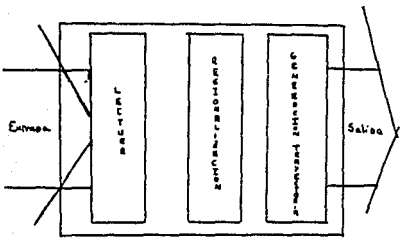


fig # 3.8.

Etapa: LECTURA.

Se encarga de la captura de datos que representa la simulación de la camara de T.V de las coordenadas de los objetos en su posición actual dentro de la mesa de trabajo, asi como la posición final a trasladarlos dentro de ésta, (ver fig #3.9).

Se divide en los siguientes módulos:

- LEE OBJETOS
- LEE POSICIONES

Módulo: LEE_OBJETOS.

Toma como datos de entrada las coordenadas dadas por el usuario de los vértices de cada objeto i, almacenándose secuencialmente en el archivo de objetos denominado object.dat, este archivo sera utilizado para el despliegue gráfico del modulo SGT.

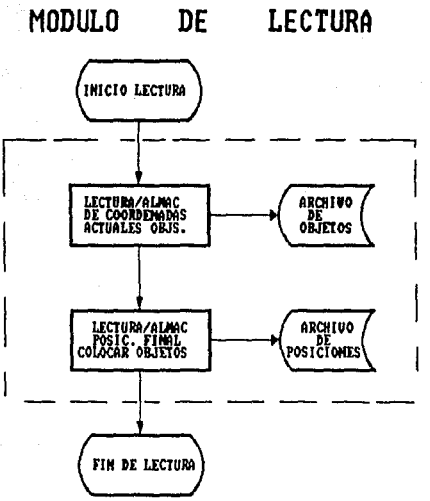

MODULO DE LECTURA
INICIO LECTURA
LECTURA/ALMAC DE COORDENADAS ACTUALES OBJS.
ARCHIVO DE OBJETOS
LECTURA/ALMAC POSIC. FINAL COLOCAR OBJETOS
ARCHIVO DE POSICIONES
FIN DE LECTURA

FIGURA 3.9

MODULO REGIONALIZADOR

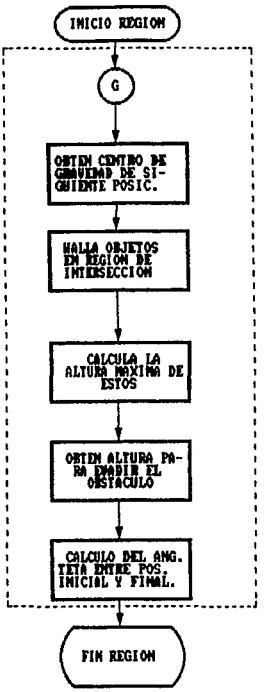


FIGURA 3.10

Módulo: LEE_POSICIONES.

Toma como entrada los datos dados por el usuario de las posición (x, y, z) y ángulo de inclinación beta que indican la posición a trasladar el objeto i, almacenandose en el archivo POS.DAT y sera utilizado para el despliegue gráfico del módulo SCT.

Etapa: REGIONALIZACION.

Para poder llevar a cabo el adecuado traslado de objetos desde una posición inicial a una final eludiendo obstáculos, se hace necesario desarrollar una fase que permita generar una región adecuada de trabajo en la que sea posible realizarlo, este módulo encuentra la región de trabajo para los dos casos siguientes:

- Cuando se desea trasladar a la pinza del manipulador desde su posición actual hasta tomar el objeto.

- Cuando se desea trasladar a la pinza del manipulador ya con el objeto tomado hasta su posición final. (Ver figura # 3.10).

Esta etapa se subdivide a su vez en los siguientes módulos:

- UBICA_ALTURA
- COORDENAD
- BUSQUEDA_OBJETO
- ALTURA_MAX
- COMPARA_STACKS

Módulo: UBICA_ALTURA.

Para realizar el reconocimiento interno de la figura, se toma de entrada al objeto N a trasladar, se utilizan desigualdades que permiten encontrar la ubicación del centro de gravedad del objeto n respecto a la posición final de traslado en la que tenemos las siguientes posibilidades (fig # 3.11). Este módulo se encarga de obtener a la salida la ubicación especifica para el medio ambiente actual de los objetos.

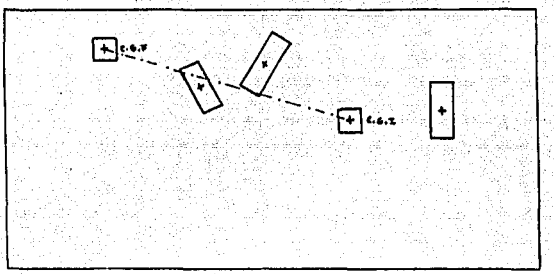


fig # 3.11.

Módulo: COORDENADAS.

Tomando de entrada alguna de estas posibilidades, efectua la suma de la longitud del area del objeto N a las posiciones actuales de los centros de gravedad con el objeto de tener a la salida las coordenadas de los cuatro puntos generando la región de trabajo (fig # 3.12).

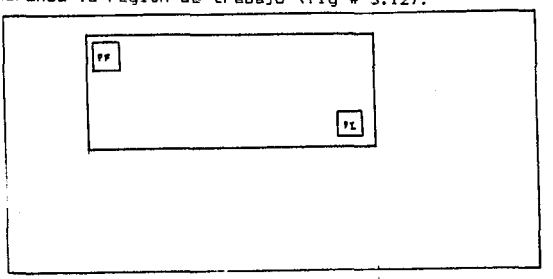


fig # 3.12.

A continuación, y en base A LAS COORDENADAS DE LOS CUATRO PUNTOS de esta región de trabajo cuadrangular calculada, se efectua el cálculo de las pendientes m1 y m2 de la ordenada1 y ordenada2 con el objeto de obtener las rectas que me definen la región de trabajo 1 para la recta 1 y la región de trabajo 2 para la recta 2 (fig # 3.13).

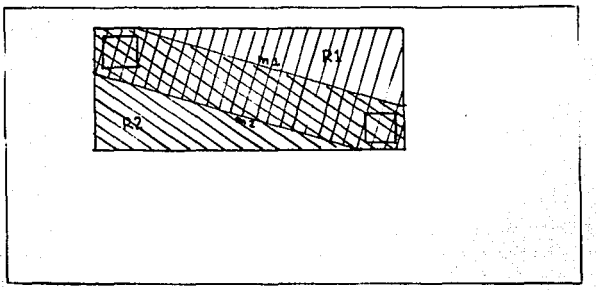


fig # 3.13.

Módulo: BUSQUEDA_OBJETO:

Teniendo de entrada estas regiones, el siguiente paso consiste en efectuar la busqueda de los objetos que se hallan contenidos en la intersección de ambas regiones, para lo cuál, se considera un procedimiento encargado de efectuar la busqueda en cualquiera de las 2 areas posibles, con las posibilidades siguientes (fig # 3.14).

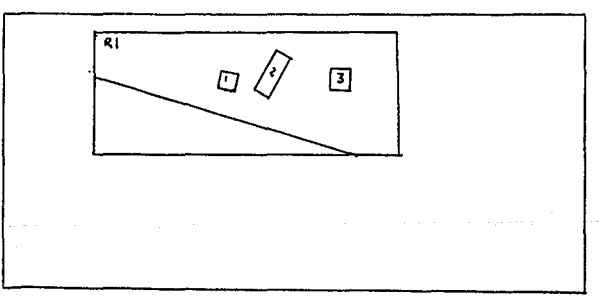


fig # 3.14.

O con las siguientes posiblilidades teniendo como datos la region 2 (fig # 3.15).

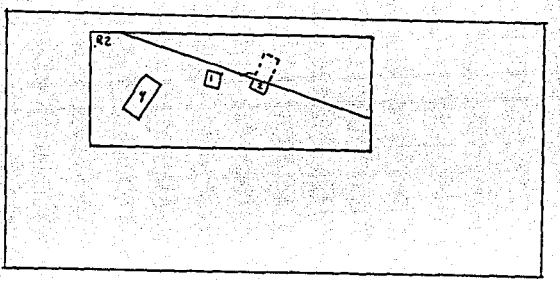


fig # 3.15.

Obteniendose un conjunto de objetos para la región 1 asi como un conjunto de objetos para la region 2, con èsto, finalmente efectuamos la busqueda del conjunto de objetos que se encuentran en ambas regiones. Almacenandose a la salida en un vector de objetos.

Mòdulo: ALTURA_MAX.

Teniendo como entrada al conjunto de objetos contenidos en la region1 y 2, se efectua la busqueda de los objetos que se hallan en ambas regiones almacenándose a la salida en un vector de intersección (fig # 3.16).

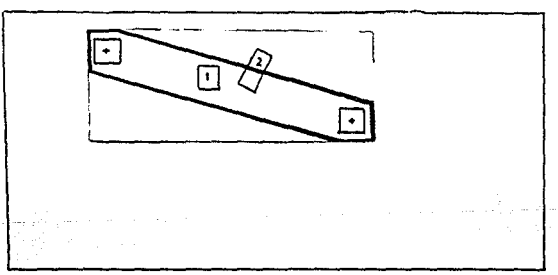


fig # 3.16.

Módulo: COMPARA ALTURAS

Finalmente, se toman de entrada las alturas de ambos regiones y se calcula a la salida la altura máxima unica y a la que se adiciona una constante de seguridad con el objeto de liberar tanto al obstáculo de mayor altura como a los demás (fig # 3.17).

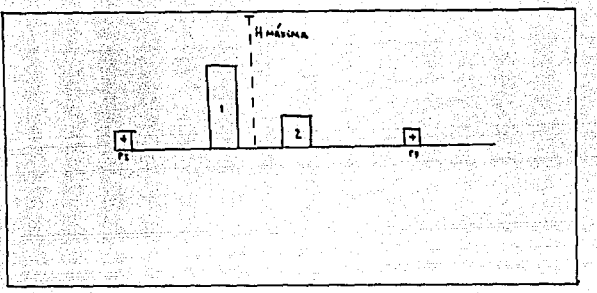


fig # 3.17.

El hecho de haber realizado de esta forma el cálculo de la región de trabajo, se debe a que con èsta región de intersección podemos encontrar únicamente a los objetos que obstaculizan el libre traslado de la pinza del manipulador a su posición final. Asimismo, si posteriormente se desea una trayectoria parabólica horizontal o con algun angulo de inclinación deseado, se puede tomar cualquier ya sea la 1 o 2 directamente.

Módulo: TRANS_ANG_BASE.

Para hacer el traslado de el objeto N desde su posición inicial hasta su posición final, se hace necesaria la utilización de un procedimiento para el cálculo del angulo teta que forman, la posición inicial del objeto respecto del eje X hasta efectuar el traslado a su posición final. Esto se realiza tomando como entrada las posiciones del centro de gravedad inicial y final de todas las posibilidades de colocación de los objetos en la mesa de trabajo y calculando a la salida la inclinación teta en grados (fig # 3.18).

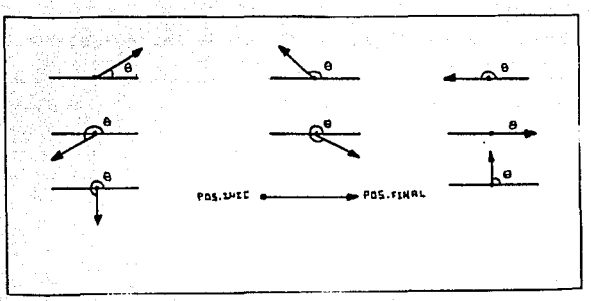


fig # 3.18.

Etapa: GENERACION DE TRAYECTORIAS.

El objetivo de esta etapa es la generación de la trayectoria desarrollada, la cual se divide en los siguientes tres segmentos considerados desde la toma del objeto, su traslado, y su colocación:

-trayectoria lineal vertical ascendente
-trayectorias paràbolicas verticales
-trayectoria lineal vertical descendente

(Ver figura # 3.19).

y se desarrolla en los siguientes modulos:

- pos_pinza
- altura_inic_pinza
- tray_no_lin
- reasigna
- desciende

La trayectoria generada por estos módulos se almacenara en el archivo PUNTS.DAT para ser utilizada por el modulo SGT de despliegue gráfico.

Mòdulo: POS_PINZA.

Es utilizado solamente al inicio de correr el sistema SCT, tomando como entrada la posición definida de antemano de la pinza del manipulador y dando a la salida la transformación homogènea Tó que representa la posición inicial de la pinza del manipulador.

MODULO GENERADOR DE TRAYECTORIAS

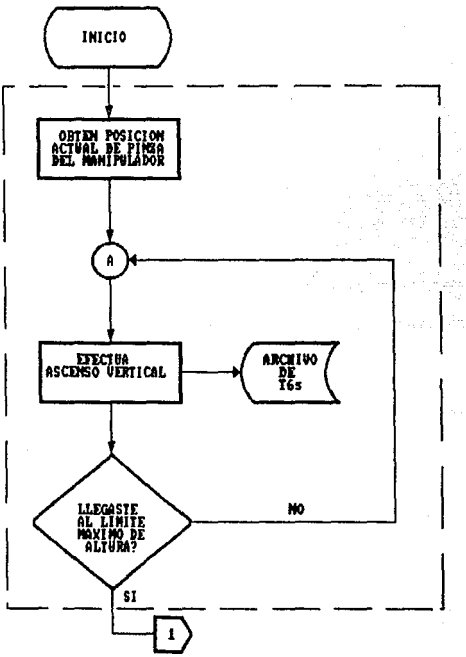


FIGURA 3.19

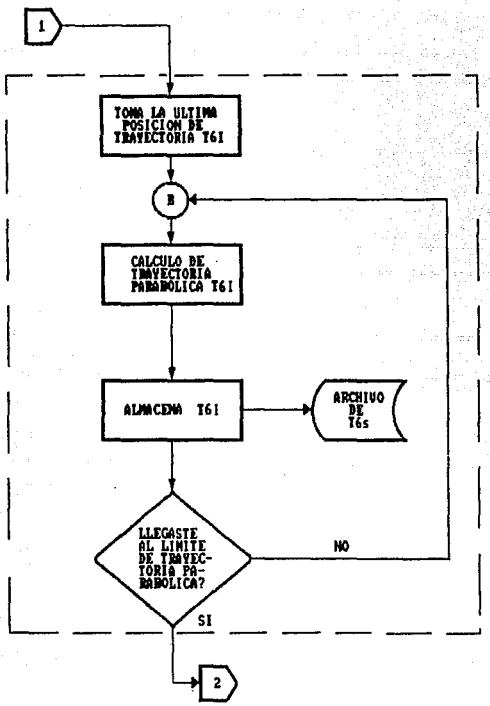


FIGURA 3.19 (CONTINUACION)

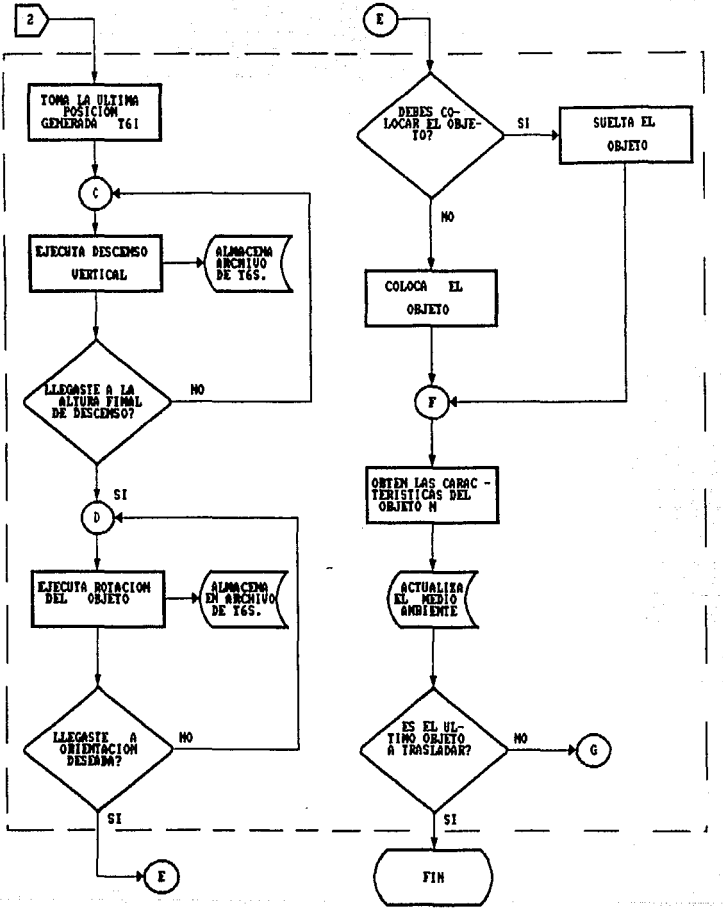


FIGURA 3.19 (CONTINUACION)

Módulo: ALTURA INICIAL DE LA PINZA.

La finalidad de este módulo, es efectuar el ascenso desde la posición inicial actual del objeto H, hasta la posicion final H'=H+a donde a, es un parámetro encontrado en la fase de regionalizacion que representa la altura máxima a evaluarse para evadir los obstáculos que impiden el libre traslado del objeto. Para esto, se toma de entrada la posición t6 ultima en la que se encuentra la pinza del manipulador, se efectua paulatinamente un incremento constante en la variable z de esta matriz, generando a la salida la matriz t6i, que contiene la posición actual del objeto H, hasta llegar a la posición final T6z=H' (fig # 3.20).

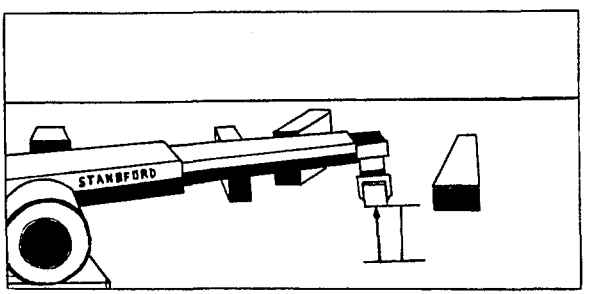


fig # 3.20.

Módulo: TRAYECTORIA PARABOLICA.

Teniendo como entrada la ultima posición de la pinza del manipulador T6i, se continua con el cálculo del segmento de trayectoria parabólica, se comiensa efectuando el cálculo de los parámetros de la parábola, considerando el valor de la altura H del módulo regionalizador, obteniendo los otros parámetros p, k, de la parábola, a continuacion, tomando un eje coordenado X-Z, Se valua la parábola en Z para un valor x inicial almacenándose en una matriz de transformación A (fig # 3.21).

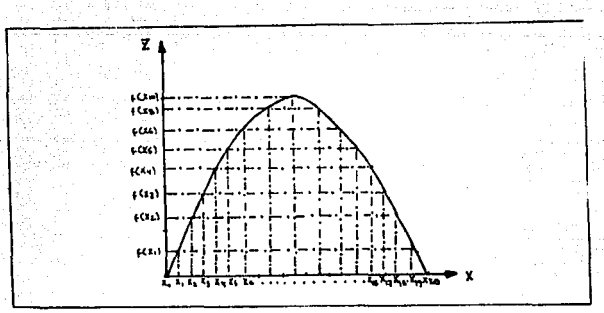


fig # 3.21.

Módulo: ROT_EJES.

Tomando de entrada la posición X del módulo anterior, se efectua el cálculo de la orientación de la pinza del manipulador al trasladarse siguiendo la proyección del segmento de recta x a x+inc, donde inc es una constante almacenandose en una matriz B a la salida (fig # 3.22).

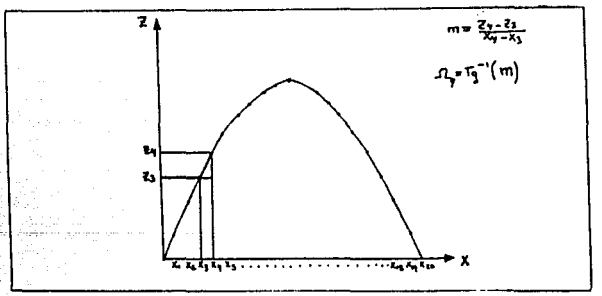


fig # 3.22.

Módulo: TRANSF_BASE

Se obtiene la matriz de transformación C referida a coordenadas base, lo cual se hace con el objeto de obtener la posición respecto del sistema coordenada de referencia base XYZ. Para ésto, se toma como entrada el angulo TETA respecto al eje Z (fig # 3.23).

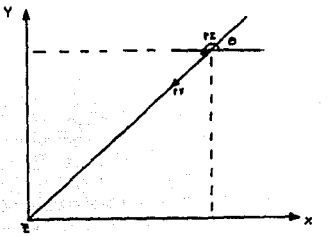


fig # 3.23.

A continuación, se efectua el producto de las transformaciones anteriores, translación por la de rotación, obteniendose a la salida la posición final de la pinza del manipulador como transformación Homogènea Tói tanto en posición como en orientación respecto a coordenadas Base (fig # 3.24).

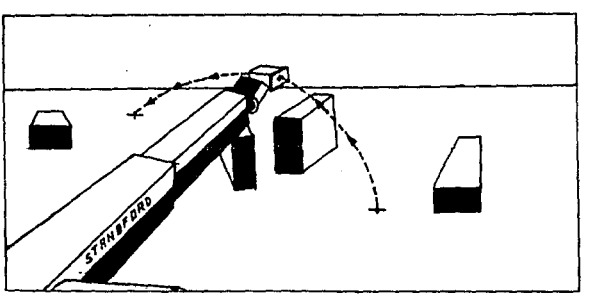


fig # 3.24.

Nuevamente, se calcula en x+inc de la pinza del manipulador para continuar con el algoritmo anterior para el calculo de la matriz T6 hasta que se llega al valor final de x+inc en el que el algoritmo se detiene.

Módulo: DESCIENDE.

Para llevar a cabo el descenso vertical, se consideran de entrada, la última posición de la trayectoria parabólica generada T6i, y la posición final a la cuál se debe colocar el objeto, calculada con anterioridad tanto en posición como en orientación. A continuación, se efectua un decremento constante en Z de la matriz de transformación T6 generando a la salida la transformación homogènea T6i hasta llegar a la posición deseada (fig # 3.25).

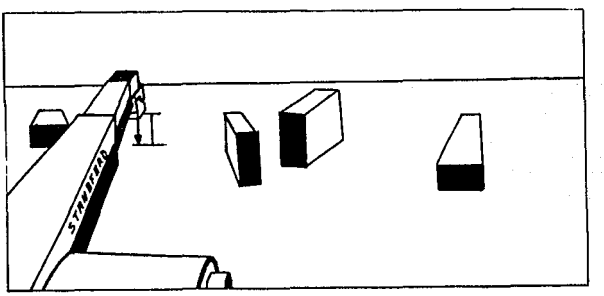


fig # 3.25.

Finalmente, para posicionar el manipulador con la orientación final deseada, se efectua un cálculo en rotación del ángulo calculado en el que se encuentra la pinza del manipulador actualmente alfa, menos un decremento angular beta, de valor pequeño, obteniendose un valor de la matriz de transformación homogènea T6i, a continuación se realiza una comparación entre esta orientación y la orientación final deseada, si no es igual, se continua con las diferencias hasta llegar al ángulo deseado Sigma que contiene la orientación deseada, efectuando la toma o colocación del objeto dependiendo de una bandera (fig # 3.26).

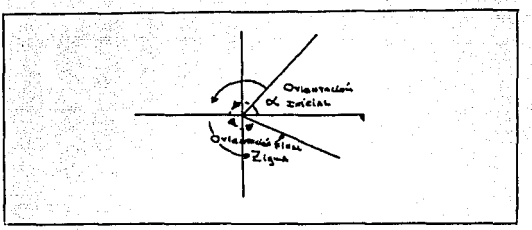


fig # 3.26.

Mòdulo: TOMA_SIG_POSIC.

Teniendo cómo entrada la posición i siguiente del archivo de objetos secuencial, se deben obtener las caracteristicas del siguiente objeto a trasladar, que contiene la posición Pn a la que se deberá trasladar el objeto, asimismo, actualiza el medio ambiente de los objetos, que ahora contendrán las posiciones de los objetos ya trasladados a su nueva posición. Finalmente, teniendo el nuevo objeto a trasladarse se reinicia con el módulo regionalizador para el cálculo de la nueva región de trabajo y generación de trayectorias, hasta terminar con todos los objetos n a trasladar la tarea especificada por el usuario.

3.6 DESCRIPCION DEL SISTEMA GRAFICADOR DE TRAYECTORIAS (SGT).

Para poder observar el resultado del procesamiento realizado por SCT, se desarrollò un sistema de graficaciòn de trayectorias (SGT), que consiste en desplegar en la pantalla del computador las posiciones y orientaciones del manipulador (en nuestro caso particular, el Stanford), asi como los puntos de la trayectoria planeada y generada por SCT, adicionalmente, nos permite mostrar a travès de letreros una mayor informaciòn para un mejor entendimiento de la diferentes etapas del proceso.

Los despliegues gràficos se realizan utilizando una representaciòn matricial, y subrutinas de un paquete gràfico.

Esto es, para la definiciòn de los movimientos a realizar por los actuadores se usaron matrices de transformaciòn homogèneas para movimientos de rotaciòn, traslaciòn y escala. En cuanto al paquete gràfico, se utilizaron ventanas, definiciòn de un sistema coordenado global, y rutinas apropiadas para efectuar la animaciòn (movimiento del robot).

En el SCT se definen las tareas que va a efectuar el manipulador, y en base a esto, se planea y se genera la trayectoria que debe seguir para realizar estas tareas; mientras que, el sistema SGT lleva a cabo la simulaciòn gràfica de lo realizado por el sistema SCT. De esta forma, tenemos que, el sistema SGT, primero, utiliza la informaciòn generada por SCT, tal como: ubicaciòn de los objetos a manipular, su orientaciòn inicial, posiciòn inicial del manipulador, y las trayectorias a seguir (que se encuentran en las matrices Tèi). Una vez obtenida esta informaciòn, se procede a calcular los cambios en las variables del manipulador, que para el caso del manipulador de Stanford son θ1, θ2, d3, θ4, θ5 y θ6. Estos cambios sirven para definir un determinado movimiento en los actuadores del manipulador (Ver mòdulo CALCULAR VARIABLES DEL MANIPULADOR y fig. # 3.28).

Una vez calculadas las matrices de transformaciòn Ti, las cuales se aplicarán a los eslabones del manipulador (definidos por vertices en coordenadas homogeneas en forma matricial) para obtener su nueva posiciòn, se obtendrán las transformaciones correspondientes a los objetos (definidos por vertices en coordenada homogeneas y de forma matricial). Ya que se obtienen la posiciòn y orientaciòn del manipulador y de los objetos, entonces se procede al desoliegue gràfico tridimensional. Para ello se usò la siguiente funciòn de mapeo :

```
xb = yt - xt
yb = zt - xt ; para   xt <> 0

xb = yt
yb = zt; para  xt = 0
```

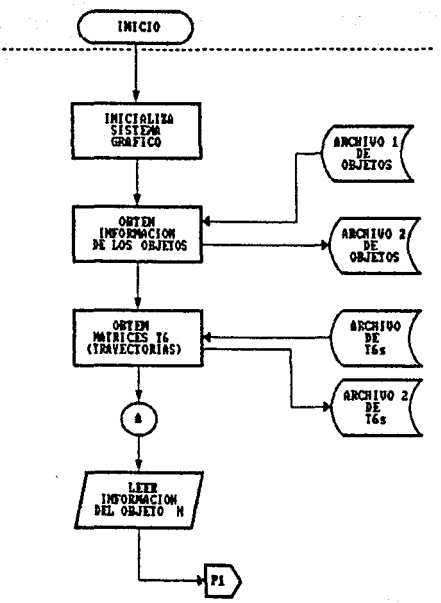

INICIO
INICIALIZA SISTEMA GRAFICO
ARCHIVO 1 DE OBJETOS
OBTEN INFORMACION DE LOS OBJETOS
ARCHIVO 2 DE OBJETOS
OBTEN MATRICES T6 (TRAYECTORIAS)
ARCHIVO DE T6s
ARCHIVO 2 DE T6s
A
LEER INFORMACION DEL OBJETO N
P1

FIGURA 3.27

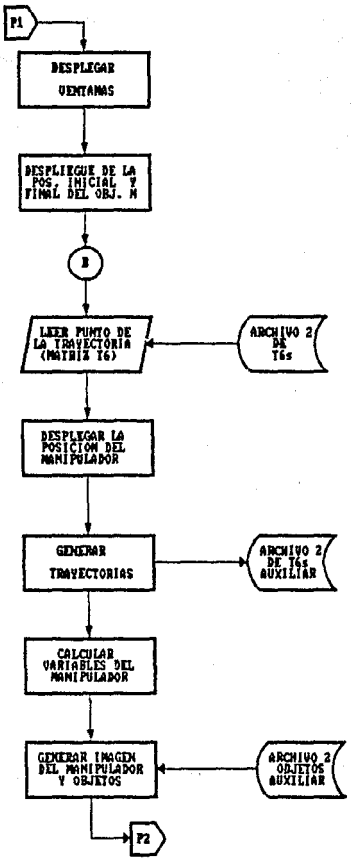


FIGURA 3.28 (CONTINUACION)

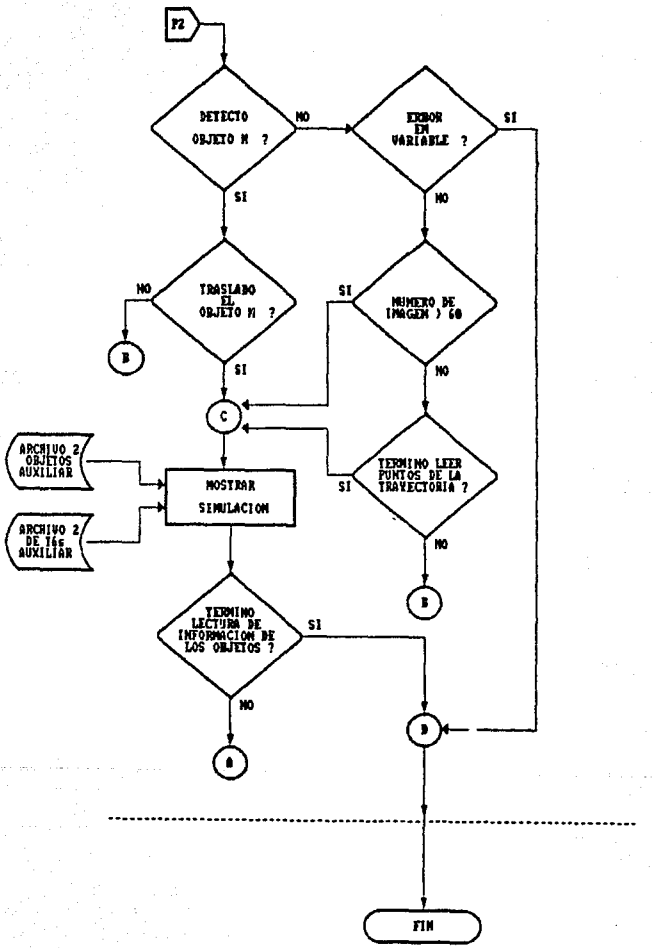


FIGURA 3.25 (CONTINUACION)

donde

xb, yb son las coordenadas bidimensionales
xt, yt, zt son las coordenadas tridimensionales

que permite interpretar en un plano puntos definidos en un ambiente tridimensional. En adición, estos despliegues gráficos se almacenan en memoria. (Ver GENERA IMAGEN DEL MANIPULADOR Y OBJETOS y fig. # 3.28). Este almacenamiento en memoria de las imágenes se hace necesario para dar un efecto de animación, simplemente con desplegar estas imagenes almacenadas a diferentes velocidades. También se grafica la trayectoria que se generó. (Ver módulo MOSTRAR SIMULACION y fig. # 3.29).

3.6.1 Organización de la implantación del SGT.

Funcionalmente, el sistema de graficación tridimensional para un manipulador, desarrollado en esta tèsis, consta de los siguientes pasos :

PASO 1- Definir las ventanas (secciones de la pantalla) en donde se va a desplegar lo siguiente: un sistema coordenado global, con respecto al cual se graficarà el manipulador y los objetos, representación gràfica del manipulador y de los objetos, sus posiciones, indicadores de errores, valores de las variables importantes del manipulador, y mensajes de detección de objetos.
Todo lo anterior, se realiza en el mòdulo Inicializa Sistema Gráfico (ver fig. # 3.27).

Módulo : INICIALIZA SISTEMA GRAFICO.

Este módulo divide la pantalla en once ventanas que nos sirven para visualizar la siguiente información :

- Valor de la posición del manipulador.
- Valor de la posición inicial y final de los objetos que estan en la región de trabajo.
- Letrero para indicar si se detectó algun objeto.
- Leds de error para indicar si alguna variable del manipulador está fuera de rango.
- Valores de las variables del manipulador.
- Imágenes del manipulador y objetos.

Además, define el encabezado para la ventana principal, la cual ocupa toda la pantalla.

También define el sistema de referencia que se va a usar para desplegar la imagen del manipulador y objetos. El sistema que se usará varía de -150 a 150 en cada eje x,y,z.

PASO 2- Leer el archivo de objetos creado en el sistema SCT y crear otro archivo de objetos con la siguiente información : número de objeto, posición inicial y final, vertices en coordenadas homogenas para representar al objeto y transformación homogenea asociada al objeto. La necesidad de crear otro archivo se debe a que el despligue gràfico de los objetos se hara utilizando diez vertices y en el SCT se usaron solo cuatro.
Esta parte se efectua en el módulo Obten Información de los Objetos (ver fig. # 3.27).

Módulo : OBTEN INFORMACION DE LOS OBJETOS.

Dado que el sistema SCT nos da un archivo que contiene información acerca de los objetos, entonces este módulo lo usará para crear otro archivo con información adicional cuya estructura es la siguiente :

- Número de objeto.
- Posición inicial y final.
- Diez vèrtices (en coordenadas homogèneas) para describir un objeto. Dos de estos servirán para situar el centro superior e inferior del objeto.
- Transformación homogènea que se aplicará al objeto.

Este módulo se divide en las siguientes partes:

Obten_pos.

Obtiene la posición inicial y final de los objetos.

Obt_transf_obj.

Cuya entrada es el centro de gravedad de los objetos y nos da la transformación de traslación asociada al objeto.

Obt_vert_obj.

Obtiene diez vèrtices en coordenadas homogèneas a partir de la información obtenida del sistema SCT. Estos vertices nos representan al objeto colocado en el sistema de referencia. Los vèrtices se almacenan en una matriz de numeros reales.

Traspu1.

Efectúa la traspuesta de la matriz que contiene a los vèrtices de un objeto o de la junturas del manipulador.

Mult4.

Realiza la multiplicaciòn de la matriz de transformaciòn homogenea (traslaciòn y/o rotaciòn) por la matriz obtenida en Traspu1. Con esto obtenemos los vertices de cada objeto en alguna posiciòn.

Traspu2.

Obtiene la traspuesta de la matriz de vèrtices para los objetos o para la junturas del manipulador. Con estos ya se tiene la matriz que se mandarà à dibujar en la pantalla.

PASO 3- Leer el archivo de matrices T6, el cual contiene las trayectorias que va a seguir el manipulador, y crear un nuevo archivo de trayectorias que se usara para el despliegue gràfico.
Esto se hace en el mòdulo Obten Matrices T6 (ver fig. # 3.27).

Módulo : OBTEN MATRICES T6 (TRAYECTORIAS).

Usará el archivo de matrices T6 que se generó en la planeación. En otras palabras se obtendrá un archivo que contiene la posición y orientación de la mano del manipulador para varios puntos de la trayectoria planeada.

PASO 4- Posicionarse en el archivo de objetos para leer la informaciòn : el número de objeto, posiciòn inicial y final, vertices del objeto y su transformaciòn homogènea. Con esto se conoce la informaciòn del objeto que se va a manipular.

PASO 5- Mostrar el ambiente gràfico en la pantalla mostrando las secciones, dentro de las cuales se van a desplegar los distintos aspectos de la simulaciòn.
Esto se hace en el mòdulo Desplegar Ventanas (ver fig. # 3.28).

Mòdulo : DESPLEGAR VENTANAS.

Despliega todas las ventanas definidas en el módulo de inicialización.

PASO 6- Mostrar la posiçòn inicial y final del objeto en la secciòn de la pantalla correspondiente.
Esto se realiza en el mòdulo Despliegue de la Posiciòn Inicial y Final del Objeto N (ver fig. # 3.28).

Mòdulo : DESPLIEGUE DE LA POSICION INICIAL Y FINAL DEL OBJETO N.

Despliega en una parte de la pantalla la posiciòn inicial y final del objeto que se va a manipular.

PASO 7- Posicionarse en el archivo de trayectorias (matrices T6) para leer los diferentes puntos de la misma; es decir, la matriz T6. Con esto ya conocemos un punto de la trayectoria que seguirà el manipulador para colocarse en la posiciòn inicial del objeto y trasladarlo hacia una posicòn final, la cual se obtuvo en el paso 4.

