

**DESARROLLO DE SOFTWARE PARA MODELADO Y
FABRICACIÓN DE OBJETOS DE MADERA USANDO UN
ROBOT SCARA**

TESIS

Que para obtener el grado de
MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

presenta

Roberto Loaeza Valerio

Dr. Leonardo Romero Muñoz

Director de Tesis

Universidad Michoacana de San Nicolás de Hidalgo

Agosto 2009

A mi familia, etc.....

Resumen

Resumen Español

Abstract

Resumen Ingles

Índice general

Dedicatoria	III
Resumen	V
Abstract	VII
Índice general	IX
Índice de figuras	XIII
Índice de tablas	XV
Índice de códigos	XVII
1. Introducción	19
1.1. Motivación	20
1.2. Antecedentes	21
1.3. Objetivos	21
1.4. Alcances	22
1.5. Metodología	22
1.6. Contribuciones	23
1.7. Descripción de los capítulos	23
2. Ambiente de programación de robots manipuladores	25
2.1. Robot	25

2.1.1. Estructura de los robots manipuladores	27
2.1.2. Clasificación de los robots	29
2.1.3. Programación de tareas	30
2.2. Revisión del Estado del Arte	32
2.2.1. Programación de tareas	32
2.2.2. Desarrollo tecnológico	34
3. Modelado de objetos	37
3.1. Sistemas de coordenadas	37
3.2. Primitivas de graficación	38
3.2.1. Punto	38
3.2.2. Línea	38
3.2.3. Círculo	39
3.2.4. Elipse	40
3.2.5. Curva de Bezier	42
3.2.6. Elementos compuestos	43
3.3. Transformaciones	44
3.3.1. Escalado	45
3.3.2. Traslación	46
3.3.3. Rotación	46
3.3.4. Proyección isométrica	48
3.4. Almacenamiento y recuperación	49
3.4.1. Almacenamiento en formato XML	49
3.4.2. Estructura de almacenamiento	50

3.5. Importación de otros formatos	51
3.5.1. Formato de Imágenes PGM	52
4. Interfaz de software	53
4.1. Diseño de la interfaz de usuario	53
4.1.1. Principios generales de un buen diseño	54
4.2. Lenguaje de programación	56
4.2.1. Java	57
4.3. Software desarrollado	58
4.3.1. Iniciando el software	58
4.3.2. Versiones realizadas	64
5. Interfaz con el robot SCARA	67
5.1. Robot SCARA	67
5.1.1. Características	67
5.1.2. Arquitectura	68
5.1.3. Limitantes	69
5.2. Conversión de modelado a acciones SCARA	69
5.2.1. Cinemática inversa	71
5.2.2. Implementación de la cinemática inversa.	74
5.3. Conectividad PC-Robot	75
5.3.1. Versiones realizadas	75
6. Pruebas	77
6.1. Introducción	77

6.2. Funcionalidad multiplataforma del software 3D	79
6.3. Conectividad multiplataforma entre software y el robot	80
6.4. Pruebas en campo	80
7. Conclusiones	83
7.1. Conclusiones	83
7.2. Sugerencias para trabajos futuros	84
7.2.1. Aplicación desarrollada	84
7.2.2. Electrónica del robot SCARA	85
Glosario	88

Índice de figuras

1.1.	Diagrama representativo de sistema de software desarrollado en esta tesis	21
2.1.	Tipos de articulaciones	28
2.2.	Categorías de los robots manipuladores y sus respectivas áreas de trabajo[Sanchez07].	30
2.3.	Programación de un robot manipulador.	31
3.1.	Sistema de coordenadas de 3 dimensiones	38
3.2.	Línea	40
3.3.	Círculo	41
3.4.	Elipse	41
3.5.	Curva de Bezier	43
3.6.	Elemento compuesto	44
3.7.	Escalamiento	46
3.8.	Traslación	47
3.9.	Rotación	48
3.10.	Ejemplo simple	51
3.11.	Ejemplo PGM	52
4.1.	Pantalla principal	59

4.2. Área de menús	61
4.3. Barra de herramientas de acceso rápido	61
4.4. Barra de herramientas	62
4.5. Vistas disponibles	64
4.6. Área de trabajo	65
5.1. Robot SCARA	68
5.2. Módulo Adapt9S12DP256 del microcontrolador 68hc12[Rodriguez08]	69
5.3. Configuración de un robot SCARA	70
5.4. Orientación del último eslabón	72
5.5. Longitudes L_1 y L_2	72
5.6. Ángulos β , ψ y Θ_1	73
6.1. Aplicación ejecutándose en diferentes plataformas	79
6.2. Ejemplo de 9 primitivas	81
6.3. Ejemplo PGM	81
6.4. Ejemplo PGM realizado en madera	82

Índice de cuadros

2.1. Listado de software CAD/CAM/CNC	34
4.1. Lenguajes de programación y Sistemas Operativos	57
5.1. Bibliotecas para comunicación RS232	75
6.1. Requisitos mínimos del sistema	78
6.2. Tiempos de carga del programa	79
6.3. Puerto serial: nombre y modo de uso	80
6.4. Pruebas y tiempos	80

Índice de códigos

3.1. Punto (primitivas.Punto.java)	39
3.2. Línea (primitivas.Linea.java)	40
3.3. Círculo (primitivas.Circulo.java)	41
3.4. Elipse (primitivas.Eipse.java)	42
3.5. Curva de Bezier (primitivas.Bezier.java)	44
3.6. Estructura de almacenamiento	50
3.7. Código PGM	52
5.1. Cinemática inversa (cinematica.Inversa.java)	74
5.2. Conexión mediante puerto serial (serial.Comunicacion.java)	76

Capítulo 1

Introducción

En la actualidad las pequeñas y medianas empresas de la región maderera del estado de Michoacán requieren tener sus procesos automatizados para poder competir en el mercado tan saturado actualmente. Desafortunadamente la mayoría de estas empresas no cuentan con todos los procesos automatizados.

Los procesos que pueden automatizarse los dividiremos en dos categorías para fines de su estudio:

- Procesos administrativos
- Procesos industriales

Los procesos administrativos abarcan todo proceso relacionado con documentación de la empresa. Algunos ejemplos de estos procesos se dan en los siguientes departamentos: recursos humanos, recursos financieros, ventas, compras, entre otros. Estos procesos están parcial o totalmente automatizados por aplicaciones de computo ofrecidas por el gobierno de forma gratuita o haciendo uso de aplicaciones de terceros

Algunas de las aplicaciones mas comunes utilizadas en esta área administrativa son: ContPAQ, Compiere, NomiPAQ, SUA entre otras.

Por otra parte, los procesos industriales normalmente manipulan una materia prima y la

transforman en un producto final, mediante el uso de maquinaria industrial. Precisamente en estos procesos se centra el trabajo de la presente tesis.

Teniendo las empresas la necesidad de contar con maquinaria industrial de precisión se ven en la necesidad de adquirir esta maquinaria en el extranjero, debido a que en el país no existen empresas que ofrezcan estos productos.

Al usar maquinaria de importación es evidente que los costos de compra, instalación y mantenimiento son elevados, por ejemplo la empresa PROCART de la localidad de Paracho Michoacán adquirió una maquinaria robótica por el costo aproximado de 1,000,000 pesos con un costo de mantenimiento de 10,000. pesos.

Observando la necesidad de la automatización en los procesos de fabricación de productos se optó por desarrollar una aplicación multiplataforma para el modelado de productos de madera haciendo uso de un robot SCARA, construido en la Facultad de Ingeniería Eléctrica de la Universidad Michoacana de San Nicolás de Hidalgo.

1.1. Motivación

Las razones principales que motivaron la elaboración de la presente tesis son las siguientes:

- En la actualidad la mayoría de las pequeñas empresas de las regiones de Michoacán, en particular las de Paracho, no cuentan con procesos automatizados en la manufactura de sus materias primas.
- Por otra parte, existen personas que tienen diseños muy innovadores de productos; pero por falta de recursos económicos para adquirir un robot que lo realice, así como un programa de cómputo para su realización se han quedado solo en ideas que no llegan a materializarse.

1.2. Antecedentes

El antecedente más cercano que se tiene, es el trabajo realizado en la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica, de la Universidad Michoacana de San Nicolás de Hidalgo, que es la construcción de un robot manipulador tipo SCARA, de cuatro grados de libertad de plataforma abierta [Rodriguez08], para el cual se desarrollará la aplicación de la presente tesis.

1.3. Objetivos

El objetivo general es realizar investigación en los aspectos computacionales relacionados con el modelado 3D de objetos físicos y su conversión en instrucciones que el robot SCARA pueda seguir para construir físicamente los modelos en madera, de forma que se desarrolle tecnología propia que promueva una industria robótica y se generen diversas aplicaciones que tengan un importante impacto social y económico.

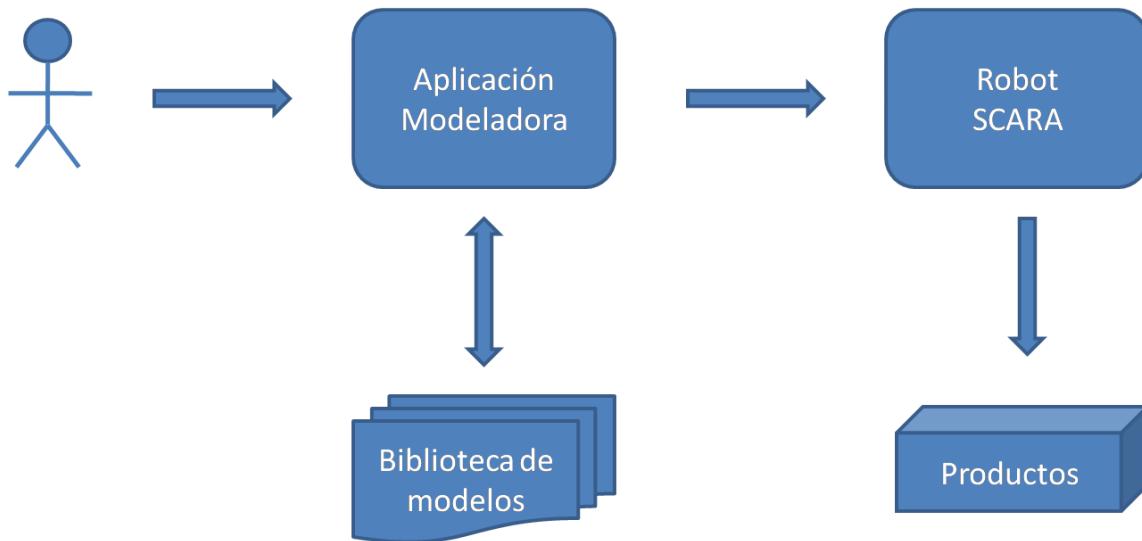


Figura 1.1: Diagrama representativo de sistema de software desarrollado en esta tesis

La figura 1.1 representa gráficamente la ubicación del software de desarrollado; etiquetado como aplicación modeladora. Donde se puede apreciar que el presente trabajo de tesis,

tiene como entrada el diseño por parte de una persona el cual es interpretado para su futura producción en un robot SCARA.

Objetivos específicos

- Desarrollar una aplicación de software para modelado de objetos 3D.
- Desarrollar una interface de comunicación con el robot SCARA.

1.4. Alcances

Los alcances previstos son:

- Implementación de algoritmos de gráficación en 3D.
 - Puntos
 - Líneas
 - Círculos
 - Elipses
 - Curvas de Bezier
- Desarrollo de una aplicación multiplataforma (que pueda ejecutarse en la mayoría de los sistemas operativos actuales) capaz de modelar objetos de madera para su producción utilizando un robot SCARA.
- Desarrollo de interfaz entre la aplicación multiplataforma para modelar objetos de madera y el robot SCARA para la producción.

1.5. Metodología

Para cumplir los objetivos mencionados anteriormente, se propone desarrollar una aplicación multiplataforma para el modelado de objetos de madera en el lenguaje de programación Java.

Para lograr una aplicación aceptable se desarrollarán prototipos de la aplicación hasta lograr una versión que tenga una velocidad aceptable a la hora de generar los modelos en diferentes vistas, así mismo que se genere con rapidez aceptable los movimientos del robot.

Con el fin de mejorar el diseño se evaluará la aplicación continuamente en la empresa de la familia Cardiel Cervantes.

1.6. Contribuciones

A continuación se describen brevemente las contribuciones de este trabajo:

- Desarrollo de una aplicación multiplataforma para la construcción de modelos 3D en madera.
- Aplicación multiplataforma para la conexión entre la aplicación de software 3D y el robot SCARA.
- Aplicación de código abierto, el código fuente se distribuirá bajo la “*GNU General Public License v2*” y puede ser accedido en la siguiente dirección de internet: <http://modelando-madera.googlecode.com> para su mejoramiento y/o ser tomado como base para futuros proyectos.

1.7. Descripción de los capítulos

En el capítulo 2 se presentará una revisión del estado del arte asociado con la paquetería de software existente para el modelado/fabricación de objetos de madera.

El capítulo 3 aborda las partes del software 3D, tanto los algoritmos de graficación como la forma de interacción con el usuario final.

En el capítulo 4 se mostrarán las principales interfaces de software para la implementación de software 3D, se mostrarán pros y contras de cada una de éstas y finalmente se detallará el funcionamiento de la interfaz realizada.

El capítulo 5 trata las características del robot SCARA, la forma de conversión del modelo a acciones que el robot entienda. También se describirá la interfaz entre el software de modelado y el robot.

En el capítulo 6 se mostrarán resultados de pruebas realizadas al software 3D, seguido de los resultados de las pruebas de conectividad entre el software y el robot.

En el capítulo 7 se aportarán las conclusiones generales, resultado de la investigación abordada e ideas para trabajo de investigación posterior a realizar en el mismo campo del conocimiento.

Capítulo 2

Ambiente de programación de robots manipuladores

Por mucho tiempo el hombre ha elaborado máquinas que imitan las partes del cuerpo humano. Los antiguos egipcios unieron brazos mecánicos a las estatuas de sus dioses, éstos brazos fueron operados por sacerdotes, quienes clamaban que el movimiento de estos era inspiración de sus dioses. Los griegos contruyeron estatuas que operaban con sistemas hidráulicos para fascinar a los adoradores de los templos.

2.1. Robot

El inicio de la robótica actual puede fijarse en la industria textil del siglo XVIII, cuando Joseph Jacquard inventa en 1801 una máquina textil programable mediante tarjetas perforadas. La revolución industrial impulsó el desarrollo de estos agentes mecánicos, entre los cuales se destacaron el torno mecánico motorizado de Babbitt (1892) y el mecanismo programable para pintar con spray de Pollard y Roselund (1939). Además de esto durante los siglos XVII y XVIII en Europa fueron construidos muñecos mecánicos muy ingeniosos que tenían algunas características de robots. En 1805, Henri Maillardert construyó una muñeca mecánica que era capaz de hacer dibujos.

26 CAPÍTULO 2. AMBIENTE DE PROGRAMACIÓN DE ROBOTS MANIPULADORES

La palabra robot se empleó por primera vez en una obra de teatro llamada "Ros-sum's Universal Robota" (Los Robots Universales de Rossum) escrita por el dramaturgo checo Karel Capek en 1920. La palabra checa 'Robota' significa servidumbre o trabajador forzado, y cuando se tradujo al inglés se convirtió en el término **robot**.

