Universidad Autónoma del Estado de Hidalgo

Instituto de Ciencias Básicas e Ingeniería

Centro de Investigación en Tecnologías de Información y Sistemas

Tesis

Análisis y diseño de una interfaz de visualización virtual del dispositivo háptico PHANToM 1.0 basada en la dinámica de Euler-Lagrange

que para obtener el grado de Maestría en Ciencias Computacionales presenta:

Herbert Lara Ordaz

Director de tesis: Dr. Omar Arturo Domínguez Ramírez

Co-Director de tesis: Dr. Ramón Soto de la Cruz

Director externo de tesis: Dr. Rene Mayorga

Pachuca Hgo.,Octubre del 2005

A todas las personas que han estado cerca de mi e influyeron de manera directa o indirecta este proyecto.

Agradecimientos

Quiero agradecer en primer lugar a mi madre, la profesora Margarita Ordaz Labra, por estar al pendiente de mi.

A mis maestros y en particular a mi asesor de tesis al Dr. Omar A. Domínguez Ramírez.

También agradezco el apoyo y compañerismo que me ha demostrado la Lic. Ma. Teresa Torres Escalante.

Resumen

La realidad virtual es una herramienta versátil que permite observar, experimentar e interactuar con escenarios simulados en la computadora. Los objetos virtuales presentan un comportamiento definido por modelos que pueden ser matemáticos, estocásticos entre otros.

Mediante la simulación del comportamento de robots se puede apreciar sus capacidades y limitaciones.

En este trabajo se presenta la metodología para visualizar virtualmente a dispositivos electromecánicos considerando a las leyes dinámicas que los rigen (cinemática y dinámica), cuyo caso de estudio es un dispositivo háptico PHANToM 1.0 de 3 grados de libertad. Se dan a conocer los modelos cinemáticos y dinámicos basados en el algoritmo Denavit-Hartenberg y la formulación de Euler-Lagrange. Se presentan resultados experimentales referidos al análisis de la dinámica y sus propiedades en movimiento libre y en lazo abierto, y se evalúa una estrategia de control no lineal para seguimiento de trayectorias.

La interfaz gráfica está desarrollada con librerías de OpenGL para Visual C++, estas librerías son muy usadas para la creación de ambientes virtuales.

Los resultados de la simulación que proporciona Matlab son guardados en un archivo que es leído posteriormente por la interfaz gráfica.

Índice general

1. Introducción **8**
- 1.1. Simulación y ambientes virtuales . . . 8
- 1.2. Simulación de robots . . . 9
- 1.3. Objetivo principal . . . 10
- 1.4. Objetivos específicos . . . 10
- 1.5. Justificación . . . 10
- 1.6. Tareas desarrolladas . . . 10
- 1.7. Hipótesis . . . 11
- 1.8. Aportes tecnológicos . . . 11
- 1.9. Herramientas empleadas . . . 11

2. Estudio del estado del arte: simulación de robots manipuladores **12**
- 2.1. Simulación de robots articulados . . . 12
- 2.2. SIMUROB . . . 14
- 2.3. Software de simulación del robot PUMA Unimation 560 basado en la cinemática directa e inversa de posición . . . 16

3. Visualizadores y simuladores **18**
- 3.1. Introducción . . . 18
- 3.2. Simulación de procesos físicos . . . 18
- 3.3. Simuladores de sistemas digitales . . . 19
  - 3.3.1. Simnon . . . 21
  - 3.3.2. 20-sim . . . 23
  - 3.3.3. Matlab . . . 23
- 3.4. Simuladores virtuales . . . 24
  - 3.4.1. AutoForm . . . 25
  - 3.4.2. Vericut . . . 25
  - 3.4.3. WorkNC-CAD . . . 26
  - 3.4.4. AutoARQ . . . 27
  - 3.4.5. Squiggle . . . 28
- 3.5. Conclusiones . . . 29

4. Dispositivo háptico PHANToM 1.0 **30**
4.1. Introducción . . . 30
4.2. PHANToM 1.0 . . . 30
4.3. Modelo cinemático . . . 31
4.3.1. Metodología para la obtención del modelo cinemático directo de posición (MCDP) . . . 31
4.3.2. Cadena cinemática . . . 32
4.3.3. Marcos ortonormales . . . 32
4.3.4. Parámetros Denavit-Hartenberg (DH) . . . 33
4.3.5. Matrices elementales y de transformación homogénea generalizada . . . 33
4.3.6. Modelo cinemático directo de posición (MCDP) . . . 34
4.3.7. Modelo cinemático inverso de posición (MCIP) . . . 35
4.3.8. Modelo cinemático directo de velocidad (MCDV) . . . 37
4.3.9. Modelo cinemático inverso de velocidad (MCIV) . . . 38
4.3.10. Modelo cinemático directo de aceleración (MCDA) . . . 38
4.3.11. Modelo cinemático inverso de aceleración (MCIA) . . . 39
4.4. Modelo dinámico . . . 39
4.4.1. Módulos de velocidad y alturas de centros de masa . . . 40
4.4.2. Energías cinéticas . . . 41
4.4.3. Energías potenciales . . . 42
4.4.4. Lagrangiano . . . 42
4.4.5. Par torsor . . . 42
4.4.6. Formulación Euler Lagrange . . . 43
4.5. Simulaciones digitales . . . 44
4.6. Control PID para robots manipuladores . . . 46
4.7. Simulación digital con un esquema de control . . . 47
4.8. Conclusiones . . . 48

5. Ambiente virtual: cinemática del PHANToM 1.0 **50**
5.1. Introducción . . . 50
5.2. Ambientes virtuales con OpenGL y Visual C++ . . . 50
5.2.1. OpenGL . . . 50
5.2.2. Visual C++ . . . 53
5.3. Desarrollo de un ambiente virtual empleando OpenGL . . . 55
5.4. Animación al ambiente virtual . . . 56
5.5. Contrucción del robot virtual . . . 58
5.6. Animación al robot virtual . . . 62
5.7. Integración y validación del modelo cinemático en el robot virtual . . . 64
5.8. Modelo conceptual . . . 66
5.8.1. Movimiento del PHANToM . . . 66
5.8.2. Enlace con Matlab . . . 67
5.8.3. Modificación del escenario, posición y orientación de la cámara . . . 68
5.8.4. Modo de operación . . . 69

5.8.5. Desplegue de opciones y letreros . . . 71
5.9. Conclusiones . . . 71

6. Conclusiones y perspectivas **72**
6.1. Conclusiones . . . 72
6.2. Trabajos a futuro . . . 73

A. Lista de abreviaturas **74**

B. Glosario **76**

C. Manual del usuario **78**
C.1. Requerimientos: . . . 78
C.2. Ejecución de la aplicación . . . 78
C.3. Operación de la Interfaz virtual . . . 79
C.4. Operación en línea . . . 81
C.4.1. Modo PHANToM . . . 81
C.4.2. Modo Simple . . . 81
C.5. Operación fuera de línea (desde archivo) . . . 81
C.5.1. Graba tarea y reproduce la tarea . . . 82
C.5.2. Seguimiento de una trayectoria de una circunferencia . . . 82
C.5.3. Seguimiento de una trayectoria de una rosa de 3 pétalos . . . 82
C.5.4. Simulación de oscilación libre de baja fricción . . . 82

D. Programas auxiliares **83**

E. Estrategias implementadas en la interfaz de visualización **84**
E.1. Switch con una tecla . . . 84
E.2. Desplegue temporal . . . 85
E.3. Lectura y escritura de datos . . . 86
E.4. Tiempo del reloj . . . 88
E.5. Implementación de zonas para activar empleando el mouse . . . 89
E.6. Desplegue de trayectoria . . . 89

F. Código de la interfaz virtual PHANToM 1.0 **91**

G. Programas de simulaciones **114**

H. Códigos para generar el archivo de trayectoria **115**
H.1. Código GrabaCircunferencia.m . . . 115
H.2. Código Circunferencia.m . . . 116
H.3. Código GrabaROSA.m . . . 117
H.4. Código Rosa.m . . . 117
H.5. Código GrabaOscilacionLBF.m . . . 118
H.6. Código OscilacionLBF.m . . . 118

I. Artículo arbitrado publicado: *Visualization and Control of Virtual Robot Manipulators* **119**

Bibliografía **120**

Índice de figuras

2.1. Ambiente Virtual en Java 3D del Robot PUMA 14
2.2. Interfaz virtual del robot IRB 1400 16
2.3. Vista de la interfaz gráfica del Robot PUMA Unimation 560 17

3.1. Resultados de simulación de un sistema [7] 19
3.2. Ambiente de Simnon 22
3.3. Portada del 20-sim 23
3.4. Portada de MatLab 24
3.5. Simulador de líneas de corte. 25
3.6. Simulador de control numérico. 26
3.7. Ejemplos de algunas formas de moldes. 27
3.8. Exterior de casa construida con AutoARQ. 28
3.9. Dibujo a mano con Squiggle. 29

4.1. Modelos PHANToM Premium 1.0, 1.5 y 3.0 31
4.2. a)Numeración de eslabones y articulaciones b)Cadena cinemática 31
4.3. Ubicación y sentido de θ_1 35
4.4. Triángulo formado por los eslabones L2, L3 y su proyección sobre el plano XY. . . 36
4.5. Localización de los centros de masa 40
4.6. Configuración incio de la simulación de oscilación libre de baja fricción 45
4.7. Gráficas de comportamiento de articulaciones para la oscilación libre con baja fricción 46
4.8. Gráfica en Matlab del seguimiento de trayectoria de una circunferencia 47
4.9. Comportamiento de los ángulos de las articulaciones para una circunferencia . . . 48
4.10. Gráfica en Matlab del seguimiento de trayectoria para una rosa de 3 pétalos . . . 49
4.11. Comportamiento de los ángulos de las articulaciones para una rosa de 3 pétalos . 49

5.1. Disposición de ejes en el espacio virtual con OpenGL 56
5.2. Diagrama para dibujar un objeto fijo y otro móvil 57
5.3. Aproximación con rectángulos 58
5.4. Descripción para dibujar 3 eslabones y proporcionarles movimiento 59
5.5. Configuración de eslabones con rectángulos 59
5.6. Paralepipedo a partir de 6 rectángulos 60
5.7. Configuración de eslabones con paralepipedos 60
5.8. Efecto de movimiento entre 2 eslabones 61

5.9. Localización del punto del efector final 61
5.10. Esferas en articulaciones 62
5.11. Orientación de los ejes de acuerdo al modelo cinemático 65
5.12. Configuración de eslabones al inicio del programa 65
5.13. Opciones para mover al PHANToM virtual 66
5.14. Enlace de Matlab con la interfáz de visualización 67
5.15. Modificación del escenario, posición y orientación de la cámara 68
5.16. Modos de operación de la interfáz de visualización 69
5.17. Comparación para las restricciones en las articulaciones 70
5.18. Desplegue de opciones 70

C.1. Entorno inicial 79

Índice de cuadros

2.1. Datos técnicos del robot PUMA [4] 13
2.2. Parámetros Denavit-Hartenberg del Robot PUMA [4] 13
2.3. Ángulos para las articulaciones del robot IRB 1400 [5] 15
2.4. Parámetros Denavit-Hartenberg del IRB 1400 [5] 16
2.5. Parámetros Denavit Hartenberg del robot PUMA Unimation 560 [6] 17

4.1. Parámetros Denavit-Hartenberg 33
4.2. Parámetros de fricción estática y dinámica [23] 44
4.3. Longitudes proporcionales y masas de los eslabones del PHANToM 1.0 [23] 44

Capítulo 1

Introducción

1.1. Simulación y ambientes virtuales

Actualmente, es necesario llevar a cabo simulaciones antes de implementar sistemas, esquemas o condiciones a un prototipo o sistema real. Es por ello, que la denominada realidad virtual (RV) está siendo ampliamente utilizada en áreas tan variadas como medicina, robótica, matemáticas, psicología, entre otras.

La RV es un entorno de 3 dimensiones donde se puede realizar la interacción con objetos o personas que no están presentes o que se localizan en algún lugar remoto.

Permite la modelación matemática para la creación de un espacio tridimensional, donde es posible generar objetos sencillos como un cubo o una esfera que son denominados primitivas; pero también los objetos pueden ser más complejos como un diseño arquitectónico, una estructura de ADN e inclusive algún estado nuevo de la materia. En un prototipo virtual es posible simular infinidad de situaciones para no arriesgar recursos.

La realidad virtual va mas allá de la simulación; se puede decir que es una simulación con las siguientes características [1]:

- Existe interacción.
- Es dinámica.
- Se produce en tiempo real.
- Y se presenta en tres dimensiones

Cuando se realiza una simulación donde los resultados son gráficas; es muy difícil que el usuario comprenda todo lo que ocurre; es más, si se trata de sistemas con movimiento en el espacio, es prácticamente imposible comprender e imaginar todo lo que está ocurriendo.

Por ello surge la necesidad de crear entornos tridimensionales que proporcionen otra perspectiva del sistema simulado [2].

La realidad virtual, ha tenido avances significativos en los últimos años, existen diferentes factores que permiten dar un mayor realismo a los mundos virtuales, uno de los principales es el tacto, por ello se han realizado estudios concernientes a los denominados *dispositivos hápticos*.

El término haptic (*háptica*) es utilizado por personas que se encargan del estudio de la psicofísica humana, quienes estudian cómo el hombre a través de sus manos, las utiliza para tocar, sentir y manipular objetos haciendo uso únicamente de sus modalidades sensoriales. [1].

Háptica es el estudio de cómo combinar el sentido humano del tacto con el mundo virtual generado por la computadora. Los sistemas actuales de realidad virtual carecen de estímulos táctiles. La investigación háptica trata precisamente de resolver este problema, consta de dos subcampos que son:

- Retroalimentación de fuerza, (kinestética) implementación de dispositivos que interactúan con los músculos y tendones y que son capaces de proporcionar una sensación de fuerza. Estos dispositivos son comúnmente robots manipuladores que proporcionan un fuerza de reacción correspondiente a los eventos del ambiente virtual.

- Retroalimentación táctil, implementación de dispositivos que interactuan con las terminales nerviosas de la piel que detectan calor, presión o texturas.

El dispositivo háptico PHANToM es un dispositivo empleado en varias partes del mundo por empresas como Boeing, NASA, etc. Una de las principales aplicaciones es en tele-manipulación para cirugías de mínima invasión.

Este dispositivo es producido por SensAble technologies y su costo depende del modelo. Por ejemplo *12.000 euros* el PHANTOM *Desktop* y 72.000 euros el PHANTOM *3.0/6DOF* [3].

Realizar un estudio del dispositivo háptico PHANToM por el momento ha sido completamente teórico, y se plantea la posibilidad de realizar un simulador virtual para estudiarlo y conocerlo mejor.

La simulación de robots cubre una necesidad importante para los estudiantes que no tienen acceso a un robot real y que lo han estudiado en forma teórica únicamente.

1.2. Simulación de robots

Existen algunos trabajos donde se ha realizado la simulación de robots clásicos, como en el Centro Superior de Informática de la Universidad de la Laguna España, donde se simula un robot PUMA [4]

En la Universidad Autónoma del Estado de Hidalgo se cuenta con un simulador también del robot PUMA [6].

En la Universidad Politécnica de Cartagena se cuenta con un robot IRB-1400 en su laboratorio para labores docentes [5].

Existen muchas herramientas para construir un robot simulado, desde VRLM, Java 3D o X3D, entre las principales y más comunes; cada una con diversas características y limitaciones.

La simulación del robot PUMA en la Universidad de la Laguna España, se hizo empleando JAVA3D, este lenguaje presenta una dificultad enorme para realizar movimientos de objetos independientes y consume bastantes recursos, generándose algunas latencias manifestadas con imágenes intermitentes, como lo mencionan en las conclusiones los autores de este proyecto.

En el caso del robot PUMA de la Universidad Autónoma del Estado de Hidalgo, emplearon Visual Basic debido al poderoso ambiente de desarrollo de su interfaz gráfica (GUI).

En el caso de la Universidad de Cartagena, emplearon visual C++ por la gestión automática de memoria y de ser un lenguaje propuesto por la compañía ABB que produce al robot.

1.3. Objetivo principal

Desarrollar una interfaz virtual que recree el comportamiento cinemático y dinámico del dispositivo háptico PHANToM 1.0 y que pueda ser extendida a cualquier dispositivo electromecánico parecido.

1.4. Objetivos específicos

1. Obtener los modelos cinemáticos para un robot de tres grados de libertad.
2. Obtener el modelo dinámico de un robot de tres grados de libertad.
3. Realizar el estudio de tareas contemplando los modelos cinemático y dinámico.

1.5. Justificación

El estudio y operación de robots generalmente se realiza con un robot físico y en muchos de los casos no es posible tener acceso a uno, debido a los altos costos que esto representa. Al emplear un robot simulado en un ambiente virtual no es necesario realizar tal inversión.

Algunos de los simuladores de robots operan satisfactoriamente, realizan tareas basadas en la cinemática, pero no contemplan la dinámica que un modelo real puede realizar. En este trabajo se propone la observación del comportamiento dinámico de un robot de 3 grados de libertad, caso particular, el dispositivo háptico PHANToM 1.0.

La realidad virtual es una herramienta para simular la operación dinámica de un robot, y colocando los modelos cinemáticos y dinámicos apropiados, se puede apreciar un comportamiento muy cercano al de un robot físico.

Con el sistema desarrollado en este trabajo se puede visualizar el comportamento de un robot de 3 grados de libertad antes de ejecutar la tarea físicamente.

Esta interfaz se podrá integrar con un teleoperador y realizar estudios del desempeño de tareas para controlar procesos remotos.

1.6. Tareas desarrolladas

Para desarrollar este trabajo fue necesario realizar las siguientes actividades:

- Obtener los modelos cinemáticos de acuerdo al algoritmo Denavit-Hartenberg.
- Obtener el modelo dinámico de acuerdo a la formulación Euler-Lagrange.
- Seleccionar el lenguaje para desarrollar la interfaz de visualización.

- Construir los eslabones del robot, empleando librerías de OpenGL.
- Dar movimiento al prototipo virtual del robot.
- Proporcionar mayor realismo al prototipo virtual, proporcionando texturas y articulaciones esféricas.
- Incorporar el modelo cinemático al ambiente virtual.
- Incorporar elementos para la interacción y modificación del aspecto.
- Incorporar los resultados de las simulaciones en Matlab del comportamiento dinámico.

1.7. Hipótesis

Empleando librerías de OpenGL para visual C++ es posible construir un visualizador del dispositivo háptico PHANToM 1.0 con bastante realismo, desarrollando la simulación en línea sin que exista interrupción en el proceso cuando el usuario interactué con él, además la visualización se realiza sin defecto intermitente debido al doble buffer empleado en OpenGL.

1.8. Aportes tecnológicos

Se proporciona una interfaz de visualización de un robot de tres grados de libertad, permitiendo hacer una observación del comportamiento cinemático y dinámico; abriendo la posibilidad de construir otro tipo de interfaz para robots con características similares.

1.9. Herramientas empleadas

Las herramientas empleadas para la realización de este proyecto son diversas, en primer lugar se emplea el algoritmo Denavit-Hartenberg y la formulación de Euler-Lagrange para la obtención de los modelos matemáticos, como también, el software Matlab 6 para la solución del modelo diferencial y generación de archivos y del programa MATEMATICA 5 para el procesamiento algebraico.

Para desarrollar la virtualización del dispositivo háptico PHANToM, se emplean librerías de OpenGL para Visual C++.

Capítulo 2

Estudio del estado del arte: simulación de robots manipuladores

A continuación se presentan algunos de los trabajos que se localizan en la internet y que tratan sobre la simulación de robots articulados, algunos son de tipo didáctico con códigos abiertos.

2.1. Simulación de robots articulados

El trabajo simulación de robots articulados fue realizado por :

- Benito José Cuesta Viera
- Juan Lucio Cruz Méndez
- Leopoldo Acosta Sánchez

En el Centro Superior de Informática de la Universidad de la Laguna, España [4].

Este trabajo es un ambiente virtual del robot PUMA, realizado en Java 3D, tiene la particularidad de tener la opción de manipular un robot PUMA real y poder observar el desempeño a través de una cámara de video.
La características físicas del robot Puma que se consideran para el desarrollo del proyecto, se presentan en la cuadro 2.1
Los parámetros Denavit-Hartenberg que se determinan y que son usados para la modelación se presentan en la cuadro 2.2
Las conclusiones que se proporcionan con relación a este proyecto son:

1. A partir de este proyecto, es posible desarrollar otros modelos de robot como el SCARA o STANDFORD.
2. La dificultad de implementar estos robots virtuales en Java 3D no es por falta de recursos en el API Java 3D, sino por el grado de conocimientos sobre técnicas de programación gráfica en 3D.

Característica	Valor
Movilidad Eje 0 (columna central)	210^0
Movilidad Eje 1 (hombro)	180^0
Movilidad Eje 2 (codo)	230^0
Elevación de la muñeca	140^0
Rotación de la muñeca	320^0
Apertura de la pinza	45mm
Presión de agarre	10N
Repetibilidad	2mm
Capacidad de elevación	1 Kg
Velocidad máxima	$46^0/seg$
Alcance (desde el centro del eje 1)	420mm
Entrada analógica	0 a +5Vcc
Entrada digital	TTL (+5Vcc)
Salida digital	TTL (+5Vcc/20mA
Alimentación	220 Vca 50-60 Hz
Tensión de salida (uso externo)	5Vcc/100mA

Cuadro 2.1: Datos técnicos del robot PUMA [4]

	d_i	θ_i	a_i	α_i
T01	L1	variable	0	90
T21	0	variable	L2	0
T23	0	variable	L3	0
T34	0	variable	0	90
T45	Lm	variable	0	0

Cuadro 2.2: Parámetros Denavit-Hartenberg del Robot PUMA [4]

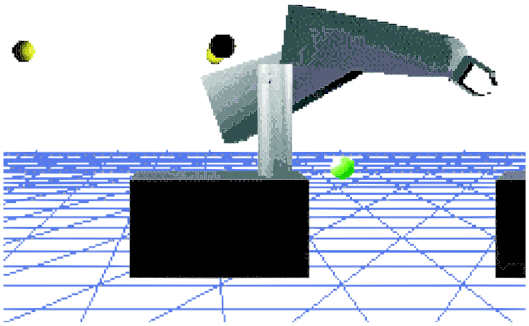


Figura 2.1: Ambiente Virtual en Java 3D del Robot PUMA

3. A veces este esfuerzo no brinda los resultados esperados, debido a que se disminuye el realismo por que el renderizado de la escena es demasiado lento.

Este proyecto se puede accesar en la página disponible en: http://www.csi.ull.es/ simvirt/index.html. El ambiente del robot simulado se presenta en la figura: 2.1

2.2. SIMUROB

SIMUROB es un proyecto desarrollado en el Departamento Ingeniería de Sistemas y Automática ETSII perteneciente a la Universidad Politécnica de Cartagena [5].

SimuRob es un simulador bajo Windows del robot antropomórfico IRB-1400; uno de estos robots se emplea para clases prácticas en la Universidad Politécnica de Cartagena.
Este simulador permite a los alumnos aprender el manejo del robot y la programación de sus movimientos, así como el conocimiento de la cinemática directa e inversa, alineación de ejes, configuraciones singulares etc.
Este robot puede ser programado por una consola que lleva consigo, esto condiciona a que solo un usuario trabaje con él, por esta razón surgió el proyecto de SimuRob para manipular a un robot IRB 1400 virtual.
Algunas características del robot IRB 1400 son mencionadas a continuación:

- El IRB 1400 fue diseñado por ABB Robotics AB, para tareas de ciclos rápidos de trabajo.
- El robot IRB-1400 es un robot de 6 grados de libertad y es utilizado para tareas de soldadura, recubrimientos, manipulación de objetos etc.
- Este robot fue concebido como una estructura abierta para establecer comunicación con sistemas externos y así formar parte de células de fabricación.
- El IRB-1400 está equipado con el sistema operativo Base Ware OS, que controla los aspectos del robot como los movimientos, que ejecuta, desarrollo y ejecución de programas, sistemas de comunicación etc.

Eje	ángulo max.	ángulo min.
1	170	-170
2	70	-70
3	70	-65
4	150	-150
5	115	-115
6	300	-300

Cuadro 2.3: Ángulos para las articulaciones del robot IRB 1400 [5]

- Los movimientos que puede ejecutar se resumen en la tabla 2.3
- El lenguaje de programación del IRB-1400 es RAPID, este lenguaje fue desarrollado también por ABB Robotics. Es un lenguaje estructurado que consiste en una serie de funciones que describen la tarea del robot; cada función tiene asociada una serie de parámetros .

A continuación se presentan las características del programa SimuRob:

1. SimuRob está escrito en Visual C++; se usó este lenguaje por ser empleado para la programación de simuladores de robots, debido a su gestión automatizada de memoria, además es un lenguaje que propone la casa ABB para la posible comunicación con el IRB-1400.
2. En el diseño y programación de SimuRob no se utilizan librerías gráficas externas, todas las funciones para la representación gráfica están implementadas en los archivos fuente del programa.
3. SimuRob permite programar movimientos y trayectorias que posteriormente pueden ser ejecutados en el robot real; el simulador incluye pinzas y pistola de soldadura.
4. El código del SimuRob es abierto y posibilita la creación de nuevos robots, probar algoritmos diferentes para la generación de trayectorias.
5. El código está abierto para realizar pruebas con la cinemática inversa.
6. SimuRob está constituido por una consola de programación, un robot virtual y una zona de edición de programas, como de muestra en la figura 2.2.

La modelación cinemática se realizó empleando las matrices de transformación homogénea y el método Denavit-Hartenberg; en la tabla 2.4 se presentan los parámetros Denavit-Hartenberg empleados.
En general, SimuRob permite:

- Simular el comportamiento cinemático del robot IRB-1400
- Guiar al robot virtual a través de una consola virtual que incluye un joystick 3D.
- Tomar diferentes objetos virtuales y cambiarlos de posición.

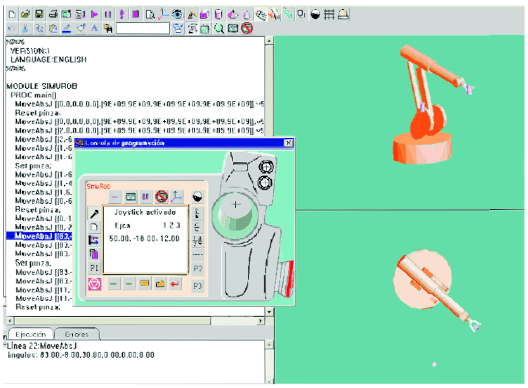


Figura 2.2: Interfaz virtual del robot IRB 1400

Eslabón i	θ_i	d_i	a_i	α_i
1	$\theta_1 + 90$	475	150	90
2	$\theta_2 + 90$	0	600	0
3	θ_3	0	120	90
4	θ_4	720	0	-90
5	θ_5	0	0	90
6	θ_6	85	0	0

Cuadro 2.4: Parámetros Denavit-Hartenberg del IRB 1400 [5]

- Ejecutar programas de movimientos mediante el guiado punto a punto de su extremo, muy similar al robot IRB-1400.
- Verificar y ejecutar programas, visualizando movimientos.
- Los programas realizados en el simulador pueden ser ejecutados en el robot real.

2.3. Software de simulación del robot PUMA Unimation 560 basado en la cinemática directa e inversa de posición

Los desarrolladores de este proyecto son:

- Jonathan Carrera Diaz
- Fernando Garrido Reséndiz
- Omar Arturo Domínguez Ramírez

En la Universidad Autónoma del Estado de Hidalgo [6]
Este trabajo simula el comportamiento cinemático de un robot PUMA Unimation 560; este programa tiene un panel de control donde se realiza la comunicación con el robot virtual, en él se accesan las coordenadas a las que se desea llevar al efector final y éste calcula la configuración

Eslabón i	α_i	a_i	d_i	θ_i
1	0	0	0	θ_1
2	-90	0	0	θ_2
3	0	a2	d3	θ_3
4	-90	a3	d4	θ_4
5	90	0	0	θ_5
6	-90	0	0	θ_6

Cuadro 2.5: Parámetros Denavit Hartenberg del robot PUMA Unimation 560 [6]

de las articulaciones, inclusive determina si no se trata de algún punto singular
La interfaz gráfica que resulta se muestra en la figura 2.3
La interfaz gráfica permite al usuario establecer las condiciones iniciales para realizar la simu-

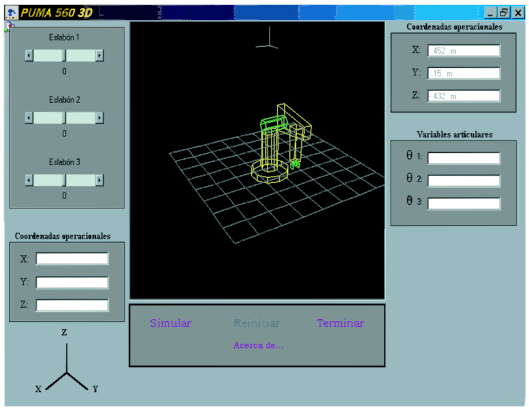


Figura 2.3: Vista de la interfaz gráfica del Robot PUMA Unimation 560

lación; una vez hecho esto, el usuario podrá observar mediante animación gráfica en tres dimensiones los movimientos del brazo robot en tiempo real, así como la simulación matemática de la cinemática directa e inversa de posición.
Este simulador es un sistema para la experimentación de la cinemática directa y la cinemática inversa para la posición del efector final de un robot industrial de seis grados de libertad, sin tomar en cuenta la orientación de dicho efector final.
Los parámetros Denavit-Hartenberg usados para el modelo del robot de este trabajo se presentan en la tabla 2.5

Para la elaboración del ambiente simulado se emplea una de las técnicas de proyección, denominadas proyecciones geométricas planares.

Capítulo 3

Visualizadores y simuladores

3.1. Introducción

En este capítulo se describe brevemente el proceso de simulación. Es importante señalar que el concepto de simulación es muy amplio y solo se menciona los que más se usan. Se describen algunos de los simuladores utilizados por ingenieros, economistas, diseñadores etc.

Describiremos rápidamente las características de Simnon, 20-sim, Matlab.

3.2. Simulación de procesos físicos

Un simulador es un programa para computadora que resuelve un problema de algún sistema físico en particular; modelado bajo las leyes que lo rigen tales como: leyes físicas, químicas, biológicas, de mercado; la información es presentada en la mayoría de las veces en función del tiempo, es decir como va evolucionando el sistema físico simulado de acuerdo al paso del tiempo.

La forma tradicional de presentar los datos de una simulación es a través de gráficas de comportamiento contra el tiempo; en el eje de las ordenadas se proporciona una característica analizada y en el eje de las abcisas se muestra el tiempo, como se muestra en la figura 3.1

Algunas simulaciones presentan la información combinando dos características, como por ejemplo las gráficas de par-velocidad de un motor, corriente-potencia de un circuito, temperatura-presión, por mencionar algunas.

