CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

Departamento de Ingeniería Eléctrica
Sección de Computación

Sistema de Visión Artificial para el Laboratorio de Robótica Virtual

Tesis que presenta:	**Araujo Díaz David**
Optando por el grado de:	**Maestro en Ciencias**
Especialidad:	**Ingeniería Eléctrica**
Opción:	**Computación**

Asesor:

Dr. Jorge Buenabad Chávez
Sección de Computación

México, D.F. a 20 de marzo del año 2002

Resumen

La invención de todo tipo de máquinas ha tenido el propósito de ayudar al hombre a controlar su medio ambiente. La complejidad de su diseño y construcción ha aumentado considerablemente a través del tiempo debido a los avances tecnológicos en diferentes áreas. Los robots hoy en día pueden realizar tareas que antes solo hacían los humanos, con algunas ventajas, como que no se cansan y por lo tanto cometen menos errores. Una de las maneras de que los robots realicen tareas aún más complejas es dotarlos de sistemas de visión, ó análisis automático de imágenes, que les permita conocer su ambiente de trabajo para poder manipular los objetos que tengan a su alcance. Además un sistema de visión para robots, junto con la infraestructura adecuada para utilizarlo en forma remota, permitirá a los interesados realizar prácticas de manipulación con un robot. Esto es particularmente benéfico dado que un robot y su infraestructura solo se encuentran disponibles en unas pocas instituciones en nuestro país.

Esta tesis presenta el diseño de un sistema de visión capaz de actualizar el espacio de trabajo de un robot industrial, permitiendo al Robot conocer la cantidad de objetos presentes en su mesa de trabajo y otras características importantes como su identidad, posición, orientación y tamaño. Los objetos pueden ser mostrados a un usuario (local ó remoto) mediante el empleo de modelos en tres dimensiones realizados con *VRML* (*Virtual Reality Modeling Language*), con el cual se obtuvieron códigos bastante reducidos que viajan a través de Internet de forma rápida. El sistema se puede integrar con un robot industrial para manipular los objetos encontrados de forma virtual o real, y de forma local o remota. Se realizaron cuatro aplicaciones para nuestro sistema de visión: el reconocimiento de figuras geométricas, el reconocimiento de dígitos, el análisis de movimiento y la estimación de altura de los objetos presentes en el espacio de trabajo de un robot industrial.

A la memoria de mi abuelita

Raquel Rodríguez Baños

A mis padres

Magdalena Josefa y Mario Israel

A mis sobrinos

Mario Esteban, Erika Jazmín y Ángel David

A mis hermanas y hermano

Norma Edith, Yamily e Israel

A mis padrinos

Hilda y Eloy

A mis cuñados

Martha Marisela y Mario

A todos mis familiares
por su apoyo y comprensión

A mi asesor:

Dr. Jorge Buenabad Chávez

A los profesores de la Sección de Computación:

Dr. Adriano de Luca Pennacchia

Dr. Arturo Díaz Pérez

Dr. Guillermo Benito Morales Luna

Dr. Jesús Vázquez Gómez

Dr. Sergio Víctor Chapa Vergara

A mis compañeros de la Sección de Computación:

Libertad Rivera Larqué

Sergina Ascelli Shaya Zamudio Vissuet

Armando Flores Ibarra

Carlos Galindo Hernández

Gabriel Ruiz Hernández

Giner Alor Hernández

Marcelino Castañeda Sánchez

Marco Antonio Ortega García

Mizael Sánchez Santiago

Ulises Zaldivar Colado

Contenido

	Página
Introducción	**I**
Resumen	IV
1. Robótica	**1**
1.1 Definición de robot	1
1.2 Arquitectura de robots	2
1.2.1 Clasificación de robots	3
1.2.2 Tecnología de funcionamiento	3
1.2.3 Geometría del espacio de trabajo	4
1.2.4 Control de movimientos	7
1.3 Sensores y percepción	7
1.4 Programación, planificación y aprendizaje	8
1.5 Integración de robots	9
1.6 Teleoperación de robots y robótica virtual	9
1.7 Mercado mundial de robots	10
Resumen	12
2. Ambientes Virtuales y VRML	**13**
2.1 Ambientes virtuales	13
2.2 Construcción de ambientes virtuales con VRML (*Virtual Reality Modeling Language*)	15
2.2.1 Un programa en VRML	16
2.3 Transformaciones en VRML	18
2.4 Puntos de vista con VRML	20
2.5 Objetos, líneas y puntos tridimensionales en VRML	21
2.6 Texto en VRML	24
2.7 Integración VRML y Java	24
Resumen	28
3. Visión Artificial	**29**
3.1 Visión artificial	29
3.2 Adquisición de la imagen	30
3.3 Preprocesamiento de una imagen	35
3.4 Extracción de características para la segmentación	35
3.4.1 Identificación de contornos en una imagen analógica mediante la derivada	35
3.4.2 Extracción de contornos de una imagen digital mediante derivada	36
3.4.3 Aproximación de Roberts	44
3.4.4 Aproximación de Prewitt	44
3.4.5 Aproximación de Sobel	44
3.4.6 Operador Laplaciano	48
3.4.7 Gradiente estocástico	48
3.5 Segmentación de imágenes	49
3.5.1 Seguimiento de contornos	50

	Página
3.5.2 Unión de bordes por búsqueda heurística	50
3.5.3 Transformada de Hough	51
3.6 Métodos para reconocimiento de patrones	52
3.6.1 Reconocimiento por ángulos	52
3.6.2 Reconocimiento por puntos clave	53
3.6.3 Reconocimiento por cambios de dirección	53
3.6.4 Reconocimiento por factor de forma	54
3.6.5 Reconocimiento por calculo de momentos	55
3.7 Reconocimiento Óptico de Caracteres (OCR)	58
3.8 Calibración de cámara	60
3.9 Áreas de aplicación de sistemas de visión en la industria	63
Resumen	64
4. Laboratorio de Robótica Virtual	**65**
4.1 Laboratorio de Robótica Virtual del CINVESTAV	65
4.1.1 Robot SCARA UNIMATE S-103	66
4.2 Modelación para el robot UNIMATE S-103	68
4.3 Desarrollo del robot virtual	71
4.4 Visión artificial en robótica	74
Resumen	74
5. Diseño e Implementación de un Sistema de Visión	**75**
5.1 Motivación	75
5.2 Aspectos de diseño	76
5.3 Diseño del sistema de visión	76
5.4 Análisis del sistema de visión	85
5.5 Alternativas al acceso remoto	91
Resumen	94
6. Tareas del Sistema de Visión	**95**
6.1 Reconocimiento de figuras geométricas	95
6.2 Reconocimiento de dígitos	98
6.3 Análisis de movimiento	104
6.4 Estimación de altura	109
6.5 Calibración de cámara	116
6.6 Reconstrucción tridimensional con VRML	117
Resumen	122
7. Conclusiones y Perspectivas	**123**
7.1 Conclusiones	123
7.2 Perspectivas	125
Apéndice A: Sistemas de Adquisición de Imágenes	**131**
a.1 Captores de imagen según el tipo de transductor fotoeléctrico	131
a.1.1 Disco de Nipkow	132
a.1.2 Disertor de imagen o tubo de Fransworth	132

	Página
a.1.3 Iconoscopio, tubo de condensación ó de acumulación	133
a.1.4 Orticon de imagen	134
a.1.5 Vidicon	134
a.1.6 Plúmbicon	135
a.1.7 Arreglos de dispositivos de transferencias de carga	135
a.1.8 Arreglos de dispositivos de inyección de cargas	137
a.2 Composición de las cámaras captadoras de imágenes electrónicas	137
a.2.1 Separador óptico	137
a.3 Lentes	138
Resumen	140
Apéndice B: Iluminación Estructurada	**141**
b.1 Sistemas de iluminación estructurada	141
b.2 Triangulación	142
b.3 Implementación de un sensor de características tridimensionales (*Hardware*)	143
b.4 Análisis de imágenes (*Software*)	145
b.5 Resultados	147
Resumen	150
Apéndice C: Transformaciones Geométricas	**151**
c.1 Transformaciones geométricas bidimensionales	151
c.1.1 Traslación	151
c.1.2 Escala	151
c.1.3 Rotación	151
c.1.4 Sesgo	152
c.1.5 Representación de objetos bidimensionales	152
c.1.6 Operaciones con transformaciones bidimensionales	152
c.2 Transformaciones geométricas tridimensionales	153
c.2.1 Traslación	153
c.2.2 Escala	153
c.2.3 Rotación	153
c.2.4 Sesgo	154
c.2.5 Proyecciones paralelas	154
c.2.6 Proyecciones ortográficas	155
c.2.7 Representación de objetos tridimensionales	155
c.2.8 Operaciones con transformaciones tridimensionales	156
Apéndice D: Patrones de Calibración y Figuras Geométricas	**159**
d.1 Patrones de calibración de cámara	159
d.2 Figuras usadas para pruebas de reconocimiento	167
Apéndice E: Contenido del CD-ROM	**173**
e.1 Archivos del CD-ROM	173
e.2 Directorios del CD-ROM y su contenido	174
Referencias	**177**

Lista de Figuras

	Página
1.1 Robot que utiliza reconocimiento de imágenes	**1**
1.2 Tipos de cadenas cinemáticas	2
1.3 Robot rotacional ó R	4
1.4 Robot prismático ó P	4
1.5 Robot prismático – prismático ó PP	4
1.6 Robot rotacional – prismático ó RP	5
1.7 Robot cartesiano, prismático – prismático – prismático ó PPP	5
1.8 Robot cilíndrico, rotacional – prismático – prismático ó RPP	5
1.9 Robot esférico, rotacional – rotacional – prismático ó RRP	6
1.10 Robot SCARA, rotacional – rotacional – prismático ó RRP	6
1.11 Robot antropomórfico, rotacional – rotacional – rotacional ó RRR	7
2.1 Caja simple en *VRML*	**16**
2.2 Ejes de coordenadas en *VRML*	17
2.3 Nodo translation	19
2.4 Nodo rotation	20
2.5 Nodo scale	20
2.6 Definición de los vértices de un triángulo isósceles con *VRML*	22
2.7 Un triángulo isósceles definido como: un sólido, con líneas y con puntos	24
2.8 Texto en el espacio virtual con *VRML*	24
2.9 Interacción de *VRML* y *Java*	27
3.1 Esquema general de procesamiento de imágenes en visión artificial	**30**
3.2 Iluminación direccional	31
3.3 Iluminación difusa mediante un anillo de luz	31
3.4 Iluminación a contraluz	31
3.5 Iluminación estructurada	32
3.6 Efecto del muestreo espacial en una imagen	33
3.7 Diferentes niveles de brillo para una imagen	33
3.8 Conectividad entre píxeles	34
3.9 Teorema de *Jordan*	34
3.10 Paradoja de conectividad	34
3.11 Proceso de extracción de contornos mediante derivada.	36
3.12 Imagen discreta	36
3.13 Imagen analógica a digital o discreta	39
3.14 Aplicación de una máscara en una imagen digital (Parte 1/4).	39
3.14 Aplicación de una máscara en una imagen digital (Parte 2/4).	40
3.14 Aplicación de una máscara en una imagen digital (Parte 3/4).	41
3.14 Aplicación de una máscara en una imagen digital (Parte 4/4).	42
3.15 Resultados al aplicar derivadas a una imagen en *MatLab*	43
3.16 Resultados al aplicar operadores gradiente a una imagen (Parte 1/3)	45
3.16 Resultados al aplicar operadores gradiente a una imagen (Parte 2/3)	46
3.16 Resultados al aplicar operadores gradiente a una imagen (Parte 3/3)	47

	Página
3.17 Operador *Laplaciano*	48
3.18 Algoritmo de seguimiento de contornos	50
3.19 Ecuación de una recta en forma polar	51
3.20 Características variables de los objetos en una imagen	52
3.21 Puntos clave para algunos objetos	53
3.22 Figuras empleadas para el cálculo del factor de forma	54
3.23 Angulo principal de la región R	56
3.24 Objetos en diferentes situaciones	57
3.25 Obtención de códigos ara *OCR*	59
3.26 Calibración de cámara	61
3.27 Modelo para calibración en dos planos	62
4.1 Componentes del Laboratorio de Robótica Virtual	**66**
4.2 Robot UNIMATE S-103	67
4.3 Dimensiones del robot UNIMATE S-103 (Acotación: Pulgadas)	68
4.4 Dimensiones del espacio de trabajo del robot UNIMATE S-103 (Acotación: Pulgadas)	69
4.5 Estructura cinemática del robot SCARA UNIMATE S-103	70
4.6 Divisiones para el robot virtual UNIMATE S-103	72
5.1 Sistema de Visión para el Robot UNIMATE S-103 con comunicaciones con conexión a través de URL	**77**
5.2 Cámara Web usada para adquisición de imágenes	78
5.3 Datos del archivo de calibración de cámara camara.cal	79
5.4 Archivo de umbrales umb.cal	79
5.5 Matriz de almacenamiento de imagen img(x,y)	79
5.6 Máscaras usadas para la extracción de contornos	80
5.7 Definición de esquina para segmentación	80
5.8 Seguimiento de contornos 4-conectados	81
5.9 Imágenes obtenidas al aplicar los algoritmos para reconocimiento (Parte 1/2)	84
5.9 Imágenes obtenidas al aplicar los algoritmos para reconocimiento (Parte 2/2).	85
5.10 Imagen de prueba	86
5.11 Gráfica con tamaño de imagen constante y número de objetos variable	87
5.12 Gráfica con número de objetos constante y tamaño de imagen variable	88
5.13 Gráfica de tamaños de archivos de salida y de imagen, con un número de objetos variable	90
5.14 Sistema de Visión para el Robot UNIMATE S-103 con comunicaciones mediante Sockets y Datagramas	93
6.1 Objetos usados para el reconocimiento de figuras geométricas	**96**
6.2 Formato del archivo de límites de momentos umb.cal	96
6.3 Archivo de umbrales umb.cal obtenido	97
6.4 Distribución de los momentos ϕ_1, ϕ_2 y ϕ_3 para figuras geométricas	97
6.5 Resultados del proceso de reconocimiento de figuras geométricas	98
6.6 Matriz para reconocimiento de dígitos	99
6.7 Dígitos empleados para la obtención de momentos	99

	Página
6.8 Archivo de umbrales umbx.cal obtenido	99
6.9 Distribución de momentos para el reconocimiento de dígitos	100
6.10 Asignación de las celdas de la matriz de 3x3	101
6.11 Ordenamientos de dígitos para asignación de posiciones en la matriz	101
6.12 Matriz resultante para el reconocimiento de dígitos	102
6.13 Resultados del proceso de reconocimiento de dígitos	102
6.14 Archivos obtenidos después del proceso de reconocimiento de dígitos	102
6.15 Sistema para reconocimiento de dígitos	103
6.16 Distancia recorrida, velocidad y aceleración de un punto en movimiento	105
6.17 Programa para análisis de movimiento	106
6.18 Imágenes de entrada para el análisis de movimiento	107
6.19 Archivo de entrada mov.ent para el análisis de movimiento	107
6.20 Archivo de salida mov.sal para el análisis de movimiento	107
6.21 Resultados para el programa de análisis de movimiento	108
6.22 Gráficas de posición, velocidad y aceleración para nuestro programa de análisis de movimiento	108
6.23 Objeto con la altura adecuada para manipularlo	110
6.24 Objeto con una altura menor que la estimada para manipularlo	110
6.25 Objeto con una altura mayor que la estimada para manipularlo	110
6.26 Circuito propuesto para la estimación de altura con un convertidor A/D	111
6.27 Estimación de altura usando luz estructurada	112
6.28 Estimación de altura por medio de rayos luminosos	112
6.29 Mecanismo usado para posicionar los haces láser para la estimación de altura.	114
6.30 Estimación de altura para dos objetos sobre la mesa de trabajo	115
6.31 Secuencia de fotografías para calibración de cámara	116
6.32 Datos obtenidos para la calibración de cámara	117
6.33 Parámetros encontrados para las figuras geométricas	117
6.34 Parámetros de las figuras geométricas	118
7.1 Sistema de visión para seguimiento de objetos celestes	**126**
7.2 Sistema de visión en la medicina	126
7.3 Circuito para implementar el algoritmo de localización del centro de los objetos	127
7.4 Cálculo del centro de un objeto por medios electrónicos	129
a.1 Disco de Nipkow	**132**
a.2 Disector de imagen o tubo de Fransworth	132
a.3 Funcionamiento de los dynodos multiplicadores	133
a.4 Iconoscopio (Tubo de condensación o acumulación)	133
a.5 Orticon de imagen	134
a.6 Vidicon	134
a.7 Plúmbicon	135
a.8 Construcción de un dispositivo de transferencia de carga	136
a.9 Funcionamiento de una celda de un dispositivo de transferencia de carga	136
a.10 Control lógico usado para transferir cargas en cada fila en un CCD	136

	Página
a.11 Composición de un separador óptico	137
a.12 Síntesis aditiva y substractiva del color	138
a.13 Características de una lente	138
a.14 Posiciones de un Zoom	139
a.15 Partes constitutivas de un Zoom	139
b.1 Modulación de un haz de luz por un objeto	**142**
b.2 Geometría de la triangulación	142
b.3 Circuito de control para el sistema de iluminación estructurada	142
b.4 Constitución de un diodo láser	144
b.5 Deformación del haz láser por un objeto	145
b.6 Fotografía del sistema de reconstrucción tridimensional	145
b.7 Imagen de un objeto para extracción de características	146
b.8 Programa de reconstrucción tridimensional mediante iluminación estructurada	146
b.9 Tiempo de procesamiento de imágenes para reconstrucción tridimensional	147
b.10 Tamaño de los archivos generados durante la reconstrucción tridimensional	148
b.11 Resultados para la reconstrucción tridimensional de una roldana	149
b.12 Resultados para la reconstrucción tridimensional de la mesa de trabajo de un robot con diversos objetos	149
c.1 Representación de objetos bidimensionales	**152**
c.2 Resultados de las transformaciones bidimensionales en un cuadrado	152
c.3 Representación de los objetos tridimensionales	156
c.4 Resultados de las transformaciones tridimensionales en un cubo	156
c.5 Vistas de la proyección paralela de una casa	157
c.6 Vistas de la proyección en perspectiva de una casa	157
c.7 Vistas de una proyección paralela de un conjunto de cubos	157
c.8 Vistas de una proyección en perspectiva de un conjunto de cubos	158
d.1 Uso de los patrones de calibración de cámara (Parte 1/2)	**159**
d.1 Uso de los patrones de calibración de cámara (Parte 2/2)	160
d.2 Patrón de calibración de cámara a 0º	161
d.3 Patrón de calibración de cámara a 15º	162
d.4 Patrón de calibración de cámara a 30º	163
d.5 Patrón de calibración de cámara a 45º	164
d.6 Patrón de calibración de cámara a 60º	165
d.7 Patrón de calibración de cámara a 75º	166
d.8 Cubo usado para pruebas de reconocimiento	167
d.9 Prisma rectangular usado para pruebas de reconocimiento	168
d.10 Cilindro usado para pruebas de reconocimiento	169
d.11 Prisma triangular usado para pruebas de reconocimiento	170
d.12 Prima hexagonal para pruebas de reconocimiento	170
d.13 Dígitos empleados para pruebas de reconocimiento	171
e.1: Menús del CD-ROM	173

	Página
e.2: Acrobat Reader 5.0	174
e.3: VRML 2.1	174
e.4: Internet Explorer 5.0	175
e.5: Kawa Professional Edition 5.0	175
e.6: Java 2 SDK	176
e.7: Borland C++ 3.0	176
e.8: WinZip 8.0	176

Lista de Tablas

	Página
1.1 Tipos de eslabones de robots	**3**
1.2 Tipos de robots dependiendo de la disposición de sus eslabones	3
2.1 Unidades convencionales usadas en *VRML*	**17**
2.2 Colores básicos en *VRML*	18
2.3 Colores metálicos con otros campos en *VRML*	18
3.1 Operadores gradiente más comunes	**44**
3.2 Factores de forma calculados y medidos con *MatLab*	55
3.3 Momentos calculados	58
3.4 Comparación entre diferentes técnicas de calibración de cámaras	60
3.5 Aplicaciones típicas de la visión artificial en inspección en la industria	63
3.6 Aplicaciones típicas para identificación de objetos en la industria mediante visión artificial	63
4.1 Parámetros de cinemática directa para el robot SCARA	**70**
5.1 Tiempos medidos cuando se incrementa el número de objetos	**86**
5.2 Tiempos medidos cuando se modifica el tamaño de imagen	88
5.3 Tamaños de archivos de salida y de Imagen, con un número de objetos variable	89
6.1 Asignación de posiciones en la matriz	**101**

Lista de Programas

	Página
2.1 Uso de las primitivas básicas de *VRML*	**16**
2.2 Nodo translation	19
2.3 Nodo rotation	19
2.4 Nodo scale	20
2.5 Nodo IndexedFaceSet	22
2.6 Nodo IndexedLineSet	23
2.7 Nodo PointSet	23
2.8 Código *VRML* para insertar texto	24
2.9 Programa JavaVRML.java (Parte 1/2)	25
2.9 Programa JavaVRML.java (Parte 2/2)	26
2.10 Código HTML para mostrar un mundo *VRML* y ejecutar el *applet* de *Java*	27
2.11 Programa root.wrl del mundo virtual vacío	27
3.1 Obtención de gradientes en *MatLab* (Parte 1/2)	**42**
3.1 Obtención de gradientes en *MatLab* (Parte 2/2)	43
3.2 Calculo del factor de forma en *MatLab*	54
4.1 Interpoladores de posición en *VRML* para el movimiento del robot	**73**
4.2 Definición de los sensores de tiempo en *VRML* para el movimiento del robot	73
4.3 Definición de las rotaciones en *VRML* para el movimiento del robot	73
5.1 Algoritmo para extracción de contornos	**80**
5.2 Algoritmo de búsqueda de esquinas para segmentación	81
5.3 Algoritmo de seguimiento de contornos para segmentación	82
5.4 Algoritmo de cálculo de momentos para reconocimiento (Parte 1/2)	82
5.4 Algoritmo de cálculo de momentos para reconocimiento (Parte 2/2)	83
6.1 Secuencia del ordenamiento por selección	**100**
6.2 Secuencia de asignación de celdas	101
6.3 Fragmento de código *VRML* para modelar un prisma triangular	118
6.4 Fragmento de código *VRML* para modelar un cubo	119
6.5 Fragmento de código *VRML* para modelar un prisma rectangular	119
6.6 Fragmento de código *VRML* para modelar un prisma hexagonal	120
6.7 Fragmento de código *VRML* para modelar un cilindro	120
6.8 Fragmento de código *VRML* para insertar números	121
7.1 Código ensamblador para comunicación serie	**128**
7.2 Algoritmo para calcular el centro de un objeto	129

Introducción

La automatización de procesos industriales, ha jugado un papel relevante en la historia humana, ha permitido la producción de bienes en un número suficiente y con una elevada calidad a costos razonables; ahorrando energía, resolviendo el problema de la carestía de mano de obra, etc. Desde la implementación de las máquinas en labores industriales, éstas se han hecho más elaboradas efectuando ya no solo las tareas simples y rutinarias, sino también, tareas complejas y peligrosas. En este sentido los sistemas que se han incorporado a dichas máquinas son más elaborados, por lo que no es raro encontrar máquinas dotadas de una cierta capacidad de inteligencia, las cuales no solo procesan información de sensores simples, también procesan informaciones acústicas e imágenes. Entre más compleja es la inteligencia de una máquina, más complejas son las tareas que puede realizar. Por ejemplo con un *sistema de visión* un robot, puede llevar a cabo labores de ensamble, ordenamiento, búsqueda y clasificación, entre otras de los objetos presentes en su área de trabajo, tal y como lo haría un *operador humano*.

Algunas de las ventajas que los robots aportan a los sistemas industriales son las siguientes [Schilling 90]:

1) *Seguridad*.- la complejidad de ciertas operaciones aumenta la posibilidad de errores, de los que a veces puede resultar daño para el operador que realiza el trabajo.

2) *Calidad*.- la consistencia de un proceso depende de la posibilidad de mantener dentro de límites establecidos factores tales como la fatiga y la monotonía derivada de la repetición sistemática.

3) *Rapidez*.- en un robot se puede prever la realización de numerosas operaciones a partir de una ordenación constante y estricta.

4) *Precisión*.- los límites de la habilidad manual humana para procesos industriales pueden ser superados mediante aparatos y herramientas de gran complejidad.

5) *Optimización de los recursos industriales*.- al eliminar toda posible interrupción del proceso debida a factores humanos, la demanda de esfuerzo a las máquinas no presenta inconvenientes de tipo ético, moral o jurídico y sí únicamente económico, de manutención y de puesta a punto.

6) *Reducción de las instalaciones*.- la infraestructura de iluminación y calefacción, por ejemplo, puede ser más simple en ambientes donde trabajen robots, en comparación con ambientes en donde trabajen humanos.

7) *Reducción de costos*.- al reducir el factor mano de obra y permitir la competitividad industrial, es posible mantener la continuidad empresarial y el subsiguiente mantenimiento de puestos de trabajo adecuados.

Un *robot* es la unión de software y hardware [Schildt 87], siendo el primero la inteligencia que controla el mecanismo. Esta inteligencia es la que diferencia a un robot de otras formas de automatización. Existen básicamente dos tipos de robots. Los primeros son los *robots fijos* de ensamble industrial, como los utilizados en el ensamble de coches, los cuales trabajan en ambientes altamente controlados para los cuales fueron diseñados. El segundo tipo son los *robots autónomos* diseñados para operar en el mundo real.

La manipulación de robots requiere de personal capacitado, sin embargo la infraestructura para generar personal con las aptitudes necesarias para llevar a cabo esta labor, se encuentra limitada a unas pocas instituciones, por lo que es necesario encontrar nuevas formas de capacitar

al personal al tiempo que se dota de inteligencia al robot, por ejemplo a través de sistemas de *visión artificial*.

La *visión artificial* [Escalera 01], es un proceso de tratamiento de imágenes que permite obtener la descripción simbólica del mundo real a partir de imágenes, la cual consiste en la obtención, caracterización e interpretación de la información de una imagen tomada del mundo real. Un sistema de visión deberá contemplar los procesos de adquisición, preprocesamiento, segmentación y clasificación de los objetos presentes en la imagen proveniente del mundo real, además podrá reconocer y modelar dichos objetos [Schildt 87, Jackson 85].

La *adquisición* de imágenes es el conjunto de operaciones que se efectúan para transformar la información luminosa de una escena en una señal digital. Se deben de tener en cuenta los factores siguientes: iluminación, tipo de cámara empleada y el proceso de digitalización de las imágenes. Las imágenes no siempre presentan un formato adecuado para su análisis por lo que el siguiente proceso es el *preprocesamiento* de una imagen, la cual se refiere a las técnicas encaminadas a realizar una mejora de la imagen como son el nivel de gris, el contraste, la eliminación de ruido, el realce de algunas características de interés, etc.

Una vez que la imagen ésta en condiciones de ser procesada, se tienen que hallar los objetos dentro de la imagen de forma independiente, esto se hace a través de la *segmentación,* en este proceso se divide una escena en objetos. Se puede realizar por métodos de discontinuidad, por métodos de similitud, etc. Cada uno de los objetos hallados puede ser clasificado por lo que la siguiente tarea es la de *clasificación* ó extracción de características para reconocimiento, mientras que el *reconocimiento* es la identificación de cada objeto en la escena mediante una etiqueta.

Para representar el mundo real a partir de un proceso de visión artificial, es posible emplear una *modelación*, está puede ser solo el etiquetado o emplear algunos modelos geométricos, como los generados con *VRML* (*Virtual Reality Modeling Language*) para *modelación* y *reconstrucción en tres dimensiones* (*3D*).

Esta tesis presenta el diseño de un *Sistema de Visión para el Laboratorio de Robótica Virtual*, como una manera de aprovechar de modo más eficiente el equipamiento disponible en los laboratorios que cuentan con la infraestructura adecuada para llevar a cabo prácticas de robótica. Nuestro sistema hará accesible tal equipamiento a usuarios locales o remotos a través de una herramienta de comunicación entre computadoras tan poderosa como *Internet*, aprovechando que está disponible en la gran mayoría de las instituciones de educación superior y por muchos particulares. Los objetivos son los siguientes:

- El sistema deberá de realizar la adquisición, procesamiento, segmentación y modelado de imágenes, así como la localización de los objetos presentes en la imagen correspondiente a la escena del espacio de trabajo de un robot industrial.

- El sistema deberá de permitir el reconocimiento y reconstrucción 3D de los objetos presentes en la imagen, usando modelos en lenguaje *VRML* y su utilización en la actualización del *espacio virtual* del *Laboratorio de Robótica Virtual*; para la realización de prácticas de *visión artificial* a distancia utilizando *Internet*.

La realización de estos objetivos involucra una gran cantidad de tareas, las cuales se resumen en los siguientes párrafos.

Para la *selección del esquema de visión* existen dos esquemas, el de cámara fija y el de cámara móvil [Schilling 90], en el primero la cámara permanece en un lugar mientras se realiza el reconocimiento; en el segundo la cámara se mueve. Como se trata de un sistema de programación fuera de línea, no resultó restrictiva la elección de un esquema de visión con *cámara fija*, por que la imagen captada por la cámara es lo suficientemente amplia como para poder observar el espacio de trabajo de interés para las tareas que el robot va a desempeñar.

Para la *adquisición y mejora de imágenes* se usan algoritmos de filtrado, de eliminación de defectos de iluminación, de ruidos de digitalización y otros efectos nocivos [Alchemy 96, Escalera 01, Chapra 96, CVPR 97, Ibarra 98]. Para cubrir este propósito se emplearon métodos basados en *extracción de contornos* y específicamente el método de *aproximación de Roberts*.

La *calibración del sistema de visión real* permitió asegurar la repetitividad y efectividad en el puesto de trabajo real de los objetos hallados por el programa creado y simulado en el *espacio virtual* [Corke 96]. Se usó el método *de transformación homogénea* por su simplicidad [Schilling 90].

Para la *segmentación de imágenes* se utilizó el *seguimiento de contornos*, para hallar el contorno de cada objeto en la escena y así poder determinar los límites pertenecientes a cada uno de ellos [Escalera 01].

La *localización de objetos* se refiere al cálculo de las coordenadas del centro de gravedad, orientación y tamaño de cada uno de los objetos presentes en la escena (puesto de trabajo del robot) en coordenadas imagen [Escalera 01, Mammone 94, Masters 94, Corke 96, Ayres 87, Granville 92, Mazaira 94, Staugaard 87]. Se empleo el método de *cálculo de momentos de Hu* programados en *lenguaje C*, por su facilidad de implementación.

Para el *reconocimiento* se crearon los espacios de clasificación acordes para cada modelo, [Escalera 01], se usaron modelos estadísticos para hallar los espacios de clasificación de dichos objetos.

La *descripción de imágenes* comprende la elección y construcción de los modelos paramétricos para reconocimiento, así como el cálculo de los valores específicos de sus parámetros [Escalera 01, Staugaard 87, VRML 97]. Se usó *VRML* como modelo de despliegue *3D* de los objetos en un *espacio virtual*.

La *creación del mundo virtual* se realizó con *VRML*, por lo que resultó ser una copia fiel del espacio de trabajo completo del robot industrial *UNIMATE S-103,* la cual ya existe en parte, pues se tiene modelado el robot [Foley 96, VRML 97]. La *actualización del mundo virtual* garantiza que los objetos vistos, modelados y reconocidos aparezcan en el *espacio virtual.* Una parte importante del espacio virtual es que los objetos reales aparecieran con todas sus características relevantes para la tarea de manipulación como son: forma, tamaño, identidad, posición y orientación.

La *reconstrucción 3D con VRML* tuvo como tarea fundamental la actualización del espacio virtual con la información visual generada a partir de los parámetros extraídos de las imágenes se reconstruyeron los objetos presentes usando modelos *VRML 2.0* [Foley 96, VRML 97].

La *comunicación con el robot* consistió en la conversión de coordenadas, de tal forma que la información espacial del puesto de trabajo coincidiera con información espacial del mundo virtual creado [Deitel 98]. La comunicación entre el sistema de visión y el robot real se realizó usando una conexión con *URL* (*Uniform Resource Locators*) empleado para localizar datos en *Internet*, programándolos en *Java*.

La tesis se encuentra organizada de la forma siguiente:

El *Capítulo 1* aborda el tema de la *robótica*, incluye los conceptos necesarios para entender la importancia del empleo de sistemas de visión artificial, en sistemas automáticos, así como las expectativas actuales y futuras de este tipo de máquinas.

En el *Capítulo 2* se revisan las técnicas de modelación virtual y reconstrucción *tridimensional* empleando *VRML*. Se tiene una breve descripción de este lenguaje para modelación virtual y como se conjuga con *Java* para combinar sus posibilidades de visualización y de cálculo.

En él *Capítulo 3* se realiza el estudio de los diferentes métodos para diseñar e implementar un sistema de *visión artificial*. Se resalta la importancia de los sistemas de procesamiento de imágenes en visión artificial y se revisan los métodos de calibración de cámara.

El *Capítulo 4* presenta la composición y los objetivos del *Laboratorio de Robótica Virtual*, se destacan las características del robot UNIMATE S-103 y se estudia la geometría y cinemática del robot necesarias para su modelación.

En el *Capítulo 5* se presenta la forma en que se implementaron los diferentes algoritmos para el programa de *visión* y de *comunicaciones*. Se muestran los resultados obtenidos para el programa de reconocimiento y el análisis de los algoritmos empleados.

El *Capítulo 6* esta dedicado a la descripción de cuatro aplicaciones del sistema de visión para el *Laboratorio de Robótica Virtual*: el reconocimiento de figuras geométricas, el reconocimiento de dígitos, el análisis de movimiento y la estimación de altura de los objetos. También se muestran los resultados para la calibración de cámara.

Nuestras conclusiones y perspectivas se describen en el *Capítulo 7*.

En el *Apéndice A* se realiza una revisión histórica de los diferentes *sistemas de adquisición de imágenes*, desde los sistemas mecánicos hasta los dispositivos semiconductores y los sistemas ópticos que emplean las cámaras.

El *Apéndice B* presenta el diseño e implementación de un sistema de iluminación estructurada para adquisición de imágenes tridimensionales y su representación con *VRML*.

En el *Apéndice C* se muestran las ecuaciones matemáticas para llevar a cabo transformaciones geométricas en dos y tres dimensiones, necesarias para la modelación.

En el *Apéndice D* se presentan los patrones de calibración de cámaras utilizados, así como las figuras geométricas y dígitos utilizados para pruebas con el sistema de visión.

Resumen

El empleo de robots ha tenido un impacto importante en el desarrollo humano. Un robot consta de dos partes una inteligencia (software) y los mecanismos (hardware). Los robots aportan seguridad, calidad, rapidez y precisión a diversas tareas industriales, por lo que se empleo seguirá ampliándose. La importancia de dotar a los robots de sistemas de visión, es permitir que se incrementen sus posibilidades de interacción con el mundo real.

La visión un sentido importante y complejo. La visión por computadora involucra una serie de procesos que requieren de la adquisición, preprocesamiento, segmentación, clasificación, reconocimiento y modelación de los objetos presentes en una imagen.

Una de las formas de aprovechar mejor los pocos recursos de robótica disponibles es haciéndolos accesibles mediante Internet, de ahí la importancia de crear laboratorios virtuales que permitan realizar practicas de robótica de forma local o remota.

Nuestro Sistema de Visión actualiza el espacio de trabajo del robot, dando las características de los objetos presentes en su mesa de trabajo, como son su identidad, forma, posición, orientación y tamaño, presentándolos a un usuario local o remoto mediante el empleo de modelos en tres dimensiones con *VRML*.

Capítulo 1: Robótica

En la industria se emplean ampliamente robots para mejorar la productividad, pues pueden realizar tareas monótonas y complejas sin errores en su operación, y también pueden trabajar en ambientes intolerables para operadores humanos. Un robot industrial maneja partes mecánicas que tienen formas y pesos particulares, por medio de al menos un brazo, una articulación y una mano, y la suficiente potencia para realizar la tarea. Además debe de poseer algunos sensores. A bajo nivel se instalan interruptores, mientras que a un alto nivel los robots emplean *medios ópticos* (como un sistema de televisión) para explorar el ambiente que los rodea, reconocer imágenes y determinar la presencia y orientación del objeto (**Figura 1.1**). Estas tareas requieren de una computadora para el procesamiento de señales consistente en el *reconocimiento de imágenes*, en base a códigos numéricos asociados con dichas imágenes.

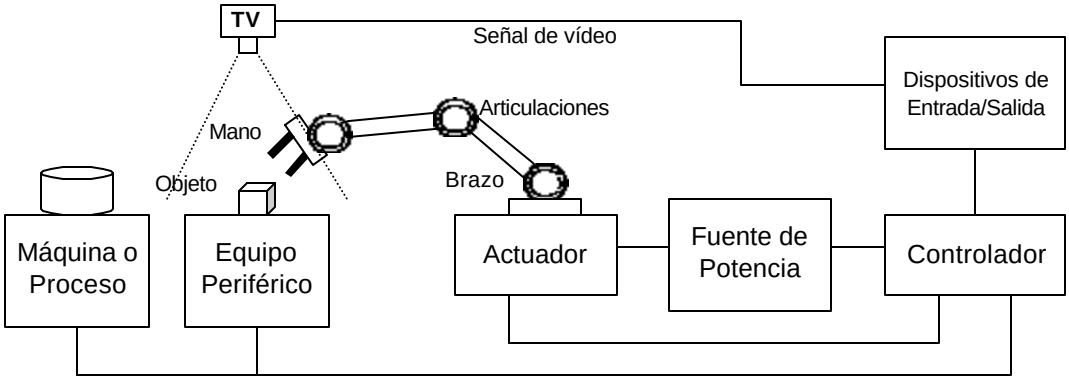


Figura 1.1: Robot que utiliza reconocimiento de imágenes.

En este capítulo se presentan algunos conceptos importantes relacionados con la robótica, así como con el desarrollo y evolución en esta área; se definen los tipos de robots de acuerdo a su arquitectura resaltando sus características más importantes; se describen los sensores que proporcionan percepción al robot, y se definen conceptos tales como teleoperación, telepresencia, webots, robots virtuales y sistemas híbridos hasta llegar a la robótica virtual. Además se presentan las perspectivas desde el punto de vista del mercado mundial que los robots tienen.

1.1 Definición de robot

El término *robot* presenta diferentes acepciones dependiendo del contexto [Schilling 90]. Sin embargo, se considera como un *robot* a toda aquella máquina capaz de realizar de forma automática funciones asignadas generalmente a personas [Larousse 80]. Existen dos tipos de robots, los *manipuladores* y los de *servicio*.

Los robots usados en plantas industriales, cuya forma es la de un brazo humano, son conocidos como *robots manipuladores* o *brazos robot*. Al igual que los brazos humanos poseen un órgano terminal, llamado *herramienta* o *mano*.

La clásica definición de la *Robotic Industries Association* (*RIA*) americana [IFR 98], ha sido sustituida por la de la norma *ISO 8373.* Según esta norma, un *robot industrial manipulador* es un manipulador programable en tres o más ejes, controlado automáticamente, reprogramable y multifuncional, que puede estar fijo en un lugar o ser móvil y cuya finalidad es la utilización en aplicaciones de automatización industrial.

No hay una definición de *robot de servicio*, pero la *International Federation of Robotics* (*IFR*) ha adoptado la siguiente definición provisional: *robot de servicio* es un robot que opera de

forma parcial o totalmente autónoma para realizar servicios útiles para el bienestar de los humanos y del equipamiento, excluyendo operaciones de manufactura. Los robots industriales manipuladores pueden ser considerados *robots de servicio* si están dedicados a operaciones diferentes de la manufactura. La *International Federation of Robotics* ha adoptado también una clasificación provisional de los robots de servicio por áreas de aplicación:

- Servicio a humanos (personal, protección, entretenimiento, ...).
- Servicio a equipos (mantenimiento, reparación, limpieza, ...).
- Otras funciones autónomas (vigilancia, transporte, adquisición de datos, ...).

1.2 Arquitectura de robots

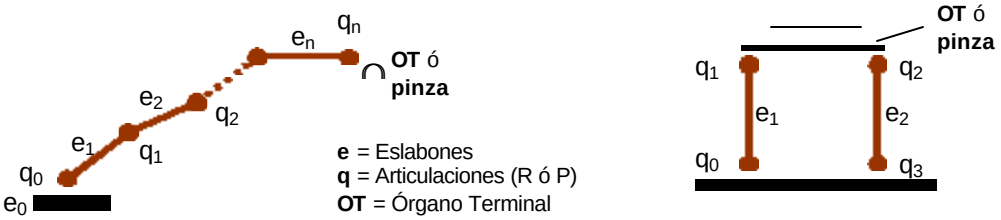
La *arquitectura* ó *estructura mecánica* de un robot determina tanto el *espacio de trabajo* como las prestaciones que pueden esperarse de un *robot* [Schilling 90]. Por este motivo numerosos estudios han intentado lograr estructuras que puedan sustituir con ventaja a las tradicionales, al menos en determinadas aplicaciones. A pesar de las numerosas propuestas realizadas, ninguna de ellas se ha abierto camino de una manera clara en el ámbito industrial. Los denominados *robots angulares* acaparan casi la mitad del mercado mundial y los más novedosos *robots paralelos* solamente representaban el 0.5% de los robots instalados hasta 1997.

En referencia a las articulaciones, dos interesantes paradigmas marcan los objetivos a alcanzar. Por un lado, la *articulación tipo nudillo* que se caracteriza por su ligereza, tamaño reducido, precisión y rapidez, y por otro, la de *tipo rodilla*, paradigma de relación entre diseño mecánico, control complejo y suspensión activa.

Los *actuadores de accionamiento directo*, evitan transmisiones que pueden dar lugar a oscilaciones o comportamientos inadecuados; parecen tener un futuro prometedor. De la misma forma, los *motores de estado sólido*, especialmente en *microrobótica* pueden tener un importante desarrollo. Dentro de este campo puede también mencionarse el diseño conjunto *actuador-control,* como un medio de conseguir mejores prestaciones del robot.

En cuanto a los sistemas de locomoción, aspecto esencial para los *robots móviles*, las ruedas siguen siendo la opción de mayor futuro, si bien combinadas con algún tipo de soporte articulado, activo o pasivo, para su utilización en terrenos irregulares.

Un robot se compone de diferentes partes como son: órgano terminal, eslabones y articulaciones, tal y como se observa en la **Figura 1.2**.



a) Cadena cinemática serie **b)** Cadena cinemática paralela

Figura 1.2: Tipos de cadenas cinemáticas.

El *órgano terminal* o *herramienta* de un robot, se encuentra montada regularmente sobre el último *eslabón* que conforma al robot. El *espacio de trabajo del robot* define los puntos que este puede alcanzar con su *órgano terminal* en el espacio tridimensional. El espacio de trabajo de un robot se encuentra determinado por el número de eslabones que lo componen, es decir, en cuantos ejes es posible que se mueva de acuerdo al tipo de *eslabón* empleado (**Tabla 1.1**). Con 3

ejes es posible obtener cinco tipos útiles de robot, aunque en la industria sólo se emplean tres tipos: el *esférico*, el *cilíndrico* y el *cartesiano* (**Tabla 1.2**).

Tipo	Notación	Símbolo	Descripción
Rotacional	R	↺	Movimiento rotatorio sobre un eje
Prismática	P	↔	Movimiento lineal a lo largo de un eje

Tabla 1.1: Tipos de eslabones de robots.

Robot	Eje 1	Eje 2	Eje 3	Total de Rotaciones
Cartesiano	*P*	*P*	*P*	*0*
Cilíndrico	*R*	*P*	*P*	*1*
Esférico	*R*	*R*	*P*	*2*
SCARA	R	R	P	2
Antropomórfico	R	R	R	3

Tabla 1.2: Tipos de robot dependiendo de la disposición de sus eslabones.

Otros conceptos relevantes, para entender el funcionamiento de los robots, son los siguientes:

a) *Grado de libertad* (*gdl*).- es el número de variables independientes que definen la postura del *órgano terminal* de un robot.
b) *Movilidad* (*mov*).- es el número de articulaciones o motores que conforman la estructura mecánica del robot.
c) *Postura*.- se compone de la orientación y posición de los eslabones del robot.
d) *Cadena cinemática serie*.- se compone de una serie de eslabones articulados de forma *serie* como se muestra en la **Figura 1.2a**.
e) *Cadena cinemática paralela*.- se compone de una serie de eslabones articulados de forma *paralela* como se muestra en la **Figura 1.2b**.

1.2.1 Clasificación de robots

Los robots pueden clasificarse de acuerdo a diferentes criterios, entre los que destacan: tecnología de funcionamiento, geometría del espacio de trabajo y por los métodos de control de movimiento [Schilling 90].

1.2.2 Tecnología de funcionamiento

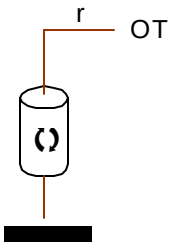
La *tecnología de funcionamiento* ó *fuente de alimentación* de los eslabones del robot es uno de los esquemas fundamentales para la clasificación de robots. Las tecnologías más comunes son la *eléctrica,* la *hidráulica* y la *neumática*. Por ejemplo, los robots manipuladores utilizan sistemas eléctricos como *servomecanismos de CD* (*Corriente Directa*) y *motores Paso a Paso* con los cuales se tiene una excelente precisión, pero son lentos y de poca potencia. Sin embargo cuando se requieren movimientos rápidos o la manipulación de cargas sustanciales, como el moldeo de acero y manipulación de partes de automóviles, los robots con sistemas hidráulicos ó neumáticos son los más empleados.

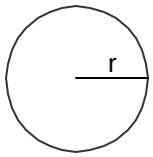
Es posible utilizar herramientas neumáticas ó hidráulicas, como órgano terminal, principalmente cuando se requiere de una manipulación que puede ser llevada a cabo mediante un solo movimiento, como subir ó bajar, ó abrir y cerrar, pero también tienen la ventaja de que es posible regular la fuerza mediante la presión de aire ó de algún fluido, con lo que se pueden manipular objetos delicados.

1.2.3 Geometría del espacio de trabajo

La *geometría del espacio de trabajo* de un robot indica en cuantos ejes es posible que se mueva su órgano terminal de acuerdo al tipo de *eslabones* empleados. En la **Tabla 1.2** se muestran diferentes configuraciones del espacio de trabajo respecto al número de eslabones empleados, estás se describen a continuación.

El *robot rotacional ó R*, se mueve de forma circular alrededor del soporte, es decir, exhibe un movimiento rotatorio alrededor de un eje. Además, es uno de los tipos más simples. Su espacio de trabajo es bidimensional, es decir, se mueve sobre un plano, pero conserva la misma distancia al origen. Se puede emplear en aplicaciones donde sólo se requiera mover objetos en un radio determinado (**Figura 1.3**).



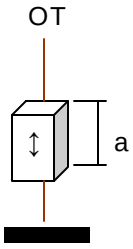


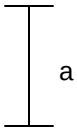
Espacio de Trabajo

Características:
gdl = 1
mov = 1
Espacio de trabajo = $2\pi r$

Figura 1.3: Robot rotacional ó R.

El *robot prismático ó P* se mueve de forma lineal, por lo que describe un desplazamiento lineal a lo largo de un eje. Su espacio de trabajo es unidimensional (solo en una dimensión). Al igual que en el robot rotacional (R); emplea de forma simple un eslabón. Se puede emplear en algunos casos donde sólo se requiera mover objetos a lo largo de una dirección (**Figura 1.4**).



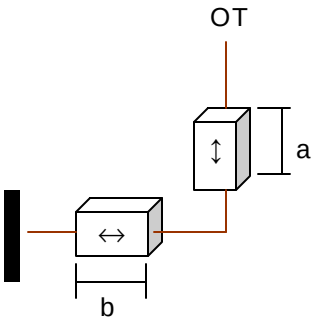


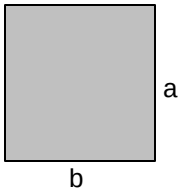
Espacio de Trabajo

Características:
gdl = 1
mov = 1
Espacio de trabajo = a

Figura 1.4: Robot prismático ó P.

El *robot prismático-prismático* o *PP*, se mueve en dos ejes, cubriendo un plano ó espacio bidimensional. A diferencia de los anteriores, su órgano terminal puede ubicarse en cualquier lugar del plano limitado solo por la longitud de sus eslabones en cada dirección. Se emplea en tareas de dibujo, corte, etc. (**Figura 1.5**).



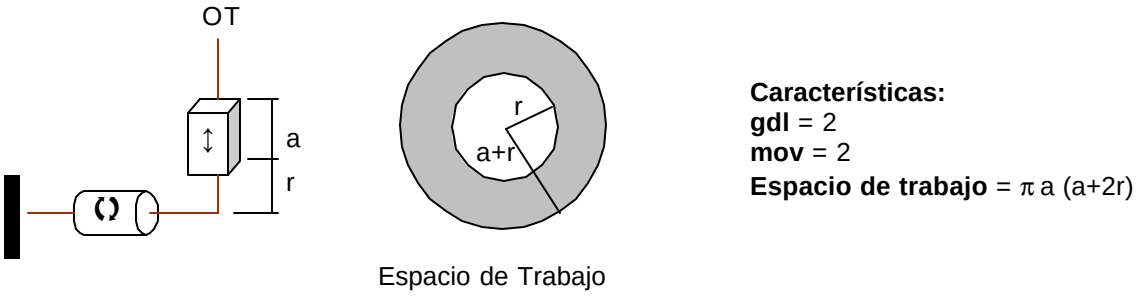


Espacio de Trabajo

Características:
gdl = 2
mov = 2
Espacio de trabajo = a b

Figura 1.5: Robot prismático-prismático ó PP.

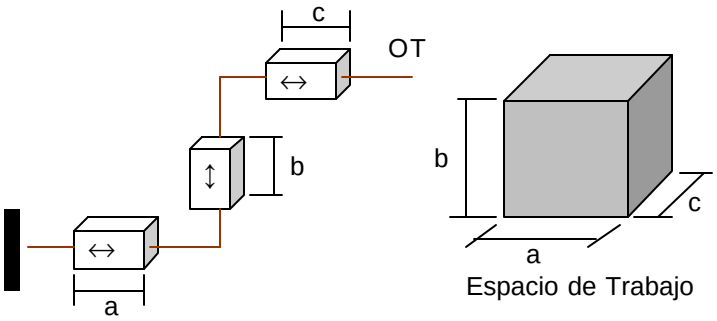
El *robot rotacional-prismático ó RP*, se mueve en dos ejes, describiendo un desplazamiento rotacional, por lo que se cubre un espacio bidimensional, en forma de corona circular (**Figura 1.6**).



Características:
gdl = 2
mov = 2
Espacio de trabajo = π a (a+2r)

Figura 1.6: Robot rotacional-prismático ó RP.

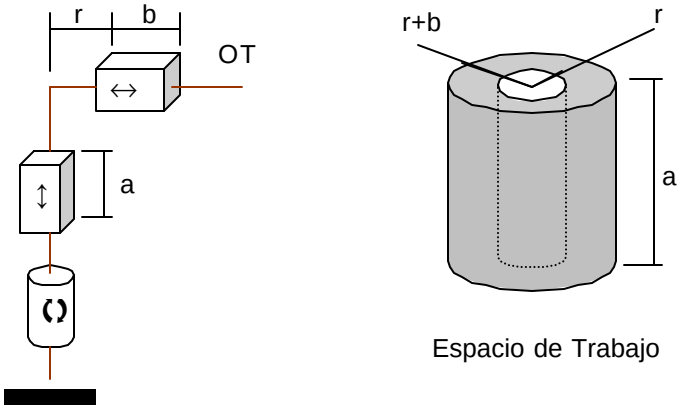
El *robot cartesiano, prismático-prismático-prismático ó PPP,* esta formado por un brazo con tres articulaciones traslacionales prismáticas ortogonales, y un órgano terminal orientado generalmente con articulaciones rotacionales. El órgano terminal se orienta en coordenadas cartesianas. Se utiliza en aplicaciones que requieren de gran precisión y rigidez, por lo que una vez que alcanza su posición difícilmente se mueve (**Figura 1.7**).



Características:
gdl = 3
mov = 3
Espacio de trabajo = a b c

Figura 1.7: Robot cartesiano, prismático-prismático-prismático ó PPP.

Un *robot cilíndrico, rotacional-prismático-prismático ó RPP*, esta construido de tal forma que presenta un movimiento de rotación azimutal, es decir, alrededor de un centro, seguida de dos articulaciones prismáticas, una horizontal y otra vertical. El órgano terminal suele tener movimiento en tres ejes concurrentes para asegurar su correcta orientación. Son útiles cuando las tareas a desarrollar, o las máquinas a las que deben apoyar, se encuentran radialmente cercanas al robot y no existen obstáculos. Se caracterizan por su precisión, aunque es menor a la de los robots cartesianos (**Figura 1.8**).



Características:
gdl = 3
mov = 3
Espacio de trabajo = πa b (b+2r)

Figura 1.8: Robot cilíndrico, rotacional-prismático-prismático ó RPP.

El *robot esférico, rotacional-rotacional-prismático ó RRP,* presenta una de las arquitecturas más empleadas en la construcción de robots industriales. Consiste de dos articulaciones rotacionales que permiten un movimiento azimutal y de elevación de una articulación prismática que le permite tener un desplazamiento. Se equipan con un órgano terminal con tres articulaciones rotacionales. Además puede tener actuadores hidráulicos que le permite manejar cargas pesadas. (**Figura 1.9**).

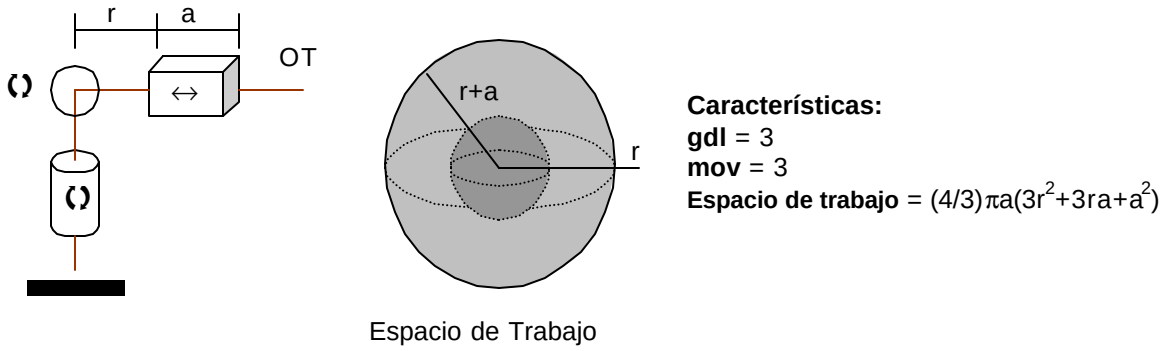


Figura 1.9: Robot esférico, rotacional-rotacional-prismático ó RRP.

Un *robot* ***SCARA*** (*Selective Compliance Assembly Robot Arm:* Brazo de Robot para Ensamble de Complianza Selectiva), *rotacional-rotacional-prismático ó RRP,* es un brazo de robot desarrollado por *Fujitsu-Fanuc* con el fin de ser un brazo compatible con el trabajo humano. Es un robot de coordenadas cilíndricas. Se mueve libremente en el plano horizontal mediante sus dos primeras articulaciones, ambas rotacionales con ejes paralelos y verticales entre sí; así se tiene entonces una complianza alta, es decir, que se deforma ligeramente en el peor de los casos. La última articulación es prismática y mueve únicamente al órgano terminal desplazándolo en alguna dirección (**Figura 1.10**).

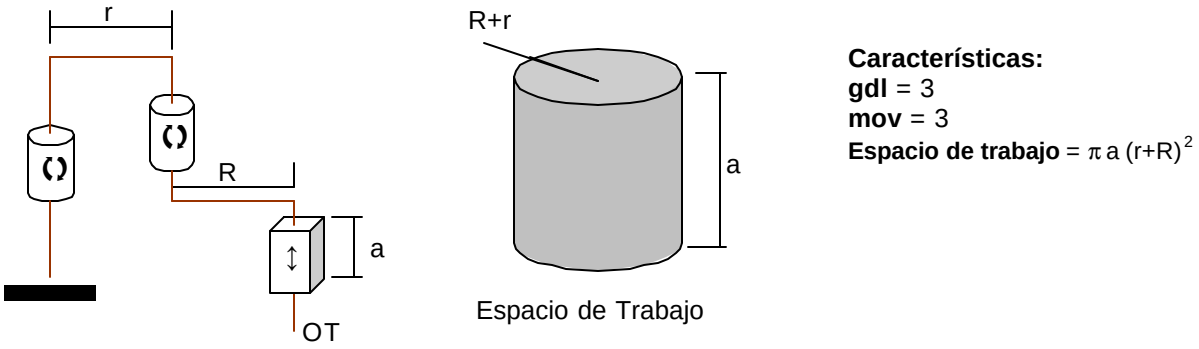


Figura 1.10: Robot SCARA, rotacional-rotacional-prismático ó RRP.

El robot *SCARA* ejecuta tareas de montaje y ensamble en el plano horizontal con mucha precisión y velocidad, pero también a un costo menor comparado con otros robots como el *PUMA* (*Programmable Universal Machine for Assembly*) que se emplea en tareas similares.

Un *robot antropomórfico, rotacional-rotacional-rotacional ó RRR,* tiene tres articulaciones rotacionales. Es un robot de coordenadas articuladas ó de coordenadas cartesianas dobles, por lo que su espacio de trabajo se asemeja a un toro en tres dimensiones. Presenta una configuración antropomórfica, ya que su anatomía se asemeja a la forma de un brazo humano (**Figura 1.11**). Es una de las arquitecturas más versátiles, puede tener tantos grados de libertad como sea necesario según la tarea a realizar.

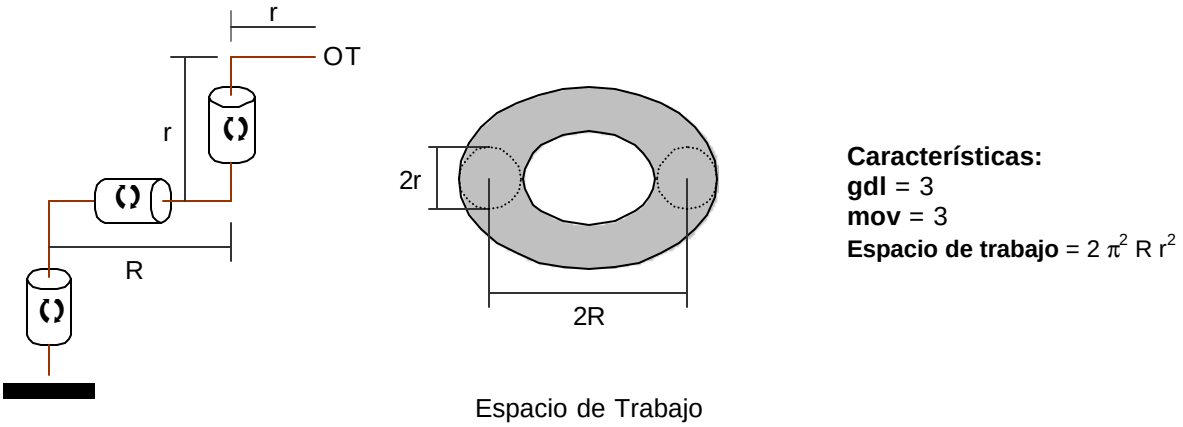


Figura 1.11: Robot antropomórfico, rotacional-rotacional-rotacional ó RRR.

1.2.4 Control de movimientos

El método de *control de movimientos* del órgano terminal es otra de las formas que existen para clasificar a los robots. Existen básicamente dos tipos de *control de movimientos* del órgano terminal: movimiento de *punto a punto*, en donde la herramienta se mueve de punto a punto en el espacio de trabajo; y el movimiento por un *camino continuo*, en donde el movimiento entre dos puntos no esta controlado por el usuario.

El movimiento punto a punto se utiliza en aplicaciones donde se tiene de forma natural un movimiento discreto. Por ejemplo cuando se colocan puntos de soldadura.

El movimiento por *camino continuo* también conocido como movimiento de *camino controlado,* describe un camino en el espacio tridimensional. La velocidad en el recorrido puede variar, por lo que el control es más complejo. Por ejemplo se tienen los robots que esparcen un líquido, o que colocan una capa de pegamento.

En los últimos años, los robots han constituido una plataforma excelente para la aplicación y ensayo de numerosas técnicas de control, entre las que cabe mencionar: el control adaptativo, el control por modos deslizantes, las técnicas de pasividad, el control difuso y el control neuronal. Muchos de los sistemas diseñados con estas técnicas de control, han sido probadas únicamente en simulación. No han sido sometidas aún a una verificación experimental que permita su validación real [Schilling 90].

No obstante, diversos fabricantes de robots han incorporado mejoras derivadas de estos diseños, y se puede apreciar una paulatina mejora en los sistemas de control por la disponibilidad de microprocesadores más rápidos y potentes. Algunos fabricantes han comenzado a incorporar en sus sistemas módulos de compensación dinámica que permiten al robot cargado seguir con precisión trayectorias a velocidades elevadas.

1.3 Sensores y percepción

La incorporación de *sensores* a los robots que permitan a estos obtener información de su entorno e interaccionar con él, ha sido mucho más lenta de lo previsto. Sensores como los de *tacto,* que en un momento dado fueron objeto de intensa investigación e incluso de comercialización en algunos casos, han quedado prácticamente olvidados. No obstante, no parece demasiado arriesgado afirmar que el desarrollo de la robótica futura, tanto de los robots manipuladores como de los robots móviles, pasará en gran parte por la incorporación de nuevos y más eficientes sensores [Ibarra 90, Schilling 90].

En robots manipuladores para tareas de montaje y mecanizado en las que existe contacto entre la pieza manipulada por el robot y el entorno, la utilización de sensores de fuerza puede tener

un incremento apreciable a mediano plazo. Esto será el resultado del abaratamiento del costo de los sensores y la disponibilidad de procesadores para el tratamiento en tiempo real de su información y de estrategias eficientes de ejecución de esas tareas que hagan uso efectivo de la información de fuerza.

Los *sistemas de visión* seguirán siendo, en cualquier caso, los más utilizados y los de mayor desarrollo futuro, tanto para los robots manipuladores (robots fijos que manipulan objetos a su alrededor) como para los robots móviles (robots capaces de moverse para manipular objetos, crear mapas geográficos, etc.), aunque para estos últimos los sensores de proximidad y distancia sigan constituyendo un elemento esencial. La *iluminación controlada* aparece como uno de los factores fundamentales de los *sistemas de visión* futuros. Aspectos como la posición y tipo de los focos, la utilización de luz estructurada y la explotación de las posibilidades de la longitud de onda y de la polarización serán, sin duda, de gran importancia en dichos sistemas.

Por otro lado, la incidencia de modelos, especialmente de representación jerárquica, de los programas oportunistas y de la selección automática de dichos modelos de acuerdo con el objetivo perseguido, tendrá una creciente importancia el procesamiento de la información de las imágenes. Igualmente, los futuros *sistemas de visión* harán un uso creciente del color, de la información en tres dimensiones y de las técnicas de visión activa como el seguimiento y el flujo óptico.

Mención especial merecen las técnicas de integración sensorial cuyo objetivo es combinar la información procedente de diversos sensores para construir y actualizar un modelo del entorno que sirva para un objetivo determinado. Estas técnicas han de permitir, por un lado, un uso más eficiente de los sensores disponibles con un incremento de la cantidad y de la calidad de la información obtenida, y por otro lado, la detección de errores y fallos en algún sensor, y la continuidad del funcionamiento, aunque degradado, del sistema.

1.4 Programación, planificación y aprendizaje

La *programación de los robots* para la ejecución de sus tareas es uno de los temas básicos para la efectiva expansión de los robots en los ambientes industriales [Ibarra 90, Schilling 90]. Uno de los temas recientes de discusión es la necesidad, ó conveniencia, de alcanzar un estándar sobre un lenguaje de programación para robots que pudiese ser convertido por software en el lenguaje específico de cada robot. Sobre este punto, diversos fabricantes han expresado opiniones contrapuestas, pero parece haber un consenso generalizado sobre la necesidad de hacer más fácil, seguro y eficiente para el usuario el desarrollo de aplicaciones robotizadas.

En esta dirección cumple un papel decisivo la *programación fuera de línea*, que no ocupa tiempo de trabajo del robot y es asistida de la simulación gráfica y de elementos como el posicionamiento relacional que facilita la obtención de las posiciones y orientaciones que ha de alcanzar el robot, para llevar a cabo sus tareas.

La *planificación de tareas* fue un tema en boga hace unos años, quedando después en un segundo plano debido a la dificultad de realizar planificadores realistas y utilizables en un entorno industrial. En estos momentos, los objetivos son claramente más modestos y la planificación se orienta más como una ayuda a la programación en tareas complejas que como un sistema autónomo ó robot. Ejemplos de este enfoque son la planificación de movimientos sin colisión, de la acomodación activa de objetos en tareas de montaje con robots y de las curvas de pulido en el acabado de piezas. En todo caso, los planificadores, para ser realistas, deben tener en cuenta la incertidumbre siempre presente en la ejecución de una tarea y la utilización de los sensores disponibles para llevar a cabo la misma.

El *aprendizaje* de las tareas que ha de realizar un robot es otra técnica prometedora aunque todavía incipiente en el entorno industrial. La introducción del aprendizaje en el campo de la robótica viene motivado básicamente por la necesidad del robot de adquirir automáticamente los conocimientos necesarios para la realización de determinadas tareas. Esta necesidad es debida, en algunos casos, a la existencia de tareas difíciles de programar pero cuya forma de realización

puede ser mostrada fácilmente al robot. Otras veces, el problema es que la información necesaria para programar el robot no es accesible y el robot ha de adquirir ese conocimiento mediante la exploración. Un caso similar se produce en entornos dinámicos en los que el robot ha de ser capaz de tener constantemente actualizado el conocimiento de los mismos.

1.5 Integración de robots

El robot industrial es cada vez más un elemento dentro de un sistema automático de producción [Ibarra 90]. Por esta razón, adquiere una gran importancia la *integración* de un robot con otros robots y con otras máquinas.

En el campo de la cooperación entre robots pueden mencionarse como temas abiertos el reparto de tareas entre distintos robots, el control combinado de fuerza y posición en la manipulación conjunta por parte de varios robots y como evitar colisiones entre ellos.

La integración de robots en celdas robotizadas (por la semejanza que guardan este tipo de implementaciones industriales con un panal de abejas) pone de manifiesto una serie de problemas hasta ahora resueltos solamente de forma parcial, como son el diseño de la arquitectura de la célula, la comunicación entre máquinas, la simulación del funcionamiento y la programación de la celda.

1.6 Teleoperación de robots y robótica virtual

La *teleoperación* es el manejo a distancia de manipuladores, de vehículos y de cualquier otro tipo de robots. Para poder teleoperar un robot, es necesario contar con una interfaz de usuario dotada de sensores capaces de detectar los movimientos realizados por el usuario, los cuales serán enviados como consignas de posición y/o velocidad para los controladores articulares del robot y por ende, de sus motores. Estas interfaces pueden ser desde una simple botonera, hasta una compleja estructura tipo exoesquelética. El requisito indispensable aquí es que los sensores de la interfaz de usuario generen señales que comanden los servomecanismos del robot, a través de un sistema de comunicación unidireccional.

La teleoperación de robots manipuladores es uno de los principales antecedentes de los robots industriales [Ibarra 90, Schilling 90]. En la industria nuclear de los años cuarenta y cincuenta se desarrollaron brazos mecánicos operados a distancia para manipular con seguridad substancias radioactivas en espacios confinados. La evolución que ha sufrido la teleoperación permite hoy en día el manejo de robots en la superficie marciana comandados desde la Tierra, con base en avanzadas tecnologías de telecomunicaciones, de computación, de control, de sensores y de electrónica, todas ellas de altas prestaciones, pero de costos inaccesibles para otro tipo de aplicaciones. En la actualidad, la teleoperación de robots presenta problemas científicos y tecnológicos interesantes desde los puntos de vista de las telecomunicaciones, computación, control automático, etc. Esto ha hecho proliferar sistemas más o menos complejos que cada vez dificultan más su clasificación y hacen confuso el uso de los términos asociados.

La dificultad de programar un robot en operaciones complejas y con capacidad de adaptación a situaciones cambiantes ha hecho resurgir la idea de la teleoperación. En un futuro próximo, la teleoperación asistida, en la que el robot es capaz de desarrollar ciertas operaciones ordenadas por el operador sin necesidad de que éste tenga que realizarlas en detalle, y la utilización del retorno sensorial, a través del cual el operador experimenta las sensaciones de la tarea, tendrán un desarrollo creciente. Actualmente estas técnicas están siendo ya aplicadas con éxito en tareas complejas de mantenimiento y en campos de reciente introducción de la robótica, como la construcción, exploración, la agricultura, la industria alimentaría y la medicina.

La teleoperación no precisa de algún tipo de retroalimentación de sensaciones hacia el usuario, cuando esta se presenta se llama telepresencia. La *telepresencia* es la percepción, por parte del usuario, de sensaciones correspondientes a la información sensorial obtenida por un robot remoto durante una cierta tarea de interacción con su entorno. Esto requiere que la interfaz

de usuario remoto sea capaz de convertir las señales correspondientes en estímulos sensoriales detectables por los sentidos del usuario (visión, tacto, sonidos, etc.). Recientemente han aparecido diversas instituciones educativas y de investigación de varios países que tienen publicadas páginas *web* en las cuales ponen diferentes tipos de robots a la disposición de cualquier usuario [Internet 4.1 y 4.2]. Generalmente se trata de robots manipuladores o de robots móviles reales susceptibles de ser teleoperados desde una página *web* en la cual también es posible observar los movimientos del robot mediante el uso de cámaras de vídeo (*web cam*). A este tipo de robots se les comienza a denominar *webots*, término que significa *robot en la web*.

Cuando se hace uso de modelos matemáticos que representan los principales aspectos físicos de un robot y se emplean diferentes técnicas de simulación gráfica, es posible tener en la computadora a un *robot virtual* cuyo comportamiento sea equivalente o similar al de un robot real. La limitación de un *robot virtual* con respecto al robot real depende de los modelos empleados para representar su comportamiento, pudiendo ser meramente geométrico (en dos o tres dimensiones) ó se puede incluir la cinemática exacta del robot real y aún su dinámica. El *robot virtual* requiere, además de los modelos, de un simulador capaz de reproducir artificialmente (virtualmente) las propiedades físicas de dichos modelos. Generalmente, los robots virtuales incluyen el modelado de su entorno, en particular de su puesto de trabajo con todos sus accesorios mecánicos. Finalmente, una de las partes fundamentales de un *robot virtual* es la interfaz gráfica con el usuario, cuyo diseño debe contemplar múltiples aspectos entre los que se encuentran la ergonomía, la estética y aún normas legales y sanitarias.

Del mismo modo que es simple entender la utilidad de combinar la teleoperación con la telepresencia para hacer más eficiente y seguro a un sistema, no debe haber dificultad para analizar o crear sistemas híbridos que combinen dos o más de alguno de los conceptos presentados en los párrafos previos. Por ejemplo, utilizando el modelo dinámico de un robot virtual, se pueden obtener señales sensoriales artificiales (virtuales) que pueden ser utilizadas en la interfaz de operación para generar el retorno de esfuerzos a fin de asegurar la telepresencia del usuario en el entorno del robot virtual simulado en la computadora. Otro sistema híbrido puede obtenerse de la combinación de la teleoperación de un robot real y de su correspondiente robot virtual, pudiendo incluir la telepresencia del usuario tanto en el puesto de trabajo del robot real como en el espacio cibernético en el que se desenvuelve el robot virtual, en donde la computadora representa la interfaz que permite al usuario hacer la teleoperación y sentir la telepresencia.

La *robótica virtual* es entonces la combinación de teleoperación y telepresencia con el auxilio de robots virtuales para la validación de programas de usuario mediante simulación, para el entrenamiento de nuevos usuarios y para la visualización de las tareas realizadas por el *webot*.

1.7 Mercado mundial de robots

El mercado mundial de robots, prevé que serán las máquinas industriales que más se difundirán en un futuro, debido al incremento en eficiencia que en los procesos productivos implica su empleo, sin embargo es posible que en el corto plazo algunas de las técnicas discutidas anteriormente no tengan una aplicación debido principalmente a los costos económicos que involucra su empleo.

En el periodo 1987 a 1997, la venta anual de robots industriales tuvo un claro crecimiento hasta 1990, año en que se vendieron 81,000 unidades, para caer en los años siguientes, debido principalmente al brutal descenso del mercado japonés, hasta alcanzar un mínimo en 1993 con poco más de 54,000 unidades vendidas [Ibarra 90, Schilling 90]. Después de una pequeña recuperación en 1994, en el año siguiente, se contempló un espectacular crecimiento de casi un 29% que se moderó a un 11% en el año de 1996 y a un 6% en el año de 1997, año en el que alcanzó la cifra mas alta de casi 85,000 unidades vendidas.

El número total estimado de robots operativos al final de 1997 supera la cifra de 711,000, con un crecimiento del 6.4% respecto al año anterior. De ellos, Japón, con prácticamente 413,000 robots, tuvo la mayor parte, y junto con los otros cinco grandes países en robótica (Estados Unidos,

Francia, Alemania, Canadá y China), totalizan casi 611,000 robots, quedando únicamente 100,000 para el resto del mundo.

El mercado anual de robots se mueve alrededor de la cifra de 5,000 millones de dólares, con un ligero descenso en 1997, pese al incremento de ventas de robots. Este hecho pone de manifiesto el continuo descenso del precio de los robots desde 1990, que en periodo 1990 - 1997 ha supuesto un decremento en dólares, que va desde algo más del 21% en Estados Unidos hasta el casi 50% en Francia. No obstante debe tenerse en cuenta que el precio de un robot, sólo representa un promedio de un 30% del costo total de un sistema de producción.

En cuanto a las áreas de aplicación, el 29.2% de los robots instalados en 1997 han estado dedicados a soldadura (13.2% de arco y 15.7% por puntos) que ha sido la aplicación mayoritaria, seguida por montaje (25.7%), manipulación (13.1%), mecanizado (8.7%) y paletización (3.1%). Estos porcentajes varían considerablemente cuando se refieren al total de robots operativos al final de 1997. En este caso, el montaje se destaca claramente con un 33.3%, pasando la soldadura a un segundo puesto, con un 23.9%. A mecanizado le corresponde el 9.6%, a manipulación, el 7.2% y al paletizado, el 2.8%. Únicamente del orden de 5,600 robots en todo el mundo *(0.9%)* están dedicados a *enseñanza e investigación*.

Por sectores industriales, la fabricación de vehículos automóviles es claramente el sector mayoritario, rozando el 30% del total, tanto en nuevas instalaciones de robots como en número de robots que actualmente están operando.

En cuanto a los tipos de robots, los de 5 ó más ejes instalados en 1997 representan el 65% del total, mientras que los de 3 y 4 ejes se reparten el resto en partes aproximadamente iguales. Los robots angulares (con al menos tres articulaciones de rotación) suponen el 47.1% de los robots instalados en 1997. A este respecto, cabe destacar el reducido número de nuevas instalaciones de robots *SCARA* (*Selective Compliance Robot Arm*) que alcanza solamente el 2.6% del total, únicamente por encima de los *robots paralelos* (cuyos brazos tienen articulaciones prismáticas o de rotación concurrentes) que tienen el 0.5%. Finalmente, los robots de trayectoria continua controlada representan la inmensa mayoría (82.4%) de los instalados en 1997.

Resumen

Un robot es un autómata capaz de realizar una tarea monótona con alta eficiencia. Las características principales de un robot son: el espacio de trabajo, la movilidad y los grados de libertad.

Un robot manipulador tiene como función principal la automatización industrial, mientras que un robot de servicio es aquel que se aplica a áreas de servicio a humanos, equipos u otros autómatas.

La arquitectura de los robots es la estructura mecánica que condiciona tanto el espacio de trabajo como las prestaciones que pueden esperarse de un robot.

Los robots se clasifican de diferentes formas: por la tecnología de funcionamiento: eléctrica, hidráulica y neumática. Por la tecnología del espacio de trabajo: rotacionales, prismáticos, cartesianos, cilíndricos, esféricos, SCARA *(Selective Compliance Assembly Robot Arm)* y antropomórficos, entre muchas otras. Por los métodos de control de movimientos: de punto a punto y de movimiento continuo.

La incorporación de sensores a los robots les permite obtener información de su entorno e interaccionar con esté de una manera más eficiente. Los *sistemas de visión* seguirán siendo, en cualquier caso, los más utilizados y los de mayor desarrollo futuro, tanto para los robots manipuladores como para los robots móviles, aunque para estos últimos los sensores de proximidad y distancia sigan constituyendo un elemento esencial.

Aunque resulta difícil hacer previsiones en el desarrollo de la robótica, algunos temas destacan de manera clara: las exigencias crecientes de fiabilidad y eficiencia, la interfaz hombre máquina a través de sistemas gráficos y programación fuera de línea, la importancia creciente de los sensores y de la integración sensorial, la interconexión entre máquinas, la coordinación entre robots y otras máquinas y la teleoperación. Igualmente, es importante mencionar los nuevos campos en expansión de aplicación de la robótica como la exploración, la agricultura, la industria alimentaria y la medicina, que complementarán en el futuro la ya tradicional robótica industrial.