Sin embargo el término robot con el paso de los años ha cambiado su significado, a continuación se listan algunas de las definiciones más aceptadas:

- Un manipulador multipropósito automáticamente controlado y reprogramable de más de tres ejes/[ISO8373:94].
- Manipulador multifuncional y reprogramable diseñado para mover material, partes, herramientas o dispositivos especializados mediante varios movimientos programados para la realización de una variedad de tareas/[IRA08].
- Mecanismo formado generalmente por elementos en serie, articulados entre sí, destinado al agarre y desplazamiento de objetos. Es multifuncional y puede ser gobernado directamente por un operador humano o dispositivo lógico[Sanchez06].

Para fines de estudio clasificaremos los robots en:

- Manipuladores. Generalmente están montados sobre una base fija que les sirve para definir su área de trabajo y su posición en la misma. Su estructura básica es la de un brazo humano.
- Móviles. Este tipo de robots tienen la capacidad de cambiar de posición por si mismos para realizar alguna tarea determinada. Para lograrlo, tienen sensores que les permiten conocer su ambiente así como actuadores que le permiten desplazarse.

La gran mayoría de los robots usados en la industria son de tipo manipuladores y en adelante solamente nos referiremos a éstos.

2.1.1. Estructura de los robots manipuladores

Básicamente la estructura de un robot manipulador es la de un brazo articulado. Un robot manipulador consta de las siguientes partes:

- Estructura mecánica.
- Sistema sensorial.
- Elementos terminales.
- Sistema de control.

Las cuales se describen brevemente a continuación.

Estructura mecánica

Desde el punto de vista mecánico, el robot está formado por una serie de elementos o eslabones unidos mediante articulaciones, que permiten un movimiento relativo entre cada dos eslabones consecutivos.

Existen diferentes tipos de articulaciones como podemos observar en la figura 2.1:

- a) La articulación de rotación suministra un grado de libertad, es decir, permite la rotación sobre el eje de la articulación.
- b) La articulación prismática, consiste en una translación a lo largo del eje de la articulación.
- c) En la articulación cilíndrica, existen dos grados de libertad, una rotación y una translación sobre el eje de la articulación.
- d) La articulación planar, se caracteriza por el movimiento de desplazamiento en un plano, tiene dos grados de libertad.
- e) Por último, la articulación esférica, combina tres giros en tres direcciones perpendiculares en el espacio.

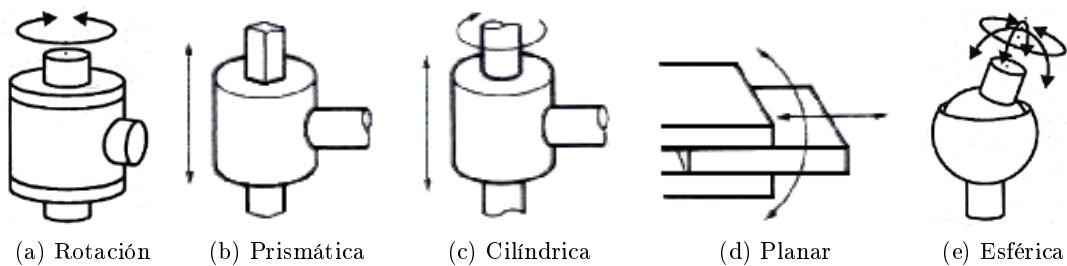


Figura 2.1: Tipos de articulaciones

Sistema sensorial.

Los sensores son transductores que convierten algún fenómeno físico en señales eléctricas que el microcontrolador puede leer [Jones98]. La principal función es trasladar la información del mundo real al mundo abstracto de la computadora, para que ésta pueda responder de forma correcta.

Algunos de los tipos de sensores son:

- De medición de desplazamiento.
- De velocidad y aceleración.
- Detectores de estructuras del ambiente del robot.
- De contacto.

Los interruptores de contacto son dispositivos simples que reportan un valor binario: cerrado o abierto, dependiendo de la ubicación de un elemento mecánico. En particular, el robot SCARA utiliza estos sensores para ubicar los límites de trabajo de cada articulación.

Elementos terminales.

Existen muchos tipos de motores, pero sólo unos cuantos son útiles en robótica, y dentro de éstos se encuentran los motores de corriente directa [Jones98]. Los motores o servomotores de CD son preferidos debido a su relativa facilidad para ser controlados, comparados con otros tipos de motores de CD y con los motores de CA [Estrada06].

Sistema de control.

El sistema de control en un robot es la parte más importante, ya que es aquí donde las acciones lógicas son convertidas en acciones físicas. Estas acciones son realizadas por actuadores y pueden depender de un evento del exterior captado por algún sensor.

2.1.2. Clasificación de los robots

Existe una gran variedad de tipos de brazos manipuladores y se pueden clasificar de distintas formas: por su estructura, por su forma de control, por su área de aplicación, por su fuente de potencia, por su geometría, por su movimiento cinemático, etc. [Spong89, Ramirez98]. Hoy en día, los manipuladores están agrupados en clases de acuerdo a la combinación de uniones usadas en su construcción [Petriu06] como puede observarse en la figura 2.2:

- Cilíndrico. Cuenta con dos articulaciones de rotación y una tipo prismática.
- Esférico. Cuenta con una articulación de rotación y dos de tipo prismática.
- S.C.A.R.A. Cuenta con dos articulaciones de rotación y una tipo prismática.
- Cartesiano. Cuenta con tres articulaciones de tipo prismática.
- Antropomórfico. Cuenta con tres articulaciones de tipo rotación.

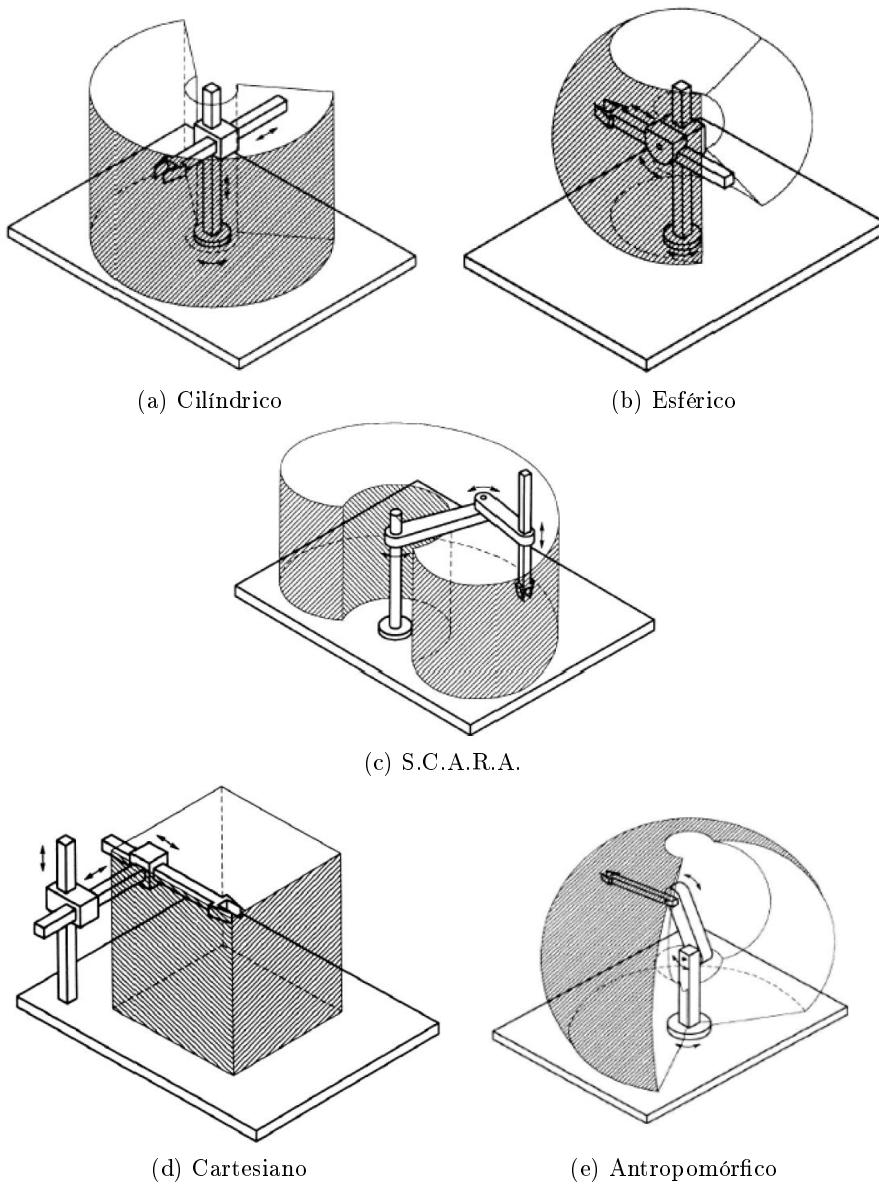


Figura 2.2: Categorías de los robots manipuladores y sus respectivas áreas de trabajo [Sanchez07].

2.1.3. Programación de tareas

La secuencia que se tiene que realizar para programar una tarea en un robot manipulador, generalmente sigue los pasos mostrados en la figura 2.3. En la figura podemos observar tres componentes que se describen a continuación.

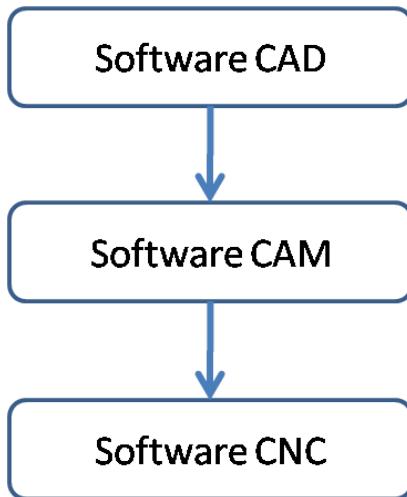


Figura 2.3: Programación de un robot manipulador.

Software CAD

El software CAD (Diseño Asistido por Computadora, del inglés “Computer Aided Design”) son sistemas informáticos para realizar tareas de creación, modificación, análisis y optimización de un diseño.

Software CAM

El software CAM (Manufactura Asistida por Computadora, del inglés “Computer Aided Manufacturing”) son sistemas informáticos para la planificación, gestión y control de las operaciones de una planta de fabricación, mediante una interfaz directa o indirecta entre el sistema informático y los recursos de producción. En una interfaz directa la computadora se conecta directamente con el proceso de producción para monitorizar su actividad y realizar tareas de supervisión y control. En una interfaz indirecta la computadora se utiliza como herramienta de ayuda para la fabricación, pero en las que no existe conexión directa con el proceso de producción.

La entrada de estos sistemas son generalmente los diseños generados con software CAD.

Software CNC

El software CNC (Control Numérico por Computadora, del inglés “Computer Numerical Control”) generalmente lee archivos generados en las aplicaciones CAM y realiza las operaciones de comunicación con el robot necesarias para convertirlas en el movimiento deseado.

2.2. Revisión del Estado del Arte

En la actualidad los robots son usados para realizar tareas peligrosas, difíciles, repetitivas y/o complicadas para los humanos. En industrias grandes se utilizan frecuentemente robots industriales en las líneas de producción. Algunas aplicaciones incluyen la limpieza de residuos tóxicos, minería y localización de minas terrestres.

Sin embargo, la manufactura continúa siendo el principal mercado donde los robots son utilizados, en particular los robots articulados son los más usados comúnmente. Las aplicaciones incluyen soldado, pintado y carga de maquinaria. En este ramo la industria automotriz ha tomado gran ventaja de esta nueva tecnología donde los robots han sido programados para reemplazar el trabajo de los humanos en muchas tareas repetitivas y peligrosas.

2.2.1. Programación de tareas

Para utilizar un robot se requiere, como ya se describió antes, una aplicación CAD-CAM-CNC que sirve de interfaz entre el usuario humano y el robot industrial. Los robots manipuladores de robots se diferencian de la automatización fija, por ser “flexibles”, es decir, son programables. No sólo son programables los movimientos de los manipuladores sino que, a través del uso de sensores y comunicación con otros tipos de automatización, los manipuladores pueden adaptarse a las variaciones requeridas para realizar su tarea.

Generalmente la programación de los robots se realiza fuera de línea y no en tiempo real.

A continuación se listan paquetes de software existentes para programar los robots:

- Microsoft Robotic Studio[mrs2008] con un costo aproximado de \$ 4,000.00 M.N. tiene las siguientes características:

- Herramienta de programación visual para crear y depurar aplicaciones robóticas. El desarrollador puede interactuar con los robots mediante interfaces basadas en web o nativas al sistema operativo (MS Windows).
 - Contiene simulación realística provista por el motor PhysX de AGEIA. Se posibilita la emulación por software o la aceleración por hardware.
 - Se permiten varios lenguajes como: Microsoft Visual Studio Express languages (Visual C#® y Visual Basic® .NET), JScript® y Microsoft IronPython 1.0 Beta 1, y lenguajes de terceros que se adecuen a la arquitectura basada en servicios.
 - Robots soportados: CoroWare's CoroBot (\$3 200), Lego Mindstorms NXT, iRobot Create y Robosoft's robots (38 a 65 K€). Estos robots son de entretenimiento o de propósito educativo.
- KUKA.Officelite[Kuka08] con un costo superior a los \$5,500,000.00 M.N. tiene las siguientes características:
- Este sistema de programación posee las mismas características que el software de sistema KUKA: para el manejo y la programación se utiliza la interfaz de usuario Original KUKA y la sintaxis KRL: un lenguaje completo.
 - Disponibilidad de todo el repertorio de funciones de las respectivas ediciones del software de sistema. Sin embargo, no se pueden conectar dispositivos de hardware periféricos.
 - Comprobación de sintaxis mediante el compilador y el interpretador disponibles; creación de programas KRL de usuario ejecutables.
 - Control completo de la ejecución de un programa de aplicación de robot. Ello permite optimizar la duración de los ciclos.
 - El Techware de KUKA para optimización de programas se puede utilizar e instalar en todo momento. De este modo, en un PC estándar se puede disponer de todo el software de sistema Original, sin necesidad de emulaciones.

- Las conexiones con el robot real se pueden simular.
- KUKA.OfficeLite no se puede utilizar para controlar un robot, solo para desarrollar aplicaciones. De la versión profesional el autor no encontró más información.

Algunos programas de software que pueden ser aplicados a la programación de tareas de robots se pueden observar en la tabla 4.1.

CAD	CAM	CNC
AutoCAD	ArtCAM	TurboCNC
SolidWorks	SolidCAM	EMC2
RhinoCAD	DeskCAM	DeskCNC
TurboCAD	MeshCAM	Mach 3
Graphite One CAD	FreeMill	KCAM

Cuadro 2.1: Listado de software CAD/CAM/CNC

2.2.2. Desarrollo tecnológico

La tecnología robótica propia en México no se ha desarrollado. Existen casos aislados de avance en manipuladores y en móviles.

En instituciones de nivel superior como el Tecnológico de Monterrey campus Guadalajara se trabaja en la planeación de movimientos para robots. En el laboratorio de robótica del Instituto Tecnológico Autónomo de México se enfocan al área de uso de robots móviles pequeños.

