Pojo

Centro de Investigación y Estudios Avanzados.

Departamento de Ingeniería Eléctrica.

Sección de Computación

Una solución al problema de la correspondencia en un par estéreo utilizando descriptores de curvas.

Tesis que presenta Joaquín SALAS[1] para obtener el grado de Maestro en Ciencias en la especialidad de Ingeniería Eléctrica. Trabajo dirigido por el Dr. Sergio CHAPA.

México. D.F.. Agosto 1991.

[1]Becario de CONACYT.

AGRADECIMIENTOS:

Este trabajo es fruto de un acuerdo de colaboración científica entre las escuelas **E.N.S.T. de Bretagne** en Francia y **CINVESTAV del I.P.N.** en México.

Francia. Quiero agradecer a los profesores **M. Allan HILLION**, Jefe del Departamento de Matématicas y Sistemas de Comunicación, y **M. Jean-Marc BOUCHER**, responsable de mi estancia, por haberme recibido tan amablemente en su departamento. Así también, a los profesores **M. Christian ROUX**, Jefe del Grupo de Tratamiento de Imágenes, y **M. Guy CAZUGEL**, quienes siempre tuvieron una palabra de apoyo para este proyecto. Igualmente, a **M. Jean-José JACQ** y **M. Jean-Paul LAURENT**, sin cuyo equipo y conocimientos técnicos, el proyecto no hubiera progresado como lo hizo. Finalmente, quiero agraceder los alumnos y personal de la E.N.S.T. de Bretagne, cuya amistad a lo largo de estos meses ha sido una de las experiencias más enriquecedoras que he tenido.

México. Esta estancia y en general mis estudios de Maestría han sido fuertemente apoyados en **CINVESTAV**. Quiero agradecer al profesor **Dr. David MUÑOZ**, Jefe del Departamento de Ingeniería Eléctrica, y a los señores **Dr. Guillermo MORALES**, Jefe de la Sección de Computación, **Dr. José Luis GORDILLO** y **Dr. Juan Manuel IBARRA**, Jefe de la Sección de Control Automático, por todo su apoyo para la realización de este intercambio que me ha permitido trabajar en un laboratorio del más alto nivel. También, a los profesores **Dra. Ana María MARTINEZ** y **Dr. Sergio CHAPA**, mis asesores académicos en México, por todas sus conversaciones y consejos. Por último, a los alumnos de las Secciones de Computación y Control Automático, quienes han propiciado un ambiente científico y humano muy rico, del cual me siento orgulloso de formar parte.

DEDICATORIA:

A Ramón y Socorro, quienes hace mucho tiempo me enseñaron que pocas cosas son más valiosas que un poco de conocimiento.

A Lenis y Daniel, quienes me han proporcionado un ambiente familiar pleno.

Introducción.

Las imágenes son proyecciones bidimensionales de escenas tridimensionales. La recuperación de la información de profundidad es un problema fundamental en aplicaciones tales como la navegación automática, la cartografía y la robótica. El objetivo de este trabajo es describir un método para encontrar la correspondencia entre los segmentos de curva de dos imágenes usando técnicas de estereoscopía.

Dadas dos imágenes y el modelo geométrico de las cámaras, la tarea de la estereoscopía es encontrar los puntos, que corresponden en una y en otra imagen, y señalan un punto único en la escena; después, tomando en cuenta la diferencia de los desplazamientos de los puntos en las imágenes (*disparidad*) y la posición de las cámaras, la información de profundidad puede ser calculada mediante una transformación geométrica.

En el Laboratorio de Tratamiento de Imágenes de la E.N.S.T. de Bretagne se trabaja sobre problemas relacionados con el análisis y procesamiento de imágenes, en ocasiones desarrollando investigación básica y en otras en colaboración con empresas de la región con el fin de aplicar esta tecnología a la solución de problemas.

Este trabajo se ubica dentro de un gran proyecto de análisis ocular que tiene como fin el diagnóstico automático de enfermedades. En particular los desarrollos de nuestro trabajo serán utilizados para la determinación de la curvatura de la superficie de los ojos y observar si se pueden obtener parámetros relativos a defectos tales como miopía o astigmatismo.

Nuestro propósito es el diseño y construcción de un modelo para resolver el problema de la correspondencia basado en técnicas de estereocopía. Proponemos un algoritmo que utiliza descriptores de segmentos de curva como base para recuperar la información de profundidad. Creemos que la representación de estructuras tridimensionales presupone la construcción de procesos de interpolación de superficies, lo cual requiere una solución al problema de la segmentación de objetos en la escena; pensamos que éste es un problema que merece atención por derecho propio y por ello no hemos propuesto una solución al problema de la reconstrucción tridimensional en este escrito.

El reporte se organiza en cuatro capítulos y un apéndice:

Capítulo 1. Se proporciona un panorama general de la estereoscopía; se

incluyen temas y trabajos previos importantes.

Capítulo 2. Desarrollamos los algoritmos necesarios para la realización del modelo de propuesto.

Capítulo 3. Se presentan los resultados que obtuvimos al experimentar con el modelo construído.

Capítulo 4. Señalamos algunas conclusiones y proponemos posibles mejoras al proyecto.

Apéndice. El trabajo computacional es presentado y descrito.

Contenido

Capítulo 1

La estereovisión.

1.1 Definición del problema.

Nuestros ojos son el modelo más próximo que tenemos para explicar el principio de la estereoscopía. Simplifiquemos enormemente el modelo y olvidemonos por un momento de la enorme complejidad de funcionamiento de los ojos; tomémoslos simplemente como dos captores visuales con la única misión de transferir imágenes del mundo físico a nuestro cerebro. Un ojo es, en el caso ideal, funcionalmente dependiente con respecto al otro por las siguientes causas:

- Ambos ojos funcionan durante el mismo período de tiempo; es decir, normalmente se abren y cierran simultaneamente, lo cual ocasiona que capten los mismos eventos.
- Los ejes focales de los ojos convergen en el punto de interés; pero en general, el punto de interes se encuentra de tal manera retirado que podemos considerar a los ejes focales como prácticamente paralelos.
- Físicamente, los ojos se encuentran orientados en la misma dirección y a la misma altura en su posición normal.
- El enfoque es el mismo, idealmente, para cada ojo.

Cada ojo capta una escena bidimensional. El cerebro, mediante un mecanismo que no hemos comprendido del todo, procesa ambas imágenes y nos hace

percibir la profundidad[1].

Ahora bien, un sistema de visión artificial cuyo fin fuera la inferencia de profundidad en una escena, debería contar con los siguientes componentes:

- Dos o más imágenes, tomadas por dos o más cámaras fijas o por una cámara desplazada.
- Un proceso, llamado *calibración*, que determine:
  - La posición de las cámaras en el momento de tomar las imágenes;
  - La profundidad de un punto que nos sirva de referencia para relacionar nuestras medidas con el mundo físico.
- Un sistema de inferencia que dadas como entradas el conjunto de imágenes y el modelo geométrico de las cámaras al momento de tomar las imágenes determine la profundidad de los puntos en la escena.

Varias técnicas del tipo *cooperativo* (ver §1.4) han sido estudiadas. Estas intentan relacionar conocimiento referente al movimiento, la textura o el flujo óptico con la estereoscopía. La integración de tal cantidad de datos constituye un problema muy complejo desde los puntos de vista de la arquitectura computacional y los algoritmos requeridos.

Por medio de la estereocopía se intenta reproducir el funcionamiento del sistema visual humano en el sentido de que, dado un conjunto de imágenes de una escena y el modelo geométrico de las cámaras en el momento de tomar las imágenes, el propósito de la estereoscopía es extraer la información de profundidad de la escena. En la estereoscopía clásica, el problema fundamental consiste en encontrar las proyecciones de una característica en la escena en las dos imágenes (*problema de la correspondencia*).

El arreglo geométrico más simple para experimentación estereoscópica se presenta en la Fig. 1.3. Los planos imágenes son coplanares y los ejes focales se encuentran a la misma altura. Los algoritmos de estereo visión deben identificar puntos de correspondencia entre las dos imágenes con el fin de encontrar la profundidad. Un punto en una imagen estará desplazado en la otra imagen una distancia que es dependiente del desplazamiento relativo de las cámaras y de la profundidad del punto en la escena. Una vez que

[1] Podemos inferir profundidad con un solo ojo, pero esto se debe a conocimiento previo, interpretación de movimiento ó inferencia a partir de la iluminación y las sombras.

la correspondencia ha sido efectuada, la profundidad puede ser calculada usando una transformación geométrica o triangulación (ver §1.2).

El proceso de búsqueda de los puntos de correspondencia puede ser simplificado en virtud del modelo geométrico descrito en el párrafo anterior. Dos consideraciones pueden ser postuladas:

- Los puntos que en una imagen aparezcan en la línea i(*línea epipolar*), aparecerán en la misma línea i en la otra imagen (*propiedad de epipolaridad*). El plano que se define entre la línea i y el punto en la escena se le llama *plano epipolar*.

- Definamos el arreglo de la Fig. 1.3 en términos de imagen derecha D e imagen izquierda I. Si un punto aparece en la mitad de la derecha de D entonces, si aparece en la imagen I, aparecerá en su mitad derecha. Igualmente, si un punto aparece en la mitad de la izquierda de I entonces, si aparece en la imagen D, aparecerá en la mitad izquierda. Esto reduce el espacio de búsqueda de un punto.

1.2 Geometría estereo.

Con el fin de aplicar la información de profundidad a la resolución de problemas, necesitamos referenciar cada punto que observamos en la imagen con respecto al mundo físico. Con tal efecto definimos dos sistemas coordenados: *el sistema coordenado de la imagen* que nos servirá para localizar puntos en la imagen y *el sistema coordenado global* en donde localizaremos cualquier otra cosa. La Fig. 1.1 ilustra la idea. Mediante el sistema coordenado global, localizamos el punto v y la cámara, que se encuentra trasladada del origen, rotada horizontalmente un ángulo θ, y con una dirección de enfoque alzada un ángulo ϕ. La proyección del punto v en la imagen, v', es medido con respecto al sistema coordenado de la imagen.

Nuestro problema es encontrar una transformación geométrica que nos exprese las coordenadas de un punto en la imagen en términos del sistema coordenado global. Definimos entonces, el plano de la imagen con respecto al sistema coordenado global mediante los siguientes tres pasos:

1. Trasladamos el centro del plano imagen al centro de la cámara mediante

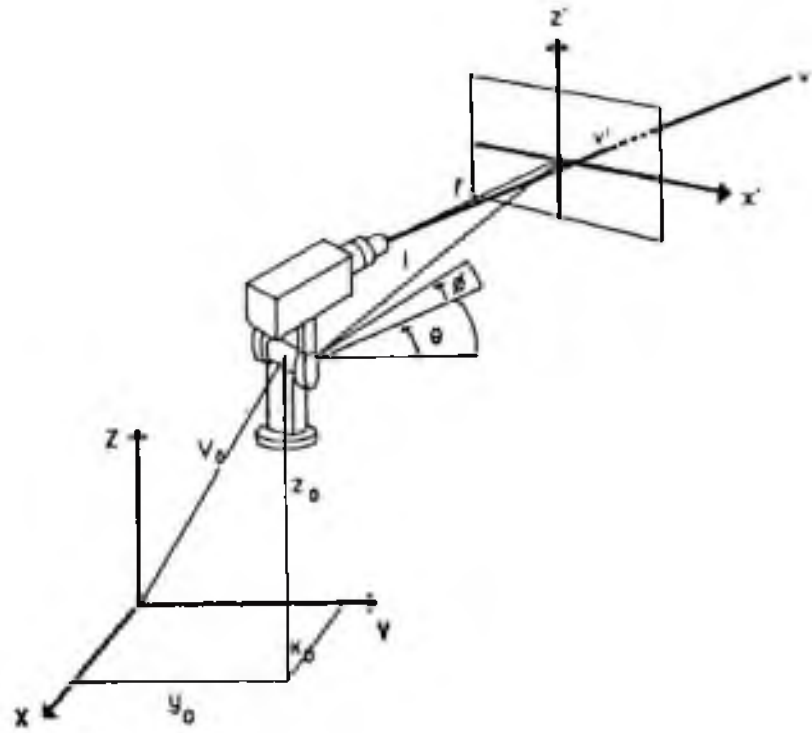


Figura 1.1: **Transformación de perspectiva.** Dos sistemas coordenados son utilizados. La cámara y el punto v son referenciados por el sistema coordenado global. El sistema coordenado de la imagen, que contiene al punto imagen v', es referenciado con respecto a la cámara. Sea v_0 la posición del centro de rotación de la cámara. La cámara tiene un giro horizontal de un ángulo θ y su dirección de enfoque está alzada un ángulo ϕ.

una matriz homogenea de traslación G expresada como:

$$G = \begin{bmatrix} 1 & 0 & 0 & -l_1 \\ 0 & 1 & 0 & -(l_2 + f) \\ 0 & 0 & 1 & -l_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.1}$$

2. Giramos horizontalmente la cámara un ángulo θ y bajamos su dirección de enfoque un ángulo ϕ, de tal manera que el eje Y (el eje de profundidad) del sistema coordenado global quede paralelo con el eje focal de la cámara; lo cual se puede representar por la matriz homogenea de rotación S, expresada como:

$$S = \begin{bmatrix} \cos\theta & -\cos\phi\sin\theta & \sin\phi\sin\theta & 0 \\ \sin\theta & \cos\phi\cos\theta & -\sin\phi\cos\theta & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

3. Trasladamos el centro de rotación de la cámara v_0 al centro del sistema global. Esto se logra aplicando el operador H, definido como una matriz homogenea por:

$$H = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

La transformación anterior, que nos expresa puntos de la imagen en el sistema coordenado global, es llamada transformación inversa de perspectiva y se expresa como:

$$v = H \times S \times G \times v' \tag{1.4}$$

Después de algunos cálculos Richard O. DUDA en [9], obtiene la localización del centro del lente de una cámara con respecto al sistema coordenado global v_0;

$$v_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} l_1\cos\theta & - & l_2\cos\phi\sin\theta & + & l_3\sin\phi\sin\theta \\ l_1\sin\theta & + & l_2\cos\phi\cos\theta & - & l_3\sin\phi\cos\theta \\ & & l_2\sin\theta & + & l_3\cos\phi \end{bmatrix} \tag{1.5}$$

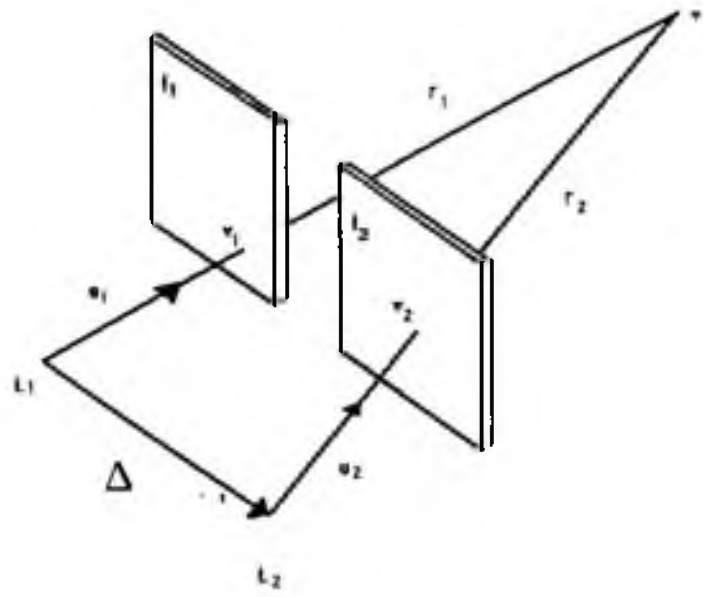


Figura 1.2: **Geometría estereo.** I_1 e I_2 representan los planos de las imágenes; $\mathbf{L_1}$ y $\mathbf{L_2}$ son los centros de las lentes; $\mathbf{r_1}$ y $\mathbf{r_2}$ son las proyecciones entre el punto $\mathbf{v}$ y los centros de las lentes, $\mathbf{u_1}$ y $\mathbf{u_2}$ son los vectores unitarios en esa dirección; $\mathbf{\Delta} = \mathbf{L_2} - \mathbf{L_1}$ es llamado el vector de base ó línea epipolar; $\mathbf{v_1}$ y $\mathbf{v_2}$ son las proyecciones de $\mathbf{v}$.

y las coordenadas de un punto v_p en la imagen con respecto al sistema coordenado global como:

$$v_p = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} +$$

$$\begin{bmatrix} (x_p + l_1)\cos\theta & - & (f + l_2)\cos\phi\sin\theta & + & (z_p + l_3)\sin\phi\sin\theta \\ (x_p + l_1)\sin\theta & + & (f + l_2)\cos\phi\cos\theta & - & (z_p + l_3)\sin\phi\cos\theta \\ & & (f + l_2)\sin\theta & + & (z_p + l_3)\cos\phi \end{bmatrix} \quad (1.6)$$

Con las ecuaciones 1.5 y 1.6 como base, analizamos ahora un arreglo arbitario de las cámaras para experimentación por estereoscopía. El arreglo se muestra en la Fig. 1.2. Enfoquemos nuestra atención en determinar la intersección de las dos proyecciones $\mathbf{r_1}$, $\mathbf{r_2}$. Idealmente, existen dos números, a

y b, tal que $a\mathbf{u}_1 = \mathbf{\Delta} + b\mathbf{u}_2$, entonces $\mathbf{v} = \mathbf{L}_1 + a\mathbf{u}_1$ nos da la respuesta buscada. Formalmente, debemos encontrar los valores de a y b que minimizen:

$$J(a,b) = \mid a\mathbf{u}_1 - (\mathbf{\Delta} + b\mathbf{u}_2) \mid^2 \tag{1.7}$$

donde podemos encontrar, después de derivar, los valores:

$$a_0 = \frac{\mathbf{u}_1 \cdot \mathbf{\Delta} - (\mathbf{u}_1 \cdot \mathbf{u}_2)(\mathbf{u}_2 \cdot \mathbf{\Delta})}{1 - (\mathbf{u}_1 \cdot \mathbf{u}_2)^2} \tag{1.8}$$

$$b_0 = \frac{(\mathbf{u}_1 \cdot \mathbf{u}_2)(\mathbf{u}_1 \cdot \mathbf{\Delta}) - (\mathbf{u}_2 \cdot \mathbf{\Delta})}{1 - (\mathbf{u}_1 \cdot \mathbf{u}_2)^2} \tag{1.9}$$

que minimizan $J(a,b)$. El valor $\mathbf{L}_1$ está dado por la ecuación 1.5. Un vector en la dirección del rayo de proyección está dado por $\mathbf{v}_i - \mathbf{L}_i$, donde $\mathbf{v}_i$ está dado por la ecuación 1.6, y entonces:

$$\mathbf{u}_i = \frac{\mathbf{v}_i - \mathbf{L}_i}{\mid \mathbf{v}_i - \mathbf{L}_i \mid} \tag{1.10}$$

donde los subíndices indican la imagen y las lentes de las cámaras.

1.2.1 Geometría simplificada

Consideremos ahora el modelo más simple para la experimentación estereoscópica representada en la Fig. 1.3.

En la figura se expresan las siguientes condiciones:

- El centro del lente de la primera cámara está en el origen.
- Los centros de las lentes de las cámaras están alineados y sobre el eje X; por lo tanto los puntos en la escena tendrán las mismas coordenadas en el eje Z en sus proyecciones sobre las imágenes; ésta es llamada *condición de epipolaridad.*
- Los ejes focales de las dos cámaras son paralelos entre si y paralelos con el eje de profundidad Y.

Para expresar matemáticamente el modelo simplificado tomamos como referencia la Fig. 1.4, en donde asumimos que la coordenada de altura Z

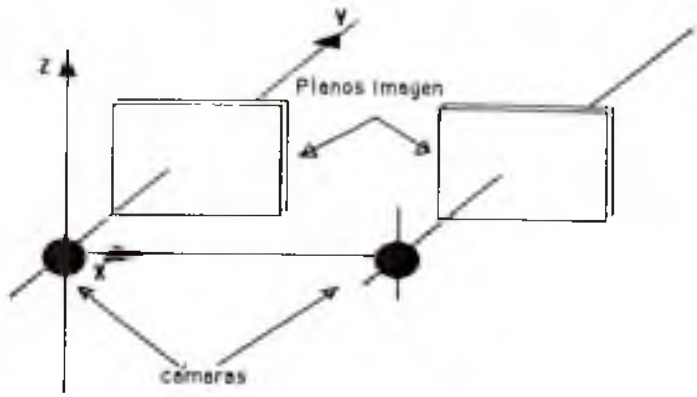


Figura 1.3: **Geometría simplificada.** Las dos cámaras están relacionadas por un desplazamiento horizontal.

es la misma para todos los puntos del plano (condición de epipolaridad) y para facilitar la comprensión de la idea, no es mencionada. Un punto p de coordenadas (x_{p_1}, y_p), se proyecta sobre dos planos imágenes separados del centro de sus lentes una distancia f. Gracias a la geometría del modelo y a la semejanza entre triángulos, las siguientes relaciones pueden ser postuladas:

$$\frac{x_1}{f} = \frac{x_{p_1}}{y_p} \tag{1.11}$$

$$\frac{x_2}{f} = \frac{x_{p_2}}{y_p} \tag{1.12}$$

y

$$d = x_{p_1} + x_{p_2} \tag{1.13}$$

donde d es la distancia entre los centros de las lentes de las cámaras, x_1 y x_2 son las distancias de las proyecciones sobre el eje X de la imagen del punto p a los centros de las lentes de las cámaras 1 y 2 respectivamente y x_{p_1} y x_{p_2} son distancias de las proyecciones del punto p a los centros de las lentes de las cámaras 1 y 2 respectivamente.

Resolviendo este sistema de ecuaciones, el cálculo de la profundidad Y se reduce a:

$$Y = \frac{fd}{x_1 - x_2} \tag{1.14}$$

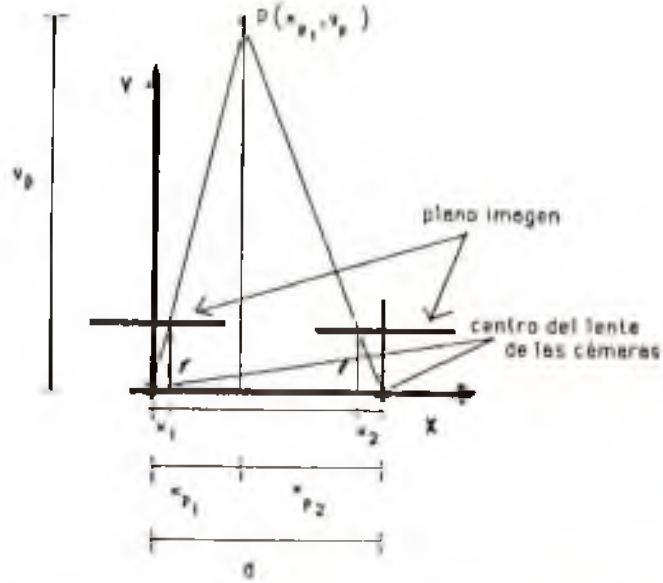


Figura 1.4: **Geometría estereo simplificada.** El punto p se proyecta sobre dos imágenes; f es la distancia focal de las lentes; x_1 y x_2 son las distancias de las proyecciones en las imágenes sobre el eje X al centro de las lentes; x_{p_1} y x_{p_2} son las distancias de las proyecciones del punto p sobre el eje X al centro de las lentes y d es la distancia entre las lentes.

que nos reduce el problema del cálculo de la profundidad de un punto, al conocimiento de la distancia focal f propia del lente, la distancia entre el centro del lente de las cámaras d y la disparidad entre los puntos.

1.3 Detección de contornos.

Una vez determinado que la solución al problema de obtención de información de profundidad se resuelve por el descubrimiento de la correspondencia entre características en ambas imágenes, debemos determinar cuales son esas características. Eric Leifur GRIMSON en [3], determinó que las características sobresalientes de una superficie se obtienen de acuerdo a los cambios que la luz experimenta al reflejarse en las superficies, es decir el cambio en la luminosidad. El cambio de luminosidad se mantiene constante entre los contornos

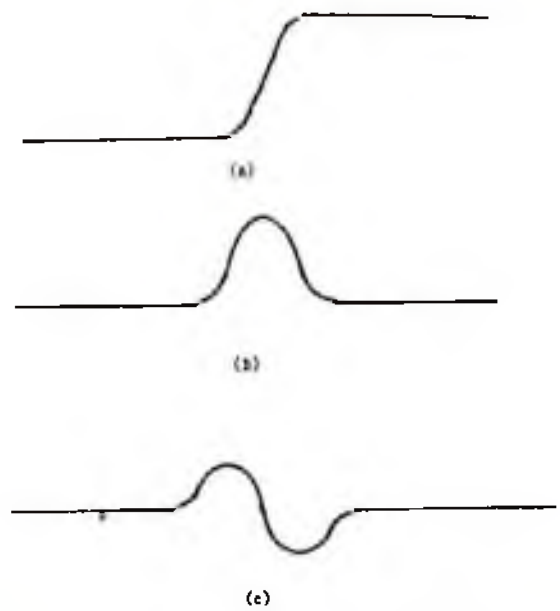


Figura 1.5: **Detección de cambios en una imagen unidimensional.** En (a) se presenta un cambio en la luminosidad. El cambio es asociado a un máximo local en la primera derivada (b) y un cruzamiento por cero en la segunda derivada (c).

de un objeto[2]. Los lugares de cambio de luminosidad tienen por lo tanto, la propiedad matemática de que la primera derivada presenta un máximo o un mínimo y la segunda derivada un cruzamiento por cero (ver Fig. 1.5).

La derivada es un operador direccional, esto significa que la operación se realiza tomando como referencia alguno de los ejes coordenados. Esto presenta la dificultad de que algunas imágenes no tienen sus cambios de luminosidad de tal manera que la derivada los detecte fácilmente. La Fig. 1.6 presenta el caso de la aplicación de la segunda derivada en la detección de los cruces por cero. La fidelidad del resultado del operador depende de la dirección en la que se aplique. Debemos aplicar la segunda derivada de tal manera que coincida con la dirección de máximo cambio, con el fin de obtener un resultado fiable.

Para resolver el problema de la determinación del máximo cambio en la

[2]Esto establece un paralelismo con la estructura física de los objetos, pues entre los contornos el cambio de profundidad se mantiene constante.

segunda derivada, se utiliza comúnmente el Operador Laplaciano ∇^2; el cual obtiene la máxima magnitud permitiendonos prescindir de la dirección en la cual se aplica la operación.

Prácticamente, el Laplaciano no es aplicado directamente sobre la imagen. Primeramente, la imagen es suavizada por medio de un filtro que degrade las altas frecuencias (filtro pasa-bajas). Se ha utilizado ampliamente la convolución de la imagen con un filtro Gaussiano:

$$G_\sigma(x,y) * I(x,y) \tag{1.15}$$

donde el operador Gaussiano G_σ, está dado por:

$$G_\sigma(x,y) = \sigma^2 \exp^{-(x^2+y^2)/(2\sigma^2)} \tag{1.16}$$

e I es la función de la imagen.

Matemáticamente, la imagen es filtrada utilizando el siguiente grupo de operadores:

$$f(x,y,\sigma) = \nabla^2 G_\sigma * I(x,y). \tag{1.17}$$

El operador de convolución $\nabla^2 G_\sigma$ está dado por:

$$\nabla^2 G_\sigma(r,\sigma) = \left(\frac{r^2 - 2\sigma^2}{\sigma^4}\right) \exp\left(\frac{-r^2}{2\sigma^2}\right) \tag{1.18}$$

donde:

$$r = \sqrt{x^2 + y^2} \tag{1.19}$$

El parámetro libre σ determina el tamaño espacial de la función.

Para obtener una descripción de los cambios en la imagen, aplicamos el filtro y localizamos los cruzamientos por cero en el resultado.