Realizar una simulación tiene ventajas, algunas de ellas se mencionan a continuación:

- Es más fácil y accesible experimentar sobre un sistema simplificado; la inversión es mínima para la construcción de algún prototipo a escala o recreándolo en la computadora, solo eligiendo un software apropiado para la representación del experimento.
- Hay acceso total a la entrada y a la salida; el usuario puede controlar las condiciones en las cuales se está desarrollando la simulación y ésta le está proporcionando toda la información que él requiera.
- En muchas ocasiones es factible hacer pruebas en un sistema simulado que no pueden hacerse en un sistema real ya que podrían llevar al desastre; no siempre las pruebas son

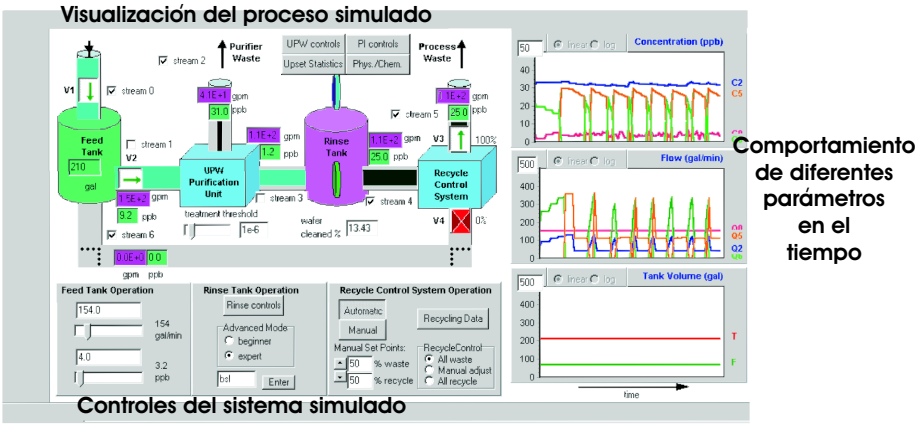


Figura 3.1: Resultados de simulación de un sistema [7]

experimentales y de observación del comportamiento, algunas son para medir el desempeño, poniendo en riesgo la integridad del equipo y de los usuarios.

- La experimentación en el sistema real puede ser imposible o contraria a la ética, si es necesario conocer los efectos de una catástrofe, pues definitivamente no es aceptable reproducirla en la realidad, como por ejemplo derrames de sustancias tóxicas, etc.

Los modelos resultantes de los sistemas para llevarlos a la simulación pueden ser:

- Deterministas: expresan matemáticamente sin incertidumbre las relaciones entre las variables. El modelo asigna unívocamente valores y/o funciones ciertas y determinadas a la información que procesa.

- Estocásticos: se expresan las relaciones con incertidumbre entre las variables mediante conceptos probabilísticos usando variables aleatorias. Dichas relaciones son descritas, es decir que los valores que toma a la largo del tiempo no son determinados con certeza. absoluta [14].

3.3. Simuladores de sistemas digitales

Los simuladores permiten verificar que varias representaciones sean equivalentes desde el punto de vista de su comportamiento. Estos programas son ejecutables y permiten ejercitar el funcionamiento de un sistema digital; además trabajan sobre una representación dada y con un conjunto de señales de entrada (vectores de prueba).

Mientras más abstracta es la representación del sistema digital, menor es el tiempo de máquina requerido para ejercitar el modelo. Sin embargo, entre más abstracto es el modelo, menor es la precisión sobre el comportamiento temporal y las formas de onda. Los simuladores a nivel de circuitos eléctricos son muy precisos, sin embargo son muy ineficientes ya que sólo permiten simular algunos cientos de compuertas en un tiempo razonable [8].

Existen diferentes formas de realizar una simulación en un sistema digital, tales como:

- Simulación analógica: el modelo planteado del sistema físico está basado principalmente en expresiones diferenciales, es decir que es continuo o continuo en intervalos; la manera de resolverlo puede ser a través de elementos externos a una computadora como por ejemplo prototipos a escala o arreglos de amplificadores operacionales, o puede ser resuelto íntegramente por un software de computadora apropiado.

- Simulación digital: sucede integramente dentro de la computadora, el modelo del sistema es completamente discreto, esto quiere decir que su comportamiento cambia solo en los instantes concretos de tiempo.

  Toda la información del modelo físico está comprendida en un programa de computadora que se encarga de resolverlo y de presentar resultados a través del monitor u otros dispositivos de salida.

- Simulación híbrida: este tipo de simulaciónes realizan una combinación entre los modelos continuo y el discreto para obtener una solución general de ambos.

Actualmente, los simuladores más utilizados son los digitales, debido a que es sencillo construirlos, modificarlos, presentar resultados. Básicamente trabajan en las computadoras digitales.

El sitio en Internet de Computer Aided Control System Design [9] hace mención de las características de algunos simuladores para ingenniería:

- **ABACUSS** (Advanced Batch And Continuous Unsteady-State Simulator): desarrollado para sistemas de ingeniería química, soporta modelos híbridos, modelos de herencia, modelos categorizados de descomposición. Resuelve ecuaciones algebraicas diferenciales de índices elevados, dinámicas y de estado estable, sensitividad dinámica y análisis de incertidumbre.

- **DAEPACK**: es una librería de software para cálculos numéricos generales. Se divide en dos librerías principales:

  1. Análisis simbólico y transformación.
  2. Cálculo numérico.

  La librería de análisis simbólico y transformación consiste en componentes para el análisis general de modelos Fortran-90 y generación automática de información requerida cuando se usen algoritmos numéricos modernos.

- **AnyLogic**: es un entorno virtual a nivel prototipo y de aplicación profesional; habilita al usuario rapidamente para construir un modelo de simulación con desempeño en el entorno. Incluye objetos físicos y usuarios humanos. La tecnología de modelado está basada en UML-RT, java y ecuaciones diferenciales algebraicas. AnyLogic ofrece un rango de librerías específicas.

- **BaSiP**: es un simulador desarrollado como manejador de producción multi-propósitos complejos para plantas productoras.
- **DOORS**: es un simulador distribuido orientado a objetos de tiempo real [10]
- **Dymola**: proporciona un poderoso modelado orientado a objetos y simulación de entornos virtuales para educación e ingeniería profesional.
- **gPROMS**: representa el estado del arte en procesos de modelado, y tecnología de simulación y optimización tecnológica.
- **HYBRSIM**: es una implementación de un modelado gráfico híbrido y herramienta de si mulación. Esto envuelve un conjunto de principios físicos que rigen los cambios discontinuos en un modelo de un sistema físico.
- **Model Vision 3.0**: es un entorno orientado a objetos para el diseño de grandes sistemas dinámicos con características para soperte para B-Charts, para comportamiento híbrido específico, métodos numéricos de ODEPACK y de la colección Hairer-Norsett-Wanner, soporta matrices y vectores de datos, incluye un dispositivo de clases estandar, librerías de animación y wizards para la rápida creación sketches animados y genera un completo y portable soporte para Win32 y Java de modelos ejecutables.
- **OmSim**: es un lenguaje orientado a objetos para modelado de eventos de sistemas dinámicos continuos y discretos en el tiempo.
- **SHIFT**: es un lenguaje de programación para describir redes dinámicas de autómatas híbridos. Tales sistemas consisten de componentes, los cuales pueden ser creados, interconectados y destruidos. Los componentes exhiben un comportamiento híbrido, que consiste en fases de tiempo continuo separadas por transiciones de eventos discretos.
- **Smile**: es una herramienta de simulación para sistemas de energía.

A continuación mencionamos algunos de los programas de análisis matemático y simulación, comunmemte utilizados:

3.3.1. Simnon

Simnon es un programa que está diseñado para resolver ecuaciones algebraicas, ecuaciones diferenciales y ecuaciones simultáneas de sistemas dinámicos.

El sistema a resolver puede ser descrito como una interconexión de subsistemas cuyo comportamiento está caracterizado por ecuaciones diferenciales o por otro tipo de expresiones.

Estos modelos pueden ser matemáticos, biológicos, económicos o de diversas áreas de la ingeniería.

Simnon es un software interactivo que facilita la tarea al usuario, escribiendo los comandos, parámetros, condiciones iniciales, etc.

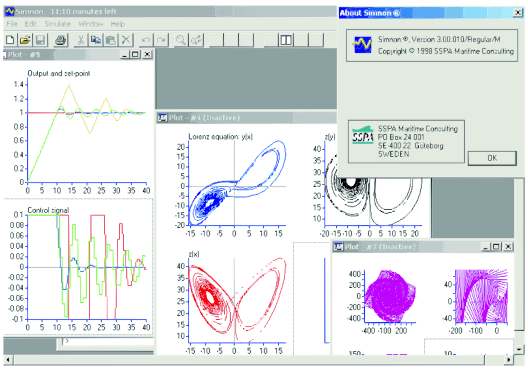


Figura 3.2: Ambiente de Simnon

Los resultados pueden ser desplegados gráficamente o numéricamente en la pantalla, las gráficas pueden ser copiadas al porta papeles de windows.

El módulo para resolver ecuaciones diferenciales no lineales, puede ser visto como una calculadora de ecuaciones diferenciales, con la diferencia de que despliega los resultandos en una gráfica que va construyendo.

Para resolver un problema con expresiones diferenciales, se pueden emplear los siguientes pasos:

1. Realizar la descripción del sistema: reescribir la ecuación en términos de estado-espacio estandar que es usado por Simnon

2. Ejecutar la simulación: analizar el resultado: en ocaciones cuando se realiza la simulación, encuentra que el procesamiento no es el esperado. Entonces es posible interrumpir la simulación.

3. Modificar parámetros y condiciones iniciales: lo interesante de la simulación es explorar diferentes soluciones que dependen de los parámetros.

Para la simulación de control de sistemas digitales; un proceso continuo controlado por computadora es descrito en Simnon como la interconexión de sistemas; el proceso puede ser representado como un sistema continuo y el control digital como un sistema discreto.

Simnon tiene un módulo para el intercambio de datos con otros programas que soporta:

- DDE-links (enlace de intercambio dinámico de datos)
- Importar y exportar series de tiempo
- Exportar datos de gráficas (plots)
- Importación lineal de sistemas invariantes en el tiempo

Simnon posee un módulo para realizar simulación en tiempo real, con este módulo es posible que una computadora controle un proceso físico continuo.

En la figura 3.2 se muestran alguos ejemplos de desplegue de datos, resultados de simulaciones.

En general Simnon cuenta con más de 40 comandos para diversos modelos de sistemas [11].

Figura 3.3: Portada del 20-sim

3.3.2. 20-sim

Es un programa de simulación y modelado que corre bajo Microsoft Windows y Unix de Sun Microsystems.
20-sim consiste en 2 ventanas principales y varios grupos de herramientas. La primera ventana es el editor y la segunda es el simulador.
El editor es usado para definir y editar los modelos; estos pueden ser creados usando:

1. Modelo de ecuaciones: éste puede ser único (modelo principal) o un modelo de bajo nivel de jerarquía (sub modelo)

2. Modelo de diagramas de bloques: igual que el modelo de ecuaciones, éste puede ser único o un sub modelo. Los diagramas muestran claramente el flujo de información y puede ser separado en distintos bloques.

3. Los modelos gráficos bond son muy usados para modelar sistemas físicos debido a que muestran claramente una resemblanza con los componentes del sistema físico.

4. Modelos con diagramas de íconos: es también usado para modelar sistemas físicos, empleando precisamente íconos de resistores, inductancias, fuentes, etc.

El modelo se puede definir construyéndolo con partes existentes almacenadas en varias librerías. En la figura 3.3 se muestra la portada de 20-sim.

20-sim es capaz de simular el comportamiento de sistemas dinámicos, tales como sistemas eléctricos, mecánicos e hidráulicos o cualquier combinación de estos sistemas.

3.3.3. Matlab

Matlab se puede definir de manera general como un lenguaje técnico de computación. Es un programa interactivo que ayuda a realizar cómputo numérico y visualización de datos, está diseñado

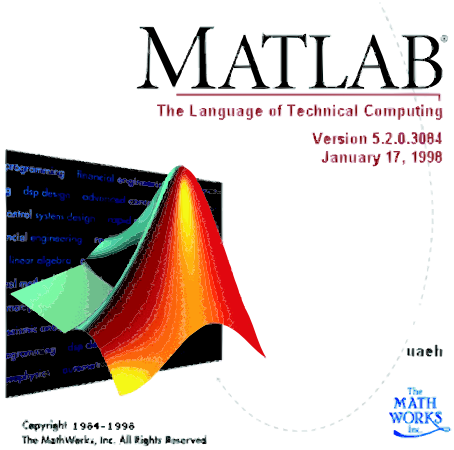


Figura 3.4: Portada de MatLab

para trabajar con matrices de gran tamaño. Esta herramienta proporciona solución para problemas en matemáticas, química, física, ingeniería, finanzas y en la mayoría de áreas donde se requieran cálculos complejos. MatLab tiene la capacidad de trabajar con LINPACK y EISPACK, que fueron desarrollados originalmente en Fortran, Matlab es a la vez un entorno de trabajo y un lenguaje de programación, cuenta con un Toolbox (caja de herramientas), que es un conjunto de M-files (programas en Matlab) construidos especialmente para resolver problemas específicos. Estas herramientas resuelven problemas en los campos de control, procesamiento de señales, identificación de sistemas entre otras. Así mismo cuenta con una sección denominada Simulink, que es una herramienta para modelar, analizar y simular físicamente sistemas matemáticos, incluyendo en su procesamiento a elementos no lineales, así como elementos continuos y discretos, ver figura 3.4 [12].

3.4. Simuladores virtuales

Las simulaciones también pueden auxiliarse de los entornos virtuales, donde se requiera una visualización; para lograrlo es necesario una transformación de números en imágenes interactivas.

Se requiere convertir cantidades masivas de datos, así como realizar bastantes cálculos complejos.

Una vez hecho todo este trabajo computacional, es muy sencillo observar la información trasformada en imágenes y es posible tener una apreciación global de los resultados de una simulación.

Existen bastantes simuladores virtuales y es dificil mencionarlos a todos, pero la mayoría están involucrados en la simulación espacial, programas para vuelos virtuales, laboratorios virtuales, simulación virtual de ecosistemas, cirugía virtual, entre otros.

El anuario 2003-2004 CaD menciona software que presenta soluciones para las aplicaciones 3D, a continuación de presentan algunas [13].

3.4.1. AutoForm

AutoForm integra una línea de módulos de software para el diseño de piezas de chapa metálica y matrices de estampación. Dispone de módulos especializados que realizan las siguientes funciones:

- Evaluación de la viabilidad de estampado de diseños de piezas durante el desarrollo del producto y en las etapas iniciales del diseño de la herramienta.
- Generación de la superficie de la matriz o herramienta de manera rápida e interactiva para validarla posteriormente.
- Validación de los diseños de la matriz y el proceso de estampado a través de simulaciones virtuales, y determinar automáticamente la geometría óptima de la herramienta y las condiciones del proceso de estampación.
- El módulo AutoForm-UserInterface proporciona un entorno virtual de integración de todos los módulos y por ello usuarios de AutoForm de diferentes departamentos o empresas pueden compartir e intercambiar datos.

Este software es utilizado por: diseñadores de producto, ingenieros, proyectistas, diseñadores de matrices, ingenieros de producción, fabricantes de herramienta, estampadores, y especialistas en simulación. La figura 3.5 presenta una muesta de una pieza después de haber sido procesada [16].

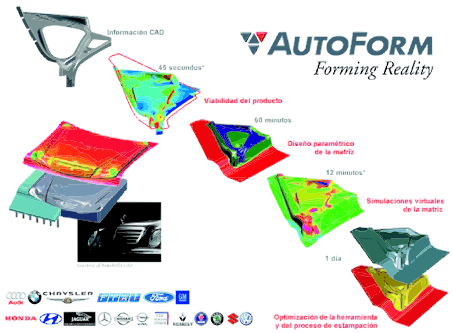


Figura 3.5: Simulador de líneas de corte.

3.4.2. Vericut

Es un software de verificación NC(control numérico). Simula el mecanizado NC para detectar errores en la trayectoria de la herramienta antes de la prueba en máquina. Este programa trabaja con códigos G , salidas CAM y funciona con UNIX y Windows. El programa tiene tres funciones principales:

1. Simulación/verificación y análisis de la pieza de trabajo. Simula interactivamente el fresado multiejes, el taladro, el torneado, el EDM y operaciones combinadas de fresado y torneado. Se verifican constantemente las medidas durante la simulación. Compara la pieza meca-

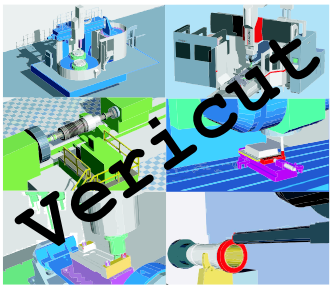


Figura 3.6: Simulador de control numérico.

nizada con la geometría original, garantizando la correspondencia entre la pieza acabada y el diseño deseado.

2. Optimización automática del recorrido de la pieza. Según las condiciones de corte y la capacidad de la herramienta NC, el módulo OptiPath de VERICUT modifica automáticamente los programas NC para volverlos más rápidos y eficaces.

3. Simulación de la máquina y del control NC. Posee la verificación de colisiones para máquinas de NC más precisa y rápida disponible. La máquina simulada es accionada por las mismas funciones de control, lo que le permite funcionar exactamente como la que tiene en su taller, y asi evitar una descompostura de máquina que podría costar mucho y retrasar todo el plan de producción.

La figura 3.6 muestra algunos escenarios del entorno virtual [17].

3.4.3. WorkNC-CAD

Software especializado para crear moldes, matrices y utencilios, disponible de forma autónoma o integrado con WorkNC. Este programa reduce el tiempo de desarrollo en múltiples etapas del proceso de diseño y fabricación, y mejora la productividad y la capacidad de reacción. Algunas de las características de este programa son:

- Los usuarios disponen de herramientas necesarias para diseñar y construir utillajes de forma autónoma sin depender de un sistema CAD.

- La integración total entre el CAD y el CAM simplifica los procesos constructivos. Durante la preparación del mecanizado WorkNC-CAD permite la extensión de superficies, definición de puntos de taladro, tapado de agujeros, extracción de curvas y contorno de resto de material para la creación automatizada de electrodos.

- Permite trabajar de forma directa sobre cualquier tipo de pieza inicial sin tener que convertirla o repararla. En pocos minutos se analizan los ángulos de incidencia, radios, superficies planas, alturas y realización de secciones dinámicas que permiten validar rápidamente la factibilidad de la pieza.

- Comparación de ficheros: frecuentemente se producen y se reciben modificaciones en la geometría de la pieza. Por ello, este software incluye funciones que comparan dos ficheros automáticamente en búsqueda de sus diferencias, que se visualizan en capas de distintos colores. Esto permite evitar errores y aporta máxima rapidez y fiabilidad.
- Separación Macho-Cavidad: en pocos minutos y simplemente seleccionando las superficies deseadas se puede separar automáticamente el macho de la cavidad en todo tipo de piezas. Las superficies verticales se colocan en una tercera capa.
- Factor de escala: este software es capaz de aplicar factores de escala individuales a los ejes X, Y y Z de una pieza. Esta función se usa por ejemplo al crear electrodos.
- Modelado 3D de elementos del utillaje: este software incluye todas las funciones necesarias para la generación de modelos de elementos activos del utillaje (superficies de junta, cortes para moldes de soplado, extensión tangencial de superficies, recorte y modificación de superficies dinámicamente, etc.)
- Creación de superficies de protección y tapado rápido de agujeros: este software permite la corrección de agujeros e imperfecciones de la pieza, y el rápido relleno de superficies complejas manteniendo tangencia con múltiples superficies adjuntas.

En la figura 3.7 se presentan algunos de los moldes en el entorno virtual [24].

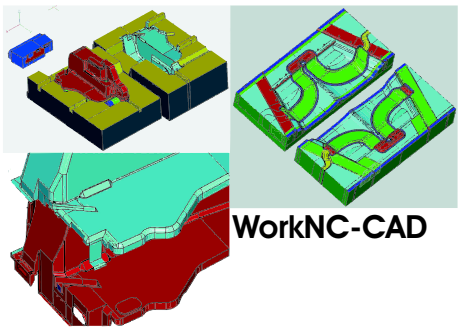


Figura 3.7: Ejemplos de algunas formas de moldes.

3.4.4. AutoARQ

AutoARQ 2000+ es una herramienta que personaliza y potencia AutoCAD para el dibujo de elementos arquitectónicos en 2D y 3D, así como para la creación de imágenes fotorrealísticas. Algunas de sus caracteristicas son:

- Incorpora un motor mucho más potente y rápido para la gestión de objetos. Trabajando sobre AutoCAD, permite, desde cualquier plano en 2D, levantar el volumen de cualquier proyecto. Para ello utiliza sencillos objetos arquitectónicos como paredes, puertas, ventanas, pilares, escaleras, forjados, etc.

- Permite la asignación a cada uno de los elementos de dibujo de códigos de medición para realizar listados de calidades y presupuestos. De igual manera automatiza la generación de listados de carpinterías, etc.
- Permite automatizar la mayoría de las labores rutinarias de modificación y ajuste. Un objeto ventana sólo puede ser copiado a un objeto pared y, automáticamente, genera el hueco. Al reconocer la escalera del piso inferior, nos indica si queremos crear su hueco, permitiendo indicar una altura de paso y ajustando el hueco al escalón correspondiente.
- Todos los elementos arquitectónicos, y alguno de AutoCAD, son objetos. Estos objetos están relacionados entre ellos, adaptándose al dibujo según sus características. De esta forma un pilar, al desplazarlo, siempre genera el arreglo y recubrimiento correspondiente a las paredes que afecta, y la pared en la que estaba situado arregla automáticamente el hueco que había dejado.

La figura 3.8 muestra una imagen de lo que se puede construir con AutoARQ. [15]

Figura 3.8: Exterior de casa construida con AutoARQ.

3.4.5. Squiggle

El programa Squiggle permite crear presentaciones de dibujos CAD en minutos aplicándoles efectos de mano alzada. Este software comparte trabajos con: AutoCAD, AutoCAD LT, AutoSketch, Allplan, Cadkey, CorelCAD, CorelDraw, Corel VisualCAD, DataCAD, DesignCAD, Drawbase, DynaCAD, EasyCAD, MicroStation, TurboCAD, o cualquier CAD, dibujo, o aplicación gráfica con salidas a ficheros HPGL/2, .DWG o .DXF.

El programa muestra el fichero en pantalla y permite variar el ancho y el color de sus entidades (hasta 255). Se puede aplicar bastantes estilos de mano alzada a cada línea. Cada línea es manipulada independientemente y los resultados no son predecibles exactamente, igual que la mano humana. El usuario parte de siete estilos predeterminados pero puede crear nuevos efectos y almacenarlos para posteriores usos. El programa también le permite exportarlos como archivos bitmap (.bmp, .tif, .jpeg o .gif), para ser importados en otras aplicaciones [25].

Figura 3.9: Dibujo a mano con Squiggle.

3.5. Conclusiones

Existe software de simulación para diversos propósitos, tratar de mencionarlos a todos es un trabajo complicado; en la dirección de internet dir.nodeworks.com se tiene un directorio de 166 links de software de simulación. Entre algunos de los programas, se tienen simuladores de procesos químicos, de sistemas dinámicos de autómatas, línea de corte de chapa, diseño arquitectónico. La simulación es un proceso compuesto principalmente de dos partes que son: la modelación, que involucra la representación de un sistema físico a través de expresiones ya sean continuas o discretas. La solución del modelo, que puede ser a través de prototipos a escala o enteramente a través de la computadora. Cabe mencionar que las simulaciones en línea dependen directamente del tamaño del modelo, como se mencionó anteriormente, entre mas abstracto sea el modelo, el tiempo de resolución es menor y se acerca a la observación en tiempo real del desempeño del sistema físico.

Éste capítulo tuvo como objetivo dar a conocer algunas de las diferentes herramientas, para simulación y visualización, mas utilizadas en el mercado y que ubican en contexto el presente trabajo de investigación aplicado a la visualización y simulación de movimientos de robots manipuladores, cuyo caso de estudio es el robot PHANToM 1.0 descrito técnicamente en el siguiente capítulo.

Capítulo 4

Dispositivo háptico PHANToM 1.0

4.1. Introducción

En este capítulo se hace una breve descripción del dispositivo háptico PHANToM 1.0 y de manera general se mencionan las características de un dispositivo clásico; más adelante se describe la forma de obtener los modelos cinemático directo e inverso de: posición, velocidad y de aceleración; para después continuar con el modelo dinámico, empleando el método Euler - Lagrange [21].

Cabe mencionar que algunas de las operaciones matriciales y simplificaciones de ecuaciones, se realizaron con el software Mathematica 5.

Empleando los modelos, se realizan algunas simulaciones sustituyendo los parámetros físicos de un dispositivo háptico específico; estas simulaciones se realizaron con el software de Matlab 6.0.

Posteriormente se describe la implementación de una estrategia de control cartesiano y se realizan nuevamente algunas simulaciones para verificar el desempeño del control.

4.2. PHANToM 1.0

El PHANToM fué desarrollado en el MIT en 1993 por el Dr. Salisbury y el Dr. Mandayam Srinivasan y es producido por la compañía SensAble Technologies.

El PHANToM (Personal HAptic iNTerface Mechanism) es un dispositivo electromecánico que maneja el movimiento y fuerza a la vez; es un elemento doble debido a que el usuario es capaz de manipular objetos virtuales y ser retroalimentado de fuerza proveniente del ambiente virtual.

El PHANToM mide la posición del efector final y proporciona una reflexión de fuerza entre un usuario humano y la computadora.

Este dispositivo ha sido dispuesto para interactuar y sentir diferentes formas de objetos virtuales.

Para operar el PHANToM el usuario se conecta al mecanismo con simplemente introducir su dedo en el thimble (dedal) o sujetando el Stylus (lápiz), como se puede apreciar en la figura 4.1

Figura 4.1: Modelos PHANToM Premium 1.0, 1.5 y 3.0

Este dispositivo detecta el movimiento y puede activar una influencia externa de fuerza, creando una ilusión completa de tocar objetos físicos sólidos.
La cantidad de fuerza reflejada depende del modelo de PHANToM, por ejemplo los modelos Premium 1.0 y Premium 1.5 proporcionan 8,5 N y el modelo Premium 3.0 proporciona 22 N.
Estos modelos se pueden apreciar en la figura 4.1 [18] [19]

4.3. Modelo cinemático

El modelo cinemático se refiere al conjunto de ecuaciones que determinan la posición, velocidad y aceleración del efector final en función de los ángulos de las articulaciones, sin tomar en cuenta las causas que producen el movimiento (fenómeno dinámico).

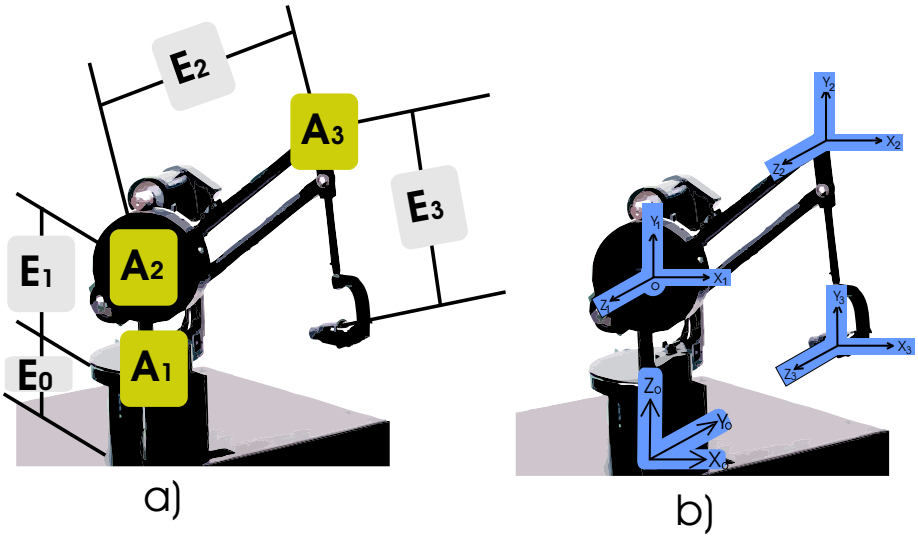


Figura 4.2: a)Numeración de eslabones y articulaciones b)Cadena cinemática

4.3.1. Metodología para la obtención del modelo cinemático directo de posición (MCDP)

La metodología para obtener el MCDP se basa en la definición de:

1. La cadena cinemática.
2. Los marcos ortonormales de referencia.
3. Los parámetros de Denavit-Hartenberg.
4. Las matrices elementales.
5. La matriz de transformación homogénea generalizada.
6. El MCDP en términos de las coordenadas cartesianas para posición y ángulos de Bryant para orientación.

4.3.2. Cadena cinemática

La cadena cinemática está constituida por los eslabones, las articulaciones y en su extremo final por un órgano terminal o efector final. Los eslabones y articulaciones se enumeran comenzando por 1 hasta n (número de grados de libertad), considerando a la base fija del robot como el eslabón cero, tal y como se muestra en la figura 4.2 a)
Existen reglas para construir cadenas cinemáticas de manipuladores, tal como las utilizadas para describir la del PHANToM 1.0 [20].

4.3.3. Marcos ortonormales

La definición de un marco ortonormal, permite tener una referencia para conocer la situación de cada uno de los eslabones y del efector final, con respecto a la base (referencia fija).
Los marcos de referencia se construyen bajo los siguientes criterios [21]:

1. A los ejes de giro de las articulaciones, se designarán como ejes $\hat{Z}$
2. El origen O se localiza en la normal común a los ejes de las articulaciones $\hat{Z}_1$ y $\hat{Z}_{i+1}$.
   Si los ejes de las articulaciones son paralelos o están alineados, la perpendicular común se selecciona arbitrariamente.
   Es recomendable seguir un criterio de simplicidad.
3. El vector $\hat{X}_{i+1}$ se define sobre la perpendicular común a los ejes de las articulaciones $\hat{Z}_i$ y $\hat{Z}_{i+1}$; es decir por el producto cruz: $\hat{Z}_i \times \hat{Z}_{i+1}$.
   Si los ejes de las dos articulaciones están alineados, la orientación de $\hat{X}_{i+1}$ es arbitraria.
4. $\hat{Y}_{i+1}$se define de tal manera que se complete un marco dextrogiro, es decir:
   $\hat{Y}_{i+1} = \hat{Z}_{i+1} \times \hat{X}_{i+1}$

Los marcos de referencia y la cadena cinemática resultante se muestra en la figura 4.2 b)

Eslabón i	θ_i	d_i	a_i	α_i
1	θ_1	0	0	$-\frac{\pi}{2}$
2	θ_2	0	L_2	0
3	θ_3	0	L_3	0

Cuadro 4.1: Parámetros Denavit-Hartenberg

4.3.4. Parámetros Denavit-Hartenberg (DH)

Los parámetros DH son cuatro términos geométricos asociados con cada elemento que permiten describir una articulación prismática o de revolución y se definen a continuación [22]:

θ_i : Es el ángulo de rotación alrededor del eje $\hat{Z}_{i-1}$ medido de $\hat{X}_{i-1}$ a $\hat{Y}_{i-1}$, el sentido está definido por la regla de la mano derecha.

d_i : Es la distancia de traslación a lo largo del eje $\hat{Z}_i$.

a_i : Es la distancia de traslación sobre el eje $\hat{X}_i$

α_i : Es el ángulo de separación del eje $\hat{Z}_i$ al eje $\hat{Z}_{i+1}$ medido respecto a $\hat{Z}_{i+1}$

Los parámetros DH que se obtienen se presentan en la tabla 4.1

4.3.5. Matrices elementales y de transformación homogénea generalizada

La matriz de transformación elemental se obtiene sustituyendo los parámetros Denavit Hartenberg en la ecuación matricial (4.1) [21].

$$ {}^{i}_{i+1}T = \begin{bmatrix} cos\theta_i & -sen\theta_i cos\alpha_i & sen\theta_i sen\alpha_i & a_i cos\theta_i \\ sen\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sen\alpha_i & a_i sen\theta_i \\ 0 & sen\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1} $$

Sustituyendo los parámetros DH se obtienen las matrices elementales : (4.2), (4.3) y (4.4),

$$ {}^{1}_{2}T = \begin{bmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2} $$

$$ {}^{2}_{3}T = \begin{bmatrix} c_2 & -s_2 & 0 & L_2 c_2 \\ s_2 & c_2 & 0 & L_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3} $$

$$ {}^{3}_{4}T = \begin{bmatrix} c_3 & -s_3 & 0 & L_3c_3 \\ s_3 & c_3 & 0 & L_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4} $$

La matriz de transformación homogénea generalizada se obtiene multiplicando las 3 matrices elementales, como se indica en la ecuación: (4.5) [21]

$$ \left[{}^{1}_{4}T\right] = \left[{}^{1}_{2}T\right]\left[{}^{2}_{3}T\right]\left[{}^{3}_{4}T\right] \tag{4.5} $$

Empleando identidades trigonométricas para la suma de ángulos y realizando simplificaciones en la matriz: (4.5) se obtiene la matriz: (4.6)

$$ {}^{1}_{4}T = \begin{bmatrix} c_1c_{23} & -c_1s_{23} & -s_1 & L_3c_1c_{23} + L_2c_1c_2 \\ s_1c_{23} & -s_1s_{23} & c_1 & L_3s_1c_{23} + L_2s_1c_2 \\ -s_{23} & -c_{23} & 0 & -L_3s_{23} - L_2s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6} $$

Para facilitar la escritura, se emplea: $\cos(\theta_1) = c_1$, $\cos(\theta_2 + \theta_3) = c_{23}$, $sen(\theta_1) = s_1$ y $sen(\theta_2 + \theta_3) = s_{23}$

4.3.6. Modelo cinemático directo de posición (MCDP)

El MCDP es el conjunto de ecuaciones que proporcionan las coordenadas operacionales a partir de los ángulos de las articulaciones, así como los ángulos de giro de Bryant aplicados al primer marco de referencia para hacerlo coincidir con el último.