PASO 8- Mostrar en la pantalla la posiciòn actual del manipulador que corresponde a la definida en la matriz T6 obtenida en el paso 7.
Esto lo efectua el mòdulo Despliegue de la Posiciòn del Manipulador (ver fig. # 3.28).

Mòdulo : DESPLEGAR LA POSICION DEL MANIPULADOR.

Despliega la posiciòn actual del manipulador, la cual esta contenida en las matrices T6.

PASO 9- Guardar en un archivo auxiliar cada punto de la trayectoria, es decir, la matriz T6 obtenida en el paso 7. Este archivo se usara para desplegar la parte de la simulaciòn gràfica, que consiste en mostrar una serie de puntos que configuran las trayectorias que va a seguir el manipulador al efectuar la tarea.
Esto se lleva a cabo en el mòdulo Generar Trayectorias (ver fig. # 3.28).

Mòdulo : GENERAR TRAYECTORIAS.

Genera un archivo que contendrà las trayectorias que seguirà el manipulador para tomar un objeto y colocarlo en alguna posiciòn. El archivo creado se usa para graficar en pantalla los puntos de las trayectoria.

PASO 10- Calcular , en base a la matriz obtenida en el paso7, los valores de los movimientos que efectuarà el manipulador, que para nuestro caso estàn representados por los cambios en las variables θ1, θ2, d3, θ4, θ5, θ6. Con esto se definen los movimientos en las junturas del manipulador.
Este paso se realiza en el mòdulo Calcular Variables del Manipulador (ver fig. # 3.28).

Mòdulo : CALCULAR VARIABLES DEL MANIPULADOR.

Tiene como propòsito calcular las variables del manipulador para cada matriz T6.

Para el cálculo de los valores de los grados de libertad, que en el caso del manipulador Stanford son θ1, θ2, d3, θ4, θ5, θ6, se usaron las ecuaciones obtenidas anteriormente para la solución de ecuaciones cinemáticas de dicho manipulador. Si vemos las ecuaciones, se nota que se necesita como entrada una matriz T6 que contiene a nx, ny, nz, sx, sy, sz, ax, ay, az, px, py, pz ; elementos que nos definen la posición y orientación de la mano del manipulador.

Para efectuar lo anterior, este módulo se divide en las partes siguientes:

Vminimo.

Para evitar divisiones entre cero, este módulo inicializa los valores de la matriz T6 con un valor muy pequeño.

Variables.

Calcula las variables del manipulador, usando las ecuaciones vistas en el capitulo 2 (ver Cinemática).

Rad_grad.

Si las variables del manipulador son ángulos, este módulo efectúa la conversión de radianes a grados.

Imprime_angs.

Imprime las variables del manipulador.

Checa_ang.

Avisa si alguna variable del manipulador está fuera de un rango preestablecido. En caso de detectar la variable erronea, entonces se prenderá una ventana como si fuera un led de error.

PASO 11- En este paso se efectua lo siguiente : Seleccionar la sección de la pantalla en la cual se va a desplegar el manipulador y objetos.
Inicializar los vértices en coordenadas homogeneas que representan a las junturas del manipulador.
Con los valores calculados en el paso 10, además de parámetros como: di, αi, ai; se obtienen las matrices Ai, y a partir de estas se obtienen las matrices Ti, las cuales se aplicarán a las junturas del manipulador para obtener su posicion actual.
Ya obtenida la posición de las partes del manipulador se lleva a cabo un mapeo de vértices en tres dimensiones a vértices en dos dimensiones. Usando los vértices en dos dimensiones y con ayuda de una rutina para dibujar lineas y\o circulos, se realiza el

despligue gráfico de las junturas que conforman al manipulador.
Ahóra empezamos a trabajar con la informaciòn del archivo de objetos obtenida en el paso 4. Primero se verifica si ya se detectò el objeto, esto es, que la posiciòn actual del manipulador sea aproximadamente igual a la posiciòn final del objeto que se va a manipular. En caso de que se detecte algun objeto entonces se manda un mensaje en una secciòn de la pantalla y se crea otro archivo de objetos con informaciòn actualizada acerca de su posiciòn; en caso contrario, no se actualiza la informacion de los objetos.
Con el archivo actualizado de los objetos se mapean los vèrtices del objeto en tres dimensiones a vèrtices en dos dimensiones,para, utilizando nuevamente la rutina de dibujar lineas, hacer el despliegue de los objetos en la pantalla. Despues de desplegar el manipulador y objetos en una secciòn de la pantalla, se guarda esta informaciòn en memoria con la ayuda de una rutina del paquete gráfico.
Todo esto se lleva a cabo en el mòdulo Generar Imagen del Manipulador y Objetos (ver fig. # 3.28).

Módulo : GENERAR IMAGEN DEL MANIPULADOR Y OBJETOS.

Esta parte tendrá como objetivo crear y dibujar en pantalla al manipulador y los objetos que estàn en su ambiente de trabajo. Este mòdulo se divide en las partes siguientes :

Manip_inic.

Nos da como salida siete matrices que representan a los eslabones del manipulador como vèrtices en coordenadas homogèneas.

Seleccionar la ventana en la cual se desplegará la imagen del manipulador y los objetos.

Mat_Ti.

Obtiene las matrices de eslabòn Ti a partir de las variables del manipulador y de las matrices Ai.

Efectua llamadas a los siguientes mòdulos :

Paramet.

Tiene como entrada a las variables del manipulador y nos da los parámetros : di , αi, ai, θi.

Grad_rad.

Convierte las variables del manipulador a radianes.

Mat_Ai

Entran los parámetros y nos las matrices Ai, las cuales se calculan usando la expresión vista en el capitulo 2 (Cinemática).

Mat_Ti.

Usando las matrices Ai calcula las matrices Ti.

El algoritmo que usa este módulo es el siguiente :

1 - Hacer i = 0.
2 - Iniciar con T0 (matriz identidad).
3 - Si i>5 ir al paso 9.
4 - Hacer i=i+1.
5 - Calcular di, αi, ai, θi .
6 - Obtener matriz Ai (la cual depende de di, αi, ai, θi) que se vio en la parte de cinemática.
7 - Hacer Ti = Ai Ti-1 .
8 - Regresar al punto 3.
9 - FIN.

Graf_manip.

Dibuja los eslabones del manipulador llamando a los siguientes módulos :

Transpu1.

Mult2.

Multiplica la tranformación Ti por los vértices del eslabón i, para que se obtengan los nuevos vértices del eslabón i.

Transpu2.

Tres_dos

Mapea los nuevos vértices de cada eslabón i a vértices en dos dimensiones.
La función de mapeo se reliza de la siguiente forma:

Si coordenada x(tridimensional) es <> 0 entonces
coordenada x(bidimensinal) =
coordenada y(tridimensional)
- coordenada x(tridimensional)

coordenada y(bidimensinal) =
coordenada z(tridimensional)
- coordenada x(tridimensional)

sino
coord. x(bidimensional) = coord. y(tridimens.)
coord. y(bidimensional) = coord. z(tridimens.)

Dib_cilindro y Dib_prisma.

- Usando los vèrtices en dos dimensiones efectuar el dibujo del manipulador.

Checa_obj.

Verifica si se detectó un objeto y en caso de que sea cierto entonces la matriz de transformación del objeto será la matriz T6. Se actualiza el archivo que contiene la información de los objetos a manipular.

Graf_objs.

Dibuja los objetos en pantalla.
Se divide en estas partes :

Mult4.

Multiplica las transformación del objeto i por los vèrtices de dicho objeto, para que se obtenga la posición y orientación del objeto i.

Tres_dos.

Mapea los vèrtices del objeto i a vèrtices en dos dimensiones.

Dib_prisma.

Realiza el dibujo de los objetos usando los vèrtices en dos dimensiones.

Almac_imag_en_memoria.

Almacena la imagen del manipulador y objetos en memoria.

PASO 12- En este paso se realiza lo siguiente : Si se detectó un objeto y se trasladó hacia su posición final, no hubo error en alguna variable del manipulador, y no se sobrepasò la memoria para almacenar imagenes, o si ya terminó de leer el archivo de trayectorias (matrices T6), entonces ir al paso 13; de otra forma ir al paso 7.

PASO 13- Usando el archivo actualizado de objetos obtenido en el paso 11, y el archivo auxiliar de trayectorias(matrices Tб), realizar el despliegue con efecto de animación ,esto es, consiste en desplegar las imagenes almacenadas en memoria utilizando una rutina del paquete gráfico. Con esto se puede visualizar el manipulador en movimiento para efectuar una tarea y siguiendo trayectorias. Tambièn se muestra la gráfica de las trayectorias que seguirá el manipulador para realizar su tarea.
Esto se realiza en el mòdulo Mostrar Simulaciòn (ver fig. # 3.29).

Mòdulo : MOSTRAR SIMULACION.

Esta parte dará animación (movimiento) al despliegue gráfico.

Para tal propósito se divide en las siguientes partes :

Menu.

Despliega un menù con las siguientes opciones:
esc : salir.
t : dibujar trayectorias.
m : desplegar manipulador en movimiento.
Da como salida la opcion elegida.

Dib_tray.

Despliega la gráfica de la trayectoria que seguirá el manipulador para efectuar su tarea (manipular un objeto). Para tal gráfica se usará el archivo de matrices Tб y el archivo que tiene la información de los objetos.

Dib_manip.

Realiza el efecto de animación para ver el movimiento del manipulador y los objetos que estan en su entorno. Para tal efecto, se desplegarán las imágenes almacenadas en memoria con un cierto retardo entre cada despliegue.

PASO 14- Si ya se terminò de leer toda la informacion del archivo de objetos entonces termina la simulaciòn ; en caso contrario ir al paso 4.

3.6.2 Simulaciones y resultados.

Para mostrar el funcionamiento del sistema planificador de trayectorias tridimensionales (SPTT), en esta sección presentamos varias simulaciones que nos permiten ver algunas de sus caracteristicas mas importantes. Para ello, elegimos una secuencia de imàgenes que representan tres tareas a efectuar por el manipulador.

Iniciamos con una presentación del SGT (ver fig. 3.30), para despues mostrar al manipulador colocado en su posición inicial (0,90,10), ver fig. 3.31.

Ejemplo 1 :

La primer tarea a efectuar por el manipulador es : trasladar un objeto A situado en una posición inicial P0, hacia una posición final Pf, siguiendo la trayectoria planeada por SCT. Las figs.3.32-3.33 muestran al manipulador trasladandose, desde su posición actual, hacia el objeto 1 ubicado en la posición P0 = (90,90,20) y librando obstàculos. En la fig.3.34, se ve el momento en que el manipulador entra en contacto con el objeto 1, es decir, cuando la posición del manipulador es igual a la posición inicial del objeto 1.

Las figs.3.35-3.37, indican la forma en que el manipulador traslada al objeto 1 hacia la posición Pf=(45,45,20); mientras que las figs.3.38-3.39, muestran las graficas de las trayectorias tridimensionales que sigue el manipulador para operar sobre el objeto 1.

FIGURA 3.30

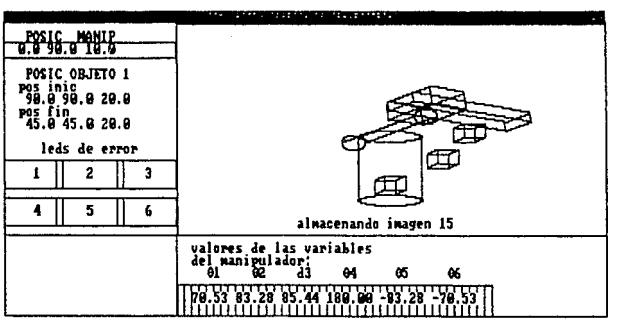


FIGURA 3.31

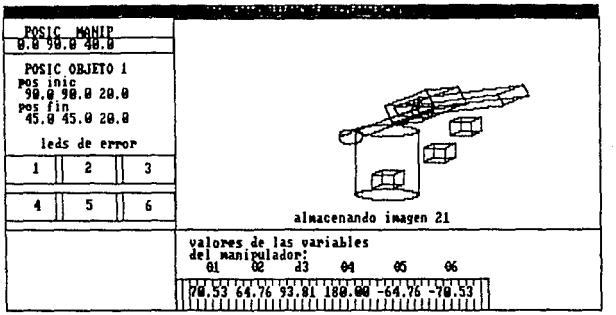


FIGURA 3.32

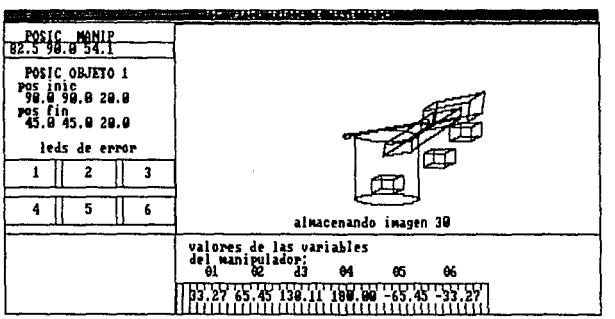


FIGURA 3.33

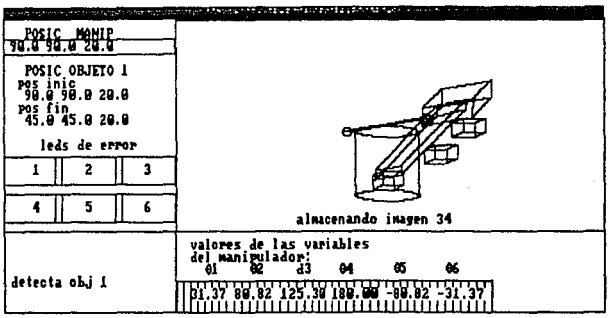


FIGURA 3.34

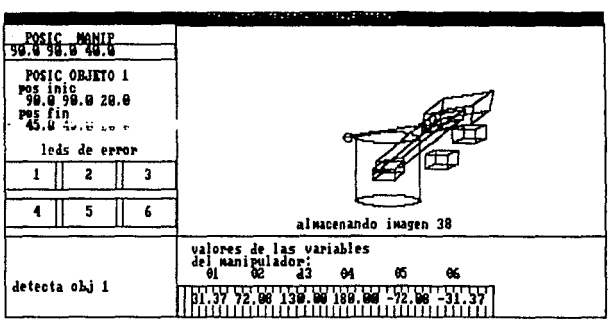


FIGURA 3.35

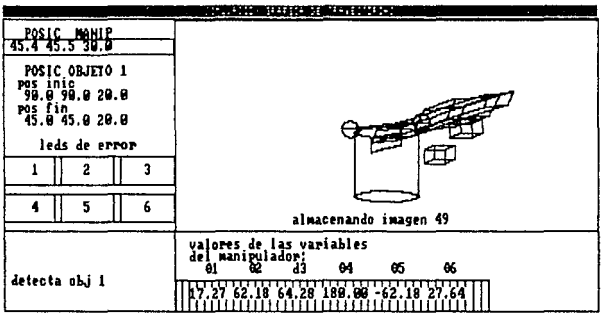


FIGURA 3.36

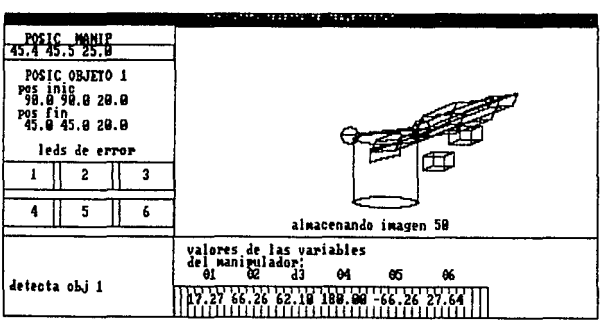


FIGURA 3.37

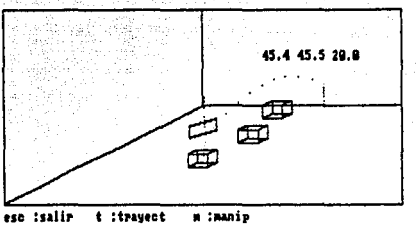


FIGURA 3.38

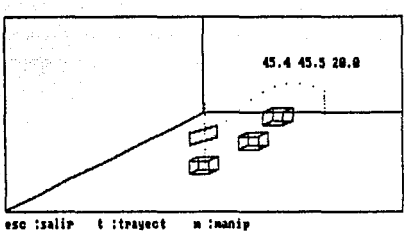


FIGURA 3.39

Ejemplo 2.

La segunda tarea a efectuar por el manipulador es : alcanzar un objeto B situado en una posicón P0' y colocarlo encima de otro objeto C situado en otra posición P1, siguiendo las trayectorias planeadas por SCT. Las figs.3.40-3.44 muestran al manipulador trasladandose, desde su posición actual, hacia el objeto 2 ubicado en la posición P0' = (52.5,90,20) y librando obstàculos. En la fig.3.45, se ve el momento en que el manipulador detecta al objeto 2, es decir, cuando la posición del manipulador es igual a la posición inicial del objeto 2.

En las figs.3.46-3.47, se observa la forma en que el manipulador traslada al objeto 2 hacia la posición P1=(45,45,40), es decir, encima del objeto 1, mientras que las figs.3.48-3.49, muestran las graficas de las trayectorias tridimensionales que sigue el manipulador para operar sobre el objeto 2.

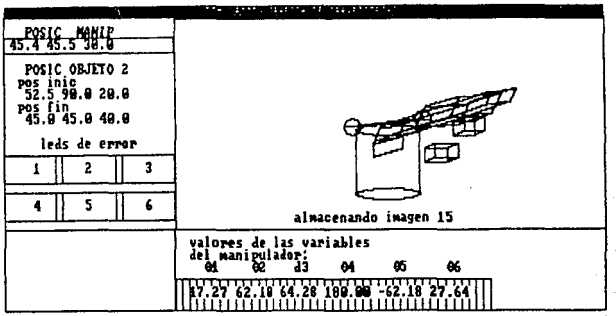


FIGURA 3.40

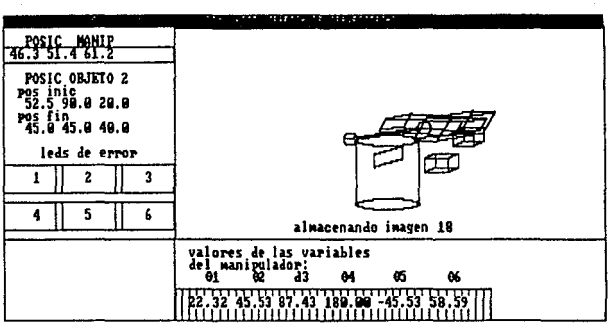


FIGURA 3.41

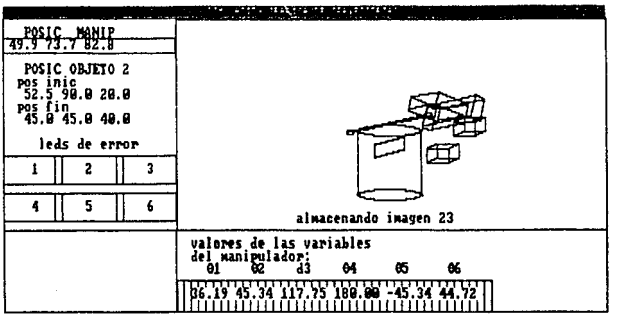


FIGURA 3.42

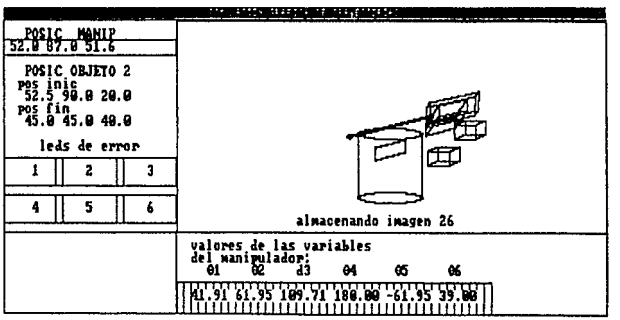


FIGURA 3.43

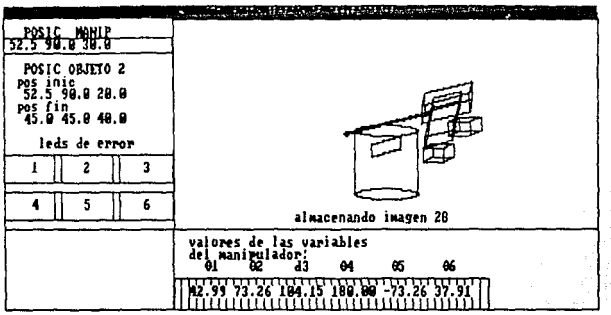


FIGURA 3.44

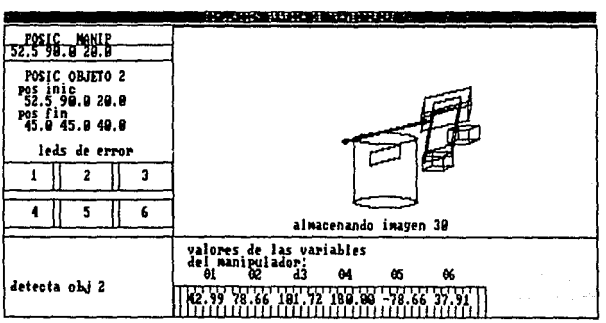


FIGURA 3.45

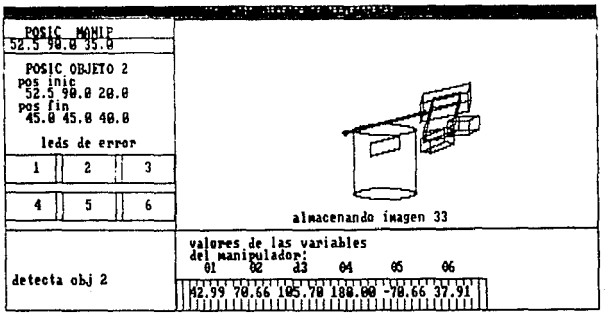


FIGURA 3.46

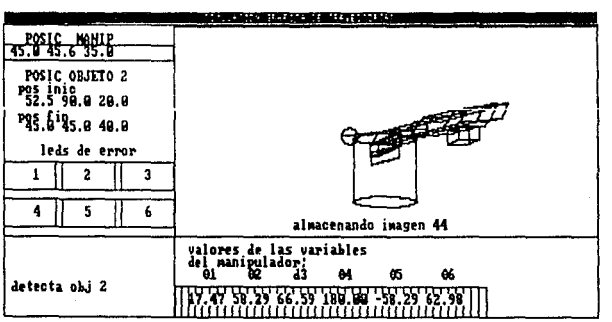


FIGURA 3.47

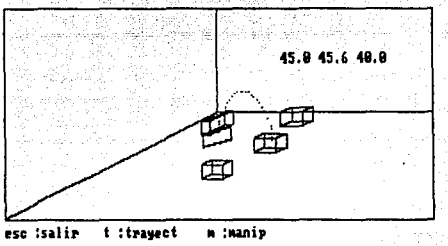


FIGURA 3.48

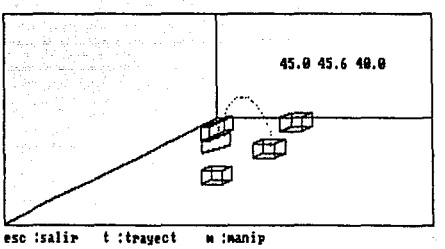


FIGURA 3.49

Ejemplo 3.

La tercer tarea para el manipulador es : trasladarse hacia un objeto situado en una posicòn P0", y colocarlo en otra posiciòn (librando los objetos que estan en las posiciones de los ejemplos 1 y 2) siguiendo las trayectorias planeadas por SCT. Las figs.3.50-3.51 muestran al manipulador trasladandose, desde su posiciòn actual, hacia el objeto 3 ubicado en la posiciòn P0" = (15,71.2,15) y librando obstàculos. En la fig.3.51, se ve el momento en que el manipulador entra en contacto con el objeto 3, es decir, cuando la posiciòn del manipulador es igual a la posiciòn inicial del objeto 3.

Las figs.3.52-3.53, indican la forma en que el manipulador traslada al objeto 3 hacia la posiciòn (71.2,15,30) librando a los objetos 1 y 2.

Las figs.3.54-3.55, muestran las gráficas de las trayectorias tridimensionales que sigue el manipulador para operar sobre el objeto 3.

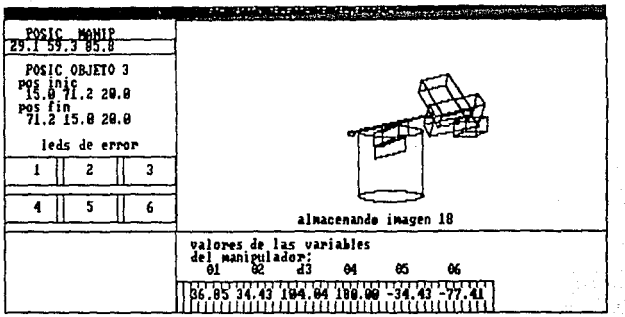


FIGURA 3.50

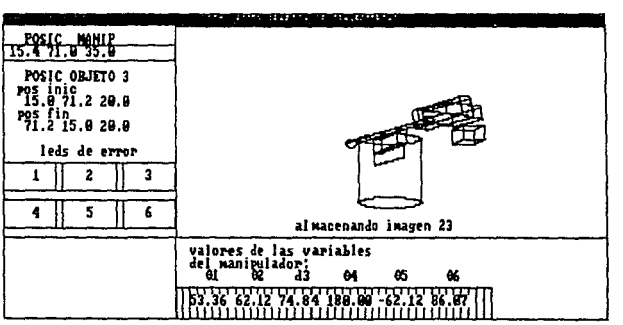


FIGURA 3.51

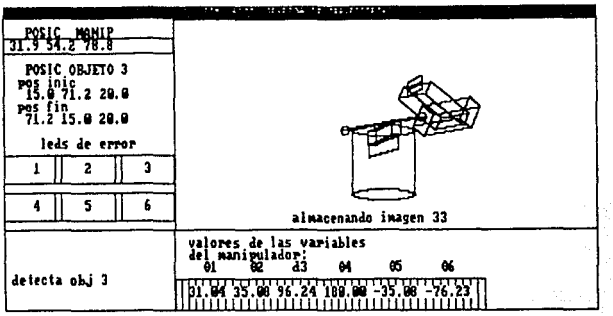


FIGURA 3.52

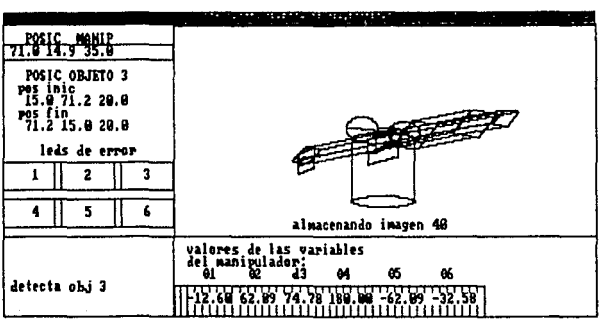


FIGURA 3.53

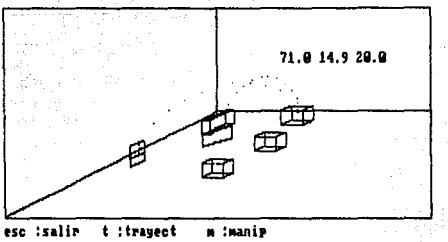


FIGURA 3.54

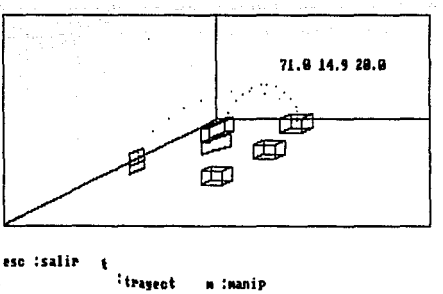


FIGURA 3.55

4 PLANEACION DE TRAYECTORIAS PARA UN ROBOT MOVIL

4.1 INTRODUCCION

La planeación de trayectorias es un problema bien estudiado en varios disciplinas, como por ejemplo: en investigación de operaciones, ingeniería y ciencias de la computación. Muchos algoritmos han sido desarrollados, usando comunmente representaciones gráficas, para determinar una trayectoria en un ambiente completamente conocido.

En el caso de la Robótica, principalmente en robots móviles, el problema de planeación de trayectorias es todavia sujeto de estudio, siendo la razón principal el hecho de que los robots operan en un ambiente parcialmente conocido; las razones de esto son varias: sensores imprecisos, un ambiente dinámico y modelos incompletos del entorno de trabajo, son algunos ejemplos.

Un sistema de planeación de trayectorias para un robot móvil necesariamente debe ser capaz de:

- Mantener un modelo interno del mundo circundante.
- Actualizar ese modelo usando información sensorial.
- Determinar una trayectoria para un objetivo especifico, basada en algún criterio de "mejor" trayectoria.
- Modificar en forma adaptativa esa trayectoria, si es necesario, durante la ejecución; basandose principalmente en nuevos datos sensoriales que entren en conflicto con la trayectoria predefinida.

En este campo, la planeación de trayectorias y los métodos de control para un robot son inseparables. Un plan no puede ser generado por el sistema de planeación de trayectorias y ser ejecutado ciegamente por el robot sin tener realimentación sensorial. Por lo tanto, muchos métodos de planeación de trayectorias han sido desarrollados para enfrentarse con la incertidumbre y no-monotonicidad inherente al problema.

En este capitulo, primero, en la sección 4.2, se describen las estrategias de control que existen para robótica móvil; segundo, en la sección 4.3, se revisan los tipos de representación usados en sistemas de planeación existentes; y tercero, en la sección 4.4, se describe detalladamente el sistema de planeación de trayectorias diseñado en esta tesis, al cual denominamos TURMOV (Trayectorias para Un Robot Móvil).

4.2 ESTRATEGIAS DE CONTROL PARA UN ROBOT MOVIL

Las características importantes del ambiente en que se desenvuelve un robot móvil son las siguientes (ref. 20):

Control impreciso

Los dispositivos sensoriales que pueden utilizarse en robots móviles fácilmente pueden dar datos imprecisos e inexactos. La correspondencia de los cambios en los sensores a los cambios en el ambiente del robot puede ser pobre; por lo tanto la realimentación en el sentido tradicional de teoría de control no es inmediatamente aplicable y sólo se puede usar como guía.

Baja dimensionalidad

El número de grados de libertad de un robót móvil, es menor que el de un robot manipulador. Esto hace que se decremente la complejidad de sus movimientos coordinados.

Error acumulativo

Los errores, si no son corregidos, tienden a incrementarse conforme avanzan los cálculos. Una trayectoria no puede ser calculada y ejecutarse sin verificar y corregir constantemente en base al modelo interno de la posición del robot dentro del entorno.

Modelo incompleto

Los modelos de representación interna para robots móviles son quizá más incompletos que en el caso de los manipuladores. El software diseñado debe cumplir con las restricciones de tiempo real impuestas por la planeación de trayectoria, la replaneación y la evitación de obstáculos.

Incertidumbre en objetos

El ambiente del robot no es siempre como se espera, aun cuando es modelado, ya que no solo se debe considerar la incertidumbre de la posición del robot, sino que también se debe considerar la de la posición de los objetos de su ambiente. Esto se debe a que los objetos pueden haberse movido desde su última observación o a que estan en movimiento relativo al mundo del robot. Esto da un marcado contraste con el ambiente altamente estructurado de un robot manipulador.

Planeación de trayectoria en linea

El robot no debe cerrar sus "ojos" mientras se mueve. El constante monitoreo para evitar colisión es esencial, aunque algunos sistemas existentes violan esta regla (ref. 23). La

planeación de trayectoria debe ser mantenida durante el movimiento del robot, para replanear en el caso de eventos inesperados; ademas de que debe ser llevada a cabo en tiempo real.

Los problemas a resolver por un robot móvil, que involucra un sistema de control, son variados: Planeación de Trayectorias, Evitación de Obstáculos, Reconocimiento de Marcas, Evaluación de su propia posición, Construcción de un Modelo Interno del Mundo, etc. La manera en la cual el control es aplicado y el intercambio de información entre los diferentes módulos constituyen la principal diferencia entre los sistemas de robot móvil desarrollados a la fecha. Existen tres mètodos: el control Monolítico, el Jerárquico y el Distribuido, los cuales se describen a continuación.

4.2.1 Sistemas de control monolítico

El control monolítico es un mètodo bastante simplista para un problema tan complejo como el del control de un robot móvil. No existe realimentación despuès de la generación de trayectoria inicial y casi todo el conocimiento de navegación es por procedimiento. Por lo tanto el sistema monolítico solo es aplicable para sistemas rápidos. Ninguno de los sistemas encontrados en la literatura son de este tipo, pero algunos sistemas comerciales (como el sistema de navegación DRV) usan un tipo de estrategia monolítica. Varios sistemas construidos con capacidades sensoriales son monolíticos, pero no se pretenden para uso general.

Uno de estos sistemas usa un mètodo de enseñansa de trayectoria, común en brazos manipuladores (ref. 22). El robot es puesto en una posición particular de la trayectoria deseada, se graba esta posición y entonces se mueve el robot al proximo punto, repitiendose este proceso hasta que se completa la trayectoria. La trayectoria global nunca es modificada y las condiciones cambiantes son manejadas usando estrategias de evitación de obstáculos.

4.2.2 Sistemas de control jerárquico

La mayoria de sistemas de control que se describen en la literatura son de tipo de jerárquico. En este tipo de sistemas hay varios niveles o jerarquias, en donde a cada nivel le corresponde realizar una cierta función. Cada nivel puede estar constituido por uno o varios módulos y estos ejecutan su función de acuerdo a los resultados de los módulos que estan en un nivel más alto; por lo tanto debe haber sincronización entre los módulos. Cada módulo realiza una función especifica y se comunican uno con otro de una manera predecible y predeterminada.

Un sistema de este tipo es el desarrollado por Hughes Artificial Inteligence (ref. 11). En el nivel más alto esta el PLANEADOR DE MISION el cual establece objetivos, realiza optimización y se encarga de la planeación de tareas. EL PLANEADOR DE LARGO ALCANCE esta en el nivel intermedio y se encarga del desarrollo de una trayectoria global sujeta a las restricciones del planeador de misión. EL PLANEADOR DE TRAYECTORIA LOCAL, el nivel más bajo, es usado para evitar obstaculos y navegación alrededor de obstaculos no modelados en los niveles altos.

4.2.3 Sistemas de control distribuido

En este tipo de sistemas, todos los módulos que lo constituyen estan en un mismo nivel, y todos contribuyen a la realización de una función global, ejecutando parte de esta función. Por lo tanto el control distribuido opera de una manera asincrona. Los módulos se comunican entre si a traves de estructuras de datos globales llamados pizarras.

Un sistema de este tipo, es el HILARE (refs. 9, 10) desarrollado por LAAS en Francia. Tiene una base de datos global que utilizan los módulos de decisión especializada (SDMs). Los SDMs son usados para planeación, navegación y análisis de escenas, mientras que la comunicación entre ellos se logra a través de una pizarra multinivel.

Otro sistema que utiliza control distribuido es el desarrollado en la Universidad Carnegie-Mellon (refs. 18,21), el cual usa fuentes de conocimiento en lugar de SDMs para proveer e interpretar información sobre un mapa local multinivel. Una ventaja de este sistema es la habilidad para facilitar la adopción de nuevas fuentes de conocimiento, módulos de conocimiento y sensores en su estructura.

4.3 ESTUDIO DE REPRESENTACIONES

4.3.1 Caracteristicas de una Buena Representación

Las características de una buena representación para robots móviles deben incluir:

- Eficiencia
- Representación de Incertidumbre
- Múltiples Sistemas de Referencia
- Independencia de Sensores
- Independencia del Vehiculo del Robot
- Soporte de Información Semántica
- Facilidad de Procesamiento Paralelo
- Soporte de Localización

Eficiencia

En el dominio del tiempo real, como la navegación en robótica, es necesario asegurar que los algoritmos usados para procesar los datos contenidos en una representación sean eficientes. Si mucha información se debe calcular, se puede exceder el tiempo de proceso. Esto se resuelve a favor del tiempo con un costo en memoria. Lo anterior también se debe considerar, pues si hay datos que dependen de otros datos en memoria, se puede perder mucho tiempo en actualizaciones.

Representación de Incertidumbre

Los sensores no proveen información exacta del mundo circundante. Ademas la información puede ser contradictoria o errónea. Por lo tanto, el manejador de la representación elegida para expresar el mundo del robot debe tomar en cuenta que algunas cosas no son siempre como se representan y que algunos datos son más confiables que otros.