1.4 Trabajos previos.

El problema fundamental en estereoscopía, llamado *el problema de la correspondencia*, es encontrar las proyecciones de las características de la escena en las dos imágenes. Estas características pueden ser puntos, rectas, curvas, superficies, etc. Tradicionalmente, podemos clasificar las soluciones al problema de la correspondencia desde dos puntos de vista:

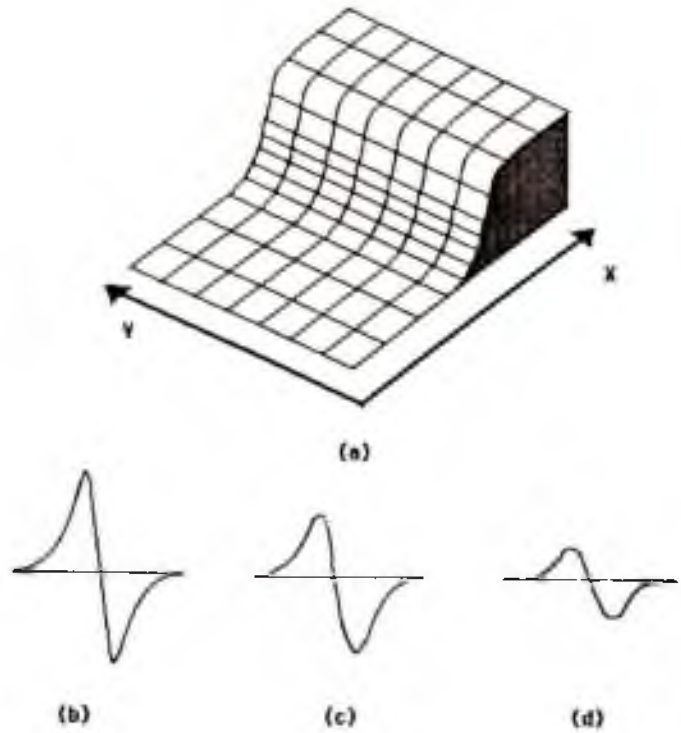


Figura 1.6: **Factores espaciales y direccionales en la aplicación de la segunda derivada.** En (a) se muestra un cambio de luminosidad y (b), (c) y (d) son las segundas derivadas en varias direcciones. En (b) la segunda derivada es tomada con respecto al eje x, y en (c) y (d) la segunda derivada es tomada a 30 y 60 grados con respecto al eje x. La segunda derivada con respecto a Y daría cero en esta zona.

Técnicas basadas en la intensidad. Se basan en la búsqueda de la correspondencia punto a punto de las imágenes tomando en cuenta la luminosidad de los pixeles en la imagen. Por ejemplo los siguientes dos algoritmos del tipo cooperativo[3]:

- Ellen C. HILDRETH en [5], presentó un algoritmo que involucra movimiento con estereoscopía, tal que la organización del movimiento en una serie de imágenes nos permiten analizar el ambiente en términos de objetos, su movimiento en el espacio y su estructura tridimensional, inferida en base al cambio en las intensidades de luz que percibe la cámara. Su análisis se divide en dos etapas. Primero la determinación de la magnitud y dirección de la velocidad de los elementos en la imagen en base a sus cambios de intensidad. Segundo los objetos son separados y su estructura tridimensional inferida.
- Nasser M. NASRABADI, Sandra P. CLIFFORD, y Yi LIU en [4], integran el flujo óptico (desplazamiento de los pixeles entre una imagen y la siguiente) con la intensidad de los puntos en la imagen para producir una función que expresará las discontinuidades de profundidad de la solución. La información de intensidad de los pixeles en las imágenes estereo es utilizada para calcular la estructura tridimensional, mientras que el flujo óptico confirma la aproximación y ofrece solución para las regiones ocluídas u ocultas.

Técnicas basadas en características. Se selecciona alguna primitiva como líneas rectas, esquinas de objetos o segmentos de curvas para el apareamiento. Los algoritmos basados en características son menos caros, computacionalmente, que los algoritmos basados en la intensidad debido a la utilización de primitivas.

A continuación presentaremos dos algoritmos para la solución del problema de la estereoscopía basados en características, que fueron de preciosa ayuda para el desarrollo del presente trabajo.

[3]Son algoritmos del tipo cooperativo aquellos que obtienen información de movimiento, flujo óptico u otro y lo integran como conocimiento a la información estereoscópica en la búsqueda de la profundidad.

Segmentos de línea recta como primitivas.

Ramakant NEVATIA y Gerard MEDIONI propusieron un algoritmo en [8], en el cual utilizan segmentos de línea recta como primitivas. Una serie de filtros, cada uno de ellos sensible a una dirección de recta diferente, es utilizado para determinar la dirección de la recta en un punto. El resultado es encadenado a otras rectas. Para cada recta se almacena un conjunto de descriptores consistente en: las coordenadas de los puntos finales, la orientación y el promedio del contraste.

El algoritmo de correspondencia.

- Para cada recta i en una imagen se construye una ventana en la otra imagen sobre la cual se buscan la rectas j que corresponden a i. Se define la función $p(i,j)$, que indica el promedio de la disparidad o diferencia entre los puntos correspondientes a las rectas i y j.

- Sobre el conjunto de características que corresponden se calcula una función de evaluación $v(i,j)$, que consiste en comparar los valores de la función $p(i,j)$ para el conjunto de rectas en la vecindad próxima de i y j. La función $v(i,j)$ se utiliza como una forma de darle robustes a la correspondencia, pues utiliza además, la restricción de continuidad en los objetos de la escena (ver Fig.1.7). Al final se elige el segmento con el valor de $v(i,j)$ que indica mayor certeza de continuidad del contorno del objeto.

Utilización de la dirección de la curva.

Nasser M. NASRABADI y Yi LIU en [7] presentaron un algoritmo que utiliza segmentos de curva como primitivas en el proceso de correspondencia. Ellos aplican el filtro laplaciano (ver §1.3) dando tres valores diferentes a la variable de control σ con lo cual obtienen tres imágenes diferentes de contorno para cada imagen del par estereo. Para cada imagen del contorno obtienen los segmentos de curva de la siguiente manera:

- Encontrar un pixel del contorno.

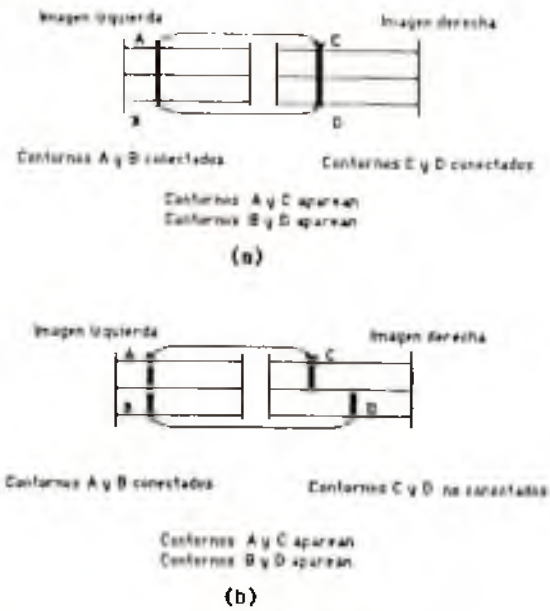


Figura 1.7: **Restricción de continuidad.** En (a) Los contornos B y D son compatibles con el conocimiento previo, mientras que en (b) no lo son.

- Determinar la dirección del promedio local del gradiente y sobre cada pixel de la vecindad ocho conectada buscar el segmento de curva. Un pixel es sujeto de búsqueda si la dirección del promedio local del gradiente no cambia demasiado con respecto al anterior.

Una tabla, llamada R tabla, es construída para cada segmento de curva. La tabla contiene para cada pixel, su orientación y su distancia al centroide de la curva.

Las entradas al algoritmo de correspondencia son los segmentos de curva extraídos de las imágenes. Cada segmento está identificado por una etiqueta, la localización de su centroide, su R-tabla, la longitud de la curva y la localización de cada pixel de contorno en la curva.

Se construye un grafo relacional para cada imagen. Los nodos del grafo son las localizaciones del centroide y los arcos las relaciones entre los centroides. Las R-tablas representan la estructura interna del nodo y las distancias entre los centroides las características estructurales de los objetos en la escena. Los grafos de las escenas son similares y podemos aplicar un algoritmo que nos busque el máximo número de asociaciones no contradictorias

o *máximo clique* sobre el grafo.

Procediendo desde el nivel de menor al mayor detalle (desde un valor σ alto hasta uno bajo se evaluan las nuevas disparidades encontradas en base a las primeras encontradas.

1.5 Informalmente: nuestro algoritmo.

Nuestro algoritmo parte de una premisa fundamental: Los contornos de los objetos son aproximables analíticamente por trozos; ésto es, podemos aproximar una función sobre una parte del contorno del objeto que está limitada por puntos de singularidad, partes donde la función no es continua. Si podemos caracterizar esos segmentos de curva naturales de los objetos habremos dado el paso de aumentar el nivel de abstracción desde niveles de gris en la imagen hasta expresiones matemáticas de los contornos. Tras lo cual el proceso de correspondencia entre los objetos se puede llevar a cabo tomando en cuenta las características de cada curva y la relación de las curvas con respecto a los objetos y por último la relación de los objetos en la escena.

La descripción de todo tipo de curvas a partir de una imagen no es tarea sencilla, el ruido por una parte y la enorme variedad de curvas por otro nos vuelven la tarea extremadamente complicada. Proponemos la aproximación de curvas mediante pequeños segmentos de línea recta; lo suficientemente pequeños como para que la naturaleza original de la curva no se pierda y lo suficientemente grandes como para que expresen de manera robusta la dirección de la curva en esa parte. Igualmente se propone la sustitución de la expresión análitica de la curva por su expresión en términos de descriptores; si el conjunto de descriptores de la curva es completo, entonces la curva será suficientemente distinguible. Por último, proponemos el empleo de la información estructural de los segmentos de curva con respecto a los objetos con el fin de eliminar ambigüedades.

El método propuesto tiene algunas restricciones de antemano. Las oclusiones no se manejan y estamos atenidos a una buena detección de contornos. El primer problema lo dejamos abierto; el segundo se soluciona con la utilización de los potentes operadores (Laplaciano del Gaussiano, Canny[4], etc.), que la teoría del procesamiento de imágenes nos ofrece.

[4]Ver la presentación de John CANNY en [11], para mayor detalle.

Capítulo 2

El modelo de estereoscopía propuesto.

2.1 Introducción.

El esquema del modelo para experimentación estereoscópica desarrollado se muestra en la Fig. 2.1. Dos imágenes son tomadas siguiendo el modelo presentado en la subsección 1.2.1. En seguida, las imágenes se sometieron a un proceso de detección de contornos de acuerdo a la teoría que se expondrá en §2.2. Los contornos detectados resultaban demasiado gruesos para realizar la extracción de contornos con nuestro modelo, por lo que antes las imágenes fueron sometidas a un algoritmo de adelgazamiento de líneas. Este algoritmo será formalmente presentado en §2.3. El modelo de extracción de contornos será introducido en §2.4. Se busca la correspondencia entre los segmentos de curva extraídos de acuerdo a nuestro alegato en §2.5 y por último la profundidad es detectada siguiendo las ecuaciones de nuestro modelo geométrico (ver subsección 1.2.1).

2.2 Filtro Gradiente.

Si $(\delta f)/(\delta x)$ y $(\delta f)/(\delta y)$ son la razón de cambio de una función f en dos direcciones perpendiculares x y y, entonces la razón de cambio en cualquier

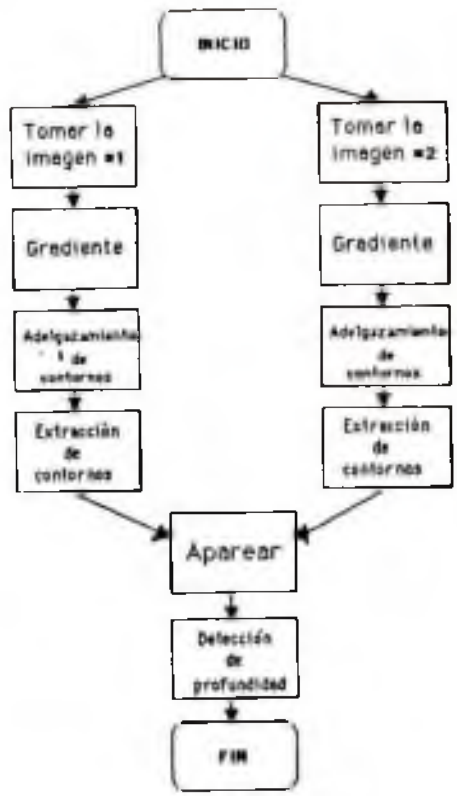


Figura 2.1: **Diagrama a bloques del modelo propuesto.**

dirección θ, medida desde el eje-x, es:

$$D_u\ f(x,y) = \frac{\delta f}{\delta x}\cos\theta + \frac{\delta f}{\delta y}\sin\theta \tag{2.1}$$

donde $D_u\ f(x,y)$ expresa la deivada direccional de $f(x,y)$. La dirección en la cual la razón de cambio tiene su máximo valor es $\arctan(\delta x/\delta y)$, su magnitud es $\sqrt{(\delta f/\delta x)^2 + (\delta f/\delta y)^2}$. El vector que tiene esta dirección y esta magnitud es llamado el *gradiente* de f.

La respuesta de la derivada direccional varía de acuerdo a la orientación del contorno. Para usar la derivada en la detección de contornos, utilizamos la magnitud del gradiente, el cual nos proporciona la razón de cambio en la dirección de máxima variación.

Para poder utilizar una computadora digital, debemos trasladar nuestras expresiones para trabajar con diferencias en lugar de derivadas y con una función I discreta en lugar de la función f continua:

$$(\Delta_x I)(x,y) \equiv I(x,y) - I(x-1,y) \tag{2.2}$$

$$(\Delta_x I)(x,y) \equiv I(x,y) - I(x,y-1) \tag{2.3}$$

Lo cual equivale a realizar la convolución de I con los siguientes vectores:

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \quad \text{y} \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{2.4}$$

La magnitud del operador gradiente sobre una imagen I se expresa como:

$$G[I(x,y)] \cong \sqrt{[I(x,y) - I(x+1,y)]^2 + [I(x,y) - I(x,y+1)]^2} \tag{2.5}$$

Resultados similares pueden alcanzarse utilizando valores absolutos:

$$G[I(x,y)] \cong | I(x,y) - I(x+1,y) | + | I(x,y) - I(x,y+1) | \tag{2.6}$$

La relación entre los pixeles es mostrada en la Fig. 2.2 (a).

Otra forma de obtener el gradiente es el llamado *Operador Roberts*, el cual utiliza las diferencias cruzadas (ver Fig. 2.2 (b)):

$$G[I(x,y)] \equiv \sqrt{[I(x,y) - I(x+1,y+1)]^2 + [I(x+1,y) - I(x,y+1)]^2} \tag{2.7}$$

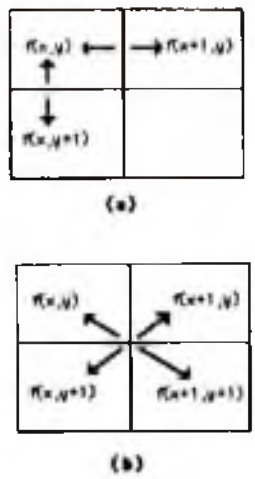


Figura 2.2: **Cálculo del filtro gradiente.**
En (a) y (b) se muestran los cálculos de las ecuaciónes 2.6 y 2.8 respectivamente.

o, usando valores absolutos:

$$G[I(x,y)] \cong | I(x,y) - I(x+1,y+1) | + | I(x+1,y) - I(x,y+1) | \quad (2.8)$$

Existen muchas maneras de utilizar el resultado del filtro gradiente; a nosotros nos interesa caracterizar los pixeles de la imagen como pertenecientes a una de dos clases: CONTORNO o FONDO; por lo cual aplicamos la siguiente relación:

$$J(x,y) = \begin{cases} \text{CONTORNO} & \text{si } G[I(x,y)] \geq \alpha \\ \text{FONDO} & \text{si no} \end{cases} \quad (2.9)$$

Donde α es un valor real arbitrario llamado *umbral de binarización.*

2.3 Adelgazamiento de líneas.

El adelgazamiento de líneas es utilizado como una etapa intermedia entre la detección de los contornos y su extracción. El objetivo del adelgazamiento de líneas es generar un *esqueleto unitario* (un esqueleto de un pixel de grosor). Es decir, cada pixel debe estar conectado con no más de dos pixeles adjacentes a

P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

Figura 2.3: **Vecinos del punto P_1 en una ventana de 3 × 3**

menos que se trate de una intersección de curvas. A continuación presentamos el algoritmo propuesto por T.Y. ZHANG y C.Y. SUEN con la mejora de Humberto SOSSA (ver [10]):

Algoritmo de adelgazamiento de líneas

Sea la función discreta $I(x, y, g)$, la representación de una imagen digital, donde $1 \leq x \leq C$, $1 \leq y \leq H$, y $0 \leq g \leq 1$. H y C son el número de hileras, columnas respectivamente; x,y y g ϵ enteros. Definamos a P_i como instancias de I llamadas puntos. I así considerado contiene solamente dos clases (CONTORNO y FONDO), de tal manera que:

$$I(i,j,k) = \begin{cases} \text{CONTORNO} & \text{si } k = 1 \\ \text{FONDO} & \text{si } k = 0 \end{cases} \tag{2.10}$$

Para el proceso de adelgazamiento se usará una ventana de 3 × 3 con el ordenamiento de pixeles que se muestra en la Fig. 2.3.

Algoritmo: Adelgazamiento de líneas.

Entradas: La imagen digital I.

Salidas : La imagen digital I' cuyas líneas forman un esqueleto unitario.

1. Realizar el siguiente conjunto de iteraciones sobre la función I:

   Iteración 1. El punto P_1 es borrado de la imagen si cumple las siguientes condiciones:

   (a) $P_2 \times P_4 \times P_6 = 0$

   (b) $P_4 \times P_6 \times P_8 = 0$

   (c) $2 \leq B(P_1) \leq 6$

(d) $A(P_1) = 1$

donde $A(P_1)$ es el número de patrones 01 en el conjunto ordenado $P_2, P_3, \ldots, P_9$, y

$$B(P_1) = \sum_{i=2}^{9} P_i \tag{2.11}$$

es el número de vecinos no cero de P_1.

Iteración 2. En esta segunda iteración, las condiciones 1a y 1b son cambiadas por:

(a) $P_2 \times P_4 \times P_8 = 0$

(b) $P_2 \times P_6 \times P_8 = 0$

y el resto permanece igual.

Iteración 3. En esta tercera iteración, el punto P_1 es borrado de la imagen si cumple una de las siguientes condiciones:

(a) $\overline{P_9} \times P_4 \times P_6 = 1$

(b) $\overline{P_3} \times P_6 \times P_8 = 1$

(c) $\overline{P_5} \times P_8 \times P_2 = 1$

(d) $\overline{P_7} \times P_2 \times P_4 = 1$

2. Repetir el paso anterior hasta que ningún punto sea borrado de la imagen.

2.4 Extracción de contornos.

La imagen original ha sido pasada por un detector de contornos en la forma del Operador Roberts. Después, los contornos han sido erosionados hasta lograr su representación como un esqueleto unitario. Ahora debemos manejar la imagen para extraer de ella una información que represente un nivel más alto de abstracción que simples niveles de gris.

Definición 2.1 Segmento de curva. *Es la secuencia conectada de pixeles definida entre dos esquinas o junciones.*

Una de las consecuencias de la definición 2.1, los segmentos de curva se definen sobre $\mathcal{Z}^\epsilon$ (el dominio de los enteros) y no sobre $\mathcal{R}^\epsilon$ el dominio de los reales.

Definición 2.2 Esquina o junción. *Es el lugar en el cual la dirección de la curva cambia más que un cierto ángulo ϑ o deja de existir.*

Nuestro propósito es encontrar los segmentos de curva entre los que se definen los objetos en la imagen, caracterizar esas curvas mediante un conjunto de descriptores y utilizar los descriptores para encontrar las curvas que corresponden. Un proceso posterior tomará las curvas que correspondieron y la información de profundidad podrá ser calculada punto a punto.

De acuerdo a las condiciones experimentales a las que ha sido sometido nuestro modelo de la cámara, los segmentos de curva varían poco; de tal manera que los descriptores se conservan dentro de un rango pequeño. Pero la extracción de la curva sigue siendo afectada por el ruido y defectos en la operación de detección de contornos.

Suposición 2.1 *Cualquier segmento de curva puede ser aproximada por segmentos de línea recta.*

El grado de exactitud de la aproximación depende de la longitud y del número de segmentos de línea recta; por lo tanto, para reducir el error se tenderá a la elección de un gran número de segmentos de recta pequeños.

Supongamos que tenemos el círculo de la Fig. 2.4 y las aproximaciones en base a tres, cuatro y ocho segmentos de rectas del mismo tamaño. Si tomamos un número suficientemente grande de segmentos de rectas (lo cual determina la longitud del segmento) obtendremos una representación tan exacta como queramos del objeto original. Así pues, las curvas presentes en una imagen podrán ser aproximadas por segmentos de línea recta y al mismo tiempo conservar las propiedades de la curva, siempre y cuando los segmentos de recta tengan una longitud suficientemente pequeña.

Desarrollo del algoritmo

Sea la función discreta $I(x, y, g)$, la representación de una imagen digital, donde $1 \leq x \leq C$, $1 \leq y \leq H$, y $0 \leq g \leq 1$. H y C son el número de hileras y columnas respectivamente; las variables x, y y g enteros. Definamos a p como una instancia de I, a la que llamaremos punto. La función I así considerada contiene solamente dos clases (CONTORNO y FONDO), de tal manera que:

$$I(i, j, k) = \begin{cases} \text{CONTORNO} & \text{si } k = 1 \\ \text{FONDO} & \text{si } k = 0 \end{cases} \tag{2.12}$$

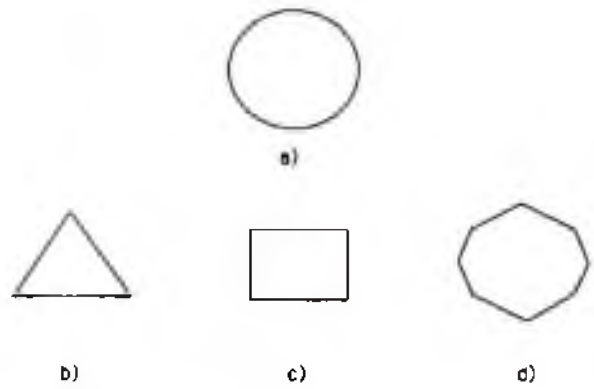


Figura 2.4: **Aproximación de un objeto por segmentos de rectas.** En (a) se presenta el objeto original, en (b), (c) y (d) se aproxima por 3, 4 y 8 segmentos de línea recta.

Definamos finalmente $V(p)$ como la vecindad de p, que corresponde a un cuadrado que tiene como centro a p y tiene longitud de lado entera y positiva y por lo demás arbitraria.

Algoritmo: Obtención de los segmentos de curva de una imagen.

Entradas: La imagen digital I, caracterizada por las clases CONTORNO y FONDO.

Salidas : El conjunto E, que representa las curvas en I.

1. Encontrar el punto $p(i, j, k)$ tal que $k \epsilon$ CONTORNO. Si no existe ir al paso 7
2. Aproximar el segmento de recta l que se describe en $V(p)$, si no existiera ir al paso 1. En otro caso guardar los puntos extremos de ese segmento, q y r, para futuras referencias.
3. Utilizar el punto q, un extremo del segmento de recta, para encontrar un segmento de recta l_1 en $V(q)$ que tendrá como puntos extremos q_1 y q_2. Comparar el segmento l_1 con el obtenido en el paso anterior; si la diferencia entre las pendientes es muy grande

hemos encontrado un extremo del segmento de curva y debemos continuar con el otro extremo (ir al paso 5).

4. Uno de los puntos extremos del segmento de recta encontrado, q_1 o q_2, es igual al centro del segmento anterior, continuar el seguimiento del segmento de curva en la dirección del otro punto (ir al paso 3).

5. Utilizar el punto r, un extremo del segmento de recta, para encontrar un segmento de recta l_2 en $V(r)$ que tendrá como puntos extremos r_1 y r_2. Comparar el segmento l_2 con el obtenido en el paso anterior; si la diferencia entre las pendientes es muy grande hemos encontrado el segundo extremo del segmento de curva y debemos continuar con otra curva. Agregar la curva recién recorrida al conjunto E (ir al paso 1).

6. Uno de los puntos extremos del segmento de recta encontrado, r_1 o r_2 es igual al centro del segmento anterior, continuar el seguimiento del segmento de curva en la dirección del otro punto (ir al paso 5).

7. Hemos finalizado el análisis de la escena.

El problema de la dimensión de la $V(p)$ queda abierto. Debe ser tan pequeño para no correr el riesgo de que dos más rectas entren a la ventana y suficientemente grande como para expresar de manera robusta una dirección. Nosotros hemos encontrado experimentalmente que un valor de 11×11 para una imagen de 768 columnas $\times$ 512 hileras es adecuado.

La Fig. 2.5 presenta una ventana, al seguir en las direcciones de los puntos p y q podemos explorar todo el segmento de curva y al mismo tiempo extraer el conjunto de descriptores de la curva. El conjunto de descriptores para cada segmento de curva incluyen:

- La longitud del segmento de curva.
- La dirección promedio del segmento de curva.
- Las coordenadas del centroide.
- Las coordenadas de los puntos inicial y final del segmento de curva.

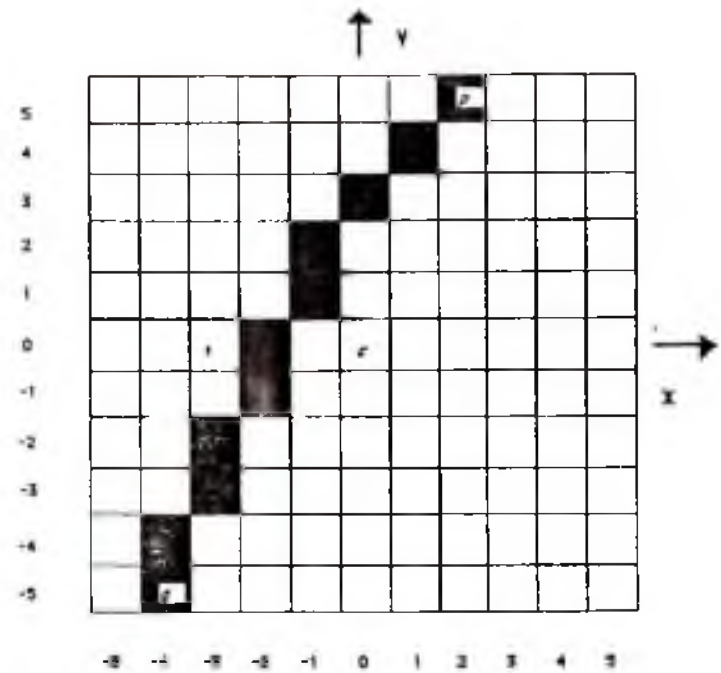


Figura 2.5: **Ejemplo de una ventana.** Las coordenadas de los puntos inicial y final son $(2, 5)$ y $(-4, -5)$ y la recta definida es aproximadamente $y = 1.5x + 3$. En la figura, c es el centro de la ventana y p y q los extremos del segmento de recta definido.

2.4.1 Determinación de los segmentos de recta.

El problema consiste en aproximar un conjunto de puntos ubicados dentro una ventana a una línea recta; para ello elegimos el método de mínimos cuadrados.

La solución natural de aproximar mediante la minimización de la distancia de los puntos a la recta conduce a ecuaciones muy complejas (ver [2]), en las cuales la eliminación de términos que llevan a la simplificación tiende a perder la respuesta para casos degenerados, como $x = k$ (donde k es una constante arbitraria). En el presente trabajo obtuvimos las soluciones para dos funciones:

- La que calcula el error asumiendo que las coordenadas x de los datos de entrada son buenos y los de y pueden ser malos.
- Asumir que las coordenadas y de los datos de entrada son buenos y las x pueden estar mal.

Escogemos los parámetros que minimizan la función que calcula el error para la suma de las distancias a la recta.