- Coordenadas operacionales: los elementos ${}^{1}_{4}T\,(1,4)\,,{}^{1}_{4}T\,(2,4)\,y{}^{1}_{4}T\,(3,4)$ de la matriz (4.6) definen las coordenadas operacionales:

$$ \mathrm{x} = \mathrm{L_3c_1c_{23}}+\mathrm{L_2c_1c_2} \tag{4.7} $$

$$ y = L_3s_1c_{23} + L_2s_1c_2 \tag{4.8} $$

$$ z = -L_3s_{23} - L_2s_2 \tag{4.9} $$

- Ángulos de Bryant: los ángulos λ,μ y ν son los ángulos de giro que se deben aplicar sobre los ejes XYZ para hacer coincidir al primer marco con el 4° marco de referencia.
  Los valores se determinan con las siguientes ecuaciones: (4.10), (4.11) y (4.12) [21].

$$ \lambda = \mathrm{atan2}(-\mathrm{E_{23}}, \mathrm{E_{33}}) \tag{4.10} $$

$$ \mu = \mathrm{atan2}(\mathrm{E_{13}}, \mathrm{E_{33}}/\mathrm{Cos}\lambda) \tag{4.11} $$

$$ \upsilon = \mathrm{atan2}(-\mathrm{E_{12}}, \mathrm{E_{11}}) \tag{4.12} $$

La función $atan2(y, x)$ es la misma función arco tangente $atan\left(\frac{y}{x}\right)$, solo que en esta se tiene un rango de 0 a 360^0, es decir un desempeño en los cuatro cuadrantes. Los ángulos resultantes de las ecuaciones (4.10),(4.11) y (4.12) son:

$$\lambda = \frac{3}{4}\pi \tag{4.13}$$

$$\mu = 0 \tag{4.14}$$

$$\upsilon = \theta_2 + \theta_3 \tag{4.15}$$

Estos ángulos son obtenidos debido a que el organo terminal es un dedal sin actuador y no una pinza actuada.

Finalmente las ecuaciones que definen el modelo cinemático directo de posición son: (4.7), (4.8), (4.9), (4.13), (4.14) y (4.15)

Los cálculos se realizan a partir de la referencia en la segunda articulación (origen definido), pero la orientación de los ejes se hace de acuerdo al marco de referencia de la primera articulación [21].

4.3.7. Modelo cinemático inverso de posición (MCIP)

Las ecuaciones que a partir de los valores de las coordenadas operacionales proporcionan el valor de las variables articulares o ángulos en las articulaciones se les denomina modelo cinemático inverso de posición (MCIP).

Obtención θ_1 : Este ángulo es sencillo de obtener, si observamos la figura 4.3 que se construye

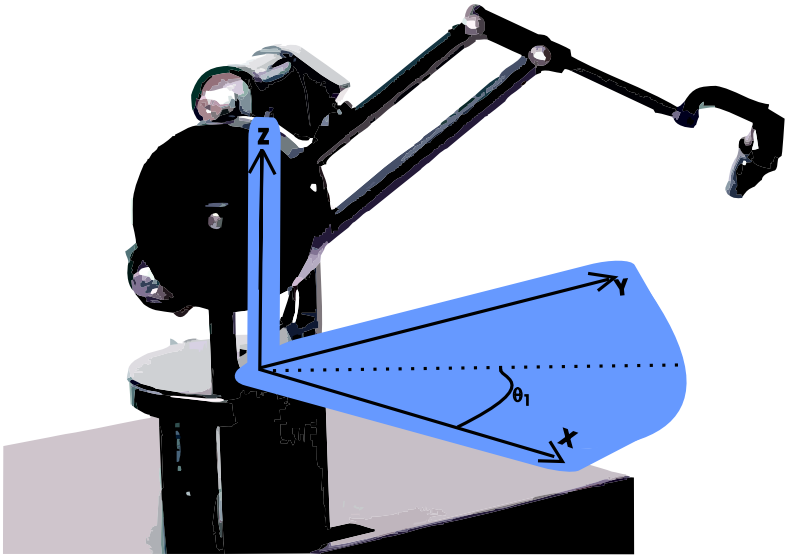


Figura 4.3: Ubicación y sentido de θ_1

en torno a la primera articulación; entonces se obtiene la ecuación (4.16)

$$\theta_1 = \text{atan2(y, x)} \tag{4.16}$$

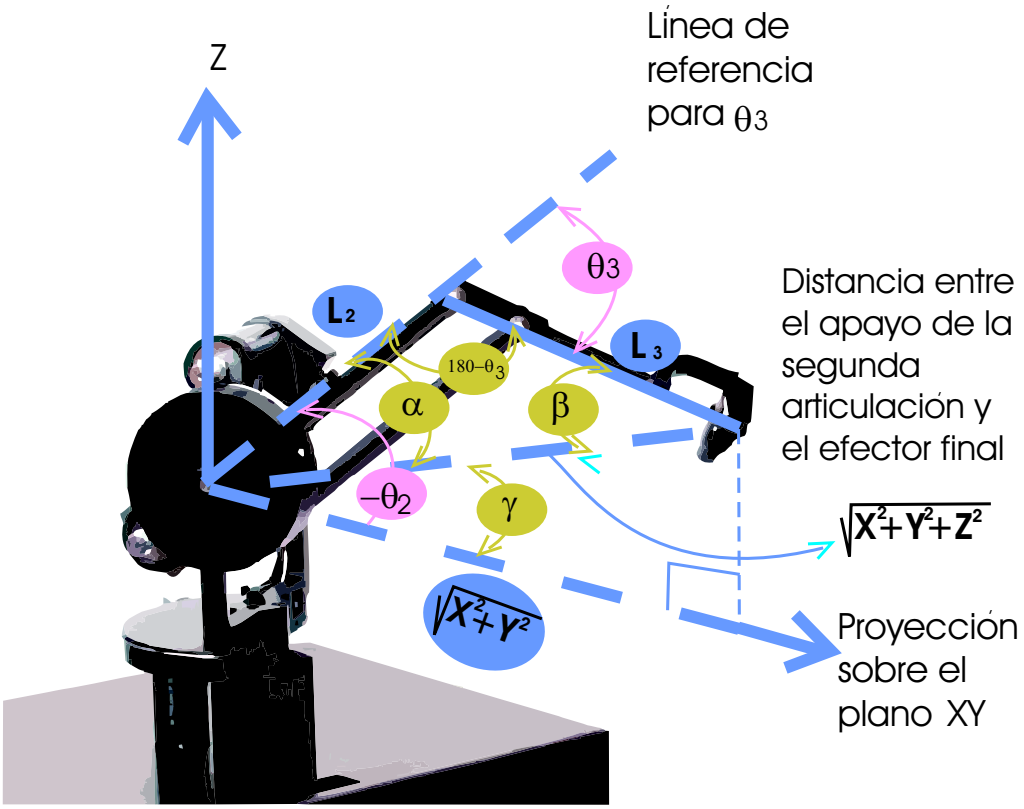


Figura 4.4: Triángulo formado por los eslabones L2, L3 y su proyección sobre el plano XY.

Obtención θ_3 : por conveniencia se determina primero el valor de θ_3 antes que el valor de θ_2. Para el análisis, consideramos el esquema derivado de la cadena cinemática, observando la figura 4.4 conformada por los eslabones L2 y L3:
Estableciendo relaciones en la figura 4.4 y aplicando la ley de cosenos se obtiene la ecuación: (4.17)

$$cos(\theta_3) = \frac{x^2 + y^2 + z^2 - L_2^2 - L_3^2}{2L_2L_3} \tag{4.17}$$

Para relacionar la función atan2, se obtiene la ecuación (4.18)

$$sen\,(\theta_3) = \sqrt{1 - \cos^2(\theta_3)} \tag{4.18}$$

para finalmente obtener la ecuación: (4.19) [21]

$$\theta_3 = atan2(sen(\theta_3), \cos(\theta_3)) \tag{4.19}$$

Obtención de θ_2 : tomando en cuenta que el ángulo θ_2 se mide en sentido de las manecillas del reloj y observando nuevamente la figura 4.4 se obtiene la ecuación: (4.20)

$$\theta_2 = -\alpha - \gamma \tag{4.20}$$

y también la ecuación: (4.21)

$$\gamma = atan2(z, \sqrt{x^2 + y^2}) \tag{4.21}$$

Sabiendo que la suma de los ángulos internos de un triángulo es de 180^0, podemos relacionar β con θ_2 y α, para obtener la ecuación: (4.22)

$$\beta = \theta_3 - \alpha \tag{4.22}$$

Desarrollando la identidad trigonométrica de $sen\,(A+B)$ y empleando la ley de senos para los ángulos α y β se llega a obtener la espresión: (4.23)

$$\alpha = atan2\left(L_3 sen\left(\theta_3\right), L_2 + L_3\cos\left(\theta_3\right)\right) \tag{4.23}$$

Sustituyendo las ecuaciones (4.23) y (4.21) en (4.20) se obtiene la ecuación: (4.24)

$$\theta_2 = -atan2\left(L_3 sen(\theta 3), L_2 + L_3 cos(\theta_3)\right) - atan2\left(z, \sqrt{x^2+y^2}\right) \tag{4.24}$$

4.3.8. Modelo cinemático directo de velocidad (MCDV)

Proporciona la velocidad operacional en términos de las coordenadas operacionales del efector final de las variables articulares y sus derivadas respecto al tiempo, como se muestra en la ecuación (4.25)

$$\dot{X} = \frac{df(q)}{dt} \tag{4.25}$$

Donde:
$\dot{X}$ = Vector de velocidades $\left[\dot{x}, \dot{y}, \dot{z}, \dot{\lambda}, \dot{\mu}, \dot{\gamma}\right]^T$
Para determinar el resultado de la ecuación (4.25), se hace uso del jacobiano como se indica en la ecuación (4.26) [21]

$$\dot{X} = [J]\,\dot{q} \tag{4.26}$$

Donde:
$\dot{q}$ = Vector de velocidades articulares $\left[\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3\right]^t$
Las entradas para la matriz jacobiana se determinan de acuerdo a la forma mostrada en la ecuación (4.27)

$$[J_{ij}] = \frac{\partial f_i}{\partial q_j} \tag{4.27}$$

Aplicando la derivada parcial a cada término del MCDP para obtener el jacobiano y multiplicando a éste por el vector de velocidades angulares, se obtiene la ecuación matricial (4.28)

$$\dot{X} = \begin{bmatrix} -\left(c_2 L_2 s_1 - c_{23} L_3 s_1\right)\dot{\theta}_1 + \left(-c_1 L_2 s_2 - c_1 L_3 s_{23}\right)\dot{\theta}_2 - \left(c_1 L_3 s_{23}\right)\dot{\theta}_3 \\ \left(c_1 c_2 L_2 + c_1 c_{23} L_3\right)\dot{\theta}_1 + \left(-L_2 s_1 s_2 - L_3 s_1 s_{23}\right)\dot{\theta}_2 - \left(L_3 s_1 s_{23}\right)\dot{\theta}_3 \\ \left(-c_2 L_2 - c_{23} L_3\right)\dot{\theta}_2 - \left(c_{23} L_3\right)\dot{\theta}_3 \\ 0 \\ 0 \\ \dot{\theta}_2 + \dot{\theta}_3 \end{bmatrix} \tag{4.28}$$

4.3.9. Modelo cinemático inverso de velocidad (MCIV)

El MCIV es un conjunto de ecuaciones que determinan las velocidades de las variables angulares en función de las operacionales tal y como se presenta en la ecuación (4.29)

$$\dot{q} = [J_*]^{-1} \dot{X}_* \tag{4.29}$$

donde:
$[J_*]^{-1}$ es la matriz inversa del jacobiano reducido, esta matriz se contruye a partir del jacobiano, solo se eliminan los 3 últimos renglones, es decir $J \in R^{3x3}$ y así poder calcular su inversa.
Entonces el MCIV que se obtiene se presenta en la ecuación matricial (4.30)

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} \frac{-s_1\dot{x}+c_1\dot{y}}{c_2L_2+c_{23}L_3} \\ \frac{c_1c_{23}\dot{x}+c_{23}s_1\dot{y}-s_{23}\dot{z}}{L_2s_3} \\ -\frac{c_1c_2L_2\dot{x}+c_1c_{23}L_3\dot{x}+c_2L_2s_1\dot{y}+c_{23}L_3s_1\dot{y}-L_2s_2\dot{z}-L_3s_{23}\dot{z}}{L_2L_3s_3} \end{bmatrix} \tag{4.30}$$

4.3.10. Modelo cinemático directo de aceleración (MCDA)

Este conjunto de ecuaciones proporciona la aceleración de los términos de las variables operacionales del efector final en función de la primera y segunda derivadas de las variables articulares.

Para determinar el MCDA se deriva respecto al tiempo nuevamente el MCDV como se indica en las ecuaciones: (4.31) y (4.31)

$$\frac{d}{dt}\left[\dot{X}\right] = \frac{d}{dt}\left[J\dot{q}\right] \tag{4.31}$$

$$\ddot{X} = J\ddot{q} + J\dot{q} \tag{4.32}$$

Las ecuaciones que se obtienen después de realizar las operaciones indicadas y simplificaciones son: (4.33), (4.34), (4.35), (4.36), (4.37) y (4.38)

$$\begin{aligned} \ddot{x} = &-c_1\left[c_2L_2\left(\dot{\theta}_1^2+\dot{\theta}_2^2\right)+c_{23}L_3\left(\dot{\theta}_1^2+\left(\dot{\theta}_2+\dot{\theta}_3\right)^2\right)+L_2\ddot{\theta}_2s_2+L_3\ddot{\theta}_2s_{23}+L_3\ddot{\theta}_3s_{23}\right]- \\ &s_1\left(c_2L_2\ddot{\theta}_1+c_{23}L_3\ddot{\theta}_1-2\dot{\theta}_1\left[L_2\dot{\theta}_2s_2+L_3\left(\dot{\theta}_2+\dot{\theta}_3\right)s_{23}\right]\right) \end{aligned} \tag{4.33}$$

$$\begin{aligned} \ddot{y} = &-s_1\left[c_2L_2\left[\dot{\theta}_1^2+\dot{\theta}_2^2\right]+c_{23}L_3\left[\dot{\theta}_1^2+\left(\dot{\theta}_2+\dot{\theta}_3\right)^2\right]+L_2\ddot{\theta}_2s_2+L_3\ddot{\theta}_2s_{23}+L_3\ddot{\theta}_3s_{23}\right]+ \\ &c_1\left[c_2L_2\ddot{\theta}_1+c_{23}L_3\ddot{\theta}_1-2\dot{\theta}_1\left(L_2\dot{\theta}_2s_2+L_3\left(\dot{\theta}_2+\dot{\theta}_3\right)s_{23}\right)\right] \end{aligned} \tag{4.34}$$

$$\ddot{z} = -c_2L_2\ddot{\theta}_2 - c_{23}L_3(\ddot{\theta}_2+\ddot{\theta}_3) + L_2\dot{\theta}_2^2s_2 + L_3\dot{\theta}_2^2s_{23} + 2L_3\dot{\theta}_2\dot{\theta}_3s_{23} + L_3\dot{\theta}_3^2s_{23} \tag{4.35}$$

$$\ddot{\lambda} = 0 \tag{4.36}$$

$$\ddot{\mu} = 0 \tag{4.37}$$

$$\ddot{v} = \ddot{\theta}_2 + \ddot{\theta}_3 \tag{4.38}$$

4.3.11. Modelo cinemático inverso de aceleración (MCIA)

El último modelo cinemático que se requiere obtener es el inverso de aceleración; este modelo proporciona el valor de la aceleración de las variables articulares a partir de las coordenadas operacionales y de las velocidades articulares, como se indica en la ecuación matricial (4.39)

$$\ddot{q} = J_{\bullet}^{-1}\left[\ddot{X}_{\bullet} - \dot{J}_{\bullet}\dot{q}\right] \tag{4.39}$$

Donde:
$\ddot{X}_{\bullet}$ es el vector de variables operacionales reducido.
$\dot{J}_{\bullet}$ es la derivada del jacobiano reducido.
Realizando las operaciones indicadas y nuevamente realizando simplificaciones se obtienen las ecuaciones: (4.40), (4.41) y (4.42).

$$\ddot{\theta}_1 = \frac{2c_1^2\dot{\theta}_1(L_2\dot{\theta}_2 s_2 + L_3(\dot{\theta}_2+\dot{\theta}_3)s_{23}) + s_1(2L_2\dot{\theta}_1\dot{\theta}_2 s_1 s_2 + 2L_3\dot{\theta}_1(\dot{\theta}_2+\dot{\theta}_3)s_1 s_{23} - \ddot{x}) + c_1\ddot{y}}{c_2L_2 + c_{23}L_3} \tag{4.40}$$

Debido a que las ecuaciones (4.41) y (4.42) son muy grandes, se escriben de otra manera para su mejor comprensión.

$$\ddot{\theta}_2 = \frac{1}{L_2 s_3}\{A_1 + A_2 + A_3 + A_4\} \tag{4.41}$$

Donde:
$A_1 = c_1^2 c_{23}\left(c_2L_2\left[\dot{\theta}_1^2 + \dot{\theta}_2^2\right] + c_{23}L_3\left[\dot{\theta}_2^2 + \left(\dot{\theta}_2+\dot{\theta}_3\right)^2\right]\right)$
$A_2 = c_{23}^2L_3\dot{\theta}_1^2 s_1^2 + c_{23}^2L_3\dot{\theta}_2^2 s_1^2 + c_2c_{23}L_2\left[\dot{\theta}_1^2+\dot{\theta}_2^2\right]s_1^2 + 2c_{23}^2L_3\dot{\theta}_2\dot{\theta}_3 s_1^2$
$A_3 = c_{23}^2L_3\dot{\theta}_3^2 s_1^2 + L_2\dot{\theta}_2^2 s_2 s_{23} + L_3\dot{\theta}_2^2 s_{23}^2 + 2L_3\dot{\theta}_2\dot{\theta}_3 s_{23}^2 + L_3\dot{\theta}_3^2 s_{23}^2$
$A_4 = c_1c_{23}\ddot{x} + c_{23}s_1\ddot{y} - s_{23}\ddot{z}$

$$\ddot{\theta}_3 = -\frac{1}{L_2L_3s_3}\{B_1 + B_2 + B_3 + B_4 + B_5 + B_6\} \tag{4.42}$$

Donde:
$B_1 = c_1^2(c_2^2L_2^2(\dot{\theta}_1^2+\dot{\theta}_2^2) + c_2c_{23}L_2L_3(2\dot{\theta}_1^2 + 2\dot{\theta}_2^2 + 2\dot{\theta}_2\dot{\theta}_3 + \dot{\theta}_3^2)$
$B_2 = c_{23}^2L_3^2(\dot{\theta}_1^2 + (\dot{\theta}_2+\dot{\theta}_3)^2)) + c_{23}^2L_3^2\dot{\theta}_1^2 s_1^2 + c_{23}^2L_3^2\dot{\theta}_2^2 s_1^2$
$B_3 = c_2^2L_2^2(\dot{\theta}_1^2+\dot{\theta}_2^2)s_1^2 + 2c_{23}^2L_3^2\dot{\theta}_2\dot{\theta}_3 s_1^2 + c_{23}^2L_3^2\dot{\theta}_3^2 s_1^2 + L_2^2\dot{\theta}_2^2 s_2^2$
$B_4 = 2L_2L_3\dot{\theta}_2^2 s_2 s_{23} + 2L_2L_3\dot{\theta}_2\dot{\theta}_3 s_2 s_{23} + L_2L_3\dot{\theta}_3^2 s_2 s_{23} + L_3^2\dot{\theta}_2^2 s_{23}^2$
$B_5 = 2L_3^2\dot{\theta}_2\dot{\theta}_3 s_{23}^2 + L_3^2\dot{\theta}_3^2 s_{23}^2 + c_1(c_2L_2 + c_{23}L_3)\ddot{x} + c_{23}L_3s_1\ddot{y}$
$B_6 = c_2L_2s_1(c_{23}L_3(2\dot{\theta}_1^2 + 2\dot{\theta}_2^2 + 2\dot{\theta}_2\dot{\theta}_3 + \dot{\theta}_3^2)s_1 + \ddot{y}) - L_2s_2\ddot{z} - L_3s_{23}\ddot{z}$
Nota: Las simplificaciones de las ecuaciones (4.40), (4.41) y (4.42) se realizaron con el apoyo del software Mathematica 5.

4.4. Modelo dinámico

El modelo dinámico se entenderá como el conjunto de ecuaciones que caracterizan a los factores que intervienen en el movimiento del dispositivo.

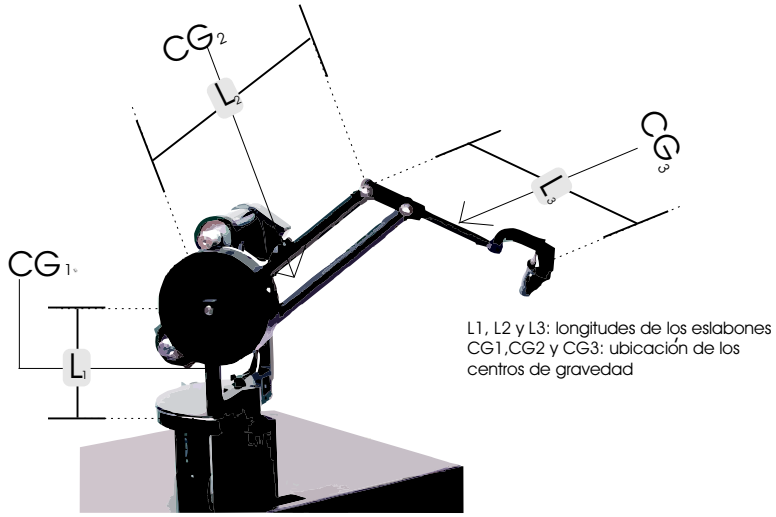


Figura 4.5: Localización de los centros de masa

Se considera que los eslabones son de material homogéneo y de forma regular, esto da lugar a localizar los centros de gravedad a la mitad de la longitud de cada eslabón como se observa en la figura 4.5

4.4.1. Módulos de velocidad y alturas de centros de masa

Las coordenadas operacionales para el primer eslabón, referidas al origen son:

$$x_1 = 0 \tag{4.43}$$

$$y_1 = 0 \tag{4.44}$$

$$z_1 = 0 \tag{4.45}$$

y sus derivadas respectivamente:

$$\dot{x}_1 = 0 \tag{4.46}$$

$$\dot{y}_1 = 0 \tag{4.47}$$

$$\dot{z}_1 = 0 \tag{4.48}$$

El módulo cuadrado de la velocidad se obtiene a través de la ecuación (4.49)

$$v_1^2 = \dot{x}_1^2 + \dot{y}_1^2 + \dot{z}_1^2 = 0 \tag{4.49}$$

y la altura del centro de masa 1 (CG1) es:

$$h_1 = -L_{CG1} \tag{4.50}$$

para el segundo eslabón, las coordenadas operacionales se obtienen a través del producto matricial: (4.51)

$$X_2 = \left[{}^1_2T\right]\left[{}^2_3T\right] \tag{4.51}$$

Resultando entonces las ecuaciones: (4.52), (4.53) y (4.54)

$$x_2 = L_2 c_1 c_2 \tag{4.52}$$

$$y_2 = L_2 s_1 s_2 \tag{4.53}$$

$$z_2 = -L_2 s_2 \tag{4.54}$$

Las derivadas de las ecuaciones anteriores son: (4.55),(4.56) y (4.57)

$$\dot{x}_2 = -L_2 s_1 c_2 \dot{\theta}_1 - L_2 s_2 c_1 \dot{\theta}_2 \tag{4.55}$$

$$\dot{y}_1 = L_2 c_1 c_2 \dot{\theta}_1 - L_2 s_1 s_2 \dot{\theta}_2 \tag{4.56}$$

$$\dot{z}_1 = -L_2 c_2 \dot{\theta}_2 \tag{4.57}$$

El módulo de velocidad simplificado que resulta y altura del centro de masa quedan determinados por las ecuaciones: (4.58) y (4.59)

$$v_2^2 = L_2^2 c_2^2 \dot{\theta}_1^2 + L_2^2 \dot{\theta}_2^2 \tag{4.58}$$

$$h_2 = -L_{CG2} s_2 = -\frac{1}{2} L_2 s_2 \tag{4.59}$$

Nota: Para la altura del centro de gravedad para el segundo eslabón se considera la coordenada operacional vertical z.
Para el tercer eslabón se hace la misma operación, empleando los MCDP y MCDV para la altura del centro de gravedad y el módulo de velocidad.
Las ecuaciones del módulo de velocidad simplificado y la altura del centro de gravedad para el eslabón tres son: (4.60), (4.61)

$$\begin{aligned} v_3^2 = &-\left(c_2 L_2 s_1 - c_{23} L_3 s_1\right)\dot{\theta}_1 + \left(-c_1 L_2 s_2 - c_1 L_3 s_{23}\right)\dot{\theta}_2 - \left(c_1 L_3 s_{23}\right)\dot{\theta}_3 + \left(c_1 c_2 L_2 + c_1 c_{23} L_3\right)\dot{\theta}_1 + \\ &\left(-L_2 s_1 s_2 - L_3 s_1 s_{23}\right)\dot{\theta}_2 - \left(L_3 s_1 s_{23}\right)\dot{\theta}_3 + \left(-c_2 L_2 - c_{23} L_3\right)\dot{\theta}_2 - \left(c_{23} L_3\right)\dot{\theta}_3 \end{aligned} \tag{4.60}$$

$$h_3 = -L_2 s_2 - L_{CG3} s_{23} = -L_2 s_2 - \frac{1}{2} L_3 s_{23} \tag{4.61}$$

4.4.2. Energías cinéticas

La energía cinética de un cuerpo es debida a su masa y a la velocidad que tiene y está definida por la ecuación (4.62)

$$K = \frac{1}{2} m v^2 \tag{4.62}$$

Las energías cinéticas para cada eslabón se indican en las ecuaciones: (4.63), (4.64) y (4.65)

$$k_1 = 0 \tag{4.63}$$

$$k_2 = \frac{1}{2} m_2 L_2^2 c_2^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 L_2^2 \dot{\theta}_2^2 \tag{4.64}$$

$$k_3 = \frac{1}{2} m_3 L_2^2 \dot{\theta}_2^2 + \frac{1}{2} m_3 L_3^2 \left(\dot{\theta}_2 + \dot{\theta}_3\right)^2 + L_2 L_3 c_3 \dot{\theta}_2 \left(\dot{\theta}_2 + \dot{\theta}_3\right) + \frac{1}{2} m_3 \left(L_2 c_2 \dot{\theta}_1 + L_3 c_{23} \dot{\theta}_1\right)^2 \tag{4.65}$$

4.4.3. Energías potenciales

Por definición, la energía potencial mecánica es la energía que adquiere un cuerpo debido a su posición dentro de una campo gravitatorio, es decir, la energía potencial está en función de la masa, la aceleración gravitacional y la altura, como se muestra en la ecuación (4.66)

$$P = mgh \tag{4.66}$$

Sustituyendo las alturas, la energía potencia para cada eslabón se indica en las ecuaciones: (4.67), (4.69) y (4.69)

$$p_1 = -m_1 g L_{CG1} \tag{4.67}$$

$$p_2 = -m_2 g L_{CG2} s_2 \tag{4.68}$$

$$p_3 = -m_3 g L_2 s_2 - m_3 g L_{CG3} s_{23} \tag{4.69}$$

4.4.4. Lagrangiano

El lagrangiano establece una relación entre las energías cinéticas y potenciales, como se muestra en la ecuación (4.70) [21]

$$L = \sum_{i=1}^{n} k_i - \sum_{i=1}^{n} p_i \tag{4.70}$$

El valor para n, en este caso de estudio será de 3. Realizando la sustitución de las energías cinéticas y potenciales y realizando las simplificaciones respectivas se obtiene la ecuación (4.71)

$$L = \left\{ \begin{array}{l} \left[\frac{1}{2} m_2 L_2^2 c_2^2 + \frac{1}{2} m_3 L_2^2 c_2^2 + m_3 L_2 L_3 c_2 c_{23} + \frac{1}{2} m_3 L_3^2 c_{23}^2\right] \dot{\theta}_1^2 + \\ \left[\frac{1}{2} m_2 L_2^2 + \frac{1}{2} m_3 L_2^2 + \frac{1}{2} m_3 L_3^2 + m_3 L_2 L_3 c_3\right] \dot{\theta}_2^2 + \left[\frac{1}{2} m_3 L_3^2\right] \dot{\theta}_3^2 + \left[m_3 L_3^2 + m_3 L_2 L_3 c_3\right] \dot{\theta}_2 \dot{\theta}_3 - \\ m_1 g L_{CG1} + m_2 g L_{CG2} s_2 + m_3 g \left[L_2 s_2 + L_{CG3} s_{23}\right] \end{array} \right\} \tag{4.71}$$

4.4.5. Par torsor

La ecuación para determinar el par torsor, esta definida por la relación de Euler - Lagrange mostrada en la ecuación (4.72)

$$T_i = \frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}_i} - \frac{\partial L}{\partial \theta_i} \tag{4.72}$$

Aplicando la ecuación (4.72), realizando las simplificaciones correspondientes, se obtienen los pares torsores 1,2 y 3 que se muestran en las ecuaciones: (4.73),(4.74) y (4.75)

$$T_1 = \left\{ \begin{array}{l} \left[\left(m_2 + m_3\right) L_2^2 c_2^2 + 2 m_3 L_2 L_3 c_2 c_{23} + m_3 L_3^2 c_{23}^2\right] \dot{\theta}_1 - \\ \left[2\left(m_2 + m_3\right) L_2^2 s_2 c_2 + 2 m_3 L_2 L_3 \left(s_{23} c_2 + s_2 c_{23}\right) + 2 m_3 L_3^2 s_{23} c_{23}\right] \dot{\theta}_1 \dot{\theta}_2 - \\ \left[2 m_3 L_2 L_3 s_{23} c_2 + 2 m_3 L_3^2 s_{23} c_{23}\right] \dot{\theta}_1 \dot{\theta}_3 \end{array} \right\} \tag{4.73}$$

$$T_2 = \left\{ \begin{array}{l} \left[\left(m_2 + m_3\right) L_2^2 + m_3 L_3^2 + 2 m_3 L_2 L_3 c_3\right] \ddot{\theta}_2 + \left[m_3 L_3^2 + m_3 L_2 L_3 c_3\right] \ddot{\theta}_3 - \left[2 m_3 L_2 L_3 s_3\right] \dot{\theta}_2 \dot{\theta}_3 \\ + \left[\left(m_2 + m_3\right) L_2^2 s_2 c_2 + m_3 L_2 L_3 \left(c_2 s_{23} + c_{23} s_2\right) + m_3 L_3^2 c_{23} s_{23}\right] \dot{\theta}_1^2 - \left[m_3 L_2 L_3 s_3\right] \dot{\theta}_3^2 \\ - m_2 g L_{CG2} c_2 - m_3 g \left[L_2 c_2 + L_{CG3} c_{23}\right] \end{array} \right\} \tag{4.74}$$

$$T_3 = \left\{ \begin{array}{l} \left(m_3 L_3^2 + m_3 L_2 L_3 c_3\right) \ddot{\theta}_2 + \left(m_3 L_3^2\right) \ddot{\theta}_3 + \left(m_3 L_2 L_3 c_2 s_{23} + m_3 L_3^2 s_{23} c_{23}\right) \dot{\theta}_1^2 + \\ \left(m_3 L_2 L_3 s_3\right) \dot{\theta}_2^2 - m_3 g L_{CG3} c_{23} \end{array} \right\} \tag{4.75}$$

4.4.6. Formulación Euler Lagrange

La formulación dinámica para un robot de eslabones articulados está definida por la ecuación 4.76 [23]

$$H\left(\theta\right) \ddot{\theta} + \left\{ B_0 + C\left(\theta, \dot{\theta}\right) \right\} \dot{\theta} + G\left(\theta\right) = \tau \tag{4.76}$$

Donde cada parte de la ecuación es:

$H\left(\theta\right) \in \Re^{n \times n}$ es una matriz que representa los componentes de inercia de los eslabones.