Multiples Sistemas de Referencia

En el mundo del robot hay objetos cuya posición puede analizarse en relación a la del robot, a la de otros objetos o con respecto a algún sistema absoluto de referencia. Puede haber incertidumbre si se considera la posición absoluta de un obstáculo dentro del mundo. Estas relaciones, así como las que hay en los objetos en si mismos (peso, anchura, fondo), y otras que no dependen de la localización absoluta del objeto, pueden ser más facilmente expresadas utilizando múltiples sistemas de referencia.

Independencia de Sensores

No todos los sensores proveen el mismo tipo de datos y varian mucho en precisión; aún cuando se haga uso de un solo dispositivo (ej. cámara CCD) hay muchas maneras de procesar los mismos datos. A fin de incorporar toda la información posible en la representación, las interfaces deben ser definidas independientes de los sensores, tanto como sea posible.

Independencia de Vehiculo del Robot

Se prefiere el uso de un vehiculo "virtual", debido a que si se considera un nuevo vehiculo, no hay necesidad de rediseñar la representación.

Soporte de Información Semántica

El razonamiento simbólico será esencial si el robot no sólo realiza la tarea de ir del punto A al punto B. Si el razonamiento espacial es implementado en el nivel de

planeación de tareas, algún medio para almacenar conocimiento simbólico debe ser provisto.

Facilidad de Procesamiento Paralelo

Para obtener respuesta en tiempo real de un modo inteligente en un ambiente parcialmente conocido, el paralelismo debe ser explotado.

Soporte de Localización

Ademas de conocer como ir de la posición inicial a la del objetivo, se debe mantener información para determinar la localización del robot.

4.3.2 Tècnicas de Representación

Varios mètodos han sido usados para representar la información necesaria para la planeación de trayectoria. Estos incluyen:

- Mètodos de Espacio Libre
- Gráficas de Vertices
- Mètodos Hibridos Espacio Libre y Gráficas de Vertices
- Campos Potenciales
- Rejilla Regular
- Arbol Cuádrico
- Autómatas
- Multi-Nivel

4.3.2.1 Mètodos de Espacio Libre

En este mètodo hay dos tècnicas: Diagramas de Voronoi y Cilindros Generalizados. En estas tècnicas se representa el espacio libre entre los obstáculos, en lugar de los obstáculos mismos. La tècnica de Diagramas de Voronoi (Ref. 17) consiste en la generación de areas cerradas, cada una conteniendo un obstáculo; estas areas se forman conectando segmentos de linea que pasan por el espacio libre que hay entre los obstáculos (ver fig. 4.1). El diagrama producido se utiliza para calcular una trayectoria de un robot movil . Las desventajas de esta tècnica son que como no se considera la información de los obstáculos, esta no se puede utilizar para planeación de alto nivel, y ademas este tipo de diagramas no pueden ser usados facilmente para planear trayectorias en ambientes cambiantes o para objetos en movimiento.

La tècnica de cilindros generalizados desarrollada por Brooks (Ref 6), consiste en la generación de areas rectangulares o caminos libres a traves del espacio que hay entre los obstáculos. La longitud de estas areas es la minima distancia que hay entre los vertices de dos obstáculos

cercanos y entre las paredes (ver fig. 4.2). Este mètodo permite traslapar los caminos libre producidos y tiene las mismas desventajas que el de Diagramas de Voronoi.

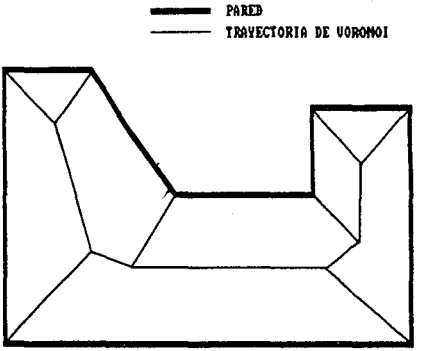


FIGURA 4.1. TECNICA DE DIAGRAMAS DE VORONOI

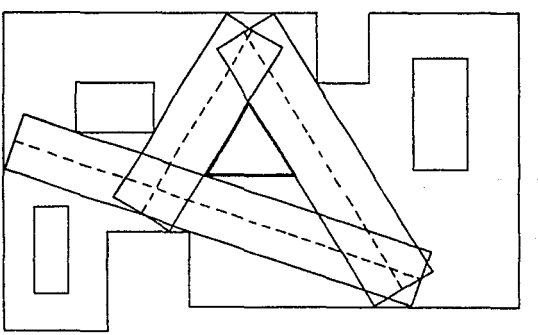


FIGURA 4.2. TECNICA DE CILINDROS GENERALIZADOS

4.3.2.2 Gráficas de Vértices

Este método consiste en modelar los vertices de un obstáculo por medio de un polígono. Los vertices se colocan a una distancia igual al radio de un circulo encerrando al robot (trivial si el robot es circular), más algún margen de tolerancia (ver figura 4.3). Aunque no se requiere suponer que el robot es circular, el hacerlo asi, simplifica mucho el problema computacional. La caracteristica de este método es que no representa explicitamente el espacio que hay entre los obstáculos. Su ventaja es que el robot es tratado como un punto y por lo tanto no ocupa espacio en la planeación de trayectorias. (ver ref. 14 para más información).

Moravec, (ref. 16), usa una representación diferente, donde los obstáculos son modelados como circulos en lugar de polígonos, y las trayectorias se construyen como una serie de tangentes a los obstáculos circulares. La planeación para una trayectoria óptima es mucho más compleja usando esta representación que con la de gráficas de vertices. Debido a esto Moravec desarrolla un algoritmo sub-óptimo que aproxima la trayectoria corta en el espacio circular y produce la trayectoria optima en 95% del tiempo. Este algoritmo rápido fue implementado en el robot ROVER.

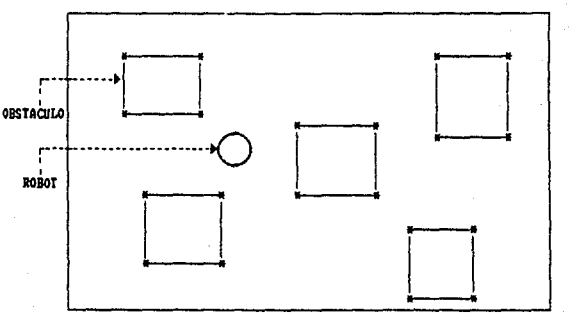


FIGURA 4.3. TECNICA DE GRAFICAS DE VERTICES

4.3.2.3 Mètodos Hibridos Espacio Libre y Gráficas de Vèrtices

Estos mètodos son comunes en robótica móvil. En el mètodo hibrido usado por Chatila y Giralt en HILARE (refs. 7,9), Crowley en FIDO (ref. 8) y otros, representan el espacio libre y los obstáculos por gráficas de vertices. El espacio entre obstáculos es subdividido en regiones convexas. La caracteristica de una region convexa es que cualquier punto dentro de esta region puede ser alcanzado desde otro punto dentro de esa misma region sin colisión (para todos los obstáculos modelados), lo que hace que se redusca la planeación de trayectoria a determinar una secuencia de piezas lineales de recorrido de regiones convexas (ver figura 4.4).

Esta representación es obtenida dinamicamente a partir de datos sensoriales. La planeación de trayectoria es complicada ya que el robot no puede ser tratado como un punto. El espacio es modelado en dos niveles, un nivel topológico, el cual mantiene información considerando lugares y su conectividad, y un nivel geomètrico, que asigna dimensiones a las componentes de la gráfica topologica. Se pueden obtener múltiples sistemas de referencia y con esta representación se permite al robot desarrollar un mapa en areas totalmente desconocidas usando sólo datos sensoriales (laser y visión). Una limitación es su restricción a ambientes interiores y obstáculos fijos. Otro problema es que la incertidumbre no se puede representar directamente.

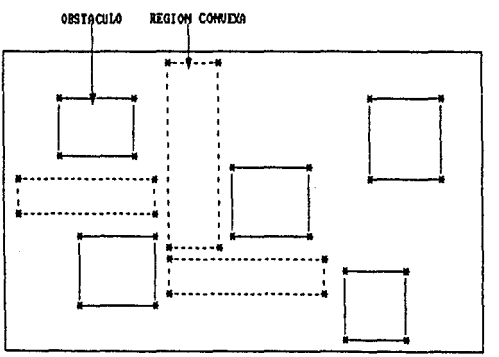


FIGURA 4.4. TECNICA DE ESPACIO LIBRE Y GRAF. DE VERTICES

4.3.2.4 Campos Potenciales

Esta representación usa una analogía gravitacional, convirtiendo los obstáculos en crestas y las trayectorias en valles. La posición inicial del robot es almacenada en una posición más alta que el objetivo y el algoritmo trata al robot como una canica rodando hacia un agujero (ver figura 4.5). Los vectores que representan la atracción del objetivo, el salvamiento de obstáculos y la aceleración existente se combinan para producir el movimiento del robot. Aunque se puede aplicar esta técnica para un robot móvil, la mayoria de trabajos hechos a la fecha tratan con planeación de trayectorias en manipuladores. Esta representación se puede utilizar en los niveles bajos de un sistema multi-nivel para un robot móvil.

La ventaja de esta técnica es que se puede representar la incertidumbre cambiando la pendiente de los obstáculos. Las cuestas muy empinadas producen un alto nivel de confianza, mientras que los obstáculos inciertos producen una lenta subida a una cuesta. Las desventajas son que no se obtienen soluciones óptimas, además de la suceptibilidad del sistema para localizar energia potencial minima, requiriendo extensiones especializadas, como es la inclusión de subobjetivos (ver ref. 12 para más información).

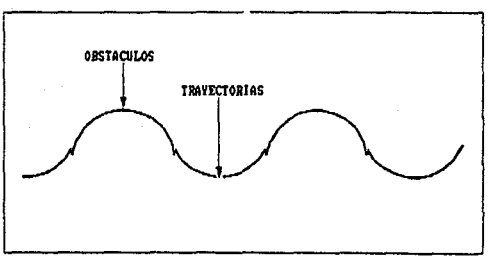


FIGURA 4.5. TECNICA DE CAMPOS POTENCIALES

4.3.2.5 Rejilla Regular

Esta tècnica consiste en representar el espacio por medio de una rejilla cartesiana de dos dimensiones. Algunos sistemas que usan esta tècnica son los desarrollados por Thorpe (ref. 20) y Mitchell (ref. 13).

Un ejemplo del uso de esta tècnica es el sistema de navegación desarrollado por Hughes Artificial Intelligence (ref. 13). Esta representación de rejilla construye un modelo en dos dimensiones del ambiente, donde cada "pixel" representa un espacio de 12.5 m por por 12.5 m. La conectividad es mantenida a través de 8 arcos a cada uno de los "pixels" vecinos (ver figura 4.6). El problema de esta representación es que si la trayectoria no varia, esta solo puede ocurrir en angulos de 45 o 90 grados de nodo a nodo, lo que hace que la trayectoria planeada sea demasiado larga y por lo tanto no óptima. Este tipo de problema se denomina sesgo de digitalización.

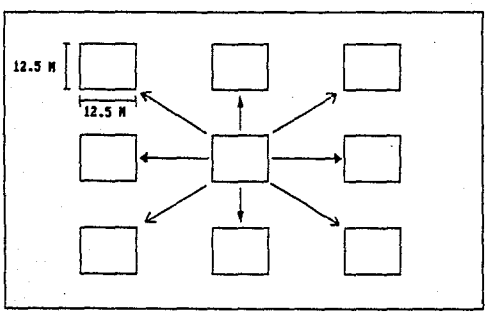


FIGURA 4.6. TECNICA DE REJILLA REGULAR

4.3.2.6 Arbol Cuádrico

Esta técnica usada por investigadores de la Universidad de Maryland (ref. 5), consiste en descomponer el espacio recursivamente en 2 (elevado a la i) por 2 (elevado a la i) areas, hasta alcanzar el nivel más bajo de resolución. Las celdas o pequeñas areas obtenidas se asocian a valores binarios, 1's para region ocupada por un obstáculo y 0's para el espacio libre, por lo que es similar al sistema de rejilla regular (ver figura 4.7). La representación obtenida se usa para planear una trayectoria, la cual consiste en una secuencia de celdas a traves del espacio libre. Para determinar esta secuencia de celdas solo se usan celdas horizontales y verticales, con el fin de tener un factor de seguridad que minimize la probabilidad de recorte de obstáculos durante el recorrido de la trayectoria. La desventaja de hacer esto es que se amplifica el problema de digitalización existente.

La ventaja de esta técnica es que los algoritmos requeridos para planear la trayectoria son más simples comparados con los de rejilla regular. Las desventajas son que la trayectoria generada puede ser burda y que hay perdida de información debido a la codificación binaria.

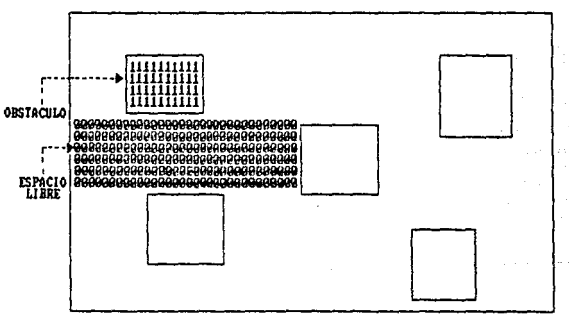


FIGURA 4.7. TECNICA DE ARBOL CUADRICO

4.4.2.7 Representación por Autómatas

Esta técnica fue desarrollada por Tachi y Komoriya (ref. 19) para el proyecto robot "perro-guia". Su caracteristica es que se utilizan autómatas para planear la trayectoria. Los estados del autómata constituyen las marcas de reconocimiento y cada estado contiene información segun el tipo de marca (comunmente solo intersecciones). Las entradas a los estados son comandos direccionales (izquierda, derecho, recto) y las salidas incluyen datos como son los ángulos del volante del vehiculo del robot y distancia a la próxima marca de reconocimiento (ver figura 4.8). Este tipo de representación usando autómatas aunque no es muy comun ha recibido mucha atención.

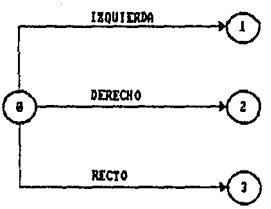


FIGURA 4.8. TECNICA DE AUTOMATAS

4.3.2.8 Representaciones Multi-Nivel

Las representaciones multi-nivel se adaptan más a sistemas con multiples niveles de control, como son los jerárquicos o los distribuidos. Lo que se busca es representar la información de manera que se adecue a diferentes tipos de procesamiento como son: estructuras de alto nivel para planeación de tareas, sistemas coordenados absolutos para recorrido de trayectorias, evitación de obstáculos, etc., (ver figura 4.9). El diseño de este tipo de sistemas debe de tomar en cuenta el manejo adecuado de la información almacenada en las diferentes estructuras, puesto que se puede presentar información inconsistente.

Una aplicación del sistema multi-nivel es el proyecto ALV de DARPA (refs, 18,21). En esta aplicación los datos para planear la trayectoria dependen de los sensores, por lo que la representación diseñada depende del tipo de sensores empleado. Se utilizan coordenadas cartesianas en tres dimensiones y un sistema de pizarra en el que se definen los objetos del ambiente en tres dimensiones. La información en el sistema es manejada a través de fuentes de conocimiento.

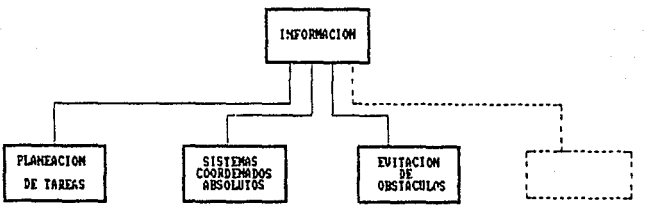


FIGURA 4.9. TECNICA MULTI-NIVEL

4.4 SISTEMA DE PLANEACION DE TRAYECTORIAS PARA UN ROBOT MOVIL (TURMOV)

Para cumplir con uno de los objetivos de este trabajo, en esta sección presentamos el diseño e implantación de un sistema computacional que nos permite realizar la planeación de trayectorias para un robot móvil, al cual hemos denominado TURMOV (Trayectorias para Un Robot MOVil). En la primera parte se describen las caracteristicas generales del sistema, haciendo enfasis en las más importantes; posteriormente se describen cada uno de los módulos que lo componen, explicando en detalle la representación usada y el algoritmo de planeación; y por último se presentan varias simulaciones en diferentes condiciones ambientales.

4.4.1 TURMOV

El sistema TURMOV consta de varios módulos, los cuales interaccionan entre si como se ve en la figura 4.10. El módulo Inicialización del Sistema Gráfico se encarga de inicializar el sistema gráfico para poder visualizar en la pantalla de la computadora el ambiente del robot móvil y la trayectoria planeada. El módulo Selección de Configuración permite al usuario seleccionar uno de los dos tipos de ambiente que se encuentran almacenados en archivos en disco en forma de coordenadas de obstáculos. El módulo Construcción de Representación define la estructura de datos o representación usada para planear las trayectorias. El ambiente del robot se considera como un conjunto de obstáculos fijos configurados en forma de un conjunto de una o más habitaciones y objetos dentro de las mismas. Las coordenadas de estos obstáculos, obtenidas de un archivo de datos, y cuya lectura constituye la generación simulada de los datos del medio ambiente, se usan para llenar una matriz de dos dimensiones, la cual constituye la representación utilizada para llevar a cabo la planeación de trayectorias. El módulo Graficación de Ambiente despliega en pantalla una representación gráfica del ambiente en que se desenvuelve el robot, con el fin de poder visualizar la simulación de las trayectorias por donde se desplaza el robot con respecto a las habitaciones y los objetos. El módulo Selección de Posición Robot permite que el usuario defina una posición inicial del robot, a partir de la cual se calculara la trayectoria adecuada para cumplir con la meta especifica, definida por el usuario a través del módulo Selección de Meta. Esta meta consiste en definir el código de habitación y el número de objeto a alcanzar.

El módulo Planeación de Trayectorias es el más importante, y su función es planear una trayectoria en forma de segmentos de recta, la cual se dirige a traves de los espacios libres que hay entre los obstáculos (objetos y paredes). Esta función la realiza con referencia a la

SISTEMA TURMOV

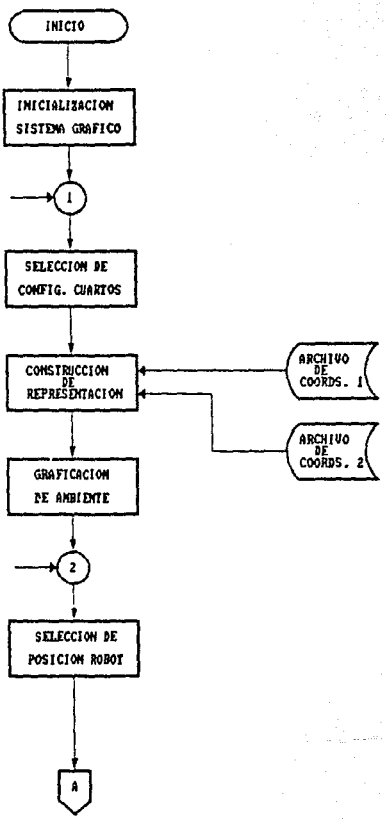


FIGURA 4.18

SISTEMA TURMOV (CONTINUACION)

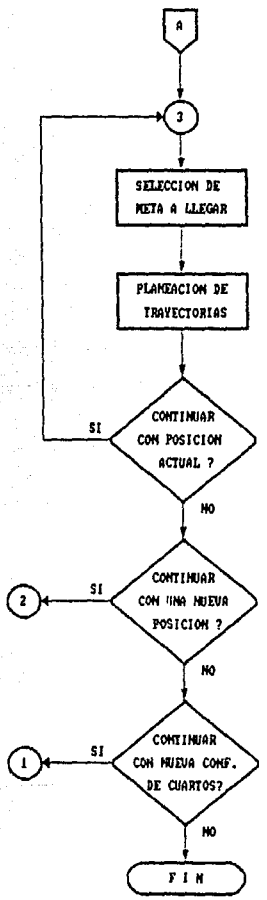


FIGURA 4.18 (CONT.)

representación actualizada del ambiente. La trayectoria se inicia desde la posición del robot y finaliza en el objeto meta especificado. Durante la planeación de trayectorias se considera al robot como un polígono de cuatro lados con ciertas dimensiones que nos permitan calcular la trayectoria considerando un margen de tolerancia entre el robot y los obstáculos.

TURMOV permite, una vez concluida la primer tarea, continuar con la misma tarea o definir una nueva. Como se ve en la figura 4.10, una vez que se ha planeado una trayectoria, es posible volver a empezar el proceso dependiendo de la decisión del usuario. Si se desea que el sistema planee una nueva trayectoria a partir de la posición actual, entonces el control del sistema se transfiere al módulo Selección de Meta; si se define un nuevo proceso de planeación desde una nueva posición inicial del robot, entonces el sistema pasa a la ejecución del módulo Selección de Posición; y por último, si se prefiere seleccionar y definir un nuevo ambiente, entonces el sistema debe ejecutarse a partir del módulo Selección de Configuración.

4.4.2 Organización y función de los Módulos del sistema TURMOV

El sistema TURMOV consta de cuatro módulos principales:

1- INICIALIZACION DEL SISTEMA GRAFICO
2- CONSTRUCCION DE REPRESENTACION
3- GRAFICACION DE AMBIENTE
4- PLANEACION DE TRAYECTORIAS

Módulo: INICIALIZACION DEL SISTEMA GRAFICO

Este módulo es el primero que se ejecuta cuando se corre el sistema y su función es inicializar el sistema gráfico, así como definir las ventanas y el sistema coordenado necesario para gráficar el ambiente del robot móvil. El sistema gráfico esta implantado en base a un conjunto de rutinas del paquete TurboGraphics. La pantalla de la computadora es Jividida en dos regiones o ventanas. Una de ellas, localizada en la parte superior de la pantalla se utiliza para dibujar el ambiente gráfico en donde se visualizan las habitaciones y los objetos, así como la trayectoria planeada; mientras que la otra, se localiza en la parte inferior de la pantalla y se utiliza para que el usuario de información al sistema o para poder visualizar los mensajes que este envia. Este módulo se corre solo una vez durante la ejecución del sistema.

Módulo: CONSTRUCCION DE REPRESENTACION

En este módulo se lleva a cabo la generación de los datos del medio ambiente, suponiendo que son enviados por una camara de televisión. Estos datos se utilizan para construir la representación interna del ambiente que se utiliza durante la planeación de trayectorias. El proceso de simulación consiste en leer de un archivo de datos un conjunto de coordenadas correspondientes a los bloques que conforman la posición de las paredes de las habitaciones y de los objetos que se encuentran en el ambiente de trabajo del robot. Estos datos se almacenan posteriormente en dos arreglos; uno que almacena las coordenadas de todos los bloques, y otro arreglo que guarda la información necesaria para identificar cada habitación y objeto del ambiente.

El primer arreglo de coordenadas es usado para llenar una matriz de dos dimensiones, la cual constituye la representación interna del ambiente usada en la fase de planeación de trayectorias (ver figura 4.11).

MODULO: CONSTRUCCION DE REPRESENTACION

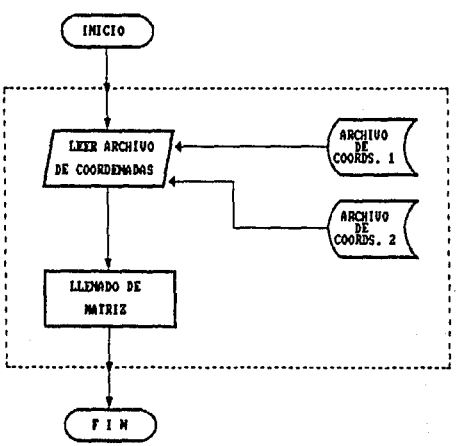


FIGURA 4.11

Esta representación es similar a la de Arbol Cuádrico, y sus caracteristicas son que es una matriz de dos dimensiones con 161 renglones por 226 columnas. Se determinó que estas dimensiones son las adecuadas para representar el medio ambiente del robot, debido a que permite que el algoritmo de planeación de trayectorias funcione en forma eficiente. La matriz contiene valores binarios, 0's para el espacio libre y 1's para el espacio ocupado por un obstáculo. De esta manera un obstáculo es representado como una celda llena de 1's, en donde el número de estos dependera de las dimensiones del obstáculo. El espacio que existe entre los obstáculos se representa por 0's, cuyo número dependera de la distancia que hay entre los diferentes obstáculos (ver figura 4.12).

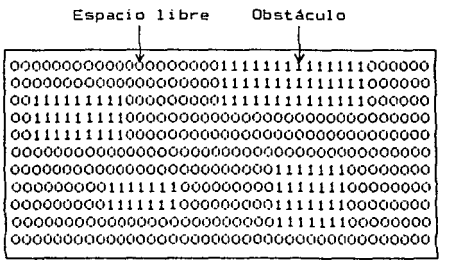


Figura 4.12. Representación del Ambiente

Se uso este tipo de representación debido a que es sencilla de manejar y facil de implementar, dando por resultado que el algoritmo diseñado para planear la trayectoria utilizando esta representación sea muy eficiente. Es posible accesar dos diferentes archivos de coordenadas, constituyendo cada uno un diferente ambiente del robot. Estos archivos de coordenadas son creados externamente a traves de un editor de texto y pueden existir varios. La creación de estos archivos es un paso necesario para poder efectuar la generación simulada de los datos del medio ambiente. La selección del archivo es llevada a cabo por el módulo Selección de Configuración de Cuartos. Para crear este archivo de coordenadas es necesario diseñar primero en papel un medio ambiente constituido por un conjunto de bloques configurados en forma de paredes de habitaciones y objetos. Este diseño se debe hacer con un sistema coordenado de 0 a 225 en X y de 0 a 160 en Y, para que coincida con la matriz de dos dimensiones utilizada para definir la representación. Cuando el diseño se concluye, las coordenadas de los obstáculos que lo constituyen se transcriben a un archivo de datos por medio del editor antes mencionado.

Módulo: GRAFICACION DE AMBIENTE

La función de este módulo es dibujar en la pantalla de la computadora la configuración de habitaciones que se acceso de disco, gráficando las paredes de las habitaciones, las puertas de las habitaciones, los objetos, asi como una escala para visualizar las dimensiones del ambiente y para poder definir una posición inicial del robot. Cada una de las habitaciones y objetos tiene asignado un número y este es dibujado también en la pantalla con el fin de que el usuario pueda definir facilmente una meta (Ver figura 4.13). Para realizar su función este módulo recibe la información de coordenadas e identificación en los arreglos creados en el módulo Construcción de Representación.

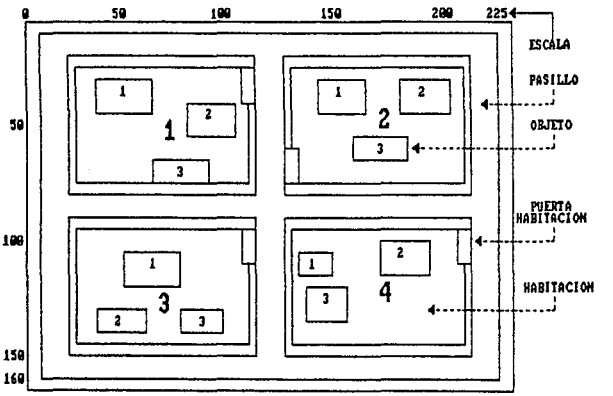


FIGURA 4.13. CONFIGURACION DE HABITACIONES

Módulo: PLANEACION DE TRAYECTORIAS

Una vez inicializado el sistema gráfico, construido la representación y graficado el ambiente, este módulo realiza uan planeación de trayectorias, la cual se inicia desde la posición inicial definida por el usuario hasta un objeto meta especificado también por el usuario. Esta trayectoria es generada en partes y cada parte esta constituida por un cierto número de segmentos de recta. Para realizar su función este módulo recibe la siguiente información: a) representación del medio ambiente y arreglo de coordenadas de

obstáculos definidos ambos en el módulo Construcción de Representación, b) Número de habitación y de objeto meta definidos en el módulo Selección de Meta y c) Posición del robot definida en el módulo Selección de Posición Robot. Dependiendo de la posición del robot y el número de habitación y objeto meta, se decide como se va a generar la trayectoria (ver figura 4.14).

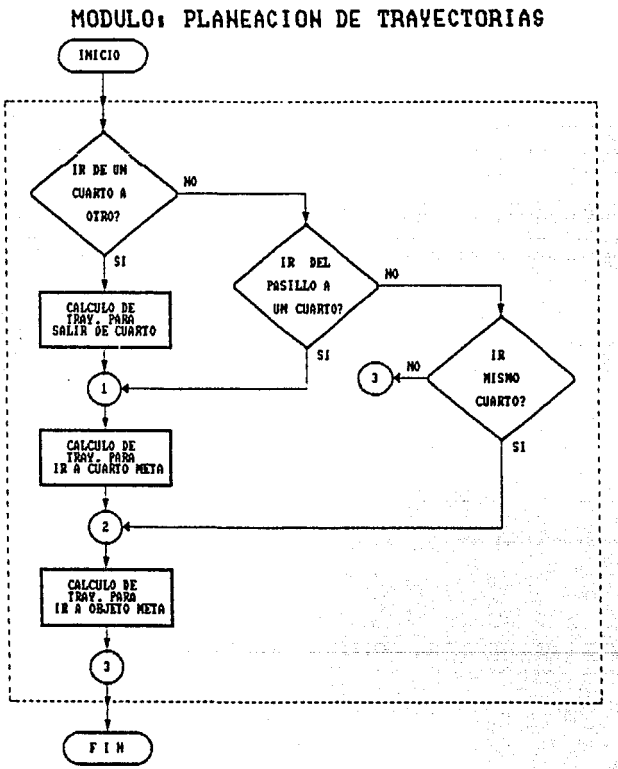


FIGURA 4.14

Por ejemplo si la posición del robot es localizada en el interior de una habitación y la meta es llegar a un objeto de otra habitación, entonces la trayectoria se genera en tres partes. La primera parte se inicia en la posición inicial del robot y finaliza en la puerta de la habitación donde se encuentra este; la segunda va desde esta puerta hasta la puerta de la habitación meta; y la tercera se inicia en esta puerta y finaliza en la vecindad del objeto meta (ver figura 4.15). Esto se hizo asi debido a que la parte del algoritmo que genera cada una de las partes de la trayectoria funciona en base al centro geométrico de un obstáculo; asi, en la primera parte de la trayectoria el centro geométrico corresponde al bloque que constituye la puerta de la habitación origen; en la segunda parte el centro geométrico es el de la puerta de la habitación meta; y en la tercera parte es el del objeto meta.

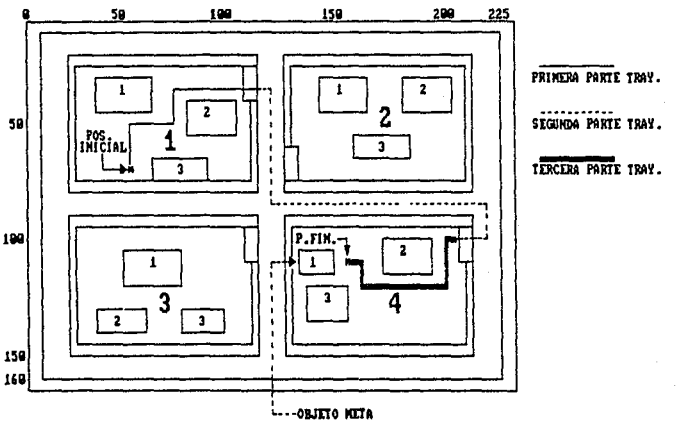


FIGURA 4.15. TRAYECTORIA GENERADA

Para generar cada una de las tres partes de la trayectoria, se determina el centro geométrico del bloque meta y se procede a ejecutar iterativamente los siguientes pasos (ver figura 4.16).

MODULO: CALCULO DE TRAYECTORIAS

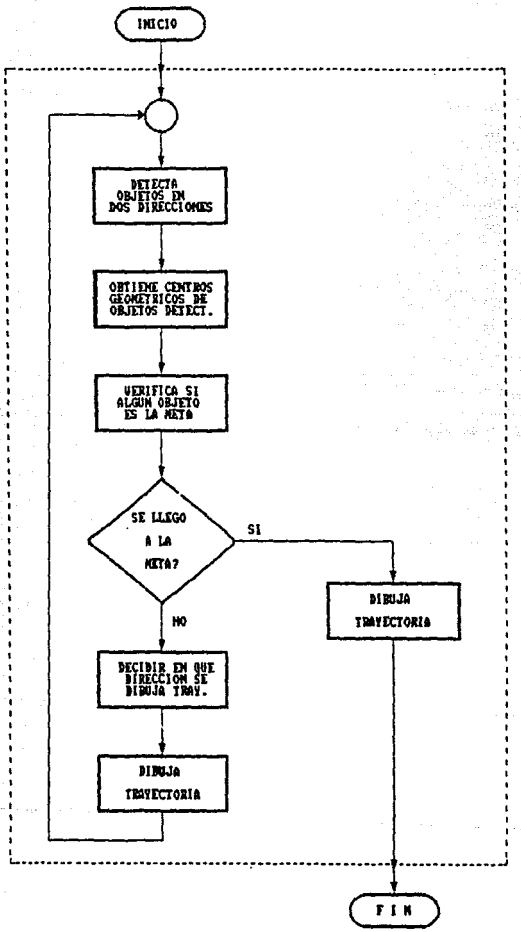


FIGURA 4.16

1- En base a este centro geomètrico y a la posición actual del robot, se determinan las dos direcciones posibles hacia donde se puede generar una trayectoria. Estas dos direcciones posibles se seleccionan de entre cuatro: hacia adelante, hacia atras, hacia la izquierda y hacia la derecha de la posición del robot. Por ejemplo si la meta esta localizada adelante y a la derecha de la posición del robot, entonces las dos direcciones posibles para generar un segmento de trayectoria son adelante y hacia la derecha (ver figura 4.17).

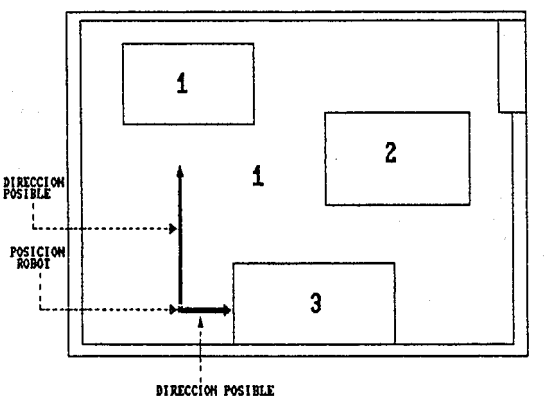


FIGURA 4.17. DIRECCIONES POSIBLES PARA GENERAR TRAY.

2- Se realiza un muestreo en la representación tomando en cuenta las dimensiones del robot con el fin de obtener las coordenadas de cualquier punto de los objetos que existen en las dos direcciones posibles. Esto se hace proyectando los dos puntos de los extremos del robot hasta la región ocupada por un objeto (ver figura 4.18).

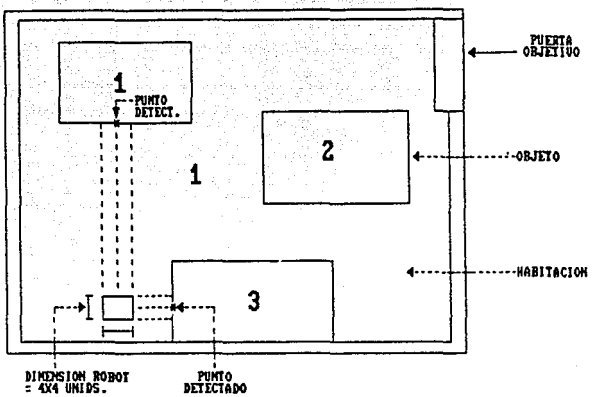


FIGURA 4.18. OBTENCION DE PUNTOS DE OBJETOS

3- Se hace una búsqueda en el arreglo de coordenadas para determinar a que objetos pertenecen los puntos detectados. Una vez obtenidas las coordenadas de las esquinas de los objetos, se determina su centro geométrico.

4- Cada uno de los centros geométricos obtenidos se comparan con el centro geométrico del bloque meta para determinar si son iguales. En el caso de coincidir alguno, quiere decir que se ha detectado el objetivo, y entonces se ejecuta el paso 7, en caso contrario se ejecuta el paso 5.

5- Teniendo como datos los puntos detectados en el paso 2, el centro geométrico del bloque meta y la posición actual del robot, se calculan las distancias en X y en Y, que hay entre la posición del robot y los objetos, y la que hay entre la posición del robot y el centro geométrico del bloque meta. Estas distancias son comparadas entre si, y a traves de un criterio se decide en que dirección se genera un segmento de trayectoria, y también el modo en que se generara: modo directo o modo por pasos.

6- Se genera un segmento de trayectoria en la dirección elegida; si es en modo directo, el segmento de trayectoria se inicia en la posición actual del robot y finaliza en una posición tomando en cuenta una tolerancia de 5 unidades entre objeto y final de la trayectoria (ver fig 4.19).