Las coordenadas de x son exactas y las de y pueden no serlo. Minimizemos:

$$S = \sum_{i=1}^{n}(y_i - ax_i - b)^2 \tag{2.13}$$

Es decir, postulemos las condiciones: $\delta S/\delta a = \delta S/\delta b = 0$. Aplicando estas condiciones obtenemos:

- $\delta S/\delta a = 0$

$$\sum_{i=1}^{n} x_i(y_i - ax_i - b) = 0 \tag{2.14}$$

- $\delta S/\delta b = 0$

$$\sum_{i=1}^{n}(y_i - ax_i - b) = 0 \tag{2.15}$$

Resolviendo para a y b llegamos a las siguientes ecuaciones:

$$a = \frac{n\sum_{i=1}^{n}(x_i y_i) - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \tag{2.16}$$

$$b = \frac{1}{n}\left\{\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i\right\} \tag{2.17}$$

Las coordenadas de y son exactas y las de x pueden no serlo. Minimizemos:

$$S = \sum_{i=1}^{n}\left(x_i - \frac{y_i - b}{a}\right)^2 \tag{2.18}$$

Aplicando las condiciones $\delta S/\delta a = \delta S/\delta b = 0$, y resolviendo para a y b, tenemos:

$$a = \frac{n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n}(x_i y_i) - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i} \tag{2.19}$$

$$b = \frac{1}{n}\left\{\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i\right\} \tag{2.20}$$

Entre los conjuntos de parámetros calculados, escogemos aquellos que minimizan la función E:

$$E = \frac{\sum_{i=1}^{n}(a_o x_i^2 + y_i^2 + b_o)^2}{a_o^2 + 1} \tag{2.21}$$

2.5 Algoritmo de correspondencia.

El proceso de correspondencia se realiza en dos etapas. Durante la primera, se construye un grafo que representa las correspondencias potenciales de cada curva de acuerdo a los valores de sus descriptores; los desempates son efectuados en base al desplazamiento observado por la mayoría de las curvas. La segunda etapa trata de encontrar la correspondencia entre los segmentos de curva que no fueron abarcados durante la etapa anterior; para ello se forman grupos de curvas que llamamos *circuitos* y que representan una sucesión de segmentos de curva muy próximas entre si .

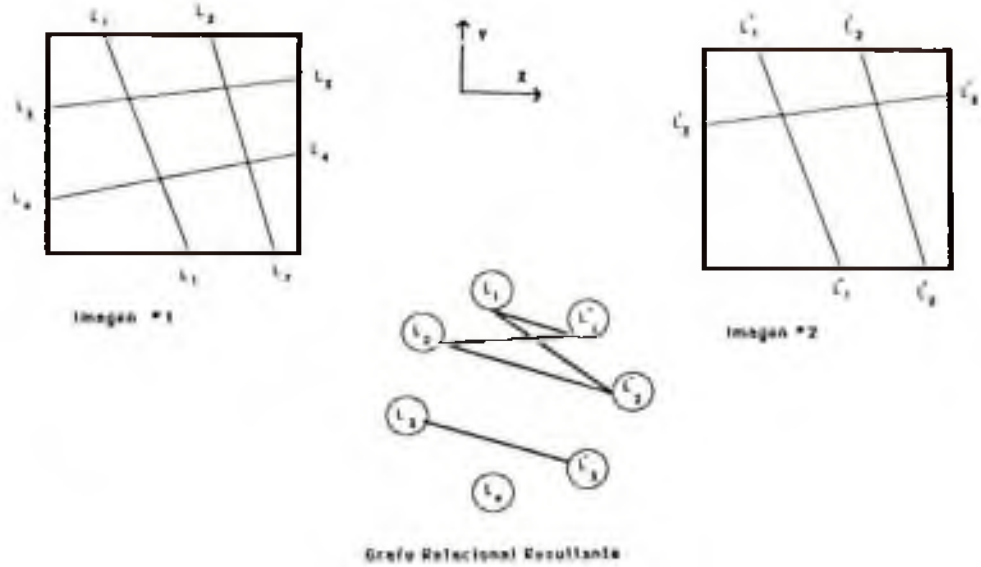


Figura 2.6: **Representación de una estructura relacional por un grafo.**
La imagen izquierda y derecha contienen los segmentos L_i y L'_i respectivamente; de ellas se deriva el grafo resultante.

2.5.1 Primera etapa.

Definamos una estructura relacional sobre el conjunto de elementos V_i que forman los segmentos de curva $v_{i,j}$ localizadas en la imagen i. Sea el conjunto P de predicados unarios p_k, definidos sobre cada elemento de V_i:

- $p_1(v_{i,j})$. El centroide de la curva,
- $p_2(v_{i,j})$. El promedio de la dirección de la curva,
- $p_3(v_{i,j})$. La longitud de la curva.

Formemos por último el predicado binario R definido sobre pares de elementos de la primera y segunda imagen:

$$R(v_{1,r}, v_{2,s}) = \begin{cases} VERDADERO & \text{si} \begin{cases} (p_1(v_{1,r}) \cong p_1(v_{2,s})) \wedge \\ (p_2(v_{1,r}) \cong p_2(v_{2,s})) \wedge \\ (p_3(v_{1,r}) \cong p_3(v_{2,s})) \end{cases} \\ FALSO & \text{en otro caso.} \end{cases} \tag{2.22}$$

con $1 \leq r \leq N_1$ y $1 \leq s \leq N_2$. Donde N_1 y N_2 representan el número de segmentos de curva en la primera y segunda imagen respectivamente. Si aplicamos el predicado R a los segmentos de curva resultantes del proceso de extracción de contornos, obtendremos un grafo de asociación entre nodos con propiedades semejantes.

Refirámonos a la Fig. 2.6, las imágenes izquierda y derecha están constituidas por los segmentos de curva L_i y L'_i respectivamente. Bajo el modelo de experimentación presentado en §1.2 y suponiendo que las líneas de epipolaridad fueran paralelas al eje X y que el predicado R es *VERDADERO*, tendríamos el grafo relacional resultante mostrado. La correspondencia que se presenta entre L_1 L'_1 y L_1 - L'_2, y entre L_2 - L'_1 y L_2 - L'_2 puede ser manejada por la naturaleza del desplazamiento del centroide presentado por la mayoría de los candidatos a corresponder; en escenas con un mayor número de segmentos de curva que corresponden esta tendencia es aún más marcada.

2.5.2 Segunda etapa.

Se da el caso que por algún defecto en la detección de contornos o aún del adelgazamiento de líneas, algunos segmentos de curvas no corresponden; esta etapa trata con ellas.

Definición 2.3 Circuito: *Es un conjunto T de segmentos de curvas en los que se conserva la propiedad de proximidad mediata o inmediata.*

Definición 2.4 Proximidad inmediata: *Dados dos segmentos de curva diferentes r y s, existe proximidad inmediata entre ellos, si la distancia entre uno de los extremos de r a uno de los extremos de s es menor que un valor arbitrario γ. El valor de γ tiende a ser pequeño y del orden de unos cuantos pixeles.*

Definición 2.5 Proximidad mediata: *Dados dos segmentos de curva diferentes p y q, existe proximidad mediata entre ellos, si existen r_1, r_2, ..., r_{k+1} para $k \geq 2$, tal que $r_1 = p$ y $r_{k+1} = q$ y hay proximidad inmediata entre r_i y r_{i+1} $\forall_{i=1,\ldots,k}$.*

Ejemplos ilustrativos de los conceptos de proximidad mediata e inmediata se presentan en la Fig. 2.7. Una imagen puede estar compuesta de varias

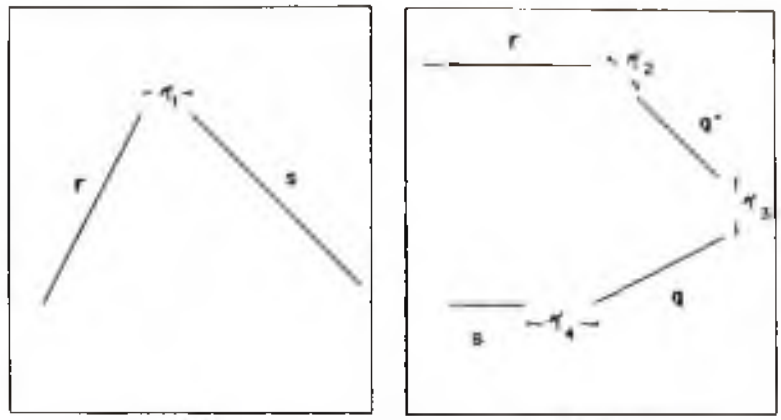


Figura 2.7: **Proximidad mediata e inmediata.** En (a) r y s tienen proximidad inmediata. En (b) r y s tienen proximidad mediata, ya que no tienen proximidad inmediata pero existe un q tal que tiene proximidad inmediata con s y ofrece proximidad mediata por intermedio de q' con r. En la figura se cumple que $\forall_i\ \gamma_i \leq \gamma$; donde γ es un valor real.

decenas de *circuitos*; idealmente, los *circuitos* representan estructuras independientes y es posible describir la escena en base a la relación de los *circuitos* que la forman.

Algoritmo: Cálculo de la correspondencia entre grupos de segmentos de curva.

Entradas: Los conjuntos T_1 y T_2, el conjunto de *circuitos* de la imagen 1 y 2 con elementos $t_{1,i}$ y $t_{2,i}$ respectivamente..

Salidas: El conjunto M de correspondencias entre grupos de curvas.

PROCEDURE PosEmpate;
VAR i, j, m, n:**INTEGER**;
BEGIN
  FOR i:=1 **TO** numero de *circuitos* en T_1
    Selecciona el *circuito-i* de T_1, $t_{1,i}$
    FOR j:=1 **TO** numero de curvas para obtener descriptores
      Seleccionar el siguiente conjunto de j-curvas del *circuito* $t_{1,i}$

```
      Calcular el valor de los descriptores del grupo-j
      FOR m:=1 TO numero de circuitos en T2
        Selecciona el circuito-m de T2, t2,m
        FOR n:=1 TO numero de curvas para obtener descriptores
          Seleccionar el siguiente conjunto de n-curvas del circuito t2,m
          Calcular el valor de los descriptores del grupo-n
          IF el predicado R = VERDADERO y
               lo hace mejor que una correspondencia anterior THEN
            Eliminar la anterior correspondencia de M que incluyera
            alguna de las curvas actuales y
            agregar la actual correspondencia.
          END
        END
      END
    END
  END
END PosEmpate;
```

Observaciones: El número de curvas para obtener descriptores, se refiere al número máximo de curvas que esperamos unir. La selección del conjunto de curvas de un *circuito* toma en cuenta solo los segmentos de curva con proximidad inmediata, ésto con el fin de disminuir la complejidad del algoritmo. El valor de los descriptores se calcula considerando que el grupo de curvas forma una curva nueva.

Capítulo 3

Resultados experimentales.

Con la teoría y algoritmos presentados en el capítulo anterior fueron desarrollados una serie de programas para computadora. Se dispuso para tal efecto de una computadora Macintosh IIfx equipado con una tarjeta para la digitalización de imágenes marca Neotech; dicha tarjeta genera una imagen de 768 columnas por 512 hileras en hasta 256 niveles de gris. Los programas fueron desarrollados en lenguaje C, pues la versión de MPW (*Machintosh Programming Workshop*) cuenta con un excelente rastreador de errores (*debuger*) que facilita la programación.

Los cálculos de profundidad son presentados como tablas de datos en los que aparece la profundidad del centroide; pues por el momento no contamos con un medio de presentación conveniente para la observación por parte de las personas. Las representaciones de estructuras tridimensionales, requieren la construcción de procesos de interpolación de superficies, lo cual requiere una solución al problema de la segmentación de objetos. Creemos que esos son problemas interesantes que merecen atención por derecho propio y no hemos propuesto una solución para ellos en este escrito.

3.1 La tasa.

3.1.1 Imágenes originales.

La Fig. 3.1 presenta las imágenes originales de una tasa. Las imágenes de la tasa fueron las únicas, en esta serie de experimentos, en las que la cámara

Figura 3.1: **Imágenes originales de la tasa.**

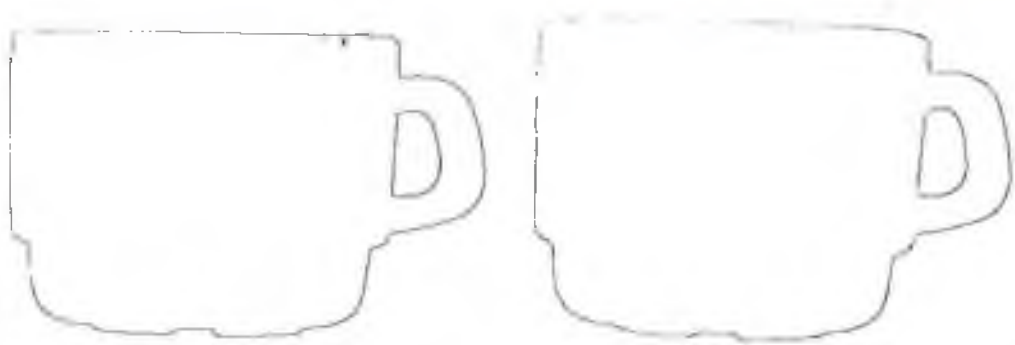


Figura 3.2: Imágenes gradiente de la tasa.

fué calibrada. La profundidad medida desde el centro del lente hasta la base donde se apoyo la tasa fué de 60 cm, la separación entre las dos imágenes fué de 4cm y la distancia del lente al plano imagen fué de 80mm.

3.1.2 Imagen gradiente.

La Fig. 3.2 presenta las imágenes gradientes obtenidas de la Fig. 3.1. Las imágenes en la Fig. 3.1 fueron tratadas para binarizar en OBJETO y FONDO mediante la manipulación de su histograma. El nivel de gris establecido para el filtro gradiente fué de 20.

Figura 3.3: **Adelgazamiento de líneas de la tasa.**

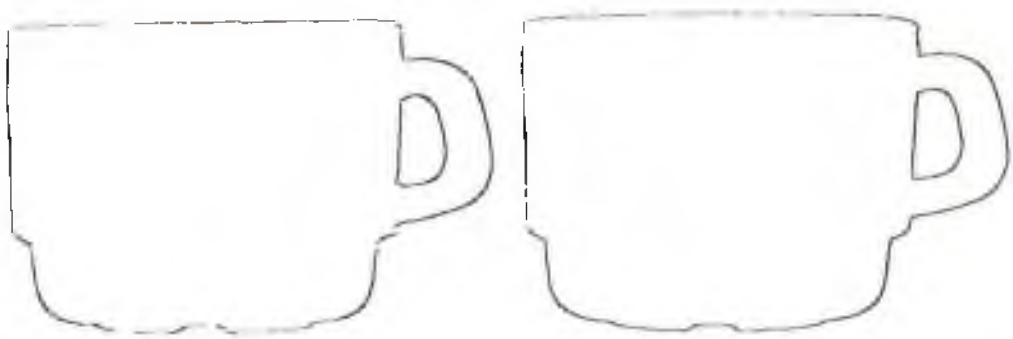


Figura 3.4: **Extracción de contornos de la tasa.**

3.1.3 Adelgazamiento de líneas.

La Fig. 3.3 presenta el resultado de aplicar el algoritmo de adelgazamiento de líncas a las imágenes de la Fig. 3.2. El resultado obtenido son líneas con grosor de un solo pixel (*esqueleto unitario*).

3.1.4 Extracción de contornos.

La Fig. 3.4 presenta el resultado de aplicar el algoritmo de extracción de contornos a las imágenes de la Fig. 3.3. La condición de continuidad se cumple siempre y cuando no existan dos segmentos de recta cuya diferencia de pendiente sea mayor a 30 grados.

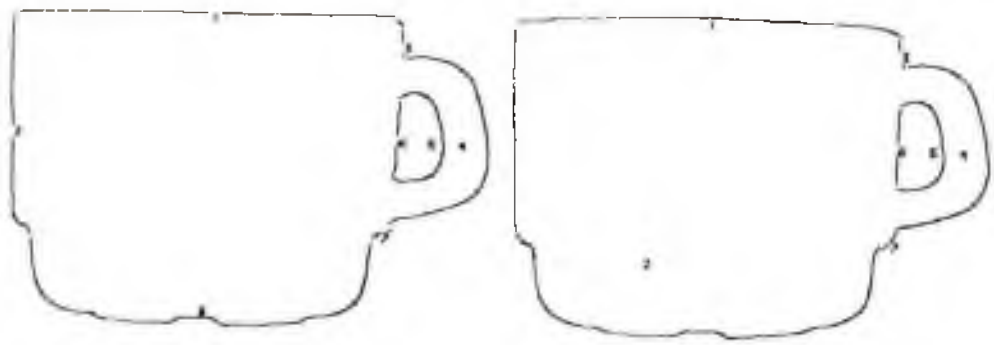


Figura 3.5: **Etiquetación de las curvas extraídas a las imágenes de la tasa.**

En la Fig. 3.5 se etiquetan las curvas de las imágenes de la Fig. 3.4. Varias observaciones se desprenden:

- La aplicación de la definición 2.1, que hacía referencia a las características de continuidad de los segmentos de curva que deseábamos extraer, es mantenida.
- La curva 2 de la imagen derecha de la Fig. 3.4 es más larga de lo que podíamos esperar. Esto se debe a qué, a lo largo del segmento de curva, no se presentaron dos segmentos de recta cuya diferencia de pendiente fuera mayor al permitido.

Las tablas 3.1 presentan los descriptores de cada uno de los segmentos de curva registrados en la Fig. 3.5. La longitud de cada segmento debe ser multiplicada por 5 para obtener el valor en pixeles. La dirección está expresada en radianes y el resto de las coordenadas toma como origen la esquina superior izquierda de la imagen, con direcciones positivas hacia la derecha y hacia abajo.

3.1.5 Apareamiento

Los segmentos de curva que aparecen en las tablas 3.1 fueron sometidas al proceso de apareamiento. Los resultados de la primera etapa aparecen en la tabla 3.2; en la primera columna aparece la etiqueta del segmento de curva

Tabla 3.1: **Descriptores de los segmentos de curva extraídos de la tasa.**

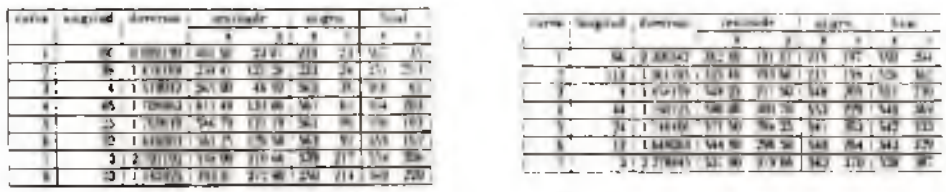


Tabla 3.2: **Resultados del apareamiento de las curvas en las imágenes de la tasa. Primera etapa.**

curva	apareada	profundidad
1	1	54.599407
3	3	54.437870
4	4	54.872761
5	5	55.073330
6	6	55.089821

de la imagen 1, en la segunda la etiqueta del segmento de curva de la segunda imagen y en la tercera columna la profundidad, que se expresa en centímetros.

Los resultados de la segunda etapa, consistente en aparear grupos de segmentos de curva aparecen en la tabla 3.3. La primera columna tiene las etiquetas de los segmentos de curva de la primera imagen y la segunda columna los de la segunda imagen.

Todas las correspondencias son correctas y la profundidad calculada de los centroides concuerda con los resultados esperados. La curva número 7 de la primera imagen no logró aparear con la 7 de la segunda imagen ya que la diferencia entre el promedio de la dirección entre las dos era muy grande.

Tabla 3.3: **Resultados del apareamiento de las curvas en las imágenes de la tasa. Segunda etapa.**

grupo #1	grupo #2
2, 8	2

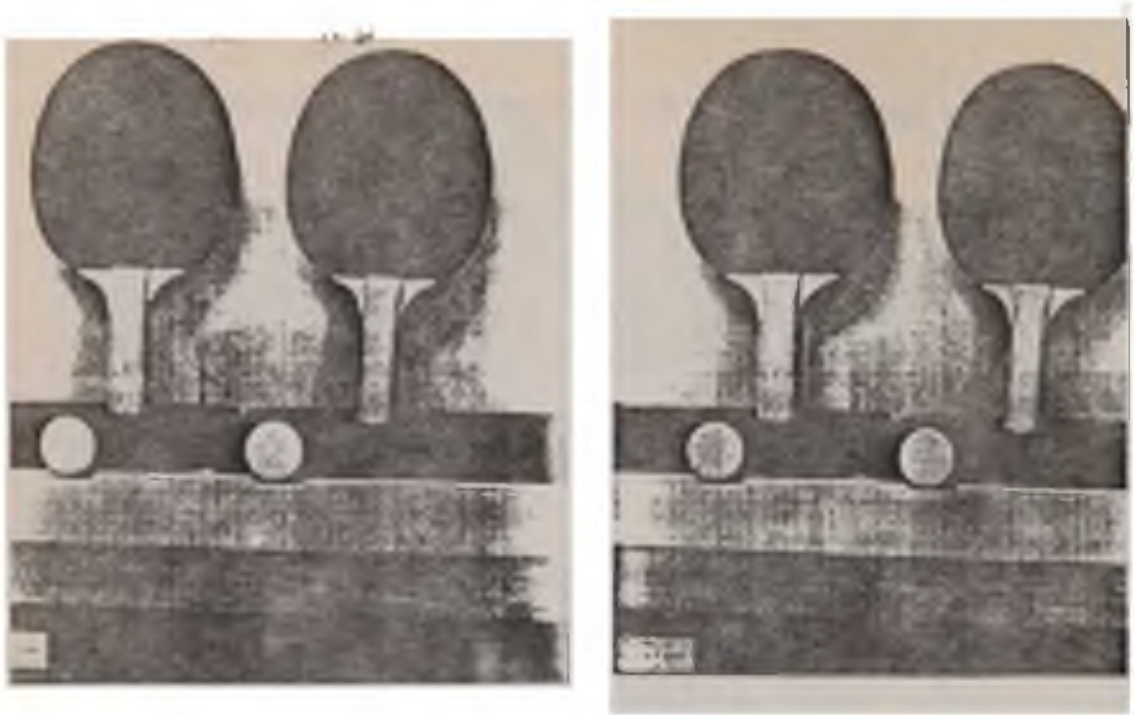


Figura 3.6: Imágenes originales de las raquetas.

3.2 Las raquetas de ping-pong.

3.2.1 Imágenes originales.

La Fig. 3.6 presenta las imágenes originales de unas raquetas y unas pelotas de ping-pong. Las imágenes del equipo de ping-pong no fueron calibradas por lo cual la información de profundidad obtenida no tiene ningún significado físico.

3.2.2 Imagen gradiente.

La Fig. 3.7 presenta las imágenes gradientes obtenidas de la Fig. 3.6. El nivel de gris establecido para binarizar la imagen fue de 20. Varias observaciones se desprenden:

- En los mangos de las raquetas se observa una zona de mucho ruido.

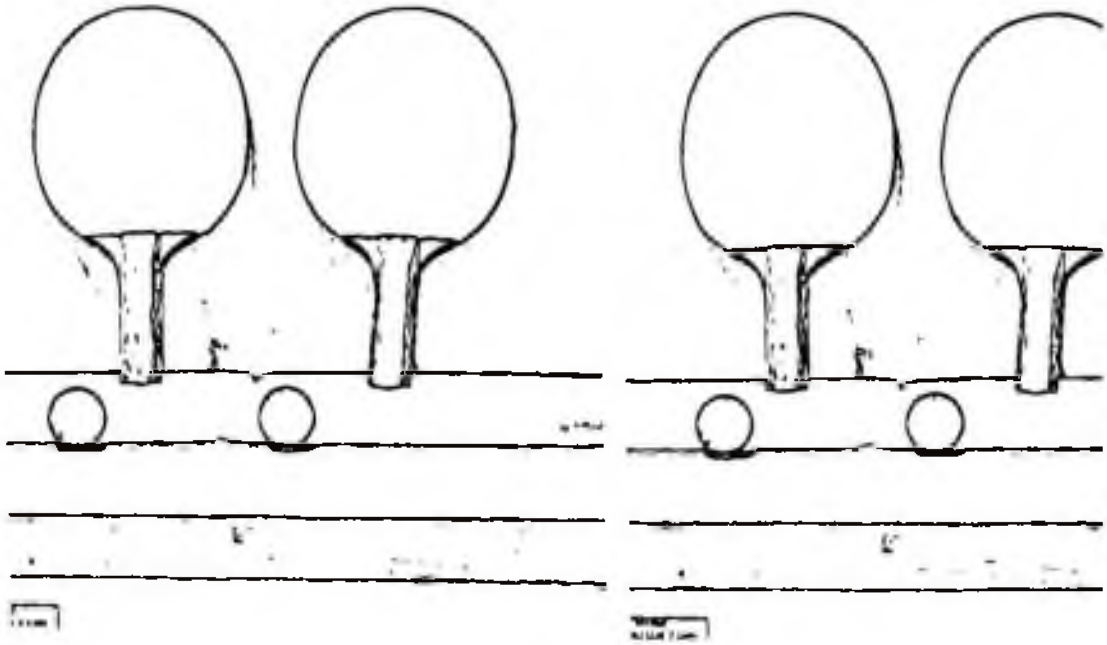


Figura 3.7: **Imágenes gradiente de las raquetas.**

- Algunas proyecciones de las sombras de las raquetas han pasado como parte de los contornos de los objetos.
- La sombra que reflejan las pelotas de ping-pong han pasado como parte del contorno.

3.2.3 Adelgazamiento de líneas.

La Fig. 3.8 presenta el resultado de aplicar el algoritmo de adelgazamiento de líneas a la Fig. 3.7. El resultado obtenido son líneas con grosor de un pixel (*esqueleto unitario*).