$B_0 \in \Re^{3 \times 3}$ es una matriz diagonal que representa los componentes de fricción en cada articulación.

$C\left(\theta, \dot{\theta}\right) \in \Re^{n \times n}$ es una matriz que representa los elementos de fuerzas de coriolis y centripetas.

$G\left(\theta\right) \in \Re^{n}$ es un vector que representa los elementos de fuerzas gravitatorias.

$\tau \in \Re^{n}$ es el vector de par de entrada.

Las matrices se definen de la siguiente forma:

$$H\left(\theta\right) = \left[\begin{array}{ccc} h_{11} & 0 & 0 \\ 0 & h_{22} & h_{23} \\ 0 & h_{32} & h_{33} \end{array} \right] \tag{4.77}$$

Donde:
$h_{11} = \left(m_2 + m_3\right) L_2^2 c_2^2 + 2 m_3 L_2 L_3 c_2 c_{23} + m_3 L_3^2 c_{23}^2$
$h_{22} = \left(m_2 + m_3\right) L_2^2 + m_3 L_3^2 + 2 m_3 L_2 L_3 c_3$
$h_{23} = m_3 L_3^2 + m_3 L_2 L_3 c_3$
$h_{32} = m_3 L_3^2 + m_3 L_2 L_3 c_3$
$h_{33} = m_3 L_3^2$

$$C\left(\theta, \dot{\theta}\right) = \left[\begin{array}{ccc} 0 & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & 0 \end{array} \right] \tag{4.78}$$

Donde:
$c_{12} = -2 \left\{ \left(m_2 + m_3\right) L_2^2 s_2 c_2 + m_3 L_3^2 s_{23} c_{23} \right\} \dot{\theta}_1 - 2 m_3 L_2 L_3 \left(s_{23} c_2 + s_2 c_{23}\right) \dot{\theta}_1$
$c_{13} = -2 \left\{ m_3 L_2 L_3 c_2 + m_3 L_3^2 c_{23} \right\} s_{23} \dot{\theta}_1$
$c_{21} = \left\{ \left(m_2 + m_3\right) L_2^2 s_2 c_2 + m_3 L_3^2 s_{23} c_{23} \right\} \theta_1 + m_3 L_2 L_3 \left(s_{23} c_2 + s_2 c_{23}\right) \theta_1$
$c_{22} = -2 m_3 L_2 L_3 s_3 \dot{\theta}_3$
$c_{23} = -m_3 L_2 L_3 s_3 \dot{\theta}_3$
$c_{31} = \left\{ m_3 L_2 L_3 c_2 + m_3 L_3^2 c_{23} \right\} s_{23} \dot{\theta}_1$

Eslabón i	Friccion estática	Fricción dinámica
1	0.3	0.1
2	0.15	0.05
3	0.05	0.01

Cuadro 4.2: Parámetros de fricción estática y dinámica [23]

Eslabón i	Longitud (dm)	Masa (Kg)
1	1.2	0.45
2	1.6	0.25
3	1.7	0.15

Cuadro 4.3: Longitudes proporcionales y masas de los eslabones del PHANToM 1.0 [23]

$c_{32} = m_3 L_2 L_3 s_3 \dot{\theta}_2$

$$G(\theta) = \begin{bmatrix} 0 \\ g_{21} \\ g_{31} \end{bmatrix} \tag{4.79}$$

Donde:
$g_{21} = -m_2 L_{CG1} c_2 - m_3 \{L_2 c_2 + L_{CG3} c_{23}\}$
$g_{31} = -m_3 L_{CG3} c_{23}$
Los términos de la matriz de fricción dependen de las características de la articulación, además estos términos tienen un valor de fricción estática cuando no hay movimiento y un valor de fricción dinámica, cuando se produce el movimiento.

En este caso, los parámetros de fricción empleados se presentan en la tabla 4.2

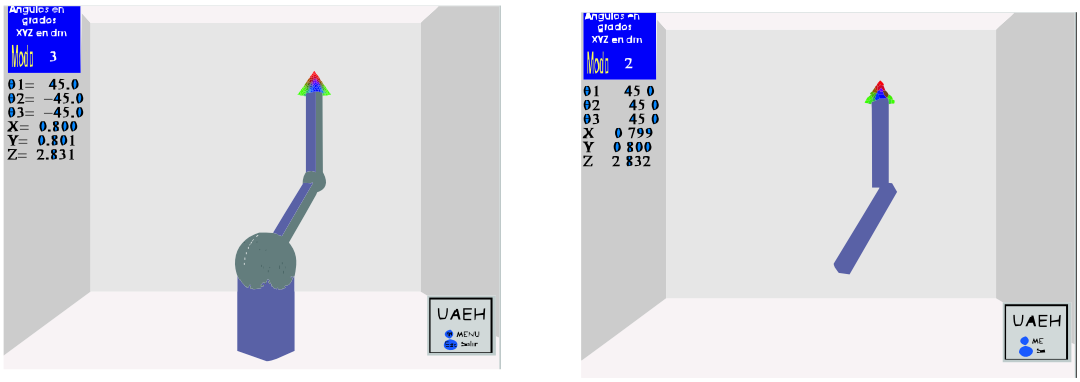
4.5. Simulaciones digitales

A continuación se presentan los resultados de la simulación de libre oscilación y con baja fricción; las longitudes y las masas de los eslabones son las que se muestran en la tabla 4.3. La simulación se inicia con las articulaciones configuradas en:

θ_1= 45 grados (0.7854 rad)
θ_2= -45 grados (-0.7854 rad)
θ_3= -45 grados (-0.7854 rad)
En la figura 4.6 *a)* se muestra la posición que adopta el robot virtual y en la 4.6 *b)* la misma posición en el modo dislocado.

Y el comportamiento de los ángulos de las articulaciones se observa en las gráficas de la figura 4.7. Como se observa en las gráficas de la figura 4.7, el ángulo θ_1, de la 1° articulación permanece inmóvil y solamente θ_2 y θ_3 cambian.

Para observar mejor la oscilación libre es conveniente retirar el 1°, presionando la tecla **F7**; también se observa que θ_3 llega a valores mayores de 360^0 y esto se visualiza como una auto



a) modo PHANToM b) modo dislocado

Figura 4.6: Configuración incio de la simulación de oscilación libre de baja fricción

penetración del 3° eslabón al 2° eslabón.

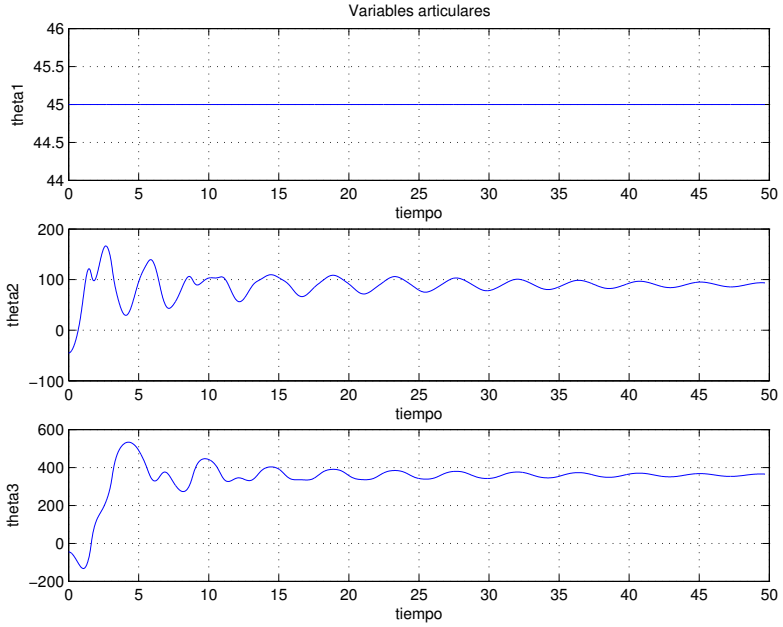


Figura 4.7: Gráficas de comportamiento de articulaciones para la oscilación libre con baja fricción

4.6. Control PID para robots manipuladores

Uno de los propósitos de esta interfaz virtual, es evualuar y verificar las estrategias de control moderno para el desempeño del robot manipulador en su espacio de trabajo.

Para el control PID para seguimiento de trayectorias con libre movimiento se tiene:
Se considera al robot con dinámica en lazo cerrado con un controlador dado por la ecuación (4.80)

$$\tau = -K_d \left[J_{-1}(q)\, \dot{X} - \hat{J}^{-1}(q)\, \dot{X}_r \right] \tag{4.80}$$

Donde K_d es una matriz de ganancia simétrica diagonal.
El seguimiento exponencial está delimitado por la ecuación (4.81)

$$Ki \geq \left\| \frac{d}{dt} \hat{J}(q)\, \hat{S}_q + \hat{J}(q)\, \frac{d}{dt} \hat{S}_q + \frac{d}{dt} \hat{J}(q)\, J(q)\, \dot{X}_r + \hat{J}(q)\, J(q)\, \dot{X}_r \right\| \tag{4.81}$$

Donde:
$\hat{J}^{-1}(q)$ se puede considerar como $J^{-1}(q)$ con un rango $(J^{-1}(q)) = n \forall q \in \Omega$
$\Omega = \left\{ q /_{rango\,(J(q))} = n \right\}$
Y Xr se define en las ecuaciones (4.82) y (4.83)

$$\dot{X}_r = \dot{X}_d - \alpha \Delta X + S(t_0) \exp^{-k(t-t_0)} - K_i \sigma \tag{4.82}$$

$$\dot{\sigma} = \operatorname{sgn}\left(\Delta \dot{X} + \alpha \Delta X - S(t_0) \exp^{-k(t-t_0)} \right) \tag{4.83}$$

Donde:
$\Delta X = X - X_d$

$K_i = K_i^T \in \Re_+^{3X3}$
sgn(*) es una entrada discontinua de la función signo, si k>0 la ganciacia α es una diagonal positiva de una matriz de 3X3

4.7. Simulación digital con un esquema de control

La primera simulación de seguimiento de trayectoria es de una círcunferencia con su centro ubicado en las coordenadas $x = 2dm, y = 2dmyz = 0dm$ y con un radio de $r = 1{,}5dm$

Las longitudes de los eslabones y masas son las mismas que se emplearon en la simulación de libre oscilación y de baja fricción, asi también la simulación se inicia con la misma configuración.

La trayectoria en 3D y los valores de los ángulos para la circunferencia se presentan en las figuras 4.8 y 4.9

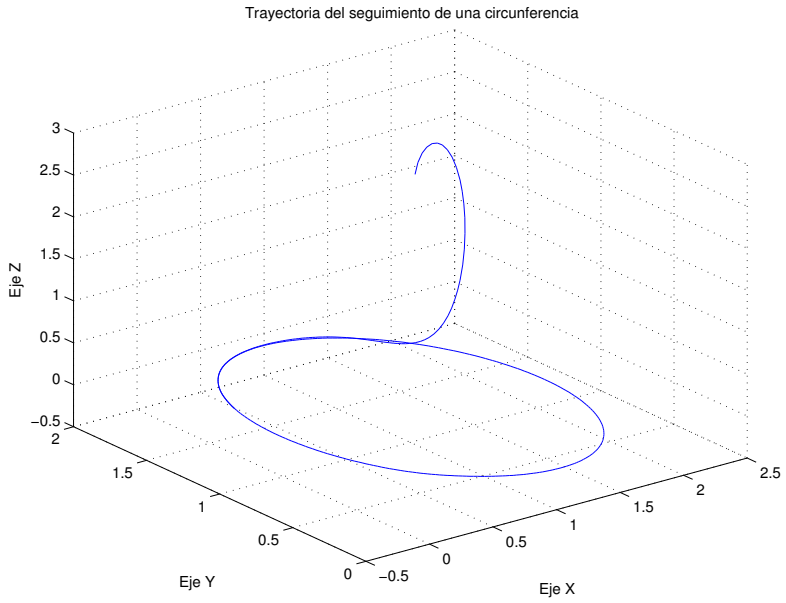


Figura 4.8: Gráfica en Matlab del seguimiento de trayectoria de una circunferencia

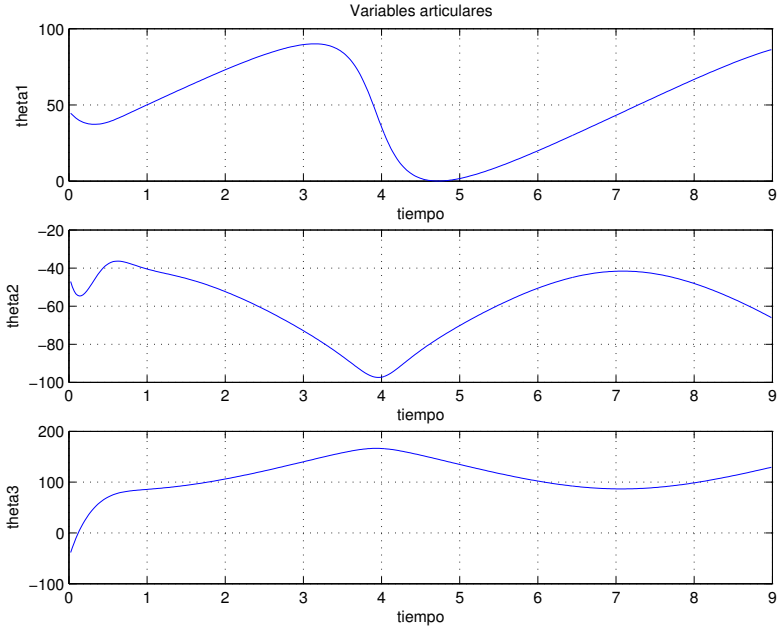


Figura 4.9: Comportamiento de los ángulos de las articulaciones para una circunferencia

Las siguientes gráficas de simulación corresponden al seguimiento de una trayectoria con forma de rosa de 3 pétalos, iniciando en la misma configuración $\theta_1 = 45$ grados, $\theta_2 = -45$ grados y $\theta_3 = -45$ grados.

El comportamiento en 3D se presenta en la gráfica de la figura 4.10 y los valores de los ángulos de las articulaciones se presentan en las gráficas de la figura 4.11

4.8. Conclusiones

En este capítulo se presenta el algoritmo Denavit-Hartenberg para sintetizar los modelos cinemáticos de posición y diferencial y la formulación Euler-Lagrange que permitió obtener las ecuaciones de movimiento del dispositivo háptico. Se establece el criterio para incorporar dichos modelos en el ambiente virtual desarrollado.

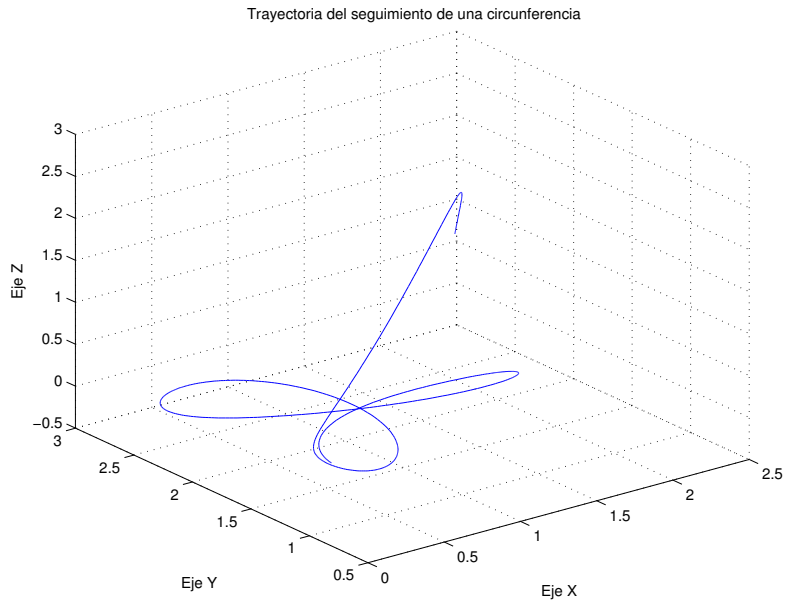


Figura 4.10: Gráfica en Matlab del seguimiento de trayectoria para una rosa de 3 pétalos

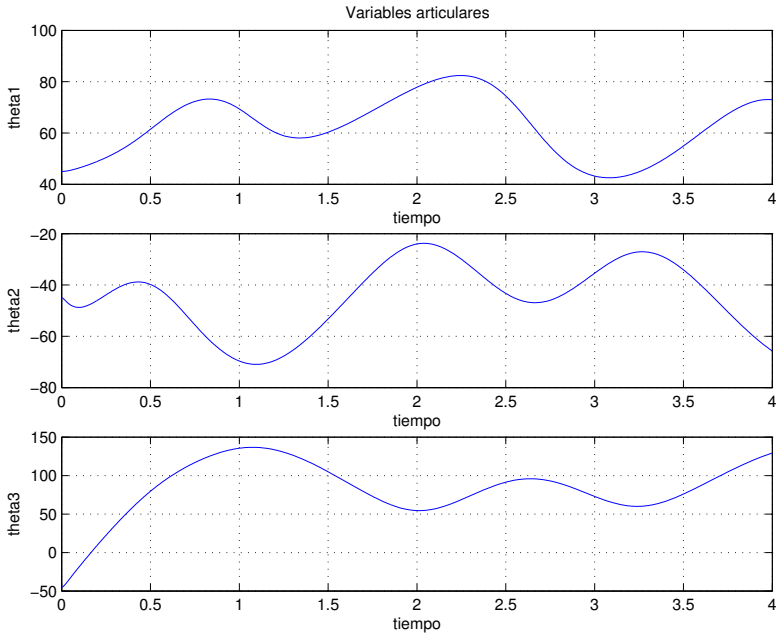


Figura 4.11: Comportamiento de los ángulos de las articulaciones para una rosa de 3 pétalos

Capítulo 5

Ambiente virtual: cinemática del PHANToM 1.0

5.1. Introducción

En este capítulo se explica porque se decidió emplear OpenGL para el desarrollo del ambiente virtual, se mencionan las caracteristicas de OpenGL y Visual C++, se describe como se construye un escenario con objetos virtuales, como se da animación a los objetos virtuales, como construye el robot virtual y como darle animación, así también como se integró la cinemática del dispositivo háptico PHANToM 1.0.

5.2. Ambientes virtuales con OpenGL y Visual C++

5.2.1. OpenGL

¿Por qué se determinó emplear OpenGL?

Después de trabajar en VRML, se llegó a la conclusión de que no cubría las necesidades para el ambiente virtual que se estaba proyectando. También se trabajo con Java 3D, pero éste lenguaje presentaba una desventaja, se complicaba demasiado el código para realizar la animación de objetos en un espacio virtual.

Al mismo tiempo que se trabajaba con Java 3D, también se hacia con OpenGL. Se localizó un sitio en internet donde se proporciona un tutorial de OpenGL y códigos [36]. En particular se revisó el tutorial donde se construye un cubo y se le da animación desde el teclado, entonces se propuso construir al robot a partir de objetos semejantes, tales como paralepípedos, y tener el control de la animación desde el teclado.

Además OpenGL es ampliamente utilizado en la creación de ambientes virtuales. Por lo tanto, se decidió emplear OpenGL.

OpenGL es una interfaz software para gráficos por hardware es decir, es un medio de enlace entre un ambiente gráfico y el equipo periférico de una computadora; también es definido como una librería de gráficos 3D, de modelación portátil y rápida.

Estas librerías proporcionan una determinada calidad para construir ambientes gráficos en 3D, debido a las características que OpenGL ofrece.

OpenGL se diseñó para ser utilizado en los sistemas operativos: Mac OS, Linux, Unix, Windows y Solaris [32]. Inicialmente se concibió para programar en máquinas nativas Silicon Graphics bajo el nombre de GL (Graphics Library), posteriormente se consideró extenderse a cualquier tipo de plataforma y asegurar así su portabilidad y extensibilidad de uso con lo que se llegó al término Open Graphics Library, es decir, OpenGL.[27]
La API OpenGL está dividida en tres librerías distintas que se mencionan a continuación:

1. OpenGL.DLL (Librería de Gráficos Abierta): podemos encontrarla también con el nombre de OpenGL32.DLL (para el caso de los sistemas operativos de 32 bits) o bien simplemente como GL.DLL, ésta es la librería principal y contiene la mayoría de las funciones que se utilizan en la aplicación; las funciones contenidas en esta librería inician con las letras gl.

2. GLU.DLL (Graphics Utility Library, Librería de Utilerías de Gráficos): también la podemos encontrar como GLU32.DLL, contiene funciones para objetos comunes a dibujar como esferas, donas, cilindros, cubos; estas figuras ya predefinidas pueden llegar a ser útiles en ciertos casos, así como funciones para el manejo de la cámara entre muchas otras; Estas funciones se pueden identificar porque llevan antepuestas en su nombre las siglas glu.

3. GLUT.DLL (GL Utility Toolkit, Equipo de Herramientas de Utilería para el desarrollo de Gráficos).- Esta también permite crear objetos complejos como GLU, aunque la principal función de ésta librería es permitir que los programas se vuelvan interactivos, o sea que posibilita la libre creación de ventanas, así como el acceso al ratón y al teclado.[27]

Existe una librería más llamada GLAUX que no es tan conocida , esta librería contiene algunos procedimientos para crear figuras prediseñadas tales como esferas, cubos, etc. como en las anteriores, la diferencia está en que cuenta con un método que permite cargar bitmaps directamente desde archivos BMP; GLAUX proporciona métodos para crear objetos GLBitmap directamente desde archivos, para ser utilizados en texturas.[27]

Todas las funciones de las librerías OpenGL32.dll y glu32.dll están disponibles cuando se usa la librería AUX para la estructura de trabajo del programa. [28]

A continuación mencionamos algunas características de OpenGL [29]:

- Primitivas geométricas que permiten construir descripciones matemáticas de objetos. Las actuales primitivas son: puntos, líneas, polígonos, imágenes y mapas de bits.

- Codificación del Color en modos RGBA (Rojo-Verde-Azul-Alfa) o de color indexado.

- Visualización y modelado que permite disponer objetos en una escena tridimensional, es posible mover la cámara por el espacio y seleccionar un posición deseada.

- Mapeado de texturas, esta característica da mayor realismo a los modelos por medio del dibujo de superficies en las caras de los objetos.

- Dadas las propiedades del material y las fuentes de luz en la habitación, OpenGL provee de comandos para calcular el color de cualquier punto de acuerdo a la iluminación del objeto.

- Doble buffering que ayuda a eliminar el defecto intermitente de las animaciones. Cada fotograma consecutivo en una animación se construye en un buffer separado de memoria y es mostrado solo cuando está completo.
- El Anti-alizado reduce los bordes escalonados en las líneas dibujadas sobre una pantalla de ordenador. Los bordes escalonados aparecen a menudo cuando las líneas se dibujan con baja resolución. El anti-alizado es una técnica común en gráficos de ordenador que modifica el color y la intensidad de los píxeles cercanos a la línea para reducir el zig-zag artificial.
- El sombreado Gouraud es una técnica usada para aplicar sombreados suaves a un objeto 3D y producir una sutil diferencia de color por sus superficies.
- El Z-buffering mantiene registros de la coordenada Z de un objeto 3D. El Z-buffer se usa para registrar la proximidad de un objeto al observador, y es también crucial para el eliminado de superficies ocultas.
- Efectos atmosféricos como la niebla, el humo y las neblinas hacen que las imágenes producidas por ordenador sean más realistas.
- El Alpha blending usa el valor Alfa (valor de material difuso) del código RGBA, y permite combinar el color del fragmento que se procesa con el del píxel que ya está en el buffer. El Alpha blending permite simular la transparencia de objetos, de tal manera que objetos colocados detrás del objeto transparente, aparezcan con un tono magenta.
- Los planos de plantilla permiten restringir el trazado a ciertas regiones de la pantalla.
- Las listas de Display permiten almacenar comandos de dibujo en una lista para un trazado posterior.
- Los Evaluadores Polinómicos sirven para soportar B-splines racionales no uniformes, esto es para ayudar a dibujar curvas de contorno suave a través de unos cuantos puntos de referencia, evitando la necesidad de acumular grandes cantidades de puntos intermedios.
- Características de Feedback, selección y elección que ayudan a crear aplicaciones que permiten al usuario seleccionar una región de la pantalla o elegir un objeto dibujado en la misma. El modo de feedback permite al desarrollador obtener los resultados de los cálculos de trazado.
- Primitivas de Raster (bitmaps y rectángulos de pixels); El formato raster (GIF, JPEG, TIFF...) es muy útil para composiciones que no precisen grandes calidades de impresión y son ideales para acompañar estudios y documentos [30].
- Operaciones con Pixels, una vez que las imágenes han sido digitalizadas y guardadas en forma de matriz en el correspondiente sector de memoria (buffer) de la computadora, resulta inmediata la posibilidad de realizar operaciones aritméticas y lógicas entre ellas [31]
- Transformaciones: rotación, escalado, perspectivas en 3D.

- Además es perceptiva a la red, de manera que es posible separar la aplicación OpenGL en un servidor y un cliente que verdaderamente produzca los gráficos. Existe un protocolo para enviar por la red los comandos OpenGL entre el servidor y el cliente. Gracias a su independencia del sistema operativo, el servidor y el cliente no tienen porque ejecutarse en el mismo tipo de plataforma, muy a menudo el servidor será una supercomputadora ejecutando una compleja simulación y el cliente una simple estación de trabajo mayormente dedicada a la visualización gráfica.

5.2.2. Visual C++

Para iniciar la descripción sobre Visual C++ y la programación orientada a objetos, se mencionarán algunos fundamentos de Windows, ya que el entorno virtual desarrollado, se encuentra bajo esta plataforma, además Windows ha sido bastante aceptado por los beneficios de su interfaz gráfica (GUI).
Windows es un entorno multitarea basado en ventanas, que representan programas, y que permite ejecución concurrente.

El modelo de programación de Windows es diferente de la antigua programación orientada a lotes de órdenes o transacciones; así que tomando como referencia la programación basada en MS-DOS describiremos algunas de sus características; el modelo de programación propuesto por Windows es totalmente diferente al modelo de ejecución secuencial de DOS. Al ser Windows un entorno multitarea los programas tienen que estar preparados para compartir los recursos de la máquina (procesador, memoria, teclado, ratón ...). Esto supone que Windows ha de disponer de métodos que permitan suspender tareas para activar otras en función de las circunstancias del momento.
Procesamiento de mensajes
Cuando se escribe un programa en C basada en MS-DOS, el único requisito es una función llamada ***main***.
El sistema operativo llama a main cuando el usuario ejecuta un programa. Si el programa necesita obtener una entrada del usuario o utilizar los servicios del sistema operativo, se llama a la función adecuada o se utiliza una biblioteca de ventanas basadas en caracteres.
Cuando el sistema operativo Windows lanza un programa, llama a la función ***WinMain*** del programa; En algún lugar, la aplicación debe tener WinMain, que realiza algunas tareas específicas, la mas importante es crear la ventana principal de la aplicación, que debe tener su propio código para procesar el mensaje que Windows le envía.
La mayoría de los mensajes de Windows están definidos de manera estricta y se pueden aplicar en todos los programas. Por ejemplo, se envía un mensaje ***WM_CREATE*** cuando se crea una ventana.
Todos los mensajes tienen parámetros de 32 bits que transportan información como la coordenada del cursor, el código de la tecla pulsada, etc., Windows envía mensajes WM_COMMAND a la ventana adecuada en respuesta a las selecciones del menú de usuario, las pulsaciones de los botones de diálogo, etc., los parámetros de los mensajes de órdenes varían en función de la disposición del menú de la ventana.
Interfaz de dispositivo gráfico de Windows

Muchos programas de MS-DOS escribían directamente en la memoria de video y en el puerto de video, esto presentaba la desventaja de suministrar software de controlador para cada modelo de tarjeta de video y de impresora; Windows introdujo un nivel de abstracción llamado la interfaz de dispositivo gráfico (GDI) que proporcionan controladores de video e impresora conectadas al sistema; En vez de dirigirse al hardware, el programa llama a las funciones de GDI que hacen referencia a una estructura de datos llamada **contexto de dispositivo**, Windows relaciona el contexto de dispositivo con el dispositivo físico y emite las instrucciones de entrada/salida adecuadas.

Programación basada en recursos

Para realizar una programación dirigida por datos en MS-DOS, es necesario codificar los datos como constantes de inicialización o proporcionar archivos de datos aparte para que los lea el programa. Cuando se programa en Windows, se almacenan los datos en un archivo de recursos utilizando varios formatos establecidos.