Los robots industriales ocupan un lugar destacado dentro de la automatización de la producción y su papel se ha ido consolidando en los últimos años. Después de un descenso en las ventas, que tuvo su mínimo en 1993, el mercado de robots ha seguido una evolución creciente. No obstante, la industria automotriz continúa siendo el sector mayoritario en cuanto a utilización de robots. El número de robots dedicados al montaje en el conjunto del mundo es mayoritario, seguido por el número de robots dedicados a labores de soldadura.

Capítulo 2: Ambientes Virtuales y VRML

La creación de mundos virtuales que representan la reconstrucción de objetos reales, es una de las áreas que tendrá más desarrollo a principios del siglo XXI. Por un lado, la tecnología en computación permitirá tener un mayor alcance. Por otro está la exigencia de los usuarios por tener estos nuevos sistemas. Los ambientes virtuales serán cada vez más empleados por usuarios comunes.

Para la programación de ambientes virtuales existen diversas herramientas, entre las que se encuentra *VRML* (*Virtual Reality Modeling Language*). *VRML* permite la creación de *mundos virtuales* con objetos en tres dimensiones y el control de estos objetos a través de sensores de movimiento, tacto, visibilidad, proximidad y tiempo, permitiendo su manipulación y/o animación en el espacio virtual. Los objetos realizados con *VRML* pueden ser visualizados desde un *navegador de Internet* (*Internet Explorer, Netscape Comunicator*, etc.), añadiendo visualizadores (*plug-in*) como *Cosmo, Platinum o el propio de Microsoft*. Los archivos de programas *VRML* se pueden crear en cualquier editor de texto. Además, *VRML* puede interactuar con otros lenguajes de programación como *Java*, haciendo posible combinar una poderosa herramienta de visualización con las ventajas de un lenguaje de programación de propósito general.

En este capítulo, se presenta la importancia de los ambientes virtuales, la programación con *VRML* y su combinación con *Java* para crear y manipular ambientes virtuales.

2.1 Ambientes virtuales

Los ambientes virtuales representan un área de la computación que se combina con otras para la creación de presentaciones realistas del mundo real [Shneiderman 98]. Estas representaciones tienen varias aplicaciones:

En *simuladores de vuelo* se emplean algunos trucos para crear ambientes reales. Las ventanas son reemplazadas por monitores de computadora de alta resolución y el sonido y la coreografía causan una impresión de realidad. La vibración y el movimiento, durante una simulación, son generados por mecanismos hidráulicos y sistemas de suspensión más o menos complejos. Esta tecnología es elaborada y hoy en día puede costar unos $100,000 dólares. Sin embargo existen simuladores de vuelo para computadoras personales a un costo mucho más accesible. Los simuladores de vuelo son complicados y muy especializados, pero son los sistemas virtuales más comunes o llamados de forma más descriptiva, *ambientes virtuales.*

Los *arquitectos* han usado las computadoras para representar dibujos en tres dimensiones durante las dos últimas décadas [Shneiderman 98]. Ahora emplean monitores más grandes, o proyectores, para ofrecer una mejor perspectiva a los clientes de sus proyectos, dando así un mayor realismo. Se adicionan animaciones, en donde el espectador se puede mover a la izquierda, derecha u observar de otra forma la imagen. Los clientes ahora pueden tener control de la animación, por ejemplo, caminando y atravesando puertas. Los proyectores de pantalla ancha pueden ser reemplazados por un casco con monitores colocado en la cabeza del espectador, que contiene además sensores de movimiento, por lo que cada cambio en la posición del cliente muestra una perspectiva real desde ese punto de vista.

Otro uso de los ambientes virtuales se encuentra en la *ciencia médica*, en donde los doctores acostumbran observar a los pacientes. La observación del interior del cuerpo humano es una tarea complicada [Shneiderman 98]. Los procedimientos en la cirugía moderna y la tecnología pueden beneficiarse mediante la observación de imágenes de vídeo del interior del paciente. El corazón puede observarse introduciendo una fibra óptica con una cámara en el extremo que puede

ser usada por manipulación directa remota. Esté dispositivo minimiza los efectos de la cirugía. También es posible tener planos tridimensionales del cuerpo humano, que pueden ser visualizados y guiados desde un monitor convencional.

Otra aplicación interesante incluye la *realidad artificial*. Se puede generar a través de instalaciones complejas de vídeo, combinando proyectores de pantalla ancha y sensores de vídeo, cuyo objetivo es imitar todos los movimientos del cuerpo como una proyección de imágenes que cubren todo el espacio visual. Por ejemplo, se pueden crear criaturas de luz que caminen a lo largo de un pasillo o a través de patrones multicolores. Si se incluyen la generación de sonido mientras se mueven, se incrementa el realismo de la experiencia.

El aspecto de *telepresencia* de la *realidad virtual* elimina las limitaciones físicas de espacio y permite que los usuarios actúen con más libertad. Los mundos virtuales generados por telepresencia pueden ser usados para tratar pacientes, pues pueden estar inmersos en una experiencia controlada, desde su punto de vista, que les permite mantener una seguridad física. En la práctica, este tipo de sistemas incorporan la conexión remota para manipulación directa y *visión remota*.

Los principios de *manipulación directa* son de gran ayuda para la gente que diseña y refina los ambientes virtuales. Los usuarios deben de ser capaces de seleccionar acciones rápidamente mediante un apuntador, un gesto, mediante la voz, etc., con un control incremental / reversible y con monitores que muestren una respuesta inmediatamente para causar una sensación de causalidad. Los objetos de las interfaces deben de ser simples, desde el punto de vista de los usuarios, y permitir el dominio y manipulación de los objetos. Alternativamente, existen los ambientes que permiten sumergirse en ellos, llamados ambientes *virtuales de escritorio* (*desktop*) o *tanques de peces* (*fishtank*) (ambos hacen referencia a la observación de pantallas estándar) y comienzan a ser de uso común [Shneiderman 98]. Utilizan principalmente gráficas tridimensionales que puede controlar el usuario para la exploración de lugares reales, visualización científica o mundos fantásticos. Muchas aplicaciones de este tipo son ejecutadas en estaciones de trabajo de alto desempeño capaces de crear imágenes rápidamente, pero que pueden ser ejecutadas en red usando *lenguajes de modelación de realidad virtual* como *VRML.*

Los investigadores en gráficas perfeccionan imágenes desplegadas en monitores, simulando efectos de iluminación, textura de superficies, reflejos y sombras. Las estructuras de datos y la ampliación de los algoritmos, permiten cruzar un objeto o un cuarto rápidamente y lentamente se empieza a practicar en computadoras comunes. Una innovación es lo que se conoce como *realidad aumentada*, en donde los usuarios observan un mundo real, y le adicionan información. Por ejemplo, se puede caminar sobre una calle, añadiéndole poca luz y neblina.

Una variante de los ambientes virtuales es llamada *conocimiento situacional*, en donde se usa una *palmtop* con un sensor de localización para controlar la pantalla. Cuando un usuario se mueve, la *palmtop* muestra en la pantalla un mapa, por ejemplo de un museo, en el que se muestra información acerca de las ciudades vecinas, las pinturas o la historia.

El desarrollo de los *ambientes virtuales* dependerá de la integración adecuada de múltiples tecnologías, como el diseño de nuevas pantallas visuales, los sensores de posición de la cabeza, de la mano, de mecanismos de retroalimentación de fuerza, integración de sonido y reconocimiento de voz, de la incorporación de otras sensaciones como calor, humedad, sabor, olor, etc.

Las *pantallas visuales* usadas en computadoras personales (de 12 a 17 pulgadas medidas diagonalmente), permiten que sean observadas a una distancia de unos 70cm, con un ángulo de 5º, las pantallas grandes (de 15 a 22 pulgadas) pueden cubrir un campo visual de 20º a 30º y los monitores montados sobre la cabeza cubren unos 100º en la horizontal y unos 60º en la vertical. Los monitores montados sobre la cabeza, muestran otros bloques de imágenes, con efectos más dramáticos y con el moviendo de la cabeza procesan nuevas imágenes, causando en el usuario una sensación de visión de 360º. Los simuladores de vuelo las aplicaciones que emplean algunas

imágenes de este tipo. La tecnología provee rápidamente de monitores de más alta resolución. Se han desarrollado monitores que proveen respuestas en tiempo real (probablemente con un retardo menor a los *100ms*) en la presentación de las imágenes a los usuarios. Los monitores de baja resolución, son aceptables para usuarios u objetos en movimiento, pero cuando se detiene el espectador, los monitores de alta resolución son necesarios para preservar la sensación visual deseada. La mejora del hardware y de los algoritmos es necesaria para desplegar formas y rellenos rápidamente con los detalles del movimiento.

La *detección de la posición de la cabeza* requiere de sensores de alta precisión (menor a 1º) y rápidos (con *100ms* de respuesta). Los movimientos del ojo pueden ser reconocidos, para encontrar el foco de atención del espectador, sin embargo se presenta una dificultad en los movimientos del usuario pues el monitor se encuentra montado sobre la cabeza.

La *detección de la posición de la mano* se realiza mediante un *guante de datos*. El problema actual con estos dispositivos es la inexactitud en las mediciones y la falta de un lenguaje de movimientos estándar. Se puede tener una precisión en las mediciones de la posición de los dedos, pero solo para uno o dos dedos. La orientación de la mano es provista por un guante montado en la mano. Los sensores en otras partes de cuerpo como las rodillas, los brazos o piernas pueden ser usados, en sistemas más complejos.

La *retroalimentación de la cantidad de fuerza aplicada* es de gran utilidad en operaciones manuales con dispositivos de control remoto para realizar experimentos en laboratorios de química o para manejar materiales nucleares para que los usuarios manejen de la mejor manera los objetos. La retroalimentación de la fuerza aplicada para manejadores de automóviles y pilotos puede ser configurada para que presente un realismo e información táctil usable. Simular la retroalimentación a través del software puede ayudar a responder preguntas sobre el comportamiento de moléculas complejas.

La *entrada/salida de sonidos* aumenta el realismo. El sonido digital es adecuado para el hardware pero las herramientas de software en ocasiones son inadecuadas. La salida de sonido de instrumentos virtuales es promisoria; fácilmente se pueden simular instrumentos existentes como un violín, pero otros instrumentos nuevos han aparecido. El reconocimiento de voz complementa los gestos de las manos en algunas aplicaciones.

Otras sensaciones pueden ser utilizadas para incrementar la experiencia. La inclinación y la vibración en los simuladores de vuelo son una inspiración para algunos diseñadores. ¿La inclinación y vibración virtual permitirán viajar a través de una costa a 60, 600 ó 6,000 kilómetros por hora, chocar contra una montaña o ir a órbita?. Otros efectos, como la música disco y las luces estroboscópicas, pueden mejorar la experiencia virtual. ¿Por qué no se incluyen olores del aire, o frío o calor con una cubierta virtual de agua?. La computadora olfativa (que detecta olores) es posible, ¿Pero será apropiada para aplicaciones prácticas?.

El *trabajo competitivo* es otra área de investigación, como los ambientes *virtuales cooperativos*, o como los diseñadores lo llaman *virtualmente construido para dos*. Dos personas trabajando en sitios remotos, pueden cada una ver y sentir las acciones de la otra en una experiencia compartida. Los juegos competitivos, como el tenis, pueden ser construidos para dos jugadores. El software puede crear la atmósfera adecuada para que ambos jueguen en la pantalla. Los sonidos realistas crean una sensación que parece real. Presumiblemente los ambientes virtuales serán empleados para la relajación y el placer de encontrarse con otras personas.

2.2 Construcción de ambientes virtuales con VRML (Virtual Reality Modeling Language)

La construcción de ambientes virtuales se realiza en base a la modelación por computadora. La *modelación* por computadora encuentra hoy en día una gran variedad de aplicaciones en áreas industriales, de negocios, de gobierno, educación y entretenimiento [Foley 96]. Un modelo se realiza en base a cálculos previos, como por ejemplo los *movimientos de un robot*. En la modelación, el usuario guía el proceso de construcción, paso a paso, eligiendo los

componentes y los datos geométricos y no geométricos. El usuario puede solicitar en cualquier momento que muestre una vista del modelo que se ha creado hasta entonces.

Los modelos son específicos para la aplicación y se crean independientemente de cualquier sistema de presentación. Por lo que un programa de aplicación debe de convertir cada porción del modelo en una representación geométrica interna, hasta crear una imagen. Este proceso se puede presentar en dos fases: una consiste en que el programa de aplicación recorre su base de datos de la aplicación que almacena el modelo para extraer las partes que se visualizarán, y otra que consiste en juntar los datos extraídos con los atributos para convertirlos en un formato que pueda enviarse al sistema gráfico.

Los datos extraídos durante el recorrido de la base de datos deben ser geométricos o convertirse en datos geométricos, los datos se pueden describir al sistema gráfico, en función tanto de primitivas como de atributos de primitivas. Una *primitiva de presentación* es generalmente un modelo geométrico como una línea, un rectángulo, un polígono, un círculo, una elipse, etc. en dos dimensiones y polígonos, poliedros, etc. en tres dimensiones. Así la fase que le sigue a la modelación es la reconstrucción en dos o tres dimensiones.

El desarrollo de modelos virtuales no es sencillo sin el soporte de un lenguaje apropiado como *VRML* (*Virtual Reality Modeling Language*). *VRML* es un lenguaje para modelación de *realidad virtual* [Alarcón 00, VRML 97]. Los archivos realizados para trabajar con esta tecnología son archivos de texto con la extensión `*.wrl` (del inglés `world`). Tiene la ventaja de que genera archivos de tamaño pequeño, por lo que viajan a través de *Internet* de forma rápida.

Para que un mundo virtual en *VRML*, pueda ser visualizado en un navegador de Internet, es necesario instalar un visor (los cuales generalmente se obtiene de forma gratuita). *Microsoft* provee su propio visor de *VRML* con el *Internet Explorer*. Otros visores son *Cosmo Player* y *Platinum.*

2.2.1 Un programa en VRML

El **Programa 2.1** muestra un ejemplo sencillo de un programa en *VRML*. Este programa dibuja una caja amarilla con dimensiones en *x* de `4 unidades`, en *y* de `3 unidades` y en *z* con `5 unidades`. Obsérvese que la geometría se define dentro de un nodo `Shape` y la apariencia dentro del *nodo* que define a la geometría. En la **Figura 2.1** se observa el resultado, al observar la caja desde una esquina.

```
1.    #VRML V2.0 utf8
2.    Shape   {
3.         geometry Box{ size 4 3 5 } #Las dimensiones son: X Y y Z
4.         appearance Appearance{
5.                material Material {diffuseColor 1 1 0}
6.                }
7.         }
```

Programa 2.1: Uso de las primitivas básicas de *VRML*.

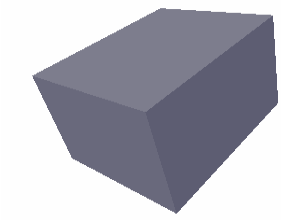


Figura 2.1: Caja simple en *VRML*.

La primera línea es con la que el navegador identifica la versión del archivo (cabecera). Para *VRML 2.0* la cabecera es la siguiente: `#VRML V2.0 utf8.` Donde `#VRML V2.0` denota el tipo y la versión, y `utf8` permite utilizar la codificación *UTF-8* para poder emplear todos los caracteres especiales del estándar *ISO 10646*.

Los comentarios en *VRML* se denotan colocando un signo # al comienzo del comentario.

La manipulación de objetos en el mundo virtual se realiza a través de los siguientes objetos definidos en *VRML*:

- *Nodos*.- son objetos geométricos tridimensionales, imágenes, colores, etc. Algunos de ellos tienen características variables que pueden definirse mediante *campos* (*fields*) los cuales funcionan como parámetros.

- *Campos*.- pueden ser univaluados (con un solo valor) o multivaluados, cuando se necesita una lista de valores.

- *Eventos*.- son mensajes que circulan entre *nodos* y permiten la variación de parámetros de un objeto durante la navegación en el *entorno virtual*. Los eventos pueden ser de *entrada* (si aceptan eventos), de *salida* (si envían eventos) o de *entrada / salida*.

En la **Tabla 2.1** se muestran las unidades convencionales empleadas en *VRML* aunque estas son arbitrarias.

Parámetro	Unidad
Distancias lineales	Metros
Ángulo	Radianes
Tiempo	Segundos
Color	RGB (Rojo, Verde y Azul)

Tabla 2.1: Unidades convencionales usadas en *VRML*.

Para definir un mundo virtual tridimensional en *VRML* se asumen tres ejes de coordenadas *eje x, eje y* y *eje z* que representan cada una de las tres dimensiones (**Figura 2.2**).

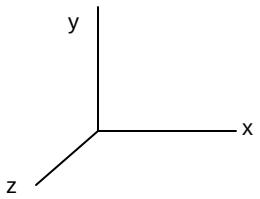


Figura 2.2: Ejes de coordenadas en *VRML*.

Cada objeto en el mundo virtual es desarrollado bajo un sistema de ejes local, por lo que para colocarlo dentro del mundo virtual, y en una posición diferente al origen será necesario aplicar una transformación para situarlo en el lugar correspondiente.

Existen en *VRML* algunas primitivas que definen una serie de objetos simples. Para poder visualizarlas, hay que usar un nodo *Shape*, el cual presenta dos campos *geometry* y *appearance*. Las primitivas para el campo *geometry* son:

- `Box { size x y z }` para dibujar un cubo de dimensiones **x, y** y **z** para cada uno de los ejes coordenados.

- `Sphere { radius r }` para dibujar una esfera con radio **r**.
- `Cone { bottomRadius r height h }` para dibujar un cono con una circunferencia en la base de radio **r** y altura **h**.
- `Cylinder { radius r height h }` para dibujar un cilindro con una circunferencia de radio **r** y altura **h**.

Las primitivas *VRML* anteriores definen la geometría de los objetos básicos. También es posible definir un color o textura. La apariencia de una primitiva se establece por medio del campo *appearance* del nodo *Shape.* Algunas primitivas de este nodo son:

- `Material`.- para definir un material de la primitiva como color y textura.
- `diffuseColor R G B`.- para establecer el color del material empleado. Se emplea el formato de color *RGB* (*R* para el rojo, *G* para el verde y *B* para el azul). Se tiene que indicar el valor de cada uno de los parámetros *RGB*, que representan la intensidad de cada uno de los colores básicos. En donde cada parámetro varía en el rango de *[0,1].* En las **Tablas 2.2** y **2.3** es posible observar como se combinan estos parámetros para proporcionar múltiples colores y texturas.

Color	Color RGB
Blanco	1 1 1
Amarillo	1 1 0
Cyan	0 1 1
Verde	0 1 0
Magenta	1 0 1
Rojo	1 0 0
Azul	0 0 1

Tabla 2.2: Colores básicos en *VRML*.

Descripción	`AmbientColor`	`DiffuseColor`	`SpecularColor`	`shininess`
Dorado	0.57 0.40 0.00	0.22 0.15 0.00	0.71 0.70 0.56	0.16
Aluminio	0.30 0.30 0.35	0.30 0.30 0.50	0.70 0.70 0.80	0.09
Cobre	0.33 0.26 0.23	0.50 0.11 0.00	0.95 0.73 0.00	0.93
Púrpura	0.25 0.17 0.19	0.10 0.03 0.22	0.64 0.00 0.98	0.08
Rojo	0.25 0.15 0.15	0.27 0.00 0.00	0.61 0.13 0.18	0.12
Azul	0.10 0.11 0.79	0.30 0.30 0.71	0.83 0.83 0.83	0.12

Tabla 2.3: Colores metálicos con otros campos de *VRML.*

2.3 Transformaciones en VRML

Las transformaciones en *VRML* se emplean para situar, escalar y orientar los objetos dentro del mundo virtual. Las transformaciones básicas son: translación, orientación y escala.

La *traslación* se define con un nodo `translation X Y Z`, que define el desplazamiento del objeto en cada una de las coordenadas. Al nodo `translation` le sigue un nodo `children` el cual contiene la lista de objetos que serán afectados por está transformación definida por el nodo `Transform`. Por ejemplo, en el siguiente programa (**Programa 2.2** y **Figura 2.3**), se muestra una esfera en el centro del mundo virtual y otra desplazada *10* unidades en los ejes *X* y *Z*, respecto a la primera.

```
1.    #VRML V2.0 utf8
2.    Shape   {              # Esfera de referencia en el centro
3.            geometry Sphere{ radius 2 }
4.            appearance Appearance{
5.                    material Material {diffuseColor 1 0 0}
6.                    }
7.            }
8.    Transform {
9.            translation 10 0 10   # Desplazamiento en X y Z
10.           children [            # Objetos a desplazar
11.                   Shape   {
12.                           geometry Sphere{ radius 2 }
13.                           appearance Appearance{
14.                               material Material {diffuseColor 1 1 0 }
15.                                   }
16.                           }
17.                   ]
18.           }
```

Programa 2.2: Nodo `translation`.

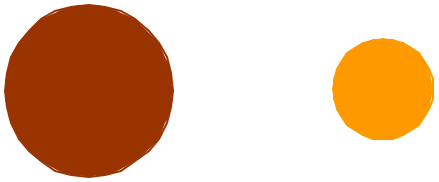


Figura 2.3: Nodo `translation`.

La *rotación* se define con el nodo `rotation X Y Z` α, en donde `X Y` y `Z` definen el vector en el espacio tridimensional de un objeto, o la orientación del eje de rotación, y α es el ángulo de rotación en *radianes* alrededor del eje definido anteriormente. Se emplea un nodo `children` que contiene los objetos que serán afectados por la rotación. Un ejemplo de la forma en que son empleados ambos nodos se tiene en el **Programa 2.3** y la **Figura 2.4,** donde se tiene una caja en el origen y dentro de ésta, otra caja con una rotación en el eje *Y* de *45º* (0.7854 rad).

```
1.    #VRML V2.0 utf8
2.    Shape   {         # Caja de referencia en el Centro
3.            geometry Box{ size 4 2 4 } # Las dimensiones son: X Y y Z
4.            appearance Appearance{
5.                    material Material {diffuseColor 1 1 0}
6.                    }
7.    }
8.    Transform{
9.            rotation 0 1 0 .7854  # Eje y ángulo de rotación
10.           children[
11.                Shape{           # Contiene  los objetos a transformar
12.                        geometry Box {size 2 4 2 }
13.                        appearance Appearance {
14.                        material Material {diffuseColor 0 0 1}
15.                                }
16.                        }
17.                ]
18.       }
```

Programa 2.3: Nodo `rotation`.

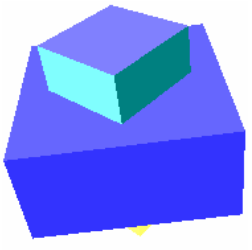


Figura 2.4: Nodo `rotation`.

El *escalamiento* de un objeto es de dos tipos: *escalado uniforme* y *escalado no uniforme*. El primero cambia el tamaño del objeto en todas las direcciones, así los objetos mantienen sus proporciones. El escalado *no uniforme* cambia el tamaño de un objeto solo en una dirección, deformando al objeto. Un ejemplo de un *escalado no uniforme* se muestra en el **Programa 2.4** y la **Figura 2.5**. Donde se tiene una caja con dimensiones unitarias, escalada en el eje *X* por *2* y en el eje *Z* por *3*.

```
1.    #VRML V2.0 utf8
2.    Transform {
3.          scale 2 1 3       # Define la escala en X Y y Z
4.          children [
5.                Shape   {
6.                        geometry Box{ size 1 1 1 }
7.                        appearance Appearance{
8.                            material Material {diffuseColor 0 1 0 }
9.                        }
10.                 }
11.         ]
12.   }
```

Programa 2.4: Nodo `scale`.

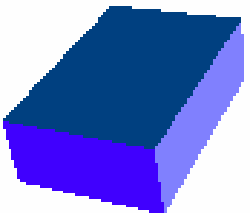


Figura 2.5: Nodo `scale`.

2.4 Puntos de vista con VRML

VRML permite definir diversos puntos de vista. Los usuarios pueden utilizarlos para moverse a partes importantes dentro del mundo virtual. Esta operación se realiza mediante el nodo `Viewpoint`.

El nodo `Viewpoint` se utiliza de la forma:

```
Viewpoint {
      position X1 Y1 Z1
      orientation X2 Y2 Z2 α
      description "Nombre"
      }
```

Donde X_1, Y_1 y Z_1 son el vector de la posición donde se sitúa el punto de vista, luego X_2, Y_2 y Z_2 definen el eje de rotación del punto de vista en un ángulo α, y `nombre` designa una etiqueta para identificar el *punto de vista*, este aparece en el navegador.

Cuando se ingresa al navegador, el punto de vista por omisión es:

```
Viewpoint {
      position 0 0 10
      orientation 0 0 1 0
      description "Punto de Vista Inicial"
      }
```

También es posible definir un ángulo de visión β, para que sólo sea visible una determinada área, se emplea el nodo `fieldOfView` de la forma:

```
Viewpoint {
      FieldOfView β           # β es el ángulo de abertura
      description "Nombre"
      }
```

2.5 Objetos, líneas y puntos tridimensionales en VRML

Las primitivas básicas permiten modelar solo unas cuantas formas geométricas, sin embargo *VRML* permite definir *objetos, líneas* y *puntos* a través de los nodos: `IndexedFaceSet`, `IndexedLineSet` y `PointSet.`

El modelado de objetos mediante el nodo `IndexedFaceSet`, se realiza definiendo los vértices y caras del objeto a través de arreglos de coordenadas de puntos y caras, de la forma siguiente:

```
geometry IndexedFaceSet{
      coord Coordinate{
            # Lista de vértices
            point [
                  x0 y0 z0, x1 y1 z1,
                  .........
                  x n-1 y n-1 zn-1, xn yn zn
                  ]
            }

      coordIndex [
            # Lista de caras
            0,1,2,...,-1, 3,4,5,...,-1,
            ............, n-2,n-1,n,-1,
            ]
      }
```

Se debe tener en cuenta las reglas siguientes:

- Los vértices tienen que ser consecutivos, siguiendo el perímetro de la cara.
- Los vértices se colocan en un *orden contrario* al de la dirección de las *manecillas del reloj*, pues la iluminación se refleja en un vector normal, hacia fuera de ésta cara. Mirando desde el lado contrario, la cara es transparente.
- El último vértice de cada cara se enlaza con el primero, para formar un perímetro cerrado.

Por ejemplo vamos a definir un triángulo isósceles en base a sus vértices, primero definimos los puntos de los vértices y luego las caras (**Figura 2.6**).

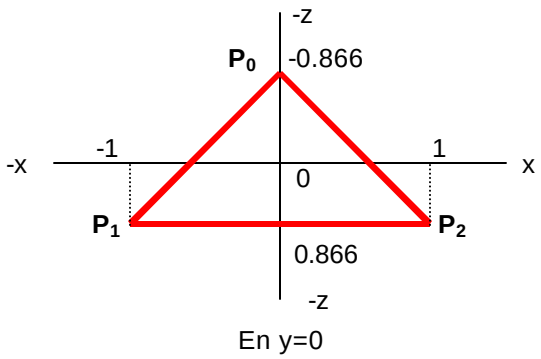


Figura 2.6: Definición de los vértices de un triángulo isósceles con *VRML*.

Con los puntos anteriores, no es difícil, definir los vértices y posteriormente las caras, en el **Programa 2.5** y en la **Figura 2.7** se muestra el uso del nodo `IndexedFaceSet.`

```
1.    # Un triángulo isósceles definido por caras y vértices
2.    #VRML V2.0 utf8
3.    Shape       {
4.          geometry IndexedFaceSet{
5.                coord Coordinate{
6.                      point [
7.                            0   0 -.866,      # Punto P0
8.                            -1  0 .866,       # Punto P1
9.                            1   0 .866        # Punto P2
10.                           ]
11.                     }

12.               coordIndex [
13.                     0,1,2,-1                # Cara iluminada arriba
14.                     ]
15.               }
16.         }
```

Programa 2.5: Nodo `IndexedFaceSet.`

El nodo `IndexedLineSet` se emplea para definir objetos mediante sus aristas para crear una figura de alambre, es decir, los puntos se enlazan mediante líneas. El nodo `IndexedLineSet` se define de la forma:

```
geometry IndexedLineSet {
            coord Coordinate{
                  # Lista de vértices
                  point [
                        x0 y0 z0,
                        x1 y1 z1,
                        .........
                        .........
                        x n-1 y n-1 zn-1,
                        xn yn zn
                        ]
                  }
```

```
        # Lista de puntos enlazados por líneas
        coordIndex [
              0,1,2,...,-1,
              3,4,5,...,-1,
              ............
              n-2,n-1,n,-1,
              ]
        }
```

De la **Figura 2.6**, podemos establecer los puntos y las líneas que se desea aparezcan en el mundo virtual, como se observa en el **Programa 2.6** y en la **Figura 2.7**.

```
1.    # Un triángulo isósceles definido por vértices y líneas
2.    #VRML V2.0 utf8
3.    Shape       {
4.          geometry IndexedLineSet {
5.                coord Coordinate{

6.                      point [
7.                            0   0 -.866,      # Punto P0
8.                            -1  0 .866,       # Punto P1
9.                            1   0 .866        # Punto P2
10.                           ]
11.                     }

12.               coordIndex [
13.                     0,1,2,0,-1              # Líneas del perímetro
14.                     ]
15.               }
16.         }
```

Programa 2.6: Nodo `IndexedLineSet`.

El nodo `PointSet` se emplea para definir puntos aislados en el espacio, que conforman un solo objeto. Por ejemplo, de la **Figura 2.6**, es posible determinar los puntos necesarios para definir un triangulo, como se muestra en el **Programa 2.7** y en la **Figura 2.7**.

```
1.    # Un triángulo isósceles definido por puntos
2.    #VRML V2.0 utf8
3.    Shape       {
4.          geometry PointSet {
5.                coord Coordinate{
6.                      point [
7.                            0   0 -.866,      # Punto P0
8.                            -1  0 .866,       # Punto P1
9.                            1   0 .866        # Punto P2
10.                           ]
11.                     }
12.               }
13.         }
```

Programa 2.7: Nodo `PointSet`.

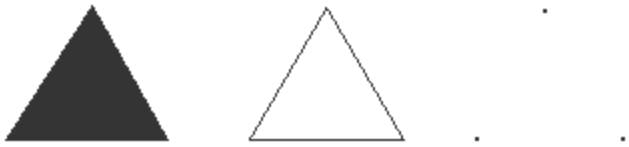


Figura 2.7: Triángulo isósceles definido como: un sólido, con líneas y con puntos.

2.6 Texto en VRML

En muchas ocasiones es necesaria la presentación de texto dentro de los ambientes virtuales, *VRML* provee una forma de agregar texto como si se tratase de objetos, a través de la primitiva `Text` del campo `geometry`, en donde también es posible definir otros parámetros de la letra como son el tipo, estilo, tamaño, etc. (**Programa 2.8** y **Figura 2.8**).

```
1.    # Texto "VRML" en un ambiente virtual
2.    #VRML V2.0 utf8
3.    Shape {
4.          geometry Text {                # Inserta Texto
5.                string [ "VRML" ]        # Texto a insertar: VRML
6.                fontStyle FontStyle {    # Define parámetros de la letra
7.                      family "ARIAL"     # Tipo de letra: ARIAL
8.                      style "BOLD"       # Estilo: negrita
9.                      size 1.5           # Define el tamaño: 1.5 Unidades
10.                     }
11.               }
12.         }
```

Programa 2.8: Código *VRML* para insertar texto.

VRML

Figura 2.8: Texto en el espacio virtual con *VRML*.

2.7 Integración VRML y Java

Indudablemente *VRML* es una poderosa herramienta de visualización, sin embargo carece del poder y versatilidad de cálculo que proporcionan los lenguajes de propósito general. Para realizar la actualización del espacio de trabajo de un robot industrial con *VRML*, es necesario crear un nuevo programa *VRML* para cada caso específico, lo cual resulta engorroso. Para resolver este problema es posible hacer que *VRML*, trabaje conjuntamente con un lenguaje de propósito general como *Java*, para que éste cree los objetos que cambian de posición, orientación y tamaño dentro de un mundo virtual fijo que puede ser por ejemplo el mismo robot (el cual permanece con su misma forma).

El lenguaje *Java* es un lenguaje de alto nivel de propósito general (parecido a *C++*), orientado a objetos, dinámico (que cambia en tiempo de ejecución), con tipos de datos duros (el lenguaje prohíbe operaciones con datos que no sean del tipo al que están destinados), tiene comprobación estática de tipos (no sucede al momento de ejecución) y es concurrente (permite varios hilos de control) [Comer 97].

Java fue desarrollado por *Sun Microsystems* en 1995 para la creación de páginas web con un contenido dinámico. Los sistemas *Java* consisten en el entorno, el lenguaje, la interfaz de programación de aplicaciones (*API, Applications Programming Interface*) de *Java* y las bibliotecas

de clases [Deitel 98]. Las *clases* son piezas predefinidas que se agrupan en categorías llamadas paquetes. *Java* esta disponible de forma gratuita a través de Internet en el sitio: `www.javasoft.com`.

En *Java* un documento dinámico se llama *applet*. Un programador crea un *applet* escribiendo un programa fuente en el lenguaje de programación *Java*, luego el compilador de *Java* traduce esté código en una representación de código de bytes (*bytecode*), el cual se carga en un servidor en la web. Cuando el visualizador solicita una *URL* (*Uniform Resource Locators*: para localizar datos en *Internet*) relacionada con la *applet*, el servidor la ejecuta. El intérprete suministra el ambiente de ejecución de *Java* y el acceso a la pantalla del usuario y a la *Internet*.

Java y *VRML* pueden interactuar, uniendo una poderosa herramienta de visualización (*VRML*) con las ventajas de un lenguaje de programación como *Java*. Para visualizar un mundo virtual en *VRML*, es necesario tener instalado en el sistema, un navegador de *Internet* capaz de aceptar *plug-in* y un visualizador de *VRML*. Para ejecutar programas en *Java* también es necesario instalar el intérprete correspondiente, un visualizador que ejecuta *Java* necesita tener un intérprete *HTML* (*HyperText Transfer Protocol*, es un protocolo el cual constituye la base de *Internet*) y un intérprete para aplicaciones elementales.

Para lograr la integración de *Java* con *VRML*, es necesario utilizar la *External Authoring Interface* (*EAI*), la cual permite la comunicación y el paso de parámetros entre mundos virtuales *VRML* y los programas escritos en *Java*. Para poder compilar un programa en *Java* que utilice clases *VRML*, es necesario copiar el paquete de *clases VRML* de la *EAI,* como librería externa de *Java*. Por ejemplo para el visor de *VRML Cosmo Player 2.1*, el archivo que contiene los paquetes de clases se llama `npcosmop21.jar` y esta ubicado en:

```
C:\Archivos de programa\CosmoSoftware\CosmoPlayer\npcosmop21.jar
```

El archivo `npcosmop21.jar` tiene que ser copiado al directorio de *Java* `C:\jdk1.3\jre\lib\ext\`, sí *Java* se encuentra instalado en `C:\jdk1.3.`

Para realizar un programa en *Java* el cual permita la interacción con un ambiente virtual en *VRML*, son necesarios al menos los siguientes archivos de programa: el programa compilado en *Java*, un intérprete en *HTML* y un ambiente virtual en *VRML*.

Por ejemplo, veamos el código en *Java* de un programa capaz de agregar y remover objetos en un mundo virtual *VRML* (una esfera en este caso). En la primera parte se importan las clases de *Java* y *VRML*, luego se definen variables y se establece la ventana de controles y dialogo con el usuario, después se definen las variables para interactuar con el navegador y visualizador de *VRML* (donde se tiene también el código *VRML* que será insertado en el mundo virtual) y al final se tiene que mediante eventos del ratón, se introducen las instrucciones de agregado y borrado de objetos en el mundo virtual (**Programa 2.9**).

```
1.  import java.awt.*;
2.  import java.applet.*;
3.  import vrml.external.field.*;
4.  import vrml.external.Node;
5.  import vrml.external.Browser;
6.  import vrml.external.exception.*;
7.  public class JavaVRML extends Applet {
8.  TextArea output = null;
9.  boolean error = false;
10. Browser browser;    /* Navegador a emplear */
11. Node root;          /* Mundo virtual que recibirá los objetos */
```

Programa 2.9: Programa `JavaVRML.java` (Parte **1**/2).

```
12. Node[] shape;       /* Nodo para VRML */
13. /* Eventos para agregar y remover un nodo en VRML */
14. EventInMFNode addChildren;   /* Define un evento de entrada a VRML */
15. EventInMFNode removeChildren;/* Define un evento de entrada a VRML */
16. /* Inicia la ventana de controles y dialogo */
17. public void init() {
18.    add(new Button("Agregar Esfera"));
19.    add(new Button("Remover Esfera"));
20.    output = new TextArea(5, 40);
21.    add(output);
22. }
23. public void start() {    /* Inicia la rutina en el navegador */
24. /* Indica el navegador en donde se despliega la información */
25. browser = (Browser) vrml.external.Browser.getBrowser(this);
26. try {
27.   /* Nombre del mundo donde se realizarán las operaciones */
28.   root = browser.getNode("ROOT");
29.   addChildren = (EventInMFNode) root.getEventIn("addChildren");
30.   removeChildren = (EventInMFNode) root.getEventIn("removeChildren");
31.   /* Código VRML a insertar en el mundo virtual */
32.   shape = browser.createVrmlFromString("Shape {\n" +
33.                  "  appearance Appearance {\n" +
34.                  "    material Material {\n" +
35.                  "      diffuseColor 0.2 0.2 0.8\n" +
36.                  "    }\n" +
37.                  "  }\n" +
38.                  "  geometry Sphere {}\n" +
39.                  "}\n");
40.   }
41. catch (InvalidNodeException e) {
42.      error = true;
43.   }
44. catch (InvalidEventInException e) {
45.      error = true;
46.   }
47. catch (InvalidVrmlException e) {
48.      error = true;
49.   }
50. }
51. /* Eventos de los botones */
52. public boolean action(Event event, Object what) {
53.   if (event.target instanceof Button) {
54.      Button b = (Button) event.target;
55.      if (b.getLabel() == "Agregar Esfera") {
56.         output.appendText("Agregando Esfera...\n");
57.         addChildren.setValue(shape);
58.      }
59.      else if (b.getLabel() == "Remover Esfera") {
60.         output.appendText("Romoviendo esfera...\n");
61.         removeChildren.setValue(shape);
62.         }
63.      }
64.   return true;
65.   }
66. }
```

Programa 2.9: Programa `JavaVRML.java` (Parte **2**/2).

Existen diversas operaciones que es posible realizar al insertar o remover objetos en un mundo virtual, como son agregar objetos, quitar objetos, cambiar la posición de los objetos, simular movimiento y llegar a la animación de los objetos.

Para agregar un objeto desde *Java* para un mundo virtual en *VRML* se emplea el nodo `addChildren.setValue()`cuyo empleo se observa en el **Programa 2.9 línea 57**, en donde se agrega el objeto ó los objetos contenidos en `shape` al mundo virtual. De igual forma es posible quitar objetos con `removeChildren.setValue()`del **Programa 2.9 línea 61**, en donde se remueven los objetos que se encuentran en `shape`.

Las operaciones de cambiar la posición de los objetos, simular movimiento y llegar a la animación de los objetos se realizan mediante una secuencia de operaciones `addChildren.setValue()` y `removeChildren.setValue()` usadas de forma alternada.

Java incluye un compilador `javac`, el cual produce un código de bytes. La entrada es un archivo fuente en *Java* y produce archivos `*.class` de las clases publicas del programa en *Java*. Para el **Programa 2.9** se tiene la instrucción `javac JavaVRML` y produce la salida `JavaVRML.class`.

Una llamada a un *applet* se realiza a través de un navegador de *Internet* por medio de un programa en *HTML*, para mostrar el *mundo virtual* en *VRML* y ejecutar el *applet* de *Java*. La primera parte del **Programa 2.10**, muestra el espacio designado para mostrar el espacio virtual de *VRML* y la segunda establece el espacio de trabajo del *applet* de *Java*.

```
1. <html>
2. <center>
3. <embed src="root.wrl" border=0 height="250" width="600">
4. <center>
5. <applet code="JavaVRML.class" width="600" height="200" mayscript>
6. </applet>
7. </html>
```

Programa 2.10: Código `HTML` para mostrar un mundo *VRML* y ejecutar un *applet de Java.*

El programa *HTML*, realiza una llamada a *VRML* y a un código de *Java*. El mundo virtual en *VRML*, puede contener cualquier cantidad de objetos o puede encontrarse vacío, solo es necesario que permita insertar el código generado desde *Java*. En el **Programa 2.11**, se muestra un ejemplo en donde es posible insertar un mundo virtual de cualquier tipo (en la línea: `DEF ROOT Group {}`), por simplicidad se tiene un mundo vacío.

```
1. #VRML V2.0 utf8
2. #Define un punto de vista
3. DEF Camera Viewpoint {position 0 0 7 }
4. DEF ROOT Group {}    # Aquí inserta Java su código VRML que genera
```

Programa 2.11: Programa `root.wrl` del mundo virtual vacío.

Figura 2.9: Interacción de *VRML* y *Java*.

Resumen

Los ambientes virtuales encuentran múltiples aplicaciones en áreas tales como aviación, arquitectura, medicina, etc. Se incluyen algunos otros desarrollos como la realidad virtual telepresencia, teleoperación, la manipulación directa remota, etc. Los ambientes virtuales dependerán de la integración adecuada de múltiples tecnologías como: pantallas, sonido, retroalimentación de fuerza, movimiento, calor, olor, etc.

La realidad virtual se puede subdividir en cooperativa y competitiva. La cooperativa se presenta cuando dos o más máquinas se enlazan para realizar una tarea conjunta, mientras que la competitiva se da cuando se desea alcanzar una meta en el menor tiempo posible.

VRML es un lenguaje de modelación para realidad virtual, con el que es posible realizar reconstrucción de objetos en tres dimensiones usando un navegador de *Internet*. Es un lenguaje sencillo, por lo que su código es regularmente pequeño, lo que permite que los programas escritos en este lenguaje viajen a través de *Internet* a gran velocidad. Presenta algunas primitivas básicas con las que es posible dibujar formas geométricas, como cubos, cilindros, esferas, conos, objetos, líneas y puntos tridimensionales. También es posible realizar transformaciones de escala, posición y orientación de los objetos o un conjunto de ellos.

VRML permite la interacción con otros lenguajes de programación como *Java*, a través de la *External Authoring Interface*. Por lo que amplia sus posibilidades de uso añadiendo el poder de un lenguaje de programación, a la visualización y reconstrucción tridimensional.

Capítulo 3: Visión Artificial

La importancia de dotar a las máquinas de visión consiste en ampliar sus posibilidades de interacción con el mundo real. Con un sentido como el de la vista, las máquinas podrían *ver* para realizar tareas análogas a las que realiza un ser humano (ensamblar, buscar, moverse, inspeccionar, etc.).

Un sistema de *visión artificial* para una máquina consiste de funciones tales como obtener, caracterizar e interpretar información a partir de imágenes provenientes del mundo real; es decir, permite obtener una descripción simbólica del mundo real a partir de imágenes. La visión artificial encuentra múltiples aplicaciones en campos tales como:

- *Industria*.- clasificación, búsqueda, ordenamiento, control de calidad, etc.
- *Medicina.*- análisis de enfermedades, operaciones con endoscopios, ayuda al diagnostico, etc.
- *Biomedicina.*- análisis de imágenes, sangre y DNA, etc.
- *Milicia.*- detección y seguimiento de objetivos, análisis del terreno, armas inteligentes, etc.
- *Robótica.*- guía de robots móviles, y de brazos de robots, etc.
- *Agricultura.*- análisis de plantaciones, análisis de imágenes de satélite, etc.
- *Identificación.*- identificación automática de huellas dactilares, reconocimiento de caras, etc.
- *Seguridad.*- para avisos de alarma al detectar o reconocer a un sujeto, etc.
- *Redes de comunicación.*- compresión de imágenes, teleconferencia, etc.
- *Control de calidad.*- verificación de etiquetas, inspección de contenedores, motores, cristales, soldaduras, circuitos impresos, etc.
- *Realidad virtual.*- para creación de entornos virtuales a partir de imágenes.
- *Astronomía.*- búsqueda, reconocimiento y seguimiento de objetos celestes.

En este capítulo se presentan algunos conceptos básicos de visión artificial. Se describen las tareas necesarias para llevarla a cabo (procesamiento de imágenes y reconocimiento), y se detallan algunos de los métodos empleados en *visión artificial* para obtener alguna descripción del mundo que pueda ser interpretada por una máquina. Se muestra una forma de realizar el reconocimiento óptico de caracteres u *OCR* por sus siglas en inglés. Luego se realiza una descripción de los métodos para la calibración de una cámara, para que el modelo de la imagen coincida con el que una máquina ó un robot reconoce.

3.1 Visión artificial

Existe una gran variedad de definiciones de lo que es la *visión artificial* [Escalera 01]. Las que mejor la describen en nuestra opinión son las siguientes:

- La *visión artificial* es un proceso de tratamiento de imágenes que permite obtener la descripción simbólica del mundo real a partir de imágenes.

- La *visión artificial* es la ciencia que estudia, utilizando computadoras, los procesos de obtención, caracterización e interpretación de la información procedentes de imágenes tomadas de un mundo tridimensional.

Un sistema de *visión artificial* consiste en una serie de tareas que se muestran en la **Figura 3.1** [Schildt 87, Jackson 85].

La *adquisición de imágenes* es el conjunto de operaciones que se efectúan para transformar la información luminosa de una escena en una señal digital. Se deben de tener en

cuenta los factores siguientes: iluminación, tipo de cámaras empleadas y el método de digitalización usado.

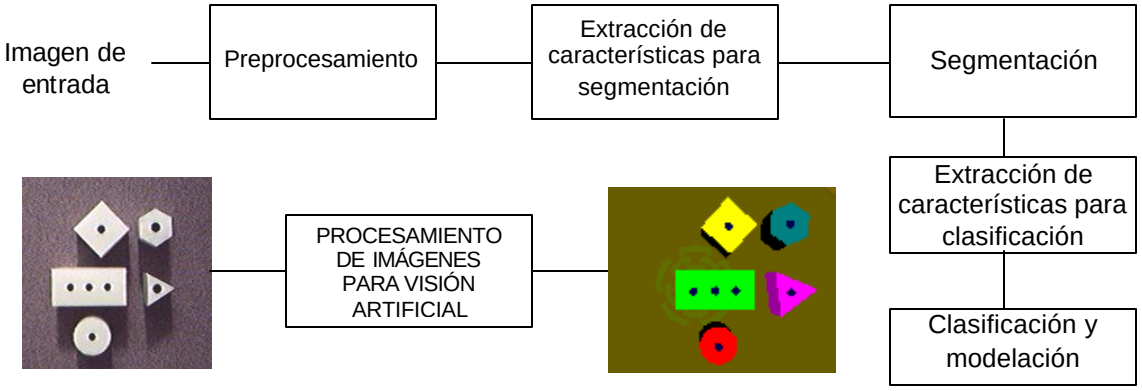


Figura 3.1: Esquema general de procesamiento de imágenes en visión artificial.

El *preprocesamiento* realiza una mejora de la imagen, modificando por ejemplo: el nivel de gris, el contraste, eliminación de ruido, realce de características, etc. La *segmentación* divide a la escena en objetos por métodos de discontinuidad, similitud, etc. La *clasificación* extrae y organiza las características de los objetos para su reconocimiento. El *reconocimiento* es la identificación de cada objeto en la escena mediante una etiqueta y la *modelación* puede ser solo el etiquetado o emplear algunos modelos geométricos como los generados a través de *VRML* para *modelación* y *reconstrucción tridimensional.*

3.2 Adquisición de la imagen

La *adquisición de imágenes* se divide en tres tareas: la formación de imágenes, la captura de imágenes y la digitalización de las mismas.

La *formación de una imagen* resulta de proyectar la información visual contenida o reflejada por los objetos que constituyen un medio ambiente sobre un plano bidimensional [Ibarra 98]. Las imágenes pueden ser producidas y captadas de forma óptica. Sin embargo, es posible obtener imágenes del mundo a partir de cualquier tipo de estimulo físico como: ultrasonido, rayos infrarrojos, luz ultravioleta, rayos X, luz láser, etc. Podemos considerar a una imagen como un arreglo bidimensional de muestras de energía. En la formación de una imagen también juegan un papel importante los sistemas de *iluminación*. Una *iluminación* adecuada permite resaltar características importantes de los objetos, ver objetos transparentes, aumentar el contraste entre el fondo y el objeto, etc. Una iluminación adecuada puede simplificar considerablemente los algoritmos de detección y extracción de características de los objetos en la escena [Escalera 01]. Los puntos que hay que tomar en cuenta para el sistema de iluminación son:

- *Precisión del sistema*.- tiene que ver con el tamaño y la precisión de las características que se desean resaltar: superficies lisas, rugosas, reflejos direccionales, color, etc.
- *Tipo de escena*.- se debe de considerar el tamaño de los objetos en la escena.
- *Profundidad de enfoque*.- es la distancia entre la cámara y el objeto.
- *Consideraciones físicas*.- se consideran las limitaciones térmicas de los objetos, el espacio para el sistema de iluminación, etc.

Se tienen además una gran diversidad de fuentes de luz [Viakon 00], como son:

- *Incandescente*.- es una iluminación de bajo costo, larga vida y ajustable por medios simples, sin embargo genera mucho calor, puede generar parpadeo, etc. Se emplea en sistemas donde se requiere iluminación frontal.

- *Fluorescente*.- son de alta eficiencia y mínima disipación térmica, pueden tener diferentes colores, tamaños y formas, emiten luz difusa y trabajan a altas frecuencias, pero no operan a bajas temperaturas y su vida útil es menor que las incandescentes. Se utilizan para iluminar escenas grandes y objetos reflectivos.
- *Diodos Emisores de Luz (LED).*- la luz que emiten es monocromática [Willians 89], tienen una vida útil larga y presentan un bajo consumo de energía. Sin embargo, presentan una emisión de baja intensidad. Se utilizan para tomas en contra luz y detección de objetos pequeños.
- *Láser*.- es una fuente de luz altamente direccional, por lo que permite la creación de diferentes patrones de luz, pero su costo es alto. Encuentra múltiples aplicaciones en medidas de objetos como: curvas tridimensionales, profundidad, espesor, etc.
- *Halógenas*.- tienen gran intensidad por un tiempo breve. Permiten el estudio de objetos en movimiento. Por otro lado necesitan fuentes especiales de iluminación y pierden su intensidad en un tiempo corto.

También existe una gran variedad de técnicas de iluminación, algunas de ellas se muestran a continuación:

- *Iluminación direccional*.- es cuando la luz se dirige en una sola dirección (**Figura 3.2**). Se pueden tener efectos como el de brillo intenso, campo oscuro, iluminación perpendicular, iluminación oblicua, iluminación colimada, etc.

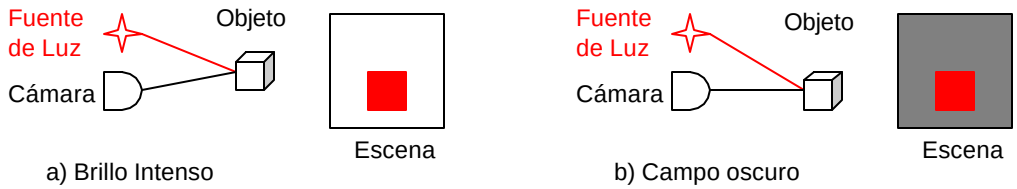


Figura 3.2: Iluminación direccional.

- *Iluminación difusa*.- se trata de hacer llegar la luz al objeto en todas las direcciones (**Figura 3.3**). Por ejemplo las fuentes que se pueden utilizar son: anillo de luz, difusores esféricos, anillos de fibra óptica y anillos de diodos *LED*.

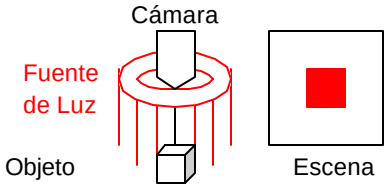


Figura 3.3: Iluminación difusa mediante un anillo de luz.

- *Iluminación a contraluz*.- se tiene cuando los objetos se colocan frente a la cámara y la fuente de luz se encuentra atrás de los objetos (**Figura 3.4**).

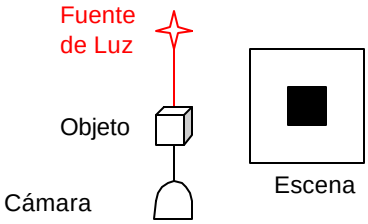


Figura 3.4: Iluminación a contraluz.

- *Iluminación estructurada*.- se basa en la iluminación con puntos, líneas o rejillas de la escena a ser procesada (ver **Figura 3.5** y el **Apéndice B: Iluminación Estructurada**). Hace posible que el mundo observado sea bidimensional, igual que la imagen; por lo que no se pierde información, ya que a cada punto en el mundo real le corresponde un punto en la imagen captada.

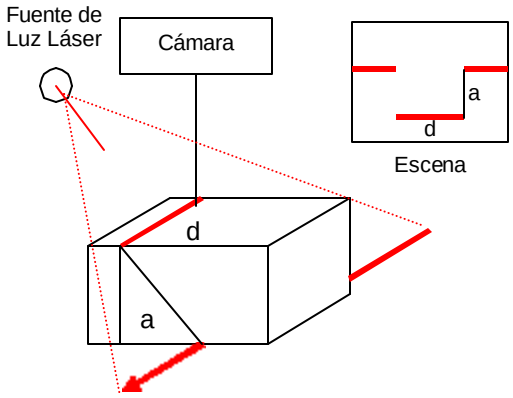


Figura 3.5: Iluminación estructurada.

- *Iluminación coaxial*.- se presenta cuando la iluminación va en la misma dirección que la luz que entra por la cámara reflejada por los objetos. Produce imágenes libres de sombras. Se emplea para iluminar agujeros interiores y/o evitar reflejos.

- *Luz polarizada*.- un rayo luminoso normal gira perpendicularmente a la dirección en la que se propaga. Cuando se elimina el giro, la luz oscila en una sola dirección, entonces se tiene *luz polarizada*. Algunos materiales transparentes muestran la capacidad de desviar estos rayos, por lo que pueden ser detectados iluminándolos de ésta forma, pues al ser iluminados con luz normal, son transparentes.

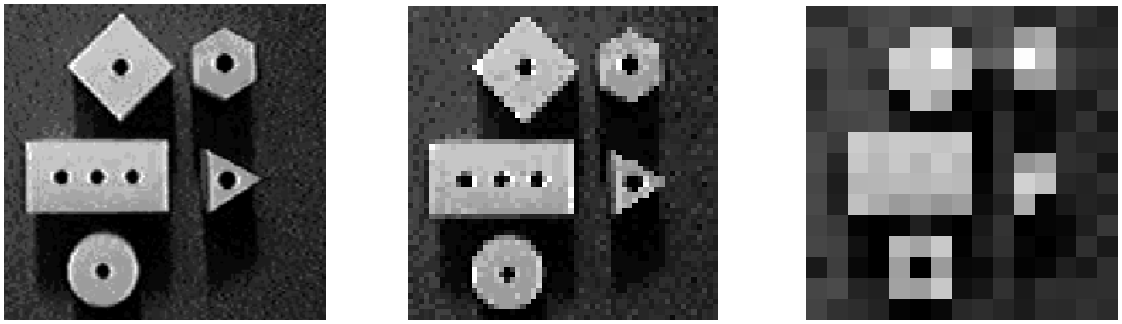
La *captura de imágenes* consiste en transformar una imagen óptica en una señal eléctrica que represente la información visual de la escena. Básicamente existen tres tipos de captores de imagen dependiendo del tipo de propiedad de los materiales que se modifique al incidir la luz sobre ellos: fotovoltaicos, fotoemisivos y fotoconductivos. Los dispositivos relacionados con cada unos de estos transductores se describen en detalle en el **Apéndice A: Sistemas de Adquisición de Imágenes**.

La *digitalización de imágenes* es el proceso de cuantificación de una imagen representándola en forma digital [Diccionario 90]. Consiste en traducir a un formato numérico una imagen para que pueda ser procesada por el sistema de visión.

La digitalización de una imagen se puede hacer tomando muestras en el espacio de forma continua por lo que a este proceso se le conoce como *muestreo espacial* (**Figura 3.6**), entre mayor sea en número de muestras mejor calidad tendrá la imagen resultante. Cada muestra es codificada de acuerdo a la cantidad de bits (q) que proporcione un convertidor analógico/digital, cada una de estas muestras es conocida como *píxel* (*elemento de imagen*), por lo que el número de *niveles de brillo* será $L=2^q$ (**Figura 3.7**). Si $q=1$, la imagen solo tiene dos niveles de brillo, por lo que se llama *imagen binaria*. Una imagen será mejor, conforme aumente su número de bits, sin embargo la cantidad de memoria empleada y el tiempo para procesarla se incrementarán. Por lo que se deberá evaluar la calidad de la información que requiere el sistema de visión para que sea lo suficientemente precisa para llevar a cabo el proceso, sin comprometer un excesivo costo en cómputo y de espacio utilizado para su almacenamiento.

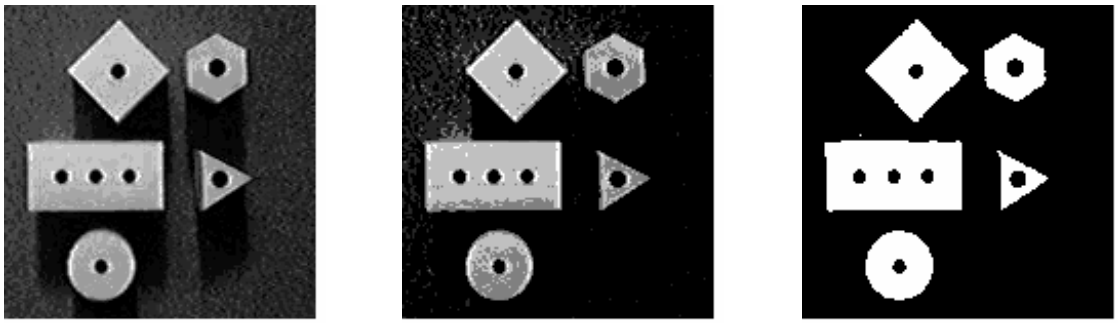
En las **Figuras 3.6** y **3.7** se observan los resultados al aplicar diferentes muestreos espaciales y niveles de brillo. Se observa que existen algunos píxeles los cuales se relacionan con

otros al compartir características comunes (por ejemplo el brillo). De las relaciones entre píxeles surgen conceptos como los de conectividad y vecindad.



256x256 píxeles | 64x64 píxeles | 15x15 píxeles

Figura 3.6: Efecto del muestro espacial en una imagen.



256 niveles de gris | 16 niveles de gris | 2 niveles de gris (Imagen Binaria)

Figura 3.7: Diferentes niveles de brillo para una imagen.

La *conectividad* describe como se enlazan los píxeles en una imagen, es decir, como se relacionan entre sí un conjunto de estos para determinar si son o no *vecinos* [Foley 96]. Dependiendo de cómo se consideren los píxeles como vecinos de los que se encuentran a su alrededor existen diferentes formas de conectividad:

La conectividad *4-conectado* para un píxel (x,y) es cuando se considera como vecinos todos aquellos píxeles que comparten un borde (**Figura 3.8a**), es decir, los cuatro píxeles: (x-1,y), (x+1,y), (x,y-1) y (x,y+1).

La conectividad *8-conectado* para un píxel (x,y) es cuando se considera como vecinos todos aquellos píxeles que comparten un borde o una esquina (**Figura 3.8b**), es decir, los ocho píxeles: (x-1,y), (x+1,y), (x,y-1), (x,y+1), (x-1,y-1), (x+1,y-1), (x-1,y+1) y (x+1,y+1).

La conectividad *6-conectado izquierdo* para un píxel (x,y) es cuando se considera como vecinos todos aquellos píxeles que comparten un borde o una esquina en la dirección *Noroeste* y *Sudeste* (**Figura 3.8c**), es decir, los seis píxeles: (x,y-1), (x-1,y-1), (x-1,y), (x+1,y), (x+1,y+1) y (x,y+1).

La conectividad *6-conectado derecho* para un píxel (x,y) es cuando se considera como vecinos todos aquellos píxeles que comparten un borde o una esquina en la dirección *Noreste* y *Sudoeste* (**Figura 3.8d**), es decir, los seis píxeles: (x,y-1), (x+1,y-1), (x+1,y), (x-1,y), (x-1,y+1) y (x,y+1).

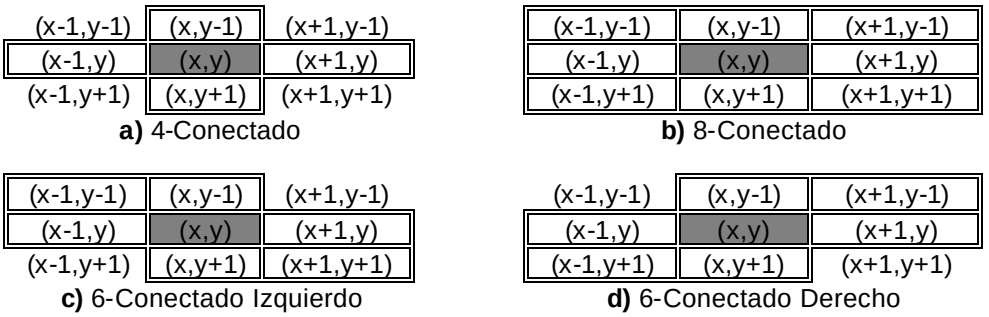


Figura 3.8: Conectividad entre píxeles.

En una imagen un conjunto de píxeles conectados de alguna manera, forman un objeto. Por lo que un objeto se encuentra formado por píxeles que en conjunto forman una curva cerrada, la cual se encuentra separada del resto de la imagen. El teorema de *Jordan* establece que para una curva cerrada simple, se puede segmentar en dos regiones, disjuntas, es decir, no conexas, dividiendo a la imagen en dos partes: la interior y la exterior (**Figura 3.9**).

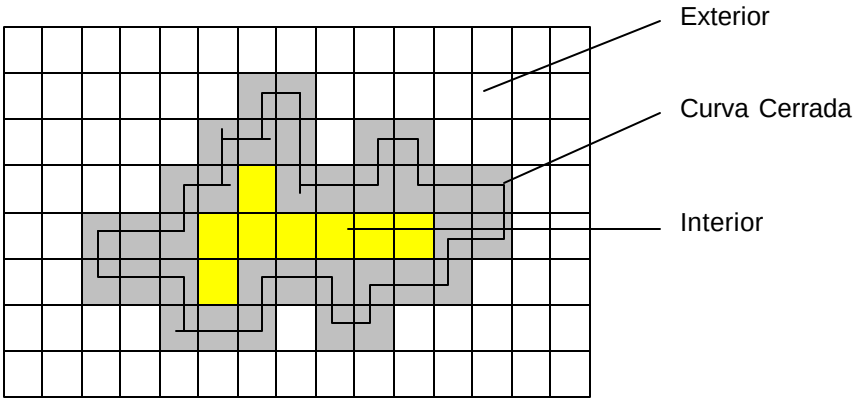


Figura 3.9: Teorema de *Jordan*.

Un píxel puede ser considerado *4-conectado* u *8-conectado* cuando tiene las mismas propiedades que alguno de los cuatro u ocho vecinos más próximos. Pero se puede ver que se llegará a una paradoja cuando queremos aplicar alguna de los dos tipos de conexión *4-conectado* u *8-conectado*. Este caso es conocido como la *paradoja de conectividad.*

Si suponemos un contorno *4-conectado* como el de la **Figura 3.10**, vemos que los segmentos *1, 2, 3* y *4* serían clasificados como disjuntos. Esto no sucede si aplicamos *8-conectado*. Sin embargo, también serán conectados los píxeles del agujero (*píxel B*) y los del exterior del anillo (*píxel A*), esta característica es la que se conoce como *paradoja de conectividad.*

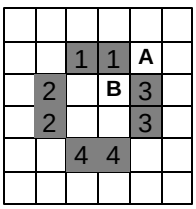


Figura 3.10: Paradoja de conectividad.

Una solución a la paradoja de conectividad suele ser considerar *8-conectado* para el objeto (borde) y *4-conectado* para el fondo. Otra alternativa es usar *6-conectado*.

3.3 Preprocesamiento de una imagen

Una vez que una imagen a sido digitalizada, está puede ser almacenada en la memoria de la computadora, pero para poder ser empleada por un programa de reconocimiento se debe de llevar a cabo un preprocesamiento de la imagen.

El *preprocesamiento* comprende aquellos algoritmos cuya finalidad es conseguir una mejora en la apariencia de la imagen original [Escalera 01]. Esa mejora consiste en resaltar determinadas características de la imagen o en su defecto eliminar aquellas que la afectan para los procesos posteriores. Básicamente existen dos grupos de algoritmos, según se trabaje en el dominio del *espacio* o de la *frecuencia*. En el primero se modifican directamente los píxeles de la imagen. Para el caso de la frecuencia, se emplean métodos basados en la *transformada de Fourier*.

El preprocesamiento de una imagen trata que esta sea lo más ideal posible, en cuanto a dos aspectos:

1.- Iluminación uniforme, este punto implica que la escena se encuentre con una iluminación uniforme, o que en su defecto cada objeto, mantenga una iluminación uniforme en su superficie.
2.- Que la ganancia entre la luz de entrada y la imagen resultante sea lineal, de esta forma la imagen obtenida no deberá de deformarse por efecto del preprocesamiento de la imagen, con el fin de que los objetos presentes en la imagen no se pierdan.

Algunos de los métodos empleados para el preprocesamiento de imágenes en el dominio del espacio son: la modificación del contraste, la modificación del histograma, etc.

3.4 Extracción de características para la segmentación

Cuando una imagen ha sido procesada se encuentra en disposición de detectar los objetos presentes en ella. Se buscan entonces las características que definen a cada objeto. Se trata de detectar contornos, esquinas, etc. Existe una gran diversidad de métodos para *extraer contornos* de una imagen. La mayoría de ellos utilizan máscaras, las cuales son desplazadas sobre toda la imagen para obtener el contorno, encontrar esquinas, etc. Las máscaras se pueden encontrar mediante el cálculo de la derivada de la imagen. Existe una gran diversidad de métodos para la detección de bordes como el método de gradiente, de aproximación de Roberts, de aproximación de Prewitt, de aproximación de Sobel, etc., todos empleados en imágenes digitales o discretas.

Antes de entrar a la descripción de los métodos de extracción de contornos para imágenes discretas, realizaremos una breve descripción del proceso empleado en imágenes analógicas para la extracción de contornos.

3.4.1 Identificación de contornos en una imagen analógica mediante la derivada

La extracción de contornos de una imagen (señal) analógica, es posible mediante el cálculo de la derivada de cada una de las líneas que conforman dicha imagen. Una imagen analógica se encuentra constituida por una serie de líneas, cada una de estas líneas puede ser tomada como una señal en función del tiempo, la derivada de está función nos identifica los cambios en la intensidad luminosa de la imagen, dichos cambios corresponden a los bordes de los objetos en la imagen. Un ejemplo de cómo se identifican los bordes en una imagen analógica, lo podemos observar en la **Figura 3.11**, en donde la imagen original ha sido dividida en cinco líneas, en cada una de las cuales se aplicará una derivada. Si se colocan los cambios en una nueva imagen de salida, se observa que el resultado obtenido son los contornos laterales del objeto de entrada, esto es debido a que la derivada se aplicó solo en la dirección horizontal, por lo que si se requieren los bordes verticales, bastará aplicar la derivada sobre líneas verticales.

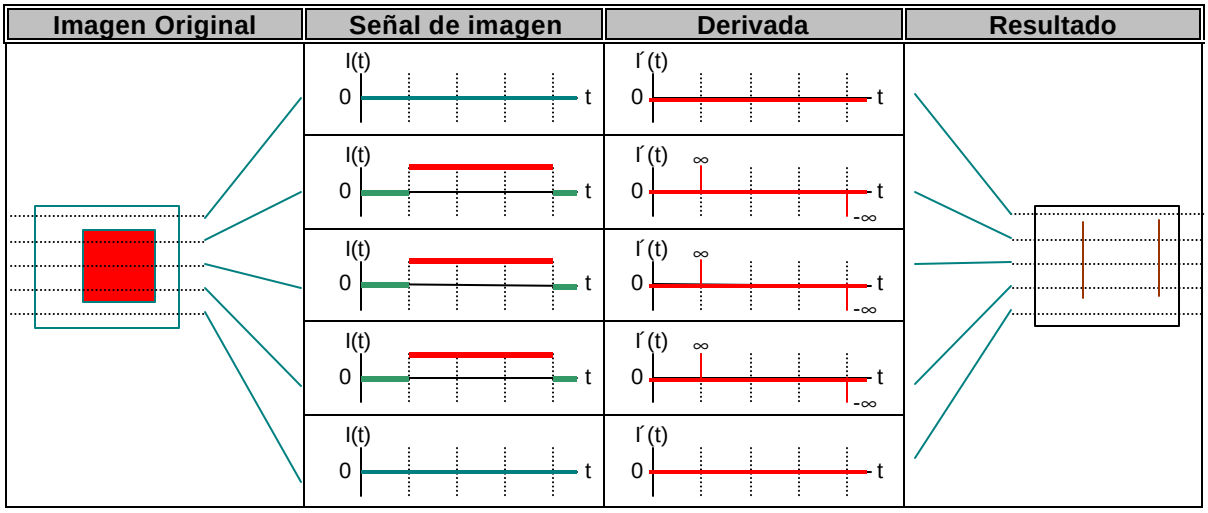


Figura 3.11: Proceso de extracción de contornos mediante derivada.

3.4.2 Extracción de contornos de una imagen digital mediante derivada

Para imágenes discretas, es posible emplear el método de la derivada para identificar contornos. Una *imagen discreta f(x,y),* se encuentra formada por una serie de puntos *(x,y),* que se encuentran distribuidos en el espacio bidimensional, es decir, forman una matriz como la que se observa en la **Figura 3.12**.

f(5,5)	x_1	x_2	x_3	x_4	x_5
y_1	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)
y_2	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)
y_3	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)
y_4	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)
y_5	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)

Figura 3.12: Imagen discreta.

Para la detección de contornos en una imagen discreta, es necesario realizar una aproximación del concepto de derivada para *espacios discretos* [Chapra 96, CVPR 97, Ibarra 98], esto se realiza a través del *gradiente de luminosidad* de una imagen *f(x,y)* en el punto *(x,y)* definida de la forma:

Gradiente de luminosidad $= \nabla f(x,y) = [\partial f(x,y)/\partial x, \partial f(x,y)/\partial y]$

La *magnitud del gradiente de luminosidad* se obtiene de:

Magnitud del gradiente de luminosidad $= |\nabla f(x,y)| = \sqrt{\{(\partial f(x,y)/\partial x)^2 + (\partial f(x,y)/\partial y)^2\}}$

Para simplificar el costo computacional que involucra el cálculo, la *Magnitud del gradiente de luminosidad* se obtiene por la **Ec. 3.1**.

$|\tilde{N}f(x,y)| = |\P f(x,y)/\P x| + |\P f(x,y)/\P y|$ ...(Ec. 3.1)

Por otro lado conocemos que la *aproximación de la derivada* por la *serie de Taylor* para el punto x_i está dada por:

$f(x_{i+1}) = f(x_i) + hf'(x_i) + h^2f''(x_i)/2 + h^3f'''(x_i)/6 + h^4f''''(x_i)/24 + \ldots\ldots$

Despejando el valor de la derivada tenemos la **Ec. 3.2**.

$f'(x_i) = [f(x_{i+1}) - f(x_i)]/h - hf''(x_i)/2 - h^2f'''(x_i)/6 - h^3f''''(x_i)/24 -\ldots\ldots$ ...(Ec. 3.2)

1) Considerando solo el primer termino de la aproximación de la derivada por la serie de *Taylor*, podemos hallar los valores aproximados para cada una de las derivadas parciales. De la **Ec. 3.2** tenemos que:

$\partial f(x,y)/\partial x = [f(x+1,y) - f(x,y)]/h$
$\partial f(x,y)/\partial y = [f(x,y+1) - f(x,y)]/h$

Como la distancia entre píxeles de una imagen es de *un píxel*, se considera *h=1*, para las ecuaciones anteriores se obtienen las **Ecs. 3.3** y **3.4**.

$\P f(x,y)/\P x = f(x+1,y) - f(x,y)$ ...(Ec. 3.3)
$\P f(x,y)/\P y = f(x,y+1) - f(x,y)$ ...(Ec. 3.4)

Sustituyendo las **Ecs. 3.3** y **3.4** en la **Ec. 3.1** tenemos:

$|\nabla f(x,y)| = |\partial f(x,y)/\partial x| + |\partial f(x,y)/\partial y| = f(x+1,y) - f(x,y) + f(x,y+1) - f(x,y)$

$|\tilde{N}f(x,y)| = f(x+1,y) + f(x,y+1) - 2*f(x,y)$

Siendo así, la máscara queda como (**Figura 3.14** y **Figura 3.15b**):

–2*(x,y)	1*(x+1,y)
1*(x,y+1)	0*(x+1,y+1)

$\Rightarrow$

–2	1
1	0

2) Considerando hasta el segundo termino de la aproximación de la derivada por la serie de *Taylor*, podemos hallar los valores aproximados para cada una de las derivadas parciales. De la **Ec. 3.2** tenemos que:

$\partial f(x,y)/\partial x = [f(x+1,y) - f(x,y)]/h - hf''(x,y)/2$
$\partial f(x,y)/\partial y = [f(x,y+1) - f(x,y)]/h - hf''(x,y)/2$

Como la distancia entre píxeles de una imagen es de *un píxel*, se considera *h=1*, para las ecuaciones anteriores se obtienen las **Ecs. 3.5** y **3.6**.

$\P f(x,y)/\P x = f(x+1,y) - f(x,y) - f''(x,y)/2$ ...(Ec. 3.5)
$\P f(x,y)/\P y = f(x,y+1) - f(x,y) - f''(x,y)/2$ ...(Ec. 3.6)

Se conoce también que la aproximación para la segunda derivada está dada por:

$f''(xi) = [f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)] / h^2 = f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)$

Aplicando las **Ecs. 3.5** y **3.6**, en la segunda derivada se obtienen las **Ecs. 3.7** y **3.8**.

$\P^2 f(x,y)/\P x = f(x+2,y) - 2f(x+1,y) + f(x,y)$ ...(Ec. 3.7)
$\P^2 f(x,y)/\P y = f(x,y+2) - 2f(x,y+1) + f(x,y)$ ...(Ec. 3.8)

Sustituyendo las **Ecs. 3.7** y **3.8** en las **Ecs. 3.5** y **3.6** obtenemos las **Ecs. 3.9** y **3.10**.

$\P f(x,y)/\P x = f(x+1,y) - f(x,y) - f(x+2,y)/2 + f(x+1,y) - f(x,y)/2$ ...(Ec. 3.9)
$\P f(x,y)/\P y = f(x,y+1) - f(x,y) - f(x,y+2)/2 + f(x,y+1) - f(x,y)/2$ ...(Ec. 3.10)

Sustituyendo las **Ecs. 3.9** y **3.10** en la **Ec. 3.1** tenemos:

$|\nabla f(x,y)| = |\partial f(x,y)/\partial x| + |\partial f(x,y)/\partial y|$

$|\nabla f(x,y)| = f(x+1,y) - f(x,y) - f(x+2,y)/2 + f(x+1,y) - f(x,y)/2 + f(x,y+1) - f(x,y) - f(x,y+2)/2 + f(x,y+1) - f(x,y)/2$

$|\tilde{N}f(x,y)| = -3*f(x,y) + 2*f(x+1,y) - f(x+2,y)/2 + 2*f(x,y+1) - f(x,y+2)/2$

Siendo así la máscara queda como (**Figura 3.15c**):

–3*(x,y)	2*(x+1,y)	-0.5*(x+2,y)
2*(x,y+1)	0*(x+1,y+1)	0*(x+2,y+1)
–0.5*(x,y+2)	0*(x+1,y+2)	0*(x+2,y+2)

$\Rightarrow$

–3	2	–0.5
2	0	0
–0.5	0	0

3) Considerando hasta el tercer termino de la aproximación de la derivada por la serie de *Taylor*, podemos hallar los valores aproximados para cada una de las derivadas parciales. De la **Ec. 3.2** tenemos que:

$\partial f(x,y)/\partial x = [f(x+1,y) - f(x,y)]/h - hf''(x,y)/2 - h^2 f'''(x,y)/6$
$\partial f(x,y)/\partial y = [f(x,y+1) - f(x,y)]/h - hf''(x,y)/2 - h^2 f'''(x,y)/6$

Como la distancia entre píxeles de una imagen es de *un píxel*, se considera *h=1*, para las ecuaciones anteriores se obtienen las **Ecs. 3.11** y **3.12**.

$\partial f(x,y)/\partial x = f(x+1,y) - f(x,y) - f''(x,y)/2 - f'''(x,y)/6$ ...(Ec. 3.11)
$\partial f(x,y)/\partial y = f(x,y+1) - f(x,y) - f''(x,y)/2 - f'''(x,y)/6$ ...(Ec. 3.12)

Se conoce también que la aproximación para la tercera derivada está dada por:

$f''(xi)=[f(x_{i+3}) - 3f(x_{i+2}) + 3f(x_{i+1}) - f(x_i)] / h^3 = f(x_{i+3}) - 3f(x_{i+2}) + 3f(x_{i+1}) - f(x_i)$

Aplicando las **Ecs. 3.11** y **3.12** en la tercera derivada se obtienen las **Ecs. 3.13** y **3.14**.