3.2.4 Extracción de contornos.

La Fig. 3.9 presenta el resultado de aplicar el algoritmo de extracción de contornos a la Fig. 3.8. La condición de continuidad se cumple siempre y

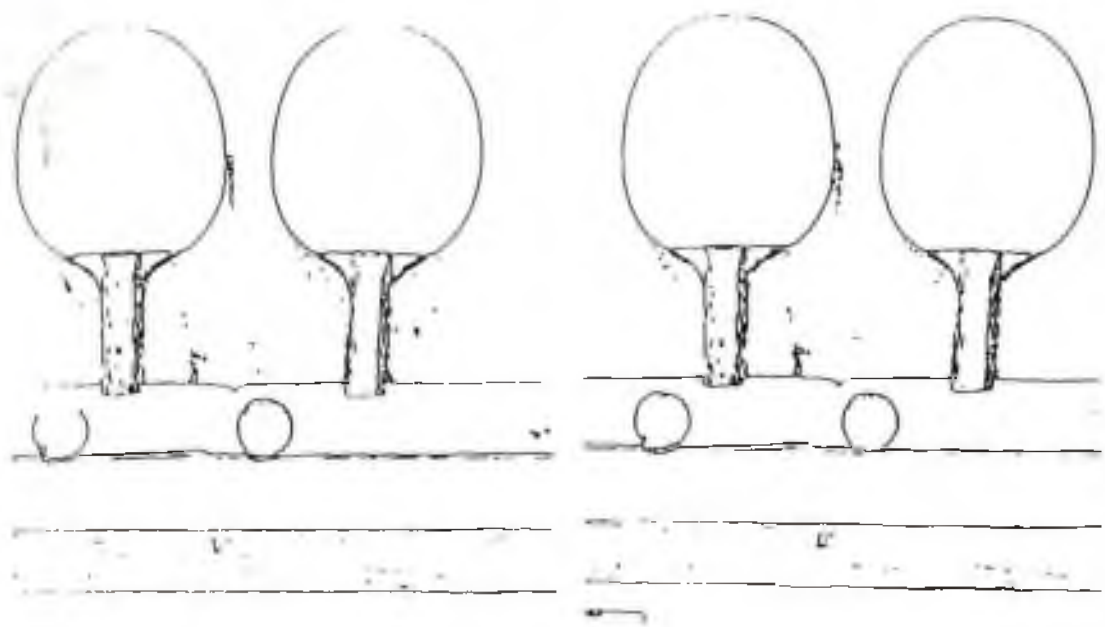


Figura 3.8: **Adelgazamiento de líneas a las imágenes de las raquetas.**

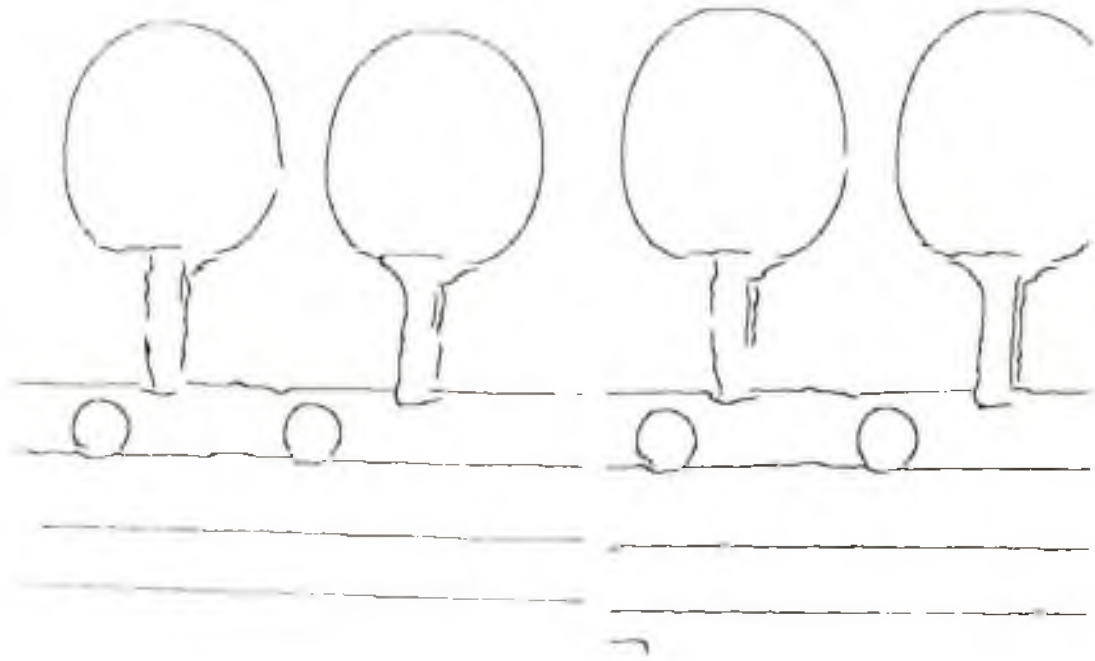


Figura 3.9: **Extracción de contornos a las imágenes de las raquetas.**

cuando no existan dos segmentos de recta cuya diferencia de pendiente sea mayor a 45 grados. En la Fig. 3.9 solo aparecen los segmentos de curva cuya longitud es mayor o igual a 25 pixeles, ello elimina bastante del ruido existente y no le resta mucha comprensibilidad a las imágenes.

En la Fig. 3.10 se etiquetan las curvas de las imágenes de la Fig.s 3.9. Varias observaciones se desprenden:

- La aplicación de la definición 2.1, que hacía referencia a las características de continuidad de los segmentos de curva que deseabamos extraer, es mantenida.
- Las zonas de los mangos han sido definidas con un grado más o menos satisfactorio, considerando el ruido existente en ese lugar desde la ima gen gradiente (Fig. 3.7).

El par de tablas 3.4, se presentan los descriptores de cada uno de los segmentos de curva registrados en las imágenes de la Fig. 3.10. La longitud

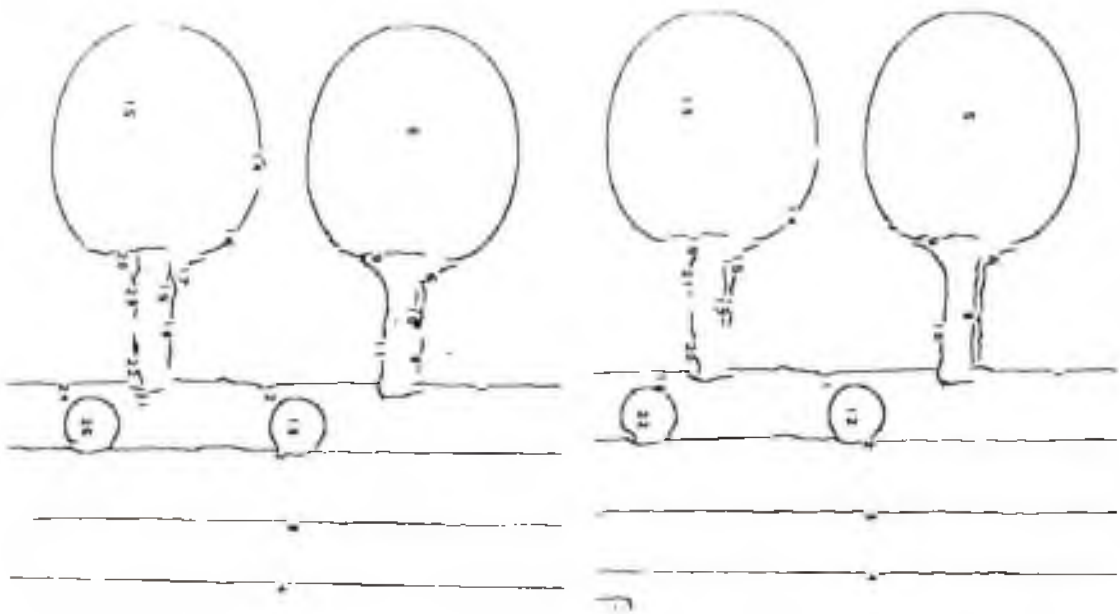
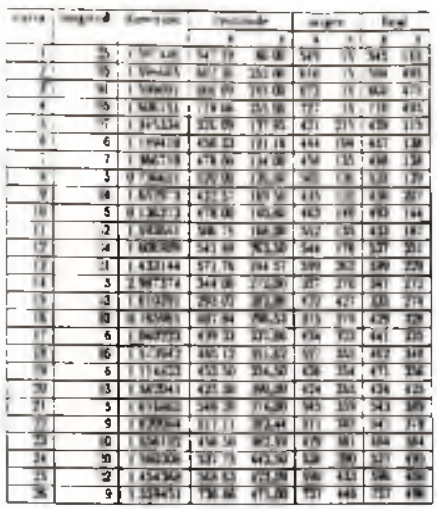
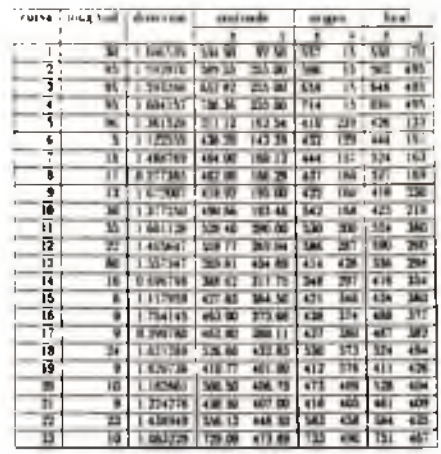


Figura 3.10: **Etiquetación de las curvas extraídas a las imágenes de las raquetas.**

Tabla 3.4: **Descriptores de los segmentos de curva extraídos de las raquetas.**





de cada segmento debe ser multiplicada por 5 para obtener el valor en pixeles. La dirección está expresada en radianes y el resto de las coordenadas toma como origen la esquina superior izquierda de la imagen, con direcciones positivas hacia la derecha y hacia abajo.

3.2.5 Apareamiento.

Los segmentos de curva que aparecen en el par de tablas 3.4 fueron sometidas al proceso de correspondencia. Los resultados de la primera etapa aparecen en la tabla 3.5; en la primera columna aparece la etiqueta del segmento de curva de la imagen 1, en la segunda la etiqueta del segmento de curva de la segunda imagen y en la tercera la profundidad, que no expresa medidas reales. Los renglones con asterisco(*) en la columna de comentario indican una correspondencia incorrecto.

A partir de los resultados de la tabla 3.5, podemos decir que el apareamiento fue razonablemente bueno. El apareamiento del segmento de curva 7 de la primera imagen con la 18 de la segunda y el de la 16 de la primera con el 23 de la segunda demuestran que nuestro concepto de *circuito* si no malo si es por lo menos incompleto. Visualmente, podíamos haber esperado que

Tabla 3.5: **Resultados de la correspondencia en las raquetas. Primera etapa.**

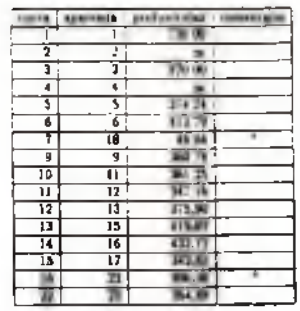


Tabla 3.6: **Resultados de la correspondencia en las raquetas. Segunda etapa.**

grupo #1	grupo #2
21, 24	18

el segmento 19 de la primera imagen corresponderá con la 20 de la segunda, pero al observar sus datos observamos que los umbrales de aceptación para la longitud no se alcanzaron.

Los resultados de la segunda etapa, consistente en buscar correspondencia entre los grupos de segmentos de curva aparecen en la tabla 3.6. La primera columna tiene las etiquetas de los segmentos de curva de la primera imagen y la segunda columna los de la segunda imagen.

Se encontró una gran cantidad de curvas que correspondeiron aunque se hicieron algunos errores. Las curvas 7 y 8 de la primera imagen no formaron par con la 7 de la segunda, en la segunda etapa del algoritmo ya que la curva 7 de la segunda y la 7 de la segunda imagen había encontrado correspondendia. En los casos de las otras curvas las diferencias entre los descriptores justifican su soltería.

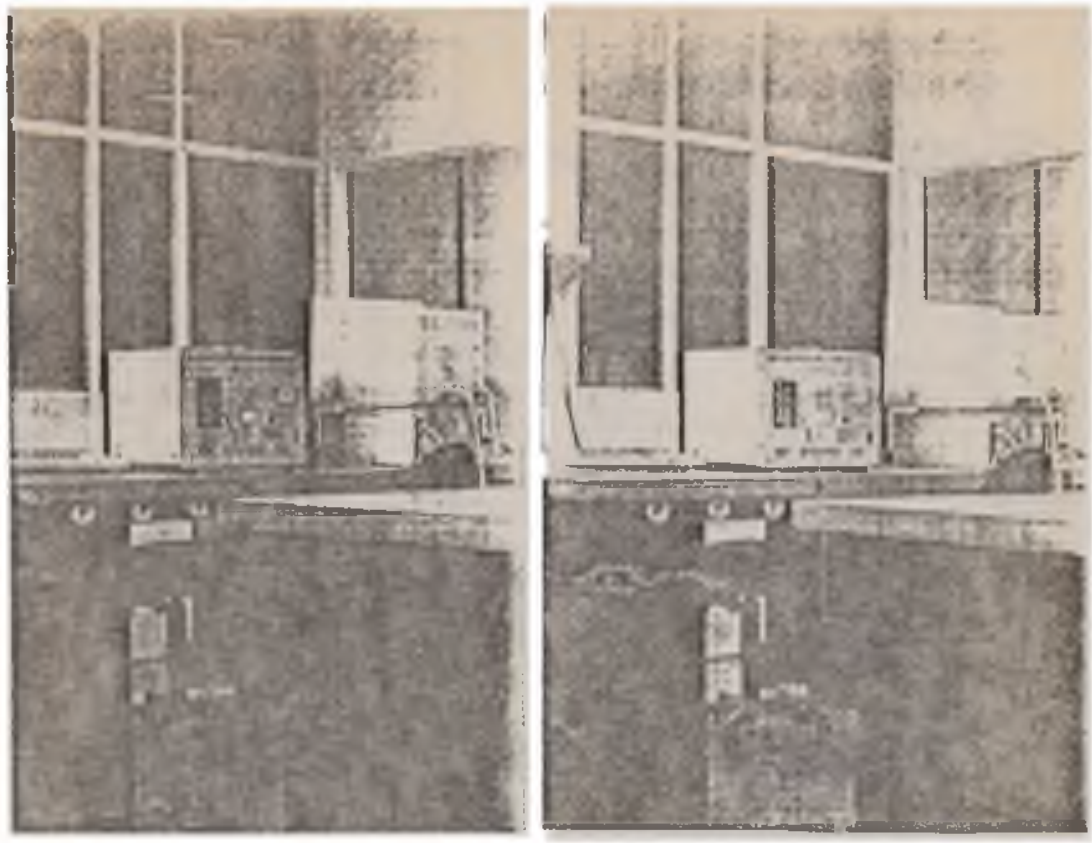


Figura 3.11: **Imágenes originales del laboratorio.**

3.3 El laboratorio.

3.3.1 Imágenes originales.

La Fig. 3.11 presenta las imágenes originales de una parte del Laboratorio de Procesamiento de Imágenes en la E.N.S.T de Bretagne. Las imágenes del laboratorio no fueron calibradas por lo cual la información de profundidad obtenida no tiene ningún significado físico.

3.3.2 Imagen gradiente.

La Fig. 3.12 presenta las imágenes gradiente obtenidas de las imágenes de la Fig. 3.11. El nivel de gris establecido para binarizar la imagen fue de 20.

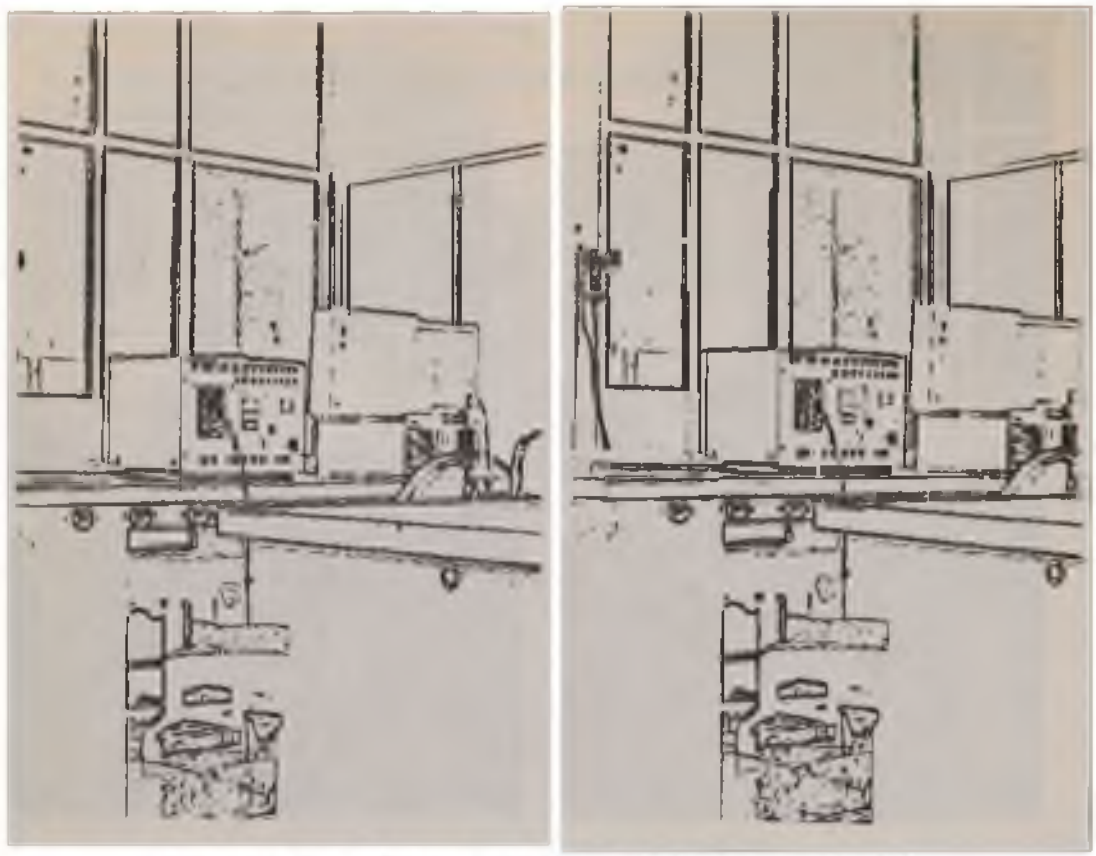


Figura 3.12: **Imágenes gradiente del laboratorio (1).**

Algunas observaciones se desprenden:

- Las partes traseras de la computadora y del monitor a la izquierda son zonas de alto detalle que no han sido muy bien definidas por el operador gradiente.

- Algunas partes, como el marco de la puerta, la computadora y las sillas presentan zonas de bajo contraste que no han sido muy bien definidas por el operador gradiente.

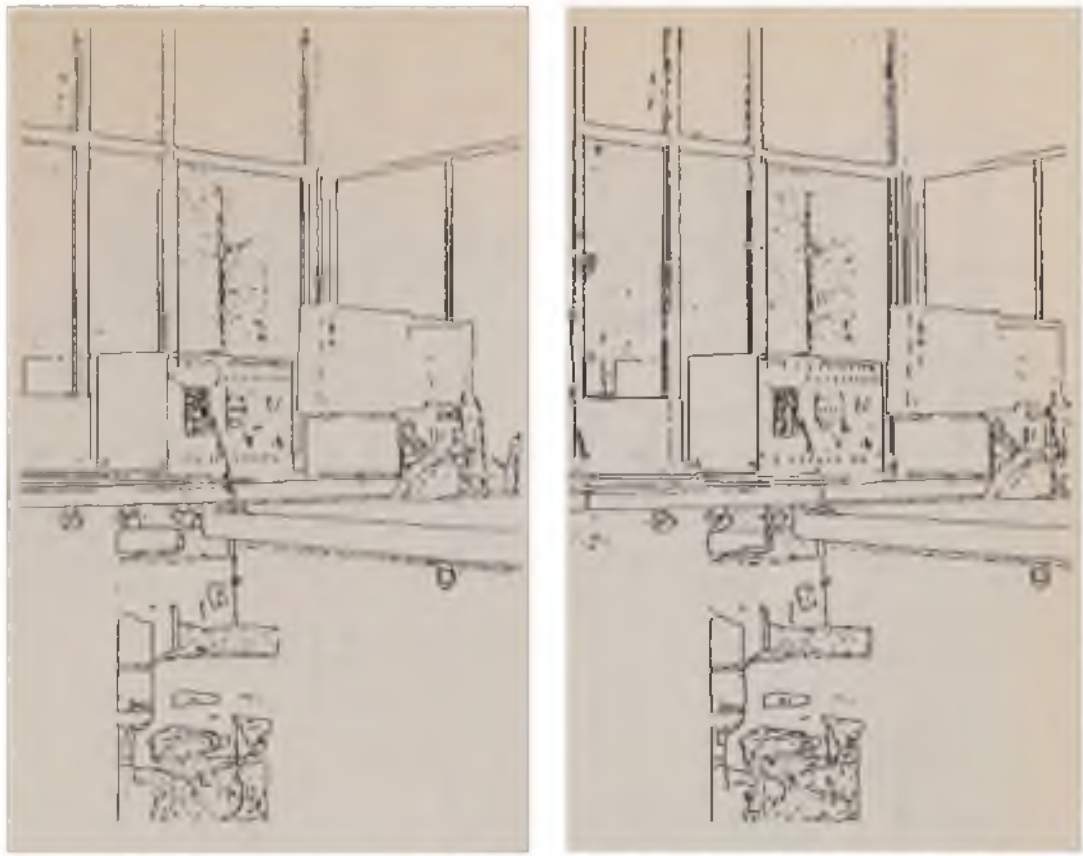


Figura 3.13: **Adelgazamiento de líneas a las imágenes del laboratorio.**

3.3.3 Adelgazamiento de líneas.

La Fig. 3.13 presenta el resultado de aplicar el algoritmo de adelgazamiento de líneas a las imágenes de la Fig. 3.12. El resultado obtenido son líneas con grosor de un pixel (*esqueleto unitario*).

3.3.4 Extracción de contornos.

La Fig. 3.14 presenta el resultado de aplicar el algoritmo de extracción de contornos a las imágenes de la Fig. 3.13. La condición de continuidad se cumple siempre y cuando no existan dos segmentos de recta cuya diferencia de pendiente sea mayor a 45 grados. En las imagenes de la Fig. 3.14 solo aparecen los segmentos de curva cuya longitud es mayor o igual a 25 pixeles,

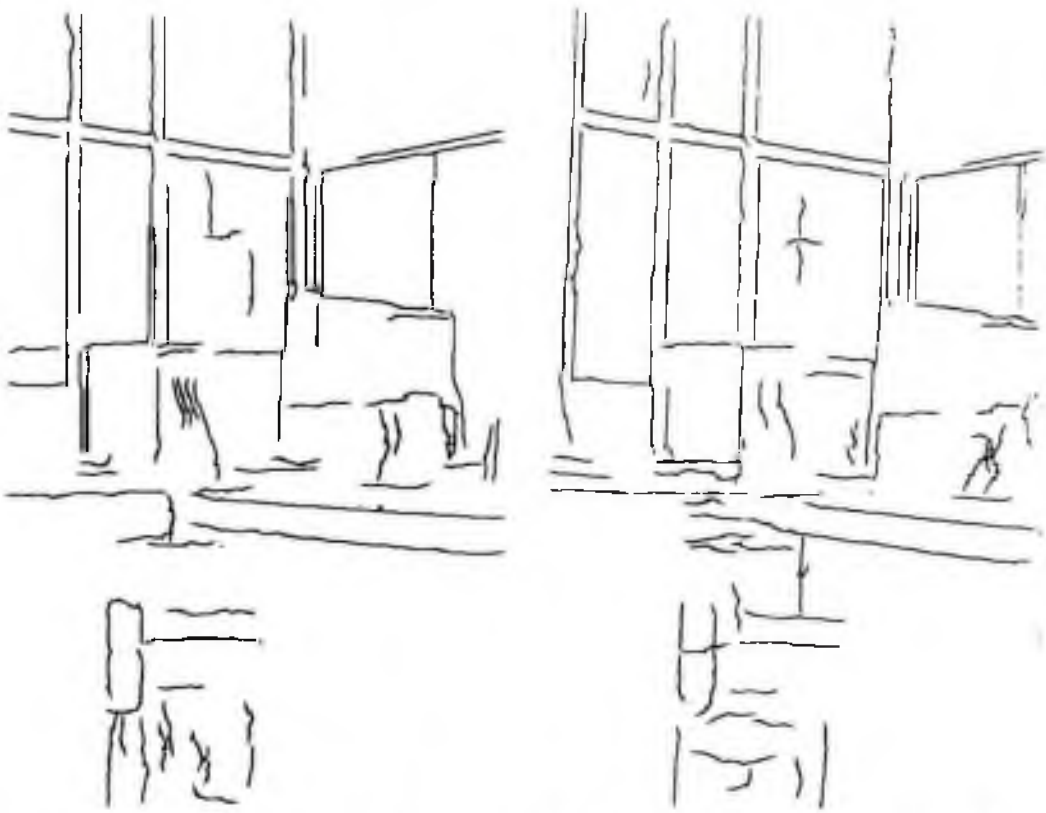


Figura 3.14: Extracción de contornos a las imágenes del laboratorio.

ello elimina bastante del ruido existente aunque le resta algo de comprensibilidad a la imagen.

En las imágenes de la Fig. 3.15 se etiquetan las curvas de las imágenes de la Fig. 3.14.

En las tablas 3.7 se presentan los descriptores de cada uno de los segmentos de curva registrados en la Fig. 3.15. La longitud de cada segmento debe ser multiplicada por 5 para obtener el valor en pixeles. La dirección está expresada en radianes y el resto de las coordenadas toma como origen la esquina superior izquierda de la imagen, con direcciones positivas hacia la derecha y hacia abajo.

Figura 3.15: **Etiquetación de las curvas extraídas de las imágenes del laboratorio.**

Tabla 3.7: **Descriptores de los segmentos de curva extraídos del laboratorio.**

curva	longitud	dirección	centroide		origen		final	
			x	y	x	y	x	y
1	20	1.393665	131.46	68.50	119	13	144	160
2	25	1.381087	141.60	104.80	129	12	150	195
3	8	0.436126	414.50	72.67	382	29	448	38
4	43	1.636669	478.57	167.80	478	13	439	321
5	43	1.615376	361.23	167.80	510	13	440	321
6	7	1.698633	417.86	37.28	389	37	443	35
7	5	1.339658	428.75	58.00	429	33	430	77
8	16	1.475413	347.50	61.35	282	71	414	54
9	8	1.378169	394.67	84.27	375	77	424	70
10	9	1.099863	394.50	72.12	383	78	426	78
11	8	1.345184	291.67	98.50	283	67	285	130
12	19	1.694983	379.31	140.00	288	70	284	219
13	20	0.784330	318.60	82.15	137	83	284	85
14	18	1.479862	368.50	115.78	384	79	373	154
15	9	1.538894	449.77	120.00	445	93	448	158
16	5	1.422217	400.00	117.19	379	121	431	131
17	8	1.768783	416.00	130.90	375	133	443	138
18	8	1.338752	373.25	183.50	375	162	375	205
19	15	0.215878	318.06	191.13	180	190	273	191
20	5	1.313644	308.00	198.19	275	201	321	198
21	13	0.000000	318.00	198.80	180	198	287	198
22	6	1.848745	478.50	214.50	425	192	424	243
23	11	1.581041	434.50	236.00	432	194	434	278
24	19	0.872984	308.60	266.77	134	308	387	307
25	8	0.000000	38.50	309.80	38	309	91	309
26	17	1.618083	80.00	250.52	17	225	142	250
27	16	0.216701	319.50	218.81	186	251	279	280
28	33	0.133824	293.50	227.50	180	225	411	234
29	17	1.618894	138.29	302.80	144	319	130	343
30	17	1.368638	158.67	303.80	158	258	144	340
31	9	1.541589	325.00	292.00	320	277	328	297
32	16	1.320171	294.00	305.50	347	267	508	666
33	5	1.447369	671.00	319.75	650	305	695	302
34	5	0.135113	712.00	288.79	691	258	732	282
35	9	1.503399	281.50	259.17	230	292	293	303
36	12	1.605458	568.41	309.50	569	254	567	343
37	7	1.518111	452.57	285.00	450	257	454	313
38	5	1.528629	317.60	286.79	316	267	296	307
39	7	1.854374	743.43	289.42	746	273	729	319
40	13	2.762319	399.00	302.36	349	316	447	283
41	8	1.892762	187.50	300.37	158	304	319	300
42	8	1.547151	505.11	328.00	507	293	503	363
43	5	1.268857	713.40	300.60	695	303	735	307
44	5	1.865952	697.28	310.20	677	317	718	311
45	8	1.345374	639.00	337.50	638	303	640	352
46	8	1.467348	317.83	330.50	317	308	321	355
47	5	2.341363	355.00	319.00	348	324	380	313
48	6	2.333679	371.50	324.33	347	333	398	321
49	5	1.437491	324.60	334.00	324	313	328	355
50	6	2.499378	373.50	330.00	348	330	398	339
51	6	1.374427	479.50	337.18	456	344	384	341
52	7	1.841664	677.28	345.67	656	346	705	337
53	20	0.533748	343.50	342.50	170	344	317	343
54	6	1.907118	685.50	344.50	671	350	730	340
55	7	1.794104	498.83	365.00	491	337	503	383
56	16	0.351503	73.50	348.12	12	348	132	347
57	13	0.377520	394.50	352.41	339	353	436	353
58	21	1.557998	456.38	435.00	458	343	480	497
59	16	1.356309	74.50	338.42	15	339	134	334
60	28	0.485262	225.50	357.19	132	342	331	338
61	11	1.989412	759.00	364.77	666	348	752	360
62	15	0.252059	314.00	362.00	259	362	370	364
63	29	0.946399	89.62	414.00	126	357	14	437
64	9	1.671870	135.00	393.00	136	358	131	425
65	12	1.349939	642.66	373.69	560	372	667	402
66	5	0.525421	579.00	369.00	558	372	600	369
67	8	1.478730	316.17	394.50	317	363	323	436
68	17	0.973700	592.15	396.79	565	367	651	397
69	5	1.221784	684.00	385.39	683	368	705	382
70	12	0.802466	787.50	398.51	662	391	753	399
71	5	1.563287	439.60	418.00	438	389	440	431
72	5	1.244881	431.00	409.50	419	398	436	429
73	13	0.750673	372.80	419.61	333	426	421	435
74	13	0.436120	372.00	438.61	333	429	421	439
75	27	0.143830	313.50	427.86	133	428	294	438
76	14	1.299614	94.50	437.42	12	441	117	438
77	33	0.946063	241.00	440.38	137	440	380	440
78	7	1.446568	112.00	466.00	112	438	109	494
79	7	1.566049	357.51	467.00	356	439	356	495
80	6	1.469779	323.50	469.50	322	441	328	494
81	9	1.729896	120.66	478.50	129	446	133	495