Los archivos pueden incluir mapas de bits, íconos, definiciones de menú, composiciones de los cuadros de dialogo y cadenas.

Gestión de memoria

Con cada versión de Windows, la gestión de memoria se simplifica, se asigna la memoria que se necesita y Windows se encarga de los detalles; algunas de las funciones de gestión de memoria de Win16, como el GlobalAlloc, se trasladaron a Win32, pero se hizo para permitir a los desarrolladores modificara el código fuente con rapidez al pasar de una versión a otra.

Bibliotecas de enlace dinámico

En el entorno de MS-DOS, todos los módulos de objeto de un programa se enlazan de manera estática durante el proceso de construcción. Windows permite el enlazado dinámico, lo que significa que pueden cargar y enlazar en tiempo de ejecución bibliotecas construidas de una forma especial. Múltiples aplicaciones pueden compartir bibliotecas de enlace dinámico (DLL), lo que ahorra memoria y espacio en disco.

Los componentes de visual C++

Está formado por dos sistemas completos de desarrollo de aplicación para Windows en un producto, se pueden desarrollar programas empleando el API Win32. Se pueden utilizar muchas herramientas de Visual C++, incluyendo los editores de recursos, para simplificar la programación de bajo nivel.

También incluye la biblioteca de plantillas de ActiveX (ATL) que se puede utilizar para desarrollar controles ActiveX para internet.

El AppWizard

Es un generador de código que crea un esqueleto de trabajo de una aplicación de Windows con características, nombres de clases y nombres de archivos código que se especifican a través de cuadros de diálogo.

El código del AppWizard es mínimo; la funcionalidad se encuentra en las clases de base del armazón de la aplicación.

El classWizard

Es un programa implementado como DLL, simplifica el mantenimiento del código de clase de Visual C++; en caso de requerir una función virtual nueva o una función de gestión de mensajes, el ClassWizard escribe los prototipos, cuerpos de de la función y si es necesario , el código para

enlazar el mensaje de Windows a la función. [33]

En Visual C++ la construcción de cualquier tipo de programa se inscribe dentro del concepto de proyecto (workspace). Un proyecto define los pasos a seguir para alcanzar la construcción de un objetivo (un programa, una DLL, etc.), en realidad es un concepto análogo a lo que se conoce como **makefile** en otros entornos típicos de desarrollo en C. En realidad Visual C++ genera para cada proyecto dos ficheros que lo definen, el fichero de workspace (con extensión wsp) y un makefile (con extensión mak) estándar que permite la utilización del mismo proyecto en otro entorno distinto.
El proyecto contiene referencias a cada uno de los ficheros fuentes (C/C++, con extensiones c y cpp respectivamente), objetos, librerías o ficheros de recursos (extensión rc) que se deben utilizar para construir el objetivo final del proyecto.
Para crear cualquier programa con Visual C++ se debe comenzar creando un proyecto para él, codificando y añadiendo los módulos necesarios a dicho proyecto, y definiendo los recursos asociados. [34]

5.3. Desarrollo de un ambiente virtual empleando OpenGL

Para construir un ambiente virtual con OpenGL es necesario tener un espacio de trabajo; una ventana donde se desarrollará la animación del ambiente virtual en este espacio de trabajo se definen algunas características tales como tamaño de la ventana, posición de la ventana, color del fondo (background), encabezado de la ventana etc. Después se proporcionan las funciones correspondientes que integran al ambiente virtual.

Por ejemplo, para dibujar una esfera se emplea la función:

```
gluSphere(quadratic, radio, param1, param2);
```

A este objeto se le proporcionan dos características principalmente, color y/o textura, para definir el color se emplea la función:

```
glColor3f (red,green,blue);
```

donde los valores para las variables red green blue van de $0{,}0f$ a $1{,}0f$.

Para asignar una textura es necesario primero haberla adquirido desde un archivo bmp; y solamente se indica cual textura se ha de colocar en el objeto a través de la función:

```
glBindTexture (GL_TEXTURE_2D textura [ Textura deseada]);
```

Si se desea conservar la textura original es necesario definir un color blanco:

```
glColor3f (1.0f,1.0f,1.0f);
```

Para dar movimiento a los objetos del ambiente virtual se hace a través de las funciones:

1. glLoadIdentity();
2. glTranslatef(Valorx, valory, valorz); glRotatef(ángulo, opcionx, opciony, opcionz);

Estas funciones proporcionan traslación y rotación a los objetos virtuales.

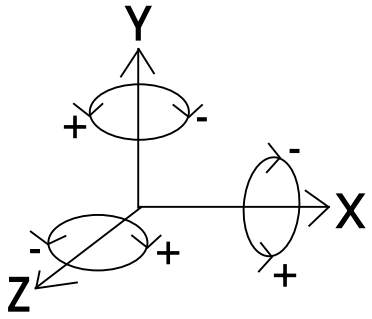


Figura 5.1: Disposición de ejes en el espacio virtual con OpenGL

5.4. Animación al ambiente virtual

Como se menciono anteriormente, la animación de los objetos en el ambiente virtual se hace con funciones que proporcionan traslación y rotación.

La función *glLoadIdentity()*, reestablece los valores de las coordenadas para el origen y también restablece los ángulos de giro de los ejes de referencia del píxel de trazo, entendiendo como reestablecer coordenadas del origen llevar al origen al centro del espacio virtual (centro de la pantalla para las coordenadas x , y y justo donde inicia el monitor para la coordenada z) y reestablecer ángulos de giro de los ejes de referencia, es llevarlos a cero (haciendo coincidir el plano x-y con la pantalla del monitor y el plano xz con la horizontal del monitor).

Esta función es muy importante para dibujar los objetos en el espacio virtual cuando se emplea la función *glTranslatef(...)* y *glRotatef (...)*, como se explica a continuación:

La función *glTranslatef (...)* modifica las coordenadas (XYZ) origen del píxel de trazo.

Por ejemplo:

```
glLoadIdentity (); //reestablece la coordenada del píxel de trazo
glTranslatef(3.0f, -2.0f, -5.0f);//traslación del origen del píxel
de trazo
```

El origen del píxel de trazo se desplazó 3 unidades a la derecha, 2 unidades hacia abajo, y 5 unidades hacia el fondo.

Es importante explicar que los ejes están definidos en el espacio de trabajo virtual de la siguiente forma:

- El movimiento sobre la horizontal se hace sobre el eje X.
- El movimiento vertical sobre el eje Y.
- hacia el fondo o acercar sobre el eje Z.

Como se muestra en la figura 5.1

Esta definición de eje se manejará de aquí en adelante para el espacio virtual construido con OpenGL, y que es diferente a la definición de ejes establecida por el modelo cinemático.

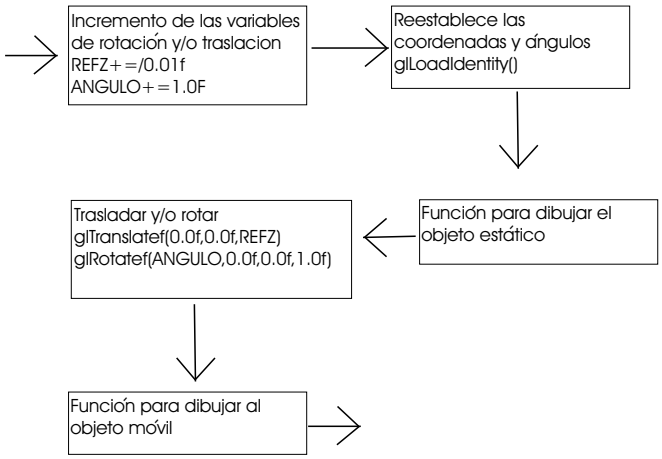


Figura 5.2: Diagrama para dibujar un objeto fijo y otro móvil

La función *glRotate3f (ANGULO, OPCIONX, OPCIONY, OPCIONZ);* proporciona un giro en grados sexagesimales para los ejes de referencia del píxel de trazo.

El parámetro *ANGULO* puede ser positivo o negativo o mayor a 360, los parámetros *OPCION X, OPCIONY y OPCIONZ* se les asignara el valor de *1.0f* si se les desea aplicar el ángulo de rotación al eje correspondiente. Y se asignara el valor de *0.0f* sino se aplica.

Ejemplo:

```
glRotatef (30.0f, 0.0f,0.0f, 1.0f); // se gira 30 grados sobre z;
```

Ahora empleando estas tres funciones se puede producir un efecto de movimiento de la siguiente forma:

Por ejemplo, si se requiere que un objeto se aleje, se puede hacer con las siguientes sentencias:

```
REFZ+=-0.1f;
glTranslatef(0.0f, 0.0f, 0.0f, REFZ);
```

El objeto se alejara en incrementos de *-0.1f* unidades. Si se requiere una apariencia de mayor velocidad, solo se aumenta el incremento , Ahora que si lo que se requiere es que un objeto gire en sentido de las manecillas del reloj, esto se hace con las siguientes sentencias:

```
ANGULO+=1.0f;
glRotatef(ANGULO, 0.0f, 0.0f, 1.0f);
```

Finalmente, si se requiere que un objeto se mueva respecto a otro, se empleará la función *glLoadIdentity();* aplicada como se muestra en la figura 5.2

El incremento de las variables de translación y de rotación puede realizarse en alguna otra parte del código, siempre y cuando se escriban antes de la función que realiza la rotación y translación.

Las funciones de OpenGL dibujan y borran los objetos para producir una animación. OpenGL emplea un doble buffer para presentar los objetos virtuales, esto quiere decir que un buffer

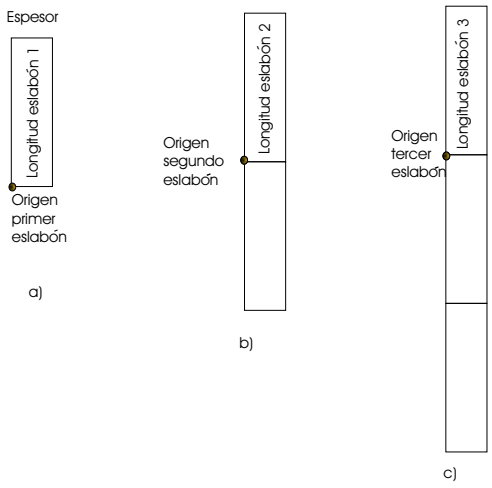


Figura 5.3: Aproximación con rectángulos

contiene el escenario virtual que se visualiza, mientras que en el segundo buffer se esta dibujando la siguiente escena, y en su momento se intercambian los buffers [35].

Este doble buffer da la apariencia de movimientos suaves y se elimina el defecto de intermitencia. Cabe aclarar que si los incrementos son muy grandes el movimiento de los objetos no es suave.

5.5. Contrucción del robot virtual

El robot PHANToM esta constituido por tres eslabones y tres articulaciones (sin tomar a la base como el eslabón cero).

Primera aproximación (con rectángulos)

Se construyó un rectángulo para representar al primer eslabón, como se muestra en la figura 5.3 a),para dibujar el segundo eslabón, primero se traslado el origen del píxel de trazo: *glTranslatef(0.0f, longitud eslabon2, 0.0f);* y se dibujo otro rectángulo para representar al segundo eslabón, lo mismo se hizo para el tercer eslabón, como se muestra figura 5.3 b) y c) respectivamente. Las funciones para dibujar un eslabón con un rectángulo son:

```
Begin (GL_QUADS);
   glVertex3f(0.0f, 0.0f, 0.0f);
   glVertex3f(ESP,0.0f, 0.0f);
   glVertex3f(ESP, LongEsl1, 0.0f);
   glVertex3f(0.0f,LongEsl1, 0.0f);
glEnd();
```

Como se puede observar, se definen las 4 esquinas del rectángulo a partir del origen del píxel de trazo.

Para dar movimiento a cada uno de los eslabones, se incorpora la función *glRotatef(...);* Antes de dibujar cada uno de los eslabones como se ilustra en la figura 5.4

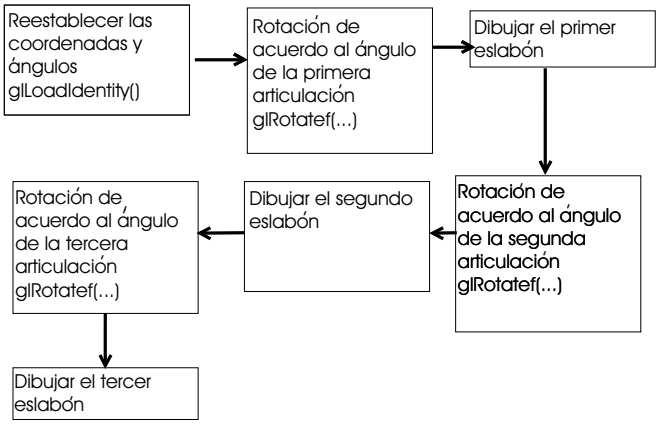


Figura 5.4: Descripción para dibujar 3 eslabones y proporcionarles movimiento

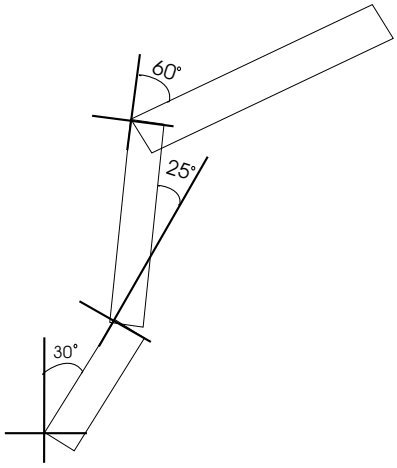


Figura 5.5: Configuración de eslabones con rectángulos

Por ejemplo, si los ejes de las articulaciones están sobre el eje Z del espacio virtual, y las articulaciones tienen la siguiente configuración: $\theta_1 = 30$, $\theta_2 = -25$ y $\theta_3 = 60$, la visualización que se obtiene es la que se muestra en la figura 5.5.

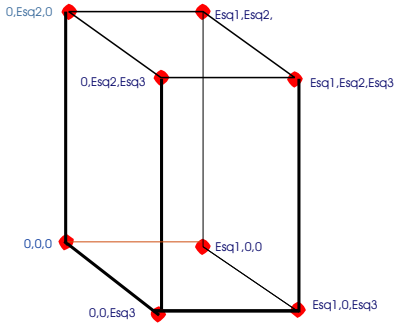


Figura 5.6: Paralepipedo a partir de 6 rectángulos

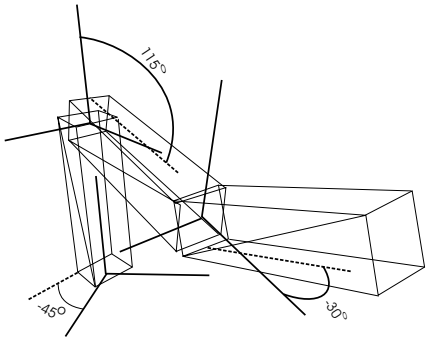


Figura 5.7: Configuración de eslabones con paralepipedos

Segunda aproximación (con paralepípedos)

Para construir un paralepípedo se necesitan 6 rectángulos para cada una de las caras como se muestra en la figura 5.6

Para dibujar un paralepípedo se implementó una función que dibuje los 6 rectángulos cuando se le pasan 3 parámetros que son: *esq1 (ancho), esq2 (alto), esq3 (largo)* como se puede mostrar en la figura 5.6

Siguiendo el mismo criterio en la primera aproximación con restángulos, los eslabones quedan alineados como se mostrará mas adelante.

Para dar movimiento a los eslabones se emplea también el criterio de la figura 5.4

Por ejemplo, si el eje de la primera articulación se mueve sobre el eje Y, el eje de las otras articulaciones se mueve sobre el eje Z y la configuración de las articulaciones son: $\theta_1 = -45$, $\theta_2 = 115$ y $\theta_3 = -30$, la visualización que se obtiene se muestra en la figura 5.7

Como se puede observar la rotación de los eslabones se hace sobre uno de los extremos de la cara superior y exterior de cada eslabón respectivamente; para lograr que la rotación se haga a la mitad de las caras de los eslabones respectivos se hace lo siguiente: antes de dibujar el paralepípedo se hace una traslación para mover el origen del píxel de trazo. Después se dibuja el paralepípedo. Nuevamente se hace una traslación para mover el origen del píxel de trazo donde inicialmente se encontraba.

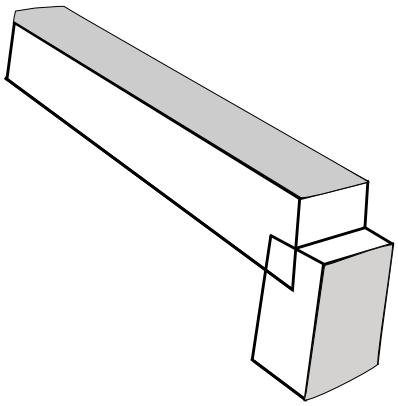


Figura 5.8: Efecto de movimiento entre 2 eslabones

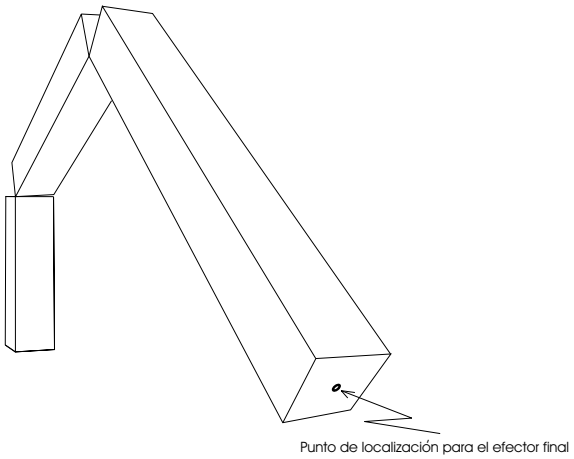


Figura 5.9: Localización del punto del efector final

Esta forma de colocar y construir los paralepípedos produce un movimiento entre los eslabones como se muestra en la figura 5.8 Implementando lo anterior se logra que el punto donde se localiza el efector final sea en el centro de la cara del extremo del paralepípedo como se muestra en la figura 5.9 La traslación se hace de la siguiente forma:

- glTranslatef(-esq1/2,0.0f,-esq3/2);
- Se dibuja el paralepípedo
- glTranslatef(esq1/2, 0.0f,esq3/2);

Construcción especial para el primer eslabón e incorporación de esferas en las articulaciones

El primer eslabón se construye mas grande colocándole una esfera en la parte superior, estas dos características proporcionan una mejor apariencia al robot. El primer eslabón se construye a partir de un cubo, es decir, que el valor de altura (longitud del eslabón) es el mismo valor para el largo y ancho.

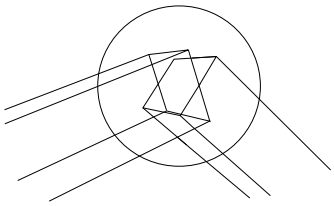


Figura 5.10: Esferas en articulaciones

La esfera se construye con un radio con valor en porcentaje a la longitud del primer eslabón. Un porcentaje que proporciona una buena apariencia es de 75 %, es decir, que el radio de la circunferencia es el 75 % de la longitud del eslabón.

Para mejorar la apariencia de la segunda y tercera articulación, se dibuja una esfera en ellas.

Debido a que el origen del píxel de trazo queda ubicado en el centro de la cara de los paralepípedos, la esfera queda bien colocada en la articulación, como se observa en la figura 5.10 Para dar una apariencia proporcional de las esferas en las articulaciones, el radio esta definido en porcentaje al espesor de cada eslabón. Un porcentaje apropiado es de 150 %.

5.6. Animación al robot virtual

Como se ha descrito anteriormente, la animación se hace a través de funciones que proporcionan traslación y rotación, en estas funciones se colocan variables que controlan la función y orientación de cada objeto en el ambiente virtual.

Para manipular estas variables, se hace de dos formas:

- Desde el teclado
- Desde un archivo

Para hacer una interacción desde el teclado, se hace lo siguiente:

- Se declara un arreglo bool:

  ```
  bool Keys[ 256];
  ```

  Para las 256 teclas y posibles combinaciones de teclas

- El método *WndProc(...)* Se verifica si una tecla fue presionada con la instrucción *WM_KEYDOWN*

  Y para conocer que tecla fue presionada hay que leer el contenido de WParam.

  Ésto se consigue asignando el valor true al elemento correspondiente del arreglo *Keys[ ]* , es decir:

  ```
  Key[WParam]=TRUE;
  ```

- Ahora, cuando la tecla es soltada, es necesario saber cual fué, ésto se consigue de forma similar cuando fué presionada, es decir:

```
WM_KEYUP { Key [WParam]=FALSE;}
```

- En el método principal de windows *WinMain (...)* se ejecuta la acción correspondiente a la tecla presionada, esta forma es conveniente cuando se presentan varias teclas al mismo tiempo. Por ejemplo, para incrementar el ángulo de la primera articulación, se estableció que con la tecla W se incrementa el valor de θ_1. Entonces el elemento correspondiente del elemento *Key[]* que corresponda a la **W** se le asigna el valor true; en el método *WindMain(...)* se tiene la condicional:

```
If (Key[W]) { teta1 +=0.1f };
```

  Este incremento seguirá mientras mantengamos presionada la tecla **W** ahora bien, si presionamos la tecla S que definimos para el incremento del ángulo de la segunda articulación θ_2, entonces lo que se visualizará es el movimiento del robot con los ángulos de la primera y segunda articulación cambiando.

Para manipular las variables que controlan la traslación y orientación desde un archivo, es necesario que se tenga previamente el archivo listo.

El protocolo para leer los datos de un archivo es el siguiente:

1. El nombre del archivo debe ser *Trayectoria.mat*
2. El tipo de archivo es de texto, ASCII.
3. El archivo debe contener 7 datos separados por un espacio en blanco.
4. El orden de las datos dede ser:
   - Tiempo
   - Valor de la variable articular 1 (θ_1)
   - Valor de la variable articular 2 (θ_2)
   - Valor de la variable articular 3 (θ_3)
   - Valor de la coordenada operacional X
   - Valor de la coordenada operacional Y
   - Valor de la coordenada operacional Z

Para tener acceso a los archivos es necesario declarar una variable de tipo file.

Cuando se selecciona la opción de visualización desde archivo primero se deben cargar los datos con la tecla que se estableció previamente para ello (**F5**). Enseguida se leen los datos de 7 en 7 verificando siempre que el primero no sea un dato nulo. La extracción y verificación se realiza de la siguiente forma:

```
FILE *KL; // variable tipo file KL =fopen(Trayectoria.mat,
r);//abriendo para leer el archivo
fscanf(KL, %f,AUXDATO);
if (AUXDATO <=NULL) {
   CONTA++;
   TIEMPO [CONTA]=AUXDATO;
   fscanf(KL, %f,AUXDATO);
   TETA1[CONTA]=AUXDATO;
   fscanf(KL, %f,AUXDATO);
   TETA2[CONTA]=AUXDATO;
   fscanf(KL, %f,AUXDATO);
   TETA3[CONTA]= AUXDATO;
}
```

La lectura de datos se hará hasta que la variable AUXDATO sea nula después de haber leído los datos se pasan a las variables que controlan la traslación y rotación cuando se presiona la tecla designada para ello (**F6**).

Se inicia un ciclo para pasar los valores del archivo a las variables articulares:

```
for (int=1, i< conta+1, i++)
{
teta1=TETA1[i];
teta2=TETA2[i];
teta3=TETA3[i];
...
}
```

El almacenamiento de datos en arreglos presenta dos inconvenientes:

- Se consume recurso de memoria.
- Animaciones muy largas en tiempo no se pueden visualizar debido a la capacidad de memoria.

Se ha destinado que los arreglos sean de 2000 datos, que proporcionan 40 segundos en promedio de visualización.

5.7. Integración y validación del modelo cinemático en el robot virtual

Para integrar el modelo cinemático del robot virtual se tiene que de acuerdo a la cadena cinemática, la posición y orientación de los ejes definida en el **capitulo 4** es la que se muestra en la figura 5.11

Se observa que la primera articulación gira sobre el eje **Y** del espacio virtual la rotación en esta articulación se hará con la función: *glRotatef(teta1,0.0f,1.0f,0.0f);* la rotación para la segunda

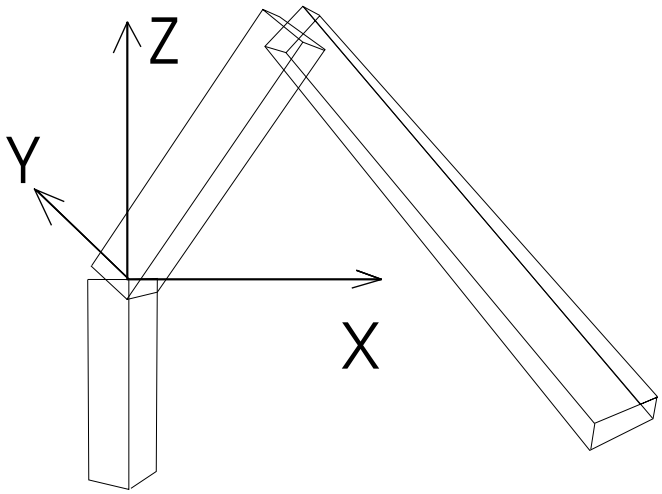


Figura 5.11: Orientación de los ejes de acuerdo al modelo cinemático

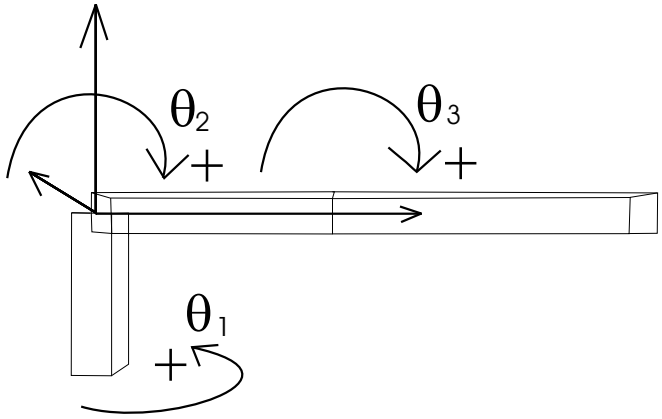


Figura 5.12: Configuración de eslabones al inicio del programa

y tercera articulación se hace sobre el eje **Z**, es decir: *glRotatef (teta2, 0.0f, 0.0f, 1.0f)* para la segunda articulación, *glRotatef (teta3, 0.0f, 0.0f, 1.0f)* para la tercera articulación.

Las ecuaciones del modelo cinemático directo de posición están dados por las ecuaciones (4.7), (4.8), (4.9), (4.13), (4.14) y (4.15).

Sustituyendo para los ángulos $\theta_1 = \theta_2 = \theta_3 = 0^0$, que es la configuración que se presenta al iniciar el programa del visualizador se obtiene:

X=6,27, Y=0 ,Z=0

Esto significa que la visualización debe ser la que se presenta en la figura 5.12.

Para conseguir que el robot se presente en la forma de la figura 5.12, se debe agregar 90^0 grados al valor del ángulo que manipula la rotación en la segunda articulación, es decir: *glRotatef(teta2+90, 0.0f, 0.0f, 1.0f);*

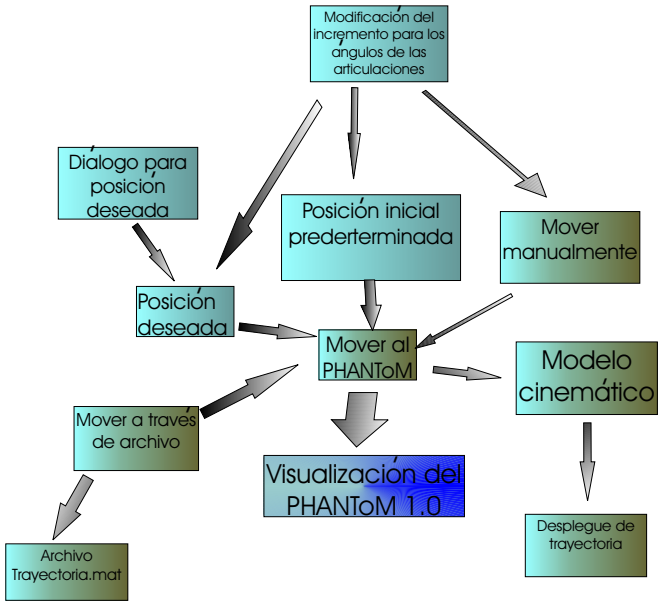


Figura 5.13: Opciones para mover al PHANToM virtual

5.8. Modelo conceptual

La interfaz de visualización se planeó para observar el desempeño cinemático del dispositivo háptico PHANToM. Conforme se fueron desarrollando las diferentes partes de la interfáz, se incorporaron diferentes opciones y funciones para la operación y visualización.

Estas funciones se explican en los diagramas conceptuales de las figuras 5.13, 5.14, 5.15, 5.16 y 5.18

5.8.1. Movimiento del PHANToM

La interfáz de visualización virtual tiene diferentes formas de dar movimiento al PHANToM, como se puede observar en el diagrama de la figura 5.13.

Las diferentes opciones para mover al PHANToM, emplean el modelo cinemático directo de posición para mostrar las coordenadas operacionales.

El MCDP es empleado para desplegar la trayectoria por donde pasó el efector final.

Mover manualmente: el usuario a través del teclado puede cambiar el valor de las articulaciones y observar directamente el movimiento resultante.

Posición inicial predeterminada: en esta opción, el PHANToM regresa a la configuración inicial de sus articulaciones, es decir, que el ángulo θ_1 regrasa a 0^o, θ_2 a -45^o y θ_3 a 90^o.

Esta es la configuración inicial que presentan las articulaciones.

Posición deseada: esta opción coloca al efector final del PHANToM sobre un punto deseado en su espacio de trabajo.

El usuario llama al diálogo *MCDP.EXE* presionando la tecla **F8**, en este diálogo se calculan las coordenadas operacionales a partir de los ángulos que proporcione el usuario, se graban los ángulos y se cierra el diálogo.

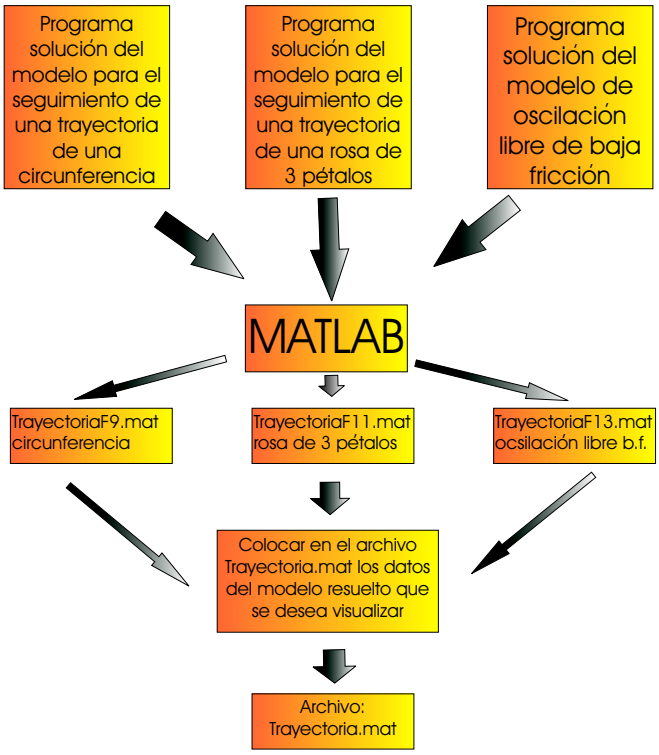


Figura 5.14: Enlace de Matlab con la interfáz de visualización

Presionando **F4**, se mueve el PHANToM a la coordenada deseada.