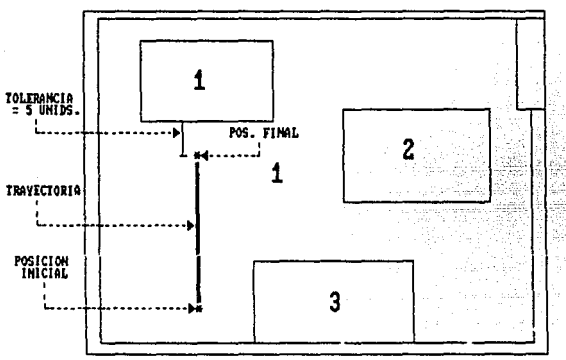


FIGURA 4.19. SEGMENTO DE TRAY. EN MODO DIRECTO

Si el segmento de trayectoria se debe generar en modo por pasos, como sucede en las condiciones de la figura 4.20, entonces comenzando desde la posición actual del robot, se generan dos puntos de trayectoria y se calcula la distancia que hay entre la posición del robot y un punto de un objeto que existe en la otra dirección posible; se vuelven a generar otros dos puntos de trayectoria y se vuelve a calcular la distancia anterior. Este proceso se repite hasta que la distancia actual difiere de la distancia anterior. Cuando esto ocurre, se generan tres puntos de trayectoria en la misma dirección (ver figura 4.20) y se ejecuta el paso 1.

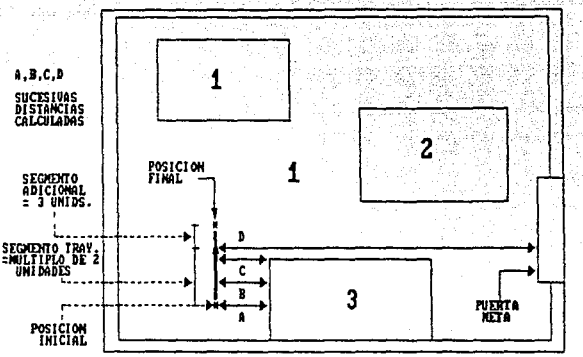


FIGURA 4.20. SEGMENTO DE TRAY. EN MODO POR PASOS

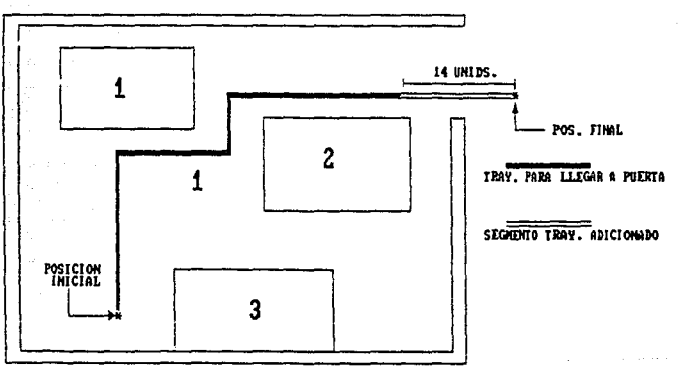


FIGURA 4.21. SEGMENTO DE TRAY. ADICIONAL

7- Se genera un segmento de trayectoria en la dirección elegida en modo directo, y como se ha alcanzado el objeto meta, finaliza la ejecución de este módulo de Planeación de Trayectorias.

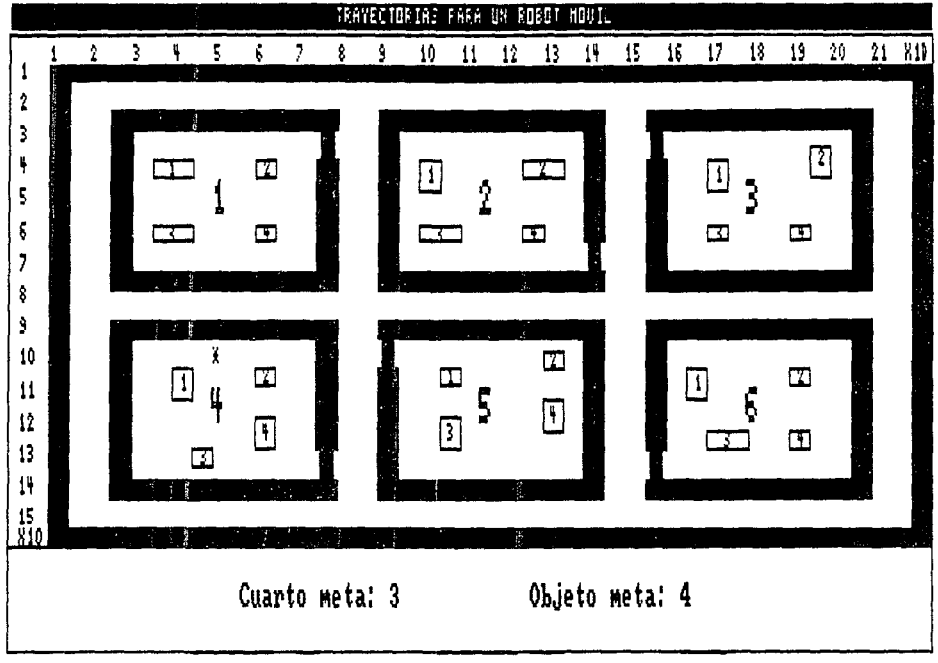
Como parte de la primera y segunda parte de la trayectoria, se debe añadir un pequeño segmento de trayectoria de 14 unidades en la misma dirección que la última calculada, hacia adentro o hacia afuera de una habitación (ver figura 4.21). Esto es necesario debido a que, como se dijo antes, para generar los segmentos de trayectoria se toman en cuenta dos direcciones posibles, y si este segmento no es generado, el algoritmo no funciona.

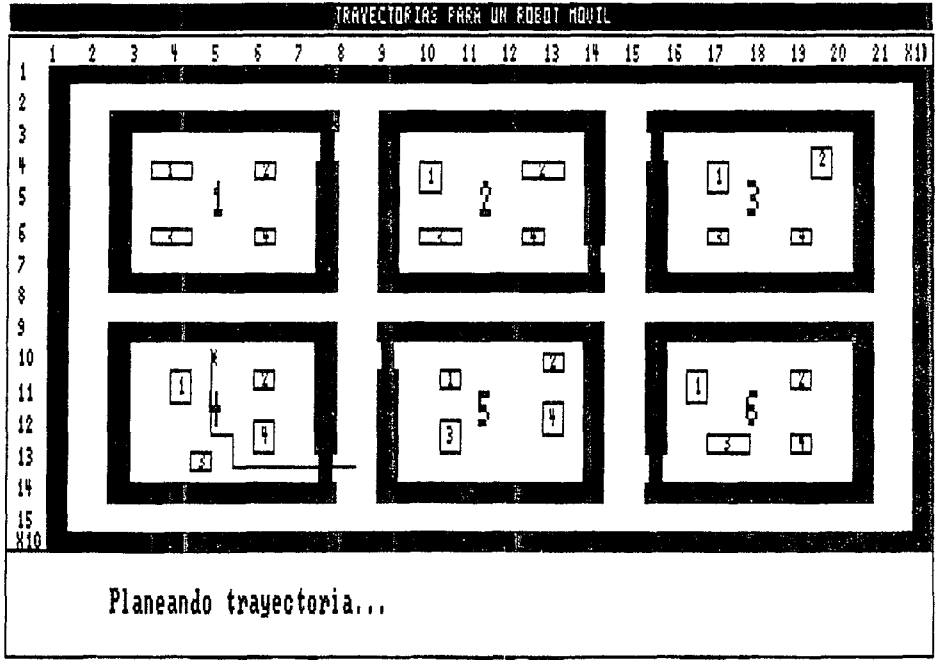
4.4.3 Simulaciones y Resultados

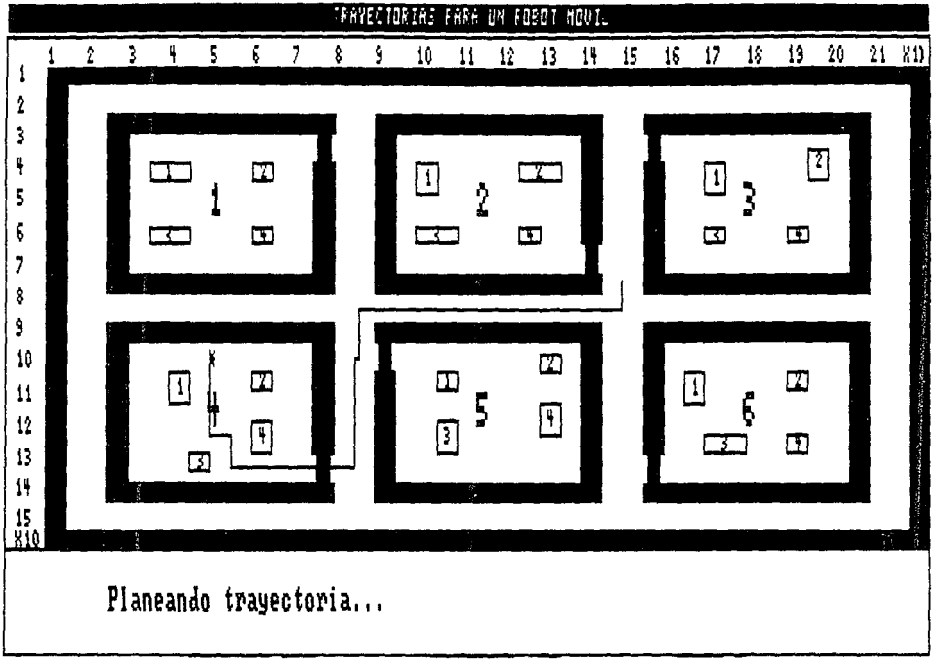
El propósito de esta sección es realizar varias simulaciones sobre el funcionamiento del sistema con el fin de mostrar como funciona el proceso de planeación de trayectorias. Cada simulación tiene como objetivo resolver un problema particular. Para cada simulación se da una descripción del problema a resolver y en seguida se dan las gráficas resultantes de imprimir lo que se presenta en la pantalla de la computadora.

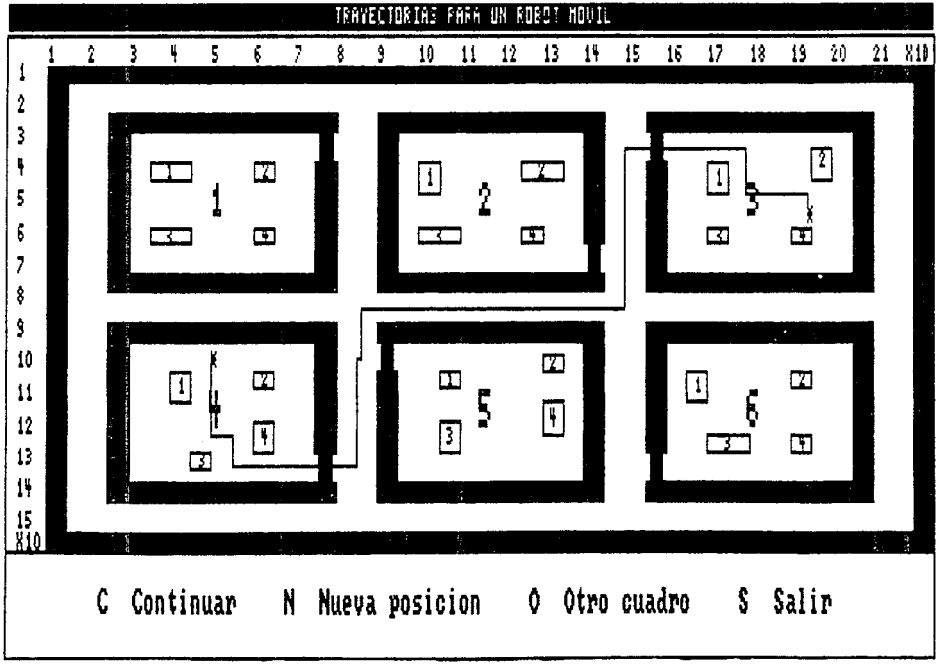
Ejemplo 1:

El robot se encuentra localizado en la habitación 1, en las coordenadas X=50 Y=100; el problema consiste en trasladar el robot movil, desde esta posición hasta la vecindad del objeto 4 de la habitación 3. Una vez dados los datos de posición y meta al sistema, este empieza a realizar la planeación de trayectoria, que se inicia desde la posición inicial para llegar hasta la puerta de la habitación 3 y continua hasta el objeto 4 de esta habitación. Todo este proceso se ilustra en las siguientes gráficas en donde se representa la posición inicial y final del robot por medio de un asterisco (*); la trayectoria generada es representada con segmentos de recta; cada una de las seis habitaciones se representan con bloques rectangulares formando un area cuadrada con un pequeño bloque adicional de menor anchura, colocado en una esquina del area y que representa la puerta; los objetos se representan por medio de bloques rectangulares y cuadrados de varios tamaños.


TRAYECTORIAS PARA UN ROBOT MOVIL
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 X10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 X10
1
2
3
4
5
6
Cuarto meta: 3
Objeto meta: 4
Xrobot: 50
Yrobot: 100
Dir:

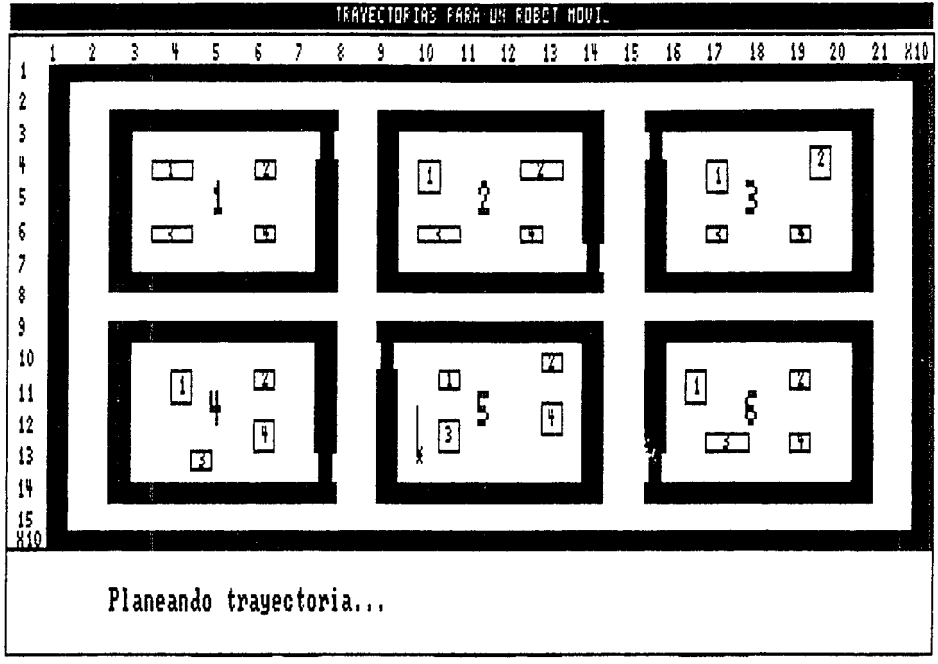

TRAYECTORIAS PARA UN ROBOT MOVIL
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 X10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 X10
Planeando trayectoria...
Xrobot: 85
Yrobot: 134
Dir: →

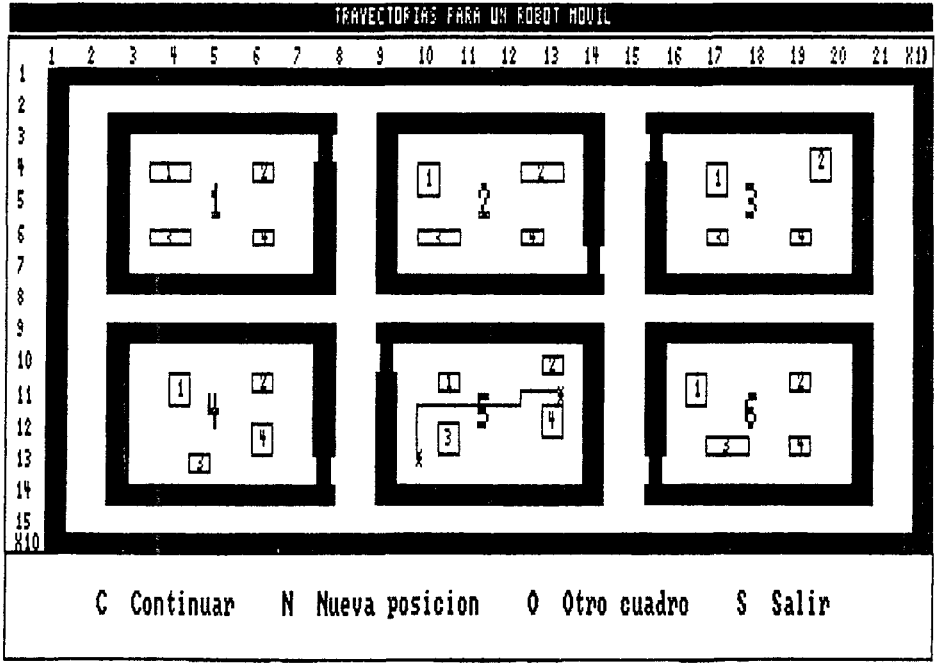

TRAYECTORIAS PARA UN ROBOT MOVIL
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 X10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 X10
1
2
3
4
5
6
Planeando trayectoria...
Xrobot: 150
Yrobot: 76
Dir: ↑


TRAYECTORIAS PARA UN ROBOT MOVIL
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 X10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 X10
C Continuar N Nueva posicion O Otro cuadro S Salir
Xrobot: 194 Yrobot: 55 Dir: ↓

Ejemplo 2:

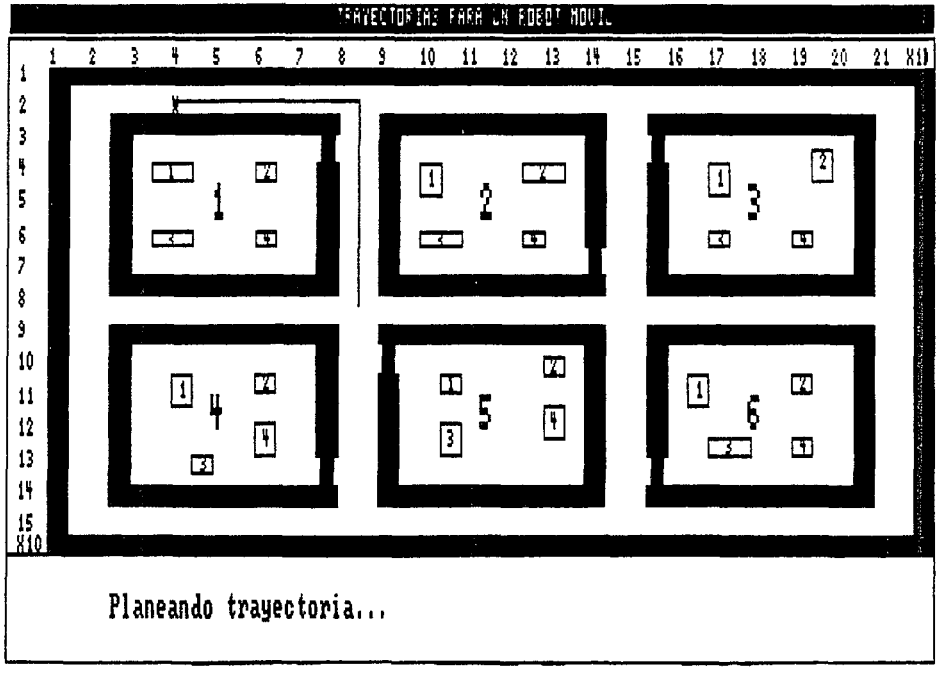
El robot se encuentra en la habitación 5, en la posición X=100 Y=130; el problema a resolver es planear el recorrido que debe hacer el robot móvil para alcanzar el objeto 2 de la misma habitación. De la misma manera que en el ejemplo 1, el sistema pide datos de posición y meta al usuario y posteriormente ejecuta la planeación de trayectorias requerida, que se inicia desde el punto anterior y llega hasta la posición del objeto especificado. Una secuencia de etapas del proceso es ilustrado en las siguientes gráficas.

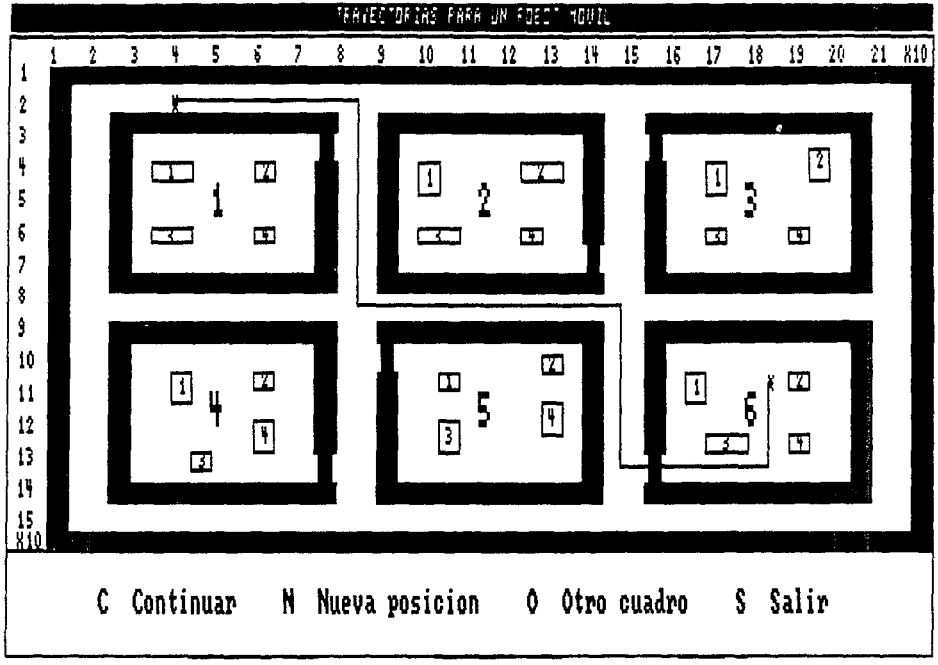

TRAYECTORIAS PARA UN ROBOT MOVIL
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 X10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 X10
1
2
3
4
5
6
Planeando trayectoria...
Xrobot: 100
Yrobot: 115
Dir: ↑


TRAYECTORIAS PARA UN ROBOT MOVIL
C Continuar N Nueva posicion O Otro cuadro S Salir
Xrobot: 134 Yrobot: 110 Dir: ↑

Ejemplo 3:

En este caso, el robot se encuentra en el pasillo, en la posición X=40 Y=20; el problema a resolver consiste en generar una trayectoria de recorrido para el robot móvil desde la posición en el pasillo hasta el objeto 2 de la habitación 6. Para resolver este problema, el sistema solicita los datos de posición y meta al usuario, y empieza a planear la trayectoria, la cual se inicia desde la posición especificada, va hasta la puerta de la habitación 6 y continua hasta el objeto 2 de esta habitación. Las siguientes gráficas ilustran la realización de esta tarea.


TRAYECTORIAS PARA UN ROBOT MOVIL
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 X10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 X10
1
2
3
4
5
6
Planeando trayectoria...
Xrobot: 85
Yrobot: 84
Dir: ↓


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 X10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 X10
1
2
3
4
5
6
C Continuar N Nueva posicion O Otro cuadro S Salir
Xrobot: 185 Yrobot: 107 Dir: →

5. DISCUSION Y CONCLUSIONES

5.1 INTRODUCCION.

El objetivo de este capitulo es realizar una evaluaciòn acerca del trabajo realizado en esta tèsis, asi como una discusiòn sobre los objetivos logrados, las aplicaciones y etapas futuras del mismo. De esta forma, en la secciòn 5.2 se hace una evaluaciòn del proyecto, en la secciòn 5.3 se analizan los objetivos logrados, en la secciòn 5.4 se tratan las aplicaciones para el SPTT y TURMOV, y, para finalizar,en la secciòn 5.5 se plantean posibles extensiones para ambos sistemas.

5.2 EVALUACION DEL PROYECTO

En este trabajo se desarrollaron dos sistemas independientes: SPTT y TURMOV, para la planeaciòn de trayectorias de un manipulador y un robot mòvil respectivamente. Por lo tanto las dos evaluaciones se hacen con respecto al sistema TURMOV y a los subsistemas SCT y SGT que componen al sistema para manipulador SPTT. El primer aspecto se refiere a los algoritmos implementados, exponiendo sus ventajas y desventajas, ademas de las similitudes y diferencias con otros sistemas ya existentes. El segundo aspecto de evaluaciòn se hace con respecto a los criterios de ingenieria de software, analizando ambos sistemas tomando en cuenta aspectos como eficiencia, modularidad, flexibilidad, y transportabilidad.

5.2.1 Algoritmos Implementados

Sistema SCT

El sistema SCT presenta ciertas similitudes respecto al sistema desarrollado por Udupa (1981), ambos sistemas utilizan un procedimiento de regionalizaciòn. Tambièn presentan diferencias, mientras Udupa utiliza dos regiones recursivas, en SCT utilizamos unicamente una; y con respecto a la generaciòn de trayectorias, la diferencia radica en que en SCT no solo calculamos trayectorias lineales, sino que tambièn hay la posibilidad de utilizar trayectorias parabòlicas, (y en posteriores adiciones, definir el tipo de trayectoria algebraica de tipo polinomial, lo cual permite efectuar operaciones de ensamble aun màs complejas que las implementadas actualmente). Otra diferencia con respecto al sistema de Udupa, radica en el tiempo de bùsqueda de la funciòn recursiva òptima para las regiones 1 y 2, mientras en su sistema el tiempo de bùsqueda crece exponecialmente a medida que tenemos un mayor número de objetos en la mesa de trabajo y su estructuraciòn dentro de esta se complica, en nuestro caso (SCT) este tiempo crece linealmente al aumentar el nùmero de objetos.

Otro sistema similar a nuestro SCT es el desarrollado por Lozano Pèrez (1983) al SCT, la similitud radica en que ambos sistemas utilizan un procedimiento geomètrico analitico para generar la región de trabajo en la que se encuentra la trayectoria de traslado. Por otro lado, las diferencias entre SCT y este sistema radican en las formas de representaciòn de la informaciòn; de Lozano Pèrez utiliza estructuras àrbol, mientras que en SCT usamos almacenamiento secuencial. Adicionalmente, el algoritmo de Lozano Perez esta limitado al manipulador cartesiano, mientras que el de SCT puede utilizarse con cualquier tipo de manipulador. Finalmente, el tiempo de computo en efectuar la busqueda en el arbol del sistema de Lozano se expande exponencialmente a medida que crece el número de objetos, mientras que en el sistema SCT lo hace linealmente.

Las ventajas ofrecidas por SCT son las siguientes :

- Precisión en la generaciòn de trayectorias para el traslado de objetos, debido a la facilidad de dar un incremento pequeño a la ecuaciòn numèrica utilizada para la trayectoria lineal y paràbolica.

- Velocidad en el proceso de generar trayectorias y regionalizaciòn.

- Independencia en el uso del tipo de manipulador ya que el SCT considera el movimiento del efector final del manipulador.

- Posibilidad de adicionar algoritmos para generar trayectorias con una geometria distinta a la considerada.

Sistema SGT

Debido a que no contamos con un manipulador real, diseñamos un sistema (SGT), que nos permite visualizar en la pantalla del computador un manipulador efectuando diversas tareas y siguiendo la trayectoria generada por SCT.

El SGT nos sirve para mostrar al usuario, en la pantalla de la computadora, los resultados obtenidos en el SCT. Para esto, se hace uso de ventanas (secciones de la pantalla), donde se visualiza al manipulador siguiendo las trayectorias definidas por SCT para efectuar tareas especificas , asi como el ambiente en el que se mueve el robot, el cual está compuesto de objetos cùbicos. Por medio del SGT, podemos verificar los algoritmos implementados en SCT para planeaciòn y generaciòn de trayectorias. En adiciòn, el SGT también permite, ver la alta precisiòn con que el manipulador efectùa su tarea. Por medio de leds de error (visualizados en seis ventanas), SGT, permite indicar cuando el manipulador no

puede acceder a una posición; esto es, cuando un movimiento de las seis junturas del manipulador está fuera area de trabajo se detiene el movimiento y se prende un led de error, con lo cual se tiene un margen de seguridad para los actuadores. Además, con SGT se pueden visualizar las trayectorias generadas por SCT para librar los obstàculos, asi como el movimiento del manipulador a diferentes velocidades.

Una desventaja del SGT es, que en varias posiciones, el manipulador se muestra distorsionado, lo cual afecta la claridad de visualización.

Sistema TURMOV

Existen varias tècnicas de planeación de trayectorias para el caso de robots móviles, las cuales difieren entre si por el tipo de representación usada en la definición del ambiente donde el robot se desempeña. La tècnica utilizada en TURMOV es muy similar a las de Rejilla Regular y Arbol Cuádrico, debido a que tanto en estas como en TURMOV se utiliza una matriz bidimensional con valores binarios para representar el ambiente del robot; TURMOV difiere del mètodo de Rejilla Regular en que en este se pueden generar trayectorias a 0, 45 y 90 grados, mientras que en nuestro caso sòlo generamos trayectorias a 0 y 90 grados. Con respecto a los mètodos de Espacio Libre, TURMOV difiere de ellos ya que este considera una representación que toma en cuenta la información del espacio libre y de los obstàculos, mientras que los mètodos de Espacio Libre consideran una representación que hace ènfasis sòlo en el espacio libre.

El mètodo de Gràficas de Vèrtices es diferente de TURMOV debido a que usa una representación que solo toma en cuenta los vèrtices de los obstáculos y supone al robot como un circulo, además de que no representa explícitamente el espacio entre obstàculos; TURMOV toma en cuenta la información de todo el perimetro de los obstàculos, el robot es representado como un polígono de cuatro lados y considera la información del espacio libre y de los obstáculos. Los mètodos hìbridos Espacio Libre-Gráficas de Vèrtices son similares a TURMOV en cuanto a la representación explícita del espacio libre y los obstàculos; pero difiere en que los mètodos hìbridos generan regiones convexas a partir del espacio libre, mientras que TURMOV trabaja directamente con la información del espacio libre y los obstàculos.

Las tècnicas de Campos Potenciales y Autòmatas difieren de TURMOV en cuanto al planteamiento de la representación, ya que mientras estos usan una analogía gravitacional y un sistema de autòmatas, TURMOV usa una representación que tiene que ver con los vèrtices de los obstàculos. Finalmente, la tècnica de Representaciones Multi-nivel son diferentes a

TURMOV, en cuanto a que maneja la información para varios niveles de control, como son la planeación de tareas y la planeación de trayectorias, mientras que el manejo de la información en el sistema Turmov es manejada solo en el nivel de planeación de trayectorias.

Entre las ventajas que presenta nuestro sistema TURMOV tenemos que la matriz bidimensional usada para representar el ambiente del robot es sencilla de manejar y facil de implementar, permitiendo que el algoritmo diseñado para generar la trayectoria sea muy eficiente. Otra ventaja se refiere al uso de un ambiente muy estructurado (habitaciones y objetos), donde solo se requiere conocer el tamaño de los obstáculos. Adicionalmente, TURMOV es independiente del vehículo de robot móvil usado.

Las limitaciones presentadas por TURMOV son: sólo permite la generación de trayectoria a ángulos de 0 y 90 grados; los obstáculos definidos en el ambiente deben estar fijos; y solo se utiliza un único sistema de referencia para poder localizar los obstáculos y el robot.

5.2.2 Ingeniería de Software

Otro aspecto importante de este trabajo es el diseño ingenieril de los sistemas desarrollados, por lo que en esta sección lo analizamos desde el punto de vista de ingeniería de software, tomando en cuenta los siguientes aspectos:

-Eficiencia
-Modularidad
-Flexibilidad
-Transportabilidad

Eficiencia

Como eficiencia se entiende la rapidez con que un sistema realiza la función para la cuál fue diseñado. Un sistema es más eficiente en cuanto tiene mayor rapidez de ejecución, lo cual es ventajoso sobre todo para el usuario.

Con respecto a SGT podemos decir que es muy eficiente, ya que basta con que accese los datos de posiciones de objetos de un archivo, para que en un tiempo mínimo planee una trayectoria de movimiento para la pinza del manipulador. Por lo cual SGT podria ser utilizado en el desarrollo de tareas en tiempo real.

El sistema de monitoreo SGT tiene una eficiencia media, ya que si bien, el sistema procesa muy rápidamente los cálculos aritméticos necesarios y el despliegue de imágenes en pantalla también es muy rápido, el proceso de ir

almacenando las imàgenes en memoria principal es lento. Lo anterior se podria resolver parcialmente utilizando un paquete de graficaciòn màs avanzado o un coprocesador matemàtico que permitiera efectuar màs ràpidamente el proceso de captura de imàgenes.

El sistema TuRMov tambièn presenta una alta eficiencia, puesto que tanto los càlculos aritmèticos como el despliegue en pantalla se realizan ràpidamente, requiriendo incluso que en la parte de despliegue gràfico, se ejecuten retardos de tiempo para que el usuario pueda observar detenidamente el proceso. Por lo tanto este sistema podría tambièn funcionar perfectamente en fases que requieran ser ejecutadas en tiempo real.

Modularidad

La modularidad se refiere a la subdivisión de un sistema en módulos, donde cada uno de ellos tiene una función única y específica. Los módulos se comunican entre sí de forma predeterminada.

Debido a que los tres sistemas desarrollados (SCT, SGT y TURMOV) fueron diseñados siguiendo las tècnicas de la programación estructurada, todos tienen una alta modularidad; donde cada módulo tiene una función específica y se comunica con los demás módulos de una manera predeterminada.

Flexibilidad

La flexibilidad se refiere a la posibilidad de agregar con facilidad más módulos a un sistema con objeto de que este pueda realizar funciones adicionales.

Debido a la alta modularidad de los tres sistemas, es fácil añadir un módulo a cualquiera de ellos, para la realización de otras funciones. En las sección de Etapas Futuras de este capitulo se describen algunas posibles funciones adicionales.

Transportabilidad

La transportabilidad se refiere a la facilidad de que un sistema funcione en una computadora distinta de la que fuè originalmente diseñado.

Sòbre el sistema SCT, se puede decir que es totalmente transportable, ya que se programó completamente en Pascal Estandar, pudiendo funcionar en cualquier computadora que soporte este lenguaje.

Para los sistemas SGT y TURMOV, se puede decir que son transportables parcialmente, ya que si bien, la parte de los sistemas que efectúan los cálculos aritmèticos utilizan Pascal Estandar, para el despliegue gráfico se utiliza un paquete gráfico (TurboGraphics) que sólo funciona en microcomputadoras PC compatibles.

5.3 OBJETIVOS LOGRADOS

Con respecto a la consecusiòn de los objetivos propuestos en la introducción (punto 1.2) se puede decir lo siguiente:

En lo que se refiere al sistema SCT, se cumplió con lo propuesto, puesto que se buscaba desarrollar un sistema capáz de planear y ejecutar trayectorias de recorrido para la pinza de un robot manipulador, y el sistema diseñado hace esto en forma eficiente, puesto que dando al sistema como entrada la posición inicial y final de los objetos que queremos trasladar, el sistema se encarga de planear una trayectoria óptima de movimiento de la pinza para recoger y llevar los objetos a las posiciones especificadas. Esta planeación se realiza considerando que el robot recibe información visual a través de un sensor (cámara de T. V.) y que además el robot tiene un ambiente configurado en forma de objetos tridimensionales de ciertas dimensiones.

Para visualizar las especificaciones del ambiente y el resultado de las simulaciones se diseño un sistema gráfico (SGT) en tres dimensiones. Este sistema SGT, nos permite tanto dibujar en la pantalla de la computadora el robot y sus movimientos, planeados por SCT, como trazar los puntos de la trayectoria que debe seguir la pinza.

En el caso del planeador de trayectorias de un robot móvil (TURMOV) se logró diseñar un sistema capáz de planear en forma eficiente una trayectoria de desplazamiento que permite al robot alcanzar un objetivo especifico. En esta planeación, al igual que en el caso del manipulador, se asume que el sistema recibe información visual de un sensor (cámara de T. V.) y además se considera un ambiente estructurado en forma de habitaciones de una casa y objetos dentro de estas habitaciones. Con respecto a la interface gráfica, esta tambièn cumple eficientemente su cometido pues al ejecutarse una trayectoria se puede ver en la pantalla de la computadora como se va trazando la trayectoria para el robot, asi como tambièn la posición y dirección actual de este.

5.4 APLICACIONES PARA SPTT Y TURMOV.

Existen varias aplicaciones que se pueden plantear con el fin de hacer que los sistemas diseñados se utilicen en forma práctica, las cuales se describen enseguida.