$\partial^3 f(x,y)/\partial x = f(x+3,y) - 3f(x+2,y) + 3f(x+1,y) - f(x,y)$ ...(Ec. 3.13)
$\partial^3 f(x,y)/\partial y = f(x,y+3) - 3f(x,y+2) + 3f(x,y+1) - f(x,y)$ ...(Ec. 3.14)

Sustituyendo las **Ecs. 3.13**, **3.14**, **3.7** y **3.8** en las **Ecs. 3.11** y **3.12** obtenemos la **Ecs. 3.15** y **3.16**.

$\partial f(x,y)/\partial x =$ $f(x+1,y) - f(x,y) - f(x+2,y)/2 + f(x+1,y) - f(x,y)/2 - f(x+3,y)/6 + f(x+2,y)/2 - f(x+1,y)/2 + f(x,y)/6$ ...(Ec. 3.15)

$\partial f(x,y)/\partial y =$ $f(x,y+1) - f(x,y) - f(x,y+2)/2 + f(x,y+1) - f(x,y)/2 - f(x,y+3)/6 + f(x,y+2)/2 - f(x,y+1)/2 + f(x,y)/6$ ...(Ec. 3.16)

Sustituyendo las **Ecs. 3.15** y **3.16** en la **Ec. 3.1** tenemos:

$|\nabla f(x,y)| = |\partial f(x,y)/\partial x| + |\partial f(x,y)/\partial y|$

$|\nabla f(x,y)| = f(x+1,y) - f(x,y) - f(x+2,y)/2 + f(x+1,y) - f(x,y)/2 - f(x+3,y)/6 + f(x+2,y)/2 - f(x+1,y)/2 + f(x,y)/6 + f(x,y+1) - f(x,y) - f(x,y+2)/2 + f(x,y+1) - f(x,y)/2 - f(x,y+3)/6 + f(x,y+2)/2 - f(x,y+1)/2 + f(x,y)/6$

$|\tilde{N}f(x,y)| = -8/3*f(x,y) + 3/2*f(x+1,y) + 3/2*f(x,y+1) - f(x+3,y)/6 - f(x,y+3)/6$

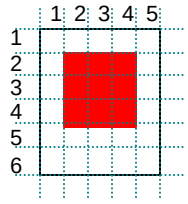
Siendo así la máscara queda como (**Figura 3.15d**):

–8/3*(x,y)	3/2*(x+1,y)	0*(x+2,y)	1/6*(x+3,y)
3/2*(x,y+1)	0*(x+1,y+1)	0*(x+2,y+1)	0*(x+3,y+1)
0*(x,y+2)	0*(x+1,y+2)	0*(x+2,y+2)	0*(x+3,y+2)
1/6*(x,y+3)	0*(x+1,y+3)	0*(x+2,y+3)	0*(x+3,y+3)

⇒

–8/3	3/2	0	–1/6
3/2	0	0	0
0	0	0	0
–1/6	0	0	0

Podemos seguir aproximando cada vez más la derivada, tomando un mayor número de términos de la serie de *Taylor*, sin embargo, se obtienen buenas aproximaciones tomando solo tres términos de esta. Veamos ahora como opera la máscara obtenida para el primer término de la serie de *Taylor* al aplicarla sobre una imagen discreta (**Figura 3.13**).



⇒

f(5,6)	x_1	x_2	x_3	x_4	x_5
y_1	0	0	0	0	0
y_2	0	1	1	1	0
y_3	0	1	1	1	0
y_4	0	1	1	1	0
y_5	0	0	0	0	0
y_6	0	0	0	0	0

a) Imagen Analógica **b)** Imagen Digital

Figura 3.13: Imagen analógica a digital o discreta.

En la **Figura 3.14** se observan los pasos a seguir para aplicar la máscara. La operación se realiza para todos los píxeles de la imagen menos para los píxeles del borde, en donde no es posible aplicar la convolución al no solaparse totalmente la máscara y la imagen. Así siempre que se realice una convolución de una imagen con una máscara se obtendrá una imagen más pequeña de dimensiones $\mathbf{f_f(x,y) = (N-(n-1))x(M-(m-1))}$ donde *N* y *M* son las dimensiones de la imagen original; y, *n* y *m* las dimensiones de la máscara. Para el ejemplo de la **Figura 3.14** se tiene:

Dimensiones de la imagen resultante = $f_f(x,y) = (5-(2-1))x(6-(2-1)) = 4x5$

–2	1
1	0

a) Máscara a aplicar

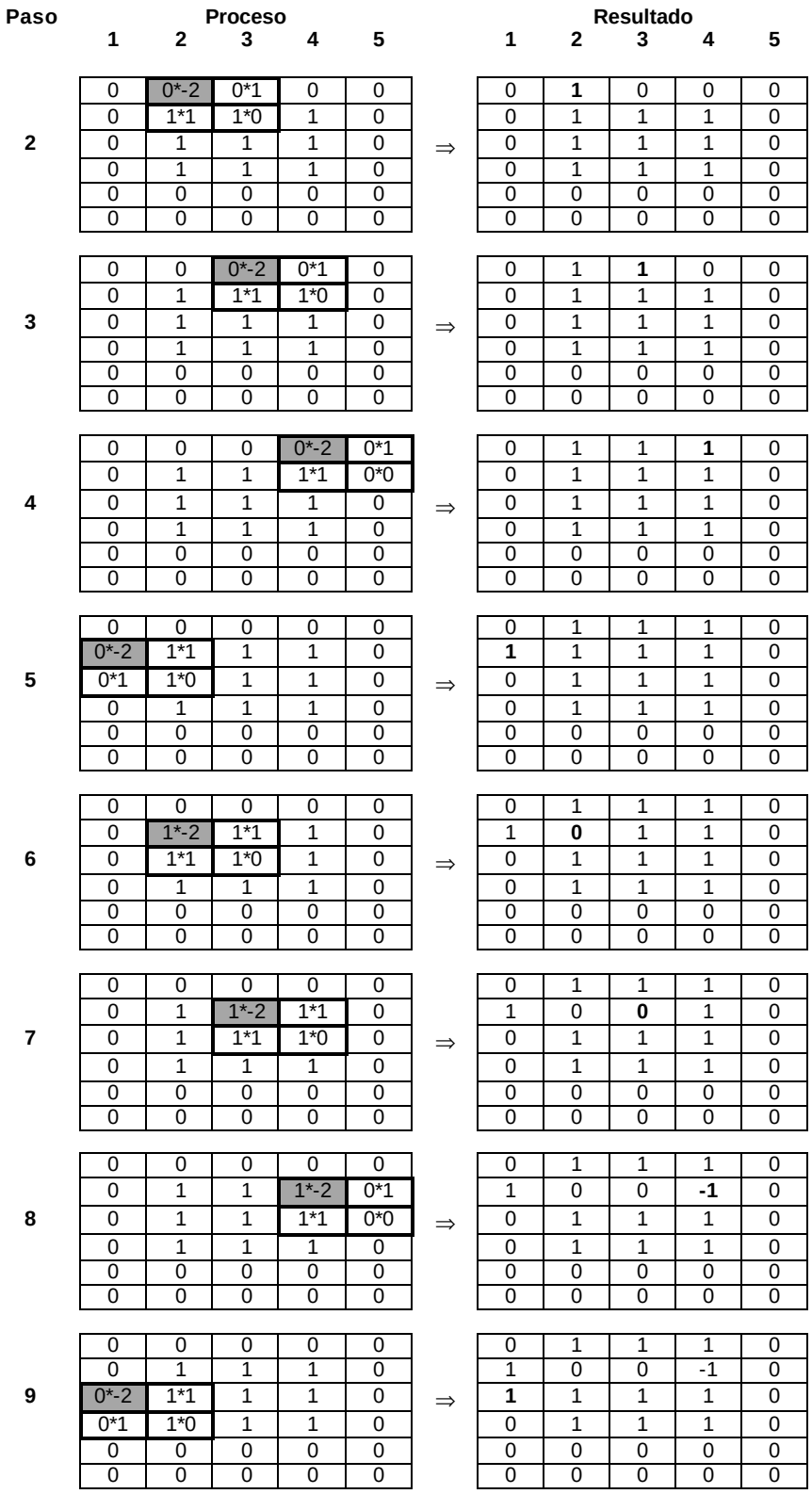
f(5,6)	x_1	x_2	x_3	x_4	x_5
y_1	0	0	0	0	0
y_2	0	1	1	1	0
y_3	0	1	1	1	0
y_4	0	1	1	1	0
y_5	0	0	0	0	0
y_6	0	0	0	0	0

b) Imagen digital original

Paso	**Proceso** 1	2	3	4	5		**Resultado** 1	2	3	4	5
	0*-2	0*1	0	0	0		**0**	0	0	0	0
	0*1	1*0	1	1	0		0	1	1	1	0
1	0	1	1	1	0	⇒	0	1	1	1	0
	0	1	1	1	0		0	1	1	1	0
	0	0	0	0	0		0	0	0	0	0
	0	0	0	0	0		0	0	0	0	0

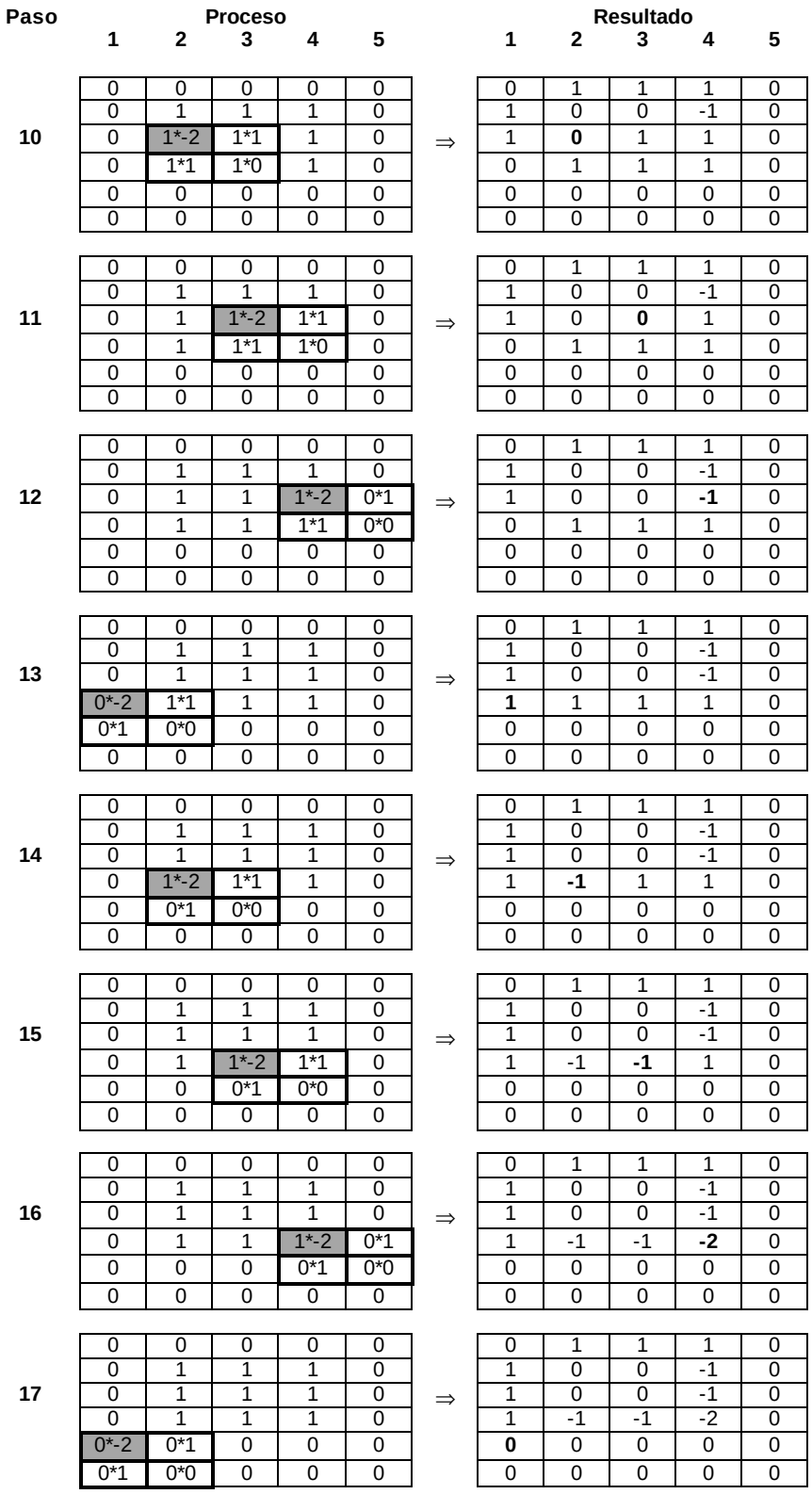
c) Procedimiento de aplicación de máscara

Figura 3.14: Aplicación de una máscara en una imagen digital (`Parte 1/4`).



c) Procedimiento de aplicación de máscara

Figura 3.14: Aplicación de una máscara en una imagen digital (`Parte 2/4`).

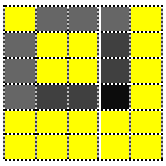


c) Procedimiento de aplicación de máscara

Figura 3.14: Aplicación de una máscara en una imagen digital (`Parte 3/4`).

Paso	Proceso 1	2	3	4	5		Resultado 1	2	3	4	5
18	0	0	0	0	0	⇒	0	1	1	1	0
	0	1	1	1	0		1	0	0	-1	0
	0	1	1	1	0		1	0	0	-1	0
	0	1	1	1	0		1	-1	-1	-2	0
	0	0*-2	0*1	0	0		0	**0**	0	0	0
	0	0*1	0*0	0	0		0	0	0	0	0
19	0	0	0	0	0	⇒	0	1	1	1	0
	0	1	1	1	0		1	0	0	-1	0
	0	1	1	1	0		1	0	0	-1	0
	0	1	1	1	0		1	-1	-1	-2	0
	0	0	0*-2	0*1	0		0	0	**0**	0	0
	0	0	0*1	0*0	0		0	0	0	0	0
20	0	0	0	0	0	⇒	0	1	1	1	0
	0	1	1	1	0		1	0	0	-1	0
	0	1	1	1	0		1	0	0	-1	0
	0	1	1	1	0		1	-1	-1	-2	0
	0	0	0	0*-2	0*1		0	0	0	**0**	0
	0	0	0	0*1	0*0		0	0	0	0	0

c) Procedimiento de aplicación de máscara



d) Imagen resultante

Figura 3.14: Aplicación de una máscara en una imagen digital (`Parte 4/4`).

Para poder apreciar la forma en que las máscaras operan sobre una imagen se muestra un programa en *MATLAB* para encontrar el contorno de una imagen a través del gradiente (**Programa 3.1**) [MatLab 99].

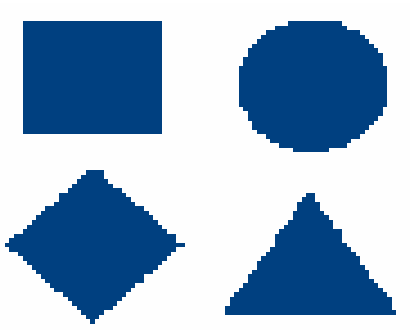
```
1.  % Aplicación de la máscara de gradiente a una imagen binaria
2.  I = imread('prueba.bmp');          %Carga la imagen a procesar
3.  imshow(I,[]);                      %Muestra la imagen de entrada
4.  pause                              %Hace una pausa
5.  % 1.- Para la máscara:
6.  %   -2   1
7.  %   1    0
8.  h=[-2,1;1,0];                      %Máscara No. 1
9.  I2=filter2(h,I);                   %Hace el filtrado con la mascara dada
10. imshow(I2,[]);                     %Muestra el resultado
11. pause                              %Hace una pausa
12. % 2.- Para la máscara:
13. %   -3      2       -.5
14. %   2       0       0
15. %   -.5     0       0
```

Programa 3.1: Obtención de gradientes en *MatLab* (`Parte 1/2`).

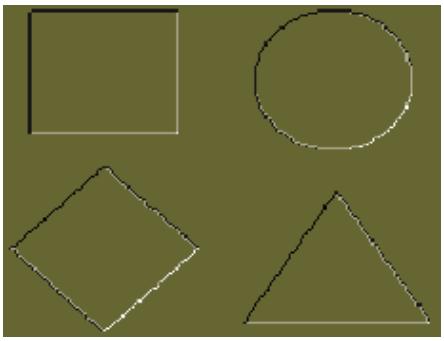
```
16. h=[-3,2,-.5;2,0,0;-.5,0,0];                          %Máscara No. 2
17. I2=filter2(h,I);                                     %Hace el filtrado con la mascara dada
18. imshow(I2,[]);                                       %Muestra el resultado
19. pause                                                %Hace una pausa
20. % 3.- Para la máscara:
21. %   -8/3     3/2     0       -1/6
22. %   3/2      0       0       0
23. %   0        0       0       0
24. %   -1/6     0       0       0
25. h=[-8/3,3/2,0,-1/6;3/2,0,0,0;0,0,0,0;-1/6,0,0,0];    %Máscara No. 3
26. I2=filter2(h,I);                                     %Hace el filtrado con la mascara dada
27. imshow(I2,[]);                                       %Muestra el resultado
```

Programa 3.1: Obtención de gradientes en *MatLab* (`Parte 2/2`).

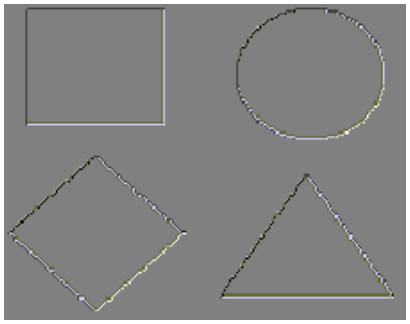
Los resultados de aplicar el programa a una imagen se muestran en la **Figura 3.15**.



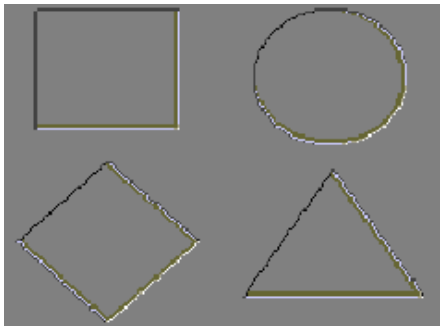
a) Imagen de entrada



b) Primer termino de la derivada



c) Dos términos de la derivada



d) Tres términos de la derivada

Figura 3.15: Resultados al aplicar derivadas a una imagen en *MatLab*.

Como es posible apreciar el empleo de máscaras permite obtener de una forma sencilla el contorno de los objetos presentes en una imagen; también es fácil de programar como se puede apreciar, con *MatLab*. La implementación en *lenguaje C* [Johnson 87] en igualmente sencilla. La complejidad del procedimiento es *O(m*n),* con *m* como el *número de columnas* y *n* el *número de filas* de la imagen debido a que el algoritmo deberá de recorrer la imagen píxel a píxel en un arreglo bidimensional.

Partiendo de la expresión para calcular el gradiente y empleando aproximaciones de derivadas de *Taylor* es posible obtener otras máscaras. Son famosas una serie de máscaras por su uso frecuente en análisis de imágenes.

3.4.3 Aproximación de Roberts

El procedimiento por aproximación de *Roberts* consiste en la aplicación de dos máscaras sucesivas para calcular el vector gradiente en las direcciones diagonales; por lo que es sensible a discontinuidades ó cambios en cualquier dirección. De este procedimiento se obtienen bordes bien definidos (**Figura 3.16**). La máscara es la siguiente:

0	1
-1	0

y

1	0
0	-1

3.4.4 Aproximación de Prewitt

La aproximación de *Prewitt* se obtiene de la aproximación del gradiente de luminosidad cuando se aplica a tres términos, de está forma la máscara para la convolución es una *matriz de 3x3* (**Figura 3.16**). Al igual que en caso anterior es sensible a discontinuidades en todas las direcciones, permite identificar bordes más gruesos que en el caso anterior.

-1	0	1
-1	*0*	1
-1	0	1

y

1	1	1
0	*0*	0
-1	-1	-1

3.4.5 Aproximación de Sobel

Es una variación de la aproximación de *Prewitt* la cual hace un énfasis mayor en la celda central (**Figura 3.16**).

-1	0	1
-2	*0*	2
-1	0	1

y

1	2	1
0	*0*	0
-1	-2	-1

En la **Tabla 3.1** se muestra un resumen con los operadores gradientes más comunes:

Operador	Dirección Horizontal	Dirección vertical
Roberts		
Prewitt		
Sobel		

Tabla 3.1: Operadores gradiente más comunes.

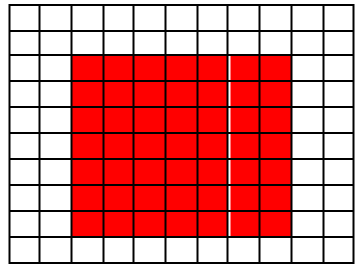
Imagen Digital

Entrada

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0

Resultado

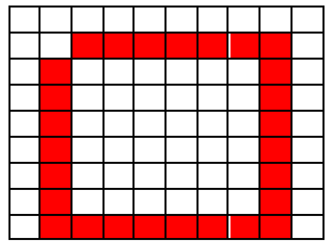
Entrada



Derivada

0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	0
0	1	0	0	0	0	0	0	-1	0
0	1	0	0	0	0	0	0	-1	0
0	1	0	0	0	0	0	0	-1	0
0	1	0	0	0	0	0	0	-1	0
0	1	0	0	0	0	0	0	-1	0
0	1	0	0	0	0	0	0	-1	0
0	1	-1	-1	-1	-1	-1	-1	-2	0

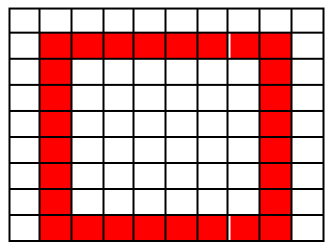
Derivada



Roberts

0	0	0	0	0	0	0	0	0	0
0	1	2	2	2	2	2	2	1	0
0	2	0	0	0	0	0	0	2	0
0	2	0	0	0	0	0	0	2	0
0	2	0	0	0	0	0	0	2	0
0	2	0	0	0	0	0	0	2	0
0	2	0	0	0	0	0	0	2	0
0	2	0	0	0	0	0	0	2	0
0	1	2	2	2	2	2	2	1	0

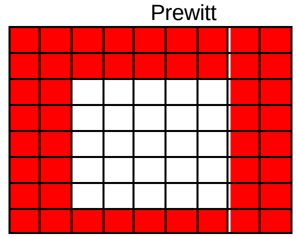
Roberts



Prewitt

2	3	3	3	3	3	3	3	2
3	4	3	3	3	3	3	4	3
3	3	0	0	0	0	0	3	3
3	3	0	0	0	0	0	3	3
3	3	0	0	0	0	0	3	3
3	3	0	0	0	0	0	3	3
3	3	0	0	0	0	0	3	3
3	4	3	3	3	3	3	4	3

Prewitt



Sobel

2	4	4	4	4	4	4	4	2
4	6	4	4	4	4	4	6	4
4	4	0	0	0	0	0	4	4
4	4	0	0	0	0	0	4	4
4	4	0	0	0	0	0	4	4
4	4	0	0	0	0	0	4	4
4	4	0	0	0	0	0	4	4
4	6	4	4	4	4	4	6	4

Sobel

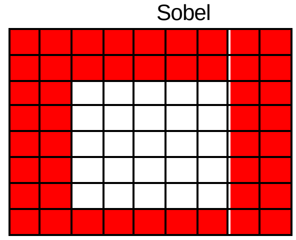
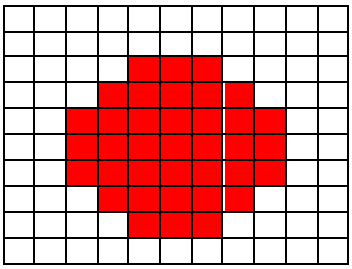


Figura 3.16: Resultados al aplicar operadores gradiente a una imagen (`Parte` **1**/3).

Imagen Digital **Resultado**

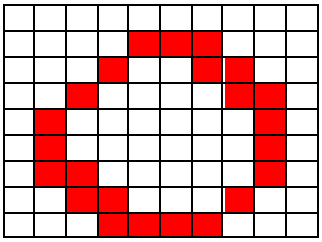
Entrada

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0



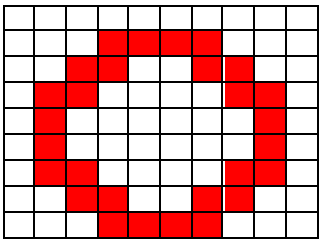
Derivada

0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	2	0	0	-1	1	0	0
0	0	2	0	0	0	0	-1	1	0
0	1	0	0	0	0	0	0	-1	0
0	1	0	0	0	0	0	0	-1	0
0	1	-1	0	0	0	0	0	-2	0
0	0	1	-1	0	0	0	-2	0	0
0	0	0	1	-1	-1	-2	0	0	0



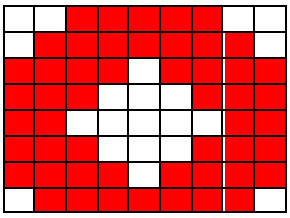
Roberts

0	0	0	0	0	0	0	0	0	0
0	0	0	1	2	2	1	0	0	0
0	0	1	1	0	0	1	1	0	0
0	1	1	0	0	0	0	1	1	0
0	2	0	0	0	0	0	0	2	0
0	2	0	0	0	0	0	0	2	0
0	1	1	0	0	0	0	1	1	0
0	0	1	1	0	0	1	1	0	0
0	0	0	1	2	2	1	0	0	0



Prewitt

0	0	2	3	3	3	2	0	0
0	2	4	4	3	4	4	2	0
2	4	4	2	0	2	4	4	2
3	4	2	0	0	0	2	4	3
3	3	0	0	0	0	0	3	3
3	4	2	0	0	0	2	4	3
2	4	4	2	0	2	4	4	2
0	2	4	4	3	4	4	2	0



Sobel

0	0	2	4	4	4	2	0	0
0	2	6	6	4	6	6	2	0
2	6	6	2	0	2	6	6	2
4	6	2	0	0	0	2	6	4
4	4	0	0	0	0	0	4	4
4	6	2	0	0	0	2	6	4
2	6	6	2	0	2	6	6	2
0	2	6	6	4	6	6	2	0

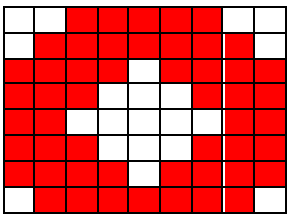
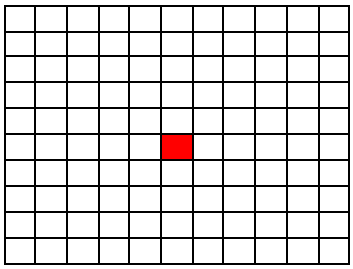


Figura 3.16: Resultados al aplicar operadores gradiente a una imagen (Parte **2**/3).

Imagen Digital **Resultado**

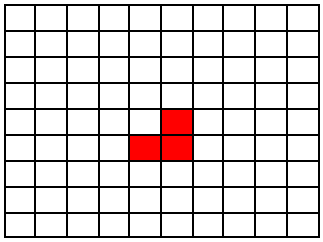
Entrada

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0



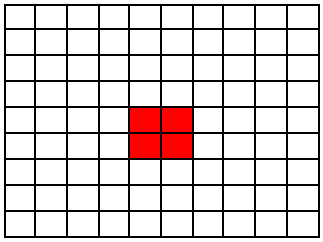
Derivada

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	-2	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0



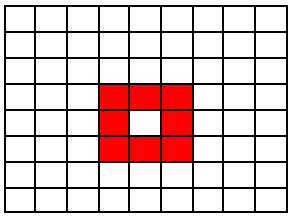
Roberts

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0



Prewitt

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	2	1	2	0	0	0
0	0	0	1	0	1	0	0	0
0	0	0	2	1	2	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0



Sobel

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	2	2	2	0	0	0
0	0	0	2	0	2	0	0	0
0	0	0	2	2	2	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

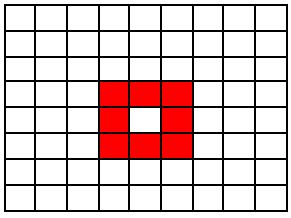


Figura 3.16: Resultados al aplicar operadores gradiente a una imagen (`Parte 3/3`).

3.4.6 Operador Laplaciano

Los métodos anteriores funcionan bien para imágenes con un alto contraste de gris o binarias, sin embargo cuando la región de cambio es muy ancha, resulta mejor aplicar derivadas de segundo orden.

El *Laplaciano* de una imagen (**Figura 3.17b**) se define como la *segunda derivada espacial de la luminosidad*. Para una imagen *f(x,y)* en el punto *(x,y),* se define por la **Ec. 3.17**.

$$\tilde{N}^2 f(x,y) = [\partial^2 f(x,y) / \partial x^2, \partial^2 f(x,y) / \partial y^2] \qquad \text{...(Ec. 3.17)}$$

Para aplicarse a imágenes digitales, es necesario discretizar la **Ec. 3.17**. Si se emplean las aproximaciones de las segundas derivadas obtenemos la **Ec. 3.18**.

$$\partial^2 f(x,y) / \partial x^2 = f(x-1,y) - 2f(x,y) + f(x+1,y)$$
$$\partial^2 f(x,y) / \partial y^2 = f(x,y-1) - 2f(x,y) + f(x,y+1)$$

$$L(x,y) = f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1) - 4f(x,y) \qquad \text{...(Ec. 3.18)}$$

Para que la aproximación sea simétrica respecto al píxel central de la máscara, se debe de considerar la aproximación del *Laplaciano* con un giro de 45°, obtiendo la **Ec. 3.19**.

$$\partial^2 f(x,y) / \partial x^2 = f(x-1,y-1) - 2f(x,y) + f(x+1,y+1)$$
$$\partial^2 f(x,y) / \partial y^2 = f(x-1,y-1) - 2f(x,y) + f(x+1,y+1)$$

$$L(x,y) = f(x-1,y-1) + f(x+1,y+1) + f(x-1,y-1) + f(x+1,y+1) - 4f(x,y) \qquad \text{...(Ec. 3.19)}$$

De las **Ecs. 3.18** y **3.19** se obtienen las máscaras:

0	1	0
1	-4	1
0	1	0

y

1	0	1
0	-4	0
1	0	1

Los operadores basados en la segunda derivada son sumamente sensibles al ruido, por lo que su aplicación se reduce a imágenes filtradas previamente (**Figura 3.17b**).

a) Imagen de Entrada **b)** Laplaciano

Figura 3.17: Operador *Laplaciano.*

3.4.7 Gradiente estocásti co

El uso de máscaras ofrece una pobre respuesta en imágenes con ruido. Una máscara estocástica construye una máscara dinámica (a diferencia de las anteriores que permanecían estáticas) para cada píxel en la imagen, intentando reducir en la medida de lo posible el efecto del ruido, obteniendo al mismo tiempo el gradiente de la imagen. Obviamente el algoritmo es muy costoso en tiempo, debido a la cantidad de cálculos necesarios, por lo que su aplicación es extremadamente reducida.

3.5 Segmentación de imágenes

Una vez que los contornos ó bordes han sido identificados, el siguiente paso es la segmentación de la imagen.

La *segmentación de imágenes* es el proceso mediante el cual una imagen se subdivide en unidades significativas [Ibarra 98]. Permite clasificar los píxeles como pertenecientes a una categoría en concreto. Algunos de los atributos que son tomados en cuenta para segmentar una imagen son: la luminancia (ó brillo) en imágenes binarias, los componentes de color en imágenes de color, los bordes y las texturas.

Los objetivos que persiguen los procesos de segmentación son: encontrar agrupaciones uniformes y homogéneas, diferenciar regiones adyacentes de la escena, hallar bordes sencillos y, sobre todo, que el resultado sea útil para los procesos siguientes de la visión artificial [Escalera 01]. La segmentación se basa en tres propiedades:

- *Similitud*.- cada uno de los píxeles de un objeto presentan propiedades parecidas.
- *Discontinuidad*.- los objetos resaltan de su entorno y por lo tanto tienen bordes bien definidos.
- *Conectividad*.- los píxeles pertenecientes al mismo objeto tienen que ser contiguos, es decir, están agrupados.

Podemos decir que una *región* (puede ser un objeto, el fondo, una textura, etc.) es un conjunto de píxeles que en una imagen representan características o atributos similares. Para poder llevar a cabo la segmentación es necesario que las regiones no se encimen. Se debe de considerar que región puede contener a otras y que la unión de todas las regiones debe de dar como resultado la imagen total.

Algunas de las técnicas utilizadas para la *segmentación de imágenes* son:

- *Segmentación en amplitud*.- se basan en los umbrales de las imágenes binarias o de color. Se emplea en imágenes donde el fondo se diferencia en forma simple de los objetos.
- *Segmentación basada en clustering*.- basada en la clasificación de las características de los píxeles de una imagen, así las regiones se clasifican en función del grado de pertenencia de las diversas características, respecto a las diversas regiones.
- *Segmentación por regiones*.- utilizan propiedades espaciales de las regiones para segmentarlas. Se tiene por ejemplo la técnica de *crecimiento de regiones* en la que se van agrupando regiones que presentan características o propiedades similares. Parte de un conjunto de píxeles llamados *semillas* a partir de las cuales se hacen crecer las regiones añadiendo píxeles a las semillas, provenientes de sus vecinos que poseen propiedades similares, hasta que se cubre el total de píxeles de la imagen.
- *Segmentación por detección de bordes*.- se lleva a cabo detectando los bordes o contornos de cada región, se usan *filtros* (máscaras), *transformada de Hough*, etc.
- *Segmentación basada en texturas*.- una *textura* es la repetición de patrones de elementos situados según reglas específicas. La segmentación por texturas utiliza características locales de una región para separarlas del resto.
- *Segmentación por etiquetado*.- consiste en separar el contorno cerrado de cada segmento. Se puede hacer siguiendo el contorno indicando los cambios de dirección o empleando el código cadena.

Algunos de los métodos que son empleados para segmentar una imagen, se describen a continuación:

3.5.1 Seguimiento de contornos

La segmentación por *seguimiento de contornos*, consiste en ir siguiendo un contorno píxel a píxel en determinadas direcciones en función de sí *se sale* o *se entra* de una determinada región, o se cruza en uno u otro sentido el contorno. Se utiliza imágenes binarias, porque los bordes de las regiones deben estar bien definidos. Por ejemplo, el algoritmo de la **Figura 3.18** realiza las siguientes acciones:

1. Selecciona un punto de comienzo dentro de la región *A* (punto **1** de la **Figura 3.18**).

2. Cambiamos de dirección hacia la izquierda y saltamos al siguiente píxel si esta dentro de la región *A* y a la derecha si esta afuera (puntos **2** y **3** de la **Figura 3.18**).

3. Continúa hasta llegar al píxel de comienzo (punto **1** de la **Figura 3.18**).

Notar que los contornos extraídos con este método son dobles, y que siempre serán cerrados aunque la región no lo sea. Existen variantes con avances de 45° o haciendo pruebas de *8-conectado*, etc., que obtienen mejores resultados.

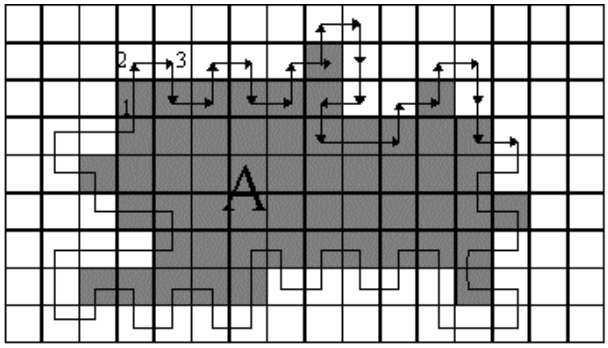


Figura 3.18: Algoritmo de seguimiento de contornos.

3.5.2 Unión de bordes por búsqueda heurística

Otro método para la segmentación de imágenes es el que se conoce como la *unión de bordes por búsqueda heurística*. Puede pensarse en la frontera de un objeto, como el camino resultante de la unión de los elementos del borde (tomando estos como píxeles sin etiquetar). Según las reglas de enlazado obtendremos un algoritmo u otro. Supóngase un gráfico con nodos x_k que representan distintos elementos del borde. Mediante una *función de evaluación* $\phi(x_k)$ obtenemos un valor de camino de un punto a otro a través del nodo x_k. En algoritmos heurísticos, para encontrar el camino que une dos puntos *A* y *B*, se examinan los nodos siguientes a uno de comienzo y se escoge como nuevo nodo de comienzo aquel que lleve al máximo ϕ. La secuencia de nodos seleccionados constituye el contorno entre *A* y *B*.

Basándose en el principio de *optimalidad de Bellman* [Richard Bellman, 1957] (si un camino es óptimo, sus subcaminos también lo son), existe un método general para encontrar el camino óptimo global en procesos multifase, mediante programación dinámica. Se basa en ir etiquetando nodos con el mejor de los caminos posibles, desde el camino inicial hasta llegar al nodo final. Se comprueban todas las posibles rutas con lo que la solución siempre es óptima. Estos métodos son muy conocidos para el encaminamiento de datagramas a través de los nodos que forman una red de comunicaciones. Obviamente la velocidad de estos algoritmos depende de la elección de $\phi(x_k)$.

3.5.3 Transformada de Hough

Otro método para la segmentación de imágenes es el que se conoce como la *Transformada de Hough*. Se utiliza para el enlace de puntos del borde y para la extracción de rectas de un contorno. Implica la transformación de coordenadas cartesianas a coordenadas polares. Podemos establecer la ecuación de una recta en coordenadas polares tal como se ve en la **Figura 3.19**. Si tenemos un punto (x_p,y_p), podrán pasar infinitas rectas por ese punto; en el plano transformado, podemos establecer esto como una función senoidal (**Ec. 3.20**).

$$r = x_p \cos q + y_p \sin q = f(q) \qquad \text{...(Ec. 3.20)}$$

Si representamos esta función para todos los puntos de una recta en el primer cuadrante, y desechamos los valores de ρ negativos, estas funciones se cortarán todas en un punto que nos definirá la ecuación de la recta $\rho_o = x \cos \theta_o + y \sin \theta_o$

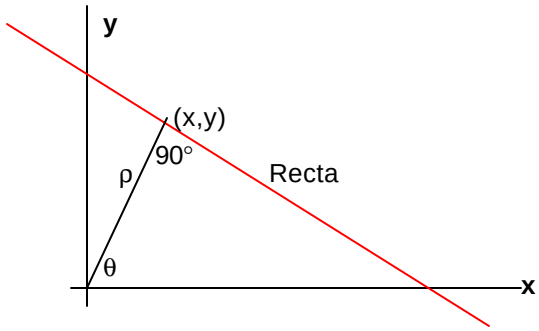


Figura 3.19: Ecuación de una recta en forma polar.

A partir de esta base teórica, se construye un método de detección de rectas en imágenes enormemente robusto. El algoritmo consiste en aplicar la transformación para todos los puntos de un borde de la imagen. El espacio transformado se discretiza de forma que se tenga una buena resolución, y las funciones senoidales resultantes de la transformación de cada píxel se van acumulando. Al final del proceso, se tiene una matriz cuyo valor máximo estará en sus índices, del valor de (ρ_o,θ_o) que definen a la recta.

Puntos de ruido en la imagen, o rectas no continuas no van a influir demasiado en la matriz de transformación, por lo que el algoritmo es muy robusto frente a estos aspectos. Esta transformada busca formas geométricas en toda la imagen encontrando los parámetros de aquella que contenga más puntos de ella.

Los dos inconvenientes principales de esta transformada son que no es capaz de encontrar los extremos de la recta, y que la transformada ha sido patentada, por lo que su uso en algún proyecto requiere del pago de derechos.

En imágenes complejas, con muchas rectas en todas direcciones, la representación del espacio transformado en tres dimensiones, genera figuras bastante complejas (catedrales) con gran cantidad de picos bastante confusos de delimitar. Cuando los puntos de la imagen a analizar proceden de una operación de gradiente; existen variantes mucho más eficaces [Duda 72] que consisten en aprovechar la información de la dirección del gradiente en cada punto, y sólo incrementar la matriz de acumulación, en aquellos puntos próximos a dicha dirección.

3.6 Métodos para reconocimiento de patrones

Después de realizar la segmentación de los objetos, el siguiente paso es el reconocimiento de patrones, es decir, el reconocimiento de los objetos presentes en la imagen de entrada.

La forma de un objeto se refiere a su perfil y a su estructura física del mismo. Para el reconocimiento de objetos se usan una serie de características que nos permiten medir sus formas. Otras características que permiten el reconocimiento de objetos son: perímetro, área, número de agujeros, radios mínimos y máximos o la relación entre ellos, esquinas, número de lados, ángulos entre lados, momentos de inercia, etc.

Un sistema de reconocimiento de patrones general debe de contemplar invariabilidad en las siguientes características de los objetos [Foley 96]:

a) *Traslación.-* es el movimiento de los puntos de un objeto en la imagen cuando todos sus píxeles siguen una dirección constante; el objeto preserva su forma solo cambia de posición (**Figura 3.20a**).

b) *Rotación.-* movimiento de los puntos de un objeto alrededor de un punto llamado eje de rotación, se define por la velocidad y aceleración angulares del objeto, el objeto no cambia de forma, solo de orientación (**Figura 3.20b**).

c) *Escala.-* es el cambio en las dimensiones de los objetos observados en una imagen, sin que la forma del objeto se vea afectada (**Figura 3.20c**).

d) *Perspectiva.-* la deformación de las imágenes causada por la perspectiva, es ocasionada por la presencia de profundidad en la imagen. Una recta que corta al cuadro tiene su perspectiva situado en un punto en el infinito llamado *punto de fuga* (**Figura 3.20d**).

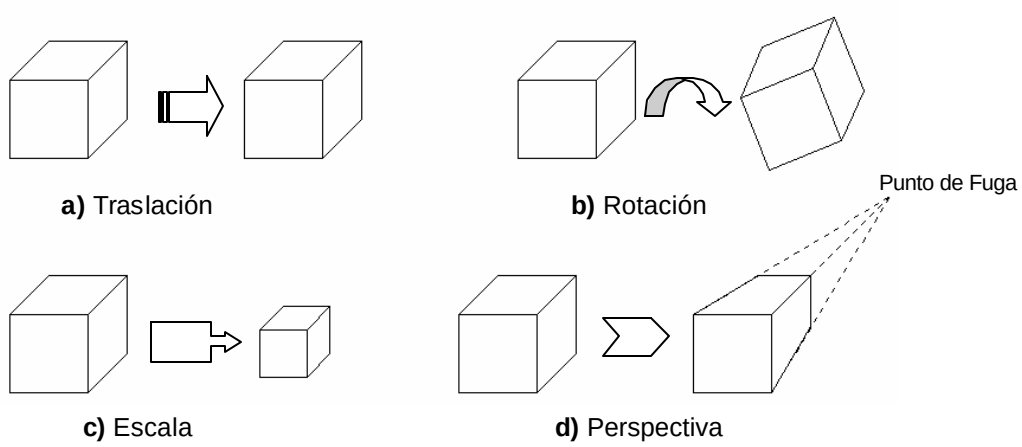


Figura 3.20: Características variables de los objetos en una imagen.

En los apartados siguientes describimos algunos de los métodos empleados para el reconocimiento de patrones contenidos en una imagen.

3.6.1 Reconocimiento por ángulos

El reconocimiento por *ángulos* solo es aplicable en un medio altamente controlado, así es posible identificar un triángulo o un cuadrado correctamente midiendo el ángulo de un vértice [Schildt 87]. Por ejemplo si solo es necesario identificar cuadrados y triángulos isósceles, entonces el reconocimiento se reduce a comprobar dos puntos en cualquier vértice. Si los puntos están en

un ángulo recto, es un cuadrado, si los puntos forman un ángulo de 60° entonces se trata de un triángulo.

La forma más sencilla de implementar el reconocimiento por ángulos es comprobar los puntos adyacentes al vértice para comprobar si se encuentran en el lugar correcto.

La ventaja de este método en su facilidad de implementación, si solo se aplica a variaciones de traslación y escala, si los objetos varían sus ángulos ligeramente o sus vértices rotan de forma ligera. Sólo se puede aplicar a situaciones muy controladas con un número mínimo de objetos variados.

3.6.2 Reconocimiento por puntos clave

El reconocimiento por *puntos clave* se aplica a objetos que presentan el mismo tamaño [Schildt 87], si se conserva el tamaño es posible analizar solo unos pocos puntos clave. Los puntos clave son escogidos de tal manera que solo uno de los objetos satisfaga las condiciones (**Figura 3.21**).

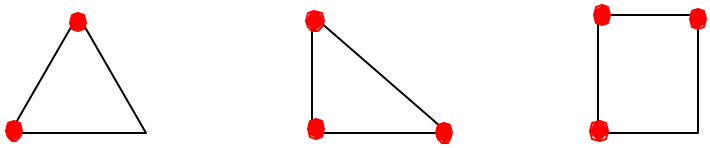


Figura 3.21: Puntos clave para algunos objetos.

Se debe de crear una rutina de reconocimiento para cada objeto a identificar, ésta deberá contemplar una pequeña variación en los puntos clave, de tal forma de que el programa se equivoque lo menos posible. Funciona bien para situaciones restringidas. Sólo permite variaciones en la traslación de los objetos. Para que reconozca correctamente la forma correcta de un objeto, éste no debe de girar, escalarse o mostrar deformaciones por efecto de la perspectiva. Esté método es más restringido que el de reconocimiento por ángulos.

3.6.3 Reconocimiento por cambios de dirección

El reconocimiento por *cambios de dirección* se basa en los cambios de dirección que se presentan cuando se recorre un contorno [Schildt 87]. El número de veces que cambia de dirección es igual número de vértices del objeto. Cada vez que se encuentra un vértice hay un cambio de dirección. El programa deberá de ser capaz de seguir una recta hasta que se intercepte con otra línea. En la intersección, deberá encontrar la siguiente línea y seguirla (regularmente se hace por medio de retroseguimiento redundante). El algoritmo tiene que almacenar los puntos del borde de los objetos que ya ha visitado para evitar que se regrese por el mismo camino, luego debe de ser capaz de regresar al punto inicial. De está forma se garantiza que se ha seguido todo el contorno del objeto.

Una de las ventajas de esté método es que no depende en lo absoluto de la orientación, escala y posición de los objetos. Se emplea frecuentemente en situaciones donde el número de objetos a reconocer es limitado o escaso. Puede equivocarse fácilmente cuando los objetos se encuentran superpuestos, sin embargo no confundirá un objeto con otro dentro de las exigencias de la situación concreta.

3.6.4 Reconocimiento por factor de forma

El *reconocimiento por factor de forma* intenta determinar la *concentración de masa* de los objetos a partir del conocimiento de su área [Mammone 94]. Cuando se tiene un objeto en la imagen, se cuentan los píxeles que lo conforman (en una imagen binaria los puntos donde existe el objeto); el número resultante es el área aproximada en píxeles del objeto en cuestión.

También es posible calcular el número de píxeles que conforman el perímetro del objeto (realizando un cálculo con los píxeles que corresponden al borde), así se pueden determinar los factores de forma de los objetos a clasificar por medio de la **Ec. 3.21**.

$$\text{Factor de forma} = 4\,\pi\,\text{Área} / \text{perímetro}^2 \qquad \text{...(Ec. 3.21)}$$

Se genera entonces una tabla que indica los índices de reconocimiento para diferentes formas. Se observa que entre más se aproxima la forma al círculo, el *factor de forma* también se aproxima a uno. Éste método presenta el inconveniente de que sólo funciona para figuras geométricas, y conforme el número de lados de las figuras se incrementa, la diferencia con respecto al círculo disminuye, para probar esto se realizó un programa en *MatLab* [MatLab 99] para el cálculo del *factor de forma* (**Programa 3.2**).

```
1.  % Programa para obtener el factor de forma de una imagen dada
2.  I = imread('circulo.bmp');          % Lee la imagen
3.  imshow(I);                          % Muestra la imagen
4.  area = bwarea(I)                    % Calcula el área total del objeto
5.  % Obtiene el área del perímetro de la imagen de entrada
6.  I = bwperim(I);                     % Obtiene una imagen con el perímetro del objeto
7.  imshow(I);                          % Muestra la imagen
8.  perimetro = bwarea(I)               % Calcula el área del perímetro del objeto
9.  % Calcula el factor de forma
10. Factor_de_forma = 4.*3.141592654*area/(perimetro*perimetro)
```

Programa 3.2: Cálculo del factor de forma en *MatLab*.

Al aplicar el programa anterior a los objetos de la **Figura 3.22** obtenemos los resultados de la **Tabla 3.2**.

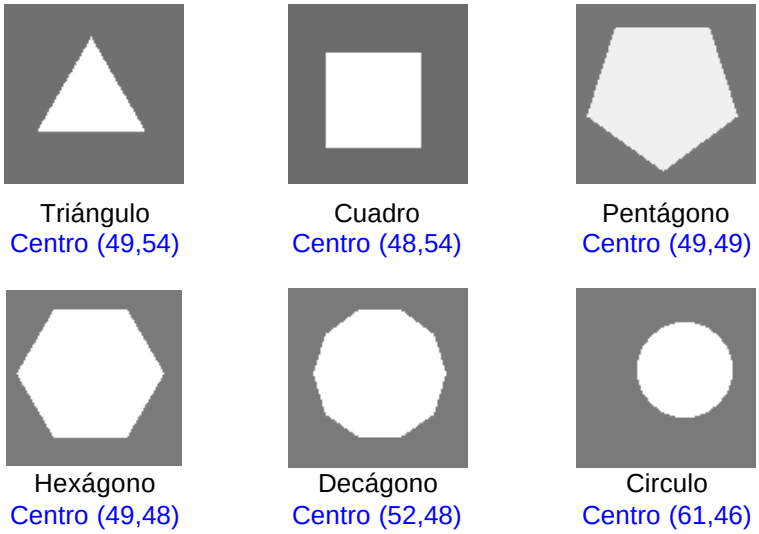


Figura 3.22: Figuras empleadas para él cálculo del factor de forma.

Figura	Factor de forma calculado	Factor de forma medido
Triángulo equilátero	0.6283	0.6498
Cuadro	0.7854	0.8120
Pentágono	0.8651	0.8989
Hexágono	0.9068	0.9451
Decágono	0.9668	0.9908
Círculo	1.0000	1.0483

Tabla 3.2: Factores de forma calculados y obtenidos con *MatLab*.

3.6.5 Reconocimiento por cálculo de momentos

Una de las formas más generales para la extracción de las características de los objetos en una imagen es el uso de *momentos* [Mammone 94, Masters 94, Corke 96, Ayres 87, Granville 92, Mazaira 94, Staugaard 87]. Los *momentos* son fáciles de calcular, y se puede alcanzar una alta velocidad empleando hardware especializado. Se emplean para encontrar la localización de un objeto (mediante su centroide). Otras características de los momentos permanecen invariables a la traslación, orientación y escala de los objetos, por lo que pueden emplearse para reconocimiento (como lo demostró *Hu en 1962*).

Los *momentos de orden (p+q)* para una *imagen binaria I(x,y)* que solo toma valores *0* ó *1*, para la región *R* de *I(x,y)* que toma valores de *1*, los *momentos de orden (p+q)* se pueden calcular a través de la **Ec. 3.22**.

$$m_{pq} = S_R S_R x^p y^q I(x,y) \quad \text{...(Ec. 3.22)}$$

Los *momentos* se pueden interpretar físicamente como una función de *distribución de masa en la imagen*. Para m_{00} como la *masa total* de la región *R*, se puede calcular la posición (x_c,y_c) del *centroide*, con las **Ecs. 3.23** a **3.30**.

$$m_{00} = S_R S_R I(x,y) \quad \text{...(Ec. 3.23)}$$
$$m_{10} = S_R S_R x\, I(x,y) \quad \text{...(Ec. 3.24)}$$
$$m_{01} = S_R S_R y\, I(x,y) \quad \text{...(Ec. 3.25)}$$
$$m_{11} = S_R S_R x\, y\, I(x,y) \quad \text{...(Ec. 3.26)}$$
$$m_{20} = S_R S_R x^2 I(x,y) \quad \text{...(Ec. 3.27)}$$
$$m_{02} = S_R S_R y^2 I(x,y) \quad \text{...(Ec. 3.28)}$$
$$x_c = m_{10} / m_{00} \quad \text{...(Ec. 3.29)}$$
$$y_c = m_{01} / m_{00} \quad \text{...(Ec. 3.30)}$$

Los *momentos centrales* μ_{pq} se calculan alrededor del *centroide* por la **Ec. 3.31**.

$$\mu_{pq} = S_R S_R (x-x_c)^p (y-y_c)^q I(x,y) \quad \text{...(Ec. 3.31)}$$

Esté tipo de momentos es invariable a la *traslación*. Algunos momentos centrales a partir de la **Ec. 3.31** se muestran en las **Ecs. 3.32** a **3.39**.

$$\mu_{00} = m_{00} \quad \text{...(Ec. 3.32)}$$
$$\mu_{11} = S_R S_R (x-x_c)(y-y_c) I(x,y) \quad \text{...(Ec. 3.33)}$$
$$\mu_{21} = S_R S_R (x-x_c)^2 (y-y_c) I(x,y) \quad \text{...(Ec. 3.34)}$$
$$\mu_{12} = S_R S_R (x-x_c)(y-y_c)^2 I(x,y) \quad \text{...(Ec. 3.35)}$$
$$\mu_{20} = S_R S_R (x-x_c)^2 I(x,y) \quad \text{...(Ec. 3.36)}$$
$$\mu_{02} = S_R S_R (y-y_c)^2 I(x,y) \quad \text{...(Ec. 3.37)}$$
$$\mu_{30} = S_R S_R (x-x_c)^3 I(x,y) \quad \text{...(Ec. 3.38)}$$
$$\mu_{03} = S_R S_R (y-y_c)^3 I(x,y) \quad \text{...(Ec. 3.39)}$$

Sin embargo en algunos casos solo se requiere conocer algunos momentos centrales (con el propósito por ejemplo de obtener el ángulo), se pueden obtener a partir de los *momentos de orden (p+q)* con las **Ecs. 3.40** a **3.46**.

$$\mu_{20}=m_{20}-m_{10}^2 / m_{00} \quad \text{...(Ec. 3.40)}$$
$$\mu_{02}=m_{02}-m_{01}^2 / m_{00} \quad \text{...(Ec. 3.41)}$$
$$\mu_{11}=m_{11}-m_{10}\, m_{01} / m_{00} \quad \text{...(Ec. 3.42)}$$
$$\mu_{12}=m_{12}-2m_{11}\, m_{01} / m_{00}-m_{02}m_{10}/m_{00}+2m_{01}^2/m_{00} \quad \text{...(Ec. 3.43)}$$
$$\mu_{21}=m_{21}-2m_{11}\, m_{10} / m_{00}-m_{20}m_{01}/m_{00}+2m_{10}^2/m_{00} \quad \text{...(Ec. 3.44)}$$
$$\mu_{03}=m_{03}-3m_{02}\, m_{01} / m_{00}+2m_{01}^3/m_{00}^2 \quad \text{...(Ec. 3.45)}$$
$$\mu_{30}=m_{30}-3m_{20}\, m_{10} / m_{00}+2m_{10}^3/m_{00}^2 \quad \text{...(Ec. 3.46)}$$

Usando el *factor de forma* de los objetos es posible calcular la *circularidad*, particular de cada uno de los objetos a identificar por la **Ec. 3.47**.

$$\text{Circularidad} = 4\,\pi\, m_{00} / \text{perímetro}^2 \quad \text{...(Ec. 3.47)}$$

Presenta un valor máximo de *circularidad = 1* para él *circulo*, otros valores se tienen en la **Tabla 3.2**. Los *segundos momentos de área* pueden considerarse como los momentos de inercia del *centroide*. De estos es posible determinar la *orientación del eje principal* (**Figura 3.23** y en la **Ec. 3.48**).

$$\theta = 0.5\, \arctan(2\,\mu_{11} / [\mu_{20} - \mu_{02}]) \quad \text{...(Ec. 3.48)}$$

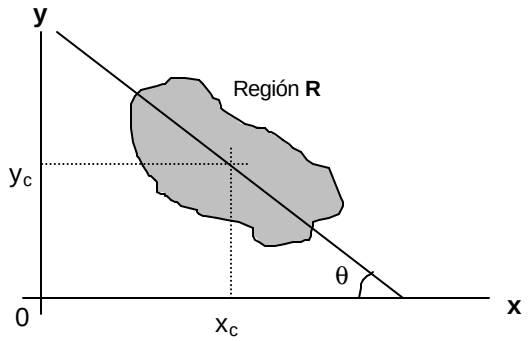


Figura 3.23: Angulo principal de la región **R.**

Los momentos *centrales normalizados* η_{pq} son invariables a *escala*. Para una región *R* se definen como se muestra en la **Ec. 3.49**.

$$\eta_{pq} = \mu_{pq} / m_{00}^{\gamma} \qquad \gamma = 0.5\,(p+q) + 1 \quad \text{...(Ec. 3.49)}$$

De donde podemos obtener las **Ecs. 3.50** a **3.56**.

$$\eta_{11} = \mu_{11} / m_{00}^2 \quad \text{...(Ec. 3.50)}$$
$$\eta_{20} = \mu_{20} / m_{00}^2 \quad \text{...(Ec. 3.51)}$$
$$\eta_{02} = \mu_{02} / m_{00}^2 \quad \text{...(Ec. 3.52)}$$
$$\eta_{21} = \mu_{21} / m_{00}^{5/2} \quad \text{...(Ec. 3.53)}$$
$$\eta_{12} = \mu_{12} / m_{00}^{5/2} \quad \text{...(Ec. 3.54)}$$
$$\eta_{30} = \mu_{30} / m_{00}^{5/2} \quad \text{...(Ec. 3.55)}$$
$$\eta_{03} = \mu_{03} / m_{00}^{5/2} \quad \text{...(Ec. 3.56)}$$

Momentos invariables (*momentos de Hu*) ó *momentos de tercer orden* proporcionan valores invariables a *traslación, escala* y *orientación* en el plano. *Hu* describe siete momentos invariables dados por las **Ecs. 3.57** a **3.63**.

$$f_1 = h_{20} + h_{02} \quad \text{...(Ec. 3.57)}$$

$$f_2 = (h_{20} - h_{02})^2 + 4h_{11}^2 \quad \text{...(Ec. 3.58)}$$

$$f_3 = (h_{30} - 3h_{12})^2 + (3h_{21} - h_{03})^2 \quad \text{...(Ec. 3.59)}$$

$$f_4 = (h_{30} + h_{12})^2 + (h_{21} + h_{03})^2 \quad \text{...(Ec. 3.60)}$$

$$f_5 = (h_{30} - 3h_{12})(h_{30} + h_{12})[(h_{30} + h_{12})^2 - 3(h_{21} + h_{03})^2] + (3h_{21} - h_{03})(h_{21} + h_{03})[3(h_{30} + h_{12})^2 - (h_{21} + h_{03})^2] \quad \text{...(Ec. 3.61)}$$

$$f_6 = (h_{20} - h_{02})[(h_{30} + h_{12})^2 - (h_{21} + h_{03})^2] + 4h_{11}(h_{30} + h_{12})(h_{21} + h_{03}) \quad \text{...(Ec. 3.62)}$$

$$f_7 = (3h_{21} - h_{03})(h_{30} + h_{12})[(h_{30} + h_{12})^2 - 3(h_{21} + h_{03})^2] + (3h_{12} - h_{30})(h_{21} + h_{03})[3(h_{30} + h_{12})^2 - (h_{21} + h_{03})^2] \quad \text{...(Ec. 3.63)}$$

Por ejemplo, para calcular los momentos para las posiciones del objeto de la **Figura 3.24**, se tienen los resultados mostrados en la **Tabla 3.3**.

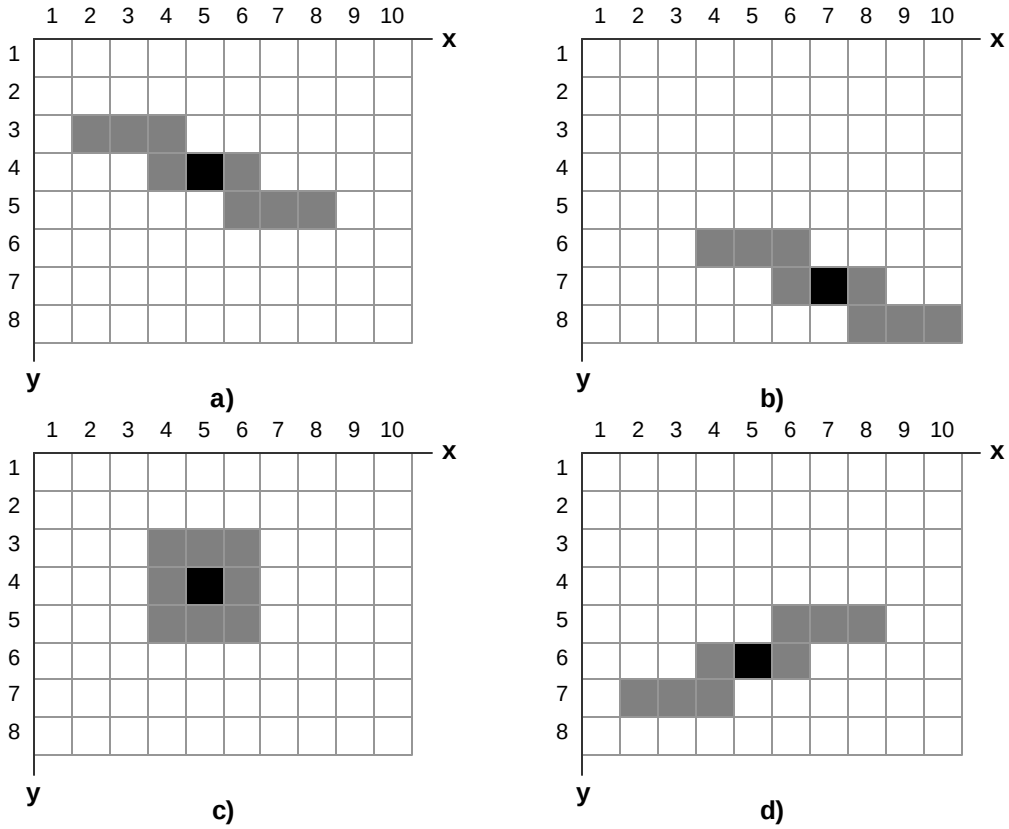


Figura 3.24: Objetos en diferentes situaciones.

Momento	Figura 3.24a	Figura 3.24b	Figura 3.24c	Figura 3.24d
m_{00}	9	9	9	9
m_{10}	45	63	45	45
m_{01}	36	63	36	54
x_c	**5**	**7**	**5**	**5**
y_c	**4**	**7**	**4**	**6**
μ_{20}	30	30	6	30
μ_{02}	6	6	6	6
μ_{11}	12	12	0	-12
Ángulo q	**22.5º**	**22.5º**	**0º**	**-22.5º**
η_{20}	0.37037	0.37037	0.0740741	0.37037
η_{02}	0.0740741	0.0740741	0.0740741	0.0740741
η_{11}	0.148148	0.148148	0	-0.148148
f_1	**0.444444**	**0.444444**	**0.148148**	**0.444444**
f_2	**0.175583**	**0.175583**	**0**	**0.175583**

Tabla 3.3: Momentos calculados.

3.7 Reconocimiento óptico de caracteres (OCR)

Cuando los objetos a reconocer son caracteres, se usa el proceso de *reconocimiento óptico de caracteres* (*OCR* del inglés: *Optical Character Recognition*). Se aplica cuando se tiene una imagen de entrada la cuál presenta algún contenido de caracteres [Escalera 01]. En esta etapa, la imagen no es más que un conjunto de puntos sin sentido (píxeles) sobre un fondo. El programa de *OCR* tiene que extraer información bajo la forma texto de estos píxeles, ha de reconocer las formas y asignarles un símbolo.

Existen dos tipos de sistemas OCR: los sistemas automáticos y los de aprendizaje. El primero puede reconocer automáticamente textos con una gran variedad de tipos de caracteres pero no todos. Con el segundo tipo, cada vez que trate un nuevo documento se tiene que enseñar al sistema los nuevos caracteres; este método requiere una mayor intervención del usuario pero permite tratar casi todos los tipos de caracteres. Ambos sistemas de *OCR* utilizan procesos de reconocimiento similares, para realizar las siguientes tareas:

1. Segmentación de línea.
2. Segmentación de palabras y caracteres.
3. Reconocimiento de caracteres.
4. Producción de un archivo de salida de texto.

La *segmentación de líneas* consiste en aislar cada línea de texto. En esta etapa, se analiza la inclinación del texto y el espacio interlineal. Los documentos más difíciles son aquellos que tienen un espacio interlineal reducido y un texto de notable inclinación.

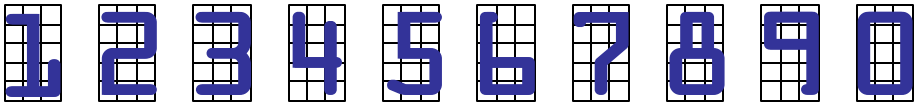
La *segmentación de palabras y caracteres* consiste en separar las palabras entre sí y aislar las diferentes letras que las componen. Si todos los caracteres tienen el mismo ancho, (intervalo fijo), esta etapa es muy fácil de ejecutar; como es el caso de los documentos mecanografiados. El problema surge cuando el ancho de las letras depende de su forma (intervalo proporcional). El caso más complicado es el de un documento cuyos caracteres tienen una forma proporcional y se tocan casualmente. Los caracteres compuestos por matrices de puntos exigen técnicas de segmentación muy específicas.

La etapa de *reconocimiento de caracteres* consiste en extraer las características de cada forma aislada y asignarle a ésta un símbolo.

El *texto de salida* es la conversión del texto reconocido en un archivo.

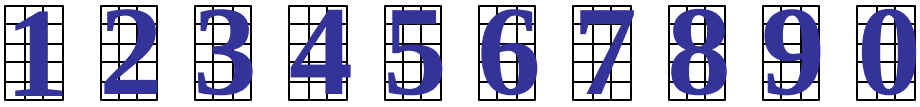
Veamos como funciona un sistema de este tipo con dígitos. En la **Figura 3.25** se tienen dos tipos de letra diferentes para los mismos dígitos. Un algoritmo *OCR* identifica cada línea y dígito. A partir del dígito, realizaría una partición de tal forma que pude obtener un código. En este ejemplo, cada carácter se particionó en una matriz de *3x5* y se asigno a cada celda un valor, dependiendo si estaba con alguna fracción de color negro (una parte del dígito). Así se obtiene un código binario para cada dígito. Se observa que para el mismo número, con dos tipos de letra diferente, el código obtenido es muy diferente.

a) Tipo de caracteres: **OCR A Extended**



No.	Fila 1			Fila 2			Fila 3			Fila 4			Fila 5			
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3	**Octal**
1	1	1	0	0	1	0	0	1	0	0	1	1	1	1	1	62,237
2	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	71,747
3	1	1	1	0	0	1	0	1	1	0	0	1	1	1	1	71,617
4	1	0	0	1	0	1	1	0	1	1	1	1	0	0	1	45,571
5	1	1	1	1	0	0	1	1	1	0	0	1	1	1	1	74,717
6	1	0	0	1	0	0	1	0	0	1	1	1	1	1	1	44,477
7	1	1	1	0	0	1	0	0	1	0	1	0	0	1	0	71,122
8	0	1	0	0	0	0	1	0	1	1	0	1	1	1	1	20,557
9	1	1	1	1	0	1	1	1	1	0	0	1	0	0	1	75,711
0	1	1	1	1	0	1	1	0	1	1	0	1	1	1	1	75,557

b) Tipo de caracteres: **Times New Roman**



No.	Fila 1			Fila 2			Fila 3			Fila 4			Fila 5			
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3	**Octal**
1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	22,222
2	0	1	0	0	0	1	0	0	0	0	0	0	1	1	1	21,007
3	0	1	1	0	1	0	0	1	1	0	0	1	1	0	0	32,314
4	0	1	1	0	1	1	0	1	1	1	1	1	0	1	1	33,373
5	0	1	1	1	0	0	1	1	1	0	0	1	1	0	0	34,714
6	0	0	0	1	0	0	1	0	1	1	0	1	1	0	1	04,555
7	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	70,000
8	1	0	1	1	0	1	0	1	0	1	0	1	1	0	1	55,255
9	0	0	1	1	0	1	1	0	1	0	0	1	0	1	0	15,512
0	0	0	0	1	0	1	1	0	1	1	0	1	0	0	0	05,550

Figura 3.25: Obtención de códigos para *OCR.*

3.8 Calibración de cámara

Antes de que el robot o cualquier otro sistema pueda usar la información obtenida a partir del proceso de reconocimiento por visión artificial, es necesario adecuarla a las unidades de medida que dichos sistemas comprendan, para esto se realiza lo que se conoce como *calibración de cámara.*

La *calibración de cámara* [Corke 96] es el proceso que permite determinar la geometría interna de la cámara, así como sus características ópticas, posición en el espacio de tres dimensiones y la orientación del cuadro de cámara con relación a las *coordenadas del sistema*. Con frecuencia la *calibración de cámara* se realiza de forma empírica, con una matriz de calibración relacionada con el plano de coordenadas del sistema. Los parámetros a obtener con la calibración de cámara son:

- *Parámetros intrínsecos*.- son todos los relacionados con el conjunto de cámara óptica, como el centro de la imagen, distancia focal, distorsión, tamaño de los píxeles, etc.
- *Parámetros extrínsecos*.- es la posición y orientación de la cámara respecto al mundo real.

Existe una gran diversidad de métodos para la calibración de cámaras. Algunos de ellos y sus características importantes se describen en la **Tabla 3.4**. Se diferencian en la forma de obtener la distorsión del lente de la cámara y los patrones de calibración empleados. Una *carta de puntos coplanares* (puntos situados en un mismo plano) puede ser generada conveniente y sin error con una impresora láser. *Marcas circulares* son empleadas por que permiten encontrar el centroide de manera sencilla, pero pueden introducir algunos errores en planos normales al plano de interés, es decir, pueden parecer elipses, con lo cual el centroide puede no coincidir con el centro de la marca, esto se corrige haciendo las marcas tan pequeñas como sea posible, aunque también se emplean otro tipo de marcas, como *líneas cruzadas, marcas con esquinas rectangulares*, etc.

Método	Tipo de patrón empleado		Distorsión
	Coplanar	No coplanar	
No- lineal Clásico		✓	✓
Transformación Homogénea		✓	
2 Planos	✓		✓
2 Estados	✓		✓

Tabla 3.4: Comparación entre diferentes técnicas de calibración de cámaras.

El *método no lineal clásico* consiste de un conjunto de técnicas desarrolladas por la comunidad de fotometría. En estas las cámaras son descritas por un modelo detallado, regularmente con más de 18 parámetros. La optimización por las técnicas de *calibración no lineal,* usa los datos de calibración para ajustar de forma iterativa los parámetros requeridos.

El *método de transformación homogénea* se emplea donde se requiere la determinación de la *orientación* y la *posición* de la cámara [Schilling 90]. En la **Figura 3.26**, se muestra el principio de calibración de cámara por el método de transformación homogénea. Para simplificar el problema podemos considerar que la orientación de la cámara es conocida.

La medición directa de la posición de la cámara no es posible de forma simple, pues se requiere medir la posición respecto al plano de la imagen, punto que se encuentra adentro de la cámara. Sin embargo colectando diferentes datos es posible obtener por inferencia dicho punto. La *distancia focal* de la cámara es *f*.

Con (x^0,y^0,z^0) como la posición desconocida del cuadro de cámara $C=\{x^c,y^c,z^c\}$, respecto a cuadro de la base del robot $B=\{x^b,y^b,z^b\}$. Se observa que de forma general la matriz de transformación de coordenadas de cámara a base esta dada por la **Ec. 3.64**.

$$T_{base}^{cámara} = \begin{bmatrix} 0 & -1 & 0 & \dots & x_0 \\ -1 & 0 & 0 & \dots & y_0 \\ 0 & 0 & -1 & \dots & z_0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \quad \text{...(Ec. 3.64)}$$

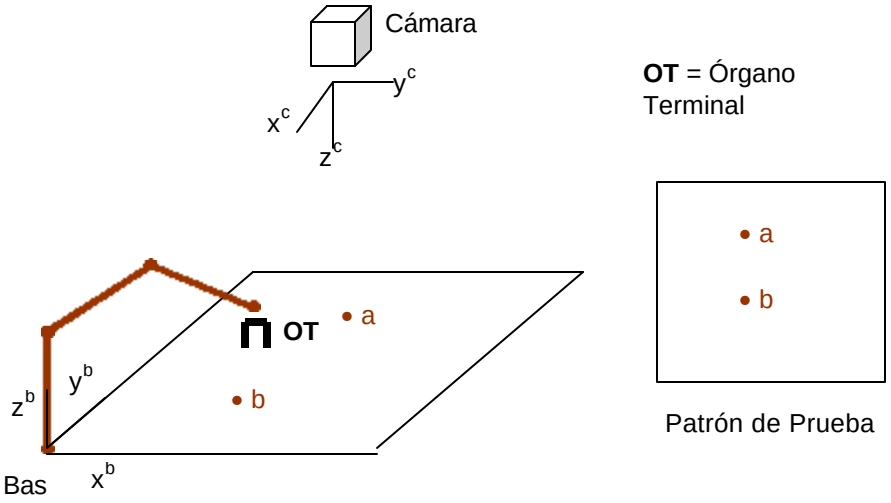


Figura 3.26: Calibración de cámara.

La posición se calibra, colocando un patrón de prueba con dos puntos en su superficie, como se muestra en la **Figura 3.26**. Donde *a* y *b* denotan los puntos de prueba con respecto al cuadro de la base del robot y a^{imagen} y b^{imagen} son las coordenadas de esos puntos en el cuadro de imagen. La relación entre ambas coordenadas de los puntos de la base del robot y de la imagen, se hace transformando las coordenadas de base del robot en coordenadas de imagen. Primero se invierte la matriz de la **Ec. 3.64**, con lo que se mapean las coordenadas de la base del robot a las coordenadas de la cámara, luego se realiza una transformación de perspectiva en donde se relacionan las coordenadas de la cámara con las coordenadas de la base del robot. La composición de ambas transformaciones describe las coordenadas de base del robot con relación a las coordenadas de la imagen (**Ecs. 3.65** y **3.66**).

$$T_{cámara}^{base} = T_{imagen}^{camara}\, T_{cámara}^{base} = T_{imagen}^{cámara} \left[T_{base}^{cámara}\right]^{-1} \quad \text{...(Ec. 3.65)}$$

$$T_{imagen}^{base} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/f & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & x_0 \\ -1 & 0 & 0 & y_0 \\ 0 & 0 & -1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

$$T_{imagen}^{base} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/f & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & y_0 \\ -1 & 0 & 0 & x_0 \\ 0 & 0 & -1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{imagen}^{base} = \begin{bmatrix} 0 & -1 & 0 & y_0 \\ -1 & 0 & 0 & x_0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/f & (f - z_0)/f \end{bmatrix} \quad \text{...(Ec. 3.66)}$$

Ahora con las coordenadas de la imagen a^{imagen}, es posible inferir las coordenadas $T_{imagen}^{base}a$. Esto se puede hacer con el *punto b*, el resultado de la simplificación son las **Ecs. 3.67**, **3.68**, **3.69** y **3.70**.

$$f(y_0 - a_2) = a_1^{imagen}(f - z_0) \quad \text{...(Ec. 3.67)}$$
$$f(x_0 - a_1) = a_2^{imagen}(f - z_0) \quad \text{...(Ec. 3.68)}$$
$$f(y_0 - b_2) = b_1^{imagen}(f - z_0) \quad \text{...(Ec. 3.69)}$$
$$f(x_0 - b_1) = b_2^{imagen}(f - z_0) \quad \text{...(Ec. 3.70)}$$

Las cuatro ecuaciones permiten conocer las coordenadas de la posición de la cámara al resolverse de forma simultánea. Para determinar (x^0, y^0, z^0), las coordenadas de *cámara a al base del robot* se transforman por la matriz de la **Ec. 3.64**. El procedimiento de calibración consiste de los pasos siguientes:

a) Considerar la estación de trabajo del robot como la que se muestra en la **Figura 3.26**.
b) Tener un patrón de prueba con al menos dos puntos *a* y *b* con sus coordenadas sobre la base del robot y las coordenadas de la imagen a^{imagen} y b^{imagen} respecto al cuadro de cámara a_1^{imagen} ı b_1^{imagen}.
c) Si *f* es la *distancia focal* efectiva de la cámara, las coordenadas de la cámara a la base del robot dadas por la matriz de transformación $T_{base}^{cámara}$, la *posición de la cámara* se puede determinar por las **Ecs. 3.71**, **3.72** y **3.73**.

$$z_0 = f\{1 + (a_2 - b_2)/(a_1^{imagen} - b_1^{imagen})\} \quad \text{...(Ec. 3.71)}$$
$$y_0 = a_2 + (f - z_0)a_1^{imagen}/f \quad \text{...(Ec. 3.72)}$$
$$x_0 = a_1 + (f - z_0)a_2^{imagen}/f \quad \text{...(Ec. 3.73)}$$

El *método de transformación homogénea* no puede extenderse para encontrar la distorsión del lente [Corke 96]. Está limitación es superada por la *calibración con dos planos*. En esté método una línea en el espacio se define por los puntos de corte entre dos planos como se observa en la **Figura 3.27**. Cada punto sobre los planos de calibración se encuentra por la **Ec. 3.74**.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_k = A_k \left[{}^iX^n \; {}^iX^{n-1} \; {}^iY \ldots {}^iY^n \ldots {}^iX \ldots {}^iY \; 1 \right]^T \quad \text{...(Ec. 3.74)}$$

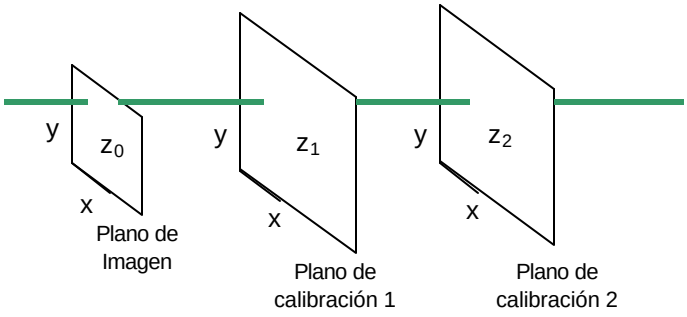


Figura 3.27: Modelo para la calibración en dos planos.