En la División de Estudios de Posgrado de la Facultad de Ingeniería Electrica de la Universidad Michoacana de San Nicolas de Hidalgo se han tenido avances significativos tanto en robots móviles así como en manipuladores:

- En el área de robots móviles, se construyó un robot móvil[Concha07] con desempeño comparable al de los robots comerciales. El lector interesado pudo consultar la tesis de licenciatura para información sobre el robot móvil desarrollado.

- En el área de manipuladores se construyeron dos robots SCARA, uno con motores de CD[Ríos07] y otro con motores de CD y de pasos[Rodríguez08].

Para las pruebas del sistema desarrollado en esta tesis, se utilizó el robot de motores de CD y pasos.

Capítulo 3

Modelado de objetos

Una aplicación de modelado de objetos proporciona una biblioteca de funciones, que pueden utilizarse para crear diseños de objetos, que posteriormente se pueden plasmar en madera o algún otro tipo de material que pueda ser moldeado por alguna de las herramientas soportadas por el robot. Estas funciones se denominan primitivas gráficas o simplemente primitivas.

Para describir un modelo, primero es necesario seleccionar un sistema de coordenadas cartesianas adecuado, que puede ser bidimensional o tridimensional. Después se describen los objetos del modelo proporcionando sus especificaciones geométricas. Por ejemplo, se define una línea recta proporcionando la posición de los dos puntos extremos.

3.1. Sistemas de coordenadas

Un sistema de coordenadas es un conjunto de valores que permiten definir exactamente la posición de un punto cualesquiera en el espacio. Debido a que se requiere poder realizar un modelo de un objeto real, se requieren el sistema de coordenadas de tres dimensiones como se muestra en la figura 3.1.

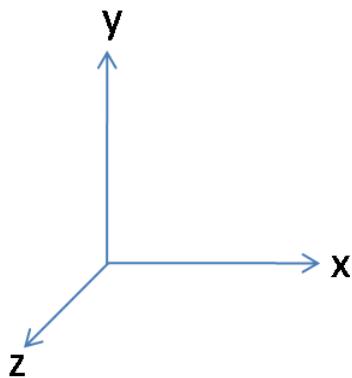


Figura 3.1: Sistema de coordenadas de 3 dimensiones

3.2. Primitivas de graficación

Los elementos que describen la geometría de los objetos se denominan normalmente primitivas geométricas. Entre las primitivas geométricas más simples son las que indican posiciones de puntos y segmentos de líneas rectas. Adicionalmente se pueden incluir círculos, elipses y curvas tipo bezier, las cuales se abordan a continuación.

3.2.1. Punto

El punto es el objeto más simple que puede representarse, es un elemento geométrico que describe una ubicación o posición en el espacio. Un punto puede determinarse en el sistema de coordenadas cartesianas mediante las distancias a los ejes principales, que se indican con dos variables (x, y) en el plano y con tres variables (x, y, z) en el espacio tridimensional.

Para fines de programación se definió un punto en el espacio con la primitiva $Punto(x, y, z)$, puede observarse su implementación en el código 3.1.

3.2.2. Línea

La línea recta es la sucesión continua e indefinida de puntos en una sola dimensión. La ecuaciones utilizadas son:

Código 3.1 Punto (primitivas.Punto.java)

```

public class Punto {
    private double x;
    private double y;
    private double z;
    public Punto(double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }
    public double getX() { return x; }
    public double getY() { return y; }
    public double getZ() { return z; }
    public void setX(double x) { this.x = x; }
    public void setY(double y) { this.y = y; }
    public void setZ(double z) { this.z = z; }
    public String getXML(int i) {
        return "\n\t<x">" + x + "</x">" +
            "\n\t<y">" + y + "</y">" +
            "\n\t<z">" + z + "</z">" ;
    }
}

```

$$x_i = x_1 + \lambda(x_2 - x_1) \quad (3.1)$$

$$y_i = y_1 + \lambda(y_2 - y_1) \quad (3.2)$$

$$z_i = z_1 + \lambda(z_2 - z_1) \quad (3.3)$$

donde (x_1, y_1, z_1) pertenecen al punto inicial del segmento de la recta, (x_2, y_2, z_2) el punto final de la recta y $\lambda \in [0, 1]$. En la figura 3.2 se puede observar un ejemplo de ésta.

Para fines de programación se definió la clase *Línea(a, b)*, donde *a* y *b* son los puntos extremos del segmento de línea, puede observarse su implementación en el código 3.2.

3.2.3. Círculo

La ecuación paramétrica es:

$$\begin{aligned} x &= r * \cos(\Theta) \\ y &= r * \sin(\Theta) \end{aligned} \quad (3.4)$$

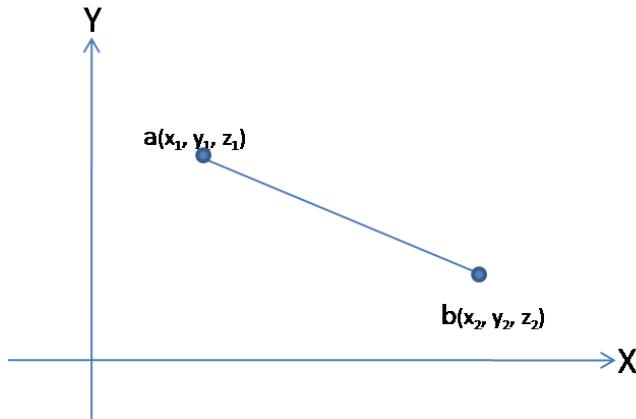


Figura 3.2: Línea

Código 3.2 Línea (primitivas.Linea.java)

```

public void linea( Punto a, Punto b)
{
    int i;
    int n=(int)(dist(a.getX(),a.getY(), b.getX(),b.getY())*ptosPixel);
    double inc_lambda = 1.0 / (n - 1.0);
    for(i=0; i < n; i++) {
        grafPto(new Punto(a.getX() + (i * inc_lambda) * (b.getX() - a.getX()),
                           a.getY() + (i * inc_lambda) * (b.getY() - a.getY()),
                           a.getZ() + (i * inc_lambda) * (b.getZ() - a.getZ()))
        );
    }
}

```

donde $\Theta \in [0, 2\pi]$ y r es el radio.

Para fines de programación se definió la clase $Circulo(c, r)$ donde c es el punto definido como centro y r es el radio o distancia del centro a cualquier otro punto.

En la figura 3.3 se puede observar un ejemplo, puede observarse su implementación en el código 3.3.

3.2.4. Elipse

La ecuación paramétrica es:

$$\begin{aligned}
 x &= r_1 * \cos(\Theta) \\
 y &= r_2 * \sin(\Theta)
 \end{aligned} \tag{3.5}$$

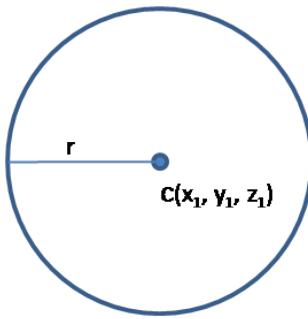


Figura 3.3: Círculo

Código 3.3 Círculo (primitivas.Circulo.java)

```

void circulo(double radio, double oX, double oY, double oZ, int n, int opc)
{
    int i;
    double inc_ang, ang;
    inc_ang = 360.0 / n * Math.PI / 180.0;
    for(i=0; i < n; i++) {
        ang = inc_ang * i;
        switch(opc) {
            case DEF.vistaYX:
                grafPto(new Punto( oX + radio * Math.cos(ang),
                    oY + radio * Math.sin(ang), oZ )); break;
            case DEF.vistaZX:
                grafPto(new Punto( oX + radio * Math.cos(ang),
                    oY, oZ + radio * Math.sin(ang))); break;
            case DEF.vistaZY:
                grafPto(new Punto( oX, oY + radio * Math.cos(ang),
                    oZ + radio_b * Math.sin(ang))); break;
        }
    }
}

```

donde $\Theta \in [0, 2\pi]$, r_1 es la distancia al punto más alejado en el eje X y r_2 es la distancia al punto más alejado sobre el eje Y.

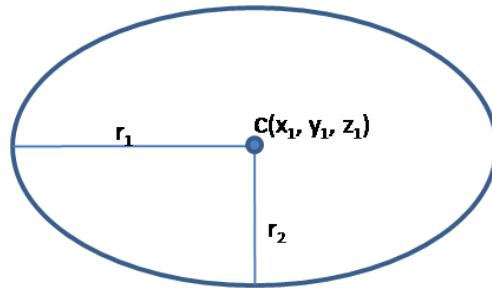


Figura 3.4: Elipse

Para fines de programación se definió la clase *Elipse* (c , r_1 , r_2) donde c es el punto definido

como centro, r_1 es la distancia al punto más alejado en el eje horizontal y r_2 es la distancia al punto más alejado sobre el eje vertical, puede observarse su implementación en el código 3.4.

En la figura 3.4 se puede observar un ejemplo.

Código 3.4 Elipse (primitivas.Elipse.java)

```
void ellipse(double radio_a, double radio_b, double oX, double oY, double oZ,
            int n, int opc)
{
    int i;
    double inc_ang, ang;
    inc_ang = 360.0 / n * Math.PI / 180.0;
    for(i=0; i < n; i++) {
        ang = inc_ang * i;
        switch(opc) {
            case DEF.vistaYX:
                grafPto(new Punto( oX + radio_a * Math.cos(ang),
                                   oY + radio_b * Math.sin(ang), oZ ));    break;
            case DEF.vistaZX:
                grafPto(new Punto( oX + radio_a * Math.cos(ang),
                                   oY, oZ + radio_b * Math.sin(ang)));      break;
            case DEF.vistaZY:
                grafPto(new Punto( oX, oY + radio_a * Math.cos(ang),
                                   oZ + radio_b * Math.sin(ang)));           break;
        }
    }
}
```

3.2.5. Curva de Bezier

La curva Bézier es un tipo de línea curva ideada por un ingeniero de Renault en los años 60 por medio de un método de descripción matemática que conseguía definir las transiciones suaves de las curvaturas.

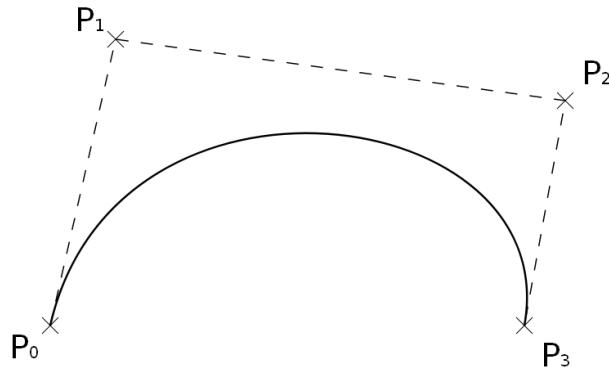


Figura 3.5: Curva de Bezier

Una curva de Bézier tiene por lo menos cuatro puntos de control y es de orden cúbico como se puede apreciar en la figura 3.5, dos de los puntos corresponden a los extremos de la curva, son denominados nodos o puntos de anclaje y los otros puntos son denominados puntos de control o manejadores y determinan la dirección con que la curvatura ingresa a los extremos. La forma paramétrica es:

$$B(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3 \quad (3.6)$$

donde $t \in [0, 1]$.

Para fines de programación se definió la clase *Bezier*(p) donde p es un conjunto definido de puntos que conforman la curva, puede observarse su implementación en el código 3.5.

3.2.6. Elementos compuestos

Un elemento compuesto se forma con uno o más primitivas u otros elementos compuestos. En la figura 3.6 se muestra un ejemplo de un elemento compuesto, que contiene cuatro primitivas. De forma formal tenemos:

$$E_c = (P_1, P_2, P_3, \dots, P_n) \quad (3.7)$$

donde $P_i \in \{Punto, Línea, Polilínea, Círculo, Elipse, Bezier\}$.

Código 3.5 Curva de Bezier (primitivas.Bezier.java)

```

public void graficar(Vector<Punto> v) {
    double step= 1/maxPts;
    double[] Pxi, Pyi, X, Y, t = step;
    Pxi = new double[v.size()]; Pyi = new double[v.size()];
    for(int i=0; i<v.size(); i++) {
        switch(opc) {
            case DEF.vistaYX: Pxi[i]=v.get(i).getX(); Pyi[i]=v.get(i).getY(); break;
            case DEF.vistaZY: Pxi[i]=v.get(i).getY(); Pyi[i]=v.get(i).getZ(); break;
            case DEF.vistaZX: Pxi[i]=v.get(i).getX(); Pyi[i]=v.get(i).getZ(); break;
        }
    }
    Xant = -1000;      Yant = -1000;
    while(t<=1) {
        for (int j = Pxi.length-1; j > 0; j--)
            for (int i = 0; i < j; i++){
                Pxi[i] = (1-t)*Pxi[i] + t*Pxi[i+1];
                Pyi[i] = (1-t)*Pyi[i] + t*Pyi[i+1];
            }
        X = Pxi[0];  Y = Pyi[0];
        if(Xant!=-1000&&Yant!=-1000) {
            switch(opc) {
                case DEF.vistaYX: new Linea(plot, new Punto(Xant, Yant, z),
                                              new Punto(X, Y, z), vista); break;
                case DEF.vistaZY: new Linea(plot, new Punto(x, Xant, Yant),
                                              new Punto(x, X, Y), vista); break;
                case DEF.vistaZX: new Linea(plot, new Punto(Xant, y, Yant),
                                              new Punto(X, y, Y), vista); break;
            }
        }
        Xant = X; Yant = Y; t += step;
    }
}

```

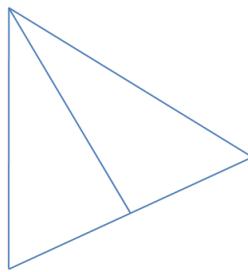


Figura 3.6: Elemento compuesto

3.3. Transformaciones

En robótica son de particular interés las transformaciones: escalado, traslación, rotación y proyección isométrica de un objeto. Estas operaciones tienen la ventaja de que se pueden

realizar mediante un producto matricial y que se pueden combinar multiplicando de antemano las matrices de transformación.

Como se mostró en las secciones anteriores, las primitivas se están definiendo mediante puntos de control, un punto (x, y, z) en el espacio 3D homogéneo será un vector columna:

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.8)$$

A continuación se muestran los productos matriciales para: escalado, traslación, rotación y proyección de un punto de control. Aplicando dichas transformaciones a todos los puntos de control de una primitiva se realiza la transformación a todos los puntos que contiene la misma.

3.3.1. Escalado

A continuación se muestra como quedaría el escalado de un punto de control:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} e_x & 0 & 0 & 0 \\ 0 & e_y & 0 & 0 \\ 0 & 0 & e_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.9)$$

Donde e_x es el factor de escalado en la dirección del eje X , e_y es el factor de escalado en la dirección del eje Y y e_z es el factor de escalado en la dirección del eje Z . El punto escalado $[x', y', z']^t$ proviene del punto original $[x, y, z]^t$.

En la figura 3.7 se muestra un ejemplo de escalamiento con $e_x = 2$ y $e_y = 2$ de los puntos P_1, P_2 y P_3 a los nuevos P'_1, P'_2 y P'_3 .