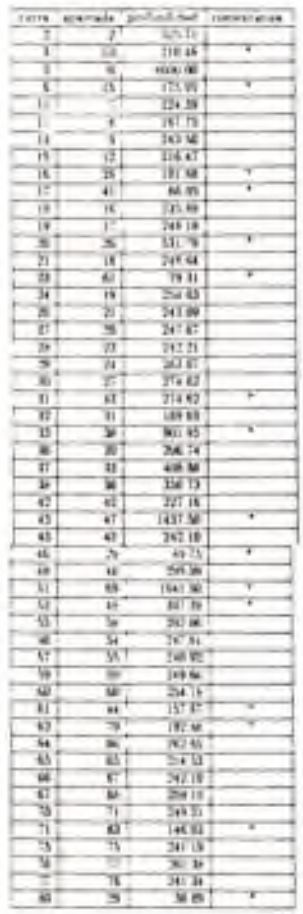
curva	longitud	dirección	centroide		origen		final	
			x	y	x	y	x	y
1	13	1.284423	134.16	62.00	127	13	144	111
2	20	1.365465	145.95	86.50	135	13	161	159
3	11	0.779163	329.00	25.56	290	29	374	30
4	13	1.636484	471.38	132.00	477	13	467	251
5	7	1.569736	377.00	35.57	369	34	403	31
6	29	1.654567	496.20	165.00	507	25	486	305
7	8	1.357106	253.12	57.50	294	36	293	90
8	17	1.620989	281.57	91.00	289	39	274	154
9	19	1.155622	308.00	47.36	138	45	278	48
10	6	1.439969	363.20	68.16	363	43	371	90
11	6	0.320540	163.50	52.00	141	52	190	52
12	6	1.561946	450.37	77.50	449	48	450	109
13	9	0.941334	413.50	66.32	379	59	448	78
14	7	2.439452	407.57	74.85	393	93	428	71
15	7	0.703743	430.14	83.57	395	78	444	90
16	8	1.402827	374.83	154.30	373	122	379	188
17	15	0.408838	217.00	154.88	181	155	273	156
18	14	0.000000	223.50	161.00	171	161	276	161
19	16	0.398845	211.50	170.56	157	166	271	171
20	16	0.326743	213.50	181.66	167	184	280	177
21	19	1.307812	90.00	187.50	18	182	130	183
22	8	0.431304	428.50	194.95	271	182	442	187
23	38	0.332617	289.50	185.57	152	191	425	134
24	18	1.720787	138.81	249.50	148	183	135	314
25	7	1.576035	436.80	214.86	435	186	425	242
26	6	1.729638	326.80	212.80	329	188	326	237
27	16	1.683915	151.50	249.50	158	190	145	309
28	6	1.949516	402.50	287.50	380	305	438	285
29	7	1.361165	341.00	229.00	342	206	344	256
30	13	1.645895	569.48	350.00	570	208	568	494
31	13	1.396434	594.12	258.00	591	259	595	307
32	11	1.127768	706.00	270.90	664	213	740	223
33	8	1.494773	433.25	262.50	432	231	438	294
34	5	1.294536	718.00	242.50	697	247	739	242
35	6	1.630179	541.50	248.83	517	244	566	269
36	5	1.494013	230.40	261.00	272	240	227	279
37	5	0.081464	511.00	250.60	490	250	531	253
38	5	1.621878	669.79	270.00	671	249	669	291
39	5	1.843468	198.00	262.60	177	264	219	264
40	5	1.834869	241.00	362.20	239	366	362	384
41	10	2.065536	365.79	300.20	348	279	474	263
42	8	1.564549	503.50	297.50	503	256	504	319
43	8	1.637259	319.12	293.50	316	304	315	335
44	11	1.389968	704.09	326.05	699	304	700	348
45	6	1.324533	640.50	289.50	638	263	641	314
46	6	0.015798	372.50	293.86	348	306	397	308
47	5	0.743811	727.20	294.20	710	290	733	312
48	6	1.216219	862.66	310.50	661	306	673	334
49	28	1.567436	430.57	394.50	437	293	463	496
50	17	0.130968	211.80	304.25	148	303	274	304
51	9	1.518787	497.86	335.00	453	294	503	364
52	5	0.099065	293.00	307.00	274	300	316	300
53	6	1.485344	582.16	308.50	549	315	546	387
54	15	0.000000	69.20	309.00	15	309	125	309
55	13	0.458838	368.00	314.33	319	313	417	309
56	6	0.546291	425.50	313.16	403	315	450	317
57	7	1.585817	479.47	323.00	480	304	479	360
58	6	1.324661	509.50	323.50	511	305	509	356
59	17	1.475317	75.80	321.50	13	330	138	315
60	24	1.681927	221.50	321.00	134	324	309	321
61	11	1.364562	429.00	312.00	430	318	431	394
62	14	1.571261	440.25	354.54	442	314	441	396
63	6	1.277524	602.00	339.00	597	314	607	362
64	9	1.702946	172.00	354.00	136	321	118	391
65	18	1.445877	627.29	330.75	599	330	667	347
66	9	1.697110	135.60	354.00	138	323	133	391
67	5	0.520441	580.00	330.00	539	334	601	330
68	5	1.516331	319.55	359.00	318	334	323	394
69	5	1.534418	455.00	346.00	455	333	456	367
70	6	1.494785	632.14	360.63	608	367	655	354
71	10	0.904832	712.50	360.00	675	362	752	361
72	6	0.000000	581.50	367.50	556	362	607	363
73	9	0.644436	392.50	382.00	358	381	427	376
74	10	0.625342	75.50	389.25	37	390	116	390
75	25	0.300000	259.00	389.75	131	390	379	390
76	41	0.751249	278.80	402.76	121	403	425	394
77	13	1.836566	68.80	403.25	19	393	117	403
78	10	1.569390	339.50	429.50	339	441	338	478
79	23	0.954081	65.40	461.85	116	462	18	477
80	10	1.733965	136.80	440.50	132	402	121	479
81	5	1.912649	77.80	414.60	56	415	98	419
82	8	1.607357	445.50	463.50	447	434	445	497
83	5	1.611341	432.00	477.00	433	457	431	494
84	22	0.568344	243.00	477.84	139	478	367	478
85	57	0.959033	315.00	486.13	17	486	418	475

3.3.5 Apareamiento.

Los segmentos de curva que aparecen en la tabla 3.7 fueron sometidas al proceso de correspondencia. Los resultados de la primera etapa aparecen en las tablas 3.8; en la primera columna aparece la etiqueta del segmento de curva de la primera imagen, en la segunda columna, la etiqueta del segmento de curva de la segunda imagen y en la tercera la profundidad, que no expresa las medidas reales.

La segunda etapa no se ejecutó, ya que se consideró que el número de correspondencias sería mínimo en comparación del esfuerzo computacional.

Tabla 3.8: **Resultados de la correspondencia en el laboratorio. Primera etapa.**



Capítulo 4

Conclusiones.

En este trabajo presentamos los fundamentos, la construcción y la experimentación de un modelo de estereovisión para encontrar la correspondencia entre segmentos de curva utilizando descriptores como primitiva. En el mod elo propuesto no se incluyó la solución al problema de la reconstrucción tridimensional.

En la primera parte del reporte se pretendío dar una panorámica global del área de estereovisión; ésto incluyó los modelos geométricos utilizados, la caracterización de imágenes y los trabajos en los que se fundó nuestro modelo fueron presentados. En la segunda parte el modelo propuesto fué desarrollado. En la tercera parte aparecieron los resultados de la experimentación con el modelo de correspondencia. Se ha agregado un apéndice que contiene la documentación de los programas para computadora escritos.

Como resultado de este trabajo podemos concluir los siguientes logros:

- El modelo fue probado en una amplia variedad de ambientes, algunos de los cuales se presentaron en la etapa de experimentación, y en todos ellos presentó algún tipo de exito.

- Para encontrar la correspondencia se utilizan como primitivas segmentos de curva. Lo cual le permite elevar el nivel de abstracción al cual se manejan las correspondencias (en comparación con la elección de líneas rectas o puntos) y reducir los tiempos de procesamiento.

- En cuanto los contornos están bien definidos y no existe mucho detalle en la escena, los resultados del modelo son buenos.

El modelo desarrollado presentó las siguientes características negativas:

- Las zonas de alto detalle no son bien trabajadas.
- La segunda etapa del proceso de correspondencia resulta bastante cara en términos de tiempo de procesamiento y obtiene resultados que no se comparan, en términos de efectividad, con este costo.
- El modelo ha demostrado ser bastante sensible al ruido y a los efectos del detector de contornos. Pero este problema puede ser manejado en la etapa de preproceso y no necesariamente significa un defecto del modelo propuesto.
- No se manejan oclusiones de partes de la escena ni ocultaciones de unos objetos tras de otros.

El modelo desarrollado puede ser mejorado en los siguientes aspectos:

- La construcción de un modelo de representación tridimensional de superficies.
- La utilización de otro tipo de detector de contornos tal como el operador de Canny.
- Una descripción más completa de las relaciones estructurales en la escena. Créemos que pueden crearse relaciones que integren los segmentos de curva en objetos en base a su relación espacial en la imagen; es decir declarar cuales segmentos de curva se encuentran arriba, abajo, a la izquierda o derecha de otra. Esto permitiría efectuar la correspondencia de manera menos quantitativa y más cualitativa. Es decir, los procesos encaminados a reconocer la estructura tridimensional en una escena no permiten información incompleta; sin embargo se puede ir generando conocimiento adicional en la medida que avanzamos hacia el resultado final.
- Con el fin de generar un sistema de visión por computadora es conveniente integrar toda la información disponible en las imágenes, tal como las texturas, el flujo óptico, el movimiento, etc. Esto modelizaría con mayor fidelidad el desempeño de los ojos humanos, en donde parece ser, se ejecutan varios procesos en paralelo y varios resultados son obtenidos al mismo tiempo.

Apéndice A

La programación del modelo.

Se presentan el conjunto de programas utilizados para experimentar con el modelo de estereoscopía desarrollado en la primera parte de este documento. Los programas en lenguaje C que siguen están diseñados para facilitar la experimentación paso a paso, la verificación de resultados y la incorporación de nuevas ideas. No se pensó en ellos como un producto para un grupo de usuarios sino como una herramienta en la búsqueda de un resultado.

Un programa para computadora tiene una complejidad inherente que no puede ser reducido de un cierto nivel de dificultad. El propósito de esta parte del documento, es presentar los algoritmos utilizados de la manera más clara posible, de tal manera que se refleje como las distintas partes del programa se van uniendo y proporcionar las referencias cruzadas que conectan esas partes.

Utilizaremos las ideas de Donald E. Knuth [12] para documentar los programas. Los programas consisten de secciones numeradas: primero viene §1, después §2, y así en adelante. Se espera que cada sección esté diseñada para entenderse por si misma. Las referencias cruzadas indican como cada sección se relaciona con otras. De esta manera un programa puede ser visto como una red que consiste de pequeños nodos y conexiones entre nodos. El todo puede ser entendido comprendiendo cada una de las partes y la relación entre las partes.

Cada sección comienza con un comentario acerca del propósito de la sección o de alguna parte remarcable del programa. La sección termina con código de un programa en C. En medio de ellas pueden existir o no, una o más macro-definiciones. Es decir, cada sección tiene tres partes:

- Comentarios,
- Macro-Definiciones y
- Código de lenguaje C.

Las secciones son *nombradas* o *no nombradas*. El código en C en una sección nombrada comienza con "<Nombre de la sección>≡" y seguirá una porción de código que reemplazará su llamada. En el caso de una sección no nombrada, solo se presentará el código de lenguaje C.

En ocasiones, una sección nombrada es dividida; es el caso de las variables, constantes, y estructuras de datos, las cuales son presentadas en el momento que creemos más oportuno. Después de la primera, estas secciones aparecerán con la forma: "<Nombre de la sección>+ ≡", lo cual indica que su contenido debe ser pegado a la primera sección.

Para facilitar la lectura del código, se han tomado algunas convenciones. Las palabras reservadas como **for**, **while**, así como los procedimientos incluídos en la librería estandar como **fprintf**, **fclose**, aparecerán en negritas; los identificadores aparecerán en itálicas y los nombres de procedimientos en letra normal. Igualmente, algunos símbolos han sido reemplazados:

símbolo:	sustituye a:
←	=
=	==
≤	<=
≥	>=
≠	!=
∧	&&
∨	||
¬	!

Igualmente, algunas funciones como la raíz cuadrada, ha sido sustituida por el símbolo matemático $\sqrt{}$; el símbolo * ha sido sustituído por × cuando se refiere a una multiplicación y algunas operaciones del lenguaje C estandar como $i++$, han sido sustituídas por $i \leftarrow i + 1$.

A.1 Almacena imagen.

El programa **AlmacenaImagen** permite almacenar la porción de memoria principal, que corresponde a la imagen digital, en la memoria secundaria de la computadora. Con el fin de aprovechar el equipo computacional existente, la imagen se almacena en la memoria incorporada a la carta de adquisición de imágenes; ésto vuelve al programa dependiente del equipo, pero las imágenes son lo suficientemente grandes (768 columnas por 512 hileras) como para justificar la decisión. Como atenuante, podemos agregar que si la memoria del ordenador es lo suficientemente grande como para reservar un espacio de memoria de 384k palabras, sin problema, los cambios al programa son mínimos.

1. La descripción comienza mostrando las porciones principales de un programa en lenguaje C, cuyas componentes serán llenadas después. Por ejemplo, la porción del programa llamada <Las constantes 3> será reemplazada por una secuencia de declaración de constantes que comienza en §3.

   <Los archivos que se preprocesan 2>

   <Las constantes 3>

   <Las variables globales 4>

   <Rutina "SaveMemory" 5>

   <Rutina "main" 6>

2. Las rutinas para el manejo de la carta de adquisición de imágenes se encuentran en la librería "igselib.h". Rutinas tales como inicializar la carta de adquisición, digitalizar una imagen, escribir un punto en la memoria, leer un punto de la memoria, etc. En esta serie de programas solo se utiliza la rutina que nos indica la dirección de memoria en donde se encuentra la carta de adquisición de imágenes. Todas las rutinas que pudieran estar relacionadas con la tarjeta se hicieron siguiendo el manejo estandar de apuntadores que permite el lenguaje C.

   El archivo "stdio.h" contiene las rutinas de entrada y salida estandar en el lenguaje C. Para mayor información consultar la documentación del lenguaje.

<Los archivos que se preprocesan 2>≡

```
#include "igselib.h"
#include "stdio.h"
```

3. La imagen digital que maneja la carta de adquisición tiene dimensiones de 768 columnas × 512 hileras. Cada pixel está representado por 8 bits, es decir tenemos 256 diferentes niveles de gris.

<Las constantes 3>≡

```
#define numColumnas (768)
#define numHileras (512)
```

4. La variable *screen* es la dirección de memoria en donde comienza el almacenamiento de la imagen.

<Las variables globales 4>≡

```
Byte screen;
```

5. Esta rutina recupera los datos de la memoria principal y los almacena en un archivo.

<Rutina "SaveMemory" 5>≡

```
void savelmage(name)
char *name;
{
Byte *s,*r;
int j;
FILE *f;
        f← fopen(name, "w");
        s←screen;j←0;
        while(j≤ numColumnas-1) {
                r←s;
                fwrite(r,numHileras+1,1,f);
                s ← s + numColumnas; j ← j + 1;
        }
        fclose(f);
}
```

numColumnas: constante, §3 numHileras: constante, §3 screen: variable, §4

6. El programa principal hace uso de la rutina "addressVideoCard", definida en la librería "igselib.h" (ver §2), que nos indica la posición en memoria en donde se encuentra la imagen. La rutina pregunta el nombre que se le dará al archivo de salida.

<Rutina "main" 6>≡

```
main() {
char *name;
char buffer[20];
      addressVideoCard(& screen);
      name ← gets(& buffer);
      saveImage(buffer);
}
```

saveImage: procedimiento, §5 screen: variable, §4

A.2 Recupera imagen.

El programa **recuperaImagen** permite colocar, una imagen digital almacenada en un archivo, en la memoria principal del ordenador. Con el fin de aprovechar el equipo computacional existente, la imagen se almacena en el área reservada por la carta de adquisición de imágenes; ésto vuelve al programa dependiente del equipo, pero las imágenes son lo suficientemente grandes (768 columnas por 512 hileras) como para justificar la decisión. Como atenuante, podemos agregar que si la memoria del ordenador es lo suficientemente grande como para reservar un espacio de 384k palabras, sin problema, los cambios al programa son mínimos.

1. La descripción comienza mostrando las porciones principales de un programa en lenguaje C, cuyas componentes serán llenadas después. Por ejemplo, la porción del programa llamada <Las constantes 3> será reemplazada por una secuencia de declaraciones de constantes que comienza en §3.

<Los archivos que se preprocesan 2>

<Las constantes 3>

<Las variables globales 4>

<Rutina "retrieveMemory" 5>

<Rutina "main" 6>

2. Las rutinas para el manejo de la carta de adquisición de imágenes se encuentran en la librería "igselib.h". Rutinas tales como inicializar la carta de adquisición, digitalizar una imagen, escribir un punto en la memoria, leer un punto de la memoria, etc. En esta serie de programas solo se utiliza la rutina que nos indica la dirección de memoria en donde se encuentra la carta de adquisición de imágenes. Todas las rutinas que pudieran estar relacionadas con la tarjeta se hicieron siguiendo el manejo estandar de apuntadores que permite el lenguaje C.

   El archivo "stdio.h" contiene las rutinas de entrada y salida estandar en el lenguaje C. Para mayor información consultar la documentación del lenguaje.

   <Los archivos que se preprocesan 2>≡

   #include "igselib.h"

   #include "stdio.h"

3. La imagen digital que maneja la carta de adquisición tiene dimensiones de 768 columnas × 512 hileras. Cada pixel está representado por 8 bits, es decir tenemos 256 diferentes niveles de gris.

   <Las constantes 3>≡

   #define *numColumnas* (768)

   #define *numHileras* (512)

4. La variable *screen* indica la dirección de memoria en donde comienza el almacenamiento de la imagen.

   <Las variables globales 4>≡

   Byte *screen*;

5. Esta rutina recupera los datos de un archivo y los transfiere a la memoria principal de la computadora.

<Rutina "retrieveMemory" 5>≡

```
void retrieveImage(name)
char *name;
{
FILE *f;
Byte *s, *r;
int  j;
     f← fopen(name, "w");
     s ← screen;
     j ← 0;
     while (j ≤ numColumnas − 1) {
          r ← s;
          fread(r,numHileras + 1,1,f);
          s ← s + numColumnas;
          j ← j + 1;
     }
     fclose(f);
}
```

numColumnas: constante, §3 numHileras:constante, §3 screen: variable, §4

6. El programa principal hace uso de la rutina "addressVideoCard", definida en la librería "igselib.h" (ver §2), que nos indica la posición en memoria en donde se encuentra la imagen. La rutina pregunta el nombre del archivo que contiene la imagen.

<Rutina "main" 6>≡

```
main() {
char   *name;
char   buffer[20];
       addressVideoCard(& screen);
       name ← gets(& buffer);
       retrieveImage(buffer);
}
```

saveImage: procedimiento, §5 screen: variable, §4

A.3 Gradiente.

El programa **gradiente** se realizó con el fin de detectar los contornos sobre la imagen. La imagen es pasada por un filtro no-lineal que corresponde a una matriz de dimensiones 2×2. El filtro es conocido como "Roberts" y se define como la suma de los valores absolutos de las diferencias entre las diagonales de la matriz.

En las vecindades de los puntos del contorno, el índice de luminosidad de la luz cambia bruscamente; por ello los valores del filtro tienden a ser altos para puntos del contorno. El programa realiza una clasificación de los puntos de la imagen de acuerdo a si pertenecen o no al contorno. Una vez verificado el valor del filtro para un punto dado, se determina si pasa o no un umbral arbitrario. Si el valor del filtro pasa de un umbral, es etiquetado como CONTORNO; si nó, es etiquetado como FONDO.

El valor del umbral es empírico, y puede cambiar de imagen a imagen. Hasta donde nosotros sabemos, no existe una manera automática de determinar este valor.

1. La descripción comienza mostrando las porciones principales de un programa en lenguaje C, cuyas componentes serán llenadas después. Por ejemplo, la porción del programa llamada <Las constantes 3> será reemplazada por una secuencia de declaración de constantes que comienza en §3.

   <Los archivos que se preprocesan 2>

   <Las constantes 3>

   <Las variables globales 5>

   <Rutina "gradient" 6>

   <Rutina "main" 7>

2. Las rutinas para el manejo de la carta de adquisición de imágenes se encuentran en la librería "igselib.h". Son rutinas tales como inicializar la carta de adquisición, digitalizar una imagen, escribir un punto en la memoria, leer un punto de la memoria, etc. En esta serie de programas solo se utiliza la rutina que nos indica la dirección de memoria en donde se encuentra la carta de adquisición de imágenes. Todas las rutinas

que pudieran estar relacionadas con la tarjeta se hicieron siguiendo el manejo estandar de apuntadores que permite el lenguaje C.

El archivo "stdio.h" contiene las rutinas de entrada y salida estandar en el lenguaje C. Para mayor información consultar la documentación del lenguaje. <Los archivos que se preprocesan 2>

#include "igselib.h"

#include "stdio.h"

3. La imagen digital que maneja la carta de adquisición tiene dimensiones de 768 columnas ×512 hileras.

   Esta es la primera de varias secciones en donde las constantes están definidas.

   <Las constantes 3>≡

   #define *numColumnas* (768)

   #define *numHileras* (512)

   Ver también 4.

4. El valor para *umbral* es función de la imagen que se esté procesando, hasta donde nosotros sabemos no hay manera de determinar su valor por medios automáticos.

   El valor de *umbral* nos sirve para etiquetar a un punto $p(x,y)$ de la imagen de acuerdo a dos clases, CONTORNO y FONDO, de la siguiente manera:

$$p(x,y) = \begin{cases} \text{CONTORNO} & \text{si } p(x,y) \geq umbral \\ \text{FONDO} & \text{en otro caso.} \end{cases} \tag{A.1}$$

   <Las constantes 3> + ≡

   #define *umbral* (15)

   #define *contorno* (255)

   #define *fondo* (0)

5. La variable *screen* indica la dirección de memoria en donde comienza la imagen.

<Las variables globales 5>≡

```
Byte screen;
```

6. El valor del gradiente $f(x,y)$, para un punto $p(x,y)$, se determina de acuerdo al filtro "Roberts" por:

$$f(x) = | p(x,y) - p(x+1,y+1) | + | p(x,y+1) - p(x+1,y) | \quad \text{(A.2)}$$

<Rutina "gradient" 6>≡

```
void gradient() {
int   i,j,nivel,a1, a2, a3, a4;
      for (j ← 0;j < numColumnas − 2;j ← j + 1) {
          for (i ← 0;i < numHileras − 2;i ← i + 1) {
              a1 ← *(screen + numColumnas × j + i);
              a2 ← *(screen + numColumnas × j + (i + 1));
              a3 ← *(screen + numColumnas × (j + 1) + i);
              a4 ← *(screen + numColumnas × (j + 1) + (i + 1));
              nivel ←| a1 − a4 | + | a3 − a2 |;
              if (nivel ≥ umbral) nivel ← contorno;
              else nivel ← fondo;
              *(screen + numColumnas × j + i) ← nivel;
          }
      }
}
```

numColumnas:constante, §3 numHileras:constante, §3 screen:variable,§5
umbral:constante, §4 contorno:constante, §4 fondo:constante, §4

7. El programa principal hace uso de la rutina **addressVideoCard**, definida en la librería "igselib.h" (ver §2), que nos indica la posición en memoria en donde se encuentra la imagen.

<Rutina "main" 7>≡

```
main() {
      addressVideoCard(&screen);
      gradient();
}
```

gradient:procedimiento, §6 screen:variable,§5

A.4 Adelgazamiento de líneas.

El programa **AdelgazaLineas** toma como entrada la imagen resultante de la aplicación del filtro gradiente y entrega como resultado una imagen representada por su esqueleto unitario. El esqueleto unitario es una estructura formada de líneas con anchura de un solo pixel. Este algoritmo forma parte del tratamiento a las imágenes antes de la aplicación de los algoritmos de correspondencia.

El algoritmo se desarrolla en tres etapas; las dos primeras corresponden al algoritmo de Zhang-Suen y la tercera corresponde a la mejora que presentó Juan Humberto SOSSA en [10].

1. La descripción comienza mostrando las porciones principales de un programa en lenguaje C, cuyas componentes serán llenadas después. Por ejemplo, la porción del programa llamada <Constantes 3> será reemplazada por una secuencia de declaraciones de constantes que comienza en §3.

   <Archivos que se preprocesan 2>

   <Constantes 3>

   <Variables globales 4>

   <Rutina de erosión 8>

   <Rutina de pos-proceso 22>

   <Rutina principal 23>

2. Las rutinas para el manejo de la carta de adquisición de imágenes se encuentran en la librería "igselib.h". Son rutinas tales como inicializar la carta de adquisición, digitalizar una imagen, escribir un punto en la memoria, leer un punto en la memoria, etc. En esta serie de programas solo se utiliza la rutina que nos indica la dirección de memoria en donde se encuentra la carta de adquisición de imágenes, todas las rutinas que pudieran estar relacionadas con la tarjeta, se hicieron siguiendo el manejo estandar de apuntadores que permite el lenguaje C.

   El archivo "stdio.h" contiene las rutinas de entrada y salida estandar en el lenguaje C. Para mayor información consultar la documentación del lenguaje.

<Archivos que se preprocesan 2>≡

#include "igselib.h"

#include "stdio.h"

3. La imagen digital que maneja la carta de adquisición tiene dimensiones de 768 columnas ×512 hileras.

   Esta es la primera de varias secciones en donde las constantes están definidas.

   <Constantes 3>≡

   #define *numColumnas* (768)

   #define *numHileras*,(512)

   Ver también 5 y 6.

4. La variable *screen* es la dirección de memoria en donde comienza el almacenamiento de la imagen.

   Esta es la primera de varias secciones en donde las variables están definidas.

   <Variables globales 4>≡

   Byte *screen*;

   Ver también 7.

5. El programa **gradiente** convirtió nuestra imagen en dos clases: CONTORNO y FONDO. Esto es necesario porque el algoritmo de erosión trabaja con dos clases. El valor de *umbral* nos sirve para etiquetar a un pixel p de la imagen, como perteneciente a una de las dos clases de la siguiente manera:

$$p = \begin{cases} \text{CONTORNO} & \text{si } p \geq umbral \\ \text{FONDO} & \text{si no} \end{cases} \tag{A.3}$$

   <Constantes 3> + ≡

   #define *umbral* (100)

6. Al final del algoritmo de erosión, un cierto número de pixeles del borde de la imagen debe ser retirado para no interferir con el algoritmo de seguimiento de contornos. La constante *minimo* es aprovechada activamente, para no calcular operaciones en el área que de cualquier manera será removida.

   <Constantes 3> + ≡

   #define *minimo* (10)

7. El valor de la variable *exito*, representa el número de puntos que fueron erosionados en la última iteración del algoritmo.

   <Variables globales 4> + ≡

   int exito;

8. El algoritmo de Zhang y Suen con la mejora de Sossa aplicado sobre una imagen binaria garantiza un esqueleto unitario. El algoritmo se desarrolla en tres etapas.