El sistema SPTT podría usarse como auxiliar en un curso de robótica a nivel universitario, a través del diseño de un cierto número de prácticas, que permitan al alumno reforzar su conocimiento sobre algún tema específico de la robótica; como por ejemplo, sobre el diseño de un manipulador y sus capacidades de manipulación de objetos, a través de transformaciones homogèneas.

Como herramienta para la investigación, SGT puede ser utilizado para observar en forma gráfica el desarrollo de otros algoritmos para la planeación de trayectorias y así poder realizar un análisis comparativo de los diferentes resultados obtenidos con cada uno de ellos y poder elegir el mejor para aplicaciones específicas. Esto podría realizarse como se hizo en esta tesis, almacenando las trayectorias generadas por esas otras tècnicas de planeación en un archivo de datos, y posteriormente que el sistema SGT accese este archivo para mostrarlas gràficamente.

Aplicado a la industria, el sistema SCT podría emplearse para hacer que un robot manipule componentes, como podría ser por ejemplo la toma de piezas de una banda sin fin en un proceso de producción y trasladarlas a otra parte. Para realizar esto, seria necesario tener un sistema de visión con los sensores (cámara de T.V.) e interfaces apropiadas para detectar las piezas de la banda y adaptar los datos detectados a un formato entendible por el sistema.

Respecto a la utilidad del sistema TURMOV, la más obvia es la aplicación de este a un robot mòvil real. Con las interfaces apropiadas e integrado con un sistema de visión, este sistema se podría utilizar para guiar un robot mòvil a traves de un ambiente como el descrito anteriormente y hacer que realizara tareas como podría ser por ejemplo que fuera a alguna habitación y nos trajera algún objeto de ese lugar.

Otra aplicación para el sistema TURMOV es que puede servir de guía para el diseño de un sistema de planeación de trayectorias con más capacidades, como podría ser por ejemplo un sistema que incluyera la capacidad de planear trayectorias no sólo a ángulos de 0 y 90 grados (izquierda, derecha, adelante, atrás), sino que también pudiera planearlas para cualquier ángulo y el robot pudiera desplazarse en forma diagonal.

5.5 ETAPAS FUTURAS

La gama de extensiones para los sistemas desarrollados es muy amplia ya que de acuerdo a las necesidades que surgen un sistema siempre puede estar mejorando y adaptándose. En seguida se describen las más inmediatas.

Una extensión que se podría hacer al sistema SCT, puede ser la adición del movimiento en forma lateral, ya que de acuerdo con lo descrito en el capítulo tres, el sistema sólo es capáz de planear las trayectorias en forma de parábolas y rectas verticales, así que cuando se presenta un objeto que tiene demasiada altura, para evitarlo el robot debe elevar la pinza demasiado, por lo que si se tuviera capacidad de movimiento lateral el sistema planearía una trayectoria en forma de parabolas y rectas horizontales.

Con respecto al sistema SGT, una posible mejora es que el sistema dibuje al robot en varias vistas, ya que para ciertas posiciones no se nota claramente como esta dispuesto el robot ni tampoco a los objetos que está manipulando, por lo que la adición de una vista como podria ser una de planta, ayudaria a apreciar mejor el movimiento del robot y los objetos que manipula. Esto es, se podrà eliminar la ambiguedad que existe en varias escenas del SGT combinando las tecnicas de despliegue visual tal como proyección en paralelo, proyección en perspectiva supresiòn de lineas ocultas, vistas esquemàticas y recortadas, supresiòn y sombreado de superficies ocultas (Donald Hern 1986).

Para el sistema TURMOV, una extensión podria ser como se dijo en la seccion de utilidad del trabajo. la adición de la planeación en forma de diagonal, o sea a cualquier ángulo, y que el robot no este limitado a moverse solo en trayectorias de 0 y 90 grados (izquierda, derecha, adelante, atrás). Otra extensión posible, aunque no en relación con la parte de planeación es la ampliación de la escala visual del ambiente del robot, lo que posibilitaria por ejemplo que cuando el robot estuviera en una habitación, en pantalla solo se vieran las paredes y objetos de esa habitación pero a una escala más grande y que cuando el robot saliera de la habitación, se visualizaran en pantalla todas las habitaciones a escala normal.

Bibliografia y Referencias Consultadas.

1. Paul Richard P. "Robot Manipulators"; The Massachusetts Institute of Technology, 1981.

2. Jose Ma. Angulo Usategui & Rafael Aviles Gonzales "Curso de Robotica"; Paraninfo 1984, España.

3. IEEE Computer Society "Tutorial on Robotics"; Washington D.C. 1986.

4. Marvin Minsky y otros autores "Robotica"; Ed. Planeta, 1985.

5. Andresen, F.P., Davis, L.S., Eastman, R., and Kambhampati, S., "Visual Algorithms for Autonomous Navigation", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Mo., pp. 856-861, 1985.

6. Brooks, R., "Solving the Find-Path Problem by Good Representation of Free Space", IEEE Systems, Man, and Cyber., SMC-13, No. 3, pp. 190-196, 1983.

7. Chatila, R. and Laumond, J.P., "Position referencing and Consistent World Modeling for Mobile Robots", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Mo., pp. 138-145, 1985.

8. Crowley, J., "Navigation for an Intelligent Mobile Robot", CMU Robotics Institute Technical Report, CMU-RI-TR-84-18, August 1984.

9. Giralt, G., "Mobile Robots", Artificial Intelligence and Robotics, ed. Brady, Gerhardt, and Davidson, Springer-Verlag, NATO ASI series, pp. 365-394, 1984.

10. Giralt, G., Chatila, R., and Vaisset, M., "An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots", Robotics Research, The First International Symposium, MIT Press, pp. 191-214, 1984.

11. Keirsey, D., Mitchell, J., Payton, D., and Preyss, E., "Multilevel Path Planning for Autonomous Vehicles", SPIE Vol. 485, Applications of Artificial Intelligence, pp. 133-137, 1984.

12. Krogh, B., "A Generalized Potential Field Approach to Obstacle Avoidance Control", RI Technical Paper, MS84-484, Soc. of Mech. Engr., 1984.

13. Mitchell, J. and Keirsey, D., "Planning Strategic paths through Variable terrain data", SPIE Vol. 485, Applications of Artificial Intelligence, pp. 172-179, 1984.

14. Mitchell, J. and Papdimitriou, C., "Planning Shortest Paths", (extended abstract), September 1985.

15. Moravec, H. and Elfes, A., "High Resolution Maps from Wide Angle Sonar", CH2152-7/85 1985 IEEE, pp. 116-121, 1985.

16. Moravec, H., "Robot Rover Visual Navigation", UMI PRESS, 1981.

17. Shamos, I. and Hoey, D., "Closest Point Problems", 16th Annual Symposium on the Foundations of Computer Science, pp. 621-628, December 1985.

18. Stentz, A. and Shafer, S., "Module Programmer's Guide to Local Map Builder for ALVan", CMU Computer Science Department, July 1985.

19. Tachi, S. and Komoriya, K., "Guide Dog Robot", ROBOTICS RESEARCH, Second Int. Symposium, ed. Hanafusa and Inoue, MIT Press, pp. 333-340, 1985.

20. Thorpe, C., "Path Relaxation: Path Planning for a Mobile Robot", CMU Robotics Institute Technical Report CMU-RI-TR-84-5, April 1984.

21. Thorpe, C., Kanade, T., and Shafer, S., "ALV System Integration Plan", DRAFT, The Robotics Institute, CMU, May 1985.

22. Tsuji, S., "Monitoring of a Building Environment by a Mobile Robot", ROBOTICS RESEARCH, Second Int. Symposium, ed. Hanafusa and Inoue, MIT Press, pp. 349-356, 1985.

23. Waxman, A.M., Le Moigne, J., and Srinivasan, B., "Visual Navigation of Roadways", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Mo., pp. 862-867, 1985.

24. Joan Ilari, Rafael Huber, Carme Torras "Applying a Tree-Graph Model of Free-Space to the Global Search of Collision-Free Paths"; Institut de Cibernètica; 1er Symposium on Robot Control; Barcelona, Noviembre 1985.

25. Joan Ilari, Rafael Huber, Carme Torras "Planificaciòn de Trayectorias sin colisiòn mediante un nuevo modelo del espacio libre"; Barcelona España.

26. A. Sanfeliu y C. Torras "Cooperaciòn Visiòn-Robot en Tareas de Ensamblaje"; Institut de Cibernetica; II Simposium Nacional de Automàtica en la Industria; Zaragoza, Noviembre 1984.

27. Carme Torras and Federico Thomas "Planning with Constrains: Application to Sensor-Based Robot Assembly Tasks"; Institut de Cibernètica; 1er Symposium on Robot Control; Barcelona, Noviembre 1985.

28. Tomas Lozano Perez "Task Level Manipulator Programming"; Massachusetts Institute of Technology, 1982.

29. William M. Newman & Robert F. Sproull "Principles of Interactive Computer Graphics" ;Second Edition, Mc. Graw Hill, 1981.