Donde A_k, es la matriz de interpolación para la calibración del plano k. El *método de calibración por dos planos* permite un cálculo rápido de la perspectiva inversa, es decir, de líneas de tres dimensiones a puntos en el plano. Las líneas obtenidas por la interpolación de las coordenadas del mundo tridimensional corresponden con las coordenadas del plano en la *calibración en dos planos*. La interpolación de la matriz de cada plano puede ser determinada probando en cada plano un patrón que contiene un arreglo de puntos que pueda ser visto por la cámara. El orden de interpolación se hace balanceando el tiempo de cómputo, la precisión y la estabilidad. En la práctica un orden dos o tres es suficiente. La interpolación de matrices implica que se ha corregido la distorsión del lente.

El esquema de calibración de *cámara de dos estados*, puede determinar los parámetros intrínsecos y extrínsecos de la cámara, mediante una simple vista de un patrón de marcas en el plano. Son en realidad marcas radiales, llamadas *alineamientos radiales*, que permiten apreciar como actúa la distorsión de la lente a lo largo de las líneas radiales desde el punto central. El algoritmo es moderadamente complejo, pero se ejecuta en un tiempo corto. No se requiere el conocimiento a priori del ciclo de reloj usado por la cámara para digitalizar (β) y los factores de escala para los píxeles (esc_x y esc_y), ni las coordenadas del punto central.

3.9 Áreas de aplicación de sistemas de visión en la industria

Las áreas de la aplicación de sistemas de visión en la industria se pueden dividir en dos grandes categorías: *inspección* e *identificación de objetos*. La *inspección de productos* es una aplicación natural de esta área, la inspección ha requerido siempre de la realización de tareas 100% visuales, en donde se usan criterios de porcentaje de defecto o de dimensión. Algunas de estas aplicaciones se muestran en la **Tabla 3.5**.

Ensamblado y procesos	Defectos en superficies	Medición
Presencia de Partes	Huecos	Diámetro
Localización de perforaciones	Decoloración	Contorno
Presencia de agujeros, hilos, etc.	Vacíos	Dimensiones de un objeto
Patrones propias de conductores	Grietas	
Verificación de etiquetas		
Detección de objetos extraños		

Tabla 3.5: Aplicaciones típicas de la visión artificial en inspección en la industria.

Otra aplicación importante de los sistemas de visión en la industria es la *identificación de objetos*. La identificación de objetos es una tarea más sofisticada que la inspección, requiere por tanto de una cierta inteligencia. En la **Tabla 3.6** se presentan algunas aplicaciones industriales para la identificación de objetos mediante visión artificial.

Manejo de Partes	Guía de Robots
Ensamble de partes	Pintura
Ordenación de partes	Ensamble de productos
Selección binaria	Soldadura

Tabla 3.6: Aplicaciones típicas para identificación de objetos en la industria mediante visión artificial.

Resumen

La *visión artificial*, consiste en el proceso de obtener, caracterizar e interpretar información a partir de imágenes, provenientes del mundo real; es decir, permite obtener una descripción simbólica del mundo real a partir de imágenes. Consiste de un conjunto de tareas como: adquisición de imágenes, preprocesamiento, segmentación, clasificación, y reconocimiento y modelación de los objetos.

La adquisición de imágenes consiste de tres procesos: la formación de imágenes, la captura de imágenes y la digitalización de las imágenes, para que puedan ser almacenadas en la computadora.

El preprocesamiento es el conjunto de las técnicas encaminadas a realizar una mejora de la imagen, modificando por ejemplo: el nivel de gris, el contraste, eliminación de ruido, realce de características, etc.

La segmentación es el proceso mediante el cual una imagen se subdivide en unidades significativas, para encontrar agrupaciones uniformes y homogéneas, diferenciar regiones adyacentes de la escena, hallar bordes sencillos y sobre todo, que el resultado sea útil para los procesos siguientes de la visión artificial. La clasificación es la extracción de características para reconocimiento. La modelación puede ser solo el etiquetado o emplear algunos modelos geométricos como los generados a través de VRML para modelación y reconstrucción tridimensional.

Para el reconocimiento existe una gran cantidad de métodos, como el reconocimiento por ángulos, puntos clave, cambios de dirección, factor de forma y momentos. Este último el más general de todos ellos, es además es inmune a variaciones de posición, escala y orientación de los objetos de una imagen.

El reconocimiento óptico de caracteres consta de las tareas: segmentación de línea, segmentación de palabras y caracteres, reconocimiento de caracteres y de la producción de un archivo de salida de texto. Procesos similares a los de procesamiento de imágenes para visión artificial.

Los métodos de calibración de cámara son el no lineal clásico, el de transformación homogénea, la calibración por dos planos y por dos estados. Sin embargo la calibración de cámara regularmente se obtiene por medios empíricos.

En los procesos industriales los sistemas de visión son de uso específico, debido a que los sistemas de reconocimiento general son demasiado complejos. Los sistemas de visión en la industria encuentran aplicaciones básicamente en dos áreas: en la inspección e identificación de objetos, por que en ellas es donde se realiza una gran cantidad de operaciones visuales.

Capítulo 4: Laboratorio de Robótica Virtual

Un *laboratorio virtual* es una representación realista realizada mediante sistemas de cómputo de un *laboratorio real* [Shneiderman 98]. Así podemos definir a un *laboratorio de robótica virtual* como aquellos en los cuales se recrea a un robot y su entorno mediante modelación virtual. Algunas universidades tienen a disposición robots virtuales teleoperados, por ejemplo la Universidad de Alicante en España lleva a cabo el proyecto de *Teleoperación de un brazo robot con ayuda de realidad virtual* [Internet 4.1], en donde es posible teleoperar un brazo de robot vía Internet. En el Laboratorio de Sistemas Inteligentes, división de ingeniería y ciencias del Instituto Tecnológico de Estudios Superiores de Monterrey, Campus Morelos, se lleva a cabo el proyecto de *Telecontrol para el robot Nomad 200* [Internet 4.2], este trabajo presenta el telecontrol para un robot móvil, utilizando la red electrónica Internet, se dividió el trabajo en tres secciones: programación, comunicación, navegación y seguridad. El proyecto *Títere* está disponible en la Universidad Politécnica de Madrid [Internet 4.3].

De esta manera los *laboratorios virtuales* surgen como una forma de utilizar de modo más eficiente el equipamiento disponible en los laboratorios que cuentan con la infraestructura adecuada para llevar a cabo prácticas de robótica haciéndolos accesibles mediante una herramienta de comunicación entre computadoras como *Internet*. Es en este contexto es en el que se encuentra el proyecto del *Laboratorio de Robótica Virtual del CINVESTAV,* accesible por *Internet*.

Este capítulo presenta los objetivos y componentes del *Laboratorio de Robótica Virtual del CINVESTAV*, la descripción geométrica y cinemática del robot UNIMATE S-103, y la modelación geométrica y cinemática del robot con *VRML* (*Virtual Reality Modeling Language*), también se resalta la importancia que los sistemas de visión tienen y tendrán en un futuro en el desempeño de máquinas automáticas.

4.1 Laboratorio de Robótica Virtual del CINVESTAV

El proyecto de *Laboratorio de Robótica Virtual del CINVESTAV*, se encuentra en el *Departamento de Control Automático* del *Centro de Investigación y de Estudios Avanzados* (*CINVESTAV*) del *Instituto Politécnico Nacional*; tiene por objetivo la realización de prácticas de programación y control de un puesto de trabajo robotizado, para efectuar tareas de manipulación usando sistemas de *visión artificial* (**Figura 4.1**). El laboratorio permite desarrollar una aplicación de manipulación robotizada asistida por *visión*, simularla en una terminal local o remota y una vez depurado el programa correspondiente, enviarlo al servidor del puesto de trabajo de manipulación robotizada (*asistida por visión*), para ejecutarlo, monitoreando su comportamiento real visualmente desde una terminal (local o remota).

El *Laboratorio de Robótica Virtual del CINVESTAV* está compuesto por los siguientes elementos:

- Robot industrial UNIMATE S-103 de arquitectura SCARA con 3½grados de libertad, controlador y sistema de programación C/ROS (Sistema Operativo del Robot).
- Accesorios de Perirrobótica controlados por el robot a través de sus canales de Entrada/Salida: mesa de ensamble, banda transportadora, tornamesa y sensores varios.
- Sistema de visión artificial: cámara CCD panasonic, frame grabber National Instruments, computadora personal y software de procesamiento.

- Servidor (incluye conexión serie con el robot).
- Cámara de monitoreo (web cam).

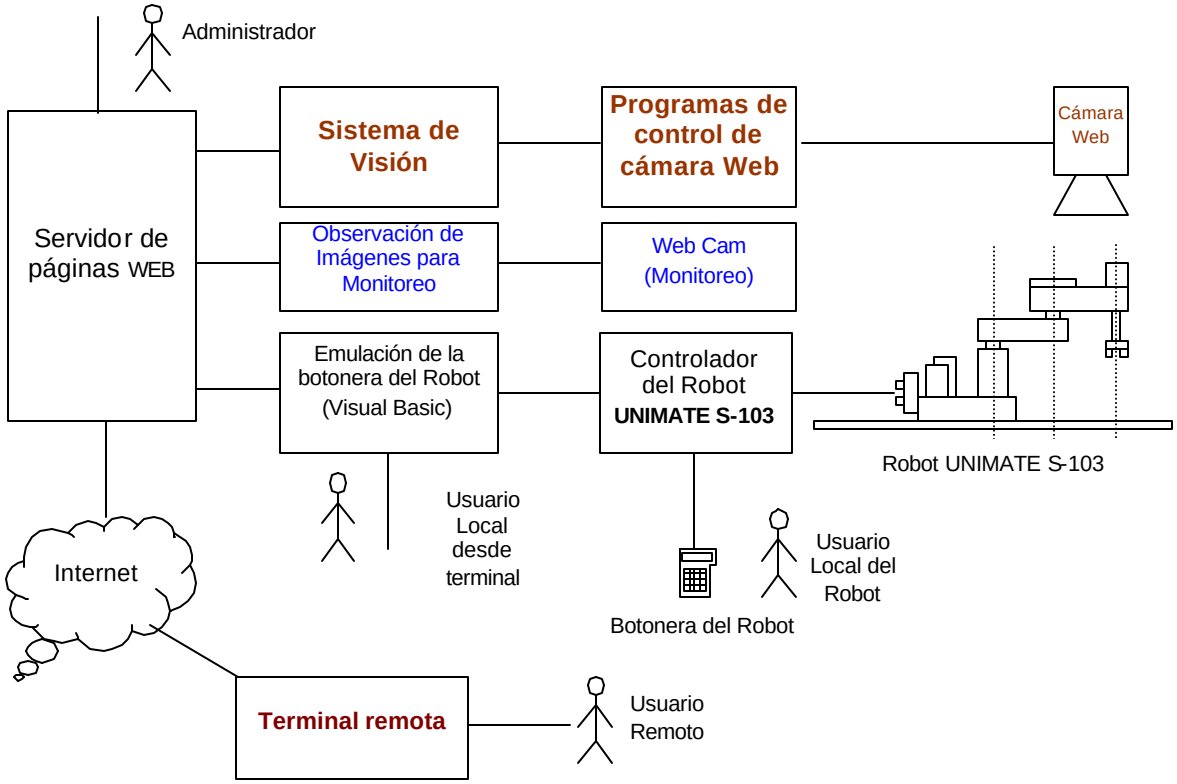


Figura 4.1: Componentes del Laboratorio de Robótica Virtual.

4.1.1 Robot SCARA UNIMATE S-103

El robot *UNIMATE S-103* es un robot industrial cuyas aplicaciones más comunes son el ensamble mecánico y electrónico (**Figura 4.2**) [Malo 93]. Sus principales características son: fiabilidad, robustez, fácil operación y alta velocidad y resolución de movimiento. Está formado por tres sistemas: mecánico, electrónico (hardware) y computacional (software).

El *sistema mecánico* del robot presenta 3½ gdl (grados de libertad) con arquitectura *SCARA* (*Selective Compliance Assembly Robot Arm*). Los primeros tres grados de libertad consisten en rotaciones alrededor de ejes verticales paralelos entre sí, correspondientes a las articulaciones hombro, codo horizontal y muñeca. La muñeca se puede ubicar en cualquier punto en un plano horizontal. La cuarta articulación consiste en un movimiento traslacional a lo largo de un eje vertical coincidente con el eje rotacional de la muñeca. Este movimiento es binario (del tipo todo o nada), por lo que, abusando del lenguaje, se le considera como ½gdl. Un último movimiento es el de apertura y cierre de los dedos de la pinza, el cual también es binario (**Figura 4.2**).

La *arquitectura computacional* para programación y control de este robot está constituida por tres tarjetas controladoras, basadas en el *microprocesador Z-80,* encargadas de controlar cada uno de los tres movimientos servocontrolados del robot. Una tarjeta general basada en el mismo circuito microprocesador se encarga de la coordinación, de la programación y de la comunicación con el exterior. Un sistema de canales binarios de entrada y de salida, así como una botonera externa (*teach pendant*) conectada a uno de los puertos serie del controlador, proporcionan una interfaz de programación y de control del robot. En el gabinete de control, también se encuentran las etapas de potencia de los tres servomecanismos del robot.

Figura 4.2: Robot UNIMATE S-103.

La *operación del robot* se hace desde el *teach pendant* ó *botonera del robot* (**Figura 4.1**), la cual posee un modo de ejecución (*exec*) desde el cual se tiene acceso a los modos operativos de programación (*pgm*), enseñanza de puntos (*teach*), ejecución de programas (*run*), establecimiento de los parámetros internos del robot (*set*), modo de borrado (*clear*) y el modo de comunicación externa, cada uno de estos modos de operación del robot se describe a continuación.

El modo de operación *set* del robot permite programar parámetros globales del robot bajo la denominación del eje cero (*axe* 0). Además permite la programación de los parámetros de eje de manera individual para las tres articulaciones servocontroladas del robot (*axe i* con *i*=1,2,3).

El modo de operación *clear*. Permite borrar el archivo de datos en el que se registran los valores de las coordenadas articulares de cada uno de los puntos de interés para un programa (*points*) así como un archivo del programa (*pgm*).

El modo de operación *pgm* del robot se crean los programas correspondientes a las tareas que deberá realizar el robot. Para ello se dispone de un conjunto de cuatro comandos de movimiento (*MP, MT, MS* y *GR*), comandos de control de flujo (*LB, GO, IF, CS, RE* y *DE*), comandos para la lectura de sensores binarios (*WT, WF, TT* y *TF*), comandos para el manejo de actuadores binarios externos (*ON* y *OFF*), comandos misceláneos (*SP, SY, ST* y *NP*) y cuatro comandos de ayuda para la edición de programas (*INS, DEL, FIX* y *NUM*).

El modo de operación *teach* del robot es el modo enseñanza, mediante el movimiento manual (*jogging*) articulación por articulación. Es posible guardar en memoria las coordenadas articulares correspondiente a cada uno de los puntos que conforman la tarea deseada. El sistema C/ROS (Sistema Operativo del Robot) acepta hasta *94 puntos diferentes*, además de la posición cero (*home*). En este modo también se permite la edición manual de los valores de las tres coordenadas articulares en cada uno de los puntos. En este modo el robot puede aceptar información de posición generada en el exterior (computadora remota) a fin de teleoperarlo.

El modo de operación *run* del robot permite ejecutar un programa cuando ya cuenta con la base de datos correspondiente a las coordenadas articulares de todos los puntos de interés. Ofrece las opciones de ejecución continua y la de paso a paso. En ambas opciones, es posible supervisar visualmente el comando ejecutado en el *display* (pantalla) del *teach pendant*.

4.2 Modelación para el robot UNIMATE S-103

Para realizar una modelación, se tiene que hallar el movimiento de cada uno de los puntos del robot a partir de sus ángulos de rotación, lo que daría como resultado una serie de ecuaciones trigonométricas. Sin embargo esta descripción resulta larga y laboriosa. Un simulador, en lugar de determinar la situación geométrica de muchos puntos, emplea matrices para simplificar el trabajo. Para modelar los movimientos de un robot en la computadora [Ogata 93], se deben de proporcionar los ángulos de rotación de las articulaciones y los desplazamientos lineales de los enlaces extensibles.

Para la modelación geométrica del robot es necesario contar con el dibujo del robot, para el robot UNIMATE S-103 se necesita conocer las dimensiones físicas del espacio de trabajo y del robot mismo. La geometría del robot UNIMATE S-103 se muestra en la **Figura 4.3** [Malo 93], el espacio de trabajo, en el que el robot tiene la posibilidad de manipular objetos, se presenta en la **Figura 4.4**.

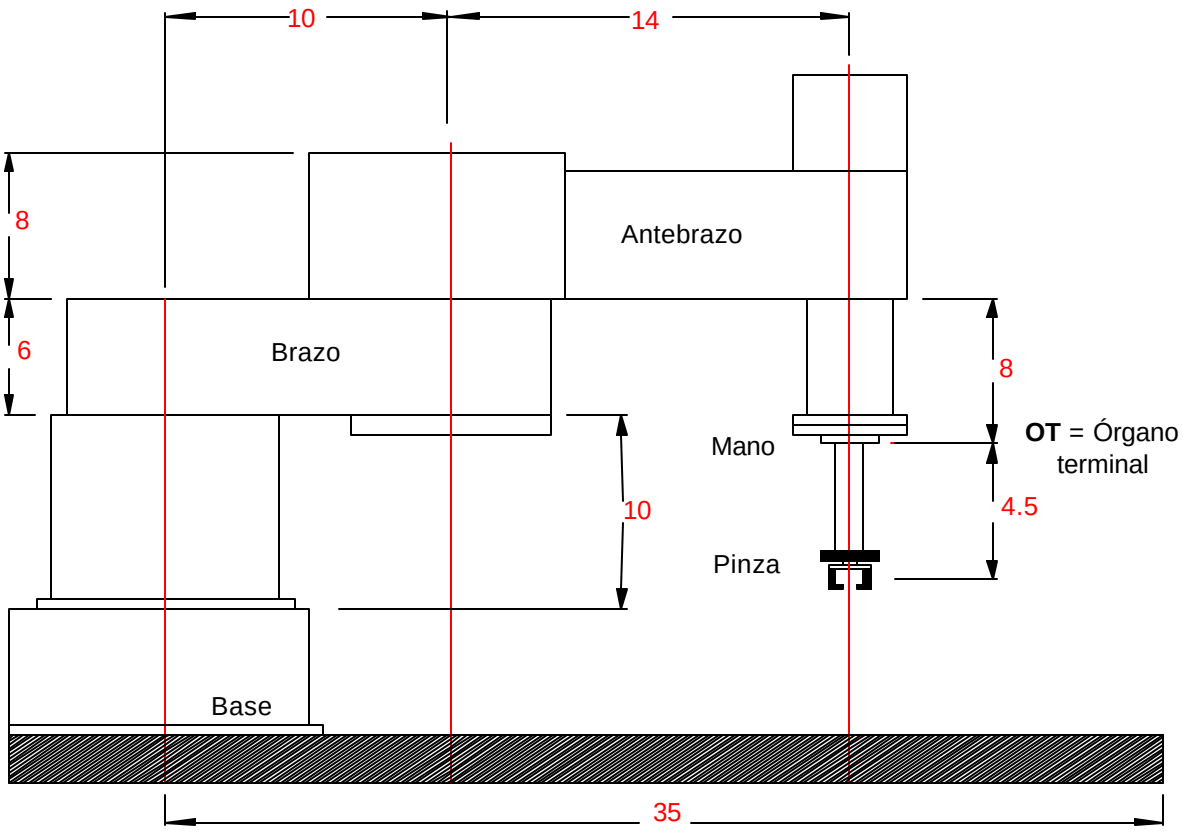


Figura 4.3: Dimensiones del robot UNIMATE S-103 (Acotación: Pulgadas).

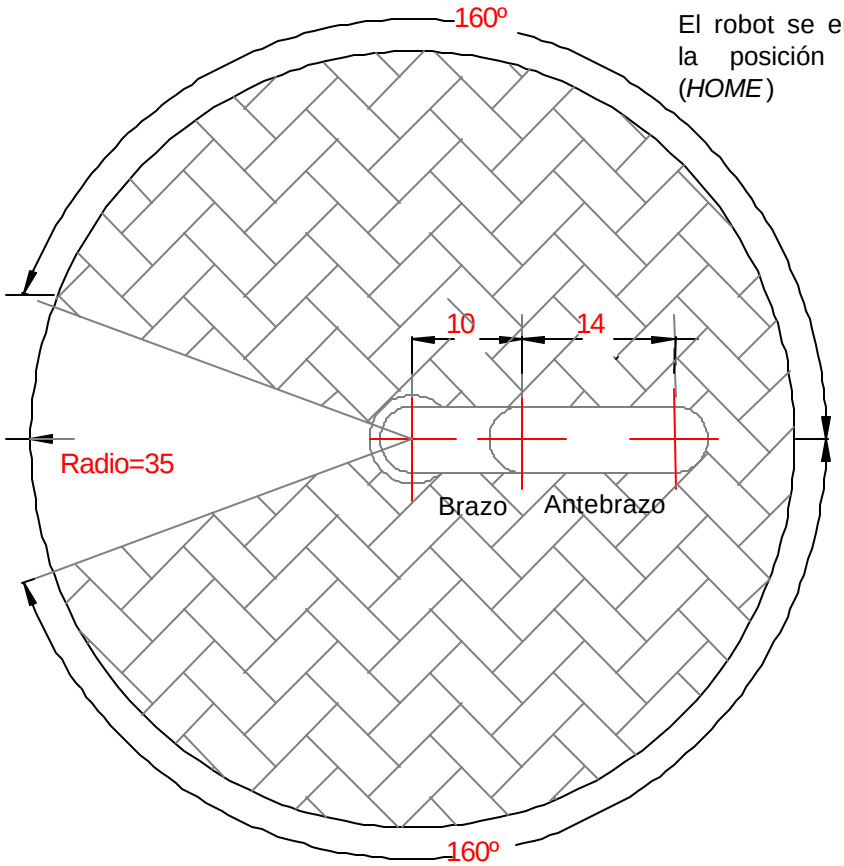


Figura 4.4: Dimensiones del espacio de trabajo del robot UNIMATE S-103 (Acotación: Pulgadas).

Para la modelación virtual, no solo es importante conocer la modelación geométrica del robot, sino también es necesario conocer la modelación cinemática o de movimiento. Se tiene dos tipos de cinemática, la directa y la inversa.

En la *cinemática directa*, se tiene el vector de coordenadas de las articulaciones del robot y se busca determinar la posición y orientación del órgano terminal respecto a las coordenadas de la base del robot.

Una de las formas de representar el modelo de cinemática de un robot, es por la *representación de Denavit-Hartenberg* [Schilling 90], que propone una notación sistemática para asignar coordenadas ortogonales por la regla de la mano derecha (implica que las rotaciones se efectúan alrededor del origen en el sentido contrario al de las manecillas del reloj, por lo que se dice que se tiene una rotación positiva). Comienza por la definición de cada uno de los eslabones y de las articulaciones de la cadena cinemática, después se hace la detección de los ejes articulares y finalmente se asocian los ejes de referencia a cada uno de los eslabones de acuerdo a las reglas siguientes:

1. El eje z^{k-1} se encuentra a lo largo del eje de cada articulación.
2. El eje x^i es normal al eje z^{k-1} y apunta hacia fuera de este.
3. El eje y_i completa los ejes de referencia según la regla de la mano derecha.
4. Cada eslabón tiene una distancia a_i, que designa la distancia de separación entre el eje z^{i-1} y el eje x^i.
5. Cada eslabón tiene un ángulo α_i de separación del eje z^{i-1} al eje z^i respecto del eje x^i.

6. Cada articulación tiene una distancia d_i, que es la distancia entre la intersección de los ejes z^{i-1} y x^i a lo largo del eje z^{i-1}.
7. Cada articulación tiene un ángulo θ_i entre el eje x^{j-1} al eje x^j respecto del eje z^{i-1}. Se utiliza la regla de la mano derecha.

Siguiendo repetidamente las reglas anteriores para los eslabones y las articulaciones tenemos el esquema del robot con sus ejes de referencia (**Figura 4.5**).

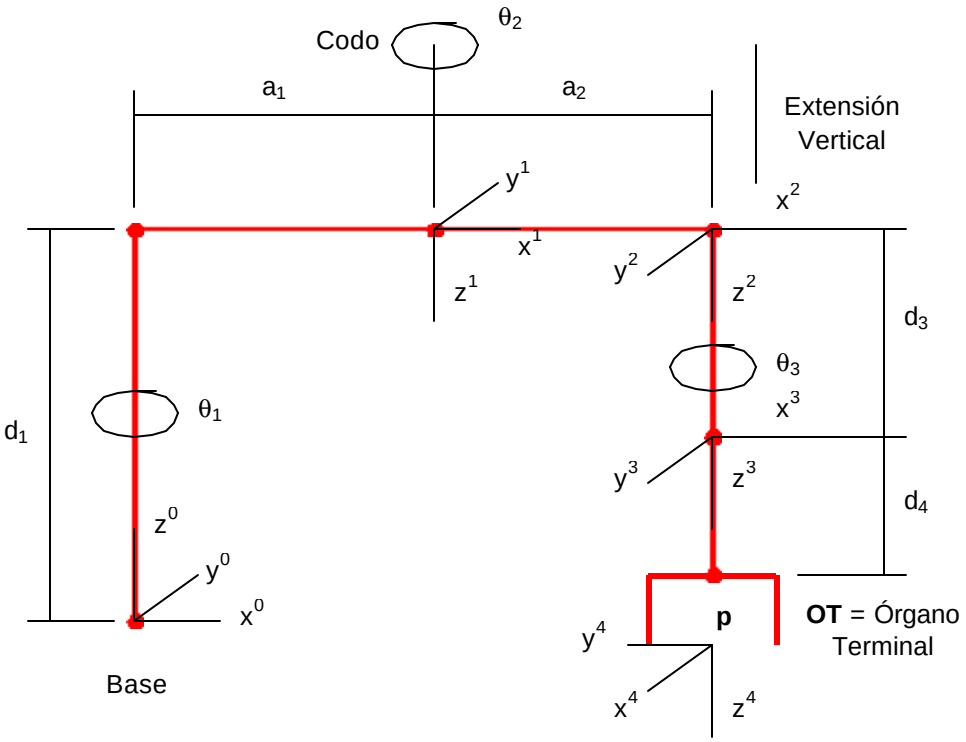


Figura 4.5: Estructura cinemática del robot SCARA UNIMATE S-103.

Para obtener la matriz de la cinemática directa $T^{OT}_{Base}(q)$ con $q = [\theta_1, \theta_2, d_3, \theta_3]^T$ completa del brazo del robot que modela su movimiento cinemático se hace:

$$T^{OT}_{Base}(q) = T_0^1 T_1^2 T_2^3 T_3^4$$

$$T^{OT}_{Base} = \left[\begin{array}{cccc} \cos(q_1 - q_2 - q_3) & sin(q_1 - q_2 - q_3) & 0 & a_1\cos(q_1) + a_2\cos(q_1 - q_2) \\ sin(q_1 - q_2 - q_3) & -\cos(q_1 - q_2 - q_3) & 0 & a_1 sin(q_1) + a_2 sin(q_1 - q_2) \\ 0 & 0 & -1 & d_1 - d_3 - d_4 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Ecuación 4.1: Matriz de la cinemática directa de un robot SCARA

Articulación	q_i	d_i	a_i	a_i
1	θ_1	d_1	a_1	0
2	θ_2	0	a_2	180º
3	0	d_3	0	0
4	θ_3	d_4	0	0

Tabla 4.1: Parámetros de cinemática directa para el robot SCARA.

El *modelo de cinemática inversa* consiste en encontrar la posición *p* y la orientación *R* del órgano terminal, buscando los valores articulares (*q*) que satisfagan la matriz de la **Ec. 4.1**

La expresión que describe la posición del órgano terminal esta dada por el vector *w* de la **Ec. 4.2**.

$$w(q_1,q_2,d_3,q_3) = \begin{bmatrix} a_1\cos q_1 + a_2\cos(q_1 - q_2) \\ a_1 sin q_1 + a_2 sin(q_1 - q_2) \\ d_1 - d_3 - d_4 \\ ---------- \\ 0 \\ 0 \\ -e^{q_3/180^\circ} \end{bmatrix} \quad \text{...(Ec. 4.2)}$$

Para la *articulación del codo* primero se tiene que encontrar el valor de q_2 que se forma con los dos primeros componentes del vector *w*, de la forma mostrada en **Ec. 4.3**.

$$q_2 = w_1^2 + w_2^2 = a_1^2 + 2a_1a_2\cos(q_2) + a_2^2 \quad \text{...(Ec. 4.3)}$$

La ecuación **4.3**, presenta dos posibles soluciones, una tomando la regla de la mano izquierda y la otra por la regla de la mano derecha (**Ec. 4.4**).

$$q_2 = \pm\arccos((w_1^2 + w_2^2 - a_1^2 - a_2^2) / (2a_1a_2)) \quad \text{...(Ec. 4.4)}$$

Para la *articulación de la base,* el ángulo de la base es θ_1 se obtiene de la expresión de la **Ec. 4.5**.

$$q_1 = \text{atan}\{[a_2\text{sen}(q_2)w_1+(a_1+a_2\cos(q_2))]/((a_1+a_2\cos(q_2)w_1-a_2\text{sen}(q_2)w_2)\} \quad \text{...(Ec. 4.5)}$$

Para la *articulación de la extensión vertical* se considera un eslabón prismático en d_3, el cual permite que el órgano terminal baje y suba, a la vez que gire sobre este eje. De forma simple se puede obtener por la **Ec. 4.6**.

$$d_3 = d_1 + d_4 - w_3 \quad \text{...(Ec. 4.6)}$$

Para la articulación del órgano terminal se conocen θ_1 y θ_2 por lo que es posible encontrar el ángulo de rotación del órgano terminal a través de la **Ec. 4.7**.

$$q_3 = q_1-q_2-\text{atan}(\text{sin}(q_1-q_2-q_3)/\cos(q_1-q_2-q_3)) \quad \text{...(Ec. 4.7)}$$

Una vez que es conocida la modelación geométrica y matemática del robot es posible realizar la representación en el espacio tridimensional y simular los movimientos del robot empleando su cinemática.

4.3 Desarrollo del robot virtual

Para crear el *mundo virtual* del Robot UNIMATE S-103 [Alarcón 00, VRML 97], se utilizó el lenguaje *VRML 2.0* (*Virtual Reality Modeling Language*), una poderosa herramienta para la creación de *mundos virtuales* con objetos tridimensionales que permite el control de estos objetos a través de sensores de movimiento, tacto, visibilidad, proximidad y tiempo.

Los objetos tridimensionales realizados con este lenguaje pueden ser visualizados desde un *navegador de Internet* (Internet Explorer, Netscape Comunicator, etc.), añadiendo

visualizadores (*plug-in*) como *Cosmo* o *Platinum*.

Para la creación del *robot virtual* se tomaron en cuenta las medidas del robot real, para dar el mayor realismo posible al mundo virtual. Después, se procedió a hacer el dibujo utilizando sólo las formas geométricas tridimensionales básicas, predefinidas en el lenguaje *VRML* como las primitivas: `Box, Cylinder, Cone` y `Sphere`, todo esto a través del nodo llamado `transform` que contiene los campos `translation, rotation y scale`, además del campo `children` donde se colocan todos los objetos que estarán sometidos a la transformación.

Para el dibujo, el robot se dividió en 7 objetos rígidos, lo que permite adaptar los movimientos articulares del robot. Las 7 partes consideradas fueron: *base* (Inmóvil), *brazo, antebrazo, mano, pinza, dedo izquierdo* y *dedo derecho*, como se ilustra en la **Figura 4.6**. En la *base*, esta contenido todo aquello que no se mueve y a partir de esta se hacen anidamientos, es decir, cada una de las partes esta contenida en la anterior y esta trasladada con respecto al eje de coordenadas de la parte en la que esta contenida. Para tener una idea mas clara de esto, *VRML* permite crear subejes y tomarlos como referencia para todo aquello que esté dibujado dentro de estos. Para hacer esto se utiliza el nodo `transform`. De esta manera el *brazo* esta dibujado justamente a partir del subeje correspondiente al eje real de movimiento y así mismo pasa con el *antebrazo, la mano* y la *pinza*; los *dedos izquierdo* y *derecho* están dibujados a partir de los subejes.

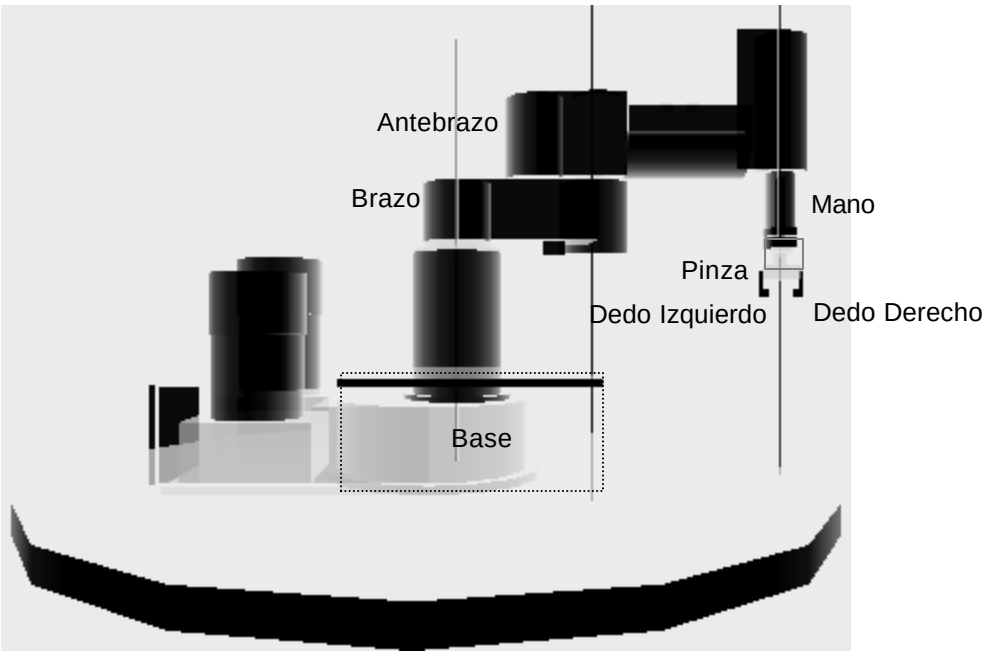


Figura 4.6: Divisiones para el robot virtual UNIMATE S-103.

Una vez realizado el modelo geométrico se procede a definir sensores e interpoladores para el movimiento del *Robot Virtual*.

Para generar los movimientos binarios de la *Pinza*: *abrir - cerrar*, así como *subir - bajar*, fue necesario crear interpoladores de posición, pues ambos son del tipo *todo ó nada*. En los interpoladores utilizados se define una coordenada inicial y una final, las cuales estarán relacionadas con el valor lógico de la variable asociada (*0* ó *1*) como se muestra en el **Programa 4.1**.

```
1. PositionInterpolator {
2.    key [0,1]
3.    keyValue[1 0 0, 0 1 0]}
```

Programa 4.1: Interpoladores de posición en *VRML* para el movimiento del robot.

En este caso el valor *0* o inicial está asociado con el punto de coordenadas *(1, 0, 0).*

Además fue necesario crear cuatro sensores de tiempo en los cuales se define un ciclo que deben realizar los movimientos de las articulaciones del robot, es decir, para subir y bajar la pinza, así como abrir y cerrar los dedos. En el **Programa 4.2** se aprecia la manera de definir los sensores de tiempo mencionados.

```
1. TimeSensor{ CycleInterval 2 }
```

Programa 4.2: Definición de los sensores de tiempo en *VRML* para el movimiento del robot.

En el **Programa 4.2** define un sensor de tiempo que tiene un ciclo de dos segundos.

En la implementación de los movimientos articulares de rotación del antebrazo, brazo y pinza, sólo se indica una rotación en el botón que tiene definido el sensor de movimiento cilíndrico, el campo `rotation` de cada una de las partes correspondientes va cambiando de acuerdo con el movimiento del ratón. Esto se hace a través de las instrucciones del **Programa 4.3**.

```
1. ROUTE SensorGirarAntebrazo.rotation_changed TO Antebrazo.rotation
2. ROUTE SensorGirarAntebrazo.rotation_changed TO BotonAntebrazo.rotation
```

Programa 4.3: Definición de las rotaciones en *VRML* para el movimiento del robot.

Aunque se han definido todos los movimientos con que cuenta el *robot real,* hace falta tener un control más exacto sobre estos. Por ejemplo, en el momento en que la *pinza* se encuentre abajo, es necesario que haga el trabajo de subir con solo tocar el mismo botón, o bien que el ángulo de rotación del *brazo* no cambie cuando se mueve el ángulo de rotación del *antebrazo*. Esto no es posible en *VRML* (**Capítulo 2: Reconstrucción Tridimensional con VRML**), pero afortunadamente nos permite utilizar otros lenguajes que son capaces de realizar estas funciones.

Este procedimiento se realiza a través del nodo `Script` (para utilizar *Java*), en donde se hacen las funciones que han de controlar los movimientos y es incluido en cada una de las rutas en donde es necesario tener ese tipo de control; igual que los anteriores nodos, solo se define y es mandado llamar por eventos de entrada, obteniéndose a partir de sus funciones los respectivos eventos de salida.

Una vez terminada la modelación, el paso siguiente es operar al robot a distancia mediante teleoperación.

El robot *UNIMATE S-103* es un brazo cuyo control es de *arquitectura cerrada* (no es posible su modificación), por lo cual es necesario adaptarse a los parámetros ya establecidos en el control del robot. La programación y el control del movimiento de este robot industrial se realizan tradicionalmente por el *teach pendant*. Esta interfaz de comunicación entre el operador y el robot físico consiste en una botonera manual basada en el *microcontrolador 6802* externa al gabinete de control del robot conectada a uno de sus puertos *seriales*. En términos prácticos el objetivo de la teleoperación consiste en sustituir este *teach pendant* (botonera) físico por uno *virtual*, tal que se pueda visualizar y utilizar desde prácticamente cualquier parte del mundo vía *Internet*.

4.4 Visión artificial en robótica

El objetivo de un sistema de percepción para robots industriales, basado en *sensores de visión*, es el de transformar una imagen del medio ambiente, proporcionada por una cámara, en una descripción de los elementos presentes en el entorno. La descripción debe de contener la información necesaria para que el sistema de control pueda efectuar los movimientos del robot que permitan la ejecución de una tarea programada.

La *visión artificial en robótica* permite a los robots actuar con características de adaptabilidad, flexibilidad y capacidad de reorganización [Ibarra 90]. La visión artificial juega un papel primordial en sistemas robóticos llamados inteligentes, y en general en sistemas flexibles de manufactura, pues permite la retroalimentación sensorial fina, que hace posible ampliar las capacidades del robot.

La utilización de sistemas de visión artificial en sistemas industriales ha aumentado en los últimos años, en tareas como: inspección, identificación, medición y control. Sin embargo estos sistemas son de propósito especial y por lo tanto con algunas limitaciones. Por otro lado se requieren sistemas de propósito general que sean capaces de reaccionar de forma rápida en cadenas de producción variables.

Resumen

El Laboratorio de Robótica Virtual del CINVESTAV permite la realización de prácticas de programación y control de un puesto de trabajo robotizado que efectúe tareas de manipulación usando sistemas de visión artificial. También es posible desarrollar una aplicación de manipulación robotizada asistida por visión, depurarla en la terminal remota ó local por simulación gráfica en tres dimensiones y enviarla al servidor del Laboratorio de Robótica Virtual del CINVESTAV para que el robot industrial las ejecute en tiempo real, monitoreando su comportamiento visualmente desde la terminal remota ó local. Uno de los componentes más importantes del laboratorio es el robot UNIMATE S-103 de arquitectura SCARA.

Para representar un robot con realidad virtual, es necesario conocer su forma y dimensiones geométricas, para que la reconstrucción tridimensional sea lo más parecida al robot real. Sin embargo hace falta conocer los modelos matemáticos del robot, para que el robot virtual se pueda manipular y mover de forma similar a como se haría con el robot real.

Los modelos matemáticos del robot, se refieren básicamente a su cinemática. En el modelo de cinemática directa, se desea conocer la posición del órgano terminal, conociendo las posiciones de las articulaciones del robot, mientras que en la cinemática inversa se conoce la posición del órgano terminal y se intenta conocer las coordenadas articulares del robot.

Conocidos los modelos matemáticos, geométricos y cinemáticos del robot, es posible su modelación virtual a través de *VRML*. Así es posible hacer que el modelo desarrollado se vea, se mueva y se comporte como el robot real y de está manera incluso sustituir al robot real por el robot virtual en la mayoría de las practicas de manejo de robots. Para mover al robot en el espacio virtual se usan sensores de tacto e interpoladores de posición y de tiempo, cuando se encuentra modelado en *VRML*, es posible manipularlo mediante Internet, a través de programas de comunicación escritos en *Java*.

La visión artificial en robótica permite a los robots actuar con características de adaptabilidad, flexibilidad y capacidad de reorganización, permitiéndoles desempeñar tareas de inspección, identificación, medición y control. Por lo que la inclusión de sistemas de visión en robots seguirá ampliándose en los próximos años.

Capítulo 5: Diseño e Implementación de un Sistema de Visión

En los capítulos anteriores se han presentado algunas de las consideraciones para el diseño e implementación de robots, ambientes virtuales, sistemas de visión artificial, y del *Laboratorio de Robótica Virtual* del CINVESTAV. En este capítulo se presenta el diseño, implementación y evaluación del sistema de visón para el robot UNIMATE S-103, alrededor del cual se desarrollo el laboratorio en comento.

Para poder dotar a un robot de un *sistema de visión*, es necesario contemplar el uso de algunos de los métodos descritos en los capítulos anteriores. Aquí describimos la forma en que se pueden implementar los algoritmos necesarios, así como la forma en que pueden ser evaluados para poder predecir su comportamiento en otros ambientes y/o tipos de imágenes.

Los algoritmos necesarios para el *reconocimiento* se realizaron empleando el *análisis de momentos Hu* descrito en el **Capítulo 3: Visión Artificial**. Para que los datos producidos por el sistema de visión sean transmitidos al usuario del laboratorio vía Internet, fue necesario realizar los programas de comunicaciones. Se presentan tres propuestas para la comunicación, una basada en *URL*, otra en *datagramas* y una más en *Sockets.*

Se realiza un análisis del tiempo empleado por el sistema de reconocimiento para llevar a cabo su labor, mediante tres pruebas, una en donde se varía el número de los objetos en la escena, otra en la que se modifica el tamaño de la imagen y una en donde se compara el tamaño de los archivos obtenidos después de todo el proceso de reconocimiento de una imagen y el tamaño del la imagen misma.

El capítulo concluye con una breve descripción de las comunicaciones en Internet a través de *Java*, que es el lenguaje usado para transferir la información de la escena proveniente del espacio de trabajo del robot, al espacio virtual del usuario y en donde se analizan las dos ultimas propuestas de comunicaciones (*datagramas* y *Sockets*).

5.1 Motivación

Como se explicó en el **Capítulo 4: Laboratorio de Robótica Virtual**, el *Laboratorio de Robótica Virtual* fue desarrollado con el propósito de realizar prácticas de programación y control de un puesto de trabajo robotizado, para efectuar tareas de manipulación usando sistemas de *visión artificial.* Su diseño incluye los módulos de controlador del robot, emulación de la botonera (*teach pendant)* en una terminal, observación de imágenes a través de una cámara web, sistema de visión, control de cámara, servidor de páginas web y una interfaz para el usuario. Actualmente se han desarrollado y se tienen funcionando los módulos de controlador del robot (de fábrica), emulación de la botonera (*teach pendant)* en una terminal (en visual basic), observación de imágenes a través de una cámara web (monitoreo remoto con programas comerciales), control de cámara (software propio de la cámara), servidor de páginas web (con html -*hypertext markup language*-) y la interfaz para el usuario (modelo del robot en VRML -*Virtual Reality Modeling Language*- y comunicaciones con CGI –*Common Gateway Interface*-).

El *Laboratorio de Robótica Virtual*, hasta antes del desarrollo del sistema de visión, solo permitía la interacción de forma limitada por parte del usuario con el robot UNIMATE S-103. Únicamente se podía manipular al robot de forma muy simple, como moverlo, o realizar programas pequeños. Con el sistema de visión ahora es posible conocer la cantidad de objetos que el robot puede manipular, pero también su posición, tamaño y orientación de los mismos. Además nuestro

sistema permite visualizar la mesa de trabajo del robot a distancia sin las incomodidades de la transferencia de imágenes a través de Internet, lo cual se logra empleando la modelación del espacio de trabajo del robot y al robot en un ambiente virtual.

Esto permite al robot UNIMATE S-103 ampliar sus capacidades de integración con el entorno, dándoles mayor libertad a los usuarios del robot al conocer las características de los objetos presentes en el espacio de trabajo del robot.

5.2 Aspectos de diseño

En sistemas industriales no existen sistemas de visión de propósito general. Los sistemas de visión usados en la industria presentan una alta especialización, es decir, son sistemas de propósito especial ó que se diseñan para una aplicación específica. Las consideraciones primarias de diseño para un sistema de visión industrial son *el costo, la velocidad, la precisión* y *la confiabilidad* [Corke 96]. La primera consideración es el *costo*, está tal vez es la limitación más grande en la implementación de un sistema de visión artificial. En ocasiones la sustitución de los humanos por sistemas de visión resulta ser muy costosa debido principalmente a los costos que involucra la calibración del sistema y a la gran diversidad de objetos que cada industria maneja.

Otros factores como son la *velocidad,* la *precisión* y la *confiabilidad* resultan ser poco relevantes, debido a que un sistema de visión presenta un mejor desempeño que su contraparte humana. En la mayoría de los casos, los sistemas de visión en tiempo real superan por mucho a los humanos. Por ejemplo, cuando se requiere contar una gran cantidad de piezas en un tiempo menor a un segundo, los humamos cometerían una gran cantidad de errores, mientras que una máquina podría contar una gran cantidad de piezas por segundo, durante el tiempo de producción con un mínimo de errores.

5.3 Diseño del sistema visión

El objetivo de un sistema de visión artificial es realizar una descripción simbólica de una imagen proveniente del mundo real. Para el caso del sistema de visión para el *Laboratorio de Robótica Virtual* el objetivo es el de localizar cada uno de los objetos que se encuentren presentes en el *espacio de trabajo* del robot UNIMATE S-103. El sistema de visión deberá de proveer al robot, de la información sobre el *tipo de objeto, localización, orientación* y *tamaño*, para que el robot pueda determinar como manipularlo y si es posible realizar alguna operación con el objeto, pero también debe de informar a los usuarios locales o remotos la situación de dichos objetos.

Una vez que se conocen los objetos presentes y sus características, el programa realiza la reconstrucción tridimensional que representa el *espacio de trabajo real del robot*, usando *VRML* (*Virtual Reality Modeling Language*). Sin embargo, para la información del usuario local o remoto resulta más práctico enviar a través de Internet un archivo con las características de los objetos y, realizar en el servidor del robot la reconstrucción en el espacio tridimensional de los objetos a partir de estos datos, por que el modelo del robot no cambia. Así solo se añaden los objetos al *robot virtual* ya modelado.

Las comunicaciones son necesarias para informar a los *usuarios locales* (son los usuarios que se encuentran en una terminal cercana al robot como se observa en la **Figura 4.1**) o *remotos* (son los usuarios que acceden al robot a través de Internet como se observa en la **Figura 4.1**) la posición no solo de los objetos, sino también del robot, de las herramientas y muebles que se encuentren alrededor del robot, etc. Para las comunicaciones se presentan tres soluciones, por conexión con *URL* (*Uniform Resource Locators*) usado en la Web para localizar datos en *Internet*, mediante el empleo de *sockets* de flujo basados en conexiones y con *datagramas* con transmisión sin conexiones. La **Figura 5.1** muestra el esquema para una conexión con *URL.*

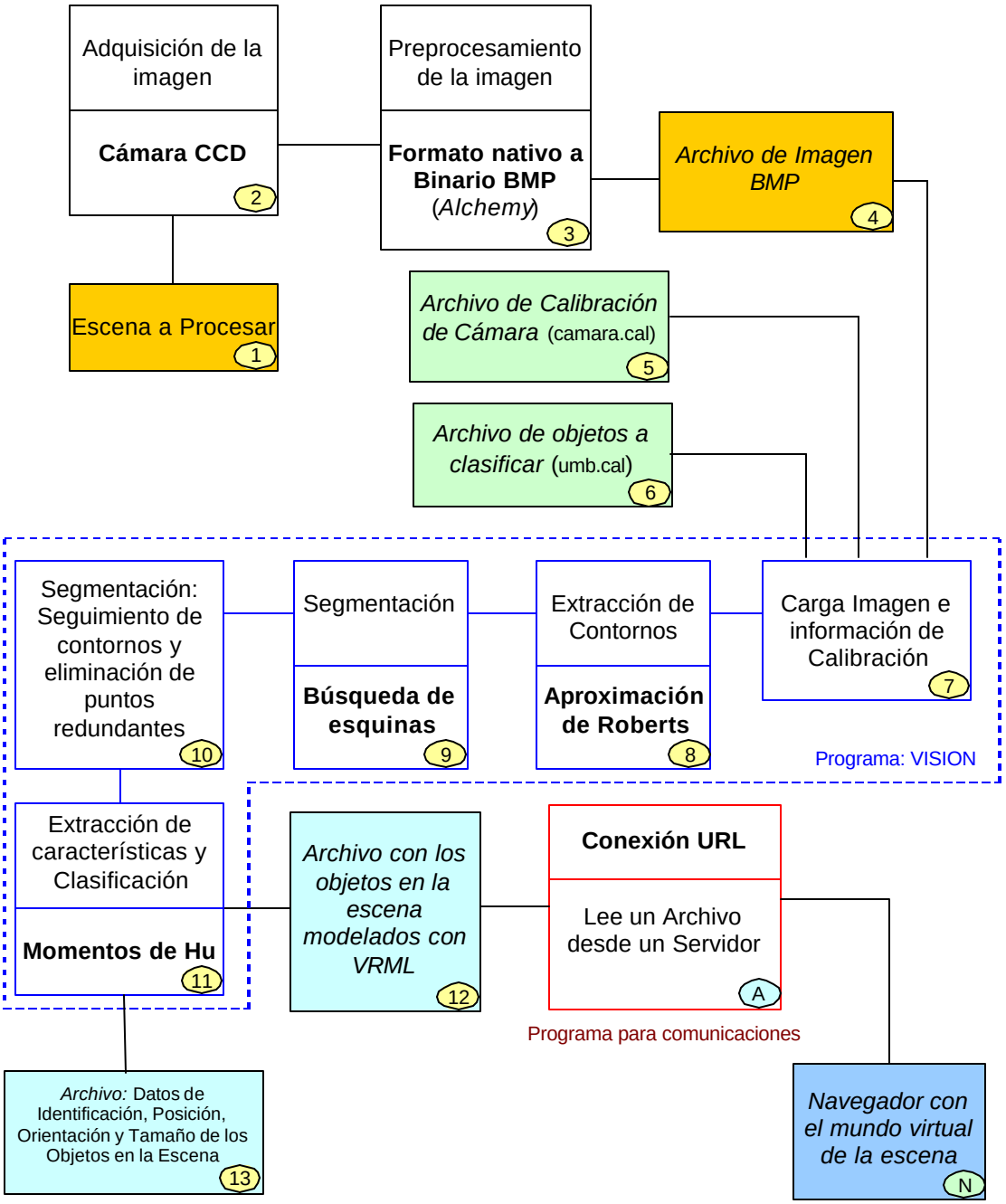


Figura 5.1: Sistema de Visión para el Robot UNIMATE S-103 con comunicaciones con conexión a través de URL.

La *escena a procesar* (**Figura 5.1 -módulo 1**), está conformada solo por el espacio de trabajo del robot UNIMATE S-103 y los objetos presentes dentro de este espacio, por lo que los objetos que se encuentren alrededor de dicho (controles, mecanismos, herramientas, etc.) espacio serán descartados.

La *adquisición de imágenes* (**Figura 5.1 -módulo 2**), se realiza a través de una cámara Web de estado sólido (**Figura 5.2** y **Apéndice A: Sistemas de Adquisición de Imágenes**). Esta cámara convierte a formato digital la imagen, por lo que puede ser inmediatamente procesada por el resto del programa. La resolución de las cámaras de este tipo es lo suficientemente buena como para poder localizar y reconocer las características de los objetos que se encuentren en su espacio de visión, de hecho solo se usa una cuarta parte aproximadamente de su resolución máxima.

Marca:	**C.U.C.US**
Resolución máxima:	**704x576 píxeles**
Sensibilidad:	**5 Lux**

Figura 5.2: Cámara Web usada para adquisición de imágenes.

Una vez que la imagen es capturada se tiene en formato de *mapa de bits* ó *BMP*, sin embargo existen una gran diversidad de formatos, si es necesario la imagen puede ser convertida a formato *BMP* mediante algún programa.

El *preprocesamiento de imágenes* (**Figura 5.1 -módulo 3**), resalta las características relevantes de una imagen para el sistema de visión. El procesamiento de imágenes se puede llevar a cabo junto con la conversión de formato, pues muchos de los programas usados, no solo cambian entre un formato a otro, sino que también permiten aplicar filtros, eliminar ruido, pasar a imágenes binarias, etc.

El programa diseñado para el reconocimiento de objetos puede leer imágenes en formato *BMP* (*Bitmap*.- imagen de mapa de bits), se usó el programa *Image Alchemy* para la conversión y preprocesamiento de las imágenes por su simplicidad y rapidez.

Alchemy [Alchemy 96] es un potente conversor de gráficos para *DOS* (*Sistema Operativo en Disco*), aunque existen versiones para otros sistemas operativos como *Windows* y *Linux*. Aparte de las conversiones entre múltiples formatos (más de 60), se pueden modificar con facilidad las medidas, paletas (colores) y otras características de la imagen. Se llama mediante la instrucción:

```
C:\>Alchemy [opciones] archivo fuente [archivo destino]
```

Para adecuar las imágenes a analizar, es necesario pasarlas al formato de *mapa de bit monocromático* (*imágenes binarias*) (**Figura 5.1 -módulo 4**), para esto se emplea la instrucción:

```
C:\>Alchemy –b –w –c2 –d0 [archivo fuente]
```

Donde:	-b	→	Imagen Blanco y Negro
	-w	→	Imagen Windows BitMap
	-c2	→	Colores de Salida = 2
	-d0	→	Difusión = 0

Otros programas pueden ser empleados para cubrir este propósito. Algunos de estos programas se describen a continuación:

Vpic.- es un conversor y visualizador de gráficos de distribución libre.

Pv.- es un conversor y visualizador de gráficos de distribución libre, con algunas particularidades en el ámbito de procesamiento de imágenes. Posee diferentes filtros y métodos de tramado.

PaintShop Pro.- permite ver y realizar la conversión entre gráficos, además de permitir la edición de los mismos, así como la corrección del color y filtros diversos.

Workshop.- es un conversor y visualizador de gráficos.

Jasc Image Robot.- es un procesador de imágenes por lotes. Permite aplicar una serie de instrucciones de conversión, filtrado y realce a una imagen o a un grupo de imágenes.

ImageCommander.- es un programa de visualización y conversión de imágenes.

El *archivo de calibración de cámara* (**Figura 5.1 -módulo 5**), contiene la información necesaria para convertir entre coordenadas de imagen a coordenadas del robot, haciendo posible que el robot pueda manipular los objetos que el sistema de visión localice (**Figura 5.3**).

```
Escala en el eje x = 9.17 píxeles/cm
Escala en el eje y = 9.11 píxeles/cm
Ángulo = 0.18º
Desplazamiento en x = 0.31 cm
Desplazamiento en y = -0.14 cm
```

Figura 5.3: Datos del archivo de calibración de cámara `camara.cal`.

El archivo de objetos a clasificar (**Figura 5.1 -módulo 6**), almacena los datos de los objetos que pueden ser reconocidos. Contiene la información relevante de cada uno de los objetos que el sistema puede reconocer (**Figura 5.4**).

Identificación	Nombre	Parámetros de identificación					
1	Triángulo	1.2000	1.4000	0.0015	0.0080	0.0000	2.0000
2	Cuadrado	2.1300	3.5000	0.0025	0.0100	0.0055	2.0000
3	Rectángulo	3.9000	5.0000	3.0000	4.0000	0.0000	2.0000
4	Hexágono	1.8000	1.9500	0.0025	0.0150	0.0000	2.0000
5	Círculo	2.0000	2.2000	0.0000	0.0035	0.0000	0.0050
6	Indicador	0.4000	0.6500	0.0000	0.0070	0.0000	2.0000

Figura 5.4: Archivo de umbrales `umb.cal`.

Cuando la imagen proveniente de la escena del espacio de trabajo del robot es capturada y procesada, el programa debe almacenarla para iniciar el proceso de *búsqueda de objetos* (**Figura 5.1 -módulo 7**). La *imagen de entrada* es *almacenada* en una matriz llamada *img(x,y),* en la cual cada bit, representa un píxel de la imagen, la matriz se muestra en la **Figura 5.5**.

0,0	1,0	....	x-1,0	x,0
0,1	1,1	....	x-1,1	x,1
....	....	....	....	....
0,y-1	1,y-1	....	x-1,y-1	x,y-1
0,y	1,y	....	x-1,y	x,y

240 líneas

40 bytes = *320* columnas

Figura 5.5: Matriz de almacenamiento de imagen `img(x,y)`.

La *imagen de entrada* presenta el formato *BMP* con un tamaño de *320x240 píxeles*, es decir, *320 columnas* por *240 líneas*, siendo este un tamaño adecuado para realizar el reconocimiento (*76,800 píxeles*). También se leen los archivos de calibración de cámara y los datos de los objetos a clasificar.

La *extracción de contornos* (**Figura 5.1 -módulo 8**), se realiza por el método de *aproximación de Roberts*, este procedimiento consiste en la aplicación de dos máscaras sucesivas para calcular el vector gradiente en las direcciones diagonales. Las máscaras se describen en el **Capítulo 3: Visión Artificial** y son las que se muestran en la **Figura 5.6**.

0	1
-1	0

y

1	0
0	-1

Figura 5.6: Máscaras usadas para la extracción de contornos.

El algoritmo usado recorre la imagen `img(x,y)` calculando el valor del punto donde se ubique la esquina de cada máscara [Schidlt 92, Ibarra 98]. Este proceso se lleva a cabo usando el **Programa 5.1**.

```
1.    xmax = ancho de la imagen
2.    ymax = alto de la imagen
3.    for(j=0;j<ymax;j++)           /* Recorre líneas                  */
4.        for(i=0;i<xmax;i++)       /* Recorre columnas                */
5.          a=img(i,j)              /* Carga el primer bit             */
6.          b=img(i+1,j+1)          /* Carga el segundo bit            */
7.          c=img(i,j+1)                  /* Carga el tercer bit       */
8.          d=img(i+1,j)                  /* Carga el cuarto bit       */
9.          img(x,y)=abs(a-b)+abs(c-d)    /* Aplica la máscara         */
10.         if(img(x,y) > 0) img(x,y) = 1 /*   Hace   binaria   la   imagen
resultante */
```

Programa 5.1: Algoritmo para extracción de contornos.

Una vez que la imagen se reduce a los contornos, es necesario fragmentarla en objetos, éste proceso es conocido como *segmentación* y se realiza mediante *búsqueda de esquinas* (**Figura 5.1 -módulo 9**). Se usa un algoritmo que busca esquinas definidas de alguna forma. En este caso se busca una esquina definida de la forma que muestra en la **Figura 5.7**.

0 (x-4,y-1)	***0*** (x-3,y-1)	***0*** (x-2,y-1)	***0*** (x-1,y-1)	***0*** (x,y-1)	***0*** (x+1,y-1)
* No importa	* No importa	* No importa	* No importa	***1*** **(x,y)** Esquina	***0*** (x+1,y)
				Contorno =*1*	Fondo =*0*

Figura 5.7: Definición de esquina para segmentación.

Para buscar las esquinas de los objetos es necesario recorrer toda la imagen, sin embargo es posible que para un mismo objeto se encuentren dos o más esquinas. Cada esquina es almacenada como un punto de coordenadas `(x,y)`, el algoritmo se muestra en el **Programa 5.2**.

```
1.          xmax = ancho de la imagen
2.          ymax = alto de la imagen
3.          p = 0
4.          for(j=0;j<ymax;j++)          /* Recorre líneas            */
5.              for(i=0;i<xmax;i++)      /* Recorre columnas          */
6.                a=img(x-4,y-1)         /* Almacena píxel x-4,y-1    */
7.                b=img(x-3,y-1)         /* Almacena píxel x-3,y-1    */
8.                c=img(x-2,y-1)         /* Almacena píxel x-2,y-1    */
9.                d=img(x-1,y-1)         /* Almacena píxel x-1,y-1    */
10.               e=img(x,y-1)           /* Almacena píxel x,y-1      */
11.               f=img(x+1,y-1)         /* Almacena píxel x+1,y-1    */
12.               g=img(x,y)             /* Almacena píxel x,y        */
13.               h=img(x+1,y)           /* Almacena píxel x+1,y      */
14.               if(a=0 & b=0 & c=0 & d=0 & e=0 & f=0 & g=1 & h=0)
                  /* ¿Es esquina? */
15.                     punto.x(p)=i     /* Guarda x                  */
16.                     punto.y(p)=j     /* Guarda y                  */
17.                     p++              /* Siguiente punto           */
```

Programa 5.2: Algoritmo de búsqueda de esquinas para segmentación.

Este proceso solo encuentra las esquinas de los objetos, para terminar la segmentación es necesario encontrar todos los puntos que corresponde al contorno de cada objeto, por lo que el siguiente proceso es el de *seguimiento de contornos*.

El *seguimiento de contornos* (**Figura 5.1 -módulo 10**), permite realizar varias operaciones importantes para el reconocimiento, por un lado sigue un contorno almacenando los puntos que lo forman, luego marca los puntos de ese contorno como visitados (para evitar que regrese sobre su propio camino). Sí se encontraron dos esquinas en un mismo objeto, en este paso se eliminan. Se puede hacer la eliminación de ruido, pues se calcula el área del contorno. Si esta es muy pequeña, corresponde seguramente a ruido por lo que no es considerada como un objeto y simplemente se desecha. Para seguimiento de contornos se emplea una conectividad *4-conectado* (**Figura 5.8**). Cada uno de los vecinos de un punto se recorre en la dirección de las manecillas del reloj.

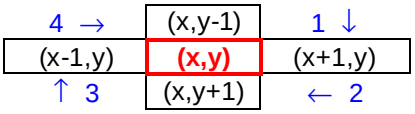


Figura 5.8: Seguimiento de contornos 4-conectados.

El algoritmo de *seguimiento de contornos* se muestra en el **Programa 5.3**, consiste básicamente en seguir el rastro de los píxeles que conforman cada contorno y marcando los píxeles como ya visitados, en caso de que el contorno no sea cerrado, se coloca un tiempo de vida después del cual el algoritmo se detiene y calcula los momentos del segmento de contorno hallado.

Una vez que son conocidas las coordenadas de los puntos que conforman un contorno, hace falta calcular los momentos para iniciar el reconocimiento.

La *extracción de características para clasificación* (**Figura 5.1 -módulo 11**), se realizó mediante el *cálculo de momentos de Hu* usando las ecuaciones de los capítulos anteriores. El algoritmo se muestra en el **Programa 5.4**, se emplean por que las características extraídas por este método son invariantes a *posición, escala* y *orientación* de los objetos bajo estudio.

```
1.    pcont=0                 /* Contador de puntos del contorno       */
2.    inicio(x,y)=punto(x,y)  /* Guarda el punto de inicio             */
3.    if(punto(x,y).visitado == TRUE)    /* ¿ Punto ya visitado?      */
4.        continua con la siguiente esquina;
5.    else
6.        punto(x,y).visitado = TRUE     /* Esquina como visitada     */
7.        i=x
8.        j=y
9.        while(inicio(x,y)!=punto(i,j))  /* Sigue el contorno        */
10.       if(punto(i,j).visitado == FALSE & punto(i+1,j)== TRUE) /* 1  */
11.            punto.sal(i,j)= punto(i+1,j)
12.            punto(i+1,j).visitado = TRUE
13.            i++
14.       if(punto(i,j).visitado == FALSE & punto(i,j+1)== TRUE) /* 2  */
15.            punto.sal(i,j)= punto(i,j+1)
16.            punto(i,j+1).visitado = TRUE
17.            j++
18.       if(punto(i,j).visitado == FALSE & punto(i-1,j)== TRUE) /* 3  */
19.            punto.sal(i,j)= punto(i-1,j)
20.            punto(i-1,j).visitado = TRUE
21.            i--
22.       if(punto(i,j).visitado == FALSE & punto(i,j-1)== TRUE) /* 4  */
23.            punto.sal(i,j)= punto(i,j-1)
24.            punto(i,j-1).visitado = TRUE
25.            j--
26.       pcont++                        /* Anexa punto hallado       */
27.       if(pcont>Umbral)               /* Elimina ruido o manchas   */
28.            calculo de momentos;      /* Va a calcular momentos    */
```

Programa 5.3: Algoritmo de seguimiento de contornos para segmentación.

```
1.    for(i=1;i<pcont-1;i++)       /* momentos de orden p+q           */
2.          m00++
3.          m10=auxx[i]+m10
4.          m01=auxy[i]+m01
5.    obj[pr].area=m00             /* Area                      */
6.    obj[pr].x=m10/m00            /* Centroide xy              */
7.    obj[pr].y=m01/m00            /* Centroide y               */
8.    for(i=1;i<pcont-1;i++)       /* Momentos centrales        */
9.          u20=(auxx[i]-obj[pr].x)*(auxx[i]-obj[pr].x)+u20
10.         u02=(auxy[i]-obj[pr].y)*(auxy[i]-obj[pr].y)+u02
11.         u11=(auxx[i]-obj[pr].x)*(auxy[i]-obj[pr].y)+u11
12.         u30=(auxx[i]-obj[pr].x)*(auxx[i]-obj[pr].x)*(auxx[i]-
obj[pr].x)+u30
13.         u03=(auxy[i]-obj[pr].y)*(auxy[i]-obj[pr].y)*(auxy[i]-
obj[pr].y)+u03
14.         u21=(auxx[i]-obj[pr].x)*(auxx[i]-obj[pr].x)*(auxy[i]-
obj[pr].y)+u21
15.         u12=(auxx[i]-obj[pr].x)*(auxy[i]-obj[pr].y)*(auxy[i]-
obj[pr].y)+u12
16.   obj[pr].ang=atan2(auxy[i]-obj[pr].y,auxx[i]-obj[pr].x)     /*
Ángulo */
```

Programa 5.4: Algoritmo de cálculo de momentos para reconocimiento (Parte **1**/2).

```
17.   obj[pr].dist=sqrt(obj[pr].dist)     /*  tamaño  del  centro  a  la
esquina     */
18.   cc1=pow(m00,2.)               /* Momentos normalizados
      */
19.   cc2=pow(m00,5./2.)
20.   n20=u20/cc1
21.   n02=u02/cc1
22.   n11=u11/cc1
23.   n30=u30/cc2
24.   n03=u03/cc2
25.   n21=u21/cc2
26.   n12=u12/cc2
27.   obj[pr].f1=fabs(n20)+fabs(n02)      /* Momentos de Hu o de tercer
orden       */
28.   obj[pr].f2=pow(n20-n02,2.)+4.*pow(n11,2.)
29.   obj[pr].f3=pow(n30-3.*n12,2.)+pow(3.*n21-n03,2.)
30.   obj[pr].f4=pow(n30+n12,2.)+pow(n21+n03,2.)
31.   obj[pr].id=0
32.   for(i=1;i<=pid;i++)                 /*     Clasificación     mediante
umbrales    */
33.         if(obj[pr].f1>=lima[i]      &&      obj[pr].f1<=limb[i]      &&
obj[pr].f2>=limc[i]
                  && obj[pr].f2<=limd[i])
34.               obj[pr].id=id[i];
```

Programa 5.4: Algoritmo de cálculo de momentos para reconocimiento (Parte **2**/2).

Posteriormente solo es necesario comparar los momentos calculados con los resultados de los análisis anteriores para saber a que categoría de objetos corresponden, lo que identifica a un objeto como perteneciente a una cierta clase. Este proceso de clasificación se lleva a cabo mediante una instrucción condicional, y simplemente se compara cada valor con los almacenados.

El programa almacena dos archivos: uno con los *objetos de la escena modelados con VRML* (**Figura 5.1 -módulo 12**) el cual puede ser visible vía Internet y otro con los *datos de los objetos hallados* (**Figura 5.1 -módulo 13**), necesario para que el robot conozca los objetos que puede manipular.

Para llevar a cabo las comunicaciones se empleo una conexión con *URL* (*Uniform Resource Locators*) (**Figura 5.1 -módulo A**), se programa en *lenguaje Java*, está aplicación estará ejecutándose para leer un archivo desde un servidor, en este caso desde el servidor de visión del *Laboratorio de Robótica Virtual*.

El *navegador de Internet* (**Figura 5.1 -módulo N**) permite visualizar la interfaz de usuario. Algunos de los navegadores empleados son *Netscape* e *Internet Explorer*. Un navegador Web proporciona una interfaz que encubre al usuario un complejo entramado de programación, como son los programas de comunicaciones.

En forma de resumen se muestran los pasos descritos anteriormente en la **Figura 5.9**, al aplicarlos a una imagen.

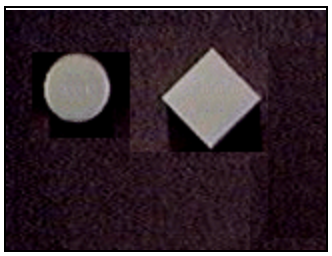
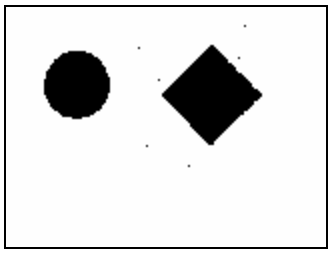
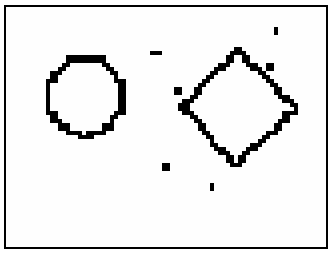
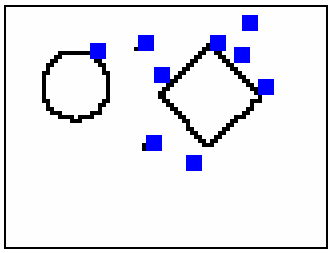
Proceso	Resultado
1) Escena a procesar. Mundo real del robot. **Figura 5.1 -módulo 1**	
2) Imagen de entrada, capturada con la cámara con un tipo BMP de 256 colores. **Figura 5.1 -módulo 2**	
3) Preprocesamiento, se convierte la imagen a BMP binaria, a través de un programa comercial. **Figura 5.1 -módulos 3 y 4**	
4) Extracción de contornos por aproximación de Roberts. **Figura 5.1 –módulos 7 y 8**	
5) Segmentación: Búsqueda de esquinas. Cada cuadro representa una esquina hallada por el algoritmo. **Figura 5.1 -módulo 9**	

Figura 5.9: Imágenes obtenidas al aplicar los algoritmos para reconocimiento (`Parte 1/2`).

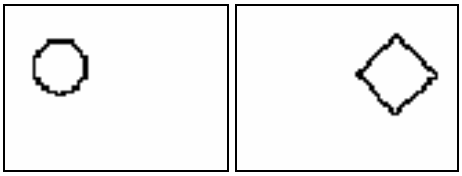
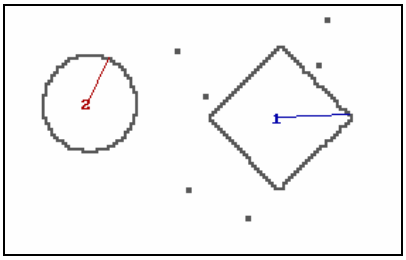
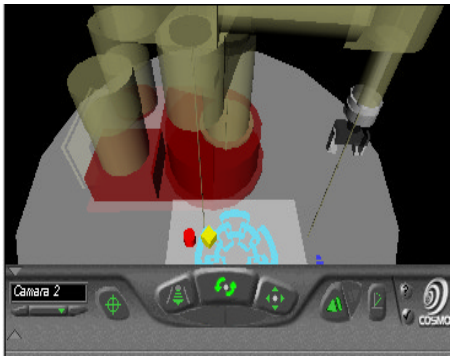
Proceso	Resultado
6) Segmentación: Seguimiento de Contornos. Por cada contorno hallado mayor a 15 píxeles, se considera como un objeto sujeto a identificación, por lo que el resto de objetos se elimina, pues se consideran como ruido. **Figura 5.1 -módulo 10**	
7) Cálculo de *momentos de Hu*. Para cada contorno hallado se calculan los parámetros necesarios para llevar a cabo una clasificación. **Figura 5.1 -módulos 11, 12 y 13**	 **Objeto 1:** Centro (x,y) = (101.8,44.4), Angulo =3.1º No. de píxeles =197 ϕ_1=2.151 ϕ_2=0.003 ϕ_3=0.014 Identificación = **Cuadrado** **Objeto 2:** Centro (x,y) = (34.4, 39.4), Angulo =65.2º No. de píxeles =131 ϕ_1=2.135 ϕ_2=0.000 ϕ_3=0.003 Identificación = **Circulo**
8) Presentación en un navegador de Internet. Consiste en la modelación de objetos y su incrustación en el mundo virtual del robot. **Figura 5.1 -módulos A y N**	

Figura 5.9: Imágenes obtenidas al aplicar los algoritmos para reconocimiento (Parte **2**/2).

5.4 Análisis del sistema de visión

En los capítulos anteriores se determinó, que la complejidad de los procedimientos empleados para el reconocimiento es cuadrática, es decir, que el tiempo empleado en el proceso se incrementa de esa forma. Para apreciar el tiempo que le toma a nuestro sistema en general, el reconocimiento de una imagen, se realizaron tres pruebas. La primera consistió en mantener el tamaño de la imagen constante mientras se incrementa el número de objetos en la escena, en la segunda se cambió el tamaño de la imagen pero se mantuvo constante el número de objetos en la escena, y en la tercera se comparó el tamaño de la imagen de entrada, en relación al archivo de salida el cual contiene la descripción de los objetos presentes en la escena.

Para la primera prueba, en donde se *mantiene constante el tamaño de la imagen y se varia el número de objetos*, el tamaño de la imagen se mantuvo fijo en *320*240* píxeles, mientras el número de objetos se incrementaba de *25* en *25* hasta llegar a *400*. Obviamente el robot nunca va a trabajar con tal cantidad de objetos (solo puede almacenar 94 puntos), pero permite obtener un modelo del comportamiento del algoritmo al incrementar el número de objetos por identificar. Los resultados nos indican el tiempo necesario que le toma al sistema de reconocimiento, detectar objetos sobre la mesa de trabajo del robot y para cuantos objetos resulta práctico emplearlo.

Los tiempos se midieron usando una *imagen sintética*. La imagen se genera distribuyendo los objetos uniformemente en toda la imagen. Cada objeto presenta un *área de 24 píxeles* con un *perímetro de 18 píxeles* y se ubican en forma alternada como se muestra en la **Figura 5.10**.

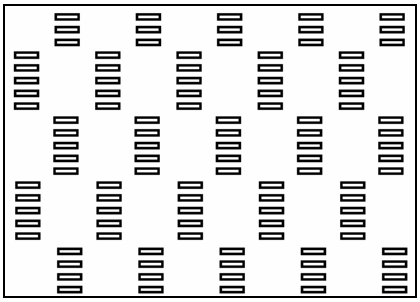


Figura 5.10: Imagen de prueba.

Se considera que los tiempos de lectura y almacenamiento de la imagen son constantes. Por cada variación de objetos se midió 50 veces el tiempo para obtenerlo de la forma más precisa posible, cada uno de estos promedios corresponde a t_1, t_2, t_3, t_4 y t_5. El resumen se muestra en la **Tabla 5.1** en donde se presentan también el resultado del *ajuste* de la curva realizado por el *método de mínimos cuadrados* [Chapra 96, Nakamura 92], y la gráfica resultante en la **Figura 5.11**.