Mover a través de archivo: esta opción fue desarrollada para observar el comportamiento dínámico, es decir tomando en cuenta los efectos gravitatorios, inerciales y de fricción.

Para utilizar esta opción, debe existir un archivo de datos llamado *Trayectoria.mat* en la carpeta *Datos*.

En este archivo contiene los valores del tiempo, variables articulares y operacionales.

El usuario primero indica que simulación desea visualizar, para obsevar el seguimiento de una trayectoria de una circunferencia, presiona la tecla **F9**, para observar el seguimiento de una trayectoria de una rosa de 3 pétalos, presiona **F11** y para la simulación de oscilación libre de baja fricción presiona **F12**.

Cuando presiona cualquiera de estas 3 teclas, se tranfieren los datos de la solución para la simulación deseada al archivo *Trayectoria.mat*.

Ahora para visualizar la simulación del comportamiento dinámico, primero carga los datos a la memoria presionando la tecla **F5** y para ejecutar la simulación, presiona la tecla **F6**.

5.8.2. Enlace con Matlab

Como se dijo anteriormente, para realizar el movimiento del PHANToM a través de archivo es necesario que exista un archivo denominado *Trayectoria.mat*; en este archivo se coloca la información que necesita leer la interfáz de visualización para ejecutar la simulación.

Como se puede observar en el diagrama de la figura 5.14, se tienen 3 programas donde se localizan los modelos matemáticos que corresponden a cada simulación; empleando Matlab se resuel-

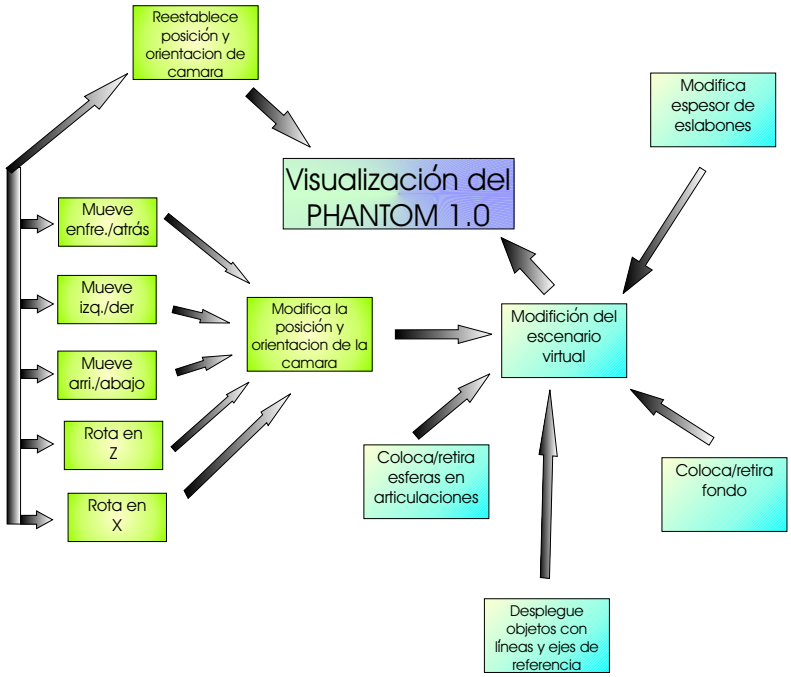


Figura 5.15: Modificación del escenario, posición y orientación de la cámara

ven estos modelos y se producen 3 archivos denominados *TrayectoriaF9.m*, *TrayectoriaF11.m* y *TrayectoriaF12.m* que corresponden a las soluciones para el seguimiento de una tayectoria de una circunferencia, seguimiento de una trayectoria de una rosa de 3 pétalos y la simulación de oscilación libre de baja fricción respectivamente.

Entonces para tranferir los datos al archivo *Trayectoria.mat* se presionan las teclas **F9**, **F11** o **F12** según sea el caso.

Si el usuario desea realizar otra simulación, empleando algún modelo diferente, solo tiene que generar el archivo solución de acuerdo al formato que se explica en la guia de usuario y darle un nombre al archivo para que la interfáz lo reconozca.

5.8.3. Modificación del escenario, posición y orientación de la cámara

Las opciones para modificar el escenario, se pueden obsevar en el diagrama de la figura 5.15.

El escenario se puede modificar de varias formas, una es retirando o colocando:

- Las esferas en las articulaciones
- El escenario de fondo

Presionando las teclas **Y**, **U** respecivamente.

Otra modificación es cambiar el espesor de las articulaciones presionando las teclas **I** , **O**

Y la última modificación al entorno virtual es para dibujar los objetos con líneas.

Estas opciones de modificación al entorno virtual se implementaron para ahorrar recursos y disminuir latencias en el renderizado de los objetos.

Los cambios de la posición y orientación de la cámara que se pueden realizar son:

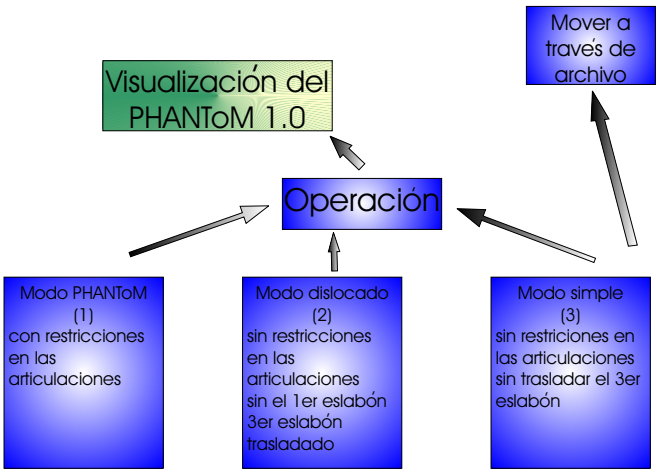


Figura 5.16: Modos de operación de la interfáz de visualización

- Alejar o acercar empleando las teclas **E**, **R**
- Mover hacia arriba, hacia abajo empleando las teclas **D**, **F**
- Hacia la derecha o hacia la izquierda empleando las teclas **C**, **V**
- Rotar sobre el eje **Z** (*eje Y para OpenGL*) y sobre el eje **X** empleando las teclas de las flechas del cursor.

También se integró una función para reestablecer los valores de la posición y orientación de la cámara.

5.8.4. Modo de operación

Como se puede observar en el diagrama de la figura 5.16, se tienen 3 modos de operación para la interfaz.

El modo PHANToM (1), que es el de inicio, contempla restricciones en los ángulos de las articulaciones, para proporcionar una operación conforme al dispositivo háptico PHANToM, como se puede observar en la comparación de la figura 5.17.

La primera articulación no tiene restrición , la segunda articulación, opera en en intervalo de -118^o a $-12{,}5^o$ y la tercera en el intervalo de $28{,}8^o$ a $110{,}3^o$.

El modo dislocado (2) no tiene restricciones para los ángulos de las articualaciones, elimina el primer eslabón y traslada al tercero; todo ello para evitar la invación del espacio de renderizado.

Este modo de operación se implementó para observar la simulación de oscilación libre de baja fricción.

El modo simple (3) no presenta restricciones en los valores de los ángulos de las articulaciones, si presenta el primer eslabón y no traslada el tercer eslabón.

Este es el modo de operación cuando se mueve al PHANToM virtual desde archivo.

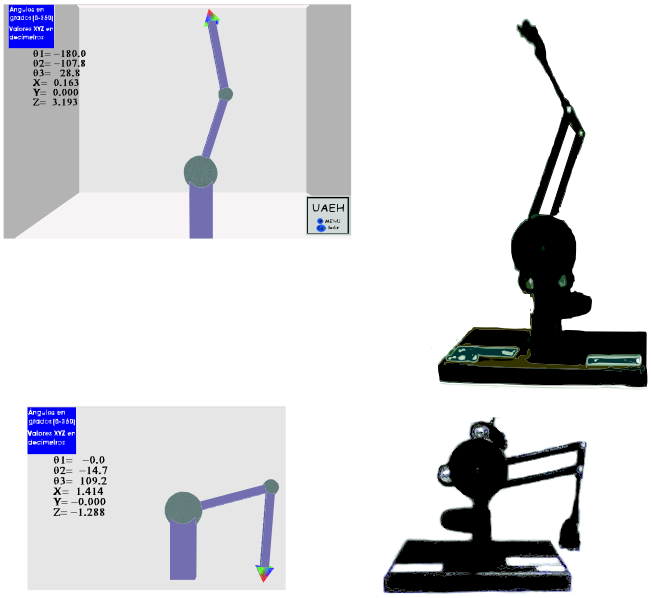


Figura 5.17: Comparación para las restricciones en las articulaciones

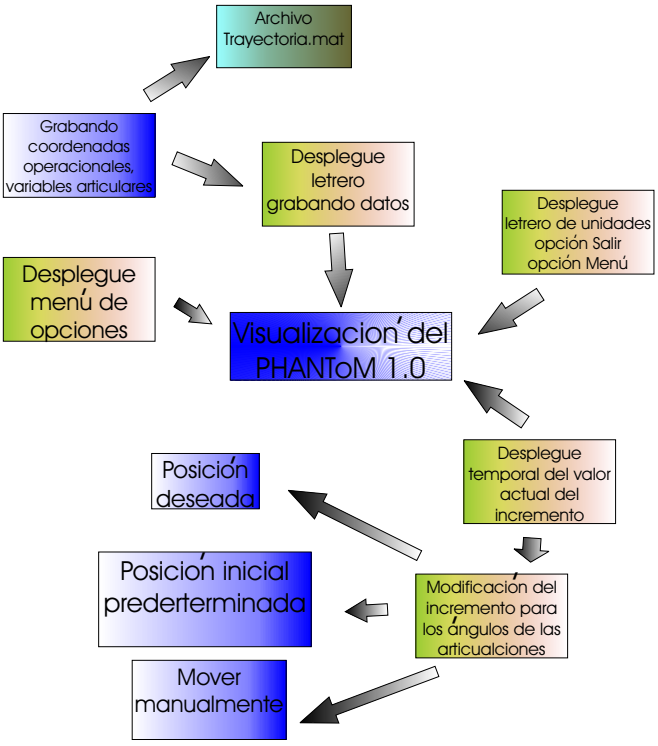


Figura 5.18: Despliegue de opciones

5.8.5. Desplegue de opciones y letreros

Las opciones y letreros que se despliegan, junto a otras funciones, se presentan en el diagrama de la figura 5.18.

Desplegue menú de opciones: aqui se muestran los elementos con los que se opera la interfaz de visualización, asi también cuando se despliega este menú, se tiene la opción de activar algunas funciones marcadas con círculos amarillos empleando el mouse.

Desplegue letrero grabando datos: este letrero se presenta cuando se inicia el proceso de grabar los valores de las variables articulares, coordenadas operacionales y el tiempo que proporciona el reloj de la computadora.

Esta información es grabada en el mismo archivo *Trayctoria.mat*, siendo así para que pueda reproducir los movimientos a través de la opción de mover al PHANToM desde archivo.

Desplegue de letrero de unidades y letrero con opciones salir y menú: El letrero de unidades está permanentemente y se emplea para indicar las unidades de las coordenadas operaciones y de los ángulos de las varibles articulares.

El letrero con opciones salir y menú, indica que con la tecla **M** se despliega el menú y con la tecla **ESC** se termina el programa.

Estas dos opciones también pueden ser activadas con el mouse, haciendo click en el círculo correspondiente.

Desplegue temporal del valor actual del incremento: este letrero se despliega cuando se modifica el incremento para los valores de los ángulos de las articulaciones, presionando las teclas **B**, **N**.

5.9. Conclusiones

Para generar el entorno virtual, es necesario tomar en cuenta todas las características relevantes del dispositivo háptico, pero sin generar latencias. Esto se ha logrado al proponer un dispositivo háptico con elementos mínimos y opciones para modificar el entorno virtual.

Capítulo 6

Conclusiones y perspectivas

6.1. Conclusiones

Se ha presentado el trabajo para la simulación de un robot PHANToM empleando modelos cinemáticos y dinámicos en un ambiente virtual, la simulación cinemática es en línea y la simulación dinámica fuera de línea.
Al inicio de este proyecto se planteó la posibilidad de emplear VRML como lenguaje para la creación de la interfaz virtual; se observó que todos los eventos y cambios de estado para la interacción con el ambiente virtual están condicionados al uso del mouse.
Para lograr producir un comportamiento complejo era necesario emplear alguna estrategia para que los eventos y la interacción fuesen desde otras fuentes, no solamente desde del mouse.
La visualización del robot PHANToM en un ambiente virtual, se desarrolló en Visual C++, empleando librerías gráficas de OpenGL, estas librerías son muy utilizadas para el desarrollo de escenas en 3D y además que Visual C++ es un lenguaje usado para la programación de aplicaciones robustas y consistentes.
La interfaz gráfica de visualización desarrollada cae en la categoría de ambiente virtual de sobremesa.
La construcción del PHANToM virtual se realizó básicamente con 3 eslabones, cada eslabón se construyó a partir de un paralepípedo y se colocaron esferas para mejorar la presentación, cabe mencionar que estas esferas así como el fondo del escenario se pueden retirar para evitar latencia en algunas computadoras.
Esta interfaz de visualización está diseñada para manipular al robot con el teclado, esta característica posibilita que se vinculen eventos capturados mediante una tarjeta de adquisición de datos, puertos serial o paralelo y tarjeta de red, para trabajar con teleoperación.
La interacción con el usuario permite que éste altere el escenario de tal forma que opere con elementos mínimos; además se proporciona un forma de penetrar en el ambiente virtual cambiando la posición de la cámara.
La modelación cinemática se realizó de acuerdo al método Denavit-Hartenberg y la modelación dinámica se realizó mediante la formulación Larange-Euler; este último modelo fue resuelto en Matlab y los resultados se presentaron en la interfaz de visualización virtual.

6.2. Trabajos a futuro

La visualización del comportamiento dinámico se realiza fuera de línea, implementando un método de integración tal como el Runge-Kutta de 4to orden en el programa de visualización, se tendrá un comportamiento en línea.

No se ha abandonado la posibilidad de emplear VRML, Java 3D e inclusive X3D para la creación de robot virtuales, investigando más acerca de estos lenguajes, estamos seguros de que se podrán obtener algunos otros resultados y se tendrán algunas ventajas y desventajas.

El PHANToM se construyó con elementos mínimos, para visualizar uno más parecido es necesario colocar el segundo eslabón paralelo.

Es necesario construir un graficador de trayectorias por donde haya pasado el efector final, por el momento éstas se realizan empleando Matlab.

Apéndice A

Lista de abreviaturas

API: Application Programming Interface

E-L Euler - Lagrange.

K Energía cinética.

P Energía potencial.

IH Interfaz háptica.

J Matriz Jacobiana.

kg Kilogramo.

ME Matriz elemental.

MTH Matriz de transformación homogénea.

mseg Milisegundos.

MCDP Modelo cinemático directo de posición.

MCIP Modelo cinemático inverso de posición.

MCDV Modelo cinemático directo de velocidad.

MCIV Modelo cinemático inverso de velocidad.

MCDA Modelo cinemático directo de aceleración.

MCIA Modelo cinemático inverso de aceleración.

N Newton.

PDH Parámetros Denavit-Hartenberg.

PHANToM: Personal HAptic iNTerface Mechanism

T Periodo.

rad Radian.

RV Realidad virtual.

RGBA: Red Green Blue Alpha

R Robot.

seg Segundos.

V Velocidad de desplazamiento.

ω Velocidad angular.

Z Impedancia.

Apéndice B

Glosario

Coordenada operacional: posición en el espacio definido por los valores x,y,z referidos a un origen.

Modo dislocado: para referirse a la traslación del último eslabón.

Entorno anfitrión: sistema o plataforma

Grados de libertad: conjunto mínimo de variables independientes que se requieren para definir la situación espacial de los eslabones.

Haptic (Haptica): término referido a la retroalimentación de fuerzas por medios sensoriales.

Interfaz: medio de comunicación entre el usuario y la máquina.

Kinestética: sensaciones derivadas de los músculos tendones y articulaciones, estimuladas por el movimiento y la tensión.

Latencia Tiempo de retrazo para ejecutar una acción o visualización una vez solicitada.

Marco dextrogiro: marco de mano derecha.

Modelo: representación abstracta de un sistema

Modelo cinemático: conjunto de expresiones que determinan el movimiento sin considerar las causas que lo producen.

Modelo dinámico: conjunto de expresiones que determinan las causas del movimiento.

Modelo grafico bond: técnica para representar un sistema físico.

OpenGL: librería abierta de gráficos.

Órgano terminal: parte última del sistema que tiene contacto con el objeto que se manipula.

Pipeline: proceso en línea, sin interrupciones

Pixel de trazo: pixel de dibujo, punto en la pantalla de gráficos que dibuja líneas.

Robot: dispositivo manipulador reprogramable y multifuncional diseñado para trasladar materiales, piezas, herramientas o aparatos a través de una serie de movimientos programados para llevar a cabo variedad de tareas.

SGI: Silicon Graphics Iridium.

Stylus: dispositivo para dibujar sobre la pantalla de algunas computadoras

Thimble gimbal: elemento para colocar el dedo del usuario (lápìz virtual)

Utillajes: relieves y cavidades.

Variable articular: valor que puede adquirir el ángulo de la articulación.

Apéndice C

Manual del usuario

La interfaz de visualización del dispositivo Háptico PHANToM 1.0 tiene diferentes formas de operarse, cada una de ellas obedece a lo que el usuario indique, a través del teclado.

Esta interfaz se desarrolló empleando librerías de OpenGL para Visual C++.

C.1. Requerimientos:

Para ejecutar satisfactoriamente la aplicación, se requiere de una computadora **PENTIUM 4**, con **64 MB** y espacio en disco de **2.33MB**; no es necesario tener instalado Visual Studio, pero si se requieren las librarías **GLU32.DLL y glut32.dll** para la interfáz de visualización, y las librerías **MSVCRTD.DLL** y **MFC42D.DLL** para el diálogo de comunicación; estas librerías pueden ubicarse en la misma carpeta de la aplicación o en la carpeta de SYSTEM32.

La resolución de la pantalla debe ser de **800 por 600** pixeles y **calidad de color de 32 bits**.

Si la resolución es mayor, es posible observar la interfaz de visualización, pero no ocupará toda la pantalla.

Si es el caso de tener una mayor resolución, no es conveniente maximizar la pantalla, debido a que no se localizarán las zonas sencibles para las funciones activadas con el mouse.

C.2. Ejecución de la aplicación

La aplicación se ejecuta al pulsar el ícono de **Visualizador.exe**.

El entrono virtual aparecerá centrado en un escenario de fondo, los valores de las coordenadas operacionales y de las variables articulares en la parte superior izquierda, en la parte inferior derecha un letrero permanente para indicar las opciones de menú y salir, como se puede observar en la figura C.1

Es importante resaltar que la ventana de tabajo está sin las pestañas para minimizar, restaurar o salir.

Para salir de la aplicación se presiona la tecla **ESC**.

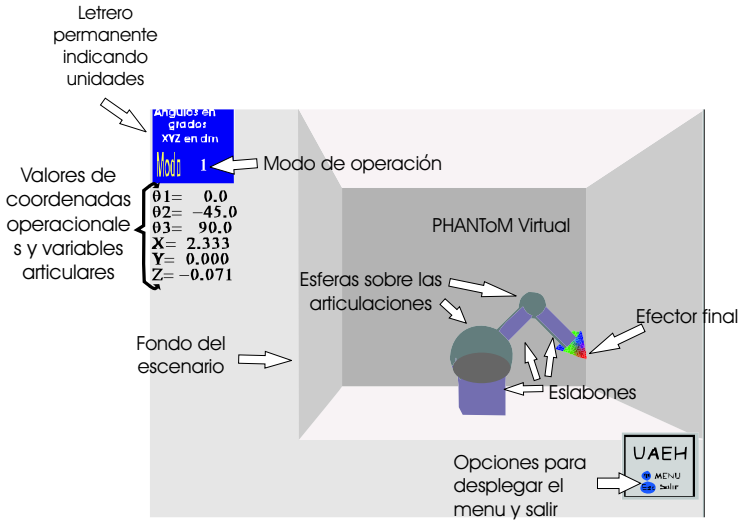


Figura C.1: Entorno inicial

C.3. Operación de la Interfaz virtual

A continuación se presenta la forma de interactuar con el entorno virtual, a través del teclado:

- Presionando la tecla **M**, aparecerá un menu de opciones en la parte derecha y presionando nuevamente la tecla **M** se desaparece.

  En el menú, aparecerán algunas funciones con círculos amarillos; estas funciones se pueden activar con el mouse.

- Presionando las teclas **Q** o **W** se modifica el valor del ángulo de la primera articulación, así también presionando las **A** **S** para la seguda y **Z** **X** para la tercera articulación.

- La posición y orientación de la cámara se modifica con las teclas:

  1. **E** **R** Para acercar o alejar
  2. **D** **F** Arriba o abajo
  3. **C** **V** Para la izquierda o derecha
  4. Empleando las teclas de las flechas $\leftrightarrow$ se hacen rotaciones sobre el eje **X** y con las teclas $\updownarrow$ sobre el eje **Z** *(eje Y para OpenGL)*

- Con las teclas **B** y **N** se modifica el valor del incremento de los ángulos de las articulaciones, para realizar movimentos en apariencia más rápidos o mas lentos.

  Cuando se presiona cualquiera de estas dos teclas, aparecerá un letrero que indica el valor actual del incremento; este valor puede reducirce hasta *0.5* y las articulaciones se moverán lentamente; y se puede aumentar sin límite y los eslabones se moverán rápidamente.

- El espesor de los eslabones también puede ser modificado con las teclas **I O**.

  Para una apariencia proporcional, el espesor inicial es de *0.3* unidades y los límites son de *0.15* a *0.6* unidades.

- Con la tecla **Y** podemos quitar o colocar el escenaio de fondo, con la tecla **U** quitar o colocar las esferas sobre las articulaciones y con la tecla **F1** se dibujan los objetos con líneas y se presentan los ejes de acuerdo a la modelación cinemática

- Con la tecla **F2** se reestablece la posición y orientación de la cámara , es decir que el punto de visión del usuario regresa al inicio.

- Con la tecla **F3** lleva a la posición inicial al robot, regresando a los valores de inicio las articulaciones ($\theta_1 = 0$,$\theta_2 = 0$ y $\theta_3 = 0$)

- Con la tecla **F4** se lleva al robot a una coordenada deseada, esta coordenada es leída desde el archivo *Posicion.pha* localizado en la carpeta *Datos*.

  Este archivo puede ser modificado empleando el diálogo **MCDP.exe** que se activa presionando **F8**

  Si se emplea **MCDP.exe** , le indicará que existen restricciones para los ángulos de las articulaciones 2 y 3, esto es debido a los valores estimados a los cuales se pueden mover las articulaciones del PHANToM real.

- Presionando la tecla **F5** se cargan los datos que contienen el resultado de alguna simulación realizada en Matlab o por una tarea grabada directamente.

  El número de datos previsto en la aplicación es de 2000; este archivo está organizado por *7* columnas: tiempo, θ_1, θ_2, θ_3, x, y, z.

  El archivo de datos se llama **Trayectoria.mat** y es accesado por la operación de la tecla **F5**, la tecla **G**

- Una vez cargados los datos, se presiona la tecla **F6** para ejecutar la simulación correspondiente.

  Para hacer una mejor representación del comportamiento del robot, es necesario conocer la posición inicial dende se inicia la tarea, para que no exista un salto bruzco de la posición donde se encuetra el robot a la posición donde se inicia la simulación.

- Presionando la tecla **G**, se inicia la grabación de datos de las configuraciones que adopta el robot; los cambios se pueden hacer directamente con las teclas para modificar las variables articulares; también si el robot se encuentra inmóvil, los datos se graban.

- Para que se muestre la trayectoria por donde se mueve el efector final, se presiona la tecla **P**; si se desea quitar o borrar la trayectoria, presione nuevamente la tecla **P**.

- Presionando la tecla **F7** varias veces, se selecciona el modo de operación, inicialmente está el modo PHANToM *(1)*, luego el modo dislocado *(2)* y el modo simple *(3)*.

  En el modo PHANToM, se tienen restricciones en los valores de las variables articulares. Estas restricciones en los ángulos se estimaron a partir del PHANToM real que se puede observar en la figura 5.17 del capítulo 5.

  El modo dislocado se implementó para realizar la simulación de oscilación libre de baja fricción y se evitara la invasión del espacio de renderizado.

C.4. Operación en línea

La operación en línea reproduce los movimientos que el usuario proporciona desde el teclado.
Las dos formas de operar el robot virtual en línea son el modo **PHANToM** y el modo **simple.**
Se tiene el modo dislocado para operar el robot virtual; este modo se implmentó para observar la simulación de oscilación libre de baja fricción que se presenta mas adelante.

C.4.1. Modo PHANToM

En este modo se reproduce el comportamiento cinemático del dispositivo PHANToM 1.0.
La primera variable articular se controla con las teclas **Q** para decrementar y **W** para incrementar θ_1.
Esta variable no tiene restricciones en sus valores.
La segunda variable articular se controla con las teclas **A** y **S** para decrementar e incrementar θ_2 respectivamente.
Esta vaiable puede tener valores entre -118^0 y $-12{,}5^0$ grados. Si el usuario intenta pasar de estos valores, el robot no se moverá; mas sin embargo si los valores fueron rebasados por estar operando en otro modo que no contempla restricciones y se selecciona el modo PHANToM, el valor de la articulación retornará a los límites restringidos, dando la apariencia de que el robot se mueve por si solo.

C.4.2. Modo Simple

Este modo de operación, no contempla restricción alguna en las variables articulares.
Los ángulos de las articulaciones pueden tomar cualquier valor, teniendo la desventaja de invadir el espacio de renderizado.
Cuando se mueve al PHANToM virtual desde archivo, este es el modo que se activa para desarrollar la simulación.

C.5. Operación fuera de línea (desde archivo)

Como se menciona en la sección *5.8.1*, esta función se implementó para observar el comportamiento dinámico del PHANToM, esta función requiere del archivo Trayectoria.mat localizado en la carpeta *Datos*.

C.5.1. Graba tarea y reproduce la tarea

Esta función se implementó para grabar una tarea manejada por el usuario desde el teclado y reproducirse posteriormente.

Primero se graba la trayectoria, presionando la tecla **G**; en el ambiente aparece un letrero indicando que se están grabando los datos de posición, en la parte central inferior.

Los datos del tiempo, variables articulares y operacionales se almacenan en el archivo *Trayectoria.mat*.

Para finalizar la grabación de la tarea, se presiona nuevamente la tecla **G**. Y para observar la tarea grabada, se cargan los valores a memoria, presionando la tecla **F5** y posteriormente se ejecuta presionando la tecla **F6**.

C.5.2. Seguimiento de una trayectoria de una circunferencia

Primero se tranfieren los datos al archivo **Trayectoria.mat**, presionando la tecla **F9** o haciendo click con el mouse en el círculo amarillo del menú desplegado.

Una vez hecho lo anterior, se cargan los datos a memoria presionando la tecla **F5** y se ejecuta la visualización presionando la tecla **F6**.

Para observar con mayor detalle el seguimiento de la circunferencia, active la función desple de trayectoria antes de ejecutarla.

C.5.3. Seguimiento de una trayectoria de una rosa de 3 pétalos

Siguiendo el procedimiento anterior, tranfiera los datos presionando la tecla **F11** o haciendo click en el cículo amarillo correspondiente al menú. Cargue los valores presionando la tecla **F5** y ejecútelos presionando la tecla **F6**.

C.5.4. Simulación de oscilación libre de baja fricción

Como se mencionó antes, tranfiera los datos precionando la tecla **F12** o haciendo click en el círculo correspondiente, cargue los valores a memoria y antes de ejecutar la simulación, pase al **modo dislocado (2)** de operación precionando varias veces la tecla **F7** y luego ejecute la simulación.

Apéndice D

Programas auxiliares

La aplicación está acompañada de dos programas auxiliares que no son indispensables para la ejecución del ambiente virtual; el primer programa es **MCDP.exe**, este programa requiere de los valores de los ángulos de las articulaciones θ_1, θ_3 y θ_3 para obtener las coordenadas operacionales **X Y Z**.

Este programa restringe los valores de los ángulos de las articulacíones *2* Y *3* para evitar la invasión del espacio de renderizado de otrar partes el robot.

Cuando un ángulo es incorrecto, se despliega un aviso en el cuadro de texto y no calcula las coordenadas además que no es posible grabar esos datos.

Cuando el valor es satisfactorio, se puede grabar la coordenada y cuando se está de vuelta en la interfaz virtual, el robot se posicionará bajo esos valores de ángulos.

El segundo programa es **Lee_Arreglo_t_q123.exe**, este programa es para presentar los datos del archivo **Trayectoria.mat** y sobre todo la cantidad de datos almacenados, la interfaz esta diseñada para leer 2000 datos, que representa aproximadamente un tiempo de estudio de 40 segundos.

Apéndice E

Estrategias implementadas en la interfaz de visualización

Se dará una explicación de como se implementaron algunos de los métodos para modificar el entorno con una tecla, desplegue temporal del valor del incremento en los ángulos de las articulaciones, la lectura y escritura de datos y el desplegue de la trayectoria.

E.1. Switch con una tecla

Este método se emplea para modificar el escenario presionando solo una tecla y regresar a la condición anterior presionandola nuevamente.
Las características que se modifican en el entorno virtual son:

1. Primer eslabón, normalmente presente.
2. Menu, normalmente no presente.
3. Esferas de las articulaciones, normalmente presentes.
4. Escenario de fondo, normalmente presente.
5. Dibujado en líneas, normalmente desactivado.
6. Asi también para iniciar y terminar la grabación de trayectoria en archivo.

El algoritmo es el siguiente:
Se declaran dos variables de tipo bool, una para la característica y la segunda como auxiliar para eliminar rebotes.
La variable de la característica tendrá el valor que corresponde a su condición normal (presente TRUE, no presente FALSE), la variable auxiliar con valor TRUE, ejemplo:

```
bool DespMenu=FALSE; el menu no aparece normalmente\\
bool Menub=TRUE; variable auxiliar
```

Al presionar la tecla correspondiente se modificará la característica, en este caso con la tecla M se despliega o no el menú.

Si se presionó la tecla **M** y *Menub* es cierta, entonces se niega el valor de DespMenu y Menub=FALSE;, es decir:

```
if (keys['M'] && Menub)
        {
            Menub=FALSE;
            DespMenu=!DespMenu;
        }
```

Es importante asignar el valor a *Menub=FALSE;* para que el cambio de la característica se haga una sola vez mientras esté presionada la tecla.

Para cambiar nuevamente la característica, debe soltarse la tecla y presionarse nuevamente; esto se logra asignando nuevamente el valor a *Menub=TRUE* una vez que se deja de presionar la tecla **M**, es decir:

```
if (!keys['M']) Menub=TRUE;
```

E.2. Desplegue temporal

Este tipo de dsplegue es necesario para mostrar el valor que tiene el incremento de las variables auxiliares para asi modificarlo; este incremento se hace con la variable St y su valor de inicio es de 1.0.
Con las teclas N y B se modifica con valores de ± 0.5.
Para que se desplegue el valor de St es necesario:

- Dibujar un letrero con el valor actual de St.
- Iniciar un contador regresivo que mientras no llegue a cero, permita dibujar el letrero.