30. Steve Harrington "Computer Graphics A Programming Approach" ;Mc. Graw Hill, 1985.

31. Donald Hern & M. Pauline Baker "Gráficas por Computadora", Prentice Hall, 1986.

32. Turbo Pascal version 3.0 "Reference Manual"; Borland International Inc., 1985.

33. Turbo Graphix Toolbox version 1.05A "Owner's Handbook"; Borland International Inc., 1985.

34. Lyle J. Graham & Tim Field "Guia del BPM/PC y XT"; Mc Graw Hill, 1986.

APENDICE A

LISTADOS DE PROGRAMAS

DE LOS SISTEMAS

"SCT" "SGT" Y "TURMOV"

```
(*********************************************************)
(*                                                       *)
(*                     ** SCT **                         *)
(*                                                       *)
(* ----------------------------------------------------- *)
(*                                                       *)
(*     Este mòdulo permite efectuar  la generaciòn       *)
(*     de la trayectorias para cualquier  tipo  de       *)
(*     manipulador.                                      *)
(*     Para la generaciòn de esta, se considero un       *)
(*     medio ambiente  con objetos poco  estructu-       *)
(*     rado, con el objeto  de  eludirlos y  poder       *)
(*     asi efectuar el traslado de  cada  uno  de        *)
(*     ellos a la posicion requerida.                    *)
(*     Finalmente, la  trayectoria  generada  se         *)
(*     almacenara con el objeto de poder  interac-       *)
(*     con el mòdulo de graficaciòn SGTM, el cual,       *)
(*     se encargara de desplegar  en  pantalla  el       *)
(*     trayecto desarrollado en este modulo, para        *)
(*     el traslado de los objetos.                       *)
(*                                                       *)
(* ----------------------------------------------------- *)
(*                                                       *)
(*  AUTORES :                                            *)
(*             Jorge Alfonso Hernandez Santis            *)
(*             Luis Adrian Letepichia Flores             *)
(*             Juan Alfonso Martinez  Padilla.           *)
(*                                                       *)
(*                                                       *)
(*  FECHA :    13 de octubre de 1988.                    *)
(*                                                       *)
(*********************************************************)

program viajero(datas,objetos,traslada,arch_pos);

type

      mat4_4=array[1..4,1..4]of real;
      vect_20=array[1..20]of real;
      vect_3=array[1..3]of real;

      {  registro de objetos  }
      objeta = record
            identif : integer;
             puntos : array[1..4,1..4] of real;
             altura : real;
              cx,cy : real;
                sig : real;
               long : real;
       end;

    { posiciones a colocar los objs }
      posica = record
       objeto : integer;
```

```
      t6 :mat4_4;
    rotz : real;
end;

 { reg_pos_pinza }
 reg1 = record
   t6 : mat4_4;
 end;

   { reg_pos_data }
  reg0 = record
 identif: integer;
    puntos : mat4_4;
    cx,cy : real;
    pos_inic:vect_3;
    pos_fin:vect_3;
  end;

    { regc_long_n }
     reg4 = record
       obj_numero : real;
       long : real;
     end;

    { reg_angs  }
    reg2 = record
     altura : real;
     sigma1 : real;
     sigma2 : real;
    end;

   {  regc_grav  }
    reg3 = record
     cx1,cy1 : real;
     cgf_x1,cgf_y1 : real;
     altura : real;
    end;


 { reg_p_ult  }
  reg5 =record
   eleva :real;
   maxima:real;
  end;

  reg6= record
   altura :real;
   objeto :integer;
  end;

  object = array[1..5]of objeta;
  posicion = array[1..5]of posica;
  vector = array[1..20]of reg6;
```

```
var
        reg_pos_data  : reg0;
        reg_pos_pinza : reg1;
           reg_angs   : reg2;
  regc_grav,regc_grav1 : reg3;
        regc_long_n    : reg4;
          reg_p_ult    : reg5;
          reg_vect     : vector;

          arch_data : file of reg0;
          arch_pos  : file of reg1;
               post : file of integer;
              datas : file of integer;
            objetos : file of object;
           traslada : file of posicion;
  reg_obja,reg_obj  : object;
            reg_pos : posicion;
               obst : object;

{ se efectua la impresion de una transformacion }
  procedure imprime( matriz : mat4_4 );

   var
        i,j : integer;

  begin

       writeln('_____________________________________
                ______________');
          for i:=1 to 4 do
          begin
              for j:=1 to 4 do
              write(' ':10,matriz[i,j]:7:3);
              writeln;
          end;
          write('_____________________________________
                ______________');
          writeln('_____________________________________');
             writeln;
             writeln;

  end;

procedure guarda_arch(reg_obj:object;reg_pos:posicion);
 var
        n,contador,f:integer;
        i,j:integer;
        matris1:mat4_4;
begin
  writeln(' ':15,'---------------');
  writeln(' ':15,'guarda_objetos');
  writeln(' ':15,'--------');
  reset(datas);
  seek(datas,0);
```

```
    read(datas,n);

    rewrite(objetos);
    contador:=0;f:=1;

    while  n>contador do
    begin
        contador:=contador + 1;
        with reg_obj[f] do
        begin
          write(objetos,reg_obj);
          for i:=1 to 4 do
          for j:=1 to 4 do
          matris1[i,j]:=puntos[i,j];
          imprime(matris1);
        end;
        f:=f+1;
    end;
    close(objetos);
 end;


procedure lee_objetos(var reg_obj:object);
 var
      n,f,contador,i,j :integer;
      matris1 :mat4_4;

begin
writeln(' ':15,'---------------');
writeln(' ':15,'imprime_objetos');
writeln(' ':15,'---------');
reset(datas);
seek(datas,0);
read(datas,n);
contador:=0;
f:=0;
reset(objetos);
while n>contador do
begin
   contador:=contador+1;
   f:=f+1;
   with reg_obj[f] do
   begin
      reset(objetos);
      writeln(' ':15,'f=',f);
      seek(objetos,f-1);
      read(objetos,reg_obj);
      for i:=1 to 4 do
       for j:=1 to 4 do
       matris1[i,j]:=puntos[i,j];
       imprime(matris1);
    end;
 end;
   close(objetos);
```

```
end;

procedure lee_posicion(var reg_pos:posicion);
 var
     f,n,i,j,contd:integer;
     matris1:mat4_4;
begin
    writeln(' ':15,'------------');
    writeln('  ':15,'lee posicion');
    writeln(' ':15,'--------');
    reset(post);
    seek(post,0);
    read(post,n);
    writeln(' ':15,'npos=',n);
    f:=1;
    reset(traslada);
    contd:=0;
    while n-1>contd do
    begin
         read(traslada,reg_pos);
         with reg_pos[f] do
         begin
                for i:=1 to 4 do
                 for j:=1 to 4 do
                 matris1[i,j]:=t6[i,j];
                 imprime(matris1);
          end;
          f:=f+1;
          contd:=contd+1;
      end;
         close(traslada);
  end;


    { defino la posicion inicial del manipulador }
  procedure pos_pinza( var reg_pos_pinza : reg1 );
   var
       i,j : integer;
  begin
    with reg_pos_pinza do
    begin
       writeln(' ':15,'pos inicial pinza');
       writeln(' ':15,'--------');
       t6[1,1]:= 1;  t6[1,2]:= 0;
       t6[1,3]:= 0;  t6[1,4]:=0;
       t6[2,1]:= 0;  t6[2,2]:=1;
       t6[2,3]:= 0;  t6[2,4]:=6;
       t6[3,1]:= 0;  t6[3,2]:= 0;
       t6[3,3]:= 1;  t6[3,4]:= 0;
       t6[4,1]:= 0;  t6[4,2]:= 0;
       t6[4,3]:= 0;  t6[4,4]:= 1;
       write('_____________________________________________');
```

```
      writeln(' ':15,'_____________________');
      for i:= 1 to 4 do
      begin
        for j:= 1 to 4 do
          write(' ':10,t6[i,j]:3);
        writeln;
      end;
       write('_____________________________________________');
       writeln(' ':15,'_____________________');
    end;
  end;

{_____________________________________________________}

{     transfiero  del registro de posiciones a la  matriz  de
posiciones     }
  procedure c_mat_pos( l:integer;a : posicion;
                                   var matriz : mat4_4 );
   var
       i,j : integer;
  begin
     writeln(' ':15,'--------');
     writeln(' ':15,'c_mat_pos');writeln(' ':15,'--------');
     with a[l] do
     begin
       for i:=1 to 4 do
       for j:=1 to 4 do
         matriz[i,j]:= t6[i,j];
     end;
  end;

{       transfiero  del  registro de objetos a la  matris  de
        objetos      }
  procedure c_mat_obj(l:integer; b : object;
                          var matriz : mat4_4 );
   var
       i,j : integer;
  begin
     writeln(' ':15,'----------');
     writeln(' ':15,'c_mat_obj');writeln(' ':15,'--------');
     with b[l] do
     begin
        for i:=1 to 4 do
        for j:=1 to 4 do
          matriz[i,j]:= puntos[i,j];
     end;
  end;


  {     multiplica dos transformaciones tipo mat4_4     }
  procedure mult_tr( t1 : mat4_4;  t2:mat4_4;
                          var producto : mat4_4 );
   var
      i,j,k : integer;
```

```
      constante : real;

  begin
       constante:=0.0;
       i:= 1; j:= 1; k:= 1;
       while i <= 4 do
       begin
       while k <= 4 do
       begin
         while j <= 4 do
         begin
            constante:= t1[i,j] * t2[j,k] + constante;
            producto[i,k]:= constante;
            j:= j+1;
         end;
         k:= k + 1; j:= 1; constante:= 0;
       end;
       writeln;
       i:= i+1; j:= 1; k:= 1;
      end;
  end;

{asigna  a  c_grav los valores dados por la  posicion  actual
  pinza y obj}
procedure reasigna( n : integer ; reg_pos_pinza : reg1 ;
                        regc_grav1 :  reg3 ;
                        var  regc_grav : reg3 );
begin
     with regc_grav do
     begin
       writeln(' ':15,'-------');
       writeln(' ':15,'reasigna');writeln(' ':15,'--------');
         cgf_x1:= regc_grav1.cx1;
       { almaceno en cgf_x la posicion inicial del objeto }
         cgf_y1:= regc_grav1.cy1;
         if n=1 then
         begin
           cx1:= reg_pos_pinza.t6[1,4];
           cy1:= reg_pos_pinza.t6[2,4];
         end
         else
          begin
             cx1:= 14;
             cy1:= 13;
          end;
      end;
  end;

{  encuentra  a  los objetos que se encuentran dentro  de  la
   region de trabajo }
procedure busqueda_obj( num : integer ;
                        xm,xmx,ym,ymx,mn,bb:real;
                        var stack_object : vect_20 );
 var
```

```
      flag : boolean; y : real;
      j,i,l,f,nobject,total,g,contador,s,t,o: integer;
      entero1 : integer;
      objetos : integer;
procedure answer1(var i :integer);
begin
  i := 0;
  with  reg_obj[f] do
  begin
   if( puntos[2,j]<= ymx) and  (puntos[1,j]<=xmx) and
              (puntos[1,j]>=xm)and(puntos[2,j]>=ym)and
              (identif<>regc_long_n.obj_numero)and
              (puntos[2,j]>=mn*(puntos[1,j])+bb)
   then
       i:=1;
   end;
end;

procedure answer2( var i :integer );
begin
  with reg_obj[f] do
  begin
    i:=0;
    if (puntos[2,j]>=ym)and(puntos[1,j]>=xm)
       and(puntos[2,j]<=mn*(puntos[1,j])+bb)
       and(puntos[2,j]<=ymx)and(puntos[1,j]<=xmx)
       and(identif<>regc_long_n.obj_numero)
    then
         i:=1;
  end;
end;

procedure answer3( var i :integer );
begin
  i := 0;
  with reg_obj[f] do
  begin
    if (puntos[2,j]>=ym) and(puntos[1,j]<=xmx)
        and(puntos[2,j]<=mn*puntos[1,j]+bb)
        and(puntos[1,j]>=xm)and(puntos[2,j]<=ymx)
        and(identif<>regc_long_n.obj_numero)
    then
         i:= 1;
 end;
end;

procedure answer4( var i : integer );
begin
  with reg_obj[f] do
  begin
    i:=0;
    if (puntos[2,j]<=ymx)and(puntos[1,j]>=xm)
       and(puntos[2,j]>=mn*puntos[1,j]+bb)
       and(puntos[1,j]<=xmx)and(puntos[2,j]>=ym)
```

```
        and(identif<>regc_long_n.obj_numero)
     then
          i:=1;
  end;
end;

 procedure answer5( var i:integer );
 begin
   i:=0;
   with reg_obj[f] do
   begin
    if (puntos[1,j]>=xm)and(puntos[1,j]<=xmx)and
       (puntos[2,j]>=ym)and(puntos[2,j]>=ym)and
     (puntos[2,j]<=ymx)and( identif<>regc_long_n.obj_numero)
    then
         i:=1;
   end;
 end;

begin
  s:=1;
  for l:=1 to 20 do
   stack_object[l]:=0;
  j:=1;
  l:=1;f:=1;
  t:=1;
  flag:=false;
  total:=1;
  i:=1;
  reset(datas);
    read(datas,objetos);
  close(datas);
  repeat
     for s:=1 to 4 do
     begin {imp objeto a analizar  }
       for t:=1 to 4 do
       write(reg_obj[f].puntos[s,t]:8:4);
       writeln;
     end;
     while j <= 4 do
     begin
       case num of
         1: answer1(o);
         2: answer2(o);
         3: answer3(o);
         4: answer4(o);
         5: answer5(o);
       end;
       if o=1 then
       begin
         stack_object[l]:=reg_obj[f].identif;
          flag:=true;
       end;
       j:= j+1;
```

```
    end;
    f:= f+1;
    if flag = true then l:= l+1;
     flag:= false;
    j:= 1;
    total:= total+1;
   until total>objetos;
 end;

{_____________________________________________________________}

{    se encuentra la interseccion de dos vectores que se
     encuentran dentro de la misma region de trabajo  }
  procedure compara_stacks( sta,stb : vect_20 ;
                            var dentro_tray :vect_20 );
   var
       i,j,k : integer;
  begin
    for i:=1 to 5 do
     dentro_tray[i]:= 0;
    writeln(' ':15,'-------------');
    writeln(' ':15,'compara staks');
    writeln(' ':15,'--------');
    writeln(' ':15,'sta');
    i:= 1; j:= 1; k:= 1;
    while i<=5 do
    begin
      while j<=5 do
      begin
        if (sta[i]=stb[j])and (sta[i]<>0) then
        begin
          dentro_tray[k]:=sta[i]; k:=k+1;
        end;
        j:= j+1;
      end;
      i:= i+1; j:= 1;
    end;
  end;


{  encuentro la altura maxina y minima de los objetos que se
   encuentran dentro del vector de objetos dentro de la
   region }
  procedure maxi_mini( a : vect_20; var  maxi,min : real);
   var
       i,n : integer;
       buff : real;
  begin
    writeln(' ':15,'maxi_mini');writeln(' ':15,'--------');
    i:= 1; n:= 1;
    repeat
      repeat
        if (a[i]<a[i+1])then
        begin
```

```
            buff:= a[i];
            a[i]:= a[i+1];
            a[i+1]:= buff;
          end;
          i:= i+1;
        until i = 4;
        i:= 1;  n:= n+1;
      until  n = 3;
      maxi:= a[1];
      min:= a[4];
    end;


    {  analizo el vector de objetos efectuando una busqueda  en
       el archivo de objs. hasta encontrar al elemento de mayor
       altura }
    procedure altura_max( reg_vect :vector ; var  max : real );
     var
         a,i,j,s : integer;
         min : real;
         stack_height : vect_20;
    begin
      writeln(' ':15,'----------');
      writeln(' ':15,'altura_max');writeln(' ':15,'--------');
      for j:=1 to 10 do
       stack_height[j]:= 0;
      for i:=1 to 5 do
       stack_height[i]:=reg_vect[i].altura;
      maxi_mini(stack_height,max,min);
    end;

{ se encuentra la region de trabajo de la pinza del manipulador  }
    procedure min_region(secuen_p:integer; regc_long_n:reg4;
                         regc_grav :reg3 ;
                         var reg_p_ult :reg5 );
     var
         flag,f : boolean;
         long : real;
         cons : real;
         maxima : real;
         interseccion,stacka,stackb : vect_20;
         m1,m2 : real;
         ordenada1,ordenada2 : real;
         xm,ym,xmx,ymx : real;
         aprim,bprim,cprim,dprim : real;

     procedure asig( n : integer;var buff : real );
     begin
       with regc_grav do
       begin
         case n of
          1 : buff:= cx1;
          2 : buff:= cy1;
          3 : buff:= cgf_x1;
```

```
        4 : buff:= cgf_y1;
        5 : buff:= cx1+long;
        6 : buff:= cx1-regc_long_n.long;
        7 : buff:= cy1+regc_long_n.long;
        8 : buff:= cy1-regc_long_n.long;
        9 : buff:= cgf_x1+regc_long_n.long;
       10 : buff:= cgf_x1-regc_long_n.long;
       11 : buff:=cgf_y1+regc_long_n.long;
       12 : buff:= cgf_y1-regc_long_n.long;
      end;
    end;
  end;

  procedure ubica_altura(interseccion:vect_20;
                                secuen_p:integer;
                                var reg_vect:vector );
   var
       s,i,j :integer;
       ultimo_obj:integer;
  begin
    writeln(' ':15,'------------');
    writeln(' ':15,'ubica_altura');
    writeln(' ':15,'--------');
    s:=1;
    i:=1;
    j:=1;
    while (interseccion[s]>0.2) or (  s<5.0) do
    begin
      for i:=1 to secuen_p - 1 do
      if reg_pos[i].objeto=interseccion[j] then
        ultimo_obj:=i;
      if reg_pos[i].t6[3,4]>reg_obj[i].puntos[3,4] then
      with reg_vect[s] do
      begin
        altura:=reg_pos[i].t6[3,4];
        objeto:=reg_obj[i].identif;
      end
      else
        with reg_vect[s] do
        begin
          altura:=reg_obj[i].puntos[3,4];
          objeto:=reg_obj[i].identif;
        end;
      s:=s+1;
      j:=j+1;
    end;
  end;

begin
   writeln(' ':15,'--------------');
   writeln(' ':15,'min_region');writeln(' ':15,'--------');
    with regc_grav do
    begin
      f:= false;
```

```
      cons:= 0.0;
      if( cx1>cgf_x1 )and( cy1>cgf_y1 )then
      begin
        flag:= true;
        asig(10,xm);asig(4,aprim);
        asig(12,ym);asig(3,cprim);
        asig(5,xmx);asig(2,bprim);
        asig(7,ymx);asig(1,dprim);
      end;
      if ( cx1<cgf_x1 )and ( cy1<cgf_y1 ) then
      begin
        flag:=true;
        asig(6,xm);asig(2,aprim);
        asig(8,ym);asig(1,cprim);
        asig(9,xmx);asig(4,bprim);
        asig(11,ymx);asig(3,dprim);
      end;
      if( cx1>cgf_x1 ) and ( cy1<cgf_y1 ) then
      begin
        flag:=false;
        asig(10,xm);asig(4,aprim);
        asig(11,ymx);asig(3,dprim);
        asig(5,xmx);asig(2,bprim);
        asig(8,ym);asig(1,cprim);
      end;
      if ( cx1<cgf_x1 ) and ( cy1>cgf_y1 ) then
      begin
        flag:=false;
        asig(6,xm);asig(2,aprim);
        asig(12,ym);asig(3,cprim);
        asig(9,xmx);asig(4,bprim);
        asig(7,ymx);asig(1,dprim);
      end;
      if( cx1=cgf_x1 ) and ( cy1>cgf_y1 ) then
      begin
        f:=true;
        asig(6,xm);asig(2,aprim);
        asig(12,ym);asig(3,cprim);
        asig(5,xmx);asig(2,bprim);
        asig(7,ymx);asig(1,dprim);
      end;
      if ( cx1=cgf_x1 ) and ( cy1<cgf_y1 ) then
      begin
        f:=true;
        asig(6,xm);asig(2,aprim);
        asig(8,ym);asig(1,cprim);
        asig(5,xmx);asig(2,bprim);
        asig(11,ymx);asig(3,dprim);
       end;
      if ( cx1>cgf_x1 ) and ( cy1=cgf_y1 ) then
      begin
        f:=true;
        asig(10,xm);asig(4,aprim);
        asig(12,ym);asig(3,cprim);
```

```
    asig(5,xmx);asig(2,bprim);
    asig(7,ymx);asig(1,dprim);
  end;
  if ( cx1<cgf_x1 ) and ( cy1=cgf_y1 ) then
  begin
    {los objs forman 0 g resp eje x}
    f:=true;
    asig(6,xm);asig(2,aprim);
    asig(8,ym);asig(1,cprim);
    asig(9,xmx);asig(4,bprim);
    asig(11,ymx);asig(3,dprim);
  end;
  if flag=true and ( f=false ) then
  begin
    {la m es + y no forma cero grados}
    { m + }
    m1:= (ymx-aprim)/(dprim-xm);
    m2:= (bprim-ym)/(xmx-cprim);
    ordenada1:= aprim-m1*xm;
    ordenada2:= ym-m2*cprim;
  end;
  if flag=false and (f=false) then
  begin
    { m - }
    {la m es - y no forma 1 ang cero grados}
    m1:= (ym-aprim)/(cprim-xm);
    m2:= (bprim-ymx)/(xmx-dprim);
    ordenada1:= aprim-m1*xm;
    ordenada2:= ymx-m2*dprim;
  end;
  {si la m es - busco objs en su region}
  if flag = false and ( f = false ) then
  begin
    busqueda_obj(1,xm,xmx,ym,ymx,m1,ordenada1,stacka);
    busqueda_obj(2,xm,xmx,ym,ymx,m2,ordenada2,stackb);
  end;
  {si la m es + busco objs en su region}
  if( flag=true ) and( f=false ) then
  begin
    busqueda_obj(3,xm,xmx,ym,ymx,m1,ordenada1,stacka);
    busqueda_obj(4,xm,xmx,ym,ymx,m2,ordenada2,stackb);
  end;
  if f=true then
  {considero todas las opciones}
  begin
    busqueda_obj(5,xm,xmx,ym,ymx,m1,ordenada1,stacka);
  end;
  if f=false then
   compara_stacks(stacka,stackb,interseccion)
  else
      interseccion:=stacka;
  ubica_altura(interseccion,secuen_p,reg_vect);
  altura_max(reg_vect,reg_p_ult.maxima);
  reg_p_ult.maxima:=2;
```

```
    end;
  end;


{ se encuentra el valor de la pendiente y del angulo teta que
  sera utilizado en el calculo de la trayect. parabolica }
  procedure trans_ang_base( regc_grav : reg3;
                                var teta : real );
  var
      consta : real;

    procedure assig( n : integer; var teta : real);
     var
         m : real;
    begin
      with regc_grav do
      begin
        m:= (cgf_y1-cy1)/(cgf_x1-cx1);
        teta:= abs(arctan(m));
        case n of
          1 : teta:= 2*3.14+teta;
          2 : teta:= 2*3.14-teta;
          3 : teta:= 3.14+teta;
          4 : teta:= 3.14-teta;
          0 : teta:= teta;
        end;
      end;
  end;

  begin
   with regc_grav do
   begin
    writeln(' ':15,'--------------');
    writeln(' ':15,'trans_ang_base');
    writeln(' ':15,'--------');
    if (cx1>cgf_x1) and (cy1=cgf_y1) then
     teta:= 180;
    if (cx1<cgf_x1)and(cy1=cgf_y1)then
     teta:= 0;
    if (cx1=cgf_x1) and (cy1<cgf_y1) then
     teta:= 3.1415926/2;
    if( cx1=cgf_x1) and (cy1>cgf_y1) then
      teta:= 3.1416+(3.1416/2);
    consta:=0;
    if (cx1>consta) and (cgf_x1>consta) then
    begin
      if (cx1<cgf_x1) and (cy1<cgf_y1) then
        assig(0,teta);
      if (cx1<cgf_x1)and(cy1>cgf_y1) then
        assig(2,teta);
      if (cx1>cgf_x1) and (cy1<cgf_y1) then
        assig(4,teta);
      if (cx1>cgf_x1) and (cy1>cgf_y1) then
        assig(3,teta);
```

```
    end;
    if(cx1<0) and(cgf_x1<0)then
    begin
      if(cx1<cgf_x1)and(cy1<cgf_y1)then
        assig(0,teta);
      if(cx1<cgf_x1)and(cy1>cgf_y1)then
        assig(2,teta);
      if (cx1>cgf_x1)and(cy1>cgf_y1) then
        assig(3,teta);
      if (cx1>cgf_x1) and (cy1<cgf_y1) then
        assig(4,teta);
    end;
    if((cx1<0)and(cgf_x1>0)or (cx1>0)and(cgf_x1<0))then
    begin
      if(cx1>cgf_x1)and(cy1>cgf_y1) then
        assig(3,teta);
      if(cx1<cgf_x1)and(cy1>cgf_y1)then
        assig(2,teta);
      if(cx1<cgf_x1)and(cy1<cgf_y1)then
        assig(0,teta);
      if (cx1>cgf_x1)and(cy1<cgf_y1)then
        assig(4,teta);
    end;
    teta:= (180*teta)/3.141592654;
  end;
end;

      {    se  efectua el calculo de tó para  la  trayectoria
           vertical    }
 procedure altura_inic_pinza(reg_p_ult: reg5;
                             var reg_pos_pinza : reg1 );

 var
      consta,i,j,inc,a: real;
 begin
   with reg_p_ult do
   begin
     inc:= 0.5;
     writeln(' ':15,' ':10,'altura inic-pinza');
     writeln(' ':15,' ':10,'-----------------');
     if( maxima<=0) or(maxima>=10) then
       maxima:=2.5;
     if reg_p_ult.eleva<0 then
       reg_p_ult.eleva:=1;
     if maxima>reg_p_ult.eleva then
     begin
       while reg_p_ult.eleva<maxima do
       begin
         reg_p_ult.eleva:=reg_p_ult.eleva+inc;
         reg_pos_pinza.t6[3,4]:= reg_p_ult.eleva;
         imprime(reg_pos_pinza.t6);
         write(arch_pos,reg_pos_pinza);
       end;
     end;
```

```
  end;
end;

{       se asignan parametros a la transformacion de
        traslacion    }
procedure transla( x,y,z : real ; var trans : mat4_4 );
begin
  trans[1,1]:=1;  trans[1,2]:=0;  trans[1,3]:=0;
  trans[1,4]:=x;
  trans[2,1]:=0;  trans[2,2]:=1;  trans[2,3]:=0;
  trans[2,4]:=y;
  trans[3,1]:=0;  trans[3,2]:=0;  trans[3,3]:=1;
  trans[3,4]:=z;
  trans[4,1]:=0;  trans[4,2]:=0;  trans[4,3]:=0;
  trans[4,4]:=1;
end;

    {   se asignan parametros a la transformac. de
        rotacion    }
procedure rot( xyz :  char; omega : real;
                          var rota : mat4_4 );
 var
     t:real;
begin

  writeln(' ':15,'rot');
  omega:= omega*3.141592654/180;
  if xyz='x'  then
  begin
    rota[1,1]:=1; rota[1,2]:=0;
    rota[1,3]:=0; rota[1,4]:=0;
    rota[2,1]:=0; rota[2,2]:=cos(omega);
    rota[2,3]:=-sin(omega); rota[2,4]:=0;
    rota[3,1]:=0; rota[3,2]:=sin(omega);
    rota[3,3]:=cos(omega); rota[3,4]:=0;
    rota[4,1]:=0; rota[4,2]:=0;
    rota[4,3]:=0; rota[4,4]:=1;
 end;
 if  xyz='y'  then
 begin
   rota[1,1]:=cos(t); rota[1,2]:=0;
   rota[1,3]:=sin(t); rota[1,4]:=0;
   rota[2,1]:=0; rota[2,2]:=1;
   rota[2,3]:=0; rota[2,4]:=0;
   rota[3,1]:=-sin(omega); rota[3,2]:=0;
   rota[3,3]:=cos(omega); rota[3,4]:=0;
   rota[4,1]:=0; rota[4,2]:=0;
   rota[4,3]:=0; rota[4,4]:=1;
 end;
  if    xyz='z'  then
  begin
   rota[1,1]:=cos(omega); rota[1,2]:=-sin(omega);
   rota[1,3]:=0;  rota[1,4]:=0;
   rota[2,1]:=sin(omega); rota[2,2]:=cos(omega);
```

```
                rota[2,3]:=0;  rota[2,4]:=0;
     rota[3,1]:=0;  rota[3,2]:=0;
     rota[3,3]:=1;  rota[3,4]:=0;
     rota[4,1]:=0;  rota[4,2]:=0;
     rota[4,3]:=0;  rota[4,4]:=1;
   end;
end;


{  actualizo  los archivos de objetos con aquellos que ya  se
actualizaron }
procedure toma_sig_posic(secuen_p:integer;reg_obj:object;
                         reg_pos:posicion;
                         var regangs : reg2;
                         var regcgravi : reg3;
                         var  reg_long_n : reg4 ;
                         var reg_obja : object;
                         var pos_obj : integer);
    var
        i,j : integer;
        beta : real;
        nobj,t,b,d : mat4_4;
  begin
    writeln(' ':15,'--------------');
    writeln(' ':15,'toma_sig_posic');
    writeln(' ':15,'--------');
    { encuentro angulo a ser colocado el objeto }
      with reg_pos[secuen_p] do
       begin
         reg_angs.sigma2:= rotz;
       end;
     { almaceno posicion final  en registro c.gravedad }
       with regc_gravi do
         begin
           cgf_x1:= reg_pos[secuen_p].t6[1,4];
           cgf_y1:= reg_pos[secuen_p].t6[2,4];
           altura:=reg_pos[secuen_p].t6[3,4];
         end;
         pos_obj:=reg_pos[secuen_p].objeto;
         {   almaceno posic inicial obj    }
        with regc_gravi do
          begin
            cx1:= reg_obj[pos_obj].cx;
            cy1:= reg_obj[pos_obj].cy;
          end;
         with regc_long_n do
           begin
             regc_long_n.obj_numero:=
             reg_obj[pos_obj].identif;
             long:= reg_obj[pos_obj].long;
           end;
          with reg_angs do
            begin
              altura:= reg_obj[pos_obj].altura;
```

```
            sigma1:= reg_obj[pos_obj].sig;
          end;
        with reg_pos_pinza do
          begin
            t[1,1]:=1;t[1,2]:=0;
            t[1,3]:=0;t[1,4]:=-regc_grav1.cx1;
            t[2,1]:=0;t[2,2]:=1;
            t[2,3]:=0;t[2,4]:=-regc_grav1.cy1;
            t[3,1]:=0;t[3,2]:=0;
            t[3,3]:=1;t[3,4]:=0;
            t[4,1]:=0;t[4,2]:=0;
            t[4,3]:=0;t[4,4]:=1;
          end;
        c_mat_obj(pos_obj,reg_obj,nobj);
        { calc beta en grados }
        beta:= reg_angs.sigma2-reg_angs.sigma1;
        { tras origen }
        mult_tr(t,nobj,t);
        imprime(t);
        {  giro origen  }
        rot('z',beta,d);
        mult_tr(d,t,t);
        {  tras posic final  }
        b[1,1]:=1; b[1,2]:=0;
        b[1,3]:=0; b[1,4]:=reg_pos[secuen_p].t6[1,4];
        b[2,1]:=0; b[2,2]:=1;
        b[2,3]:=0; b[2,4]:=reg_pos[secuen_p].t6[2,4];
        b[3,1]:=0; b[3,2]:=0;
        b[3,3]:=1; b[3,4]:=0;
        b[4,1]:=0; b[4,2]:=0;
        b[4,3]:=0; b[4,4]:=1;
        mult_tr(b,t,t);
        { almaceno reg.coord nuevos puntos
                                  trasladado y rotado}
        with reg_obja[pos_obj] do
         begin
           for i:=1 to 4 do
           for j:=1 to 4 do
           puntos[i,j]:= t[i,j];
           cx:= reg_pos[secuen_p].t6[1,4];
           cy:= reg_pos[secuen_p].t6[2,4];
           { transfiero coordenadas a reg_obj  }
           long:= reg_obj[pos_obj].long;
           altura:= reg_obj[pos_obj].altura;
           reg_angs.sigma1:= reg_obj[pos_obj].sig;
           identif:= reg_obj[pos_obj].identif;
         end;
      end;

procedure tray_no_lin(var reg_pos_pinza :reg1;
                          regc_grav:reg3; teta :real;
                                    var ang_sal :real);
{ me da el valor de t6 en distintos valores para la
                              trayectoria no lineal }
```

```
  var
      a,d,z,x,y : real;
      pfx,pix,pfy,piy : real;
      mat_4 : mat4_4;
      inc_region : real;
      oo,s,inc  : real;
      h,k,p2 : real;
      omega : real;
      consta:real;
      hh,g,contador_1:real;
   { se encuentra el valor de la matris de rotacion
                        para la  trayectoria parabolica }
procedure rot_ejes( reg_pos_pinza : regi;
                        var mat_rot : mat4_4);
  var
       a1,a2, z1,z2,mp1,mp2: real;
       bb1,bb2,mm : real;
       incx, a : real;
   begin
    with reg_pos_pinza do
      begin
        writeln(' ':15,'-------');
        writeln(' ':15,'rot-ejes');
        writeln(' ':15,'--------');
        a1:= t6[1,4];
        incx:= 0.2;
        bb1:= sqr(a1-h);
        bb2:= 4*p2;
        z1:= bb1/bb2+k;
        a2:= a1+incx;
        bb1:= sqr(a2-h);
        bb2:= 4*p2;
        z2:= bb1/bb2+k;
        mp1:= (z2-z1);
        mp2:= a2-a1;
        bmm:=mp1/mp2;
        omega:= arctan(mp1/mp2);
        omega:= (omega*180)/3.1415926;
        omega:= 90-omega;
        (*calculo de rotacion*)
        rot('y',omega,mat_rot);
      end;
  end;

  { se encuentra la matris de transformacion
    referida a las coordenadas base del manipulador }
  procedure  transf_base( ax,ay,az : real;
                              var  t6_base : mat4_4);
    var
         i,j : integer;
         a,b,c : mat4_4;
     begin
       writeln(' ':15,'-----------');
       writeln(' ':15,'transf_base');
```

```
    writeln(' ':15,'--------');
    transla(ax,ay,az,a);
    rot('z',teta,b);
    mult_tr(a,b,b);
    mult_tr(b,t6_base,t6_base);
  end;
  begin
    writeln(' ':15,'--------------');
    writeln(' ':15,'tray no lineal');
    writeln(' ':15,'--------');
      writeln(' ':15,' alturaz=  ',
                          reg_pos_pinza.t6[3,4]:8:4);
   a:= reg_pos_pinza.t6[3,4];
 with regc_grav do
   begin
        pfx:= cgf_x1;
        pfy:= cgf_y1;
        pix:= cx1;
        piy:= cy1;
        inc_region:= 0.2;
        inc:= 0.2;
        x:= 0; y:= 0; z:= 0;
        {     vertices de la parabola      }
        {  parametros de la parabola  }
         h:= sqrt(sqr(pfx-pix)+sqr((pfy-piy)))/2;
         d:= 2*h;
         k:= a+inc_region;
         aa:= sqr(x-h);
         aa1:= 4*(z-k);
         p2:= aa/aa1;
        end;
      contador_1:=0.0;
      consta:=8.0;
      hh:=0;g:=0;
       while x<=d  do
        begin
          contador_1:=contador_1+1.0;
          g:=g+1;
        with reg_pos_pinza do
          begin
          {     calculo de z en funcion de x       }
          aa4:= sqr(x-h);
          aa3:= 4*p2;
          z:= aa4/aa3+k;
         t6[1,1]:=1; t6[1,2]:=0; t6[1,3]:=0;
         t6[2,1]:=0; t6[2,2]:=1; t6[2,3]:=0;
         t6[3,1]:=0; t6[3,2]:=0; t6[3,3]:=1;
         t6[4,1]:=0; t6[4,2]:=0; t6[4,3]:=0;
         t6[1,4]:=x;
         t6[2,4]:=0;
         t6[3,4]:=z;
         t6[4,4]:=1;
        with reg_pos_pinza do
          begin
```

```
              { calculo transf rotacion }
              rot_ejes(reg_pos_pinza.mat_4);
              {    calculo transf. traslacion e>
         with regc_grav do
              transf_base(cx1,cy1,reg_p_ult,maxima,t6);
              {    calculo de transf. translacion*rot    }
              mult_tr(t6,mat_4,t6);
              imprime(t6);
              x:= x+inc;
              ang_sal:= omega;
              writeln(' ':15,'ang_sal= ',ang_sal:8:4);
              {  almaceno en archivo  }
              write(arch_pos,reg_pos_pinza);
            end;
        end;
    end;
end;

{     me da las posiciones para la etapa de ascenso
                                  y descenso vertical      }
procedure  asc_rota_vert(  n,pos_obj,secuen_p:integer;sigma1,
sigma2 : real; reg_obj:object;   var  reg_pos_pinza : reg1;
                        reg_pos:posicion;var reg_p_ult:reg5);
          procedure desciende(var reg_pos_pinza :reg1);
                                      var inalt:real;
            begin
              inalt:=0.1;
              with reg_pos_pinza do
                begin
                  t6[3,4]:=t6[3,4]-inalt;
                  writeln(' ':15,'altura pinza=',t6[3,4]);
                  imprime(t6);
                  write(arch_pos,reg_pos_pinza);
                end;
             end;
    var
        incang    : real;
        mat_b : mat4_4;
    begin
      writeln('-------------');
      writeln('asc_rota_vert');writeln(' ':15,'--------');
      writeln(' ':15,'sigma1=',sigma1,'sigma2=',sigma2);
       with reg_pos_pinza do
         begin
           writeln(' ':15,' altura= ',
                                reg_pos_data.puntos[3,4]:8:4);
     end;
     incang:=0.3;
      with reg_pos_pinza do
         begin
           reg_pos_pinza.t6[3,4]:=reg_p_ult.maxima;
            if sigma1<sigma2 then
              begin
                with reg_pos_pinza do
```

```
                    begin
                     repeat
                       rot('z',sigma1,mat_b);
                       sigma1:=sigma1+incang;
                       mult_tr(t6,mat_b,t6);
                       imprime(t6);
                       write(arch_pos,reg_pos_pinza);
                      until sigma1>=sigma2;
                  end;
                end;
              if sigma1>sigma2 then
                begin
                  with reg_pos_pinza do
                    begin
                     repeat
                       rot('z',sigma1,mat_b);
                       sigma1:=sigma1-incang;
                       mult_tr(t6,mat_b,t6);
                       bimprime(t6);
                       write(arch_pos,reg_pos_pinza);
                      until sigma1<=sigma2;
                     end;
                 end;
          writeln('regobj=',reg_obj[pos_obj].puntos[3,4]);
          writeln('reg pos=',reg_pos[secuen_p].t6[3,4]);
          writeln('pos_obj=',pos_obj);
          writeln('secuen p=',secuen_p);
           with reg_pos_pinza do
             begin
               case n of
                 1: begin
                      repeat
                       desciende(reg_pos_pinza);
                      until t6[3,4]-
                            0.1<reg_obj[pos_obj].puntos[3,4];
                     end;
                 2: begin
                      repeat
                        desciende(reg_pos_pinza);
                    until t6[3,4]-
                          0.1<reg_pos[secuen_p].t6[3,4];
                    end;
               end;
            reg_p_ult.eleva:=reg_pos_pinza.t6[3,4];
        end;
    end;
  end;
  {_____}
procedure transfiere(reg_c_grav1:reg3;reg_c_grav:reg3);
    begin
      with regc_grav do
        begin
          writeln(' ':15,'transfiere');
          writeln(' ':15,'--------');
```

```
          cx1:=regc_grav1.cx1;
          cgf_x1:=regc_grav1.cgf_x1;
          cy1:=regc_grav1.cy1;
          cgf_y1:=regc_grav1.cgf_y1;
        end;
    end;
{_____________________________________________________________}
procedure trayectorias(reg_obj:object;reg_pos:posicion);
   var
       secuen_p : integer;
       bteta    : real;
       ang_sal  : real;
       sigma2,sigma1 : real;
       pos_obj  : integer;
       i,j : integer;
        numobj : integer;
    begin
      reset(datas);
      read(datas,numobj);
      write('num objs=',numobj);
      reg_p_ult.eleva:=0;
      secuen_p :=1;
      reg_p_ult.maxima:=0;
      pos_pinza(reg_pos_pinza);
      toma_sig_posic(secuen_p,reg_obj,reg_pos,
                     reg_angs,regc_g rav1,regc_long_n,
                                          reg_obja,pos_obj);
   {calculo coordenadas p.inicial pinza y pos.obj a
                                        tomar del reg.c.g}
   reasigna(1,reg_pos_pinza,regc_grav1,regc_grav);
     repeat
       for i:=1 to 4 do
         begin
           for j:=1 to 4 do
              reg_pos_data.puntos[i,j]:=
                reg_obj[pos_obj].puntos[i,j];
         end;
       with reg_pos_data do
         begin
           cx:=reg_obj[pos_obj].cx;
           cy:=reg_obj[pos_obj].cy;
           identif:=reg_obj[pos_obj].identif;
           pos_fin[1]:=regc_grav1.cgf_x1;
           pos_fin[2]:=regc_grav1.cgf_y1;
           pos_fin[3]:=reg_pos[secuen_p].t6[3,4];
           bpos_inic[1]:=regc_grav1.cx1;
           pos_inic[2]:=regc_grav1.cy1;
           pos_inic[3]:=reg_obj[pos_obj].puntos[3,4];
           write(arch_data,reg_pos_data);
         end;
         min_region(secuen_p,regc_long_n,regc_grav,reg_p_ult);
         trans_ang_base(regc_grav,teta);
         altura_inic_pinza(reg_p_ult,reg_pos_pinza);
         tray_no_lin(reg_pos_pinza,regc_grav,teta,ang_sal);
```

```
        sigma1:=reg_angs.sigma1;
        asc_rota_vert(1,pos_obj,secuen_p,0,sigma1,
                      reg_obj,reg_pos_pinza,reg_pos,reg_p_ult);
        {obtengo coordenadas de p.inicial objeto y p.final
                                                 a colocarse}
  transfiere(regc_grav1,regc_grav);
  min_region(secuen_p,regc_long_n,regc_grav,reg_p_ult);
  trans_ang_base(regc_grav,teta);
  altura_inic_pinza(reg_p_ult,reg_pos_pinza);
  tray_no_lin(reg_pos_pinza,regc_grav,teta,ang_sal);
  sigma1:=reg_angs.sigma1;
  sigma2:=reg_angs.sigma2;
  asc_rota_vert(2,pos_obj,secuen_p,sigma1,
          sigma2,reg_obj,reg_pos_pinza,reg_pos,reg_p_ult);
  {transfiere al reg. objetos el reg. trasladado}
  reg_obj[pos_obj]:=reg_obja[pos_obj];
   {inc.apunt registro posiciones}
   secuen_p:=secuen_p+1;
  toma_sig_posic(secuen_p,reg_obj,reg_pos,
           reg_angs,regc_grav1,regc_long_n,reg_obja,pos_obj);
   {obtengo coordenadas de ult.posic pinza y
                                  nueva pos.objeto a tomar}
   reasigna(1,reg_pos_pinza,regc_grav1,regc_grav);
until secuen_p=numobj+1;
end;
{_____________________________________________________________}
{              P   R   I   N   C   I   P   A   L              }
{_____________________________________________________________}
 begin
          assign(arch_data,'b:arch_dta.dta');
          assign(datas,'b:datos.dta');
          assign(arch_pos,'b:punts.dta');
          assign(post,'b:pos.dta');
          assign(objetos,'b:object.dta');
          assign(traslada,'b:posicion.dta');
          lee_objetos(reg_obj);
          lee_posicion(reg_pos);
          rewrite(arch_pos);
          rewrite(arch_data);
          trayectorias(reg_obj,reg_pos);
          close(arch_pos);
          close(arch_datas);
          guarda_arch(reg_obj,reg_pos);
  end.
```

```
(*********************************************************)
(*                                                       *)
(*                      ** SGT **                        *)
(*                                                       *)
(* ----------------------------------------------------- *)
(*                                                       *)
(*      Este modulo permite simular el posiciona-        *)
(*      miento de un manipulador con seis grados         *)
(*      de libertad. En este caso se hara la  si-        *)
(*      mulaciòn del manipulador desarrollado en         *)
(*      la Universidad de Stanford.                      *)
(*      Una vez que se hizo el  posicionamiento          *)
(*      entonces, el manipulador sera capaz  de          *)
(*      tomar  un  objeto  y  trasladarlo  hacia         *)
(*      alguna posiciòn, claro esta, siguiendo           *)
(*      una trayectoria previamente planeada  en         *)
(*      en  el   programa   de   PLANEACION   DE         *)
(*      TRAYECTORIAS.                                    *)
(*                                                       *)
(* ----------------------------------------------------- *)
(*                                                       *)
(*    AUTORES :                                          *)
(*              Luis Adrian Letepichia Flores.           *)
(*              Juan Alfonso Martinez  Padilla.          *)
(*              Jorge Alfonso Hernadez Santis.           *)
(*                                                       *)
(*    FECHA :    13 de octubre de 1988.                  *)
(*                                                       *)
(*********************************************************)

program manipulador;
{$i typedef.sys}
{$i graphix.sys}
{$i kernel.sys}
{$i windows.sys}

 procedure manipula;
  const
      rax=1;{ rango en x }
      ray=1;{ rango en y }
      raz=2.5;{ rango en z}
      sx=15;{ escala en x }
      sy=15;{ escala en y }
      sz=20;{ escala eb z }
      d2=30;{   se   usara  para  calcular  los  angulos   de
              movimiento del manipulador }
  type
     arr2_vert_manip=array[0..6,1..10,1..2] of real;
     arr3_vert_manip_obj=array[1..10,1..4] of real;
     arr_puntos=array[1..4,1..4] of real;
     arr_tvert_obj=array[1..4,1..10] of real;
     reg_tó=record
            nx,ny,nz,ox,oy,oz,
            ax,ay,az,px,py,pz:real;
```

```
    end;(* reg_t6 *)
    reg_angs=record
       t1,t2,d3,t4,t5,t6:real;
    end;(* reg_angs *)
    posiciones=record
       px,py,pz:real;
    end;
    reg_puntos=record
      t6:arr_puntos;
    end;
    reg_obj=record
         nobj:integer;
         vert_obj:arr3_vert_manip_obj;
         pos_ini,pos_final:posiciones;
         transf_obj:arr_puntos;
     end;
    archi_obj=file of reg_obj;
    archi_t6=file of reg_t6;

var
   arch_t6:archi_t6;  { archivo de matrices t6
                        para la simulacion }
   arch_g_t6:archi_t6; { archivo para guardar la trayectoria
                     para el objeto que se va a manipular }
   arch_obj,aux_arch_obj:archi_obj;{ archivo de objetos para
                                     la  simulacion }
   objn:reg_obj;  { registro de objetos para la simulacion }
   detecta_obj:boolean; { para detectar un objeto }
   rad,aspect_loc:real;{para el radio inicial de un circulo}
   rang1,rang2,rang3,rang4,rang5,rang6:real;{ rangos
                                        para la posicion
                                        inicial y final
                                       de los objetos }
   t6aux:reg_t6;  { registro de t6s para la simulacion }
   cont_fig:integer;{ contadores de imagenes }
   angulos:reg_angs;{para valores de los grados de libertad}
   err_ang:boolean; { variable para detectar si el rango de
                      movimiento   de   cada   parte   del
                      manipulador es aceptable  }
   tvert_obj:arr_tvert_obj; {para traspuesta de los vertices
                             del objeto  }
   ch:char;{para oprimir una tecla}

  (*****************************************************)
   { define una ventana }

   procedure vent(ind:integer;x1,y1,x2,y2:real);
   begin

      definewindow(ind,trunc(xmaxglb*x1/10),
                       trunc(ymaxglb*y1/10),
                  trunc(xmaxglb*x2/10),
                       trunc(ymaxglb*y2/10));
   end;(* vent *)
```

```
{ este procedimiento obtiene la primer traspuesta }

procedure traspu1(at1:arr3_vert_manip_obj;
                  var att1:arr_tvert_obj);
 var
   i,j:integer;
begin

   for i:=1 to 10 do
   for j:=1 to 4 do
     att1[j,i]:=at1[i,j];

end;(* traspu1 *)


{ este procedimiento obtiene la segunda traspuesta }
procedure traspu2(ata:arr_tvert_obj;
                  var atat:arr3_vert_manip_obj);
 var
   i,j:integer;
begin

   for j:=1 to 10 do
   for i:=1 to 4 do
     atat[j,i]:=ata[i,j];

end;(* traspuesta2 *)

{  este  procedimiento multiplica dos matrices  y  da  una
   matriz    resultante }
procedure mult4(t:arr_puntos;att:arr_tvert_obj;
                var ata:arr_tvert_obj);
var
  ii,j,k:integer;
  sigma:real;
begin
   for ii:=1 to 4 do
   begin
     for k:=1 to 10 do
     begin
       sigma:=0.0;
       for j:=1 to 4 do
         sigma:=sigma+t[ii,j]*att[j,k];
       ata[ii,k]:=sigma;
     end;
   end;
end;(* mult4 *)

{ este procedimiento convierte vertices a dos dimensiones
  para poder hacer dibujos }

procedure tres_dos(i:integer;mt:arr3_vert_manip_obj;
                   var md:arr2_vert_manip);
var
```

```
    j:integer;
  begin
     for j:=1 to 10 do
     begin
        if mt[j,1]<>0 then
        begin
           md[i,j,1]:=mt[j,2]-mt[j,1];
           md[i,j,2]:=mt[j,3]-mt[j,1];
        end
        else
          begin
             md[i,j,1]:=mt[j,2];
             md[i,j,2]:=mt[j,3];
          end;
      end;
   end;(* tres_dos *)

   { este procedimiento dibuja una juntura  p .
     como entrada se da el numero de juntura  i,
     asi como los vertices  md  en dos dimensiones      }
   procedure dib_prisma(i:integer;md:arr2_vert_manip);
   begin
      drawline(md[i,1,1],md[i,1,2],md[i,2,1],md[i,2,2]);
      drawline(md[i,1,1],md[i,1,2],md[i,4,1],md[i,4,2]);
      drawline(md[i,2,1],md[i,2,2],md[i,3,1],md[i,3,2]);
      drawline(md[i,3,1],md[i,3,2],md[i,4,1],md[i,4,2]);
      drawline(md[i,4,1],md[i,4,2],md[i,8,1],md[i,8,2]);
      drawline(md[i,1,1],md[i,1,2],md[i,5,1],md[i,5,2]);
      drawline(md[i,2,1],md[i,2,2],md[i,6,1],md[i,6,2]);
      drawline(md[i,3,1],md[i,3,2],md[i,7,1],md[i,7,2]);
      drawline(md[i,5,1],md[i,5,2],md[i,6,1],md[i,6,2]);
      drawline(md[i,5,1],md[i,5,2],md[i,8,1],md[i,8,2]);
      drawline(md[i,6,1],md[i,6,2],md[i,7,1],md[i,7,2]);
      drawline(md[i,7,1],md[i,7,2],md[i,8,1],md[i,8,2]);
   end;(* dib_prisma *)

   (*****************************************************)

  { define las ventanas que se van a usar }
  procedure def_vents;
   begin (* def_vents *)
      { ventana para dibujar el manipulador }
       vent(2,3,0.6,10,7.4);

      { ventana para imprimir las variables del manipulador }
       vent(3,3,9,8.2,10);

      {   ventana para la posicion inicial y final del
          manipulador }
       vent(4,0,7.4,2.8,10);

      { ventanas para prender 6 leds }
        vent(5,0,5,0.9,5.9);
        vent(6,1,5,1.9,5.9);
```

```
      vent(7,2,5,2.9,5.9);
      vent(8,0,6.2,0.9,7.1);
      vent(9,1,6.2,1.9,7.1);
      vent(10,2,6.2,2.9,7.1);

     { ventana para la posicion del manipulador }
      vent(11,0,1.2,2.6,1.7);
end; (* def_vents *)

{ define encabezado para la ventana principal }

procedure def_encab;
begin
  { encabezado para la ventana principal }
    defineheader(1,'SIMULACION GRAFICA DE TRAYECTORIAS');
    setheaderon;
end;(* def_encab *)

{ define un sistema coordenado }

procedure def_sist;
begin
  defineworld(1,-150,-150,150,150);
  selectworld(1);
end;(* def_sist *)

{ presentacion del programa }

procedure presentacion;
begin
   selectwindow(1);
   drawborder;
end;(* presentacion *)

{ obtiene informacion de los objetos que estan en la region
  de trabajo }

procedure obten_objs(objn:reg_obj;
                     var arch_obj,aux_arch_obj:archi_obj);
type
   arr_pos=array[1..3] of real;
   reg2_obj=record
        identif:integer;
        puntos:arr_puntos;
        cx,cy:real;
        pos_inic,pos_fin:arr_pos;
   end;
   arch2_obj=file of reg2_obj;
var
  objetos:arch2_obj; { archivo de objetos creado
                       en la planeacion de
                            trayectoria }
   objn2:reg2_obj;{   registro  de  objetos  usado  en   la
                      planeacion}
```

```
 i,j:integer;

procedure obten_pos(pos_inic,pos_fin:arr_pos;
                    var pos_ini,pos_final:posiciones);

begin
      with pos_ini do
      begin
         px:=pos_inic[1]*sx;
         py:=pos_inic[2]*sy;
         pz:=pos_inic[3]*sz;
      end;
      with pos_final do
      begin
         px:=pos_fin[1]*sx;
         py:=pos_fin[2]*sy;
         pz:=pos_fin[3]*sz;
      end;
end;(* obten_pos *)

procedure obt_transf_obj(ptx,pty:real;var t0:arr_puntos);
begin
   t0[1,1]:=1; t0[1,2]:=0; t0[1,3]:=0; t0[1,4]:=ptx;
   t0[2,1]:=0; t0[2,2]:=1; t0[2,3]:=0; t0[2,4]:=pty;
   t0[3,1]:=0; t0[3,2]:=0; t0[3,3]:=1; t0[3,4]:=0;
   t0[4,1]:=0; t0[4,2]:=0; t0[4,3]:=0; t0[4,4]:=1;
end; (* transfo *)

procedure obt_vert_obj(cx,cy:real;p:arr_puntos;
                       var vo:arr3_vert_manip_obj);
 var
   i:integer;
   va1,va1,va2,va3,va4,va5,va6,va7,va8:real;

begin
  va1:=p[3,1]*sz;  (* multiplica coord.  z (altura)   por
                      factor de escala *)
  va1:=p[1,1]*sx; va2:=p[2,1]*sy;  { multiplica }
  va3:=p[1,2]*sx; va4:=p[2,2]*sy;  { coord. x,y }
  va5:=p[1,3]*sx;   va6:=p[2,3]*sy;   { por factor de
                                        escala }
  va7:=p[1,4]*sx; va8:=p[2,4]*sy;
  { obtiene  8  vert. en coord. homogeanas para los
    objetos }
  vo[1,1]:=va1;  vo[1,2]:=va2;  vo[1,3]:=va1;
  vo[2,1]:=va3;  vo[2,2]:=va4;  vo[2,3]:=va1;
  vo[3,1]:=va5;  vo[3,2]:=va6;  vo[3,3]:=va1;
  vo[4,1]:=va7;  vo[4,2]:=va8;  vo[4,3]:=va1;
  vo[5,1]:=va1;  vo[5,2]:=va2;  vo[5,3]:=0;
  vo[6,1]:=va3;  vo[6,2]:=va4;  vo[6,3]:=0;
  vo[7,1]:=va5;  vo[7,2]:=va6;  vo[7,3]:=0;
  vo[8,1]:=va7;  vo[8,2]:=va8;  vo[8,3]:=0;
  vo[9,1]:=cx;    vo[9,2]:=cy;    vo[9,3]:=va1;
  vo[10,1]:=cx;   vo[10,2]:=cy;   vo[10,3]:=0;
```

```
    for i:=1 to 10 do
       vofi,4]:=1;
 end;(* obt_vert_obj *)

begin (* obten_objs *)
    assign(objetos,'arch_dta.dta');
    assign(arch_obj,'arch_obj.dta');
    assign(aux_arch_obj,'arch_aux.dta');
      reset(objetos);
      rewrite(arch_obj);
      rewrite(aux_arch_obj);
    while not eof(objetos) do
    begin
        read(objetos,objn2);
        with objn,objn2 do
        begin
          {obtiene numero de objeto}
           nobj:=identif;
          {obtiene posicion inicial y final}
           obten_pos(pos_inic,pos_fin,pos_ini,pos_final);
          { inicializa transformacion de objeto }
          obt_transf_obj(-cx*sx,-cy*sy,transf_obj);
          obt_vert_obj(cx*sx,cy*sy,puntos,vert_obj);
          traspu1(vert_obj,tvert_obj);
          mult4(transf_obj,tvert_obj,tvert_obj);
          traspu2(tvert_obj,vert_obj);
          { obtiene la transformacion de objeto que se va a
            usar }
          obt_transf_obj(cx*sx,cy*sy,transf_obj);
        end;
        write(arch_obj,objn);{  escribe  el registro en  el
                                  archivo }
        write(aux_arch_obj,objn);
   end;
      close(objetos);
      close(arch_obj);
      close(aux_arch_obj);
end; (* obten_objs *)

procedure obten_tós(tóa:reg_puntos;tóaux:reg_tó;
                        var tós:tós_arch;
                           var arch_tó:archi_tó);
type
   tós_arch=file of reg_puntos;
var
   tós:tós_arch;{ archivo de tós creado en la planeacion }
   tóa:reg_puntos;{registro de tós usado en la planeacion }
begin
    assign(tós,'punts.dta');
    assign(arch_tó,'arch_tó.dta');
    reset(tós);
    rewrite(arch_tó);
    while not eof(tós) do
    begin
```

```
        read(t6s,t6a);
        with t6aux,t6a do
        begin
          nx:=t6[1,1]; ox:=t6[1,2]; ax:=t6[1,3];
          px:=t6[1,4]*sx;
          ny:=t6[2,1]; oy:=t6[2,2]; ay:=t6[2,3];
          py:=t6[2,4]*sy;
          nz:=t6[3,1]; oz:=t6[3,2]; az:=t6[3,3];
          pz:=t6[3,4]*sz;
         end;
        write(arch_t6,t6aux);
     end;
     close(t6s);close(arch_t6);
 end;(* obten_t6s *)

 { imprime todas las ventanas }
procedure despl_vent;
 var
   i:integer;
begin
   for i:=1 to 11 do
   begin
       selectwindow(i);
       drawborder;
   end;
end;(* despl_vent *)

procedure imp_pos_t6(t6aux:reg_t6);
begin
    { selecciona ventana para imprimir la posicion actual }
    gotoxy(4,3);writeln('POSIC  MANIP');
    selectwindow(11);
    setbackground(0);
    drawborder;
    with t6aux do
    begin
       gotoxy(2,4);write(px:4:1,' ',py:4:1,' ',pz:4:1);
    end;
end;(* imp_pos_t6 *)

procedure imp_obj_pos(objn:reg_obj);
begin
     gotoxy(4,6);write('POSIC OBJETO ',objn.nobj);
     with objn,pos_ini do
     begin
        gotoxy(3,7);write('pos inic');
        gotoxy(3,8);write(px:5:1,py:5:1,pz:5:1);
     end;
     with objn,pos_final do
     begin
        gotoxy(3,9);write('pos fin');
        gotoxy(3,10);write(px:5:1,py:5:1,pz:5:1);
     end;
end;(* imp_obj_pos *)
```

```
{ genera la imagen del manipulador y objetos }

procedure genera_imagen(t6aux:reg_t6;objn:reg_obj;
                        cont_fig:integer;angulos:reg_angs;
                        var detecta_obj:boolean;
                        var aux_arch_obj:archi_obj);

 const
    pi=3.141592654;
 type
    arr=array[0..6,1..4,1..4] of real;
 var
  aux_objn:reg_obj; { variable auxiliar para los objetos }
  {para  vertices  en tres dimensiones de los  enlaces  del
   manipulador}.
  at,bt,ct,dt,et,ft,gt,atat:arr3_vert_manip_obj;
  ad:arr2_vert_manip;{ vertices en dos dimensiones
                       para hacer el dibujo }
  t:arr;   { matrices de transformacion ti }
  att,ata:arr_tvert_obj;{ matriz traspuesta }
  i,j:integer;
  ran1,ran2,ran3,ran4,ran5,ran6:real; { rangos }

 {   este   procedimiento inicializa los eslabones  del
     manipulador con vertices en coordenadas homogeneas en
     tres dimensiones }

 procedure manip_inic(var at,bt,ct,dt,et.
                       ft,gt:arr3_vert_manip_obj);
  var
     i:integer;
     lx,ly,lz:real;
 begin
    { enlace 1 }
    lx:=12;{ longitud en  x }
    ly:=9; { longitud en y }
    lz:=100;{ longitud en z }
    at[1,1]:=lx;  at[1,2]:=-ly; at[1,3]:=-10;
    at[2,1]:=lx;  at[2,2]:=ly;  at[2,3]:=-10;
    at[3,1]:=-lx; at[3,2]:=ly;  at[3,3]:=-10;
    at[4,1]:=-lx; at[4,2]:=-ly; at[4,3]:=-10;
    at[5,1]:=lx;  at[5,2]:=-ly; at[5,3]:=-lz;
    at[6,1]:=lx;  at[6,2]:=ly;  at[6,3]:=-lz;
    at[7,1]:=-lx; at[7,2]:=ly;  at[7,3]:=-lz;
    at[8,1]:=-lx; at[8,2]:=-ly; at[8,3]:=-lz;
    { centro superior }
    at[9,1]:=0; at[9,2]:=0; at[9,3]:=-10;
    { centro inferior }
    at[10,1]:=0; at[10,2]:=0; at[10,3]:=-lz;

    { enlace 2 }
    lx:=9;{ longitud en  x }
    ly:=9; { longitud en y }
    lz:=20;{ longitud en z }
```

```
bt[1,1]:=lx;  bt[1,2]:=-ly; bt[1,3]:=lz;
bt[2,1]:=lx;  bt[2,2]:=ly;  bt[2,3]:=lz;
bt[3,1]:=-lx; bt[3,2]:=ly;  bt[3,3]:=lz;
bt[4,1]:=-lx; bt[4,2]:=-ly; bt[4,3]:=lz;
bt[5,1]:=lx;  bt[5,2]:=-ly; bt[5,3]:=-lz;
bt[6,1]:=lx;  bt[6,2]:=ly;  bt[6,3]:=-lz;
bt[7,1]:=-lx; bt[7,2]:=ly;  bt[7,3]:=-lz;
bt[8,1]:=-lx; bt[8,2]:=-ly; bt[8,3]:=-lz;
bt[9,1]:=0;  bt[9,2]:=0;  bt[9,3]:=lz;
bt[10,1]:=0; bt[10,2]:=0; bt[10,3]:=-lz;

{ enlace 3 }
lx:=12;{ longitud en  x }
ly:=12; { longitud en y }
lz:=40;{ longitud en z }
ct[1,1]:=lx;  ct[1,2]:=-ly; ct[1,3]:=0;
ct[2,1]:=lx;  ct[2,2]:=ly;  ct[2,3]:=0;
ct[3,1]:=-lx; ct[3,2]:=ly;  ct[3,3]:=0;
ct[4,1]:=-lx; ct[4,2]:=-ly; ct[4,3]:=0;
ct[5,1]:=lx;  ct[5,2]:=-ly; ct[5,3]:=-lz;
ct[6,1]:=lx;  ct[6,2]:=ly;  ct[6,3]:=-lz;
ct[7,1]:=-lx; ct[7,2]:=ly;  ct[7,3]:=-lz;
ct[8,1]:=-lx; ct[8,2]:=-ly; ct[8,3]:=-lz;

{ complemento del enlace 3 }
lx:=7;{ longitud en  x }
ly:=7; { longitud en y }
lz:=135;{ longitud en z }
dt[1,1]:=lx;  dt[1,2]:=-ly; dt[1,3]:=0;
dt[2,1]:=lx;  dt[2,2]:=ly;  dt[2,3]:=0;
dt[3,1]:=-lx; dt[3,2]:=ly;  dt[3,3]:=0;
dt[4,1]:=-lx; dt[4,2]:=-ly; dt[4,3]:=0;
dt[5,1]:=lx;  dt[5,2]:=-ly; dt[5,3]:=-lz;
dt[6,1]:=lx;  dt[6,2]:=ly;  dt[6,3]:=-lz;
dt[7,1]:=-lx; dt[7,2]:=ly;  dt[7,3]:=-lz;
dt[8,1]:=-lx; dt[8,2]:=-ly; dt[8,3]:=-lz;

{ enlace 4 }
lx:=6;{ longitud en  x }
ly:=6; { longitud en y }
lz:=10;{ longitud en z }
et[1,1]:=lx;  et[1,2]:=-ly; et[1,3]:=lz;
et[2,1]:=lx;  et[2,2]:=ly;  et[2,3]:=lz;
et[3,1]:=-lx; et[3,2]:=ly;  et[3,3]:=lz;
et[4,1]:=-lx; et[4,2]:=-ly; et[4,3]:=lz;
et[5,1]:=lx;  et[5,2]:=-ly; et[5,3]:=-lz;
et[6,1]:=lx;  et[6,2]:=ly;  et[6,3]:=-lz;
et[7,1]:=-lx; et[7,2]:=ly;  et[7,3]:=-lz;
et[8,1]:=-lx; et[8,2]:=-ly; et[8,3]:=-lz;
et[9,1]:=0;  et[9,2]:=0;  et[9,3]:=lz;
et[10,1]:=0; et[10,2]:=0; et[10,3]:=-lz;

{ enlace 5 }
lx:=4;{ longitud en  x }
```

```
    ly:=4; { longitud en y }
    lz:=5;{ longitud en z }
    ft[1,1]:=lx;  ft[1,2]:=-ly; ft[1,3]:=25;
    ft[2,1]:=lx;  ft[2,2]:=ly;  ft[2,3]:=25;
    ft[3,1]:=-lx; ft[3,2]:=ly;  ft[3,3]:=25;
    ft[4,1]:=-lx; ft[4,2]:=-ly; ft[4,3]:=25;
    ft[5,1]:=lx;  ft[5,2]:=-ly; ft[5,3]:=-lz;
    ft[6,1]:=lx;  ft[6,2]:=ly;  ft[6,3]:=-lz;
