Capítulo 3
Selección de herramientas

Antes de detallar nuestro trabajo, es conveniente de acuerdo a nuestra perspectiva, definir cuáles son los elementos de un sistema de visión por computadora.

Es necesario manifestar que para desarrollar una aplicación que utilice el análisis de imágenes por computadora se tiene disponible cierto conjunto de elementos como las fuentes de luz necesarias para iluminar los objetos y los algoritmos encargados de la clasificación de estos objetos.

Para poder obtener una imagen se tienen condiciones que deben ser atendidas y una de ellas muy importante es que los objetos deben estar iluminados. Existen otras condiciones que por su variabilidad e impredecibilidad, son complejas y dependen en muchos casos de los entornos exteriores (condiciones climatológicas). Asi, los entornos interiores también poseen sus propias variables como la iluminación siendo muy importante este elemento en cualquier sistema de visión porque puede ser fijada.

La misión del elemento iluminación, es resaltar ciertos detalles de un objeto que interesen a la aplicación concreta, ocultar aquellos que impidan ver dichos detalles y garantizar que se están captando los objetos siempre con la misma intensidad. El tipo de iluminación depende del tipo de material y aplicación.

Los rayos reflejados por los objetos se captan y convierten en una señal eléctrica para que se puedan tratar por la computadora. Estas tareas son realizadas por una cámara cuya parte óptica concentra los rayos luminosos de la escena encuadrada sobre el elemento sensor de la cámara.

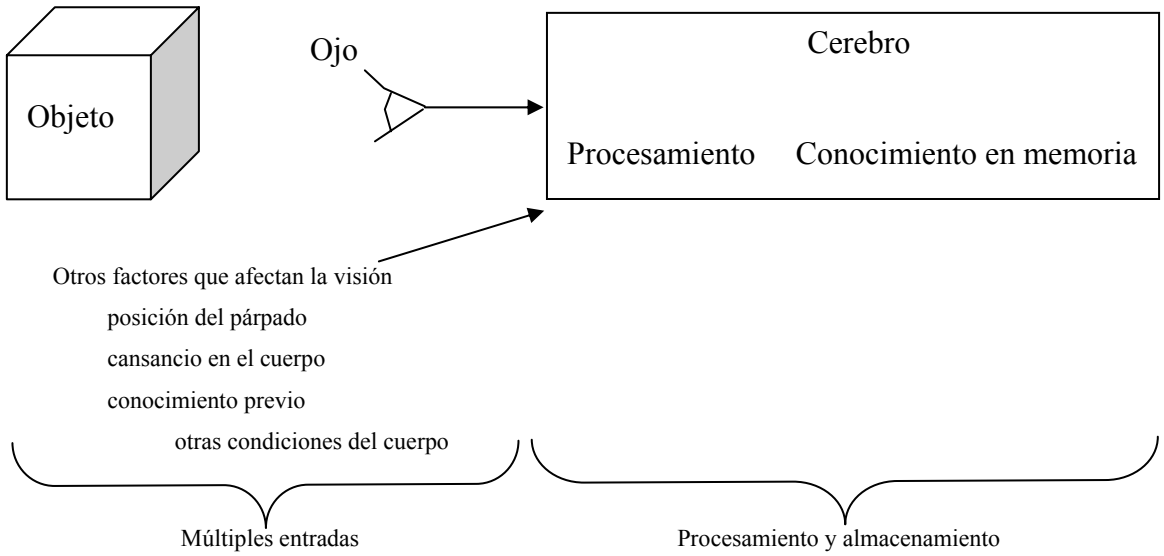
La máquina, a través de una tarjeta de adquisición y procesamiento de imágenes, trabaja las imágenes como señales digitales y obtiene una señal analógica para representar dicha imagen en un monitor de video. La imagen digital se introduce a memoria de la tarjeta de adquisición o de la propia computadora central y entonces se podrán aplicar diversas técnicas de preprocesamiento de imágenes y transformaciones morfológicas [González,1992.]

Los sistemas de visión suelen ser una parte de un sistema global. De tal modo, las tarjetas de entrada-salida ayudan a efectuar la integración del sistema con el resto de los dispositivos instalados: robots, y cintas transportadoras, por ejemplo.

3.1 Procesamiento de imágenes digitales

La visión humana es uno de los más importantes sentidos que tiene el ser humano, pues provee la información necesaria al cerebro (percepción, aprendizaje) para reconocer objetos, imágenes, colores, señales, letras, y luego adquirir el conocimiento y aprendizaje necesarios para interactuar con el ambiente y tomar decisiones, entre otras cosas.