Objetos (n)	t_1(s)	t_2(s)	t_3(s)	t_4(s)	t_5(s)	Media del tiempo (s)	Desviación Estándar	Imágenes / segundo	Ajuste
25	0.0385	0.0385	0.0385	0.0440	0.0385	0.0396	0.0022	25.2778	0.0396
50	0.0495	0.0440	0.0385	0.0440	0.0440	0.0440	0.0035	22.7500	0.0445
75	0.0549	0.0495	0.0440	0.0495	0.0549	0.0505	0.0041	19.7826	0.0493
100	0.0495	0.0549	0.0549	0.0549	0.0549	0.0538	0.0022	18.5714	0.0542
125	0.0549	0.0659	0.0549	0.0604	0.0659	0.0604	0.0049	16.5454	0.0591
150	0.0604	0.0604	0.0659	0.0659	0.0604	0.0626	0.0027	15.9649	0.0640
175	0.0714	0.0714	0.0659	0.0659	0.0714	0.0692	0.0027	14.4444	0.0689
200	0.0714	0.0769	0.0714	0.0714	0.0769	0.0736	0.0027	13.5821	0.0738
225	0.0769	0.0769	0.0824	0.0769	0.0824	0.0791	0.0027	12.6389	0.0787
250	0.0824	0.0824	0.0824	0.0824	0.0824	0.0824	0.0000	12.1333	0.0836
275	0.0934	0.0824	0.0934	0.0879	0.0879	0.0890	0.0041	11.2346	0.0885
300	0.0934	0.0934	0.0934	0.0934	0.0934	0.0934	0.0000	10.7059	0.0934
325	0.0934	0.1044	0.0934	0.1044	0.0934	0.0978	0.0054	10.2247	0.0983
350	0.1044	0.1044	0.1044	0.0989	0.1044	0.1033	0.0022	9.6808	0.1032
375	0.1154	0.1044	0.1099	0.1044	0.1099	0.1088	0.0041	9.1919	0.1081
400	0.1099	0.1154	0.1099	0.1154	0.1154	0.1132	0.0027	8.8349	0.1130

Tamaño de la imagen = 320*240 píxeles — Computadora = Pentium III @ 450MHz
Área de los Objetos = 24 Píxeles — Perímetro de los Objetos = 18 Píxeles

Tabla 5.1: Tiempos medidos cuando se incrementa el número de objetos.

Se observa que la gráfica presenta un *comportamiento lineal*, lo que significa que el tiempo de ejecución varía de forma directamente proporcional a la forma en que se incrementen el número

de objetos en la escena. La ecuación que mejor se ajusta a la gráfica es la de una recta (**Ec. 5.1**).

Tiempo de ejecución (s) = 0.0346525 + 0.000195959n ...(Ec. 5.1)

Donde *n* es el número de objetos presentes en la escena.

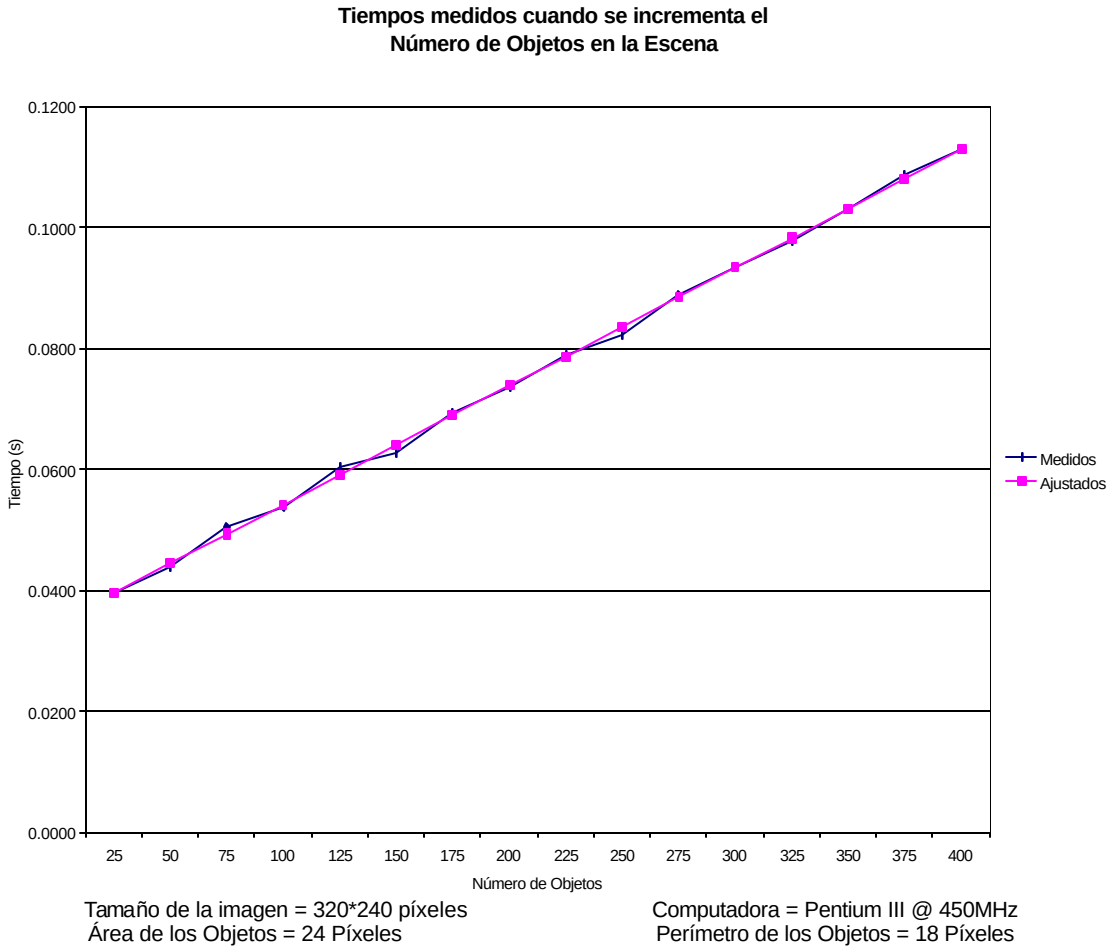


Figura 5.11: Gráfica con tamaño de imagen constante y número de objetos variable.

Para la *segunda prueba* se mantiene constante el número de objetos y varía el tamaño de la imagen. El tamaño de la imagen en la escena se incrementó desde una muy pequeña (*80*60 píxeles*), en la cual apenas se lograrían distinguir algunos detalles, hasta una de tamaño apropiado que permite reconocer la mayor parte de los objetos que pudiesen encontrarse en la *mesa de trabajo* del robot industrial (*320*240 píxeles*).

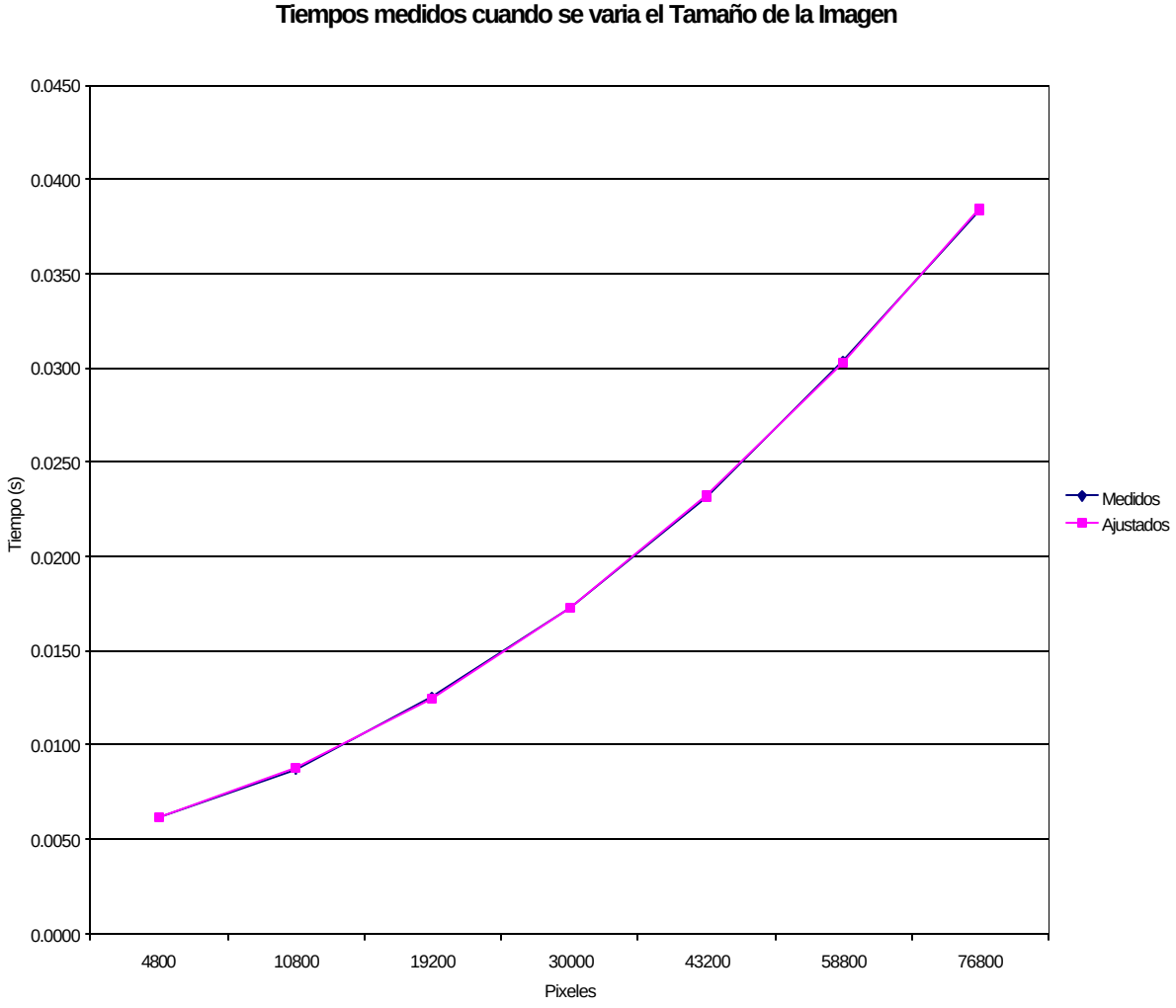
Los tiempos se midieron usando una imagen sintética, por lo que los tiempos para la lectura y almacenamiento de la imagen se consideran constantes. Por cada variación en el tamaño de la imagen se midió 100 veces el tiempo, para obtenerlo de la forma más precisa posible, cada uno de estos promedios corresponde a t_1, t_2, t_3, t_4 y t_5. Aquí es necesario incrementar el número de medidas por que los tiempos involucrados son menores.

El propósito de esta prueba es observar como se mejora o empeora el tiempo tomando imágenes de menor o mayor calidad, lo que implica un procesamiento de un mayor número de puntos (píxeles) por parte del programa. El resumen con los resultados obtenidos se muestra en la **Tabla 5.2** en donde se presentan también el resultado del *ajuste* de la curva realizado por el *método de mínimos cuadrados* [Chapra 96, Nakamura 92], y en la **Figura 5.12**.

Tamaño		Píxeles (p)	t_1(s)	t_2(s)	t_3(s)	t_4(s)	t_5(s)	MEDIA (s)	Desviación Estándar	Ajuste
x	y									
80	60	4800	0.0060	0.0060	0.0066	0.0060	0.0060	0.0062	0.0002	0.0061
120	90	10800	0.0088	0.0088	0.0082	0.0093	0.0082	0.0087	0.0004	0.0088
160	120	19200	0.0126	0.0126	0.0126	0.0126	0.0121	0.0125	0.0002	0.0125
200	150	30000	0.0176	0.0170	0.0176	0.0170	0.0170	0.0174	0.0003	0.0172
240	180	43200	0.0231	0.0231	0.0231	0.0231	0.0236	0.0232	0.0002	0.0230
280	210	58800	0.0308	0.0297	0.0308	0.0297	0.0308	0.0303	0.0005	0.0299
320	240	76800	0.0385	0.0385	0.0379	0.0385	0.0385	0.0383	0.0002	0.0378

Número de Objetos = 20 Computadora = Pentium III @ 450MHz
Área de los Objetos = 24 Píxeles Perímetro de los Objetos = 18 Píxeles

Tabla 5.2: Tiempos medidos cuando se modifica el tamaño de imagen.



Número de Objetos = 20 Computadora = Pentium III @ 450MHz
Área de los Objetos = 24 Píxeles Perímetro de los Objetos = 18 Píxeles

Figura 5.12: Gráfica con número de objetos constante y tamaño de imagen variable.

Se observa que el tiempo de ejecución en este caso se incrementa de forma más rápida que para el caso anterior, debido a que la complejidad de los algoritmos involucrados es *O(n*m),*

donde *n* es el número de filas y *m* es el número de columnas. Al ajustar la ecuación que representa este incremento se obtuvo un polinomio cuadrado (**Ec. 5.2**).

$$\text{Tiempo de ejecución (s)} = 0.00401998 + 0.0000043951p + 1.1049*10^{-13}\,p^2 \qquad \text{...(Ec. 5.2)}$$

Con *p* como el número de píxeles que conforman una imagen de la escena.

Los tiempos de ejecución resultan ser lo suficientemente pequeños, como para procesar varias imágenes durante un segundo.

En la *tercera prueba* los tamaños del archivo de salida se compararon con relación al tamaño de imagen**.** El número de objetos en este caso se incremento de *25* en *25* hasta llegar a *400*, observando en esta ocasión el archivo de salida generado, con el propósito conocer la conveniencia de enviar a través de *Internet* el archivo con la descripción de los objetos o la imagen completa. Los resultados se muestran en la **Tabla 5.3** y en la **Figura 5.13**.

No. de Objetos	Tamaño de Archivo de Salida Medido (Bytes)	Tamaño de Archivo de Salida Calculado (Bytes)	Tamaño de Imagen de Entrada (Bytes)
25	685.00	638.18	9,662.00
50	1,355.00	1,320.53	9,662.00
75	2,013.00	2,002.88	9,662.00
100	2,681.00	2,685.24	9,662.00
125	3,341.00	3,367.59	9,662.00
150	4,014.00	4,049.94	9,662.00
175	4,701.00	4,732.29	9,662.00
200	5,391.00	5,414.65	9,662.00
225	6,080.00	6,097.00	9,662.00
250	6,767.00	6,779.35	9,662.00
275	7,458.00	7,461.70	9,662.00
300	8,143.00	8,144.06	9,662.00
325	8,836.00	8,826.41	9,662.00
350	9,519.00	9,508.76	9,662.00
375	10,214.00	10,191.11	9,662.00
400	10,895.00	10,873.47	9,662.00

Tamaño de la imagen = 320*240 píxeles — Computadora = Pentium III @ 450MHz
Área de los Objetos = 24 Píxeles — Perímetro de los Objetos = 18 Píxeles

Tabla 5.3: Tamaños de archivos de salida y de Imagen, con un número de objetos variable.

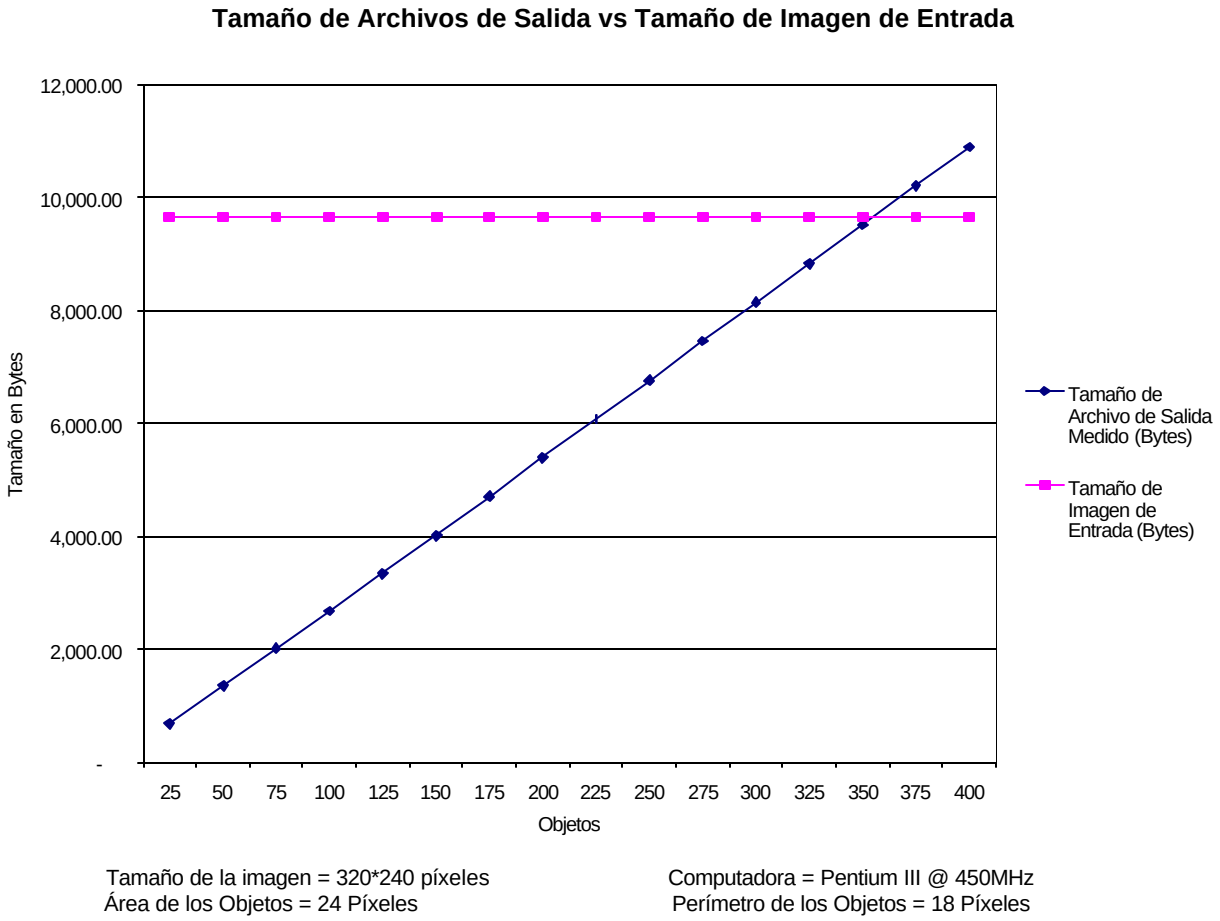


Figura 5.13: Gráfica de tamaños de archivos de salida y de imagen, con un número de objetos variable.

La ecuación que representa el crecimiento de los archivos de salida, conforme se incrementa el número de objetos se muestra en la **Ec. 5.3**.

Tamaño del archivo en Bytes = -44.175 +27.2941n **...(Ec. 5.3)**

Donde: *n* es el número de objetos es la escena.

Se observa que el número de objetos a reconocer produce un archivo menor o igual tamaño de la imagen de entrada para *n=355*, es decir, después de 355 objetos es mejor enviar la imagen y no el archivo de reconocimiento, pues el tamaño de la imagen es menor que para cuando el número de objetos es menor a *355*, en tal caso es mejor enviar el archivo de salida para la reconstrucción virtual. Sin embargo el robot UNIMATE S-103, solo puede almacenar hasta 94 puntos de su espacio de trabajo, si suponemos que almacena un objeto en cada punto, el tamaño del archivo con la descripción de los objetos *no* será mayor a *2,500 bytes*.

El programa para el sistema de visión para el *Laboratorio de Robótica Virtual* termina con los procedimientos de comunicación a través de Internet, realizados en el lenguaje de programación *Java*, por lo que a continuación se realiza una breve descripción de la forma en que se trabaja con este lenguaje.

5.5 Alternativas al acceso remoto

Java ofrece comunicaciones basadas en *sockets* [Deitel 98] que permiten a las aplicaciones manejar datos en redes como si fuera una entrada/salida de archivos. Tiene además *sockets de flujo* y *sockets de datagramas*. Con los *sockets de flujo* un proceso establece una conexión con otro proceso. Mientras la conexión existe, los datos fluyen entre los procesos en un flujo continuo. Así se dice que los *sockets de flujo* proporcionan un servicio orientado a conexiones. El protocolo empleado para la transmisión es *TCP* (*Transmission Control Protocol*).

Con los *sockets de datagrama*, se transmiten paquetes individuales de información, el protocolo empleado es *UDP* (*User Datagram Protocol*) (**Figura 5.14 –módulos B1, B2 y B3**), es un servicio sin conexiones y no garantiza que los paquetes lleguen en una forma en particular. De hecho los paquetes pueden perderse, duplicarse e incluso llegar en desorden. Sin embargo los servicios sin conexiones ofrecen mayor velocidad pero menor confiabilidad que los servicios orientados a conexiones.

El esquema empleado para la comunicación es el de *cliente – servidor* (**Figura 5.14 –módulo B2**). El *cliente* (**Figura 5.14 –módulo B1**) solicita la realización de una acción y el *servidor* (**Figura 5.14 –módulo B3**) realiza la acción devolviendo el resultado (si existe) al *cliente*. El *cliente* primero trata de establecer una conexión con el *servidor*. El servidor puede aceptar o rechazar la conexión. Si acepta la conexión; el *cliente* y el *servidor* se comunicarán a través de *sockets*. Cuando ya no se necesita la conexión, el *cliente* y el *servidor* cierran la conexión.

El *socket* es una norma de facto [Comer 97]. Una biblioteca de *sockets* puede ser ofrecida por las aplicaciones *API* (*Interfaz de programación de aplicaciones*) de un sistema que no ofrece *sockets* originales. Cuando una aplicación crea un *socket*, recibe un *descriptor* entero pequeño para hacer referencia a este. Si un sistema usa el mismo espacio de descriptor para los *sockets* y otras entradas y salidas, es posible emplear la misma aplicación para comunicación de red como para la transferencia local de datos.

Internet ofrece muchos protocolos [Deitel 98]. El protocolo *http* (*HyperText Transfer Protocol*) constituye la base del *World Wide Web* la cual emplea *URL* (*Uniform Resource Locators*) (**Figura 5.1**) para localizar datos en *Internet*. Si se conoce una *URL* de archivos *HTML* (*HyperText Markup Language*), se puede acceder a estos a través de *http*.

Para realizar una conexión de flujo de *URL* para leer un archivo desde un *servidor* en *Java*, se usa un objeto `URL` para abrir un `InputStream` de un archivo que esta en el servidor, leer el contenido del archivo y actualizar el mundo virtual. Para abrir una conexión de flujo con el archivo en el *servidor* se usa el enunciado:

```
input = fileURL.openStream();
```

Para crear un *servidor* a través de *sockets* de flujo se crea un `ServerSocket` y un `Socket`. El *servidor registra* el *número de puerto disponible*. El *cliente* pedirá conectarse con el *servidor* a través de ese *puerto*. El `ServerSocket` establece el puerto en el que el *servidor* esperará las conexiones de los *clientes*. Cada conexión se maneja con un `socket`. Es necesario crear un *flujo de salida* para que el *servidor* pueda enviar datos y un *flujo de entrada* para que pueda recibir datos.

Con *sockets* los programas en *Java* perciben la entrada y salida de la red como la entrada y la salida desde archivos. Los *sockets* ocultan gran parte de la complejidad de la programación de la red vista por el programador.

El *servidor* invoca el método `getOutputStream` para el *socket* a fin de obtener una referencia del flujo asociado, en el cual puede enviar información. Cuando se completa la transmisión el *servidor* cierra la conexión invocando el método `close` pare el *socket*.

Para el *cliente*, se usa un *socket* para conectarse con el servidor, un `inputStream` para recibir información del servidor y un `OutputStream` para enviar información al *servidor*.

La transmisión orientada a *conexiones* funciona de forma similar a como opera el servicio telefónico, en el que se marca y se recibe una conexión, que se mantiene mientras dura la llamada.

La transmisión *sin conexiones* con *datagramas* se parece al servicio postal. Si un mensaje grande no cabe en un solo sobre, se divide en varios fragmentos que se colocan en sobre individuales numeradas secuencialmente. Luego todas las cartas se envían juntas. Las cartas pueden llegar en orden, en desorden o no llegar. El receptor reensambla las piezas en orden secuencial antes de intentar entender el mensaje.

Para el uso de *datagramas* en *java*, la clase `Server` define `DatagramPacket` para crear paquetes con los que se enviará y recibirá la información, y `DatagramSocket` para enviar y recibir los paquetes.

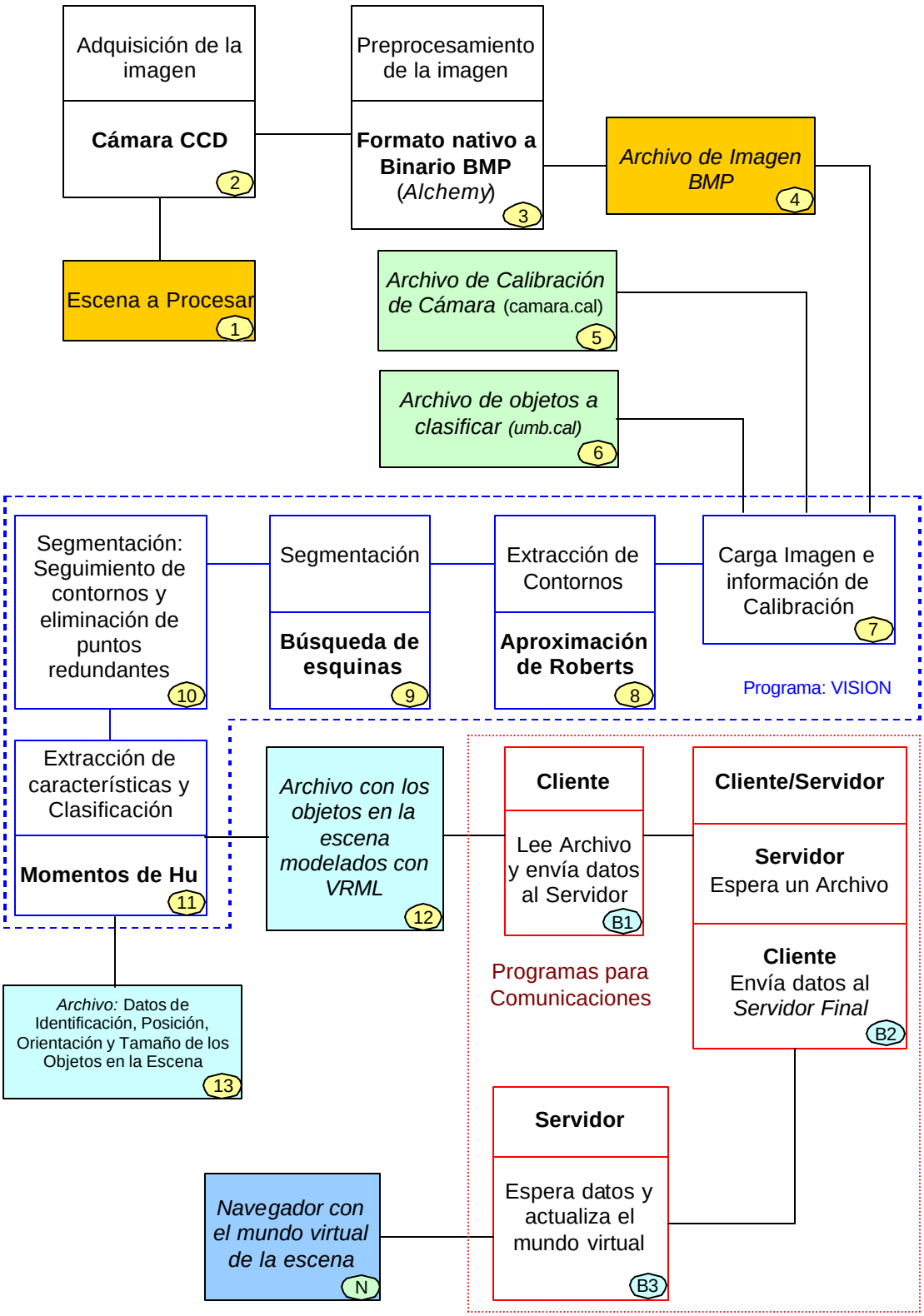


Figura 5.14: Sistema de Visión para el Robot UNIMATE S-103 con comunicaciones mediante Sockets y Datagramas.

Resumen

Para el diseño, implementación y evaluación del sistema de visón para el robot UNIMATE S-103, es necesario contemplar el uso de algunos de los métodos descritos en los capítulos anteriores.

En la implementación de un sistema de visión es necesario especificar claramente los procesos que serán empleados para cada una de las tareas que realiza. Se debe de especificar el tipo de formato empleado, se debe de determinar el tamaño de imagen adecuado para el reconocimiento, el tiempo necesario para el reconocimiento, el tipo de preprocesamiento a realizar, considerar los algoritmos para la segmentación y extracción de características, etc. Los algoritmos dependen de la aplicación específica, en nuestro sistema de visión destinado a dotar de este sentido al robot UNIMATE S-.103 del Laboratorio de Robótica Virtual; se decidió realizar el preprocesamiento mediante programas que se encuentran disponibles de forma gratuita en Internet, el formato elegido fue *BMP* (mapa de bits) de *320x240 píxeles*, la extracción de contornos se realizó por el método de *aproximación de Roberts*, la segmentación la realizamos por búsqueda de esquinas y seguimiento de contornos, y la extracción de características para clasificación, se obtuvo por el método de *cálculo de momentos de Hu*.

Se realizaron tres pruebas al sistema de reconocimiento. Para la primera se observó que para una cantidad de objetos que se incrementa en la escena, el tiempo se incrementa de forma lineal, sin embargo el tiempo es lo suficientemente pequeño, lo que permite procesar varias imágenes por segundo. En la segunda prueba, se incrementa el número de píxeles en la imagen, así el tiempo se incrementa de forma cuadrática, por lo que queda claro que esté es uno de los factores más importantes a considerar en la implementación del sistema de visión. Una última prueba demostró que para una cantidad reducida de objetos en la escena (hasta un poco más de 300), es más conveniente enviar el archivo con la descripción de cada uno de ellos, pero cuando se rebasa dicha cantidad es mejor enviar la imagen de la escena tal cual.

Para enviar los archivos a un usuario remoto, es necesario implementar los programas de comunicación, un lenguaje que permite realizar dichas operaciones de una forma sencilla es *Java*. Para las comunicaciones se presentan tres soluciones, por conexión con *URL* empleado para localizar datos en Internet, otra mediante el empleo de *sockets de flujo* basados en conexiones y otra con *datagramas* con transmisión sin conexiones.

Capítulo 6: Tareas del Sistema de Visión

En el **Capítulo 5: Diseño e Implementación de un Sistema de Visión**, presentamos el diseño e implementación de un sistema de visión para el *Laboratorio de Robótica Virtual*. El sistema desarrollado es aplicable solo a algunas aplicaciones específicas, como el reconocimiento de figuras simples, que si bien es suficiente para desarrollar prácticas de manipulación de objetos, en otras es necesario realizar algunas rutinas específicas de la aplicación, como veremos en el presente capítulo.

En este capítulo se describen algunas tareas del sistema de visión para el *Laboratorio de Robótica Virtual* que permitirán utilizarlo no solo para la docencia e investigación, sino también en sistemas industriales, y en áreas como física y astronomía. Presentamos cuatro tareas: reconocimiento de figuras geométricas, reconocimiento de dígitos, análisis de objetos en movimiento y estimación de la altura de los objetos.

Para cada una de estas tareas se realiza una breve descripción, se presenta la forma de obtener los parámetros de cada uno de los objetos para su reconocimiento, se muestran los problemas que se presentan en la implementación y los algoritmos necesarios para su resolución.

En la implementación del sistema de visión se empleo una *cámara fija*, por que facilitó su integración al programa. El programa requiere de una calibración simple y los errores producidos por el sistema pueden ser evaluados e incluso eliminados. Se describe la forma de calibrar la cámara para las aplicaciones anteriores.

Al final se realiza la modelación de los objetos hallados en el espacio tridimensional con *VRML* (*Virtual Reality Modeling Language*), para su visualización a través de *Internet*. Este último apartado es importante cuando los objetos reconocidos requieren de una presentación visual y puede omitirse si el robot solo requiere conocer las características de los objetos en la escena.

6.1 Reconocimiento de figuras geométricas

El reconocimiento de *figuras geométricas* es importante, por la cantidad de aplicaciones que se presentan. Algunas de estas aplicaciones son: **1)** en la navegación de robots móviles, al colocar marcas en paredes y pisos (incluso con los mismos letreros que ya existen), es posible que robot siga una trayectoria y encuentre un objetivo determinado; **2)** en el guiado de robots industriales, para localizar objetos y poder manipularlos (aplicación descrita en este apartado); **3)** en el análisis de plantaciones, al detectar el crecimiento de algunas plantas que se pueden aproximar a figuras geométricas como círculos, cuadrados, etc.; **4)** en control de calidad, para verificación de etiquetas, inspección de contenedores, inspección de circuitos impresos, etc.; **5)** en biomedicina, para el análisis del *DNA*, al hallar cadenas especificas de proteínas.

Las *figuras geométricas* son fáciles de reconocer por los métodos descritos en los capítulos anteriores, y también son fáciles de modelar pues solo requieren de las primitivas básicas de *VRML*. Nuestro sistema es capaz de localizar cada uno de los objetos geométricos que se encuentren presentes en el *espacio de trabajo* de un robot industrial, por lo que el sistema de visión debe de proveer al robot, de la información sobre el *tipo de objeto, localización, orientación* y *tamaño*, para que el robot pueda determinar como manipularlo.

Una vez que se conocen los objetos presentes y sus características, el programa realiza la reconstrucción tridimensional de los objetos que representen de forma fiel el *espacio de trabajo real del robot*; para este propósito se empleo *VRML*.

Para que el programa pueda reconocer cada uno de los objetos presentes en su mesa de trabajo, es necesario llevar a cabo una descripción detallada de cada uno de ellos. Está descripción se obtiene mediante procedimientos estadísticos, en donde se realiza una cierta cantidad de mediciones de cada uno de los parámetros buscados y se seleccionan los parámetros de engloben a la mayoría. En nuestro caso se tomaron los valores máximos y mínimos para cada uno de estos parámetros, para así poder reconocer a todos los objetos de una categoría.

Primero hay que determinar la cantidad y el tipo de objetos que el sistema de visión será capaz de reconocer. En nuestro sistema la cantidad de objetos se limitó a seis *figuras geométricas*: triángulo, cuadrado, rectángulo, hexágono, círculo y un círculo indicador (**Figura 6.1**).

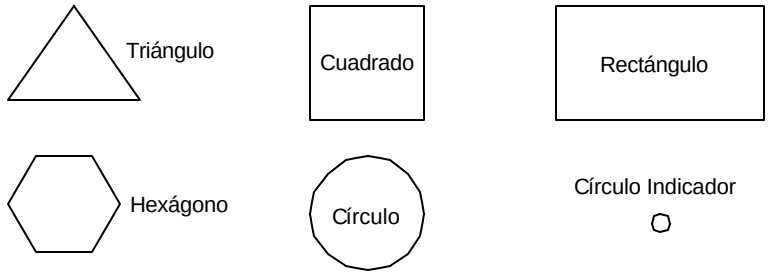


Figura 6.1: Objetos usados para el reconocimiento de figuras geométricas.

Posteriormente se determinan los parámetros de cada uno de estos objetos de acuerdo al método empleado, en nuestro sistema el método empleado es el de reconocimiento por el método de *momentos de Hu*, el cual consiste en determinar una serie de parámetros numéricos (ϕ_1, ϕ_2, ϕ_3, …ϕ_h), los cuales determinan las características de cada categoría de objetos.

Para hallar los parámetros necesarios para cada objeto fue necesario tomar varias fotografías (alrededor de 50) de cada objeto en diferentes posiciones tratando de abarcar todo el espacio de trabajo del robot, y rotándolos cada vez. Así se obtuvieron las variaciones mínimas y máximas que representen las características propias de cada objeto; por lo que el archivo que almacena dicha información, tiene un par de valores por cada momento, que corresponden a los límites inferior y superior de los valores medidos para cada momento. El formato del archivo se muestra en la **Figura 6.2**.

Número de Objetos = n					
1	Nombre 1	Límite Inferior ϕ_1	Límite Superior ϕ_1	Límite Inferior ϕ_2	Límite Superior ϕ_2…
2	Nombre 2	Límite Inferior ϕ_1	Límite Superior ϕ_1	Límite Inferior ϕ_2	Límite Superior ϕ_2…
.....	.....	.....	.....	.....	.....
n	Nombre n	Límite Inferior ϕ_1	Límite Superior ϕ_1	Límite Inferior ϕ_2	Límite Superior ϕ_2…

Figura 6.2: Formato del archivo de límites de momentos `umb.cal`.

Donde el *número de objetos n*, es el número de objetos que el sistema puede reconocer; el *nombre*, puede ser cualquier cadena de texto sin espacios, solo sirve para designar al objeto; los *límites inferior* f_1, *superior* f_1, *inferior* f_2 y *superior* f_2 corresponden a los *momentos de Hu* del objeto correspondiente. Para el caso de *reconocimiento de figuras geométricas*, solo fue necesario el cálculo de los primeros tres *momentos de Hu*, aunque si el número de objetos es menor, la cantidad de momentos también sería menor.

Los límites para los *momentos de Hu* hallados para los objetos propuestos se muestran en la **Figura 6.3** y las *figuras geométricas* se pueden armar con los patrones del **Apéndice D Patrones de Calibración y Figuras Geométricas**.

6 ← Número de objetos

Identificación	Nombre	f_1 Inferior	f_1 Superior	f_2 Inferior	f_2 Superior	f_3 Inferior	f_3 Superior
1	Triángulo	1.2000	1.4000	0.0015	0.0080	0.0000	2.0000
2	Cuadrado	2.1300	3.5000	0.0025	0.0100	0.0055	2.0000
3	Rectángulo	3.9000	5.0000	3.0000	4.0000	0.0000	2.0000
4	Hexágono	1.8000	1.9500	0.0025	0.0150	0.0000	2.0000
5	Circulo	2.0000	2.2000	0.0000	0.0035	0.0000	0.0050
6	Indicador	0.4000	0.6500	0.0000	0.0070	0.0000	2.0000

Figura 6.3: Archivo de umbrales `umb.cal` obtenido.

Las gráficas de estos momentos permiten observar los límites más claramente (**Figura 6.4**). Por una parte se tienen las figuras geométricas que se diferencian perfectamente de las demás desde el cálculo del primer momento, como son el *triángulo (1)*, el *rectángulo (3)*, el *hexágono (4)* y los *círculos indicadores (6),* mientras que el *cuadrado (2)* y el *círculo (5)* se traslapan entre los puntos `2.13` y `2.2`, por lo que si alguno de sus valores cae en este intervalo, el sistema de visión los confundirá. Para evitar ambigüedades es necesario tomar un mayor número de momentos, por lo que para el caso de figuras geométricas, fue necesario tomar los primeros tres *momentos de Hu*.

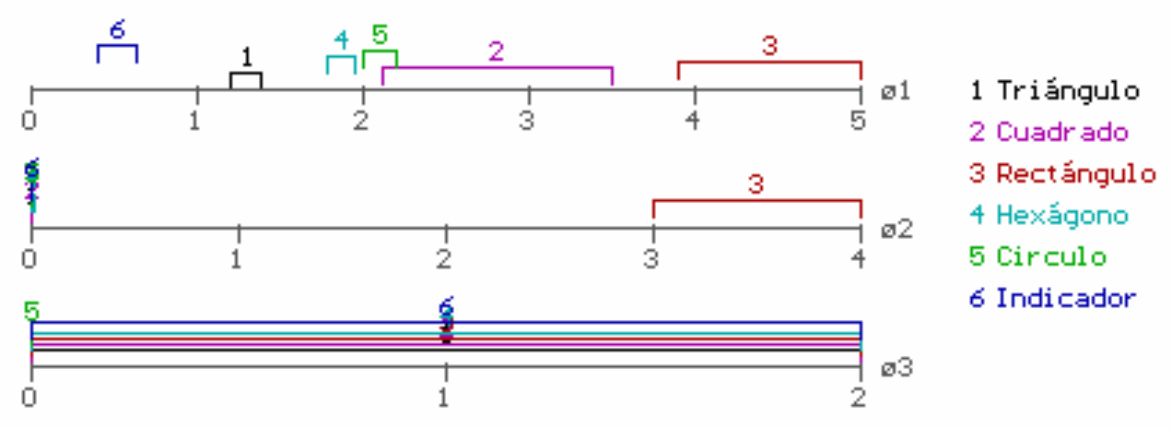
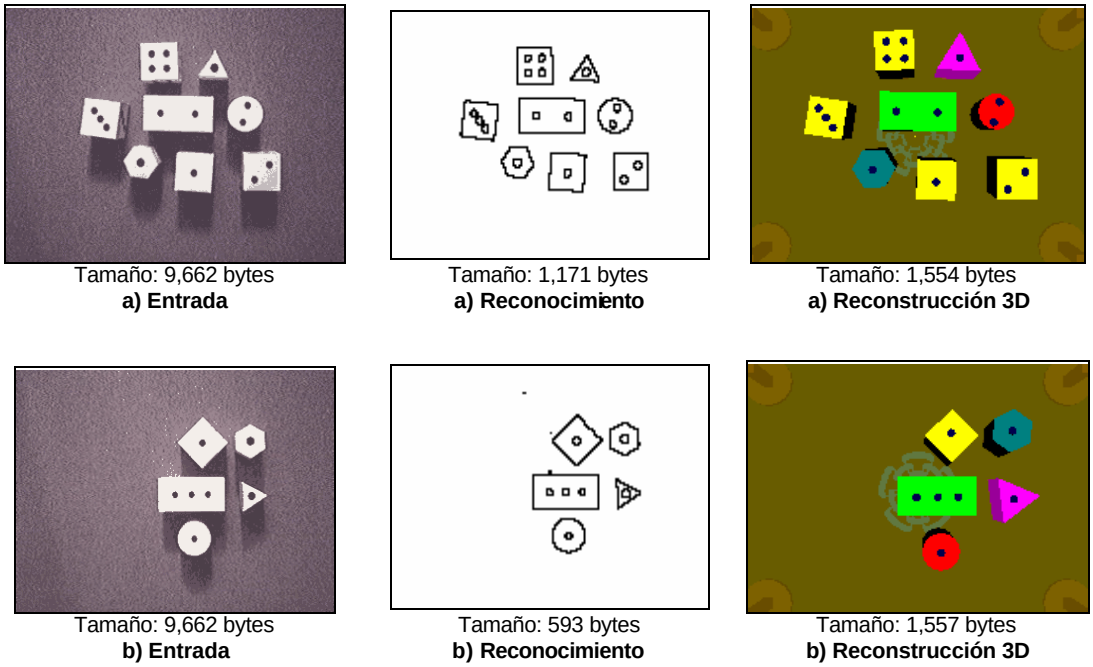


Figura 6.4: Distribución de los momentos ϕ_1, ϕ_2 y ϕ_3 para figuras geométricas.

Es interesante hacer notar de la **Figura 6.4** que la mayoría de las figuras geométricas se pueden diferenciar calculando solo un momento ϕ_1, es decir, si no se requiere reconocer el círculo o el cuadrado, bastará con el cálculo de un solo momento para reconocer todas las figuras geométricas restantes.

En la **Figura 6.5** se muestran los resultados obtenidos al aplicar nuestro sistema de visión a varias tomas fotográficas, se observa que fue posible reconocer todos los objetos presentes en las diferentes escenas que fueron fotografiadas, algunas de ellas con un poco de ruido para verificar la eliminación del mismo.

La **Figura 6.5** presenta en primera instancia la fotografía que es la entrada al nuestro sistema de reconocimiento, el tamaño del archivo de imagen generado es constante, pues la resolución siempre es la misma. En el proceso de reconocimiento, se obtiene un archivo con los objetos reconocidos por nuestro sistema, el tamaño de este archivo varia considerablemente, pues depende de la cantidad de objetos en la escena. En la última parte de la **Figura 6.5** se muestra la imagen obtenida después de realizar la reconstrucción tridimensional de la escena con *VRML*, el tamaño de este archivo no cambia significativamente, debido a la reducida cantidad de objetos que se agregan al mundo virtual.



Tamaño: 9,662 bytes **a) Entrada** | Tamaño: 1,171 bytes **a) Reconocimiento** | Tamaño: 1,554 bytes **a) Reconstrucción 3D**

Tamaño: 9,662 bytes **b) Entrada** | Tamaño: 593 bytes **b) Reconocimiento** | Tamaño: 1,557 bytes **b) Reconstrucción 3D**

Figura 6.5: Resultados del proceso de reconocimiento de figuras geométricas.

Para el reconocimiento y reconstrucción de *figuras geométricas*, el sistema de reconocimiento implementado, puede ser empleado sin ninguna modificación, debido a la simplicidad que representa la obtención de parámetros de cada uno de los objetos bajo estudio, sin embargo este proceso puede requerir de un poco de tiempo, pues para poder garantizar que se reconocerán los objetos en la mayor parte de los casos, es necesario llevar a cabo una gran cantidad de fotografías de los objetos en diferentes posiciones y orientaciones. La cantidad de momentos es mayor entre mayor sea la cantidad de objetos a reconocer, para evitar ambigüedades entre los objetos a reconocer.

6.2 Reconocimiento de dígitos

Los sistemas de reconocimiento de dígitos, son empleados en la industria por ejemplo para el control de inventarios, a través de la identificación automática mediante sistemas de visión de los números contenidos en las cajas; se pueden emplear para identificación de productos, si estos contienen en alguna parte un número por medio del cual puedan ser identificados [Erdei 92]. Sin embargo, el empleo de los sistemas de visión para este propósito, se encuentra muy restringido, debido a las facilidades que actualmente ofrece el uso de los códigos de barras, como son la existencia de estándar internacionales, que los equipos para su lectura son más económicos que los empleados en sistemas de visión y principalmente a que se han usado con gran éxito desde mediados del siglo pasado. A pesar de ello no deja de ser un tema interesante desde el punto de vista académico el reconocimiento de dígitos debido a los problemas inherentes a determinadas aplicaciones como la que aquí se describe.

En este caso, se requiere que el robot manipule una serie de cubos con un número impreso en uno de sus costados (con los dígitos 1, 2, 3, 4, 5, 6, 7 y 8), cada uno de los cubos deberá de encontrarse distribuido en una celda de una *matriz de 3x3*, quedando una de estas celdas vacía, para que el robot pueda utilizarla para mover un cubo a la vez dentro de la matriz como se muestra en la **Figura 6.6**. Para esto fue necesario idear un método que permitiera identificar las coordenadas de las celdas de la matriz y del lugar de la celda vacía, por lo que se anexaron algunos módulos al sistema de reconocimiento descrito en el **Capítulo 5 Diseño e Implementación de un Sistema de Visión**.

Matriz 3x3	**Columna 1**	**Columna 2**	**Columna 3**
Fila 1	1	2	3
Fila 2	4	5	6
Fila 3	7	8	Espacio Vacío

Figura 6.6: Matriz para reconocimiento de dígitos.

En los capítulos anteriores se realizó una pequeña descripción del funcionamiento de los sistemas de *Reconocimiento Óptico de Caracteres* u *OCR*. El proceso comienza cuando se tiene una imagen de entrada. En esta etapa, la imagen no es más que un conjunto de puntos sin sentido (píxeles) sobre un fondo. El programa de *OCR* tiene que extraer información bajo la forma texto de estos píxeles; ha de reconocer las formas y asignarles un símbolo. Los sistemas de *OCR* utilizan procesos de: segmentación de línea, segmentación de palabras y caracteres, reconocimiento de caracteres y producción de un archivo de salida de texto.

El *reconocimiento de dígitos* mediante el cálculo de *momentos de Hu*, no es el mejor método para reconocimiento de dígitos, sin embargo, como la cantidad de dígitos es pequeña es posible emplearlo. De igual forma que se hace con los *objetos geométricos*, la obtención de parámetros para cada uno de los dígitos, se realiza tomando fotografías de cada uno de ellos en diferentes posiciones y orientaciones, los valores para los *momentos de Hu*, solo que en esta ocasión la cantidad de momentos para realizar un reconocimiento aceptable es de cuatro momentos, pues la cantidad de objetos es mayor.

Un factor importante a considerar en esta aplicación, es que los *momentos de Hu* obtenidos solo funcionan para el tipo de caracteres empleados para la obtención de parámetros, por lo que sí se requiere reconocer los dígitos con un tipo de letra diferente es necesario obtener otros parámetros. Por que los parámetros cambian de acuerdo al tipo de letra empleado pues la forma de cada carácter cambia en cada tipo de letra. Los dígitos usados para la obtención de umbrales fueron realizados con el tipo de letra **Albertus** (**Figura 6.7** y **Apéndice D: Patrones de Calibración y Figuras Geométricas**).

Figura 6.7: Dígitos empleados para la obtención de momentos.

Después de realizar múltiples fotografías de cada número en diferentes posiciones y orientaciones, se obtuvieron los límites de la **Figura 6.8** y cuyas gráficas se muestran en la **Figura 6.9**, estos se almacenan en un archivo de texto llamado: `umbx.cal`.

De la **Figura 6.9** se observa que se traslapan los valores para la mayoría de los dígitos en cada uno de los momentos, sin embargo al emplear cuatro momentos, es posible evitar ambigüedades en el reconocimiento de cada uno de ellos.

8 ← Número de objetos

Identificación	Nombre	f_1 Inferior	f_1 Superior	f_2 Inferior	f_2 Superior	f_3 Inferior	f_3 Superior	f_4 Inferior	f_4 Superior
1	Uno	1.500	1.750	1.600	2.200	0.100	0.250	0.000	1.000
2	Dos	1.120	1.170	0.350	0.450	0.010	0.030	0.000	1.000
3	Tres	1.250	1.400	0.550	0.730	0.008	0.150	0.015	0.025
4	Cuatro	1.450	1.600	0.380	0.500	0.200	0.350	0.000	1.000
5	Cinco	1.200	1.350	0.450	0.600	0.025	0.075	0.001	0.014
6	Seis	1.550	1.780	0.550	0.715	0.205	0.300	0.000	1.000
7	Siete	1.350	1.800	0.725	1.200	0.450	0.999	0.000	1.000
8	Ocho	1.900	2.100	0.500	0.700	0.005	0.020	0.000	1.000

Figura 6.8: Archivo de umbrales `umbx.cal` obtenido.

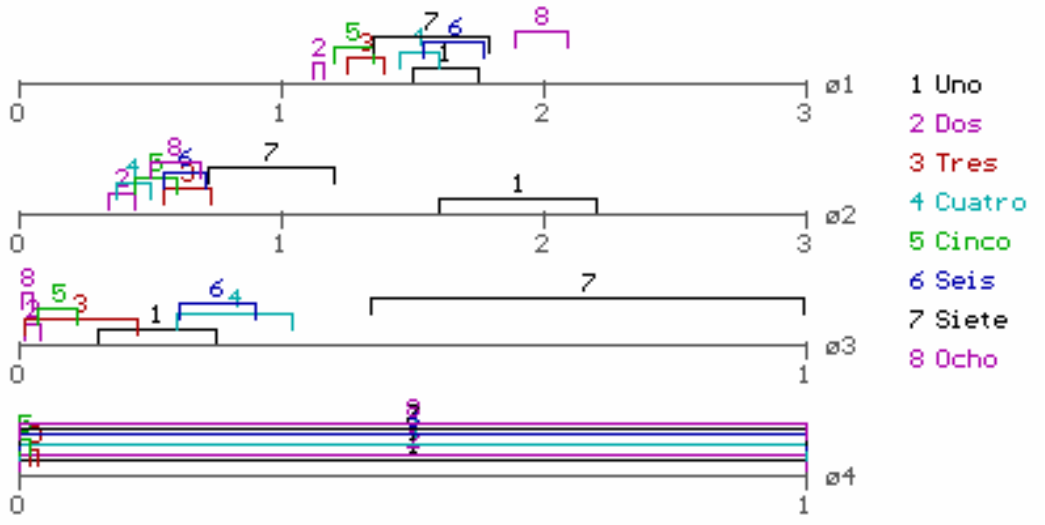


Figura 6.9: Distribución de momentos para el reconocimiento de dígitos.

El proceso de obtención de momentos para cada uno de los dígitos es similar al de reconocimiento de *figuras geométricas*, también el proceso de reconocimiento diseñado en el **Capitulo 5: Diseño e Implementación de un Sistema de Visión**, es el mismo hasta el cálculo de *momentos de Hu* (**Figura 6.15 -módulos 1** al **13, A** y **N**), sin embargo hace falta asignar a cada dígito reconocido, no solo su posición en la imagen y en la mesa de trabajo del robot, sino también determinar a que fila y columna pertenecen así como las coordenadas de la celda vacía, está no se encuentra presente en la imagen pero el robot la puede utilizar para manipular el resto de los objetos.

Para asignar las coordenadas a cada una de las celdas de la matriz hay que realizar un ordenamiento de las coordenadas de cada dígito hallado, para acomodarlos de esta forma en la celda de la matriz que le corresponda. Para esto se emplea el método de *ordenamiento por selección*, aplicado a cada dígito en cada una de sus dos coordenadas.

El *ordenamiento por selección* consiste en la secuencia de pasos mostrada en el **Programa 6.1** (**Figura 6.15 –módulo a**) [Heileman 98, Tenenbaum 93, Cormen 90].

```
1. Comenzar por la primera entrada.
2. Ver el resto de las entradas una por una. Cuando se encuentra una
   mayor se intercambia con la primera.
3. Ahora se reduce el listado. Se comienza por la segunda entrada y se
   busca en las entradas restantes. Se repite el paso anterior y se
   repite esté proceso sucesivamente.
4. Se continúa hasta que se realizan todos los procesos.
```

Programa 6.1: Secuencia del ordenamiento por selección.

El análisis del algoritmo consiste en observar, cuantas comparaciones se realizan, primero se realizan *n-1* comparaciones, luego *n-2* y así sucesivamente, por lo que la complejidad se obtiene por la **Ec. 6.1**.

$$(n - 1) + (n - 2) + ... + 1 = n*(n - 1) / 2 \qquad \text{...(Ec. 6.1)}$$

Se observa que el tiempo de ejecución del algoritmo tiene una complejidad de $O(n^2)$, sin embargo como el número de elementos que tiene que ordenar es extremadamente pequeño, no influye en el desempeño del programa de reconocimiento de dígitos.

Una vez que se tienen ordenados los dígitos identificados de acuerdo a sus posiciones con respecto a la referencia *X* y respecto a la referencia *Y*, es posible proceder a asignarles una posición en el tablero junto con la posición de la celda vacía.

Para realizar esto se desarrollo un algoritmo que analiza los resultados de los ordenamientos respecto a *X* y a *Y* (**Figura 6.15 –módulo b**), colocándolos en sus correspondientes celdas y dejando los valores que sobran para la celda vacía. El algoritmo desarrollado para este propósito sigue los pasos mostrados en el **Programa 6.2**.

```
1. Asignar al primer valor de cada ordenamiento las posiciones X1 y Y1,
   e ir llenando la matriz de la forma mostrada en la Figura 6.10.
2. Seguir asignando valores, mientras la diferencia entre el valor actual
   y el valor anterior no supere a la mitad de los valores asignados a X1
   y Y1.
3. Los espacios de X y Y no asignados corresponden a la posición de la
   celda vacía.
```

Programa 6.2: Secuencia de asignación de celdas.

x_1,y_1	x_2,y_1	x_3,y_1
x_1,y_2	x_2,y_2	x_3,y_2
x_1,y_3	x_2,y_3	x_3,y_3

Figura 6.10: Asignación de las celdas de la matriz de 3x3.

Veamos mediante un ejemplo como opera el algoritmo. Supongamos que se tienen los valores que se muestran en la **Tabla 6.1** y en la **Figura 6.11**.

#	X	Y
1	139.9	114.6
2	138.0	46.6
3	142.9	186.1
4	75.7	116.6
5	202.7	183.6
6	78.0	45.4
7	203.2	41.5
8	203.2	116.0

Datos de Origen

#x	X	Columnas
4	75.7	x_1
6	78.0	x_1
2	138.0	x_2
1	139.9	x_2
3	142.9	x_2
5	202.7	x_3
7	203.2	x_3
8	203.2	x_3

Ordenados respecto a **X**

#y	Y	Filas
7	41.5	y_1
6	45.4	y_1
2	46.6	y_1
1	114.6	y_2
8	116.0	y_2
4	116.6	y_2
5	183.6	y_3
3	186.1	y_3

Ordenados respecto a **Y**

Tabla 6.1: Asignación de posiciones en la matriz.

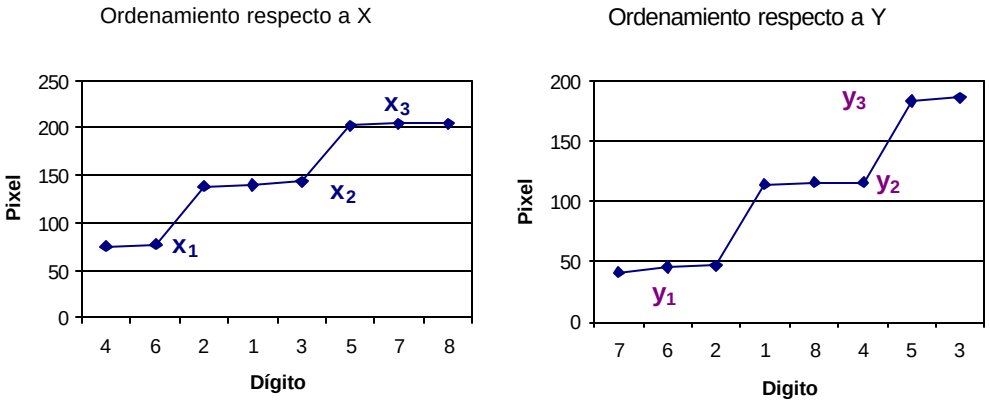


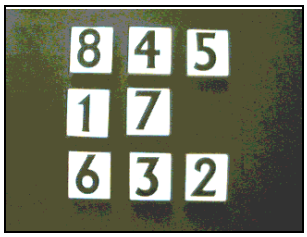
Figura 6.11: Ordenamientos de dígitos para asignación de posiciones en la matriz.

De la **Figura 6.11** se observa, por ejemplo, que el número *4* aparece en la posición x_1 y y_2, mientras que el lugar vacío corresponde a x_1 y y_3, pues es en estas posiciones donde solo aparecen dos valores para *X* y *Y*, de igual forma se puede encontrar la posición de los dígitos restantes. El resultado final aparece en la **Figura 6.12**.

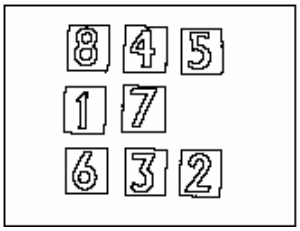
6	2	7
4	1	8
	3	5

Figura 6.12: Matriz resultante para el reconocimiento de dígitos.

Los resultados obtenidos son adecuados para este tipo de reconocimiento, tomando en cuenta que los límites de muchos de los dígitos reconocidos se traslapan en algunos de los momentos. En la **Figura 6.13** se ilustran los resultados aplicados a un par de fotografías de entrada.

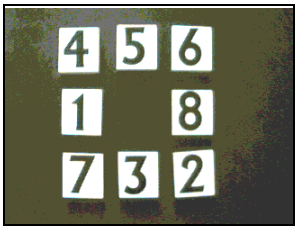


Tamaño: 9,662 bytes
a) Entrada: Ejemplo 1

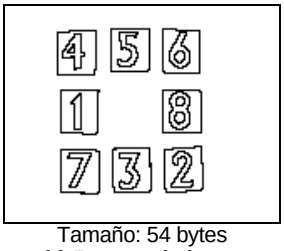


Tamaño: 54 bytes
a) Reconocimiento

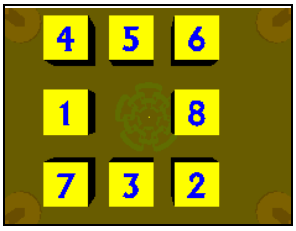
Tamaño: 10,777 bytes
a) Reconstrucción 3D



Tamaño: 9,662 bytes
b) Entrada: Ejemplo 2



Tamaño: 54 bytes
b) Reconocimiento



Tamaño: 10,775 bytes
b) Reconstrucción 3D

Figura 6.13: Resultados del proceso de reconocimiento de dígitos.

Los *archivos de salida del proceso de reconocimiento*, son archivos bastante pequeños, pues contienen los números en las posiciones correspondientes a cada celda (de la primera a la novena), el cero representa al lugar vacío y los números restantes a x_1, x_2, x_3, y_1, y_2 y y_3 a las coordenadas de los centros de las celdas (matriz de 3x3) (**Figura 6.14**).

```
Digito 1    Digito 2    Digito 3…          … Digito 8
x1    x2    x3     Coordenadas de los centros en el eje X
y1    y2    y3     Coordenadas de los centros en el eje Y
```

Formato del Archivo de Salida

```
8     4     5     1     7     0     6     3     2
87.2        152.3       214.2
44.6        109.2       181.4
```

a) Ejemplo 1

Figura 6.14: Archivos obtenidos después del proceso de reconocimiento de dígitos.

En la **Figura 6.15** se tiene el sistema de visión para reconocimiento de dígitos con los anexos (**módulos a** y **b)** necesarios para encontrar las coordenadas de la matriz y la celda vacía.

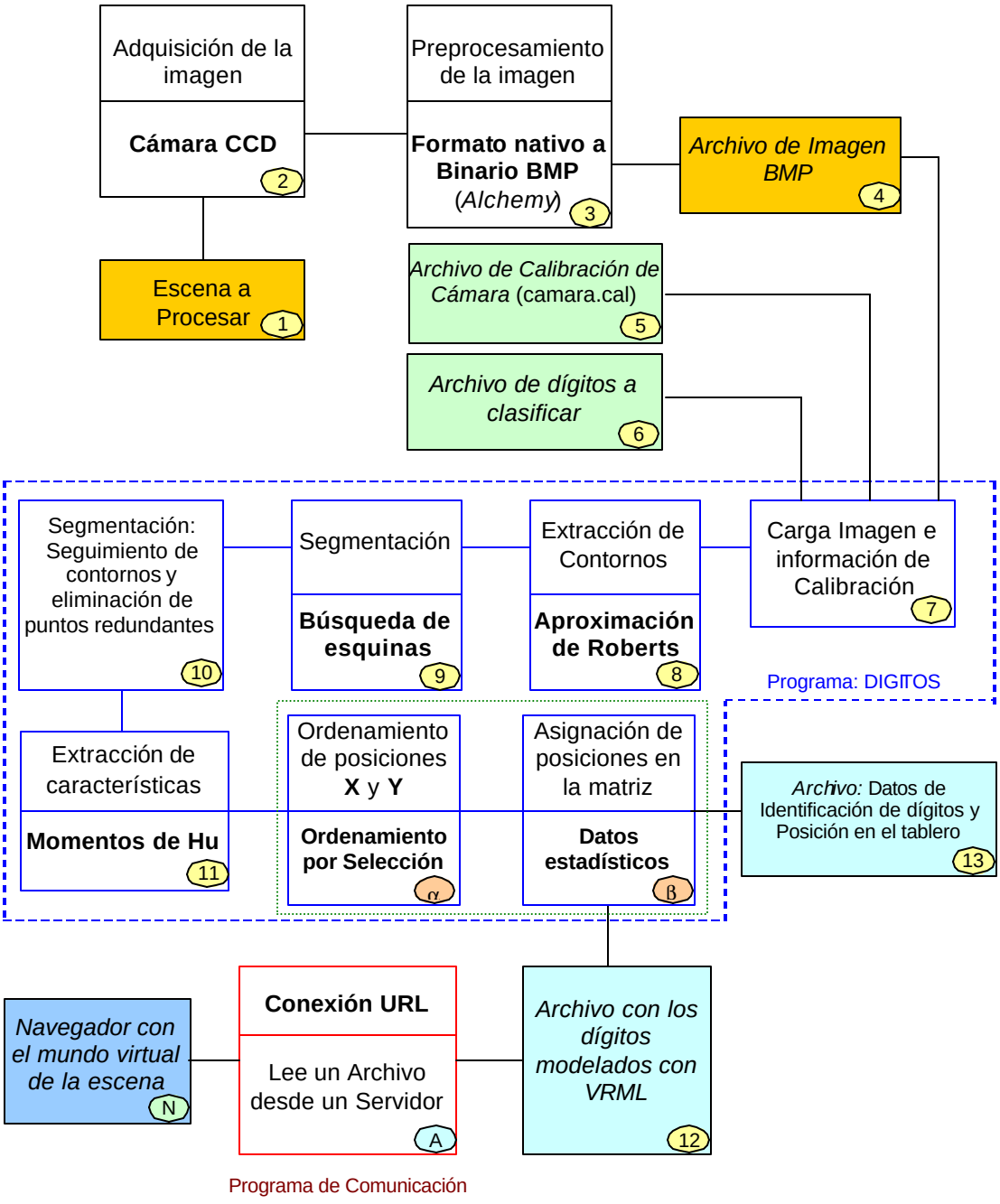


Figura 6.15: Sistema para reconocimiento de dígitos.

El reconocimiento de dígitos requiere de una mayor cantidad de *momentos de Hu* por que la cantidad de objetos es mayor que para el caso de figuras geométricas, además de otros módulos para encontrar las coordenadas de las celdas de la matriz y de la celda vacía. Sin embargo el tiempo empleado para realizar el reconocimiento de dígitos es lo suficientemente pequeño, lo que permite realizar el reconocimiento de dígitos varias veces por segundo.

6.3 Análisis de movimiento

Hasta aquí se ha descrito el reconocimiento de objetos fijos en la escena. Sin embargo, algunas aplicaciones requieren el seguimiento de objetos en movimiento. Por ejemplo en aplicaciones militares se emplea para detección y seguimiento de objetivos; en control de tráfico, para el control de tráfico vario; en seguridad, en la vigilancia de edificios, etc. Por lo que describimos el análisis de movimiento a continuación.

El análisis por *momentos de Hu* puede ser también empleado para el seguimiento de objetos que se mueven en la escena a baja velocidad ó a una velocidad lo suficientemente pequeña como para que el sistema de visión pueda detectarlo. Mediante está técnica es posible guiar a un robot para poder tomar el objeto en una posición especifica o incluso para mover el sistema de cámara y mantener al objeto lo más cercano al centro del cuerpo en la escena. Las imágenes empleadas pueden provenir de una película, en algún formato (`*.mpeg, *.jpeg, *.mov, *.vif, *.avi, *.gif, *.dat,` etc.) o a través de tomar muchas fotografías a intervalos de tiempo determinados. Empecemos examinando qué es el movimiento.

El fenómeno más obvio y fundamental que observamos a nuestro alrededor es el *movimiento*. Prácticamente todos los procesos imaginables pueden describirse como el movimiento de ciertos objetos. Decimos que un objeto está en movimiento relativo respecto a otro cuando su posición, medida con relación al segundo cuerpo, está cambiando con el tiempo. Por otra parte si esta posición relativa no cambia con el tiempo, el objeto se encuentra en reposo relativo. Tanto el reposo como el movimiento son conceptos relativos; es decir, dependen de la condición del objeto respecto a un cuerpo que sirve de referencia.

Para analizar el movimiento de un objeto, es necesario conocer algunas definiciones. La *cinemática*, estudia el movimiento de los cuerpos en función del tiempo. La *velocidad* de un objeto es la primera derivada de la distancia respecto al tiempo (**Ecs. 6.2** y **6.3**).

$$v = ds / dt \qquad \text{...(Ec. 6.2)}$$

$$v = s / t \quad \text{para } v = \text{constante} \qquad \text{...(Ec. 6.3)}$$

Donde:
v = Velocidad en m/s
s = Distancia en m
t = Tiempo en s

La *aceleración* es la primera derivada de la velocidad en relación al tiempo (**Ec. 6.4**).

$$a = dv / dt = d^2s / dt^2 \qquad \text{...(Ec. 6.4)}$$

Donde:
a = Aceleración en m/s^2
v = Velocidad en m/s
s = Distancia en m
t = Tiempo en s

La *velocidad angular* o *frecuencia angular* es la primera derivada respecto al tiempo del ángulo de giro o rotación (**Ecs. 6.5**, **6.6** y **6.7**).

$$w = df / dt \qquad \text{...(Ec. 6.5)}$$

$$w = f / t \quad \text{para } w = \text{constante} \qquad \text{...(Ec. 6.6)}$$

$$w = 2 p f \qquad \text{...(Ec. 6.7)}$$

Donde:
w= Velocidad angular en rad/s
f = Ángulo en rad
f = Frecuencia en $1/s$
t = Tiempo en s

La *aceleración angular* es la primera derivada de la *velocidad angular* respecto al tiempo (**Ec. 6.8**).

$$a = dw / dt = d^2 f / dt^2 \quad \text{...(Ec. 6.8)}$$

Donde:
a = Aceleración angular en rad/s^2
w= Velocidad angular en rad/s
f = Ángulo en rad
f = Frecuencia en $1/s$
t = Tiempo en s

Con la *gráfica de la forma del movimiento,* en función del tiempo es posible obtener la velocidad en un instante determinado a través del calculo de la *primera derivada de la función* (**Ec. 6.9**) (**Figura 6.16a**).

$$v » Ds / Dt \quad \text{...(Ec. 6.9)}$$

Si se deriva la función de la forma del movimiento en relación al tiempo se obtiene una gráfica de la aceleración en un punto determinado (**Ec. 6.10**) (**Figura 6.16b**).

$$a » Dv / Dt \quad \text{...(Ec. 6.10)}$$

En la **Figura 6.16c** muestra la variación de la *aceleración*. Si $a>0$ la aceleración corresponde a un *aumento en la velocidad*, si $a<0$ la aceleración corresponde a una *disminución en la velocidad*.

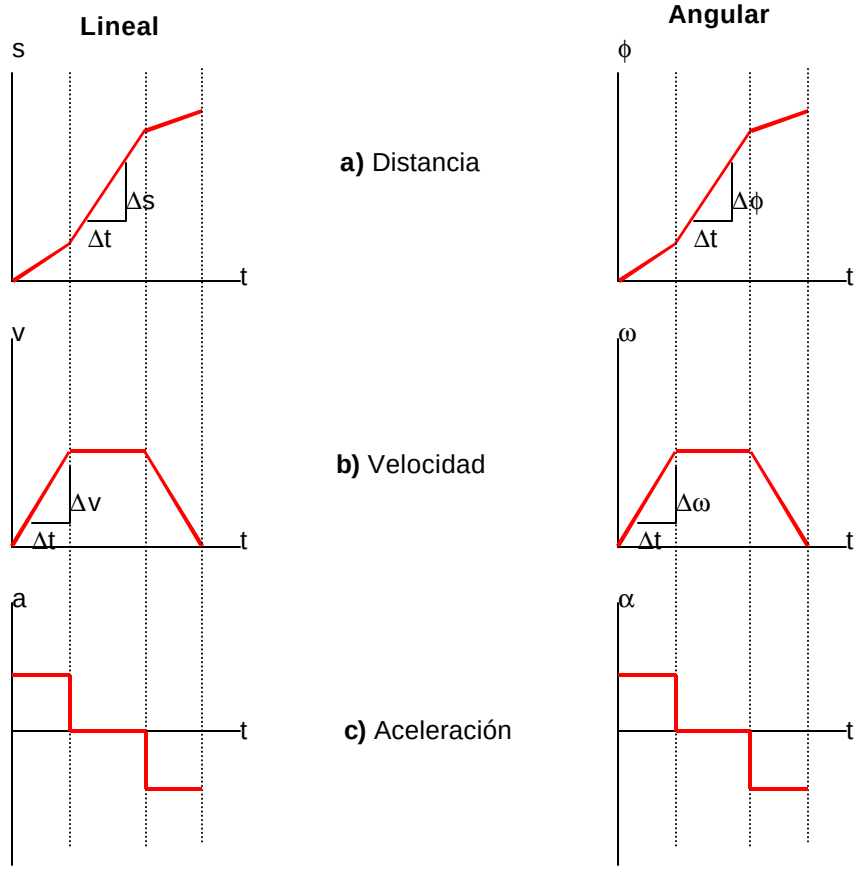


Figura 6.16: Distancia recorrida, velocidad y aceleración de un punto en movimiento.

A diferencia de las aplicaciones anteriores, el análisis del movimiento de un objeto sobre una escena requiere de un proceso de reconocimiento un poco más simple que el empleado en las aplicaciones anteriores. En la **Figura 6.17** se muestra nuestro sistema para llevar a cabo el análisis de movimiento de un objeto sobre la mesa de trabajo del robot UNIMATE S-103, siendo posible su utilización en otras aplicaciones sin necesidad de modificaciones mayores. Los **módulos 1** al **12** de la **Figura 6.17** fueron descritos en el **Capítulo 5: Diseño e Implementación de un Sistema de Visión**, y solo cambia el formato de archivos de salida (**Figura 6.17 –módulo W**), el cuál contiene una serie de números que representan el movimiento del *centro de masa* calculado mediante *momentos de Hu* de un objeto moviéndose en la escena.

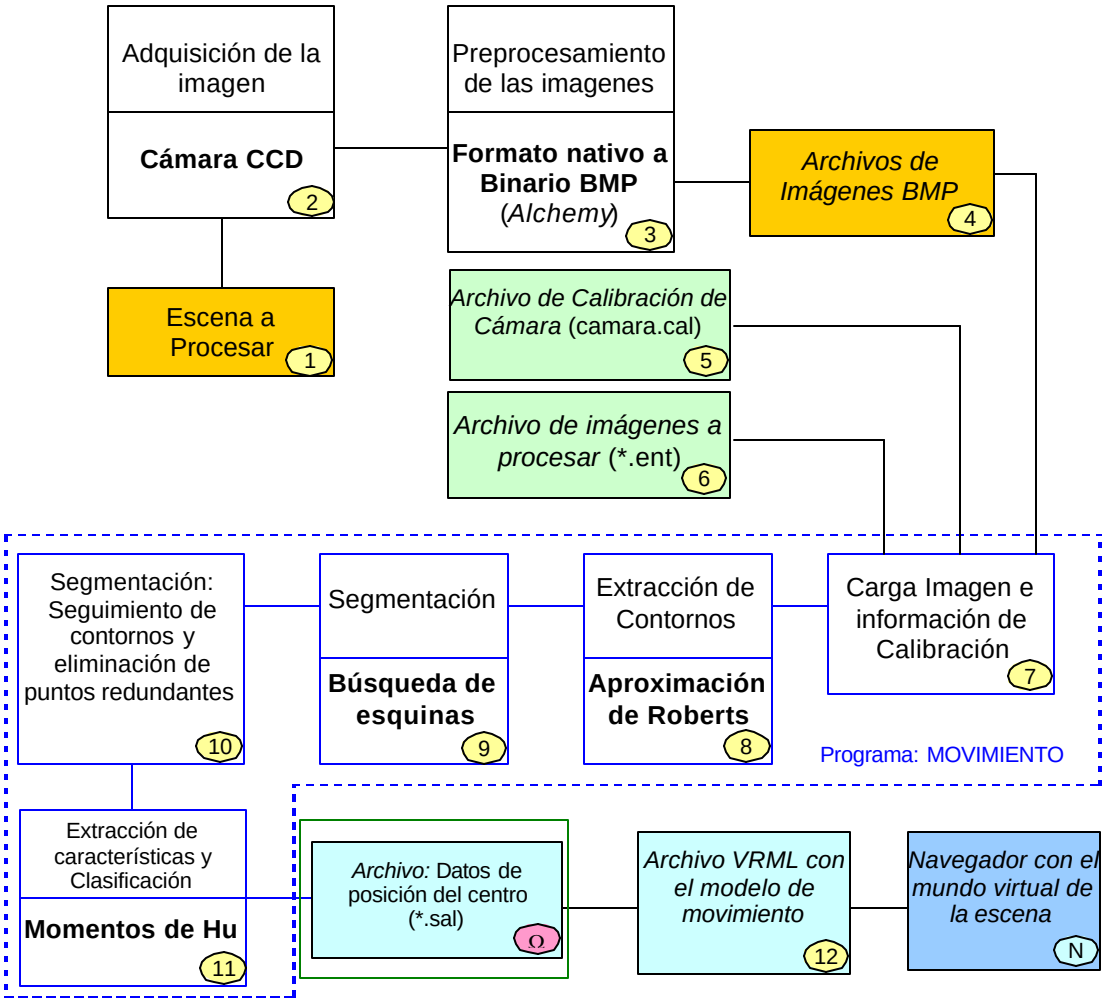


Figura 6.17: Programa para análisis de movimiento.

Para el análisis de movimiento, es necesario conocer las imágenes tomadas en un intervalo de tiempo determinado. Por lo que en un archivo se encuentran los nombres de las imágenes y el intervalo entre ellas, este intervalo se emplea para determinar la velocidad y aceleración del objeto, una vez que su posición es calculada. Por ejemplo para la serie de imágenes de la **Figura 6.18** correspondientes a una pelota moviéndose en un plano inclinado, tenemos el archivo de entrada de la **Figura 6.19** con los parámetros de las imágenes.