    ft[7,1]:=-lx; ft[7,2]:=ly;  ft[7,3]:=-lz;
    ft[8,1]:=-lx; ft[8,2]:=-ly; ft[8,3]:=-lz;
    ft[9,1]:=0;  ft[9,2]:=0;  ft[9,3]:=25;
    ft[10,1]:=0; ft[10,2]:=0; ft[10,3]:=-lz;

    { enlace 6 }
    lx:=12;{ longitud en  x }
    ly:=12; { longitud en y }
    lz:=20;{ longitud en z }
    gt[1,1]:=lx;  gt[1,2]:=-ly; gt[1,3]:=lz;
    gt[2,1]:=lx;  gt[2,2]:=ly;  gt[2,3]:=lz;
    gt[3,1]:=-lx; gt[3,2]:=ly;  gt[3,3]:=lz;
    gt[4,1]:=-lx; gt[4,2]:=-ly; gt[4,3]:=lz;
    gt[5,1]:=lx;  gt[5,2]:=-ly; gt[5,3]:=-lz;
    gt[6,1]:=lx;  gt[6,2]:=ly;  gt[6,3]:=-lz;
    gt[7,1]:=-lx; gt[7,2]:=ly;  gt[7,3]:=-lz;
    gt[8,1]:=-lx; gt[8,2]:=-ly; gt[8,3]:=-lz;
    gt[9,1]:=0;  gt[9,2]:=0; gt[9,3]:=lz;
    gt[10,1]:=0; gt[10,2]:=0; gt[10,3]:=-lz;

    { hace que los vertices esten en coordenadas homogeneas }
    for i:=1 to 10 do
    begin
     at[i,4]:=1;
     bt[i,4]:=1;
     ct[i,4]:=1;
     dt[i,4]:=1;
     et[i,4]:=1;
     ft[i,4]:=1;
     gt[i,4]:=1;
    end;
end;(* manip_inic *)

  { este procedimiento calcula las matrices de
    transformacion que se aplicaran a los enlaces del
    manipulador y asi obtener la nueva posicion de cada
    enlace }

procedure mat_ti(angulos:reg_angs;var t:arr);
type
  arr9=array[1..6] of real;
var
  a:arr;
  teti,tetai,alfi,alfai,di,aai:real;
  i,j,k:integer;
  dii,alfaii,aaii,tetaii:arr9;
```

```
{  este procedimiento obtiene los parametros para  poder
 obtener las matrices de transformacion  Ai }

procedure    parametros(angulos:reg_angs;
                        var dii,alfaii,aaii,
                                    tetaii:arr9);
var
  i:integer;
begin
   (* tabla de parametros para el stanford *)
   (* valores de alfai *)
   alfaii[1]:=-90;
   alfaii[2]:=90;
   alfaii[3]:=0;
   alfaii[4]:=-90;
   alfaii[5]:=90;
   alfaii[6]:=0;

   (* valores de ai *)
   for i:=1 to 6 do
     aaii[i]:=0;

   (* valores de di *)
   dii[1]:=0;
   dii[2]:=30;
   dii[3]:=angulos.d3;
   for i:=4 to 6 do
     dii[i]:=0;

   (* valores de tetai *)
   with angulos do
   begin
      dii[3]:=d3;
      tetaii[1]:=t1;
      tetaii[2]:=t2;
      tetaii[3]:=0;
      tetaii[4]:=t4;
      tetaii[5]:=t5;
      tetaii[6]:=t6;
   end;
end;(* parametros *)

{ este procedimiento convierte los angulos en radianes }

procedure grad_rad(teti:real;var teta:real);
begin
   teta:= (teti*pi)/180;
end;(* grad_rad *)

{ este procedimiento calcula las matrices Ai }

procedure    mat_ai(di,alfai,aai,tetai:real;i:integer;var
                a:arr);
```

```
var
   n1,n2,n3,n4:real;
begin
   n1:=cos(tetai);n2:=sin(tetai);
   n3:=cos(alfai);n4:=sin(alfai);
   a[i,1,1]:=n1;a[i,1,2]:=-n2*n3;
   a[i,1,3]:=n2*n4;a[i,1,4]:=aai*n1;
   a[i,2,1]:=n2;a[i,2,2]:=n1*n3;
   a[i,2,3]:=-n1*n4;a[i,2,4]:=aai*n2;
   a[i,3,1]:=0;a[i,3,2]:=n4;a[i,3,3]:=n3;a[i,3,4]:=di;
   a[i,4,1]:=0;a[i,4,2]:=0;a[i,4,3]:=0;a[i,4,4]:=1;
end;(* mat_ai *)

{ este procedimiento calcula las matrices  Ti }

procedure mats_ti(i:integer;a:arr;var t:arr);
 var
  ii,j,k:integer;
  sigma:real;
begin
   for ii:=1 to 4 do
   begin
      for k:=1 to 4 do
      begin
         sigma:=0.0;
         for j:=1 to 4 do
           sigma:=sigma+t[i-1,ii,j]*a[i,j,k];
         t[i,ii,k]:=sigma;
      end;
   end;
end;(* mats_ti *)


begin (* mat_ti *)
   (* inicia algoritmo para obtener Ai y Ti *)

   i:=0;
   (* matriz identidad *)
   for j:=1 to 4 do
   begin
      for k:=1 to 4 do
        t[i,j,k]:=0;
      t[i,j,j]:=1;
   end;

   { obtiene los parametros del manipulador }
   parametros(angulos,dii,alfaii,aaii,tetaii);

   while i<6 do
   begin
      i:=i+1;
      di:=dii[i];alfi:=alfaii[i];
      aai:=aaii[i];teti:=tetaii[i];
```

```
        { convierte grados a radianes }
        grad_rad(teti,tetai); grad_rad(alfi,alfai);

        (* calcula matrices de enlace Ai *)
        mat_ai(di,alfai,aai,tetai,i,a);

        (* obtiene matrices de transformacion Ti *)
        mats_ti(i,a,t);
     end;
  end;(* mat_ti *)

  { multiplica matriz do matices y nos da una resultante }

  procedure mult2(i:integer;t:arr;att:arr_tvert_obj;
                  var ata:arr_tvert_obj);
  var
    ii,j,k:integer;
    sigma:real;
  begin
     for ii:=1 to 4 do
     begin
        for k:=1 to 10 do
        begin
           sigma:=0.0;
           for j:=1 to 4 do
             sigma:=sigma+t[i,ii,j]*att[j,k];
           ata[ii,k]:=sigma;
        end;
     end;
  end;(* mult2 *)


  { este procedimiento dibuja una juntura  i ;
    como entrada se da el numero de juntura  i,
    asi como los vertices de la juntura md  en
    dos dimensiones      }

  procedure dib_cilind(i:integer;md:arr2_vert_manip);
   var
     eje_a,eje_b,xcentro,xcentro2,ycentro,ycentro2:real;
     ch:char;

   { calcula la longitud del eje mayor o eje menor }

   procedure dist(px1,py1,px2,py2:real;var eje:real);
   begin
      eje:=sqrt(sqr(abs(px2)-abs(px1))+sqr(abs(py2)-
       abs(py1)));
   end;(* dist *)

   { dibuja los circulos superior e inferior del cilindro }

   procedure dib_cir(x,y,rad_aux,asp_aux:real);
     var
```

```
        i:integer;
        rad,asp,f2,n1:real;
        resp:char;
  begin
     rad:=abs((1.5*rad_aux)/94);
     if (rad_aux>=11)and(rad_aux<13)  then
       n1:=9
     else
      if (rad_aux>=13)and(rad_aux<16) then
        n1:=10
      else
       if rad_aux<11 then
       begin
         n1:=8;
         rad_aux:=10;
       end
       else
        if rad_aux=8 then
          n1:=7
        else
          n1:=11;
     f2:=abs(rad_aux/(abs(rad_aux)-n1));
     asp:=abs((asp_aux*f2)/65);
     for i:=1 to 1 do                    {draw circles}
     begin
       setaspect(asp);drawcircle(x,y,rad);
     end;

   end;(* dib_cir *)
  begin
     xcentro:=md[i,9,1];ycentro:=md[i,9,2];
     dist(md[i,9,1],md[i,9,2],md[i,3,1],md[i,3,2],eje_a);
     dist(md[i,9,1],md[i,9,2],md[i,4,1],md[i,4,2],eje_b);
     dib_cir(xcentro,ycentro,eje_a,eje_b);

     xcentro2:=md[i,10,1];ycentro2:=md[i,10,2];
     dib_cir(xcentro2,ycentro2,eje_a,eje_b);

     { completa el cilindro }
     drawline(xcentro+eje_a,ycentro,xcentro2+eje_a,ycentro2);
     drawline(xcentro-eje_a,ycentro,xcentro2-eje_a,ycentro2);
  end;(* dib_cilind *)

  { dibuja el manipulador }

  procedure graf_manip(at,bt,ct,dt,
                       et,ft,gt:arr3_vert_manip_obj);
  begin
     i:=0;
     tres_dos(i,at,ad);{ convierte los vertices de la base
                         del manipulador a dos dimensiones }
     while i<6 do
     begin
        i:=i+1;
```

```
        case i of
          1:begin
              traspu1(bt,att);
            end;

          2:begin
              traspu1(ct,att);
            end;

          3:begin
              traspu1(dt,att);
            end;

          4:begin
              traspu1(et,att);
            end;

          5:begin
              traspu1(ft,att);
            end;

          6:begin
              traspu1(gt,att);
            end;
        end;(* end case *)
        mult2(i,t,att,ata);
        traspu2(ata,atat);
        tres_dos(i,atat,ad);
    end;(* end while *)

    { dibuja las junturas del manipulador }
     dib_cilind(0,ad);
     dib_cilind(1,ad);
     dib_prisma(2,ad);
     dib_prisma(3,ad);
{      dib_cilind(4,ad);
      dib_cilind(5,ad);
      dib_prisma(6,ad); }

end; (* graf_manip *)

{ checa si el manipulador detecto el objeto que va a
  manipular. Si ocurre esto, entonces, se obtendra la
  transformacion que se va aplicar al objeto y se
  actualizara la ubicacion del objeto }

procedure checa_obj(objn:reg2_obj  ;var checa_obj:boolean;
                    var aux_arch_obj:archi_obj);

 { obtiene la transformacion que se va aplicar al objeto
   que se esta manipulando }

 procedure asig_transf(t6aux:reg_t6;var tr_ob:arr_puntos);
 begin
```

```
    with t6aux do
    begin
       tr_ob[1,1]:=nx; tr_ob[2,1]:=ny; tr_ob[3,1]:=nz;
       tr_ob[1,2]:=ox; tr_ob[2,2]:=oy; tr_ob[3,2]:=oz;
       tr_ob[1,3]:=ax; tr_ob[2,3]:=ay; tr_ob[3,3]:=az;
       tr_ob[1,4]:=px; tr_ob[2,4]:=py; tr_ob[3,4]:=pz;
       tr_ob[4,1]:=0;  tr_ob[4,2]:=0;  tr_ob[4,3]:=0;
                tr_ob[4,4]:=1;
    end;
  end;(* asig_transf *)

 { actualiza la ubicacion de todos los objetos }

 procedure actualiz_objs(auxi_objn:reg_obj;
                         var aux_arch_obj:archi_obj);
 var
  aux2_arch_obj:archi_obj;{  archivo  auxiliar  para  hacer
                             cambios a los objetos }
  aux2_objn:reg_obj; { variable auxiliar para los objetos }
 begin
    assign(aux2_arch_obj,'arch2_aux.dta');
    reset(aux_arch_obj);
    rewrite(aux2_arch_obj);
    while not eof(aux_arch_obj) do
    begin
       read(aux_arch_obj,aux2_objn);
       if aux2_objn.nobj=auxi_objn.nobj then
          write(aux2_arch_obj,auxi_objn)
       else
          write(aux2_arch_obj,aux2_objn);
     end;
     close(aux_arch_obj);
     close(aux2_arch_obj);
     { actualiza }
     rewrite(aux_arch_obj);
     reset(aux2_arch_obj);
     while not eof(aux2_arch_obj) do
     begin
        read(aux2_arch_obj,aux2_objn);
        write(aux_arch_obj,aux2_objn);
     end;
     close(aux_arch_obj);
     close(aux2_arch_obj);
 end;(* actualiz_objs *)

begin
 {prepara trasformacion para aplicar al objeto}
   with objn,pos_ini do{ rango de posicion inical }
   begin
      ran1:=px-rax; ran2:=px+rax;
      ran3:=py-ray; ran4:=py+ray;
      ran5:=pz-raz; ran6:=pz+raz;
   end;
   if detecta_obj=true then
```

```
    begin
       asig_transf(t6aux,objn.transf_obj); { iguala
                                             transf_obj a
                                               t6aux para
                                     aplicarse al objeto }
       actualiz_objs(objn,aux_arch_obj);{ actualiza archivo
                                            de
                                          objetos }
    end;

    { pregunta si el manipulador llego a la posicion
      inicial del objeto }
      if   (((t6aux.px>=ran1)and(t6aux.px<=ran2))
           and ((t6aux.py>=ran3)and(t6aux.py<=ran4))
               and     ((t6aux.pz>=ran5)and(t6aux.pz<=ran6)))
    then
     begin
        gotoxy(2,23);write('detecta obj ',objn.nobj);
        detecta_obj:=true; { detecta objeto }
     end;

end; (* checa_obj *)

{ dibuja todos los objetos en su nueva ubicacion }

procedure graf_objs(aux_arch_obj:archi_obj);
begin
   { dibuja los objetos }
   reset(aux_arch_obj);
   while not eof(aux_arch_obj) do
   begin
      selectwindow(2);
      drawborder;
      read(aux_arch_obj,aux_objn);
      with aux_objn do
      begin
          traspu1(vert_obj,att);
          mult4(transf_obj,att,ata);
          traspu2(ata,atat);
          tres_dos(0,atat,ad);
          dib_prisma(0,ad);
       end;
    end;
    close(aux_arch_obj);
end;(* graf_objs *)

{ almacena la imagen del manipulador y objetos en memoria
  para su posterior uso en el efecto de animacion }
procedure almacena_en_memoria(cont_fig:integer);
begin
  vent(cont_fig,3,0.6,10,7.4); { define ventana }
  storewindow(cont_fig); {almacena la ventana}
  gotoxy(40,18);write('almacenando imagen ',cont_fig);
end;(* almacena_en_memoria *)
```

```
begin (* genera_imagen *)
   i:=0;
   selectwindow(2);{ selecciona ventana para el manipulador}
   setbackground(0);
   drawborder;
     { analiza manipulador}
   manip_inic(at,bt,ct,dt,et,ft,gt);{ inicializa enlaces }
   mat_ti(angulos,t);{ obtiene matrices Ti }
   graf(at,bt,ct,dt,et,ft,gt);

   {analiza objetos }
    checa_obj(objn,detecta_obj,aux_arch_obj);
    graf_objs(aux_arch_obj);

   {almacena los graficos en la memoria de la maquina }
   almacena_en_memoria(cont_fig);
end;(* genera_imagen *)


{calcula  los  valores  de  las variables del manipulador }

procedure calc_ang(t6aux:reg_t6;var err_ang:boolean;
                   var angulos:reg_angs);
      (* procedimiento  que  da  un  valor  minimo  a  las
variables = 0 *)

      { asigna un valor minimo a la matriz T6 para evitar
        divisiones entre cero }

      procedure vminimo(var t6:reg_t6);
      var
         vmin:real;
      begin
         with t6 do
         begin
            vmin:=0.00001;
            if nx=0 then nx:=vmin;
            if ny=0 then ny:=vmin;
            if nz=0 then nz:=vmin;
            if ox=0 then ox:=vmin;
            if oy=0 then oy:=vmin;
            if oz=0 then oz:=vmin;
            if ax=0 then ax:=vmin;
            if ay=0 then ay:=vmin;
            if az=0 then az:=vmin;
            if px=0 then px:=vmin;
            if py=0 then py:=vmin;
            if pz=0 then pz:=vmin;
         end;(* with *)
      end;(* vminimo *)

      (* procedimiento que calcula las variables de juntura
         usando las ecuaciones que se vieron en el capitulo
         de cinematica *)
```

```
procedure variables(t6:reg_t6;var estado:reg_angs);
var
   s4,c4,s5,c5,s6,c6,r,cuad:real;
begin
   with t6,estado do
   begin
      r:=sqrt(sqr(px)+sqr(py));
      if r = d2 then r:=r +0.00000000001;
      cuad:=sqr(r)-sqr(d2);
      if cuad<=0 then cuad:=0.00000000001;
      t1:=arctan(py/px)-arctan(d2/(sqrt(cuad)));
      t2:=arctan((cos(t1)*px+sin(t1)*py)/pz);
      d3:=sin(t2)*(cos(t1)*px+sin(t1)*py)+cos(t2)*pz;
      s4:=-sin(t1)*ax+sin(t1)*ay;
      c4:=cos(t2)*(cos(t1)*ax+sin(t1)*ay)-sin(t2)*az;
      t4:=arctan(s4/c4);
      s5:=cos(t4)*(cos(t2)*(cos(t1)*ax+sin(t1)*ay)-
          sin(t2)*az)+
          sin(t4)*(-sin(t1)*ax+cos(t1)*ay);
      c5:=sin(t2)*(cos(t1)*ax+sin(t1)*ay)+cos(t2)*az;
      t5:=arctan(s5/c5);
      s6:=-cos(t5)*(cos(t4)*(cos(t2)*(cos(t1)*
            ox+sin(t1)*oy)-
          sin(t2)*oz)+sin(t4)*(-sin(t1)*
            ox+cos(t1)*oy))+
          sin(t5)*(sin(t2)*(cos(t1)*ox+sin(t1)*oy)
            +cos(t2)*oz);
    c6:=-sin(t4)*(cos(t2)*(cos(t1)*ox+sin(t1)*oy)-
          sin(t2)*oz)+
          cos(t4)*(-sin(t1)*ox+cos(t1)*oy);
      t6:=arctan(s6/c6);
      if t5<0 then t4:=t4+pi;
   end;(* with *)
end;(* variables *)

{ convierte radianes a grados }

procedure rad_grad(var angulos:reg_angs);
  var
    f:real;
begin
    f:=180/pi;
    with angulos do
    begin
      t1:=t1*f;
      t2:=t2*f;
      d3:=d3;
      t4:=t4*f;
      t5:=t5*f;
      t6:=t6*f;
    end;
end;(* rad_grad *)
```

```
 (* este procedimiento imprime los angulos y
    traslaciones *)

procedure imprime_angs(estado:reg_angs);
begin
   with estado do
   begin
      selectwindow(3);{    selecciona    ventana    para
      imprimir las variables del manipulador}
      setbackground(0);
      setbackground(1);
      drawborder;
      gotoxy(26,20);      writeln('valores    de    las
                              variables');
      gotoxy(26,21);writeln('del manipulador:');
      gotoxy(26,22);
      write('  ',chr(233),'1    ',chr(233),'2    ',
                        'd3    ',chr(233),'4     ');
      write(chr(233),'5     ',chr(233),'6');
      gotoxy(26,24);
      write(t1:4:2,' ',t2:4:2,' ',d3:4:2,' ',
            t4:4:2,' ',t5:4:2);
      write(' ',t6:4:2);
    end;(* with *)
end;(* imprime_angs *)

{  en  esta  parte  se detecta  si  alguna  parte  de
   manipulador esta fuera de rango.
   en  caso  de  que se detecte un  error,entonces  se
   prendera un led indicador de error. }

procedure checa_ang(angulos:reg_angs;
 var err_ang:boolean);

  { activa una ventana }

  procedure prende_led(i:integer;var err_ang:boolean);
  begin

    err_ang:=true;
    selectwindow(i);
    setbackground(30);
    drawb_rder;

  end;(* prende_led *)

  { pone el numero de eslabon en una ventana }

  procedure imp_num_enlace;
  begin

      gotoxy(6,12);writeln('leds de error');
      gotoxy(5,14);writeln(1);
      gotoxy(12,14);writeln(2);
```

```
          gotoxy(20,14);writeln(3);
          gotoxy(5,17);writeln(4);
          gotoxy(12,17);writeln(5);
          gotoxy(20,17);writeln(6);
       end;(* imp_num_enlace *)

     begin (* checa_ang *)
       with angulos do
       begin
          if (t1<-360)or(t1>90) then
              prende_led(5,err_ang)
          else
            if (t2<-90)or(t2>90) then
                prende_led(6,err_ang)
            else
              if (d3<0)or(d3>200) then
                  prende_led(7,err_ang)
              else
                if (t4<-360)or(t4>360) then
                    prende_led(8,err_ang)
                else
                  if (t5<-180)or(t5>180) then
                      prende_led(9,err_ang)
                  else

                    if (t6<-360)or(t6>360) then
                        prende_led(10,err_ang);
         end;
         imp_num_enlace;
     end;(* checa_ang *)

  begin (* calc_ang *)
       vminimo(t6aux);
       err_ang:=false;{ inicializa variable para  detectar
                        si  es que hay algun error en los
                        valores  de  los  grados de
                        libertad del  manipulador }

       variables(t6aux,angulos);{  calcula los valores  de
              los grados de libertad del manipulador }
       rad_grad(angulos);{ convierte a radianes }
       imprime_angs(angulos);{ imprime }
       checa_ang(angulos,err_ang);{  checa  que  no   haya
                                      error};
end;(* calc_ang *)

    {  dibuja el manipulador en movimiento para efectuar su
       tarea, y ambien grafica las trayectorias que se
       generaron}

procedure imp_simul(error_ang:boolean;cont_fig:integer;
                    var aux_arch_obj:archi_obj;
                    var arch_t6:archi_t6);
 var
```

```
t6aux:reg_t6;
ch:char;
i:integer;

{ despliega menu de simulacion }

procedure menu(var ch:char);
begin
  gotoxy(24,20);
  write('esc :salir    t :trayect     m :manip');
  read(kbd,ch);
end;(* menu *)

{ dibuja las trayectorias que seguira el robot para
  manipular objetos .  Tambien se dibujaran los
  objetos en su posicion inicial y final }

procedure   dib_tray(t6aux:reg_t6;var
          t_obj:reg_i_obj; var arch_t6:archi_t6);
 var
   file_obj:archi_obj;
   ad2:arr2_vert_manip;

  { mapea de tres a dos dimensiones }

  procedure tres2_dos(i:integer;t6aux:reg_t6;
                      var md:arr2_vert_manip);
  var
     j:integer;
  begin
    with t6aux do
    if px<>0 then
    begin
      md[i,1,1]:=py-px;
      md[i,1,2]:=pz-px;
    end
    else
    begin
      md[i,1,1]:=py;
      md[i,1,2]:=pz;
    end;
  end;(* tres2_dos *)

  { dibuja los ejes del sistema tridimensional }

  procedure dib_ejes;
  begin
    drawline(0,0,-500,-500);(* eje x *)
    drawline(0,0,500,0);(* eje y *)
    drawline(0,0,0,500);(* eje z *)
  end;(* dib_ejes *)

 { dibuja los objetos }
 procedure dibuj_objs(var aux_arch_obj:archi_obj);
```

```
 var
   ad2:arr2_vert_manip;
   att,ata:arr_tvert_obj;{ matriz traspuesta }
   atat:arr3_vert_manip_obj;
   aux_objn:reg_obj;
 begin
   reset(aux_arch_obj);
   while not eof(aux_arch_obj) do
   begin
     selectwindow(2);
     drawborder;
     read(aux_arch_obj,aux_objn);
     with aux_objn do
     begin
       traspu1(vert_obj,att);
       mult4(transf_obj,att,ata);
       traspu2(ata,atat);
       tres_dos(0,atat,ad2);
       dib_prisma(0,ad2);
     end;
   end;
   close(aux_arch_obj);
 end;{ dibuj_objs }

begin  (* dib_tray *)
  setbackground(0); { limpia      }
  drawborder;       {    ventana  }
  dib_ejes;

  { dibuja los objetos sin manipular }
  assign(file_obj,'arch_obj.dta');
  dibuj_objs(file_obj);
  {dibuja la trayectoria en una ventana}
  reset(arch_t6);
  while not eof(arch_t6) do
  begin
     selectwindow(2);
     drawborder;
    {  obtiene  una nueva matriz t6 }
    read(arch_t6,t6aux);
    with t6aux do
    begin
      gotoxy(60,6);write(px:5:1,py:5:1,pz:5:1);
      read(kbd,ch);
    end;
    { para graficar trayectoría }
       tres2_dos(0,t6aux,ad2);
       drawterm tw(ad2[0,1,1],ad2[0,1,2],1,'.');
  end;(* while *)
  close(arch_t6);

  { dibuja objetos ya manipulados }
  dibuj_objs(aux_arch_obj);
end; (* dib_tray *)
```

```
      { se efectua la animacion, es decir, se ve el
        movimiento del manipulador para realizar su
        tarea }

      procedure dib_manipu(cont_fig:integer);
      var
        ret,delta,ind_imag:integer;
      begin
        ret:=1;  {retardo para controlar la velocidad del
                  manipulador}
        setbackground(0); { limpia      }
        drawborder;       {    ventana }
        while ret <> 0 do
         begin
           gotoxy(40,17);write(chr(26),' dar retardo:');
           gotoxy(54,17);read(ret);
           if ret>=0 then
           for ind_imag:=12 to cont_fig do
           begin
             delay(ret);
             restorewindow(ind_imag,0,0);
           end
           else
           begin
             restorewindow(cont_fig,0,0);
           end;
         end;
       end; { dib_manipu }

begin (* imp_simul *)
     if err_ang=false then
     begin
       clearscreen;
       selectwindow(2);
       setbackground(0); { limpia          }
       drawborder;       {    ventana     }
       gotoxy(30,3);write(chr(2),'              iniciando
                                  simulacion');
       repeat
          menu(ch);
         case ord(ch) of
         27:begin { aviso de salida }
              for i:=1 to 30 do
              begin
                sound(800);
                sound(80);
              end;
              nosound;
            end;

          84,116:dib_tray(t6aux,aux_arch_obj,arch_t6);
                                  { elige
                            dibujo de la trayectoria}
```

```
            77,109:dib_manipu(cont_fig); { elige movimiento
                                           del
                                            manipulador }

             else
                begin
                  for i:=1 to 10 do
                   sound(400); { avisa que hay error }
                  nosound;
                end;
            end;(* case *)
          until ord(ch)=27;
        end; { if }
        clearscreen;{ limpia pantalla }
    end;(* imp_simul *)

begin(* manipula *)
  def_vents;{ define la ventanas que se usaran}

  def_encab;{ define encabezado para la ventana principal }

  presentacion;{ presentacion del programa }

  def_sist;{ define sistema coordenado a usar }

  rad:=1.5;                  { incicializa variables     }
                             {   para dibujar            }
  aspect_loc:=getaspect;     {   circulos o elipses      }

  (* inicia simulacion *)

  gotoxy(30,15);write('esperar un momento.....');

  { obtiene informacion de los objetos a usar }
  obten_objs(objn,arch_obj,aux_arch_obj);
  gotoxy(30,15);write(' esperar un momento.....');

  { obtiene informacion de las trayectorias (matrices T6) }
  obten_t6s(t6aux,arch_t6);

  err_ang:=false;
  reset(arch_obj);  { nos coloca al inico del archivo de
                      objetos }
  reset(arch_t6);   { al inicio del archivo de matrices t6 }
  assign(arch_g_t6,'arch_g');

  gotoxy(30,15);write('oprimir una tecla....');
  read(kbd,ch);

  {'incia algoritmo de simulacion grafica }

  repeat { objetos }
     read(arch_obj,objn); { lee objeto }
     rewrite(arch_g_t6);{nos coloca al inicio para grabar la
```

```
                        trayectoria  del objeto que  se
                        esta manipulando }

    despl_vent;{ despliega todas las ventanas }

    imp_obj_pos(objn);{ imprime posicion del objeto }

    (* determina rangos de la posicion final del objeto *)
    with objn,pos_final do
    begin
      rang1:=px-rax; rang2:=px+rax;
      rang3:=py-ray; rang4:=py+ray;
      rang5:=pz-raz; rang6:=pz+raz;
    end;

    detecta_obj:=false; { no detecta objeto }
    cont_fig:=11; {inicializa contador de imagenes }
    repeat  { matrices t6 }
       read(arch_t6,t6aux); {lee la matriz t6}
       write(arch_g_t6,t6aux);   {  graba  punto  de  la
       trayectoria }
       cont_fig:=cont_fig+1;{  contador  para el numero  de
                                    imagenes
                              que se van a generar en la
                               simulacion}

       imp_pos_t6(t6aux);   {  imprime la posicion contenida
                              en la matriz t6}

       { calcula las variables del manipulador y nos  dice
          si alguna variable esta fuera de rango}
       calc_ang(t6aux,err_ang,angulos);

       if err_ang=false then  {si no hay error }
       { genera las imagenes del manipulador y objetos y
         las guarda en memoria }
       genera_imagen(t6aux,objn,cont_fig,angulos,
                     detecta_obj,aux_arch_obj);

    until (err_ang=true) or (cont_fig=61) or (eof(arch_t6))
          or  ((detecta_obj=true)
                 and
                ((t6aux.px>=rang1)and(t6aux.px<=rang2))
                  and ((t6aux.py>=rang3)
                  and(t6aux.py<=rang4))
                  and ((t6aux.pz>=rang5)
                   and(t6aux.pz<=rang6)));

    close(arch_g_t6);

    { despliega la simulacion, es decir dibuja las imagenes
      contenidas  en  memoria  para  dar  el  efecto  de
      animacion }
    imp_simul(err_ang,cont_fig,aux_arch_obj,arch_g_t6);
```

```
    until (eof(arch_obj)) or (err_ang=true) ;

    close(arch_obj);
    close(arch_t6);
    setaspect(aspect_loc);  { variable para dibujar circulos o
                              elipses}

    (* termina simulacion *)
end;(* manipula *)

{------ PROGRAMA PRINCIPAL -----}

begin (* manipuladores *)
  initgraphic;{ pasa a modo grafico }
  manipula;{ efectua simulacion grafica }
  leavegraphic;{ sale de modo grafico }
end (* manipuladores *).
```

```
(*************************************************************)
(*                                                           *)
(*                      ** TURMOV **                         *)
(*                                                           *)
(* --------------------------------------------------------- *)
(*                                                           *)
(*   Este programa realiza  la planeación de  trayectoria   *)
(*   para un robot móvil. El ambiente de este consiste en    *)
(*   un conjunto de bloques de varios tamaños, los cuales    *)
(*   se configuran para formar habitaciones y  objetos de    *)
(*   una casa.  Los datos  sensoriales del medio ambiente    *)
(*   son simulados.                                          *)
(*                                                           *)
(* --------------------------------------------------------- *)
(*                                                           *)
(*         AUTORES: JUAN ALFONSO MARTINEZ PADILLA            *)
(*                  LUIS ADRIAN LETEPICHIA FLORES            *)
(*                  JORGE ALFONSO HERNANDEZ SANTIS           *)
(*                                                           *)
(*         FECHA:   13 de octubre de 1988.                   *)
(*                                                           *)
(*************************************************************)


Program TurMov(input,output,region);

{$I Typedef.sys}
{$I graphix.sys}
{$I kernel.sys}
{$I windows.sys}
{$I hatch.hgh}

Const
   maxobj= 120;
   maxcos= 20;
   maxcua= 12;
   maxdir= 2;
   maxren= 160;
   maxcol= 225;
   pi=     3.141592654;

Type
   arreglo1= record
                x,y: integer;
             end;

   arreglo2= record
                c1,c2,cg: arreglo1;
                som:      integer;
             end;

   arreglo3= array[1..maxobj] of arreglo2;

   arreglo4= (ade,atr,izq,der);
```

```
    arreglo5= array[1..maxdir] of arreglo4;

    arreglo6= record
                  ci,cd,
                  cg,tray,
                  pos:     arreglo1;
                  dir:     arreglo4;
              end;

    arreglo7= array[1..maxdir] of arreglo6;

    arreglo8= array[0..maxcos] of arreglo2;

    arreglo9= record
                  ci,c2:   arreglo1;
                  objetos: arreglo8;
                  numcos:  integer;
              end;

    arreglo10= array[0..maxcua] of arreglo9;

    arreglo11= array[0..maxren,0..maxcol] of char;

Var
  region:             text;
  mapa:               arreglo11;
  posrob:             arreglo1;
  listobj:            arreglo3;
  listcua:            arreglo10;
  cuameta,cuactual,
  objmeta,numobj,
  numcua:             integer;
  nomarch:            wrkstring;
  ch:                 char;


{ esta funcion determina si una casilla del arreglo mapa
tiene un 1 }

Function PT (col,ren:integer):boolean;

begin
    if (mapa[ren,col]='0') then
      PT:=false
    else if (mapa[ren,col]='1') then
      PT:=true;
end;  { PT }


{  este proc.  produce un retardo y puede detectar una  tecla
   para abortar el retardo y realizar alguna funcion }

procedure delax(n: real);
```

```
var
  i:real;
  quit:boolean;

begin
   i:=0;
   ch:=' ';
   repeat
      i:=i+1;
      quit:=false;
      if keypressed then
      begin
         read(kbd,ch);
         quit:=(ch=^C);
         if (ch=^C) and keypressed then
         begin
            read(kbd,ch);
            quit:=(ch='O');
            ch:=' ';
         end;
      end;
      if quit then
      begin
         leavegraphic;
         halt;
      end;
   until (ch=^M) or (i>=n);
end; { delax }


{ este proc. borra una linea de la pantalla }

Procedure limpia_linea (i:integer);

Begin
   gotoxy(2,i);
   write('
                        ');
End;  { limpia_linea }


{ este proc. escribe un mensaje en la pantalla }

Procedure msg (s:wrkstring);

begin
   limpia_linea(22);
   gotoxy(10,22);
   write(s);
end;
```

```
{----------MODULO: CONSTRUCCION DE REPRESENTACION----------}

Procedure const_repres(nomarch:wrkstring; var listobj:
                       arreglo3; var listcua:arreglo10;
                       var numobj,numcua:integer);

{ este proc. lee de un archivo las coordenadas de los objetos,
las almacena en el arreglo listobj. Tambien se leen los
indices de listobj y se guardan en listcua. Se leen tambien
las coords. de las esquinas de los cuartos. Posteriormente
asigna 1's en las casillas del arreglo mapa en donde hay un
objeto y 0'S para las demas casillas }

Var
  i,j,k,indice:integer;

Begin
   msg('Espere un momento...');
   i:=0;
   assign(region,nomarch);
   reset(region);
   repeat
      i:=i+1;
      with listobj[i] do
      begin
         readln(region,c1.x,c1.y,c2.x,c2.y,som);
         cg.x:= round(c1.x+(c2.x-c1.x)/2);
         cg.y:= round(c1.y+(c2.y-c1.y)/2);
      end;
   until (listobj[i].som < 0);
   numobj:=i-1;
   i:=0;
   while not eof(region) do
   with listcua[i] do
   begin
      readln(region,c1.x,c1.y,c2.x,c2.y);
      j:=0;
      while not eoln(region) do
      begin
         read(region,indice);
         objetos[j]:=listobj[indice];
         j:=j+1;
      end;
      numcos:= j-1;
      readln(region);
      i:=i+1;
   end;
   numcua:=i-1;

   for i:=0 to maxren do
   for j:=0 to maxcol do
     mapa[i,j]:='0';
   for i:=1 to numobj do
   with listobj[i] do
```

```
   begin
      for j:=c1.x to c2.x do
      for k:=c1.y to c2.y do
        mapa[k,j]:='1';
   end;
end; { const_repres }


{-------------MODULO: GRAFICACION DE AMBIENTE---------------}

{ este proc. dibuja en la pantalla la configuracion de
habitaciones, asi como los objetos que hay en estas }

Procedure dibuja_region (listobj:arreglo3; numobj:integer;
                         listcua:arreglo10; numcua: integer);

Var
  i:integer;
  xcoord,ycoord: real;
  numtext: wrkstring;

begin
   for i:=1 to numobj do
   with listobj[i] do
   if (som > 0) then
   begin
      DrawSquare(c1.x,c1.y,c2.x,c2.y,false);
      str(som,numtext);
      DrawTextW(cg.x-1,cg.y-2,1,numtext);
   end
   else DrawSquare(c1.x,c1.y,c2.x,c2.y,true);
   for i:=1 to numcua do
   with listcua[i] do
   begin
      xcoord:= c1.x + (c2.x - c1.x)/2 - 3;
      ycoord:= c1.y + (c2.y - c1.y)/2 - 3;
      str(i,numtext);
      DrawTextW(xcoord,ycoord,2,numtext);
   end;
   for i:=1 to 21 do
   begin
      str(i,numtext);
      DrawTextW(i*10,5,1,numtext);
   end;
   DrawTextW(218,5,1,'X10');
   for i:=1 to 15 do
   begin
      str(i,numtext);
      DrawTextW(3,i*10,1,numtext);
   end;
   DrawTextW(3,156,1,'X10');
end; { dibuja_region }
```

```
{ este proc. produce un sonido de beep }

Procedure beep;

begin
   Sound(2500);
   Delay(100);
   Nosound;
end;  { beep }


{ este proc. determina el centro geometrico del objetivo }

Procedure fija_cg_meta (listcua:arreglo10; cuarto:integer;
                        objmeta: integer; local:boolean;
                        var cgmeta: arreglo1);
begin
   with listcua[cuarto] do
   begin
      if (not local) then
        cgmeta:= objetos[0].cg
      else
        cgmeta:= objetos[objmeta].cg;
   end;
end;  { fija_cg_meta }


{ este proc. determina en que habitacion se encuentra el
  robot o si se encuentra en el pasillo }

Procedure fija_cuarto (posrob:arreglo1; listcua:arreglo10;
                       numcua:integer; var cuactual:integer);
var
  i:integer;
  hallado:boolean;

begin
   i:=1;
   cuactual:=0;
   hallado:=false;
   while (i <= numcua) and (not hallado) do
   with listcua[i] do
   begin
      if (posrob.x >= c1.x) and (posrob.x <= c2.x) and
         (posrob.y >= c1.y) and (posrob.y <= c2.y) then
      begin
         cuactual:= i;
         hallado:= true;
      end
      else i:= i + 1;
   end;
end; { fija_cuarto }
```

```
{ este proc. selecciona las direcciones hacia donde el robot
  se puede mover para llegar al objetivo }

Procedure fija_direccion (posrob,cgmeta: arreglo1;
                          var direccion: arreglo5);

Var
  px,py: real;

Begin
   px:= cgmeta.x - posrob.x;
   py:= cgmeta.y - posrob.y;
   if (px > 0) then
     direccion[1]:= der
   else
     direccion[1]:= izq;
   if (py > 0) then
     direccion[2]:= atr
   else
     direccion[2]:= ade;
end;  { fija_direccion }


{ este proc. calcula los parametros de vision, tales como
  distancia y posicion
  en las dos direcciones fijadas en el proc. anterior }


Procedure toma_foto (posrob: arreglo1; direccion: arreglo5;
                                          var lista: arreglo7);
Var
  i,j,longrob: integer;
  x1,y1,x2,y2,x3,y3,alfa: real;
  hayobjeto1,hayobjeto2: boolean;

begin
   beep;
   longrob:=4;     { este parametro indica la longitud del
                     robot }
   j:= 0;
   for i:=1 to maxdir do
   begin
      x2:= posrob.x;
      y2:= posrob.y;
      if (direccion[i] = izq) or (direccion[i] = der) then
      begin
         y1:= y2 - (longrob/2);
         y3:= y2 + (longrob/2);
         x1:= x2;
         x3:= x2;
      end
        else
      begin
         x1:= x2 - (longrob/2);
```

```
            x3:= x2 + (longrob/2);
            y1:= y2;
            y3:= y2;
         end;
         case direccion[i] of
            izq: alfa:= pi;
            der: alfa:= 0;
            atr: alfa:= pi/2;
            ade: alfa:= pi * 1.5;
         end;
         repeat
             x1:= x1 + cos(alfa);
             y1:= y1 + sin(alfa);
             x2:= x2 + cos(alfa);
             y2:= y2 + sin(alfa);
             x3:= x3 + cos(alfa);
             y3:= y3 + sin(alfa);
             hayobjeto1:= PT(trunc(x1),trunc(y1));
             hayobjeto2:= PT(trunc(x3),trunc(y3));
         until (hayobjeto1) or (hayobjeto2);
         j:= j + 1;
         lista[j].dir:= direccion[i];
         lista[j].tray.x:= trunc(x2);
         lista[j].tray.y:= trunc(y2);
         lista[j].cg.x:= 0;
         lista[j].cg.y:= 0;
         if (hayobjeto1) then
         begin
            lista[j].pos.x:= trunc(x1);
            lista[j].pos.y:= trunc(y1);
         end
           else if (hayobjeto2) then
         begin
            lista[j].pos.x:= trunc(x3);
            lista[j].pos.y:= trunc(y3);
         end;
      end;
end;   { toma_foto }


{ este proc. determina los centros geometricos de los objetos
  detectados }

Procedure obtiene_centros (listcua:arreglo10; cuarto:integer;
                           var lista:arreglo7);
var
  i,j,num: integer;
  hallado: boolean;

begin
   num:= listcua[cuarto].numcos;
   for i:=1 to maxdir do
   begin
      j:=0;
```

```
      hallado:= false;
      while (j <= num) and (not hallado) do
      begin
         with listcua[cuarto].objetos[j],lista[i] do
         if (pos.x >= (c1.x)) and (pos.x <= (c2.x)) and
            (pos.y >= (c1.y)) and (pos.y <= (c2.y)) then
         begin
            lista[i].cg:= listcua[cuarto].objetos[j].cg;
            lista[i].ci:= listcua[cuarto].objetos[j].c1;
            lista[i].cd:= listcua[cuarto].objetos[j].c2;
            hallado:= true;
         end
           else
             j:= j + 1;
      end;
   end;
end;  { obtiene_centros }


{ este proc. verifica si algun objeto detectado es el
  objetivo }

Procedure verifica_meta (lista: arreglo7; cgmeta: arreglo1;
                         var llega: boolean;
                         var indice: integer);
Var
  i: integer;
  hallado: boolean;

begin
   i:=1;
   indice:=1;
   llega:= false;
   hallado:= false;
   while (i <= maxdir) and (not hallado) do
   with lista[i] do
   begin
      if (cg.x = cgmeta.x) and (cg.y = cgmeta.y) then
      begin
         llega:= true;
         indice:= i;
         hallado:= true;
      end
        else
          i:= i + 1;
   end;
end;  { verifica_meta }


{ este proc. decide en que direccion debe moverse el robot y
  si se debe activar el proc. detecta_cam }

Procedure decide_dir(posrob,cgmeta: arreglo1;lista:arreglo7;
                     var indice:integer;var detectar:boolean);
```

```
Var
  i,xind,yind: integer;
  xcuad,ycuad,xmeta,ymeta,tol: real;

begin
tol:= 6;  { parametro que indica la tolerancia entre el
            robot y un objeto }
   detectar:= false;
   for i:=1 to maxdir do
   with lista[i] do
   begin
      case dir of
         izq,der: begin
                     xcuad:= abs(tray.x - posrob.x);
                     xmeta:= abs(cgmeta.x - posrob.x);
                     xind:= i;
                  end;
         ade,atr: begin
                     ycuad:= abs(tray.y - posrob.y);
                     ymeta:= abs(cgmeta.y - posrob.y);
                     yind:= i;
                  end;
      end; { case }
   end;

   if (xcuad < tol) and (ycuad <= ymeta) then
     indice:= yind
   else if (xcuad <= xmeta) and (ycuad < tol) then
     indice:= xind
   else if (xcuad >= xmeta) and (ycuad < tol) then
   begin
      indice:= xind;
      detectar:= true;
   end
   else if (xcuad < tol) and (ycuad >= ymeta) then
   begin
      indice:= yind;
      detectar:= true;
   end
   else if (xcuad <= xmeta) and (ycuad <= ymeta) then
   begin
      if (xcuad > ycuad) then
        indice:= xind
      else
        indice:= yind;
   end
   else if (xcuad < tol) and (ycuad >= ymeta) then
   begin
      indice:= yind;
      detectar:= true;
   end
   else if (xcuad >= xmeta) and (ycuad < tol) then
   begin
      indice:= xind;
```

```
      detectar:= true;
   end
   else if (xcuad >= xmeta) and (ycuad <= ymeta) then
   begin
      if (xmeta < ymeta) then
        indice:= yind
      else
      begin
         indice:= xind;
         detectar:= true;
      end;
   end
   else if (xcuad <= xmeta) and (ycuad >= ymeta) then
   begin
      if (xmeta < ymeta) then
      begin
         indice:= yind;
         detectar:= true;
      end
      else
        indice:= xind;
   end
   else if (xcuad >= xmeta) and (ycuad >= ymeta) then
   begin
      if (xmeta < ymeta) then
        indice:= yind
      else
        indice:= xind;
      detectar:= true;
   end;
end;  { decide_dir }


{ este proc. calcula la distancia del robot a un objeto en
  una cierta direccion }

Procedure detecta_cam (posrob: arreglo1; dircomp: arreglo4;
                       var dist:integer);

Var
  alfa,xcoord,ycoord: real;
  hayobjeto: boolean;

begin
   case dircomp of
      izq: alfa:= pi;
      der: alfa:= 0;
      atr: alfa:= pi/2;
      ade: alfa:= pi * 1.5;
   end; { case }
   xcoord:= posrob.x;
   ycoord:= posrob.y;
   dist:= 0;
   repeat
```

```
        dist:= dist + 1;
        xcoord:= xcoord + cos(alfa);
        ycoord:= ycoord + sin(alfa);
        hayobjeto:= PT(trunc(xcoord),trunc(ycoord));
    until (hayobjeto);
end;  { detecta_cam }


{ este proc. traza una trayectoria de una direccion y
  longitud especificadas }

Procedure traza_tray (dir: arreglo4; coord:real;
                      var posrob:arreglo1);

Var
  i,tiempo:integer;

begin
   tiempo:=25;
   case dir of

      izq: begin
             gotoxy(64,25);
             write(chr(27));
             i:= round(posrob.x);
             while(i>=coord) do
             begin
                drawline(posrob.x,posrob.y,i,posrob.y);
                gotoxy(25,25);
                write(i:3);
                delax(tiempo);
                posrob.x:=i;
                i:=i-1;
             end;
           end;
      der: begin
             gotoxy(64,25);
             write(chr(26));
             i:= round(posrob.x);
             while (i<=coord) do
             begin
                drawline(posrob.x,posrob.y,i,posrob.y);
                gotoxy(25,25);
                write(i:3);
                delax(tiempo);
                posrob.x:=i;
                i:=i+1;
             end;
           end;
     `ade: begin
             gotoxy(64,25);
             write(chr(24));
             i:= round(posrob.y);
             while (i>=coord) do
```

```
            begin
               drawline(posrob.x,posrob.y,posrob.x,i);
               gotoxy(46,25);
               write(i:3);
               delax(tiempo);
               posrob.y:=i;
               i:=i-1;
            end;
         end;
      atr: begin
               gotoxy(64,25);
               write(chr(25));
               i:= round(posrob.y);
               while (i<=coord) do
               begin
                  drawline(posrob.x,posrob.y,posrob.x,i);
                  gotoxy(46,25);
                  write(i:3);
                  delax(tiempo);
                  posrob.y:=i;
                  i:=i+1;
               end;
            end;
   end; { case }
end;  { traza_tray }


{ este proc. ejecuta la trayectoria calculada en los proc.
  anteriores }

Procedure ejecuta_tray(lista: arreglo7; indice: integer;
                       detectar:boolean;var posrob:arreglo1);
Var
  dist,dist2,tol,paso,coord: integer;
  dir,dircomp: arreglo4;

begin
   msg('Ejecutando trayectoria...');
   dir:= lista[indice].dir;
   if (not detectar) then
   begin
      tol:=  5;    { este parametro indica la distancia a  que
                     queda el robot de un objeto despues de
                     ejecutar una trayectoria directa }
      case dir of
         izq: coord:= lista[indice].tray.x + tol;
         der: coord:= lista[indice].tray.x - tol;
         ade: coord:= lista[indice].tray.y + tol;
         atr: coord:= lista[indice].tray.y - tol;
       end;
       traza_tray (dir,coord,posrob);
   end  { if }
   else
   begin
```

```
   paso:= 2;  { este parametro indica la longitud del paso
                cuando se ejecuta una trayectoria por pasos }
  if (indice=1) then
       dircomp:= lista[2].dir
     else
       dircomp:= lista[1].dir;
     detecta_cam(posrob,dircomp,dist);
     repeat
        case dir of
           izq: coord:= posrob.x - paso;
           der: coord:= posrob.x + paso;
           ade: coord:= posrob.y - paso;
           atr: coord:= posrob.y + paso;
        end;
        dist2:=dist;
        traza_tray(dir,coord,posrob);
        detecta_cam(posrob,dircomp,dist);
     until (abs(dist - dist2) > 0);
     paso:=3;
        case dir of
           izq: coord:= posrob.x - paso;
           der: coord:= posrob.x + paso;
           ade: coord:= posrob.y - paso;
           atr: coord:= posrob.y + paso;
        end;
        traza_tray(dir,coord,posrob);
   end; { if }
end;  { ejecuta_tray }


{ este proc. realiza las funciones de abrir y cerrar la
  puerta de algun cuarto asi como calcular la trayectoria
  para entrar o salir de ese cuarto }

Procedure sal_ent_cua (listcua:arreglo10; cuarto:integer;
                       lista:arreglo7; indice:integer;
                       var posrob:arreglo1);
Var
  paso:integer;
  coord:real;
  dir:arreglo4;

begin
   SetColorBlack;
   msg('Abriendo puerta...');
   with listcua[cuarto].objetos[0] do
     Hatch(c1.x,c1.y,c2.x,c2.y,1);
   delax(1500);
   SetColorWhite;
   paso:= 14;
   dir:= lista[indice].dir;
   case dir of
     izq: coord:= posrob.x - paso;
     der: coord:= posrob.x + paso;
```

```
      ade: coord:= posrob.y - paso;
      atr: coord:= posrob.y + paso;
    end;
    traza_tray(dir,coord,posrob);
    msg('Cerrando puerta...');
    with listcua[cuarto].objetos[0] do
      Hatch(c1.x,c1.y,c2.x,c2.y,1);
    delax(1500);
end;  { sal_ent_cua }


{---------MODULO: INICIALIZACION DE SISTEMA GRAFICO---------}

{ este proc. define las ventanas para dibujar las
  habitaciones y los mensajes }

procedure define_ventana;

begin
    InitGraphic;
    DefineWindow(1,0,0,xmaxglb,trunc(ymaxglb*7.9/10));
    DefineWindow(2,0,trunc(ymaxglb*7.9/10),xmaxglb,
                 trunc(ymaxglb*9.4/10));
    DefineWindow(3,0,0,xmaxglb,ymaxglb);
    DefineWindow(4,trunc(xmaxglb*2.5/10),trunc(ymaxglb*2/10),
                 trunc(xmaxglb*7.5/10),trunc(ymaxglb*5/10));
DefineHeader(1,'TRAYECTORIAS PARA UN ROBOT MOVIL');
   DefineWorld(1,0,160,225,0);
    DefineWorld(2,0,99,99,0);
end;  { define_ventana }


{ este proc. dibuja en pantalla la presentacion del programa}

Procedure presenta_prog;

var
  i:integer;

begin
    SelectWorld(2);
    SelectWindow(3);
    DrawBorder;
    SelectWindow(4);
    DrawBorder;
    SelectWindow(3);
    DrawTextW(5,7,2,'PRESENTACION DE:');
    DrawTextW(27,32,8,'TURMOV');
    DrawTextW(5,65,3,'TRAYECTORIAS PARA UN ROBOT MOVIL');
    DrawTextW(5,88,2,'Autores: J.M. L.L. J.H.');
    gotoxy(50,23);
    write('Presione ENTER para continuar');
    SelectWindow(4);
    i:=0;
```

```
   repeat
      InvertWindow;
      delax(800);
      i:=i+1;
   until (ch=^M) or (i=20);
   SelectWindow(3);
   SetBackground(0);
   SetHeaderOn;
   SelectWorld(1);
   SelectWindow(2);
   DrawBorder;
   SelectWindow(1);
   DrawBorder;
end; { presenta_prog }


{ este proc. prende y apaga el caracter que representa
  al robot }

Procedure flashea(posrob:arreglo1);

begin
   repeat
      DrawTextW(posrob.x,posrob.y,1,'*');
      delay(250);
      SetColorBlack;
      DrawTextW(posrob.x,posrob.y,1,'*');
      delay(250);
      SetColorWhite;
   until keypressed;
end;  { flashea }


{------------MODULO: SELECCION DE POSICION ROBOT------------}

{ este proc. lee de pantalla la posicion inicial del robot }

Procedure lee_pos (var posrob:arreglo1);

Var
  i:integer;
  hayerror:boolean;

begin
   repeat
      limpia_linea(22);
      limpia_linea(25);
      gotoxy(18,22);
      write('Posicion en X:            Posicion en Y:');
      gotoxy(33,22);
      read(posrob.x);
      gotoxy(58,22);
      read(posrob.y);
      hayerror:=PT(posrob.x-2,posrob.y) or
```

```
                PT(posrob.x+2,posrob.y) or
                PT(posrob.x,posrob.y-2) or
                PT(posrob.x,posrob.y+2);
      if hayerror then
      begin
         beep;
         msg('ERROR,  posicion invalida  * teclear nuevamente
              los datos *');
         delax(2500);
      end;
   until (not hayerror);
   gotoxy(17,25);
   write('Xrobot:                Yrobot:                Dir: ');
   gotoxy(25,25);
   write(posrob.x:3);
   gotoxy(46,25);
   write(posrob.y:3);
   DrawTextW(posrob.x,posrob.y,1,'*');
end; { lee_dato1 }


{-----------------MODULO: SELECCION DE META-----------------}

{ este proc. lee de pantalla la identificacion del objetivo }

Procedure lee_meta (listcua:arreglo10; numcua:integer;
                    posrob:arreglo1; var cuameta,
                    objmeta: integer);

Var
  hayerror:boolean;

begin
   repeat
      limpia_linea(22);
      gotoxy(21,22);
      write('Cuarto meta:              Objeto meta:');
      gotoxy(34,22);
      flashea(posrob);
      read(cuameta);
      gotoxy(59,22);
      flashea(posrob);
      read(objmeta);
      hayerror:=(cuameta<1)or(cuameta>numcua)or(objmeta<1)or
                (objmeta>listcua[cuameta].numcos);
      if hayerror then
      begin
         beep;
         msg('ERROR,  meta invalida  * teclear nuevamente los
              datos *');
         delax(2500);
      end;
   until (not hayerror);
end;  {lee_meta }
```

```
{------------MODULO: SELECCION DE CONFIGURACION-------------}

{ este proc. determina que base de datos se va a accesar de
  disco }

Procedure lee_cuadro(var nomarch:wrkstring);

Var
  cuad:integer;
  hayerror:boolean;

begin
   limpia_linea(25);
   repeat
      limpia_linea(22);
      gotoxy(15,22);
      write('Cuadro a accesar [1,2]:');
      gotoxy(39,22);
      read(cuad);
      hayerror:=(cuad<1)or(cuad>2);
      if hayerror then
      begin
         beep;
         msg('ERROR, cuadro invalido  * teclear nuevamente el
              dato *');
         delax(2500);
      end;
   until (not hayerror);
   case cuad of
      1: nomarch:='REGION1.DTA';
      2: nomarch:='REGION2.DTA';
   end;
end; { lee_cuadro }


{ este proc. escribe en pantalla una breve explicacion sobre
  como usar el programa  }

Procedure lee_ayuda;

Var
  ren,lin,i:integer;
  ch:char;

{ este proc. escribe lentamente una cadena de caracteres }

Procedure writelento(s:wrkstring);

begin
   ren:=ren+1;
   lin:=lin+1;
   gotoxy(10,3+ren);
   for i:=1 to length(s) do
   begin
```

```
        write(s[i]);
        gotoxy(10+i,3+ren);
        delax(10+random(100));
    end;
    if (lin=15) or (lin=20) then
    begin
        ren:=0;
        msg('Presione ENTER para continuar');
        delax(30000.0);
        limpia_linea(22);
        SetBackground(0);
        DrawBorder;
    end;
end; { writelento }

begin
    repeat
        msg('Necesita instrucciones? S/N [N]: ');
        read(kbd,ch);
        write(Upcase(ch));
    until Upcase(ch) in ['S','N',#13];
    if (Upcase(ch)='S') then
    begin
        ren:=0; lin:=0;
        writelento('Este  es  un programa de trayectorias
                    para un robot móvil. El');
        writelento('ambiente del robot esta estructurado en
                    forma de habitaciones');
        writelento('de una casa con objetos dentro de estas
                    habitaciones. El');
        writelento('objetivo  del  programa  es  que  dada  una
                    posición inicial del');
        writelento('robot   y  una  meta  consistentes  de   la
                    habitación y el objeto');
        writelento('dentro de esta, se planee y se traze una
                    trayectoria de');
        writelento('recorrido del robot.');
        writelento(' ');
        writelento('El primer dato que pide el programa es que
                    configuración de');
        writelento('habitaciones se va accesar de la base datos
                    en disco.');
        writelento('Enseguida se pide las coordenadas x-y de la
                    posición inicial');
        writelento('del robot, y finalmente se debe dar la meta
                    del robot, la');
        writelento('cual consiste en un numero de habitación y
                    el numero del');
        writelento('objeto dentro de esta habitación.');
        writelento(' ');
        writelento('Despues de que el programa ha concluido el
                    objetivo, se tienen');
        writelento('las  opciones  de continuar con  esa  misma
                    posición, dar una');
```

```
      writelento('nueva posición, accesar otra configuración
                  de habitaciones y');
      writelento('realizar otra simulación con una nueva meta
                  o tambien por');
      writelento('supuesto se tiene la opción de salir del
                  programa.');
   end;
end; { lee_ayuda }


{--------------MODULO: CALCULO DE TRAYECTORIAS--------------}

{ este es el proc. principal de planeacion y ejecucion de
  trayectoria de acuerdo al centro geometrico de un objetivo }

Procedure plan_ejec_tray (cgmeta:arreglo1; listcua:arreglo10;
                          cuarto:integer; var posrob:arreglo1;
                          var lista:arreglo7;
                          var indice:integer);
var
  direccion: arreglo5;
  detectar,llega: boolean;

begin
   repeat
      msg('Planeando trayectoria...');
      fija_direccion(posrob,cgmeta,direccion);
      toma_foto(posrob,direccion,lista);
      obtiene_centros(listcua,cuarto,lista);
      verifica_meta(lista,cgmeta,llega,indice);
      if (not llega) then
      begin
         decide_dir(posrob,cgmeta,lista,indice,detectar);
         ejecuta_tray(lista,indice,detectar,posrob);
      end
      else ejecuta_tray(lista,indice,false,posrob);
   until (llega);
end;  { plan_ejec_tray }


{-------------MODULO: PLANEACION DE TRAYECTORIAS------------}

{ este proc. planea los subobjetivos que se tienen que
  alcanzar, a partir del objetivo global definido en el
  programa principal }

Procedure plan_objetivo(listcua:arreglo10; cuactual,cuameta,
                        objmeta:integer;var posrob:arreglo1);
var
  cgmeta:arreglo1;
  lista:arreglo7;
  indice:integer;

begin
```

```
   if (cuameta = cuactual) then
     begin
        fija_cg_meta(listcua,cuactual,objmeta,true,cgmeta);
        plan_ejec_tray(cgmeta,listcua,cuactual,posrob,lista,
                       indice);
     end
     else if (cuactual > 0) then
     begin
        fija_cg_meta(listcua,cuactual,objmeta,false,cgmeta);
        plan_ejec_tray(cgmeta,listcua,cuactual,posrob,lista,
                       indice);
        sal_ent_cua(listcua,cuactual,lista,indice,posrob);
        fija_cg_meta(listcua,cuameta,objmeta,false,cgmeta);
        plan_ejec_tray(cgmeta,listcua,0,posrob,lista,indice);
        sal_ent_cua(listcua,cuameta,lista,indice,posrob);
        fija_cg_meta(listcua,cuameta,objmeta,true,cgmeta);
        plan_ejec_tray(cgmeta,listcua,cuameta,posrob,lista,
                       indice);
     end
     else if (cuactual = 0) then
     begin
        fija_cg_meta(listcua,cuameta,objmeta,false,cgmeta);
        plan_ejec_tray(cgmeta,listcua,cuameta,posrob,lista,
                       indice);
        sal_ent_cua(listcua,cuameta,lista,indice,posrob);
        fija_cg_meta(listcua,cuameta,objmeta,true,cgmeta);
        plan_ejec_tray(cgmeta,listcua,cuameta,posrob,lista,
                       indice);
    end;
end; { plan_objetivo }


{--------------------MODULO: PRINCIPAL----------------------}

begin
   define_ventana;
   presenta_prog;
   lee_ayuda;
   lee_cuadro(nomarch);
   const_repres(nomarch,listobj,listcua,numobj,numcua);
   dibuja_region(listobj,numobj,listcua,numcua);
   lee_pos(posrob);
   repeat
     lee_meta(listcua,numcua,posrob,cuameta,objmeta);
     fija_cuarto(posrob,listcua,numcua,cuactual);
     plan_objetivo(listcua,cuactual,cuameta,objmeta,posrob);
     limpia_linea(22);
     gotoxy(9,22);
     write('C   Continuar      N   Nueva posicion      O   Otro
           cuadro     S  Salir');
     repeat
        DrawTextW(posrob.x,posrob.y,1,'*');
        flashea(posrob);
        read(kbd,ch);
```

```
    until Upcase(ch) in ['C','N','O','S'];
    case Upcase(ch) of

    'C': begin
           SetBackground(0);
           DrawBorder;
           dibuja_region(listobj,numobj,listcua,numcua);
           DrawTextW(posrob.x,posrob.y,1,'*');
           gotoxy(64,25);
           write(' ');
           cuactual:=cuameta;
         end;
    'N': begin
           SetBackground(0);
           DrawBorder;
           dibuja_region(listobj,numobj,listcua,numcua);
           lee_pos(posrob);
         end;
    'O': begin
           SetBackground(0);
           DrawBorder;
           lee_cuadro(nomarch);
           const_repres(nomarch,listobj,listcua,numobj,
                        numcua);
           dibuja_region(listobj,numobj,listcua,numcua);
           lee_pos(posrob);
         end;
    end;
    until Upcase(ch)='S';
  LeaveGraphic;
end.  { robmovil }
```