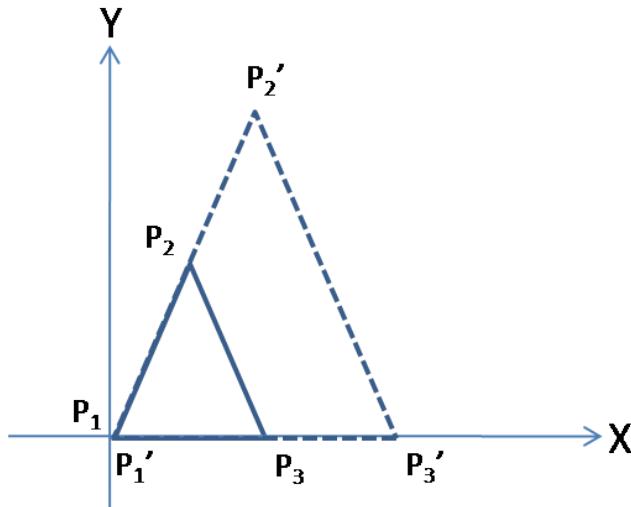


Figura 3.7: Escalamiento

3.3.2. Traslación

La traslación puede ser expresada como el producto:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.10)$$

Donde t_x es el desplazamiento en la dirección del eje X , t_y es el desplazamiento en la dirección del eje Y y t_z es el desplazamiento en la dirección del eje Z . El punto trasladado $[x', y', z']^t$ proviene del punto original $[x, y, z]^t$.

En la figura 3.8 se muestra un ejemplo de traslación con $t_x = 2$ de los puntos P_1 , P_2 y P_3 a los nuevos P'_1 , P'_2 y P'_3 .

3.3.3. Rotación

La rotación alrededor del eje x de un ángulo Θ está definido por:

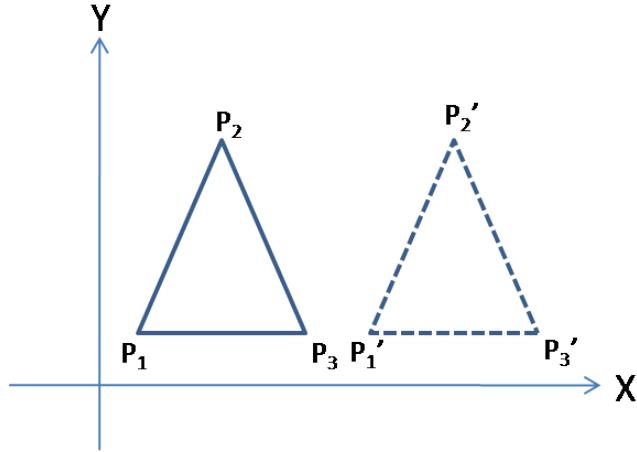


Figura 3.8: Traslación

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Theta) & -\sin(\Theta) & 0 \\ 0 & \sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.11)$$

La rotación alrededor del eje y de un ángulo Θ esta definido por:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\Theta) & 0 & -\sin(\Theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\Theta) & 0 & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.12)$$

La rotación alrededor del eje z de un ángulo Θ esta definido por:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\Theta) & -\sin(\Theta) & 0 & 0 \\ \sin(\Theta) & \cos(\Theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.13)$$

Donde Θ es el ángulo a rotar en el eje correspondiente. El punto rotado $[x', y', z']^t$ proviene

del punto original $[x, y, z]^t$.

En la figura 3.9 se muestra un ejemplo de rotación en el eje X con $\Theta = 45$ de los puntos P_1, P_2 y P_3 a los nuevos P'_1, P'_2 y P'_3 .

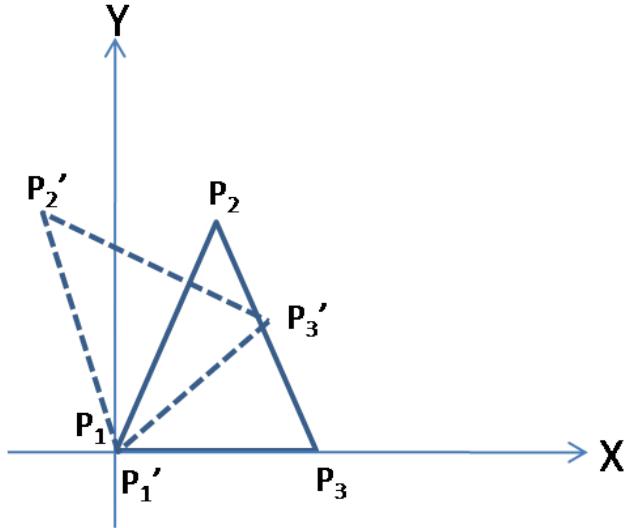


Figura 3.9: Rotación

3.3.4. Proyección isométrica

La proyección isométrica es un método de representación visual de objetos de 3 dimensiones en 2 dimensiones.

Existen 8 diferentes vistas isométricas, dependiendo del octante que se haya elegido. Esta transformación mapea un punto en el espacio 3D en un plano visto desde el primer octante[Carlbom78].

$$\begin{bmatrix} x' \\ y' \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) & 0 \\ 0 & -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.14)$$

donde $\alpha = \arcsin(\tan(30^\circ))$ y $\beta = 45^\circ$. En la ecuación 3.14 se realiza un rotación al rededor del eje Y dado por β , seguido por una rotación alrededor del eje X dado por α , y finalmente se realiza una proyección sobre el plano x-y.

3.4. Almacenamiento y recuperación

A pesar de las mejoras en la tecnología de los dispositivos de almacenamiento, de la mejor protección frente a los virus y de las mejoras en la formación, siguen ocurriendo desastres y caídas de sistemas en mayor número, haciendo de la pérdida de datos un acontecimiento cada vez más habitual.

Teniendo en cuenta lo anterior se eligió usar un formato flexible tanto para la implementación como para la recuperación en caso de alguna perdida parcial de la información, se eligió el formato XML. Brevemente enseguida se explican las características del formato XML.

3.4.1. Almacenamiento en formato XML

El formato XML (del inglés eXtensible Markup Language) es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium[Bray08] como una propuesta de estándar para el intercambio de información estructurada entre diferentes plataformas. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Algunas de las ventajas de este formato son:

- Independiente de la plataforma.
- Soporta código Unicode.
- El mismo documento define la estructura y los campos así como los valores respectivos.
- Está basado en estándares internacionales.

3.4.2. Estructura de almacenamiento

La estructura de almacenamiento puede observarse en el código 3.6 y su representación gráfica en la figura 3.10. Como puede verse la estructura es simple, compacta y fácil de entender.

Código 3.6 Estructura de almacenamiento

```
<modelador>

<fig tipo="1" >
    <x0>32.0</x0><y0>32.0</y0><z0>70.0</z0>
</fig>
<fig tipo="2" >
    <x0>23.0</x0><y0>45.0</y0><z0>70.0</z0>
    <x1>138.0</x1><y1>45.0</y1><z1>70.0</z1>
</fig>
<fig tipo="3" >
    <x0>23.0</x0><y0>61.0</y0><z0>70.0</z0>
    <x1>23.0</x1><y1>107.0</y1><z1>70.0</z1>
    <x2>134.0</x2><y2>108.0</y2><z2>70.0</z2>
    <x3>132.0</x3><y3>62.0</y3><z3>70.0</z3>
    <x4>132.0</x4><y4>62.0</y4><z4>70.0</z4>
</fig>
<fig tipo="4" >
    <x0>76.0</x0><y0>143.0</y0><z0>70.0</z0>
    <x1>105.0</x1><y1>152.0</y1><z1>70.0</z1>
</fig>
<fig tipo="5" >
    <x0>179.0</x0><y0>68.0</y0><z0>70.0</z0>
    <x1>199.0</x1><y1>124.0</y1><z1>70.0</z1>
</fig>
<fig tipo="6" >
    <x0>138.0</x0><y0>171.0</y0><z0>70.0</z0>
    <x1>195.0</x1><y1>166.0</y1><z1>70.0</z1>
    <x2>225.0</x2><y2>130.0</y2><z2>70.0</z2>
    <x3>230.0</x3><y3>55.0</y3><z3>70.0</z3>
</fig>
</modelador>
```

En el segmento de código 3.6 puede observarse:

- Es posible tener una gran cantidad de primitivas en un diseño.
- Los puntos de control de una primitiva pueden no tener límite, es decir, una primitiva puede tener N puntos de control. Sólo aplica a la polilínea y curva de bezier.

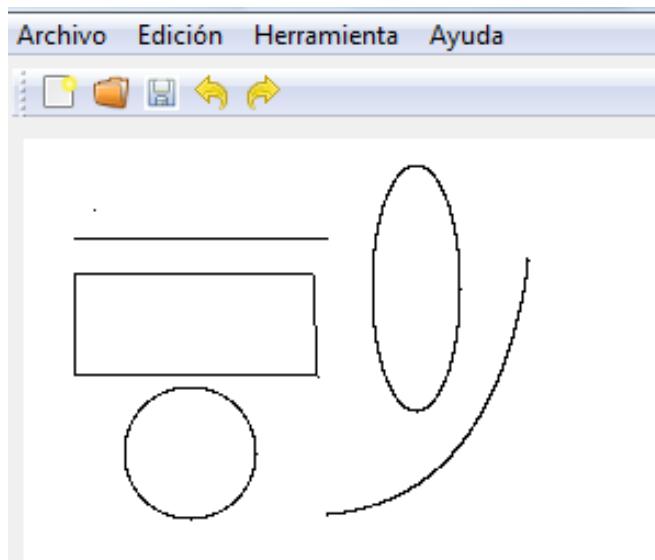


Figura 3.10: Ejemplo simple

- Cada primitiva es identificada mediante un número entero (tipo):
 - Punto (1)
 - Línea (2)
 - Poli-Línea (3)
 - Círculo (4)
 - Elipse (5)
 - Curva de Bezier (6)

3.5. Importación de otros formatos

Una herramienta importante en toda aplicación de diseño es la capacidad de importar formatos externos a ella.

Como primer formato de importación se desarrolló un importador para el formato de imágenes PGM. A petición del Sr. Rafael Cardiel Cervantes se agregó esta capacidad con la intención de hacer figuras en relieve sobre madera a partir de los tonos de grises de una imagen en formato PGM. En seguida se describe brevemente dicho formato.

3.5.1. Formato de Imágenes PGM

El formato PGM (del inglés Portable GrayMap) es un formato para imágenes en tonos de gris de 8 bits sin compresión, diseñado para el fácil intercambio entre plataformas[Poskanzer03]. Existen dos formatos de almacenamiento: texto plano y binario; solamente se trabaja con el formato de texto plano.

Este formato está compuesto por dos partes(ver código 3.7 para un ejemplo):

- Un encabezado que da información sobre el tamaño de la imagen (largo y ancho) seguido del máximo color expresado (en el rango de 0 a 255)
- El cuerpo de la imagen que incluye la información de cada pixel.

La imagen codificada en formato PGM de la figura 3.11 corresponde al formato presentado en el código 3.7.

Código 3.7 Código PGM

```
P2
10 7
255
255 255 255 255 255 255 255 255 255 255
255 0 0 0 255 255 164 164 164 255
255 0 255 0 255 255 164 255 255 255
255 0 0 0 255 255 164 164 255 255
255 0 255 0 255 255 164 255 255 255
255 0 255 0 255 255 164 164 164 255
255 255 255 255 255 255 255 255 255 255
```



Figura 3.11: Ejemplo PGM

Capítulo 4

Interfaz de software

La interfaz de usuario es uno de los apartados con más relevancia de un sistema de cómputo. En la actualidad todo sistema debe contener una herramienta con la que el usuario pueda ordenar a la computadora qué hacer. Una interfaz de usuario mal diseñada puede causar que el mejor sistema de cómputo sea ineficaz y por lo tanto sea desechado, es por esto que se debe realizar un buen diseño de la interfaz de usuario, la cual incluye:

- Consideraciones sobre el diseño de la interfaz gráfica.
- La elección de un lenguaje de programación apropiado.

En este capítulo se presentan algunos elementos importantes del diseño de interfaces gráficas, el lenguaje de programación y el software de modelado desarrollado por el autor.

4.1. Diseño de la interfaz de usuario

La Interfaz Gráfica de Usuario (IGU) es la parte más importante de los sistemas computarizados debido a que el usuario interactúa de forma directa con ella. Las metas de una IGU son lograr que el trabajo con la computadora sea fácil, productivo y agradable [Galitz07].

La IGU se conforma de dos componentes: entrada y salida. La entrada es la forma en que los usuarios comunican sus necesidades a la computadora. La salida es el medio por el cual la computadora muestra los resultados de las operaciones requeridas por el usuario.

Propiamente, una interfaz de usuario provee la mezcla de: mecanismo de entrada y salida que de manera eficiente satisfagan las necesidades del usuario, capacidades, y limitaciones en la forma más eficiente.

4.1.1. Principios generales de un buen diseño

Enseguida se revizan algunos principios generales para la IGU que deben ser tomados en cuenta [Galitz07]:

Accesibilidad

El sistema deberá poder ser usado por personas de diversas capacidades y limitantes. Originalmente este término fue usado para referirse a sistemas accesibles a usuarios con discapacidades. Actualmente el término accesibilidad se refiere a cubrir las necesidades de la mayoría de los usuarios.

Las principales características de un sistema accesible son: Perceptibilidad, Operabilidad y Simplicidad.

Disponibilidad

Todos los aspectos de un sistema deberán de estar disponibles en cualquier momento. Sólo deberán no estar disponibles en aquellas situaciones que no tenga sentido.

Claridad

La interfaz debe ser clara en apariencia visual y conceptual. Los elementos visuales deben ser entendibles, relacionados con objetos del mundo real. Los conceptos y textos deben ser simples y no confusos.

Consistencia

Consistencia es la uniformidad en apariencia y localización de los objetos dentro de la IGU. Es importante ya que puede reducir la necesidad de adquirir nuevas habilidades para una actividad que puede ser similar a otra. Si un nuevo sistema impone necesidades de aprender nuevas habilidades en los usuarios, este puede convertirse en un sistema no productivo e innecesario.

Un sistema deberá lucir y operar de forma consistente, es decir:

- Una acción realizada sobre los mismos datos siempre deberá devolver el mismo resultado.
- La posición de los elementos estándares no cambiará.

Control

El usuario debe tener control sobre las acciones que está realizando el sistema.

Eficiencia

En cada paso de un proceso, se debe mostrar al usuario toda la información y herramientas necesarias para terminar el proceso. El usuario no debe tener la necesidad de buscar información ni herramientas en lugares externos al proceso activo.

Tiempo de respuesta

Cuando un usuario realice una solicitud el sistema debe contestar lo más rápido que le sea posible. Si al realizar una solicitud el sistema no responde en un tiempo considerable, puede ser considerado como que el sistema ha fallado.

Recuperación

El sistema debe contar con un sistema de recuperación que permita deshacer una acción. Con este mecanismo se reduce en gran medida los errores de usuarios nuevos. El punto de

retorno puede ser hacia la acción anterior, pantalla anterior o al inicio de un determinado periodo de tiempo.

El objetivo es mantener la estabilidad, es decir, cuando el usuario realice una acción errónea que pueda llevar a una situación peligrosa exista alguna forma de regresar al punto anterior estable. La recuperación debe ser obvia, automática y simple de realizar.