   <Rutina de erosión 8>≡

   void erosion()

   {**int** *i,j,densidad,patrones,nivel*;

   <Primera etapa 9>

   <Segunda etapa 10>

   <Tercera etapa 11>

   }

9. La imagen entera es examinada. Para cada pixel P_1 se determina si pertenece a la clase CONTORNO; si es así, el pixel $p(x, y)$ es erosionado si su vecindad (ver Fig. 2.3) cumple con las siguientes cuatro propiedades:

   Propiedad A. La suma de los pixeles en la vecindad de P_1 que pertenecen a la clase CONTORNO es mayor que dos y menor que seis.

   Propiedad B. El número de patrones 01 en el conjunto ordenado $P_2, P_3, ..., P_9$ es igual a uno.

Propiedad C. Alguno de los pixeles P_2, P_4, P_6, pertenece a la clase FONDO.

Propiedad D. Alguno de los pixeles P_4, P_6, P_8, pertenece a la clase FONDO.

<Primera etapa 9>≡

```
for (i ← minimo;i ≤ numHileras − minimo;i ← i + 1){
    for (j ← minimo;j ≤ numColumnas − minimo;j ← j + 1) {
        if ( *(screen + numColumnas × i + j) > umbral) {
            densidad ←propiedadA(j,i);
            if ((densidad ≥ 2) ∧ (densidad ≤ 6)) {
                patrones ←propiedadB(j,i);
                if (patrones = 1) {
                    if ((propiedadC(j,i)=0)∧(propiedadD(j,i)=0)) {
                        *(screen + numColumnas × i + j) ← 0;
                        exito ← exito + 1;
                    }
                }
            }
        }
    }
}
```

densidad:variable,§8 — exito:variable,§7
i:variable,§8 — j:variable,§8
minimo:constante, §5 — numColumnas:constante,§3
numHileras:constante,§3 — patrones:variable,§8
propiedadA:procedimiento, §12 — propiedadB:procedimiento, §13
propiedadC:procedimiento, §14 — propiedadD:procedimiento, §15
screen:variable,§4 — umbral:constante,§6

10. La imagen entera es examinada. Para cada pixel P_1 se determina si pertenece a la clase CONTORNO; si es así, el pixel $p(x, y)$ es erosionado si su vecindad (ver Fig. 2.3) cumple con las siguientes cuatro propiedades:

Propiedad A. La suma de los pixeles en la vecindad de P_1 que pertenecen a la clase CONTORNO es mayor que dos y menor que seis.

Propiedad B. El número de patrones 01 en el conjunto ordenado $P_2, P_3, ..., P_9$ es igual a uno.

Propiedad C'. Alguno de los pixeles P_2, P_4, P_8, pertenece a la clase FONDO.

Propiedad D'. Alguno de los pixeles P_2, P_6, P_8, pertenece a la clase FONDO.

<Segunda etapa 10>≡

```
for (i ← minimo;i ≤ numHileras − minimo;i ← i + 1){
    for (j ← minimo;j ≤ numColumnas − minimo;j ← j + 1) {
        if ( *(screen+numColumnas×i+j)¿umbral) {
            densidad←propiedadA(j,i);
            if ((densidad ≥ 2) ∧ (densidad ≤ 6)) {
                patrones←propiedadB(j,i);
                if (patrones = 1) {
                    if ((propiedadC'(j,i)=0) ∧ (propiedadD'(j,i)=0)) {
                        *(screen + numColumnas × i + j) ← 0;
                        exito ← exito + 1;
                    }
                }
            }
        }
    }
}
```

densidad:variable,§8
i:variable,§8
minimo:constante, §5
numHileras:constante,§3
propiedadA:procedimiento, §12
propiedadC':procedimiento, §16
screen:variable,§4
exito:variable,§7
j:variable,§8
numColumnas:constante,§3
patrones:variable,§8
propiedadB:procedimiento, §13
propiedadD':procedimiento, §17
umbral:constante,§6

11. La imagen entera es examinada. Para cada pixel P_1 se determina si pertenece a la clase CONTORNO; si es así, el pixel $p(x, y)$ es erosionado si su vecindad (ver Fig. 2.3) cumple con alguna de las siguientes cuatro propiedades:

Propiedad 1. Todos los pixeles $\overline{P_9}, P_4, P_6$, pertenece a la clase CONTORNO.

Propiedad 2. Todos los pixeles $\overline{P_3}, P_6, P_8$, pertenece a la clase CONTORNO.

Propiedad 3. Todos los pixeles $\overline{P_5}, P_8, P_2$, pertenece a la clase CONTORNO.

Propiedad 4. Todos los pixeles $\overline{P_7}, P_2, P_4$, pertenece a la clase CONTORNO.

<Tercera etapa 11>≡

```
for (i ← minimo;i ≤ numHileras − minimo;i ← i + 1){
   for (j ← minimo;j ≤ numColumnas − minimo;j ← j + 1) {
      if ( *(screen + numColumnas × i + j) > umbral) {
         if (propiedad1(j,i)≠ 0) {
            *(screen + numColumnas × i + j) ← 0;
            exito ← exito + 1;
         }
         else{if (propiedad2(j,i)≠ 0) {
               *(screen + numColumnas × i + j) ← 0;
               exito ← exito + 1;
            }
            else{if (propiedad3(j,i)≠ 0) {
                  *(screen + numColumnas × i + j) ← 0;
                  exito ← exito + 1;
               }
               else {if(propiedad4(j,i)≠ 0){
                     *(screen + numColumnas × i + j) ← 0;
                     exito ← exito + 1;
                  }
               }
            }
         }
      }
   }
}
```

```
}
```

exito:variable,§7 *i*:variable,§8
j:variable,§8 *minimo*:constante, §5
numColumnas:constante,§3 *numHileras*:constante,§3
propiedad1:procedimiento, §18 propiedad2:procedimiento, §19
propiedad3:procedimiento, §20 propiedad4:procedimiento, §21
screen:variable,§4 *umbral*:constante,§6

12. Verificamos el número de pixeles en la vecindad 3×3 de P_1 que pertenece a la clase CONTORNO:

```
int propiedadA(i,j)
int i,j;
{ int x,y,result,nivel;
      result ← 0;
      for (x ← i − 1;x ≤ i + 1;x ← x + 1){
          for (y ← j − 1;y ≤ j + 1;y ← y + 1){
              if (¬((x = i) ∧ (y = j))) {
                  nivel ← *(screen + numColumnas × y + x);
                  if (nivel > umbral) result ← result + 1;
              }
          }
      }
      return (result);
}
```

numColumnas:constante,§3 *screen*:variable,§4 *umbral*:constante,§6

13. Contabilizamos el número de patrones 01 en la vecindad de P_1:

```
int propiedadB(i,j)
int i,j;
{ int v[10],x,y,next,nivel;
      v[2] ← *(screen + numColumnas × (j − 1) + i);
      v[3] ← *(screen + numColumnas × (j − 1) + (i + 1));
      v[4] ← *(screen + numColumnas × j + (i + 1));
      v[5] ← *(screen + numColumnas × (j + 1) + (i + 1));
      v[6] ← *(screen + numColumnas × (j + 1) + i);
```

```
    v[7] ← *(screen + numColumnas × (j + 1) + (i − 1));
    v[8] ← *(screen + numColumnas × j + (i − 1));
    v[9] ← *(screen + numColumnas × (j − 1) + (i − 1));
    next ← 0;
    for (x ← 2; x < 9; x ← x + 1){
        if ((v[x] < umbral) ∧ (v[x + 1] > umbral)) next ← next + 1;
    }
    if ((v[9] < umbral) ∧ (v[2] > umbral)) next ← next + 1;
    return (next);
}
```

numColumnas:constante,§3 screen:variable,§4 umbral:constante,§6

14. Verificamos que por lo menos uno de los valores P_2, P_4, P_6 sea cero:

```
int propiedadC(x,y)
int x,y;
{int p2, p4, p6;
    p2 ← *(screen + numColumnas × (y − 1) + x);
    p4 ← *(screen + numColumnas × y + (x + 1));
    p6 ← *(screen + numColumnas × (y + 1) + x);
    return (p2 × p4 × p6);
}
```

numColumnas:constante,§3 screen:variable,§4

15. Verificamos que por lo menos uno de los valores P_4, P_6, P_8 sea cero:

```
int propiedadD(x,y)
int x,y;
{int p4, p6, p8;
    p4 ← *(screen + numColumnas × y + (x + 1));
    p6 ← *(screen + numColumnas × (y + 1) + x);
    p8 ← *(screen + numColumnas × y + (x − 1));
    return (p4 × p6 × p8);
}
```

numColumnas:constante,§3 screen:variable,§4

16. Verificamos que por lo menos uno de los valores P_2, P_4, P_8 sea cero:

```
int propiedadC'(x,y)
int x,y;
{int p2, p4, p8;
    p2 ← *(screen + numColumnas × (y − 1) + x);
    p4 ← *(screen + numColumnas × y + (x + 1));
    p8 ← *(screen + numColumnas × y + (x − 1));
    return (p2 × p4 × p8);
}
```

numColumnas:constante,§3 screen:variable,§4

17. Verificamos que por lo menos uno de los valores P_2, P_6, P_8 sea cero:

```
int propiedadD'(x,y)
int x,y;
{int p2, p6, p8;
    p2 ← *(screen + numColumnas × (y − 1) + x);
    p6 ← *(screen + numColumnas × (y + 1) + x);
    p8 ← *(screen + numColumnas × y + (x − 1));
    return (p2 × p6 × p8);
}
```

numColumnas:constante,§3 screen:variable,§4

18. Verificamos que todos los valores $\overline{P_9}, P_4, P_6$ sean uno:

```
int propiedad1(x,y)
int x,y;
{int p9', p4, p6;
    p9' ← *(screen + numColumnas × (y - 1) + (x − 1));
    p9' ← (p9' > umbral)?0 : 1;
    p4 ← *(screen + numColumnas × y + (x + 1));
    p6 ← *(screen + numColumnas × (y + 1) + x);
    return (p9' × p4 × p6);
}
```

numColumnas:constante,§3 *screen*:variable,§4 *umbral*:constante,§6

19. Verificamos que todos los valores $\overline{P_3}, P_6, P_8$ sean uno:

```
int propiedad2(x,y)
int x,y;
{int p3', p6, p8;
     p3' ← *(screen + numColumnas × (y − 1) + (x + 1));
     p3' ← (p3' > umbral)?0 : 1;
     p6 ← *(screen + numColumnas × (y + 1) + x);
     p8 ← *(screen + numColumnas × y + (x − 1));
     return (p3' × p6 × p8);
}
```

numColumnas:constante,§3 *screen*:variable,§4 *umbral*:constante,§6

20. Verificamos que todos los valores $\overline{P_5}, P_8, P_2$ sean uno:

```
int propiedad3(x,y)
int x,y;
{int p5', p8, p2;
     p5' ← *(screen + numColumnas × (y + 1) + (x + 1));
     p5' ← (p5' > umbral)?0 : 1;
     p8 ← *(screen + numColumnas × y + (x − 1));
     p2 ← *(screen + numColumnas × (y − 1) + x);
     return (p5' × p8 × p2);
}
```

numColumnas:constante,§3 *screen*:variable,§4 *umbral*:constante,§6

21. Verificamos que todos los valores $\overline{P_7}, P_2, P_4$ sean uno:

```
int propiedad4(x,y)
int x,y;
{int p7', p2, p4;
     p7' ← *(screen + numColumnas × (y + 1) + (x − 1));
     p7' ← (p7' > umbral)?0 : 1;
     p2 ← *(screen + numColumnas × (y − 1) + x);
```

```
    p4 ← *(screen + numColumnas × y + (x + 1));
    return (p7 × p2 × p4);
}
```

numColumnas:constante,§3 screen:variable,§4 umbralconstante,§6

22. Con el fin no interferir con el algoritmo de seguimiento de contornos (que es la siguiente etapa en el proceso) un conjunto de pixeles del borde de la imagen es pasado a la clase FONDO.

<Rutina de pos-proceso 22>≡

```
void valida()
{int i,j;
    for (i ← 0; i ≤ numHileras − 1;i ← i + 1){
        for (j ← 0;j < 20;j ← j + 1){
            *(screen + numColumnas × i + j) ← 0;
            *(screen + numColumnas × i + (numColumnas − j − 1)) ←
        }
    }
    for (i ← 0; i ≤ numColumnas − 1;i ← i + 1){
        for (j ← 0;j < 20;j ← j + 1){
            *(screen + numColumnas × j + i) ← 0;
            *(screen + numColumnas × (numHileras − j − 1) + i) ← 0;
        }
    }
}
```

numColumnas:constante,§3 numHileras:constante,§3 screen:variable,§4

23. El programa principal hace uso de la rutina **addressVideoCard**, definida en la librería "igselib.h" (ver §2), que nos indica la posición en memoria en donde se encuentra definida la imagen.

El número de iteraciones del algoritmo de erosión depende de la imagen original. El proceso se continua mientras algún pixel haya sido erosionado por la iteración anterior. Al final una rutina convierte en la clase FONDO los bordes de la imagen con el fin de no interferir en algunos procesos del algoritmo de seguimiento de contornos.

<Rutina principal 23>≡

```
main(){
        initVideoCard(&screen);
        exito ← 1;
        while (exito > 0)
                erosion();
        valida();
}
```

erosion:procedimiento, §8 *exito*: variable, §7 *screen*:variable,§4
valida:procedimiento, §22 *screen*:variable,§4

A.5 Extracción de contornos.

Hasta este momento, la matriz que almacena la imagen tiene información que caracteriza a los pixeles que contiene, como pertenecientes a una de dos clases: FONDO o CONTORNO. Nuestro propósito ahora es definir una relación que agrupe los pixeles de la clase CONTORNO en subclases llamadas *circuitos*. La relación que buscamos debe estar relacionada con la morfología de los objetos; de ésta manera, en subsiguientes imágenes, encontraremos la misma subclase[1].

1. La descripción comienza mostrando las porciones principales de un programa en lenguaje C, cuyas componentes serán llenadas después. Por ejemplo, la porción del programa llamada <Las constantes 3> será reemplazada por una secuencia de declaraciones de constantes que comienza en §3.

   <Los archivos que se preprocesan 2>

   <Las constantes 3>

   <Estructuras de datos 10>

   <Las variables globales 5>

   <Rutinas auxiliares 6>

   <Rutina "main" 41>

[1] Siempre y cuando no haya oclusiones y las imágenes estén "cerca" una de otra.

2. Las rutinas para el manejo de la carta de adquisición de imágenes se encuentran en la librería "igselib.h". Son rutinas tales como inicializar la carta de adquisición, digitalizar una imagen, escribir un punto en la memoria, leer un punto de la memoria, etc. En esta serie de programas solo se utiliza la rutina que nos indica la dirección de memoria en donde se encuentra la carta de adquisición de imágenes. Todas las rutinas que pudieran estar relacionadas con la tarjeta se hicieron siguiendo el manejo estandar de apuntadores que permite el lenguaje C.

   El archivo "stdio.h" contiene las rutinas de entrada y salida estandar en el lenguaje C. Para mayor información consultar la documentación del lenguaje.

   <Los archivos que se preprocesan 2>

   #include "igselib.h"

   #include "stdio.h"

3. La imagen digital que maneja la carta de adquisición tiene dimensiones de 768 columnas ×512 hileras.

   Esta es la primera de varias secciones en donde las constantes están definidas.

   <Las constantes 3>≡

   #define *numColumnas* (768)

   #define *numHileras* (512)

   Ver también 4, 14, 29, 30, 35.

4. Los valores de lógicos, verdadero o falso, son definidos en esta sección.

   <Las constantes 3> + ≡

   #define *FALSE* (0)

   #define *TRUE* (1)

5. Las variables *memo* y *screen* expresan dos direcciones de memoria. La dirección *memo* nos indica el lugar en donde se almacena la imagen que actualmente se analiza. La dirección *screen* nos indica dos cosas: el lugar en donde se almacenan los resultados del análisis y el inicio de

la memoria de video de la tarjeta de digitalización; ello nos permite ver los resultados del análisis.

Esta es la primera de varias secciones en donde las variables están definidas.

<Las variables globales 5>≡

Byte *$*memo$*$*screen$;

Ver también 8 y 9.

6. Las rutinas que constituyen el cuerpo del programa están divididas en las siguientes partes:

   <Rutinas auxiliares 6>≡

   < Inicialización de memoria de video 7>

   <Seguimiento de contornos 16>

   <Análisis de vecindad 31>

   <Exposición de resultados 28>

7. Los resultados del proceso de seguimiento de contornos son visualizados directamente en un monitor. Esto se logra guardando los resultados del proceso en la memoria de video de la tarjeta de digitalización. Al utilizar la tarjeta, nuestro programa se vuelve dependiente del equipo pero, las facilidad de visualizar los resultados y la existencia de un bloque de memoria capaz de albergar una imagen nos inclinaron a tomar esta decisión. Sin embargo, si las ventajas dejan de existir, las rutinas se han programado para que los cambios sean mínimos.

   < Inicialización de memoria de video 7>≡

```
void initMemory() {
int i,j;
    for(i ← 0; i ≤ numHileras − 1; i ← i + 1){
        for(j ← 0; j ≤ numColumnas − 1; j ← j + 1){
            *(screen + i × numHileras + j) ← 0;
        }
    }
}
```

numHileras:constante, §3 numColumnas:constante, §3 screen:variable,§5

8. La variable *f* nos indica el archivo en donde se almacenarán los descriptores de cada curva.

<Las variables globales 5> + ≡

```
FILE *f;
```

9. La variable *numero* nos indica el número de curvas que fueron encontradas en cada escena.

<Las variables globales 5> + ≡

```
int numero;
```

10. Un pixel o punto está representado por el número de columna x y el número de hilera y. El origen se encuentra en el margen superior izquierdo, con direcciones positivas hacia la derecha y abajo.

Esta es la primera de varias declaraciones de estructuras de datos.

<Estructuras de datos 10>≡

```
struct point{
    int x,y;
} ;
```

Ver también 11, 12, 13.

11. La siguiente estructura almacena características importantes de un segmento de recta. Las coordenadas (x_o, y_o) y (x_f, y_f) son los puntos inicial y final del segmento. El ángulo del segmento con respecto a la horizontal está dado por *angle*, la cual toma valores entre 0 y π radianes; la variable *slope* es el valor de la función tangente de la variable *angle*. La coordenada (cx, cy) representa el centro de la ventana en donde se encontró el segmento.

<Estructuras de datos 10> + ≡

```
struct recta{
    float x_o, x_f, y_o, y_f, angle, slope;
    int cx,cy;
};
```

12. El área en donde se define una ventana está delimitada por su margen superior izquierdo y su margen inferior derecho; las coordenadas (x_o, y_o) y (x_f, y_f) respectivamente.

<Estructuras de datos 10> + ≡

```
struct windType {
    int x_o, y_o, x_f, y_f;
};
```

13. El análisis de los contornos recupera los siguientes descriptores de la curva:

- La longitud n. Cada unidad representa 5 pixeles.
- El promedio de dirección de la curva *media*, medido en radianes.
- La posición del centroide. La coordenada (cx,cy).
- Las coordenadas del punto inicial y final de la curva: (x_1, y_1) y (x_2, y_2) respectivamente.

<Estructuras de datos 10> + ≡

```
struct curve{
    int n;
    float media;
    float cx,cy;
    int x_1, y_1, x_2, y_2;
};
```

14. Con el fin de no abandonar el área de la imagen, nuestro algoritmo exige que una parte de los bordes sea suprimido. Este borde suprimido tiene una anchura de *minimo* pixeles.

<Las constantes 3> + ≡

```
#define minimo (10)
```

15. El nivel de gris que divide entre las dos clases: FONDO y CONTORNO esta dado por *umbral*.

<Las constantes 3> + ≡

```
#define umbral (40)
```

16. El algoritmo de seguimiento de contornos propuesto se basa en el supuesto de que cualquier curva puede ser aproximada por segmentos de recta siempre y cuando estos segmentos de recta sean suficientemente pequeños. Con el fin de aplicar este principio a una imagen binaria expuesta a ruido definimos una ventana de análisis, dentro de la cual intentaremos encontrar segmentos de recta. Las dimensiones de la ventana se definen de acuerdo a los siguientes principios:

    - Las dimensiones de una ventana deben ser suficientemente grandes para encontrar dentro un número suficientemente grande de pixeles como para que una recta sea aparente.
    - Las dimensiones de una ventana deben ser suficientemente pequeñas para no involucrar más de una recta a la vez.

    Las ventanas que se definen conforme se analiza la curva se traslapan entre ellas con el fin de darle robustes a la decisión de seguimiento de una dirección.

    <Seguimiento de contornos 16>≡

```
int tracking() {
struct windType window[6];
int             i,j,k,nivel,numWindow,numLine;
struct point    p,q,centroFijo,centroMovil,next;
struct recta    actual,siguiente,saveLine[6];
float           pendiente;
struct curve    curva;
    numero ← 0; j ← minimo
    while(j ≤ numHileras − (minimo + 1)){
        i ← minimo;
        while(i ≤ numColumnas − 1 − minimo){
            nivel ← *(memo + j × numHileras + i);
            if(nivel ≥ umbral){
                p.x ← i; p.y ← j;
                if (Neighborhood( &p, &actual)=TRUE) {
                    < Analisis de la posicion actual 17 >
                }
            }
```

```
            i ← i + 1;
        }
        j ← j + 1;
    }
    return(numero);
}
```

curve:tipo, §13
Neighborhood:función, §31
numColumnas:constante, §3
recta:tipo, §11
numColumnas:constante, §3
memo:variable,§5
numHileras:constante, §3
point:tipo, §10
screen:variable,§5
windType:tipo, §12

17. Cada posición de la imagen es evaluada para determinar ocurrencias de segmentos de recta. Una vez encontrada evidencia de que un segmento existe se le sigue en dirección de sus extremos. Al mismo tiempo un conjunto de descriptores de la curva son calculados. Conforme es posible, los pixeles que han sido analizados son borrados para no interferir con otros procesos.

    El procedimiento toma cuenta de que las curvas tengan una longitud significativa. Si una curva no tiene una longitud suficientemente grande, simplemente no se le toma en cuenta. Esta convención permite eliminar algo del ruido presente en la imagen.

    Al final, los descriptores son almacenados en un archivo para que un proceso posterior busque la correspondencia entre las curvas de dos imágenes.

    <Analisis de la posición actual 17>≡

    ```
    <Inicio de curva 18>
    numLine ← 1;
    saveLine[numLine] ← actual;
    <Inicio del salvaguardo de areas de ventanas 24>
    centroFijo ← p; centroMovil ← centroFijo;
    p.x ← actual.x_o + actual.cx;
    p.y ← actual.y_o + actual.cy;
    q.x ← actual.x_f + actual.cx;
    q.y ← actual.y_f + actual.cy;
    ```

```
next ← p; k ← 2;
pendiente ← actual.slope;
< Seguimiento de la curva 19>
if(numLine ≥ 2){
    <Final de curva 22>
    <Almacen de descriptores de la curva 23>
    for(k ← 1; k ≤ 2; k ← k + 1) DDA(saveLine[k]);
    <Termina salvaguardo de areas de ventanas 27>
}
```

actual:variable,§16 centroFijo:variable,§16 centroMovil:variable,§16
DDA:procedimiento,§28 k:variable,§16 next:variable,§16
numLine:variable,§16 p:variable,§16 pendiente:variable,§16
q:variable,§16 saveLine:variable,§16

18. Una curva comienza a ser analizada.

<Inicio de curva 18>≡

```
curva.n ← 1;
curva.media ← actual.angle;
curva.cx ← p.x; curva.cy ← p.y;
```

actual:variable,§16 curva:variable,§16 p:variable,§16

19. Una curva ha sido detectada. Lo que procede ahora es tratar de llegar tan lejos como podamos en dirección de los dos extremos de la curva. Mientras exista una secuencia de pixeles en dirección de un extremo y no encontremos un cambio brusco en la dirección de la curva continuamos en el presente extemo; de otro modo, regresamos al lugar donde encontramos el primer segmento y seguimos hacia el otro lado.

< Seguimiento de la curva 19>≡

```
while(k > 0){
    if(Neighborhood(&q, &siguiente) = TRUE){
        <Continuacion del salvaguardo de areas de ventanas 25>
        if(Continuity(actual, siguiente)){
```

```
            numLine ← numLine + 1;
            if(numLine ≤ 2){
                saveLine[numLine] ← siguiente;
            }
            else DDA(siguiente);
            <Continuacion de curva 20>
            p ← q;
            proyecta1(siguiente,centroMovil, &q);
            actual ← siguiente;
            centroMovil ← p;
        }
        else {
            <Termina un extremo de areas de ventanas 26>
            centroMovil ← centroFijo;
            actual.slope ← pendiente;
            q ← next; k ← k − 1;
            <Termina un extremo de la curva 21>
        }
    }
    else {
        <Termina un extremo de areas de ventanas 26>
    }
  }
}
```

actual:variable,§16 centroFijo:variable,§16 centroMovil:variable,§16
Continuity:funcion,§36 DDA:procedimiento,§28 Neighborhood:funcion,§31
numLine:variable,§16 p:variable,§16 pendiente:variable,§16
proyecta1:procedimiento,§37 q:variable,§16 siguiente:variable,§16
saveLine:variable,§16

20. Se continúa el análisis de la curva.

<Continuación de curva 20>≡

```
curva.n ← curva.n + 1;
curva.media ← curva.media + siguiente.angle;
curva.cx ← curva.cx + q.x;
curva.cy ← curva.cy + q.y;
```

curva:variable,§16 q:variable,§16 siguiente:variable,§16

21. Un extremo de la curva ha sido alcanzado. Podemos determinar uno de los puntos extremos de la curva.

<Termina un extremo de la curva 21>≡

```
if (k > 0) {
    curva.x_1 ← (float)q.x; curva.x_1 ← (float)q.y;
}
```

curva:variable,§16 k:variable,§16 q:variable,§16

22. Hemos arrivado al extremo final de la curva.

<Final de curva 22>≡

```
numero ← numero + 1;
curva.x_2 ← (float)q.x; curva.y_2 ← (float)q.y;
curva.cx ← curva.cx/curva.n;
curva.cy ← curva.cy/curva.n;
curva.media ← curva.media/curva.n;
```

curva:variable,§16 numero:variable,§16 q:variable,§16

23. Los descriptores de la curva: la longitud, la dirección promedio, la posición del centroide y los puntos inicial y final son almacenados en un archivo.

<Almacén de descriptores de la curva 23>≡

```
fprintf(f,"%d ",numero);
fprintf(f,"%d ",curva.n);
fprintf(f,"%f ",curva.media);
fprintf(f,"%f ",curva.cx);
fprintf(f,"%f ",curva.cy);
```

```
fprintf(f,"%d ",curva.x1);
fprintf(f,"%d ",curva.x1);
fprintf(f,"%d ",curva.x2);
fprintf(f,"%d ",curva.y2);
```

curva:variable,§16 f:variable,§5 numero:variable,§16

24. Un sistema de ventanas que se traslapan es puesta en funcionamiento para seguir el camino de la curva. Este sistema de ventanas nos servirá para borrar, en el momento oportuno, los pixeles que ya fueron analizados.

<Inicio del salvaguardo de áreas de ventanas 24>≡

```
numWindow ← 1;
saveWindow(&window[numWindow], p);
```

numWindow:variable,§16 p:variable,§16 saveWindow:procedimiento,§38
window:variable,§16

25. Las áreas que corresponden a las ventanas uno y dos no pueden ser borradas inmediatamente en virtud del translape que entre ellas existe. En lugar de ello, hay un corrimiento entre las ventanas tres, cuatro y cinco.

<Continuación del salvaguardo de áreas de ventanas 25>≡

```
numWindow ← numWindow + 1;
saveWindow(&window[numWindow], q);
if(numWindow = 5){
    eraseWindow(window[3]);
    window[3] ← window[4]; window[4] ← window[5];
    numWindow ← 4;
}
```

eraseWindow:procedimiento,§39 numWindow:variable,§16
saveWindow:procedimiento,§38 window:variable,§16

26. Cuando un extremo de la curva ha sido alcanzado, debemos eliminar los puntos hasta el momento han sido analizados pero no borrados. Esto incluye de la tercera ventana en adelante y la mitad de la segunda que apunta hacia el extremo alcanzado.