A P E N D I C E B

M A N U A L D E U S U A R I O D E L O S S I S T E M A S "S C T", "S G T" Y "T U R M O V"

TABLA DE CONTENIDO

1. Introducción.

SCT - Sistema Controlador de Trayectorias - es un sistema que permite la planeación y generación de trayectorias tridimensionales para un manipulador con seis grados de libertad, basándose en información del entorno que rodea al robot. El entorno estará compuesto de objetos. El sistema será capaz de trazar una trayectoria para que el manipulador se coloque en algún objeto (librando obstáculos) y otra para que el manipulador traslade el objeto hacia una posición (también librando obstáculos).

SGT - Sistema Graficador de Trayectorias - que permitirá visualizar gráficamente las trayectorias generadas por el sistema SCT. También, podrá mostrar información acerca de las variables del manipulador, y de la posición del robot y los objetos a manipular. Además, visualizará el movimiento del manipulador para seguir las trayectorias planeadas.

TURMOV - Sistema de trayectorias para un robot móvil - es un sistema de planeación y ejecución de trayectorias para un robot movil. El ambiente del robot está estructurado en forma de habitaciones de una casa común, cada habitación conteniendo un cierto número de objetos los cuales podrían ser una silla, una mesa, etc. El objetivo del sistema es que, dada una posición inicial del robot (en coordenadas x-y) y una meta (la habitación a la que debe llegar y un objeto dentro de esta habitación), se planee y se trace una trayectoria de recorrido del robot para que este pueda llegar a la meta especificada. La trayectoria es construida a base de segmentos de recta perpendiculares, los cuales permiten guiar al robot móvil en cuatro direcciones posibles (izquierda, derecha, adelante, atras).

Los sistemas mencionados se programaron en lenguaje Pascal ya que permite crear programas de alta calidad y estructuración. Para los sistemas SGT y TURMOV se usó, además, el paquete Turbographics.

Se utilizó una microcomputadora compatible PC con capacidad de 640 KBytes de memoria principal y con una tarjeta de gráficos CGA (para correr los sistemas SGT y TURMOV).

El presente manual tiene como objetivo guiar al usuario en el uso de los sistemas SCT, SGT y TURMOV, permitiendo utilizarlos sin ser necesario su pleno conocimiento.

Antes de intentar correr los sistemas, se necesita verificar lo siguiente :

- Se debe tener un disco flexible del sistema operativo versión 3.10 en adelante.

- Se debe tener un disco que contenga los archivos necesarios para correr los sistemas. Esto se verá más adelante.

- Se debe contar con una computadora compatible con PC que contenga una tarjeta para gráficos (opcional para el sistema SCT).

- Se debe tener a la mano este manual.

- Se requiere leer la tésis para entender los resultados obtenidos en una sesión.

2. Inicialización.

Esta parte es obligatoria, y consta de los pasos siguientes:

a) Insertar el disco flexible del sistema operativo en la unidad de discos.

b) Encender la computadora.

c) Esperar a que el sistema operativo sea cargado en la memoria principal.

d) Esperar a que aparesca el promt A> , lo cual indica que la computadora está lista para aceptar ordenes.

e) Sacar el disco del sistema operativo e insertar el disco que contiene el còdigo ejecutable de los sistemas desarrollados en esta tèsis.

f) Teclear DIR , para checar que esten los archivos del paquete TURBOGRAPHICS que son :

- ERROR.MSG
- 4X6.FON
- 8X8.FON
- 14X9.FON

Este inciso es opcional para cuando se use el sistema SCT.

3. Cómo correr y usar el sistema SCT.

Para correr y usar el sistema SCT se deben seguir estos pasos :

a) Efectuar la Inicialización.

b) Teclear DIR , para verificar si se encuentran los siguientes archivos :

- SCT.COM
- LECTURA.COM
- POSIC.DTA
- OBJECT.DTA

c) Si ya se verificó el inciso b, entonces, teclear :

LECTURA

lo que ocasionará que se ejecute el programa de captura de datos para el sistema SCT.

Para usar el programa de CAPTURA se tendrá que leer las lineas que siguen :

El programa de CAPTURA se encarga de almacenar las posiciones actuales de los objetos y su posiciones de traslado ,por tanto, me pedirá como entrada, las coordenadas actuales de los objetos, definidas como:

$$\text{OBJETO } i = \begin{vmatrix} px1 & px2 & px3 & px4 \\ py1 & py2 & py3 & py4 \\ pz1 & pz2 & pz3 & pz4 \end{vmatrix}$$

donde px1, py1, pz1 son las coordenadas del vèrtice uno de la base del objeto i, px2, py2, pz2 son las del vèrtice dos, etc, asimismo me pedirá los siguientes datos para cada objeto i:

- ALTURA MAXIMA DEL OBJETO i
- CENTRO DE GRAVEDAD DE LA BASE DEL OBJETO i
- LONGITUD TRANSVERSAL DEL OBJETO i

El programa CAPTURA.COM es capáz de:

- almacenar objetos nuevos y borrar anteriores
- adición de objetos a los ya existentes
- imprimir los archivos de objetos y posiciones ya almacenados.

Finálmente, cabe decir, que si los archivos de objetos y posiciones fueran almacenados ya con anterioridad, no es necesario entrar a correr este programa, ya que ya se encuentran generados los archivos de datos y de posiciones, denominados como:

- OBJECT.DTA
- POSIC.DTA

e) Ya efectuada la captura, entonces, teclear :

SCT

lo que hará que se ejecute el programa de planeación y generación de trayectorias tridimensionales.

Esta ejecución dará como resultado dos archivos que contienen la información de los objetos y las trayectorias que seguirá el manipulador. Estos archivos son :

- ARCH.DTA
- PUNTS.DTA

4. Cómo correr y usar el sistema SGT.

Para efectuar la corrida del sistema SGT, se deben seguir los pasos siguientes :

a) Correr el sistema SCT.

b) Teclear DIR , para verificar los siguientes archivos :

- SGT.COM
- ARCH.DTA
- PUNTS.DTA

c) Ya verificado el inciso b, entonces, teclear :

SGT

lo cual efectuará la ejecución del despliegue gráfico tridimensional del manipulador siguiendo las trayectorias apropiadas para efectuar sus tareas.

d) Una vez realizado c) , aparecerá en pantalla los resultados mostrados en el capitulo 3 del subtema de simulaciones y resultados.

Uso del sistema SGT.

El uso de este programa es muy sencillo. primero hay que oprimir una tecla dos veces. lo cual hará que vayan apareciendo en la pantalla una secuencia de imágenes del manipulador y objetos, además de la siguiente información :

- Posición actual del manipulador.
- Posición inicial y final del objeto a manipular.
- Leds de error para ver si alguna variable del manipulador esta fuera de un rango prestablecido.
- Letrero para indicar la detección de un objeto.
- Valores de las variables del manipulador.

Después que termina la secuencia anterior aparece un menú con las siguientes opciones :

esc : salida.

t : dibujar grafica de las trayectorias.

m : mostrar el manipulador en movimiento.

Se debe elegir la opción.

Este menú aparecerá tantas veces como manipulaciones de objetos se vayan a realizar. Esto es, durante la manipulación de 3 objetos el menú aparecerá 3 veces.

5. Cómo correr y usar el sistema TURMOV.

Para llevar a cabo la corrida del sistema TURMOV, se tendrán que seguir estos pasos :

a) Efectuar la Inicialización.

b) Teclear DIR , para verificar que esten los siguientes archivos :

- TURMOV.COM
- REGION1.DTA
- REGION2.DTA

c) Si ya existen los archivos del inciso b, entonces, teclear :

TURMOV

lo cual hará que se ejecute la planeación y despliegue gráfico para un robot móvil.

d) Ya realizado lo anterior se mostrarán los resultados en la forma establecida en el capitulo 4.

Uso del sistema TURMOV.

Después de hacer lo anterior aparecera en la pantalla la presentación del programa (ver capitulo 4), si queremos pasar a lo siguiente teclear ENTER, si no, el sistema automáticamente pasara después de un periodo de tiempo corto. En este momento aparece el mensaje:

Instrucciones? S/N [N]:

Si queremos instrucciones acerca del programa teclear S, en caso contrario teclear ENTER ó N. Después de hacer esto aparece el mensaje:

Cuadro a accesar [1,2]:

Con esto podemos seleccionar una de las dos configuraciones de habitaciones que existen en un archivo en disco. Teclear 1 ó 2 según sea la que queramos. Una vez hecho esto se dibuja en pantalla la configuración de habitaciones y aparece el mensaje:

Posición en X: Posición en Y:

Esto indica que tecleemos la posiciòn inicial del robot para X y Y, la cual debe estar en escala de 10. Al hacer esto aparece en pantalla un asterisco en la posiciòn especificada, el cual representa al robot movil, y se despliega otro mensaje de solicitud de datos:

Cuarto meta: Objeto meta:

Este mensaje indica que se teclee el No. de habitaciòn y el objeto al que queremos que el robot llegue. En la pantalla estan numeradas las habitaciones y los objetos.

Al dar los datos anteriores el sistema empieza a realizar la planeaciòn y ejecuciòn de trayectoria, la cual se puede ver como se va trazando en la pantalla, ademàs de que se va indicando la posiciòn y direcciòn actual del robot.

Cuando el robot concluye el objetivo, se despliega en pantalla el mensaje:

C Continuar N Nueva Posicion O Otro cuadro S Salir

Esto constituye cuatro opciones que podemos elegir presionando la tecla indicada:

- Con C se puede volver a elegir un objetivo y la planeaciòn se hara a partir de la posiciòn actual del robot.
- Con N podemos fijar una nueva posiciòn y un nuevo objetivo para el robot.
- Con O se puede elegir otra configuraciòn de habitaciones y realizar todo el proceso anterior.
- Con S termina la ejecuciòn del sistema y se devuelve el control al sistema operativo.