Simplicidad

La simplicidad es reconocida cuando cualquier usuario puede de manera simple entender y usar un sistema con la mínima experiencia y documentación.

Visibilidad

Los sistemas son más usables cuando de una manera clara indican el estado y los resultados de las acciones realizadas por los usuarios.

4.2. Lenguaje de programación

En la actualidad existe una gran cantidad de lenguajes de programación, la mayoría de propósito general y el resto de propósito específico, a continuación se listan los más populares de ambas categorías: C, Basic, Java, MS .Net, Perl y PHP.

Estos podrían ser los más utilizados en la actualidad, pero debido a la heterogeneidad de sistemas operativos y arquitecturas, no todos los lenguajes de programación son aplicables en todos los casos.

En el caso de los lenguajes compilados es poco probable su funcionalidad en diferentes plataformas debido a la dependencia de tipo de procesador que adquieren al momento de compilar, por ejemplo si se compila en un procesador tipo RISC es imposible ejecutarlo en un procesador tipo CISC.

Por otro lado, los lenguajes interpretados son muy portables a todas las arquitecturas y sistemas operativos, debido a que no dependen de un tipo de arquitectura, pero requieren de

que exista un programa nativo que los interprete, es decir, que si la empresa desarrolladora del lenguaje interpretado no libera una versión de su intérprete para un sistema operativo en una arquitectura determinada, los programas escritos en este lenguaje no se ejecutarán.

	Windows	Linux	MacOS	Portabilidad
C	X	X	X	
Basic	X			
Java	X	X	X	X
.Net	X	X		
Perl	X	X	X	
PHP	X	X		X

Cuadro 4.1: Lenguajes de programación y Sistemas Operativos

En la tabla 4.1 se puede apreciar los alcances de los lenguajes de programación más desarrollados y más difundidos en la actualidad, dejando claro que Java es el lenguaje de programación con mayor portabilidad, debido a que se cuenta con intérpretes para todas las arquitecturas y además que no se requiere modificar el código fuente para ejecutarse en éstas. En seguida se describe brevemente dicho lenguaje de programación.

4.2.1. Java

Java es una plataforma de software desarrollada por Sun Microsystem, de forma que los programas creados en ella, puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y dispositivos computacionales.

Esta plataforma está compuesta por:

- El Lenguaje de programación.
- La máquina virtual de Java (JVM).
- Un conjunto de bibliotecas estándar, conocidas como API.

El lenguaje de programación usa una sintaxis similar al lenguaje C++, incorpora sincronización, manejo de tareas e interfaces como un mecanismo alternativo a la herencia múltiple de C++.

La máquina virtual de Java (JVM) es un programa nativo (un ejecutable de una plataforma específica) capaz de interpretar y ejecutar código binario especial (llamado Java Bytecode), el cual es generado a partir del compilador de Java. Es por esto que Sun Microsystem ha liberado versiones de su JVM para las arquitecturas y sistemas operativos más utilizados.

Debido al desarrollo de JVM en los sistemas operativos, las aplicaciones desarrolladas en Java se convierten en aplicaciones multiplataforma. Es por esto que para desarrollar la aplicación de modelado se utilizó Java. En seguida se describe las secciones más relevantes de la aplicación.

4.3. Software desarrollado

El software desarrollado fue incluyó la biblioteca SWT desarrollada por IBM, la cual obtiene una apariencia idéntica a las aplicaciones nativas al sistema operativo, logrando cumplir así unos de los principios fundamentales de un buen diseño de interfaz gráfica “consistencia”. Es decir, mediante la biblioteca SWT se uniformiza la apariencia con las aplicaciones del sistema operativo anfitrión.

4.3.1. Iniciando el software

Al iniciar la aplicación nos encontramos con una pantalla como la que se muestra en la figura 4.1, donde se pueden resaltar cinco secciones importantes:

- Área de menús (1).
- Barra de herramientas de acceso rápido (2).
- Barra de herramientas de expansión (3).
- Área de trabajo (4).
- Descripción del punto actual (5).

A continuación se describe brevemente cada una de estas secciones.

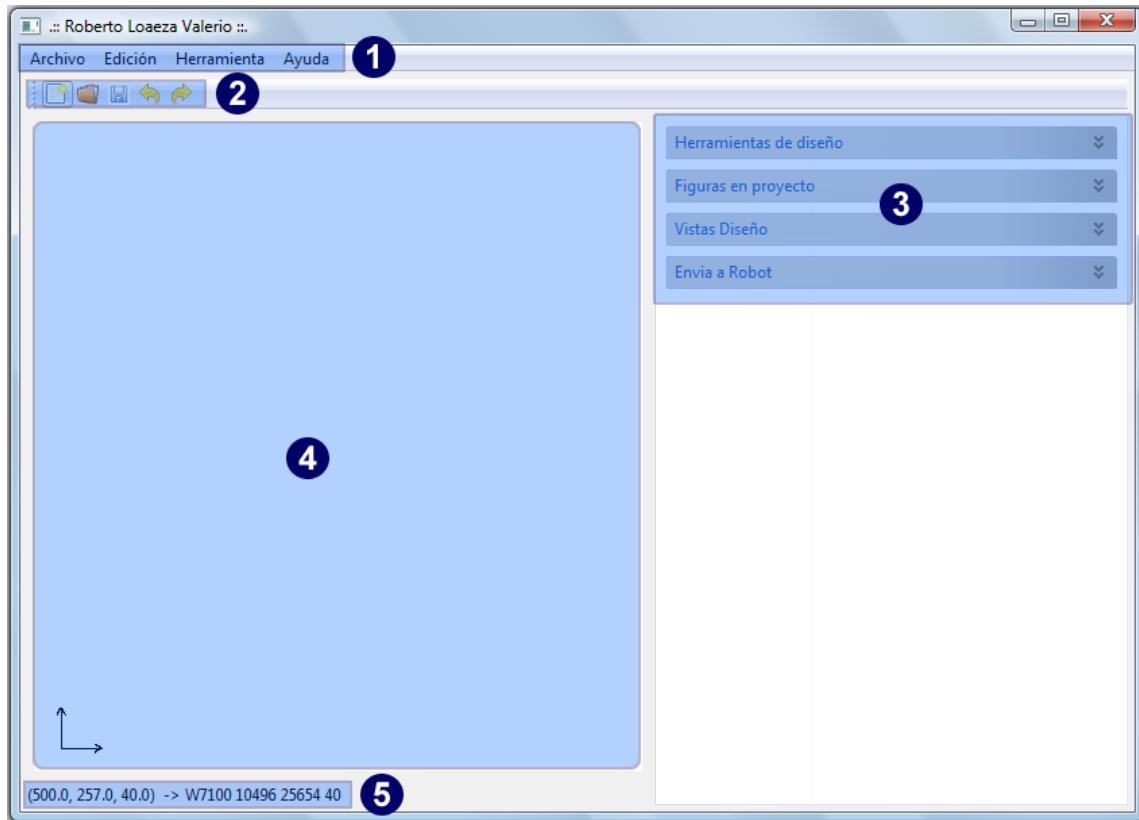


Figura 4.1: Pantalla principal

Área de menús

Esta área contiene las acciones que debe presentar todo sistema computacional. En la figura 4.2(a) se muestra las acciones básicas sobre manipulación de archivos:

- Nuevo. Se utiliza para crear un nuevo diseño o un diseño sin nombre. Todos los archivos recibirán la extensión xml. Si se escogiera un nombre existente, se mostrará un mensaje indicando la existencia de un diseño con ese nombre y dando la opción de sustituirlo.
- Abrir. Con esta orden se cargan los diseños existentes, creados con anterioridad, para su visualización, modificación o implementación.

- Importar. Permite transformar archivos en formato PGM en un diseño entendible por el programa.
- Cerrar. Este comando permite cerrar el diseño actual sin almacenar los cambios efectuados al diseño.
- Guardar. Para almacenar los diseños realizados durante una sesión de trabajo. Si el proyecto tiene nombre solo se actualizará, en caso contrario se solicitará un nombre para el diseño.
- Guardar Como. Permite almacenar el diseño actual con un nombre diferente al que tiene, si aún no tiene nombre se solicitará un nombre para el diseño.
- Salir. Este comando sale del programa sin guardar los cambios efectuados en el diseño actual, sólo preguntando si en realidad desea salir.

En la figura 4.2(b) se muestra las acciones disponibles de edición:

- Deshacer. Permite deshacer la ultima acción realizada.
- Rehacer. Permite rehacer una accion deshecha.
- Eliminar primitiva. Este comando permite eliminar una primitiva seleccionada del diseño.
- Duplicar primitiva. Este comando duplica una primitiva seleccionada.
- Preferencias del sistema. Permite seleccionar opciones referentes al sistema, como nombre del puerto y estilo de área de trabajo.

En la figura 4.2(c) se muestra las acciones pertinentes que permiten que el robot realice el diseño activo. Contiene las opciones de:

- Activar robot. Con esta orden se realizan dos operaciones: Se comprueba que el puerto serial mediante el cual se realizará la conexión sea el indicado y se realiza un proceso para bloquear el uso del puerto y así evitar fallas en la comunicación con el robot.

- Iniciar proceso robot. Este comando inicia el proceso de transformación del diseño a comandos entendibles por el robot SCARA y los envía al mismo.

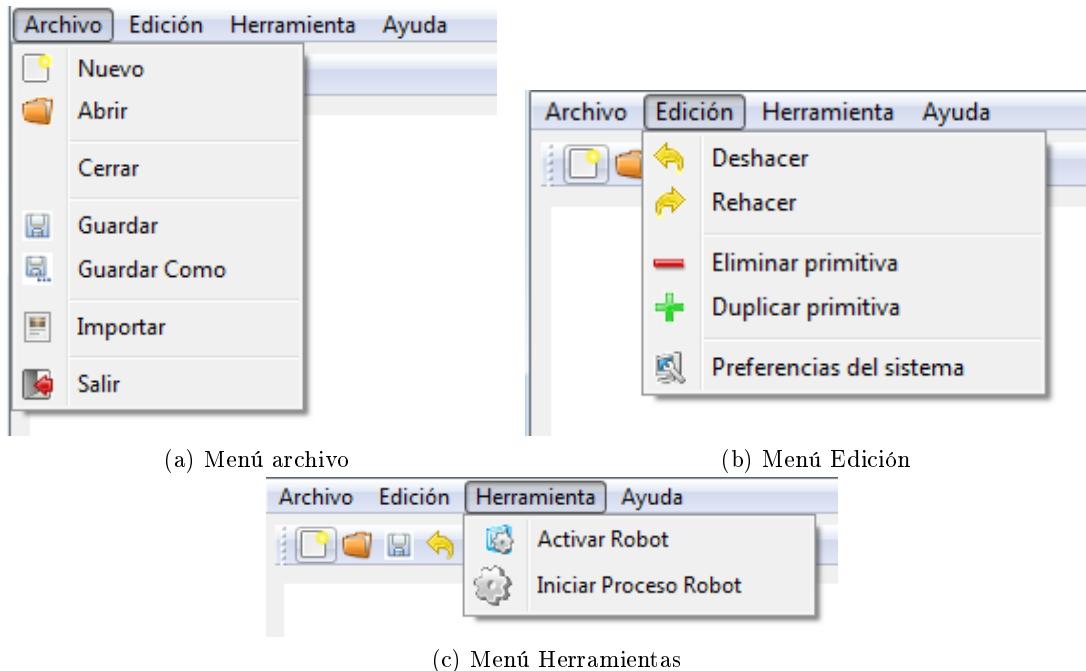


Figura 4.2: Área de menús

Barra de herramientas de acceso rápido

Contiene acceso rápido a tareas comunes como crear un nuevo diseño, abrir un diseño, así como la acciones de deshacer y rehacer. Esta sección puede apreciarse en la figura 4.3 en la parte superior izquierda, justo debajo del área de menús.

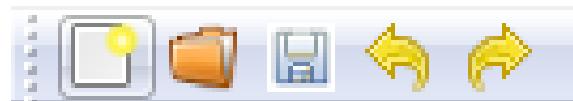


Figura 4.3: Barra de herramientas de acceso rápido

Barra de herramientas de expansión

Estas herramientas están agrupadas en diferentes contenedores: “Herramientas de diseño”, “Figuras en proyecto”, “Vistas” y “Envia robot” para una mejor funcionalidad cumpliendo así con los principios de un buen diseño de la interfaz gráfica: consistencia, eficiencia, disponibilidad y simplicidad. A continuación se describe brevemente cada una de éstas.

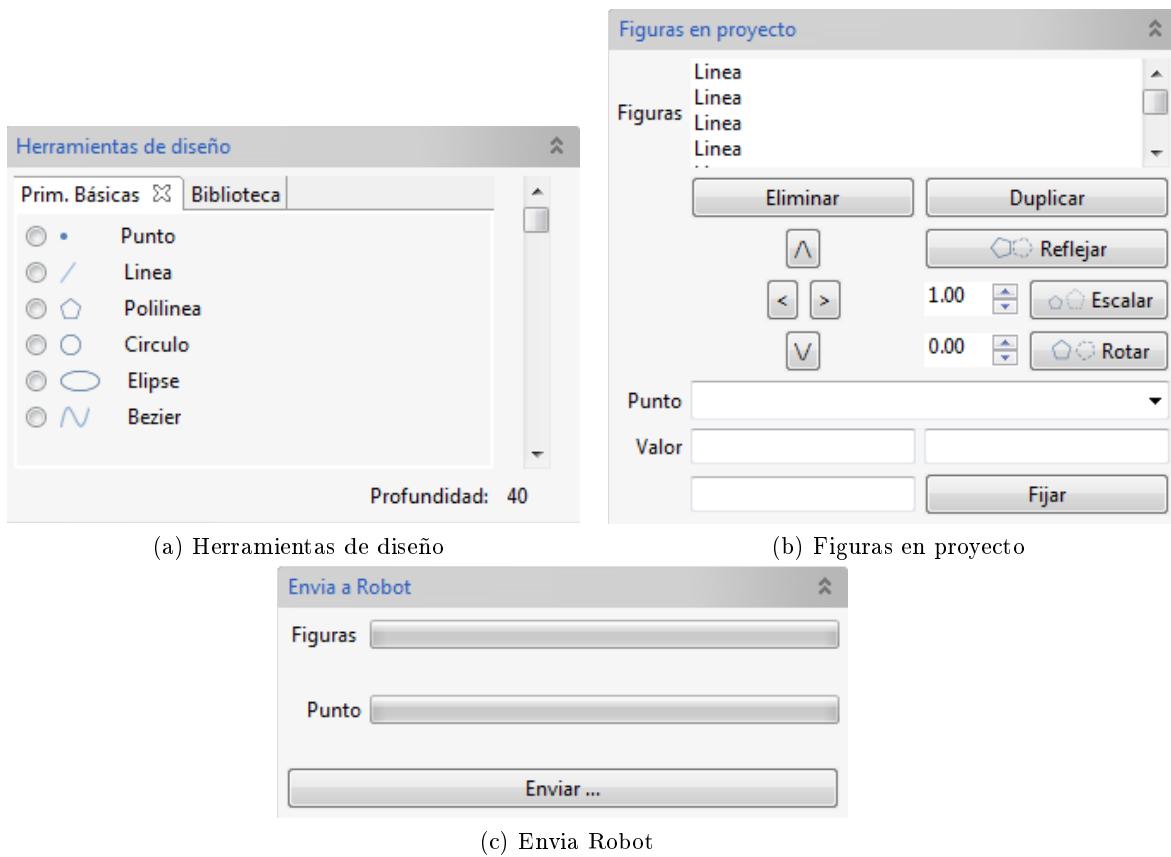


Figura 4.4: Barra de herramientas

- **Herramientas de diseño.** Este contenedor tiene las herramientas de diseño predefinidas así como los elementos compuestos generados por el usuario como se puede apreciar en la figura 4.3(a).
 - Las herramientas predefinidas son: Punto, Línea, Polilínea, Círculo, Elipse y curvas de Bezier.