Representación del sistema de visión del ser humano.



Una imagen en nuestros días juega un rol muy importante como un medio de comunicación masiva porque logra trasmitir en ocasiones más información que un texto [Schilling, 1990]. Debido a esto, los investigadores empezaron a trabajar para procesar y transmitir automáticamente la información óptica que contiene la imagen; alrededor de 1964 hubo un avance importante: el Laboratorio Jet Propulsión (Pasadena, California) realizó el procesamiento digital de las imágenes de la luna que provenían de un satélite. Así, una nueva rama de la ciencia llamada *procesamiento de imágenes* surgía y se ha desarrollado en áreas como las telecomunicaciones, y en la investigación médica y científica.

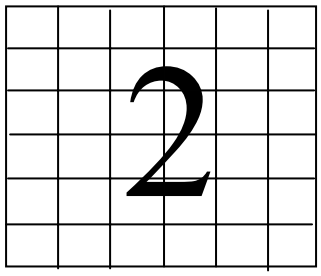
El procesamiento de una imagen digital consiste en transformarla a un formato digital y procesarla en una computadora. Para el desarrollo de este sistema, nos enfocaremos en el análisis de la imagen usando técnicas computacionales para el proceso o modificación de los datos de la misma después de haberla obtenido (por medio de la simulación de un proveedor como un satélite, o una cámara digital) y

digitalizado con la computadora. Para manipular la imagen, ésta se representa en una matriz *h* de tamaño *mxn*:

$$h = \begin{pmatrix} h(1,1) & h(1,2) & \dots & h(1,m) \\ h(1,2) & h(2,2) & \dots & h(2,m) \\ \vdots & \vdots & & \vdots \\ h(n,1) & h(n,2) & \dots & h(n,m) \end{pmatrix}$$

Cada elemento de la matriz representa a un píxel (picture element) de la imagen, el elemento más pequeño y simple que se controla en pantalla. Estos elementos son enteros que van de un rango de 0 a 255 para imágenes compuestas por 8 bits por píxel.

Ejemplo: representación de un número dentro de una región (conjunto de píxeles) cuadrada de tamaño 6x6.



0	0	0	0	0	0
0	0	5	7	0	0
0	0	1	8	0	0
0	0	2	2	0	0
0	0	0	0	0	0

La pantalla se forma por un arreglo de píxeles. Con *n* bits por píxel se pueden lograr 2^n colores distintos.

La escala de grises nos ayuda con el nivel de iluminación. El número total de niveles en una escala de grises es usualmente una potencia de 2:

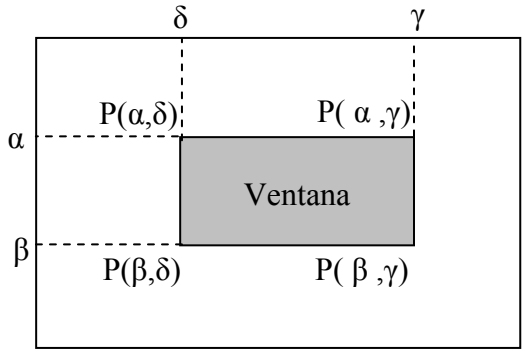
bits por píxel	# de colores	Rango de la escala de grises
1	2 (Blanco y Negro)	0,1
2	4	0 a 3
3	8	0 a 7
4	16	0 a 15
8	256	0 a 255
15	32 768	0 a 32 767
16	65 536	65 536
24	16 777 216	16 777 215
32	4 294 967 296	4 294 967 295
48	2.81474x10 14	2.81473x10 14
64	1.84467x10 19	1.84466x10 19

Un color se define en un espacio tridimensional que se compone de tres elementos. Estos elementos son RGB(Red, Green, Blue). Así, cada píxel tiene su propio color.

Los objetivos del procesamiento de imágenes en forma digital son:

1. Realzar las características de la imagen
2. Extraer información de la imagen

A una sub región de la imagen se le llama ventana y es acotada por 4 esquinas definidas como P(β,δ), P(β ,γ), P(α,δ), P(α ,γ) según se muestra a continuación:



Las operaciones aritméticas básicas (suma, resta y multiplicación) se aplican en el procesamiento de imágenes, un ejemplo de suma/resta de imágenes es el siguiente:

$$c[i][j] = a[i][j] \pm b[i][j]$$

y la multiplicación sería de la siguiente forma en donde *c* representa una constante: $b[i][j] = c \cdot a[i][j]$