Para hacer lo anterior se requeren 2 variables auxiliares:

DespSt variable de tipo boole que se usa como bandera para dibujar o no el letrero, su valor de inicio es FALSE (no dibuje letrero)

CantidadSt variable de tipo entero que se usa para la contabilidad regresiva.

El funcionamiento es el que sigue:
Cada vez que se realiza un cambio, se ejecuta el método Steep()

```
if (keys['N']){St+=0.05f;Steep();}
if (keys['B']){St+=-0.05f;Steep();}
```

En el método Steep() se pueden realizar 3 cosas:

- Si el incremento St<0.05 entonces fija el valor *St=0.05*, para evitar que el incremento no sea cero o negativo.

- Si la negación de DespSt es cierta, es decir que si el letrero no esta desplegado, entonces se indica que se despliegue el letrero (*DespSt=TRUE*) y se inicia el temporizador con *CantidadSt=500*
- Si el letrero está desplegado y el contador regresivo llego a cero, entonces ya no despliegue el letrero (DespSt=FALSE)

Entonces si se despliega el letrero, se ejecuta el método DesplegueSt(); en este método es donde se decrementa la varaible CantidadSt y de proporcionan las características para dibujar la ventana que muestra el incremento en las articulaciones; los símbolos "Dq"presentan en el entorno $\Delta\theta$

E.3. Lectura y escritura de datos

Es necesario leer datos de un archivo que sea el resultado de alguna simulación para recrearla en el entrono virtual; además se deben escribir datos para grabar alguna trayectoria que realice el usuario.
Estos datos se pueden utilizar para graficar la trayectoria en matlab o reproducir nuevamente la tarea en el entorno virtual.
Para tener un flujo de datos de entrada-salida es necesario incluir el archivo cabecera stdio.h, además se declara un puntero del tipo file para leer y escribir datos en archivos:

FILE *KL;

Las opciones de lectura o escritura están listas para usarse una vez que se ejecuta la aplicación; el archivo Trayectoria.mat es donde se tiene la información para leerse y también es este archivo se graban los datos de la trayectoria.
Para grabar y dejar de grabar se emplea la tecla G y para leer datos se emplea la tecla F5.
Se emplean variables auxiliares tales como:

graba: variable de tipo boole para indicar que se graben los datos

archivob: variable del tipo boole que indica si el archivo fue generado satisfactoriamente para grabar.

lee: variable de tipo boole para indicar que se pude seguir leyendo datos de archivo.

gb: variable del tipo boole para la estrategia de switch.

DatoObtenido: variable del tipo bool para indicar que los datos ya fueron extraidos del archivo.

ciclo: variable del tipo entero para llevar una cuenta del número de datos que se han obtenido.

No_Datos: variable del tipo entero para guardar la cantidad de datos obtenidos.

AuxDato: variable auxiliar del tipo flotante usada para guardar temporalmente el dato extraido del archivo.

dato_arch: variable de tipo entero para limitar la cantidad de datos que se pueden almacenar, en este caso 1999+1 dato.

VecTie[2000], VecQ1[2000]**,...** arreglo de variables flotantes para almacenar tiempo y las variables articulares, resultado de una simulación o una trayectoria grabada, estos 2000 datos proporcionan una simulación de 40 segundos proximadamente.

Lectura de datos:
Los datos, como se había mencionado anteriormente de leen de un archivo llamado *Trayectoria.mat*; este archivo puede ser leído también por matlab.

Al presionar la tecla **F5** se ejecuta el método ObtenDatoTrayecto(); en este método se indica que la lectura de datos se ha iniciado: lee=TRUE; a través del puntero KL se abre el archivo en modo de lectura:

```
KL = fopen("Datos/Trayectoria.mat","r");
```

Si se abrió exitosamente el archivo; se posiciona la lectura en el primer registro y se inicia un ciclo de extracción de datos, controlado con la varieble ciclo y el método *SacaDato()*.
Los datos están ordenados de 7 en 7 de la siguiente forma:

1. tiempo
2. θ_1
3. θ_2
4. θ_3
5. $\dot{\theta}_1$
6. $\dot{\theta}_2$
7. $\dot{\theta}_3$

Primero se obtiene el dato tiempo y se verifica que no sea nulo; si es nulo ya se leyeron todos los datos del archivo y se almacena el número de datos en la variable No_Datos=ciclo-1; se indica que ya no hay datos para leer lee=FALSE; y se retorna:
cuando lee=FALSE termina el ciclo y se cierra el archivo fclose(KL).
Pero si no es nulo, entonces obtiene los siguientes 6 datos; los datos 5,6 y 7 no se usan, pero se tienen presentes en el archivo; finalmente se asigna AuxDato=NULL para alistar el siguiente dato.

Grabar datos:
Al presionar la tecla **G**, se realiza lo siguente:

- Se asigna gb=FALSE, para eliminar rebotes
- Se niega la variable graba.

- se asigna a la variable gtime=graba.
- Se verifica: si graba es falso entonces se cierra el archivo y se asigna el valor archivob=FALSE, que indica que no hay mas datos que grabar.

En el método que dibuja el entorno, se verifica si graba es cierta, si lo es, se realiza lo siguiente:

- Si gtime es cierta, entonces se asigna gtime=FALSE, esto para generar una sola vez el archivo Trayectoria.mat y también para obtener una sola vez la referencia del tiempo del reloj de la computadora.
- Se indica que se desplegue el letrero de grabando datos.

También en este método, se ejecuta el método Posición(), donde se
calculan las coordenadar operacionales a través del MCDP que se explica en el capitulo 3 y se verifica si graba es cierta, si lo es entonces se graban los datos del tiempo,θ_1,θ_2 y θ_3 asi como 3 espacios vacios para tener los 7 datos que se explican en los programas de simulación.

E.4. Tiempo del reloj

Para tener acceso al tiempo del reloj de la computadora se requieren de los archivos cabecera:

```
#include <time.h>   //cabecera para el tiempo del reloj
#include<sys/types.h>
#include <sys/timeb.h>
```

También se declaran algunas variables y estructuras:

```
//variables para el tiempo
double segF,segC,tiempo,MiliSeg;
time_t ltime;
struct_timeb tstruct;
```

Cuando se inicia la grabación de datos, se obtiene por una sola vez la referencia de tiempo:

```
time( &ltime );
segF=ltime;   //referencia de tiempo de inicio
```

Cuando se ejecuta el método Posición() y se obtiene el tiempo en segundos de reloj, se le resta el tiempo de referencia para tener un tiempo que inició desde cero, también se obtiene el tiempo en milisegundos; este tiempo en milisegundos se divide entre mil para sumarlo al tiempo en segundos.

```
time( &ltime );
segC=ltime;
tiempo=segC-segF;
_ftime( &tstruct );
MiliSeg=tstruct.millitm;
...
if (archivob) fprintf(KL, "%f %f %f %f %f %f %f\n",
    tiempo+MiliSeg/1000.0,Cteta1,Cteta2,Cteta3,ValorX,ValorY,ValorZ);
```

El tiempo en segundo es corrido, o sea que cuando llegue a 59.999 no se hara cero, sino que continuará.

E.5. Implementación de zonas para activar empleando el mouse

La pantalla es un plano ordenado con coordenadas x,y y el mouse tiene un valor en x y en y. Estos valores son capturados por las variables *mouse_x* y *mouse_y*, empleando las funciones:

```
mouse_x = LOWORD(lParam);
mouse_y = HIWORD(lParam);
```

para las cooredenas x, y respectivamente.

Conociendo la posición del mouse en la pantalla, se evalua en diferentes ecuaciones de círculos, por ejempo:

```
Menu_Circulo=sqrt(pow((mouse_x-184),2)+pow((mouse_y-519),2));//
area del boton menu
```

Y de tal forma que si se cumple la condición:

```
if (14>Menu_Circulo) DespMenu=!DespMenu;
```

Significa que el mouse está en el área de dicho círculo y se indica ejecutar la función correspondiente.

E.6. Desplegue de trayectoria

Para dibujar líneas por donde va pasando el efector final es necesario que el visualizador almacene los puntos por donde ya pasó dicho elemento; para que al re-dibujar el escenario aparezca la trayectoria.

Los puntos se almacenarán todo el tiempo, excepto si el efector final no se mueve o la distancia entre el punto anterior y el actual es corta.

Para lograr lo anterior, es necesario verificar que el punto anterior no sea el mismo punto actual o muy cercano.

Es necesario que se maneje una tecla para iniciar el desplegue de la trayectoria y final del desplegue.

Cuando el usuario indica que se despliegue la trayectoria, se inicia el almacenamiento de los puntos o coordenadas operacionales de la forma como se muestra a continuación:

```
if(DespliegueTrayecto)
    {
        DistaActual=sqrt(pow((Xpasa-ValorX),2)+pow((Ypasa-ValorY),2)+pow((Zpasa-ValorZ)
        if(DistaActual>0.01 && GuarDato<2001)
        {
            GuarDato++;
            AnteX[GuarDato]=ValorX;
            AnteY[GuarDato]=ValorY;
            AnteZ[GuarDato]=ValorZ;
            Xpasa=ValorX;
            Ypasa=ValorY;
            Zpasa=ValorZ;
        }

    }
```

Para dibujar el desplegue de la trayectoria, se emplean los valores almacenados como se puede observar a continuación:

```
    if(DespliegueTrayecto)
    {
        glBegin(GL_LINE_STRIP);
        glColor3f(0.0f,0.0f,0.0f);
            for(int va=1;va<GuarDato+1;va++)
            {
                glVertex3f(AnteX[va],AnteZ[va],-AnteY[va]);
            }
        glEnd ();
    }
```

Apéndice F

Código de la interfaz virtual PHANToM 1.0

La interfaz virtual está desarrollada con librerías de OpenGL aplicadas en visual C++; para correr la aplicación es necesario tener instaladas las librerías glu32.dll y GLUT32.DLL, como se explica en el apéndice C.
Las partes relevantes de los códigos se muestran a continuación (*el código completo se localiza en el CD de esta tesis*):

```
/*Programa Interfaz Grafica PHANToM 1.0
por Herbert Lara Ordaz
asesor Omar A. Dominguez Ramirez
Citis UAEH 2004
Pachuca, Hgo.
Empleando tutoriales 01,06,07 y 13 NeHe por Jeff Molofee
*/
#include <time.h>               //cabecera para el tiempo del reloj
.
.
.
//variables para el tiempo
double segF,segC,tiempo,MiliSeg;
time_t ltime;
struct _timeb tstruct;
//VARIABLES DE ESLABONES definicion en InitGL()
GLfloat LonEslA,LonEslB,LonEslC,EspEsl,Lim_Esp_Alto,Lim_Esp_Bajo;
//Variable para dislocar eslabon
GLfloat Disloca=0.0f;
//variables del mouse
int mouse_x,mouse_y;
//Cantidad temporizador del mensaje st
```

```
int cantidadSt=0;
//Declaracion de la base de la fuente
GLuint  base;
//COMBINACION COLORES BACKGROUND
GLfloat CRojo= 0.56f;
GLfloat CVerde=0.54f;
GLfloat CAzul=1.0f;
//DESPLIEGUE TRAYECTORIA
double DistaActual;
double AnteX[2000],AnteY[2000],AnteZ[2000];
double Xpasa=4.434;
double Ypasa=0.0;
double Zpasa=0.134;
int GuarDato=0;
//variable para 3 estados
int estados=0;
//variable para la aceleracion simulada
float Acelera=0.0;
float Acelera_max=24.0;//aceleracion maxima
//variable para cambiar el paso de incremento
GLfloat St=1.0f;
//***************banderas*************************************************************
bool archivob=FALSE;        //verifica archivo correctamente abierto
bool gtime=FALSE;
bool PosicionInicial=FALSE; //Introducción de datos para movimientos
bool graba=FALSE;           //para gabar datos
bool Fondo=TRUE;            //presenta el fondo
bool Esfera=TRUE;           //dibuja las esferas
bool eje=FALSE;             //dibuja los ejes
bool DespMenu=FALSE;        //presenta el menu
bool DespSt=TRUE;           //presenta el cambio del incremento para theta
bool BandAce;               //aceleración
bool RecorreDinamicab=FALSE;//recreacion comportamiento dinamico
bool lee=TRUE;              //se puede leer dato de archivo
bool DespliegueTrayecto=FALSE;//DESPLIEGUE TRAYECTORIA
//variable para elimenar rebotes en el fondo, menu y esferas
bool efe4b=TRUE;            //posiciona segun dato de archivo
bool efe1b=TRUE;            //auxiliar para ejes
bool gb=TRUE;               //auxiliar para grabar
bool Yb=TRUE;               //auxiliar fondo
bool Ub=TRUE;               //auxiliar esferas
bool Menub=TRUE;            //auxiliar menu
bool efe5b=TRUE;            //carga vectores de datos
```

```
bool efe6b=TRUE;            //ejecuta comportamiento dinamico
bool Despliegueb=TRUE;      //DESPLIEGUE TRAYECTORIA ***
bool DatoObtenido=FALSE;    //no hay datos obtenidos
bool Reestablece_Inmersion=FALSE;//resstablecer la inmerción en el ambiente
bool efe7b=TRUE;            //auxiliares para eliminar rebotes
bool efe8b=TRUE;
bool efe9b=TRUE;
bool efe11b=TRUE;
bool efe12b=TRUE;
bool Eslabon_Uno[2],OneEsla[2];//para establecer estados
bool Salida=FALSE;
bool Mouse_Inicialb=FALSE;
bool Mouse_Deseadab=FALSE;
bool Mouse_Cargab=FALSE;
bool Mouse_Ejecutab=FALSE;
bool Mouse_Dislocab=FALSE;
//*****************************************************************************
//variables de angulo de rotacion Para VISUALIZACION (valores inicialesf)
GLfloat teta1=90.0f;            //primera variable articular
GLfloat teta2=45.0f;            //segunda variable articular
GLfloat teta3=90.0f;                //tercera variable articular
GLfloat Cteta1,Cteta2,Cteta3;   /*variables articulares para cálculos
                                (Cteta angulos para visualizacion)*/
double te1,te2,te3;             //angulos en radianes (para calculos)
double ValorX,ValorY,ValorZ;    //variables de posicion
//diferencia de posicion inicial y final, para auto posicionar
int Diferenciateta1,Diferenciateta2,Diferenciateta3;
int ErrorTotal; //Error total de autoposicion
int ErrorT1,ErrorT2,ErrorT3;//Errores locales de cada theta
//Variables para cambiar la posicion
float PosicionTheta1,PosicionTheta2,PosicionTheta3;
//VARIABLES DE INMERSION (XYZ) y angulos de giro
GLfloat refX=-4.6f;
GLfloat refY=-1.1f;
GLfloat refZ=-6.5f;
GLfloat AnguloY=0.0f;
GLfloat AnguloX=0.0f;
GLfloat Diferencia_Escena=0;

GLfloat Diferencia_refX=0;
GLfloat Diferencia_refY=0;
GLfloat Diferencia_refZ=0;
GLfloat Diferencia_AnguloX=0;
```

```
GLfloat Diferencia_AnguloY=0;
// variables auxiliares para extraer datos de archivo
int ciclo=0;
int No_Datos;
float AuxDato;
//variable del area del menu y salir ESC
float Menu_Circulo;
//variable para el menu
float Menu_Mouse;
//arreglo de variables para trayectoria leida de archivo
int dato_arch=1999;//********** 1999+1 datos para leer ******
float VecTie[2000],VecQ1[2000],VecQ2[2000],VecQ3[2000];
//Metodo para leer datos de archivo a traves del puntero KL
int SacaDato()
{
    fscanf(KL,"%f",&AuxDato);
    VecTie[ciclo]=AuxDato;
    if(VecTie[ciclo]==NULL && ciclo>0) //Ya se obtuvieron todos los datos
    {
        if (ciclo>0) No_Datos=ciclo-1;// regreso de a un dato anterior
        lee=false;      // no se puede leer mas datos de archivo
        return 0;
    }
    if(ciclo==dato_arch)// restriccion para leer solo dato_arch+1 datos
    {
        lee=false;
        return 0;
    }
    fscanf(KL,"%f",&AuxDato);
    VecQ1[ciclo]=AuxDato;
    fscanf(KL,"%f",&AuxDato);
    VecQ2[ciclo]=AuxDato;
    fscanf(KL,"%f",&AuxDato);
    VecQ3[ciclo]=AuxDato;
    fscanf(KL,"%f",&AuxDato);//datos sin usar
    fscanf(KL,"%f",&AuxDato);
    fscanf(KL,"%f",&AuxDato);
    AuxDato=NULL;
    return 0;
}
//Menu con opciones para el mouse
void MenuMouse()
{
```

```
    Menu_Mouse=sqrt(pow((mouse_x-23),2)+pow((mouse_y-433),2));// area del boton f1
    if (14>Menu_Mouse) eje=!eje;

    Menu_Mouse=sqrt(pow((mouse_x-23),2)+pow((mouse_y-465),2));// area del boton f2
    if (14>Menu_Mouse) Reestablece_Inmersion=TRUE;
    Menu_Mouse=sqrt(pow((mouse_x-23),2)+pow((mouse_y-499),2));// area del boton f3
    if (14>Menu_Mouse) Mouse_Inicialb=TRUE;
    Menu_Mouse=sqrt(pow((mouse_x-184),2)+pow((mouse_y-214),2));// area del boton f4
    if (14>Menu_Mouse) Mouse_Deseadab=TRUE;
    Menu_Mouse=sqrt(pow((mouse_x-184),2)+pow((mouse_y-246),2));// area del boton f5
    if (14>Menu_Mouse) Mouse_Cargab=TRUE;;
    Menu_Mouse=sqrt(pow((mouse_x-184),2)+pow((mouse_y-279),2));// area del boton f6
    if (14>Menu_Mouse) Mouse_Ejecutab=TRUE;
    Menu_Mouse=sqrt(pow((mouse_x-184),2)+pow((mouse_y-311),2));// area del boton F7 disl
    if (14>Menu_Mouse) Mouse_Dislocab=TRUE;
    Menu_Mouse=sqrt(pow((mouse_x-184),2)+pow((mouse_y-386),2));// area del boton F8
    if (14>Menu_Mouse) system("MCDP.EXE");
    Menu_Mouse=sqrt(pow((mouse_x-184),2)+pow((mouse_y-421),2));// area del boton f9
    if (14>Menu_Mouse) system("F9.bat");
    Menu_Mouse=sqrt(pow((mouse_x-184),2)+pow((mouse_y-452),2));// area del boton f10
    if (14>Menu_Mouse) system("F11.bat");
    Menu_Mouse=sqrt(pow((mouse_x-184),2)+pow((mouse_y-487),2));// area del boton f11
    if (14>Menu_Mouse) system("F12.bat");
    Menu_Circulo=sqrt(pow((mouse_x-184),2)+pow((mouse_y-519),2));// area del boton menu
    if (14>Menu_Circulo) DespMenu=!DespMenu;

}
//metodo para dibujar letreros de los ejes
void CuadroXYZ(int TexEje)
{
    glBindTexture(GL_TEXTURE_2D, texture[TexEje]);//seleccion de la textura
    glBegin(GL_QUADS);
        glTexCoord2f(0.0f,0.0f);glVertex3f(-0.2f,-0.2f,0.0f);   //bajo izq
        glTexCoord2f(1.0f,0.0f);glVertex3f(0.2f,-0.2f,0.0f);    //bajo der
        glTexCoord2f(1.0f,1.0f);glVertex3f(0.2f,0.2f,0.0f); //arriba der
        glTexCoord2f(0.0f,1.0f);glVertex3f(-0.2f,0.2f,0.0f);    //arriba izq
    glEnd();
}
//Metodo para el comportamiento de informacion de archivo
int ComportaDinamica()
{
    while(ciclo<No_Datos)
    {
```

```
        teta1=VecQ1[ciclo]+90;
        teta2=VecQ2[ciclo]+90;
        teta3=VecQ3[ciclo];
        ciclo++;
        return 0;
    }
    RecorreDinamicab=FALSE;
    return 0;
}
//metodo para leer y almacenar los datos leido de archivo
void ObtenDatoTrayecto()
{
    lee=TRUE;
    KL = fopen("Trayectoria.mat","r");
    if( KL == NULL ) DatoObtenido=FALSE;
    else
    {
        fseek(KL,0L,SEEK_SET);//se soloca en el primer registro
        for(ciclo=0;lee;ciclo++)
        {

            SacaDato();
        }
        DatoObtenido=TRUE;
    }
    fclose( KL );
}
//metodo para calcular los valores para presentar en ambiente
void Posicion() //metodo calcula posicion (coordenadas XYZ)
{
    te1=Cteta1/CoAng;//conversion a radianes
    te2=Cteta2/CoAng;
    te3=Cteta3/CoAng;
    //Modelo cinematico directo de posicion
    ValorX=LonEslC*cos(te1)*cos(te2+te3)+LonEslB*cos(te1)*cos(te2);
    ValorY=LonEslC*sin(te1)*cos(te2+te3)+LonEslB*sin(te1)*cos(te2);
    ValorZ=-LonEslC*sin(te2+te3)-LonEslB*sin(te2);
    if(DespliegueTrayecto)
    {
        DistaActual=sqrt(pow((Xpasa-ValorX),2)+pow((Ypasa-ValorY),2)+pow((Zpasa-ValorZ)
        if(DistaActual>0.01 && GuarDato<2001)
        {
            GuarDato++;
```

```
            AnteX[GuarDato]=ValorX;
            AnteY[GuarDato]=ValorY;
            AnteZ[GuarDato]=ValorZ;
            Xpasa=ValorX;
            Ypasa=ValorY;
            Zpasa=ValorZ;
        }

    }
    //condiciopn para grabar los datos en archivo
    if (graba)
    {
        time( &ltime );
        segC=ltime;
        tiempo=segC-segF;
        _ftime( &tstruct );
        MiliSeg=tstruct.millitm;
        if (archivob) fprintf(KL, "%f %f %f %f %f %f %f\n",
            tiempo+MiliSeg/1000.0,Cteta1,Cteta2,Cteta3,ValorX,ValorY,ValorZ);
    }
}
.
.
.

//Metodo para cambiar el Incremento de los angulos de las articulqciones
void DesplegueSt()
{
        cantidadSt--;                   //Decremento Temporizador
        glBegin(GL_QUADS);
        glColor3f(1.0f,0.90f,0.40f);
        glVertex3f(-0.05f,-2.1f,0.0f);  //bajo izq
        glVertex3f(-0.05f,-1.4f,0.0f);  //arriba izq
        glVertex3f(0.45f,-1.4f,0.0f);   //arriba der
        glVertex3f(0.45f,-2.1f,0.0f);   //bajo der
        glEnd();
        glColor3f(0.0f,0.0f,0.0f);
        glRasterPos2f(0.0f,-1.70f);
        glPrint("Dq");                  //Delta Theta
        glRasterPos2f(-0.1f,-1.86f);
        glPrint("%7.3f",St);
        glColor3f(1.0f,1.0f,1.0f);
}
```

```
void DesplegueGraba()                    //indica grabando datos
{
        glColor3f(1.0f,1.0f,1.0f);
        glBindTexture(GL_TEXTURE_2D, texture[0]);   //seleccion de la textura
        glPushMatrix();
        glBegin(GL_QUADS);
        glTexCoord2f(0.0f,0.0f);glVertex3f(-0.55f,-1.1f,2.30f); //bajo izq
        glTexCoord2f(1.0f,0.0f);glVertex3f(-0.15f,-1.1f,2.30f); //bajo der
        glTexCoord2f(1.0f,1.0f);glVertex3f(-0.15f,-0.75f,2.30f);    //arriba der
        glTexCoord2f(0.0f,1.0f);glVertex3f(-0.55f,-0.75f,2.30f);    //arriba izq
        glEnd();

}
void DesplegueMenu()                              //metodo dibuja menu
{
    glLoadIdentity();
    glColor3f(1.0f,1.0f,1.0f);                    //blanco para textura
    glBindTexture(GL_TEXTURE_2D, texture[1]);     //seleccion de la textura
    glPushMatrix();
        glBegin(GL_QUADS);
            glTexCoord2f(0.0f,0.0f);glVertex3f(-0.085f,-0.06f,-0.15f);//izq abajo
            glTexCoord2f(1.0f,0.0f);glVertex3f(-0.01f,-0.06f,-0.15f);//der abajo
            glTexCoord2f(1.0f,1.0f);glVertex3f(-0.01f,0.015f,-0.15f);//der arriba
            glTexCoord2f(0.0f,1.0f);glVertex3f(-0.085f,0.015f,-0.15f);//izq arriba

        glEnd();

}
int Reestablece_Escenario()
{
    Diferencia_refX=abs(refX+4.6f);
    Diferencia_refY=abs(refY+1.1f);
    Diferencia_refZ=abs(refZ+6.5f);
    Diferencia_AnguloX=abs(AnguloX);
    Diferencia_AnguloY=abs(AnguloY);
    Diferencia_Escena=abs(Diferencia_refX+Diferencia_refY+Diferencia_refZ+
                    Diferencia_AnguloX+Diferencia_AnguloY);
    while (Diferencia_Escena>2.0f)
    {
        if (Diferencia_refX>0.1f)
        {
            if (refX<-4.6f) refX+=0.1f;
            else refX+=-0.1f;
```

```
        }
        if (Diferencia_refY>0.1f)
        {
            if (refY<-1.1f) refY+=0.1f;
            else refY+=-0.1f;
        }
        if (Diferencia_refZ>0.1f)
        {
            if (refZ<-6.5f) refZ+=0.1f;
            else refZ+=-0.1f;
        }
        if (Diferencia_AnguloX>1.0f)
        {
            if (AnguloX<0) AnguloX+=1.0f;
            else AnguloX+=-1.0f;
        }
        if (Diferencia_AnguloY>1.0f)
        {
            if (AnguloY<0) AnguloY+=1.0f;
            else AnguloY+=-1.0f;
        }
        return 0;
    }
    Reestablece_Inmersion=FALSE;
    return 0;

}
void Aceleracion()                      //metodo cambia factor aceleracion
{
    if (Acelera<Acelera_max){Acelera=Acelera+0.01f;}
    BandAce=FALSE;
}

void ImprimePosicion()                  //metodo despliega posicicion (XYZ)
{
    glTranslatef(0.0f,0.0f,-5.0f);  //Traslacion para el texto
    glColor3f(0.0f,0.0f,0.0f);
    glRasterPos2f(-2.45,1.0f);
    glPrint("q1=%7.1f",Cteta1);     //Theta 1,con ajuste de signo para la referencia
    glRasterPos2f(-2.45,0.85f); //Theta 2
    glPrint("q2=%7.1f",Cteta2);     //Theta 2
    glRasterPos2f(-2.45,0.7f);
    glPrint("q3=%7.1f",Cteta3);
```

```
    //posicion XYZ
    glRasterPos2f(-2.45,0.55f);
    glPrint("C=%7.3f",ValorX);      //Posicion X (C imprime X)
    glRasterPos2f(-2.45,0.4f);
    glPrint("U=%7.3f",ValorY);          //Posicion Y (U imprime Y)
    glRasterPos2f(-2.45,0.25f);
    glPrint("Z=%7.3f",ValorZ);          //Posicion Z
}
void Steep()                //Metodo controla el incremento
{
    if (St<0.05f) St=0.05f; //fija incremento minimo
    if (!DespSt)
        {
            DespSt=TRUE;
            cantidadSt=500; //inicia al temporizador
        }
    if (DespSt & (cantidadSt<1)) DespSt=FALSE;  //finaliza temporizador
}
void Triangulo()// dibuja piramide a colores para el efector final
{
    glTranslatef(0.0f,-EspEsl,0.0f);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, EspEsl, 0.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f(-EspEsl,-EspEsl, EspEsl);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f( EspEsl,-EspEsl, EspEsl);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, EspEsl, 0.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f( EspEsl,-EspEsl, EspEsl);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f( EspEsl,-EspEsl, -EspEsl);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, EspEsl, 0.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f( EspEsl,-EspEsl, -EspEsl);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f(-EspEsl,-EspEsl, -EspEsl);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, EspEsl, 0.0f);
        glColor3f(0.0f,0.0f,1.0f);
```

```
        glVertex3f(-EspEsl,-EspEsl,-EspEsl);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f(-EspEsl,-EspEsl, EspEsl);
    glEnd();
}
//metodo para construir un paralepipedo
void Rectangulo(GLfloat esq1,GLfloat esq2,GLfloat esq3)
{
    glTranslatef(-esq1/2,0.0f,-esq3/2); //cambio a la parte central de la figura
    if (eje) glBegin(GL_LINE_LOOP);
    else glBegin(GL_QUADS);
        //frente
        glColor3f(0.4f,0.35f,0.45f);
        glVertex3f(0.0f,0.0f,0.0f);             //bajo der
        glVertex3f(0.0f,esq2,0.0f);             //arriba der
        glVertex3f(esq1,esq2,0.0f);             //arriba izq
        glVertex3f(esq1,0.0f,0.0f);             //bajo izq
        //abajo
        glVertex3f(0.0f,0.0f,0.0f);             //arriba der
        glVertex3f(esq1,0.0f,0.0f);             //arriba izq
        glVertex3f(esq1,0.0f,esq3);             //bajo izq
        glVertex3f(0.0f,0.0f,esq3);             //bajo der
        //arriba
        glVertex3f(0.0f,esq2,0.0f);             //bajo der
        glVertex3f(esq1,esq2,0.0f);             //bajo izq
        glVertex3f(esq1,esq2,esq3);             //arriba izq
        glVertex3f(0.0f,esq2,esq3);             //arriba der
        //atras
        glVertex3f(0.0f,0.0f,esq3);             //arriba der
        glVertex3f(0.0f,esq2,esq3);             //bajo der
        glVertex3f(esq1,esq2,esq3);             //bajo izq
        glVertex3f(esq1,0.0f,esq3);             //arriba izq
        //derecho
        glColor3f(0.45f,0.4f,0.5f);
        glVertex3f(esq1,0.0f,0.0f);             //bajo der
        glVertex3f(esq1,esq2,0.0f);             //arriba der
        glVertex3f(esq1,esq2,esq3);             //arriba izq
        glVertex3f(esq1,0.0f,esq3);             //bajo izq
        //izquierdo
        glVertex3f(0.0f,0.0f,0.0f);             //bajo izq
        glVertex3f(0.0f,esq2,0.0f);             //arrina izq
        glVertex3f(0.0f,esq2,esq3);             //arriba der
        glVertex3f(0.0f,0.0f,esq3);             //bajo der
```

```
    glEnd();
    glTranslatef(esq1/2,0.0f,esq3/2);/*restauracion del cambio a
                                        la parte central de la figura*/
}
//metodo para dibujar los ejes del marco de referencia
void RectanguloEjes(GLfloat esj1,GLfloat esj2,GLfloat esj3) //genera ejes
{
    glBegin(GL_QUADS);
    glColor3f(0.0f,0.10f,0.59f);
//frente el desplazamiento de 1.0f para coincidir el origen del MCDP
        glVertex3f(0.0f,0.0f,0.0f);             //bajo der
        glVertex3f(0.0f,esj2,0.0f);             //arriba der
        glVertex3f(esj1,esj2,0.0f);             //arriba izq
        glVertex3f(esj1,0.0f,0.0f);             //bajo izq
        //abajo
        glVertex3f(0.0f,0.0f,0.0f);             //arriba der
        glVertex3f(esj1,0.0f,0.0f);             //arriba izq
        glVertex3f(esj1,0.0f,esj3);             //bajo izq
        glVertex3f(0.0f,0.0f,esj3);             //bajo der
        //arriba
        glVertex3f(0.0f,esj2,0.0f);             //bajo der
        glVertex3f(esj1,esj2,0.0f);             //bajo izq
        glVertex3f(esj1,esj2,esj3);             //arriba izq
        glVertex3f(0.0f,esj2,esj3);             //arriba der
        //atras
        glVertex3f(0.0f,0.0f,esj3);             //arriba der
        glVertex3f(0.0f,esj2,esj3);             //bajo der
        glVertex3f(esj1,esj2,esj3);             //bajo izq

        glVertex3f(esj1,0.0f,esj3);             //arriba izq
        //derecho
        glColor3f(0.0f,0.20f,0.69f);
        glVertex3f(esj1,0.0f,0.0f);             //bajo der
        glVertex3f(esj1,esj2,0.0f);             //arriba der
        glVertex3f(esj1,esj2,esj3);             //arriba izq
        glVertex3f(esj1,0.0f,esj3);             //bajo izq
        //izquierdo
        glVertex3f(0.0f,0.0f,0.0f);             //bajo izq
        glVertex3f(0.0f,esj2,0.0f);             //arrina izq
        glVertex3f(0.0f,esj2,esj3);             //arriba der
        glVertex3f(0.0f,0.0f,esj3);             //bajo der
    glEnd();
}
```

```
//cambia la posicion del robot a una deseada leida desde archivo
int Reposiciona()
{
    Diferenciateta1=PosicionTheta1-teta1;
    Diferenciateta2=PosicionTheta2-teta2;
    Diferenciateta3=PosicionTheta3-teta3;
    //calcula el error
    ErrorTotal=abs((Diferenciateta1+Diferenciateta2+Diferenciateta3)/(180.0)*100);
    while(ErrorTotal>1)
    {
        ErrorT1=abs(Diferenciateta1/90.0*100);
        ErrorT2=abs(Diferenciateta2/90.0*100);
        ErrorT3=abs(Diferenciateta3*100);
        if(ErrorT1>1)
        {
            if(Diferenciateta1<0)teta1+=-St;
            else teta1+=St;
        }
        if(ErrorT2>1)
        {
            if(Diferenciateta2<0)teta2+=-St;
            else teta2+=St;
        }
        if(ErrorT3>1)
        {
            if(Diferenciateta3<0)teta3+=-St;
            else teta3+=St;
        }
        return 0;
    }

    PosicionInicial=FALSE;
    return 0;
}
//letrero fijo izquierda inferior ESC y MENU
void MenuPermanente()
{
    glColor3f(1.0f,1.0f,1.0f);
        glBindTexture(GL_TEXTURE_2D, texture[3]);   //seleccion de la textura
        glPushMatrix();
        glBegin(GL_QUADS);
        glTexCoord2f(0.0f,0.0f);glVertex3f(1.1f,-1.0f,2.30f);   //bajo izq
        glTexCoord2f(1.0f,0.0f);glVertex3f(1.5f,-1.0f,2.30f);   //bajo der
```

```
        glTexCoord2f(1.0f,1.0f);glVertex3f(1.5f,-0.65f,2.30f);  //arriba der
        glTexCoord2f(0.0f,1.0f);glVertex3f(1.1f,-0.65f,2.30f);  //arriba izq
        glEnd();
        //LETRERO ACLARA UNIDADES
        glBindTexture(GL_TEXTURE_2D, texture[7]);   //seleccion de la textura
        glPushMatrix();
        glBegin(GL_QUADS);

        glTexCoord2f(0.0f,0.0f);glVertex3f(-2.5f,1.2f,-0.1f);   //bajo izq
        glTexCoord2f(1.0f,0.0f);glVertex3f(-1.7f,1.2f,-0.1f);   //bajo der
        glTexCoord2f(1.0f,1.0f);glVertex3f(-1.7f,2.f,-0.1f);    //arriba der
        glTexCoord2f(0.0f,1.0f);glVertex3f(-2.5f,2.f,-0.1f);    //arriba izq
        glEnd();
        glRasterPos2f(-2.0f,1.3f);
        glPrint("%d",estados+1);
}
//genera escenario de fondo
void EscenaFondo()
{
    if (eje) glBegin(GL_LINE_LOOP);
    else glBegin(GL_QUADS);
    glColor3f(0.95f,0.92f,0.93f);
    //abajo piso
        glVertex3f(-3.5f,-LonEslA,2.5f);    //bajo izq
        glVertex3f(-3.5f,-LonEslA,-3.5f);   //arriba izq
        glVertex3f(3.5f,-LonEslA,-3.5f);    //arriba der
        glVertex3f(3.5f,-LonEslA,2.5f); //bajo der
    //izquierdo pared
        glColor3f( 0.29f,0.39f,0.84f);
        glVertex3f(-3.5f,-LonEslA,2.5f);    //arriba izq
        glVertex3f(-3.5f,4.5f,2.5f);    //arriba der
        glVertex3f(-3.5f,4.5f,-3.5f);   //bajo der
        glVertex3f(-3.5f,-LonEslA,-3.5f);   //bajo izq
    //atras fondo
        glColor3f(0.24f,0.28f,0.48f);
        glVertex3f(3.5f,4.5f,-3.5f);    //arriba der
        glVertex3f(3.5f,-LonEslA,-3.5f);    //bajo der
        glVertex3f(-3.5f,-LonEslA,-3.5f);   //bajo izq
        glVertex3f(-3.5f,4.5f,-3.5f);   //arriba izq
    //arriba techo
        glColor3f( 0.95f,0.92f,0.93f);
        glVertex3f(-3.5f,4.5f,-3.5f);   //bajo izq
        glVertex3f(-3.5f,4.5f,2.5f);    //arriba izq
```

```
        glVertex3f(3.5f,4.5f,2.5f);     //arriba der
        glVertex3f(3.5f,4.5f,-3.5f);    //bajo der
    //derecho pared
        glColor3f( 0.29f,0.39f,0.84f);
        glVertex3f(3.5f,4.5f,-3.5f);    //arriba der
        glVertex3f(3.5f,4.5f,2.5f);     //bajo der
        glVertex3f(3.5f,-LonEslA,2.5f); //bajo izq
        glVertex3f(3.5f,-LonEslA,-3.5f);    //arriba izq
    glEnd();

}
.
.
.

int InitGL(GLvoid)                      // Todo el Setup de OpenGL en esta seccion
{
    /*********************************
    definición de las dimensiones de los eslabones en centímetros
    *********************************/
    LonEslA= 1.2f;//unidades en decimetros
    LonEslB= 1.6f;
    LonEslC= 1.7f;
    EspEsl= LonEslA/4.0f;
    Lim_Esp_Alto=2.0f*EspEsl;
    Lim_Esp_Bajo=0.5f*EspEsl;
    Eslabon_Uno[0]=FALSE;//NO DISLOC
    OneEsla[0]=TRUE;//SI RESTRICCION
    Eslabon_Uno[1]=TRUE;//SI DISLO
    OneEsla[1]=FALSE;//NO RESTR
    Eslabon_Uno[2]=FALSE;//NO DISLO
    OneEsla[2]=FALSE;//NO RESTR
.
.
.

}
/*************************
*Dibuja toda la recreacion*
*************************/
int DrawGLScene(GLvoid)
{
    // Limpia pantalla y Depth Buffer
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    Posicion();                 //Calcula la posicion
    ImprimePosicion();          //Dibuja los valores de los angulos y la posicion XYZ
    MenuPermanente();
        //activa el control del cambio de incremento
    if (DespSt & (cantidadSt>0))DesplegueSt();
    if (graba)
    {
        if(gtime)       //obtener el tiempo solo una vez abrir el archivo solo una vez
        {
            gtime=FALSE;
            KL=fopen ("Trayectoria.mat", "w");
            if (KL==NULL) archivob=FALSE;// no es posible grabar hay un problema
            else archivob=TRUE;
            time( &ltime );
            segF=ltime;     //referencia de tiempo de inicio
        }
        DesplegueGraba();
    }
    glTranslatef(refX+6.2f,refY,refZ);//traslada al phantom segun varables de inmersion
    glRotatef(AnguloY,0.0f,1.0f,0.0f);
    glRotatef(AnguloX,1.0f,0.0f,0.0f);
    if (Fondo)EscenaFondo();
if (eje)                        //Dibuja ejes de referencia
    {   //colocacion de letrero para los ejes
        //equis
        glColor3f(1.0f,1.0f,1.0f);
        glTranslatef(2.30f,0.0,0.0f);
        CuadroXYZ(4);
        glTranslatef(-2.30f,0.0,0.0f);
        //ye
        glTranslatef(0.0f,0.0,-2.30f);
        CuadroXYZ(5);
        glTranslatef(0.0f,0.0,2.30f);
        //zeta
        glTranslatef(0.0f,2.3,0.0f);
        CuadroXYZ(6);
        glTranslatef(0.0f,-2.3,0.0f);
        //eje
        //glTranslatef(0.0f,-LonEslA,0.0f);//Ajuste hacia el origen
        RectanguloEjes(2.0f,0.05f,0.05f);//eje x modelo
        RectanguloEjes(0.05f,2.0f,0.05f);//eje Z modelo
```

```
    RectanguloEjes(0.05f,0.05f,-2.0f);//eje y modelo
    //glTranslatef(0.0f,LonEslA,0.0f);//Ajuste hacia el origen
}
if(DespliegueTrayecto)
{
    glBegin(GL_LINE_STRIP);
    glColor3f(0.0f,0.0f,0.0f);
        for(int va=1;va<GuarDato+1;va++)
        {
            glVertex3f(AnteX[va],AnteZ[va],-AnteY[va]);
        }
    glEnd ();
}
glRotatef(teta1,0.0f,1.0f,0.0f);            //rotacion theta 19
//dibujando esferas
if (Esfera && !eje)
{
    glBindTexture(GL_TEXTURE_2D, texture[2]);//seleccion de la textura
    glPushMatrix();
    glColor3f(1.0f,1.0f,1.0f);;
    gluSphere(quadratic,EspEsl*2.30f,32,32);
}
if(!Eslabon_Uno[estados] | OneEsla[estados])
{
    glTranslatef(0.0f,-LonEslA,0.0f);       //baja "Y" 2 veces LongEslA
    Rectangulo(3.0f*EspEsl,LonEslA,3.0f*EspEsl);                    //Dibuja 1er esl
    glTranslatef(0.0f,LonEslA,0.0f);    //traslado para dibujar el 2do rectangulo
}
glRotatef(teta2,1.0f,0.0f,0.0f);
if (Esfera && !eje)
{
    glTranslatef(0.0f,LonEslB,0.0f);
    glColor3f(1.0f,1.0f,1.0f);
    gluSphere(quadratic,EspEsl,32,32);
    glTranslatef(0.0f,-LonEslB,0.0f);
}
Rectangulo(EspEsl,LonEslB,EspEsl);                  //dibuja el 2do eslabon
glTranslatef(Disloca,LonEslB,0.0f);
glRotatef(teta3,1.0f,0.0f,0.0f);
Rectangulo(EspEsl,LonEslC-EspEsl,EspEsl);                   //dibuja el 3er eslabon
glTranslatef(0.0f,LonEslC,0.0f);
Triangulo();
if (DespMenu)DesplegueMenu();
```

```
    return TRUE;                                        // Todo bien, continua
}
.
.
.
        /* Funciones y valores de teclas
        teclas utilizadas
        QW ER AS DF ZX CV BN M G Y U IO F1 F2 F3 F4 F5 F6  F7 P ESC
        */
        if (keys['Q']){Aceleracion();teta1+=-St-Acelera;}   //Q disminuye teta1
        if (keys['W']){Aceleracion();teta1+=St+Acelera;}    //W aumenta teta1
        if (keys['A']){Aceleracion();teta2+=-St-Acelera;}   //A disminuye teta2
        if (keys['S']){Aceleracion();teta2+=St+Acelera;}    //S aumenta teta 2
        if (keys['Z']){Aceleracion();teta3+=-St-Acelera;}   //Z disminuye teta 3
        if (keys['X']){Aceleracion();teta3+=St+Acelera;}    //X aumenta teta 3
        if (keys['E']){refZ+=0.1f;}                         //E ACERCANDO
        if (keys['R']){refZ+=-0.10f;}                       //R ALEJANDO
        if (keys['D']){refY+=0.1f;}                         //D ARRIBA
        if (keys['F']){refY+=-0.1f;}                        //F ABAJO
        if (keys['C']){refX+=-0.1f;}                        //C IZQUIERDA
        if (keys['V']){refX+=0.1f;}                         //V DERECHA
        if (keys['N']){St+=0.05f;Steep();}                  //N disminuye incremento
        if (keys['B']){St+=-0.05f;Steep();}                 //B aumenta incremento
        /*** Rotacion del escenario ***/

        if (keys[VK_LEFT])AnguloY--;
        if (keys[VK_RIGHT])AnguloY++;
        if (keys[VK_DOWN])AnguloX++;
        if (keys[VK_UP])AnguloX--;
        //*** MENU ***
        if (keys['M'] && Menub)
        {
            Menub=FALSE;
            DespMenu=!DespMenu;
        }
        if (!keys['M']) Menub=TRUE;
        //*** DESPLIEGUE TRAYECTORIA ***
        if (keys['P'] && Despliegueb)
        {
            Despliegueb=FALSE;
            DespliegueTrayecto=!DespliegueTrayecto;
            if(!DespliegueTrayecto)
            {
```

```
        GuarDato=0;

    }
}
if (!keys['P']) Despliegueb=TRUE;
//*** FONDO ***
if (keys['Y'] && Yb)
{
    Yb=FALSE;
    Fondo=!Fondo;
}
if (!keys['Y']) Yb=TRUE;
//*** ESFERAS ***
if (keys['U'] && Ub && !eje)
{
    Ub=FALSE;
    Esfera=!Esfera;
}
if (!keys['U']) Ub=TRUE;
//*** ESPESOR ESLABON ***

if (keys['O'])
{
    if (!(EspEsl>Lim_Esp_Alto))EspEsl+=0.005f;
}//aumenta espesor
if (keys['I'])
{
    if (!(EspEsl<Lim_Esp_Bajo)) EspEsl+=-0.005f;//aumenta espesor
}
//*** CARGA Y REPOSICIONA DESDE ARCHIVO ***
if ((keys[VK_F4] && efe4b)| Mouse_Deseadab)
{
    efe4b=FALSE;
    KL=fopen("Datos/Posicion.pha","r");
    if (KL==NULL)
    {
        PosicionTheta1=90.0;

        PosicionTheta2=90.0;
        PosicionTheta3=0.0;
    }
    else
    {
```

```
        fseek( KL, 0L, SEEK_SET );
        fscanf(KL,"%f",&PosicionTheta1);
        fscanf(KL,"%f",&PosicionTheta2);
        fscanf(KL,"%f",&PosicionTheta3);
        PosicionTheta1+=90;//ajuste para la visualizacion
        PosicionTheta2+=90;
        fclose(KL);
    }
    PosicionInicial=TRUE;
    Mouse_Deseadab=FALSE;
}
if (!keys[VK_F4])efe4b=TRUE;
//*** DESPLEGUE EJES ***
if (keys[VK_F1] && efe1b)
{
    efe1b=FALSE;
    eje=!eje;
}
if (!keys[VK_F1])efe1b=TRUE;
//*** REESTABLECE INMERSION ***
if (keys[VK_F2])
{
    Reestablece_Inmersion=TRUE;
}
//*** REPOSICIONA INICIAL ***
if (keys[VK_F3]|Mouse_Inicialb)
{
    PosicionTheta1=90;
    PosicionTheta2=45;
    PosicionTheta3=90;
    PosicionInicial=TRUE;
    Mouse_Inicialb=FALSE;
}
/*** OBTINE DATOS PARA TRAYECTORIA ***/
if ((keys[VK_F5] && efe5b)|Mouse_Cargab)
{
    ObtenDatoTrayecto();
    if (!DatoObtenido)efe5b=FALSE;
    Mouse_Cargab=FALSE;
}
if (!keys[VK_F5]) efe5b=TRUE;
/*** Ejecuta datos de archivo ***/
if ((keys[VK_F6] && efe6b)|Mouse_Ejecutab)
```

```
{
    efe6b=FALSE;
    if (DatoObtenido)RecorreDinamicab=TRUE;
    ciclo=0;            //reinicia la variable
    Mouse_Ejecutab=FALSE;
}
if (!keys[VK_F6]) efe6b=TRUE;
//*** GRABA DATOS EN ARCHIVO ***
if (keys['G'] && gb)
{
    gb=FALSE;
    graba=!graba;
    gtime=graba;//bandera para justar el tiempo transcurrido
    if (!graba)
    {
        fclose(KL);
        archivob=FALSE;
    }
}
if ((keys[VK_F7] && efe7b)|Mouse_Dislocab)
{
    efe7b=FALSE;
    estados++;
    if(estados>2) estados=0;
    if (Eslabon_Uno[estados]) Disloca=EspEsl;
    else Disloca=0.0f;
    Mouse_Dislocab=FALSE;

}
if (!keys[VK_F7]) efe7b=TRUE;
if (!keys['G'] )gb=TRUE;
if (keys[VK_F8] && efe8b)
{
    efe8b=FALSE;
    system("MCDP.EXE");
}
if (!keys[VK_F8]) efe8b=TRUE;
if (keys[VK_F9] && efe9b)
{
    efe9b=FALSE;
    system("F9.bat");
}
if (!keys[VK_F9]) efe9b=TRUE;
```

```
if (keys[VK_F11] && efe11b)
{
    efe11b=FALSE;
    system("F11.bat");
}
if (!keys[VK_F11]) efe11b=TRUE;
if (keys[VK_F12] && efe12b)
{
    efe12b=FALSE;
    system("F12.bat");
}
if (!keys[VK_F12]) efe12b=TRUE;
//*** ACELERACION ***
if (BandAce) {Acelera=0.0f;}
BandAce=TRUE;
//*** REPOSICIONANDO **
if (PosicionInicial)
{
    Reposiciona();
}
/*recreacion comportamiento dinamico*/
if (RecorreDinamicab)
{
    ComportaDinamica();
}
if(Reestablece_Inmersion)
{
    Reestablece_Escenario();
}
//limitacion y correccion por invacion de las articulaciones
if(!RecorreDinamicab && !PosicionInicial && OneEsla[estados])
{
    //*** LIMITE 2DA ARTICULACION ***
    if(teta2>77.59)
    {
        teta2+=-St-Acelera;
    }
    if(teta2<-18.68)
    {
        teta2+=St+Acelera;
    }
    //*** LIMITE 3RA ARTICULACION ***
    if(teta3>110.26)
```

```
            {
                teta3+=-St-Acelera;
            }
            if(teta3<28.74)
            {
                teta3+=St+Acelera;
            }
        }
        //*** ajuste de angulos de visualizacion para calculos***
        Cteta1=teta1-90;
        Cteta2=teta2-90;
        Cteta3=teta3;
    }
    // Shutdown
    KillGLWindow();                                 // Elimina la ventana
    return (msg.wParam);                            // Slir del programa
}
```