- Elementos compuestos. Aquí aparecerán todos aquellos diseños almacenados en un directorio denominado “galeria” como patrones para ser insertados como si se tratara de una herramienta predefinida.
- Figuras en proyecto. Aquí se mantiene una lista de figuras que se encuentran en el diseño actual. A cada figura existente se pueden aplicar las transformaciones mencionadas en la sección 3.3 rotación, escalado, traslación así como operaciones de duplicación y eliminación de primitivas existentes utilizando los controles que se aprecian en la figura 4.3(b).
- Vistas. Permiten cambiar la vista del diseño, como se muestra en la figura 4.5.
 - Frente. Permite la vista del plano x-y.
 - Arriba. Permite la vista del plano x-z.
 - Lateral. Permite la vista del plano y-z.
 - Proyección isométrica. Muestra una proyección en perspectiva del diseño activo.
- Envía robot. Permite el envío del diseño activo al robot SCARA.
 - Inicia / Detiene el envío hacia el robot SCARA, además de que muestra un barra de progreso como se muestra en la figura 4.3(c).

Área de trabajo

El área de trabajo es la parte central donde se pueden realizar los diferentes trazos haciendo uso de la herramienta de diseño, puede apreciarse en la figura 4.6.

En esta área de trabajo es donde se visualizarán todos los trazos realizados.

Descripción del punto actual

La descripción del punto actual se muestra en la figura 4.1 en la parte inferior izquierda, esta nos presenta información sobre la posición actual donde se encuentra el cursor dentro

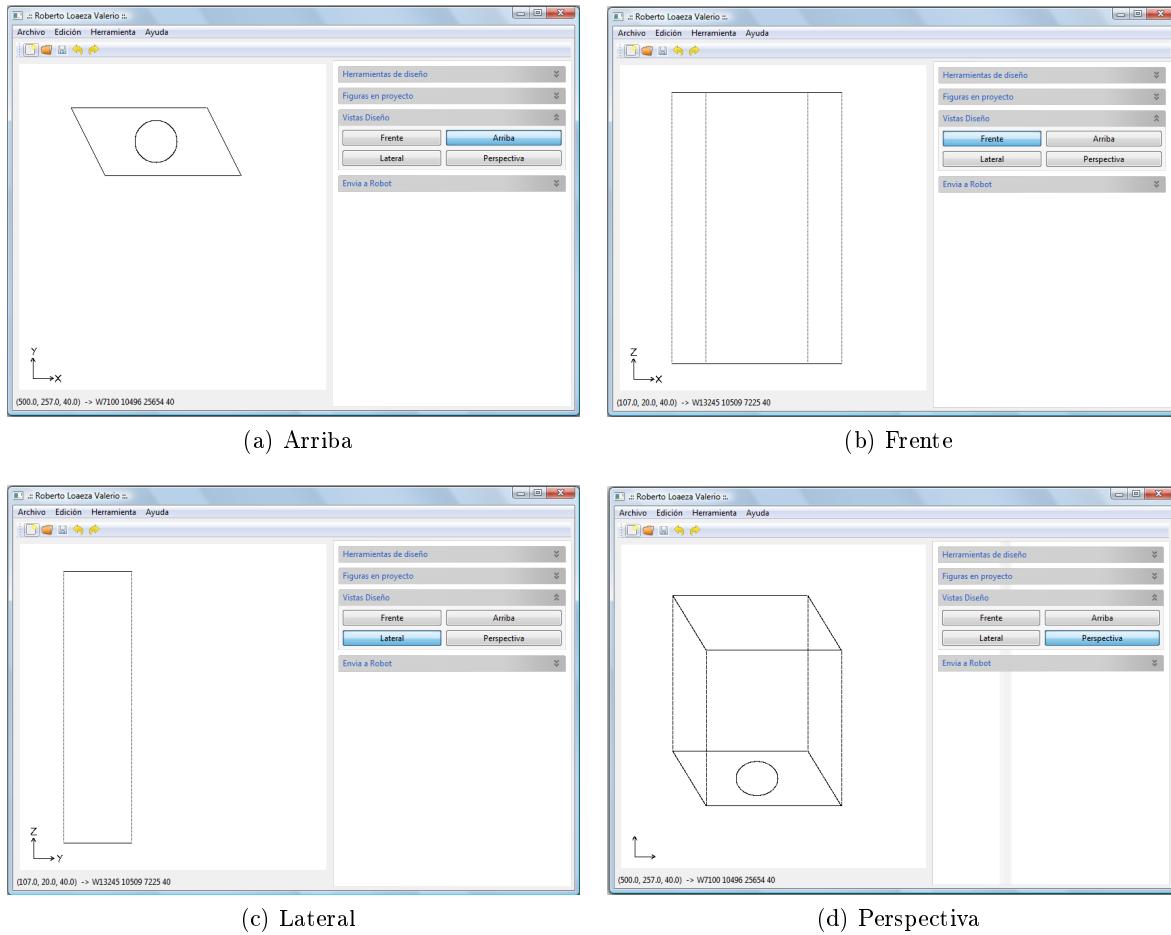


Figura 4.5: Vistas disponibles

del área de trabajo del robot, además muestra el comando para mover el robot SCARA a la posición relativa.

4.3.2. Versiones realizadas

Las versiones de software desarrolladas se listan a continuación con una pequeña descripción de los componentes usados y su desempeño.

Versión 1

En la versión 1 se implementó usando Java y OpenGL (usando jogl para realizar el enlace con OpenGL [Davison07]) pero lamentablemente fue muy inestable debido a que si la

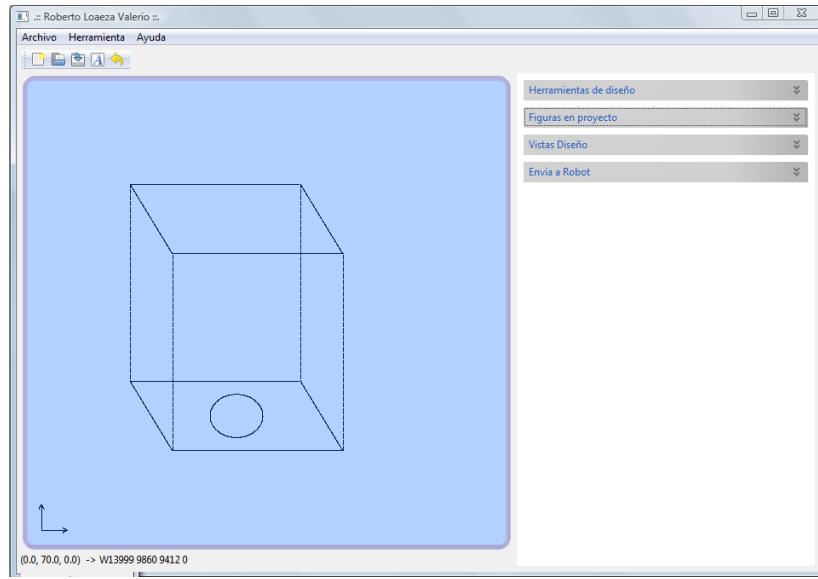


Figura 4.6: Área de trabajo

computadora no tiene una tarjeta de video dedicada el procesador tiene que realizar la emulación y esto conlleva a una operación lenta de la aplicación. Siendo uno de los puntos más relevantes la velocidad de la aplicación, en específico para la comunicación con el robot mediante el puerto serial. Fue desechada esta implementación.

Versión 2

En ésta versión se implementarón dos hilos para intentar eliminar el problema del envío retrasado por la emulación de OpenGL, sin embargo no siempre se lograba enviar al robot con la velocidad y continuidad necesaria. Esta versión también fue desechada.

Versión 3

En la versión 3 se decidió abandonar OpenGL por los motivos mencionados anteriormente y se inicio la implementación con primitivas gráficas de Java (canvas). Se logró eliminar el problema del envío al robot.

Versión 4

En esta última versión se cambiarán los controles de Java.swing(jframe, jbutton, etc) por los correspondientes de la biblioteca SWT (shel, button, etc) debido a que usando la biblioteca SWT la aplicación logra tener una apariencia idéntica a las aplicaciones nativas del sistema operativo.

Capítulo 5

Interfaz con el robot SCARA

La interfaz entre el software modelador y el robot es una parte muy importante, ya que si ésta llega a colapsar o mal-funcionar, se verá reflejado en un trabajo final no satisfactorio.

5.1. Robot SCARA

El robot manipulador con el cual se trabajó se puede apreciar en la figura 5.1. Este robot SCARA de 4 grados de libertad construido en la Facultad de Ingeniería Electrica de la Universidad Michoacana de San Nicolás de Hidalgo. La estructura física fue realizada por una empresa dedicada a la robótica, denominada “Cervantes Co.”, ubicada en Paracho Michoacán. La programación del controlador del robot SCARA fue realizada por Omar Rodríguez Páez [Rodriguez08].

5.1.1. Características

Las características más relevantes acerca del robot son:

- 4 Grados de libertad.
- Altura de 60 cm.
- Alcance máximo de su brazo, 50 cm.

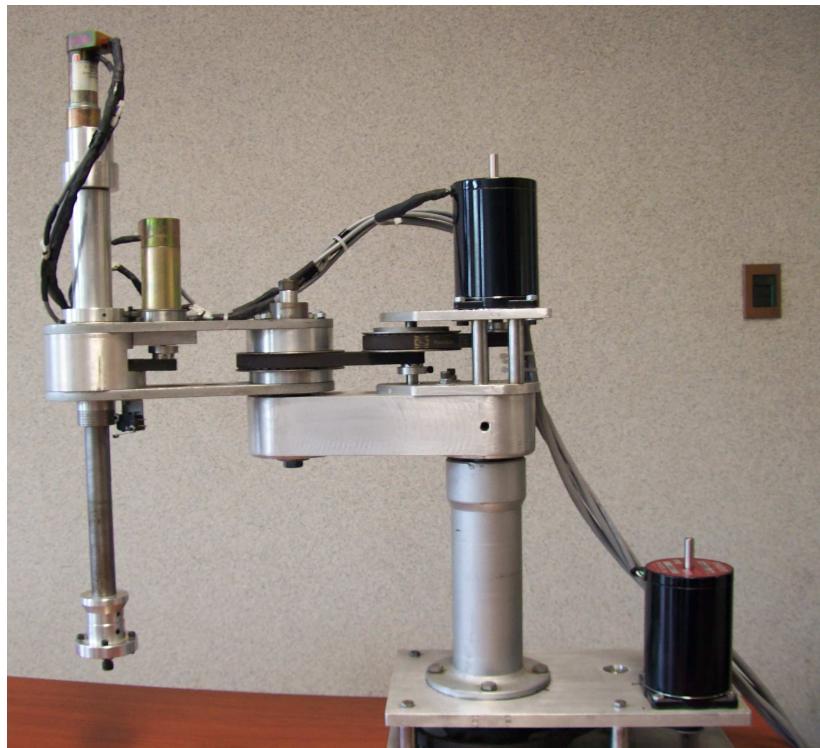


Figura 5.1: Robot SCARA

- Construido en acero y aluminio.
- Peso aproximado de 45 Kg.

5.1.2. Arquitectura

A continuación se listan algunas de las características más relevantes del robot utilizado en el presente trabajo:

Sistema sensorial.

El robot cuenta con sensores:

- Sensores Odométricos. Estos sensores mide la posición o rango de rotación de un eje.
- Sensores de límite o de posición límite de cada articulación. Es un dispositivo muy simple que provee una gran fiabilidad para conocer con certeza las posiciones de inicio y final de cada uno de los ejes del robot SCARA.

Elementos terminales.

El robot cuenta con dos motores de pasos, uno para el hombro y otro para el codo, éstos permiten un movimiento sobre los ejes X y Y; y dos motores de CD con escobillas para la llamada rotación de muñeca y para definir la profundidad.

Sistema de control.

El robot utiliza un sólo módulo Adapt9S12DP256 con el microcontrolador 9S12DP256C de Motorola [Arts04], el cual se muestra en la figura 5.2.

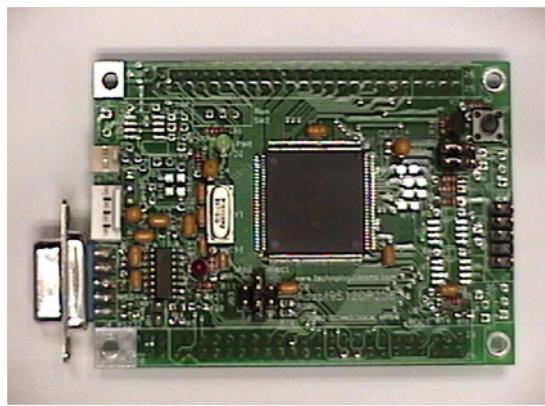


Figura 5.2: Módulo Adapt9S12DP256 del microcontrolador 68hcs12[Rodriguez08]

5.1.3. Limitantes

Possiblemente la limitante más importante viene dada por el rango reducido del área de trabajo, dejando un limitado conjunto de posibles aplicaciones. También cabe mencionar la parte de control ya que esta se realiza en forma secuencial y no paralela, es decir no se permite el movimiento simultáneo de varias articulaciones.

5.2. Conversión de modelado a acciones SCARA

Debido a que en el modelo sólo contamos con puntos definidos dentro del sistema de coordenadas cartesianas de tres dimensiones es necesario calcular el conjunto de ángulos de las

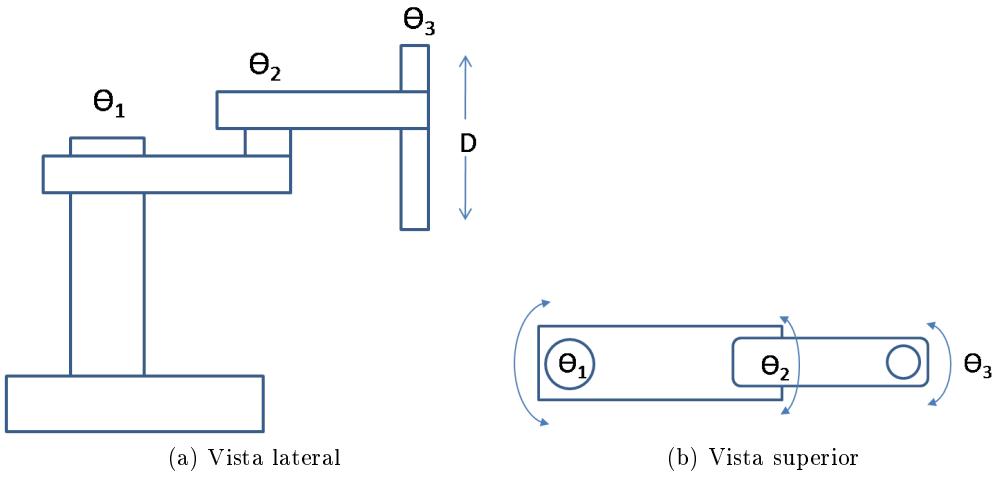


Figura 5.3: Configuración de un robot SCARA

articulaciones del robot para lograr el resultado deseado.