Una subrutina para la suma se muestra a continuación:

```
int add(a,b,c,n1,m1,n2,m2)
  image a, b, c;
  int n1, m1, n2, m2;  //n1, m1 coordenadas iniciales, n2, m2 coordenadas finales, las 4
  int i,j,img;              //coordenadas definen el tamaño de la imagen.
  for(i=n1;i<n2;i ++)
      for(j=m1;j<m2;j++)
          {
               img=a[i][j]+b[i][j];
               if(img>255)
                    c[i][j]=255;
               else
                    c[i][j]=img;
          }
return 0;
```

La implementación de esta rutina permite observar que si ocurre un sobre flujo (el rango de cada elemento de la imagen va de [0 a 255]), el correspondiente píxel es

negro en un fondo blanco; y en el caso de que no haya sobre flujo, el correspondiente píxel aparece blanco en un fondo negro [Schilling, 1990].

El algoritmo para la sustracción es semejante al anterior, pero con una característica muy importante: puede ser usado para detectar cambios ocurridos durante un intervalo de tiempo mientras las dos imágenes que se comparan sean de la misma escena. A continuación se muestra un ejemplo de esto:

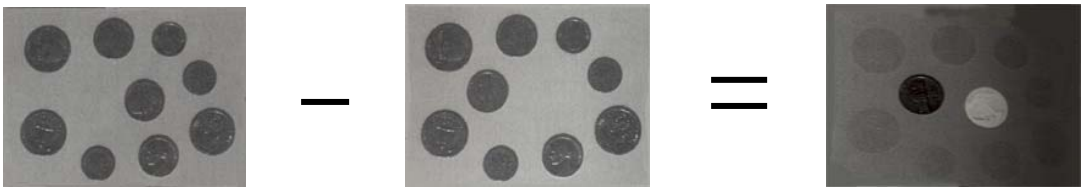


Imagen A — Imagen B — *Imagen C resultante de la sustracción*

Para esto, los valores de la matriz resultante deben ser positivos, y esto se logra usando alguno de los siguientes dos métodos:

1. Reescalar la imagen; se toma al número negativo más pequeño como igual a 0 y al número positivo más grande como el valor máximo de la escala de grises (por ejemplo sí tomamos 8 bits el valor asignado sería 255) [Russ,1992]. Cada elemento de la matriz toma el siguiente valor:

   B = rango(elemento actual – número negativo menor)/(número positivo mayor - número negativo menor)*

   donde rango en este caso es 255

2. Obtener el valor absoluto de la matriz resultante de la sustracción; usaremos este método para que todos los valores que se obtengan sean positivos [Russ,1992].

Así, las imágenes que se obtengan del simulador, una por una dentro de cierto intervalo, se comparará con lo almacenado en la base de datos a través de una sustracción para detectar si cambió o no el tráfico de cierta calle o ruta.

El tamaño de las imágenes aéreas (ortofotos) de San Andrés Cholula ya se tendrá definido, así como la localización, nombre de las calles dentro de la imagen, y la trayectoria de la red de la ciudad después de haber realizado un proceso a cada imagen que implique una parte de la región.

El simulador proveerá imágenes cuyas diferencias serán detectadas por el algoritmo de sustracción. Se conocerá el estado inicial de la imagen, y los valores de los píxeles de interés. Para nuestro proyecto, el estado inicial será cuando todas las calles de la ciudad estén vacías. Partiendo de este punto compararemos las imágenes.

Si la matriz resultante de la operación implica valores iguales a 0, entonces no hubo ningún cambio y definiremos a la ruta analizada como transitable. Si los valores de nuestro estado inicial cambiaron y tales cambios se mantienen durante un tiempo considerablemente largo, entonces la ruta analizada esta congestionada o el flujo es grande. Dependiendo de los resultados de la sustracción de imágenes, se valida si la calle puede ser transitada o no.

3.2 Teoría de grafos, fundamentos

Los grafos son objetos matemáticos usados para modelar redes de trabajo, estructuras de datos, calendarización de procesos, cálculos y otros sistemas donde las relaciones entre los objetos del sistema son importantes.

Un grafo G(V,E) consiste de un conjunto V de elementos (*vértices)* y un conjunto E de pares desordenados de miembros de V (*edges* o *aristas)*. Los vértices de un grafo son los puntos, y las aristas pueden ser las líneas que conectan cada par de puntos si imaginamos una representación geométrica del grafo.