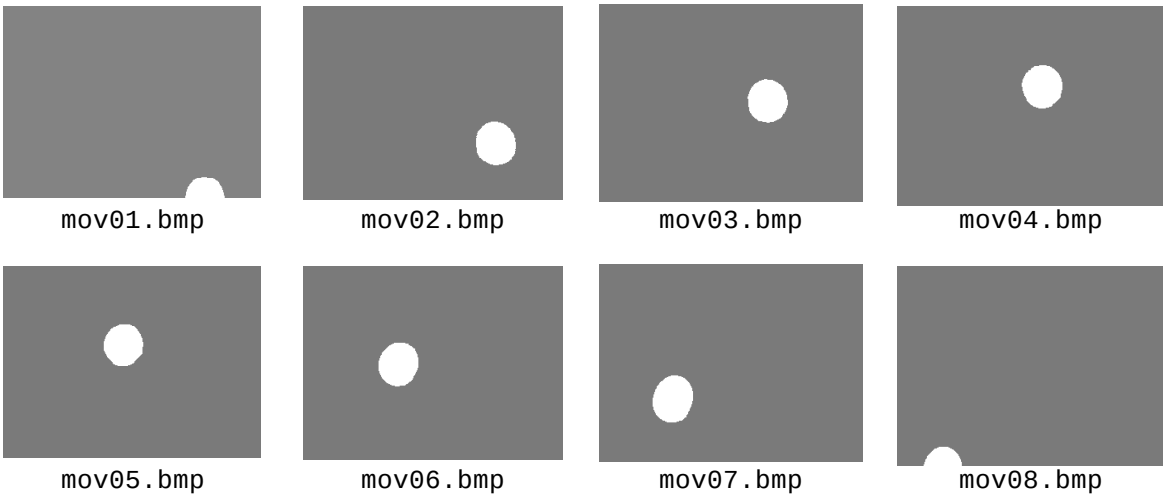


Figura 6.18: Imágenes de entrada para el análisis de movimiento.

Archivo `mov.ent`:	**Comentarios:**
`8`	Número de imágenes (n).
`0.1875005`	Tiempo entre imágenes en segundos.
`0`	Tipo de imagen, identificación de objeto.
	Lista de imágenes:
`mov01.bmp`	Imagen 1.
`mov02.bmp`	Imagen 2.
`mov03.bmp`	.
`mov04.bmp`	.
`mov05.bmp`	.
`mov06.bmp`	.
`mov07.bmp`	.
`mov08.bmp`	Imagen n.

Figura 6.19: Archivo de entrada `mov.ent` para el análisis de movimiento.

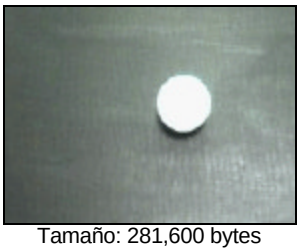
Con el archivo de la **Figura 6.19** y las imágenes de entrada de la **Figura 6.18** obtuvimos el *archivo de salida* de la **Figura 6.20** con las posiciones del centro de masa de cada una de las fotografías. Este archivo de posiciones es empleado por el programa `movimiento` (en *lenguaje C*) para reconstruir el movimiento en un mundo virtual en tres dimensiones (en lenguaje *VRML*).

Archivo `mov.ent`:			**Comentarios:**
`8`			Número de imágenes (n).
`0.187501`			Tiempo entre imágenes en segundos.
`0`			Tipo de imagen, identificación de objeto.
			Lista de coordenadas del centro de masa y tamaño del objeto en cada imagen
`265.7`	`222.8`	`44`	Posición 1: x_{c1} y_{c1} Área 1
`236.1`	`170.3`	`209`	Posición 2: x_{c2} y_{c2} Área 2
`203.4`	`118.0`	`201`	.
`173.6`	`97.2`	`199`	.
`148.4`	`99.3`	`199`	.
`116.5`	`122.2`	`203`	.
`88.6`	`164.1`	`209`	.
`70.5`	`225.5`	`35`	Posición n: x_{cn} y_{cn} Área n

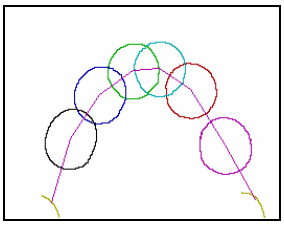
Figura 6.20: Archivo de salida `mov.sal` para el análisis de movimiento.

Un archivo de posiciones es relativamente pequeño comparado con el conjunto de imágenes de entrada (en mapa de bits ó BMP) ó con la película misma. Por lo que su envío a través de Internet u otro medio es más eficiente. Por otro lado, la representación del entorno a

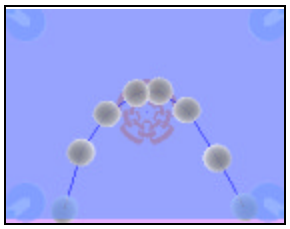
través de *VRML* permite la visualización del movimiento del objeto en la escena desde cualquier ángulo, distancia, perspectiva, etc. por lo que se tiene una mayor flexibilidad respecto a las películas tradicionales (**Figura 6.21**).



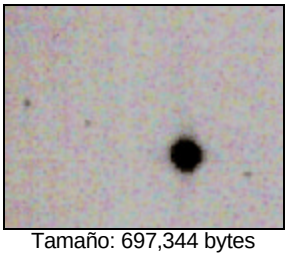
Tamaño: 281,600 bytes
a) Entrada: Película `mov.avi`



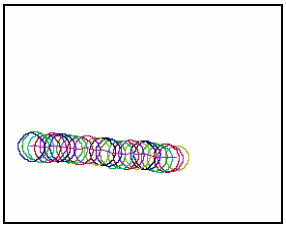
Tamaño: 178 bytes
a) Análisis del movimiento



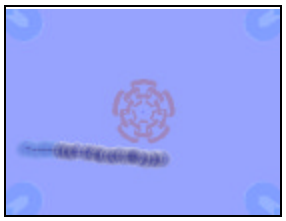
Tamaño: 10,252 bytes
a) Reconstrucción 3D



Tamaño: 697,344 bytes
b) Entrada: `Jupiter.avi`



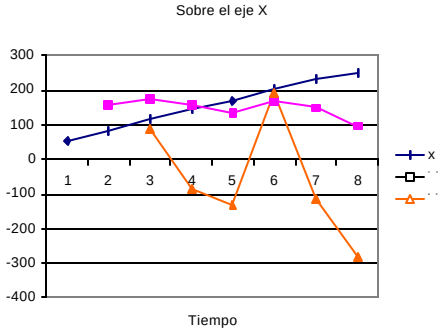
Tamaño: 570 bytes
b)) Análisis del movimiento



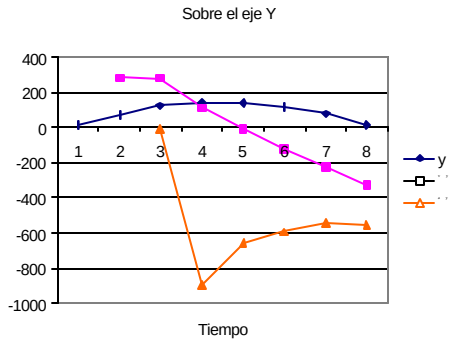
Tamaño: 18,766 bytes
b) Reconstrucción 3D

Figura 6.21: Resultados para el programa de análisis de movimiento.

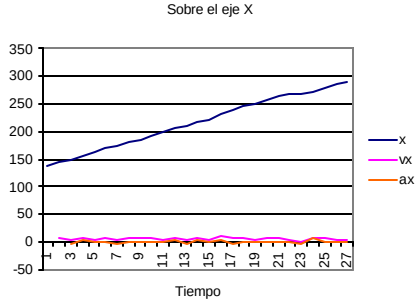
Con relación al movimiento del objeto, nuestro programa también proporciona gráficas de *velocidad contra tiempo* y aceleración contra tiempo, que muestran como varían ambas variables con relación al movimiento del objeto captado en cada uno de los ejes (**Figura 6.22**).



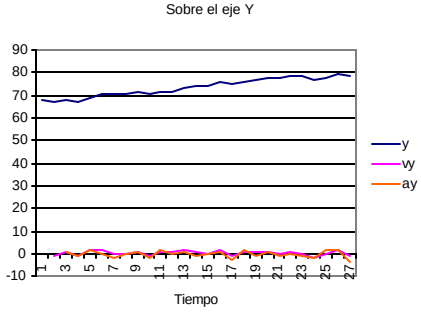
a) Entrada: Película `mov.avi`



b) Entrada: Película `mov.avi`



c) Entrada: `Jupiter.avi`



d) Entrada: `Jupiter.avi`

Figura 6.22: Gráficas de posición, velocidad y aceleración para nuestro programa de análisis de movimiento.

Estos datos pueden servir para que un robot no colisione con un objeto en movimiento, para que siga la trayectoria de un objeto ó para que mueva una cámara para seguirlo. Pero también es aplicable a otras áreas como la astronomía para el seguimiento de objetos celestes, como es posible apreciar en la **Figura 6.21**.

6.4 Estimación de Altura

Hasta este punto se ha considerado que los objetos presentan una altura adecuada para ser manipulados por el robot. Pero hace falta considerar el caso en que los objetos no presenten una altura adecuada para su manipulación (pueden ser solo figuras planas).

Para el caso del robot UNIMATE S-103, la altura de los objetos puede llegar a dañar los sistemas mecánicos del robot, pues como se recordará su órgano terminal es del tipo todo ó nada, es decir, se desplaza a toda su longitud, sin tener la posibilidad de quedar en un punto intermedio.

La altura de los objetos presentes en la mesa de trabajo de un robot industrial, es considerada como un problema resuelto en la mayoría de los textos, debido a que los robots disponen de dispositivos y/o sistemas de captación de altura adecuados. Sin embargo, en nuestro caso, la manipulación de objetos por parte del robot ha supuesto que la altura es adecuada, pero si la altura no lo es, entonces hay que proveer al robot de los mecanismos necesarios para evitar colisiones entre el objeto y el robot. Proponemos las siguientes cuatro soluciones.

1) *Estimación de altura de los objetos presentes en una imagen por el robot.-* se basa en el hecho de que la mayoría de los robots industriales presentan, además de los mecanismos necesarios para realizar sus movimientos dentro de su espacio de trabajo, sensores que les indican su posición en coordenadas articulares y canales adicionales de entrada/salida a través de los cuales pueden manejar herramientas de perirrobótica. Por medio de estos canales adicionales, es posible que un robot interactúe con el mundo exterior. Los canales pueden ser empleados para parar el cierre del órgano terminal, por ejemplo, al momento de que se detecte que el objeto a sido sujetado correctamente. En el robot UNIMATE S-103, estos canales pueden ser usados para saber si el órgano terminal se encuentra abajo o arriba y sí la mano esta cerrada o abierta, pero también si un objeto se encuentra entre los dedos del robot.

Si se dota al robot de un sensor de proximidad en su órgano terminal (no lo presenta en este momento), entonces, el robot puede detectar la altura del objeto al momento en que lo va a manipular, recordando que la posición y orientación de los objetos presentes en la mesa de trabajo del robot es determinada por el sistema de reconocimiento a partir de una fotografía del espacio de trabajo del robot, este sistema considera que todos objetos reconocidos presentan una altura fija adecuada para su manipulación.

La altura puede así ser estimada por un sensor de proximidad o presencia que el robot tiene en su órgano terminal. El robot al colocarse sobre el objeto y realizar los movimientos necesarios para tomarlo, puede conocer la altura a través de un canal, en donde se encuentra conectado el sensor de proximidad o presencia. Se pueden presentar tres casos, considerando que nuestro programa de reconocimiento considera una altura inicial para cada objeto en la escena:

a) *Objeto con la altura adecuada para manipularlo.-* en este caso el órgano terminal llegará al final de su desplazamiento y detectará el objeto en el sensor de proximidad o presencia, por lo que lo tomará normalmente al objeto, y el modelo del objeto no cambia, por lo que la altura considerada desde el inicio permanece igual (**Figura 6.23**).
b) *Objeto con una altura menor que la estimada para manipularlo.-* es este caso el órgano terminal llegó al final de su desplazamiento pero el sensor de proximidad o presencia no se activo, por lo que el objeto no puede ser manipulado, en este caso, se deberá de modificar la altura preestablecida en el modelo, y para movimientos posteriores del robot, la posición de este objeto puede ser ignorada, pues no estorba (**Figura 6.24**).

c) *Objeto con una altura mayor que la estimada para manipularlo*.- es este caso el órgano terminal no ha llegado al final de su desplazamiento, pero el sensor de proximidad o presencia se activo, por lo que en el programa del robot, deberá de estar contemplada esta posibilidad, en cuyo caso el órgano terminal (para el robot UNIMATE S-103), deberá de elevar el órgano terminal para que no se presenten daños en el objeto, ni en el robot. Se deberá de modificar la altura preestablecida en el modelo, y la posición y tamaño deberán de ser tomados en cuenta para que no se presenten colisiones al momento de mover al robot a otros puntos, pues el objeto si estorba (**Figura 6.25**).

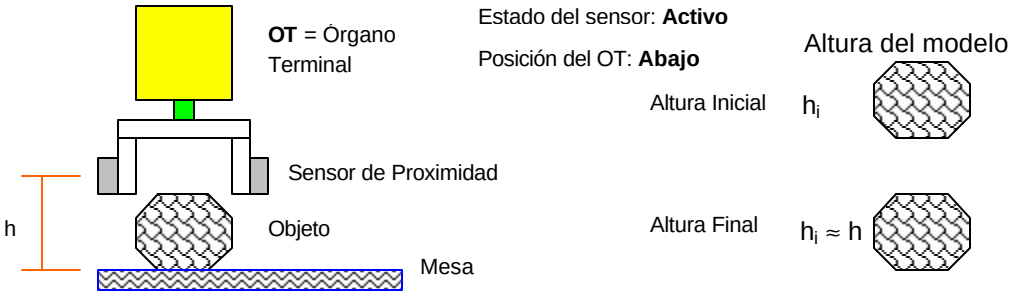


Figura 6.23: Objeto con la altura adecuada para manipularlo.

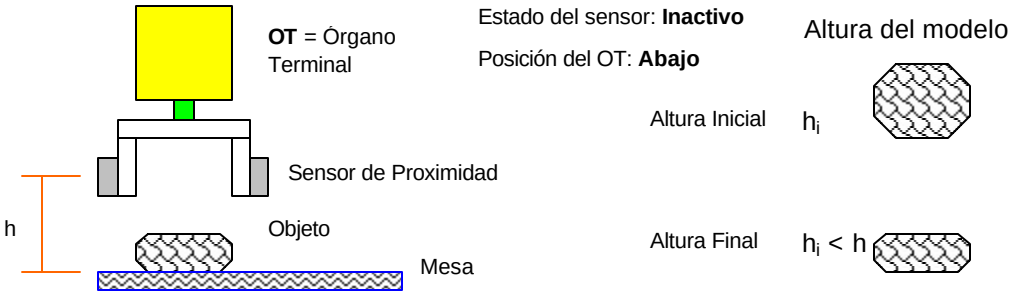


Figura 6.24: Objeto con una altura menor que la estimada para manipularlo.

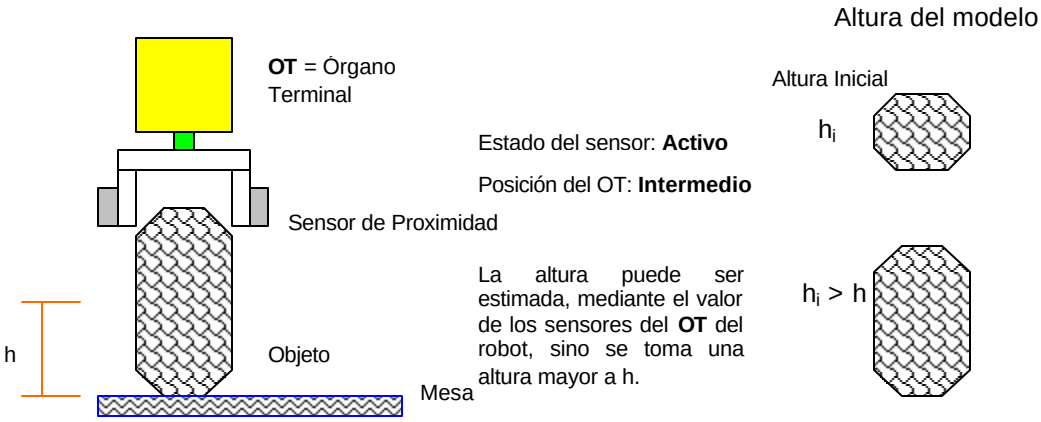


Figura 6.25: Objeto con una altura mayor que la estimada para manipularlo.

Como puede observarse, este sistema únicamente proporciona solo tres valores para la altura (adecuado, pequeño o grande), aún así es posible emplearlo si la altura no es un factor importante a considerar para la modelación de objetos.

Si la altura se requiere conocer con mayor precisión, es necesario emplear otros sensores más elaborados, como el que se presenta a continuación.

2) *Estimación de altura a través de un convertidor analógico / digital (A/D).-* la altura de los objetos puede ser medida a través de sensores que conviertan la altura a la que se encuentra el órgano terminal del robot en un número. Para este fin, implementamos un c*ircuito detector de presencia y de un circuito para medir la posición vertical del órgano terminal*, el cual consiste básicamente de un sensor de presencia y un convertidor analógico/digital. De está forma es posible que el robot conozca la presencia de un objeto próximo a su órgano terminal y la posición vertical del órgano terminal, con la cuál es posible realizar una estimación de la altura con una buena exactitud. El circuito que se diseñó para llevar a cabo este propósito se muestra en la **Figura 6.26**.

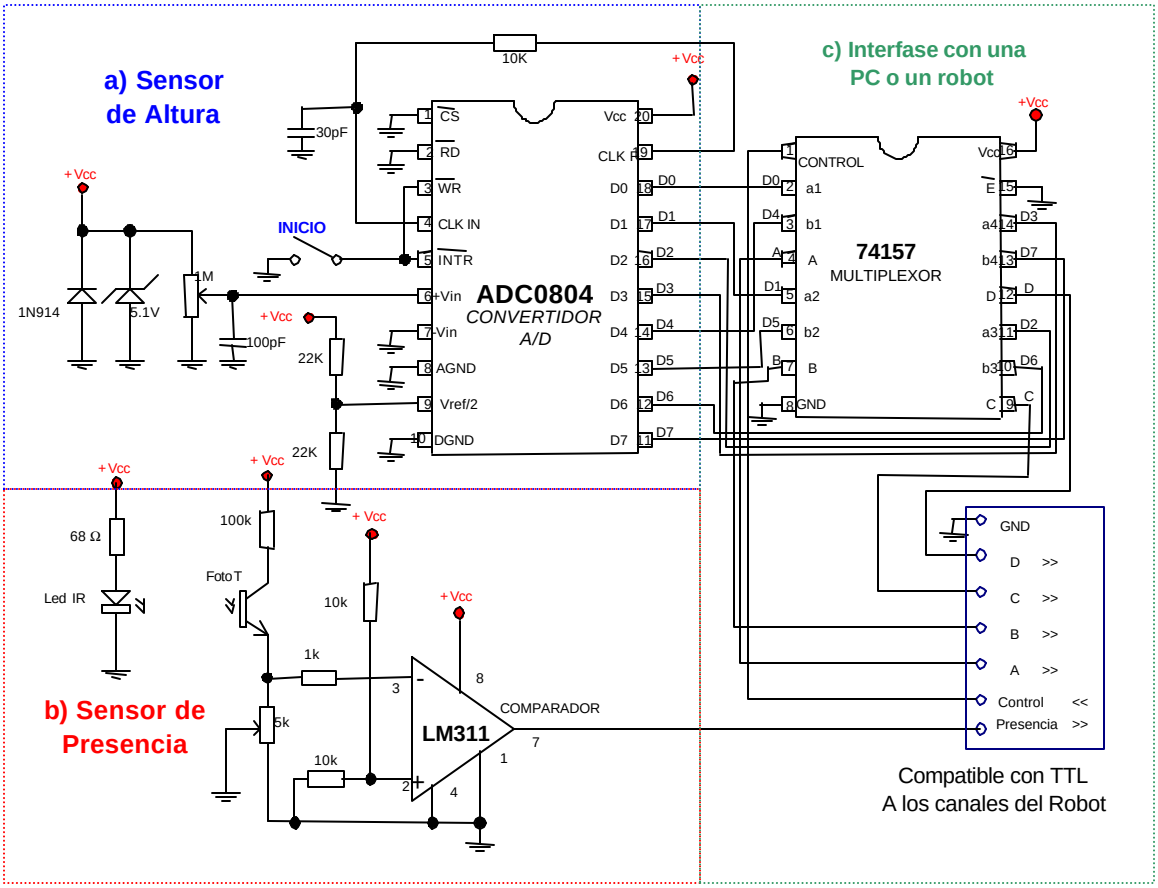


Figura 6.26: Circuito propuesto para la estimación de altura con un convertidor A/D.

El circuito de la **Figura 6.26** presenta tres circuitos: a) un *sensor de altura*, el cuál se encuentra constituido por un convertidor A/D de ocho bits (el circuito *ADC0804* proporciona hasta 255 valores digitales para una entrada analógica y es un convertidor A/D de doble rampa), y un medidor de altura simple, el cual consiste en un potenciómetro lineal que se encuentra conectado mecánicamente al órgano terminal del robot, proveyendo la distancia de desplazamiento del órgano terminal; b) un *sensor de presencia*, destinado a proporcionar al robot ó a la computadora el momento en que se debe de tomar la lectura de la altura, al estar presente el objeto en el órgano terminal del robot; y c) una *interfase* entre el circuito y una computadora o el robot, la cual consiste en un multiplexor digital para que los datos del convertidor A/D puedan ser transferidos a través de un puerto paralelo (computadora) ó a través de solo cuatro líneas.

Aunque este circuito proporciona la altura del objeto con gran exactitud, no es posible medir objetos que sean menores al desplazamiento máximo del órgano terminal del robot, de hecho ningún método basado en el órgano terminal lo podrá realizar. Para la estimación de altura para cualquier objeto es posible por medios ópticos para estimación de altura.

3) La *estimación de altura por extracción de características mediante iluminación estructurada.*- utiliza luz estructurada para encontrar todas las características de los objetos presentes en la mesa de trabajo del robot (la implementación se presenta en el **Apéndice B: Iluminación Estructurada**). Todas las características de los objetos pueden ser conocidas mediante un barrido realizado en la mesa de trabajo del robot con luz estructurada. Así se conoce la altura directamente, junto con las demás medidas de los objetos como se aprecia en la **Figura 6.27**.

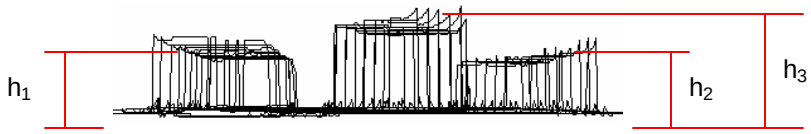


Figura 6.27: Estimación de altura usando luz estructurada.

Si el sistema de adquisición de imágenes mediante luz estructurada se encuentra calibrado, al ser utilizado en la mesa de trabajo del robot provee de la información acerca del tamaño (largo, ancho y altura), posición y orientación, de todos los objetos presentes en la mesa de trabajo del robot.

En el **Apéndice B: Iluminación Estructurada** se realiza el estudio de un sistema de iluminación estructurada, en donde se explica como realizar el cálculo de la profundidad (ó altura) de las superficies que son alcanzadas por dicha iluminación. Su empleo, puede estar limitado al tiempo que lleva realizar la cantidad de fotografías necesarias para realizar una reconstrucción del espacio de trabajo de un robot; esté tiempo puede llegar a ser superior a un segundo, aunque depende de la resolución que se requiera para la identificación de los objetos.

4) *Estimación de la altura por triangulación usando dos puntos de luz*.- el procedimiento se basa en el cálculo de la profundidad de una superficie cuando incide un par de puntos luminosos sobre ella. Nuestro procedimiento para estimar la altura consiste en lo siguiente:

1. Conocer el tamaño del punto formado por dos rayos de luz que convergen en la mesa de trabajo del robot.
2. El ángulo entre ambos rayos es conocido, o de cualquier forma es posible medirlo.
3. El sistema de visión (cámara y nuestro programa de reconocimiento), localiza la posición del centro de masa de los objetos en dos dimensiones.
4. Con la posición de los objetos el robot o un mecanismo auxiliar apuntan los rayos a cada uno de los centros de masa.
5. Una vez que los rayos apunten a los centros de masa de los objetos, se fotografían, apareciendo como dos puntos o una línea. A partir de está fotografía es posible calcular la altura de los objetos de forma aproximada mediante los cálculos que se describen a continuación (**Figura 6.28**).

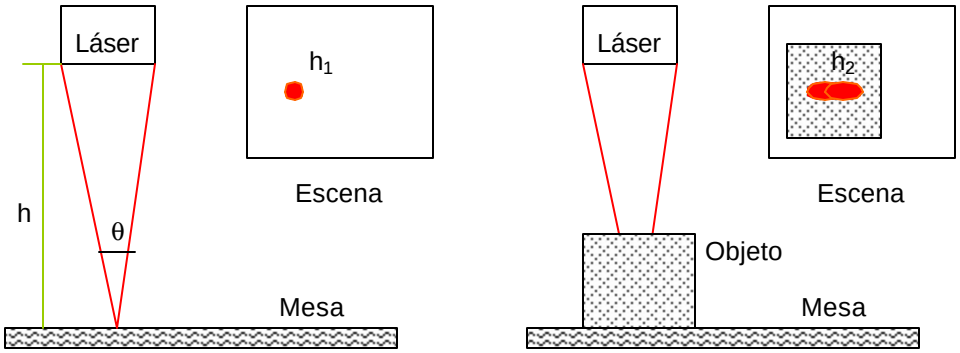


Figura 6.28: Estimación de altura por medio de rayos luminosos.

Para realizar la estimación de altura, son necesarios los parámetros siguientes:

h = Altura del dispositivo que genera los haces, es decir, la distancia entre la mesa de trabajo y el generador de luz láser (dos haces láser).

h_1 = Ancho del punto cuando los haces convergen en la mesa. Es el diámetro promedio del punto que generan los haces al incidir sobre la superficie de la mesa de trabajo.

q = Ángulo que forman ambos haces. Esté es un parámetro importante, pues entre mayor sea el ángulo los haces deben de encontrase mas separados, lo cual dificulta su posicionamiento sobre los objetos, por el contrario si se encuentran muy juntos es posible que al momento de posicionarlos sobre los objetos la diferencia entre las imágenes de los haces sobre la superficie de la mesa y sobre los objetos no sea suficiente para realizar una estimación de la altura de los mismos.

h_2 = Tamaño del punto cuando incide sobre el objeto. Este es el promedio de la separación de los haces cuando inciden sobre la superficie de los objetos.

U_1 = Umbral inferior. Es el límite inferior, por debajo del cuál el robot no puede manipular al objeto. Se encuentra determinado por el desplazamiento que tiene el órgano terminal.

U_2 = Umbral superior. Es el límite superior, a partir del cuál el robot no puede manipular al objeto. Se encuentra determinado por el desplazamiento que tiene el órgano terminal.

La *separación entre haces* (**S_x**) cuando inciden sobre la superficie de un objeto se calcula por medio de la **Ec. 6.11**.

$$S_x = h_2 - h_1 \quad \text{con } x = 1 \text{ ó } 2 \qquad \text{...(Ec. 6.11)}$$

Si se conoce una separación **S_1** es posible estimar la altura del objeto, conociendo la altura de un objeto conocido (**S_2**). A través de la relación **R** de la **Ec. 6.12** es posible determinar si el objeto se puede manipular por el robot incluso sin conocer la altura exacta del mismo.

$$R = S_2 / S_1 \qquad \text{...(Ec. 6.12)}$$

Se presentan tres casos cuando se conoce el valor de **R**:

$R < U_1$ $\Rightarrow$ El robot no puede manipular el objeto pues presenta una altura menor que la normal (altura adecuada para su manipulación).

$R > U_2$ $\Rightarrow$ El robot no puede manipular el objeto pues tiene una altura mayor que la normal (altura adecuada para su manipulación).

$U_1 < R < U_2$ $\Rightarrow$ El robot puede manipular el objeto, cuando el valor **R** se encuentre en el promedio de los valores de los umbrales **U_1** y **U_2**.

Cuando se conoce la altura promedio (**P_p**) de los objetos que si puede tomar el robot, es posible conocer la altura del objeto bajo estudio a través de la **Ec. 6.13**.

$$\text{Altura estimada} = P_p R \qquad \text{...(Ec. 6.13)}$$

Por ejemplo: para los objetos en la mesa de trabajo de un robot se tomaron los siguientes parámetros (**Figura 6.29** y **6.30a**):

h = Distancia del dispositivo que genera los haces a la mesa de trabajo = *40 cm*
h_1 = Ancho del punto cuando los haces convergen en la mesa de trabajo (**Figura 6.30b**)
= *7.702391 píxeles*
q = Ángulo que forman ambos haces = *8º*

Para dos objetos en la mesa de trabajo (**Figura 6.30a**) del robot se encontró que:

h_{21} = Tamaño del punto cuando los haces inciden sobre el *objeto 1* (**Figura 6.30c**)
= *13.152681 píxeles*

Con este dato podemos calcular **$S_1 = h_{21} - h_1 = 5.45029$**

h_{22} = Tamaño del punto cuando los haces inciden sobre el *objeto 2* (**Figura 6.30d**)
= *18.234343 píxeles*

Con **h_{22}** podemos calcular la altura por medio de las **Ecs. 6.12** a **6.13**.

$S_2 = h_{22} - h_1 = 10.531952$

$R = S_2 / S_1 = 1.93236543$

El resultado **R** implica que el *objeto 2* mide aproximadamente el doble que el *objeto 1* Ahora para conocer la altura del *objeto 2*, es necesario conocer la altura del *objeto 1* (**P**)

Altura del objeto 1 = P = 4 cm

Podemos entonces calcular la altura del *objeto 2* como:

Altura estimada para el objeto 2 = PR = 7.72946174 cm

Sin embargo, la altura real para el objeto 2 es la siguiente:

Altura real para el objeto 2 = 8 cm

Así podemos realizar una estimación del error que este método ocasiona al medir la altura de un objeto que se encuentra sobre la mesa de trabajo de un robot.

Error = ([Altura real] – [Altura medida])*100 / [altura real]

Error = (8 - 7.72946174) * 100 / 8

Error = 3.3818%

Se observa que el error es muy pequeño, por lo que con este método se alcanza una buena aproximación de la altura de los objetos.

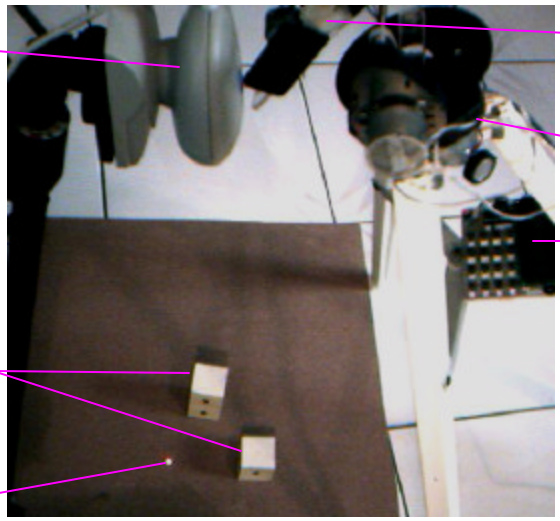
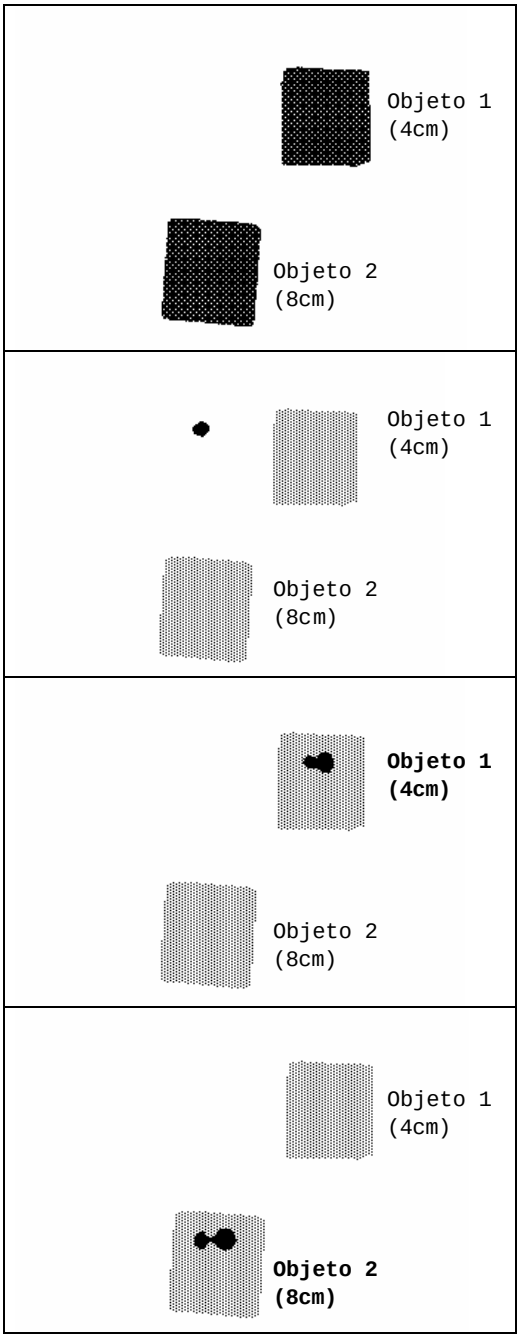


Figura 6.29: Mecanismo usado para posicionar los haces láser para la estimación de altura.



a) Objetos de diferentes alturas sobre la mesa de trabajo, hallados por nuestro sistema de reconocimiento.

b) Puntos láser sobre la mesa de trabajo. Se encuentran a un ángulo y distancia a la cual convergen en la mesa de trabajo del robot.

c) Puntos láser sobre el *objeto 1*. Un mecanismo ha movido los haces hacia el centro de masa del primer objeto.

d) Puntos láser sobre el *objeto 2*. Un mecanismo ha movido los haces hacia el centro de masa del segundo objeto.

Los resultados muestran que el *Objeto 2* es 1.93 veces más alto que el *Objeto 1*, por lo que si conocemos la altura del *Objeto 1* podemos calcular la altura del *Objeto 2*.

Figura 6.30: Estimación de altura para dos objetos sobre la mesa de trabajo.

Todas las aplicaciones anteriores consideran que el robot puede hallar y manipular al objeto sobre la mesa de trabajo. Para que el robot pueda manipular los objetos, es necesario que el sistema de reconocimiento se encuentre calibrado, es decir, que exista una relación entre los puntos de cada imagen procesada por el sistema de reconocimiento y las coordenadas que cada robot maneja. En el siguiente apartado se realiza la descripción de cómo realizar la calibración de cámara para este fin.

6.5 Calibración de cámara

Para que un robot pueda manipular los objetos que aparecen en su mesa de trabajo (como en las tareas anteriores), es necesario que sus sistema de cámara se encuentre calibrado. El proceso de *calibración de cámara* es necesario para establecer una relación entre las coordenadas de imagen con relación a las coordenadas del robot, de tal forma que cuando el sistema de visión ubique un objeto el robot lo pueda manipular. Es decir, a cada punto en la imagen le corresponde un punto en el sistema de coordenadas del robot. El proceso de calibración de cámara sirve también para hallar el tamaño en centímetros de los objetos presentes en la escena (espacio de trabajo del robot).

Nuestro programa emplea el método de calibración de cámara por *transformación homogénea*. Consiste en colocar un par ó más objetos en el espació de trabajo de un robot, por lo que se conocen sus posiciones de acuerdo a las coordenadas del robot, luego se realiza una ó más fotografías, midiendo el número de píxeles entre los centros de los objetos.

La distancia entre pares de objetos se emplea para determinar la relación entre píxeles y otras unidades (como centímetros) que el robot pueda reconocer. La distancia entre un par de puntos u objetos se puede calcular a través de la **Ec. 6.14**.

$$\textbf{Distancia entre dos puntos} = \sqrt{[(x_1 - x_2)^2 + (y_1 - y_2)^2]} \qquad \textbf{...(Ec. 6.14)}$$

Para la *calibración de la cámara* en nuestro sistema, se deben de colocar sobre la mesa de trabajo del robot y en el centro de la mesa de trabajo, una serie de patrones de calibración (**Apéndice D: Patrones de Calibración y Figuras Geométricas**). Estos *patrones de calibración* están constituidos con cuatro puntos cada uno (con un diámetro de *1 cm*), separados a *10 cm*, que en conjunto forman las esquinas de un cuadrado de *10x10cm*. Se tienen seis patrones para realizar la calibración de cámara (a 24 puntos), cada patrón difiere del anterior en que los puntos se encuentran con una rotación de 15º, es decir, se tiene un patrón para 0º, 15º, 30º, 45º, 60º y 75º. En cada fotografía de cada patrón (6 en total), se calcula: la *escala en el eje x* (escx), la *escala en el eje y* (escy), en *ángulo de rotación* respecto al eje cero del robot (ang) y las *coordenadas del centro* (xc,yc) del cuadrado formado por los cuatro puntos de cada patrón (Figura 6.30).

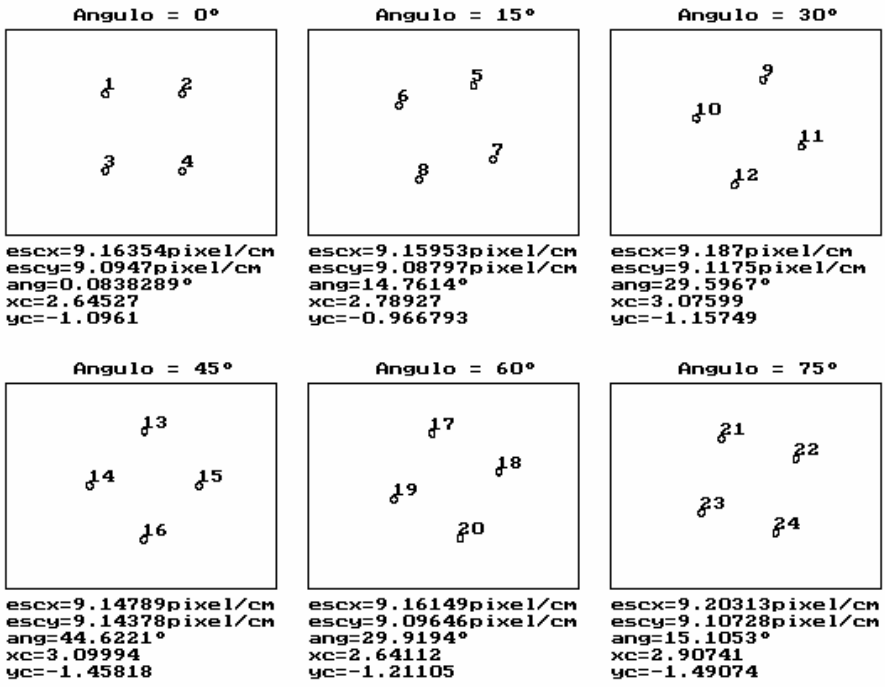


Figura 6.31: Secuencia de fotografías para calibración de cámara.

El resultado es el promedio de los valores obtenidos de cada una de las fotografías de cada patrón. Para el ejemplo mostrado de la **Figura 6.31** obtenemos los resultados mostrados en la **Figura 6.32**.

```
Escala en el eje x = 9.17 píxeles/cm
Escala en el eje y = 9.11 píxeles/cm
Ángulo = 0.18º
Desplazamiento en x = 0.31 cm
Desplazamiento en y = -0.14 cm
```

Figura 6.32: Datos obtenidos para la calibración de cámara.

En la **Figura 6.32**, la escala en el eje *x* resulta ser un poco mayor con relación a la escala en el eje *y*, debido a que la cámara posiblemente presentaba una ligera inclinación en ese eje (*x*). El ángulo resultante (*ángulo*) es debido a que el *eje horizontal* (*x*) de la cámara difiere un poco del eje horizontal de la mesa de trabajo del robot, pues no hay forma de alinear con precisión absoluta la cámara con la mesa de trabajo. Así tenemos que existe un desplazamiento entre ambos centros, y por está razón siempre será necesario realizar algún tipo de calibración de cámara para asegurar que el robot pueda manipular los objetos presentes en su espacio de trabajo.

6.6 Reconstrucción tridimensional con VRML

Para poder representar los objetos encontrados en las aplicaciones anteriores en un mundo virtual, es necesario llevar a cabo un procesamiento de los datos obtenidos de cada una de ellas. Para la reconstrucción tridimensional de los objetos bajo estudio se utilizan los parámetros que almacena el programa de reconocimiento y que para el caso de figuras geométricas se muestran en la **Figura 6.33**.

1	2	3	→ Identificación
217.580948	131.111893	177.866028	→ Centro **X** en píxeles
68.228569	121.000000	120.535889	→ Centro **Y** en píxeles
-0.569332	-0.759948	-1.515486	→ Angulo en radianes (α)
17.119450	24.677608	37.599552	→ Tamaño en píxeles (*m*)
Triángulo	**Cuadrado**	**Rectángulo**	

4	5	6	→ Identificación
158.479675	215.045120	137.826080	→ Centro **X** en píxeles
63.260162	125.473686	114.913040	→ Centro **Y** en píxeles
-1.656315	-1.687306	-2.007422	→ Angulo en radianes (α)
17.323471	17.592960	4.318154	→ Tamaño en píxeles (*m*)
Hexágono	**Círculo**	**Indicador**	

Figura 6.33: Parámetros encontrados para las figuras geométricas.

Cada uno de estos parámetros representa las características de los objetos como se muestra en la **Figura 6.34** (ver también el **Apéndice C: Transformaciones geométricas** y el **Apéndice D: Patrones de calibración y figuras geométricas**). Estos parámetros son:

a) `Identificación`.- es un número el cual representa al objeto bajo estudio y es asignado por nuestro programa de reconocimiento para las categorías de objetos dadas.

b) `Coordenadas X y Y`.- centro de masa de cada objeto localizado por el programa de reconocimiento.

c) `Ángulo` (α).- ángulo en radianes que representa la rotación del eje principal (eje del centro de masa) respecto al *eje x* horizontal.

d) `Tamaño en píxeles` (*m*).- representa el tamaño del eje principal calculado por el método de *momentos de Hu*.

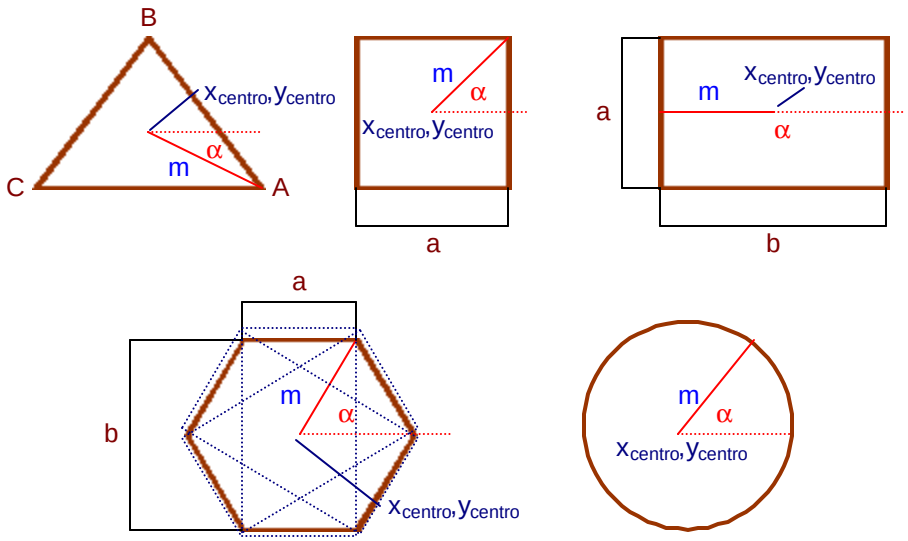


Figura 6.34: Parámetros de las figuras geométricas.

Cada uno de estos parámetros se utiliza de forma distinta dependiendo del objeto en cuestión. Describimos a continuación a manera de ejemplo el empleo de cada uno de ellos para los objetos de la **Figura 6.34**. Se emplean algunas de las ecuaciones del **Apéndice C: Transformaciones Geométricas**.

1) El *triángulo* se modela tomando como base sus vértices, por lo que hay que calcular los puntos *A, B* y *C* alrededor del centro (**Ecs. 6.15** a **6.17**). **H** es la altura del objeto estimada.

$$A(x,y) = [\ x_{centro} + m\ cos(a),\ y_{centro} + m\ sin(a)\ ] \quad \text{...(Ec. 6.15)}$$
$$B(x,y) = [\ x_{centro} + m\ cos(a+2p/3),\ y_{centro} + m\ sin(a+2p/3)\ ] \quad \text{...(Ec. 6.16)}$$
$$C(x,y) = [\ x_{centro} + m\ cos(a+4p/3),\ y_{centro} + m\ sin(a+4p/3)\ ] \quad \text{...(Ec. 6.17)}$$

La modelación con *VRML* se muestra en el **Programa 6.3**.

```
1. Transform {
2.    translation xcentro ycentro 0 # Desplaza el origen al centro del objeto
3.          children [
4.                Shape { geometry IndexedFaceSet{
5.                      coord Coordinate{
6.                            point[Ax Ay 0,  # Coordenadas del punto A cara 1
7.                                  Bx By 0,  # Coordenadas del punto B cara 1
8.                                  Cx Cy 0,  # Coordenadas del punto C cara 1
9.                                  Ax Ay H,  # Coordenadas del punto A cara 2
10.                                 Bx By H,  # Coordenadas del punto B cara 3
11.                                 Cx Cy H] # Coordenadas del punto C cara 4
12.                           }
13.                     coordIndex [2,1,0,-1,
14.                                 3,4,5,-1
15.                                 0,1,4,3,-1,
16.                                 0,3,5,2,-1,
17.                                 2,5,4,1,-1]
18.   }     }     ]     }
```

Programa 6.3: Fragmento de código *VRML* para modelar un prisma triangular.

Para el resto de los objetos de la **Figura 6.34** se desplazan las coordenadas de referencia en el espacio virtual, al centro del objeto (x_{centro}, y_{centro}), de éste modo el origen queda en el centro de cada objeto al momento de realizar la modelación.

2) Para el *cuadrado* se desplaza el origen al centro del objeto (x_{centro}, y_{centro}), se rota (el origen) y se calcula uno de los lados del cuadrado para su modelación (**Ecs. 6.18** y **6.19**). **H** es la altura del objeto estimada.

$$a = m \cos(\alpha) \quad \text{...(Ec. 6.18)}$$

$$\text{Ángulo de rotación} = \alpha + \pi/4 \quad \text{...(Ec. 6.19)}$$

La modelación con *VRML* se muestra en el **Programa 6.4**.

```
1. Transform {
2.    translation xcentro ycentro 0 # Desplaza el origen al centro del objeto
3.    rotation 0 1 0 a + p/4    # Rota al origen
4.          children [
5.                Shape {
6.                      geometry Box { size a a H } # Tamaño del cubo
7.                }
8.          ]
9.    }
```

Programa 6.4: Fragmento de código *VRML* para modelar un cubo.

3) Para el *rectángulo* se desplaza el origen al centro del objeto (x_{centro}, y_{centro}), se rota el (origen) y se calculan los lados del rectángulo para su modelación (**Ecs. 6.20** a **6.22**). **H** es la altura del objeto estimada.

$$a = m \quad \text{...(Ec. 6.20)}$$

$$b = 2\,m \quad \text{...(Ec. 6.21)}$$

$$\text{Ángulo de rotación} = \alpha \quad \text{...(Ec. 6.22)}$$

La modelación con *VRML* se muestra en el **Programa 6.5**.

```
1. Transform {
2.    translation xcentro ycentro 0 # Desplaza el origen al centro del objeto
3.    rotation 0 1 0 a          # Rota al origen
4.          children [
5.                Shape {
6.                      geometry Box { size a b H } # Tamaño del cubo
7.                }
8.          ]
9.    }
```

Programa 6.5: Fragmento de código *VRML* para modelar un prisma rectangular.

4) El *hexágono* se modela sobreponiendo tres rectángulos, por lo que para cada rectángulo hay que desplazar el origen al centro del objeto (x_{centro}, y_{centro}), calcular los lados de cada rectángulo y su ángulo de rotación (**Ecs. 6.23** a **6.27**). **H** es la altura del objeto estimada.

$$a = m \quad \text{...(Ec. 6.23)}$$

$$b = \tilde{O}3\,m \quad \text{...(Ec. 6.24)}$$

$$\text{Ángulo de rotación 1} = \alpha + \pi/6 \quad \text{...(Ec. 6.25)}$$

$$\text{Ángulo de rotación 2} = \alpha + \pi/2 \quad \text{...(Ec. 6.26)}$$

$$\text{Ángulo de rotación 3} = \alpha + 5\pi/6 \quad \text{...(Ec. 6.27)}$$

La modelación con *VRML* se muestra en el **Programa 6.6**.

```
1. Transform {
2.    translation xcentro ycentro 0 # Desplaza el origen al centro del objeto
3.    rotation 0 1 0 a + p/6    # Rota al origen
4.          children [
5.                Shape {
6.                      geometry Box { size a b H } # Tamaño del cubo
7.                }
8.          ]
9.    }
10. Transform {
11.   translation xcentro ycentro 0 # Desplaza el origen al centro del objeto
12.   rotation 0 1 0 a + p/2    # Rota al origen
13.         children [
14.               Shape {
15.                     geometry Box { size a b H } # Tamaño del cubo
16.               }
17.         ]
18.   }
19. Transform {
20.   translation xcentro ycentro 0 # Desplaza el origen al centro del objeto
21.   rotation 0 1 0 a + 5p/6   # Rota al origen
22.         children [
23.               Shape {
24.                     geometry Box { size a b H } # Tamaño del cubo
25.               }
26.         ]
27.   }
```

Programa 6.6: Fragmento de código *VRML* para modelar un prisma hexagonal.

5) Para el *círculo* y el *círculo indicador* se desplaza el origen al centro del objeto (x_{centro},y_{centro}) para su modelación (**Ec. 6.28**). **H** es la altura del objeto estimada.

Radio = m **...(Ec. 6.28)**

La modelación con *VRML* se muestra en el **Programa 6.7**.

```
1. Transform {
2.    translation xcentro ycentro 0 # Desplaza el origen al centro del objeto
3.          children [
4.                Shape {       # Tamaño del cilindro
5.                      geometry Cylinder { radius m height H }
6.                }
7.          ]
8.    }
```

Programa 6.7: Fragmento de código *VRML* para modelar un cilindro.

Como es posible apreciar en los programas anteriores, las figuras geométricas pueden modelarse con *VRML* a través de primitivas básicas. Para la modelación del mundo virtual, un programa en *Java* lee un archivo como el de la **Figura 6.33** y realiza los cálculos necesarios para modelar los objetos, luego son insertados en el código de *VRML* el cual contiene al robot y su medio ambiente.

En el caso de reconocimiento de dígitos, es necesario asignarle a cada dígito un cubo y a cada cubo un número. Para hacerlo en *VRML*, se emplea el fragmento de código mostrado en el **Programa 6.8**.

```
1.    Shape {
2.          geometry Text {                # Inserta Texto
3.                string [ "2" ]           # Número 2 a insertar
4.                fontStyle FontStyle {    # Define el tipo de letra
5.                      family "ALBERTUS" # Tipo de letra: ALBERTUS
6.                      style "BOLD"      # Estilo: negrita
7.                      size 1.5          # Tamaño: 1.5 (Unidades VRML)
8.                      }
9.                }
10.         }
```

Programa 6.8: Fragmento de código *VRML* para insertar números.

El código anterior se repite para cada uno de los dígitos identificados, por lo que el programa en *Java* repite el proceso una cantidad de veces igual a la cantidad de dígitos hallados.

Resumen

La implementación del sistema de visión se realizó para cuatro tareas: para reconocimiento de figuras geométricas, para reconocimiento de dígitos, para el análisis de objetos en movimiento y para la estimación de altura.

Las *figuras geométricas* son fáciles de reconocer por los métodos de extracción de contornos, búsqueda de esquinas, seguimiento de contornos y reconocimiento por cálculo de *momentos de Hu*. También son fáciles de modelar pues solo requieren de las primitivas básicas de *VRML*. El objetivo es el de localizar cada uno de los objetos que se encuentren presentes en el espacio de trabajo del robot, por lo que el sistema de visión deberá de proveer al robot, de la información sobre el tipo de *objeto, localización, orientación* y *tamaño*, para que el robot pueda determinar como y si es posible manipularlo. La distribución de los *momentos de Hu* (tres para está aplicación), permite ver que algunos límites se diferencian perfectamente de las demás, como son *el triángulo, el rectángulo, el hexágono* y los *círculos indicadores*, mientras que el *círculo y el cuadrado* casi se juntan en sus límites superior e inferior. Muchas de las figuras pueden ser reconocidas calculando solo el primer *momento de Hu*.

Para el *reconocimiento de dígitos*, existen sistemas de reconocimiento de caracteres cuyo proceso se asemeja al reconocimiento por momentos pues consisten en una segmentación de línea, segmentación de palabras y caracteres, reconocimiento de caracteres y producción de un archivo de salida de texto. En los sistemas *OCR*, se deben de tener tantos códigos para reconocimiento como tipos de caracteres se deseen reconocer, sin embargo permiten realizar el reconocimiento de un gran número de símbolos. El reconocimiento de dígitos por momentos consiste en la caracterización de los parámetros de cada uno de los dígitos, como si estos fueran un objeto. Se encontró que es necesario calcular una mayor cantidad de momentos (cuatro en este caso), debido principalmente a que la cantidad de objetos se incrementó. También fue necesario ampliar el programa para poder ordenar cada digito en una celda de una matriz de 3x3 y encontrar las coordenadas del lugar vacío.

Para el *análisis de movimiento*, realiza una modelación virtual que simula de la forma más fiel posible el movimiento de los objetos proveniente de una película, una serie de fotografías, etc. En este caso el programa de reconocimiento simplemente calcula el centro de masa del objeto que se mueve, trazando su trayectoria en el mundo virtual y calculando su velocidad y aceleración instantánea.

Cuando se requiere conocer la *altura de los objetos*, el sistema de visión también puede determinar la altura de los mismos mediante triangulación, al hacer incidir un par de rayos de luz sobre un objeto, conociendo el ángulo entre los haces de luz y la altura de un objeto conocido, es posible calcular la altura de los demás con un mínimo de error.

El sistema de calibración de cámara puede ser empleado para hallar el tamaño en centímetros de los objetos presentes en la escena (espacio de trabajo del robot), se empleó el *Método de Transformación Homogénea*, el cual consiste en colocar un par de objetos alrededor del centro del espació de trabajo del robot, por lo que se conocen sus posiciones de acuerdo a las coordenadas del robot, luego se realiza una fotografía y se mide el número de píxeles entre los centros de ambos objetos, conociendo la distancia en pixeles entre ambos objetos.

Capítulo 7: Conclusiones y Perspectivas

7.1 Conclusiones

Esta tesis ha presentado el diseño e implementación de un sistema de visión artificial para el *Laboratorio de Robótica Virtual* del *Departamento de Control Automático* del *Centro de Investigación y de Estudios Avanzados* del *Instituto Politécnico Nacional*. Nuestro sistema de reconocimiento fue implementado para el robot UNIMATE S-103, en donde fue posible reconocer figuras geométricas, cubos con números, objetos en movimiento y estimar la altura de objetos varios, partiendo de una imagen discreta captada por medio de una cámara web convencional.

Los objetos reconocidos por nuestro sistema de visión son representados mediante su posición, orientación y tamaño, parámetros a partir de los cuales se puede realizar la reconstrucción de dichos objetos en un espacio tridimensional, en el lugar mismo del reconocimiento o a través de Internet por usuarios remotos. El empleo de modelos virtuales facilita su transferencia a través de Internet pues, como se demostró la transmisión de las imágenes es más lenta que el envió de modelos, y también se logra una presentación más clara, aún para los usuarios locales. La manipulación del mundo virtual permite ver al robot, al medio ambiente y a los objetos que manipula, desde cualquier ángulo, perspectiva o escala, abriendo grandes posibilidades, no solo en la investigación (por ejemplo para planeación de movimientos) sino también en el área docente, al permitir a los estudiantes acceder de forma remota al robot e incluso les permite trabajar con el robot virtual con los objetos necesarios para realizar prácticas de robótica, para posteriormente aplicarlas al robot real. Cabe hacer mención que el comportamiento, forma y funcionamiento del robot virtual son idénticos al robot real, gracias al empleo del lenguaje de modelación virtual *VRML* (*Virtual Reality Modeling Language*).

La reconstrucción de los objetos presentes en el espacio de trabajo de un robot se llevó a cabo mediante el uso del lenguaje de modelación virtual *VRML*, el cual resultó ser un lenguaje de programación sencillo de usar y que produjo códigos bastante reducidos, por lo que es ideal para usarse en sistemas en donde se requiera representar ambientes reales en la computadora.

Nuestro sistema puede ser utilizado en la industria bajo dos puntos de vista: para la capacitación de personal y en labores industriales.

a) Para la capacitación de personal.- permite la capacitación del personal al realizar las tareas necesarias en los diferentes procesos industriales sin necesidad de usar el equipo disponible, pues estos procesos pueden ser trasladados a mundos virtuales para el entrenamiento del personal. Así tenemos por un lado, que no se emplean los recursos de la empresa para el entrenamiento del personal (solo algunas computadoras), y por otro, permite predecir fallas, errores de planeación, accidentes, etc.; al mismo tiempo que el personal se capacita.

b) En labores industriales.- encuentra aplicación en dos grandes áreas: inspección e identificación de objetos. Para el caso de la inspección es posible su aplicación inmediata en labores tan simples como detectar la presencia de un objeto ó para hallar algunos defectos, hasta actividades complejas como la medición y modelación de objetos industriales. Sin embargo, en la identificación de objetos, es necesario dotar a nuestro sistema de la información relacionada con cada uno de los objetos que sea necesario identificar, para cubrir necesidades tales como el reconocimiento de objetos para su ordenamiento (por ejemplo para paquetería), ensamble de mecanismos (por ejemplo en la industria automotriz y electrónica), en el seguimiento de formas (por ejemplo en pintura y soldadura), etc.

Nuestro sistema de visión artificial está constituido por sistemas de adquisición, procesamiento, segmentación y modelado de imágenes, que permiten la localización de los objetos presentes en la imagen correspondiente a la escena del espacio de trabajo del robot y la reconstrucción tridimensional de los objetos presentes en una imagen, usando modelos en lenguaje *VRML* para la actualización del *espacio virtual* del *Laboratorio de Robótica Virtual*.

Para la localización de los objetos se calcularon las coordenadas del centro de gravedad de cada uno de los objetos presentes en la escena (puesto de trabajo del robot), su orientación y tamaño por cálculo de momentos invariables de *Hu*. El reconocimiento consistió en la creación de un cierto espacio de clasificación acorde con el modelo, implementando esquemas de comparación vectorial de los objetos modelados.

La implementación de un sistema de calibración geométrica permitió asegurar la repetitividad y efectividad de las medidas realizadas por nuestro sistema de reconocimiento en el puesto de trabajo real. Se realizó a través del método de transformación homogénea.

Mediante el análisis realizado al programa de reconocimiento, se estableció que conforme el número de objetos en la escena se incrementa, el tiempo empleado para la ejecución lo hace en un tiempo proporcional. Un tiempo de ejecución reducido como el que presenta este algoritmo permite realizar varios reconocimientos por segundo, aún si el número de objetos es moderadamente grande. Por otro lado, el tamaño de imagen si influye notablemente en el desempeño del algoritmo, es este caso la complejidad obtenida de forma empírica es de $O(p^2)$, con lo cual al incrementar el tamaño de la imagen el tiempo crece de forma drástica. Sin embargo para un tamaño adecuado para el reconocimiento (*320*240 píxeles*) el tiempo de ejecución obtenido es lo suficientemente bajo, como para realizar varias operaciones de reconocimiento en un segundo. También se encontró que el número de objetos que es conveniente enviar como archivo es de aproximadamente unos 350, después de dicha cantidad es conveniente enviar la imagen de entrada.

Se establece que existe la posibilidad de emplear información visual para que un robot pueda manipular objetos, con lo que sé amplia el campo de interacción del robot que emplee este tipo de sistemas. Los resultados indican que los sistemas de reconocimiento por medios digitales pueden reconocer objetos de forma rápida, lo que permite asegurar que el robot pueda tener información sobre la localización de un objeto, debido a que la inercia de los sistemas mecánicos del robot impide que este ejecute tareas a velocidades comparables a las operaciones del sistema de visión.

Se obtuvieron buenos resultados para las aplicaciones propuestas de reconocimientos de objetos geométricos, reconocimiento de dígitos, análisis de objetos en movimiento y estimación de la altura de los objetos. Esto se debió a que como la cantidad de objetos es pequeña, permitió establecer límites lo suficientemente separados como para no tener ambigüedad en el reconocimiento.

En resumen, tenemos las siguientes conclusiones:

- Se realizó un estudio de las áreas de interés para cubrir los objetivos de esta tesis, tal como robótica, reconstrucción tridimensional y visión artificial. En robótica, se destacaron conceptos tales como robots, robots manipuladores y robots de servicio, además de la clasificación de este tipo de autómatas, se explicó la forma en que los sensores permiten que los robots interactúen de mejor manera con su ambiente. Para la reconstrucción tridimensional se realizó una pequeña descripción del lenguaje para modelación virtual *VRML*, indicando como es posible combinarlo con otros lenguajes de uso general como *Java* para incrementar su capacidad y versatilidad de cálculo. En visión artificial se describieron los procesos para realizar el reconocimiento, así como los algoritmos necesarios para cada y uno de los procesos involucrados.

- En esté trabajo también se describe el objetivo y los recursos disponibles en el *Laboratorio de Robótica Virtual*, y como nuestro sistema de visión permitió realizar un mejor empleo del mismo en las áreas docente y de investigación.

- Nuestro trabajo presentó los aspectos necesarios para el diseño e implementación de un sistema de visión general, el cual puede ser adaptado a diversas aplicaciones. Se describieron de forma detallada los algoritmos para llevar a cabo el reconocimiento de objetos.

- Se demostró cómo es posible emplear nuestro sistema de visión de forma inmediata en aplicaciones como el reconocimiento de figuras geométricas. Además se mostró como es posible emplearlo junto con otros sistemas para medir la altura de los objetos.

- Se desarrollaron algoritmos para la asignación de los dígitos en cada una de las celdas de una matriz lógica de 3x3 y se programaron los algoritmos necesarios para realizar el reconocimiento de estos objetos.

- Se implementaron los programas necesarios para llevar a cabo la representación del robot y su medio ambiente mediante *VRML*, así como la actualización con los objetos hallados por nuestro sistema de reconocimiento.

- La velocidad de los algoritmos resultó ser buena, al permitir analizar las imágenes varias veces por segundo.

- Se encontró que para una imagen de tamaño constante el tiempo necesario para el reconocimiento varía de forma directamente proporcional al número de objetos a reconocer, por lo que si tomamos en cuenta que el robot UNIMATE S-103 solo puede almacenar hasta 94 puntos (los cuales pueden ser solo objetos) el tiempo necesario para llevar a cabo el reconocimiento es de tan solo 50 milésimas de segundo (Pentium III @ 450Mhz).

- Para un número de objetos constante, pero con un tamaño de imagen variable, los resultados fueron alentadores, a pesar de que la complejidad es cuadrática. Así para 20 objetos en la escena obtuvimos un tiempo de 38 milésimas de segundo (Pentium III @ 450Mhz) para una imagen de 76,800 píxeles.

- También se demostró que para imágenes que contengan un número de objetos mayor a 350, el archivo generado es mayor que el archivo de la imagen original. Sin embargo para 94 objetos el tamaño es menor 2.5Kbytes, lo que resulta ser casi la cuarta parte del tamaño del archivo de la imagen original.

7.2 Perspectivas

Los sistemas de visión serán cada vez más empleados para muchos tipos de aplicaciones en el futuro. Los robots serán los primeros beneficiados del uso de sistemas de visión, pero lo están empezando a ser todos los sistemas autónomos.

Además de los sistemas industriales es posible aplicar los sistemas de visión en otras áreas como la astronomía, medicina, milicia, análisis de imágenes, agricultura, seguridad, redes de comunicación, control de calidad, modelación de ambientes virtuales, etc.

Pero también es posible implementar los algoritmos en circuitos especializados de visión. Así sería posible tener equipo electrónico capaz de seguir, encontrar, reconocer, etc., objetos específicos con un bajo costo (mucho menor al de una computadora).

1) Este sistema puede ser empleado con éxito en área como la astronomía, para localización y seguimiento de objetos celestes, en el futuro existirán observatorios astronómicos automáticos. Una sugerencia para adaptar nuestro sistema de visión a un telescopio astronómico, es la que se muestra en la **Figura 7.1**.

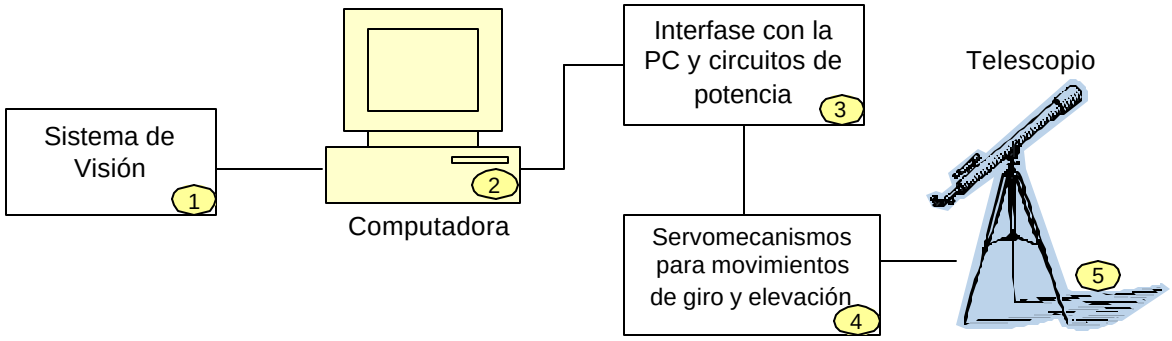


Figura 7.1: Sistema de visión para seguimiento de objetos celestes.

Nuestro sistema de visión (**Figura 7.1 –módulo 1**) se encargará de proporcionar las coordenadas del objeto celeste captado por el telescopio, para que un computadora (**Figura 7.1 –módulo 2**) ajuste los parámetros de posición del telescopio que permitan centrar al objeto ó seguirlo e incluso calcular su trayectoria, con el eje principal del telescopio (**Figura 7.1 –módulo 5**), a través de una interfase que contendrá los dispositivos de potencia necesarios para accionar los servomecanismos de movimiento del telescopio (**Figura 7.1 –módulos 3** y **4**).

2) En la medicina, se puede emplear para modelar el espacio de trabajo de la endoscopia, con lo cual el médico podrá observar de forma precisa los diferentes órganos, y en un futuro podrá ser reemplazado por robot para realizar tareas en las cuales el paciente no corra riesgo (**Figura 7.2**).

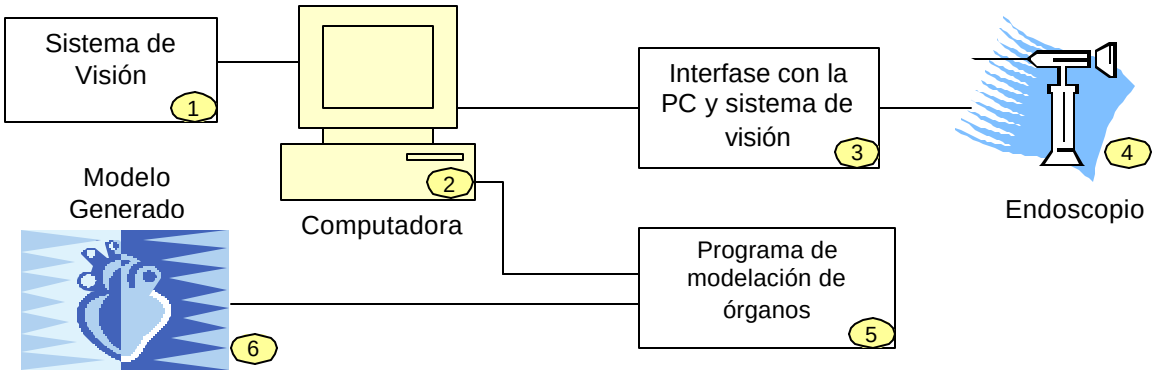


Figura 7.2: Sistema de visión en la medicina.

Nuestro sistema de visión proporciona (**Figura 7.2 –módulo 1**) en este caso los parámetros para modelos reconocidos. Con imágenes del interior del cuerpo humano provenientes de un endoscopio (**Figura 7.2 –módulo 4**). Producirá modelos tridimensionales en *VRML* de los órganos internos del cuerpo humano y del endoscopio (**Figura 7.2 –módulos 5** y **6**).

3) Para la implementación en un circuito del algoritmo de localización de objetos, proponemos el circuito de la **Figura 7.3**, el cuál se encuentra dividido en tres etapas, convertidor analógico/digital (*A/D*) para la señal compuesta de video (*NTSC-M* norma *RS-170A*), un microcontrolador encargado de hallar el centro de un objeto en la imagen y un dispositivo para

comunicación a través del puerto serie *RS-232C*, que permite comunicar los resultados a otros dispositivos similares, computadoras, etc.

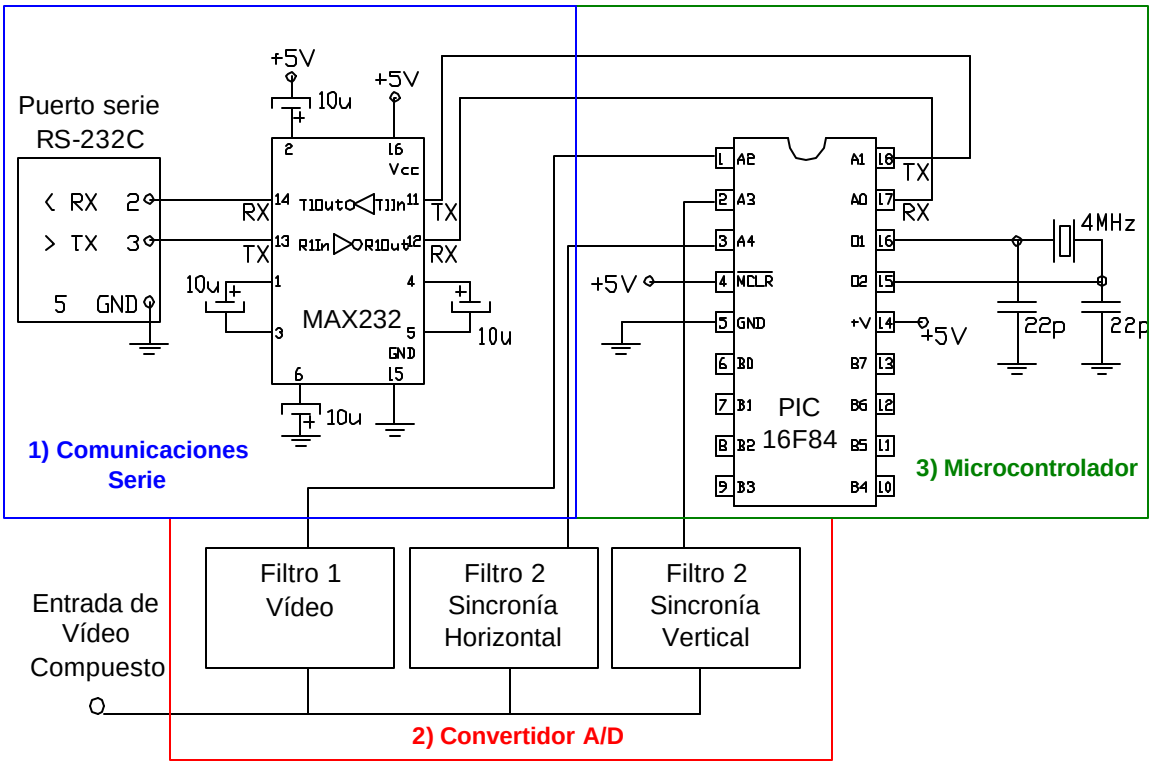


Figura 7.3: Circuito para implementar el algoritmo de localización del centro de los objetos.

El circuito encargado de las *comunicaciones* (**Figura 7.3 –módulo 1**), está constituido por un circuito *adaptador de línea* (*MAX232*). Los programas para enviar datos del microcontrolador al mundo exterior se encuentran programados en la memoria del microcontrolador (**Programa 7.1**).

Una señal compuesta de vídeo, está constituida por la señal de vídeo, y las señales de sincronización de la señal, por lo que para realizar una *conversión A/D* simple (**Figura 7.3 –módulo 2**), se requiere de al menos tres filtros. Uno destinado a convertir la señal de vídeo en una señal binaria, a través de un comparador de ventana; no interesa que la señal sea de color, un amplificador operacional de bajo costo no podrá responder a la frecuencia de la subportadora de color. Los otros dos filtros se encargarán de proporcionar las señales que indican los pulsos horizontal y vertical; estos también pueden ser implementados por medio de comparadores de ventana.

En el microcontrolador (**Figura 7.3 –módulo 3**) se encuentran los programas de comunicación serie, de localización de las coordenadas de los objetos y se pueden programar señales, las cuales pueden transferirse a través de sus ocho líneas digitales del `puerto B`. Se trata de un circuito microcontrolador *PIC16F84* de 8 bits comercializado en México por *Microchip* [Microchip 98]. Entre sus características podemos enumerar las siguientes:

- Una memoria de programa de 1Kx14 bits de tipo Flash.
- Memoria de datos EEPROM de 64 bytes.
- Memoria RAM de propósito general de 68 bytes.
- 13 líneas de entrada/salida con control individual de dirección.

- Frecuencia de funcionamiento máxima de 10Mhz.
- 35 instrucciones.
- Arquitectura *Harvard* (buses independientes para la memoria de instrucciones y datos).
- Arquitectura *RISC* (*Reduce Instruction Set Computer*).
- Soporta segmentación (*Pipe-Line* ó segmentación del procesador)

```
1. ;-------------------------------------------------------------------------
2. ; SUBRUTINA: InitSerialPort
3. ; Inicia el puerto serie
4. ;-------------------------------------------------------------------------
5. InitSerialPort
6. BSF    Serial_TX                ; Salida ALTA por omisión
7. MOVLW  10                       ; Retardo para envió
8. MOVWF  Serial_LoopCount
9. InitSerialDelay
10.MOVLW  Serial_BitDelay
11.CALL   DelayWByFour
12.DECFSZ Serial_LoopCount, f
13.BTFSS  STATUS, Z                ; Si es cero termina
14.GOTO   InitSerialDelay
15.RETURN
16.;-------------------------------------------------------------------------
17.; SUBRUTINA: TransmitDataByte
18.; Transmite el byte del registro W a la salida serie
19.;-------------------------------------------------------------------------
20.TransmitDataByte
21.MOVWF  Serial_CurrentByte       ; Almacena el byte a enviar
22.BCF    Serial_TX                ; Envía el bit de inicio
23.MOVLW  Serial_BitDelay          ; Retardo de 1 bit
24.CALL   DelayWByFour
25.MOVLW  8                        ; Envía el Byte bit a bit
26.MOVWF  Serial_LoopCount
27.TxDataByteLoop
28.BSF    STATUS, C
29.RRF    Serial_CurrentByte, f    ; Lee la bandera de acarreo C
30.BTFSS  STATUS, C
31.BCF    Serial_TX                ; Si el bit = 0 entonces TX = 0
32.BTFSC  STATUS, C
33.BSF    Serial_TX                ; Si el bit = 1 entonces TX = 1
34.MOVLW  Serial_BitDelay          ; Retardo de 1 bit
35.CALL   DelayWByFour
36.DECFSZ Serial_LoopCount, f      ; Decrementa el contador
37.GOTO   TxDataByteLoop
38.BSF    Serial_TX                ; Envía el bit de PARO
39.MOVLW  Serial_BitDelay          ; Retardo de 1 bit
40.CALL   DelayWByFour
41.RETURN                          ; Regresa
```

Programa 7.1: Código ensamblador para comunicación serie.

El algoritmo empleado para encontrar de forma aproximada la localización de un objeto en la señal de vídeo compuesta, se muestra en el **Programa 7.2**. Debido a que este tipo de microcontroladores no manejan instrucciones complejas, las operaciones del programa son muy simples, pero permiten localizar objetos con una buena precisión.

1. En microcontrolador toma muestras a intervalos regulares de tiempo, empezando inmediatamente después del pulso de sincronía vertical y el primer pulso horizontal.
2. El microcontrolador deberá llevar dos cuentas, una para X y otra para Y de la forma:
   a. Cálculo del centro en X
      i. Empezar una cuenta desde el inicio e incrementarla una vez por cada muestra (x_1)
      ii. Empezar otra cuenta mientras aparezca la imagen (x_2)
      iii. Hacer la operación de cálculo del centro en X del objeto:

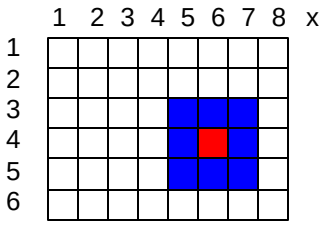
$$x_c = x_1 + x_2 / 2 + \text{acarreo de la división}$$

   b. Cálculo del centro en Y
      i. Empezar una cuenta desde el inicio e incrementarla una vez por cada pulso horizontal (y_1)
      ii. Empezar otra cuenta mientras aparezca la imagen (y_2)
      iii. Hacer la operación de calculo del centro en Y del objeto:

$$y_c = y_1 + y_2 / 2 + \text{acarreo de la división}$$

Programa 7.2: Algoritmo para calcular el centro de un objeto.