Sea $(\Theta_1, \Theta_2, \Theta_3, D)$ una configuración del robot donde Θ_1 se refiere a la rotación de la articulación del brazo, Θ_2 se refiere a la rotación del codo, Θ_3 se refiere a la rotación de la muñeca y D la profundidad de la herramienta, como se ilustra en la figura 5.3.

Sea (x, y, z, Θ) la posición y orientación final de la herramienta.

En robótica se conoce como el problema de cinemática directa al problema de determinar la posición y orientación de la herramienta (x, y, z, Θ) si conocemos la configuración de las articulaciones $(\Theta_1, \Theta_2, \Theta_3, D)$. Si lo que deseamos es encontrar la configuración de las articulaciones $(\Theta_1, \Theta_2, \Theta_3, D)$ a partir de la especificación de una posición y orientación deseada de la herramienta, el problema se conoce como cinemática inversa.

Para nuestros fines, lo que tenemos son los puntos deseados y necesitamos la configuración del robot para alcanzar dichos puntos, es decir, requerimos solucionar el problema de cinemática inversa.

5.2.1. Cinemática inversa

En la cinemática inversa se conoce la posición y la orientación del elemento terminal, referido a la base y se desea determinar los ángulos articulares para alcanzar dicha posición.

Existen varias soluciones a este problema:

- Soluciones Cerradas(analíticas): Solución algebraica y solución geométrica.
- Soluciones Numéricas Iterativas.

Debido a su naturaleza iterativa, las soluciones numéricas son generalmente mucho más lentas que la solución de forma cerrada. En este trabajo sólo nos enfocaremos en las soluciones analíticas, en particular en la solución geométrica, porque para la configuración sencilla del robot SCARA, presenta una solución muy rápida y sencilla.

Solución Geométrica

Recordando algunas identidades trigonométricas:

$$\frac{\sin(A)}{a} = \frac{\sin(B)}{b} = \frac{\sin(C)}{c} \quad (5.1)$$

$$a^2 = b^2 + c^2 - (2bc) \cos(A) \quad (5.2)$$

,

$$\cos(\Theta) = \cos(-\Theta) \quad (5.3)$$

Primero se sabe que la orientación del último eslabón es la suma de las variables articulares, como se muestra en la figura 5.4.

$$\Theta = \Theta_1 + \Theta_2 + \Theta_3 \quad (5.4)$$

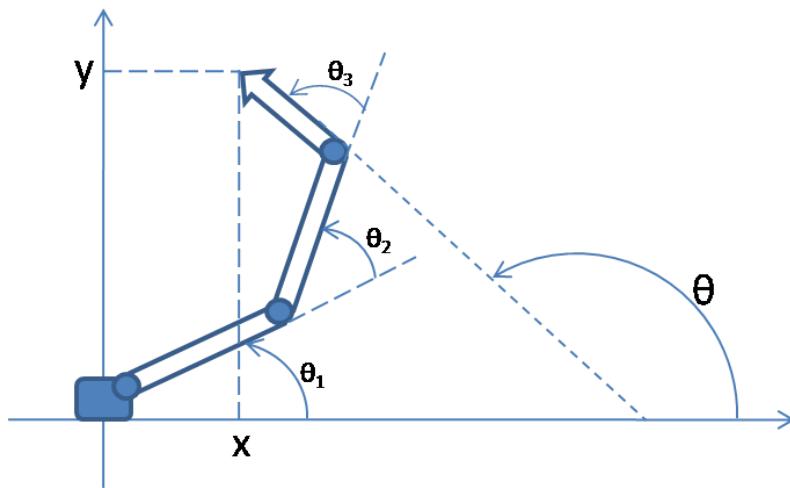
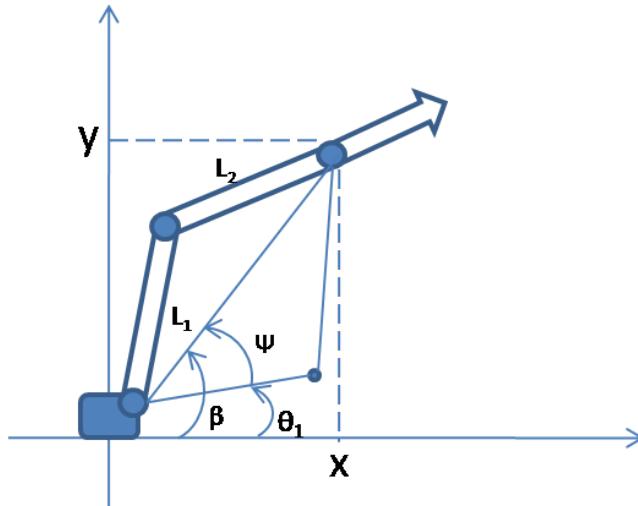


Figura 5.4: Orientación del último eslabón

Se calcula θ_2 aplicando ec. 5.2 (observar figura 5.5):

$$x^2 + y^2 = l_1^2 + l_2^2 - (2l_1l_2) \cos(180 - \Theta_2) \quad (5.5)$$

Figura 5.5: Longitudes L_1 y L_2

debido a que:

$$\cos(180 - \Theta_2) = -\cos(\Theta_2) \quad (5.6)$$

Resulta:

$$\cos(\Theta_2) = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2} \quad (5.7)$$

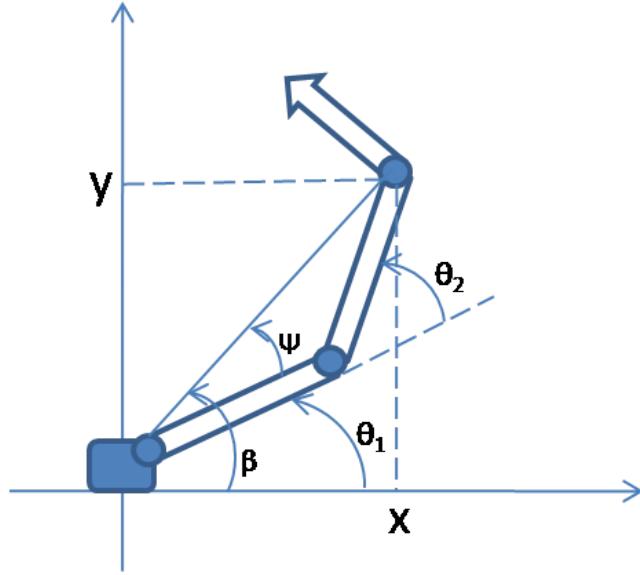


Figura 5.6: Ángulos β , ψ y Θ_1

Se calcula θ_1 :

Si se definen dos ángulos como se muestra en la figura 5.6 se cumple:

$$\Theta_1 = \beta - \psi \quad (5.8)$$

El ángulo β se calcula:

$$\sin(\beta) = \frac{y}{\sqrt{x^2 + y^2}} \quad (5.9)$$

y para el ángulo ψ se usa ec. 5.1 y se tiene:

$$\cos(\psi) = \frac{x^2 + y^2 + l_1^2 - l_2^2}{2l_1\sqrt{x^2 + y^2}} \quad (5.10)$$

Finalmente se calcula θ_3 usando la siguiente ecuación:

$$\Theta_3 = \Theta - \Theta_1 - \Theta_2 \quad (5.11)$$

5.2.2. Implementación de la cinemática inversa.

La implementación de la solución geométrica para la cinemática inversa se muestra en el código 5.1.

Código 5.1 Cinemática inversa (cinematica.Inversa.java)

```

package cinematica;
import primitivas.Punto;
public class Inversa {
    double Res_M[] = new double[]{
        (360.0 / 800.0) * (8.0 / 44.0) * (8.0 / 60.0) * (Math.PI / 180.0),
        (360.0 / 800.0) * (8.0 / 32.0) * (8.0 / 36.0) * (Math.PI / 180.0),
        (360.0 / 2000.0) * (1.0 / 19.7) * (8.0 / 20.0) * (Math.PI / 180.0),
        (1.0 / 2000.0) * (1.0 / 6.3) * (25.4 / 11.0)
    };
    int Def_M[] = new int[]{ 5000, 5350, 42500, 150000 };
    double longs[] = new double[] { 200.0, 200.0 };
    double teta_global = 90.0*(Math.PI/180.0);
    double minx = -200; double miny = 66;
    double div = 2.4;
    public void get_angles(Punto p, int M[]){
        get_angles((p.getX()/div)+minx, (p.getY()/div)+miny, M);
        M[3]=(int)p.getZ();
    }
    public void get_angles(double x, double y, int M[]) {
        double x2 = Math.pow(x, 2);
        double y2 = Math.pow(y, 2);
        double l1 = Math.pow(longs[0], 2);
        double l2 = Math.pow(longs[1], 2);
        double angs[] = new double[3];
        double beta, phi;
        beta = Math.asin( y / ( Math.sqrt ( x2 + y2 ) ) );
        if (x < 0)
            beta = Math.PI - beta;
        phi = Math.acos( ( x2 + y2 + l1 - l2 ) / ( 2 * longs[0] * Math.sqrt(x2 + y2) ) );
        angs[0] = beta - phi;
        angs[1] = Math.acos( ( x2 + y2 - l1 - l2 ) / ( 2 * longs[0] * longs[1] ) );
        angs[2] = (teta_global - angs[0] - angs[1]);
        M[0] = (int) (angs[0] / Res_M[0]) + Def_M[0];
        M[1] = (int) (angs[1] / Res_M[1]) + Def_M[1];
        M[2] = (int) (angs[2] / Res_M[2]) + Def_M[2];
        M[3] = Def_M[3];
    }
}

```

5.3. Conectividad PC-Robot

Debido a que el robot SCARA usa conexión mediante el puerto serial (RS232) primero se realizó una interfaz capaz de comunicar la aplicación desarrollada en lenguaje Java, con el sistema de control del robot SCARA.

Existen varias bibliotecas para la comunicación serial, compatibles con lenguaje Java. Algunas bibliotecas aún están en desarrollo, otras sólo son para un determinado sistema operativo. En la tabla 5.1 se muestran algunas de las bibliotecas más usadas para la conexión mediante puerto serial.

Biblioteca	Multiplataforma	Tipo de licencia
javaconn	No	GPL
rxtx	Si	GPL

Cuadro 5.1: Bibliotecas para comunicación RS232

Debido a que la biblioteca rxtx da soporte a la mayoría de las plataformas (MS-Windows, Linux, Mac OS) se optó por utilizar esta biblioteca además de que es transparente a la hora de programar, es decir no requiere hacerse ningún cambio para que la comunicación se realice mediante puerto serial en las diferentes plataformas.

En el código 5.2 se muestra la implementación de la conexión con el robot SCARA mediante el puerto serial.

5.3.1. Versiones realizadas

Los cambios más significativos de la implementación que llevaron a cambios drásticos se mencionan a continuación con el nombre de versiones.

Versión 1

En la versión 1 se implementó la conexión con el robot SCARA usando un la biblioteca javaconn. Ésta versión operó de manera eficiente en el sistema operativo Linux, sin embargo

Código 5.2 Conexión mediante puerto serial (serial.Comunicacion.java)

```

package serial;
/* ... */
public class Comunicacion {
    /* ... */
    public void connect (String nombrePuerto) throws Exception {
        CommPortIdentifier portIdentifier = CommPortIdentifier.getPortIdentifier(nombrePuerto);
        if ( portIdentifier.isCurrentlyOwned() ) {
            System.out.println("Error: El puerto está en uso");
        }
        else {
            CommPort commPort = portIdentifier.open(this.getClass().getName(),2000);
            if ( commPort instanceof SerialPort ) {
                SerialPort serialPort = (SerialPort) commPort;
                serialPort.setSerialPortParams(velPto,SerialPort.DATABITS_8,
                    SerialPort.STOPBITS_1,SerialPort.PARITY_NONE);
                in = serialPort.getInputStream();
                out = serialPort.getOutputStream();
            }
            else {
                System.out.println("Error: Solo soporta puerto serial.");
            }
        }
    }
    public String leer() throws IOException {
        byte[] buffer = new byte[2];
        in.read(buffer,0, 1);
        String s="";
        do {
            in.read(buffer,0, 2);
        } while(new String(buffer).equalsIgnoreCase("W")==false);
        return "W";
    }
    public String escribe(String str) throws IOException {
        for(int i=0; i<str.length(); i++) {
            out.write((int)str.charAt(i));
        }
        return leer();
    }
}

```

en MS-Windows no operó del todo bien, en ocasiones el sistema se volvía inestable y la conexión con el puerto serial se perdía. Esta versión fue deshechada.

Versión 2

En ésta versión se cambió la biblioteca de conexión de javaconn a rxtx dando resultados muy buenos, es decir ya no se detectaron problemas en la conexión en ambos sistemas operativos (Linux y MS Windows). También la biblioteca rxtx permitió bloquear el puerto serial dando una mayor seguridad durante el proceso de envío de comandos al robot.

Capítulo 6

Pruebas

En este capítulo se describe brevemente los requerimientos del sistema así como la forma de instalar las aplicaciones y bibliotecas en las diferentes plataformas. También se muestran los resultados de la realización de pruebas usando el robot SCARA y el programa desarrollado en esta tesis.

6.1. Introducción

Para poder ejecutar la aplicación desarrollada es necesario tener un entorno adecuado para su ejecución.

Requisitos mínimos del sistema

El equipo necesario para ejecutar la aplicación desarrollada será el mostrado en la tabla 6.1.

Instalando Java

Descargar e instalar Java Runtime Environment 1.6 o superior de la página oficial de Sun Microsystem: <http://java.sun.com/javase/downloads/index.jsp>. Si el sistema operativo an-

Concepto	Valor
Procesador	Velocidad de 1 Ghz o superior
Memoria	256 Mb como mínimo
Almacenamiento	20 Mb
Sistema operativo	MS-Windows XP o superior
	Linux kernel 2.6 o superior
	MacOS X Tiger o superior
Java	JRE 1.6 o superior
Bibliotecas	RXTX 2.2 o superior
	SWT 3.4 o superior

Cuadro 6.1: Requisitos mínimos del sistema

anfitrío es MS Windows solo es necesario ejecutar el archivo ejecutable descargado.

Instalando bibliotecas

Para instalar la biblioteca necesaria para la comunicación con el robot es necesario descargar la versión más actual del sitio oficial <http://users.frii.com/jarvi/rxtx/> y descomprimir los archivos correspondientes al sistema operativo anfitrío y mover los archivos a una carpeta común con la aplicación desarrollada.

Finalmente la biblioteca SWT necesaria para la interfaz gráfica es necesario descargar la versión más actual del sitio oficial <http://eclipse.org/swt/> y descomprimir los archivos correspondientes al sistema operativo anfitrío y mover los archivos a una carpeta común con la aplicación desarrollada.