La red de calles de San Andrés Cholula puede ser tratada como un grafo para conocer cual es la ruta óptima en determinado momento. Cada esquina que conecte calles se define como un nodo y aquella que sea el fin de una calle será definida como un nodo terminal; por lo tanto, los vértices adyacentes limitarán a una calle que será una arista incidente con dichos vértices.

Los puntos origen y destino que de acuerdo a la petición de usuario estarán cambiando, formarán parte del un sub grafo (ruta óptima) del grafo original.

De acuerdo a esto, se puede definir un path desde el punto origen al destino en la red vial como una secuencia alternada de vértices y aristas distintas, y además donde los vértices sucesivos v_i y v_{i+1} son puntos extremos de la arista intermedia e_i.

El grafo debe ser tipo conectado para poder determinar que la longitud más pequeña de un vértice a otro es llamado *shortest path* y su longitud se denomina distancia del vértice origen al vértice destino [McHugh,1990].

Además, el grafo debe ser dirigido (digrafo), esto implica agregar direcciones en sus aristas. El grafo representativo de la red vial es un multi grafo porque permite mas de una arista entre un par de vértices lo cual puede ser visualizado en la estructura de calles de alguna ciudad. De esta manera, podemos generalizar el tipo de grafo en cuanto

al direccionamiento de sus aristas, se trata de un grafo mezclado ya que se permitirían aristas dirigidas y no dirigidas.

La simple representación estática de grafos implica tener representaciones ligadas complicadas pero más económicas en términos de requerimientos de almacenamiento y además facilitan los cambios dinámicos de los grafos.

El concepto de grafos planos es completamente aplicable en nuestro sistema, un grafo plano puede ser dibujado o plasmado en un plano de tal manera que las aristas plasmadas se intersecten sólo en los vértices del grafo.

3.2.1 Técnicas de programación

Existen algunas técnicas para implementar la solución de problemas como los grafos y podemos ver más detalles acerca de esto en el apéndice C [Lau, 1989].

3.2.2 Ruta más corta (*shortest path*)

El problema de las rutas más cortas es un ejemplo de los múltiples problemas combinatorios que pueden ser solucionados de manera óptima combinando las soluciones óptimas de los sub problemas.

Si tenemos un grafo con distancias positivas asignadas a cada arista, es fácil mostrar que las sub rutas de un camino más corto entre un par de vértices son ellos mismos las rutas más cortas entre sus puntos finales; esta propiedad puede ser usada para diseñar un algoritmo que inductivamente construya el camino más corto entre dos puntos.

Si una solución óptima a un problema puede ser descompuesta en soluciones óptimas para sub problemas, entonces se usa el principio de optimalidad. Esta es la característica de definición de problemas a solucionarse por programación dinámica.

Podemos usar para este tipo de problema, procedimientos de búsqueda en forma de árbol. El algoritmo de la ruta más corta de Dijkstra encuentra la ruta más corta de un vértice inicial a uno final *extendiendo un árbol de búsqueda de ruta más corta cuya raíz es el vértice inicial y llega hasta el vértice final* que es punto final de la ruta más corta.

El formato general de un procedimiento de búsqueda de árbol es el siguiente: empieza en un vértice inicial y avanza hasta llegar a un vértice especial. Debido a que la búsqueda usualmente carece de un criterio global para guiar su dirección, su avance está basado en información puramente local. Por ejemplo, si una arista o un vértice ha sido

visitado previamente o es útil a la optimización. La búsqueda nunca avanza a un vértice que ya ha sido explorado para evitar re-explorar áreas que ya han sido visitadas.

Esto le da a la búsqueda su carácter de árbol, a partir de que las aristas avanzan a lo largo de la búsqueda en oposición a aquellas que son examinadas pero no seguidas, se induce un árbol en el grafo que se está procesando. Conforme la búsqueda progresa, eventualmente llega a ser bloqueada y es forzada a alterar su dirección. Si el bloqueo está en una dirección particular, el algoritmo puede intentar avanzar la búsqueda desde algún punto previamente alcanzado. Por otro lado, un bloqueo completo usualmente señala la terminación de por lo menos la fase actual del algoritmo.

Por lo tanto, una búsqueda depende del criterio usado para determinar si avanzar en el árbol de búsqueda a un vértice dado o a lo largo de una arista, y de otras determinantes tales como:

- ¿cómo son los vértices procesados incompletamente o aristas punteadas reservadas o recalendarizadas para el viaje subsecuente? y,
- ¿cómo son manejadas las brechas a un vértice especial, y los bloqueos parciales o completos?