La división como tal no existe en el microcontrolador, sin embargo es posible realizar corrimientos lógicos. Como la división es entre un número sobre dos, entonces es necesario realizar solo un corrimiento para obtener dicha división. En la **Figura 7.4**, podemos ver como opera el **Programa 7.2** sobre una imagen simple.



a) Imagen de vídeo:
8 muestras
6 líneas horizontales

b) Cálculo del centro:

Datos para **X**: $x_1 = 4$ $x_2 = 3$
$\mathbf{x_c} = 4 + (0011 \rightarrow 1) + 0 = 4 + 0001 + 1 = \mathbf{6}$

Datos para **Y**: $y_1 = 2$ $y_2 = 3$
$\mathbf{y_c} = 2 + (0011 \rightarrow 1) + 0 = 2 + 0001 + 1 = \mathbf{4}$

Resultado: **Centro del objeto (6,4)**

Figura 7.4: Cálculo del centro de un objeto por medios electrónicos.

Apéndice A
Sistemas de Adquisición de Imágenes

Conocer los diferentes tipos de sistemas de adquisición de imágenes, es interesante, debido a las diferentes tecnologías empleadas a lo largo del tiempo. Cada sistema de adquisición de imágenes presenta ventajas y desventajas, por lo que conocerlas es importante en al momento de implementar un sistema de visión, debido principalmente a los costos que involucra dicho sistema. En el **Capítulo 3: Visión Artificial** definimos al proceso de adquisición de imágenes y en el **Capítulo 5: Diseño e Implementación de un Sistema de Visión** presentamos la forma en que se emplea un sistema de este tipo dentro del sistema de visión. En esté apéndice describimos en detalle las características de los sistemas de adquisición de imágenes electrónicas, así como de algunos de los sistemas de enfoque empleados desde la aparición de las primeras cámaras de video, hasta las cámaras de estado sólido.

Los sistemas de adquisición de imágenes son el primer paso del proceso que realiza un *sistema de visión*, de ahí la importancia de estudiar y conocer los diferentes sistemas que han surgido a lo largo del tiempo.

Las cámaras o sensores de imágenes captan la información luminosa de una escena, para posteriormente transmitirla a una computadora como una señal analógica ó digital. Los elementos capaces de traducir la información luminosa en una señal eléctrica son conocidos como transductores (aunque esté concepto se refiere a cualquier conversión de una forma de energía a otra).

a.1 Captores de imagen según el tipo de transductor fotoeléctrico

Un transductor fotoeléctrico, se encarga de transformar la energía luminosa (fotones) en energía eléctrica (electrones). De acuerdo al transductor empleado se tienen básicamente tres tipos de transductores fotoeléctricos, de acuerdo a la variable física que varia al incidir la luz.

1) Los *transductores fotovoltaicos* se basan en la variación de voltaje que ocasiona la incidencia luminosa en algunos materiales (como el silicio amorfo). Su respuesta es lenta, pero su eficiencia es alta. No son los más empleados por que su respuesta es lenta.

2) Los *transductores fotoconductivos* se basan en el cambio de conductividad y tienen una buena respuesta a la excitación lumínica. Algunos de los dispositivos que se basan en esta propiedad son: el vidicon, el plúmbicon (leadicon) y los dispositivos *CCD* (*Charge Coupled Device*). Tienen un tiempo de vida útil alta. Los *CCD* son dispositivos de silicio, son los más empleados actualmente por su reducido tamaño y bajo consumo de potencia, debido a lo que permite reducir enormemente el tamaño de las cámaras.

3) Los *transductores fotoemisivos* se basan en la emisión de electrones y tienen una excelente velocidad de respuesta. Algunos dispositivos que emplean la emisión de electrones son: el disector de imagen, el iconoscopio y el orticon. Presentan una vida útil baja.

Cada uno de estos transductores ha sido empleado en algunos dispositivos desde el Siglo XIX. A continuación presentaremos el funcionamiento de algunos de los dispositivos de adquisición de imágenes que fueron empleados según su orden de aparición histórico, presentando su funcionamiento y sus características mas relevantes.

a.1.1 Disco de Nipkow

El *disco de Nipkow* fue realizado en 1884 por *Paul Nipkow* (**Figura a.1**) [Teuchert 46]. Se compone de un disco que lleva en su superficie una serie de agujeros rectangulares dispuestos en espiral, de forma que, al girar, el borde inferior de uno corresponde al borde superior del siguiente. La separación entre dos agujeros es igual al ancho de la imagen a transmitir.

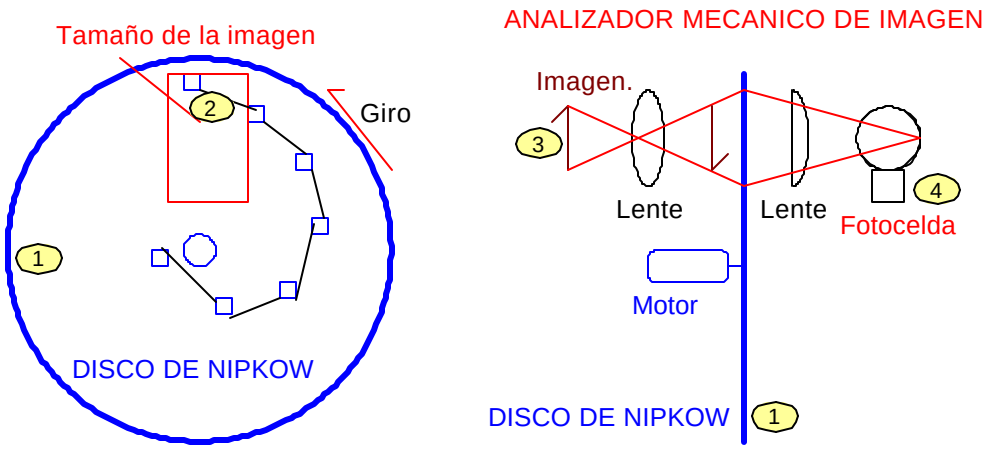


Figura a.1: Disco de Nipkow.

El sistema se conoce como *analizador mecánico de imagen*. Al girar el disco **(1),** la luz solo pasa por un agujero **(2)** a la vez, por lo que permite explorar una imagen completa **(3)** y transmitirla eléctricamente punto por punto. La imagen explorada debe de estar lo suficientemente iluminada para que la fotocelda **(4)** funcione. El *disco de Nipkow* permitió las primeras transmisiones de televisión y de transmisión en color, realizadas por la *BBC* de Londres a principios del siglo XX.

a.1.2 Disector de imagen o tubo de Fransworth

El *disector de imagen o tubo de Fransworth* fue el primer tubo de vacío dispuesto para captar imágenes [Ramirez 70]. Se compone de las partes que se muestran en la **Figura a.2**.

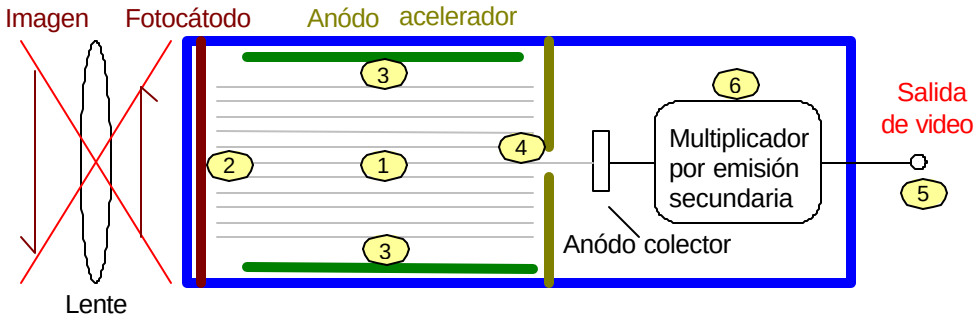


Figura a.2: Disector de imagen o tubo de Fransworth.

La imagen se explora moviendo la emisión de electrones **(1)** por medio de *bobinas de deflexión* que se encuentran afuera del tubo. El *fotocátodo* **(2)** está constituido por *cesio* evaporado, con lo cual se tienen alrededor de 40,000 elementos de imagen. Cada elemento se comporta como una *fotocelda emisiva*, es decir, emite electrones al incidir la luz sobre ella, estos electrones son acelerados por un *ánodo acelerador* **(3)** y movidos por un campo magnético, pasando por un agujero de *0.3mm* de diámetro **(4).** La variación en los campos magnéticos permite explorar un elemento cada vez, formándose al final una señal eléctrica de salida **(5)** que corresponde a las intensidades luminosas de cada parte de la imagen.

El *fotocátodo* **(2)** produce electrones proporcionalmente a la intensidad de la luz que incide sobre cada uno de sus elementos. La *sección multiplicadora* **(6)** (multiplicador por emisión secundaria), se coloco para elevar el nivel de la señal de salida, pues la resistencia del fotocátodo produce ruido. La sección multiplicadora está formada por *dynodos*, que al recibir haz electrónico, por emisión secundaria multiplican el número de electrones (**Figura a.3**). Los electrones en la sección multiplicadora viajan en espiral, debido a que el campo eléctrico es perpendicular a la dirección del electrón.

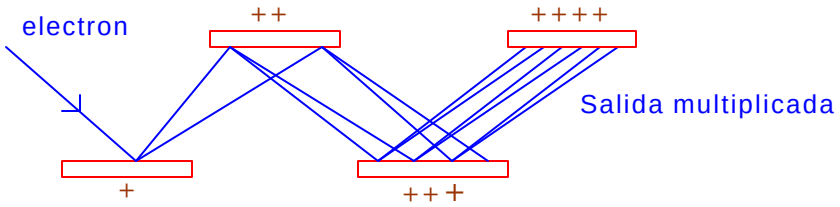


Figura a.3: Funcionamiento de los dynodos multiplicadores.

a.1.3 Iconoscopio, tubo de condensación ó de acumulación

El *Iconoscopio, tubo de condensación ó de acumulación*, fue ideado por V. K. Zworykin. Se compone de los elementos mostrados en la **Figura a.4** [Ramirez 70].

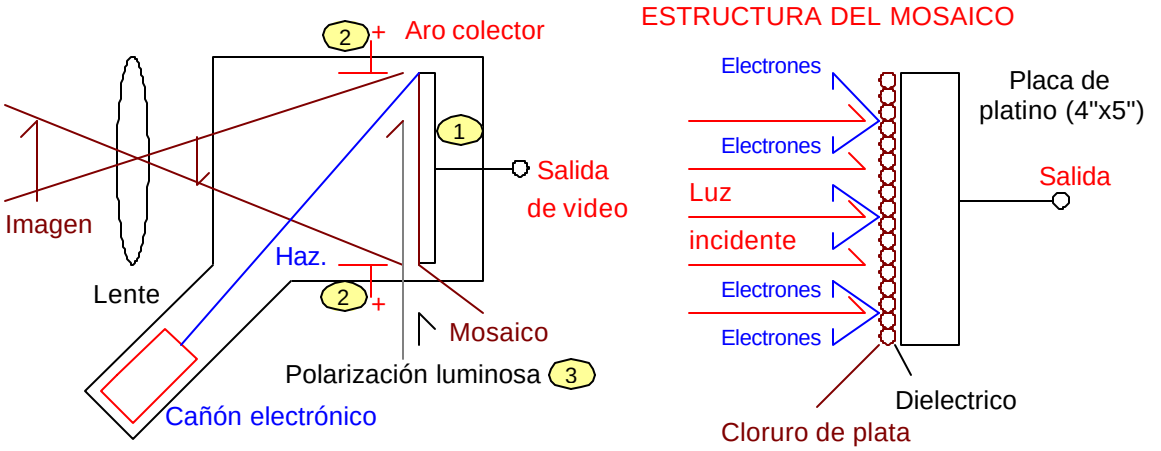


Figura a.4: Iconoscopio, tubo de condensación ó de acumulación.

Un rayo de luz incidente arranca electrones de la capa de *cloruro de plata* del *mosaico* **(1)** (la estructura del *mosaico* permite almacenar cargas eléctricas por medio de pequeñas capacidades), dejando cargas negativas alrededor del punto de incidencia. Estas cargas son absorbidas por el *aro colector* **(2).** Al incidir el haz electrónico, estas capacidades se descargan produciendo una corriente proporcional a la luz incidente. Es posible que la emisión secundaria causada por el haz electrónico, llegue a ser más grande que la que produce la luz de la escena. Para evitar esto se requiere entonces que la cantidad de luz en la escena sea elevada, bajando la eficiencia del tubo. Para contrarrestar este efecto provocado por le emisión secundaria se le da al tubo una *polarización luminosa* externa **(3),** la cual llega paralela al mosaico. Con esto se cancela la emisión secundaria. En esté tubo se tiene el problema de *microfonismo* por su gran tamaño, es decir, cuando una señal eléctrica provoca la vibración de los dispositivos eléctricos y electrónicos provocados por el tamaño del dispositivo, entre más grande sea el dispositivo aumenta el *microfonismo*.

a.1.4 Orticon de imagen

El *orticon de imagen* se encuentra constituido por los componentes que se ilustran en la **Figura a.5** [Ramirez 70].

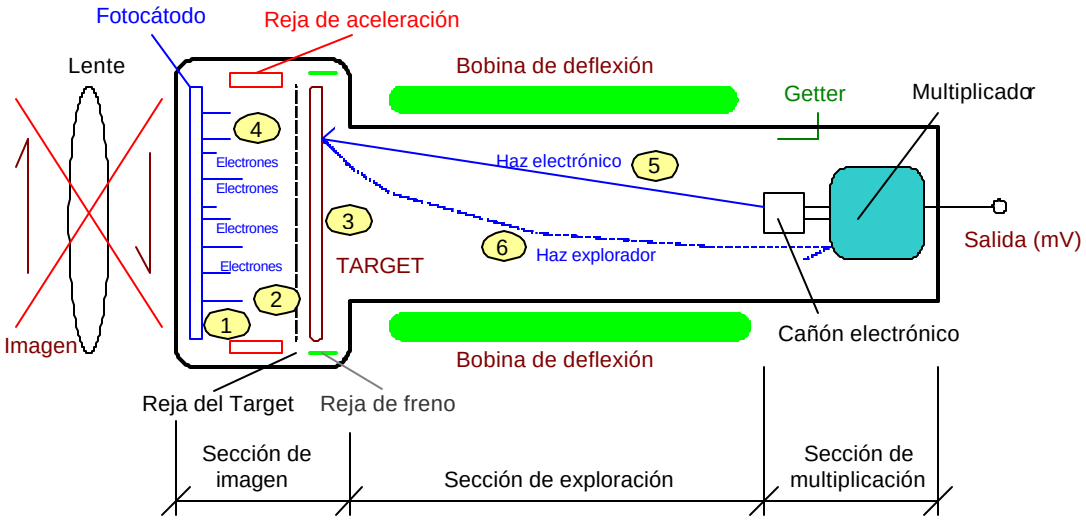


Figura a.5: Orticon de imagen.

Los electrones desprendidos del *fotocatódo* **(1)** por efecto de la iluminación de la escena son acelerados y pasan de la *reja del target* **(2)** al *target* **(3).** Al llegar los electrones al *target* **(3)** producen una emisión secundaria **(4)** que es absorbida por la *reja del target* **(2),** mientras por el lado contrario el haz electrónico **(5)** es acelerado y después frenado para que no se impacte en el *target* **(2);** regresa al cañón en una proporción equivalente a la luz incidente proveniente de la escena, este haz es llamado haz explorador **(6).** El *target* es el el equivalente al *mosaico* del *orticon*. Para trabajar esté dispositivo necesita una temperatura de 30°C ± 2.5°C; pero también cuando se toma un punto brillante, su negativo se observa un tiempo después.

a.1.5 Vidicon

El *vidicon* es el primer tubo que emplea transductores fotoresistivos, es decir, que su resistencia varia de acuerdo a la cantidad de luz que recibe (**Figura a.6**) [Ramirez 70].

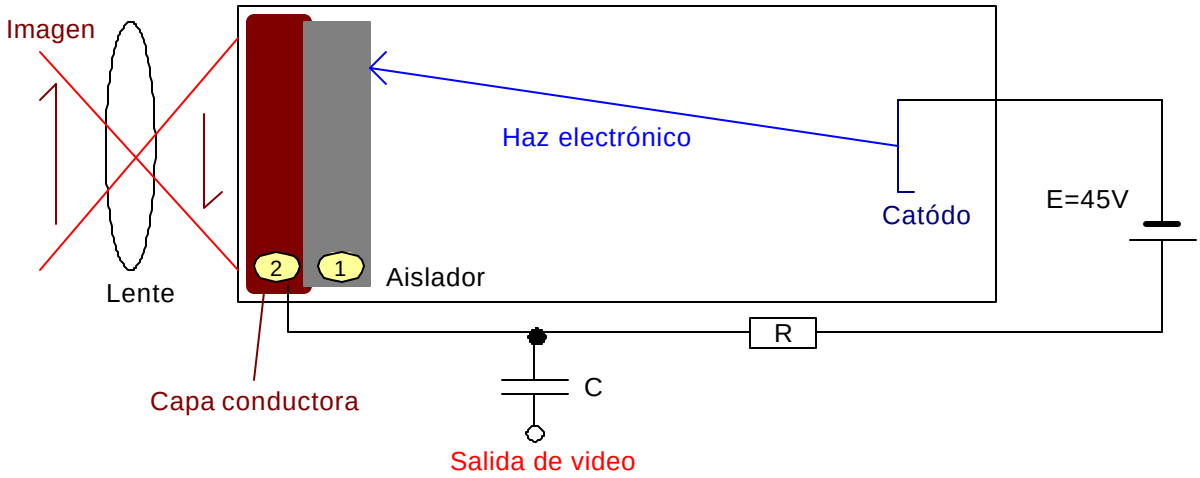


Figura a.6: Vidicon.

La capa del *aislador* **(1)** varia su impedancia de acuerdo a la cantidad de luz recibida por una capa conductora **(2).** Como aislador se emplea *antimonio;* así se tiene que la variación de impedancia es por ejemplo que para un *punto negro Z=20MW* y que para un *punto blanco Z=2MW.*

Sus ventajas son: es de tamaño reducido, bajo peso, el voltaje para formar el haz es de 250V y no presenta microfonismo. Sin embargo su problema es la *corriente de negro,* es decir, que existe una corriente o señal de vídeo con una imagen en negro ó cuando no incide luz proveniente de una escena, por lo cual trabaja con mucha luz. Otro problema es la *inercia en el valor resistivo*, por lo que la imagen sigue presente después de que desapareció. Este defecto se conoce como *LAG* (atraso), es decir, la imagen se atrasa respecto a la anterior.

a.1.6 Plúmbicon

El *plúmbicon* se conoce también como *leadicon* (Inglaterra) y *saticon* (Japón), es un tubo de cámara de *oxido de plomo PbO* (**Figura a.7**) [Ramirez 70].

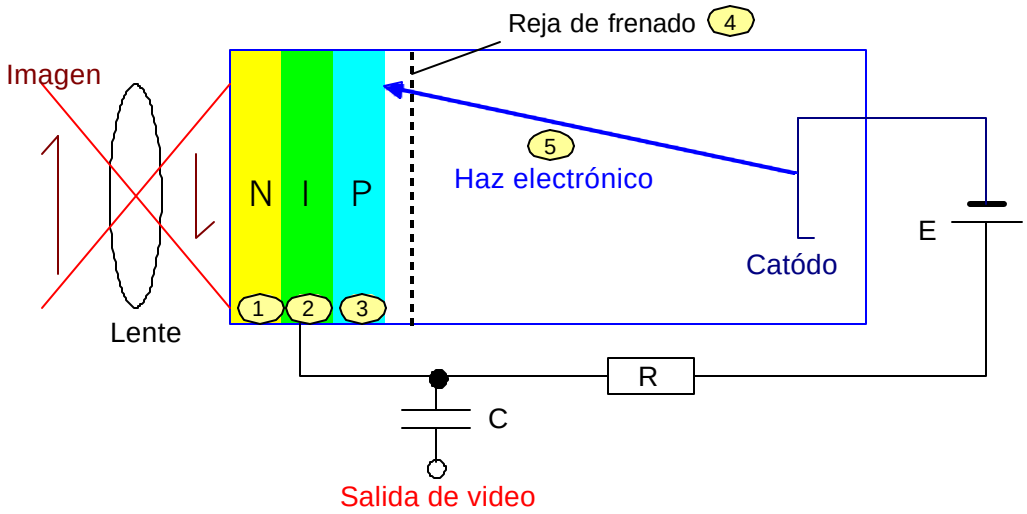


Figura a.7: Plúmbicon.

Ahora se tienen 3 capas, una semiconductora *tipo N* **(1),** una *intrínseca I* **(2)** y una semiconductora *tipo P* **(3),** las tres forman un diodo tipo *PIN*. Funciona de forma similar al *Vidicon*, solo que se agrega una capa más con lo que se crea un diodo. Cuando no hay luz la impedancia es muy grande. Con esto se incremento la eficiencia del tubo respecto al *vidicon*, pues ahora se requiere poca luz para producir una buena imagen. La *reja de frenado* **(4)** se colocó para que el haz electrónico **(5)** no golpee a la primera capa del diodo *PIN*, por lo que a este tipo de exploración de la imagen se conoce como *exploración de baja velocidad*. Su vida útil es de unas 8,000 horas. Es un tubo pequeño y no presenta microfonismo.

a.1.7 Arreglos de dispositivos de transferencia de carga

Los arreglos de *dispositivos de transferencia de carga CCD* (*Charge Coupled Device*) son los más empleados en los sistemas actuales de cámara, debido que presentan una gran sensitividad sobre un amplio espectro de luz, bajo consumo de potencia, tamaño reducido, peso ligero y una vida útil mayor a la de cualquier tubo [Staugaard 87].

Un *CCD* se construye por integración de un arreglo de transistores *MOSFET* (*Metal Oxide Semiconductor Field Effect Transistor*). En la **Figura a.8** se muestra una porción de un arreglo de este tipo. Cada transistor constituye una celda individual que conforma una serie de capacitares **(1)** integrados los cuales almacenan una serie de cargas de forma temporal.

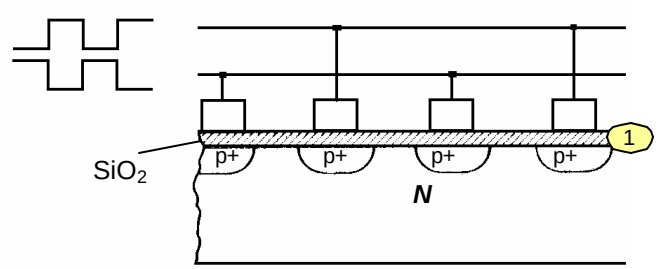


Figura a.8: Construcción de un dispositivo de transferencia de carga.

En la **Figura a.9** se presenta una celda. Se observa como los materiales *tipo P* **(2)** se encuentran aislados por el material *tipo N* **(3),** así es como se conforma la *fuente* **(4)** y el *drenaje* **(5)** de un dispositivo *MOSFET*, ambas zonas están separadas por una parte del substrato *tipo N*, llamado *región de agotamiento* **(6).**

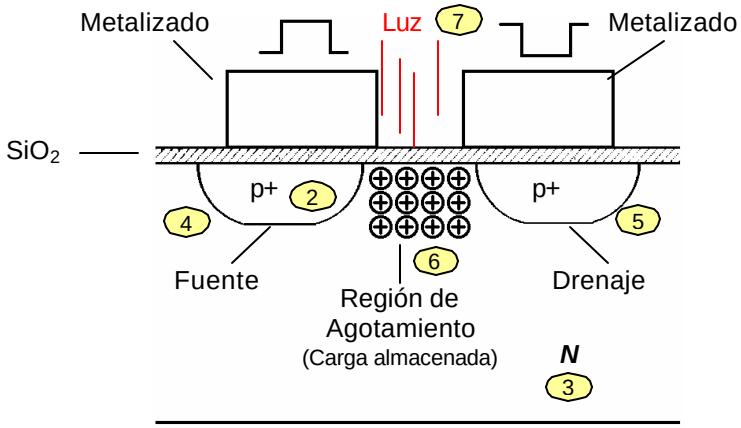


Figura a.9: Funcionamiento de una celda de un dispositivo de transferencia de carga.

Cuando los fotones **(7)** inciden liberan electrones en la región de agotamiento **(6).** Estos generan iones positivos, por lo que se genera una carga positiva en la región de agotamiento **(6).** Está carga es directamente proporcional a la intensidad de la luz que incide sobre el arreglo. De está forma podemos considerar que se tiene una serie de capacitores *MOS* (*Metal Oxide Semiconductor*), cada uno con una carga proporcional a la intensidad de la luz en ese punto.

Las cargas individuales son transferidas de una celda *CCD* a la siguiente, aplicando pulsos de reloj entre la *fuente* y el *drenaje* con una fase de 180° entre ellos. De esta forma cada fila en el arreglo funciona como un *registro de corrimiento serie*. Cuando se recorre de forma continua cada fila se obtiene una señal de vídeo (**Figura a.10**).

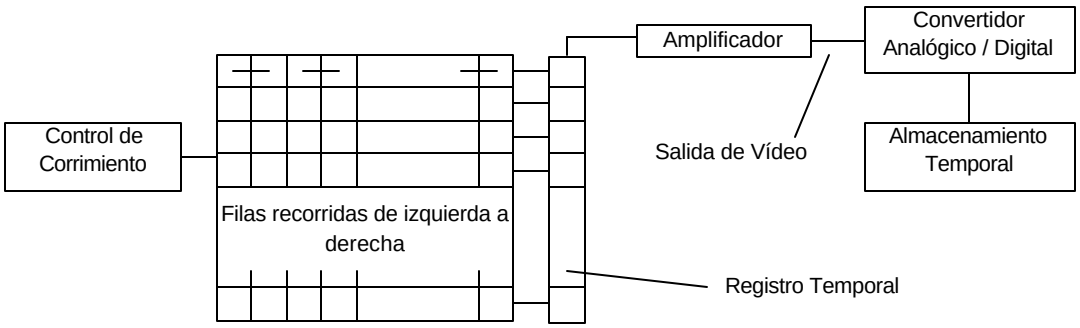


Figura a.10: Control lógico usado para transferir cargas en cada fila en un CCD.

Sin embargo el almacenamiento de cargas puede ser más rápido que la generación de la señal de vídeo, por lo que se usa una *memoria temporal de cuadro* (*frame buffer*) para almacenar temporalmente la salida entera del arreglo y ser almacenada en la computadora.

Las cámaras *CCD* están disponibles en arreglos de una ó dos dimensiones. Los arreglos de una dimensión se conocen como *Dispositivos Lineales de Imágenes LID* (*Linear Imaging Device*) y los de dos dimensiones son llamados *Dispositivos de área de imagen AID* (*Area Image Device*).

a.1.8 Arreglos de dispositivos de inyección de carga

Los *dispositivos de inyección de carga CID* (*Charge Injection Devices*) operan con un almacenamiento de carga similar a los *CCD*. Cuando la luz incide sobre una celda de estado sólido *CID*, una carga que es proporcional a la luz es inyectada en la celda. Son arreglos de filas y columnas, y su lectura hace referencia a las filas y columnas correspondientes, es decir, de la misma forma en que se da lectura a una memoria de computadora. Como resultado, el almacenamiento de cargas no se obtiene por un corrimiento, mas bien por un control lógico de lectura, por esto es llamado un dispositivo de *acceso paralelo*.

a.2 Composición de las cámaras captadoras de imágenes electrónicas

Un sistema de adquisición, no solo esta compuesta por los sistemas transductores de señales luminosas a eléctricas, sino también tienen una parte óptica, encargada de enfocar la imagen en los captores de imagen.

Una cámara electrónica de video está compuesta de dos partes: la *parte electrónica*, que esta formada por *los captores de imagen* y la *parte óptica*. La *parte óptica* de la cámara esta formada por: una parte interna (separadores ópticos) y poruna parte externa (lentes ó Zoom), la primera parte se explicó en los apartados anteriores, es los apartados siguientes veremos la parte óptica de la cámara.

a.2.1 Separador óptico

La función de un *separador óptico* es descomponer una imagen en sus componentes *rojo*, *verde* y *azul*, para poder ser procesada por tres tubos en *B/N* (blanco y negro) y obtener una señal en color [Escalera 01]. Un separador óptico se forma de cristales *dicroicos* (**Figura a.11**).

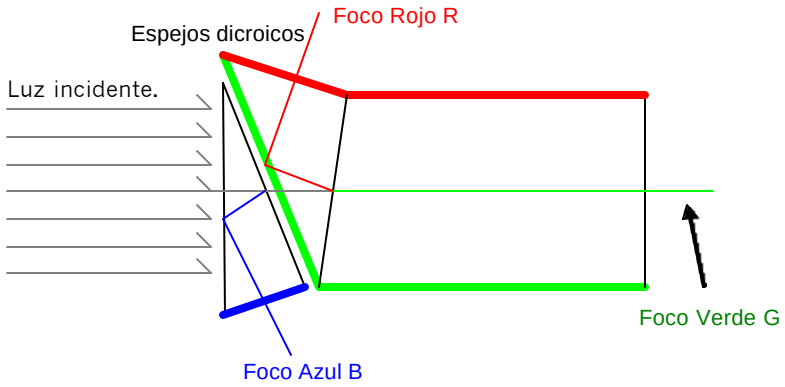
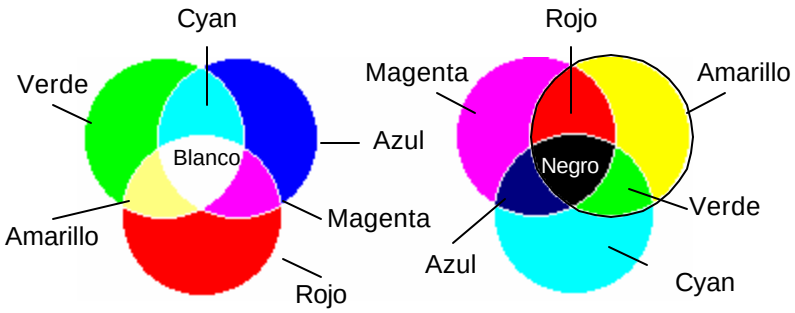


Figura a.11: Composición de un separador óptico.

Un espejo *dicroico* es aquel que responde a una longitud de onda de la luz, dichos espejos se controlan mediante su espesor. En cada uno de los focos se encuentra un tubo de cámara destinado a capturar la intensidad del color correspondiente (rojo, verde o azul). El tubo *verde* siempre se coloca en el eje óptico del sistema, por que se encuentra a la mitad del espectro entre el rojo y el azul.

La descomposición de la escena en tres colores corresponde a la forma en que se cree que el ojo humano funciona. Sin embargo también tiene que ver con la manera en que se realiza la reconstrucción de la imagen en un cinescopio de televisión.

El ojo humano ve mediante un *sistema substractivo* de luz, la imagen en la cámara (que será presentada en un cinescopio) se crea por un *sistema aditivo* en la **Figura a.12** se observan las diferencias entre ambos mecanismos de descomposición de la luz.



a) Síntesis Aditiva **b)** Síntesis substractiva

Figura a.12: Síntesis aditiva y substractiva del color.

El *color blanco* que es la referencia en sistemas de imágenes en color se crea mediante la relación de *luminancia* dada por la **Ec. a.1**.

$$\text{Blanco} = \text{LUMINANCIA} = Y = 0.3R + 0.59G + 0.11B \qquad \text{...(Ec. a.1)}$$

La **Ec. a.1** puede ser utilizada para convertir imágenes de color en B/N, para que puedan ser procesadas por nuestro sistema de reconocimiento.

a.3 Lentes

El primer componente de un sistema de adquisición es desde luego el sistema óptico formado por lentes, pues estás se encuentran en contacto con la imagen de la escena a procesar.

Un *lente* es más eficiente entre mas luz deje pasar [Stollberg 78]. Las características relacionadas con los lentes se observan en la **Figura a.13**.

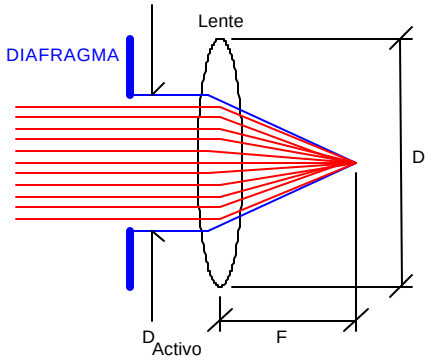


Figura a.13: Características de una lente.

Donde:
D .- es el diámetro de la lente.
F .- es la distancia focal de la lente. Determina el tamaño de la imagen.
$\mathbf{D_{Activo}}$.- es el diámetro real por el que pasa la luz, se regula mediante el *diafragma*, solo cambia el diámetro pero el foco sigue en el mismo lugar.

La *relación del lente f* es un indicativo de la eficiencia de la lente. Entre mayor sea, mejor es la lente por que la cantidad de luz es mayor. En algunos casos es mayor a 1, por que el diámetro del lente es mayor que su foco (**Ec. a.2**).

$$f = F/D_{Activo} \qquad \text{...(Ec. a.2)}$$

Para un lente ideal F=*1*, 0% de luz reflejada, 0% de distorsión cromática y 0% de distorsión geométrica. Los *F* comerciales son: $F_{comerciales}$ = 1.2, 1.4, 2.4, 2, 4, 8, 16, 22, 32, etc.

El *tamaño de la imagen* esta determinado por la relación que exista entre el foco y la imagen formada por la lente. Así se tienen:

a) Normal.- Tiene una captación angular media de 45°.

b) Gran angular. - Tiene una captación angular grande 180° ó más.

c) Telefoto.- Tiene una captación angular pequeña entre 4° y 5°.

Por ejemplo para un *F=20mm*, seria un lente *gran angular* para una imagen de *8mm*, mientras que seria una *telefoto* para una imagen de *35mm*, debido al tamaño de la imagen que forma sobre el sensor.

Los *Zoom* son lentes de distancia focal variable. En realidad son un conjunto de lentes que equivalen a uno solo de distancia focal variable con características de alta calidad. Son lentes de costo elevado por que los elementos que los componen deben de ser muy finos para lograr las características deseadas. Con este tipo de lentes no existen errores geométricos ni cromáticos. Los lentes que forman un *Zoom* pueden ser de distintas clases de cristales, acrílicos y plásticos para lograr una excelente transparencia (**Figuras a.14** y **a.15**).

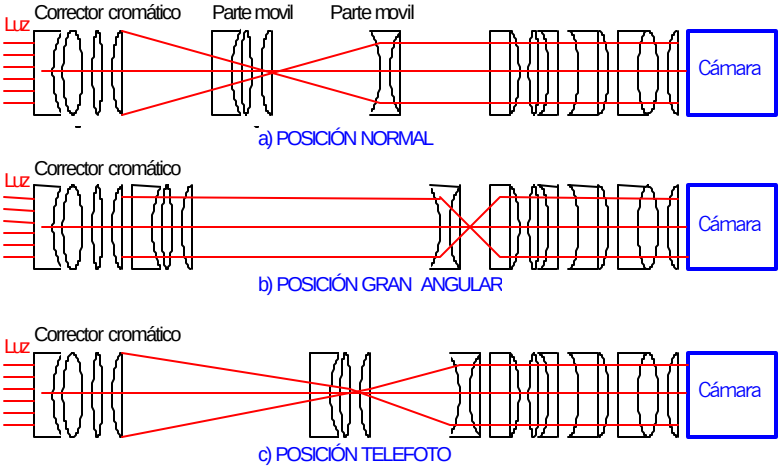


Figura a.14: Posiciones de un Zoom.

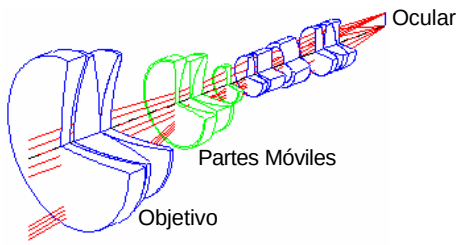


Figura a.15: Partes constitutivas de un Zoom.

La nomenclatura *Zoom 10X,* implica una distancia focal máxima es de *10xF*, donde *F* es el foco que marca el *telefoto*, pero no es la relación de la lente. *Por ejemplo:* Para un *Zoom=10X* y *F=15* implica que *F* varia de la forma: *F=15 a 150.*

Resumen

Los sistemas de adquisición de imágenes son el primer elemento para el proceso de visión artificial. Permiten traducir una imagen en una serie de señales eléctricas que la computadora puede interpretar para almacenarla y procesarla.

Existen tres tipos de transductores para la adquisición de imágenes: los foltovoltaicos que producen un voltaje proporcional a la luz incidente, los fotoemisivos que emiten electrones al incidir luz sobre de ellos y los fotoconductivos, los cuales cambian su impedancia de forma proporcional a la cantidad de luz.

Por aparición histórica, los primeros sistemas fueron los mecánicos como el disco de Nipkow, luego los fotoemisivos como el disertor de imagen, el iconoscopio y el orticon, por ultimo los fotoconductivos como el vidicon, el plúmbicon y los dispositivos de transferencia de carga ó CCD. Los CCD son los que aún se siguen usando por su bajo peso, confiabilidad, bajo consumo de potencia, larga vida, etc.

Una cámara para adquisición de imágenes consta no solo de los dispositivos captores de imagen sino también separadores ópticos y lentes. Un separador óptico tiene como función principal descomponer una imagen en colores, de forma substractiva, obteniendo entonces tres señales eléctricas correspondientes a los colores rojo, verde y azul. Las lentes son las encargadas de enfocar la escena en el sensor de imagen. Las lentes son más eficientes entre más luz dejen pasar. Sin embargo para captar imágenes de alta calidad son necesarios conjuntos de lentes llamados Zoom.

Apéndice B
Iluminación Estructurada

La *iluminación estructurada* es una forma simple de obtener patrones tridimensionales de una escena. Este tipo de iluminación puede ser empleada para la modelación de objetos sin perdida de información, debido a que fragmenta una escena tridimensional a rebanadas bidimensionales (recuérdese que una cámara solo puede capturar escenas bidimensionales). El procedimiento que describimos en esté apéndice fue definido en el **Capítulo 3: Visión Artificial,** y puede ser aplicado para la reconstrucción tridimensional y/o reconocimiento de los objetos vistos en los **Capítulos: 2. Ambientes Virtuales y VRML** y **5. Diseño e Implementación de un Sistema de Visión.**

Los sistemas de *iluminación estructurada* para reconstrucción tridimensional no son nuevos, sin embargo su empleo se encuentra limitado debido a los costos en cómputo y económicos que representa su implementación. Este apéndice presenta una propuesta de un sistema de este tipo que emplea una cámara web y una fuente de luz láser comercial de bajo costo. Además se propone realizar la reconstrucción tridimensional con *VRML* (*Virtual Reality Modeling Language*), el cual genera códigos pequeños que facilitan su intercambio por Internet. Se realizaron algunas pruebas con piezas industriales obteniendo resultados aceptables para aplicaciones tales como: detectar la presencia de un objeto, orientación, clasificación, control de calidad, cálculo de medidas, etc. La reconstrucción por *VRML* resultó ser una de las mejores en presentación y espacio requerido para la representación en tres dimensiones de una escena.

b.1 Sistemas de iluminación estructurada

Los sistemas de *iluminación estructurada* son una de herramienta sencilla para obtener características de objetos en tres dimensiones. Un sistema de visión basado en *iluminación estructurada* obtendrá fácilmente información en tres dimensiones de los objetos bajo estudio. Además la *iluminación estructurada* permite la visualización casi natural de los objetos en tercera dimensión, por lo que se puede examinar una gran cantidad de características que de otra forma sería más costoso (en tiempo y cómputo) obtener.

Los sistemas de adquisición de características tridimensionales en base a *iluminación estructurada* iluminan uniformemente toda la escena con un patrón. Actualmente existen láseres que se obtienen por un precio bajo en el mercado [Schilling 90], por lo que estos patrones pueden ser creados a partir de luz láser. Se pueden usar además lentes y máscaras ópticas, con lo que se tienen una gran variedad de patrones de luz que pueden ser proyectados en la escena donde se encuentran los objetos de interés.

Cuando se presenta un objeto en la escena se tiene una *modulación* del patrón luminoso, la cual representa las características en tres dimensiones de la perspectiva del objeto. Examinando el patrón de modulación de luz, se puede inferir la presencia del objeto, las dimensiones y la orientación de las superficies que han sido iluminadas con el patrón. Si este proceso es repetido sobre diferentes posiciones del objeto es posible obtener un modelo tridimensional del objeto bajo estudio.

Con un patrón simple de luz puede ser posible detectar la presencia y la altura del objeto en la escena. Un sistema sencillo consiste en la creación de una *línea de luz* que es proyectada sobre la escena (está línea puede ser creada a partir de un haz láser) y una cámara fija que se encarga de capturar la imagen así generada (**Figura b.1**). Si el *eje x* de la cámara coincide con la línea de luz, entonces se observa una porción del haz deformado, que coincide con las características del objeto.

La distancia abajo del *eje x* proporciona una indicación de la *altura* del objeto, mientras la línea que aparece sobre el objeto proporciona su *ancho*. Si el objeto se traslada a lo largo del *eje y* ó este permanece estacionario pero se mueve el haz a lo largo del *eje y*, entonces se obtienen las características tridimensionales del objeto bajo estudio, como el ancho, la altura y la longitud.

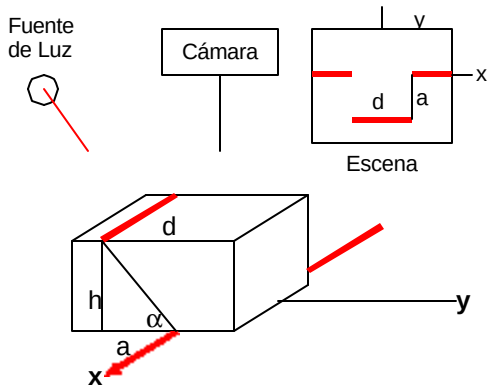


Figura b.1: Modulación de un haz de luz por un objeto.

El *ancho* del objeto en píxeles es *d*, con *a* como la distancia en píxeles entre la línea *d* y el *eje x*, es posible obtener la *altura (h)* en píxeles de la forma que se muestra en la **Ec. b.1**.

$$h = a \tan \alpha \qquad \text{...(Ec. b.1)}$$

El *largo* se obtiene contando el número de imágenes en donde aparece una modulación del *eje x*, y conociendo el desplazamiento del objeto ó del haz por cada toma, por lo que bastará multiplicar el número de tomas en las que aparece una modulación por su desplazamiento por toma.

b.2 Triangulación

Un método común para medir la profundidad de un punto en particular, es usando *triangulación* [Kanade 87, Schilling 90]. Para estimar la profundidad es posible utilizar un arreglo con una fuente de luz y una cámara como la que se muestra en la **Figura b.2**, la fuente de luz puede provenir de un *haz de luz láser* u otra capaz de producir un rayo fino de luz. De la instalación del sistema se puede determinar *d* como la distancia horizontal entre el haz incidente y la lente de la cámara y α como el ángulo entre el haz incidente y la horizontal.

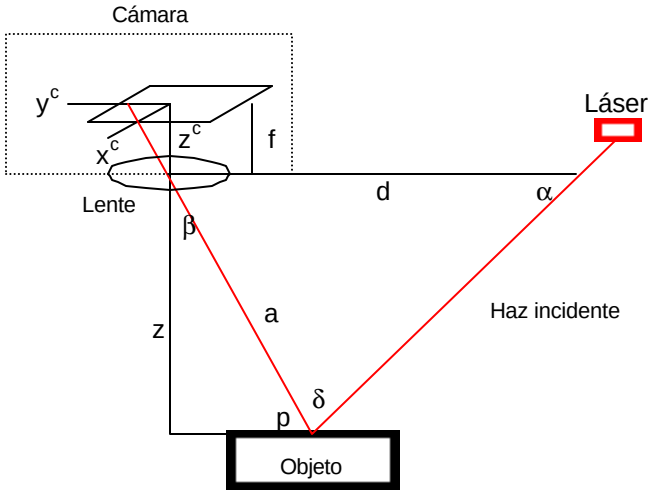


Figura b.2: Geometría de la triangulación.

El haz incidente llega al objeto en el punto *p*. Las coordenadas del punto *p* respecto al cuadro de cámara son $C=\{x^c,y^c,z^c\}$; donde β es el ángulo que forman *C* y la distancia focal efectiva *f*. β puede ser calculado empleando relaciones trigonométricas. Conociendo *C* y *f* puede calcularse β como el ángulo entre el haz incidente y el reflejo que llega a la cámara. Usando leyes de senos y conociendo α y *d* se puede calcular *a*. Por último *a* puede ser usada para conocer la *distancia vertical z*, con lo que se conoce la profundidad del punto en cuestión (**Ecs. b.2** y **b.3**).

$$\beta=\arctan(C/f) \qquad \text{...(Ec. b.2)}$$

$$z=[d \sin\alpha \cos\beta]/\cos(\alpha-\beta) \qquad \text{...(Ec. b.3)}$$

b.3 Implementación de un sensor de características tridimensionales (Hardware)

Para detectar las características en tres dimensiones de un objeto empleamos una *línea de luz* la cual nos proporciona, para una toma determinada, el contorno con las características del objeto en la rebanada correspondiente, es decir, se tiene una *modulación* del haz de luz en la escena.

Uno de los primeros aspectos a considerar en el diseño de este tipo de sistemas es como generar el *haz de luz*. Considerando que en la industria es muy probable que el objeto se deslice en algún eje (como en una banda transportadora), el haz láser puede mantenerse inmóvil, mientras el objeto se mueve tomando imágenes a intervalos iguales de tiempo.

Para propósitos de prueba, se realizó un circuito encargado de controlar *un motor paso a paso* **(1)** que permite mover los objetos en el *eje y*, con gran precisión, realizando una toma (fotografía) en cada paso. El circuito además presenta un control para el *motor de corriente continua* (CC) **(2)** destinado a mover un espejo para crear una *línea de luz láser*, aunque es conocido que también es posible generarla por medios ópticos. Tiene también un *excitador del láser*, este no es muy complejo debido a que se empleo un láser comercial de baja potencia, por lo que un circuito *TTL* (*Transistor-Transistor Logic*) es suficiente para su funcionamiento. Todo el circuito es compatible con niveles *TTL*, por lo que se puede conectar de forma directa a una computadora. El circuito construido se presenta en la **Figura b.3**, se activa de forma manual, tanto en el encendido del láser como cada paso del motor que controla el movimiento sobre el *eje y*.

El *haz láser* usado, provino de un diodo láser comercial (**Figura b.4**). Este tipo de diodos requiere para su funcionamiento de un circuito similar al de un *diodo emisor de luz* (*LED*), sin embargo la mayoría también integra un sensor de intensidad de luz emitida, el cual se emplea para mantener constante la intensidad del haz. Para los diodos que operan en el espectro visible (regularmente de *color rojo*), el circuito viene integrado en el mismo substrato que conforma el diodo láser, por lo que para su operación sólo se requiere colocar un resistor en serie a una fuente de alimentación, de hecho solo consume una corriente un poco mayor a la de los *LED* comunes.

Para mover el *rayo de luz* es necesario emplear algún medio óptico ó mecánico, por lo que se eligió un espejo rotatorio (mecánico) para mover el haz de un lado a otro de la escena de forma constante, sin embargo la velocidad a la que se mueve el haz debe de ser cuidadosamente ajustada, pues depende de la velocidad del obturador de la cámara que se encuentre tomando cada una de las imágenes necesarias para la reconstrucción de los objetos. En el circuito de la **Figura b.3**, se observa la forma en que se ajusta la velocidad del motor *de CC*, a través de un potenciómetro que controla la corriente de la base de un transistor. Hay que cuidar que la frecuencia angular del espejo giratorio sea diferente a algún múltiplo o submúltiplo de la frecuencia vertical de la cámara, por ejemplo, si la frecuencia vertical de la cámara es de *30Hz* la frecuencia angular a la que gira el espejo, deberá de ser un múltiplo diferente de está, sin embargo dependiendo de la potencia del láser empleado está frecuencia podrá ser tan elevada como sea necesario.

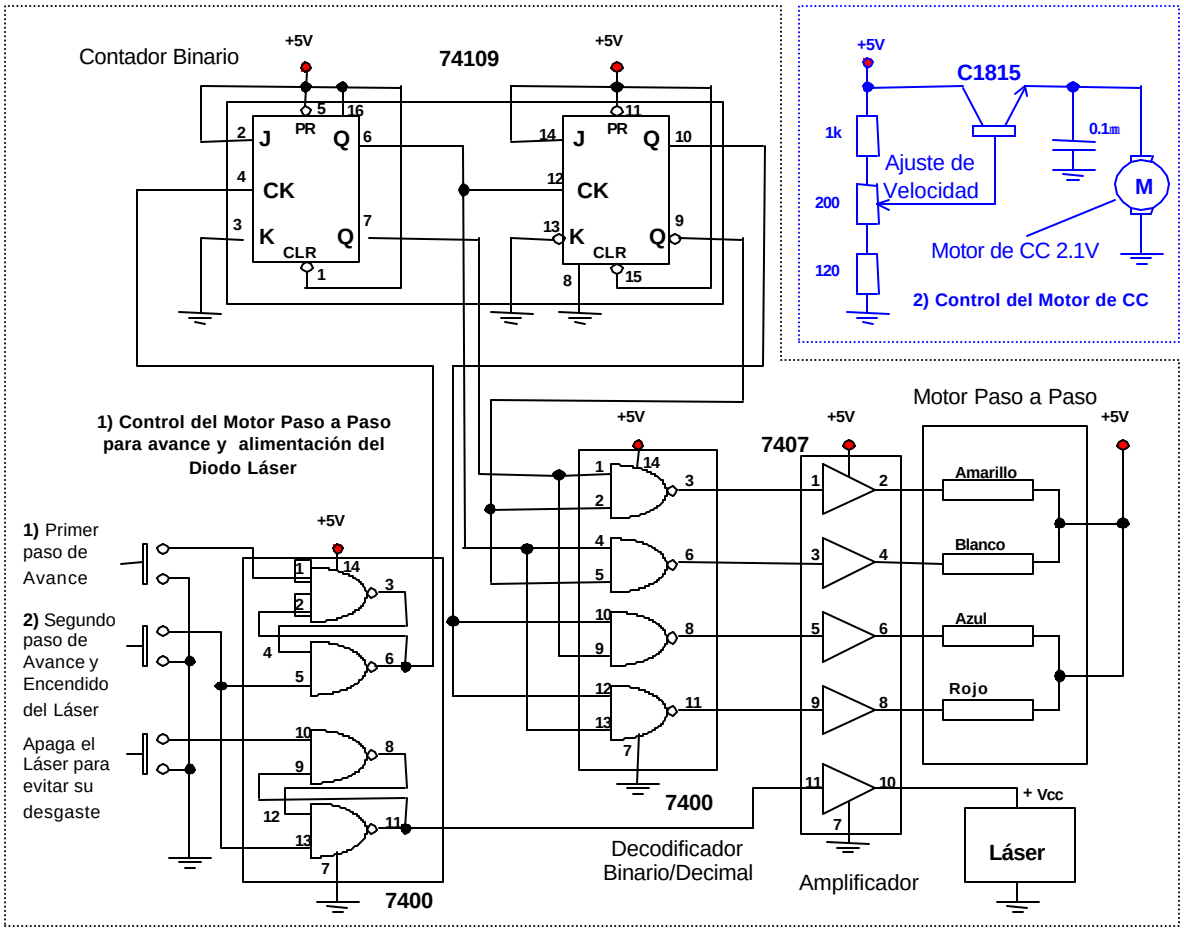


Figura b.3: Circuito de control para el sistema de iluminación estructurada.

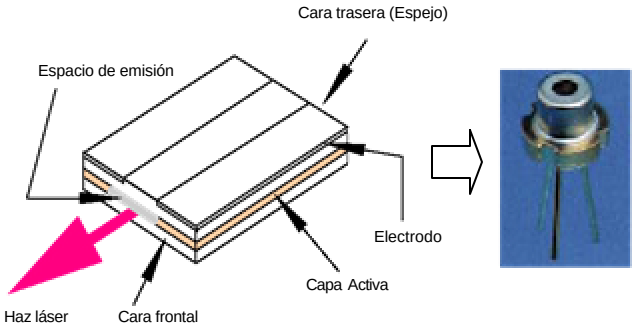


Figura b.4: Constitución de un diodo láser.

El láser se mueve perpendicular a la escena, por lo que para poder captar la perspectiva del objeto es necesario que la cámara mantenga un ángulo, el cuál permita captar todo el objeto, como se ilustra en la **Figura b.5**.

Si consideramos que se conoce el ángulo b que forma la cámara respecto a su vertical podemos calcular a como se muestra en la **Ec. b.4**.

$$a = 180° - b \qquad \text{...(Ec. b.4)}$$

Y luego emplear las ecuaciones de apartado **b.1** y **b.2** para él cálculo de las características de los objetos. El ancho de la toma *p,* es igual a la cantidad de píxeles horizontales de la cámara.

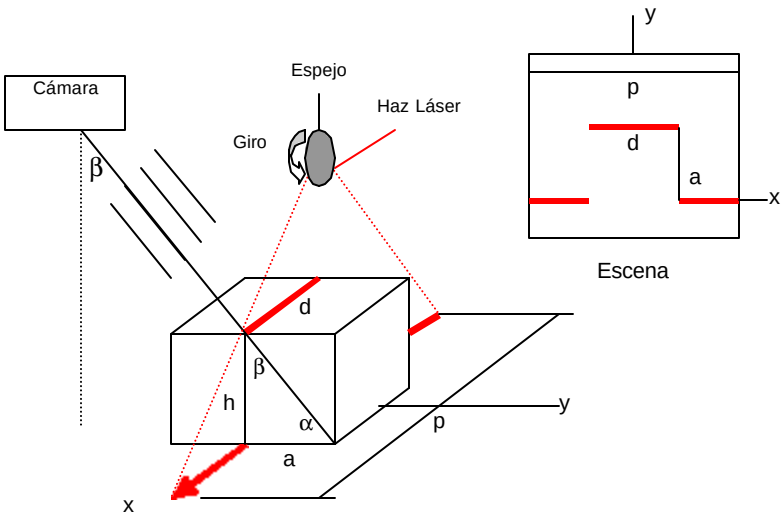


Figura b.5: Deformación del haz láser por un objeto.

Existe una limitación importante, en la distancia de separación entre objetos para evitar que unos no cubran parcialmente a otros, esta distancia es *a*, por lo que la altura también se encuentra limitada, de otra forma el sistema detectara solo al objeto más próximo a la cámara y de forma parcial a los objetos del resto de la escena.

El sistema de reconstrucción tridimensional mediante *iluminación estructurada*, se muestra en la **Figura b.6**, se observan algunos objetos bajo estudio, el láser, los circuitos, mecanismos y la cámara web.

Figura b.6: Fotografía del sistema de reconstrucción tridimensional.

b.4 Análisis de las imágenes (Software)

Para obtener mejores resultados es necesario ajustar los parámetros de entrada de la cámara para que sea sensible al haz de luz, es decir, que el haz sea preferentemente lo más brillante de la escena. Para ajustar los parámetros del programa es necesario proporcionar los datos de la cantidad de centímetros en el *eje x*, el avance en el *eje y,* y el ángulo de la cámara *α*. El programa calcula el *ancho* y la *altura* de cada imagen contando el número de píxeles que existen en los dos ejes, y la longitud del objeto contando la cantidad de imágenes en las que aparece una

modulación del haz láser (**Figura b.7**).

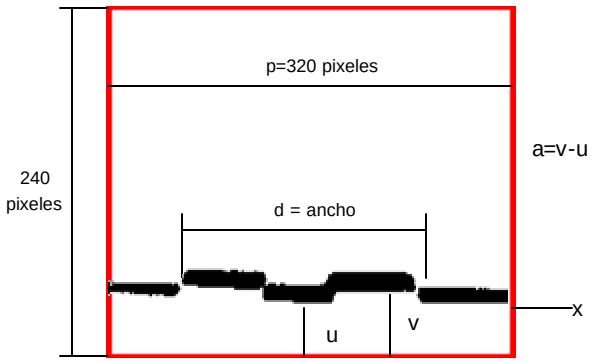


Figura b.7: Imagen de un objeto para extracción de características.

El programa diseñado sigue los pasos que a continuación se enumeran (**Figura b.8**) para obtener un modelo en *VRML* (*Virtual Reality Modeling Language*) de la escena explorada.

1) Recibe como entrada las fotografías tomadas de la escena **(1).** Y los archivos de nombre de las imágenes **(2)** y de calibración de cámara **(3).**
2) Transforma las imágenes a un formato de *mapa de bits monocromático* **(4),** en caso de ser necesario. Las fotografías tomadas deben quedar en el formato de *mapa de bits monocromático*, para poder ser leídas **(5)** y procesadas por el programa.
3) Realiza el análisis de cada una de las fotografías encontrando las profundidades del objeto **(6).**
4) Ajusta los ejes a las medidas reales, por medio de los archivos de calibración, y calcula los puntos en el espacio **(7)** para obtener la matriz 3D **(8).**
5) Escribe el código *VRML* que representa los datos tridimensionales obtenidos, en una modelación con de líneas y otra con mallas **(9)** (**Figuras b.11b, b.11c, b.12b y b.12c**).
6) Presenta los datos en una gráfica tridimensional en *Lenguaje* C **(10)** [Foley 96, Johnson 87] (**Figuras b.11a y b.12a**).
7) Genera archivos de la matriz tridimensional y una matriz con puntos y vértices en tres dimensiones para usarse con *Java* **(11).**

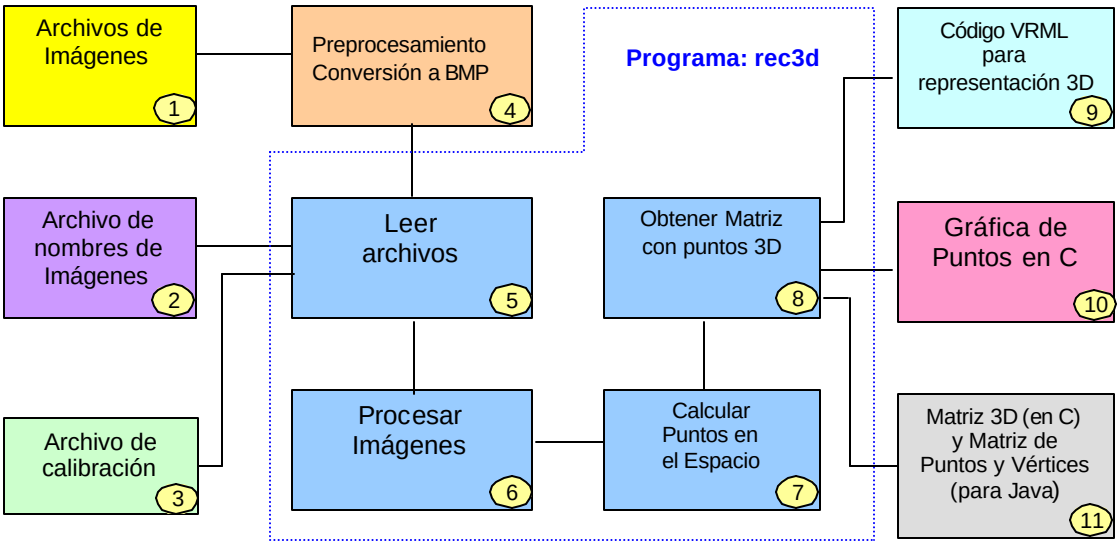


Figura b.8: Programa de reconstrucción tridimensional mediante iluminación estructurada.

b.5 Resultados

La velocidad angular a la que se mueve el espejo giratorio, se debe ajustar para que en una toma, no se corte y aparezca como una línea continua, para laceres de baja potencia como el empleado, la frecuencia no puede ser muy elevada. Se obtuvieron buenos resultados empleando *frecuencias angulares* menores de *20Hz*, para un *láser óptico* de *680nm* con *3mW* de potencia.

El tiempo necesario obtenido para procesar *imágenes* de *76,800 píxeles* (*320x240* píxeles), se muestra en la **Figura b.9** y es posible calcularlo mediante la **Ec. b.5**.

$$\text{Tiempo de Procesamiento (s)} = -0.0443281 + 0.0132724n - 0.00010914n^2 \quad \text{...(Ec. b.5)}$$

Donde: **n** = es el número de imágenes de la escena bajo análisis.

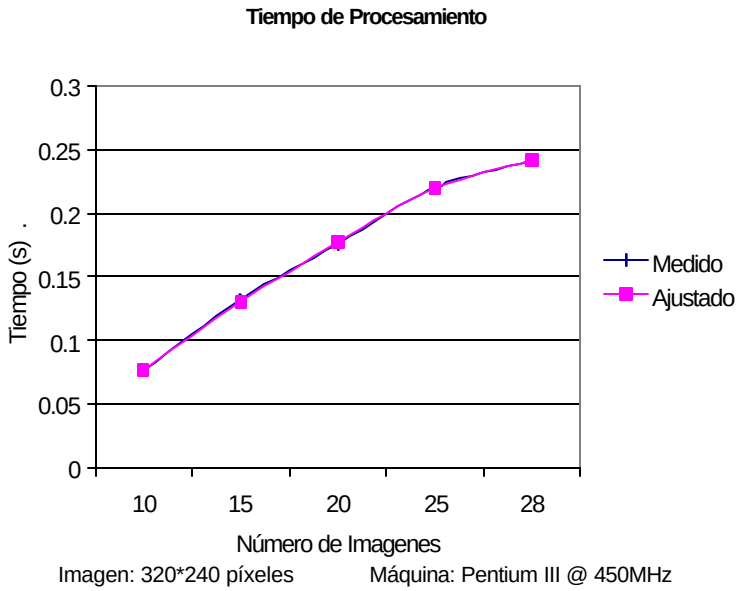


Figura b.9: Tiempo de procesamiento de imágenes para reconstrucción tridimensional.

El tiempo obtenido es lo suficientemente bajo, como para que las imágenes sean procesadas en un tiempo aproximadamente igual al de la velocidad de *sincronización vertical* de televisión usado en México, que es de *33.3 milisegundos*.

El tamaño de los archivos generados (**Figura b.10**) con la información necesaria para realizar la reconstrucción tridimensional (3D) se puede calcular por medio de las **Ecs. b.6**, **b.7**, **b.8** y **b.9**.

$$\text{Matriz de Reconstrucción (bytes)} = 19.8 + 1{,}244.7n + 0.594286n^2 - 0.0133333n^3 \quad \text{...(Ec. b.6)}$$

$$\text{Archivo para Java (bytes)} = 124{,}785 + 4{,}494.76n \quad \text{...(Ec. b.7)}$$

$$\text{Archivo de VRML con Líneas (bytes)} = 4{,}869.6 + 9{,}721.3n + 13.42n^2 \quad \text{...(Ec. b.8)}$$

$$\text{Archivo de VRML con Malla (bytes)} = 7{,}288.8 + 1{,}767.31n + 13.4143n^2 \quad \text{...(Ec. b.9)}$$

Donde: **n** = es el número de imágenes de la escena bajo análisis.

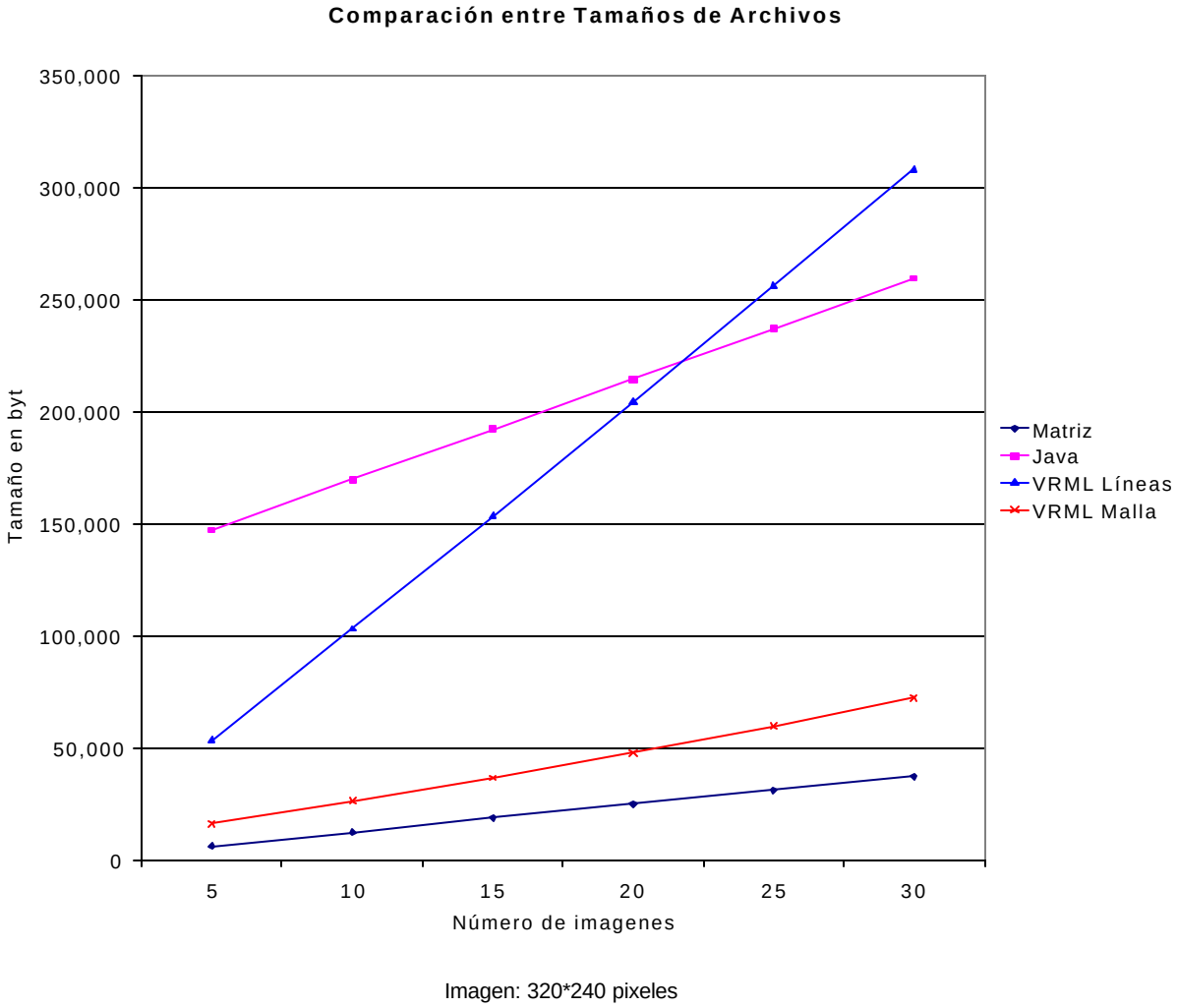


Figura b.10: Tamaño de los archivos generados durante la reconstrucción tridimensional.

En la **Figura b.11** se muestran los resultados obtenidos al colocar en la escena una *roldana con un diámetro de 2 pulgadas*, realizando un total de 27 tomas cada una con un tamaño de *320*240* píxeles, con *2mm* de separación en el *eje y*. En la **Figuras b.12** se muestran los resultados obtenidos al colocar en la escena tres objetos de diferente altura, realizando un total de 28 tomas cada una con un tamaño de *320*240* píxeles, con *2mm* de separación en el *eje y*.

La calibración se realiza de forma muy simple, para ello es necesario conocer cuanto se desliza el objeto cada vez que se realiza una toma, este valor en milímetros corresponde al *dato y* del archivo de calibración, luego se toma una fotografía de un objeto de dimensiones conocidas para hallar el número de píxeles correspondientes a la modulación en el *eje x* y calcular cuantos píxeles por milímetro se tienen en dicho eje.

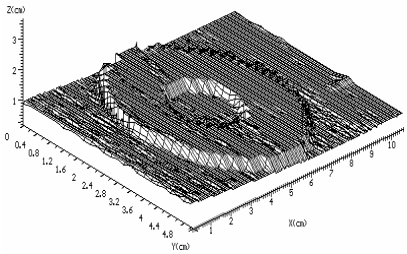
El programa realizado genera los siguientes archivos de salida:

1) *Matriz de puntos en el espacio tridimensional*.- contiene los puntos calculados en donde los dos primeros números representan el número de tomas y el ancho de la imagen (para obtener los *ejes x* y *y*), después viene el conjunto de puntos en que representan al *eje z*. Se almacena con el nombre: **`nombre_archivo`**`.mtz`.

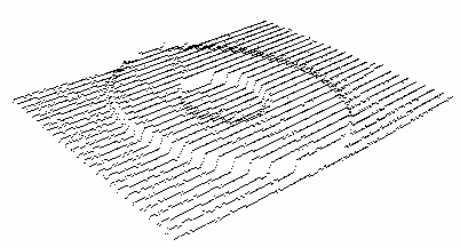
2) *Matriz de vértices y líneas.*- este archivo es usado para la reconstrucción tridimensional en lenguajes como *Java*, un programa de visualización tridimensional viene con los ejemplos de *Java JDK 1.3* en el directorio: `jdk1.3\demo\applets\WireFrame`. Los archivos de este tipo se guardan como: **`nombre_archivo`**`.pts`.

3) *Modelo Virtual con Líneas.*- es un archivo *VRML*, el cual puede ser ejecutado en un navegador (una vez instalado en *Plug-In* correspondiente). El modelo se almacena con el nombre: `m`**`nombre_archivo`**`.wrl`.

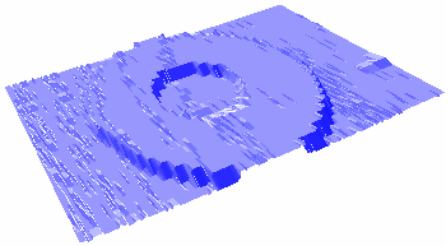
4) *Modelo Virtual con Malla.*- es un archivo *VRML*, el cual puede ser ejecutado en un navegador. El modelo se almacena con el nombre: `l`**`nombre_archivo`**`.wrl`.



a) Lenguaje C

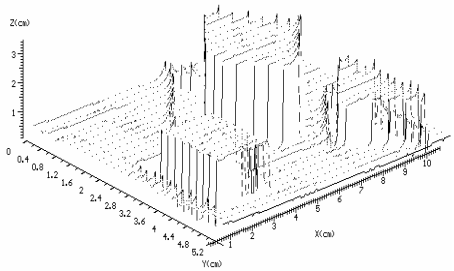


b) Líneas en *VRML*

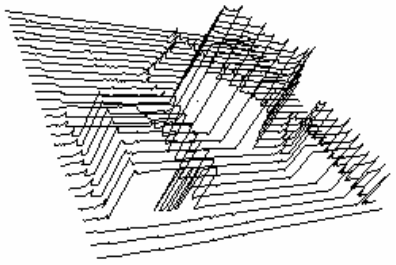


c) Malla en *VRML*

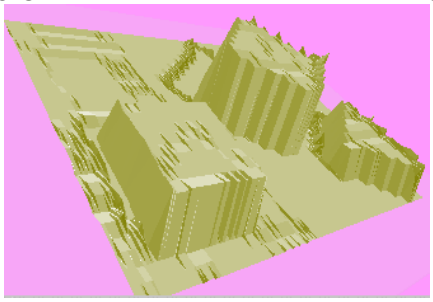
Figura b.11: Resultados para la reconstrucción tridimensional de una roldana.



a) Lenguaje C



b) Líneas en *VRML*



c) Malla en *VRML*

Figura b.12: Resultados para la reconstrucción tridimensional de la mesa de trabajo de un robot con diversos objetos.

Resumen

Los sistemas de caracterización de objetos en 3D son antiguos, sin embargo su empleo se ha limitado debido principalmente a los costos que involucran. Debido a que en los últimos años el costo de los equipos de computo y de elementos de captura de imágenes (cámaras web) se ha reducido por lo que es posible implementar sistemas como el aquí descrito para múltiples aplicaciones industriales a costos reducidos.

La implementación del hardware aquí presentada, es una tarea sencilla si se tienen algunos conceptos básicos de electrónica, e incluso puede no ser necesario si, por ejemplo, se conocen los parámetros cinemáticos de una banda transportadora, así solo es necesario ajustar los archivos de calibración del programa.

La cámara web es una cámara convencional, que solo requiere de ajustes necesarios para captar de forma adecuada el haz de luz láser. El diodo láser puede hallarse con facilidad y a un bajo precio en el mercado.

Como se ha demostrado, el tiempo de ejecución para crear el modelo de la escena es muy bajo, por lo que permitiría un alto rendimiento del sistema, empleándolo; por ejemplo, como un sistema de inspección de partes automotrices ó como un medio de obtener parámetros (medidas) de objetos que se presenten en una banda transportadora, para control de calidad, para identificación, para análisis de piezas metálicas desde un puesto de trabajo remoto mediante el uso de *Internet* y *VRML*, etc.

Además se encontró que los archivos generados por el programa para la reconstrucción en 3D mediante mallas en *VRML*, son pequeños comparados con el archivo para *Java* y el mismo *VRML* pero empleando líneas, por lo que la mejor opción es usar *VRML* en lugar de *Java* para este tipo de reconstrucciones.

Sistemas como esté pronto serán necesarios en las cadenas productivas, que requieran un alto rendimiento y un buen control de calidad, y de esta forma elevar la productividad, para poder competir en los nuevos ordenes económicos mundiales.

Apéndice C
Transformaciones Geométricas

La presentación de objetos en la computadora requiere de la manipulación matemática de cada uno de los puntos (ó píxeles) que componen un objeto en la pantalla. Está representación enriquece y facilita el trabajo de los usuarios.

Cada objeto es representado internamente como un conjunto de puntos que guardan alguna relación entre sí, para facilitar su manejo y las operaciones entre ellos. En los **Capítulos: 2. Ambientes Virtuales y VRML, 5. Diseño e Implementación de un Sistema de Visión** y **6. Tareas del Sistema de Visión,** se emplean operaciones con los objetos reconocidos o modelados para su representación en forma gráfica en dos o tres dimensiones.

En esté apéndice se presentamos las transformaciones geométricas bidimensionales y tridimensionales, necesarias para representar información en forma gráfica en la pantalla de la computadora. Las transformaciones de traslación, escala y rotación son fundamentales para trabajar con operaciones gráficas en espacios de dos y tres dimensiones.

c.1 Transformaciones geométricas bidimensionales

Para un punto (ó píxel en el caso de imágenes) en el plano *P(x,y)* podemos realizar las operaciones siguientes en el espacio bidimensional [Foley 96]:

c.1.1 Traslación

Para *trasladar* un objeto, se añaden los valores de las coordenadas de traslación, a cada uno de los puntos que lo conforman (**Ecs. c.1** y **c.2**).

$$x' = x + d_x \qquad \text{...(Ec. c.1)}$$
$$y' = y + d_y \qquad \text{...(Ec. c.2)}$$

Donde:
- *x, y* = Punto de entrada
- *dx* = Desplazamiento en *x*
- *dy* = Desplazamiento en *y*

c.1.2 Escala

La *escala* permite cambiar el tamaño de los objetos. El escalamiento se lleva a cabo respecto al origen. Es *diferencial* si el escalamiento es distinto para ambos *ejes (x,y)* y *uniforme* si es el mismo (**Ecs. c.3** y **c.4**).

$$x' = s_x * x \qquad \text{...(Ec. c.3)}$$
$$y' = s_y * y \qquad \text{...(Ec. c.4)}$$

Donde:
- *x, y* = Punto de entrada
- s_x = Escala en *x*
- s_y = Escala en *y*

c.1.3 Rotación

La *rotación* puede hacer que un objeto gire en un *ángulo* α (**Ecs. c.5** y **c.6**).

$$y' = x * \cos\alpha - y * \sin\alpha \qquad \text{...(Ec. c.5)}$$
$$y' = x * \sin\alpha + y * \cos\alpha \qquad \text{...(Ec. c.6)}$$

Donde:
- *x, y* = Punto de entrada
- α = Angulo de rotación

c.1.4 Sesgo

El *sesgo* es de dos tipos: sesgo sobre *el eje y* y sesgo sobre el *eje x*. En el *sesgo* las líneas paralelas permanecen paralelas (**Ecs. c.7**, **c.8**, **c.9** y **c.10**).

Sesgo sobre el *eje X*

$$x' = x + a * y \quad \text{...(Ec. c.7)}$$
$$y' = y \quad \text{...(Ec. c.8)}$$

Sesgo sobre el *eje Y*

$$x' = x \quad \text{...(Ec. c.9)}$$
$$y' = b * x + y \quad \text{...(Ec. c.10)}$$

Donde: *x, y* = Punto de entrada
a, b = Constantes de proporcionalidad

c.1.5 Representación de objetos bidimensionales

Para poder realizar las operaciones con mayor facilidad, se representa cada punto por sus coordenadas *(x,y)* y las rectas que los unen, numerando los puntos y uniéndolos mediante sus índices como se muestra en la **Figura c.1**.