<Termina un extremo de áreas de ventanas 26>≡

```
for(k ← 3; k ≤ numWindow; k ← k + 1){
    eraseWindow(window[l]);
}
eraseWindow2(window[1],window[2]);
numWindow ← 1;
```

eraseWindow:procedimiento,§39 eraseWindow2:procedimiento,§40 k:variable,§16
numWindow:variable,§16 window:variable,§16

27. Los dos extremos de la curva han sido alcanzados. Ahora resta eliminar los pixeles que han sido analizados para no interferir con el análisis de otras curvas.

<Termina salvaguardo de áreas de ventanas 27>≡

```
for(k ← 1; k ≤ numWindow; k ← k + 1){ eraseWindow(window[k]);
}
```

eraseWindow:procedimiento,§39 k:variable,§16 numWindow:variable,§16
window:variable,§16

28. Este algoritmo dibuja una línea recta sobre la memoria de video, con el fin de visualizar el resultado del seguimiento de contornos. Se programó con el propósito de no hacer uso de las librerías de la tarjeta. Con ésto logramos que el programa, no obstante ser dependiente del equipo, pueda ser fácilmente modificable para adaptarlo a otro.

El algoritmo toma como entradas las coordenadas del punto inicial y final del segmento. Calcula el eje de desplazamiento mayor y comen zando desde el punto inicial, se incrementa un Δ sobre el eje de despla zamiento menor dibujando los puntos que van formando la recta hasta llegar al punto final.

```
void DDA(lineDescriptor)
struct recta lineDescriptor;
{int i,j,x1,x2,z1,y2;
float x,y,Δx,Δy,l;
    i ← lineDescriptor.cx; j ← lineDescriptor.cy;
    x1 ← (int)(lineDescriptor.xo) + i;
    z1 ← (int)(lineDescriptor.yo) + j;
    x2 ← (int)(lineDescriptor.xf) + i;
    y2 ← (int)(lineDescriptor.yf) + j;
    if(¬((x1 = x2) ∧ (z1 = y2))){
        l ←| x2 − x1 |;
        if(| y2 − z1 |> l)l ←| y2 − z1 |;
        Δx ← (x2 − x1)/l; Δy ← (y2 − z1)/l;
        x ← x1 + 0.5; y ← z1 + 0.5;
        for(i ← 1; i ≤ (int)l; i ← i + 1){
            *(screen + numHileras × (int)y + (int)x) ← 255;
            x ← x + Δx; y ← y + Δy;
        }
    }
}
```

numHileras:constante,§3 recta:tipo,§10 screen:variable,§5

29. Para algunas comparaciones entre reales, es preciso definir un valor real ϵ de valor pequeño.

    <Las constantes 3> + ≡

    ```
    #define ϵ (0.0001)
    ```

30. La dimensión de la ventana que hemos definido es de 11×11; es un valor arbitario que puede cambiar de imagen a imagen. Aquí *windowSize* representa el máximo valor absoluto que puede tomar una coordenada dentro de la ventana.

    <Las constantes 3> + ≡

    ```
    #define windowSize (5)
    ```

31. Sobre una vecindad, definida por un pixel que pertenece al contorno, buscamos un segmento de recta utilizando el método de aproximación

de una recta a un conjunto de puntos por mínimos cuadrados. Después encontramos los parámetros de la recta que se define y finalmente encontramos los puntos final e inicial del segmento.

<Análisis de vecindad 31>≡

```
int Neighborhood(p,line)
struct point *p;
struct recta *line;
{int i,j,nivel,index,xr[5],yr[5],result;
float x,y,m,b,xData[122],yData[122];
float valorA1, valorB1, valorA2, valorB2, error1, error2,valor;
float sumaAbsX,sumaAbsY,sumaX,sumaY,sumaXY,
      sumaX2,sumaY2,cociente;
   < Censo de la vecindad 32>
   if(index >← 5){
      <Aproximacion por minimos cuadrados 33>
      <Limites del segmento 34>
   } else{result ← FALSE; }
   return (result);
}
```

point:tipo,§10 recta:tipo,§10

32. La zona que corresponde a la ventana es barrida. Al mismo tiempo, la información con respecto a los parámetros de la recta que se aproximará son recabados.

< Censo de la vecindad 32>≡

```
index ← 0
sumaX ← 0.0
sumaY ← 0.0
sumaXY ← 0.0;
sumaX2 ← 0.0;
sumaY2 ← 0.0;
sumaAbsX ← 0.0; sumaAbsY ← 0.0;
for(i ← p.y − 5; i ≤ p.y + 5; i ← i + 1){
```

```
for(j ← p⃗.x − 5; j ≤ p⃗.x + 5; j ← j + 1){
    nivel ← *(memo + i × numHileras + j);
    if(nivel ≥ umbral){
        x ← j − p⃗.x; y ← i − p⃗.y;
        index ← index + 1;
        xData[index] ← (float)x; yData[index] ← (float)y;
        sumaAbsX ← sumaAbsX + | x |;
        sumaAbsY ← sumaAbsY + | y |;
        sumaX ← sumaX + (float)x;
        sumaY ← sumaY + (float)y;
        sumaXY ← sumaXY + (float)(x × y);
        sumaX² ← sumaX² + (float)(x × x);
        sumaY² ← sumaY² + (float)(y × y);
    }
  }
}
```

i:variable,§31	index:variable,§31	j:variable,§31
memo:variable,§5	nivel:variable,§31	numHileras:constante,§3
sumaAbsX:variable,§31	sumaAbsY:variable,§31	sumaX:variable,§31
sumaXY:variable,§31	$sumaX^2$:variable,§31	sumaY:variable,§31
$sumaY^2$:variable,§31	x:variable,§31	xData:variable,§31
y:variable,§31	yData:variable,§31	umbral:constante,§3
recta:tipo,§10	x:variable,§31	

33. Con el fin de aproximar una recta al conjunto de pixeles en la ventana, debemos elegir una función a minimizar. Hemos elegido la función que minimiza la distancia de los puntos a la recta propuesta. Definimos dos hipótesis con respecto a las coordenadas de los pixeles que pertenecen al contorno y están dentro de la ventana:

    - En las coordenadas (X,Y), la parte X puede estar mal pero la parte Y es correcta.
    - La parte Y puede estar mal pero la parte X es correcta.

    En base a esas hipótesis calculamos los parámetros m y b (la pendiente y el cruce con el eje Y, respectivamente) de dos ecuaciones. La ecuación que minimiza nuestra función de error es elegida como la resultante del proceso.

<Aproximación por mínimos cuadrados 33>≡

```
cociente ← (index × sumaXY − sumaX × sumaY);
if(| cociente |> ε){
    valorA1 ← (index × sumaY² − sumaY × sumaY)/cociente;
    valorB1 ← (sumaY − valorA1 × sumaX)/index;
}
else {
    if(| sumaXY |< ε){
        if(| sumaAbsX |< ε){
            valorA1 ← 1000.0;
            valorB1 ← 0.0;
        }
        else {
            if(| sumaAbsY |< ε){
                valorA1 ← 0.0;
                valorB1 ← (sumaY − valorA1 × sumaX)/index;
            }
        }
    }
    else {valorA1 ← 1000.0;
        valorB1 ← 0.0;
    }
}
error1 ← 0.0;
for(i ← 1; i ≤ index; i ← i + 1){
    valor ← yData[i] − valorA1 × xData[i] − valorB1;
    error1 ← error1 + (valor × valor)/(1.0 + valorA1 × valorA1);
}
cociente ← (index × sumaX² − sumaX × sumaX);
if(| cociente |> ε){
    valorA2 ← (index × sumaXY − sumaX × sumaY)/cociente;
}
elsevalorA2 ← 1000.0;
valorB2 ← (sumaY − valorA2 × sumaX)/index;
error2 ← 0.0;
for(i ← 1; i ≤ index; i ← i + 1){
```

```
    valor ← (yData[i] − valorA2 × xData[i] − valorB2);
    error2 ← error2 + (valor × valor)/(1.0 + valorA2 × valorA2);
}
if(error1 < error2){
    m ← valorA1; b ← valorB1;
}
else{m ← valorA2; b ← valorB2;
}
```

b:variable,§31	cociente:variable,§31	ε:constante,§29
error1:variable,§31	error2:variable,§31	i:variable,§31
index:variable,§31	m:variable,§31	sumaAbsX:variable,§31
sumaAbsY:variable,§31	sumaX:variable,§31	sumaXY:variable,§31
sumaY:variable,§31	valor:variable,§31	valorA1:variable,§31
valorA2:variable,§31	valorB1:variable,§31	valorB2:variable,§31
xData:variable,§31	yData:variable,§31	

34. La recta que se ha encontrado es acotada por los límites de la ventana. Con el fin de determinar los puntos extremos del segmento, se calcula la intersección de la recta con los límites de la ventana. La ventana tiene dimensiones 11×11 por lo que las intersecciones se hacen con las rectas: $Y = 5$, $Y = -5$, $X = 5$, y $X = -5$.

<Límites del segmento 34>≡

```
index ← 1;
cociente ← m × windowSize + b;
if(| cociente |≤ windowSize){
    xr[index] ← windowSize; yr[index] ← cociente;
    index ← index + 1;
}
cociente ← −m × windowSize + b;
if(| cociente |≤ windowSize){
    xr[index] ← −windowSize; yr[index] ← cociente;
    index ← index + 1;
}
if(| m |> ε){
    cociente ← (windowSize − b)/m;
    if(| cociente |≤ windowSize){
```

```
            xr[index] ← cociente; yr[index] ← windowSize;
            index ← index + 1;
        }
        cociente ← (−windowSize − b)/m;
        if(| cociente |≤ windowSize){
            xr[index] ← cociente; yr[index] ← −windowSize;
            index ← index + 1;
        }
    }
    line.x_o ← xr[1]; line.y_o ← yr[1];
    line.x_f ← xr[2]; line.y_f ← yr[2];
    line.slope ← m;
    valor ← arctan(m);
    if(valor < 0.0)valor ← valor + π;
    line.angle ← valor;
    line.cx ← p̄.x; line.cy ← p̄.y;
    result ← TRUE;
}
```

b:variable,§31	*cociente*:variable,§31	c:constante,§29
index:variable,§31	*m*:variable,§31	p:constante
π:constante	*valor*:variable,§31	xr:variable,§31
yr:variable,§31	*windowSize*:constante,§30	

35. Para probar la no variabilidad de la dirección de una curva, proponemos que las rectas de ventanas vecinas tengan una diferencia de orientación pequeña. Los valores de Θ_{30} y Θ_{150} han sido determinados experimentalmente y pueden llegar a cambiar de imagen a imagen.

    <Las constantes 3> + ≡

    #define Θ_{30} (0.523598)

    #define Θ_{150} (2.61799)

36. Pretendemos que las curvas que elijamos tengan la característica de ser analíticamente expresables, lo cual se puede traducir como curvas continuas. Una de nuestras hipótesis de base consiste en considerar que una curva continua, parte de un objeto en la imagen, es una característica morfológicamente invariable del objeto; de tal manera que,

si tomamos otra imagen, no muy distante de la primera, la curva se mantendrá aproximadamente con las mismas características.

La comprobación de la dirección de una curva es una de las pruebas fundamentales del algoritmo. Consiste en determinar si la diferencia entre las direcciones de los segmentos de recta son tan pequeñas como para considerarlos como parte de una sola curva.

```
int Continuity(line1, line2)
struct recta line1, line2;
{ float slope,radianes;
int found;
    slope ← (line1.slope − line2.slope)/(1.0 + line1.slope × line2.slope);
    radianes ← arctan(slope);
    if(radianes < 0.0)radianes ← radianes + π;
    found ← ((radianes < Θ30) ∨ (radianes > Θ150))?TRUE : FALSE;
    return (found);
}
```

recta:tipo,§11 Θ_{150}:constante,§35 Θ_{30}:constante,§35

37. El propósito de esta rutina es calcular el centro de la siguiente ventana a analizar. En base a las dos direcciones posibles de la recta en la ventana actual, determinamos cual es el punto más alejado al centro de la ventana anterior y este punto es nuestro siguiente centro.

```
void proyecta1(sline,p,q)
struct recta sline;
struct point p,*q;
{
    if((| p.x − (sline.x0 + sline.cx) |< 3.0)∧
            (| p.y − (sline.y0 + sline.cy) |< 3.0){
        q⃗.x ← sline.xf + sline.cx; q⃗y ← sline.yf + sline.cy;
    }
    else { q⃗.x ← sline.x0 + sline.cx; q⃗.y ← sline.y0 + sline.cy; }
}
```

point:tipo,§10 recta:tipo,§11

38. Se define el área sobre la cual se ha trabajado y en este momento no debe ser borrado ningún punto.

```
void saveWindow(window,p)
struct windType *window;
struct point p;
{
    window.x_o ← p.x − 5; window.y_o ← p.y − 5;
    window.x_f ← p.x + 5; window.y_f ← p.y + 5;
}
```

point:tipo,§10 windType:tipo,§12

39. Se borran los puntos pertenecientes a las ventanas que ya han sido analizadas y cuyos puntos deben ser borrados con el fin de no ser tomados en cuenta por otras curvas.

```
void eraseWindow(window)
struct windType window;
{ int i,j;
    for(i ← window.y_o; i ≤ window.y_f; i ← i + 1){
        for(j ← window.x_o; j ≤ window.x_f; j ← j + 1){
            *(memo + numHileras × i + j) ← 0;
        }
    }
}
```

memo:variable,§5 numHileras:constante,§3 windType:tipo,§12

40. En virtud del sistema de seguimiento de curvas, en la que ocurre un translape con el fin de darle robustes a la decisión de seguir un contorno, existe la necesidad de borrar la mitad de una ventana. Este procedimiento se ejecuta una vez por cada curva, en el lugar en donde fué encontrado el primer segmento y en la dirección del primer extremo de la curva.

```
void eraseWindow2(window1, window2)
struct windType window1, window2;
{ int i,j;
int save[11][11];
    for(i ← window1.yo; i ≤ window1.yf; i ← i + 1){
        for(j ← window1.xo; j ≤ window1.xf; j ← j + 1){
            save[i − window1.yo][j − window1.xo] ←
                        *(memo + numHileras × i + j);
        }
    }
    for(i ← window2.yo; i ≤ window2.yf; i ← i + 1){
        for(j ← window2.xo; j ≤ window2.xf; j ← j + 1){
            *(memo + numHileras × i + j) ← 0;
        }
    }
    for(i ← window1.yo; i ≤ window1.yf; i ← i + 1){
        for(j ← window1.xo; j ≤ window1.xf; j ← j + 1){
            *(memo + numHileras × i + j) ←
                        save[i − window1.yo][j − window1.xo];
        }
    }
}
```

memo:variable,§5 numHileras:constante,§3 windType:tipo,§12

41. El programa analiza dos imágenes, "a" y "b" una por una. La imagen en turno es almacenada en una área de memoria reservada, cuya dirección comienza en *memo*; el resultado del análisis se almacena en la memoria de la tarjeta de digitalización, cuya dirección comienza en *screen*; lo que permite ver el resultado en un monitor especial.

    La descripción de cada una de las curvas es almacenada en el archivo "output2" y el número de curvas por imagen se guarda en el archivo "output3".

    <Rutina "main" 41>≡

```
main() {
```

```
FILE *g;
    f ← fopen("output2","w");
    g ← fopen("output3","w");
    addressVideoCard(&screen);
    memo ← (Byte*)malloc(numColumnas × numHileras);
    retrieveImage("a");
    initMemory();
    numero ←tracking();
    fprintf(g,"%d",numero);
    retrieveImage("b");
    initMemory();
    numero ←tracking();
    fprintf(g,"%d",numero);
    fclose(g);
    free(memo);
    fclose(f);
}
```

f:variable,§8
memo:variable,§5
numColumnas:constante,§3
retrieveMemory:programa
tracking:funcion, §16
initMemory:procedimiento, §7
numero:variable,§9
numHileras:constante,§3
screen:variable,§5

A.6 Correspondencia.

El programa de seguimiento de contornos hereda un conjunto de descriptores de segmentos de curva para cada imagen. A partir de ellos debemos recuperar la información de profundidad. Para ello proponemos un algoritmo de dos etapas:

- Tomamos el conjunto de descriptores de las curvas de la primera imagen y los comparamos con los descriptores de la segunda imagen. Algunas curvas de la primera imagen tendrán una curva en la segunda imagen que les corresponderá, otras tendrán varias y otras no tendrán ninguna curva candidata a corresponderles. Para las curvas que tienen por lo menos una correspondencia, deducimos las curvas que realmente corresponden en la imagen basándonos en el promedio de desplazamiento

que observan los centroides de las curvas que son candidatos a corresponder.

- Las curvas que no han encontrado su correspondiente son consideradas. Se investiga el efecto que tiene el calcular descriptores para un conjunto de curvas de la primera imagen al hacerlas corresponder con otro conjunto de la segunda imagen. El caracter explosivo de esta búsqueda es minimizada al hacer uso de las siguientes reglas:
  - Solo las curvas que cumplen un cierto criterio de longitud son consideradas.
  - Las curvas de la imagen son agrupadas en varios conjuntos de acuerdo a un criterio de proximidad entre extremos de curvas. Estos conjuntos son llamados *circuitos* y representan, físicamente, estructuras que idealmente deben existir en los objetos. Las curvas que corresponden pertenecen a uno de estos *circuitos*.

1. La descripción comienza mostrando las porciones principales de un programa en lenguaje C, cuyas componentes serán llenadas después. Por ejemplo, la porción del programa llamada < Constantes 9 >, será reemplazada por una secuencia de declaraciones de constantes que comienza en §9.

   < Archivos "include" que se preprocesan 2 >

   < Constantes 9 >

   < Estructuras de datos 4 >

   < Variables globales 6 >

   < Rutinas auxiliares 3 >

   < Rutina Principal 40 >

2. El archivo "stdio.h" contiene las rutinas de entrada y salida estandar en el lenguaje C. Para mayor información consultar la documentación del lenguaje.

   < Archivos "include" que se preprocesan 2 >≡

   ```
   #include "stdio.h"
   ```

3. Las rutinas que constituyen el cuerpo del programa están divididas en las siguientes partes:

   < Rutinas auxiliares 3 >≡

   < Recuperación de información 7 >

   < Estudio de la estructura 8 >

   < Correspondencia 16 >

   < Cálculo de profundidad 38 >

   < Resultados 39 >

4. La estructura "node" posee datos acerca de los descriptores de los segmentos de curva, ayudas para el proceso de correspondencia y resultados:

   Descriptores:

   - n. Longitud de la curva. Cada unidad representa cinco pixeles.
   - *media*. Dirección promedio de la curva, expresado en radianes.
   - (x, y). Coordenadas de la posición del centroide. El origen del sistema de coordenadas se encuentra en el margen superior izquierdo; el desplazamiento positivo en las direcciones X y Y es hacia la derecha y hacia abajo respectivamente.
   - (x_o, y_o), (x_f, y_f). Coordenadas de los puntos inicial y final del segmento de curva.

   Ayudas:

   - *label*. Número de curva.
   - Δx, Δy. Valor absoluto del desplazamiento en los ejes X y Y, de la curva con respecto a su curva correspondiente.
   - *longGrupo*. Se utiliza cuando la curva ha encontrado correspondencia junto a un grupo de curvas, representa la longitud del grupo. Cada unidad representa cinco pixeles.

- *circuito.* Las curvas de los objetos son agrupadas de acuerdo a si algún extremo de una curva está "cerca" de un extremo de otra curva diferente a ella misma. Idealmente, estos agrupamientos de curvas representan estructuras independientes de los objetos. Estas estructuras son bautizadas como *circuitos* y son utilizadas durante el proceso de correspondencia de grupos de curvas.

Resultados:

- *z.* Coordenada de profundidad del centroide calculada.
- *matched.* Curva con la que se ha encontrado correspondencia.
- *group.* Se utiliza cuando la curva ha encontrado correspondencia como un grupo de curvas, representa el identificador del grupo.

Esta es la primera de varias secciones, en donde se definen estructuras de datos.

< Estructuras de datos 4 >≡

```
struct node{
        float media;
        float x,y,z;
        float Δx, Δy;
        float longGrupo;
        int   label,n,matched,group;
        int   x_o, y_o, x_f, y_f;
        int   circuito;
};
```

Ver también 5, 12, 13 y 14.

5. Una imagen está formada por un conjunto de curvas y el número de *circuitos* que estas curvas forman.

< Estructuras de datos 4 > + ≡

```
struct image{
        struct node *nodo;
                int index;
};
```

6. Las imágenes están representadas por los descriptores de sus curvas. Las variables *imagen*[0] e *imagen*[1] guardan información de las curvas de la primera y segunda imagen respectivamente.

   Esta es la primera de varias secciones, en donde se definen variables globales.

   < Variables globales 6 >≡

   ```
   struct image imagen[2];
   ```

   Ver también 28 y 35.

7. Los descriptores de las curvas son almacenadas en el archivo "output2" por el proceso de extracción de contornos. El número de curvas que se extrajeron en cada imagen está almacenado en el archivo "output3". El espacio para almacenar esta información en memoria principal es reservado al momento de ejecución del programa.

   < Recuperación de información 7 >≡

   ```
   void loadInformation(){
   int i;
   FILE *f,*g;
   struct node msr;
       f ← fopen("output3","r");
       g ← fopen("output2","r");
       fscanf(f,"%d",&imagen[0].index);
       imagen[0].nodo← (struct node *)
           calloc(imagen[0].index+1,sizeof(msr));
       for(i ← 1; i ≤ imagen[0].index; i ← i + 1){
           fscanf(g,"%d",&msr.label);
           fscanf(g,"%d",&msr.n);
           fscanf(g,"%f",&msr.media);
           fscanf(g,"%f",&msr.x);
           fscanf(g,"%f",&msr.y);
           fscanf(g,"%d",&msr.x_o);
           fscanf(g,"%d",&msr.y_o);
           fscanf(g,"%d",&msr.x_f);
           fscanf(g,"%d",&msr.y_f);
   ```

```
        msr.group ← 0; msr.matched ← 0;
        msr.longGrupo ← 99999.9;
        msr.circuito ← 0;
        imagen[0].nodo[i] ← msr;
    }
    fscanf(f,"%d",&imagen[1].index);
    imagen[1].nodo← (struct node *)
        calloc(imagen[1].index+1,sizeof(msr));
    for(i ← 1; i ≤ imagen[1].index; i ← i + 1){
        fscanf(g,"%d",&msr.label);
        fscanf(g,"%d",&msr.n);
        fscanf(g,"%f",&msr.media);
        fscanf(g,"%f",&msr.x);
        fscanf(g,"%f",&msr.y);
        fscanf(g, "%d", &msr.x_o);
        fscanf(g, "%d", &msr.y_o);
        fscanf(g, "%d", &msr.x_f);
        fscanf(g, "%d", &msr.y_f);
        msr.group ← 0; msr.matched ← 0;
        msr.longGrupo ← 99999.9;
        msr.circuito ← 0;
        imagen[1].nodo[i] ← msr;
    }
    fclose (g);
    fclose (f);
}
```

imagen:variable, §6 node:tipo, §4

8. Un *circuito* es un conjunto de curvas C, tal que dos curvas diferentes r y s pertenecen a C si entre alguno de sus extremos existe una separación de unos pocos pixeles. Es decir, las curvas adyacentes que han sido calificadas como pertenecientes a partes diferentes del objeto en virtud de su orientación distinta, son agrupadas como pertenecientes a una sola estructura.

   La siguiente rutina busca todos los *circuitos* que existen en el conjunto de curvas que constituyen la imagen.

< Estudio de la estructura 8 >≡

```
void createStructure(c)
struct image *c;
{
int i,circuito;
    circuito ← 0;
    for(i ← 1; i ≤ c.index; i ← i + 1){
        if (c.nodo[i].circuito = 0){
            circuito ← circuito + 1;
            unidad(c,i,circuito);
        }
    }
    c.circuitos ← circuito;
}
```

image:tipo, §5 unidad:procedimiento, §10

9. Para poder ser consideradas como parte de un mismo *circuito*, los extremos de dos curvas no deben estar más alejados de *separacion* pixeles.

   Esta es la primera de varias secciones, en donde se definen constantes.

   < Constantes 9 >≡

   ```
   #define separacion (5)
   ```

   Ver también 11, 18, 19, 20, 21, 24, 27.

10. Para cada curva, se buscan algunas otras cuyos extremos estén próximos. Un conjunto así constituido da origen a un *circuito*. Un *circuito* es, idealmente, una estructura dependiente de la morfología del objeto y por lo tanto la podemos utilizar en el proceso de correspondencia de curvas.

```
void unidad(c,i,circuito)
struct image *c;
int i,circuito;
```

```
{
struct node a,b;
float x, y, dist_1, dist_2, dist_3, dist_4;
int j;
    if (c.nodo[i].circuito = 0){
        c.nodo[i].circuito ← circuito;
        a ← c.nodo[i];
        for (j ← 1; j ≤ c.index; j ← j + 1){
            if (j! ← i){
                b ← c.nodo[j];
                x ← (float)(a.xo − b.xo);
                y ← (float)(a.yo − b.yo);
                dist_1 ← √(x × x + y × y);
                x ← (float)(a.xo − b.xf);
                y ← (float)(a.yo − b.yf);
                dist_2 ← √(x × x + y × y);
                x ← (float)(a.xf − b.xo);
                y ← (float)(a.yf − b.yo);
                dist_3 ← √(x × x + y × y);
                x ← (float)(a.xf − b.xf);
                y ← (float)(a.yf − b.yf);
                dist_4 ← √(x × x + y × y);
                if((dist_1 < separacion)∨
                        (dist_2 < separacion)∨
                        (dist_3 < separacion)∨
                        (dist_4 < separacion))
                    unidad(c, j, circuito);
            }
        }
    }
}
```

imagen:variable, §6 node:tipo, §4 separacion:constante, §9
unidad:procedimiento, §10

11. Para cada curva se almacenarán hasta un máximo de *numCurvasAmbiguas* posibles correspondencias.

< Constantes 9 > + ≡

```
#define numCurvasAmbiguas (10)
```

12. La siguiente estructura servirá para almacenar una posible correspondencia entre dos curvas. Estará insertada en otra estructura que se referirá a la primera imagen y por lo tanto aquí solo necesitamos información sobre la segunda imagen.

    La variable *numCurva* se refiere a la curva que es potencialmente correspondiente. La variable *descriptor* proporciona el desplazamiento entre las curvas que potencialmente corresponden.

    < Estructuras de datos 4 > + ≡

```
struct suspected{
        float descriptor ;
        int   numCurva ;
}
```

13. Una curva puede encontrar varias otras, que tengan descriptores parecidos a ella. La siguiente estructura de datos almacena estos posibles conflictos.

    < Estructuras de datos 4 > + ≡

```
struct hilera{
        struct suspected item[numCurvasAmbiguas+1];
        int              index;
}
```

14. La siguiente estructura de datos sirve como referencia de la primera imagen. Cada nodo de la primera imagen tendrá una hilera disponible en la que se registrarán las posibles correspondencias.

    < Estructuras de datos 4 > + ≡

```
struct matrix{
```

```
        struct hilera *item;
        int           index;
}
```

15. La variable *match* representa el grafo relacional que tienen como nodos dos componentes; es decir, los identificadores de las curvas de la primera y segunda imagen cuyos descriptores son parecidos unos con otros.