El criterio que controla una búsqueda depende del problema. Así, la decisión de avanzar a lo largo de una arista puede depender de si la arista tiene una capacidad para flujo adicional o alguna característica alterna.

Una brecha a un vértice especial usualmente indica la detección de una ruta desde la raíz del árbol cruzando el árbol hasta un vértice especial, el cual puede ser la solución al problema, o una ayuda al proveer una sub solución óptima actual. Ahora, un completo bloqueo de la búsqueda desde su punto inicial actual, aunque en general indicando la terminación al menos de la fase actual del algoritmo. También usualmente tiene algunas implicaciones que dependen del problema, tal como que el sub grafo alcanzado por la búsqueda es irrelevante al problema de optimización, o que no hay solución al problema (tal como una falla para alcanzar el vértice destino con el algoritmo de Dijkstra) [Harary,1990].

3.2.3 Depth First Search

Es la técnica de búsqueda de grafo más prominente, y puede ser usada para identificar una variedad de propiedades estructurales de un grafo tales como su orientabilidad, la identidad de sus componentes fuertes o bloques. Es la clásica instancia

de un algoritmo recursivo de procesamiento de grafos, tiene un eficiente desempeño y muchos algoritmos basados en ella, adquieren su eficiencia.

3.2.4 Dígrafos *(Digraph)*

Un digrafo con pesos (o red de trabajo) es un digrafo con pesos valuados como reales o longitudes asignadas a cada arista. Equivalentemente, un dígrafo de pesos es una tripleta (*V*,*E*,*w*) donde *V* y *E* tienen la interpretación usual y *w* es una función que mapea los elementos de *E* en los reales. La longitud de una ruta en un dígrafo de pesos es la suma de las longitudes de las aristas en la ruta.

Dado un digrafo con pesos asignados a sus aristas, podemos modelar el problema de la ruta más corta entre un par de vértices en un grafo. El objetivo es construir un conjunto de variables de aristas x_{ij} de tal manera que el conjunto de aristas (i,j) para cada x_{ij} que no es cero, determine un camino más corto entre dos vértices. Se podrían definir los pesos de las aristas por w_{ij}. Así, el modelo de programación entera es:

1) $0 <= x_{ij} <= 1$, para cada arista (i,j),

2) $\sum_j x_{ij} <= 1$, $\sum_j x_{ji} <= 1$, para cada vértice i,

3) $\sum_j x_{ij} = \sum_j x_{ji}$, para cada vértice i<>s,t

4) $\sum_j x_{sj} = \sum_j x_{jt} = 1$

5) $\sum_j x_{tj} = \sum_j x_{js} = 0$

6) $\min \sum_i w_{ij} x_{ij}$

La desigualdad número 1) asegura los caracteres 0 ó 1 de las variables de cada arista. El grafo contemplado para este modelo es no dirigido y por lo tanto no hay condición de simetría. La desigualdad 2) asegura que a lo más hay una arista no cero dentro y fuera de cualquier vértice; la ecuación 3) asegura que hay un número igual de aristas no cero dentro y fuera de cada vértice que no sean el vértice inicial *s* y el vértice terminal *t*. Ahora, la ecuación 4) forza a que debe haber exactamente una arista de salida

no cero en *s* y exactamente una arista de entrada no cero en *t*, mientras que la ecuación 5) forza a que debe existir una arista de entrada no cero en *s* y ninguna de salida en *t*.

La función objetivo 6) solo suma los pesos de las aristas no ceros. La solución óptima entonces define la ruta más corta desde *s* a *t*, dado que el grafo satisface la siguiente condición: el grafo no contiene ciclos de tal manera que el peso de las aristas son negativos. Así, la solución óptima podría corresponder a una unión disjunta de una ruta y a un conjunto de ciclos disjuntos, en vez de que la ruta más corta se produzca [Behzad, 1979].

3.2.5 El algoritmo de Dijkstra: vértice a vértice

Un camino más corto entre un par de vértices *x* y *y* en un dígrafo de pesos es una ruta de *x* a *y* de la menor longitud. La distancia de *x* a *y* es definida como la longitud del camino más corto de *x* a *y*. Presentamos un modelo que ilustra la aplicación de las rutas más cortas.