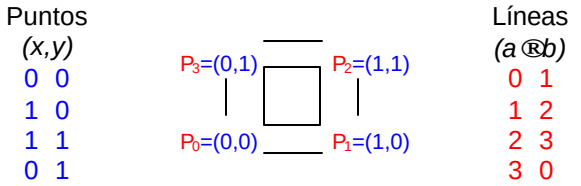
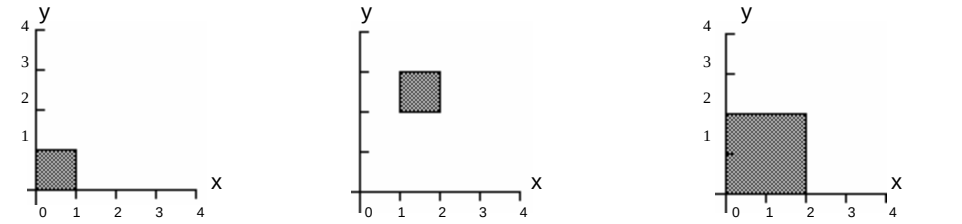


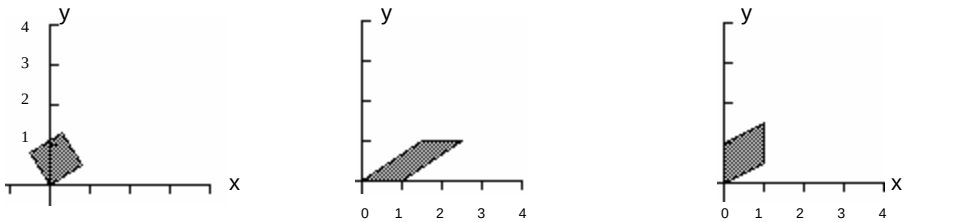
Figura c.1: Representación de objetos bidimensionales.

c.1.6 Operaciones con transformaciones bidimensionales

En la **Figura c.2** se observan los resultados de aplicar las transformaciones bidimensionales a un cuadrado.



a) Objeto original **b)** Traslación con d_x=1 y d_y=2 **c)** Escala con s_x=2 y s_y=2



d) Rotación de α=30º **e)** Sesgo en el eje X de 1.5 **f)** Sesgo en el eje Y de 0.5

Figura c.2: Resultados de las transformaciones bidimensionales en un cuadrado.

c.2 Transformaciones geométricas tridimensionales

Para un punto (ó píxel en el caso de imágenes) en el espacio *P(x,y,z)* podemos realizar las operaciones siguientes en el espacio tridimensional [Foley 96]:

c.2.1 Traslación

La *traslación* se usa para mover a un objeto en el espacio. Es necesario añadir las coordenadas en cada uno de los tres ejes (**Ecs. c.11**, **c.12** y **c.13**).

$$x' = x + d_x \quad \text{...(Ec. c.11)}$$
$$y' = y + d_y \quad \text{...(Ec. c.12)}$$
$$z' = z + d_z \quad \text{...(Ec. c.13)}$$

Donde: x, y, z = Punto de entrada
d_x = Desplazamiento en x
d_y = Desplazamiento en y
d_z = Desplazamiento en z

c.2.2 Escala

La *escala* cambia el tamaño en alguna de las dimensiones de un objeto en el espacio. Es necesario aplicar la *escala* a cada una de las coordenadas de sus puntos (**Ecs. c.14**, **c.15** y **c.16**).

$$x' = s_x * x \quad \text{...(Ec. c.14)}$$
$$y' = s_y * y \quad \text{...(Ec. c.15)}$$
$$z' = s_z * z \quad \text{...(Ec. c.16)}$$

Donde: x, y, z = Punto de entrada
s_x = Escala en x
s_y = Escala en y
s_z = Escala en z

c.2.3 Rotación

En el espacio tridimensional, existen tres *rotaciones* una por cada eje (**Ecs. c.17**, **c.18**, **c.19**, **c.20**, **c.21**, **c.22**, **c.23**, **c.24** y **c.25**).

Sobre el *eje X*

$$x' = =x \quad \text{...(Ec. c.17)}$$
$$y' = y * \cos \alpha_x - z * \sin \alpha_x \quad \text{...(Ec. c.18)}$$
$$z' = y * \sin \alpha_x + z * \cos \alpha_x \quad \text{...(Ec. c.19)}$$

Donde: x, y, z = Punto de entrada
α_x = Angulo de rotación

Sobre el *eje Y*

$$x' = x * \cos \alpha_y + z * \sin \alpha_y \quad \text{...(Ec. c.20)}$$
$$y' = y \quad \text{...(Ec. c.21)}$$
$$z' = - x * \sin \alpha_y + z * \cos \alpha_y \quad \text{...(Ec. c.22)}$$

Donde: x, y, z = Punto de entrada
α_y = Angulo de rotación

Sobre el *eje Z*

$x' = x * \cos a_z - y * \sin a_z$ ...(Ec. c.23)
$y' = x * \sin a_z + y * \cos a_z$ ...(Ec. c.24)
$z' = z$ ...(Ec. c.25)

Donde: x, y, z = Punto de entrada
a_z = Angulo de rotación

c.2.4 Sesgo

En el espacio tridimensional, existen tres tipos de *sesgo*, uno por cada plano (**Ecs. c.26**, **c.27**, **c.28**, **c.29**, **c.30**, **c.31**, **c.32**, **c.33** y **c.34**).

Sobre el *plano XY*

$x' = x$ ...(Ec. c.26)
$y' = y + a_y * z$ ...(Ec. c.27)
$z' = a_x * x + z$ ...(Ec. c.28)

Donde: x, y, z = Punto de entrada
a_x = Constante de proporcionalidad en x
a_y = Constante de proporcionalidad en y

Sobre el *plano XZ*

$x' = x + a_x * z$ ...(Ec. c.29)
$y' = y$ ...(Ec. c.30)
$z' = a_z * x + z$ ...(Ec. c.31)

Donde: x, y, z = Punto de entrada
a_x = Constante de proporcionalidad en x
a_z = Constante de proporcionalidad en z

Sobre el *plano YZ*

$x' = x + a_z * z$ ...(Ec. c.32)
$y' = y + a_y * z$ ...(Ec. c.33)
$z' = z$ ...(Ec. c.34)

Donde: x, y, z = Punto de entrada
a_y = Constante de proporcionalidad en y
a_z = Constante de proporcionalidad en z

c.2.5 Proyecciones paralelas

Las *proyecciones paralelas*, permiten modelar la profundidad de los objetos, para que su apariencia sea más realista.

Sobre el *eje Z = 0*. La *proyección* se realiza a un punto sobre el eje *Z*, por lo que el objeto sufre una deformación que tiende a juntar sus líneas en el punto $z = ¥$ (**Ecs. c.35**, **c.36**, **c.37**, **c.34**, **c.39**, **c.40**, **c.41**, **c.42** y **c.43**).

$x' = x *(d + x)/ z$ Para $z ¹ 0$ ...(Ec. c.35)
$y' = y *(d + y)/ z$ Para $z ¹ 0$ ...(Ec. c.36)
$z' = z$ Para $z ¹ 0$ ...(Ec. c.37)
$x' = x$ Para $z = 0$ ...(Ec. c.38)

$y' = y$ Para $z=0$ **...(Ec. c.39)**
$z' = z$ Para $z=0$ **...(Ec. c.40)**

Donde: x, y, z = Punto de entrada
d = Distancia del origen al *Punto de Vista*

Sobre el *eje Z en infinito*.

$x' = x * (d + |x|)/(z + d)$ **...(Ec. c.41)**
$y' = y * (d + |y|)/(z + d)$ **...(Ec. c.42)**
$z' = z$ **...(Ec. c.43)**

Donde: x, y, z = Punto de entrada
d = Distancia del origen al Punto de vista

c.2.6 Proyecciones ortográficas

Las *proyecciones ortográficas* nos permiten visualizar las características del objeto visto desde cada uno de los ejes de coordenadas, es decir, es posible ver el frente, el lado y la parte de arriba de los objetos. Consiste en hacer cero las coordenadas del eje correspondiente y rotar los puntos de tal forma que la vista quede en el *plano XY*. Se tienen tres vistas ortográficas (**Ecs. c.44**, **c.45**, **46**, **c.47**, **c.48**, **c.49**, **c.50**, **c.51** y **c.52**).

Sobre el *eje x = 0*

$x' = z * sin(90°)$ **...(Ec. c.44)**
$y' = y$ **...(Ec. c.45)**
$z' = 0$ **...(Ec. c.46)**

Donde: x, y, z = Punto de entrada

Sobre el *eje y = 0*

$x' = x$ **...(Ec. c.47)**
$y' = - z * sin(-90)$ **...(Ec. c.48)**
$z' = 0$ **...(Ec. c.49)**

Donde: x, y, z = Punto de entrada

Sobre el *eje z = 0*

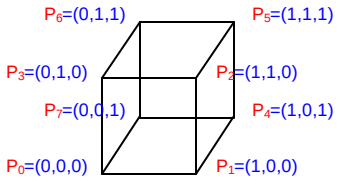
$x' = x$ **...(Ec. c.50)**
$y' = y$ **...(Ec. c.51)**
$z' = 0$ **...(Ec. c.52)**

Donde: x, y, z = Punto de entrada

c.2.7 Representación de objetos tridimensionales

Para poder realizar las operaciones de transformación en el espacio tridimensional, es necesario representar cada punto por sus coordenadas *(x,y,z)* y las rectas que los unen numerando los puntos y uniéndolos mediante sus índices como se muestra en la **Figura c.3**.

Puntos
(x,y,z)
0 0 0
1 0 0
1 1 0
0 1 0
1 0 1
1 1 1
0 1 1
0 0 1

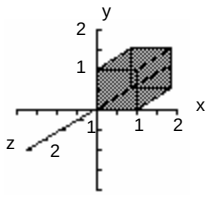


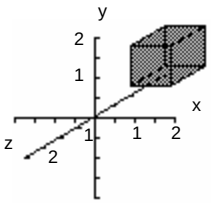
Líneas
(a b)
0 1
1 2
2 3
3 0
4 5
5 6
6 7
7 4
1 4
2 5
3 6
0 7

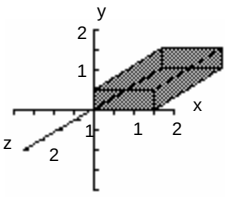
Figura c.3: Representación de objetos tridimensionales.

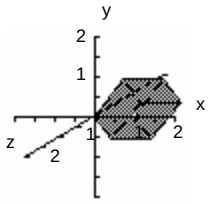
c.2.8 Operaciones con transformaciones tridimensionales

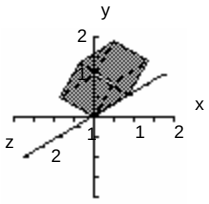
En la **Figura c.4** se observan los resultados de aplicar las transformaciones tridimensionales a un cubo.

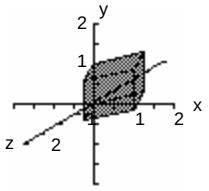

a) Objeto Original

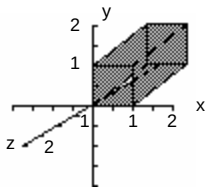

b) Traslación con d_x=2, d_y=2 y d_z=2

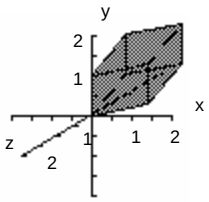

c) Escala con s_x=1.5, s_y=0.5 y s_z=2

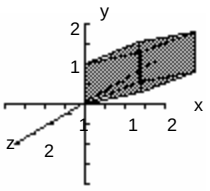

d) Rotación de α_x=60º

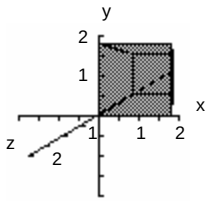

e) Rotación de α_z=60º

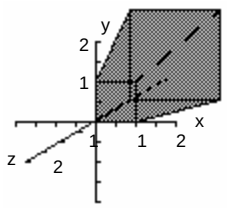

f) Rotación de α_y=60º


g) Sesgo en el plano YZ (0.5,0.5)


h) Sesgo en el plano XY (0.5,0.5)


i) Sesgo en el plano XZ (0.5,0.5)


j) Proyección paralela en z = ∞


k) Proyección paralela en z = 25

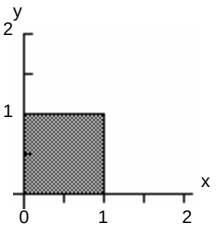

l) Vista Ortogonal z=0

Figura c.4: Resultados de las transformaciones tridimensionales en un cubo.

También es posible encontrar las vistas ortográficas de objetos complejos como se muestra en las **Figuras c.5, c.6 c.7** y **c.8.**

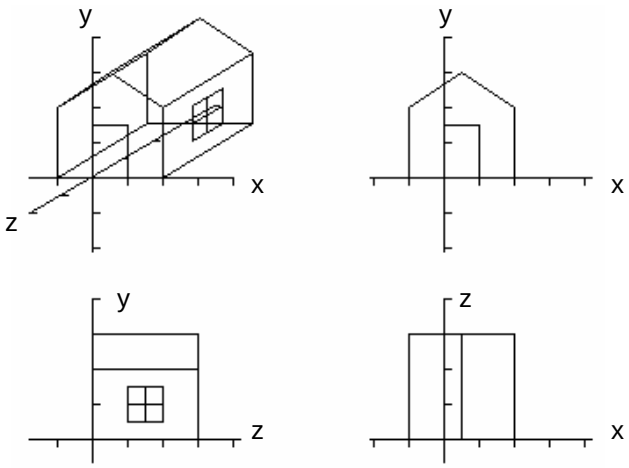


Figura c.5: Vistas de una proyección paralela de una casa.

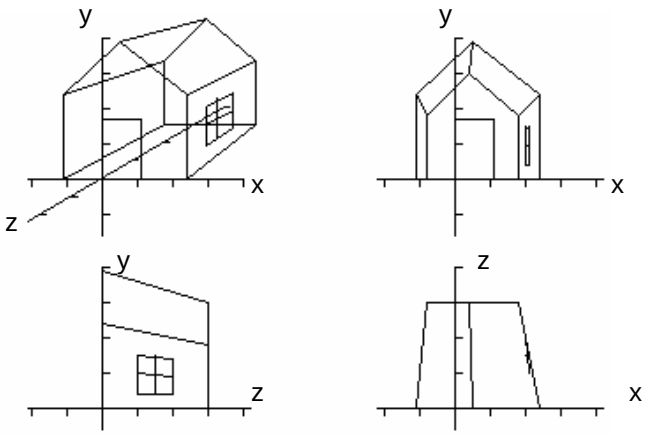


Figura c.6: Vistas de una proyección en perspectiva de una casa.

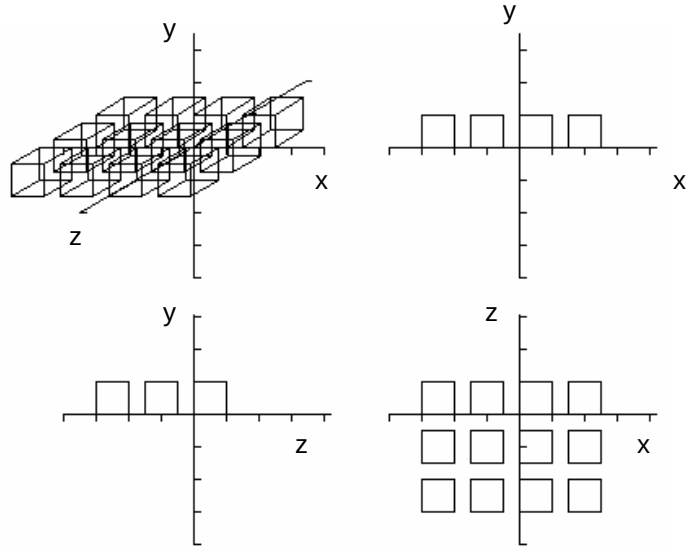


Figura c.7: Vistas de una proyección paralela de un conjunto de cubos.

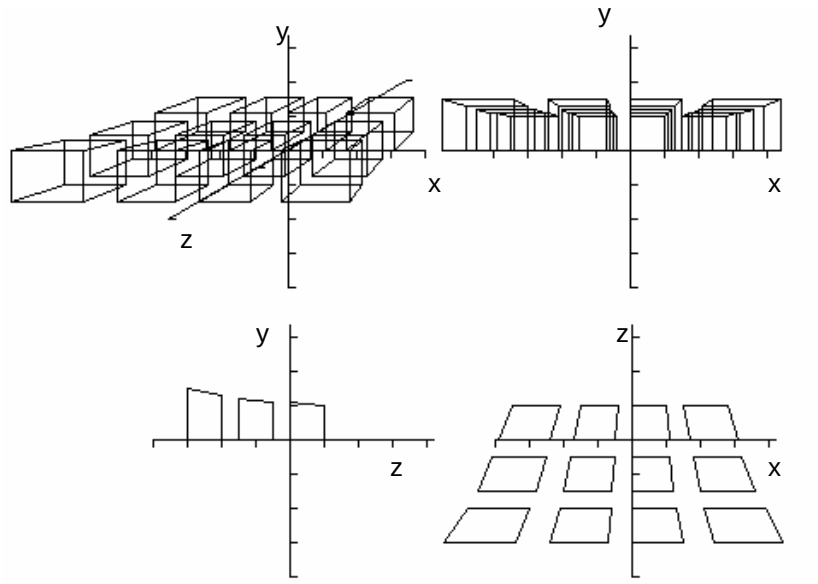


Figura c.8: Vistas de una proyección en perspectiva de un conjunto de cubos.

Apéndice D
Patrones de Calibración y Figuras Geométricas

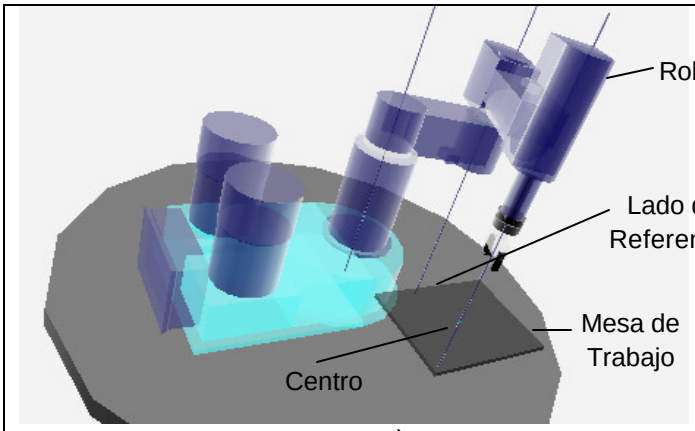
Dentro del desarrollo de un sistema de visión se deben de contemplar las formas de calibrarlo, es decir, que sea posible conocer las coordenadas espaciales de un objeto en base a las coordenadas de una imagen. También se debe de considerar que tipo y cantidad de figuras será capaz de reconocer, pues de esto depende el número de momentos necesario para llevar a cabo el reconocimiento y reconstrucción tridimensional.

En este apéndice presentamos la forma en que se lleva a cabo la calibración de cámara descrita en el **Capítulo 3: Visión Artificial** y proporcionamos los patrones de *calibración de cámara* empleados en los **Capítulos: 5. Diseño e Implementación de un Sistema de Visión** y **6. Tareas del Sistema de Visión**. También damos las *figuras geométricas* empleadas y los dígitos utilizados para realizar las pruebas de reconocimiento de patrones mediante el cálculo de momentos de *Hu* del **Capítulo 6: Tareas del Sistema de Visión**.

d.1 Patrones de calibración de cámara

La *calibración de cámara* determina la relación entre las coordenadas del mundo real con las coordenadas de la imagen, para poder determinar la posición real de los objetos. Para llevarla a cabo se utilizan algunos parámetros de la cámara, como son: la distancia focal, la distorsión de la lente, la inclinación de la cámara respecto a la mesa de trabajo del robot, etc. Para la obtención de estos parámetros se emplean los patrones de calibración de cámara. Los patrones de calibración de cámara, pueden ser figuras bien definidas, en nuestro caso están compuestas por un conjunto de *puntos circulares de 8mm* de diámetro, situados en las esquinas de un cuadrado (tamaño 10x10cm).

El objetivo de usar patrones con marcas (círculos en nuestro caso) es que en la imagen obtenida a partir de una fotografía, se encuentre bien definido el centro de cada una de ellas. Si por alguna causa no es posible calcular de forma precisa el centro de las marcas, estas pueden hacerse de un diámetro menor o incluso utilizar otras figuras como cuadrados o cruces. Su empleo para nuestra aplicación se muestra en la **Figura d.1**.



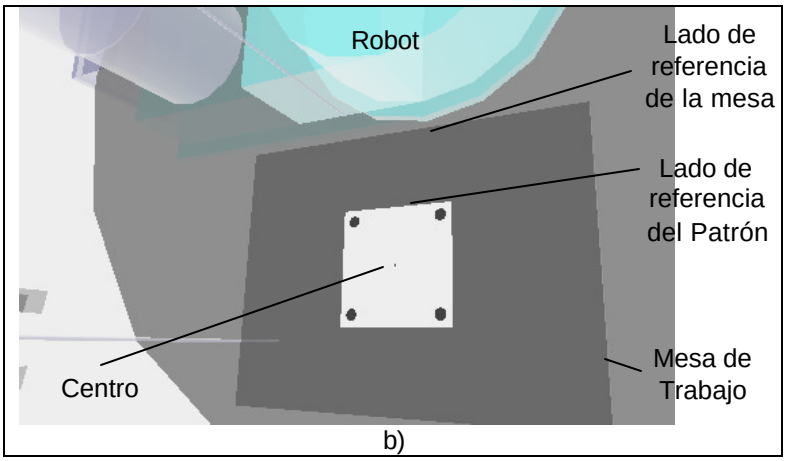
a)

1) El robot marca el centro de su mesa de trabajo (**a**).

2) Considerar un lado de la mesa de trabajo como referencia.

3) Colocar el primer patrón de calibración (0º) haciendo coincidir su centro con el centro marcado por el robot y el lado de referencia del patrón con el lado de referencia de la mesa de trabajo (**b**).

Figura d.1: Uso de los patrones de calibración de cámara (`Parte 1/2`).



4) Tomar la fotografía con el patrón de 0º y sustituirlos por los demás patrones de 15º, 30º, 45º, 60º y 75º, tomando en cada uno la imagen correspondiente.

Figura d.1: Uso de los patrones de calibración de cámara (`Parte` **`2`**`/2`).

Si bien es posible realizar la calibración de cámara con el empleo de un solo patrón, es mejor usar varios patrones debido a las deformaciones geométricas que los sistemas de captación de imágenes puedan agregar a la escena. En nuestro caso decidimos usar seis patrones de calibración, cada uno con cuatro puntos circulares como marcas de calibración. Cada uno de estos patrones difiere del anterior en la posición que guardan los puntos respecto al eje horizontal del sistema de visión. Así tenemos los patrones calibrados en 0º, 15º, 30º, 45º, 60º y 75º. Los patrones de calibración de cámara se muestran en las **Figuras d.2**, **d.3**, **d.4**, **d.5**, **d.6** y **d.7**.

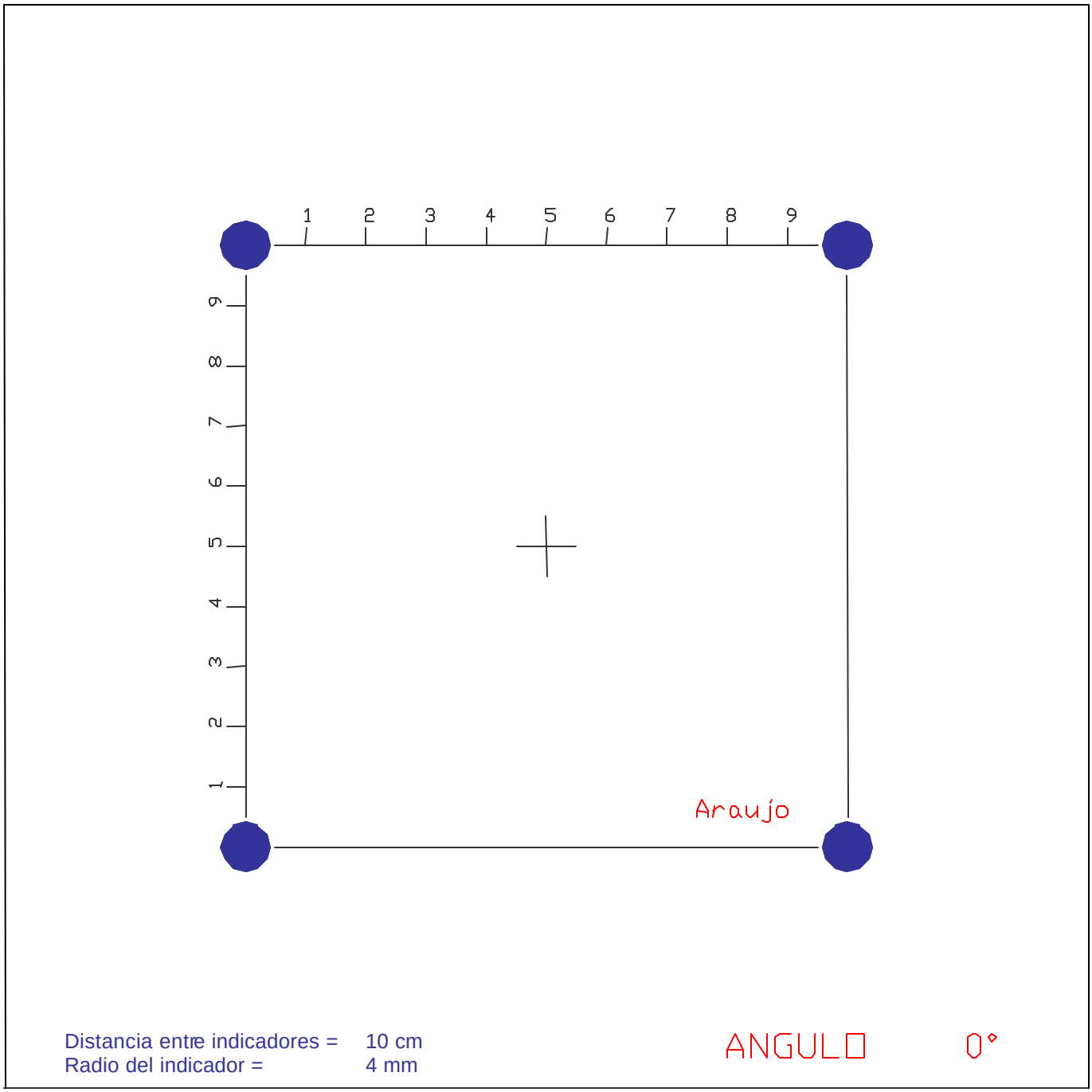


Figura d.2: Patrón de calibración de cámara a 0º.

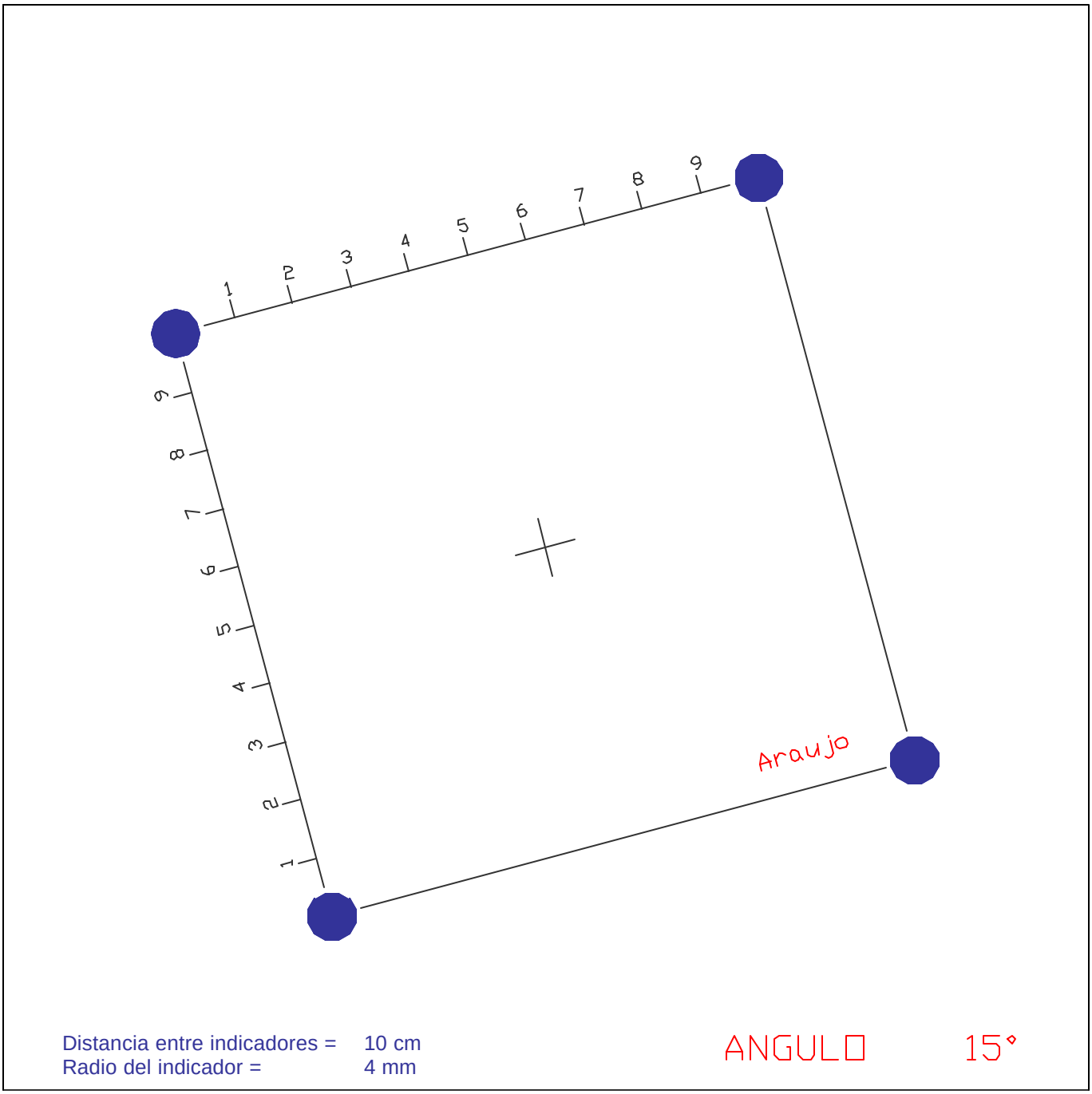


Figura d.3: Patrón de calibración de cámara a 15º.

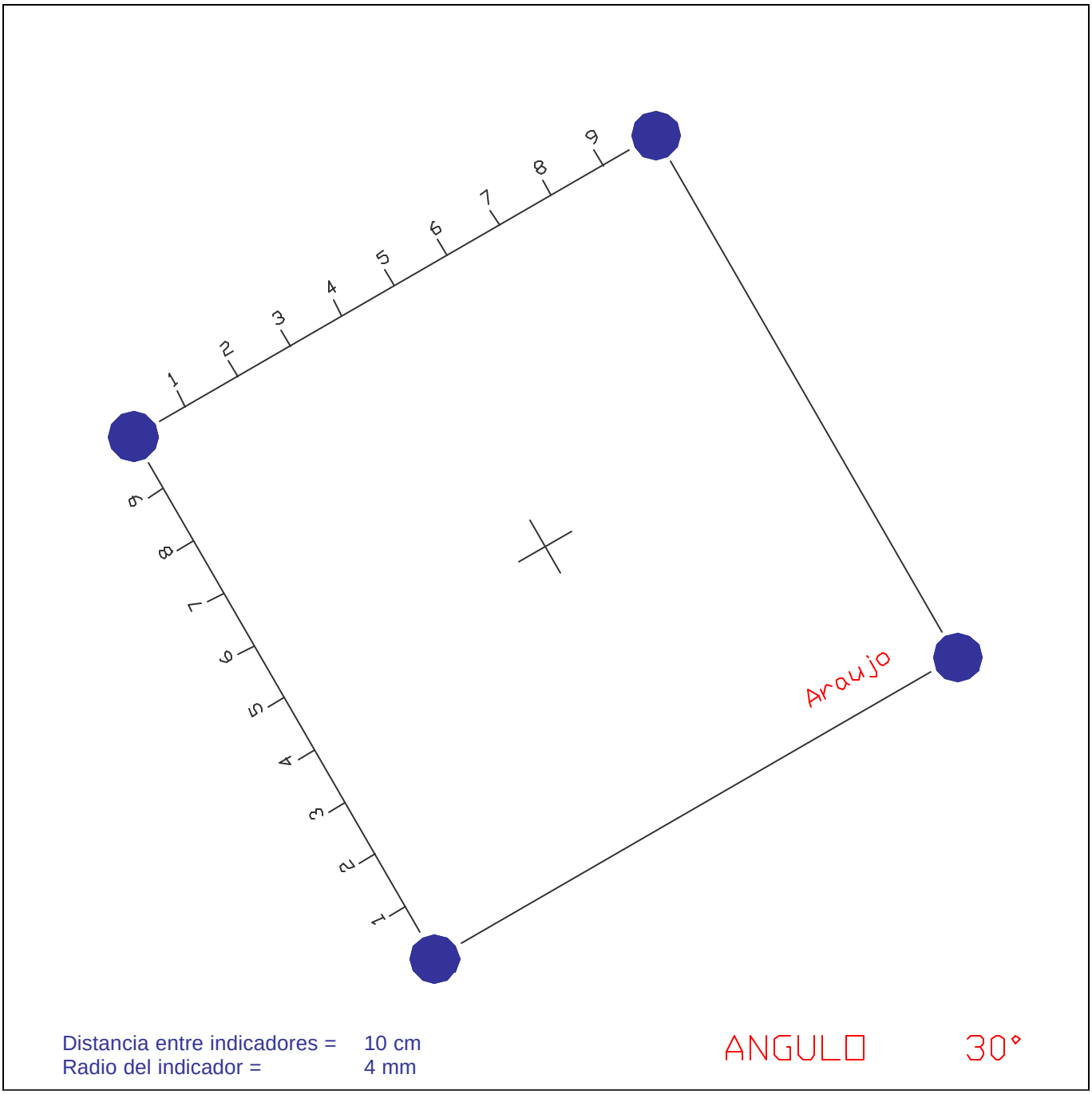


Figura d.4: Patrón de calibración de cámara a 30º.

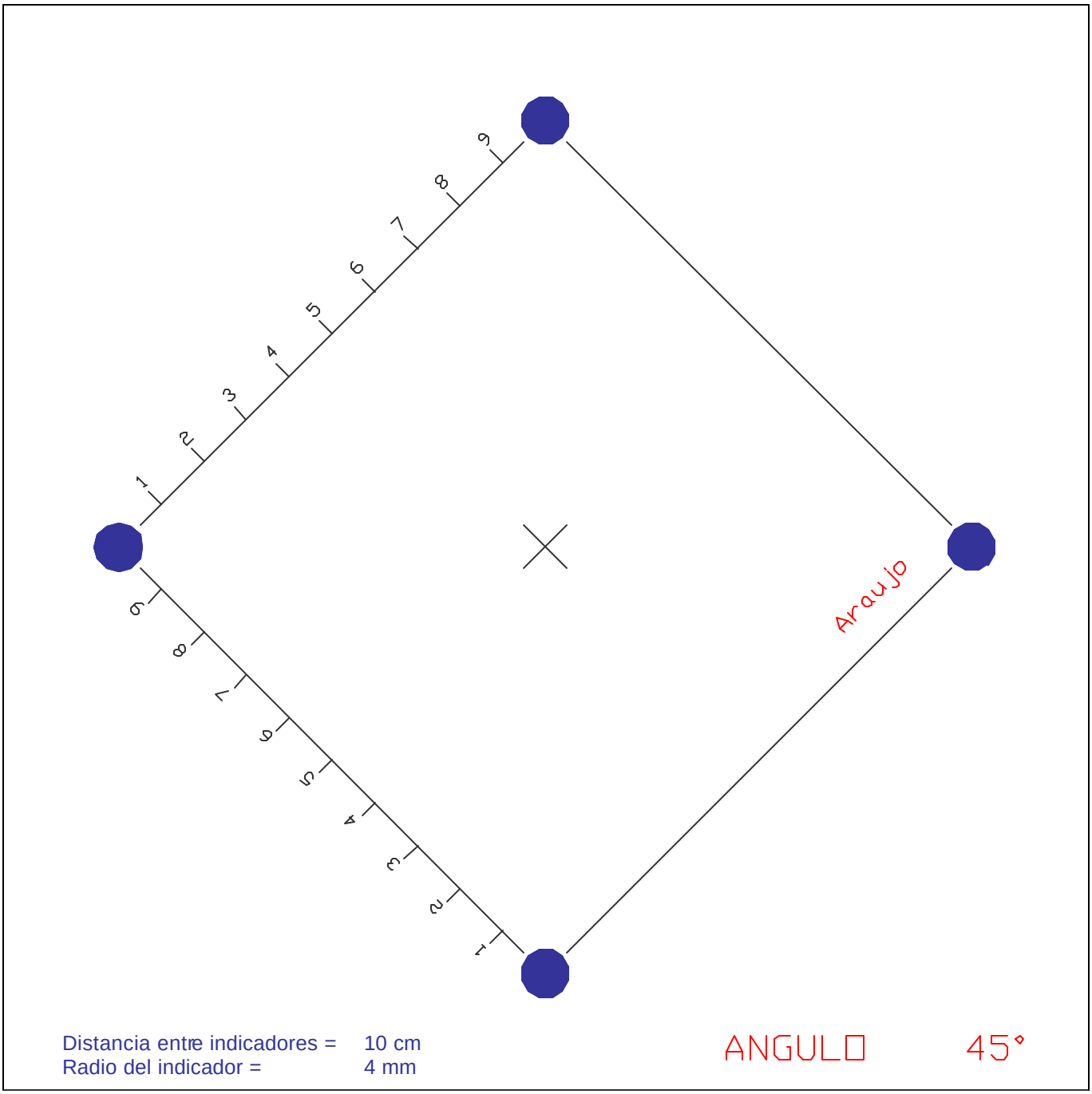


Figura d.5: Patrón de calibración de cámara a 45º.

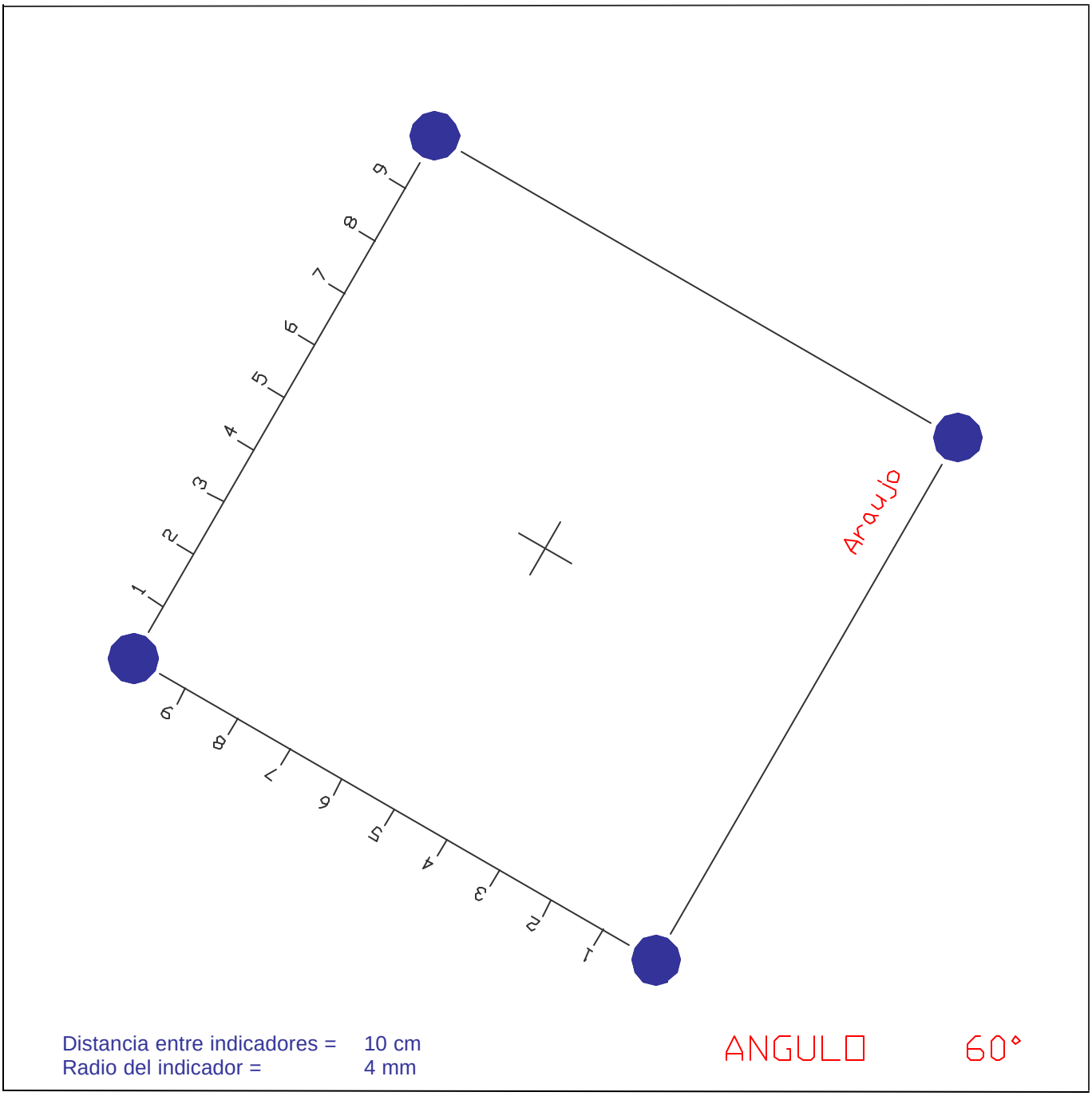


Figura d.6: Patrón de calibración de cámara a 60º.

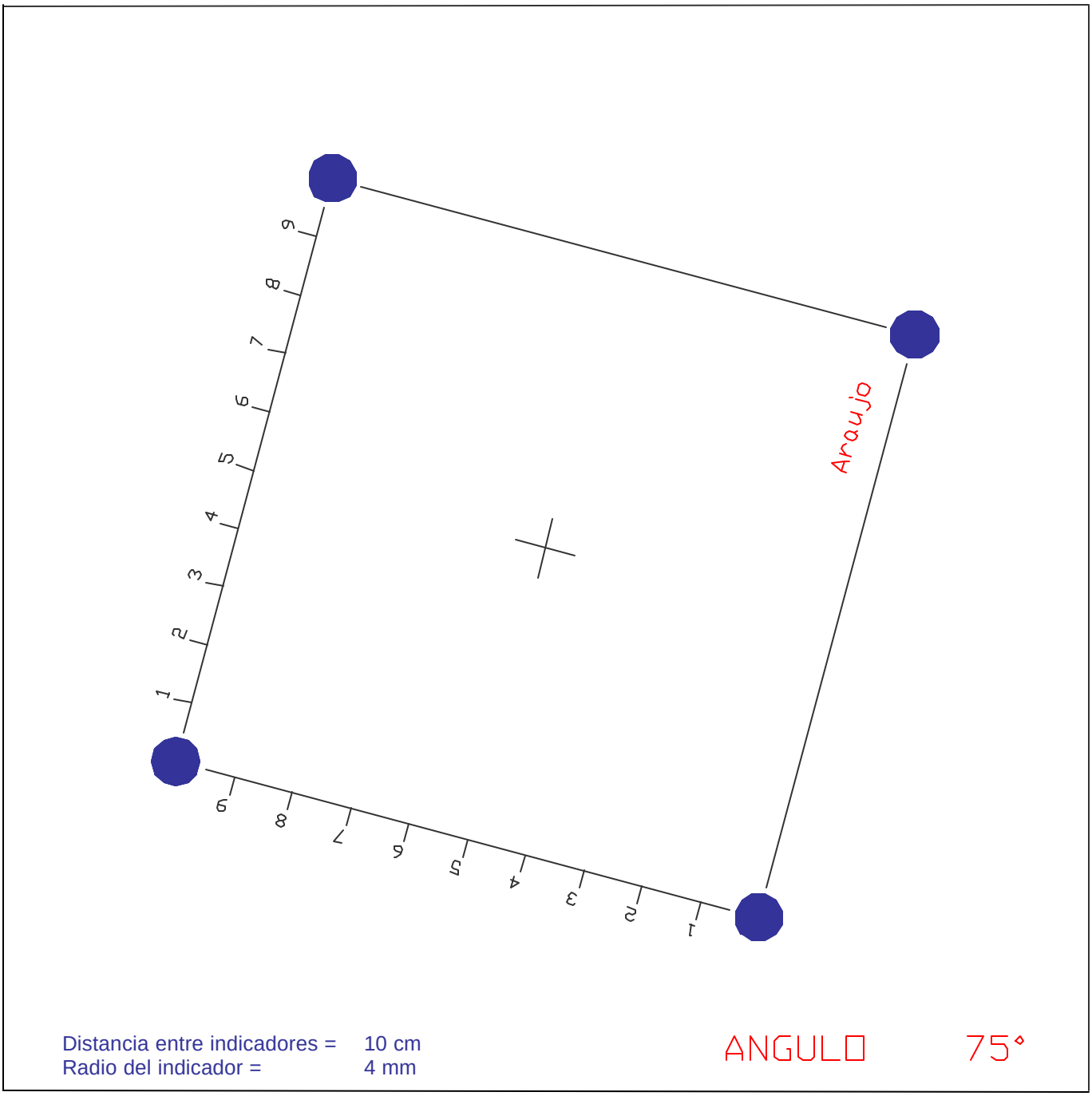
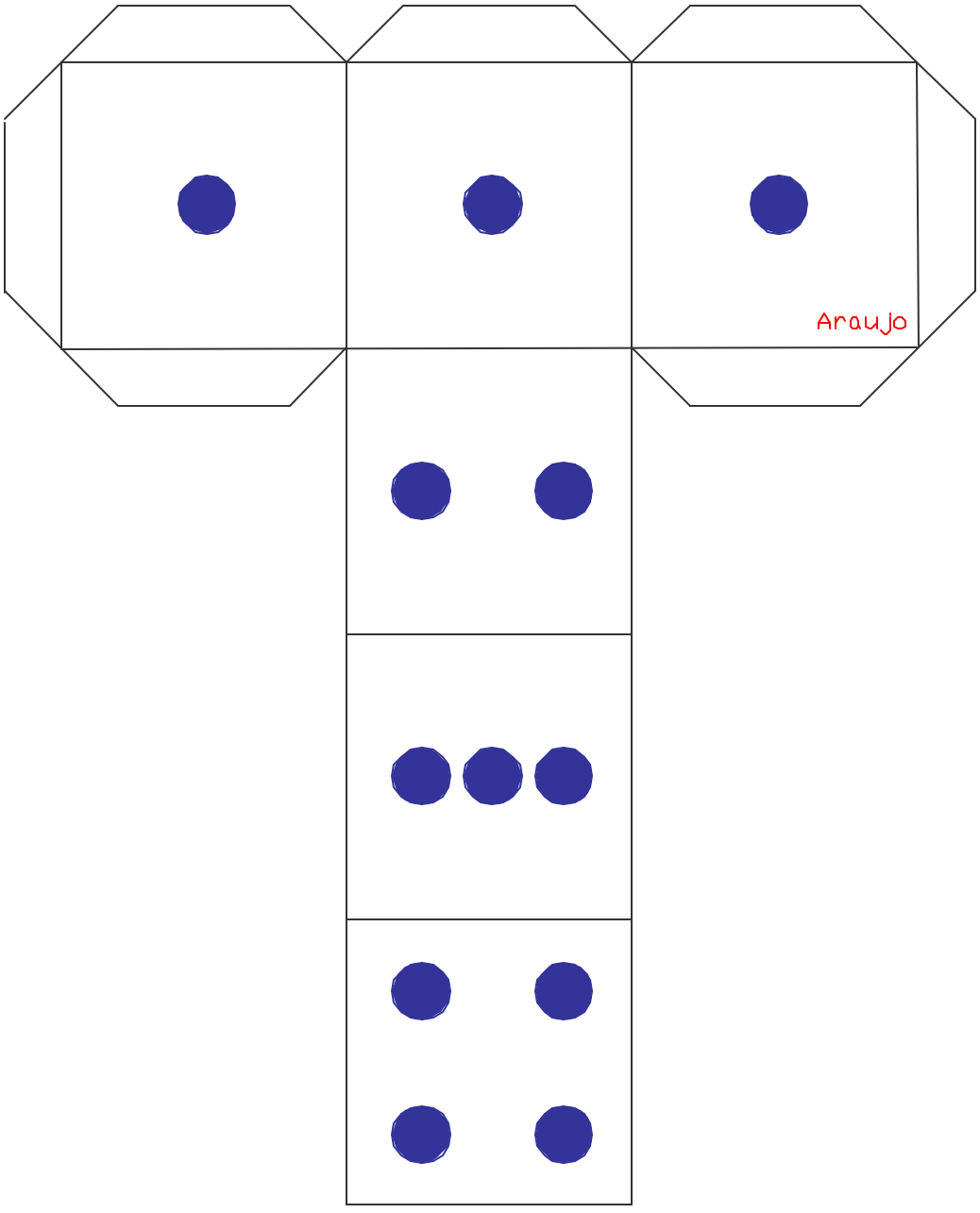


Figura d.7: Patrón de calibración de cámara a 75º.

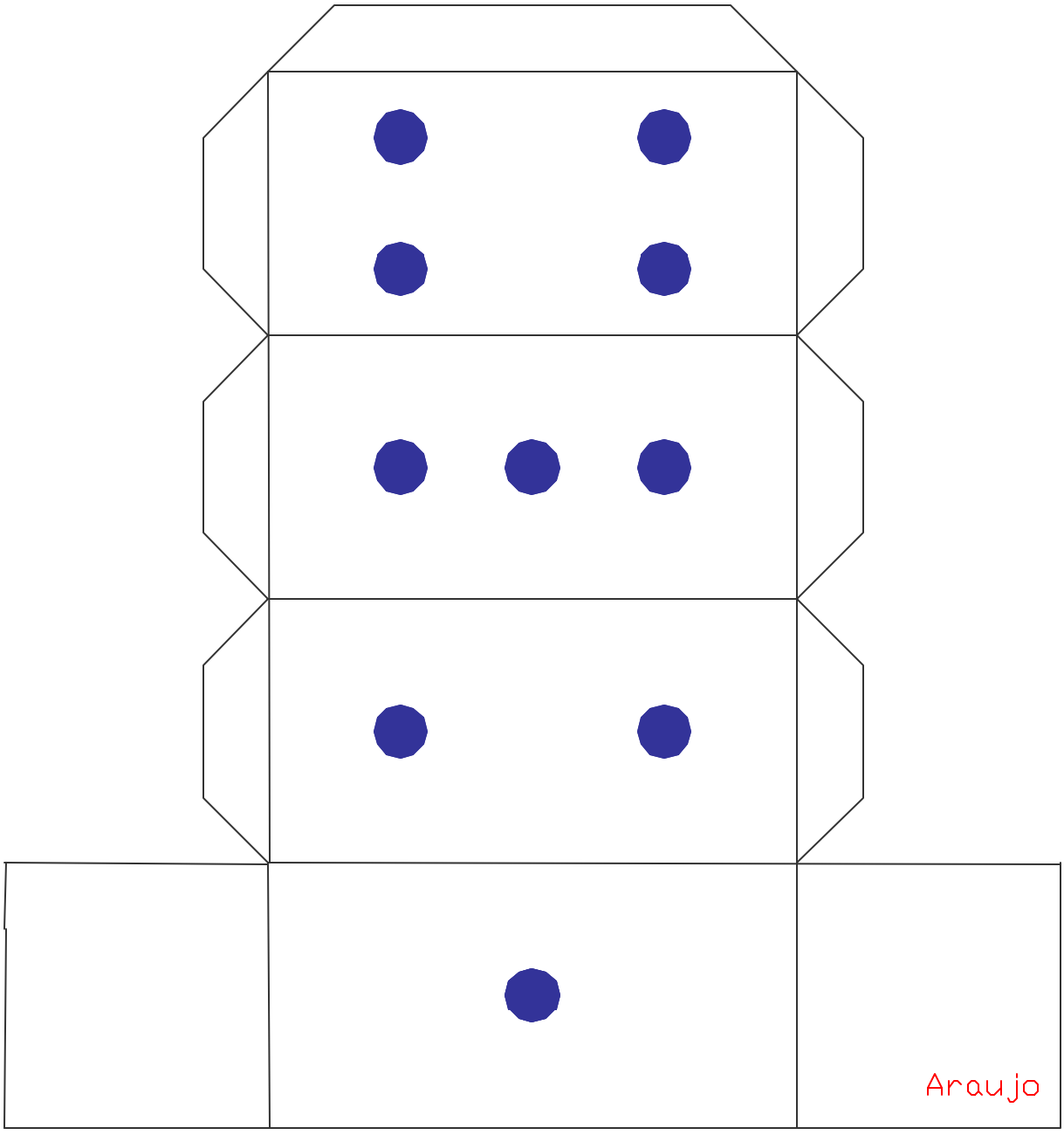
d.2 Figuras usadas para pruebas de reconocimiento

Una vez realizada la *calibración de cámara*, es necesario encontrar los parámetros de los objetos que nuestro sistema será capaz de reconocer. En las **Figuras d.8**, **d.9**, **d.10**, **d.11** y **d.12** se muestran las *figuras geométricas* usadas para las pruebas de reconocimiento. La **Figura d.13** muestra los *dígitos* usados para reconocimiento de dígitos.



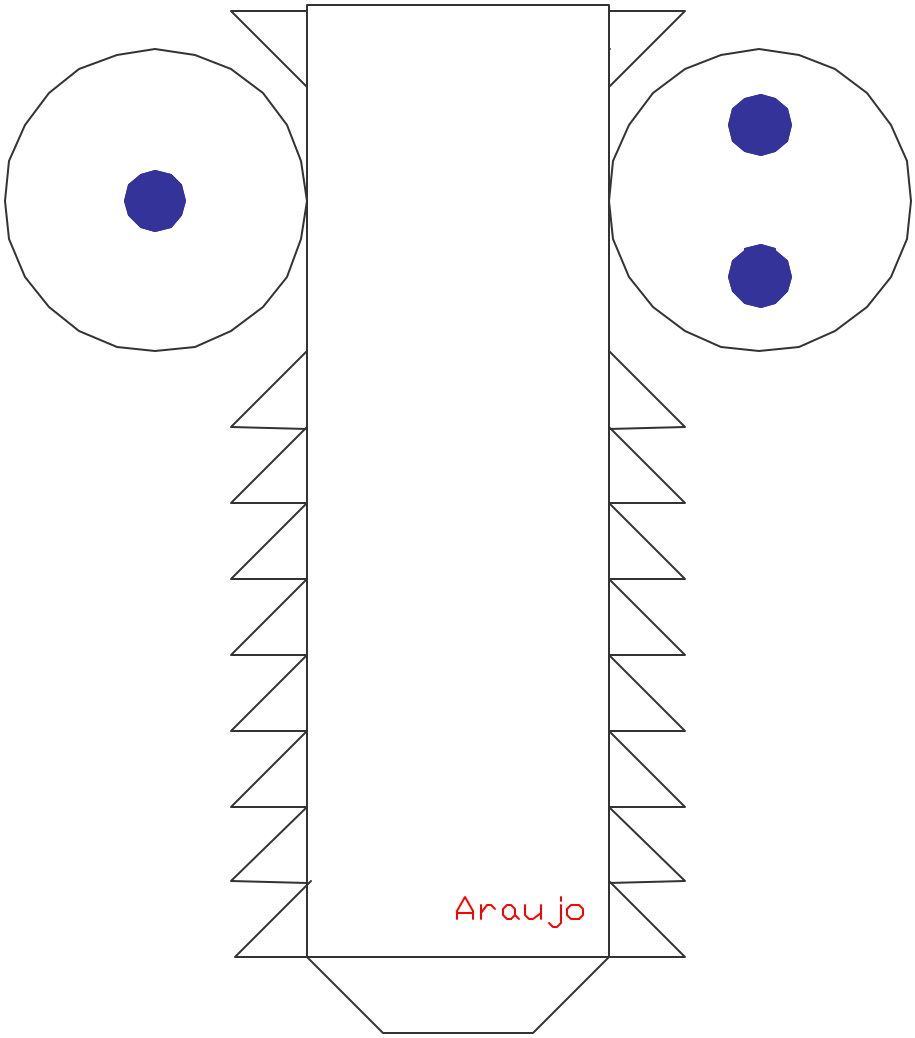
Lado del cubo = 4 cm
Radio del indicador = 4 mm

Figura d.8: Cubo usado para pruebas de reconocimiento.



Ancho del rectángulo = 8 cm
Alto del rectángulo = 4 cm
Radio del indicador = 4 mm

Figura d.9: Prisma rectangular usado para pruebas de reconocimiento.



Radio del cilindro = 2 cm
Alto del cilindro = 4 cm
Radio del indicador = 4 mm

Figura d.10: Cilindro usado para pruebas de reconocimiento.

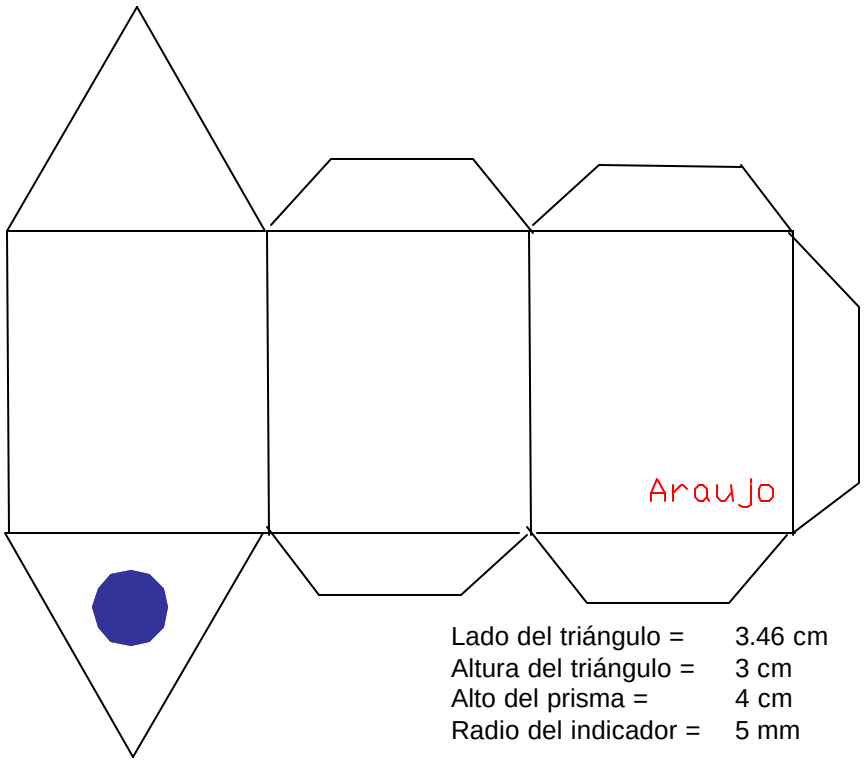


Figura d.11: Prisma triangular usado para pruebas de reconocimiento.

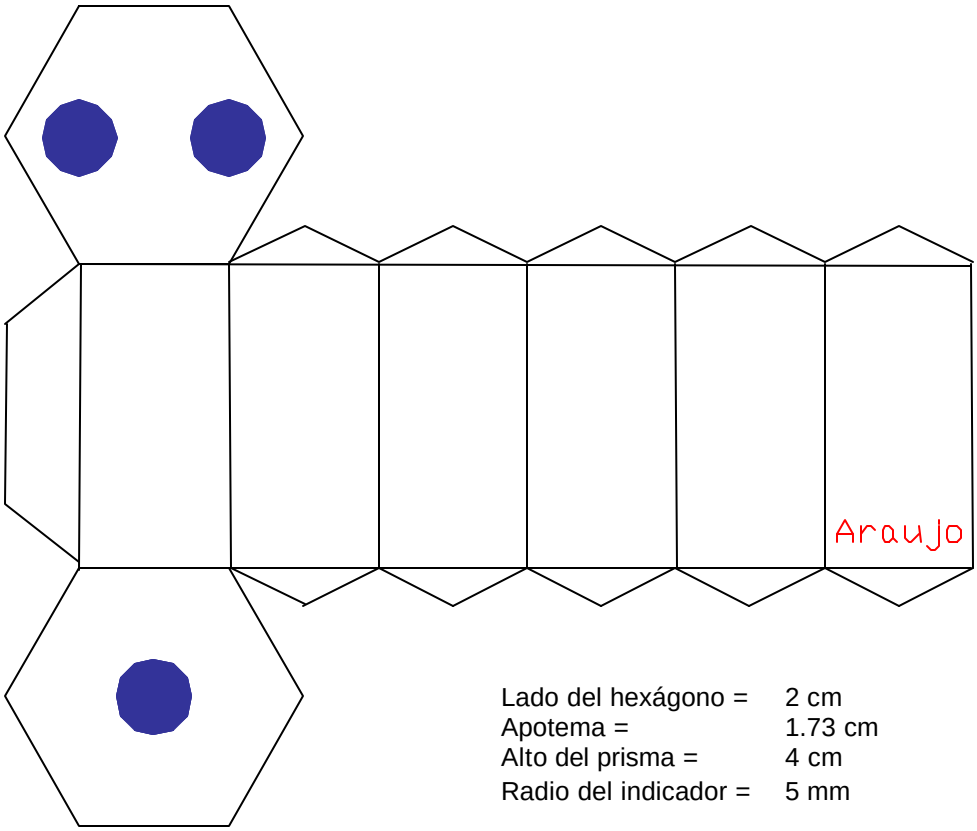
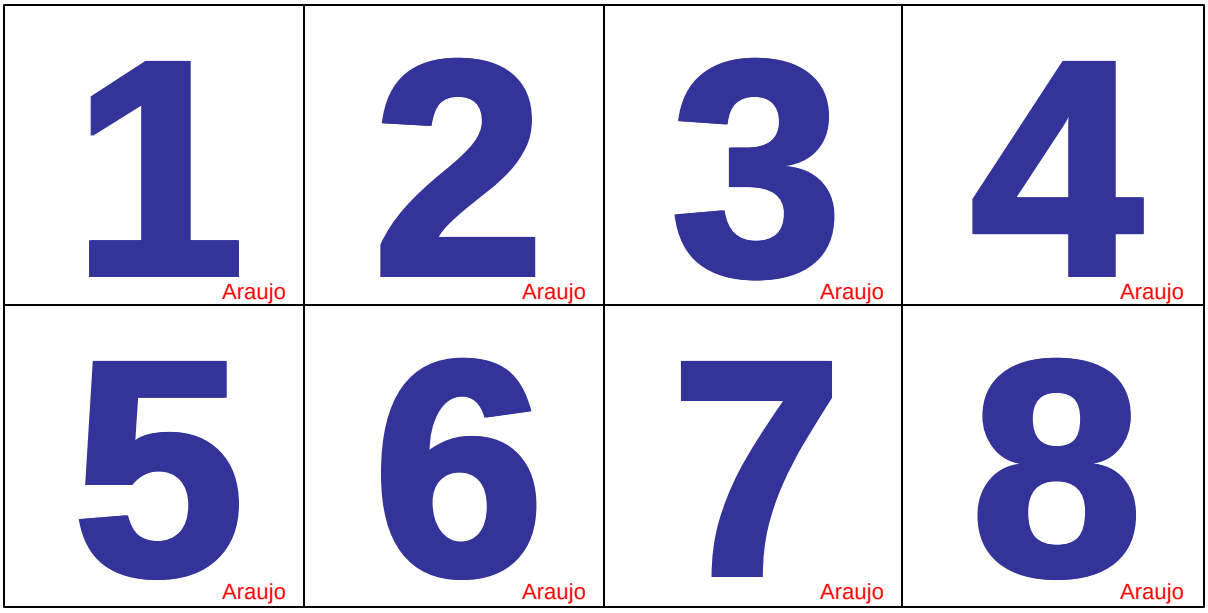


Figura d.12: Prisma hexagonal usado para pruebas de reconocimiento.



Tipo de fuente: Albertus
Tamaño de fuente: 115

Figura d.13: Dígitos empleados para pruebas de reconocimiento.

Apéndice E
Contenido del CD-ROM

El CD-ROM incluido tiene por objetivo dar a conocer las herramientas necesarias para llevar a cabo tareas tales como programación, presentación y distribución de programas, así como los resultados obtenidos durante el desarrollo de la presente tesis. Los programas comerciales incluidos se obtuvieron de Internet, por lo que son versiones de demostración, sin embargo la mayor parte de ellos son de distribución libre.

e.1 Archivos del CD-ROM

Los archivos contenidos en el directorio raíz del CD-ROM se emplean para realizar la presentación del menú, la función de cada uno es la siguiente:

`autorun.inf` Manejador de Windows de `AutoPlay`, para ejecutarse automáticamente.
`menu.apm` Configuración del menú.
`menu.exe` Es el menú ejecutable.

Al ejecutarse el menú, aparece como se observa en la **Figura e.1**.

a) Pantalla de presentación

b) Menú de opciones

c) Menú de programas fuente

d) Menú de programas para instalación

Figura e.1: Menús del CD-ROM

e.2 Directorios del CD-ROM y su contenido

El contenido de los directorios de CD-ROM se describe a continuación:

1. Acrobat.- contiene el programa `Acrobat Reader 5.0` (**Figura e.2**), que es un visualizador gratuito que permite navegar e imprimir archivos con formato **PDF** (*Portable Document Format*). La instalación de este programa es necesario para poder ver los documentos de tesis y programas fuente contenidos en el CD-ROM.

Figura e.2: Acrobat Reader 5.0

2. Cosmo.- contiene el programa de visualización de mundos virtuales `VRML 2.1` (**Figura e.3**), su distribución es gratuita. Permite visualizar los mundos virtuales de los ejemplos y los que se obtienen con los programas desarrollados, sin embargo, es indispensable tener instalador un navegador de Internet como `Internet Explorer`.

Figura e.3: VRML 2.1

3. DATA.- contiene los datos y archivos de apoyo (imágenes, sonidos y configuraciones) para el menú.

4. Ejecutables.-contiene los programas ejecutables en `DOS` (*Disk Operating System*) y `HTML-Java` (Ejecutar `indice.exe`). Los programas contenidos son los siguientes:

- **ANTENA**.- programa de prueba para generar patrones de radiación de antenas en 3D en VRML, permite grabar un archivo VRML dependiendo la longitud de onda de la antena
- **CAMARA**.- programa para calibración de cámara, según el procedimiento descrito en el **Capítulo 6. Tareas del Sistema de Visión.**
- **DIGITOS**.- programa de análisis y reconocimiento de dígitos, con el método descrito en el **Capítulo 6. Tareas del Sistema de Visión.**
- **INDICE**.- programa que muestra todos los ejemplos y programas contenidos en este directorio.
- **MOV01**.- programa para el análisis de movimiento de obajetos frente a la cámara, su descripción se encuentra en el **Capítulo 6. Tareas del Sistema de Visión.**
- **PIRÁMIDE**.- permite dibujar pirámides en el espacio 3D con VRML.
- **R3D**.- programa para la reconstrucción 3D a partir de las imágenes obtenidas con un sensor tridimensional a través de luz láser.
- **SOLAR**.- simula un sistema solar en el espacio 3D en VRML, con solo tener los datos en un archivo de texto.
- **VISION**.- programa de análisis y reconocimiento de objetos geométricos según el procedimiento descrito en **Capítulo 6. Tareas del Sistema de Visión.**

Los programas en Java se encuentran contenidos en los subdirectorios siguientes:

- **URL4.-** programa de reconocimiento de figuras geométricas, el cual emplea Java para la actualización del espacio virtual, se ejecuta presionando `..\url4\ejecutar.exe↵`
- **URL5.-** programa de reconocimiento de dígitos, el cual emplea Java para la actualización del espacio virtual, se ejecuta presionando `..\url5\ejecutar.exe↵`
- **ANTENAS.-** programa para generar patrones de radiación de antenas, el cual emplea Java para la actualización del espacio virtual, se ejecuta presionando `..\antenas\ejecutar.exe↵`

5. Explorer.- contiene una versión personalizada de Internet Explorer 5.0 (**Figura e.4**), de distribución gratuita. Es necesario para ejecutar la mayoría de las aplicaciones contenidas en este CD-ROM y para navegar en Internet.

Figura e.4: Internet Explorer 5.0

6. Fuentes.- contiene los códigos en lenguaje C, Java, HTML y VRML de los programas fuentes utilizados, en formato *PDF*.

- **DIGITO.-** programas empleados para el reconocimiento de dígitos, contiene el programa en lenguaje C, Java, HTML y VRML.
- **MOVIMIENTO.-** programas empleados para el análisis de movimiento, contiene el programa en lenguaje C, Java, HTML y VRML.
- **REC3D.-** programas empleados para la reconstrucción 3D, contiene el programa en lenguaje C, HTML y VRML.
- **THREE3D.-** programas empleados para la reconstrucción 3D, contiene el programa en Java y HTML.
- **UNIMATE.-** programa para la visualización del robot UNIMATE S-103, contiene el programa HTML y VRML.
- **VISION.-** programas empleados para el reconocimiento de figuras geométricas, contiene el programa en lenguaje C, Java, HTML y VRML.

7. Internet.- contiene las páginas de Internet con ejemplos, teoría y resultados obtenidos para el sistema de visión, programación de ambientes virtuales, Java y VRML.

8. Kawa.- contiene el programa Kawa Professional Edition 5.0 (**Figura e.5**), es un ambiente integral de desarrollo para aplicaciones en Java. Permite editar, probar y depurar programas en Java de forma sencilla.

Figura e.5: Kawa Professional Edition 5.0

9. Presentacion.- contiene una presentación en PowerPoint que describe el desarrollo y resultados de la tesis.

10. prog_dos.- programas especialmente diseñados para entender y/o demostrar algunos conceptos necesarios en visión artificial (ejecutar `indice.exe`).

11. Sdk13.- programa `Java 2 SDK`, Versión `1.3.0`. SDK es un ambiente de desarrollo para construir aplicaciones, applets y componentes que pueden desplegarse en la plataforma de `Java 2`.

Figura e.6: Java 2 SDK

12. TCC.- contiene el programa `Borland C++ 3.0` (**Figura e.7**), para desarrollar programas en lenguaje `C`. Permite editar, modificar, probar, ejecutar y depurar programas en lenguaje `C/C++`.

Figura e.7: Borland C++ 3.0

13. Tesis.- contiene los documentos de tesis en formato *PDF*. Tiene dos versiones de la tesis, una larga (`vision.pdf`), ideal para principiantes y una versión corta (`tesis.pdf`) dirigida a aquellos que poseen conocimientos en el área.

14. Winzip.- contiene el programa `WinZip 8.0` (**Figura e.8**). Es un excelente programa de compresión y descompresión de archivos.

Figura e.8: WinZip 8.0

Referencias

[Alarcón 00] De Alarcón Alvarez, E. y Parés I Burguès, N., **"Manual Práctico de VRML 2.0",** Ed. Biblioteca Técnica de Programación, España, 2000, 187pp.

[Alchemy 96] **"Image Alchemy 1.8",** Handmade Software, U.S.A., 1996, 226pp.

[Ayres 87] Ayres, Frank, **"Cálculo Diferencial e Integral",** Ed. McGraw-Hill, México, 1987, 345pp.

[Comer 97] Comer, Douglas E., **"Redes de computadoras. Internet e Interredes"**, Ed. Prentice-Hall, México, 1997, 506pp.

[Corke 96] Corke, Peter I.**, "Visual Control of Robots: high-performance visual servoing"**, Ed. John Wiley & Sons, Gran Bretaña, 1996, 353pp.

[Cormen 90] Cormen, Thomas H., Charles E. Leiserson y Ronald L. Rivest, **"Introduction to Algorithms",** Ed. MIT Press y McGraw-Hill, 1990, 1025pp.

[CVPR 97] Chenyang Xu y Jerry L. Prince, **"Gradient Vector Flow: A New External Force for Snakes"** , IEEE, 1997, 66-71pp.

[Chapra 96] Chapra, Steven C. y Raymond P. Canale, **"Métodos Numéricos para Ingenieros, con aplicaciones en computadoras personales",** Ed. McGraw-Hill, México, 1996, 641pp.

[Deitel 98] Deitel, H.M. y Deitel, P.J., **"Como programar en Java",** Ed. Prentice-Hall, México, 1998, 1056pp.

[Diccionario 90] **"Diccionario de informatica",** Ed. Oxford University Press, México, 1990, 758pp.

[Duda 72] Duda, R. y Hart P, **"Use the Hough Transform to Detect Lines and Curves in Pictures",** Ed. Communications of the ACM, Vol. 15, No. 1, U.S.A., 1972, 11-15 pp.

[Erdei 92] Erdei. E. Guillermo, **"Código de barras. Diseño, impresión y control de calidad"**, Ed. Mc. Graw Hill, México, 1992, 192pp.

[Escalera 01] De la Escalera, Arturo, **" Visión por Computador, Fundamentos y Métodos",** Ed. Prentice Hall, España, 2001, 304pp.

[Foley 96] Foley, James D., van Dam, Andreis, Feiner, Steven K., Hughes, Jonh F. y Phillips, Richard L., **"Introducción a la Graficación por Computador"**, Ed. Addison Wesley Iberoamericana, U.S.A., 1996, 650pp.

[Granville 92] Granville, Willian Anthony, **"Cálculo Diferencial e Integral",** Ed. Limusa, México, 1992, 677pp.

[Heileman 98] Heileman, Gregory L., **"Estructuras de Datos, Algoritmos y Programación Orientada a Objetos",** Ed. McGraw-Hill, 1998, 305pp.

[Ibarra 90] Ibarra Zannatha, Juan Manuel, **"Notas del curso de robótica",** Departamento de Control Automático, CINVESTAV, México, 1990, 60pp.

[Ibarra 98] Ibarra Zannatha, Juan Manuel, **"Visión Artificial para Robots",** Departamento de Control Automático, CINVESTAV, México, 1998, 69pp.

[IFR 98] "**World Robotics 1998**", Ed. Naciones Unidas y la International Federation of Robotics (IFR), New York y Genova, 1998, 299 pp.

[Internet 4.1] `www.disclab.ua.es/gava/proyectos/teleoperacion/Cliente/portada.html`

[Internet 4.2] `www.mor.itesm.mx/~lsi/redii/telecontrol/control_distancia.html`

[Internet 4.3] `138.100.76.157`

[Jackson 85] Jackson, Philip C., **"Introduction to Artificial Intelligence",** Ed. Dover Publications, U.S.A., 1985, 453pp.

[Johnson 87] Johson, Nelson, **"Advanced Graphics in C: Programming and Techniques",** Ed. McGraw-Hill, U.S.A., 1987, 669pp.

[Kanade 87] Kanade, Takco, **"Three-Dimensional Machine Vision",** Ed. Kluwer, U.S.A., 1987, 609pp.

[Larousse 80] **"Pequeño Larousse de Ciencias y Técnicas",** Ed. Larousse, México, 1980, 1056pp.

[Malo 93] Malo Tamayo, Alejandro, **"Robot UNIMATE Serie 100 Modelo 130",** Ed. CINVESTAV, México, 1993, 42pp.

[Mammone 94] Mammone, Richard J., **"Artificial Neuronal Networks for Speech and Vision",** Ed. Chapman & hall, UK, 1994, 586pp.

[Masters 94] Masters, Timothy, **"Signal and Image Processing with Neuronal Networks a C++ Sourcebook",** Ed. John Wiley & Sons, U.S.A., 1994 , 417pp.

[MatLab 99] **"Image Processing Toolbox. For use with MATLAB",** The Math Works inc. U.S.A., 1999, 400pp.

[Mazaira 94] Mazaira, Israel Morales, **"Manipulación de objetos en movimiento con el robot UNIMATE S-130, asistido por visión",** CINVESTAV, México, 1994, 91pp.

[Microchip 98] "**PIC16F8X, 18-pin Flash/EEPROM 8-bit Microcontrollers**", Ed. Microchip, U.S.A., 1998, 124pp.

[Nakamura 92] Nakamura, Shoichiro**, "Métodos Numéricos Aplicados con Software",** Ed. Prentice may, México, 1992, 570pp.

[Ogata 93] Ogata, Katsuhiko, **"Ingeniería de Control Moderna",** Ed. Prentice Hall, México, 1993, 1020pp.

[Ramirez 70] Ramirez Villareal, Humberto, **"Curso Rápido de Radio",** Ed. Diana, México, 1970, 263pp.

[Schildt 87] Schildt, Herbert, **"Utilización de C en inteligencia artificial",** Ed. McGraw-Hill, España, 1987, 340pp.

[Schildt 92] Schildt, Herbert, **"Turbo C/C++. Manual de Referencia"**, Ed. McGraw-Hill, México, 1992, 240pp.

[Schilling 90] Schilling, Robert J., **"Fundamentals of Robotics Analysis and Control",** Ed. Prentice Hall, U.S.A., 1990, 424pp.

[Shneiderman 98] Shneiderman Ben, '**Designing the Users Interface, Strategies for effective Human-Computer Interaction**", 3ra edición, Ed. Addison-Wesley, U.S.A., 1998, 638pp.

[Staugaard 87] Staugaard, Andrew C., **"Robotics and IA: An Introduction to Applied Machine Intelligence",** Ed. Prentice Hall, U.S.A., 1987, 373pp.

[Tenenbaum 93] Tenenbaum, Aaron M., Yedidyah Langsam y Moshe A. Augenstein, **"Estructuras de Datos en C",** Ed. Prentice Hall, 1993, 673pp.

[Teuchert 46] Hans Teuchert, **"Alta Frecuencia y Radiotecnía",** Ed. Labor, Argentina, 1946, 474pp.

[Viakon 00] "**Manual del Electricista**", Ed. Viakon- Conductores Monterrey, México, 2000, 115pp.

[VRML 97] **"The Virtual Reality Modeling Language",** Internet: `www.vrml.org/Specifications/VRML`, 1997, 236pp.

[Willians 89] Willians, Arthur B., **"Microprocesadores, Dispositivos Periféricos, Optoelectrónicos y de Interfaz",** Ed. McGraw-Hill, México, 1989, 280pp.