    < Variables globales 6 > + ≡

    ```
    struct matrix match;
    ```

16. El proceso de correspondencia se realiza en tres etapas. La primera se basa en tratar las curvas de forma insolada. Se crea una estructura relacional en la que, para cada curva de la primera imagen, se consideran todas las curvas que potencialmente le corresponden en la segunda imagen. En la segunda etapa se consideran los desplazamientos de los centroides y en base a ello se resuelven los conflictos que pudieran resultar de la primera etapa. Por último, en la tercera etapa se calculan los descriptores de conjuntos de curvas en base a la información estructural llmada *circuitos.*

    < Apareamiento >≡

    < Inicializacion de la estructura de correspondencia 17 >

    < Apareamiento entre curvas 22 >

    < Apareamiento entre estructuras 23 >

    < Apareamiento posterior 25 >

17. La estructura *match* es formalmente un grafo relacional. En la variable *match.item*[*i*] se almacenará el grupo de curvas que corresponderán con la curva *i*.

    < Inicialización de la estructura de correspondencia 17 >≡

    ```
    void initMatchStructure() {
    struct hilera row;
    int i;
        match.index ← imagen[0].index;
    ```

```
    match.item ←(struct hilera *)
        calloc(match.index+1,sizeof(row));
    for (i ← 0; i ≤ match.index; i ← i + 1){
        match.item[i].index ← 0;
    }
}
```

hilera:tipo, §13 match:variable, §15

18. La imagen tiene dimensiones 768×512 pixeles. El número de columnas será utilizado para el almacenamiento de todas las posibles diferencias entre renglones.

    < Constantes 9 > + ≡

    #define *numHileras* (512)

19. Los centroides de las curvas deben tener la coordenada X muy semejante. Esta diferencia se representa por la constante *separaciónImagenes*.

    < Constantes 9 > + ≡

    #define *separacionImagenes* (40)

20. La diferencia entre el promedio de dirección entre las curvas debe ser pequeño. Esta diferencia se representa por la constante θ_{20}.

    < Constantes 9 > + ≡

    #define θ_{20} (0.34906585)

21. La diferencia entre las longitudes de las curvas debe ser pequeño. Esta diferencia se representa por la constante *cocienteLong*, que representa el valor del cociente entre las longitudes de la curva menor y la de mayor longitud.

    < Constantes 9 > + ≡

    #define *cocienteLong* (0.8)

22. Cada curva de la primera imagen es comparada con cada curva de la segunda. Cuando los descriptores de ambas son similares consideramos que ha habido correspondencia y la registramos para futuras referencias. Después analizamos el comportamiento general de los desplazamientos de los centroides de las curvas que correspondieron; con ésto obtenemos información útil para resolver ambigüedades. Los descriptores utilizados en la comparación son:

Longitud. El cociente de la curva de menor longitud con el de mayor deberá ser mayor de *cocienteLong*.

Dirección. La diferencia entre el promedio de las direcciones deberá ser menor de θ_{20} grados.

Desplazamiento del Centroide. La diferencia entre la coordenada X deberá ser menor de *separaciónImagenes* pixeles.

< Apareamiento entre curvas 22 >≡

```
void matchSingleLine(){
int i,j,index;
int histograma[numHileras],maximo,minimo;
float Δl, Δx, Δy, Δmedia, m1, m2;
    for (i ← 0; i < numHileras; i ← i + 1)histograma[i] ← 0;
    i ← 1;
    while (i ≤ imagen[0].index){
        j ← 1;
        m1 ← tan(imagen[0].nodo[i].theta);
        while (j ≤ imagen[1].index){
            Δl ← (imagen[0].nodo[i].n > imagen[1].nodo[j].n)?
                (float)imagen[1].nodo[j].n/(float)imagen[0].nodo[i].n :
                (float)imagen[0].nodo[i].n/(float)imagen[1].nodo[j].n;
            Δx ←| imagen[0].nodo[i].cx − imagen[1].nodo[j].cx);
            Δy ←| imagen[0].nodo[i].cy − imagen[1].nodo[j].cy);
            m2 ← tan(imagen[1].nodo[j].theta);
            Δmedia ← arctan((m2 − m1)/(1 + m1 ×, m2));
            if (Δmedia < 0.0)
                Δmedia ← Δmedia + π;
            if (Δmedia > π/2.0)Δmedia ← π − Δmedia;
```

```
            if ((Δx ≤ separacionImagenes)∧
                (Δl ≥ cocienteLong) ∧ (Δmedia ≤ θ20)){
                if (match.item[i].index < numCurvasAmbiguas){
                    match.item[i].index ← match.item[i].index + 1;
                    index ← match.item[i].index;
                    match.item[i].item[index].numCurva ← j;
                    match.item[i].item[index].descriptor ← Δy;
                    histograma[(int)Δy] ← histograma[(int)Δy] + 1;
                }
            }
            j ← j + 1;
        }
        i ← i + 1;
    }
    maximo ← 0;
    for (i ← 0; i < numHileras; i ← i + 1){
        if (histograma[i] > maximo){
            maximo ← i;
        }
    }
    for (i ← 1; i ≤ match.index; i ← i + 1){
        if (match.item[i].index ≥ 1){
            minimo ← 9999;
            for (j ← 1; j ≤ match.item[i].index; j ← j + 1){
                if (| (match.item[i].item[j].descriptor − (float)maximo) <
                    (float)minimo){
                    minimo ←
                        | (int)match.item[i].item[j].descriptor − maximo |;
                    index ← j;
                }
            }
            match.item[i].index ← 1;
            match.item[i].item[1] ← match.item[i].item[index];
        }
    }
}
```

cocienteLong:constante, §21 *hilera*:tipo, §13
imagen:variable, §6 *match*:variable, §15
numCurvasAmbiguas:constante, §11 *numHileras*: constante, §18
separacionImagenes:constante, §19 θ_{20}:constante, §20

23. Los resultados de la correspondencia son almacenados con las estructuras que contienen los descriptores de las curvas. El desplazamiento entre el centroide de las curvas es calculado y registrado; este desplazamiento será útil para el cálculo de la profundidad.

< Apareamiento entre estructuras 23 >≡

```
void matchCompletGraph(){
int i,k;
float Δx, Δy;
    for (i ← 1; i ≤ match.index; i ← i + 1){
        if (match.item[i].index > 0){
            k ← match.item[i].item[1].numCurva;
            Δx ←| imagen[0].nodo[i].cx − imagen[1].nodo[k].cx);
            Δy ←| imagen[0].nodo[i].cy − imagen[1].nodo[k].cy);
            imagen[0].nodo[i].Δx ← Δx;
            imagen[0].nodo[i].Δy ← Δy;
            imagen[1].nodo[k].Δx ← Δx;
            imagen[1].nodo[k].Δy ← Δy;
            imagen[0].nodo[i].matched ← k;
            imagen[1].nodo[k].matched ← i;
        }
    }
}
```

imagen:variable, §6 *match*:variable, §15

24. Los valores lógicos de Verdadero y Falso son representados por las constantes *TRUE* y *FALSE* respectivamente.

< Constantes 9 > + ≡

```
#define FALSE (0)
#define TRUE (1)
```

25. Las curvas que no han encontrado su correspondiente en el proceso anterior son tratadas. Sobre cada *circuito* son tomadas, en cada caso de ser posible, grupos de curvas a los cuales se les calculan sus descriptores conjuntos. Se buscará la correspondencia entre grupos de las dos imágenes y en caso de que los descriptores sean parecidos, se declarará la correspondencia.

< Apareamiento posterior 25 >≡

```
void postMatching() {
int m,i;
int v[numCurves+1];
    for (i ← 1; i ≤ imagen[0].circuitos; i ← i + 1){
        m ← 1;
        while (m ≤ 3){
            if (initVector1(&v[0],m,i)=TRUE) {
                subPostMatch(&v[0], resto1, m);
            }
            m ← m + 1;
        }
    }
}
```

imagen: variable, §6 initVector1: procedimiento, §26 numCurves: variable, §27
resto1: variable, §28 subPostMatch: procedimiento, §29

26. Sobre la primera imagen, se toman las primeras m curvas del *circuito* k. Este algoritmo tiene una complejidad muy elevada en términos de tiempo de cálculo; para reducirlo tomamos solo las curvas que sobrepasen cierta longitud.

```
int initVector1(v,m,k)
int *v,m,k;
{
```

```
int i,bon;
    resto1 ← 0;
    for (i ← 1; i ≤ imagen[0].index; i ← i + 1){
        if ((imagen[0].nodo[i].circuito = k)∧
            (imagen[0].nodo[i].matched = 0)){
            if (imagen[0].nodo[i].n ≥ 5){
                resto1 ← resto1 + 1;
                vector1[resto1] ← i;
            }
        }
    }
    if (resto1 ≥ m){
        for (i ← 1; i ≤ resto1; i ← i + 1){
            v[i] ← (i ≤ m)?TRUE : FALSE;
        }
        bon ← TRUE;
    }
    else bon ← FALSE;
    return (bon);
}
```

imagen:variable, §6 resto1:variable, §28 vector1:variable, §28

27. El número máximo de curvas que se permite por imagen es igual a *numCurves*.

< Constantes 9 > + ≡

#define *numCurves* (200)

28. Las variables que se presentan en esta sección se utilizan en el proceso de correspondencia de *circuitos*. Las variables $vector_1$ y $vector_2$ tienen $resto_1$ y $resto_2$ valores significativos respectivamente. La variable $resto_1$ y $resto_2$ representan el número de curvas en el *circuito*.

< Variables globales 6 > + ≡

int $vector_1[numCurves + 1], vector_2[numCurves + 1], resto_1, resto_2$;

numCurves:constante, §27

29. Un vector de longitud n' es recibido. Representa las curvas de un *circuito*. La variable m indica cuantos elementos del *circuito* son utilizados para calcular los descriptores. El número de combinaciones está dado por:

$$\binom{n'}{m} \tag{A.4}$$

Algoritmo. Dado un arreglo v unidimensional de longitud n', con elementos 1 en las primeras m posiciones a la izquierda y el resto 0s, calcular el conjunto C de combinaciones de n' tomando m.

- Si $n' = m$, entonces tenemos un elemento de C.
- Si $m = 1$, entonces tenemos un elemento de C y tendremos $n' - 1$ corriendo un elemento del vector hacia la derecha.
- El conjunto C se forma con los resultados de aplicar el algoritmo con los últimos $n' - 1$ elementos y con los primeros $n' - 1$ elementos.

```
void subPostMatch(v, n', m)
int *v, m, n';
{
int i ,a[numCurves+1];
    for (i ← 1; i ≤ resto1; i ← i + 1)a[i] ← v[i];
    ordenar(&a[0], resto_1, n', m);
    if (n' = m){
        postMatching1(&a[0]);
    }
    else {
        if (m=1) {
            i ← 1;
            while (i ≤ n'){
                postMatching1(&a[0]);
                shift(&a[0], resto_1, n' − i);
                i ← i + 1;
            }
        }
        else {
            subPostMatch(&a[0], n' − 1, m − 1);
```

```
            subPostMatch(&a[0], n' − 1, m);
        }
    }
}
```

numCurves:constante, §27 ordenar:procedimiento, §34
postMatching1:procedimiento, §30 $resto_1$:variable, §28
shift:procedimiento, §33 subPostMatch:procedimiento, §29

30. Sobre las curvas que no ha encontrado correspondencia en el proceso anterior son tratadas. Sobre cada *circuito* son tomadas, en cada caso de ser posible, grupos de curvas a los cuales se les calculan sus descriptores conjuntos. Se buscará la correspondencia entre grupos de las dos imágenes y en caso de que los descriptores sean parecidos, se declarará la correspondencia.

```
void postMatching1(w)
int *w;
{
int m,i;
int v[numCurves+1];
    for (i ← 1; i ≤ imagen[1].circuitos; i ← i + 1){
        m ← 1;
        while (m ≤ 3){
            if (initVector2(&v[0],m,i)=TRUE) {
                subPostMatch1(w, &v[0], resto_2, m);
            }
            m ← m + 1;
        }
    }
}
```

imagen:variable, §6 initVector2:procedimiento, §31 numCurves:variable, §27
$resto_2$:variable, §28 subPostMatch1:procedimiento, §32

31. Sobre la segunda imagen, se toman las primeras m curvas del *circuito* k. Este algoritmo tiene una complejidad muy elevada en términos

de tiempo de cálculo; para reducirla tomamos solo las curvas que sobrepasen cierta longitud.

```
int initVector2(v,m,k)
int *v,m,k;
{
int i ,bon;
    resto2 ← 0;
    for (i ← 1; i ≤ imagen[1].index; i ← i + 1){
        if ((imagen[1].nodo[i].circuito = k)∧
                (imagen[1].nodo[i].matched = 0)){
            if (imagen[1].nodo[i].n ≥ 5){
                resto2 ← resto2 + 1;
                vector2[resto2] ← i;
            }
        }
    }
    if (resto2 ≥ m){
        for (i ← 1; i ≤ resto2; i ← i + 1){
            v[i] ← (i ≤ m)?TRUE : FALSE;
        }
        bon ← TRUE;
    }
    else bon ← FALSE;
    return (bon);
}
```

imagen:variable, §6 resto2:variable, §28 vector2:variable, §28

32. Un vector de longitud n' es recibido. Representa las curvas de un *circuito*. La variable m indica cuantos elementos del *circuito* son utilizados para calcular los descriptores. El número de combinaciones está dado por:

$$\begin{pmatrix} n' \\ m \end{pmatrix} \tag{A.5}$$

Algoritmo. Dado un arreglo v unidimensional de longitud n', con elementos 1 en las primeras m posiciones a la izquierda y el resto 0s, calcular el conjunto C de combinaciones de n' tomando m.

- Si $n' = m$, entonces tenemos un elemento de C.
- Si $m = 1$, entonces tenemos un elemento de C y tendremos $n' - 1$ corriendo un elemento del vector hacia la derecha.
- El conjunto C se forma con los resultados de aplicar el algoritmo con los últimos $n'-1$ elementos y con los primeros $n'-1$ elementos.

```
void subPostMatch1(w, v, n', m)
int *w, *v, n', m;
{
int i ,a[numCurves+1];
    for (i ← 1; i ≤ resto_z; i ← i + 1)a[i] ← v[i];
    ordenar(&a[0], resto_z, n', m);
    if (n' = m){
        try(w,&a[0]);
    }
    else {
        if (m=1) {
            i ← 1;
            while (i ≤ n'){
                try(w,&a[0]);
                shift(&a[0], resto_z, n' − i);
                i ← i + 1;
            }
        }
        else {
            subPostMatch1(w, &a[0], n' − 1, m − 1);
            subPostMatch1(w, &a[0], n' − 1, m);
        }
    }
}
```

numCurves:constante, §27
resto:variable, §28
subPostMatch1:procedimiento, §32
ordenar:procedimiento, §34
shift:procedimiento, §33
try:procedimiento, §36

33. Este procedimiento realiza un corrimiento hacia la derecha, del 1 más a la derecha, en el vector que indica las curvas presentes en correspondencia en el conjunto.

```
void shift(v,n,l)
int *v,n,l;
{
    v[n − l] ← FALSE;
    v[n − l + 1] ← TRUE;
}
```

FALSE:constante, §24 *TRUE*:constante, §24

34. Este procedimiento se asegura que un vector v de longitud n tenga m unos en sus primeras n' posiciones.

```
void ordenar(v, n, n', m)
int *v, n, n', m;
{
int i ,k;
    k ← 0;
    for (i ← 1; i ≤ n'; i ← i + 1){
        if (v[n-i +1]=TRUE) {
            k ← k + 1;
        }
        if (k ≠ m){
            for (i ← 1; i ≤ m; i ← i + 1){
                v[n − n' + i] ← TRUE;
            }
            v[n − n'] ← FALSE;
        }
}
```

FALSE:constante, §24 *TRUE*:constante, §24

35. La variable numeroGrupo nos sirve de consecutivo para expresar cual es el siguiente grupo de curvas, en el proceso de correspondencia de conjunto de curvas, que se forma.

< Variables globales 6 > + ≡

```
int numeroGrupo;
```

36. Los valores descriptivos de dos grupos de curvas son calculados; si están arriba de cierto umbral se determina si son mejores que los obtenidos previamente para las curvas que integran el grupo. De ser así la correspondencia se acepta.

Los descriptores utilizados en la comparación son:

Longitud. El cociente de la curva de menor longitud con el de mayor deberá ser mayor de *cocienteLong*.

Direccion. La diferencia entre el promedio de las direcciones deberá ser menor a θ_{20} grados.

Desplazamiento del Centroide. La diferencia entre la coordenada X deberá ser menor de *separacionImagenes* pixeles.

```
void try(w,v)
int *w,*v;
{
int i, xnum1, xnum2, n, good, k, match1, match2, grupo;
float xtheta1, xcentr1, ycentr1, m1, m2,
      xtheta2, xcentr2, ycentr2,
      Δl, Δx, Δy, Δtheta;
  xtheta1 ← 0.0; xcentr1 ← 0.0; ycentr1 ← 0.0; xnum1 ← 0;
  for (k ← 1; k ≤ resto1; k ← k + 1){
    if (w[k]=TRUE) {
      i ← vector1[k];
      n ← xnum1;
      xnum1 + ← imagen[0].nodo[i].n;
      xtheta1 ← (xtheta1 × n + imagen[0].nodo[i].theta×
            imagen[0].nodo[i].n)/xnum1;
```

```
            xcentr_1 ← (imagen[0].nodo[i].cx × imagen[0].nodo[i].n + xcentr_1 ×
                    n)/xnum_1;
            ycentr_1 ← (imagen[0].nodo[i].cy × imagen[0].nodo[i].n + ycentr_1 ×
                    n)/xnum_1;
        }
    }
    xtheta_2 ← 0.0; xcentr_2 ← 0.0; ycentr_2 ← 0.0; xnum_2 ← 0;
    for (k ← 1; k ≤ resto_2; k ← k + 1){
        if (v[k]=TRUE) {
            i ← vector_2[k];
            n ← xnum_2;
            xnum_2+ ← imagen[1].nodo[i].n;
            xtheta_2 ← (xtheta_2 × n + imagen[1].nodo[i].theta×
                    imagen[1].nodo[i].n)/xnum_2;
            xcentr_2 ← (imagen[1].nodo[i].cx × imagen[1].nodo[i].n + xcentr_2×
                    n)/xnum_2;
            ycentr_2 ← (imagen[1].nodo[i].cy × imagen[1].nodo[i].n + ycentr_2×
                    n)/xnum_2;
        }
    }
    Δx ←| xcentr_1 − xcentr_2 |
    m_1 ← xtheta_1; m_2 ← xtheta_2
    Δtheta ← arctan((m_2 − m_1)/(1 + m_1 × m_2))
    if (Δtheta < 0.0)Δtheta+ ← π
    if (Δtheta > pi/2.0)Δtheta ← π − Δtheta
    Δl ← (xnum_1 > xnum_2)?(float)xnum_2/(float)xnum_1 :
            (float)xnum_1/(float)xnum_2
    if ((Δx ≤ 30.0) ∧ (Δtheta ≤ θ_20)∧
            (Δl ≥ cocienteLong)){
        k ← 1; good ← TRUE;
        while ((k ≤ resto_1) ∧ (good = TRUE)){
            if (w[k] = TRUE){
                i ← vector_1[k];
                match_1 ← i;
                if (imagen[0].nodo[i].error > xnum_1)good ← FALSE;
            }
```

```
    k ← k + 1;
}
k ← 1;
while ((k ≤ resto_2) ∧ (good = TRUE)){
    if (v[k]=TRUE) {
        i ← vector_2[k];
        match_2 ← i;
        if (imagen[1].nodo[i].error > xnum_2)good ← FALSE;
    }
    k ← k + 1;
}
if (good=TRUE) {
    Δy ←| ycentr_1 − ycentr_2 |;
    grupo ← imagen[0].nodo[match_1].group;
    for (k ← 1; k ≤ imagen[0].index; k ← k + 1){
        if (imagen[0].nodo[k].group = grupo){
            imagen[0].nodo[k].group ← 0;
        }
    }
    for (k ← 1; k ≤ imagen[1].index; k ← k + 1){
        if (imagen[1].nodo[k].group=grupo){
            imagen[1].nodo[k].group ← 0;
        }
    }
    k ← 1;
    while (k ≤ resto_1){
        if (w[k]=TRUE) {
            i ← vector_1[k];
            imagen[0].nodo[i].error ← xnum_1;
            imagen[0].nodo[i].group ← numeroGrupo;
            imagen[0].nodo[i].Δx ← Δx;
            imagen[0].nodo[i].Δy ← Δy;
        }
        k ← k + 1;
    }
    k ← 1;
```

```
            while (k ≤ resto2){
                if (v[k]=TRUE) {
                    i ← vector2[k];
                    imagen[1].nodo[i].error ← xnum2;
                    imagen[1].nodo[i].group ← numeroGrupo;
                    imagen[1].nodo[i].Δx ← Δx;
                    imagen[1].nodo[i].Δy ← Δy;
                }
                k ← k + 1;
            }
            numeroGrupo ← numeroGrupo + 1;
        }
    }
}
```

cocienteLong:constante, §21
numeroGrupo:constante, §35
resto$_1$:variable, §28
separacionImagenes:constante, §19
TRUE:constante, §24
vector$_2$:variable, §28
imagen: variable, §6
ϕ:constante, 3.141592...
resto$_2$:variable, §28
θ_{20}:constante, §20
vector$_1$:variable, §28

37. Este parámetro es tomado de las especificaciones de operacion de la lente que utilizamos para tomar las imágenes.

< Constantes 9 > + ≡

#define distanciaFocal (50)

38. En caso de existir calibración correspondera a la ecuación:

$$z = \frac{f \times d}{\Delta y} \tag{A.6}$$

Donde f es la distancia focal, d representa la separación entre las imágenes al momento de tomar las fotos y Δy representa el desplaza miento de los puntos en su coordenada Y.

< Cálculo de profundidad 38 >≡

```
void calculeDepth() {
int i;
    for (i ← 1; i ≤ imagen[0].index; i ← i + 1){
    imagen[0].nodo[i].cz ←;
    (distanciaFocal × separacionImagenes)/(imagen[0].nodo[i].Δy);
    }
}
```

distanciaFocal:constante, §37 *imagen*:variable, §6
separacionImagenes:constante, §19

39. Los resultados del proceso de correspondencia son almacenados en un archivo. Los resultados muestran: la curva de referencia, su correspondencia, y en caso de existir calibración, la profundidad del centroide. También se incluyen los resultados cuando se han tomado grupos de curvas; en este caso, la profundidad corresponde al centroide formado por el conjunto de desciptores.

La profundidad de los puntos de la curva puede ser igualmente extraída; pero solo teníamos a nuestro alcance el número que corresponde a la coordenada de profundidad lo cual no clarificaría los resultados.

< Resultados 39 >≡

```
void saveInformation()
{
int i,grupo,limite,j;
    f ← fopen("output1","w");
    fprintf(f," I M A G E N ");
    for (i ← 1; i ≤ imagen[0].index; i ← i + 1){
        if ((imagen[0].nodo[i].matched ≠ 0)∧
                (imagen[0].nodo[i].group = 0)){
            fprintf(f, "curva : %d ∨ match : %d ∨ grupo : %d ∨ z : %f n",
                i, imagen[0].nodo[i].matched, imagen[0].nodo[i].group,
                imagen[0].nodo[i].cz);
        }
    }
    limite ← 0;
```

```
    for (i ← 1; i ≤ imagen[0].index; i ← i + 1){
        grupo ← imagen[0].nodo[i].group;
        if ((imagen[0].nodo[i].matched ≠ 0)∧
                (grupo > limite)){
            fprintf(f," grupo:");
            for (j ← 1; j ≤ imagen[0].index; j ← j + 1){
                if ((imagen[0].nodo[j].matched ≠ 0)∧
                    (imagen[0].nodo[j].group = grupo)){
                    fprintf(f,"%d ",j);
                }
            }
            for (j ← 1; j ≤ imagen[1].index; j ← j + 1){
                if ((imagen[1].nodo[j].matched ≠ 0)∧
                        (imagen[1].nodo[j].group = grupo)){
                    fprintf(f,"%d ",j);
                }
            }
            fprintf(f," %f ",imagen[0].nodo[i].cz);
            limite ← grupo;
        }
    }
    fclose (f);
}
```

imagen.variable, §6

40. Las curvas son recuperadas de un archivo heredado por el proceso de seguimiento de contornos. una vez recuperada esa información se procede a construir las relaciones estructurales entre las curvas de las imágenes.

    La correspondencia tiene una etapa en donde se pretende involucrar cada curva de la primera imagen con cada una de las curvas de la segunda imagen. Las ambigüedades son resueltas en base a la tendencia de las curvas candidatas a corresponder.

    Para las curvas que no encontraron correspondencia se propone un algoritmo en el que, basándonos en características estructurales de los

objetos en las imágenes, se calculan descriptores para conjuntos de curvas; estos descriptores son sometidos a un proceso de correspondencia donde los mejores pares son los que sobreviven.

Por último, si existe un proceso de calibracion la profundidad del centroide es calculada.

< Rutina Principal 40 >≡

```
main() {
    numerogrupo ← 1;
    loadInformation();
    createStructure(&imagen[0]);
    createStructure(&imagen[1]);
    initMatchStructure();
    matchSingleLine();
    matchCompletGraph();
    postMatching();
    calculeDepth();
    saveInformation();
}
```

calculeDepth:procedimiento, §38
matchCompletGraph:procedimiento, §23
imagen:variable, §6
numeroGrupo:variable, §35
saveInformation:procedimiento, §39
createStructure:procedimiento, §8
matchSingleLine:procedimiento, §22
initMatchStructure:procedimiento, §17
postMatching:procedimiento, §25

Lista de Figuras

Lista de Tablas

Bibliografía

[1] Gerard MEDIONI and Ramakant NEVATIA. *Segment Based Stereo Matching.* Computer Vision, Graphics, and Image Processing. **31**.2 - 18 (1985).

[2] Robert F. CHURCHHOUSE. *Numerical Methods Vol.III* John Wiley & Sons. (1981).

[3] William Eric Leifur GRIMSON. *From Images To Surfaces.* The MIT Press. (1981).

[4] Nasser M. NASRABADI, Sandra P. CLIFFORD, Yi LIU. *Integration of stereo vision and optical flow by using an energy-minimization approach.* J. Opt. Soc. Am.**6**.No 6. 900-907.(1989).

[5] Ellen C. HILDRETH. *Computations Underlying the Measurement of Visual Motion.*Artificial Intelligence. **23** 309-354. (1984).

[6] S.A. LLOYD. *Stereo matching using intra- and inter- row dynamic programming.* Pattern Recognition Letters. **4** 273-277 (1986).

[7] Nasser M. NASRABADI y Yi LIU.*Stereo vision correspondence using a multichannel graph matching technique.*Image and Vision Computing. **7**, No **4**. 237-245 (1989).

[8] Gerard MEDIONI y Ramakant NEVATIA. *Segment-Based Stereo Matching.* Computer Vision, Graphics, and Image Processing. **31**. 2-18. (1985).

[9] DUDA y HART. *Pattern Classification and Scene Analysis.* John Wiley & Sons, Inc. (1973).

[10] Juan Humberto SOSSA. *An improved parallel algorithm for thinning digital patterns.* Pattern Recognition Letters. **10.** 77-80. (1989).

[11] John CANNY. *A Computational Approach to Edge Detection.* IEEE Transactions on Pattern Analysis and Machine Intelligence. **8, No. 6.** 679 698. (1986).

[12] Donald E. KNUTH. *TEX: The program.* Addison Wesley publishing company, 1986.

EL JURADO DESIGNADO POR LA SECCION DE COMPUTACION DEL DEPARTAMENTO DE INGENIERIA ELECTRICA, APROBO LA TESIS: "Una Solución al Problema de - la Correspondiencia en un Par Estéreo Utilizando Descriptores de Curvas".

EL DIA 16 DEL MES agosto DEL AÑO 1991. .

DESARROLLADO POR EL: Ing. Joaquín Salas Rodríguez

OBTENIENDO EL GRADO DE: Maestro en Ciencias en Ingeniería Eléctrica/ Sección de Computación.

Sergio Víctor Chapa Vergara
Profesor Investigador Aux. de
Ingeniería Eléctrica/Computación.

Petra Wiederhold Grauert De Matos
Profra. Auxiliar Nivel 1-A,
de Control Automático.

José Luis Gordillo Moscoso
Prof. Investigador del Centro
de Inteligencia Artificial del
ITESM, CAMPUS Monterrey.