3.2.5.1 Modelo: grafos de visibilidad

Podemos plantearlo si consideramos el problema de encontrar los caminos más cortos a través de una región planar con obstáculos. Específicamente, si consideramos a *x* y a *y* un par de puntos en el plano, y se supone que se desea encontrar el camino más corto entre tales puntos que este libre en sus interiores de un conjunto de regiones poligonales R. Se puede modelar este problema como uno de ruta más corta en un grafo de pesos introduciendo el *grafo de visibilidad* asociado con la configuración. Por lo tanto, se podría denotar a Vert(R) como los vértices de los obstáculos poligonales de R. Por lo tanto, los vértices del grafo es la unión de Vert(R) y de (*x*,*y*). Se dirá que un vértice U en el conjunto de vértices del grafo es *visible* de un vértice V dentro de los vértices del grafo si el segmento de línea UV corta los interiores de ninguna de las regiones de R. En caso de que u sea visible de v, incluimos el segmento de línea (u,v) como una arista del grafo, y se permite que la longitud euclidiana de la arista (u,v) sea el peso de (u,v). Por lo tanto, la ruta libre de obstáculos más corta de *x* a *y* es solo la ruta entre *x* y *y* en el grafo.

3.2.5.2 Método de Dijkstra

Este es un algoritmo que resuelve el siguiente problema: permite a un grafo ser un dígrafo de pesos cuyos pesos de las aristas son positivos, y permite a *x* y a *y* constituir un par de vértices del grafo en análisis. El algoritmo encuentra la ruta más corta entre *x* e *y* dentro del grafo cuya longitud o visibilidad no es cero. Usa una técnica de árbol de búsqueda y se basa en la observación de que el vértice *k* más cercano a un vértice dado *x*, es el vecino de uno de los vértices *j* más cercanos a *x*, para algunos $j<k$, por lo tanto es posible fácilmente encontrar el vértice más cercano *k* a *x*.

Si Near(*j*) denota el vértice *j* más cercano a *x*, entonces Dist(*u*) denota la distancia de *x* a cualquier vértice *u* , y Length(*u,w*) denota la longitud de la arista de *u* a cualquier vecino *w* de *u*. Por lo tanto, el vértice *k* más cercano a *x* es aquel vértice *v* que minimiza:

$$\text{Dist(Near(j))} + \text{Length(Near(j),v)},$$

donde el mínimo es tomado sobre todo $j<k$. Por lo tanto, para encontrar la distancia a *y*, se debe primero encontrar de manera inductiva las distancias a todos los vértices más cercanos a *x* que a *y*. Los vértices más distantes de *x* son encontrados usando un procedimiento de búsqueda que explora el grafo de una manera de árbol. Esta búsqueda induce a un sub digrafo del grafo original llamado árbol de búsqueda, el cuál en turno contiene un subárbol distinguido llamado un subárbol de la ruta más corta que tiene la propiedad de que las distancias más cortas a todos sus vértices son conocidas. El proceso de búsqueda es continuado hasta que el vértice objetivo *y* es incorporado en el subárbol de la ruta más corta.

En cada fase del proceso de búsqueda, un nuevo vértice *v* falso en el árbol de búsqueda pero no en el subárbol de la ruta más corta es incorporado en el subárbol de ruta más corta, y el árbol de búsqueda es entonces extendido de *v* a sus vecinos. Para cada vecino *w* de *v*, se apunta *w* hacia *v* como su predecesor en el árbol de búsqueda si *w* no fue visitado previamente, o si *w* ya fue visitado, pero el camino a través del árbol de *x* a *w* es más largo que el camino a través del árbol de *x* a *v* más la arista de *v* a *w*.

Inicialmente, el árbol de búsqueda *fans out* desde *x* a sus vecinos inmediatos, definiendo una estrella en *x*. El segundo vértice más cercano a *x* es aquel vértice en la estrella que aparenta estar al final de una arista más corta incidente con *x*. Esto depende de manera crítica de la positividad de los pesos de las aristas. Se puede distinguir entre el árbol de búsqueda que incluye todos aquellos vértices muy lejanos encontrados por el procedimiento de búsqueda (el cuál sería la estrella) y el subárbol de camino más corto

que consiste de *x* y sus vecinos más cercanos.

Después de *k* etapas, el subárbol de camino más corto del árbol de búsqueda, contiene los *k* vértices más cercanos a *x*. La ruta a través de este árbol desde *x* a cualquiera de estos vértices es una ruta más corta. Por otro lado, el camino de árbol desde *x* a los vértices en el árbol de búsqueda, los cuales no están aún en el subárbol de la ruta más corta, implica estimaciones de rutas más cortas únicamente. El vértice *(k+1)-st* más cercano a *x* es aquél vértice *v* en la etapa *k* del árbol de búsqueda cuya distancia estimada desde *x* es un mínimo entre todos aquellos vértices que no están en el árbol de ruta más corta [Even, 1979].