Instalando la aplicación desarrollada

Para ejecutar la aplicación desarrollada en esta tesis hay dos formas:

- Usar algun cliente svn, svn es un sistema de control de versiones usado para mantener el código fuente actual así como sus versiones previas[Collins-Sussman08], y descargar la versión en desarrollo y compilarla desde el código fuente. Para esto se deben realizar los siguientes comandos:

```
[user@site]$ svn checkout http://modelando-madera.googlecode.com/svn/trunk/ modelando-madera-read-only
```

```
[user@site]$ javac ide.Tesis.java
```

```
[user@site]$ java ide.Tesis
```

- Descargar la última versión liberada por el autor desde la página oficial <http://modelando-madera.googlecode.com> y ejecutarla dando doble click sobre el archivo o bien ejecutando el comando:

```
[user@site]$ java -jar modelador.jar
```

6.2. Funcionalidad multiplataforma del software 3D

Como se puede apreciar en la figura 6.1 los controles usados para la manipulación de la aplicación existen en las plataformas más usadas comúnmente.

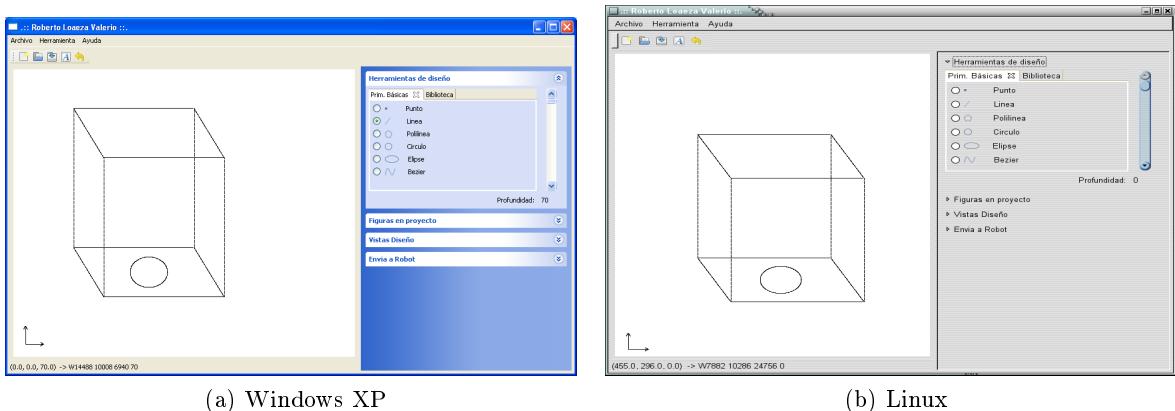


Figura 6.1: Aplicación ejecutándose en diferentes plataformas

En la tabla 6.2 puede observarse que la aplicación funcionó perfectamente en los sistemas operativos más comunes, además que cumple con la integración con el tema de pantalla que el sistema operativo define.

Sistema Operativo (s.o.)	Carga del programa	Redibujo (11000 figuras)	Integración con el s.o.
Linux (openSuSE 11.0)	1.20 seg	7.0 seg	Sí
Mac OS (tiger)	2 seg	7.20 seg	Sí
MS-Windows (Vista)	1.20 seg	7.50 seg	Sí

Cuadro 6.2: Tiempos de carga del programa

6.3. Conectividad multiplataforma entre software y el robot

Las pruebas de conectividad fueron realizadas con éxito gracias a la biblioteca rxtx que soporta la mayoría de los sistemas operativos más usados actualmente. En la tabla 6.3 pueden apreciarse los únicos cambios realizados en cada uno de los sistemas operativos, esto debido a que el puerto cambia de nombre en cada uno de los sistemas operativos.

Sistema Operativo (s.o.)	Nombre del puerto	Permite modo exclusivo de puerto
Linux (openSuSE 11.0)	/dev/ttyUSB	Si
Mac OS (tiger)	/dev/tty.usbserial	Si
MS-Windows (Vista)	COM5	Si

Cuadro 6.3: Puerto serial: nombre y modo de uso

Debido a que los sistemas operativos mostrados en la tabla 6.3 soportan el modo exclusivo de un puerto en específico no se detectó ninguna interferencia por algún otro programa que intentara utilizar el mismo puerto dando como resultado una plena seguridad de que una vez activado el puerto la comunicación con el robot SCARA se realizará con plena seguridad.

6.4. Pruebas en campo

Los tiempos promedios realizados en campo utilizando madera suave se pueden ver en la tabla 6.4.

Objeto	Numero de puntos	Tiempo de procesamiento en robot
Punto	1	1 segundo
Línea (10 cm)	500	5 segundos
Círculo (radio: 2 cm)	450	7 segundos
Elipse (2 y 1 cm para sus radios)	400	6 segundos
Bezier (aprox 10 cm)	700	8 segundos
Imagen a relieve (15x15 cm)	11000	21.5 minutos

Cuadro 6.4: Pruebas y tiempos

En la figura 6.2 puede observarse la implementación de nueve primitivas realizadas en madera en aproximadamente 1 minuto. Además en la misma figura se realizó esta tarea 10 veces

para apreciar la precisión con la que trabaja el robot SCARA, como se puede observar parece que solo se hubiera realizado una vez.



Figura 6.2: Ejemplo de 9 primitivas

En la figura 6.3 se muestra un ejemplo de una imagen en formato PGM y su respectiva implementación en madera se puede observar en la figura.



Figura 6.3: Ejemplo PGM

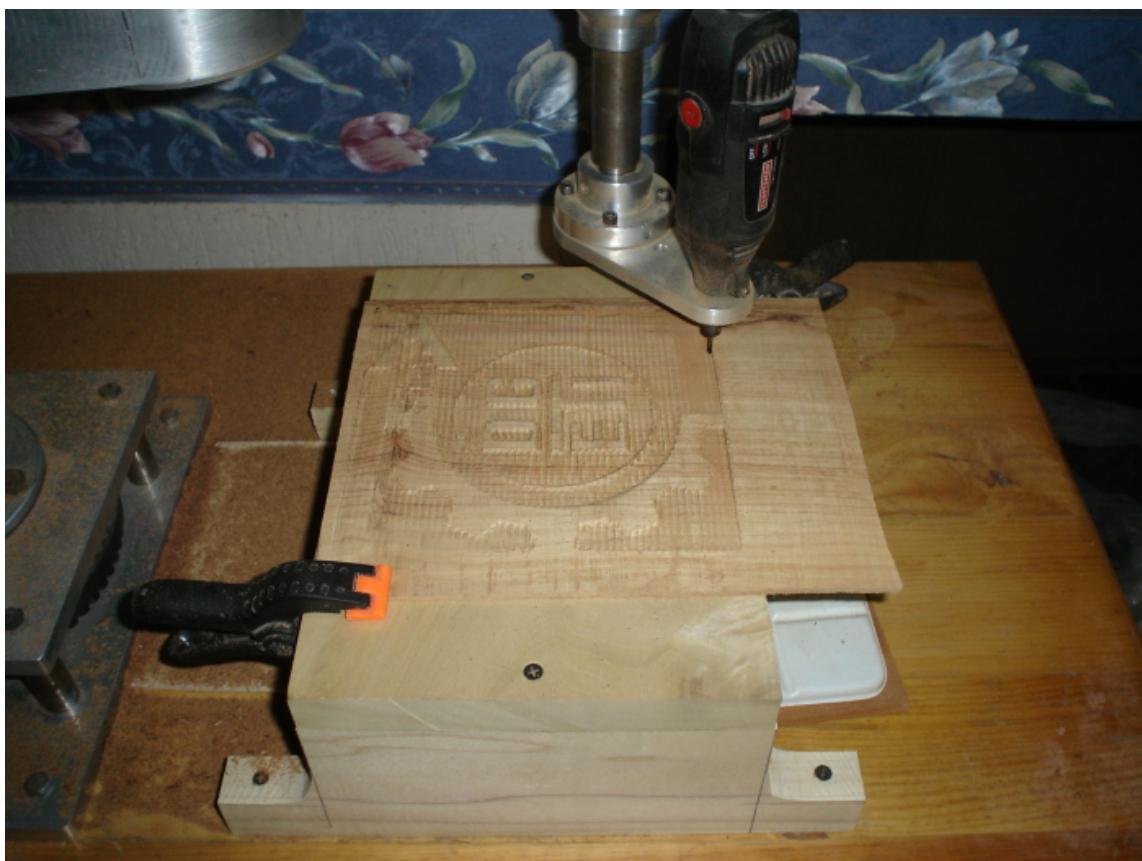


Figura 6.4: Ejemplo PGM realizado en madera

Capítulo 7

Conclusiones

La aplicación desarrollada termina con una gran cantidad de esfuerzos realizados para tener un primer robot industrial de bajo costo realizado completamente con tecnología y conocimiento propio. Este desarrollo consta de:

- Desarrollo mecánico de un robot SCARA, elaborado por el Sr. Rafael Cardiel Cervantes.
- Desarrollo electrónico/lógico de un robot SCARA, Trabajo realizado por Omar Ríos y Omar Rodríguez en sus tesis de licenciatura de la Facultad de Ingeniería Eléctrica de la Universidad Michoacana de San Nicolás de Hidalgo [Ríos07, Rodríguez08].
- Desarrollo de una aplicación para diseñar y realizar tareas utilizando un robot SCARA.

Siendo en la parte final donde se ubica la aplicación desarrollada en la presente tesis.

7.1. Conclusiones

- Se eligió Java como lenguaje de programación debido a su gran virtud de poder ser ejecutado en la mayoría de las plataformas sin la necesidad de alterar el código fuente o recompilar el mismo.

- Se diseñó una aplicación modular dando la posibilidad de que en un futuro a la aplicación se pueda agregar nuevos módulos o bien mejorar los existentes sin la necesidad de modificar el código completo.
- Se desarrollaron diferentes clases para las primitivas Punto, Línea, Círculo, Elipse, Bezier para un manejo más eficiente a la hora de su programación.
- Se destaca el uso de la biblioteca SWT que permite que la aplicación desarrollada luzca como una aplicación nativa del sistema operativo anfitrión.
- Se cumplieron los objetivos fundamentales que son la creación de una aplicación de software libre para el robot de bajo costo desarrollado en la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica de la Universidad Michoacana de San Nicolás de Hidalgo.
- Se culminó el proceso de la generación de un robot industrial de bajo costo realizado con tecnología y conocimiento nacional.
- Es posible usar el robot en la industria de la madera, en procesos de realizar detalles de algunas partes de las guitarras.

7.2. Sugerencias para trabajos futuros

Las limitaciones mecánicas y electrónicas del robot para el cual fue desarrollada la aplicación de este trabajo mostró puntos interesantes en los cuales se tiene que desarrollar nuevas soluciones. A continuación se mencionan algunos puntos en los que se podría mejorar o desarrollar.

7.2.1. Aplicación desarrollada

1. Finalizar el importador/exportador de código G, el estándar del software CNC para robots, para la aplicación. Actualmente su desarrollo se encuentra en fase de prueba y no fue incluido en esta tesis por no ser parte de la propuesta inicial.

2. Revisar los códigos fuentes disponibles del software EMC2, software desarrollado por la comunidad de software libre para robots CNC y generar un módulo de compatibilidad con la aplicación desarrollada en esta tesis.

7.2.2. Electrónica del robot SCARA

1. Acondicionar la conexión actual del robot SCARA de serial (rs232) a paralelo, ya que ésta última es utilizada por la mayoría de los programas actuales de CNC.
2. Reprogramar el controlador del robot SCARA para realizar movimientos simultáneos, debido a que actualmente los realiza en secuencia y esto ocasiona movimientos no deseados.

Glosario

CISC Complex-Intruction-Set Computing.

CRM Customer Relationship Management.

ERP Enterprise Resource Planning.

ISO International Organization for Standardization.

JIRA Japan Industrial Robot Association.

OpenGL Open Graphics Library.

PGM Portable GrayMap.

RISC Reduced-Intruction-Set Computing.

SCARA Selective Compliant Articulated Robot Arm.

SVN Subversion

XML Extensible Markup Language.

Bibliografía

- [WIKI-ROBOT] Wikipedia, Industrial Robot, 2008,
[<http://en.wikipedia.org/wiki/Industrial_robot>](http://en.wikipedia.org/wiki/Industrial_robot).
- [Kurfess05] Kurfess, T. Robotics and Automation Handbook. Ed. CRC PRESS, 2005.
- [Gibilisco03] Gibilisco, S. Concise Encyclopedia of Robotics, Ed. McGraw-Hill, 2003
- [Davison07] Davison, A. Pro Java 6 3D Game Development Java 3D, JOGL, JInput and JOAL APIs. Ed. Apress, 2007
- [Ollero] Ollero, A. Robótica: Manipuladores y robots móviles, Marcombo Editorial.
- [Galitz07] Galitz, W. The Essential Guide to User Interface Design, Ed. Wiley, 2007.
- [Rodriguez08] Rodriguez, O. Diseño y Construcción de un Robot SCARA utilizando motores de CD de pasos y con escobillas, Tesis de Licenciatura, UMSNH, 2008
- [Estrada06] Estrada, C. Diseño de un sistema de control de seguimiento de trayectorias para un robot móvil. Tesis de Maestría. UMSNH, 2006.
- [Sanchez07] Sánchez, O. Cinemática de los manipuladores, Universidad Huelva, Huelva España. 2007

- [Sanchez06] Sanchéz, O. Robot, Universidad Huelva, Huelva España. 2006
- [IRA08] Instituto de Robótica de América, 2008
- [Kurfess05] Kurfess, T .Robotics and automation handbook. Ed. CRC Press, 2005
- [Delrieux00] Delrieux, C. Introducción a la Computación Gráfica. Dpto Ingeniería Eléctrica, Universidad nacional del sur. 2000.
- [Jones98] Jones, J. L., Flynn, A. M., y Seiger, B. A. Mobile Robots. Cambridge University Press, 1998.
- [mrs2008] Microsoft Robotics Studio, 2008, <<http://www.msdn.microsoft.com/en-us/robotics/default.aspx>>.
- [Kuka08] Kuka robotics, 2008, <<http://www.kuka-robotics.com>>.
- [PROBOTIX09] CNC Stepper Motor Driver System & CNC Router Kits, 2009, <<http://www.probotix.com>>
- [Concha07] Concha, A. Diseño y construcción de un robot móvil, Tesis de licenciatura, UMSNH, 2007.
- [Ríos07] Ríos, O. Diseño e implementación del control de un brazo manipulador scara. Tesis de licenciatura. UMSNH, 2007.
- [ISO8373:94] ISO Standard 8373, Manipulating Industrial Robots. 1994
- [Carlbom78] Carlbom, I., Paciorek, J. Planar Geometric Projections and Viewing Transformations, ACM Computing Surveys, 1978.
- [Bray08] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E. Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation, 2008 <<http://www.w3.org/TR/2008/REC-xml-20081126/>>.
- [Poskanzer03] Poskanzer, J., PGM Format Specification, 2003
<<http://netpbm.sourceforge.net/doc/pgm.html>>.

[Collins-Sussman08] Collins-Sussman, B., Fitzpatrick, B., Pilato, C., Version Control with Subversion, Publicado: TBA, 2008.