Apéndice G

Programas de simulaciones

Estos progrmas se ejecutan con Matlab y resuelven la simulación del comportamiento cinemático y dinámico para tareas diferentes del robot virtual:

1. Trazo de una circunferencia (GrabaCircunferencia.m)
2. Trazo de una rosa de 3 pétalos (GrabaROSA.m)
3. Oscilación libre debaja fricción (GrabaOscilacionLBF.m)

Estos programas están localizados en la carpeta de Datos y pueden ser modificados desde Matlab. Las 3 simulaciones parten de la conficuración del robot siguiente:

1. θ_1=0.7854 rad (45 grados)
2. θ_2=-0.7854 rad (-45 grados)
3. θ_3=-0.7854 rad (-45 grados)

Estas configuraciones de inicio pueden ser modificadas en el programa correspondiente.

Los programas GrabaCircunferencia, GrabaROSA y GrabaOscilacionLBF también resuelven la simulación pero además generan un archivo de resultados ordenados; *7* datos: el tiempo, las *3* variables articulares y las 3 velocidades articulares, que corresponden a la tarea: *TrayectoriaF9.mat*, *TrayectoriaF11.mat* y *TrayectoriaF12.mat* que son usados para la simulación, como se explicó anteriormente.

Apéndice H

Códigos para generar el archivo de trayectoria

Como se describió en el manual de usuario, el robot puede desarrollar una tarea que es leida desde un archivo llamado *Trayectoria.mat* localizado en la carpeta **Datos**.

Para generar este aechivo, se tienen varios programas en Matlab que generan diferentes archivos para las trayectorias correspondientes.

Los programas principales son: *GrabaCircunferencia.m*, *GrabaROSA.m* y *GrabaOscilacionLBF.m*.

De manera general, estos programas hacen uso de la función ode45 de Matlab para resolver las ecuaciones diferenciales de los modelos planteados.

H.1. Código GrabaCircunferencia.m

Este programas generan un archivo llamado *TrayectoriaF9.mat* y contiene la información para realizar una tarea de una circunferencia con su centro ubicado en (1.0,1.0,0.0) y de radio de 1.0; iniciando el movimiento desde los valores de articulaciones: $\theta_1 = 45^0$, $\theta_2 = -45^0$, $\theta_3 = -45^0$.

Se proporciona también las longitudes de los eslabones (L1=1.2, L2=1.6 y L3=1.7)

```
% Script 1 para ejecutar el Programa que Efectúa un Control PD para Seguimiento de
% una Circunferencia en el Dispositivo Haptico Phantom en su Estructura de Posicion
% 3 grados de libertad.
% Elaboraron: Omar Arturo Domínguez Ramírez/Herbert Lara Ordaz CITIS UAEH
% 2005
clear
global dx
dx=zeros(9,1);
options=odeset('MaxStep',0.1,'InitialStep',0.1);
%definicion de parametros del robot longitud de eslabones, angulos
.
```

.
.

H.2. Código Circunferencia.m

Este programa es requerido por el programa GrabaCircunferencia, aqui se tienen definidos algunos parámetros como son las masas y centros de masa de los eslabones asi también las matrices del modelo.

```
% Programa que Efectúa un Control PD para Seguimiento de una Circunferencia en el
% Dispositivo Haptico Phantom en su Estructura de Posicion 3 grados de libertad.
%
%
% Seguimiento de una Circunferencia en el Plano XY con Ecuación (x-h)^2+(y-k)^2=(r)^2
% Elaboraron: Omar Arturo Domínguez Ramírez/Herbert Lara Ordaz CITIS UAEH
% 2005
function dx = Circunferencia(t,x)
%Parámetros supuestos del Robot Phantom y Ganancias
global dx
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% longitud de los eslabones
l1=1.2;
l2=1.6;
l3=1.7;
% Referencia Deseada en el Plano Operacional: Circunferencia C(h,k) y Radio r
h=1;
k=1;
r=1;
w=1;
% masas de los eslabones
m1=0.45;
m2=0.25;
m3=0.15;
.
.
.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                        % Integradores
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dx(1,1)=x(2);
dx(2,1)=qpp(1);
```

```
dx(3,1)=x(4);
dx(4,1)=qpp(2);
dx(5,1)=x(6);
dx(6,1)=qpp(3);
dx(7,1)=Xp(1);
dx(8,1)=Xp(2);
dx(9,1)=Xp(3);
```

H.3. Código GrabaROSA.m

Con este programa se genera un archivo para desarrollar una trayectoria de una rosa de tres pétalos:

```
% Script 1 para ejecutar el Programa que Efectúa un Control PD para Seguimiento de
% una rosa de 3 petalos en el Dispositivo Haptico Phantom en su Estructura de Posicion
% 3 grados de libertad.
%
% Elaboraron: Omar Arturo Domínguez Ramírez/Herbert Lara Ordaz CITIS UAEH
clear
global dx
dx=zeros(9,1);
options=odeset('MaxStep',0.1,'InitialStep',0.1);
%definicion de parametros del robot longitud de eslabones, angulos
.
.
.
```

H.4. Código Rosa.m

El programa GrabaRosa.m requiere del programa ROSA.m, que se muestra a continuación:

```
% Programa que Efectúa un Control PD para Seguimiento de una rosa de 3 petalos
% Dispositivo Haptico Phantom en su Estructura de Posicion 3 grados de
% libertad
function dx = Rosa(t,x)
%Parámetros supuestos del Robot Phantom y Ganancias
.
.
.
```

H.5. Código GrabaOscilacionLBF.m

Este programa genera un archivo llamado TrayectoriaF12.m que contiene la información del comportamiento de movimiento libre y de baja fircción.

```
% Script 1 para ejecutar el Modelo Dinamico del Dispositivo
% Haptico Phantom 1.0, con T=0
% Simulaciòn de Coordenadas y Velocidades Generalizadas, y Coordenadas Operacionales
% Elaboraron: Omar Arturo Domínguez Ramírez/Herbert Lara Ordaz CITIS UAEH
% 2005
clear
global dx
dx=zeros(9,1);
options=odeset('MaxStep',0.1,'InitialStep',0.1);
%definicion de parametros del robot longitud de eslabones, angulos
.
.
.
```

H.6. Código OscilacionLBF.m

Este programa también es requerido por el programa GrabaOscilacionLBF. en este se tienen los datos de las masas de los eslabones (m1=0.45, m2=0.25 y m3=0.15) y sus logitudes (l1=1.2, l2=1.6 y l3=1.7)

```
% Phantom en su Estructura de Posicion 3 grados de libertad.
% Elaboraron: Omar Arturo Domínguez Ramírez/Herbert Lara Ordaz CITIS UAEH
% 2005
function dx = OscilacionLBF(t,x)
%Parámetros supuestos del Robot Phantom y Ganancias
global dx
% longitud de los eslabones
l1=1.2;
l2=1.6;
l3=1.7;
% masas de los eslabones
m1=0.45;
m2=0.25;
m3=0.15;
.
.
.
```

Apéndice I

Artículo arbitrado publicado: *Visualization and Control of Virtual Robot Manipulators*

Bibliografía

[1] Torres Escalante Ma. Teresa Realidad Virtual Universidad Autónoma del Estado de Hidalgo 1999.

[2] Fernando Carbonero Martínez, Creación de entornos 3D con Open-GL y EcosimPro: aplicaciones a la robótica jornada de usuarios de EcosimPro mayo 2001 Madrid España

[3] Tecnologías del tácto http://gayuba1.datsi.fi.upm.es/ suruena/haptic/

[4] Benito José Cuesta Viera, Juan Lucio Cruz Méndez, Leopoldo Acosta Sánchez, Simulación de robots articulados, http://www.cyc.dfis.ull.es/simrob/pdf/proyecto.pdf

[5] Marina Beltrán Blanco, Jorge J. Feliu Batle, José Manuel Cano Izquierdo, SIMUROB, simulador del robot IRB-1400 XXV Jornadas de Automática, ciudad Real, septiembre 2004 www.informatik.uni-bremen.de/simrobot/win16_e.htm

[6] Jonathan Carrera Diaz, Fernando Garrido Reséndiz, Omar Arturo Dominguez Ramirez, Software de simulación del robot PUMA Unimation 560 basado en la cinemática directa e inversa de posición, UAEH 2000

[7] G. W. Rubloff, Dynamic Simulation: Guiding Manufacturing from Process Mechanisms to Factory Operations, http://www.isr.umd.edu/IPDPM/papers/AVS00-dynsim.pdf

[8] Hemeroteca Virtual ANUIES, http://www.hemerodigital.unam.mx/ANUIES

[9] Computer Aided Control System, Design Hybrid Dynamic Systems, Hybrid System Software, http://www-er.df.op.dlr.de/cacsd/hds/software.shtml

[10] Mechatronische Systeme, http://imat.mb.uni-magdeburg.de/Forschung /Schwerpunkte/Schwerpunkte.htm

[11] Páginas de Simnon, http://www.sspa.se/software/simnon.html, http://www.sspa.se/download/sw/tutorial part 3 4.pdf, http://www.mpassociates.gr/software/catalog/sci/simnon/simnon.html

[12] Página principal de Matlab, http://www.mathworks.com/

[13] Guía del software AEC, Mecánica, Gis, CAE, 3D y TDM http://www.edimicros.es/autocad/anuario/anuario-muestra2003.pdf

[14] Sistemas dinámicos y modelos matematicos, Facultad de ciencias exactas ingeniería y agrimesura, Universidad Nacional Rosario http://www.fceia.unr.edu.ar/dsf/files/ A SisDin MM.PDF

[15] Pagina Principal de AutoARQ 2000+, http://www.asuni.es/autoarq/index.asp

[16] Página principal AutoForm, http://www.autoform.com

[17] Página principal Vericut, http://www.samec.es/default.htm

[18] El dispositivo haptico PHANToM, Un dispositivo para tocar objetos virtuales, http://www.sensable.com/community/asme.htm

[19] PHANToM Haptic Device Implemented in a Projection Screen Virtual Environment, http://www.vrac.iastate.edu/∼jmvance/htmls/2003Htmls/Pub.03-PHANToM.htm

[20] Anibal Ollero Baturone, ROBÓTICA Manipuladores y robots móviles, Alfaomega 2001

[21] M.C. Omar A. Dominguz Ramirez Apuntes de seminario de investigaion 1,2 y 3 CITIS UAEH Pachuca, Hgo. 2002

[22] Cinemática del robot, Ingeniería de sistemas y automática, http://lorca.umh.es/isa/es/asignaturas/crss/tema4.pdf

[23] Omar A. Domínguez Ramírez, Herbert Lara Ordaz Ramón Soto de la Cruz, Virgilio López-Morales, Visualization and Control of Virtual Robot Manipulators, Citis, UAEH, CICINDI 2004, ISBN:970-36-0189-8

[24] Página principal WorkNC-CAD, http://www.sescoi.es/sescoies.nsf/worknccad.html

[25] Dibujo y diseño sistema para dibujo en 2D y 3D. http://www.interempresas.net/MetalMecanica/FeriaVirtual /ResenyaProducto.asp?R=3286

[26] Cristina Cañero Morales, Apuntes de OpenGL y GLUT Universidad Politécnica de Madrid Departamento de E.L.A.I. http://www.elai.upm.es/spain/Asignaturas/InfoInd/teoria/Apuntes

[27] Carlos García Trujillo, Programación con Delphi y OpenGL http://glscene.tripod.com

[28] Apuntes de Tópicos avanzados de graficación. Maestria en ciencias computacionles CITIS UAEH

[29] Miguel Angel Seúlveda, Que es OpenGL?, http://www.linuxfocus.org/Castellano /January1998/article15.html

[30] Cartografía y contenidos, http://www.maptel.com/catalogo/cartografia/MAPAS.asp

[31] Procesamiento en el dominio del espacio, http://gente.pue.udlap.mx/∼mramirez/notaspdi/contenido.htm♯ OPERACIONES LOGICO-ARITMETICAS.

[32] Que es OpenGL, http://enigmaprogramming.4t.com/tutoriales/gltutorial01/glTutorial01.html

[33] David J. Kruglinski, George Shepheerd, Scot Wingo, Programación avanzada con Microsoft Visual C++, Mc. Graw Hill Mexico 2000

[34] Que es visual c++, http://www.monografias.com/trabajos5/visualcurso/visualcurso.shtml♯intro1

[35] GLUT Tutorial, initialization, http://www.lighthouse3d.com/opengl/glut/index.php3?2

[36] Jeff Molofee, NeHe Productions: OpenGL Lesson 01 http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=01