Debido a que este algoritmo es muy importante en nuestro sistema, en el siguiente capítulo lo describiremos más a detalle.

3.3 Rectas, fundamentos

La teoría acerca de las líneas rectas se usa para administrar la información obtenida después del procesamiento de imágenes y después de la apicación de la transformada de Hough.

Recordemos que si a y b son dos números reales, la función $f(x) = ax+b$ es una recta.

Si $a = 0$, entonces $f(x) = b$ es la función constante, es una recta paralela al eje x.

Si $a \mathrel{!}= 0$ y $b = 0$, $f(x) = a*x$ es una recta que pasa por (0,0), ya que:

si $x \mathrel{!}= 0$, (x, y) pertenece a la recta si y solo si $(y/x) = tan(alfa) = a$;

donde a es la pendiente de la recta

La pendiente es la relación entre el cambio en sentido vertical (de las *y*) y el cambio en sentido horizontal (de las *x*), al ir de un punto a otro de la recta. Indica la inclinación de la recta en relación a los ejes coordenados.

Ahora, si $b \mathrel{!}= 0$, $f(x) = a*x + b$ es una recta paralela a la anterior y pasa por el punto $(0,b)$. La ecuación $y = a*x + b$ corresponde a la de una recta, es decir, la recta es el *lugar geométrico* de los puntos del plano que satisfacen la ecuación. O sea, un punto $P(x_0,y_0)$ está en la recta (definida por $y = a*x + b$) o la recta pasa por ese punto, si y sólo si $y_0 = a*x_0 + b$.

A partir de que $y-y_0 = a*(x-x_0)$, podemos decir que $a = ((y-y_0)/(x-x_0))$ es otra forma de obtener la pendiente de la recta para $(x-x_0) \mathrel{!}= 0$.

Bien, si tenemos a dos puntos $P1(x_1,y_1)$ y $P2(x_2,y_2)$, se tiene una única recta que pasa por ellos; la ecuación de la recta por la que pasa $P1$ es $y-y_1 = a(x-x_1)$ y aquella por la que pasa $P2$ es $y-y_2 = a(x-x_2)$.

Así, la recta por la que pasan los dos es $y-y_1 = ((y_2-y_1)/(x_2-x_1))*(x-x_1)$ para $(x_2-x_1) != 0$.

El paralelismo entre dos rectas se define si tienen la misma pendiente, entonces la constante de la ecuación puede variar en cada una.

Dos rectas son perpendiculares si sus pendientes a y a' satisfacen la ecuación $a = 1/(a')$.

El uso de cada una de las siguientes formas depende de los datos disponibles:

Forma general: $Ax + By + C = 0$

Forma dos puntos: $y-y_1 = ((y_2-y_1)/(x_2-x_1))*(x-x_1)$

Forma punto-pendiente: $y - y_1 = m(x - x)$

Forma pendiente-intersección: $y = mx + b$

Forma intersecciones: $(x/a) + (y/b) = 1$, donde a y b son valores de x y y de dos puntos diferentes que intersectan otras dos rectas

3.4 Transformada de Hough, fundamentos

Si tenemos n puntos en una imagen y deseamos encontrar subconjuntos de esos puntos que formen líneas rectas, tenemos una solución que implica encontrar todas las rectas determinadas por cada par de puntos y así encontrar todos los subconjuntos de puntos que están cerca de rectas concretas. El inconveniente es que es necesario encontrar $((n*(n-1))/2)$ líneas y hacer $((n^2*(n-1))/2)$ comparaciones (de cada punto para todas las líneas). Ya que el costo computacional es muy alto, entonces se puede usar la **transformada de Hough** (1962) de P.V.C., el cuál es un método con un rol importante en la visión artificial [Whelan, Molloy,2001].

Se tiene un punto (x_i,y_i) y la ecuación explícita de la recta $y_i=a*x_i+b$; existe un número infinito de rectas que pasan por (x_i,y_i) pues todas ellas satisfacen la ecuación al variar los valores de a y b.

Como la ecuación puede tomar la forma $b=-x_i*a+yi$, consideremos el plano ab (*espacio paramétrico*) y entonces estamos ante la ecuación de una recta para un par fijo (x_i,y_i).

Otro punto (x_j,y_j) tiene asociada una recta en tal espacio, la cuál se intersectará con la línea asociada a (x_i,y_i) en (a',b') donde a' es la pendiente y b' es la constante de la recta que contiene a (x_i,y_i) y a (x_j,y_j) en el plano xy.

Todos los puntos de esta recta tendrán rectas en el espacio que se intersectan en (a',b').

La ventaja computacional de Hough surge al subdividir el espacio paramétrico en "celdas acumuladoras".

Los intervalos $(a_{mín},a_{máx})$ y $(b_{mín},b_{máx})$ son los esperados para los valores de la pendiente (a') y de la constante (b'). La celda de coordenadas (i,j), con valor acumulador A(i,j), es el cuadrado con coordenadas del espacio paramétrico (a_i,b_j). Al inicio, todas valen cero. Para cada punto (x_k,y_k) en la imagen, se iguala *a* al valor de la subdivisión permitido en el eje "a", y se obtiene el valor *b* correspondiente con la ecuación $b=-x_k*a+y_k$. Los valores de *b* que se obtienen se aproximan en el eje "b" al valor permitido más cercano.

Si una elección de a_p genera b_q, se incrementa su acumulador, $A(p,q):=A(p,q)+1$. Al final del proceso, un valor *M* en A(*i,j*) indica *M* puntos en el plano *xy* sobre la recta $y=a_i*x+b_j$. La precisión de linealidad de tales puntos se da por el número de subdivisiones en el plano *ab* [De la Escalera, 2001].

Una subdivisión del eje "a" en *k* partes, indica que para cada punto (x_k,y_k) se obtienen *k* valores de *b* que son los *k* posibles valores de *a*. Si hay *n* puntos de imagen, esto implica *n*k* cálculos, lo cual es un procedimiento lineal en "*n*". El producto *n*k* no se acerca al número de operaciones inicial, a no ser que *k* sea similar o superior a *n.*

El problema al usar la ecuación de la recta $y=a*x+b$ es que la pendiente y la constante tienden a infinito si la recta es vertical, optamos por $x*cos(teta)+y*sen(teta) = rho$.

Esta representación usa igualmente una tabla y acumuladores. El procedimiento es idéntico, en lugar de líneas rectas, se tienen curvas sinusoidales en el plano (teta,rho). *M* puntos de la recta $x*cos\ (teta)_j+y*sen\ (teta)_j = rho_i$ generarán *M* curvas sinusoidales que se cortan en el espacio paramétrico en $(rho_i,teta_j)$.

Al incrementar *teta* y obtener *rho*, habrá *M* entradas en el acumulador *A(i,j)* asociado a la celda $(rho_i,teta_j)$.

Este método se aplica a cualquier función de forma $g(x,c) = 0$, donde x es un vector de coordenadas y *c* es un vector de coeficientes. Así, los puntos del círculo $(x-c_1)^2+(y-c_2)^2 = (c_3)^2$ se detectan usando el mismo proceso sólo que ahora hay 3 parámetros (c_1,c_2,c_3), un espacio paramétrico tridimensional, con celdas cúbicas y acumuladoras de la forma A(i,j,k).

Se incrementa c_1 y c_2 para obtener c_3, se actualiza el acumulador de (c_1,c_2,c_3).

La complejidad de esta transformada depende del número de coordenadas y coeficientes de la función.

Su aplicación principal es en técnicas de segmentación, en extracción de características, compresión y reconocimiento de imágenes. Se usa para el enlace de puntos de borde y extracción de rectas; implica transformar coordenadas cartesianas a polares.

No es capaz de encontrar los extremos de la recta, ha sido patentada y su uso en algún proyecto requiere el pago de royaltis a la familia Hough.

En realidad, la transformada puede ser usada para reconocer patrones, ya que presenta buenos resultados cuando la imagen implica mucho ruido o deformaciones; además es robusta a problemas de oclusión y discontinuidades de bordes [www.anc.ed.ac.uk/~amos].

3.5 Conclusiones

El desarrollo de nuestro sistema está basado en la teoría descrita en este capítulo, cada una de las diversas herramientas a usarse tiene como respaldo una teoría completa que sirve para la comprensión y claridad de su utilidad dentro de nuestro trabajo. Realmente es interesante interpretar la teoría y más tarde implementar aplicaciones basadas en el conocimiento adquirido. En este capítulo se ha mostrado un poco de lo que se analizó para tener clara la estrategia de cómo tratar los problemas implícitos en el desarrollo del sistema. Conforme avance nuestra descripción, abordaremos parte de dicha teoría con el objetivo de mostrar en que herramienta está siendo útil determinado conocimiento.

Indudablemente, la investigación relacionada a cualquier campo de estudio constituye una herramienta indispensable para el desarrollo, en términos generales, del ser humano.